

Дьяконов В. П.

МАТЛАВ

Полный самоучитель

В. П. Дьяконов

MATLAB

Полный самоучитель



УДК 32.973.26-018.2

ББК 004.438

Д93

Д93 Дьяконов В. П.

MATLAB. Полный самоучитель. – М.: ДМК Пресс, 2012. – 768 с.: ил.

ISBN 978-5-94074-652-2

Самоучитель по массовой матричной системе MATLAB, занимающей лидирующее место в области численных научно-технических вычислений, расчетов и моделирования. Основное внимание уделено описанию основ применения и языка программирования базовой системы MATLAB, реализации численных методов вычислений и визуально-ориентированному проектированию графического интерфейса пользователя (GUI). Описаны особенности интерфейса MATLAB, операторы, функции и средства программирования. Приведены сотни примеров применения MATLAB в учебных, научно-технических и математических вычислениях и расчетах. Для студентов, преподавателей и аспирантов университетов и вузов различного профиля, инженеров и научных работников.

MATLAB and Simulink are registered trademark of The MathWorks Inc.
Blockset, Toolbox and its components are trademark of The MathWorks Inc.

УДК 32.973.26-018.2

ББК 004.438

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.

© Дьяконов В. П., 2012

ISBN 978-5-94074-652-2

© Оформление, издание, ДМК Пресс, 2012

Краткое содержание

| | |
|---|-----|
| Введение | 35 |
| Благодарности и адреса для связи | 40 |
| Урок 1. ПЕРВОЕ ЗНАКОМСТВО С MATLAB | 41 |
| Урок 2. ЗНАКОМСТВО С ИНТЕРФЕЙСОМ ПОЛЬЗОВАТЕЛЯ | 91 |
| Урок 3. ПРОГРАММНЫЕ СРЕДСТВА МАТЕМАТИЧЕСКИХ ВЫЧИСЛЕНИЙ | 151 |
| Урок 4. ОПЕРАЦИИ С ВЕКТОРАМИ И МАТРИЦАМИ | 193 |
| Урок 5. ТИПЫ ДАННЫХ – МАССИВЫ СПЕЦИАЛЬНОГО ВИДА | 233 |
| Урок 6. ПРОГРАММНЫЕ СРЕДСТВА ОБЫЧНОЙ ГРАФИКИ | 277 |
| Урок 7. ПРОГРАММНЫЕ СРЕДСТВА СПЕЦИАЛЬНОЙ ГРАФИКИ | 341 |

| | |
|--|------------|
| Урок 8. ПРОГРАММНЫЕ СРЕДСТВА ЧИСЛЕННЫХ МЕТОДОВ | 383 |
| Урок 9. ПРОГРАММНЫЕ СРЕДСТВА ОБРАБОТКИ ДАННЫХ..... | 441 |
| Урок 10. РАБОТА СО СТРОКАМИ, ФАЙЛАМИ И ЗВУКАМИ | 493 |
| Урок 11. ТИПОВЫЕ СРЕДСТВА ПРОГРАММИРОВАНИЯ | 531 |
| Урок 12. ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ GUI | 591 |
| Урок 13. ОБЗОР РАСШИРЕНИЙ MATLAB | 659 |
| Урок 14. СТЫКОВКА MATLAB С ИЗМЕРИТЕЛЬНЫМИ ПРИБОРАМИ | 717 |
| Список литературы | 743 |
| Предметный указатель | 747 |

Содержание

| | |
|---|----|
| Введение | 35 |
| Благодарности и адреса для связи | 40 |
| Урок 1. Первое знакомство с MATLAB | 41 |
| 1.1. Назначение и особенности системы MATLAB ... | 42 |
| 1.1.1. Начальные сведения о матрицах | 42 |
| 1.1.2. Назначение матричной системы MATLAB | 43 |
| 1.1.3. Системные требования к установке | 44 |
| 1.1.4. Инсталляция системы MATLAB 7 + Simulink 6 | 45 |
| 1.1.5. Файловая система MATLAB | 45 |
| 1.2. Начало работы с MATLAB | 46 |
| 1.2.1. Запуск MATLAB и работа в режиме диалога | 46 |
| 1.2.2. Понятие о сессии работы с системой MATLAB | 47 |
| 1.2.3. Новый и старый облики системы MATLAB | 48 |
| 1.2.4. Операции строчного редактирования | 49 |
| 1.2.5. Команды управления окном | 49 |
| 1.3. Простые вычисления в MATLAB | 50 |
| 1.3.1. MATLAB в роли мощного научного калькулятора | 50 |
| 1.3.2. Форма вывода и перенос строки в сессии | 53 |
| 1.3.3. Запуск примеров применения MATLAB из командной строки | 54 |
| 1.4. Основные объекты MATLAB | 55 |

| | |
|---|-----------|
| 1.4.1. Понятие о математическом выражении | 55 |
| 1.4.2. Действительные и комплексные числа | 55 |
| 1.4.3. Форматы чисел | 56 |
| 1.4.4. Константы и системные переменные | 57 |
| 1.4.5. Текстовые комментарии в программах | 58 |
| 1.4.6. Переменные и присваивание им значений | 58 |
| 1.4.7. Уничтожение определений переменных | 59 |
| 1.4.8. Операторы и встроенные функции MATLAB | 60 |
| 1.4.9. Применение оператора : (двоеточие) | 62 |
| 1.4.10. Функции пользователя | 63 |
| 1.4.11. Сообщения об ошибках и исправление ошибок | 64 |
| 1.5. Формирование векторов и матриц | 66 |
| 1.5.1. Задания векторов и матриц и доступ к их элементам | 66 |
| 1.5.2. Задание векторов и матриц с комплексными элементами | 67 |
| 1.5.3. Понятие о матричных операциях и магические матрицы | 68 |
| 1.5.4. Конкатенация (объединение) матриц | 69 |
| 1.5.5. Удаление столбцов и строк матриц | 70 |
| 1.6. Операции с рабочей областью, текстом сессии и редактором m-файлов | 70 |
| 1.6.1. Дефрагментация рабочей области | 70 |
| 1.6.2. Сохранение рабочей области сессии | 71 |
| 1.6.3. Ведение дневника | 71 |
| 1.6.4. Загрузка рабочей области сессии | 72 |
| 1.6.5. Работа с редактором m-файлов | 73 |

| | |
|---|-----------|
| 1.6.6. Завершение вычислений и работы с системой | 73 |
| 1.7. Интерактивная справка из командной строки | 74 |
| 1.7.1. Вызов списка разделов интерактивной справки | 74 |
| 1.7.2. Справка по конкретному объекту | 75 |
| 1.7.3. Справка по группе объектов | 76 |
| 1.7.4. Справка по ключевому слову | 77 |
| 1.7.5. Дополнительные справочные команды | 77 |
| 1.8. Работа с демонстрационными примерами с командной строки | 78 |
| 1.8.1. Вызов списка демонстрационных примеров | 78 |
| 1.8.2. Пример – вывод изображения поверхности | 79 |
| 1.8.3. Что больше – $e^{\pi i}$ или πi^e ? | 80 |
| 1.8.4. Встроенные фигуры | 81 |
| 1.8.5. Просмотр текстов примеров и m-файлов | 81 |
| 1.9. Знакомство с двумерной графикой MATLAB | 82 |
| 1.9.1. Особенности двумерной графики MATLAB | 82 |
| 1.9.2. Графики функций одной переменной | 83 |
| 1.9.3. Графики ряда функций | 84 |
| 1.9.4. Графическая функция fplot | 85 |
| 1.10. Знакомство с трехмерной графикой MATLAB | 86 |
| 1.10.1. Построение трехмерных графиков | 86 |
| 1.10.2. Вращение графиков мышью | 87 |
| 1.10.3. Контекстное меню графиков | 88 |

Урок 2. Знакомство с интерфейсом

| | |
|--|-----------|
| пользователя | 91 |
| 2.1. Интерфейс основного окна MATLAB | 92 |
| 2.1.1. Средства панели инструментов | 92 |
| 2.1.2. Браузер рабочей области | 93 |
| 2.1.3. Команды просмотра рабочей области who и whos | 95 |
| 2.1.4. Браузер файловой структуры | 96 |
| 2.2. Работа с меню | 96 |
| 2.2.1. Команды, операции и параметры | 96 |
| 2.2.2. Меню системы MATLAB | 97 |
| 2.2.3. Меню File | 97 |
| 2.2.4. Установка путей доступа файловой системы | 98 |
| 2.2.5. Настройка элементов интерфейса | 99 |
| 2.2.6. Меню Edit – средства редактирования документов | 99 |
| 2.2.7. Интерфейс по умолчанию | 100 |
| 2.3. Основы редактирования и отладки m-файлов ... | 100 |
| 2.3.1. Интерфейс редактора/отладчика m-файлов | 100 |
| 2.3.2. Цветовые выделения и синтаксический контроль .. | 102 |
| 2.3.3. Понятие о файлах-сценариях и файлах-функциях .. | 102 |
| 2.3.4. Панель инструментов редактора и отладчика | 103 |
| 2.4. Новинки графического интерфейса MATLAB | 104 |
| 2.4.1. Новая позиция меню Graphics | 104 |
| 2.4.2. Работа с окном 2D-графики MATLAB | 104 |
| 2.4.3. Работа с редактором графики MATLAB | 105 |
| 2.4.4. Построение графиков из их каталога | 108 |

| | |
|---|------------|
| 2.4.5. Некоторые другие особенности применения редактора графики | 111 |
| 2.4.6. Новый вид окна MATLAB | 114 |
| 2.5. Интерфейс графических окон | 115 |
| 2.5.1. Обзор интерфейсов графических окон | 115 |
| 2.5.2. Панель инструментов камеры обзора | 117 |
| 2.5.3. Меню инструментов Tools | 117 |
| 2.5.4. Вращение графиков мышью | 117 |
| 2.5.5. Операции вставки | 118 |
| 2.6. Основы форматирования графиков | 118 |
| 2.6.1. Форматирование двумерных графиков | 118 |
| 2.6.2. Форматирование линий графиков | 118 |
| 2.6.3. Работа с инструментом Plot Tool | 120 |
| 2.6.4. Работа с редактором графики MATLAB | 120 |
| 2.6.5. Форматирование линий графиков и маркеров опорных точек | 121 |
| 2.6.6. Форматирование линий и маркеров для графика нескольких функций | 123 |
| 2.6.7. Форматирование осей графиков | 124 |
| 2.6.8. Позиция Tools меню окна графики | 124 |
| 2.6.9. Нанесение надписей и стрелок прямо на график ... | 125 |
| 2.6.10. Применение графической «лупы» | 126 |
| 2.6.11. Построение легенды и шкалы цветов на графике ... | 127 |
| 2.6.12. Работа с камерой 3D-графики | 128 |
| 2.7. Работа с Мастером импорта данных | 130 |
| 2.7.1. Открытие окна Мастера импорта данных | 130 |
| 2.7.2. Информация об импортируемых бинарных данных | 132 |

| | |
|--|------------|
| 2.7.3. Импорт данных mat-формата | 133 |
| 2.7.4. Импорт данных текстового формата | 133 |
| 2.7.5. Об экспорте данных | 135 |
| 2.8. Работа со справкой и демонстрационными примерами | 135 |
| 2.8.1. Запуск справочной системы Help Desk | 136 |
| 2.8.2. Справка по функциям и полнотекстовый обзор | 137 |
| 2.8.3. Работа с демонстрационными примерами | 139 |
| 2.9. Интерфейс и новые возможности MATLAB R2007 | 140 |
| 2.9.1. Интерфейс MATLAB R2007a по умолчанию | 140 |
| 2.9.2. Упрощенный интерфейс MATLAB R2007a | 141 |
| 2.9.3. Редактор/отладчик программ и файлов MATLAB R2007a | 143 |
| 2.9.4. Справка MATLAB R2007a | 144 |
| 2.9.5. Новые возможности MATLAB R2007a,b | 144 |
| 2.9.6. Интерфейс и справка MATLAB R2007b | 145 |
| 2.9.7. Общая настройка MATLAB R2007b | 146 |
| 2.9.8. Производительность реализаций MATLAB R2007a,b | 148 |

Урок 3. Программные средства математических вычислений

151

| | |
|---|------------|
| 3.1. Вычислительные и логические операции | 152 |
| 3.1.1. Арифметические матричные операторы и функции ... | 152 |
| 3.1.2. Операции отношения | 153 |
| 3.1.3. Логические операции и операторы | 155 |

| | |
|---|-----|
| 3.2. Специальные символы | 156 |
| 3.2.1. Специальные символы | 156 |
| 3.2.2. Системные переменные и константы | 159 |
| 3.3. Работа со специальными данными | 161 |
| 3.3.1. Поразрядная обработка данных | 161 |
| 3.3.2. Обработка множеств | 162 |
| 3.3.3. Работа с функциями времени и даты | 165 |
| 3.4. Встроенные элементарные функции | 168 |
| 3.4.1. Алгебраические и арифметические функции | 168 |
| 3.4.2. Тригонометрические и обратные тригонометрические функции | 173 |
| 3.4.3. Вычисление гиперболических и обратных гиперболических функций | 177 |
| 3.5. Числовые функции | 180 |
| 3.5.1. Округление и смена знака чисел | 180 |
| 3.5.2. Операции с комплексными числами | 181 |
| 3.6. Специальные математические функции | 182 |
| 3.6.1. Функции Эйри | 182 |
| 3.6.2. Функции Бесселя | 183 |
| 3.6.3. Бета-функция и ее варианты | 186 |
| 3.6.4. Эллиптические функции и интегралы | 187 |
| 3.6.5. Функции ошибки | 188 |
| 3.6.6. Интегральная показательная функция | 189 |
| 3.6.7. Гамма-функция и ее варианты | 189 |
| 3.6.8. Ортогональные полиномы Лежандра | 190 |
| 3.6.9. Полигамма-функция ψ_i | 191 |

Урок 4. Операции с векторами

| | |
|---|-----|
| и матрицами | 193 |
| 4.1. Создание матриц с заданными свойствами | 194 |
| 4.1.1. Создание единичной матрицы | 194 |
| 4.1.2. Создание матрицы с единичными элементами | 194 |
| 4.1.3. Создание матрицы с нулевыми элементами | 195 |
| 4.1.4. Создание линейного массива равноотстоящих точек | 195 |
| 4.1.5. Создание вектора равноотстоящих в логарифмическом масштабе точек | 196 |
| 4.1.6. Создание массивов со случайными элементами ... | 196 |
| 4.1.7. Создание массивов с логическими значениями элементов | 198 |
| 4.2. Операции с матрицами | 199 |
| 4.2.1. Конкатенация матриц | 199 |
| 4.2.2. Создание матриц с заданной диагональю | 200 |
| 4.2.3. Перестановки элементов матриц | 201 |
| 4.2.4. Вычисление произведений | 201 |
| 4.2.5. Суммирование элементов массивов | 203 |
| 4.2.6. Функции формирования матриц | 204 |
| 4.2.7. Поворот матриц | 205 |
| 4.2.8. Выделение треугольных частей матриц | 205 |
| 4.2.9. Операции с пустыми матрицами | 206 |
| 4.3. Создание и вычисление специальных матриц | 207 |
| 4.3.1. Сопровождающие матрицы | 207 |
| 4.3.2. Тестовые матрицы | 207 |

| | |
|---|------------|
| 4.3.3. Матрицы Адамара | 208 |
| 4.3.4. Матрицы Ганкеля | 208 |
| 4.3.5. Матрицы Гильберта | 209 |
| 4.3.6. Матрицы магического квадрата | 210 |
| 4.3.7. Матрицы Паскаля | 210 |
| 4.3.8. Матрицы Россера | 211 |
| 4.3.9. Матрицы Теплица | 212 |
| 4.3.10. Матрица Вандермонда | 212 |
| 4.3.11. Матрицы Уилкинсона | 213 |
| 4.4. Матричные операции линейной алгебры | 213 |
| 4.4.1. Матричные функции | 213 |
| 4.4.2. Вычисление нормы и чисел обусловленности матрицы | 215 |
| 4.4.3. Определитель и ранг матрицы | 217 |
| 4.4.4. Определение нормы вектора | 217 |
| 4.4.5. Определение ортонормированного базиса матрицы | 218 |
| 4.4.6. Функции приведения матрицы к треугольной форме | 219 |
| 4.4.7. Определение угла между двумя подпространствами | 219 |
| 4.4.8. Вычисление следа матрицы | 220 |
| 4.4.9. Разложение Холецкого | 220 |
| 4.4.10. Обращение матриц – функции inv , pinv | 221 |
| 4.4.11. LU- и QR-разложение | 222 |
| 4.4.12. Вычисление собственных значений и сингулярных чисел | 225 |
| 4.4.13. Приведение матриц к форме Шура и Хессенберга | 227 |

- 4.5. О скорости выполнения матричных операций ... 231
 - 4.5.1. О повышении скорости вычислений в старых версиях MATLAB 231
 - 4.5.2. Ситуация со скоростью вычислений в MATLAB 7.* ... 231

Урок 5. Типы данных – массивы

- специального вида 233**
- 5.1. Разреженные матрицы 234
 - 5.1.1. Роль и назначение разреженных матриц 234
 - 5.1.2. Элементарные разреженные матрицы 234
 - 5.1.3. Преобразование разреженных матриц 237
 - 5.1.4. Работа с ненулевыми элементами разреженных матриц 240
 - 5.1.5. Функция `spru` визуализации разреженных матриц .. 241
 - 5.1.6. Алгоритмы упорядочения 242
- 5.2. Применение разреженных матриц 245
 - 5.2.1. Смежные матрицы и графы 245
 - 5.2.2. Пример построения фигуры `bucky` 246
 - 5.2.3. Оцифровка узлов графа 246
 - 5.2.4. Применение разреженных матриц в аэродинамике 247
 - 5.2.5. Визуализация разреженных матриц, возведенных в степень 249
 - 5.2.6. Демонстрационные примеры на визуализацию разреженных матриц 250
- 5.3. Функции разреженных матриц 250
 - 5.3.1. Норма, число обусловленности и ранг разреженной матрицы 250

| | |
|---|------------|
| 5.3.2. Функции разложения Холецкого для разреженных матриц | 252 |
| 5.3.3. LU-разложение разреженных матриц | 253 |
| 5.3.4. Собственные значения и сингулярные числа разреженных матриц | 255 |
| 5.4. Многомерные массивы | 255 |
| 5.4.1. Понятие о многомерных массивах | 255 |
| 5.4.2. Применение оператора : в многомерных массивах | 257 |
| 5.4.3. Удаление размерности у многомерного массива | 258 |
| 5.4.4. Доступ к отдельному элементу многомерного массива | 258 |
| 5.4.5. Создание страниц, заполненных константами и случайными числами | 259 |
| 5.4.6. Функции ones, zeros, rand и randn | 259 |
| 5.4.7. Объединение многомерных массивов | 260 |
| 5.4.8. Функция преобразования размеров многомерного массива reshape | 261 |
| 5.5. Работа с размерностями массивов | 262 |
| 5.5.1. Вычисление числа размерностей массива | 262 |
| 5.5.2. Перестановки размерностей массивов | 262 |
| 5.5.3. Сдвиг размерностей массивов | 263 |
| 5.5.4. Удаление единичных размерностей | 264 |
| 5.6. Тип данных – структуры | 264 |
| 5.6.1. Структура записей | 264 |
| 5.6.2. Создание структур и доступ к их компонентам | 265 |
| 5.6.3. Функция создания структур | 266 |
| 5.6.4. Проверка имен полей и структур | 266 |
| 5.7. Функции полей структур | 267 |

| | |
|--|------------|
| 5.7.1. Функция возврата имен полей | 267 |
| 5.7.2. Функция возврата содержимого полей структуры ... | 267 |
| 5.7.3. Функция присваивания значений полям | 268 |
| 5.7.4. Удаление полей | 268 |
| 5.7.5. Применение массивов структур | 268 |
| 5.8. Массивы ячеек | 269 |
| 5.8.1. Создание массивов ячеек | 269 |
| 5.8.2. Создание ячеек с помощью функции cell | 270 |
| 5.8.3. Визуализация массивов ячеек | 271 |
| 5.8.4. Создание массива символьных ячеек из массива строк | 271 |
| 5.8.5. Присваивание с помощью функции deal | 272 |
| 5.8.6. Тестирование имен массивов ячеек | 273 |
| 5.8.7. Функции преобразования типов данных | 273 |
| 5.9. Многомерные массивы ячеек | 274 |
| 5.9.1. Создание многомерных массивов ячеек | 274 |
| 5.9.2. Вложенные массивы ячеек | 275 |

Урок 6. Программные средства обычной графики

277

| | |
|---|------------|
| 6.1. Графики функций и данных | 278 |
| 6.1.1. Построение графиков отрезками прямых | 278 |
| 6.1.2. Графики в логарифмическом масштабе | 282 |
| 6.1.3. Графики в полулогарифмическом масштабе | 283 |
| 6.1.4. Столбцовые диаграммы | 284 |
| 6.1.5. Гистограммы | 285 |

| | |
|---|------------|
| 6.1.6. Лестничные графики | 286 |
| 6.1.7. Графики с зонами погрешности | 287 |
| 6.1.8. Графики дискретных отсчетов функции | 288 |
| 6.2. Визуализация в полярной системе координат | 289 |
| 6.2.1. Графики в полярной системе координат | 289 |
| 6.2.2. Угловые гистограммы | 290 |
| 6.3. Визуализация векторов | 291 |
| 6.3.1. Графики векторов | 291 |
| 6.3.2. График проекций векторов на плоскость | 292 |
| 6.4. Основы трехмерной графики | 293 |
| 6.4.1. Контурные графики | 293 |
| 6.4.2. Создание массивов данных для трехмерной графики | 294 |
| 6.4.3. Графики поля градиентов | 296 |
| 6.4.4. Графики поверхностей | 297 |
| 6.4.5. Сетчатые 3D-графики с окраской | 298 |
| 6.4.6. Сетчатые 3D-графики с проекциями | 301 |
| 6.4.7. Построение поверхности столбцами | 301 |
| 6.5. Улучшенные средства визуализации 3D-графики | 302 |
| 6.5.1. Построение поверхности с окраской | 302 |
| 6.5.2. Построение поверхности и ее проекции | 305 |
| 6.5.3. Построение освещенной поверхности | 306 |
| 6.5.4. Средства управления подсветкой и обзором фигур | 307 |

| | |
|---|------------|
| 6.5.5. Построение графиков функций трех переменных .. | 308 |
| 6.5.6. График трехмерной слоеной поверхности | 310 |
| 6.5.7. Трехмерные контурные графики | 310 |
| 6.6. Текстовое оформление графиков | 312 |
| 6.6.1. Установка титульной надписи | 312 |
| 6.6.2. Установка осевых надписей | 312 |
| 6.6.3. Ввод текста в любое место графика | 312 |
| 6.6.4. Позиционирование текста с помощью мыши | 315 |
| 6.7. Форматирование графиков | 316 |
| 6.7.1. Вывод пояснений и легенды | 316 |
| 6.7.2. Маркировка линий уровня на контурных графиках... | 319 |
| 6.7.3. Управление свойствами осей графиков | 319 |
| 6.7.4. Включение и выключение сетки | 321 |
| 6.7.5. Наложение графиков друг на друга | 322 |
| 6.7.6. Разбиение графического окна | 324 |
| 6.7.7. Изменение масштаба графика | 324 |
| 6.8. Цветовая окраска графиков | 327 |
| 6.8.1. Установка палитры цветов | 327 |
| 6.8.2. Установка соответствия между палитрой цветов и масштабom осей | 328 |
| 6.8.3. Окраска поверхностей | 328 |
| 6.8.4. Установка палитры псевдоцветов | 329 |
| 6.8.5. Создание закрашенного многоугольника | 330 |
| 6.8.6. Окраска плоских многоугольников | 331 |
| 6.8.7. Вывод шкалы цветов | 332 |
| 6.8.8. Цветные плоские круговые диаграммы | 333 |

| | |
|---|------------|
| 6.8.9. Окрашенные многоугольники в пространстве | 334 |
| 6.8.10. Цветные объемные круговые диаграммы | 335 |
| 6.8.11. Другие команды управления световыми эффектами | 335 |
| 6.9. Другие возможности графики | 336 |
| 6.9.1. Построение цилиндра | 336 |
| 6.9.2. Построение сферы | 337 |
| 6.9.3. 3D-графика с треугольными плоскостями | 338 |

Урок 7. Программные средства специальной графики

341

| | |
|--|------------|
| 7.1. Анимационная графика | 342 |
| 7.1.1. Движение точки на плоскости | 342 |
| 7.1.2. Движение точки в пространстве | 342 |
| 7.1.3. Основные средства анимации | 344 |
| 7.1.4. Вращение фигуры – логотипа MATLAB | 344 |
| 7.1.5. Волновые колебания мембраны | 345 |
| 7.2. Основы дескрипторной графики | 347 |
| 7.2.1. Объекты дескрипторной графики | 347 |
| 7.2.2. Создание графического окна и управление им | 347 |
| 7.2.3. Создание координатных осей и управление ими ... | 348 |
| 7.2.4. Пример применения объекта дескрипторной графики | 348 |
| 7.2.5. Дескрипторы объектов | 348 |
| 7.2.6. Операции над графическими объектами | 350 |
| 7.2.7. Свойства объектов – команда get | 351 |
| 7.2.8. Изменение свойств объекта – команда set | 352 |

| | |
|---|-----|
| 7.2.9. Просмотр свойств | 352 |
| 7.2.10. Примеры дескрипторной графики | 353 |
| 7.2.11. Иерархия объектов дескрипторной графики | 356 |
| 7.2.12. Справка по дескрипторной графике | 357 |
| 7.3. Галерея трехмерной графики | 357 |
| 7.3.1. Доступ к галерее | 357 |
| 7.3.2. Примеры построения фигур из галереи | 359 |
| 7.4. Графический интерфейс пользователя GUI | 362 |
| 7.4.1. Основные команды для создания GUI | 362 |
| 7.4.2. Простой пример создания объектов GUI | 364 |
| 7.4.3. Примеры программирования GUI | 365 |
| 7.4.4. Программирование анимации поверхности с разной скоростью | 366 |
| 7.4.5. Программирование визуализации звукового сигнала | 367 |
| 7.5. Графическая поддержка цвета | 369 |
| 7.5.1. Цветовые системы и OpenGL | 369 |
| 7.5.2. Управление прозрачностью графических объектов | 370 |
| 7.5.3. Примеры построения изображений со свойствами прозрачности | 370 |
| 7.6. Расширенная техника визуализации вычислений | 372 |
| 7.6.1. Задание Path-объектов | 372 |
| 7.6.2. Построение среза черепной коробки человека | 373 |
| 7.6.3. Расширенная визуализация трехмерных объектов | 375 |

| | |
|---|-----|
| 7.6.4. Выделение части объема | 375 |
| 7.6.5. Визуализация струи в пространстве | 377 |
| 7.6.6. Визуализация электрических разрядов | 378 |
| 7.6.7. Анимация явления подъема предметов вихрями ... | 379 |
| 7.6.8. Применение «конусной» графики для визуализации струй..... | 381 |

Урок 8. Программные средства

численных методов 383

8.1. Решение систем линейных уравнений (СЛУ) 384

8.1.1. Элементарные средства

 384

8.1.2. Решение систем линейных уравнений с ограничениями

 386

8.1.3. Решение систем линейных уравнений с комплексными элементами

 387

8.2. Решение СЛУ с разреженными матрицами 388

8.2.1. Точное решение, метод наименьших квадратов и сопряженных градиентов

 388

8.2.2. Двухнаправленный метод сопряженных градиентов

 390

8.2.3. Устойчивый двухнаправленный метод

 392

8.2.4. Метод сопряженных градиентов

 392

8.2.5. Квадратичный метод сопряженных градиентов

 393

8.2.6. Метод минимизации обобщенной невязки

 393

8.2.7. Квазиминимизация невязки – функция qmr

 394

8.3. Вычисление корней функций 394

8.3.1. Вычисление корней функций одной переменной ...

 394

8.3.2. Графическая иллюстрация поиска корней функций

 395

| | |
|---|------------|
| 8.3.3. Поиск корня с помощью функций <code>fsolve</code> и <code>solve</code> | 396 |
| 8.3.4. Решение систем нелинейных уравнений | 397 |
| 8.4. Вычисление минимумов функций | 398 |
| 8.4.1. Минимизация функции одной переменной | 398 |
| 8.4.2. Минимизация функций ряда переменных симплекс-методом | 398 |
| 8.4.3. Минимизация тестовой функции Розенброка | 400 |
| 8.4.4. Другие средства минимизации функций нескольких переменных | 400 |
| 8.5. Аппроксимация производных | 403 |
| 8.5.1. Аппроксимация лапласиана | 403 |
| 8.5.2. Аппроксимация производных конечными разностями | 404 |
| 8.5.3. Вычисление градиента функции | 407 |
| 8.6. Численное интегрирование | 408 |
| 8.6.1. Интегрирование методом трапеций | 408 |
| 8.6.2. Интегрирование методом квадратур | 409 |
| 8.6.3. Вычисления двойных и тройных интегралов | 410 |
| 8.7. Математические операции с полиномами | 411 |
| 8.7.1. Определение полиномов | 411 |
| 8.7.2. Умножение и деление полиномов | 412 |
| 8.7.3. Вычисление полиномов | 412 |
| 8.7.4. Вычисление корней полинома | 414 |
| 8.7.5. Вычисление производной полинома | 415 |
| 8.7.6. Решение полиномиальных матричных уравнений | 415 |
| 8.7.7. Разложение полиномов на простые дроби | 416 |

| | |
|---|-----|
| 8.8. Обыкновенные дифференциальные уравнения (ОДУ) | 416 |
| 8.8.1. Определение ОДУ | 416 |
| 8.8.2. Решатели ОДУ | 417 |
| 8.8.3. Использование решателей систем ОДУ | 419 |
| 8.9. Примеры решения дифференциальных уравнений | 422 |
| 8.9.1. Пример на движение брошенного вверх тела | 422 |
| 8.9.2. Примеры решения системы ОДУ Ван-дер-Поля | 423 |
| 8.9.3. Вычисление реакции системы второго порядка на заданное воздействие | 426 |
| 8.9.4. Решение уравнений Лотки–Вольтерра двумя методами | 427 |
| 8.9.5. Решение системы Лотки–Вольтерра с запаздывающим аргументом | 429 |
| 8.9.6. Решение системы дифференциальных уравнений с двухсторонними граничными условиями | 431 |
| 8.9.7. Моделирование странного аттрактора Лоренца | 432 |
| 8.9.8. Решение жесткой алгебраически-дифференциальной системы уравнений | 433 |
| 8.9.9. Доступ к примерам на решение дифференциальных уравнений | 435 |
| 8.9.10. Решения дифференциальных уравнений в частных производных | 437 |

Урок 9. Программные средства обработки данных

441

| | |
|--------------------------------------|-----|
| 9.1. Обработка данных массивов | 442 |
|--------------------------------------|-----|

| | |
|--|-----|
| 9.1.1. Нахождение максимального и минимального элементов массива | 442 |
| 9.1.2. Сортировка элементов массива | 443 |
| 9.1.3. Нахождение средних и срединных значений | 445 |
| 9.1.4. Вычисление стандартного отклонения | 447 |
| 9.1.5. Вычисление коэффициентов корреляции | 447 |
| 9.1.6. Вычисление матрицы ковариации | 448 |
| 9.2. Геометрический анализ данных | 449 |
| 9.2.1. Триангуляция Делоне | 449 |
| 9.2.2. Вычисление выпуклой оболочки | 450 |
| 9.2.3. Вычисление площади полигона | 451 |
| 9.2.4. Анализ попадания точек внутрь полигона | 452 |
| 9.2.5. Построение диаграммы Вороного | 453 |
| 9.3. Преобразование Фурье | 454 |
| 9.3.1. Основные определения | 454 |
| 9.3.2. Одномерное прямое быстрое преобразование Фурье | 455 |
| 9.3.3. Многомерное прямое преобразование Фурье | 457 |
| 9.3.4. Перегруппировка массивов | 458 |
| 9.3.5. Одномерное быстрое обратное преобразование Фурье | 459 |
| 9.4. Свертка и дискретная фильтрация | 460 |
| 9.4.1. Свертка прямая и обратная | 460 |
| 9.4.2. Свертка двумерных массивов | 461 |
| 9.4.3. Дискретная одномерная фильтрация | 461 |
| 9.4.4. Двумерная фильтрация | 464 |
| 9.4.5. Коррекции фазовых углов | 464 |

| | |
|--|-----|
| 9.5. Интерполяция и аппроксимация данных | 465 |
| 9.5.1. Полиномиальная регрессия | 465 |
| 9.5.2. Фурье-интерполяция периодических функций | 467 |
| 9.5.3. Интерполяция на неравномерной сетке | 467 |
| 9.5.4. Одномерная табличная интерполяция | 469 |
| 9.5.5. Двумерная табличная интерполяция | 470 |
| 9.5.6. Трехмерная табличная интерполяция | 472 |
| 9.5.7. N-мерная табличная интерполяция | 473 |
| 9.5.8. Интерполяция кубическим сплайном | 474 |
| 9.6. Специальные виды интерполяции | 475 |
| 9.6.1. Сравнение видов двумерной интерполяции поверхности | 475 |
| 9.6.2. Сравнение видов интерполяции при контурных графиках | 478 |
| 9.6.3. Пример многомерной интерполяции | 479 |
| 9.6.4. 3D-геометрический анализ и интерполяция | 479 |
| 9.6.5. Другие представления сложных фигур | 482 |
| 9.7. Обработка данных в графическом окне | 484 |
| 9.7.1. Доступ к средствам обработки данных в графическом окне | 484 |
| 9.7.2. Полиномиальная регрессия для табличных данных ... | 485 |
| 9.7.3. Оценка погрешности аппроксимации | 487 |
| 9.7.4. Расширенные возможности окна приближения кривых | 488 |
| 9.7.5. Сплайновая и эрмитовая интерполяции в графическом окне | 490 |
| 9.7.6. Графическая визуализация разложения в ряд Тейлора | 492 |

| | |
|---|-----|
| Урок 10. Работа со строками, файлами и звуками | 493 |
| 10.1. Обработка строковых данных | 494 |
| 10.1.1. Основные функции обработки строк | 494 |
| 10.1.2. Операции над строками | 496 |
| 10.1.3. Преобразование символов и строк | 500 |
| 10.1.4. Функции преобразования систем счисления | 502 |
| 10.1.5. Вычисление строковых выражений | 503 |
| 10.2. Работа с файлами | 504 |
| 10.2.1. Открытие и закрытие файлов | 505 |
| 10.2.2. Операции с двоичными файлами | 507 |
| 10.2.3. Операции над форматированными файлами | 510 |
| 10.2.4. Позиционирование файла | 514 |
| 10.2.5. Специализированные файлы | 517 |
| 10.3. Работа с файлами изображений | 518 |
| 10.3.1. Информация о графическом файле – imfinfo | 518 |
| 10.3.2. Чтение изображения из файла – imread | 520 |
| 10.3.3. Запись изображения в файл – imwrite | 522 |
| 10.4. Работа со звуковыми данными | 526 |
| 10.4.1. Функции для работы со звуками | 526 |
| 10.4.2. Функции звука в MATLAB 6.1/6.5 | 526 |
| 10.4.3. Демонстрация возможностей работы со звуком ... | 527 |
| | |
| Урок 11. Типовые средства программирования | 531 |
| 11.1. Основные понятия программирования | 532 |

| | |
|---|-----|
| 11.1.1. Назначение языка программирования MATLAB | 532 |
| 11.1.2. Основные средства программирования | 533 |
| 11.1.3. Основные типы данных | 533 |
| 11.1.4. Виды программирования | 535 |
| 11.1.5. Двойственность операторов, команд и функций ... | 536 |
| 11.1.6. Некоторые ограничения | 537 |
| 11.1.7. Исполнение программных объектов | 538 |
| 11.2. М-файлы сценариев и функций | 538 |
| 11.2.1. Структура и свойства файлов-сценариев | 538 |
| 11.2.2. Структура М-файла-функции | 540 |
| 11.2.3. Статус переменных в функциях | 541 |
| 11.2.4. Команда глобализации переменных global | 543 |
| 11.2.5. Использование подфункций | 543 |
| 11.2.6. Частные каталоги | 544 |
| 11.3. Обработка ошибок и комментарии | 545 |
| 11.3.1. Вывод сообщений об ошибках | 545 |
| 11.3.2. Функция lasterr и обработка ошибок | 546 |
| 11.3.3. Комментарии | 547 |
| 11.4. Функции с переменным числом аргументов | 547 |
| 11.4.1. Функции подсчета числа аргументов | 547 |
| 11.4.2. Переменные varargin и varargout | 549 |
| 11.5. Особенности работы с m-файлами | 550 |
| 11.5.1. Выполнение m-файлов-функций | 550 |
| 11.5.2. Создание Р-кодов | 551 |
| 11.6. Управляющие структуры | 552 |

| | |
|--|------------|
| 11.6.1. Диалоговый ввод | 552 |
| 11.6.2. Условный оператор if...elseif...else...end | 553 |
| 11.6.3. Циклы типа for...end | 554 |
| 11.6.4. Циклы типа while...end | 556 |
| 11.6.5. Конструкция переключателя switch...case...end ... | 558 |
| 11.6.6. Конструкция try...catch...end | 559 |
| 11.6.7. Операторы break, continue и return | 560 |
| 11.6.8. Пустые матрицы в структурах if и while | 561 |
| 11.6.9. Создание паузы в вычислениях | 561 |
| 11.7. Основы объектно-ориентированного программирования | 561 |
| 11.7.1. Основные понятия | 561 |
| 11.7.2. Классы объектов | 562 |
| 11.7.3. Создание класса или объекта | 563 |
| 11.7.4. Проверка принадлежности объекта к заданному классу | 564 |
| 11.7.5. Другие функции объектно-ориентированного программирования | 564 |
| 11.8. Handle- и inline-функции | 565 |
| 11.8.1. Задание handle-функции | 565 |
| 11.8.2. Вычисление и применение handle-функций | 566 |
| 11.8.3. Inline-функции | 566 |
| 11.8.4. Преобразования handle- и inline-функций | 567 |
| 11.9. Отладка программ | 567 |
| 11.9.1. Общие замечания по отладке m-файлов | 567 |
| 11.9.2. Команды отладки программ | 568 |
| 11.9.3. Вывод листинга m-файла с пронумерованными строками | 568 |

| | |
|---|------------|
| 11.9.4. Установка, удаление и просмотр точек прерывания | 569 |
| 11.9.5. Управление исполнением m-файла | 570 |
| 11.9.6. Просмотр рабочей области | 570 |
| 11.9.7. Профилирование m-файлов | 571 |
| 11.9.8. Создание итогового отчета | 572 |
| 11.9.9. Построение диаграмм Парето | 574 |
| 11.9.10. Работа с системой контроля версий | 574 |
| 11.10. Профилирование программ в MATLAB 7 | 575 |
| 11.10.1. Утилита профилирования программ Profiler и ее запуск | 575 |
| 11.10.2. Пример профилирования программы | 576 |
| 11.10.3. Профилирование избранных функций программы | 577 |
| 11.10.4. Профилирование строк общего программного кода | 577 |
| 11.11. Общение MATLAB с операционной системой .. | 579 |
| 11.11.1. Работа с папками | 579 |
| 11.11.2. Выполнение команд !, dos, unix и vms | 580 |
| 11.11.3. Общение с Интернетом из командной строки | 580 |
| 11.11.4. Некоторые другие команды | 581 |
| 11.12. Поддержка Java | 582 |
| 11.12.1. Информация о средствах поддержки Java | 582 |
| 11.12.2. Java-объекты | 582 |
| 11.12.3. Специфика применения Java-объектов | 584 |
| 11.12.4. Java-массивы | 586 |
| 11.13. Компиляция MATLAB-программ | 587 |
| 11.13.1. Для чего нужна компиляция MATLAB-программ | 587 |

| | |
|--|-----|
| 11.13.2. Конфигурирование расширения MATLAB Compiler | 587 |
| 11.13.3. Компиляция m-файла-функции | 588 |
| 11.13.4. Исполнение откомпилированного файла | 589 |

Урок 12. Визуальное программирование

| | |
|---|-----|
| GUI | 591 |
| 12.1. Средства визуального программирования GUIDE | 592 |
| 12.1.1. Состав и назначение средств программирования GUIDE | 592 |
| 12.1.2. Открытие окна инструмента GUIDE | 593 |
| 12.1.3. Окно создания нового приложения с GUI | 594 |
| 12.1.4. Свойства объектов GUI | 597 |
| 12.1.5. Пример задания кнопки и работа с инспектором свойств объектов | 600 |
| 12.1.6. Вид всех компонентов и редактирование их свойств | 602 |
| 12.2. Работа с заготовками примеров | 604 |
| 12.2.1. Простой пример вычисления массы вещества | 604 |
| 12.2.2. Пример на построение графиков из списка | 609 |
| 12.3. Детальная работа с инструментом GUIDE | 612 |
| 12.3.1. Установка опций окна компонентов | 612 |
| 12.3.2. Работа с меню File | 614 |
| 12.3.3. Ввод компонентов и их редактирование | 615 |
| 12.3.4. Средства обзора приложения | 618 |
| 12.3.5. Операции разметки объектов | 619 |
| 12.3.6. Операции позиции Tools меню | 620 |

| | |
|---|------------|
| 12.3.7. Конструирование меню окна приложения с GUI ... | 622 |
| 12.3.8. Конструирование контекстного меню окна приложения с GUI | 629 |
| 12.3.9. Применение рамки и группы кнопок | 634 |
| 12.3.10. Интерпретация программы приложения | 639 |
| 12.3.11. Несколько советов по созданию приложений с GUI | 641 |
| 12.4. Стандартные диалоговые окна MATLAB | 642 |
| 12.4.1. Набор диалоговых окон | 642 |
| 12.4.2. Справка по диалоговым окнам и их свойства | 643 |
| 12.4.3. Работа с простыми диалоговыми окнами | 644 |
| 12.4.4. Диалоговые окна множественного типа | 645 |
| 12.4.5. Диалоговые окна файловых операций | 647 |
| 12.4.6. Диалоговые окна установки цвета и шрифтов | 649 |
| 12.4.7. Диалоговые окна параметров страницы и печати | 650 |
| 12.4.8. Другие диалоговые окна | 654 |
| | |
| Урок 13. Обзор расширений MATLAB | 659 |
| 13.1. Состав расширений MATLAB | 660 |
| 13.1.1. Классификация расширений системы MATLAB+Simulink | 660 |
| 13.1.2. Главный пакет расширения Simulink 5/6 | 660 |
| 13.2. Примеры работы с Simulink | 662 |
| 13.2.1. Пример моделирования системы Ван-дер-Поля ... | 662 |
| 13.2.2. Nonlinear Control Design Blockset | 663 |
| 13.2.3. Digital Signal Processing (DSP) Blockset | 665 |
| 13.2.4. Пакет расширения Fixed-Point Blockset | 667 |

| | |
|--|------------|
| 13.2.5. Пакет расширения Stateflow | 667 |
| 13.2.6. Пакет расширения SimPower System | 668 |
| 13.2.7. Report Generator для MATLAB и Simulink | 669 |
| 13.2.8. Real Time Windows Target и WorkShop | 670 |
| 13.3. Пакеты математических вычислений..... | 670 |
| 13.3.1. Symbolic Math Toolbox..... | 671 |
| 13.3.2. NAG Foundation Toolbox | 671 |
| 13.3.3. Spline Toolbox | 672 |
| 13.3.4. Statistics Toolbox | 673 |
| 13.3.5. Optimization Toolbox | 674 |
| 13.3.6. Partial Differential Equations Toolbox | 675 |
| 13.3.7. Fuzzy Logic Toolbox | 677 |
| 13.3.8. Neural Networks Toolbox | 678 |
| 13.4. Пакеты анализа и синтеза систем управления .. | 680 |
| 13.4.1. Control System Toolbox | 680 |
| 13.4.2. Robust Control Toolbox | 682 |
| 13.4.3. Model Predictive Control Toolbox | 684 |
| 13.4.4. Communications Toolbox..... | 685 |
| 13.4.5. m-Analysis and Synthesis | 685 |
| 13.4.6. Quantitative Feedback Theory Toolbox..... | 686 |
| 13.4.7. LMI Control Toolbox | 686 |
| 13.5. Пакет идентификации систем | 687 |
| 13.6. Пакеты для обработки сигналов и изображений | 689 |
| 13.6.1. Signal Processing Toolbox | 689 |
| 13.6.2. Image Processing Toolbox | 691 |
| 13.6.3. Wavelet Toolbox | 695 |

| | |
|---|-----|
| 13.7. Прочие пакеты прикладных программ | 699 |
| 13.7.1. Financial Toolbox | 699 |
| 13.7.2. Mapping Toolbox | 700 |
| 13.7.3. Data Acquisition Toolbox и Instrument Control Toolbox | 701 |
| 13.7.4. Database toolbox | 703 |
| 13.7.5. Excel Link | 703 |
| 13.7.6. Virtual Reality Toolbox..... | 703 |
| 13.7.7. MATLAB Compiler | 704 |
| 13.8. Пакеты расширения MATLAB 6.5 | 704 |
| 13.8.1. Curve Fitting Toolbox | 704 |
| 13.8.2. Instrument Control Toolbox | 706 |
| 13.8.3. Developer’s Kit for Texas Instruments DSP | 707 |
| 13.8.4. Dials & Gauges Blockset | 708 |
| 13.8.5. Mechanical System Blockset..... | 708 |
| 13.9. Новейшие пакеты расширения MATLAB 7+Simulink 6 | 709 |
| 13.9.1. Назначение и возможности пакета Bioinformatics Toolbox | 709 |
| 13.9.2. Пакет расширения Genetic Algorithm and Direct Search Toolbox | 711 |
| 13.9.3. Пакет расширения Video and Image Processing Blockset | 713 |

| | |
|--|------------|
| Урок 14. Стыковка MATLAB с измерительными приборами | 717 |
| 14.1. Работа измерительных приборов с системой MATLAB | 718 |

| | |
|---|------------|
| 14.1.1. Современные измерительные приборы | 718 |
| 14.1.2. Порты для подключения измерительных приборов к компьютеру | 719 |
| 14.2. Стыковка компьютера с цифровым осциллографом | 721 |
| 14.2.1. Современные цифровые осциллографы с USB-портом | 721 |
| 14.2.2. Применение пакета расширения MATLAB – Instrument Control Toolbox | 722 |
| 14.2.3. Идентификация осциллографа | 723 |
| 14.2.4. MATLAB-программы для работы с цифровыми осциллографами | 724 |
| 14.2.5. Спектральный анализ осциллограмм в MATLAB ... | 728 |
| 14.2.6. Построение спектрограмм осциллограмм в MATLAB | 732 |
| 14.3. Управление генераторами произвольных сигналов от системы MATLAB | 735 |
| 14.3.1. От множества генераторов к одному генератору произвольных сигналов | 735 |
| 14.3.2. Управление генераторами серии AFG3000 от системы MATLAB | 736 |
| 14.4. Применение MATLAB при совместной работе генератора и цифрового осциллографа | 739 |
| 14.5. Встраивание MATLAB в осциллографы, построенные на основе платформы ПК | 741 |
| Список литературы | 743 |
| Предметный указатель | 747 |

Введение

Среди бурно развивающихся систем компьютерной математики СКМ [1], в первую очередь ориентированных на численные расчеты, особо выделяется матричная математическая система MATLAB. Из-за большого числа поставляемых с системой пакетов расширения MATLAB (в новейшей реализации MATLAB R2007a,b их уже 82) эта система является и самой большой из СКМ, ориентированных на персональные компьютеры. Объем ее файлов уже превышает 3 Гб. Система фактически стала мировым стандартом в области современного математического и научно-технического программного обеспечения.

Эффективность MATLAB обусловлена прежде всего ее ориентацией на матричные вычисления [2, 3] с программной эмуляцией параллельных вычислений и упрощенными средствами задания циклов. Последние версии системы поддерживают 64-разрядные микропроцессоры и многоядерные микропроцессоры, например Intel Core 2 Duo и Quad, что обеспечивает высочайшие показатели по скорости вычислений и скорости математического имитационного моделирования.

В MATLAB удачно реализованы средства работы с многомерными массивами, большими и разреженными матрицами и многими типами данных. Система прошла многолетний путь развития от узко специализированного матричного программного модуля, используемого только на больших ЭВМ, до универсальной интегрированной СКМ, ориентированной на массовые персональные компьютеры класса IBM PC, AT и Macintosh, рабочие станции UNIX и даже суперкомпьютеры. MATLAB имеет мощные средства диалога, графики и *комплексной визуализации* вычислений.

Система MATLAB предлагается разработчиками (корпорация The MathWorks, Inc.) как лидирующий на рынке, в первую очередь на предприятиях военно-промышленного комплекса, в энергетике, в аэрокосмической отрасли и в автомобилестроении язык программирования *высокого уровня* для технических вычислений, расширяемый большим числом пакетов прикладных программ – *расширений*. Самым известным из них стало расширение Simulink, обеспечивающее блочное имитационное моделирование различных систем и устройств. Но и без пакетов расширения MATLAB представляет собой мощную операционную среду для выполнения огромного числа математических и научно-технических расчетов и вычислений и создания пользователями своих пакетов расширения и библиотек процедур и функций. Новые версии системы имеют встроенный компилятор и позволяют создавать исполняемые файлы.

Типовой комплекс MATLAB + Simulink (рис. 0.1) содержит инструментальные «ящики» Toolboxes с большим числом пакетов расширения MATLAB и Blocksets для расширения возможностей системы визуально-ориентированного блочного имитационного моделирования динамических систем Simulink. Они приобретаются избранно и отдельно от системы MATLAB + Simulink. В разработке пакетов расширения для MATLAB принимают участие многие научные школы мира и ве-

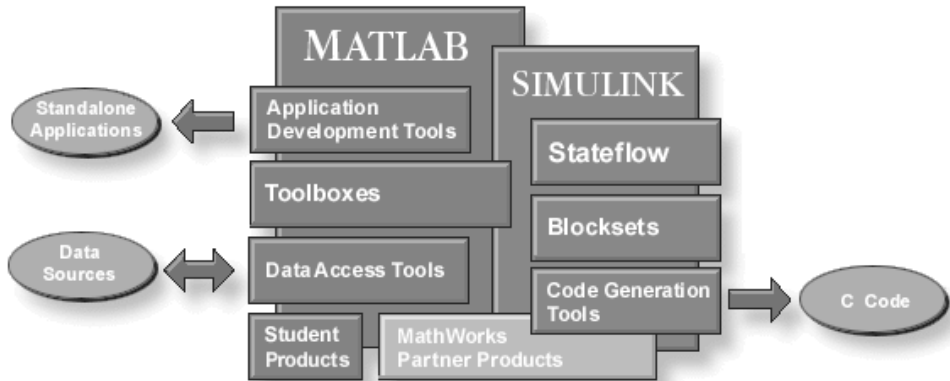


Рис. 0.1. Структура системы MATLAB + Simulink

дущие университеты. Многие пакеты охватывают крупные направления науки и техники, такие как оптимизация отклика нелинейных систем, моделирование устройств и систем механики и энергетики, обработка сигналов и изображений, вейвлеты, биоинформатика, генные алгоритмы, нечеткая логика, нейронные сети и т. д.

В России первой книгой по системе MATLAB стала небольшая книга автора [4], выпущенная еще в 1993 г. В последующие полтора десятка лет было опубликовано множество книг по различным версиям этой мощной и бурно развивающейся системы [5–49]. Так, только на Web-узле корпорации The MathWorks, Inc., разработавшей эту систему, указано уже более 1000 книг. Среди них есть и книги автора (рис. 0.2), подготовленные в рамках программы поддержки подготовки книг корпорации The MathWorks, Inc. (далее просто The MathWorks).

К сожалению, в России мало публикаций по новейшим версиям системы MATLAB и ее расширению Simulink. Лишь после 1997 г. появился ряд книг по системам MATLAB и отдельным пакетам расширения этой системы [4–43]. Из них следует особо отметить трехтомник автора [13–15] по версиям MATLAB 6.* и пятитомник по MATLAB 6.5 и (обзорно) по MATLAB 7.0 [16–20]. Последние версии системы относятся к классу систем MATLAB 7.*, так как система MATLAB 7.5 имеет второе название MATLAB R2007b. Это указывает на то, что системы MATLAB 7.* имеют куда общие возможности, чем различия, что и делает целесообразным описание их в одной книге.

Увы, но объем книг по системе MATLAB и пакетам ее расширения непрерывно растет, как и их стоимость. Достаточно отметить, что последние книги только по системе MATLAB 7.0 (без пакетов расширения) учебного характера имеют объем более 1100 [28] и более 750 [29] страниц. Пять упомянутых томов автора по системе MATLAB с пакетами расширения насчитывают уже более 2800 страниц [16–20] и, будучи ориентированными на профессионалов, выпущены в твердом переплете небольшим тиражом. Разумеется, такие книги довольно дороги и не очень доступны.

| | |
|--------------------------|--|
| Dutch | Mathematical Packages of Expansion for MATLAB: Special Handbook Dyakonov / Kruglov Piter, 2001 |
| Finnish | |
| French | |
| German | MATLAB 5 with Packages of Expansion Dyakonov / Abramenkova / Kruglov Knowledge, 2001 |
| Greek | |
| Hungarian | |
| Italian | MATLAB 6.5 SP1/7.0 and Simulink 5/6: Bases of Application Dyakonov Solon-R, 2005 |
| Japanese | |
| Korean | |
| Norwegian | MATLAB 6.5 SP1/7.0 and Simulink 5/6: Mathematics and Modeling Dyakonov Solon-R, 2005 |
| Polish | |
| Portuguese | |
| Romanian | MATLAB 6.5 SP1/7.0 and Simulink 5/6: Signal Processing and Filter Design Dyakonov Solon-R, 2005 |
| Russian | |
| Serbian | |
| Slovak | MATLAB 6.5 SP1/7/7 SP1 and Simulink 5/6: Working with Images and Video Dyakonov Solon-R, 2005 |
| Spanish | |
| Swedish | |
| Turkish | MATLAB 6.5SP1/7/7 SP1/7 SP2 and Simulink 5/6: Artificial Intelligence and Bioinformatic Instruments Dyakonov / Kruglov SOLON-Press, 2006 |
| Ukrainian | |
| Vietnamese | |
| Join Book Program | |
| | MATLAB 6/6.1/6.5 and Simulink 4/5 in Mathematics and Simulation Dyakonov Solon-R, 2003 |
| | MATLAB 6/6.1/6.5 and Simulink 4/5: Principles of Use Dyakonov Solon-R, 2002 |
| | MATLAB 6: An Educational Course |

Рис. 0.2. Книги автора по системе MATLAB и пакетам ее расширения на интернет-сайте корпорации The MathWorks

Фирменная документация по системе (англоязычная) представлена уже многими десятками книг, например [44–49]. Она настолько разрослась, что разработчики MATLAB были вынуждены прекратить поставки ее в виде PDF-файлов на отдельном CD-ROM (и даже DVD в последних версиях MATLAB R2007a,b) и разместили ее на своем интернет-сайте. Однако из-за большого объема ее файлов скачать документацию весьма проблематично даже для тех наших пользователей, которые имеют доступ в Интернет. Кроме того, вся фирменная документация англоязычная и труднодоступна для чтения и перевода, так как содержит огромное число специальных англоязычных терминов, перевод которых отнюдь не прост и не тривиален.

Все это делает книги по MATLAB доступными лишь для малой части наших инженеров, научных работников, аспирантов, студентов и преподавателей университетов и вузов. Достаточно отметить, что стоимость всего одной книги среднего объема по системе MATLAB в России составляет примерно трехмесячную выплату на приобретение литературы, предусмотренную в государственных вузах и университетах Российской Федерации для преподавателей.

Такая мощная система, как MATLAB, должна быть отражена в книгах различного толка и стиля: справочниках, руководствах пользователя, учебных изданиях,

монографиях и т. д. И автор многие годы старался подготовить книги по MATLAB разного назначения и стиля.

Но особенно желательным является издание серии относительно небольших учебных курсов и самоучителей по системе MATLAB и ее приложениям. Что касается учебных курсов, то, несмотря на полезный опыт подготовки их по старым версиям MATLAB [7, 8], почвы для такого рода книг в России пока нет. Это связано с тем, что в подавляющем большинстве наших вузов и университетов в их учебных программах изучение специальных курсов по MATLAB не предусмотрено. А в курсах по численным методам и математическому моделированию чаще всего предполагается, что система MATLAB будет изучаться студентами самостоятельно или в ходе выполнения вычислительной практики. В самостоятельном изучении MATLAB заинтересованы также инженеры, научные работники, аспиранты и преподаватели университетов и вузов. Именно поэтому подготовка серии самоучителей по такой мощной системе, как MATLAB, представляется весьма современной и наиболее полезной в наше время.

Данная книга первая в этой серии книг. Она основана на материале книг [8, 16], но посвящена только базовой системе MATLAB (вторая книга этой серии будет посвящена пакету расширения Simulink). В ней впервые в нашей литературе описаны новейшие реализации системы MATLAB R2006/2006a/2006b/2007a/2007b и ее как общие, так и специальные применения. Указанные версии системы имеют практически одинаковые (в рамках самоучителя) возможности и некоторые описанные в книге отличия в интерфейсе. По возможности сохранены наглядные примеры применения системы MATLAB из прежних книг, но в эту книгу включено множество и новых примеров, особенно программ на языке программирования системы.

Отличительными особенностями данного самоучителя являются:

- ориентация на читателей, желающих самостоятельно освоить базовую систему MATLAB новых реализаций, и преподавателей вузов, готовящих по ней учебные курсы;
- ориентация на описание системы MATLAB прежде всего как языка программирования, ориентированного на массовые численные методы вычислений и научно-технические расчеты;
- компактность материала при сохранении достаточной полноты изложения подлежащего изучению материала и справочных данных;
- описание новейших реализаций системы MATLAB R2006/2006a/2006b/2007a/2007b;
- ориентация основного материала книги на текущие версии MATLAB с выделением и достаточно полным описанием новаций самых последних версий MATLAB R2007a,b;
- более систематизированное изложение материала;
- значительно расширенный материал по численному решению дифференциальных уравнений различного типа;
- описание большого числа новых примеров для новейших реализаций MATLAB;

- описание впервые в нашей литературе применения системы MATLAB в измерительной технике (урок 14);
- введение ряда новых материалов по визуально-ориентированному программированию средств графического интерфейса пользователя – GUI;
- расширенный обзор пакетов расширения, в который включен ряд новейших пакетов расширения, появившихся только в последних реализациях MATLAB;
- доступность основного материала пользователям предшествующих версий MATLAB 6.*;
- исключение из книги ряда несущественных деталей, которые затрудняют изучение системы MATLAB и с которыми можно познакомиться самостоятельно по ее обширной справке;
- разделение материала книги на отдельные уроки, каждый из которых, в зависимости от глубины изучения материала, может потребовать примерно от 4 до 6 часов занятий.

Внедрение системы MATLAB в учебный процесс вузов России и стран СНГ находится лишь в начальной стадии. В большинстве наших вузов и университетов пока нет отдельного курса по этой системе, но спецкурсы по ней кое-где уже появились. Как правило, изучение MATLAB выполняется студентами самостоятельно в рамках курсов по численным методам вычислений и математическому моделированию. В таком изучении заинтересованы и преподаватели университетов и вузов, аспиранты, научные работники и инженеры. Им и адресована данная книга. Она может служить не только самоучителем по системе MATLAB, но и достаточно полным учебным курсом по этой системе и даже кратким справочником по ней. Подробное оглавление книги вполне заменяет тематический указатель.

Благодарности и адреса для связи

Автор выражает благодарность представителям корпорации The MathWorks, Inc., Courtney Esposito, Naomi Fernandes и Meg Vulliez. Подготовка автором книг по системе MATLAB и ее расширениям уже многие годы включается в планы поддержки этой корпорацией издания книг по системе MATLAB в разных странах мира и обеспечивается самыми свежими лицензионными программными средствами с обширной документацией по ним. В частности, благодаря этому автор заведомо получил новейшие реализации системы MATLAB, описанные в данной книге, и смог своевременно и достаточно полно проработать данный материал к моменту широкого появления их на нашем рынке.

Доктору технических наук, профессору Владимиру Круглову и кандидату физико-математических наук, доценту Роману Кристаллинскому автор выражает признательность за постоянный обмен мнениями и просмотр некоторых материалов этой книги. Автор благодарит также Генерального директора ЗАО «Смоленский Телепорт» (www.keytown.com) Григория Рухамина за предоставление услуг Интернета в ходе работы над книгой, что позволило посредством прямой оперативной связи с сайтом фирмы The MathWorks, Inc., быть в курсе обновлений системы MATLAB и использовать самую свежую информацию.

С автором можно связаться по электронной почте (vpdyak@keytown.com). Автор заранее выражает признательность всем читателям, которые готовы сообщить свое открытое мнение о данной книге и поделиться соображениями по ее улучшению. К анонимным репликам в свой адрес автор относится отрицательно, полагая, что дискуссия по книгам научного и учебного характера может быть только открытой и честной.

Кроме электронной почты, замечания можно направлять по месту работы автора: 214000, г. Смоленск, ул. Пржевальского, 4, Смоленский государственный университет. Вы можете отправлять свои письма и по адресу издательства, выпустившего книгу.

Связаться с фирмой The MathWorks вы можете, посетив сайт www.mathworks.com. Ее официальные почтовые реквизиты следующие:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7101
E-mail: info@TheMathWorks.com

Первое знакомство с MATLAB

| | |
|--|----|
| 1.1. Назначение и особенности системы MATLAB | 42 |
| 1.2. Начало работы с MATLAB | 46 |
| 1.3. Простые вычисления в MATLAB | 50 |
| 1.4. Основные объекты MATLAB | 55 |
| 1.5. Формирование векторов и матриц | 66 |
| 1.6. Операции с рабочей областью, текстом сессии и редактором m-файлов | 70 |
| 1.7. Интерактивная справка из командной строки | 74 |
| 1.8. Работа с демонстрационными примерами с командной строки | 78 |
| 1.9. Знакомство с двумерной графикой MATLAB | 82 |
| 1.10. Знакомство с трехмерной графикой MATLAB | 86 |

Система компьютерной математики MATLAB – сложный программный продукт. Его освоение целесообразно делать в два захода: вначале стоит изучить общие возможности системы и лишь затем приступить к основательному, нередко избранному знакомству с MATLAB. Этот урок посвящен описанию общих возможностей системы MATLAB и может рассматриваться как краткое введение по системе. Он особенно полезен пользователям, впервые приступившим к работе с системой MATLAB.

1.1. Назначение и особенности системы MATLAB

1.1.1. Начальные сведения о матрицах

Поскольку MATLAB – матричная система, разумно начать ее описание с начальных сведений о векторах и матрицах.

Двумерный массив чисел или математических выражений принято называть *матрицей* [2, 3]. А одномерный массив называют *вектором*. Векторы могут быть двух типов: вектор-строка и вектор-столбец. Примеры векторов и матриц даны ниже:

$$[1 \ 2 \ 3 \ 4]$$

Вектор-строка из 4 элементов

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Вектор-столбец из 3 элементов

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{bmatrix}$$

Матрица размера 3×4 с элементами-числами

$$\begin{bmatrix} a & a+b & a+b/c \\ x & y*x & z \\ 1 & 2 & 3 \end{bmatrix}$$

Матрица с элементами разного типа

Векторы и матрицы характеризуются *размерностью* и *размером*. Размерность определяет структурную организацию массивов в виде строки (размерность 1), страницы (размерность 2), куба (размерность 3) и т. д. Так что вектор является одномерным массивом, а матрица представляет собой двумерный массив с размерностью 2. MATLAB допускает задание и использование многомерных массивов, но в этом уроке мы ограничимся пока только описанием одномерных и двумерных массивов – векторами и матрицами.

Размер вектора – это число его элементов, а размер матрицы определяется произведением числа ее строк m и столбцов n . Обычно *размер матрицы* указывают как $m \times n$. Матрица называется *квадратной*, если $m = n$, то есть число строк матрицы равно числу ее столбцов.

Векторы и матрицы могут иметь имена, например \mathbf{V} – вектор или \mathbf{M} – матрица. В данной книге имена векторов и матриц набираются в основном прямым полужирным шрифтом. Элементы векторов и матриц рассматриваются как *индексированные переменные*, например:

- \mathbf{V}_2 – второй элемент вектора \mathbf{V} ;
- $\mathbf{M}_{2,3}$ – третий элемент второй строки матрицы \mathbf{M} .

Индексы у векторов и матриц в MATLAB имеют целочисленные номера, которые начинаются с 1. Даже обычные числа рассматриваются в MATLAB как матрицы размера 1×1 .

1.1.2. Назначение матричной системы MATLAB

MATLAB – одна из старейших, тщательно проработанных и проверенных временем систем автоматизации математических и научно-технических расчетов, построенная на расширенном представлении и применении матричных операций [1–49]. Это нашло отражение в названии системы – MATrix LABoratory – *матричная лаборатория*. Применение матриц как основных объектов системы способствует резкому уменьшению числа циклов, которые очень распространены при выполнении матричных вычислений на обычных языках программирования высокого уровня, и облегчению реализации параллельных вычислений.

Одной из основных задач при создании системы MATLAB всегда было предоставление пользователям мощного *языка программирования*, ориентированного на технические и математические расчеты и способного превзойти возможности традиционных языков программирования, которые многие годы использовались для реализации численных методов. При этом особое внимание уделялось как повышению скорости вычислений, так и адаптации системы к решению самых разнообразных задач пользователей.

MATLAB реализует три важные концепции программирования:

- процедурное модульное программирование, основанное на создании модулей – процедур и функций;
- объектно-ориентированное программирование, особенно ценное в реализации графических средств системы;
- визуально-ориентированное программирование, направленное на создание средств графического интерфейса пользователя GUI (Graphics User Interface).

Язык программирования MATLAB относится к классу *интерпретаторов*. Это значит, что любая команда системы распознается (интерпретируется) по ее имени (идентификатору) и немедленно исполняется в командной строке, что обеспечивает легкую проверку по частям любого программного кода. Одновременно интерпретирующий характер языка программирования MATLAB означает, что с первых строк описания средств этой системы фактически описывается ее язык программирования.

Важными достоинствами системы являются ее *открытость* и *расширяемость*. Большинство команд и функций системы реализованы в виде m-файлов текстового формата (с расширением *.m*) и файлов на языке C/C++, причем все файлы доступны для модификации. Пользователю дана возможность создавать не только отдельные файлы, но и библиотеки файлов для реализации специфических задач. Любой набор команд в справке можно тут же исполнить с помощью команды `Evaluate Selection` контекстного меню правой клавиши мыши.

1.1.3. Системные требования к установке

Новые версии системы MATLAB, например MATLAB R2006*/2007*, – весьма громоздкий программный комплекс, который требует до 1500–3200 Мб дисковой памяти (в зависимости от конкретной поставки, полноты справочной системы и числа устанавливаемых пакетов прикладных программ). Поэтому система поставляется на трех компакт-дисках (CD-ROM) или на DVD. К сожалению, из поставки новых версий исключены PDF-файлы англоязычной документации, но доступ к ним открыт через Интернет. Однако получение их при низкой скорости доступа (до 56 Кбит/с) в наших условиях весьма проблематично [67, 68]. Это усиливает роль русскоязычной литературы по системе MATLAB.

Для успешной установки новых версий MATLAB необходимы следующие минимальные средства:

- компьютер с микропроцессором не ниже Pentium и математическим сопроцессором, рекомендуются процессоры Pentium III, Pentium IV, Pentium M или AMD Athlon, Athlon XP, Athlon MP (последние версии MATLAB 2007a,b поддерживают двухъядерные процессоры, например серий Intel Core 2 Duo, а MATLAB R2007b поддерживает четырехъядерные процессоры, например Intel Core 2 Quad);
- устройство считывания компакт-дисков (привод CD-ROM или DVD), мышь, 8-разрядный графический адаптер и монитор, поддерживающие не менее 256 цветов;
- операционная система Windows XP/2000/NT/Vista (допускается Windows NT4 с сервис-пакетами 5 или 6a);
- ОЗУ емкостью 256 Мб для минимального варианта системы (рекомендуется иметь память 512 Мб и выше);
- до 3200 Мб дискового пространства при полной установке всех расширений и всех справочных систем (345 Мб при установке только MATLAB со справкой);
- свободный USB-порт для подключения ключа, открывающего доступ к системе.

Для использования расширенных возможностей системы нужны графический ускоритель, Windows-совместимые звуковая карта и принтер, текстовый процессор Microsoft Word 97/2000/XP [67] для реализации Notebook, компиляторы языков Си/Си++ и/или ФОРТРАН для подготовки собственных файлов расширения и браузер Netscape Navigator 4.0 и выше или Microsoft Internet Explorer 5.0

и выше. Для просмотра файлов справочной системы в формате PDF нужна программа Adobe Reader или Adobe Acrobat 5.0 и выше.

Далее рассматриваются только реализации системы, работающие с операционными системами класса Windows. Все примеры даны для систем класса MATLAB, запущенных в среде Windows XP.

1.1.4. Инсталляция системы MATLAB 7 + Simulink 6

Система MATLAB 7.* (R2006*/2007*) + Simulink 6.* /7 поставляется на трех CD-ROM или на одном DVD. Для инсталляции ее с другими пакетами расширения достаточно установить первый CD-ROM и дождаться его автоматического запуска (или запустить его, как обычно). После распаковки и установки файлов инсталлятора на короткое время появляется окно с эмблемой MATLAB, а затем первое окно инсталлятора.

Инсталляция системы MATLAB неоднократно описывалась, и потому детали ее в этой книге опущены. Отметим лишь, что в первом окне инсталлятора надо установить опцию **Install** для инсталляции или опцию обновления лицензии и получения кода PLP (Personal License Password). Этот код является группой из 20 цифр. Установим Install и нажмем мышью кнопку **Next>**. Появится окно для ввода данных пользователя (имени и названия организации) и, главное, кода PLP. Этот код записывается в виде длинного числа и указывается на диске при продаже MATLAB или запрашивается у MathWorks по Интернету. Каждый легальный пользователь MATLAB ныне имеет свои страницы на этом интернет-сайте с данными о лицензии, ее сроках и комплекте поставки MATLAB. С этой страницы можно получить и коды PLP.

Дальнейшие операции производятся в соответствии с простыми указаниями окон инсталлятора. Инсталляция занимает немало времени – около получаса даже на современных ПК. Ничего нового в инсталляции нет и в новейших версиях MATLAB R2007a,b, лишь недавно появившихся на рынке программных средств России. Система поставляется на DVD.

1.1.5. Файловая система MATLAB

MATLAB состоит из многих тысяч файлов, находящихся во множестве папок. Полезно иметь представление о содержании основных папок, поскольку это позволяет быстро оценить возможности системы. Кроме того, нередко надо обеспечить путь к нужным для работы файлам системы, иначе содержащиеся в них команды не будут работать.

В MATLAB особое значение имеют файлы двух типов – с расширениями **.mat** и **.m**. Первые являются бинарными файлами, в которых могут храниться значения переменных. Вторые представляют собой текстовые файлы, содержащие внешние программы, определения команд и функций системы. Именно к ним от-

носятся большая часть команд и функций, в том числе задаваемых пользователем для решения своих специфических задач. Нередко встречаются и файлы с расширением **.c** (коды на языке Си), файлы с откомпилированными кодами MATLAB с расширением **.mex** и др. Исполняемые файлы имеют расширение **.exe**.

Особое значение имеет папка **MATLAB/TOOLBOX/MATLAB**. В ней содержится набор стандартных m-файлов системы. Просмотр этих файлов позволяет детально оценить возможности поставляемой конкретной версии системы.

Полный состав файлов каждой папки (их список содержится в файле **contents.m**) можно вывести на просмотр с помощью команды `help имя`, где `имя` – название соответствующей подпапки. Ознакомиться с файловой системой MATLAB несложно с помощью Проводника Windows или любого файлового менеджера. В MATLAB 2007b директории лучше упорядочены, чем в предшествующих версиях.

1.2. Начало работы с MATLAB

1.2.1. Запуск MATLAB и работа в режиме диалога

Интерфейс и возможности трех последних версий системы MATLAB в рамках предназначения данной книги как самоучителя различаются незначительно. Однако при описании отдельных деталей системы желательно ориентироваться на какую-либо конкретную версию системы. В качестве таковой вначале рассмотрим подверсию MATLAB R2006b. Она распространена намного больше, чем новейшие MATLAB R2007a,b. О новых возможностях их будет сказано немного позднее. Пока же отметим, что в рамках материалов, характерных для данной книги-самоучителя, разница между различными версиями базовой системы MATLAB практически отсутствует.

MATLAB (к примеру, R2006b) обычно запускается из главного меню операционной системы Windows XP или активизацией ярлыка с логотипом системы на рабочем столе Windows. После запуска MATLAB на экране появляется основное окно системы MATLAB, показанное на рис. 1.1. Оно имеет обычные средства управления размерами, скрытия и закрытия. В окне командного режима показано окно About MATLAB, которое выводится одноименной командой в позиции Help меню и позволяет уточнить версию системы.

Система готова к проведению вычислений в *командном режиме*. Полезно знать, что в начале запуска автоматически выполняется команда `matlabrc`, которая исполняет загрузочный файл **matlabrc.m** и файл **startup.m**, если таковой существует. Эти файлы текстового формата выполняют начальную настройку терминала системы и задают ряд ее параметров.

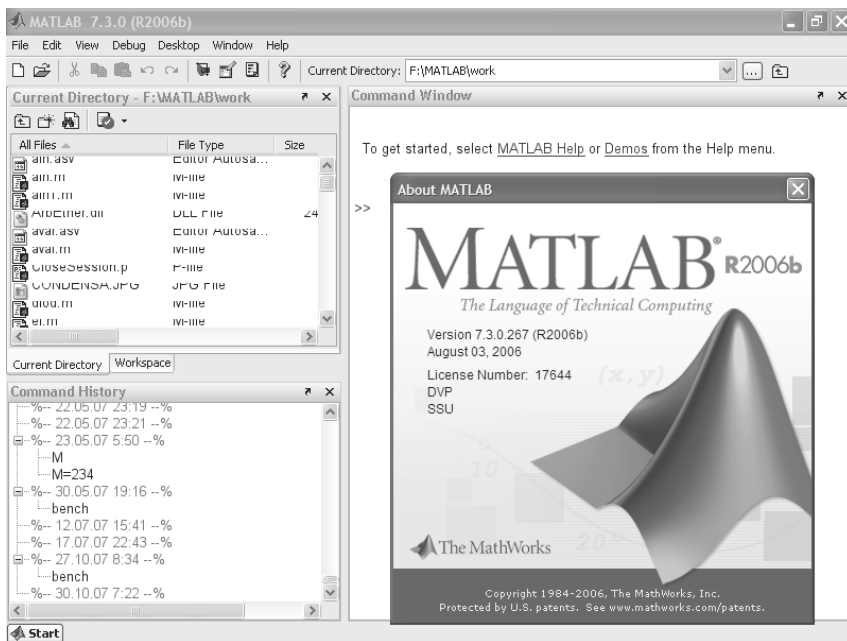


Рис. 1.1. Окно системы MATLAB 2006b после запуска

1.2.2. Понятие о сессии работы с системой MATLAB

Сеанс работы с MATLAB принято именовать *сессией* (session). Сессия в сущности является текущим документом, отражающим работу пользователя с системой MATLAB. В ней имеются строки ввода, вывода и сообщений об ошибках. Входящие в сессию определения переменных и функций, расположенные в рабочей области памяти, но не саму сессию можно записать на диск (файлы формата **.mat**), используя команду **save** (**Сохранить**). Команда **load** (**Загрузить**) позволяет считать с диска данные рабочей области. Фрагменты сессии можно оформить в виде дневника с помощью команды **diary** (**Дневник**). Позже мы обсудим эти команды подробно.

Полезно обратить внимание на возможность использования контекстного меню правой клавиши мыши в момент выделения той или иной позиции рабочего меню. Как и во всех приложениях операционных систем Windows XP/2000/NT4, это меню дает доступ ко всем возможным в данный момент операциям.

1.2.3. Новый и старый облики системы MATLAB

Вид окна системы MATLAB (рис. 1.1), выводимого изначально, вполне отвечает канонам современного интерфейса Windows-приложений. Пользовательский интерфейс многооконный и имеет ряд средств прямого доступа к различным компонентам системы. В панели инструментов имеется меню просмотра файловой системы с кнопкой его открытия.

В левой части общего окна системы имеются окна доступа к компонентам системы **Launch Pad/Workspace (Панель запуска/Рабочая область)** и окно **Current Directory** (текущей папки). Под ними расположено окно **Command History**, содержащее список выполненных команд. Щелкнув мышью по любой команде, ее можно перенести в текущую строку окна командного режима MATLAB.

Меню MATLAB R2006 стало контекстно-зависимым. Изменение внешнего вида интерфейса отведено командам позиции `Desktop` меню. Если оставить только командное окно, то интерфейс MATLAB будет иметь упрощенный вид – см. рис. 1.2. Такой вид интерфейса был характерен для старых версий системы. Многие пользователи находят его наиболее приемлемым.

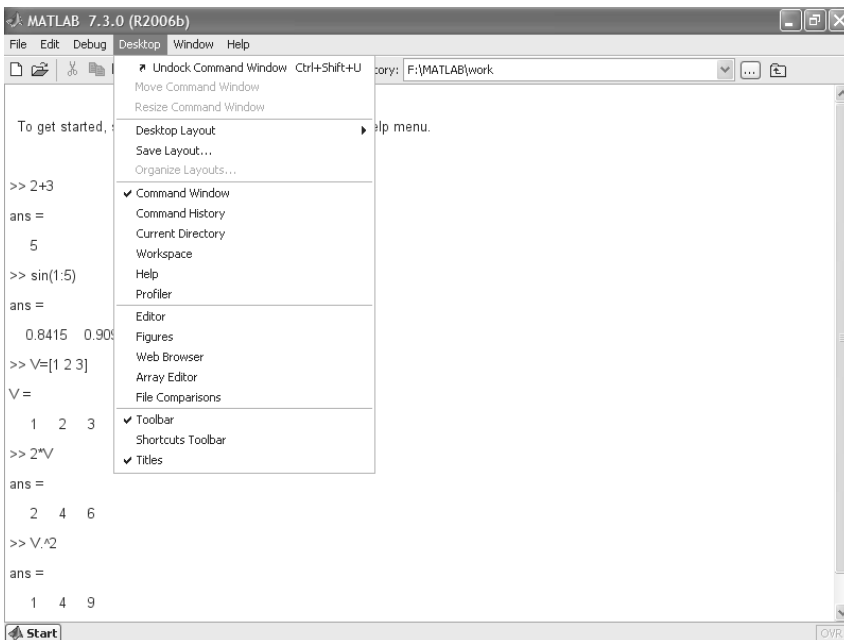


Рис. 1.2. Упрощенный интерфейс системы MATLAB 2006b

1.2.4. Операции строчного редактирования

При работе с MATLAB в командном режиме действует простейший строчный редактор. Его работа знакома любому пользователю ПК еще со времен работы с приложениями под операционную систему MS-DOS и в детальном описании не нуждается. Ограничимся указанием команд строчного редактирования, которые представлены в табл. 1.1.

Таблица 1.1. Команды строчного редактора MATLAB

| Комбинация клавиш | Назначение |
|---|--|
| → или Ctrl+b | Перемещение курсора вправо на один символ |
| ← или Ctrl+f | Перемещение курсора влево на один символ |
| Ctrl+→ или Ctrl+r | Перемещение курсора вправо на одно слово |
| Ctrl+← или Ctrl+l | Перемещение курсора влево на одно слово |
| Home или Ctrl+a | Перемещение курсора в начало строки |
| End или Ctrl+e | Перемещение курсора в конец строки |
| ↑ и ↓ или Ctrl+p и Ctrl+n | Перелистывание предыдущих команд вверх или вниз для подстановки в строку ввода |
| Del или Ctrl+d | Стирание символа справа от курсора |
| ← или Ctrl+h | Стирание символа слева от курсора |
| Ctrl+k | Стирание до конца строки |
| Esc | Очистка строки ввода |
| Ins | Включение/выключение режима вставки |
| PgUp | Перелистывание страниц сессии вверх |
| PgDn | Перелистывание страниц сессии вниз |

Обратите особое внимание на применение клавиш ↑ и ↓. Они используются для подстановки после маркера строки ввода >> ранее введенных строк, например для их исправления, дублирования или дополнения. При этом указанные клавиши обеспечивают перелистывание ранее введенных строк снизу вверх или сверху вниз. Такая возможность существует благодаря организации специального стека, хранящего строки с исполненными ранее командами.

1.2.5. Команды управления окном

Полезно сразу усвоить некоторые команды управления окном командного режима:

- `clc` – очищает экран и размещает курсор в левом верхнем углу пустого экрана;
- `home` – возвращает курсор в левый верхний угол окна;
- `echo <file_name> on` – включает режим вывода на экран текста Script-файла (файла-сценария);
- `echo <file_name> off` – выключает режим вывода на экран текста Script-файла;
- `echo <file_name>` – меняет режим вывода на противоположный;
- `echo on all` – включает режим вывода на экран текста всех m-файлов;

- `echo off all` – отключает режим вывода на экран текста всех `m`-файлов;
- `more on` – включает режим постраничного вывода (полезен при просмотре больших `m`-файлов);
- `more off` – отключает режим постраничного вывода (в этом случае для просмотра больших файлов надо пользоваться линейкой прокрутки).

В новых версиях MATLAB обе команды `clc` и `home` действуют аналогично – очищают экран и помещают курсор в левый верхний угол окна командного режима работы.

1.3. Простые вычисления в MATLAB

1.3.1. MATLAB в роли мощного научного калькулятора

Интерпретирующий язык программирования системы MATLAB создан таким образом, что любые (подчас весьма сложные) вычисления можно выполнять в режиме *прямых вычислений*, то есть без подготовки программы пользователем. При этом MATLAB выполняет функции суперкалькулятора и работает в *режиме командной строки*.

Работа с системой носит диалоговый характер и происходит по правилу «задал вопрос – получил ответ». Пользователь набирает на клавиатуре вычисляемое выражение, редактирует его (если нужно) в командной строке и завершает ввод нажатием клавиши **ENTER**. В качестве примера на рис. 1.2 показаны простейшие и вполне очевидные вычисления.

Даже из таких простых примеров можно сделать некоторые поучительные выводы:

- для указания ввода исходных данных используется символ `>>`;
- данные вводятся с помощью простейшего строчного редактора;
- для блокировки вывода результата вычислений некоторого выражения после него надо установить знак `;` (точка с запятой);
- если не указана переменная для значения результата вычислений, то MATLAB назначает такую переменную с именем `ans`;
- знаком присваивания является привычный математикам знак равенства `=`, а не комбинированный знак `:=`, как во многих других языках программирования и математических системах;
- встроенные функции (например, `sin`) записываются строчными буквами, и их аргументы указываются в *круглых скобках*;
- результат вычислений выводится в строках вывода (без знака `>>`);
- диалог происходит в стиле «задал вопрос – получил ответ».

Следующие примеры (см. рис. 1.3) иллюстрируют применение системы MATLAB для выполнения еще ряда простых векторных операций. На рисунке представлено также окно браузера файловой системы, который имеется на вкладке Current Directory, видной на рис. 1.1. В командном режиме вызов окна браузера файловой системы удобнее производить из панели инструментов активизацией кнопки после списка директорий системы MATLAB. Возможны случаи отказа от

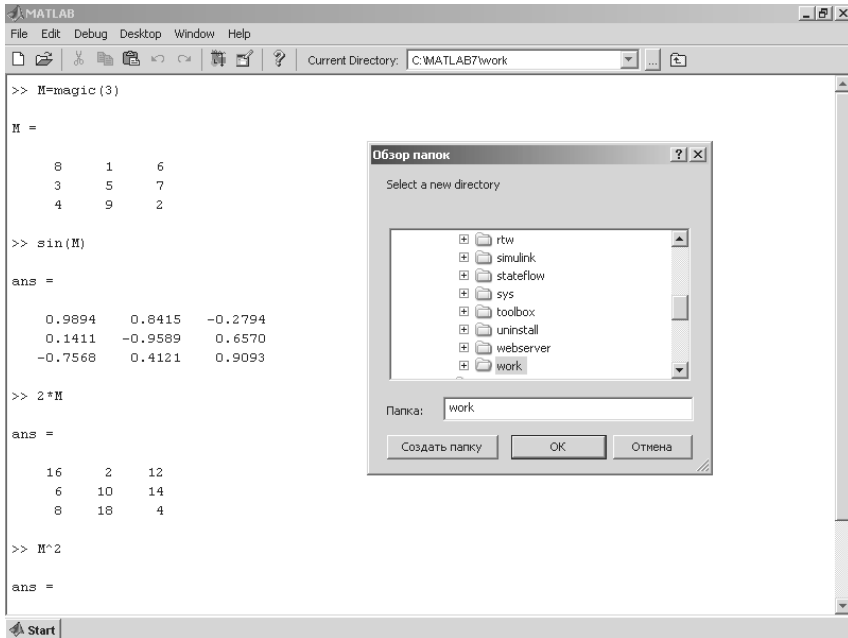


Рис. 1.3. Примеры операций с матрицей

вычислений при неправильно установленной текущей директории, если нужные для вычислений m-файлы не обнаруживаются.

В большинстве математических систем вычисление $\sin(V)$ или $\exp(V)$, где V – вектор, сопровождалось бы выдачей ошибки, поскольку функции \sin и \exp должны иметь аргумент в виде скалярной величины. Однако MATLAB – матричная система, а вектор является разновидностью матрицы с размером $1 \times n$ или $n \times 1$. Поэтому в нашем случае результат вычислений будет вектором того же размера, что и аргумент V , но элементы возвращаемого вектора будут синусами или экспонентами от элементов вектора V .

Матрица задается в виде ряда векторов, представляющих ее строки и заключенных в квадратные скобки. Для разделения элементов векторов используется пробел или запятая, а для отделения одного вектора от другого – точка с запятой. Для выделения отдельного элемента матрицы M используется выражение вида $M(j, i)$, где M – имя матрицы, j – номер строки и i – номер столбца.

Для просмотра содержимого массивов удобно использовать браузер рабочего пространства Workspace. Каждый вектор и матрица в нем представляются в виде квадратика с ячейками, справа от которого указывается размер массива. Двойной щелчок по квадратiku мышью ведет к появлению окна редактора массивов Array Editor – его окно показано на рис. 1.4. Работа с редактором массивов вполне очевидна – возможен не только просмотр элементов массивов, но и их редактирование и замена.

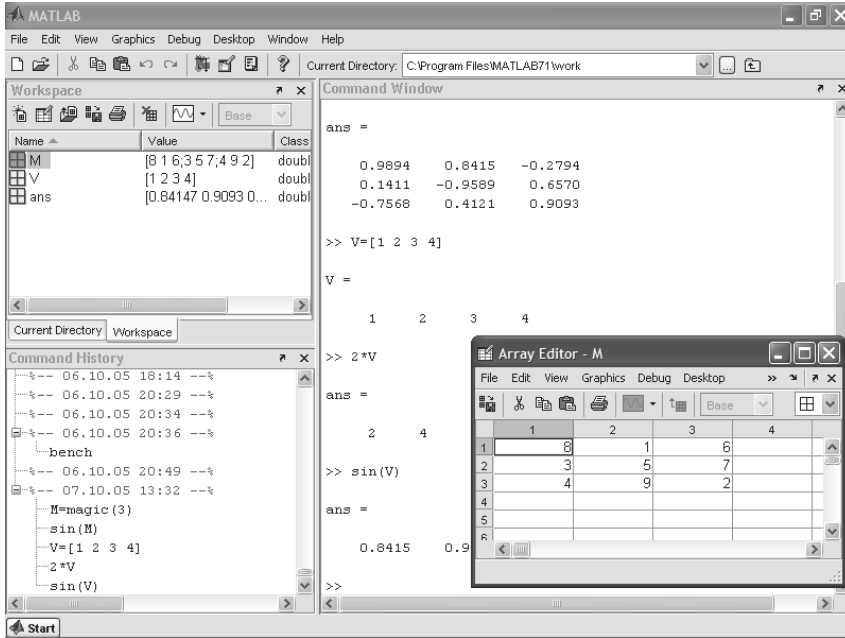


Рис. 1.4. Работа с редактором массивов

Как видно из приведенных примеров, ввод исходных выражений для вычислений в системе MATLAB осуществляется в самом обычном текстовом формате. В этом же формате выдаются результаты вычислений, за исключением графических. Приведем примеры записи вычислений, выполненных системой MATLAB в командной строке и размещенных в тексте книги:

To get started, select "MATLAB Help" from the Help menu.

```
>> 2+3
```

```
ans =
```

```
5
```

```
>> sin(1)
```

```
ans =
```

```
0.8415
```

```
>> type sin
```

```
sin is a built-in function.
```

```
>> help sin
```

```
SIN Sine.
```

```
SIN(X) is the sine of the elements of X.
```

```
Overloaded methods
```

```
help sym/sin.m
```

```
>> V=[1 2 3 4]
```

```
V =
```

```
1     2     3     4
```

```

>> sin(V)
ans =
    0.8415 0.9093 0.1411 -0.7568
>> 3*V
ans =
     3     6     9    12
>> V^2
??? Error using ==> ^
Matrix must be square.
>> V.^2
ans =
     1     4     9    16
>> V+2
ans =
     3     4     5     6
>>

```

Обратите внимание на форму ответов при выполнении простых операций без указания переменной, которой присваивается результат. В таких случаях MATLAB сам назначает переменную `ans`, которой присваивается результат и значение которой затем выводится на экран.

1.3.2. Форма вывода и перенос строки в сессии

Следует отметить особенности вывода в системе MATLAB. Вывод начинается с новой строки, причем числовые данные выводятся с отступом, а текстовые – без него. Для экономии места в данной книге в дальнейшем вывод будет даваться без перевода на новую строку. Например, вывод вектора-строки

```
ans =
     3     4     5     6
```

будет дан в виде:

```
ans = 3 4 5 6
```

Исключением является вывод векторов столбцов и матриц – тут будет сохранена более наглядная и присущая MATLAB по умолчанию форма вывода.

В некоторых случаях вводимое математическое выражение может оказаться настолько длинным, что для него не хватит одной строки. Тогда часть выражения можно перенести на новую строку с помощью знака многоточия «...» (3 или более точек), например:

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Максимальное число символов в одной строке командного режима – 4096, а в `m`-файле – не ограничено, но со столь длинными строками работать неудобно. В ранних версиях в одной строке было не более 256 символов.

1.3.3. Запуск примеров применения MATLAB из командной строки

MATLAB имеет множество примеров применения, часть из которых можно запускать прямо из командной строки. Например, команда

```
>> bench
```

запускает m-файл bench.m демонстрационного примера тестирования системы (рис. 1.5).

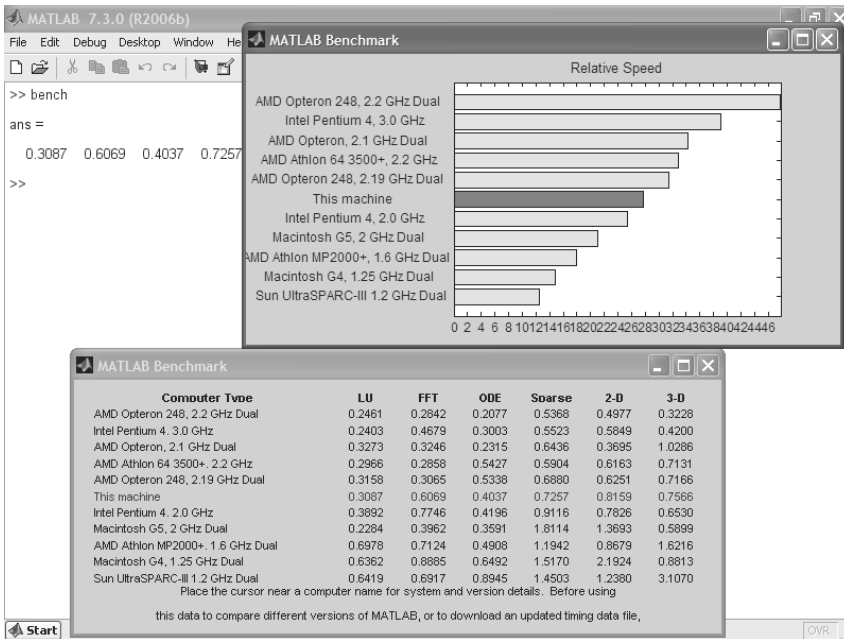


Рис. 1.5. Пример тестирования MATLAB 2006b на скорость выполнения различных операций

Здесь ПК автора на процессоре Pentium 4 HT 2,6 ГГц занял среднее место. Лучшие места заняли в основном ПК на основе самых новейших двухъядерных микропроцессоров или на двух процессорах. В уроке 2 будут описаны результаты тестирования новейших версий MATLAB на новейших четырехъядерных ПК.

1.4. Основные объекты MATLAB

1.4.1. Понятие о математическом выражении

Центральным понятием всех математических систем является *математическое выражение*. Оно задает то, что должно быть вычислено в численном (реже символьном) виде. Вот примеры простых математических выражений, записанных в MATLAB и в математике.

| В MATLAB: | В математике: |
|----------------|----------------|
| 2+3; | 2+3 |
| 2^3*sqrt(y)/2; | 2+3*003*****/2 |
| 2.301*sin(x) | 2,301sin(x) |
| 4+exp(3)/5 | 4+e3/5 |

Разница в записи вполне очевидна. В MATLAB выражения записываются в виде одной строки и вместо разделительной запятой в числах применяется разделительная точка. Математические выражения строятся на основе чисел, констант, переменных, операторов, функций и разных спецзнаков. Ниже даются краткие пояснения сути этих понятий. Специфика MATLAB в том, что математические выражения задаются в виде одной строки. Например, 2^3 записывается как 2^3. Знак ; (точка с запятой) в конце строки ввода блокирует вывод результата вычислений, например:

```
>> 2^3;
```

Однако специальная переменная ans (от *answer* – ответ) позволяет вывести результат вычислений:

```
>> ans
ans = 8
```

1.4.2. Действительные и комплексные числа

Число – простейший объект языка MATLAB, представляющий количественные данные. Числа можно считать константами. Числа используются в общепринятом представлении о них. Они могут быть целыми, дробными, с фиксированной и плавающей точкой. Возможно представление чисел в хорошо известном научном формате с указанием мантиссы и порядка числа. Ниже приводятся примеры представления действительных чисел:

```
0
-3
2.301
123.456e-24
-234.456e10
```

Как нетрудно заметить, в мантиссе чисел целая часть отделяется от дробной не запятой, а точкой, как принято в большинстве языков программирования. Для отделения порядка числа от мантиссы используется символ e . Знак «плюс» у чисел не проставляется, а знак «минус» у числа называют *унарным минусом*. Пробелы между символами в числах не допускаются.

Числа могут быть *комплексными*: $z = \text{Re}(x) + \text{Im}(x) * i$. Такие числа содержат действительную $\text{Re}(z)$ и мнимую $\text{Im}(z)$ части. Мнимая часть имеет множитель i или j , означающий корень квадратный из -1 :

```
3i
2j
2+3i
-3.141i
-123.456+2.7e-3i
```

Функция `real(z)` возвращает действительную часть комплексного числа, $\text{Re}(z)$, а функция `imag(z)` – мнимую, $\text{Im}(z)$. Для получения модуля комплексного числа используется функция `abs(z)`, а для вычисления фазы – `angle(z)`. Ниже даны простейшие примеры работы с комплексными числами:

```
>> i
ans = 0 + 1.0000i
>> j
ans = 0 + 1.0000i
>> z=2+3i
z = 2.0000 + 3.0000i
>> abs(z)
ans = 3.6056
>> real(z)
ans = 2
>> imag(z)
ans = 3
>> angle(z)
ans = 0.9828
```

Операции над числами по умолчанию выполняются в формате, который принято считать форматом *с двойной точностью* (правильнее сказать с двойной разрядностью).

1.4.3. Форматы чисел

Для установки определенного *формата* представления чисел используется команда

```
>> format name
```

где `name` – имя формата. Для иллюстрации различных форматов рассмотрим вектор, содержащий два элемента-числа:

```
x=[4/3 1.2345e-6]
```

В различных форматах их представления будут иметь следующий вид:

| | | |
|----------------|-------------------------|-------------------------|
| format short | 1.3333 | 0.0000 |
| format short e | 1.3333E+000 | 1.2345E-006 |
| format long | 1.3333333333333338 | 0.000001234500000 |
| format long e | 1.3333333333333338E+000 | 1.2345000000000000E-006 |
| format bank | 1.33 | 0.00 |

Задание формата сказывается только на *форме вывода* чисел. Вычисления все равно происходят в формате двойной точности, а ввод чисел возможен в любом удобном для пользователя виде.

1.4.4. Константы и системные переменные

Константа – это предварительно определенное числовое или символьное значение, представленное уникальным именем (идентификатором). Числа (например, 1, -2 и 1.23) являются безымянными *числовыми константами*.

Другие виды констант в MATLAB принято называть *системными переменными*, поскольку, с одной стороны, они задаются системой при ее загрузке, а с другой – могут переопределяться. Основные системные переменные, применяемые в системе MATLAB, указаны ниже:

- *i* или *j* – мнимая единица (корень квадратный из -1);
- *pi* – число $p = 3,1415926\dots$;
- *eps* – погрешность операций над числами с плавающей точкой (2^{-52});
- *realmin* – наименьшее число с плавающей точкой (2^{-1022});
- *realmax* – наибольшее число с плавающей точкой (2^{1023});
- *inf* – значение машинной бесконечности;
- *ans* – переменная, хранящая результат последней операции и обычно вызывающая его отображение на экране дисплея;
- NaN – указание на нечисловой характер данных (Not-a-Number).

Вот примеры применения системных переменных:

```
>> 2*pi
ans = 6.2832
>> eps
ans = 2.2204e-016
>> realmin
ans = 2.2251e-308
>> realmax
ans = 1.7977e+308
>> 1/0
Warning: Divide by zero.
ans = Inf
>> 0/0
Warning: Divide by zero.
ans = NaN
```

Как отмечалось, системные переменные могут *переопределяться*. Можно задать системной переменной *eps* иное значение, например $\text{eps}=0.0001$. Однако

важно то, что их значения по умолчанию задаются сразу после загрузки системы. Поэтому неопределенными, в отличие от обычных переменных, системные переменные не могут быть никогда.

Символьная константа – это цепочка символов, заключенных в апострофы, например:

```
'Hello my friend!'  
'Привет'  
'2+3'
```

Если в апострофы помещено математическое выражение, то оно *не вычисляется* и рассматривается просто как цепочка символов. Так что '2+3' не будет возвращать число 5. Однако с помощью специальных функций преобразования символьные выражения могут быть преобразованы в вычисляемые. Соответствующие функции преобразования будут рассмотрены в дальнейшем.

1.4.5. Текстовые комментарии в программах

Поскольку MATLAB используется для достаточно сложных вычислений, важное значение имеет наглядность их описания. Она достигается, в частности, с помощью текстовых комментариев. *Текстовые комментарии в программах* вводятся с помощью символа %, например так:

```
% It is factorial function
```

В новых версиях MATLAB отпала проблема ввода комментариев с символами кириллицы. Так что подобный комментарий также вполне приемлем:

```
% Это функция вычисления факториала
```

Обычно первые строки m-файлов служат для описания их назначения, которое выводится на экран дисплея после команды

```
>> help Имя_файла
```

Считается правилом хорошего тона вводить в m-файлы достаточно подробные текстовые комментарии. Без таких комментариев даже разработчик программных модулей быстро забывает о сути собственных решений.

1.4.6. Переменные и присваивание им значений

Переменные – это имеющие имена объекты, способные хранить некоторые, обычно разные по значению, данные. В зависимости от этих данных переменные могут быть числовыми или символьными, векторными или матричными. Переменные являются широко распространенными объектами в математике и программировании.

На языке программирования MATLAB можно задавать переменным определенные значения. Для этого используется операция *присваивания*, вводимая знаком равенства:

```
Имя_переменной = Выражение
```

Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваивается переменной. Так, если это выражение – вектор или матрица, то переменная будет векторной или матричной. Переменная, имеющая единственное значение, рассматривается как матрица размера 1×1 .

Имя переменной (ее *идентификатор*) может содержать сколько угодно символов, но запоминается и идентифицируется только 31 начальный символ. Имя любой переменной не должно совпадать с именами других переменных, функций и процедур системы, то есть оно должно быть уникальным. Имя должно начинаться с буквы, может содержать буквы, цифры и символ подчеркивания `_`. Недопустимо включать в имена переменных пробелы и специальные знаки, например `+`, `-`, `*`, `/` и т. д., поскольку в этом случае правильная интерпретация выражений становится невозможной.

Желательно использовать содержательные имена для обозначений переменных, например `speed_1` для переменной, обозначающей скорость первого объекта. Переменные могут быть обычными и *индексированными*, то есть элементами векторов или матриц (см. выше). Могут использоваться и *символьные* переменные, причем символьные значения заключаются в апострофы, например `s = 'Demo'`. Имена переменных рекомендуется задавать только латинскими буквами, цифрами и различными символами (не допускается применение символов операторов).

1.4.7. Уничтожение определений переменных

В памяти компьютера переменные занимают определенное место, называемое *рабочей областью* (workspace). Для очистки рабочей области используется функция `clear` в разных формах, например:

- `clear` – уничтожение определений всех переменных;
- `clear x` – уничтожение определения переменной `x`;
- `clear a, b, c` – уничтожение определений нескольких переменных.

Уничтоженная (стертая в рабочей области) переменная становится неопределенной. Использовать неопределенные переменные нельзя, и такие попытки будут сопровождаться выдачей сообщений об ошибке. Приведем примеры задания и уничтожения переменных:

```
>> x=2*pi
x = 6.2832
>> V=[1 2 3 4 5]
V = 1 2 3 4 5
>> MAT
```



```

??? Undefined function or variable 'MAT'.
>> MAT=[1 2 3 4; 5 6 7 8]
MAT =
     1     2     3     4
     5     6     7     8
>> clear V
>> V
??? Undefined function or variable 'V'.
>> clear
>> x
??? Undefined function or variable 'x'.
>> M
??? Undefined function or variable 'M'.

```

Обратите внимание на то, что сначала выборочно стерта переменная V , а затем командой `clear` без параметров стерты все остальные переменные.

Неопределенные переменные используются при выполнении символьных вычислений. Специально система MATLAB для выполнения таких вычислений не предназначена. Однако они возможны с помощью пакета расширения символьной математики Symbolic Math.

1.4.8. Операторы и встроенные функции MATLAB

Оператор – это специальное обозначение для определенной операции над данными – *операндами*. Например, простейшими арифметическими операторами являются знаки суммы $+$, вычитания $-$, умножения $*$ и деления $/$. Операторы используются совместно с операндами. Например, в выражении $2+3$ знак $+$ является оператором сложения, а числа 2 и 3 – операндами. Операторы также являются распространенными объектами математических выражений и языков программирования.

Следует отметить, что большинство операторов относятся к матричным операциям, что может служить причиной серьезных недоразумений. Например, операторы умножения $*$ и деления $/$ вычисляют произведение и частное от деления двух массивов, векторов или матриц. Есть ряд специальных операторов, например оператор \backslash означает деление *справа налево*, а операторы $.*$ и $./$ означают, соответственно, *поэлементное* умножение и *поэлементное* деление массивов.

Следующие примеры поясняют сказанное на примере операций с векторами:

```

>> V1=[2 4 6 8]
V1 = 2 4 6 8
>> V2=[1 2 3 4]
V2 = 1 2 3 4
>> V1/V2
ans = 2
>> V1.*V2
ans = 2 8 18 32
>> V1./V2
ans = 2 2 2 2

```

Полный список операторов можно получить, используя команду `help ops`. Приведем начало обширного полного списка операторов, содержащего арифметические операторы:

```
>> help ops
Operators and special characters.
Arithmetic operators.
Plus      - Plus          +
Uplus     - Unary plus    +
Minus     - Minus        -
Uminus    - Unary minus   -
Mtimes    - Matrix multiply *
times     - Array multiply .*
mpower    - Matrix power  ^
power     - Array power   .^
mldivide  - Backslash or left matrix divide \
mrdivide  - Slash or right matrix divide /
ldivide   - Left array divide .\
rdivide   - Right array divide ./
kron      - Kronecker tensor product
```

.....

Функции – это имеющие уникальные имена объекты, выполняющие определенные преобразования своих аргументов и при этом возвращающие результаты этих преобразований. *Возврат результата* – отличительная черта функций. При этом результат вычисления функции с одним выходным параметром подставляется на место ее вызова, что позволяет использовать функции в математических выражениях, например функцию \sin в `2*sin(pi/2)`.

Функции в общем случае имеют список аргументов (параметров), заключенный в круглые скобки. Например, функция Бесселя записывается как `bessel (NU, X)`. В данном случае список параметров содержит два аргумента – NU в виде скаляра и X в виде вектора. Многие функции допускают ряд форм записи, отличающихся списком параметров. Если функция возвращает несколько значений, то она записывается в виде

```
[Y1, Y2, ...]=func(X1, X2, ...),
```

где Y1, Y2, ... – список *выходных* параметров и X1, X2, ... – список *входных* аргументов (параметров).

Со списком элементарных функций можно ознакомиться, выполнив команду `help elfun`, а со списком специальных функций – с помощью команды `help specfun`. Функции могут быть *встроенными* (внутренними) и *внешними*, или *m-функциями*. Так, встроенными являются наиболее распространенные элементарные функции, например $\sin(x)$ и $\exp(y)$, тогда как функция $\sinh(x)$ является внешней функцией. Внешние функции содержат свои определения в m-файлах. Задание таких функций возможно с помощью специального редактора m-файлов, который мы рассмотрим чуть позже. Встроенные функции хранятся в откомпилированном ядре системы MATLAB, в силу чего они выполняются предельно быстро.

1.4.9. Применение оператора : (двоеточие)

Очень часто необходимо произвести формирование упорядоченных числовых последовательностей. Такие последовательности нужны, например, для создания векторов со значениями абсциссы при построении графиков или при создании таблиц. Для этого в MATLAB используется оператор : (двоеточие) в виде:

Начальное_значение:Шаг:Конечное_значение

Данная конструкция порождает возрастающую последовательность чисел, которая начинается с начального значения, идет с заданным шагом и завершается конечным значением. Применение этой конструкции резко уменьшает потребность в задании программных циклов.

Если Шаг не задан, то он принимает значение 1. Если конечное значение указано меньшим, чем начальное значение, – выдается сообщение об ошибке. Примеры применения оператора : даны ниже:

```
>> 1:5
ans = 1      2      3      4      5
>> i=0:2:10
i = 0      2      4      6      8      10
>> j=10:-2:2
j = 10     8      6      4      2
>> V=0:pi/2:2*pi;
>> V
V = 0      1.5708      3.1416      4.7124      6.2832
>> X=1:-.2:0
X = 1.0000      0.8000      0.6000      0.4000      0.2000      0
>> 5:2
ans =
Empty matrix: 1-by-0
```

Как отмечалось, принадлежность MATLAB к матричным системам вносит коррективы в назначение операторов и приводит, при неумелом их использовании, к казусам. Рассмотрим следующий характерный пример:

```
>> x=0:5
x = 0      1      2      3      4      5
>> cos(x)
ans = 1.0000      0.5403      -0.4161      -0.9900      -0.6536      0.2837
>> sin(x)/x
ans = -0.0862
```

Вычисление массива косинусов здесь прошло корректно. А вот вычисление массива значений функции $\sin(x)/x$ дает неожиданный, на первый взгляд, эффект – вместо массива с шестью элементами вычислено единственное значение!

Причина «парадокса» здесь в том, что оператор / вычисляет отношение двух матриц, векторов или массивов. Если они одной размерности, то результат будет одним числом, что в данном случае и выдала система. Чтобы действительно получить вектор значений $\sin(x)/x$, надо использовать специальный оператор *поэлементного* деления массивов – ./ . Тогда будет получен массив чисел:

```
>> sin(x)./x
Warning: Divide by zero.
ans = NaN    0.8415    0.4546    0.0470   -0.1892   -0.1918
```

Впрочем, и тут без особенностей не обошлось. Так, при $x = 0$ значение $\sin(x)/x$ дает устраняемую неопределенность вида $0/0 = 1$. Однако, как и всякая численная система, MATLAB классифицирует попытку деления на 0 как ошибку и выводит соответствующее предупреждение. А вместо ожидаемого численного значения выводится символьная константа NaN, означающая, что неопределенность $0/0$ – это все же не обычное число.

Выражения с оператором `:` могут использоваться в качестве аргументов функций для получения множественных их значений. Например, в приводимом ниже примере вычислены функции Бесселя порядка от 0 до 5 со значением аргумента 0,5:

```
>> bessel(0:1:5,1/2)
ans = 0.9385    0.2423    0.0306    0.0026    0.0002    0.0000
```

А в следующем примере вычислено шесть значений функции Бесселя нулевого порядка для значений аргумента от 0 до 5 с шагом 1:

```
>> bessel(0,0:1:5)
ans = 1.0000    0.7652    0.2239   -0.2601   -0.3971   -0.1776
```

Таким образом, оператор `:` является весьма удобным средством задания регулярной последовательности чисел. Он широко используется при работе со средствами построения графиков. В дальнейшем мы расширим представление о возможностях этого оператора.

1.4.10. Функции пользователя

Хотя ядро новых версий системы MATLAB содержит уже более 1000 встроенных функций (не считая функций, определенных в десятках пакетов расширения), всегда может понадобиться какая-то нужная пользователю функция. Язык программирования системы MATLAB предоставляет ряд возможностей для задания функций пользователя. Одна из таких возможностей заключается в применении функции `inline`, аргументом которой надо в апострофах задать выражение, задающее функцию одной или нескольких переменных. В приведенном ниже примере задана функция двух переменных – суммы квадратов $\sin(x)$ и $\cos(y)$:

```
>> sc2=inline('sin(x).^2+cos(y).^2')
sc2 =
    Inline function:
    sc2(x,y) = sin(x).^2+cos(y).^2
```

Можно также задавать свои функции в виде m-файлов. Например, можно в окне редактора m-файлов (открывается командой **New** в меню **File**) создать m-файл с именем `sc2` и листингом:

```
function y=sc2(x,y)
y=sin(x).^2+cos(y).^2
```

Записав его на диск, можно командой `type sc2` вывести листинг созданной функции:

```
>> type sc2
function y=sc2(x,y)
    y=sin(x).^2+cos(y).^2
```

Обращение к функции, созданной описанными методами, задается как `sc2(x,y)`, где на место `x` и `y` подставляются значения переменных – аргументов функции пользователя. Например:

```
>> sc2(1,2)
ans = 0.8813
>> sc2(2,1)
y = 1.1187
ans = 1.1187
```

Можно также создать так называемую *handle-функцию* (именуемую также *анонимной функцией*) с помощью оператора `@`:

```
>> fh=@sc2;
```

К такой функции можно обращаться с помощью функции исполнения функций `feval(fh,x,y)`:

```
>> feval(fh,1,2)
y = 0.8813
ans = 0.8813
>> feval(fh,2,1)
y = 1.1187
ans = 1.1187
```

1.4.11. Сообщения об ошибках и исправление ошибок

Большое значение при диалоге с системой MATLAB и отладке программ в ней имеет *диагностика ошибок*. Рассмотрим ряд примеров, поясняющих технику диагностики. Введем, к примеру, ошибочное выражение

```
>> sqr(2)
```

и нажмем клавишу **ENTER**. Система сообщит об ошибке:

```
??? Undefined function or variable 'sqr'.
```

Это сообщение говорит о том, что не определена переменная или функция, и указывает, какая именно, – `sqr`. В данном случае, разумеется, можно просто набрать правильное выражение. Однако в случае громоздкого выражения лучше воспользоваться редактором. Для этого достаточно нажать клавишу \downarrow для перелистывания предыдущих строк. В результате в строке ввода появится выражение

```
>> sqr(2)
```

с курсором в его конце. В MATLAB можно теперь нажать клавишу **Tab**. Система введет подсказку, анализируя уже введенные символы. Из предложенных системой трех операторов выбираем **sqr**. Теперь с помощью клавиши \downarrow вновь выбира-

ем нужную строку и, пользуясь клавишей ←, устанавливаем курсор после буквы **r**. Теперь нажмем клавишу **T**, а затем клавишу **ENTER**. Выражение примет следующий вид:

```
>> sqrt(2)
ans = 1.4142
```

Если бы был только один вариант окончания введенных символов, то после нажатия клавиши **Tab** система бы закончила наш ввод без перевода строки. Вычисления дают ожидаемый результат – значение квадратного корня из двух.

В системе MATLAB внешние определения используются точно так же, как и встроенные функции и операторы. Никаких дополнительных указаний на их применение делать не надо. Достаточно лишь позаботиться о том, чтобы используемые определения действительно существовали в виде файлов с расширением **.m**. Впрочем, если вы забудете об этом или введете имя несуществующего определения, то система отреагирует на это звуковым сигналом (звонок) и выводом сообщения об ошибке:

```
>> hsin(1)
??? Undefined function or variable 'hsin'.
>> sinh(1)
ans = 1.1752
```

В этом примере мы забыли (нарочно), какое имя имеет внешняя функция, вычисляющая гиперболический синус. Система подсказала, что функция или переменная с именем **hsin** не определена – ни как внутренняя, ни как **m**-функция. Зато далее мы видим, что функция с именем **sinh** есть в составе функций системы MATLAB – она задана в виде **M**-функции, хранящейся на жестком диске. Между тем в последнем примере мы не давали системе никаких указаний на то, что следует искать именно внешнюю функцию! И это вычисление прошло так же просто, как вычисление встроенной функции, такой как **sin**.

Иногда в ходе вывода результатов вычислений появляется сокращение **NaN** (от слов *Not a Number* – не число). Оно обозначает неопределенность, например вида $0/0$ или Inf/Inf , где **Inf** – системная переменная со значением машинной бесконечности. Могут появляться и различные предупреждения об ошибках (на английском языке). Например, при делении на 0 конечного числа появляется предупреждение «Warning: Divide by Zero.» («Внимание: деление на ноль»). Диапазон чисел, представимых в системе, лежит от 10^{-308} до 10^{+308} .

Вообще говоря, в MATLAB надо отличать *предупреждение* об ошибке от *сообщения* о ней. *Предупреждения* (обычно после слова **Warning**) не останавливают вычисления и лишь предупреждают пользователя о том, что диагностируемая ошибка способна повлиять на ход вычислений. *Сообщение* об ошибке (после знаков **???**) останавливает вычисления. Система контроля за ошибочными ситуациями в MATLAB 2007b была существенно переработана и стала более корректной.

1.5. Формирование векторов и матриц

1.5.1. Задания векторов и матриц и доступ к их элементам

MATLAB – система, специально предназначенная для проведения сложных вычислений с векторами, матрицами и массивами. При этом она по умолчанию предполагает, что каждая заданная переменная – это вектор, матрица или массив. Все определяется конкретным значением переменной. Например, если задано $X=1$, то это значит, что X – это вектор с единственным элементом, имеющим значение 1, а точнее даже матрица с размером 1×1 . Если надо задать вектор из трех элементов, то их значения следует перечислить в квадратных скобках, разделяя пробелами или запятыми. Так, например, присваивание

```
>> V=[1 2 3]
V = 1 2 3
```

задает вектор V , имеющий три элемента со значениями 1, 2 и 3 (его можно считать и матрицей размера 3×1). После ввода вектора система выводит его на экран дисплея. Заметим, для вектора столбца нужно разделять элементы знаками «;» (точка с запятой):

```
>> V=[1; 2; 3]
V =
     1
     2
     3
```

Задание матрицы требует указания нескольких строк и нескольких столбцов. Для разграничения строк используется знак «;» (точка с запятой). Этот же знак в конце ввода предотвращает вывод матрицы или вектора (и вообще любой операции) на экран дисплея. Так, ввод

```
>> M=[1 2 3; 4 5 6; 7 8 9];
```

задает квадратную матрицу, которую можно вывести:

```
>> M
M =
     1     2     3
     4     5     6
     7     8     9
```

Возможен ввод элементов матриц и векторов в виде арифметических выражений, содержащих любые доступные системе функции, например:

```
>> V= [2+2/(3+4), exp(5), sqrt(10)];
>> V
V = 2.2857    148.4132    3.1623
```

Для указания отдельного элемента вектора или матрицы используются выражения вида $V(i)$ или $M(i, j)$. Например, если задать

```
>> M(2, 2)
ans = 5
```

то результат будет равен 5. Если нужно присвоить элементу $M(i, j)$ новое значение x , следует использовать выражение

```
M(ij)=x
```

Например, если элементу $M(2, 2)$ надо присвоить значение 10, следует записать

```
>> M(2, 2)=10
```

Вообще говоря, в тексте программ MATLAB лучше не использовать i и j как индексы, так как i и j – обозначение квадратного корня из -1 . Но можно использовать I и J .

Выражение $M(i)$ с одним индексом дает доступ к элементам матрицы, развернутым в один столбец. Такая матрица образуется из исходной, если подряд выписать ее столбцы. Следующий пример поясняет подобный доступ к элементам матрицы M :

```
>> M=[1 2 3; 4 5 6; 7 8 9]
```

```
M =
```

```
     1     2     3
     4     5     6
     7     8     9
```

```
>> M(2)
```

```
ans = 4
```

```
>> M(8)
```

```
ans = 6
```

```
>> M(9)
```

```
ans = 9
```

```
>> M(5)=100;
```

```
>> M
```

```
M =
```

```
     1     2     3
     4    100     6
     7     8     9
```

Здесь уместно отметить, что размер векторов и матриц в данной книге учебного характера ограничен. Однако система MATLAB способна работать с очень большими векторами и матрицами. Например, последняя версия MATLAB 2007b может работать с матрицами размера $n \times n$, где максимальное значение $n = 2^{48} - 1$, тогда как предшествующие версии имели максимальное значение $n = 2^{31}$ (это тоже очень большое значение, но куда меньшее, чем у MATLAB 2007b). При этом размеры файла, который может хранить матрицу, могут достигать 18 Гб.

1.5.2. Задание векторов и матриц с комплексными элементами

Из курса математики [54] читатель знает о существовании комплексных чисел вида $a + b * i$, где a – действительная часть числа, b – мнимая часть и i – мнимая

единица (корень квадратный из -1). Возможно задание векторов и матриц с комплексными элементами, например:

```
>> i=sqrt(-1);
>> CM = [1 2; 3 4] + i*[5 6; 7 8]
```

или

```
>> CM = [1+5*i 2+6*i; 3+7*i 4+8*i]
```

Это создает матрицу:

```
CM =
    1.0000 + 5.0000i    2.0000 + 6.0000i
    3.0000 + 7.0000i    4.0000 + 8.0000i
```

Разумеется, возможно разделение элементов не только пробелами, но и запятыми.

1.5.3. Понятие о матричных операциях и магические матрицы

Наряду с операциями над отдельными элементами матриц и векторов система позволяет производить операции умножения, деления и возведения в степень сразу над всеми элементами, то есть над массивами. Для этого перед знаком операции ставится точка. Например, оператор $*$ означает умножение для векторов или матриц, а оператор $.*$ – поэлементное умножение всех элементов массива. Так, если M – матрица, то $M.*2$ даст матрицу, все элементы которой умножены на скаляр – число 2. Впрочем, для умножения матрицы на скаляр оба выражения – $M*2$ и $M.*2$ – оказываются эквивалентными.

Имеется также ряд особых функций для задания векторов и матриц. Например, функция `magic(n)` задает магическую матрицу размера $n \times n$, у которой сумма всех столбцов, всех строк и даже диагоналей равна одному и тому же числу:

```
>> M=magic(4)
M =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> sum(M)
ans = 34  34  34  34

>> sum(M')
ans = 34  34  34  34

>> sum(diag(M))
ans = 34

>> M(1,2)+M(2,2)+M(3,2)+M(4,2)
ans = 34
```

Уже сама по себе возможность создания такой матрицы с помощью простой функции `magic` заинтересует любителей математики. Но векторных и матричных функций в системе множество, и мы их детально рассмотрим в дальнейшем.

Напомним, что для стирания переменных из рабочей области памяти служит команда `clear`.

1.5.4. Конкатенация (объединение) матриц

Описанный способ задания матриц позволяет выполнить операцию *конкатенации* – объединения малых матриц в большую матрицу. Например, создадим вначале магическую матрицу размера 3×3:

```
>> A=magic(3)
```

```
A =
```

```
     8     1     6
     3     5     7
     4     9     2
```

Теперь можно построить матрицу, содержащую четыре матрицы:

```
>> B=[A A+16;A+32 A+16]
```

```
B =
```

```
     8     1     6    24    17    22
     3     5     7    19    21    23
     4     9     2    20    25    18
    40    33    38    24    17    22
    35    37    39    19    21    23
    36    41    34    20    25    18
```

Полученная матрица имеет уже размер 6×6. Вычислим сумму ее столбцов:

```
>> sum(B)
```

```
ans = 126    126    126    126    126    126
```

Любопытно, что она одинакова для всех столбцов. А для вычисления суммы строк используем команду

```
>> sum(B, '')
```

```
ans = 78    78    78    174    174    174
```

Здесь запись `B, ''` означает транспонирование матрицы `B`, то есть замену строк столбцами. На этот раз сумма оказалась разной. Это отвергает изначально возникшее предположение, что матрица `B` тоже является магической. Для истинно магической матрицы суммы столбцов и строк должны быть одинаковыми:

```
>> D=magic(6)
```

```
D =
```

```
    35     1     6     26    19    24
     3    32     7     21    23    25
    31     9     2     22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
```

```
>> sum(D)
```

```
ans = 111    111    111    111    111    111
```

```
>> sum(D, '')
```

```
ans = 111    111    111    111    111    111
```

Более того, для магической матрицы одинаковой является и сумма элементов по основным диагоналям (главной диагонали и главной антидиагонали).

1.5.5. Удаление столбцов и строк матриц

Для формирования матриц и выполнения ряда матричных операций возникает необходимость удаления отдельных столбцов и строк матрицы. Для этого используются пустые квадратные скобки – []. Проделаем это с матрицей M:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
```

```
M =
     1     2     3
     4     5     6
     7     8     9
```

Удалим второй столбец, используя оператор : (двоеточие):

```
>> M(:,2)=[ ]
```

```
M =
     1     3
     4     6
     7     9
```

А теперь, используя оператор : (двоеточие), удалим вторую строку:

```
>> M(2,:)=[ ]
```

```
M =
     1     3
     7     9
```

1.6. Операции с рабочей областью, текстом сессии и редактором m-файлов

1.6.1. Дефрагментация рабочей области

Переменные и определения новых функций в системе MATLAB хранятся в особой области памяти, именуемой *рабочей областью*. По мере задания одних переменных и стирания других рабочая область перестает быть непрерывной и начинает содержать «дыры» и всякий «мусор». Это рано или поздно может привести к ухудшению работы системы или даже к нехватке оперативной памяти. Правда, подобная ситуация становится возможной, если вы работаете с достаточно большими массивами данных.

Во избежание непроизводительных потерь памяти при работе с объемными данными (а векторы, матрицы и массивы относятся к таковым) следует использовать команду `pack`, осуществляющую дефрагментацию рабочей области. Эта команда переписывает все определения рабочей области на жесткий диск, очищает

рабочую область и затем заново считывает все определения без «дыр» и «мусора» в рабочую область.

1.6.2. Сохранение рабочей области сессии

MATLAB позволяет сохранять значения переменных в виде бинарных файлов с расширением **.mat**. Для этого служит команда `save`, которая может использоваться в ряде форм:

- `save fname` – записывается рабочая область всех переменных в файле бинарного формата с именем **fname.mat**;
- `save fname X` – записывает только значение переменной `X`;
- `save fname X Y Z` – записывает значения переменных `X`, `Y` и `Z`.

После параметров команды `save` можно указать ключи, уточняющие формат записи файлов:

- `-mat` – двоичный MAT-формат, используемый по умолчанию;
- `-ascii` – ASCII-формат единичной точности (8 цифр);
- `-ascii -double` – ASCII-формат двойной точности (16 цифр);
- `-ascii -double -tabs` – формат с разделителем и метками табуляции;
- `v4` – запись MAT-файла в формате версии MATLAB 4;
- `-append` – добавление в существующий MAT-файл.

Возможно использование слова `save` и в формате функции, а не команды, например:

```
save('fname', 'var1', 'var2')
```

В этом случае имена файлов и переменных задаются строковыми константами.

Следует отметить, что возможности сохранения всего *текста сессии*, формируемой в командном режиме, команда `save` не дает. Если же это нужно – используется команда `diary`, описанная ниже.

1.6.3. Ведение дневника

Мы отмечали, что сессии не записываются на диск стандартной командой `save`. Однако если такая необходимость есть, можно воспользоваться специальной командой для ведения так называемого *дневника сессии*:

- `diary file_name` – ведет запись на диск всех команд в строках ввода и полученных результатов в виде текстового файла с указанным именем;
- `diary off` – приостанавливает запись в файл;
- `diary on` – вновь начинает запись в файл.

Таким образом, чередуя команды `diary off` и `diary on`, можно сохранять нужные фрагменты сессии в их формальном виде. Команду `diary` можно задать и в виде функции `diary('file')`, где строка `'file'` задает имя файла. Следующий пример поясняет технику применения команды `diary`:

```
>> diary myfile.m
>> 1+2
ans = 3
```

```
>> diary off
>> 2+3
ans = 5
>> diary on
>> sin(1)
ans = 0.8415
>> diary off
```

Нетрудно заметить, что в данном примере первая операция $1 + 2 = 3$ будет записана в файл **myfile.m**, вторая операция $2 + 3 = 5$ не будет записана, третья операция $\sin(1) = 0,8415$ снова будет записана. Таким образом, будет создан Script-файл следующего вида:

```
1+2
ans = 3
diary off
sin(1)
ans = 0.8415
diary off
```

Он приведен в том виде, как записан, то есть с пробелами между строк. Одна из распространенных ошибок начинающих пользователей – попытка запустить подобный файл в командной строке указанием его имени:

```
>> myfile
??? ans =
Missing variable or function.
Error in ==> C:\MATLAB\bin\myfile.m
On line 3 ==> ans =
```

Обычно это приводит к ошибкам, так как данный файл – это просто текстовая запись команд и результатов их выполнения, не проверяемая на корректность и содержащая ряд строк, ошибочных с позиций синтаксиса языка программирования MATLAB – например, выражения `ans =`. Зато команда `type` позволяет просмотреть текст такого файла со всеми записанными действиями:

```
>> type myfile
1+2
ans = 3
diary off
sin(1)
ans = 0.8415
diary off
```

Во избежание отмеченных казусов рекомендуется записывать файл с расширением, отличным от **.m**, например **.txt**. Это позволит встраивать подобные текстовые файлы дневника сессии в документы, содержащие ее описание.

1.6.4. Загрузка рабочей области сессии

Для загрузки рабочей области ранее проведенной сессии (если она была сохранена) можно использовать команду `load`:

- `load fname ...` – загрузка ранее сохраненных в файле **fname.mat** определений со спецификациями на месте многоточия, подобными описанным для

команды `save` (включая ключ `-mat` для загрузки файлов с расширением `.mat` обычного бинарного формата, используемого по умолчанию);

- `load('fname', ...)` – загрузка файла **fname.mat** в форме функции.

Если команда (или функция) `load` используется в ходе проведения сессии, то произойдет замена текущих значений переменных теми значениями, которые были сохранены в считываемом МАТ-файле.

Для задания имен загружаемых файлов может использоваться знак `*`, означающий загрузку всех файлов с определенными признаками. Например, `load demo*.mat` означает загрузку всех файлов с началом имени **demo**, например **demo1**, **demo2**, **demoa**, **demob** и т. д. Имена загружаемых файлов можно формировать с помощью операций над строковыми выражениями.

1.6.5. Работа с редактором *m*-файлов

Любую последовательность команд в MATLAB можно оформить в виде *m*-файла, называемого скрипт-файлом. Для создания и редактирования таких файлов служит специальный редактор *m*-файлов. Его пустое окно открывается командой **New (Новый файл)**, которую можно ввести активизацией кнопки с тем же названием в панели инструментов или из позиции **File** меню окна MATLAB.

К примеру, введем такой скрипт-файл вычисления суммы чисел 2 и 3 и построения графика синусоиды:

```
2+3
x=0:0.1:15
y=sin(x)
plot(x,y)
```

Пример ввода листинга этого файла в окне редактора/отладчика *m*-файла показан на рис. 1.6. Введенный файл можно пустить из окна редактора, исполнив команду **Run** в позиции **Debug (Отладка)** меню окна редактора. В результате будет вычислено выражение $2 + 3$ и число 5 появится в окне сессии MATLAB. Будет также построен в отдельном окне график синусоидальной функции. Все это и видно на рис. 1.6.

Редактор/отладчик *m*-файлов – это в сущности специализированный текстовый редактор, предназначенный для записи и отладки программ на языках системы MATLAB, отдельных их фрагментов, процедур и функций. Строки листинга нумеруются, и в них можно вставлять специальные точки останова для отладки сложных программ. В этих точках можно контролировать и изменять значения переменных.

1.6.6. Завершение вычислений и работы с системой

Иногда из-за ошибок в программе или из-за сложности решаемой задачи MATLAB «зацикливается» и перестает выдавать результаты, либо непрерывно выдает их, хотя в этом уже нет необходимости. Для прерывания вычислений в этом случае достаточно нажать одновременно клавиши **Ctrl** и **C** (латинскую).

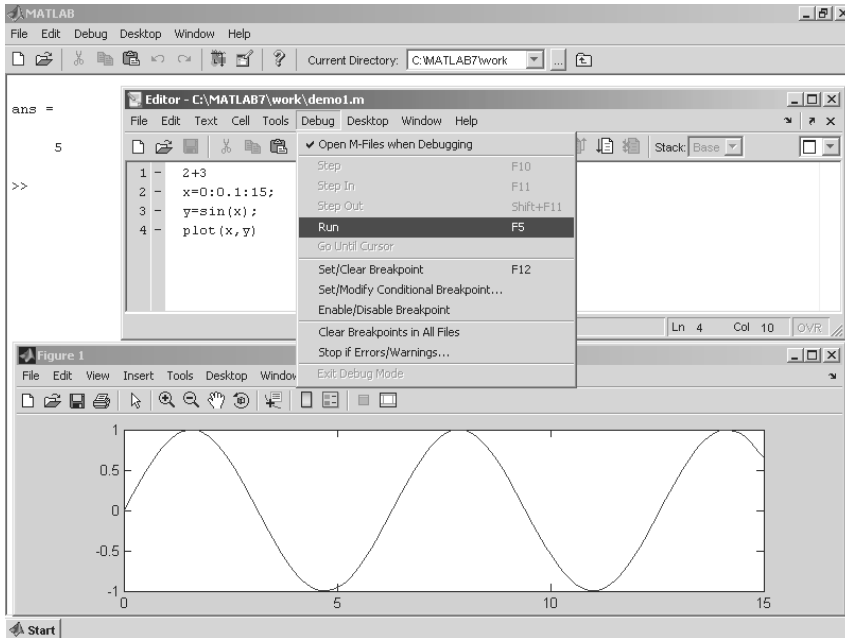


Рис. 1.6. Пример задания *m*-файла построения графика синусоиды

Для завершения работы с системой можно использовать команды `quit`, `exit` или комбинацию клавиш **Ctrl+Q**. Если необходимо сохранить значения всех переменных (векторов, матриц) системы, то перед этим следует дать команду `save` нужной формы. Команда `load` после загрузки системы считывает значения этих переменных и позволяет начать работу с системой с того момента, когда она была прервана.

1.7. Интерактивная справка из командной строки

1.7.1. Вызов списка разделов интерактивной справки

MATLAB имеет интерактивную систему помощи, которая реализуется в командном режиме с помощью ряда команд. Одной из них является команда

```
>> help
HELP topics:
matlab\general      - General purpose commands.
Matlab\ops          - Operators and special characters.
Matlab\lang         - Programming language constructs.
```

```

Matlab\elmat      - Elementary matrices and matrix
                  manipulation.
Matlab\elfun      - Elementary math functions.
Matlab\specfun    - Specialized math functions.
.....

```

Она выводит весь список папок, содержащих m-файлы с определениями операторов, функций и иных объектов, присущих конкретной реализации системы MATLAB. Ввиду большого размера списка приведены только первые несколько его строк. Рекомендуется просмотреть этот список для установленной на ПК пользователя системы MATLAB.

Следует отметить, что набор входящих в список средств зависит от набора пакетов расширения, которыми располагает конкретная версия системы MATLAB, заказанная или полученная пользователем. Кроме того, для доступа к пакетам расширения может потребоваться указание пути к их файлам на диске в панели браузера файловой системы.

1.7.2. Справка по конкретному объекту

Для получения справки по какому-либо конкретному объекту используются команды

```

>> help имя
или
>> doc имя

```

где имя – имя объекта, для которого требуется вывод справочной информации. Мы уже приводили пример помощи по разделу операторов ops. Ниже дается пример для функции вычисления гиперболического синуса, намеренно введенной с неверным указанием имени:

```

>> help hsin
hsin.m not found.

```

Нетрудно заметить, что система помощи сообщает, что для функции с именем hsin соответствующий m-файл отсутствует. Введем имя верно:

```

>> help sinh
    SINH Hyperbolic sine.
        SINH(X) is the hyperbolic sine of the elements of X.
Overloaded methods
    help sym/sinh.m

```

Теперь полученное сообщение содержит информацию о функции sinh. Сообщается, что данная функция возвращает значение гиперболического синуса для элементов вектора **X**.

Хотя имена функций в MATLAB задаются малыми (строчными) буквами, в сообщении справочной системы имена функций и команд выделяются большими (прописными) буквами. Этот не слишком удачный прием использован для выделения заголовка текста справки в виде имени функции. В данной книге мы отказались от такого приема, вводящего начинающих пользователей в заблуждение.

Аналогичным образом можно получить справку по константам и другим объектам языка MATLAB. Ниже дан пример обращения к справке о числе π :

```
>> help pi
      PI              3.1415926535897
      PI = 4*atan(1) = imag(log(-1)) = 3.1415926535897
```

При всей примитивности справки `help` надо отметить ее высокую эффективность. Особенно популярна интерактивная справка у пользователей, привыкших к работе с языками программирования, которые используются в среде операционной системы MS-DOS. Справка `doc имя` выводит более полную информацию в окне помощи в формате HTML.

1.7.3. Справка по группе объектов

Пользователя системы MATLAB часто интересует набор функций, команд или иных понятий, относящихся к определенной группе объектов. Выше были указаны имена основных групп объектов системы MATLAB. Ниже дан пример вызова справки по группе объектов `timefun`:

```
>> help timefun
Time and dates.
Current date and time.
  Now          - Current date and time as date number.
  Date         - Current date as date string.
  Clock        - Current date and time as date vector.
Basic functions.
  Datenum      - Serial date number.
  Datestr      - String representation of date.
  Datevec      - Date components.
Date functions.
  Calendar     - Calendar.
  Weekday      - Day of week.
  Eomday       - End of month.
  Datetick     - Date formatted tick labels.
Timing functions.
  Cputime      - CPU time in seconds.
  Tic          - Start stopwatch timer.
  Toc          - Stop stopwatch timer.
  Etime        - Elapsed time.
  Pause        - Wait in seconds.
```

После уточнения состава определенной группы объектов можно получить детальную справку по любому выбранному объекту. Как это делается, было описано выше. Обратите особое внимание на приведенные выше временные функции – они широко используются для оценки скорости выполнения тех или иных команд и вычислений.

1.7.4. Справка по ключевому слову

Ввиду обилия в системе MATLAB m-функций важное значение имеет поиск m-функций по *ключевым словам*. Для этого служит команда

```
lookfor Ключевое слово
```

или

```
lookfor 'Ключевые слова'
```

В первом случае ищутся все m-файлы, в заголовках которых встречается заданное ключевое слово, и заголовки обнаруженных файлов выводятся на экран. Следует отметить, что широкий поиск по одному ключевому слову может подчас привести к выводу многих десятков определений и длится довольно долго.

Для уточнения и сокращения зоны поиска следует использовать вторую форму команды lookfor. Вот пример ее применения:

```
>> lookfor 'inverse sin'
```

```
ASIN Inverse sine.
```

```
IS2RC Convert inverse sine parameters to reflection coefficients.
```

```
RC2IS Convert reflection coefficients to inverse sine parameters.
```

```
ASIN Symbolic inverse sine.
```

В данном случае для поиска использованы слова 'inverse sin', то есть задан поиск арксинуса. Число найденных определений зависит от того, с каким числом пакетов прикладных программ (пакетов расширений) поставляется версия системы MATLAB.

В следующем уроке мы рассмотрим гораздо более эффективные средства справочной системы, ориентированные на работу в стиле приложений операционных систем Windows 95/98/Me/2000/NT4/XP/Vista с графическим пользовательским интерфейсом.

1.7.5. Дополнительные справочные команды

В командном режиме можно получить справочные данные с помощью ряда команд:

- `computer` – выводит сообщение о типе компьютера, на котором установлена текущая версия MATLAB;
- `help script` – выводит сообщение о назначении m-файлов сценариев (Script-файлов);
- `help function` – выводит сообщение о назначении и структуре m-файлов функций;
- `info` – выводит информацию о фирме MathWorks с указанием адресов электронной почты;
- `subscribe` – позволяет создать файл с бланком регистрации;

- `ver` – выводит информацию о версиях установленной системы MATLAB и ее компонентов;
- `version` – выводит краткую информацию об установленной версии MATLAB;
- `version -java` – выводит информацию об установленной в составе MATLAB версии Ява (Java);
- `what` – выводит имена файлов текущего каталога;
- `what name` – выводит имена файлов каталога, заданного именем `name`;
- `whatsnew name` – выводит на экран содержимое файлов `readme` заданного именем `name` класса для знакомства с последними изменениями в системе и в пакетах прикладных программ;
- `which name` – выводит путь доступа к функции с данным именем.

Как правило, эти команды выводят довольно обширные сообщения, например команда `ver` выводит данные о текущей версии MATLAB и всех ее пакетов расширения. Краткую информацию о текущей версии выводит команда `version`. Например, для версии MATLAB R2007a:

```
>> version
ans =
7.4.0.287 (R2007a)
```

1.8. Работа с демонстрационными примерами с командной строки

1.8.1. Вызов списка демонстрационных примеров

Одним из самых эффективных методов знакомства со сложными математическими системами является ознакомление со встроенными примерами их применения. Система MATLAB содержит многие сотни таких примеров – по примеру практически на каждый оператор или функцию. Наиболее поучительные примеры можно найти в разделе `demos` справки или выполнив команду:

```
>> help demos
Examples and demonstrations.
  Type 'demo' at the command line to browse more demos of
  MATLAB, the Toolboxes, and Simulink.
  demo                - Run demonstrations.
Mathematics.
  intro                - Basic Matrix Operations
  inverter             - Inverses of Matrices
  buckydem            - Graphs and Matrices
  sparsity             - Sparse Matrices
  matmanip            - Matrix Manipulation
  integerMath         - Integer Arithmetic Examples
  .....
```

Здесь весьма длинный список примеров обрезан. Мы настоятельно рекомендуем пользователям системы MATLAB просмотреть с десятков примеров из интересующих их областей. Список примеров может несколько различаться для различных версий MATLAB.

1.8.2. Пример – вывод изображения поверхности

Мы уже приводили наглядный пример вызова из командной строки примера на вычисление скорости работы системы MATLAB. Исполнив команду

```
>> wernerboy
```

можно наблюдать изображение сложной поверхности Вернера–Боя, показанной на рис. 1.7 в окне графики.

Это построение прекрасно иллюстрирует технику функциональной окраски сложных поверхностей и фигур, именуемую рендерингом. Данная техника обеспечивает высокую степень реалистичности поверхностей с учетом условий освещения и свойств отражения света от материалов с определенными свойствами.

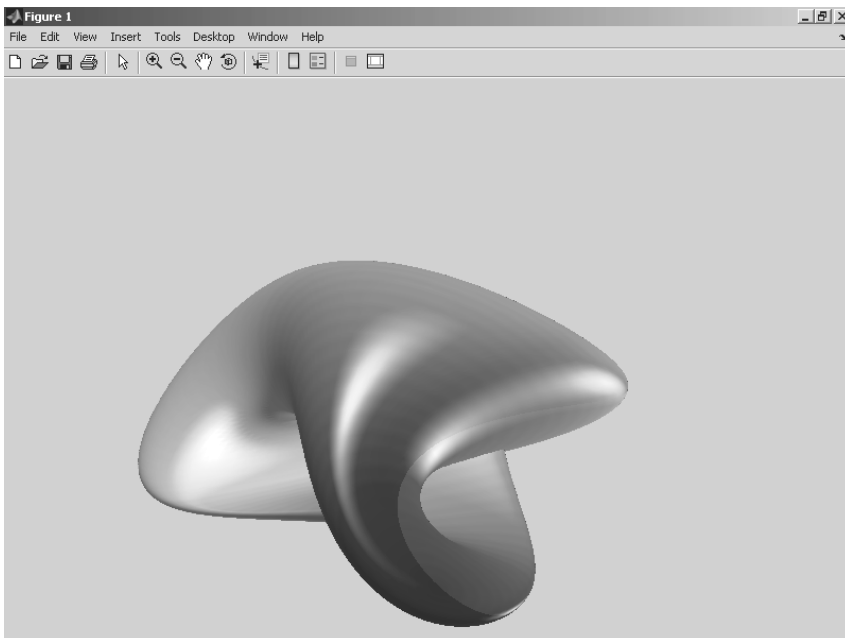


Рис. 1.7. Вывод изображения поверхности Вернера–Боя

1.8.3. Что больше – e^{π} или π^e ?

Рассмотрим еще один простой пример, дающий ответ на сакраментальный вопрос о том, какое значение больше – e^{π} или π^e ? Для запуска этого примера надо исполнить команду

```
>> e2pi
```

и наблюдать красочное шоу – графики степенных функций x^y и y^x с построением на них линий заданных функций и оценкой их значений – рис. 1.8. Этот пример – наглядная демонстрация перехода от узких понятий к более широким.

Разумеется, вы могли бы вместо приятного обозрения графического представления примера просто вычислить соответствующие значения:

```
>> e=exp(1)
e = 2.7183
>> e^pi
ans = 23.1407
>> pi^e
ans = 22.4592
```

Так можно легко убедиться в том, что все же e^{π} больше, чем π^e . Можно проверить это и помощью логического оператора сравнения $>$ (результат 1 означает, что неравенство выполняется и дает логическое значение TRUE):

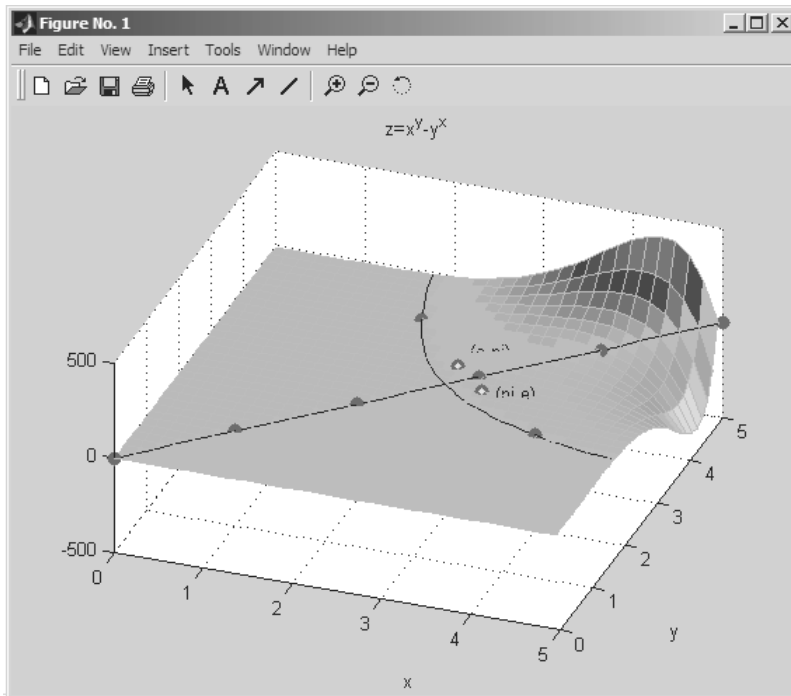


Рис. 1.8. Исполнение примера e2pi

```
>> e^pi>pi^e  
ans = 1
```

1.8.4. Встроенные фигуры

MATLAB имеет ряд встроенных фигур, которые можно легко выводить на построение простым указанием их названия. Так, введя команду `knot`, можно задать построение еще одной сложной пространственной фигуры узла с функциональной окраской (рис. 1.9). Можно убедиться в том, что имеется возможность вращать полученную фигуру. Если в прежних версиях это делалось вместе с обрамляющим фигуру параллелепипедом, то в MATLAB 7 SP2 параллелепипед уже не выводится. В данном примере показан также вывод шкалы цветовых оттенков – справа от фигуры.

1.8.5. Просмотр текстов примеров и m-файлов

Как программная среда MATLAB открыта для пользователя. Любой m-файл системы, например файл демонстрационных примеров, можно просмотреть с помощью любого текстового редактора, редактора и отладчика m-файлов, встроенного в систему, или с помощью команды

`type Имя_М-файла`

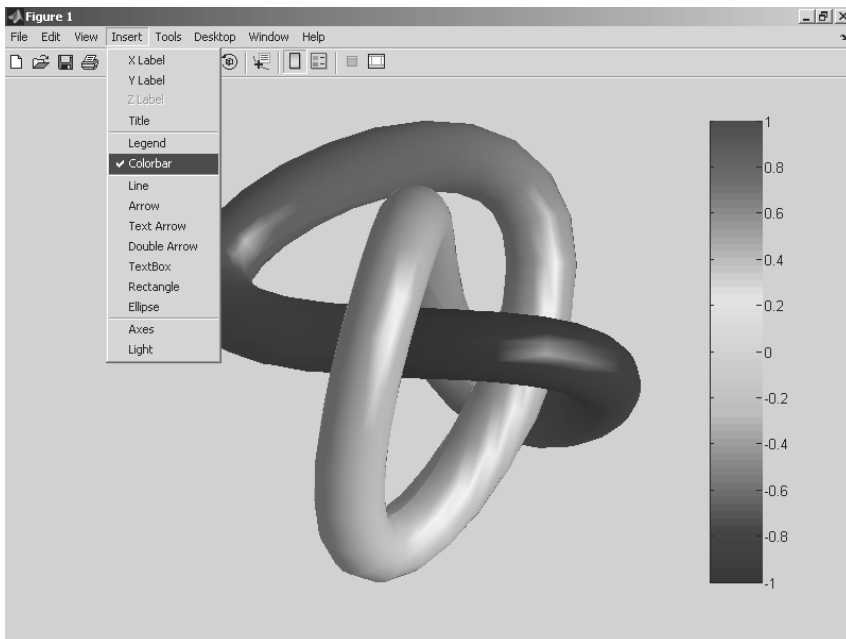


Рис. 1.9. Построение фигуры-узла

Например, если вы хотите просмотреть текст файла демонстрационного примера `e2pi`, то нужно выполнить команду:

```
>> type e2pi
```

Используя команду `help`, можно получить справку по любой конкретной функции или команде.

1.9. Знакомство с двумерной графикой MATLAB

1.9.1. Особенности двумерной графики MATLAB

Для визуализации вычислений в MATLAB широко используется машинная графика [66]. Графика в MATLAB имеется двух типов:

- обычная двумерная и трехмерная растровая графика;
- специальная дескрипторная (*handle*) графика.

Пока мы остановимся на обычной графике. С ней связано представление о *графических объектах*, имеющих определенные свойства. В большинстве случаев об объектах можно забыть, если только вы не занимаетесь объектно-ориентированным программированием задач графики. Связано это с тем, что большинство команд высокоуровневой графики, ориентированной на конечного пользователя, автоматически устанавливают свойства графических объектов и обеспечивают воспроизведение графики в нужной системе координат, палитре цветов, масштабе и т. д. Применение графики MATLAB практически исключает необходимость в сложных математических вычислениях, обычно необходимых для построения графиков и описанных в [66].

Средства графики в новых версиях MATLAB существенно дополнены. Новая позиция **Graphics** меню содержит три команды:

- **New Figure** – открывает пустое окно графики;
- **Plot Tools** – открывает окно нового мощного редактора графики;
- **More Plots...** – открывает окно доступа к различным видам графики.

Первая команда очевидна, а две другие будут детально описаны ниже.

На более низком уровне решения задач используется ориентированная на опытного программиста *дескрипторная графика* (*Handle Graphics*), при которой каждому графическому объекту в соответствие ставится особое описание – *дескриптор*, на который возможны ссылки при использовании графического объекта. Дескрипторная графика позволяет осуществлять визуальное программирование объектов пользовательского интерфейса – управляющих кнопок, текстовых панелей и т. д.

1.9.2. Графики функций одной переменной

Графики в MATLAB строятся в отдельных масштабируемых и перемещаемых окнах. Возьмем вначале простейший пример – построение графика синусоиды. Следует помнить, что MATLAB (как и другие СКМ) строит графики функций по ряду точек, соединяя их отрезками прямых, то есть осуществляя линейную интерполяцию функции в интервале между смежными точками. Зададим интервал изменения аргумента x от 0 до 10 с шагом 0,1. Для построения графика достаточно вначале задать вектор $x=0:0.1:15$, а затем использовать команду построения графиков `plot(sin(x))`.

Итак, для построения графика синусоиды надо исполнить следующие команды:
`x=0:0.1:15; y=sin(x); plot(x,y)`

При этом будут построены окно графика и сам график синусоидальной функции – рис. 1.10. Они идентичны показанным на рис. 1.6, где был дан пример построения такого же графика из *m*-файла, листинг которого введен в окно редактора/отладчика *m*-файлов.

В этих примерах вектор x задает интервал изменения независимой переменной от 0 до 15 с шагом 0,1. Почему взят такой шаг, а не, скажем, 1? Дело в том, что `plot`

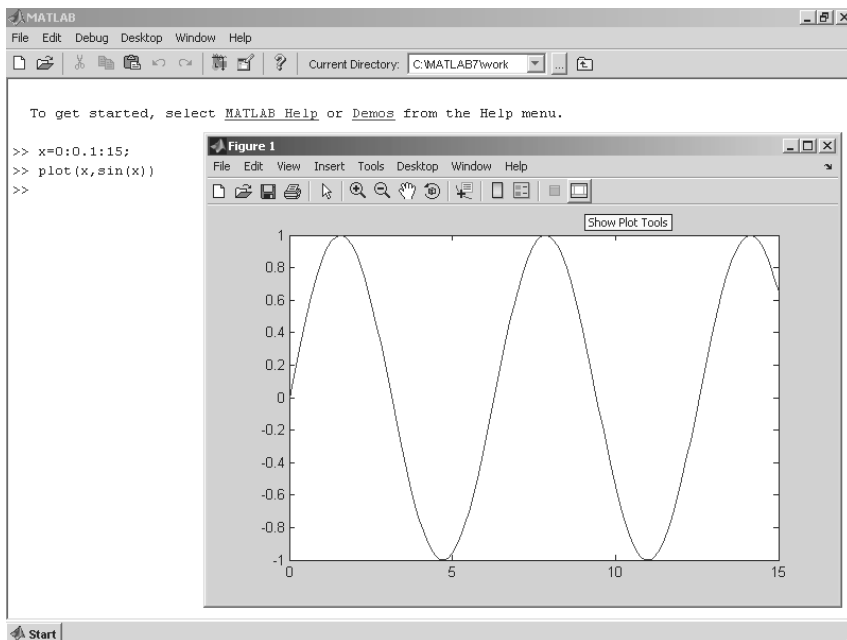


Рис. 1.10. Построение графика синусоиды из окна командного режима работы MATLAB

строит не истинный график функции $\sin(x)$, а лишь заданное числом элементов вектора x число точек. Эти точки затем просто соединяются отрезками прямых того или иного стиля и цвета, то есть осуществляется кусочно-линейная интерполяция данных графика. При 100 точках полученная кривая глазом воспринимается как вполне плавная, но при 10–20 точках она будет выглядеть состоящей из отрезков прямых.

1.9.3. Графики ряда функций

Более подробное описание графического окна будет дано в следующем уроке. А пока пойдем дальше и попытаемся построить графики сразу трех функций: $\sin(x)$, $\cos(x)$ и $\sin(x)/x$. Прежде всего отметим, что эти функции могут быть обозначены переменными, не имеющими явного указания аргумента в виде $y(x)$:

```
>> y1=sin(x); y2=cos(x); y3=sin(x)/x;
```

Такая возможность обусловлена тем, что эти переменные являются векторами – как и переменная x . Теперь можно использовать одну из ряда форм команды `plot`:

```
plot(a1, f1, a2, f2, a3, f3, ...),
```

где a_1, a_2, a_3, \dots – векторы аргументов функций (в нашем случае все они – x), f_1, f_2, f_3, \dots – векторы значений функций, графики которых строятся в одном окне. В нашем случае для построения графиков указанных функций мы должны записать следующее:

```
>> plot(x, y1, x, y2, x, y3)
```

Можно ожидать, что MATLAB в этом случае построит, как обычно, точки графиков этих функций и соединит их отрезками линий. Но, увы, если мы выполним эти команды, то никакого графика не получим вообще. Не исключен даже сбой в работе системы. Причина этого казуса уже обсуждалась в предыдущем уроке – при вычислении функции $y_3 = \sin(x) / x$: если x представляет собой массив (вектор), то нельзя использовать оператор матричного деления `/`.

Этот пример еще раз наглядно указывает на то, что чисто поверхностное применение даже такой мощной системы, как MATLAB, иногда приводит к досадным срывам. Чтобы все же получить график, надо вычислять отношение $\sin(x)$ к x с помощью оператора поэлементного деления массивов `./`. Этот случай поясняет рис. 1.11. Кстати, на нем показана открытой позиция **Tools (Инструменты)** меню графического окна, которая открывает доступ к многочисленным командам форматирования графиков.

Обратите внимание на то, что хотя на этот раз MATLAB построил графики всех трех функций, в окне командного режима появилось предупреждение о делении на 0 – в момент, когда $x=0$ – «Warning: Divide by zero». Таким образом, `plot` «не знает» о том, что неопределенность $\sin(x)/x=0/0$ устранимая и дает 1. Это недостаток практически всех систем для численных вычислений.

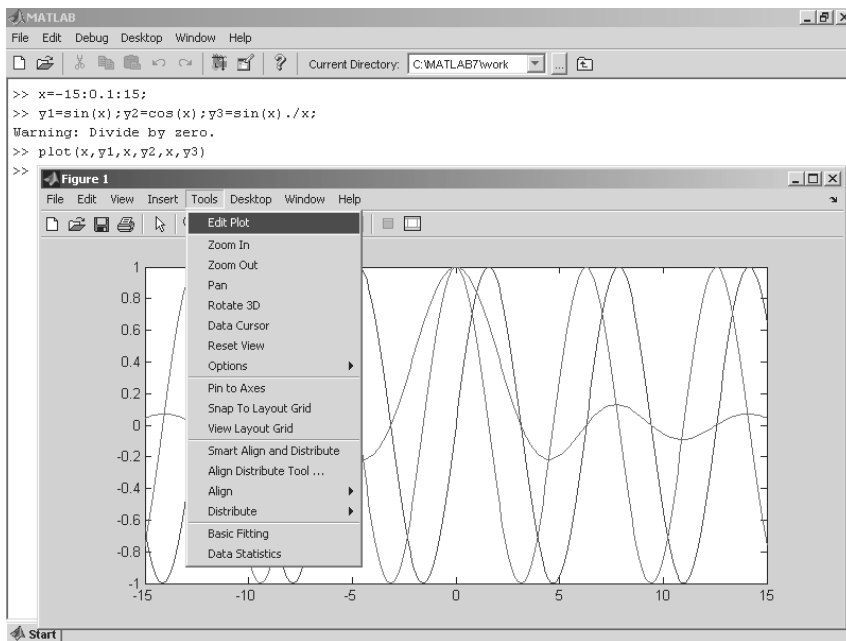


Рис. 1.11. Построение графиков трех функций

1.9.4. Графическая функция *fplot*

Разумеется, MATLAB имеет средства для построения графиков и таких функций, как $\sin(x)/x$, которые имеют устранимые неопределенности. Не обсуждая эти средства подробно, просто покажем, как это делается, с помощью другой графической команды – `fplot`:

```
fplot('f(x)', [xmin xmax])
```

Она позволяет строить график функции $f(x)$, заданной в символьном виде, в интервале изменения аргумента x от x_{\min} до x_{\max} без фиксированного шага изменения x . Один из вариантов ее применения демонстрирует рис. 1.12. Хотя в процессе вычислений предупреждение об ошибке (деление на 0) выводится, но график строится правильно, при $x=0$ $\sin(x)/x=1$. Обратите также внимание на две используемые команды: `clear` (**Очистить**) – очистка графического окна и `grid on` (**Сетка**) – включение отображения сетки, которая строится пунктирными линиями.

На рис. 1.12 представлено также меню **Insert** (**Вставка**) окна графики. С его помощью можно задать вставки в графическое окно различных объектов, например легенд – обозначений кривых графиков, шкалы цветов и т. д. На рис. 1.12 представлены примеры вставки легенды и шкалы цветов `Colorbar`.

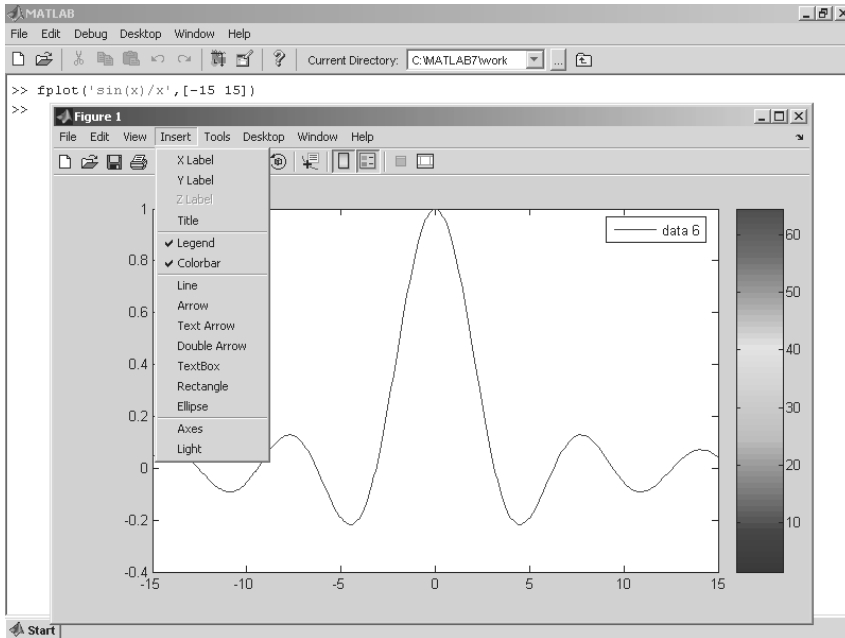


Рис. 1.12. Построение графика $\sin(x)/x$ функцией `fplot`

Обратите внимание и на позицию **File (Файл)** меню окна графики. Она содержит типовые файловые операции. Однако они относятся не к файлам документов, а к файлам графиков. В частности, можно присваивать имя записываемым на диск рисункам с графиками.

1.10. Знакомство с трехмерной графикой MATLAB

1.10.1. Построение трехмерных графиков

Столь же просто обеспечивается построение графиков сложных поверхностей, представленных функцией двух переменных $z=f(x,y)$. Такую графику называют трехмерной, или 3D-графикой. Надо только знать, какой командой реализуется тот или иной график. Например, для построения графика поверхности и ее проекции в виде контурного графика на плоскость под поверхностью достаточно использовать следующий фрагмент программы:

```
% Пример построения поверхности и ее проекции
[X,Y]=meshgrid(-5:0.1:5);
Z=X.*sin(X+Y);
meshc(X,Y,Z)
```

Первая задает разметку сетки будущей поверхности с интервалом изменения x и y от -5 до 5 с шагом $0,1$. Вторая задает выражение для вычисления значений z в узлах сетки. Наконец, третья команда строит собственно график поверхности. Окно с построенным графиком показано на рис. 1.13. Раньше пришлось бы убить много дней на составление и отладку нужной для построения такого графика программы. В MATLAB же можно в считанные секунды изменить задающую поверхность функцию $Z(X, Y)$ и тут же получить новый график поверхности с окраской, в данном случае заданной вектором Z , и с ее проекцией на плоскость XY . На рис. 1.13 показано также открытое меню **Help (Помощь)** окна трехмерной графики.

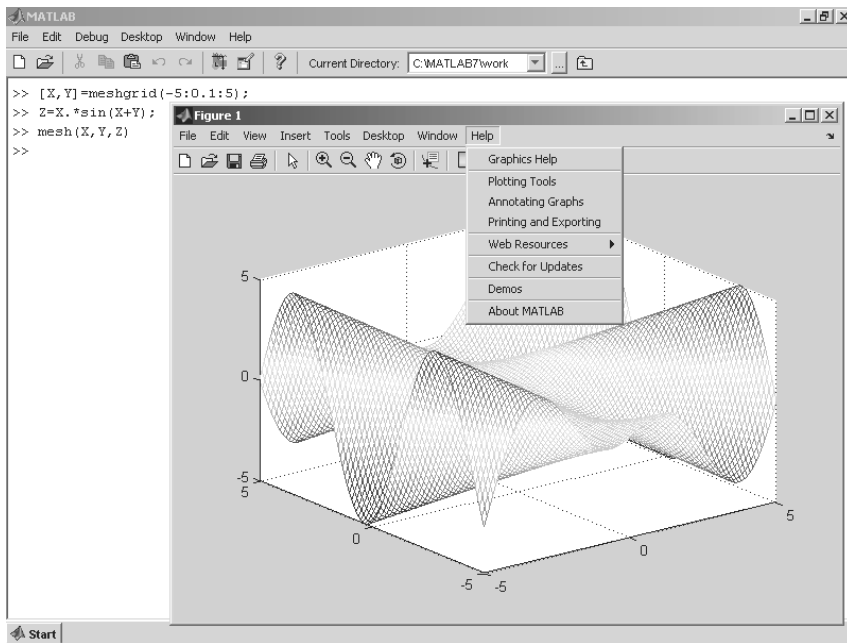


Рис. 1.13. Окно с графиками поверхности и ее проекции на плоскость под фигурой

Мы ограничимся этими примерами построения графиков как достаточно простыми и типовыми. Из них следует важный вывод: для решения той или иной частной задачи надо знать соответствующие команды и функции. В этом вам помогут как данная книга, так и справочная система MATLAB.

1.10.2. Вращение графиков мышью

Можно поворачивать построенную фигуру мышью и наблюдать ее под разными углами. Рассмотрим эту возможность на примере построения логотипа системы MATLAB – мембраны. Для этого, введя команду `logo`, получим исходный график, представленный на рис. 1.14.

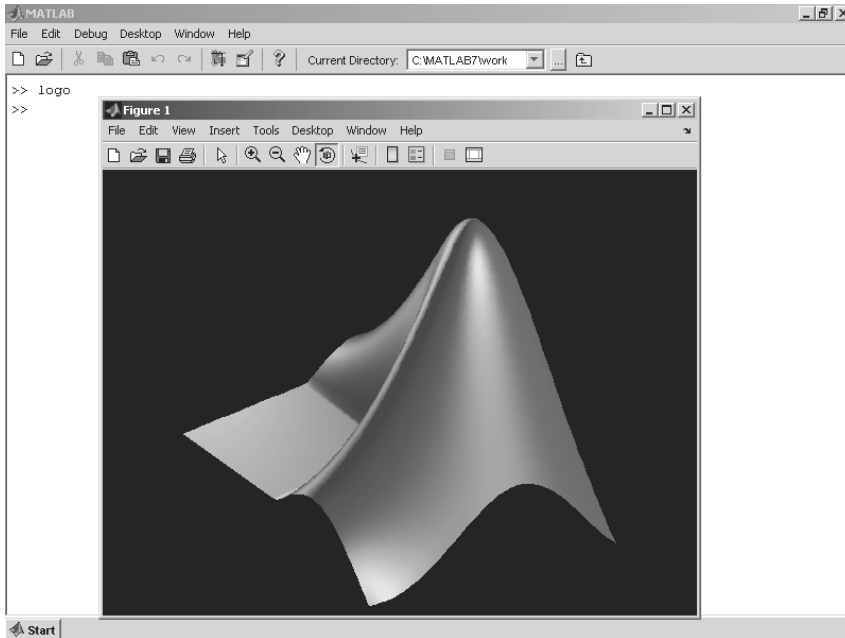


Рис. 1.14. Построение мембраны – логотипа системы MATLAB

Для вращения графика достаточно активизировать последнюю справа кнопку панели инструментов с изображением пунктирной окружности со стрелкой. Теперь, введя курсор мыши в область графика и нажав левую кнопку мыши, можно круговыми движениями заставить график вращаться (рис. 1.15).

Любопытно, что в новых версиях MATLAB вращать можно и двумерные графики, наблюдая поворот плоскости, в которой они построены.

1.10.3. Контекстное меню графиков

Для переключения в режим редактирования графика нужно щелкнуть на кнопке **Edit Plot (Редактировать график)** с изображением курсора-стрелки. В этом режиме графиком можно управлять с помощью контекстного меню, вызываемого щелчком правой кнопки мыши. Вид этого меню при курсоре, расположенном в области трехмерного графика вне построенных трехмерных графических объектов, показан на рис. 1.16. С помощью мыши можно также выделить график. Щелчок левой клавишей выводит набор точек (прямоугольников) в области рисунка (см. рис. 1.16). Теперь на график можно наносить стрелки, поясняющие надписи (кнопка с буквой **A**) и т. д.

Еще раз напоминаем, что контекстное меню правой клавиши мыши позволяет оперативно выполнять любые команды, в том числе и не относящиеся к графике.

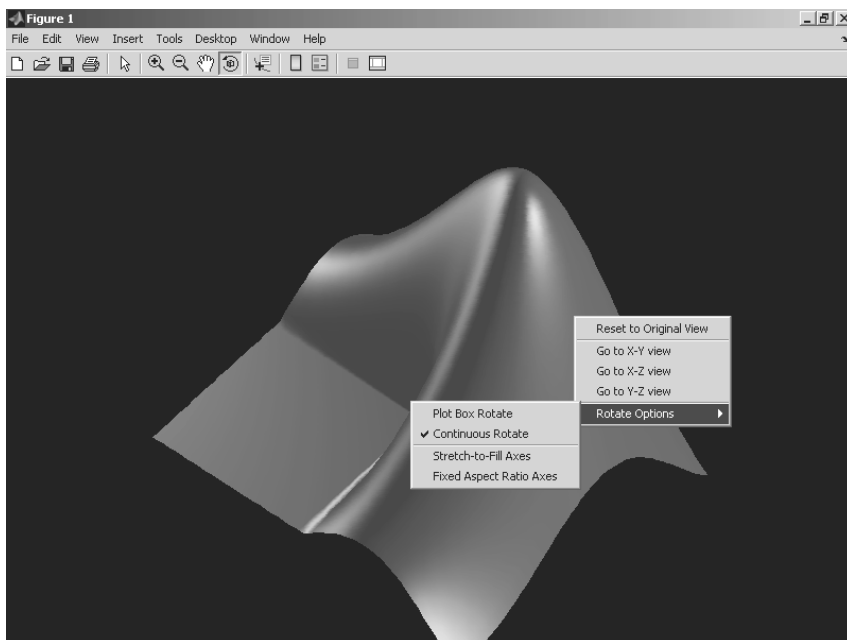


Рис. 1.15. Вращение трехмерной фигуры мышью

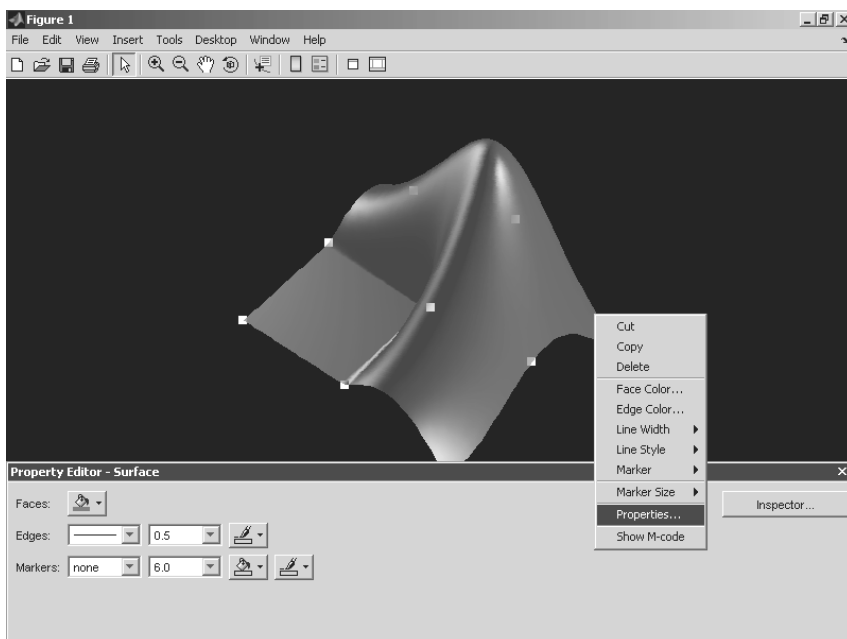


Рис. 1.16. График в состоянии редактирования и контекстное меню

Знакомство с интерфейсом пользователя

| | |
|---|-----|
| 2.1. Интерфейс основного окна MATLAB | 92 |
| 2.2. Работа с меню | 96 |
| 2.3. Основы редактирования и отладки m-файлов | 100 |
| 2.4. Новинки графического интерфейса MATLAB | 104 |
| 2.5. Интерфейс графических окон | 115 |
| 2.6. Основы форматирования графиков | 118 |
| 2.7. Работа с Мастером импорта данных | 130 |
| 2.8. Работа со справкой и демонстрационными примерами | 135 |
| 2.9. Интерфейс и новые возможности MATLAB R2007 | 140 |

Первые версии системы MATLAB имели очень простой интерфейс для работы в командной строке. Но в современных версиях используется новый многооконный интерфейс и имеется множество возможностей для управления им. Этот урок ориентирован на достаточно полное знакомство с интерфейсом системы MATLAB 2006b как наиболее характерным для всех версий системы MATLAB, рассмотренных в данном самоучителе. В конце урока мы рассмотрим интерфейс последних (на момент подготовки данной книги) подверсий системы MATLAB – MATLAB R2007a,b.

2.1. Интерфейс основного окна MATLAB

Вся совокупность средств общения пользователя с системой MATLAB именуется *интерфейсом пользователя*. В уроке 1 мы детально познакомились с одной из компонент этого интерфейса – средствами работы в режиме командной строки. В этом уроке мы продолжим знакомство с другой важной компонентой – *графическим интерфейсом пользователя GUI (Graphics User Interface)*. Он во многом похож на GUI любого приложения под операционную систему Windows, но имеет множество специфических особенностей, присущих именно системе MATLAB [16, 44, 45]. С ними мы познакомимся наиболее подробно (визуально-ориентированному программированию и проектированию GUI посвящен отдельный урок 12).

2.1.1. Средства панели инструментов

Как любая программа, MATLAB имеет основное окно с титульной строкой, строкой меню, панелью инструментов, строкой статуса и другими компонентами. Начинаящим пользователям удобно знакомиться с работой в системе с помощью панели инструментов, расположенной под строкой меню. Она дает наиболее простой и удобный, особенно для начинающих пользователей, способ работы с системой MATLAB.

Панель инструментов основного окна MATLAB довольно проста и содержит знакомые большинству пользователей кнопки (см. рис. 1.1). Ниже они перечислены слева направо:

- **New M-file (Новый m-файл)** – выводит пустое окно редактора m-файлов;
- **Open file (Открыть файл)** – открывает окно для загрузки m-файла;
- **Cut (Вырезать)** – вырезает выделенный фрагмент и помещает его в буфер;
- **Copy (Копировать)** – копирует выделенный фрагмент в буфер;
- **Paste (Вставить)** – переносит фрагмент из буфера в текущую строку ввода;
- **Undo (Отменить)** – отменяет предшествующую операцию;
- **Redo (Повторить)** – восстанавливает последнюю отмененную операцию;
- **Simulink** – открывает окно браузера библиотек Simulink;
- **GUIDE** – открывает окно создания и расширения интерфейса;
- **Help (Помощь)** – открывает окно справки.

Набор кнопок панели инструментов обеспечивает выполнение наиболее часто необходимых команд и вполне достаточен для повседневной работы с системой. О назначении кнопок говорят и всплывающие подсказки, появляющиеся, когда курсор мыши устанавливается на соответствующую кнопку. Они имеют вид желтого прямоугольника с текстом короткой справки.

Кнопка **New M-file** открывает окно редактора/отладчика m-файлов. Работу с этим средством мы обсудим позже. Кнопка **Open file (Открыть файл)** служит для загрузки в редактор/отладчик ранее созданных m-файлов, например входящих в пакет расширения (Toolbox) системы или разработанных пользователем. Она открывает стандартное окно, которое является типичным элементом интерфейса Windows-приложений.

Кнопки **Cut (Вырезать)**, **Copy (Копировать)** и **Paste (Вставить)** реализуют наиболее характерные команды работы с буфером обмена (Clipboard). Первые две операции относятся к выделенным фрагментам сессии или текста m-файлов (если они выполняются в окне редактора/отладчика). Для выделения объектов можно использовать мышь, перемещая курсор по тексту при нажатой левой кнопке, или клавиши со стрелками в комбинации с клавишей **Shift**.

В MATLAB можно использовать контекстное меню, появляющееся при нажатии правой кнопки мыши. Например, установив курсор мыши на выделенный фрагмент матрицы **M** и нажав правую кнопку, можно увидеть меню, показанное на рис. 5.6. В нем, кстати, дублируется позиция с командой **Copy (Копировать)**. Есть и ряд других доступных в данный момент команд. Обратите внимание, что в момент подготовки магической матрицы **M** ее имя появилось в окне браузера рабочей области – в правой части экрана. При этом матрица представляется изображением таблицы.

Часто, выполнив какую-то операцию, мы замечаем, что она оказалась ошибочной. При работе в MATLAB такой ситуации пугаться не стоит – нажатие кнопки **Undo (Отменить)** панели инструментов приведет к отмене последнего действия, выполненного в текущей строке. Операции в предыдущих строках документа этой командой не отменяются. Если оказалось, что вы зря произвели отмену последней операции, то ее легко восстановить, введя с панели инструментов операцию **Redo (Восстановить)**.

Кнопка **GUIDE** появилась в реализации MATLAB R2006b. Она открывает окно создания и изменения объектов интерфейса пользователя, показанное на рис. 2.1.

В дальнейшем мы будем пользоваться только стандартными средствами интерфейса. Читатель, желающий изменить этот интерфейс, легко освоит возможности окна рис. 2.1 (см. урок 12) и может поэкспериментировать с ним.

2.1.2. Браузер рабочей области

Нетрудно догадаться, что имена (идентификаторы) переменных различного типа и их значения хранятся в памяти компьютера. Эту область памяти именуют *рабочей областью*. В левой части окна системы MATLAB имеется окно специального браузера рабочей области – **Workspace Browser**. Он служит для просмотра ресурсов рабочей области памяти. Браузер дает наглядную визуализацию содержимого рабочей области – рис. 2.2. В частности, в нем имеются данные обо всех заданных переменных, векторах, матрицах и массивах.

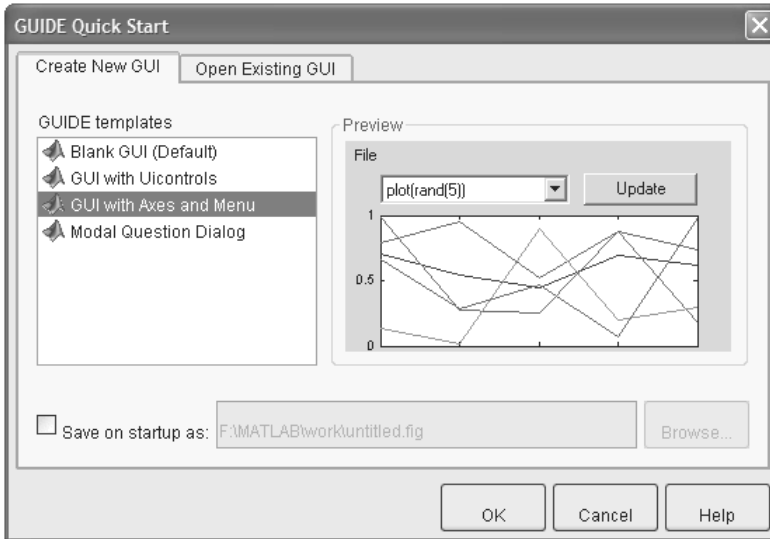


Рис. 2.1. Окно создания и изменения объектов системы MATLAB R2006b

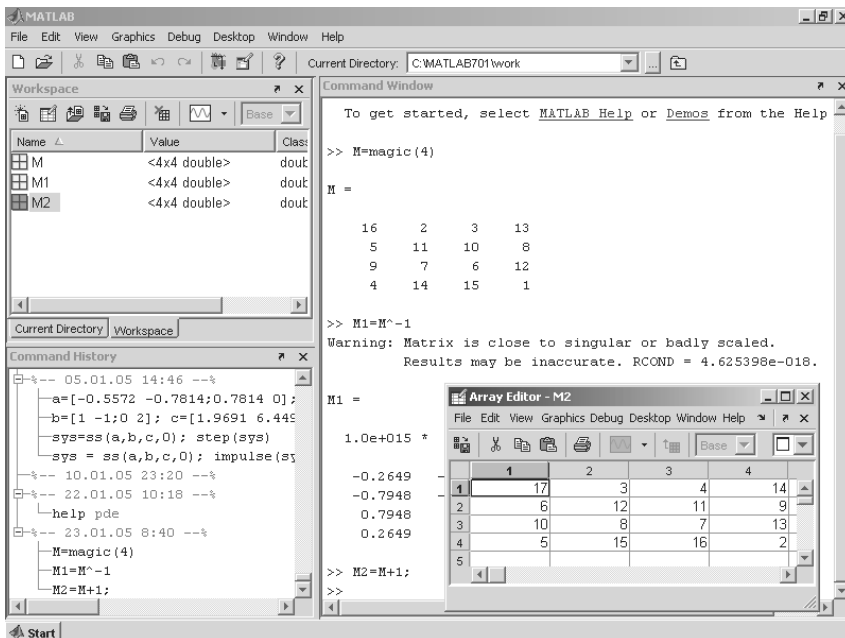


Рис. 2.2. Пример просмотра рабочей области

Окно браузера рабочей области выполняет и другие важные функции – позволяет просматривать существующие в памяти объекты, редактировать их содержимое и удалять объекты из памяти. При работе с браузером рабочего пространства в меню появляются две новые позиции – View (обзор массивов) и Graphics (специальные возможности графики).

Для вывода содержимого объекта достаточно выделить его имя с помощью мыши и щелкнуть на кнопке **Open (Открыть)**. Объект можно открыть и двойным щелчком на его имени в списке. Откроется окно редактирования массива **Array Editor**, показанное на рис. 2.2 применительно к матрице **M2**.

Окно редактирования матрицы дает удобный доступ для редактирования любого элемента матрицы по правилам, принятым при работе с электронными таблицами. Основное из них – быстрый доступ к любому элементу матрицы. Можно менять тип значений элементов, выбирая его из списка, предоставляемого меню **Numeric format (Формат чисел)**. В окне также выводятся данные о числе строк и столбцов матрицы.

Заметим, что в новых версиях MATLAB работа с браузером рабочей области стала еще более удобной, чем в прежних версиях этой системы. Это достигнуто за счет того, что главные возможности работы с браузером перенесены в позицию меню **View**.

2.1.3. Команды просмотра рабочей области *who* и *whos*

Следует отметить, что просмотр рабочей области возможен и в командном режиме, без обращения к браузеру **Workspace Browser**. Команда *who* выводит список определенных переменных, а команда *whos* – список переменных с указанием их размера и объема занимаемой памяти.

Пример: создать три переменные и просмотреть их в рабочем пространстве.

```
>> x=1.234;
>> V=[1 2 3 4 5];
>> M=magic(4);
>> who
Your variables are:
M      V      x
>> whos
      Name   Size   Bytes  Class
      M      4x4    128    double array
      V      1x5    40     double array
      X      1x1    8      double array
Grand total is 22 elements using 176 bytes
```

Если вы хотите просмотреть данные одной переменной, например *M*, следует использовать команду *whos M*. Естественно, просмотр рабочей области с помощью браузера рабочей области (**Workspace Browser**) более удобен и нагляден.

2.1.4. Браузер файловой структуры

Для просмотра файловой структуры MATLAB служит специальный браузер файловой системы (**Path Browser**), который запускается при обычной загрузке системы. Если был установлен упрощенный интерфейс, то для запуска браузера файловой системы используется окно **Current Directory (Текущая папка)**. На рис. 2.3 показано окно этого браузера, выведенное отдельно.

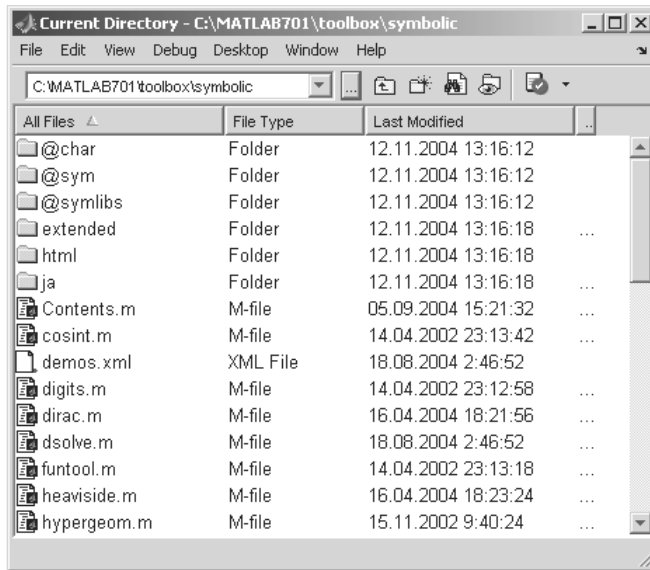


Рис. 2.3. Окно браузера *Path Browser*

Исполнив команду **Open (Открыть)** из контекстного меню правой клавиши мыши или дважды щелкнув по выделенной строке с именем файла, можно ввести этот файл в окно редактора/отладчика m-файлов. При этом редактор запустится автоматически и его окно с готовым для редактирования выбранным файлом появится на экране.

2.2. Работа с меню

2.2.1. Команды, операции и параметры

Каждая открытая позиция строки основного меню содержит различные операции и команды. Выделенная команда или операция выполняется при нажатии клавиши **Ввод (Enter)**. Выполнение команды можно также осуществить щелчком мыши или нажатием на клавиатуре клавиши, соответствующей выделенному

символу в названии команды. Для ряда команд указаны «горячие» клавиши или комбинации клавиш, обеспечивающие быстрое выполнение той или иной команды с вводом с клавиатуры.

Между командами и операциями нет особых отличий, и в литературе по информатике их часто смешивают. Для определенности мы будем считать *командой* действие, которое исполняется немедленно. А *операцией* – действие, которое требует определенной подготовки, например открытие окна для установки определенных параметров.

Параметр (option) – это значение определенной величины, действующее во время текущей сессии. Параметрами обычно являются указания на применяемые наборы шрифтов, размеры окна, цвет фона и т. д.

2.2.2. Меню системы MATLAB

Перейдем к описанию основного меню системы MATLAB. Начнем с меню версий MATLAB 6.*, все еще часто применяемых. Их меню содержит всего шесть пунктов:

- **File** – работа с файлами;
- **Edit** – редактирование сессии;
- **View** – вывод и скрытие панели инструментов;
- **Web** – доступ к интернет-ресурсам;
- **Window** – переключение и закрытие окон;
- **Help** – доступ к справочным подсистемам.

Позиция **Web** дублирует возможности стандартных браузеров, например Microsoft Internet Explorer, и не всегда работоспособна. Возможно, поэтому она удалена в новых реализациях MATLAB. Работа с позицией **Window** вполне очевидна, а позиция **Help** открывает доступ к справке. Отметим особенности работы трех первых позиций меню системы MATLAB любой версии.

2.2.3. Меню File

Меню **File** содержит ряд операций и команд для работы с файлами:

- **New** – открытие подменю с позициями;
- **M-file** – открытие окна редактора/отладчика m-файлов;
- **Figure** – открытие пустого окна графики;
- **Model** – открытие пустого окна для создания Simulink-модели;
- **GUI** – открытие окна разработки элементов графического интерфейса пользователя;
- **Deployment Project** – открытие окна разработки;
- **Open** – открытие окна загрузки файла;
- **Close Command Windows** – закрытие окна командного режима работы (оно при этом исчезает с экрана);
- **Import data** – открытие окна импорта файлов данных;
- **Save Workspace As...** – открытие окна записи рабочей области в виде файла с заданным именем;

- **Set Path** – открытие окна установки путей доступа файловой системы;
- **Preferences...** – открытие окна настройки элементов интерфейса;
- **Print...** – открытие окна печати всего текущего документа;
- **Print Selection...** – открытие окна печати выделенной части документа;
- **Exit** – завершение работы с системой.

Большинство окон, открывающихся этими операциями, хорошо известны пользователям любыми приложениями Windows. Поэтому остановимся на описании только тех окон, которые специфичны для систем класса MATLAB. Кстати, состав команд позиции **File** во всех описанных в книге версиях системы MATLAB идентичен.

2.2.4. Установка путей доступа файловой системы

Поскольку MATLAB работает со множеством файлов, расположенных в разных папках (директориях), то не исключены случаи неправильной работы из-за указания неверного пути доступа к файлам. Для коррекции этого пути в ходе работы с MATLAB служит *редактор доступа файловой системы*. Его окно открывается операцией **Set Path...** (**Установить путь**) – рис. 2.4.

Окно дает список папок с файлами системы MATLAB. Имеется возможность переноса папок вверх или вниз по списку, уничтожения их и переименования. По умолчанию задается правильная установка путей доступа, так что данными воз-

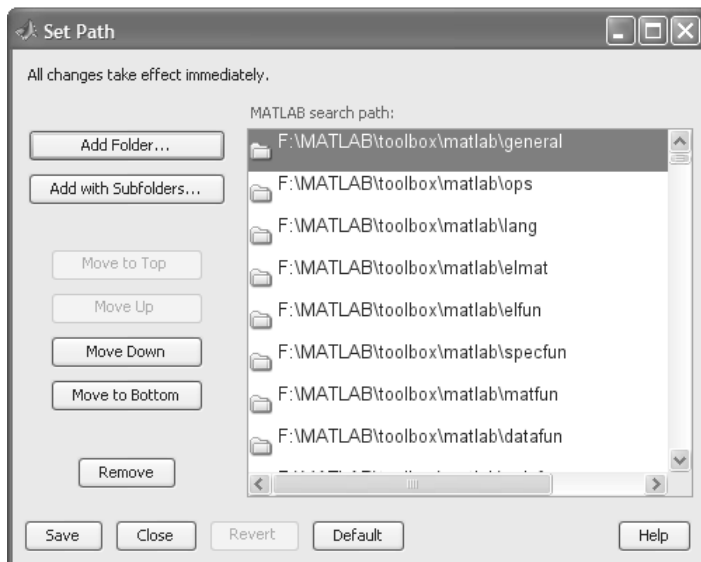


Рис. 2.4. Окно редактора путей доступа файловой системы

возможностями стоит пользоваться только в особых обстоятельствах, например при случайном переносе папок в другое место или при их переименовании.

2.2.5. Настройка элементов интерфейса

Команда **Preferences...** (**Предпочтения**) в версиях MATLAB 6.* выводит окно детальной настройки элементов интерфейса (рис. 2.5).

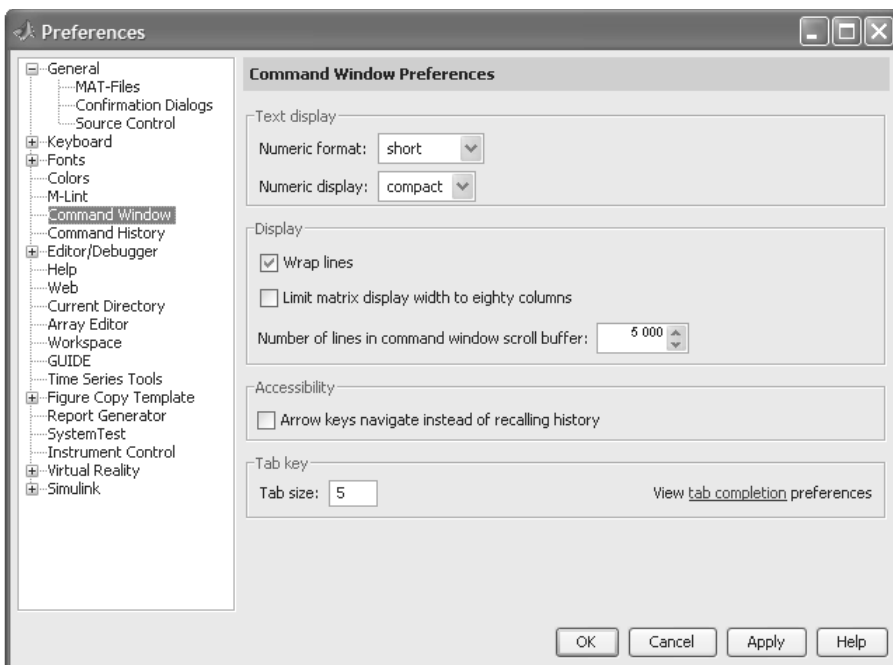


Рис. 2.5. Окно настройки элементов интерфейса

В левой части этого окна имеется древообразный список элементов интерфейса системы, а в правой части – поле задания параметров для выбранного типа элементов. Число параметров и видов этого окна велико, но заинтересованный читатель разберется с нужными ему параметрами без особого труда.

2.2.6. Меню **Edit** – средства редактирования документов

Меню **Edit** содержит операции редактирования, типичные для большинства приложений Windows. Это меню имеет следующие операции и команды:

- **Undo (Отменить)** – отмена результата предшествующей операции;

- **Redo (Повторить)** – отмена действия последней операции Undo;
- **Cut (Вырезать)** – вырезание выделенного фрагмента и перенос его в буфер;
- **Copy (Копировать)** – копирование выделенного фрагмента в буфер;
- **Paste (Вставить)** – вставка фрагмента из буфера в текущую позицию курсора;
- **Paste to Workspace...** – открытие окна вставки в рабочее пространство (новая команда);
- **Clear (Очистить)** – операция очистки выделенной области;
- **Select All (Выделить)** – выделение всей сессии;
- **Delete (Стереть)** – уничтожение выделенного объекта;
- **Find...** – открытие окна поиска объекта (текста) в командном окне (новая команда);
- **Find files...** – открытие окна поиска заданного файла (новая команда);
- **Clear Command Windows (Очистить командное окно)** – очистка текста сессии (с сохранением созданных объектов);
- **Clear Command History (Очистить окно истории команд)** – очистка окна истории;
- **Clear Workspace** – очистка окна браузера рабочей области.

Назначение ряда указанных команд и операций уже обсуждалось. Отметим лишь, что команда **Clear Command Window** очищает окно командного режима работы и помещает курсор в верхний левый угол окна. Однако все определения, сделанные в течение стертых таким образом сессий, сохраняются в памяти компьютера. Напомним, что для очистки экрана используется также команда `clc`, вводимая в командном режиме. Три новые команды (они отмечены выше) вполне очевидны.

2.2.7. Интерфейс по умолчанию

Новые версии MATLAB имеют довольно много элементов интерфейса. Пользователь может легко менять вид интерфейса, закрывая или открывая те или иные окна. Основные средства для этого в новых версиях сосредоточены в позиции меню **Desktop**.

Иногда полезно вернуться к виду интерфейса по умолчанию. Для этого можно воспользоваться командой **Desktop Layout** ⇒ **Default** в позиции **Desktop** меню. Вид окна системы MATLAB R2006b по умолчанию показан на рис. 2.6.

2.3. Основы редактирования и отладки m-файлов

2.3.1. Интерфейс редактора/отладчика m-файлов

Программы в системе MATLAB представлены m-файлами. Для подготовки, редактирования и отладки m-файлов (а в MATLAB R2006a,b/R2007a – и файлов различных языков программирования) служит специальный многооконный ре-

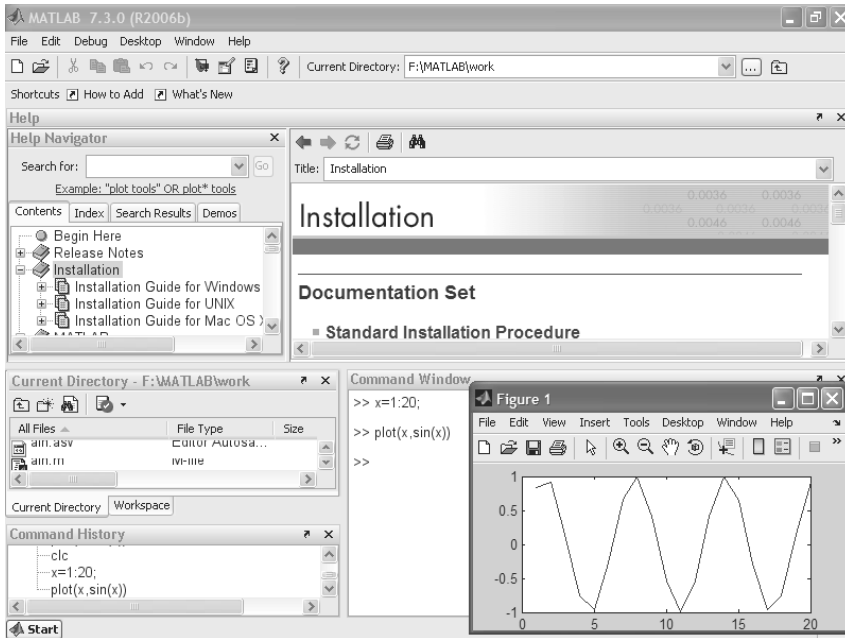


Рис. 2.6. Интерфейс MATLAB R2006b по умолчанию

доктор. Он выполнен как типичное приложение Windows. Редактор можно вызвать командой `edit` из командной строки или командой **New** \Rightarrow **M-file** из меню **File**. После этого в окне редактора можно создавать свой файл, пользоваться средствами его отладки и запуска. Перед запуском файла его необходимо записать на диск, используя команду **File** \Rightarrow **Save as** в меню редактора.

На рис. 1.6 было показано окно редактора/отладчика MATLAB R2006a с текстом простого файла `demo.m` в окне редактирования и отладки. Подготовленный текст файла (это простейшая и наша первая программа на языке программирования MATLAB) можно записать на диск. Для этого используется команда **Save As**, которая применяет стандартное окно Windows для записи файла с заданным именем.

После записи файла на диск можно заметить, что команда **Run** в меню **Tools (Инструменты)** редактора становится активной (до записи файла на диск она пассивна) и позволяет произвести запуск файла. Запустив команду **Run**, можно наблюдать исполнение m-файла – в нашем случае это вычисление выражения $2 + 3$ и построение рисунка с графиком синусоидальной функции в графическом окне (рис. 1.6 снизу).

Для удобства работы с редактором/отладчиком строки программы в нем нумеруются в последовательном порядке. Редактор является многооконным. Окно каждой программы оформляется как вкладка. Редактор-отладчик позволяет легко просматривать значения переменных. Для этого достаточно подвести к имени

переменной курсор мыши и задержать его – появится всплывающая подсказка с именем переменной и ее значением.

2.3.2. Цветовые выделения и синтаксический контроль

Редактор/отладчик m-файлов выполняет синтаксический контроль программного кода по мере ввода текста. При этом используются следующие цветовые выделения:

- ключевые слова языка программирования – синий цвет;
- операторы, константы и переменные – черный цвет;
- комментарии после знака % – зеленый цвет;
- символьные переменные (в апострофах) – зеленый цвет;
- синтаксические ошибки – красный цвет.

Благодаря цветовым выделениям вероятность синтаксических ошибок резко снижается.

Однако далеко не все ошибки диагностируются. Ошибки, связанные с неверным применением операторов или функций (например, применение оператора – вместо + или функции $\cos(x)$ вместо $\sin(x)$ и т. д.), не способна обнаружить ни одна система программирования. Устранение такого рода ошибок (их называют семантическими) – дело пользователя, отлаживающего свои алгоритмы и программы.

2.3.3. Понятие о файлах-сценариях и файлах-функциях

Здесь полезно отметить, что m-файлы, создаваемые редактором/отладчиком, делятся на два класса:

- *файлы-сценарии*, не имеющие входных параметров;
- *файлы-функции*, имеющие входные параметры полноценные процедуры.

Видимый в окне редактора на рис. 1.6 файл является файлом-сценарием, или Script-файлом. Данный файл не имеет списка входных параметров и является примером простой процедуры без параметров. Он использует *глобальные переменные*, то есть такие переменные, значения которых могут быть изменены в любой момент сеанса работы и в любом месте программы. Для запуска файла-сценария из командной строки MATLAB достаточно указать его имя в этой строке.

Файл-функция отличается от файла-сценария прежде всего тем, что созданная им функция имеет *входные параметры*, список которых указывается в круглых скобках. Используемые в файле-функции переменные являются *локальными переменными*, изменение значений которых в теле функции никоим образом не

влияет на значения, которые те же самые переменные могут иметь за пределами функции.

2.3.4. Панель инструментов редактора и отладчика

Редактор имеет свое меню и свою панель инструментов. Она представлена на рис. 2.7.

Назначение кнопок панели инструментов редактора/отладчика (слева направо) следующее:

- **New** – создание нового m-файла;
- **Open** – вывод окна загрузки файла;
- **Save** – запись файла на диск;
- **Print** – печать содержимого текущего окна редактора;
- **Cut** – вырезание выделенного фрагмента и перенос его в буфер;
- **Copy** – копирование выделенного объекта в буфер;
- **Paste** – размещение фрагмента из буфера в позиции текстового курсора;
- **Undo** – отмена предшествующей операции;
- **Redo** – повтор отмененной операции;
- **Find text** – нахождение указанного текста;
- **Show function** – показ функции;
- **Set/Clear Breakpoint** – установка/сброс точки прерывания;
- **Clear All Breakpoints** – сброс всех точек прерывания;
- **Step** – выполнение шага трассировки;
- **Step In** – пошаговая трассировка с заходом в вызываемые m-файлы;
- **Step Out** – пошаговая трассировка без захода в вызываемые m-файлы;
- **Save and Run** – запись и сохранение;
- **Exit Debug Mode** – выход из режима отладки.

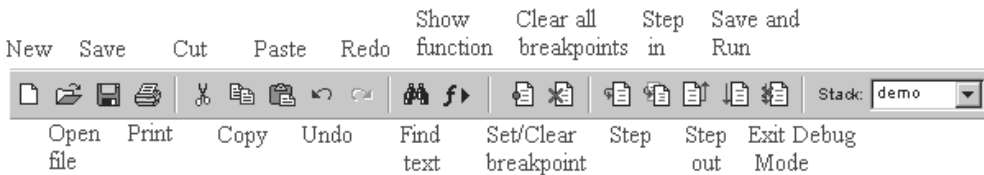


Рис. 2.7. Панель инструментов редактора/отладчика

Подробное описание работы с редактором/отладчиком дано в уроке, посвященном средствам программирования.

2.4. Новинки графического интерфейса MATLAB

2.4.1. Новая позиция меню *Graphics*

Из самых серьезных нововведений в версиях системы MATLAB 7.* надо отметить новые возможности редактирования и создания графики. Так, новая позиция **Graphics** меню содержит всего три основные команды:

- **New Figure** – открывает пустое окно графики;
 - **Plot Tools** – открывает окно редактора графики;
 - **More Plots...** – открывает окно доступа к различным видам графики.
- Первая команда очевидна, а две другие будут детально описаны ниже.

2.4.2. Работа с окном 2D-графики MATLAB

Рассмотрим работу с окном графики вначале на примере 2D-графики. Рисунок 2.8 дает пример построения графиков двух простых функций. Соответствующие команды заданы в командной строке MATLAB. На рис. 2.8 показано также окно графики с построенными кривыми. Пока ничем выдающимся это построение не отличается.

Обратите внимание на то, что построение графиков сопровождается появлением в рабочем пространстве трех массивов – переменных x , y и z . Они видны в окне вьювера рабочего пространства. Полезно подметить и то, что в окне графика на панели инструментов появились новые кнопки:

- **Data cursor** – вывести данные по месту установки курсора;
- **Insert Color Bar** – вставить в рисунок панель цветов;
- **Insert Legends** – вставить легенду;
- **Hide Plot Tools** – скрыть окно редактора графики;
- **Show Plot Tools** – показать окно редактора графики.

Все эти команды очевидны. Стоит только отметить первую команду: если навести курсор на место кривой графика и щелкнуть левой клавишей мыши, то появится всплывающее окошко с координатами точки. При этом действует система автоматического слежения курсором за кривой. Многие команды по обработке графиков в графическом окне соответствуют уже описанным командам, их можно вводить как из меню окна графики, так и с помощью контекстного меню – на рис. 2.8 представлена позиция меню **Insert** в открытом состоянии.

Окно графиков имеет обычное меню. Освоив меню системы MATLAB, нетрудно разобраться и с позициями меню графического окна. Позже, при описании форматирования графиков, мы рассмотрим основные из команд, доступ к которым дает меню графического окна.

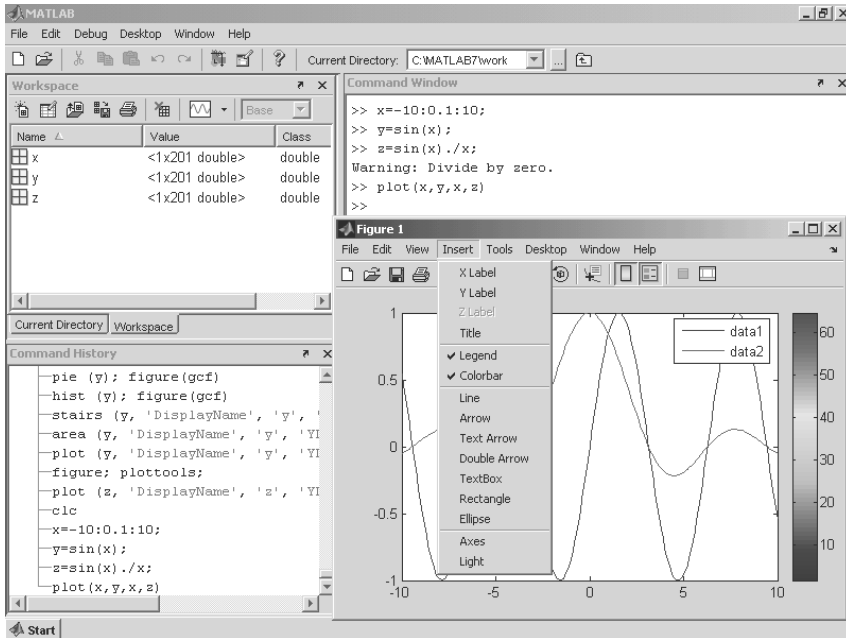


Рис. 2.8. Построение графиков двух функций

2.4.3. Работа с редактором графики MATLAB

Редактор графики – это принципиально новое графическое средство системы MATLAB 7.* (R2006a,b). Он служит как для редактирования уже созданных M-файлами или командами в командной строке графиков, так и для создания графиков заданного пользователем типа. Редактор графики можно ввести из окна графики, из меню этого окна и из меню **Graphics** окна системы MATLAB.

Если введены массивы, нужные для построения графики, и открыт браузер рабочего пространства, то позиция Graphics меню расширяется набором команд для построения возможных типов графиков. Это хорошо видно по рис. 2.9, на котором показана открытая позиция Graphics. В ней видны команды построения обычного графика, столбикового графика, круговой диаграммы и т. д.

Исполним, например, команду `plot(y)` и увидим появившееся окно редактора графики – рис. 2.10. В левой его части имеется панель **Figure Palette**, которая имеет три области:

- **New Subplot** – открытие новых 2D- и 3D-графиков;
- **Variables** – выбор из списка переменной-массива, по данным которого строится очередной график;

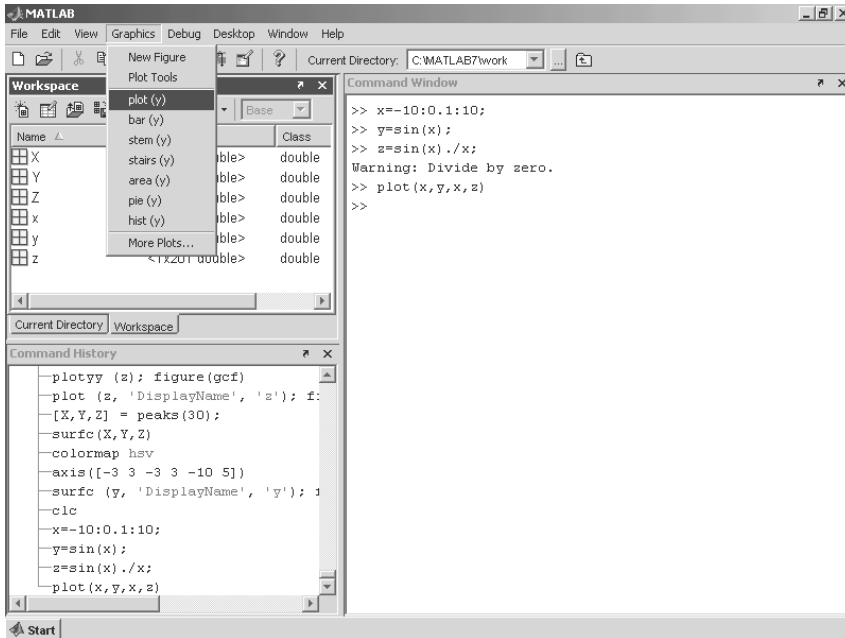


Рис. 2.9. Окно системы MATLAB с открытой позицией Graphics меню

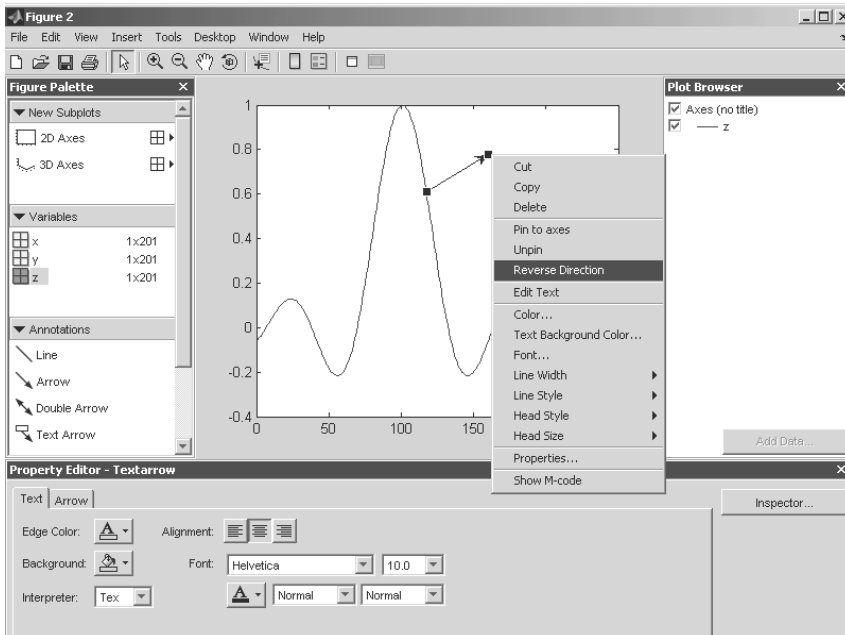


Рис. 2.10. Пример работы с редактором графики для случая 2D-графики

- **Annotations** – выбор средств украшения графиков (стрелки, текстовые надписи, окружности и эллипсы, прямоугольники и т. д.).

Выбрав, к примеру, переменную z и щелкнув по ее имени мышью, можно наблюдать построение графика соответствующей функции – в данном случае $\sin(x)/x$. Воспользовавшись средствами Annotations, можно разнообразить график, например нанести на него стрелку для текстовой надписи, что и показано на рис. 2.10. Показано также контекстное меню правой клавиши мыши, которое дает доступ к операциям, возможным для данного графика. В частности, есть команда *Reverse Direction* для изменения направления стрелки. Вид окна редактора после изменения направления стрелки и введения текстовой надписи показан на рис. 2.11.

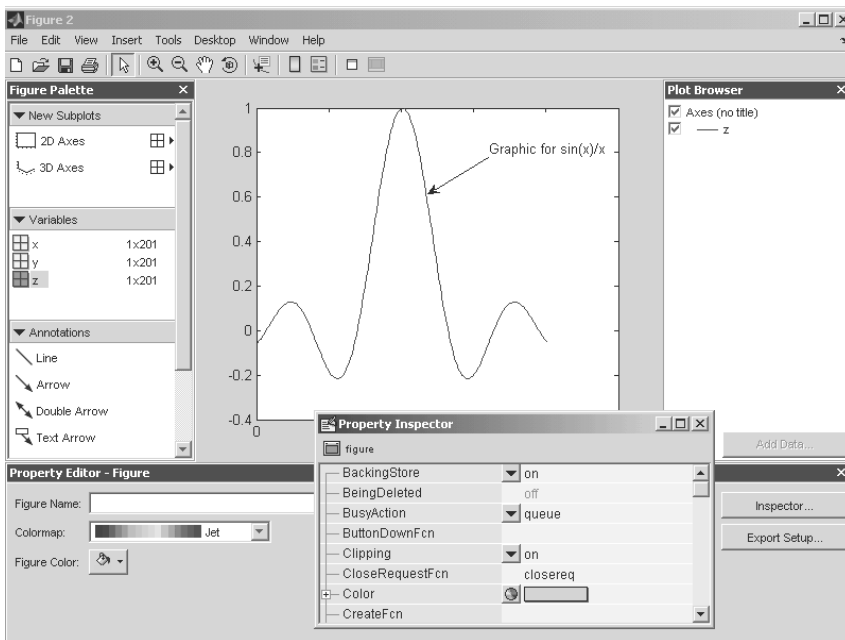


Рис. 2.11. Окно редактора графики после работы над 2D-графиком

Рисунок 2.12 демонстрирует возможность построения двух графиков в разных окнах редактора графики. Как происходит разбивка окна, мы рассмотрим чуть позже. Обратите внимание на еще одно средство контроля – панель свойств редактора, расположенную снизу. Она контекстно-зависимая, и на рис. 2.12 ей соответствует установка осей Axes (на рис. 2.11 эта панель относится к фигуре – Figure). Здесь можно задать титульную надпись, ввести линии разметки окна, построить прямоугольник вокруг графика, проверить и изменить пределы изменения переменных, назначить линейный или логарифмический типы графика, изменить шрифт надписей, вызвать инспектор графика и т. д.

Для углубленного изучения графика и его основных свойств служит инспектор графики. Он вызывается из окна редактора графики активизацией кнопки **Inspector**. Окно инспектора графики показано на рис. 2.12 снизу. С помощью этого окна можно установить любые свойства объектов графика, например цвет линий, их тип и т. д. и т. п. О многих из них обычный пользователь может и не догадываться, но профессионал оценит такую возможность весьма положительно.

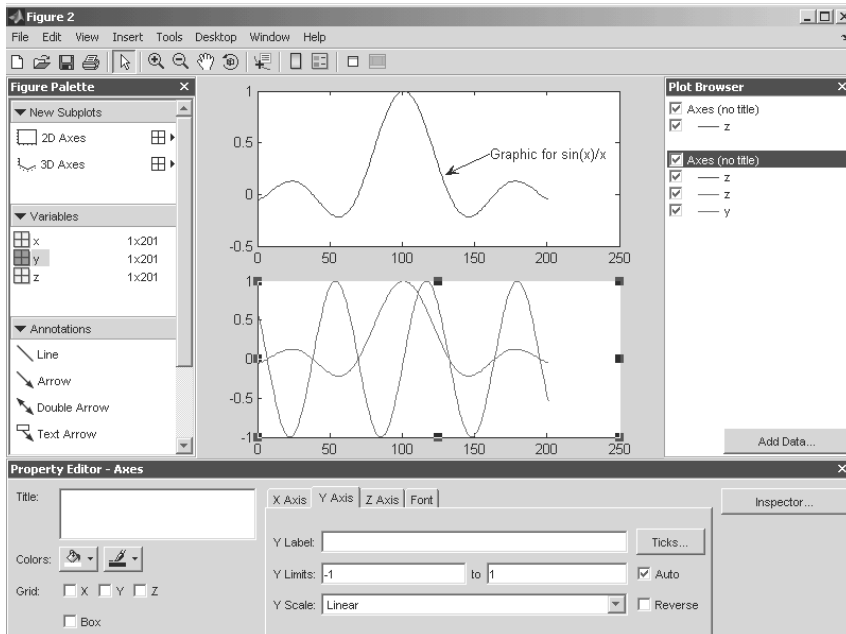


Рис. 2.12. Окно редактора графики с двумя графиками

2.4.4. Построение графиков из их каталога

Если есть данные для построения графика (например, в виде переменной-массива), то для построения графика достаточно задать его тип. Для этого в позиции меню **Graphics** следует исполнить команду **More Plots...**. Появится окно с каталогом графиков **Plot Catalog** – рис. 2.13. В нем надо указать имя переменной, хранящей данные для построения графика, например z для наших экспериментов.

По вертикали окно каталога графиков делится на три области:

- **Categories** – категория графиков;
- **Plot Types** – тип графиков в пределах каждой категории;
- **Description** – описание выделенного (выбранного) графика.

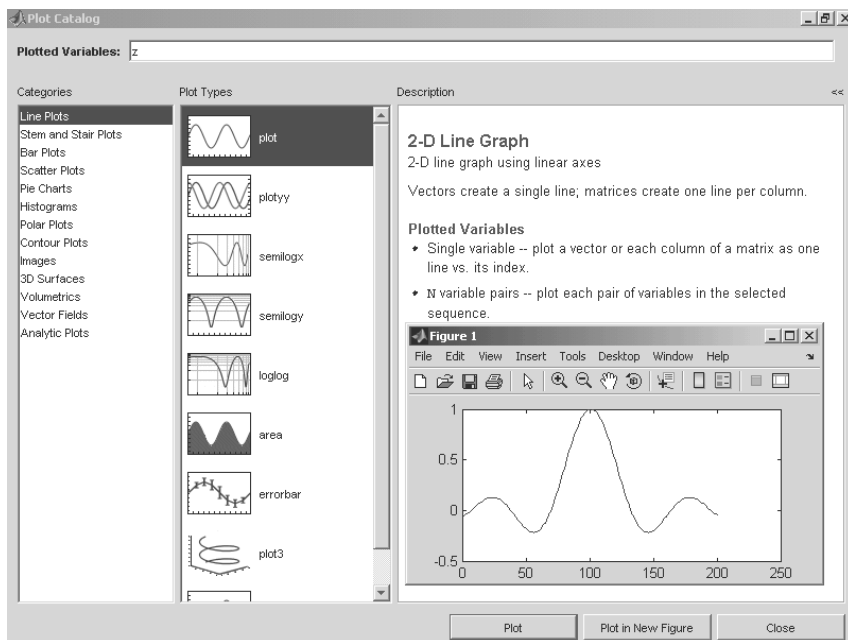


Рис. 2.13. Окно каталога графиков

В каталоге представлены все типы графиков, которые способна строить система MATLAB 7, включая особую handle-графику. Каталог дает наглядный вид графиков, что облегчает их выбор и в дальнейшем построение.

В простых случаях, если пользователь уверен в возможности построения графика по выбранным данным, достаточно нажать кнопку **Plot** внизу области Description. Будет построен график в стандартном окне графики системы MATLAB – см. это окно в области Description на рис. 2.13. Кнопка **Plot in New Figure** позволяет приступить к построению новой фигуры.

Теперь попытаемся построить график трехмерной поверхности с линиями равного уровня на плоскости под графиком. Окно каталога в этом случае показано на рис. 2.14 с установкой на категорию 3D Surface. Выберем подходящий тип графика – сверху второй. В разделе описания Description появится статья с описанием назначения и построения выбранного графика.

Поскольку в данном случае мы не подготовили данных для построения графика, то использовать кнопку **Plot** нельзя – будет получено сообщение об ошибке. Однако в разделе описания в конце его есть гиперссылка `surf`, которая открывает раздел справки по нужной функции, – рис. 2.15.

В конце справки по данной функции можно найти пример ее применения. Его надо выделить, что показано на рис. 2.16, и перенести в окно MATLAB для исполнения. Можно и сразу проделать это с помощью контекстного меню правой клавиши мыши.

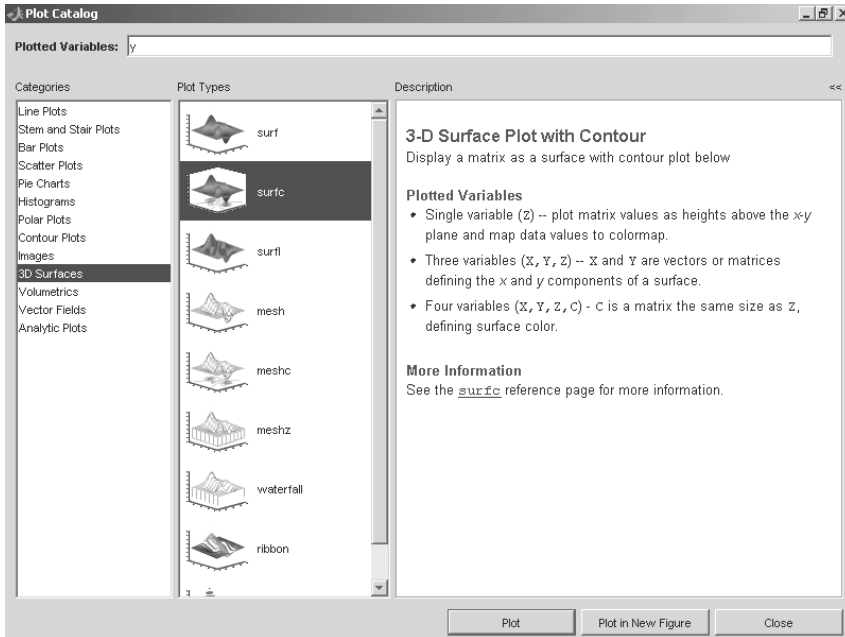


Рис. 2.14. Окно каталога графиков с категорией 3D Surface

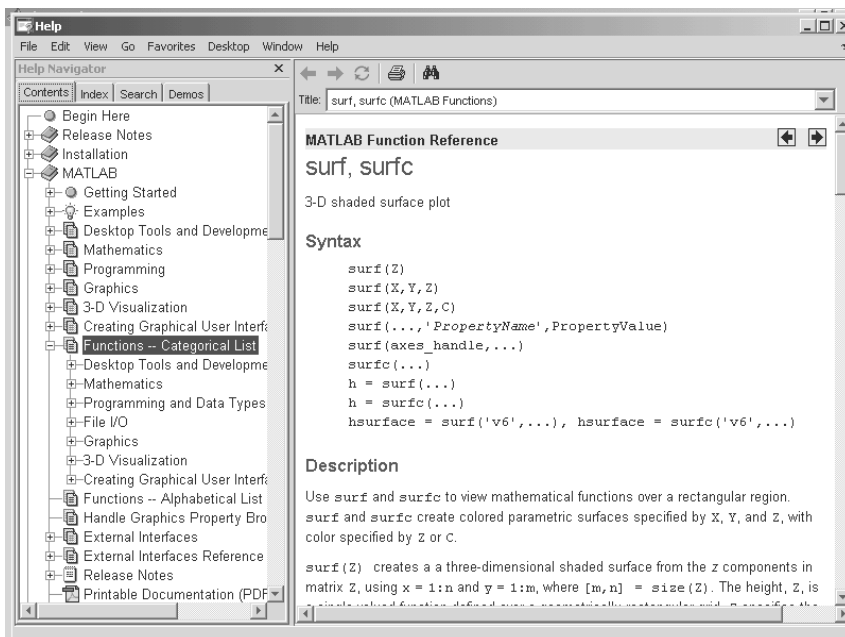


Рис. 2.15. Окно справки по функции surfz

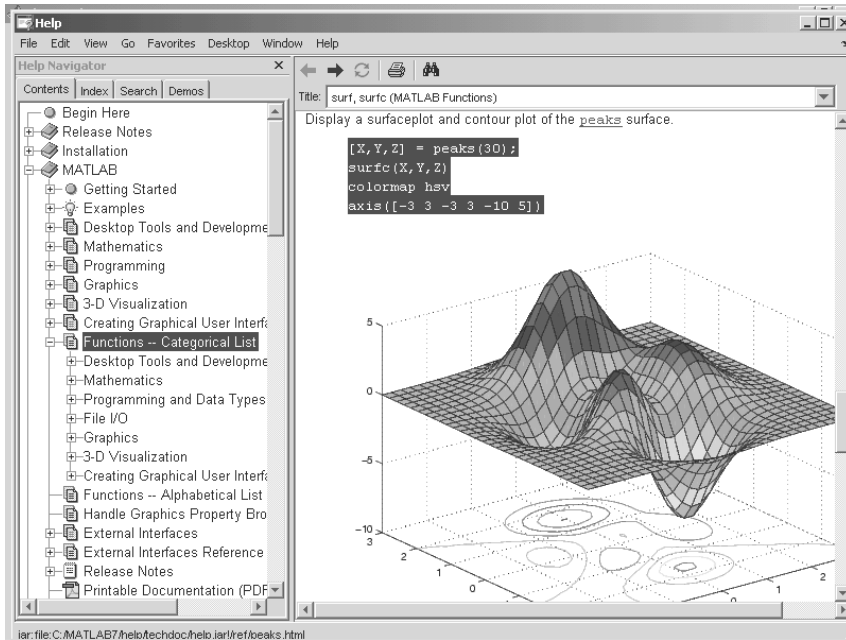


Рис. 2.16. Окно справки по функции `surf` с примером ее применения

Исполнив пример, можно наблюдать построение графика в окне трехмерной графики. Этот случай показан на рис. 2.17. Теперь все данные графика размещаются в рабочем пространстве и можно воспользоваться редактором графики для дальнейшей обработки графика.

2.4.5. Некоторые другие особенности применения редактора графики

Отметим еще несколько особенностей применения редактора графики. Мы уже говорили о возможности построения множества графиков в одном окне редактора. Для этого достаточно активизировать темный треугольник у квадратика с разбиением на части в области редактора Figure Palette. Появится шаблон для задания макета окна – на рис. 2.18 показано выделение одного окна, но можно выделить и большее число окон.

Если данных для построения графика нет, то окно графики в окне редактора графики будет пустым – рис. 2.19. Если шаблон графики выбран с несколькими графиками, то появятся несколько окон графиков.

Чтобы построить данные, в пустом окне редактора графики надо добавить данные. Для этого достаточно исполнить команду **Add Data...**, доступную, например, из контекстного меню правой клавиши мыши (рис. 2.20) в случае, когда курсор

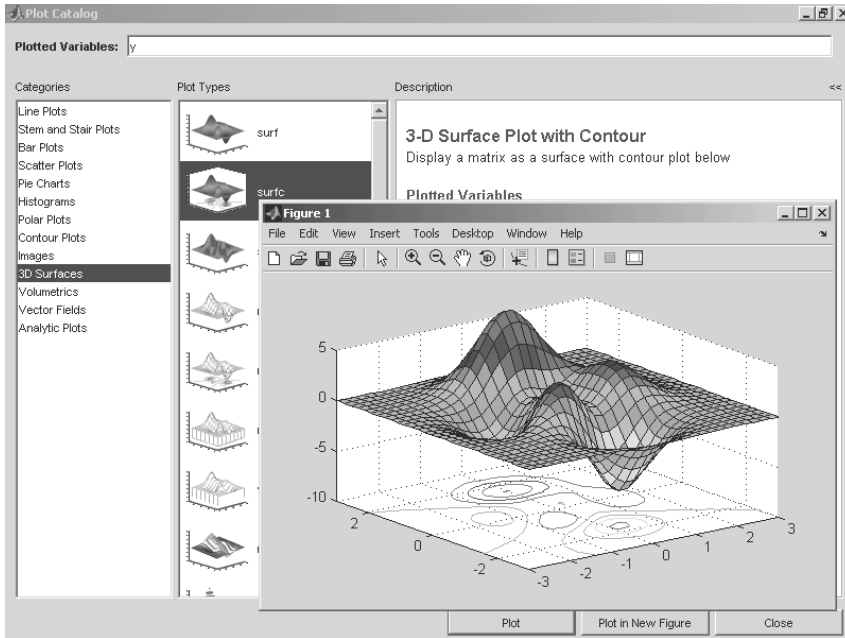


Рис. 2.17. Пример построения графика поверхности и контурного графика

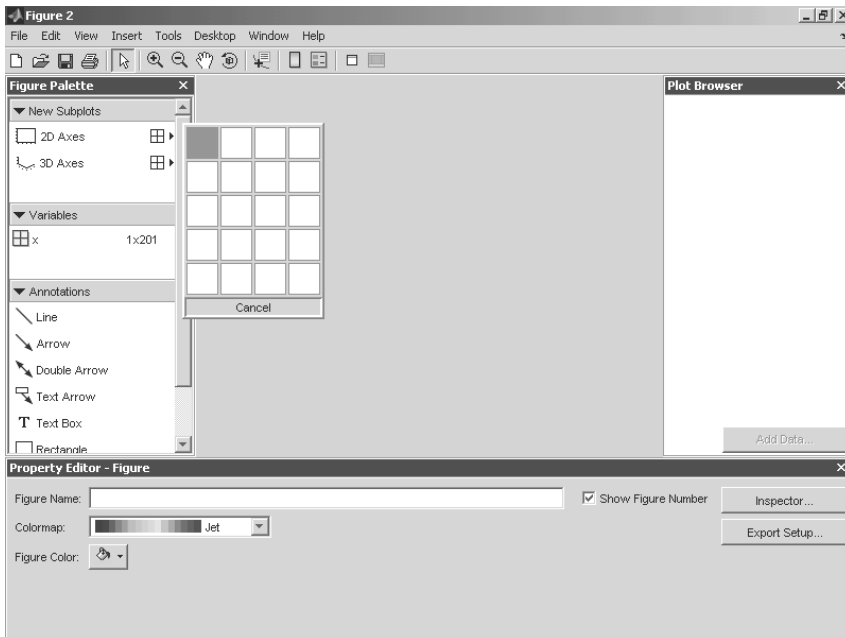


Рис. 2.18. Пример задания шаблона окна графики внутри редактора графики

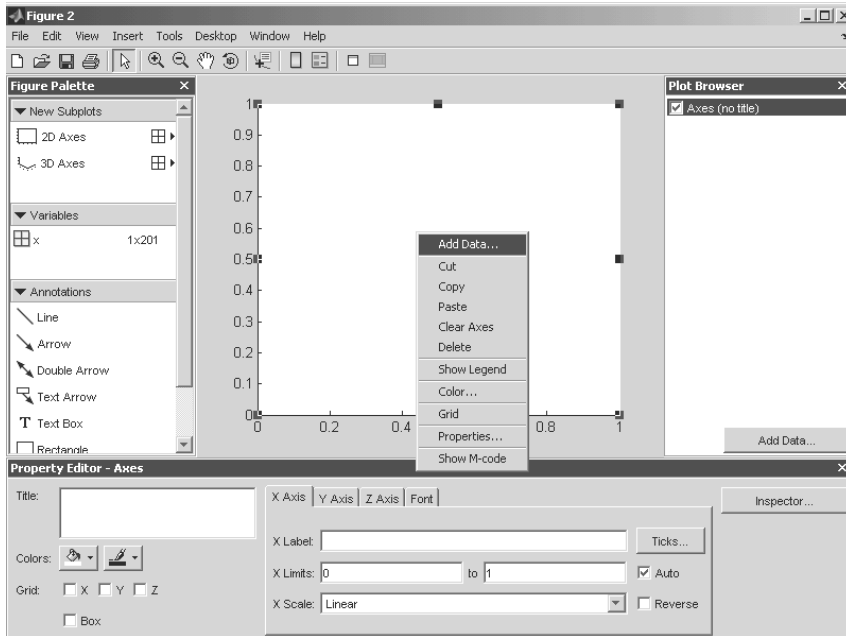


Рис. 2.19. Пример вывода пустого окна графики

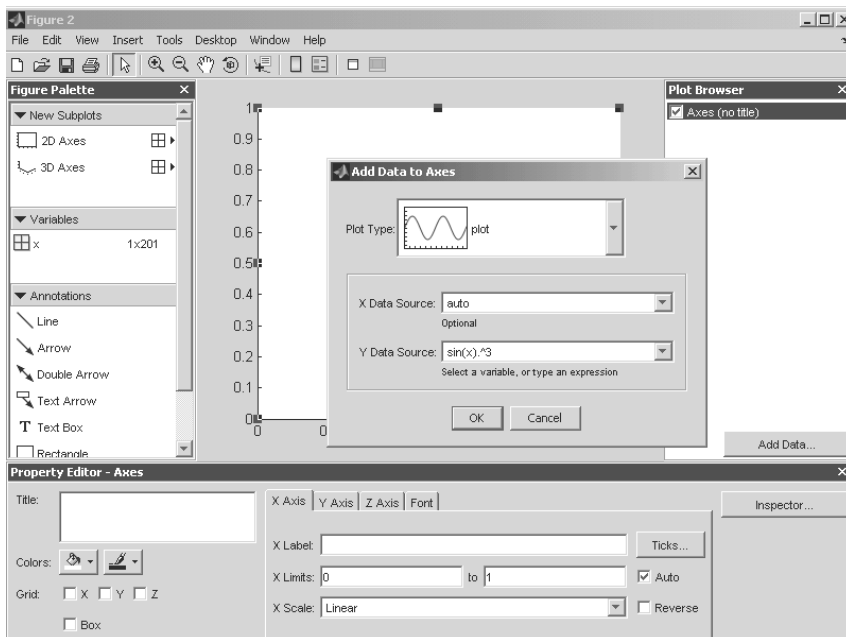


Рис. 2.20. Подготовка к добавлению данных

мыши размещен внутри пустого окна. Эту команду можно исполнить и из панели браузера графики (в правой части его окна). Появится окно **Add Data to Axes**.

В этом окне имеется список типов графиков, а также данные для установки данных. Например, если мы хотим построить график функции $\sin(x)^3$, то можно задать данные по оси x как автоматически формируемые, а по оси y ввести выражение $\sin(x)^3$. Нажав кнопку **OK**, получим построенный график в окне редактора графики – см. рис. 2.21.

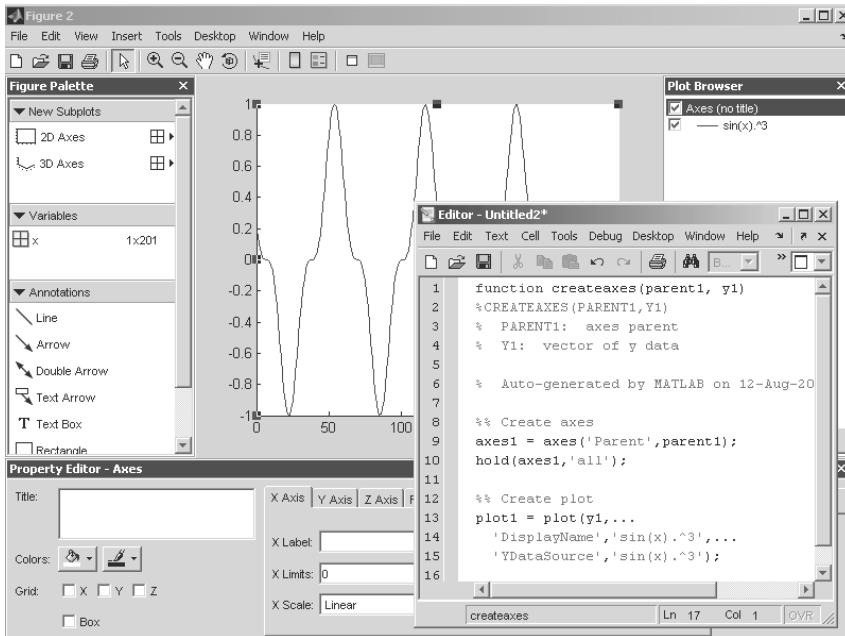


Рис. 2.21. Построенный по введенным данным график

В контекстном меню правой клавиши мыши (см. рис. 2.36) имеется команда **Show M-code** для просмотра М-файла, обеспечивающего построение заданного графика. Эта команда вызывает появление окна редактора М-кодов с текстом файла. Это окно показано на рис. 2.38 снизу.

2.4.6. Новый вид окна MATLAB

Команда **All Tabbed** в позиции **Desktop Layout** меню **Desktop** системы MATLAB 7/R2006a,b придает окну системы новый вид, показанный на рис. 2.22. Основные компоненты окна теперь представлены вкладками, переключатель которых дан в нижней части окна.

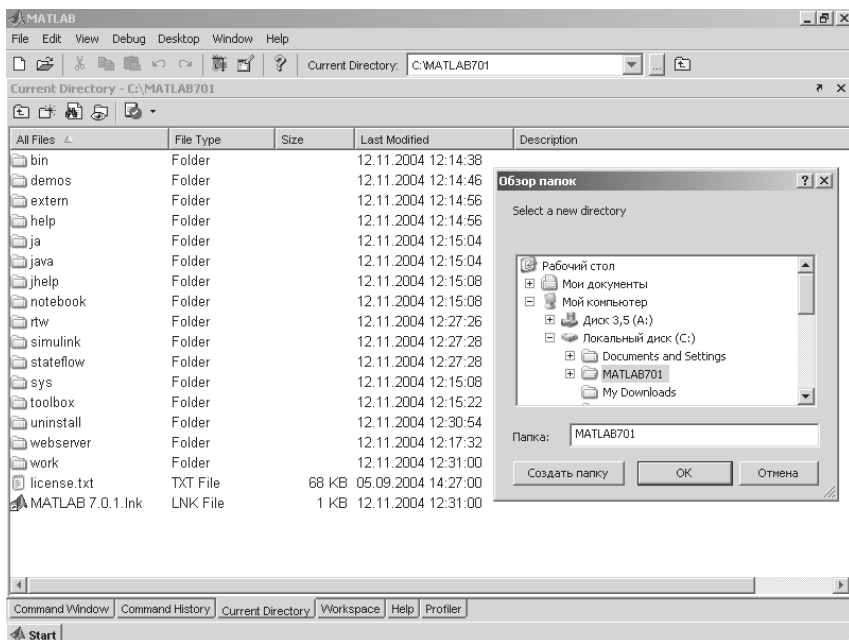


Рис. 2.22. Новое окно MATLAB

На рис. 2.22 открыта вкладка текущей директории **Current Directory**. В правой части окна показано также окно обзора папки, которое вызывается активизацией кнопки ... справа от списка директорий в строке меню. Ничего нового этот вид окна не дает, но он может понравиться многим пользователям. С видом окна при других открытых вкладках читатель может познакомиться самостоятельно.

2.5. Интерфейс графических окон

2.5.1. Обзор интерфейсов графических окон

Графическое окно MATLAB представлено на рис. 2.23. Это обычное масштабируемое и перемещаемое окно Windows-приложений. MATLAB может создавать множество таких окон.

Меню этого окна похоже на меню окна командного режима работы системы MATLAB. Однако при внимательном просмотре заметен ряд отличий.

Прежде всего в меню **Edit** окна графики наряду со стандартными операциями работы с буфером есть ряд новых команд:

- **Copy Figure (Копировать рисунок)** – копирование в буфер рисунка (графики);
- **Copy Options (Копировать параметры)** – копирование параметров графика;
- **Figure Properties (Свойства рисунка)** – вывод окна свойств графика;

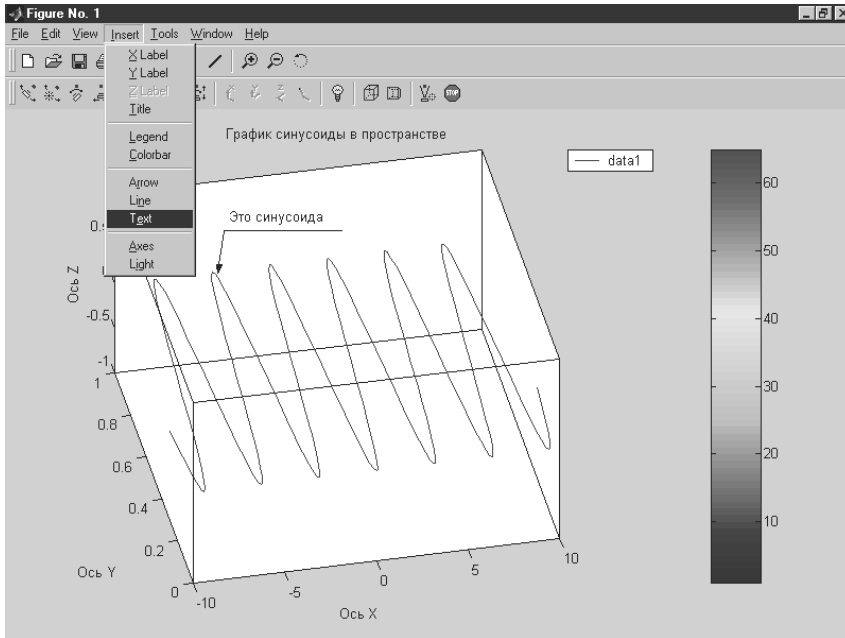


Рис. 2.23. Графическое окно MATLAB

- **Axes Properties (Свойства осей)** – вывод окна свойств осей графика;
- **Current Object Properties (Свойства текущего объекта)** – вывод окна свойств текущего объекта.

Примечание

Большинство графиков, которые описываются в книге, представлены копиями только самих графиков, а не всего графического окна. Для получения таких копий использовалась команда `Copy Figure` из меню `Edit` окна графики или просто вырезалась нужная часть копии экрана, получаемой нажатием клавиши `Print Screen`. Такое представление делает приведенные рисунки одинаковыми для всех версий MATLAB, от MATLAB 5.0 и выше.

Для вывода свойств графиков, их осей и текущих объектов используется окно свойств графиков с соответствующими вкладками, работа с которым описывалась в уроке 1.

Стоит остановиться на одной команде в позиции **File** меню графического окна. Это новая команда **Generate M-file...**, введенная в MATLAB 7. Она запускает генератор программного кода на языке MATLAB, который при его исполнении строит графическое окно с имеющимся на нем графиком. Коды выводятся в окне редактора M-файлов, могут редактироваться и записываться как MATLAB-функ-

ция. Указанная возможность – один из штрихов во внедрении в MATLAB элементов визуально-ориентированного программирования.

2.5.2. Панель инструментов камеры обзора

Отличительной особенностью окна графики еще в версии MATLAB 6.0 стало появление второй инструментальной панели со средствами форматирования трехмерной (3D) графики. Эта панель (она видна на рис. 2.23 под основной панелью инструментов) выводится командой **View** ⇒ **Camera Toolbar**.

Данная панель управляет некоторой воображаемой фотокамерой (или просто камерой), с помощью которой как бы наблюдается объект. Кнопки имеют наглядные изображения, поясняющие их действия. В связи с этим их подробное описание лишено смысла – проще опробовать их в действии.

2.5.3. Меню инструментов Tools

Действия кнопок панели инструментов камеры обзора продублированы в меню **Tools (Инструменты)**. Состав команд указанного подменю, начиная с версии MATLAB 6.0, существенно изменен и обновлен. Теперь в нем имеются следующие команды:

- **Edit Plot (Редактировать график)** – редактирование графика;
- **Zoom In (Увеличение)** – увеличение масштаба графика;
- **Zoom Out (Уменьшение)** – уменьшение масштаба графика;
- **Rotate 3D (Вращение 3D)** – вращение в пространстве;
- **Move Camera (Передвинуть камеру)** – установка камеры обзора;
- **Camera Motion (Передвижение камеры)** – установка перемещения камеры обзора;
- **Camera Axes (Оси камеры)** – установка координатных осей при работе с камерой;
- **Camera Reset (Установка начального состояния камеры)** – сброс установок камеры;
- **Basic Fitting** – проведение аппроксимации и регрессии;
- **Data Statistics** – получение статистических данных для точек графика.

Две последние позиции этого меню дают весьма оригинальные возможности обработки точек графика – выполнение регрессии множеством методов с выводом (где это возможно) уравнения регрессии на график и вычисление статистических параметров для этих точек.

2.5.4. Вращение графиков мышью

Хорошее впечатление оставляет возможность вращения графиков мышью – прием, введенный в целый ряд систем компьютерной математики (Mathcad, Maple и Mathematica). При вводе этой команды вокруг фигуры появляется обрамляющий ее параллелепипед, который можно вращать мышью (при нажатой левой кнопке)

в том или ином направлении. Отпустив кнопку мыши, можно наблюдать график в пространстве. Интересно, что эта возможность действует даже в отношении двумерных графиков (см. рис. 2.23). При этом вращается плоскость, в которой расположен график. Эта плоскость размещается в упомянутом параллелепипеде.

2.5.5. Операции вставки

В позиции **Insert (Вставка)** меню графического окна есть средства вставки для рисунков. Рисунок 2.23 показывает пример рисунка, в котором выполнены основные операции вставки с помощью команд меню **Insert (Вставка)**. Это нанесение надписей по осям, титульной надписи, надписи внутри рисунка, стрелки, отрезка прямой, легенды и шкалы цветов. На этом рисунке меню **Insert** показано в открытом состоянии.

2.6. Основы форматирования графиков

2.6.1. Форматирование двумерных графиков

Графики в системе MATLAB строятся обманчиво просто. Связано это с тем, что многие свойства графиков установлены по умолчанию. К таким свойствам относятся вывод или скрытие координатных осей, положение их центра, цвет линии графика, ее толщина и т. д. и т. п.

В новых версиях MATLAB для изменения свойств графиков (их форматирования) используются принципы визуального контроля за стилем (видом) всех объектов графиков. Это позволяет легко, просто и наглядно придать графикам должный вид перед записью их в виде файлов на диск. Можно сказать, что в этой части реализованы отдельные принципы визуально-ориентированного программирования графических средств.

Далее мы рассмотрим возможности форматирования графиков, которые, образно говоря, лежат на поверхности. Систематизированное описание интерфейса системы MATLAB, в том числе интерфейса графических окон, дается в следующем уроке.

2.6.2. Форматирование линий графиков

В новой версии MATLAB форматирование графиков стало более строгим и удобным, чем в более ранних версиях. Для этого используются команды **Figure Properties (Свойства фигуры)** и **Axis Properties (Свойства осей)** со всеми необходимыми настройками.

При построении графиков появляется графическое окно. Иногда оно бывает скрыто ранее имеющимися окнами как системы MATLAB, так и других работающих в среде Windows XP/2000/NT4 приложений. Если вы не увидели графика,

заданного для построения, то поищите его в списке открытых окон (приложений), нажимая клавиши **Alt+Tab**, и выберите из списка нужное окно. Окна графики имеют изображение логотипа системы MATLAB. По умолчанию они выводятся с панелью инструментов с рядом кнопок вполне очевидного назначения.

Щелкнув на кнопке **Edit Plot (Редактировать график)** в панели инструментов окна графики и щелкнув по графику, можно заметить, что график выделился: вокруг него появилась рамка. Теперь, указав курсором мыши на тот или иной объект графика и щелкнув снова левой клавишей, можно наблюдать выделение объекта и появление окна его форматирования.

Например, указав в режиме редактирования мышью на линию графика или поверхность (и дважды быстро щелкнув левой клавишей), можно увидеть окно свойств графика, показанное на рис. 2.24 снизу для поверхности. Часть окна графики с выделенным графиком видна сверху. Обратите внимание на появление на линии графика ряда черных квадратиков – они используются для указания курсором мыши именно на линию графика, а не на другие объекты.

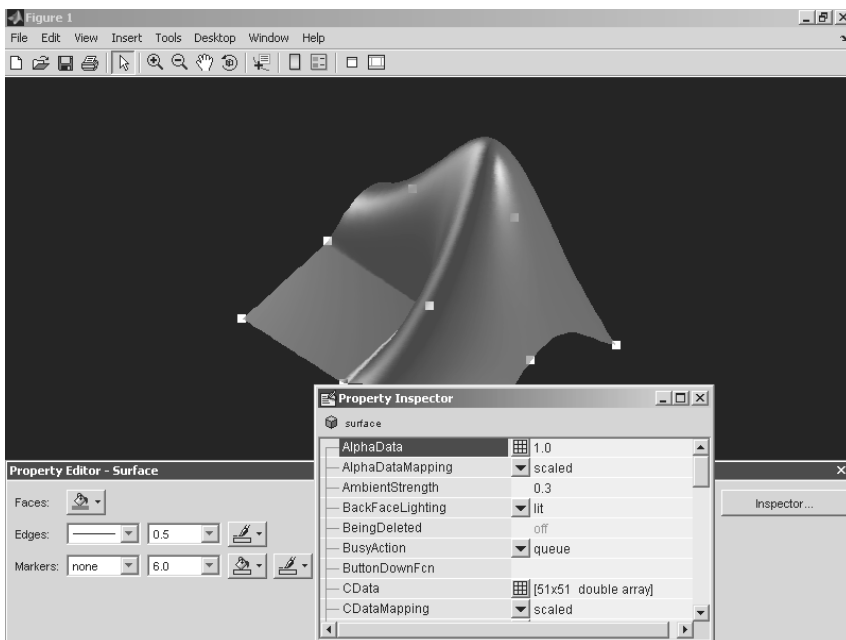


Рис. 2.24. Окно свойств графики и пример работы с инспектором графики

В окне свойств графика можно установить стиль отображения линии, то есть ее вид (например, сплошная линия или пунктирная), ширину и цвет, а также параметры маркеров, отмечающих опорные точки графиков. Работа с окном свойств достаточно очевидна. Полную информацию о свойствах графика позволяет вывести инспектор графики – новое средство, введенное в систему MATLAB 7. Окно

инспектора, показанное на рис. 2.24, выводится при активизации кнопки **Inspector...** В данном случае инспектор отображает свойства поверхности.

2.6.3. Работа с инструментом *Plot Tool*

Рассмотрим работу с редактором графики. Обратите внимание на то, что построение трехмерных графиков сопровождается появлением в рабочем пространстве трех массивов – переменных x , y и z . Они видны в окне выювера рабочего пространства. Полезно подметить и то, что в окне графика в панели инструментов появились новые кнопки:

- **Data cursor** – вывести данные по месту установки курсора;
- **Insert Color Bar** – вставить в рисунок панель цветов;
- **Insert Legends** – вставить легенду;
- **Hide Plot Tools** – скрыть окно редактора графики;
- **Show Plot Tools** – показать окно редактора графики.

Все эти команды очевидны. Стоит только отметить первую команду: если навести курсор на место кривой графика и щелкнуть левой клавишей мыши, то появится всплывающее окошко с координатами точки. При этом действует система автоматического слежения курсором за кривой. Многие команды по обработке графиков в графическом окне соответствуют уже описанным командам, их можно вводить как из меню окна графики, так и с помощью контекстного меню – на рис. 1.16 представлена позиция меню **Insert** в открытом состоянии.

2.6.4. Работа с редактором графики *MATLAB*

Редактор графики – это принципиально новое графическое средство новых версий системы *MATLAB*. Он служит как для редактирования уже созданных *M*-файлами или командами в командной строке графиков, так и для создания графиков заданного пользователем типа. Редактор графики можно ввести из окна графики, из меню этого окна и из меню **Graphics** окна системы *MATLAB*.

Окно редактора графики показано на рис. 2.25. В левой части окна в панели свойств фигур **Figure Palette** имеется список **New Subplots**, позволяющий задавать (с помощью выводимого макета) одно или несколько окон для представления графиков. Под ним находится список переменных графики **Variables**. Активизация той или иной переменной ведет к построению графика представленных в ней данных. Можно строить несколько графиков в одном окне или каждый график в отдельном окне внутри окна представления графиков – оно видно в середине. Рисунок 2.25 демонстрирует возможность построения трех графиков разного типа в разных окнах. Список **Annotations** содержит графические элементы, которые можно ввести в график, осуществив его «раскраску».

В правой части окна имеется панель браузера графики **Plot Browser**. На нем дан список всех графических элементов, из которых состоят отображаемые графики. Любой элемент можно отключить или включить, удалив или установив знак птички в маленьком прямоугольнике, который имеется перед названием гра-

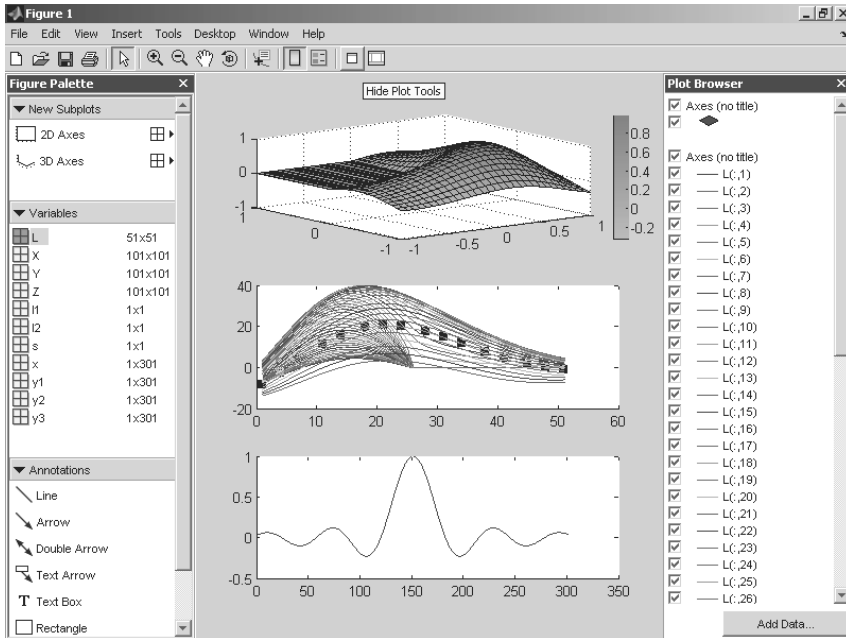


Рис. 2.25. Работа с редактором графики

фического элемента. Работа с панелью инструментов редактора графики вполне очевидна. Последние две кнопки этой панели позволяют закрывать и открывать окно редактора графики.

2.6.5. Форматирование линий графиков и маркеров опорных точек

Для форматирования линий двумерного графика достаточно указать курсором мыши на линию графика и щелкнуть левой клавишей мыши. Линия будет выделена характерными черными квадратиками – рис. 2.26. С помощью редактора свойств графики (в данном случае линии одного графика) или контекстного меню правой клавиши мыши можно задать стиль, толщину и цвет выделенной линии.

Линии двумерных графиков строятся по опорным точкам. В нашем случае опорные точки задаются ранжированной переменной x , имеющей ряд значений от -15 до $+15$ с шагом $0,1$. Эти точки появляются на графике, и с помощью редактора линий графики можно выбрать стиль маркера. На рис. 2.27, к примеру, показано построение графика с маркерами опорных точек в виде окружностей.

Можно задавать размеры маркеров, цвет их закрашки и цвет окантовки. Для выбора цвета используется палитра цветов, видимая в правом нижнем углу редак-

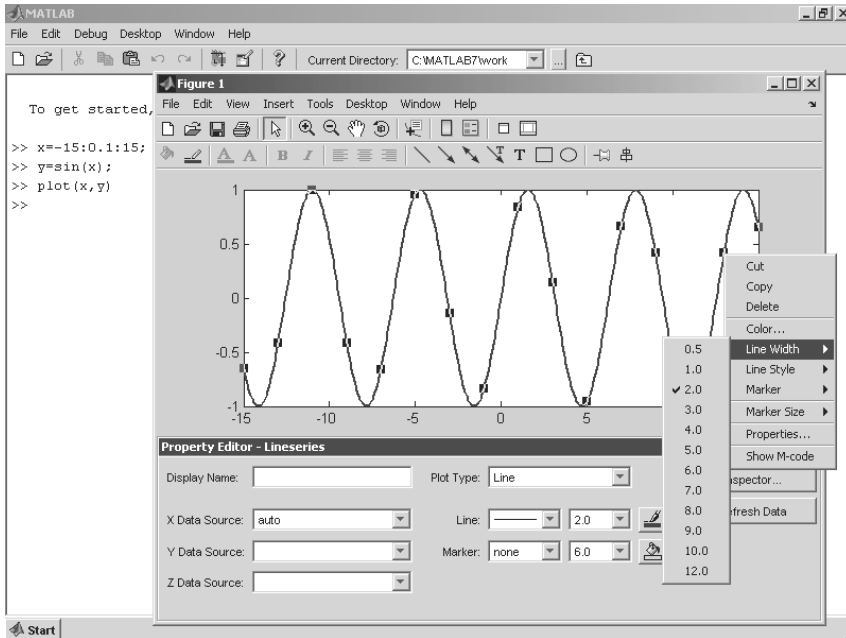


Рис. 2.26. Пример форматирования линий двумерного графика

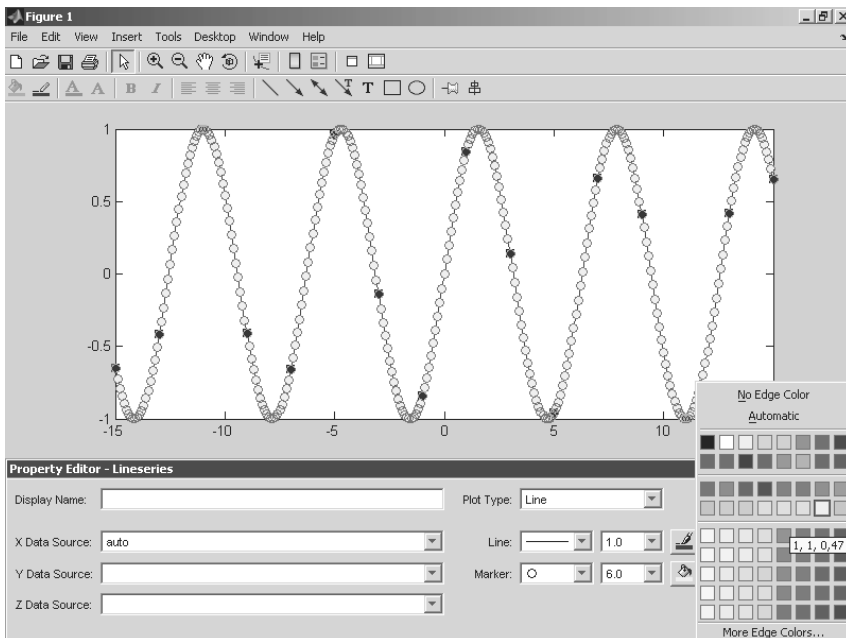


Рис. 2.27. Пример задания параметров маркеров и построения графика с ними

тора линий и маркеров. Маркеры можно задавать в виде окружностей, прямоугольников, крестиков, ромбиков и т. д. Применение маркеров иногда делает графики более наглядными.

2.6.6. Форматирование линий и маркеров для графика нескольких функций

Если строится график нескольких функций, то можно форматировать линии и маркеры каждой кривой отдельно. Выполним следующие команды:

```
>> x=-6:.1:6;
>> plot(x, sin(x), x, sin(x).^3, x, sin(x).^5);
```

Рисунок 2.28 показывает пример такого форматирования для графика, полученного исполнением этих команд. Для форматирования каждой линии надо указать на нее курсором мыши и дважды щелкнуть ее левой клавишей. Соответствующая линия будет выделена, и ее параметры можно менять с помощью редактора свойств линии – его окно видно на рис. 2.28 снизу.

Кстати, обратите внимание на то, как заданы степени синуса. Записать эти выражения в виде $\sin(x)^2$ и $\cos(x)^2$ будет грубейшей ошибкой, поскольку x – здесь вектор. Операторы \cdot^{\wedge} в данном случае дают поэлементное возведение в степень, что и нужно для построения графиков этих функций.

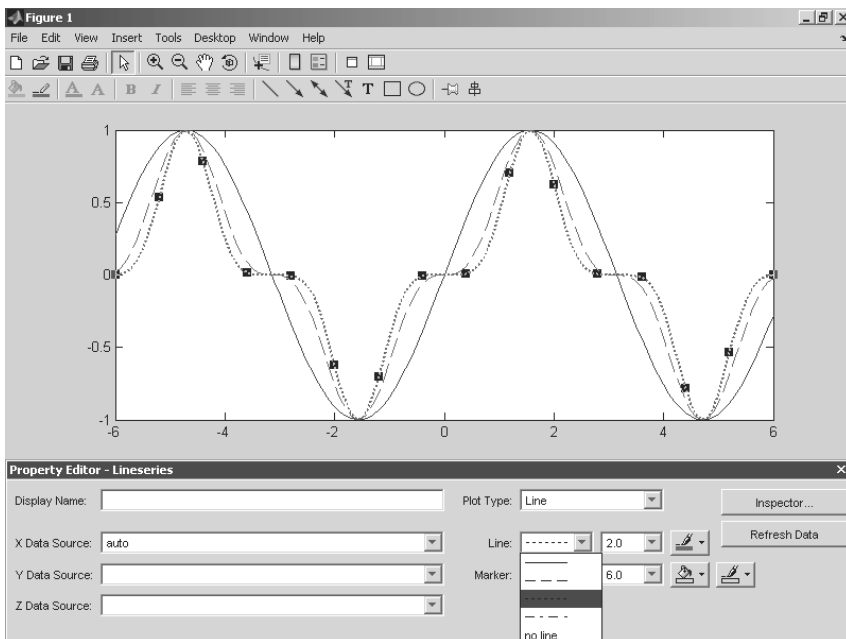


Рис. 2.28. Пример форматирования для графика трех функций

2.6.7. Форматирование осей графиков

Аналогично описанным выше правилам выполняется форматирование и других объектов графиков. Например, указав курсором мыши на оси графиков (на них тоже есть метки в виде черных квадратиков) и дважды щелкнув левой клавишей мыши, можно увидеть появление окна форматирования редактора свойств (рис. 2.29), настроенного на форматирование осей.

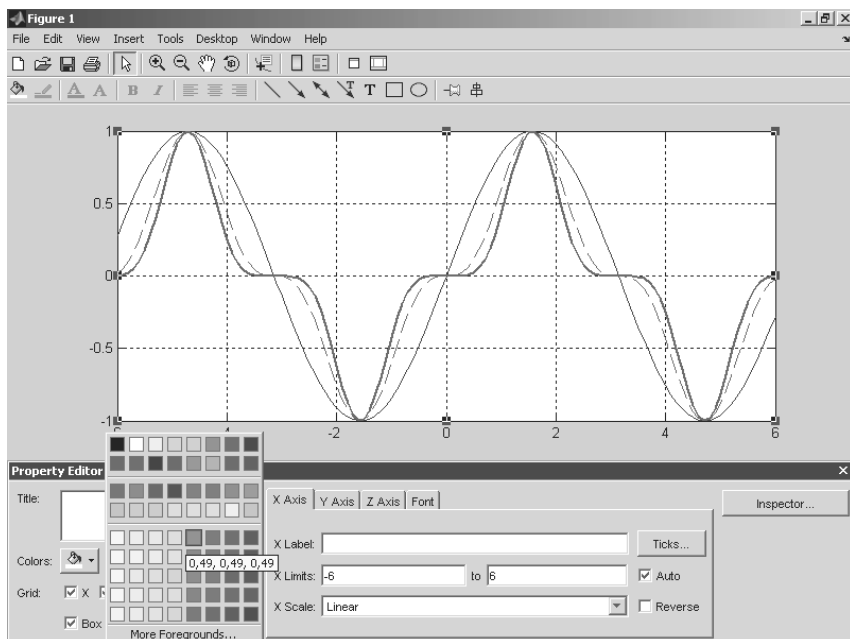


Рис. 2.29. Пример форматирования осей графика

Окно графического редактора свойств графики имеет множество вкладок, настройки которых довольно очевидны, и ничто не мешает читателю поэкспериментировать с ними несколько минут. Это позволит понять простоту и одновременно высокую эффективность средств форматирования объектов графики. Например, вы можете задать линейный или логарифмический масштаб осей (вкладка **Scale**), нормальное или инверсное направление осей (X , Y , а в случае трехмерных графиков и Z), показ сетки (параметр **Grid Show**), изменить стиль осей и цвета фона (вкладка **Style (Стиль)**), нанести у осей надписи (вкладка **Label (Ярлык)**) и прочее.

2.6.8. Позиция **Tools** меню окна графики

Большие возможности в форматировании графиков открывает позиция **Tools (Инструменты)** окна графики. Ее команды представлены на рис. 2.30. Здесь мож-

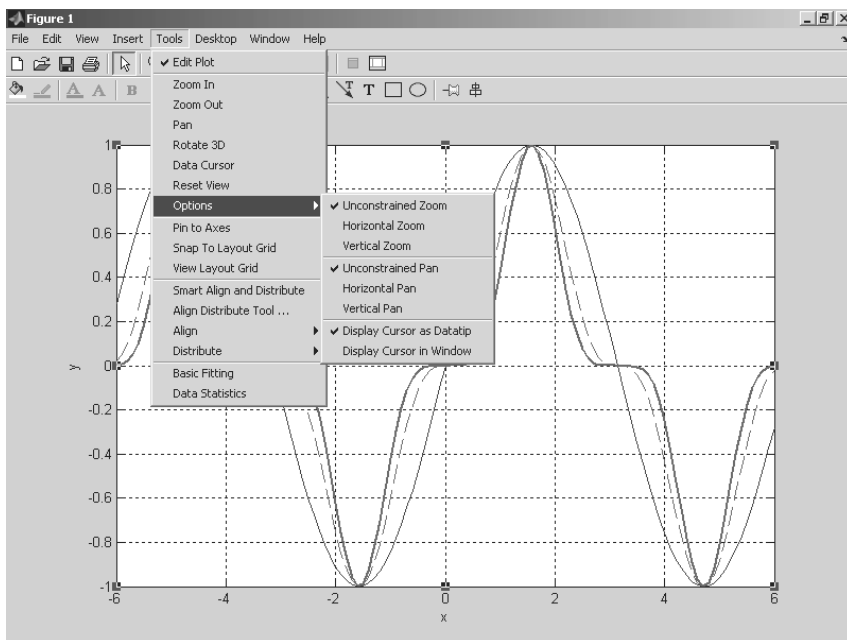


Рис. 2.30. Окно графики с открытой позицией **Tools** меню

но задать опции редактирования графика, изменения его размера, вращения, вывода графического курсора для определения координат любой точки графика и многие другие опции и команды.

Особо стоит отметить обработку графика внутри его окна – опцию **Basic Fitting** и опцию представления статистики графику **Data Statistics**. Работу с ними мы рассмотрим чуть ниже.

2.6.9. Нанесение надписей и стрелок прямо на график

Дополнительно на график можно нанести надписи с помощью кнопки панели инструментов с буквой **A**. Место надписи фиксируется щелчком мыши. Полученную таким образом надпись можно выделить и перенести мышью в любое другое место. Рисунок 2.31 показывает процесс создания еще двух надписей с переносом их текстового блока в нужное место. Надписи сделаны с разным размером символов и разным стилем. Особенно приятно, что при задании на надписи возведения в степень знаком \wedge надпись на экране отображается в естественном математическом виде (степень в виде верхнего индекса).

На рис. 2.31, в частности, показано задание надписей разным стилем, а также задание стрелки с помощью соответствующей кнопки панели инструментов. Эту стрелку в режиме редактирования графика можно перемещать и вращать мышью,

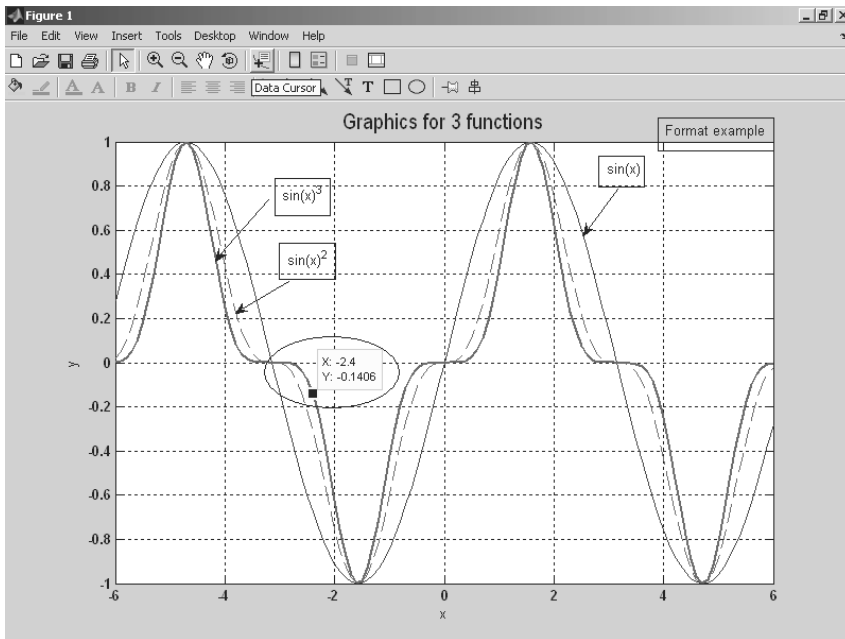


Рис. 2.31. Окончательно отформатированный график трех функций

а еще менять ее длину. Можно также наносить на график и обычные линии (без стрелки).

2.6.10. Применение графической «лупы»

На панели инструментов есть кнопки с изображением лупы и знаками + и -. С их помощью можно исполнять команды Zoom In (+) (**Увеличить**) и Zoom Out (-) (**Уменьшить**). Это позволяет увеличивать или уменьшать масштаб просмотра изображения. При этом команда Zoom In интересна еще одной возможностью – с ее помощью можно выделять часть графика перемещением мыши с нажатой левой клавишей – рис. 2.31. Область выделения отмечается прямоугольником из тонких точечных линий.

Отпустив левую клавишу мыши, можно наблюдать построение выделенной части графика на всем окне – рис. 2.32. С помощью команды Zoom Out можно восстановить график в прежнем масштабе. Таким образом реализуется графическая «лупа».

К сожалению, правильно отображается только выделенная часть кривых графика. Прочие графические объекты, например надписи и стрелки, остаются на своих местах, так что их положение относительно линий графика нарушается.

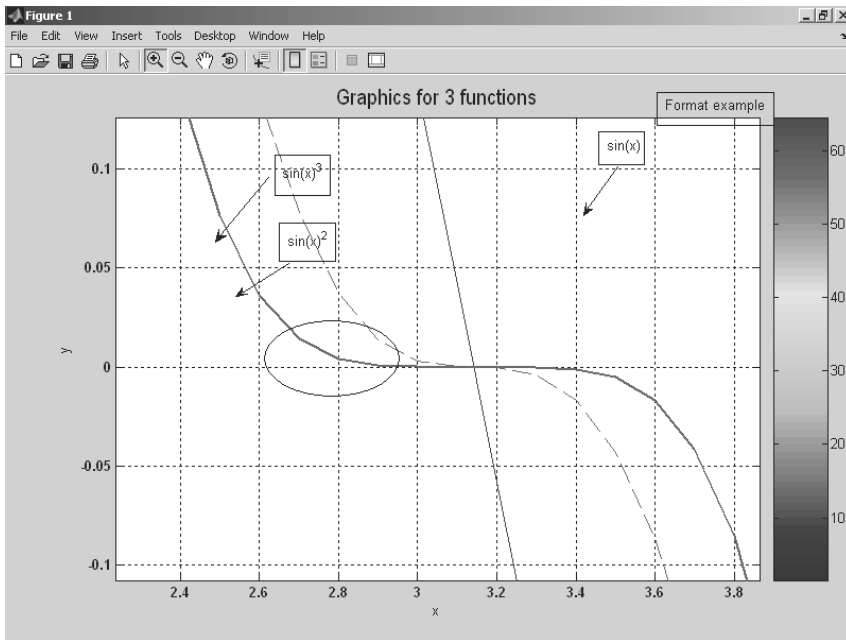


Рис. 2.32. Пример просмотра части графика

2.6.11. Построение легенды и шкалы цветов на графике

Дополнительно можно изменить размеры графика (см. меню **Tools (Инструменты)** и его команды **Zoom In (Увеличить)** и **Zoom Out (Уменьшить)**), начать поворот графика мышью (команда **Rotate 3D**), добавить отрезок прямой или иной графический примитив (подменю **Add**) и подключить к графику *легенду* – пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, – см. пример на рис. 1.11. Если график содержит три кривые, то легенда будет представлять собой обозначение этих трех линий в правом верхнем углу рисунка. Каждая линия имеет тот же цвет, что и на графике (и тот же стиль). Возможен также вывод шкалы цветов – см. рис. 1.12 и рис. 2.23.

Следует еще раз отметить, что все описанные возможности форматирования графиков доступны и программным способом, путем задания соответствующих графических команд, параметров и примитивов. Например, команда `text(x, y, 'legend')` позволяет задать надпись 'legend' с началом, имеющим координаты (x, y) . Если после первого апострофа перед текстом поместить параметр `\leftarrow`, то надпись (легенда) появится после стрелки с острием, обращенным влево. Аналогично параметр `\rightarrow` после надписи задает вывод

стрелки после надписи с острием, обращенным вправо. Эта возможность позволяет пометить не только кривые, но и отдельные точки на них. Возможно также применение команды `legend('s1', 's2', ...)`, выводящей легенду обычного вида – отрезки линий графиков с поясняющими надписями 's1', 's2' и т. д.

2.6.12. Работа с камерой 3D-графики

В отличие от двумерных (2D) графиков, форматирование трехмерных графиков содержит ряд дополнительных возможностей. Покажем их на простом примере построения 3D-графики с помощью следующих простых команд:

```
>> Z=peaks(40);  
>> mesh(Z);
```

Здесь первая команда создает массив точек поверхности с помощью одного из ряда встроенных в ядро системы MATLAB готовых описаний таких поверхностей. Вторая команда просто строит эту поверхность по опорным точкам с использованием интерполяции для промежуточных точек. Таким образом создается цветная каркасная поверхность, как бы сотканная из разноцветных проволок. На рис. 2.33 показано построение этой поверхности вместе со специальной панелью инструментов трехмерной графики, названной в оригинале **Camera (Камера)**.

Несмотря на множество кнопок, пользование панелью инструментов 3D-графики достаточно просто, если представить себе, что вы смотрите на предмет через

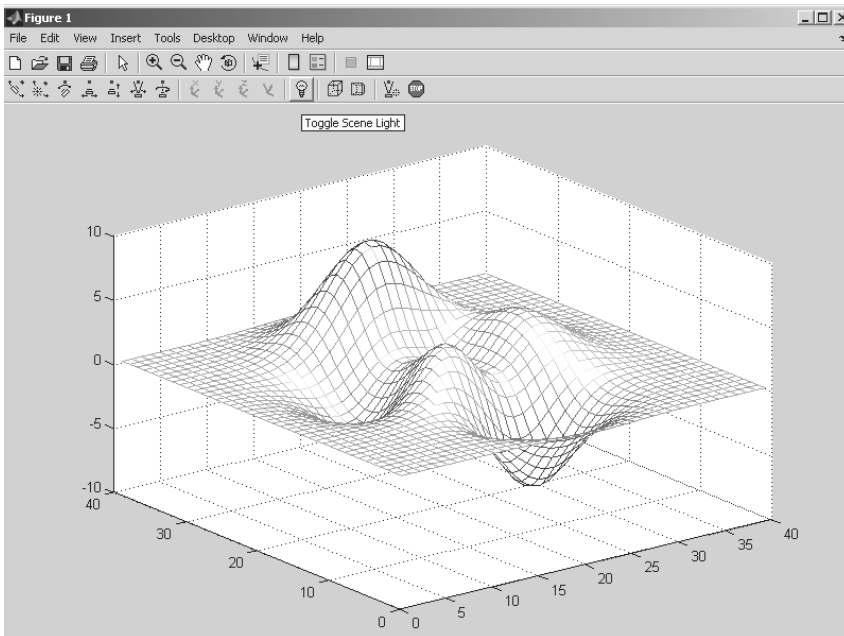


Рис. 2.33. Пример построения каркасного 3D-графика

объектив фотокамеры. Наглядные рисунки на кнопках поясняют смысл их действия – это перемещение и вращение 3D-рисунков относительно тех или иных координатных осей, включение отображения перспективы, изменение цветовой схемы и др.

Рисунок 2.34 показывает, что приемы форматирования двумерной графики можно использовать при работе с трехмерной графикой – вывод надписи на график, вывод легенды и шкалы цветов.

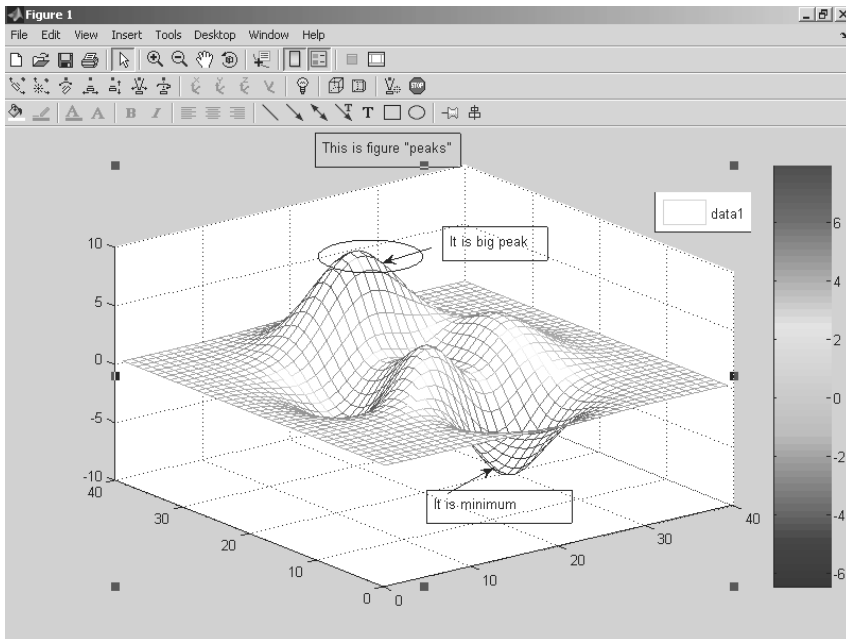


Рис. 2.34. Пример форматирования трехмерного графика

Для управления положением и вращением трехмерного графика можно использовать клавиши перемещения курсора. Эффект вращения и перемещения (приближения) графика иллюстрирует рис. 2.35, где показан график после его поворота при нажатой клавише →. В отличие от поворота мышью (также возможного), перемещение и повороты с помощью клавиш курсора при выбранном типе перемещения дают плавное перемещение или вращение фигуры. Таким образом осуществляется анимация (оживление) трехмерной графики.

Следует отметить, что скорость вращения фигур при анимации во многом зависит от скорости работы ПК, на котором установлена система MATLAB, от примененной видеокарты и ее настроек. На старых ПК она может оказаться довольно низкой.

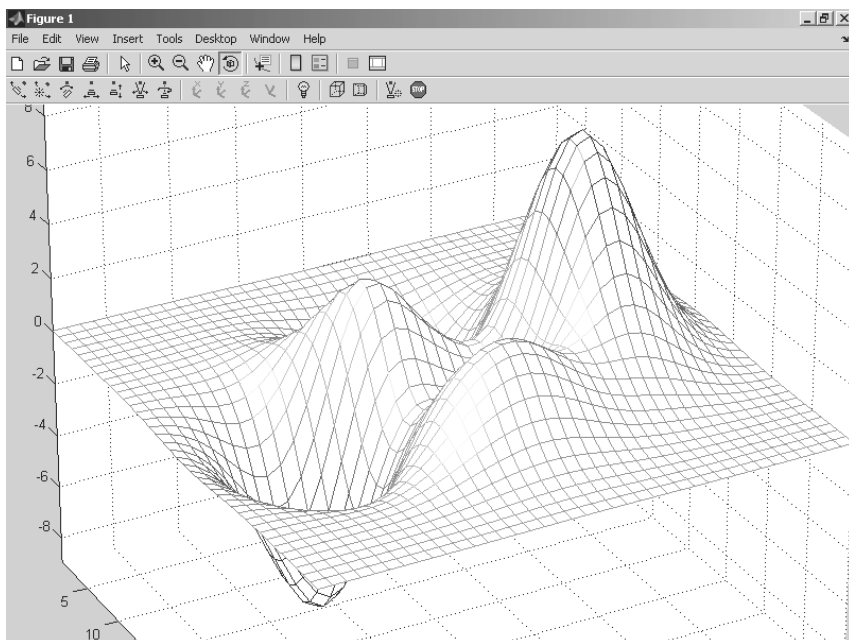


Рис. 2.35. Стоп-кадр вращения трехмерного графика

2.7. Работа с Мастером импорта данных

2.7.1. Открытие окна Мастера импорта данных

MATLAB предоставляет расширенные возможности по импорту данных. Для этого служит специальный Мастер импорта Import Wizard. Рассмотрим доступ к нему.

При исполнении команды **Import Data...** в меню **File** открывается окно **Import** – рис. 2.36. В сущности, это обычное окно открытия файлов. Внизу его представлен открытый список типов файлов, содержащих импортируемые файлы.

Выбрав нужный файл, можно наблюдать появление окна импорта данных **Import Wizard**. На рис. 2.37 это окно показано с загруженным файлом `saпoe.tif`, который представляет цифровую фотографию, заданную в формате TIF. Это окно имеет две большие панели: слева – с данными о загруженных файлах, а справа – с поначалу пустым окном просмотра массива, представляющего данные.

Вид окна Мастера импорта достаточно прост, и его возможности вполне очевидны. Однако они зависят от того, какие данные (например, текстовые, электронных таблиц, бинарные и т. д.) импортируются.

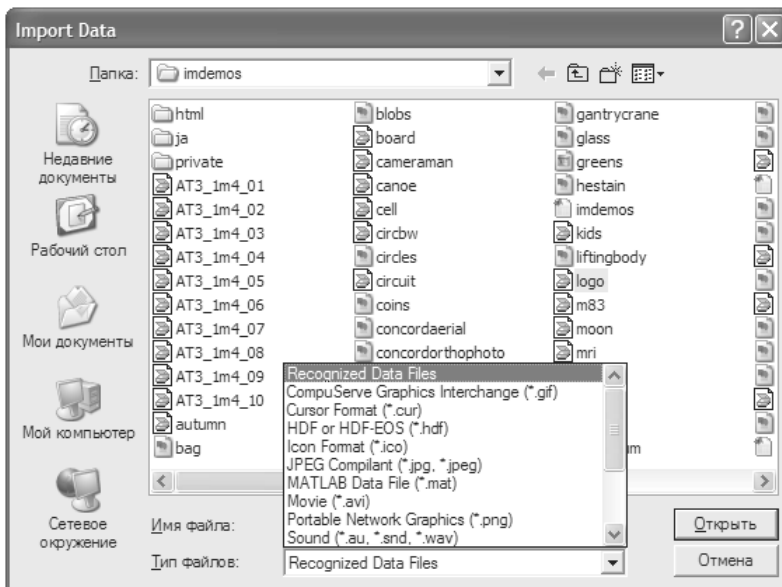


Рис. 2.36. Окно **Import** импорта файлов с открытым списком их типов

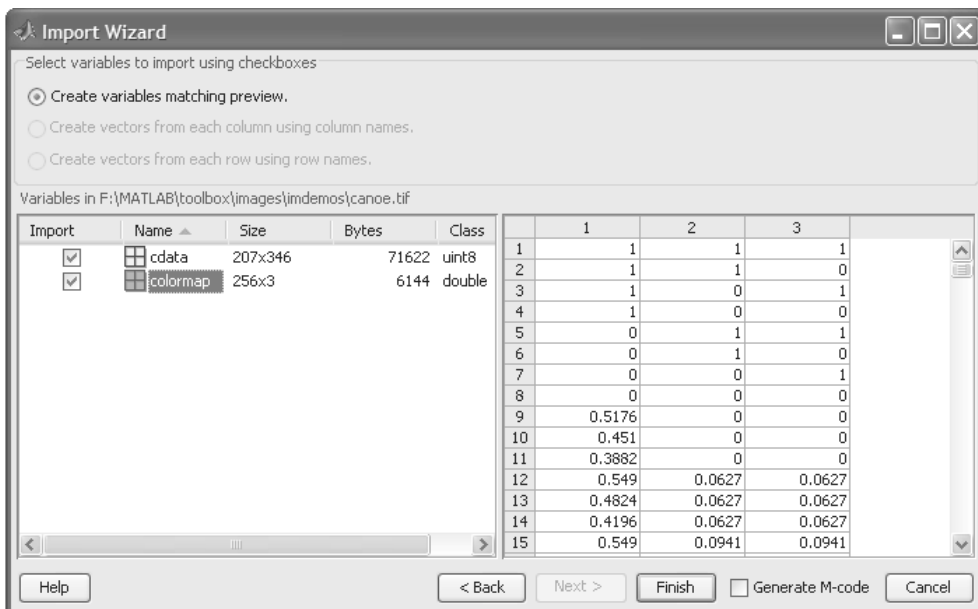


Рис. 2.37. Окно Мастера импорта данных **Import Wizard**

2.7.2. Информация об импортируемых бинарных данных

Правая панель Мастера импорта при работе с бинарными файлами изображений отображает содержимое двух вкладок:

- **Image Preview** – предварительный просмотр изображений (к сожалению, только небольших);
- **dvpr** – просмотр содержимого массива с текущим именем `dvpr` (при загрузке другого файла имя этой вкладки будет иным).

Для просмотра изображения надо нажать кнопку **Show Image**. При этом запускается специальный выювер изображений **Image Viewer**. Часть изображения можно выделить с помощью мыши, если при ее нажатой левой клавише поместить нужную часть изображения в образующийся при движении мыши прямоугольник. Отпустив левую клавишу мыши, можно наблюдать увеличенный фрагмент этого изображения. Просто щелчок левой или правой клавиши мыши увеличивает или уменьшает масштаб изображения. К сожалению, просмотр возможен только для маленьких изображений.

В правой панели можно наблюдать цифровые данные о загруженном массиве бинарных данных – рис. 2.37. Для этого достаточно открыть вкладку с именем файла. Более удобно использовать браузер рабочего пространства. Нажав кнопку **Finish**, можно занести импортируемый массив данных в рабочее пространство в виде одноименной с ним переменной. Если после этого открыть браузер рабочего пространства, то, щелкнув мышью по имени массива-переменной, можно наблюдать все данные о загруженном массиве бинарных данных – рис. 2.38.

Браузер рабочего пространства, как уже отмечалось, дает возможность доступа к любому элементу массива (даже многомерного), что открывает широкие возможности в самом тонком редактировании массивов, в том числе представляю-

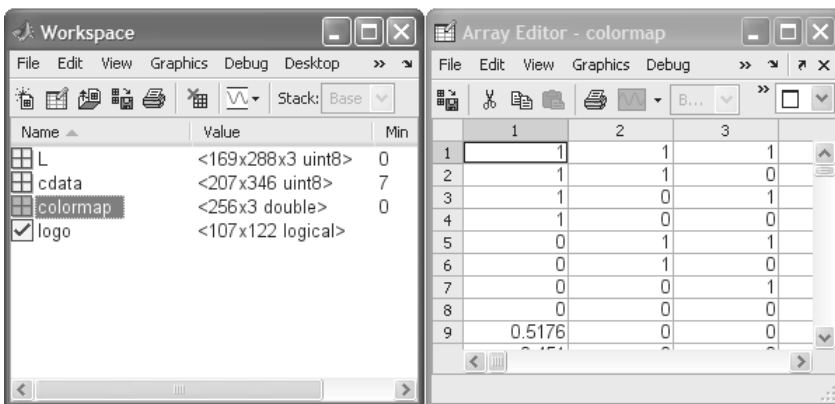


Рис. 2.38. Просмотр переменной `colormap`, созданной в рабочем пространстве Мастером импорта

щих изображения. Однако это очень кропотливая работа, которая более просто и наглядно делается современными редакторами изображений.

2.7.3. Импорт данных *mat*-формата

Многие встроенные функции системы MATLAB представлены файлами *mat*-формата. Мастер импорта дает удобные средства для их загрузки и просмотра. На рис. 2.39 представлен вид окна Мастера импорта после загрузки файла *logo.mat*, задающего вывод логотипа системы MATLAB.

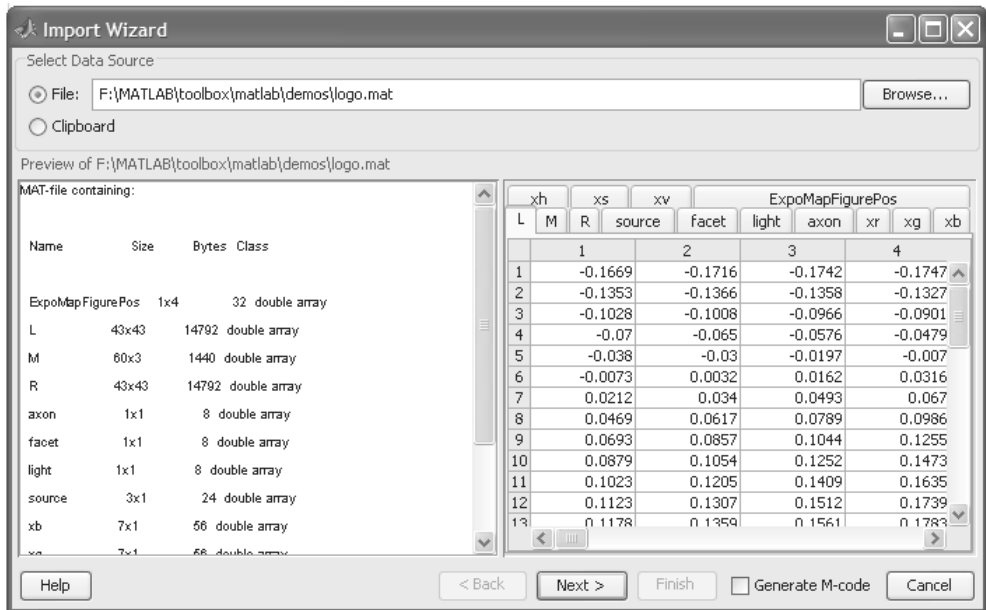


Рис. 2.39. Первое окно Мастера импорта с данными файла *logo.mat*

Нетрудно заметить, что данный файл содержит множество переменных, список которых дан в левой части окна. Нажав кнопку **Next**, можно перейти к другому окну Мастера импорта, показанному на рис. 2.40. Здесь в сущности представлена та же информация, но в несколько ином виде. Переменные представлены как массивы, и в их списке можно выделить только нужные из них.

Нажатие клавиши **Finish** теперь загрузит все массивы в рабочую область и создаст переменную с именем файла.

2.7.4. Импорт данных текстового формата

Аналогичным образом происходит импорт данных текстового формата. На рис. 2.41 показано первое окно Мастера импорта при загрузке текстового файла. В качестве его взят файл лицензии *license.txt*.

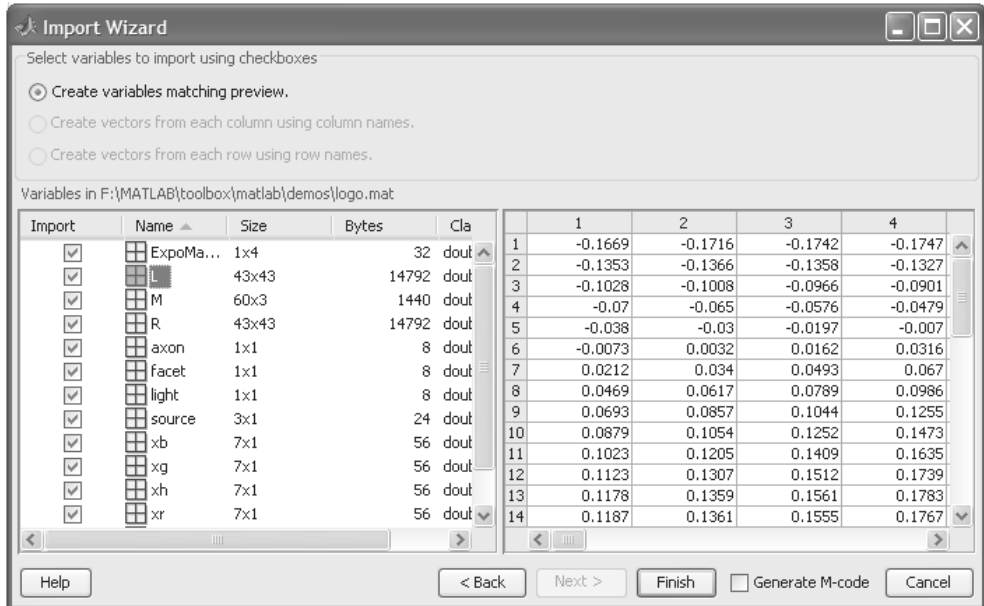


Рис. 2.40. Второе окно Мастера импорта с данными файла logo.mat

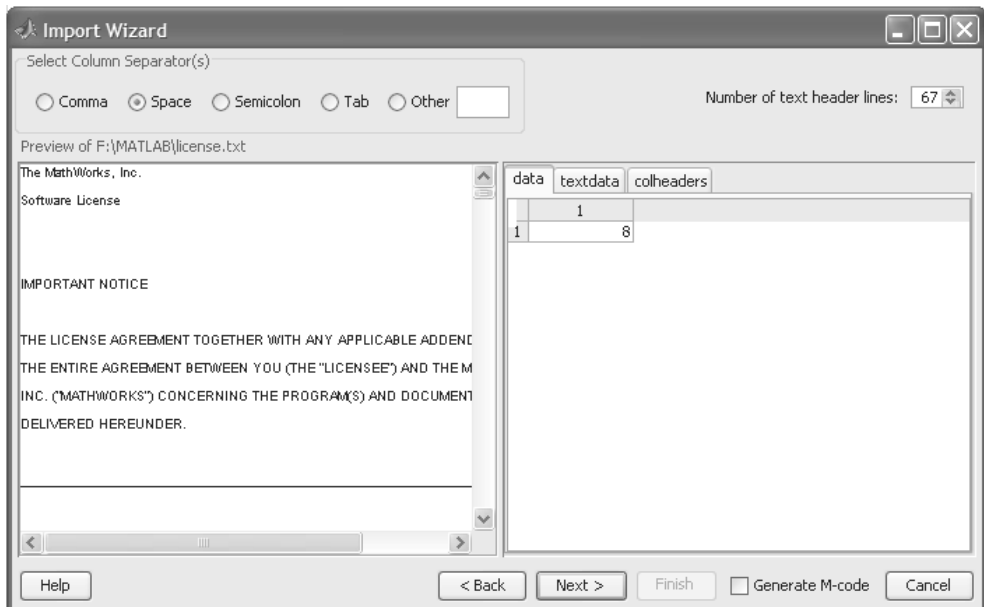


Рис. 2.41. Первое окно Мастера импорта с данными файла license.txt

Нетрудно заметить, что в левой части окна представлено начало полного текста лицензии. Нажав кнопку **Next**, можно наблюдать второе окно с данными этого файла – рис. 2.42. Теперь в левом окне представлен каталог данных файла, а в правом – начало текста лицензии.

Работа с импортируемыми текстовыми файлами вполне очевидна.

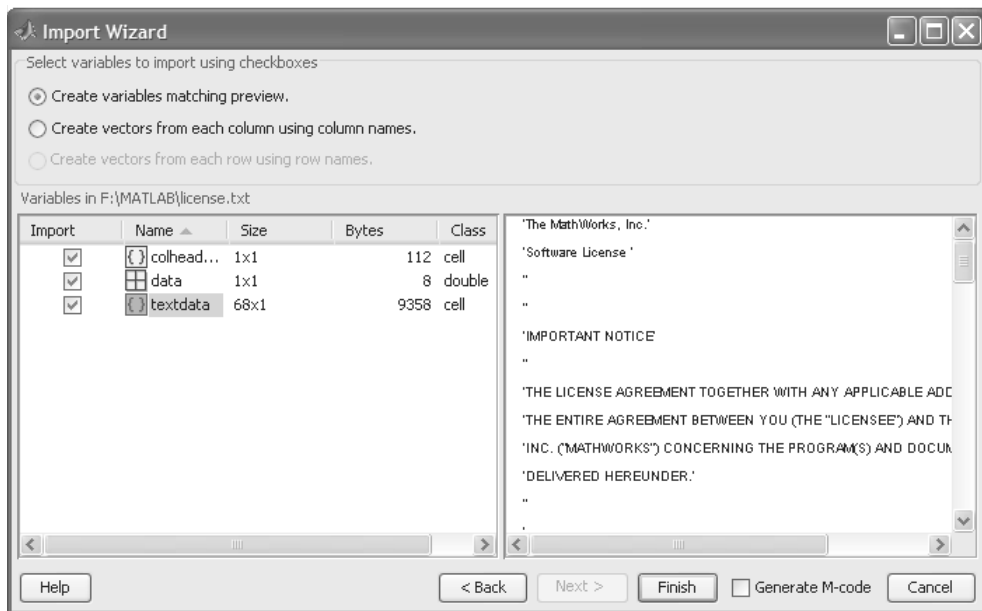


Рис. 2.42. Второе окно Мастера импорта с данными файла *license.txt*

2.7.5. Об экспорте данных

Данные, размещенные в рабочей области системы MATLAB, можно экспортировать. Для этого используются обычные файловые операции записи файлов в текстовые или иные форматы. Подробное описание этих операций, а также операций импорта с командами, вводимыми из командной строки, дано в уроке, описывающем обработку и сохранение данных. Операции импорта и экспорта файлов открывают обширные возможности по обмену данными между системой MATLAB и другими программами, а также по управлению различными внешними устройствами.

2.8. Работа со справкой и демонстрационными примерами

Как и любая современная программа, MATLAB имеет электронную справочную систему, или, проще, *справку*. Она представляет собой набор электронных статей,

оформленных в виде HTML-файлов. Такая организация справочной системы имеет два очевидных преимущества: для просмотра файлов, когда система помощи находится только на сервере, может использоваться любой браузер Интернета, и при этом имеется возможность обновления документации с WWW-сайтов. Справка в новых версиях MATLAB весьма детальная и огромная – размещается на двух CD-ROM. Увы, но она англоязычная!

2.8.1. Запуск справочной системы Help Desk

Для запуска справочной подсистемы следует использовать команду **Full Product Family Help** в позиции меню **Help**. При этом запустится браузер и откроется основное окно справки, показанное на рис. 2.43.

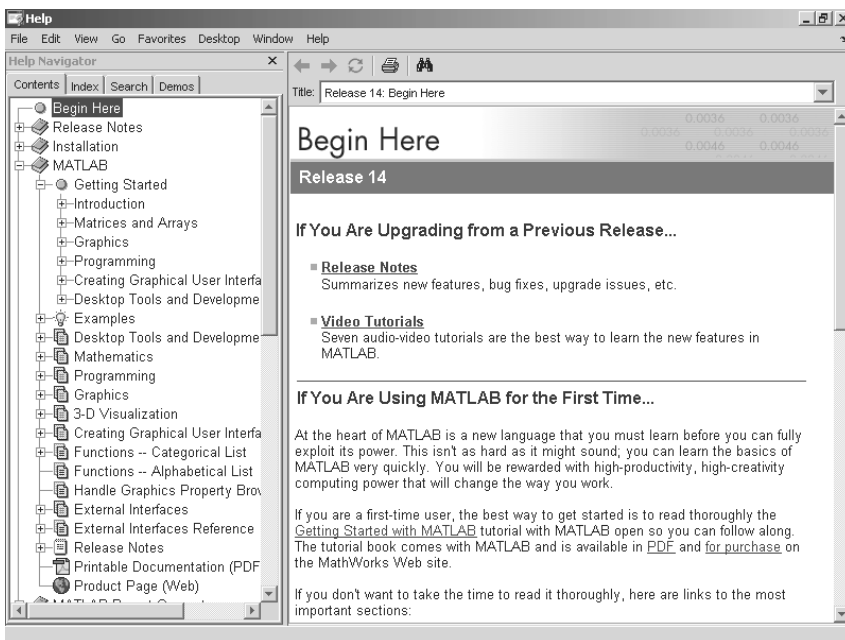


Рис. 2.43. Основное окно справки по системе MATLAB 7 (R14)

Каждый раздел справочной системы представлен в виде гипертекстовой ссылки (это подчеркнутые снизу надписи), активизация которой приводит к переходу на соответствующую HTML-страницу.

2.8.2. Справка по функциям и полнотекстовый обзор

Будучи крупной математической системой, MATLAB имеет многие сотни функций, запомнить свойства и синтаксис которых трудно даже пользователю-профессионалу. Да и нужно ли? Справочная система MATLAB позволяет найти информацию по нужной функции в считанные секунды. Для этого в левой части окна рис. 2.43 надо найти раздел **MATLAB Functions Alphabetical List**. При этом откроется окно рис. 2.44 с гиперссылками функций в алфавитном порядке.

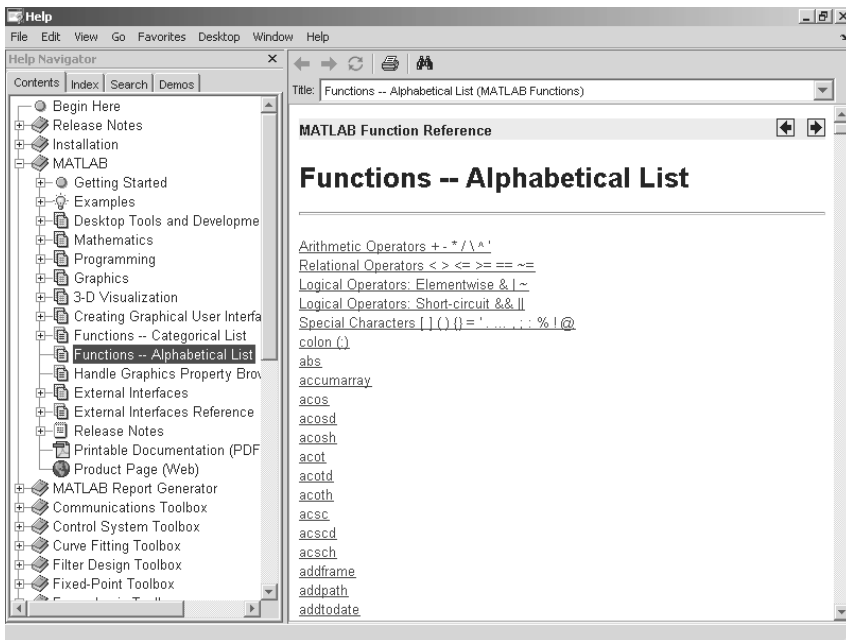


Рис. 2.44. Окно со списком функций MATLAB 7 в алфавитном порядке

Уточняя доступ к нужной функции в левой части окна справки или используя соответствующую гиперссылку в правой части окна, можно выйти на список функций и выбрать нужную для просмотра. Рисунок 2.45 дает пример просмотра функции `abs`, вычисляющей абсолютное значение аргумента в виде числа или массива чисел.

В справке по этой функции (и всем другим функциям) дается пример, который можно выделить и перенести в окно сессии MATLAB для исполнения и сравнения с приведенным в справке результатом. Выделенные фрагменты кодов (но не результаты вычислений) можно сразу исполнить с помощью команды **Evaluate Selection** контекстного меню правой клавиши мыши.

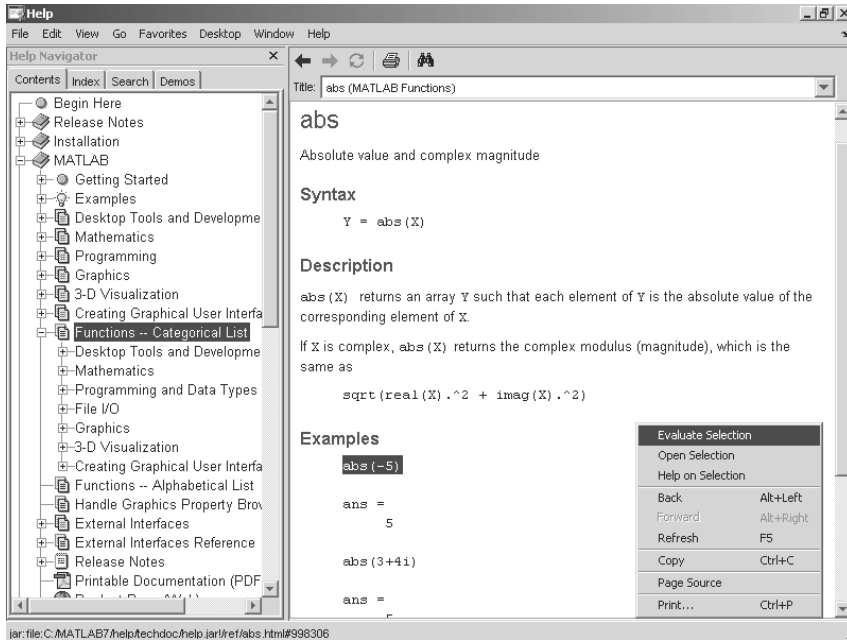


Рис. 2.45. Пример просмотра справки по функции `abs`

Нетрудно заметить, что в левой части окна имеется ряд вкладок:

- **Contents** – справка по контексту (оглавлению разделов);
- **Index** – справка по индексному указателю (алфавитному каталогу);
- **Search** – поиск справки по заданной ключевой фразе или слову.

Заметим, что вкладка **Favorite** – доступ к онлайн-справке через Интернет – в новой версии MATLAB 7 отсутствует. Похоже, что оперативная работа в Интернете с помощью встроенных средств MATLAB себя не вполне оправдала. Оно и понятно – трудно конкурировать со специально разработанными для работы в Интернете браузерами, такими как Microsoft Internet Explorer или Opera.

Работа со средствами справки очевидна для каждого пользователя, имеющего хотя бы небольшой опыт работы со справочными системами. Отметим лишь, что позиция **Search** позволяет как искать данные по отдельной фразе или слову, так и находить все разделы справки, где эта фраза или слово встречаются. Пример такого рода справки представлен на рис. 2.46.

Обратите внимание на то, что теперь в левой части окна справки имеется перечень гиперссылок на все разделы справки, в которых встречается заданное слово **abs**. А в правой части окна дается раздел, который выделен в левой части окна, – в нашем случае это начало обширного листа с алфавитным каталогом всех функций системы MATLAB.

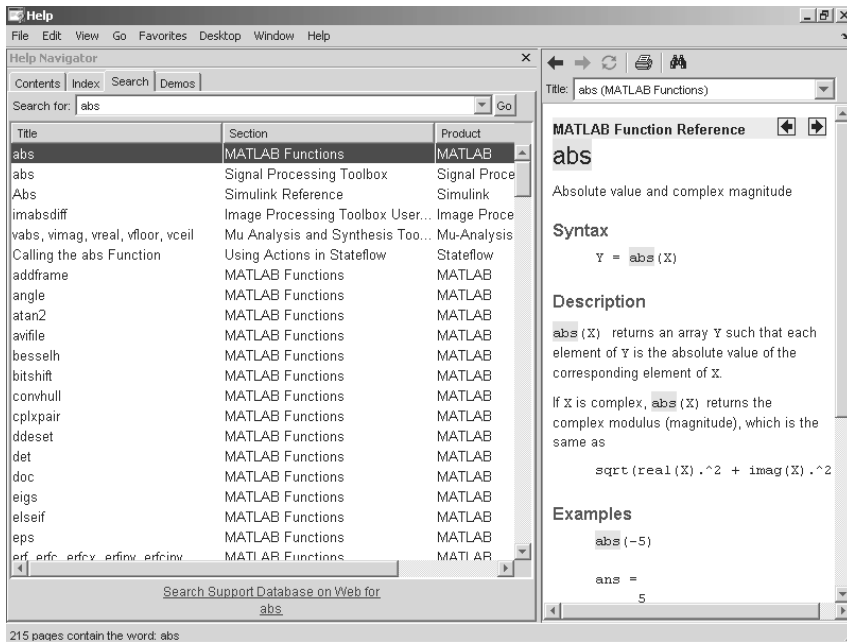


Рис. 2.46. Пример поиска всех разделов справки, в которых упомянута функция `abs`

Совершенно аналогично происходит работа со справочной системой любого расширения системы MATLAB. Достаточно найти гиперссылку на нужный пакет расширения в левой части окна справки и уточнить раздел справки – она появится в правой части окна справки. Хотя окно справки и позиция **Help** меню содержат ряд нами не описанных элементов и команд, освоение работы со справкой вряд ли вызовет даже у начинающего пользователя трудности – справочная система рассчитана на работу по интуиции, она проста и наглядна.

2.8.3. Работа с демонстрационными примерами

В меню **Help** имеется команда **Demos**, содержащая доступ к галерее примеров применения системы MATLAB. При запуске этой команды появляется окно демонстрационных примеров **MATLAB Demo Window**, показанное на рис. 2.47. Это же окно можно вызвать выполнением команды `demo` в командном режиме или открыть вкладку **Demos** в окне справки.

В MATLAB 7 наглядному представлению демонстрационных примеров уделено большое внимание. Эти примеры являются важной составной частью системы. В MATLAB 7 демонстрационные примеры представлены их наглядными графиче-

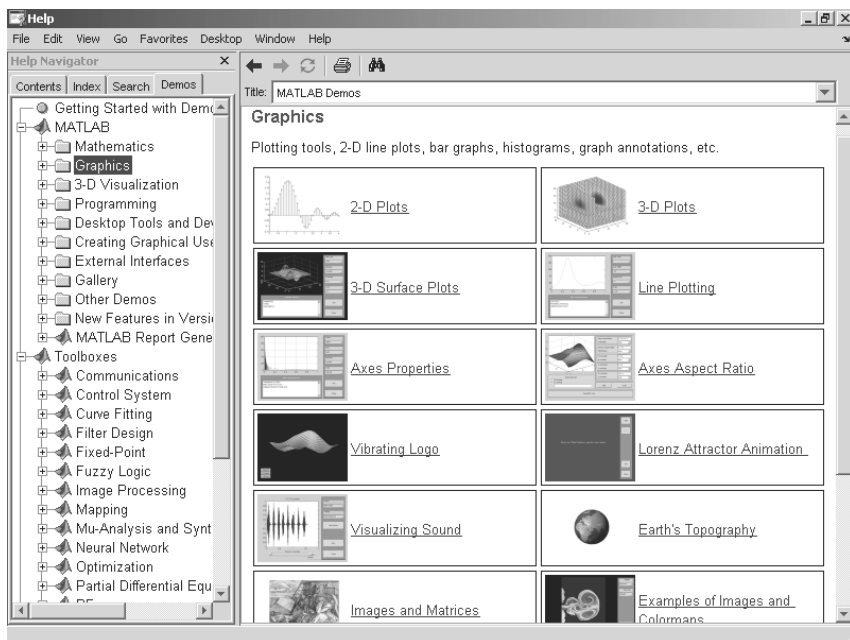


Рис. 2.47. Доступ к демонстрационным примерам

ческими образами – см. рис. 2.47. Выбрав раздел примеров (щелчком мыши), следует затем выбрать нужный пример. На рис. 2.48 представлен один из примеров на построение изображения рентгеновского снимка.

Читателю рекомендуется просмотреть с десяток-другой примеров применения системы MATLAB, что позволит оценить ее уникальные возможности в решении самых разнообразных задач науки и техники. Нередко можно найти пример, близкий к решаемой пользователем задаче, и тогда его изучение и модификация могут существенно уменьшить время, нужное для окончательного решения этой задачи.

2.9. Интерфейс и новые возможности MATLAB R2007

2.9.1. Интерфейс MATLAB R2007a по умолчанию

Интерфейс систем MATLAB всегда отличался консервативностью и мало менялся от версии к версии. Однако последние годы разработчик системы – корпорация MathWorks стала выпускать на рынок по 2–3 новые подверсии ежегодно. Назвать их полноценными новыми версиями можно с известной степенью условности,

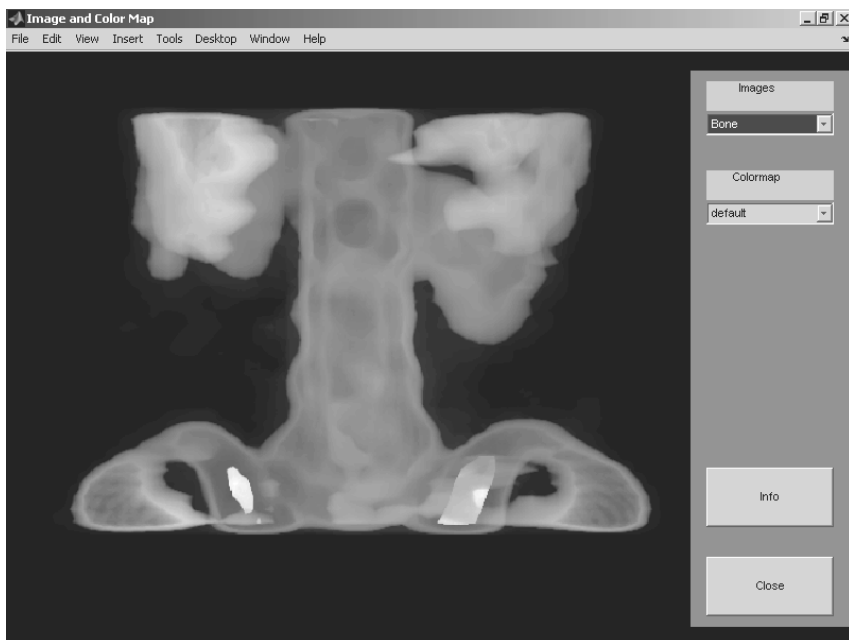


Рис. 2.48. Пример просмотра изображения рентгеновского снимка

поскольку изменения в базовой системе (напомним, это MATLAB 7.*) были незначительны. Тем не менее мы опишем эти изменения на примере новейших (на момент подготовки книги) подверсий системы MATLAB R2007a,b.

Интерфейс системы MATLAB R2007a, используемый по умолчанию (Default), представлен на рис. 2.49. Здесь представлены окно командного режима **Command Window**, главное меню, панель инструментов, вкладки текущей директории **Current Directory**, рабочего пространства **Workspace** и истории команд **Command History**. Кроме того, выведены окна редактора m-файлов и информации о системе **About MATLAB**.

Сравнение рис. 2.49 с рис. 1.1 показывает, что налицо практически полная преемственность общего вида интерфейса и его деталей. Лишь окно **About MATLAB** четко указывает на конкретную версию системы и дату ее выпуска (для MATLAB 2007a это конец января 2007 г.).

2.9.2. Упрощенный интерфейс MATLAB R2007a

Для задания вида интерфейса служит позиция главного меню **Desktop**. На рис. 2.50 показан упрощенный интерфейс MATLAB 2007a, в котором присутствуют только титульная строка, меню, панель инструментов, окно командного режима и строка

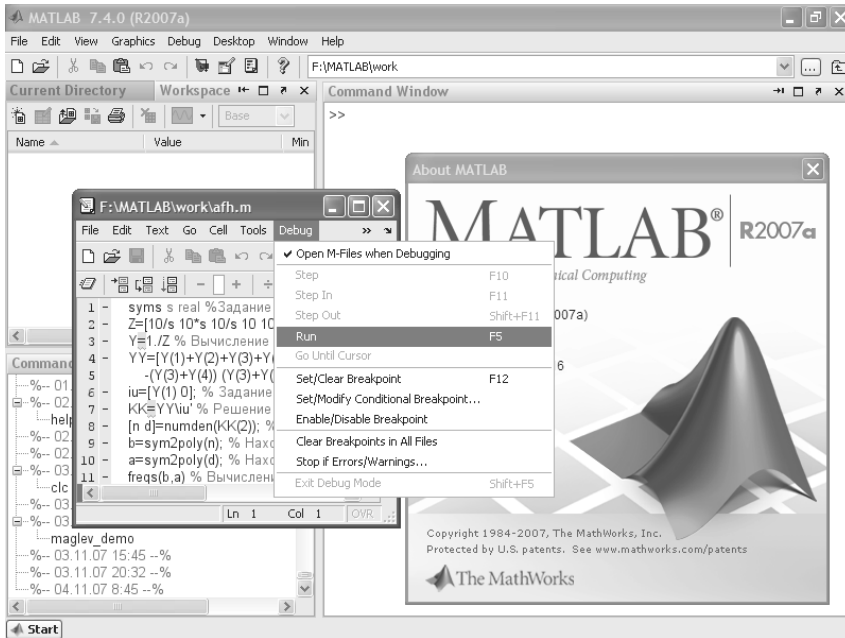


Рис. 2.49. Интерфейс системы MATLAB 2007a по умолчанию

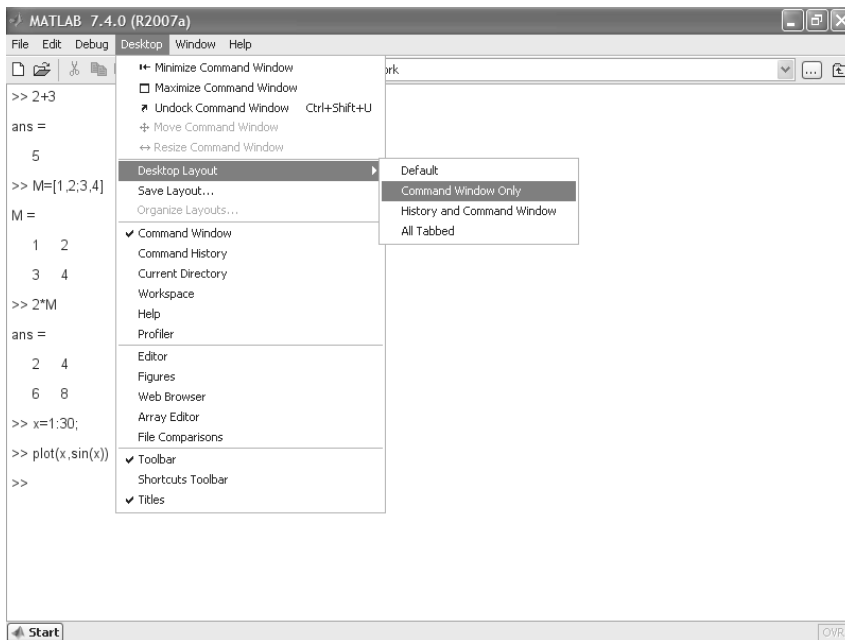


Рис. 2.50. Упрощенный интерфейс системы MATLAB 2007a

статуса. На рисунке показаны примеры простейших вычислений и открытый список команд позиции **Desktop** главного меню.

Сравнение рис. 2.50 с рис. 1.2 показывает, что различия в упрощенном интерфейсе между последними подверсиями MATLAB R2006/R2007 практически отсутствуют – разница видна лишь в названии, указанном в титульной строке.

2.9.3. Редактор/отладчик программ и файлов MATLAB R2007a

Как и в предшествующих версиях систем MATLAB, в системе MATLAB R2007a имеется простой и удобный редактор/отладчик программ и файлов. Его окно можно вывести в уменьшенном виде, как на рис. 2.49, так и в виде окна, выводимого вместе с окном командного режима, – рис. 2.51.

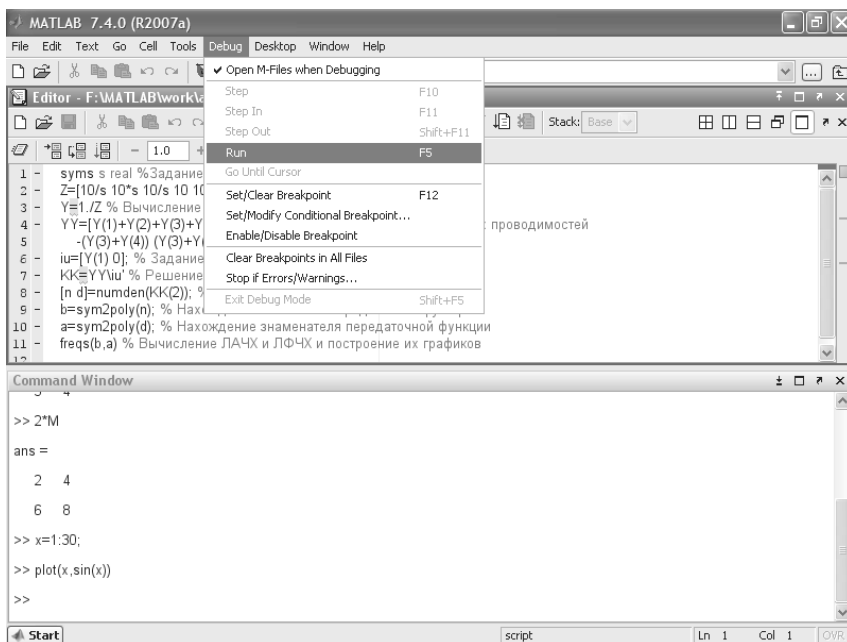


Рис. 2.51. Окна редактора/отладчика и командного режима работы MATLAB 2007a

На рис. 2.51 показан список команд позиции **Debug (Отладка)**, которые обычно используются в ходе отладки программ. Подробности работы с редактором/отладчиком программ описаны в уроке 11. Они практически не зависят от применяемой конкретной версии системы MATLAB.

2.9.4. Справка MATLAB R2007a

Вызов справки в MATLAB R2007a вполне обычный. На рис. 2.52 показано окно справки для раздела **Realize Note (Замечания по реализации)**. Этот раздел позволяет детально оценить новые возможности данной реализации MATLAB и системные требования к работе с ней.

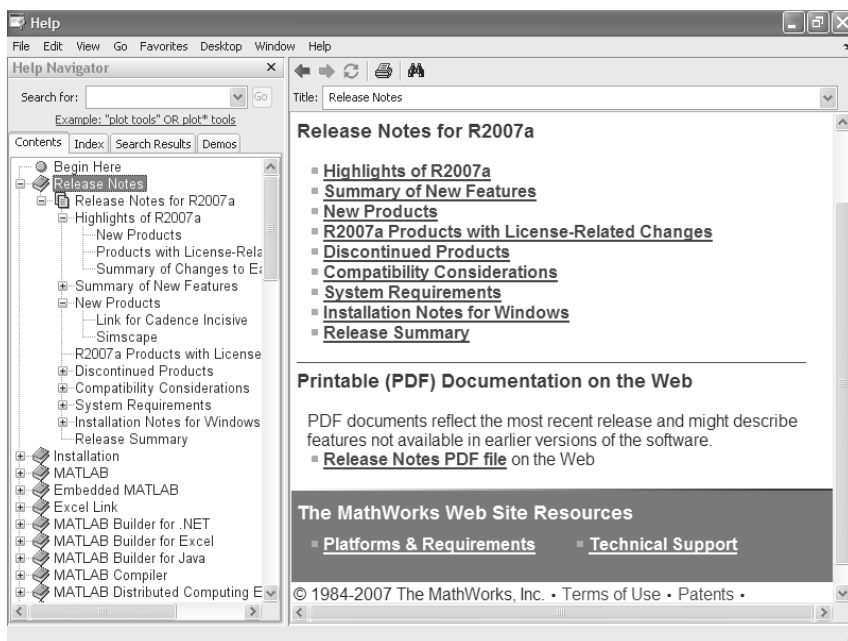


Рис. 2.52. Окно справки системы MATLAB 2007a по разделу Realize Note

2.9.5. Новые возможности MATLAB R2007a,b

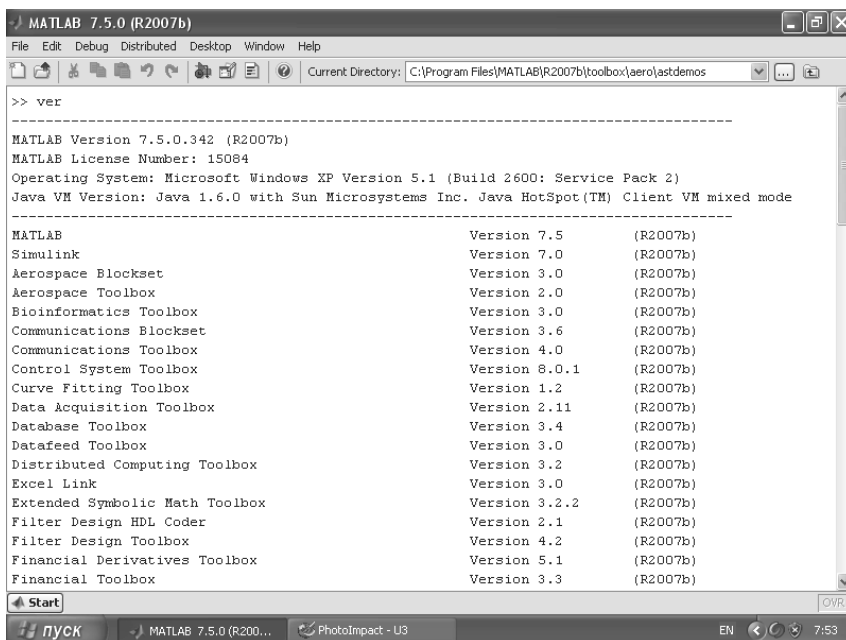
Из знакомства со справкой по данному разделу следует, что MATLAB R2007a,b обеспечивает следующие новые возможности:

- расширенный (до 82 пакетов) набор пакетов расширения, многие из которых существенно обновлены и модернизированы;
- поддержка многопоточных (Hyper Threading) вычислений в математических функциях;
- поддержка ядром системы многоядерных (multicore) микропроцессоров;
- выполнение до четырех параллельных алгоритмов;
- поддержка распределенных вычислений на рабочем столе;

- новая позиция меню **Distributed** в MATLAB R2007b, ориентированная в будущем на расширенную конфигурацию системы;
- ускорение за счет компиляции вычислений статистики, выполняемых в формате чисел с плавающей точкой;
- система управления ToolboxGeneration нелинейных моделей в расширении по идентификации систем;
- повышение скорости работы генетических алгоритмов и алгоритмов прямого поиска в пакете расширения по ним.

2.9.6. Интерфейс и справка MATLAB R2007b

Последней на момент подготовки данной книги версией системы MATLAB была версия MATLAB R2007b. Эта новейшая версия характеризуется прежде всего существенно обновленным набором пакетов расширения. Их список можно вывести командой `ver` в командном окне. На рис. 2.53 показан упрощенный интерфейс MATLAB R2007b с выводом только окна командного режима, в котором представлен результат выполнения команды `ver`. В окне представлен лишь список пакетов расширения, которые входят в данную версию. Существенно изменился номер версии главного пакета расширения системы MATLAB – Simulink (появилась версия Simulink 7).



```
MATLAB 7.5.0 (R2007b)
File Edit Debug Distributed Desktop Window Help
Current Directory: C:\Program Files\MATLAB\R2007b\toolbox\ aero\astdemos

>> ver

-----
MATLAB Version 7.5.0.342 (R2007b)
MATLAB License Number: 15084
Operating System: Microsoft Windows XP Version 5.1 (Build 2600: Service Pack 2)
Java VM Version: Java 1.6.0 with Sun Microsystems Inc. Java HotSpot(TM) Client VM mixed mode
-----
MATLAB                               Version 7.5      (R2007b)
Simulink                              Version 7.0      (R2007b)
Aerospace Blockset                    Version 3.0      (R2007b)
Aerospace Toolbox                     Version 2.0      (R2007b)
Bioinformatics Toolbox                 Version 3.0      (R2007b)
Communications Blockset                Version 3.6      (R2007b)
Communications Toolbox                 Version 4.0      (R2007b)
Control System Toolbox                 Version 8.0.1    (R2007b)
Curve Fitting Toolbox                  Version 1.2      (R2007b)
Data Acquisition Toolbox               Version 2.11     (R2007b)
Database Toolbox                       Version 3.4      (R2007b)
Datafeed Toolbox                       Version 3.0      (R2007b)
Distributed Computing Toolbox          Version 3.2      (R2007b)
Excel Link                             Version 3.0      (R2007b)
Extended Symbolic Math Toolbox         Version 3.2.2    (R2007b)
Filter Design HDL Coder                 Version 2.1      (R2007b)
Filter Design Toolbox                  Version 4.2      (R2007b)
Financial Derivatives Toolbox           Version 5.1      (R2007b)
Financial Toolbox                       Version 3.3      (R2007b)
```

Рис. 2.53. Упрощенный интерфейс системы MATLAB R2007b

Интерфейс MATLAB R2007b по умолчанию показан на рис. 2.54. Он практически не отличается от интерфейса предшествующей реализации – за исключением появления новой позиции меню **Distributed**. Оно пока имеет всего две позиции: **Select Configuration (Выбор конфигурации)** и **Manage Configuration...**. Первая позиция имеет единственную позицию **Local** – выбор локальной конфигурации, а вторая позиция выводит окно, которое показано на рис. 2.4 в правом верхнем углу окна командного режима.

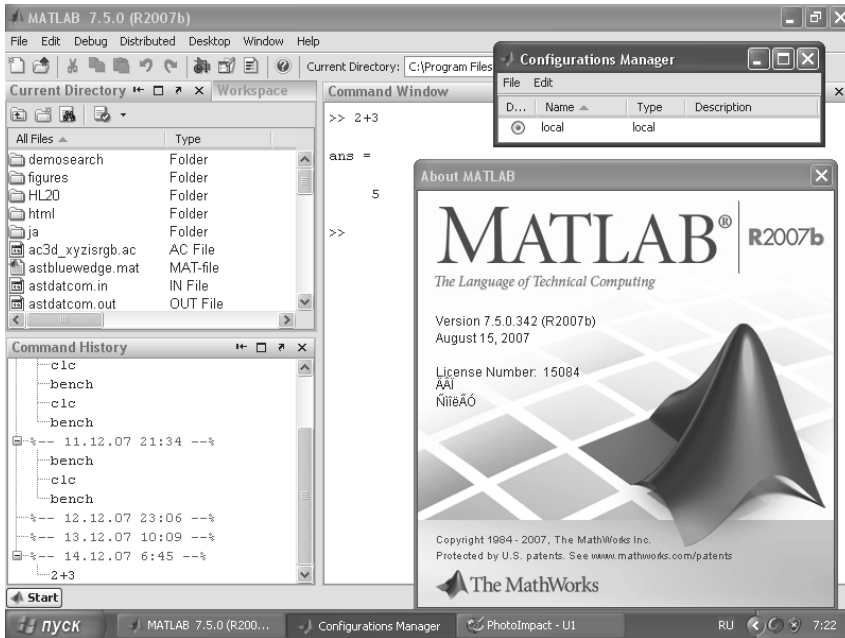


Рис. 2.54. Интерфейс системы MATLAB R2007b по умолчанию

Окно справки системы MATLAB R2007b показано на рис. 2.55. Принципы организации справки и доступа к ее разделам в новой версии системы не изменились. Однако загрузка разделов справки заметно ускорена, кроме того, она сопровождается появлением пиктограммы с изображением земного шара и логотипа MATLAB над ним. Под пиктограммой имеется надпись Loading.

2.9.7. Общая настройка MATLAB R2007b

Общая настройка систем MATLAB сосредоточена в окне предпочтений **Preferences**, которое открывается соответствующей командой в позиции **File** меню. На рис. 2.56 показано это окно для MATLAB R2007b. Сравнив это окно с аналогичным окном более ранней версии MATLAB (рис. 2.5), нетрудно заметить, что число установок в последней версии MATLAB заметно возросло, хотя вид окна сохранился.

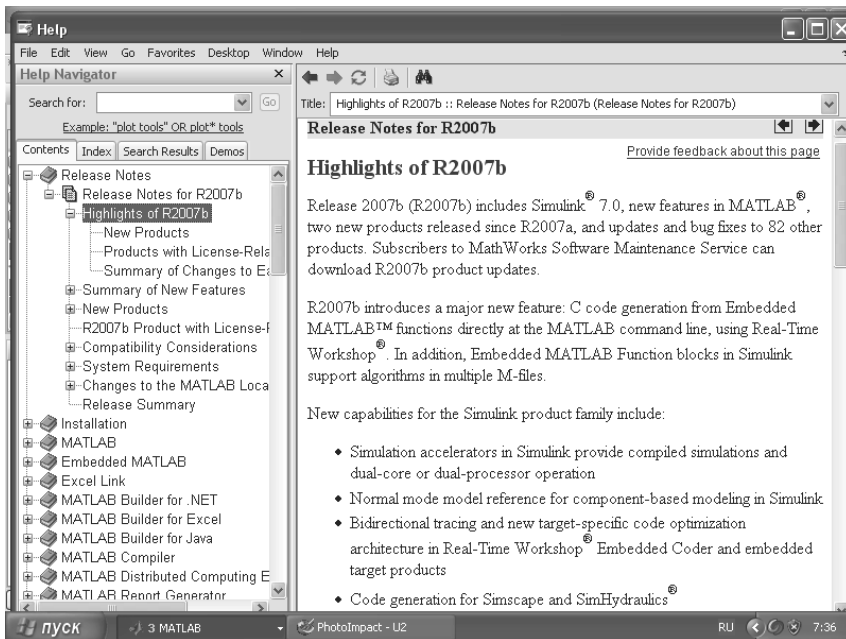


Рис. 2.55. Интерфейс справки системы MATLAB R2007b

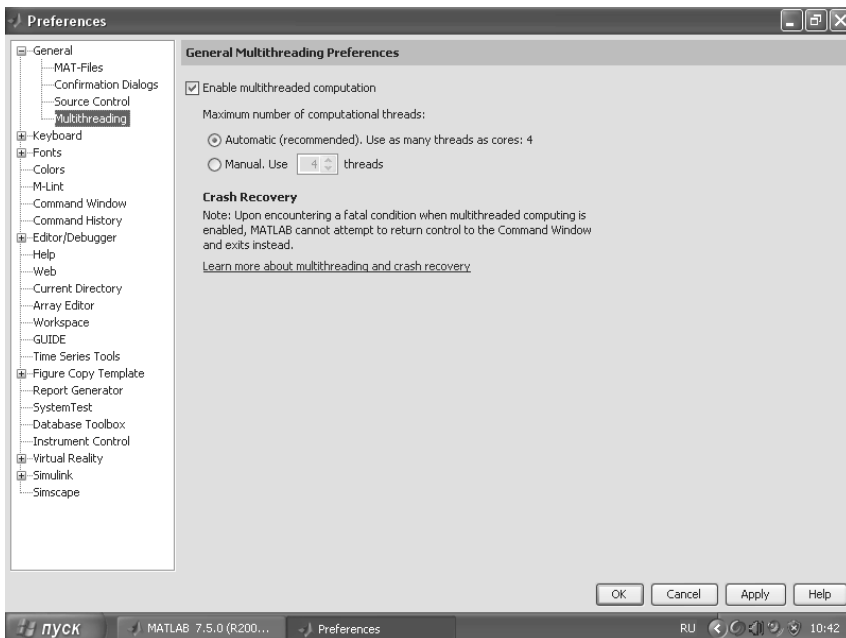


Рис. 2.56. Окно предпочтений системы MATLAB R2007b

Из новых установок отметим лишь одну – поддержку многих потоков в микропроцессорах (вкладка **Multithreading**). По умолчанию эта возможность отключена. При ее включении появляется возможность задать поддержку в автоматическом режиме или вручную до 4 потоков.

2.9.8. Производительность реализаций MATLAB R2007a,b

На рис. 2.57 показаны результаты тестирования компьютера на процессоре Pentium 4 HT 2,6 ГГц в среде системы MATLAB R2007a (MATLAB 7.4.0). Их можно трактовать двояко. С одной стороны, компьютер занял вполне достойное место – по производительности его обходят только ПК с двухъядерными процессорами (их число в тесте, кстати, заметно возросло). С другой – сравнение с результатами тестирования для предшествующей версии MATLAB 2006b (рис. 1.5) показывает, что скорость вычислений практически не изменилась.

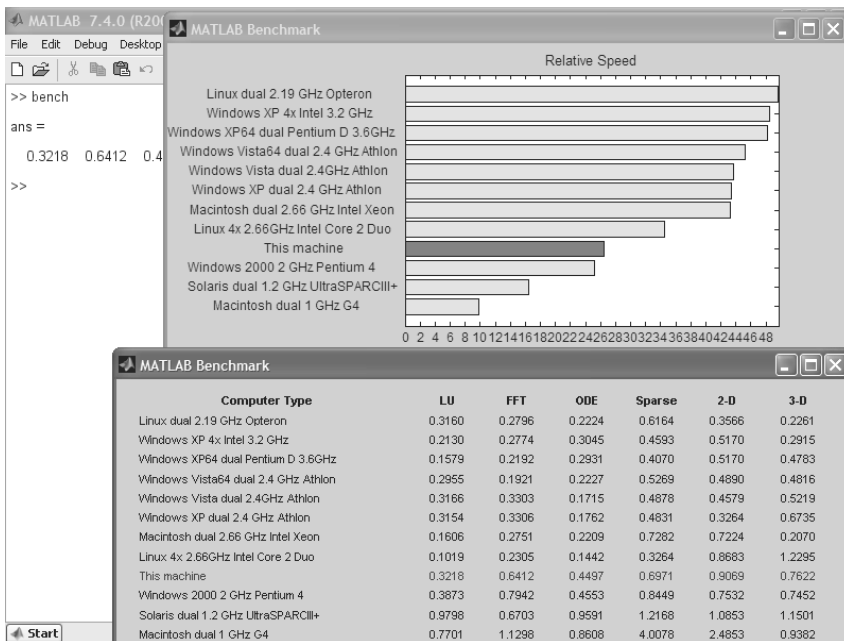


Рис. 2.57. Результаты выполнения теста на скорость вычислений для MATLAB 2007a

А что ждет пользователя, использующего новейшую версию MATLAB 2007b (MATLAB 7.5.0) при использовании новейших ПК с четырехъядерными процессорами? Ответы на эти вопросы дает рис. 2.58, на котором показано окно команд-

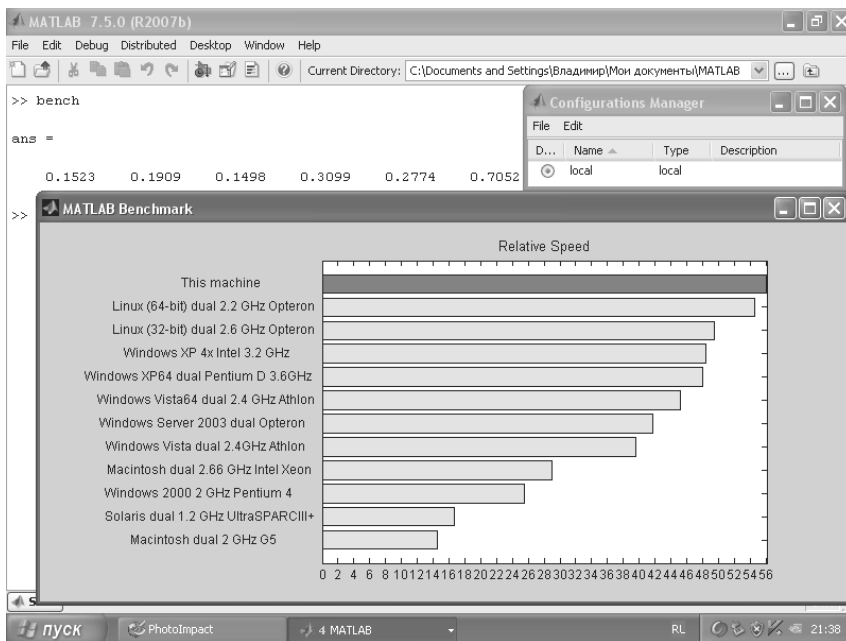


Рис. 2.58. Результаты выполнения теста на скорость вычислений для MATLAB 2007b при работе с четырехъядерным микропроцессором Core 2 Quad

ного режима MATLAB R2007b с результатами тестирования системы на скорость вычислений.

На этот раз ПК с четырехъядерным процессором показал самые высокие показатели среди ПК, отмеченных в тесте. Приведенные данные свидетельствуют, что наибольший выигрыш по скорости счета новейшие реализации MATLAB дают при применении новейших ПК с двухъ- и четырехъядерными процессорами. Они пока довольно дороги, но цены на них быстро падают. Видимо, учитывая высокую стоимость легальных программных продуктов MATLAB (десятки тысяч долларов для полных версий систем), не имеет смысла спешить с заменой установленных версий на более новые.

Исходя из этого, последующий материал дан так, что он вполне пригоден как для новейших версий MATLAB 2007a/ R2007b), так и для куда более распространенных предшествующих реализаций MATLAB R2006a,b и даже более ранних. В рамках учебной направленности данной книги детали различий разных версий MATLAB не являются принципиальными и подробно не обсуждаются.

Программные средства математических вычислений

| | |
|--|-----|
| 3.1. Вычислительные и логические операции | 152 |
| 3.2. Специальные символы | 156 |
| 3.3. Работа со специальными данными | 161 |
| 3.4. Встроенные элементарные функции | 168 |
| 3.5. Числовые функции | 180 |
| 3.6. Специальные математические функции | 182 |

Системы MATLAB являются проблемно ориентированной на математические вычисления программной средой. В этом уроке описаны наиболее важные программные средства для выполнения массовых математических расчетов самого общего характера в этой среде. Описание сопровождается множеством примеров, которые читателю рекомендуется повторить, используя командный режим работы и М-файлы примеров. Все описанные операции могут использоваться в составе программ на языке программирования системы MATLAB любой версии.

3.1. Вычислительные и логические операции

3.1.1. Арифметические матричные операторы и функции

Арифметические операторы задают выполнение арифметических операций. В MATLAB практически все операторы предназначены для выполнения операций над матрицами (табл. 3.1). Ввиду очевидности арифметических операторов их подробное описание опущено.

Таблица 3.1. Арифметические операторы и функции MATLAB

| Функция | Название | Оператор | Синтаксис |
|----------|---|----------|-----------|
| plus | Плюс | + | M1+M2 |
| uplus | Унарный плюс | + | +M |
| minus | Минус | - | M1-M2 |
| uminus | Унарный минус | - | -M |
| mtimes | Матричное умножение | * | M1*M2 |
| times | Позлементное умножение массивов | .* | A1.*A2 |
| mpower | Возведение матрицы в степень | ^ | M1^x |
| power | Позлементное возведение массива в степень | .^ | A1.^x |
| mldivide | Обратное (справа налево) деление матриц | \ | M1\M2 |
| mrdivide | Деление матриц слева направо | / | M1/M2 |
| ldivide | Позлементное деление массивов справа налево | .\ | A1.\A2 |
| rdivide | Позлементное деление массивов слева направо | ./ | A1./A2 |
| kron | Тензорное умножение Кронекера | kron | kron(X,Y) |

Обратите внимание на то, что каждый оператор имеет аналогичную по назначению функцию. Например, оператору матричного умножения * соответствует функция mtimes (M1, M2). Примеры применения арифметических операторов уже не раз приводились, так что ограничимся несколькими дополнительными примерами (здесь и далее пробел после ans при однострочном выводе опущен):

```
>> A=[1 2 3];
>> B=[4 5 6];
>> B-A
```

```

ans = 3           3           3
>> minus(B,A)
ans = 3           3           3
>> A.^2
ans = 1           4           9
>> power(A,2)
ans = 1           4           9
>> A.\B
ans = 4.0000      2.5000      2.0000
>> ldivide(A,B)
ans = 4.0000      2.5000      2.0000
>> rdivide(A,B)
ans = 0.2500      0.4000      0.5000

```

Соответствие функций операторам и командам в системе MATLAB является одним из основных положений программирования. Оно позволяет одновременно использовать элементы как операторного, так и функционального программирования.

Следует отметить, что в математических выражениях системы MATLAB операторы имеют определенный *приоритет исполнения*:

- 1) круглые скобки;
- 2) операции транспонирования и возведения в степень;
- 3) одноместные операции (унарные + и -, логическое отрицание ~);
- 4) арифметические операции умножения и деления;
- 5) арифметические операции сложения и вычитания;
- 6) оператор сечения массива ;;
- 7) операторы отношения;
- 8) логические операторы и т. д.

Для изменения приоритета операций в математических выражениях используются круглые скобки. Степень вложения скобок не ограничивается.

3.1.2. Операции отношения

Для выполнения операций отношения предназначены *операторы отношения*. Они служат для сравнения двух величин, векторов или матриц. Все операторы отношения имеют два операнда, например x и y , и записываются, как показано в табл. 3.2.

Таблица 3.2. Операторы и функции отношения

| Функция | Название | Оператор | Пример |
|---------|------------------|----------|--------|
| eq | Равно | == | $x==y$ |
| ne | Не равно | ~= | $x~=y$ |
| lt | Меньше чем | < | $x<y$ |
| gt | Больше чем | > | $x>y$ |
| le | Меньше или равно | <= | $x<=y$ |
| ge | Больше или равно | >= | $x>=y$ |

Данные операторы выполняют поэлементное сравнение векторов или матриц одинакового размера и возвращают значение 1 (True), если элементы идентичны, и значение 0 (False) в противном случае. Если операнды – действительные числа, то применение операторов отношения тривиально: например, `eq(2, 2)` дает 1, а `le(5, 3)` дает 0.

Следует отметить, что операторы `<`, `<=`, `>` и `>=` при комплексных операндах используются для сравнения только действительные части операндов – мнимые отбрасываются. В то же время операторы `==` и `~=` ведут сравнение с учетом как действительной, так и мнимой частей операндов, например:

```
>> (2+3i)>=(2+i)
ans = 1
>> (2+3i)>(2+i)
ans = 0
>> abs(2+3i)>abs(2+i)
ans = 1
>> (2+3i)==(2+i)
ans = 0
>> (2+3i)~=(2+i)
ans = 1
```

Если один из операндов – скаляр, происходит сравнение всех элементов второго операнда-массива со значением этого скаляра:

```
M =
     -1         0
      1         2
>> M>=0
ans =
         0         1
         1         1
```

В общем случае операторы отношения сравнивают два массива одного размера и выдают результат в виде массива того же размера:

```
>> M>[0 1; 1 0]
ans =
         0         0
         0         1
```

Таким образом, спектр применения операторов отношения в системе MATLAB шире, чем в обычных языках программирования, поскольку операндами являются не только числа, но и векторы, матрицы и массивы. Возможно применение операторов отношения и к символьным выражениям:

```
>> 'b'>'a'
ans = 1
>> 'abc'=='abc'
ans = 1         1         1
>> 'cba'<'abc'
ans = 0         0         1
```

В этом случае символы, входящие в выражения, представляются своими ASCII-кодами. Строки воспринимаются как векторы, содержащие значения кодов. Все это надо учитывать при использовании управляющих структур языка программирования, в которых широко применяются операторы отношения (см. урок 11).

3.1.3. Логические операции и операторы

Логические операторы и соответствующие им функции служат для реализации поэлементных логических операций над элементами одинаковых по размеру массивов (табл. 3.3). Они широко используются в управляющих структурах программ (см. урок 11) и составляют основу логического программирования.

Таблица 3.3. Логические операторы и функции MATLAB

| Функция | Название | Обозначение |
|---------|--|-------------|
| And | Логическое И (AND) | & |
| Or | Логическое ИЛИ (OR) | |
| Not | Логическое НЕ (NOT) | ~ |
| Xor | Исключающее ИЛИ (EXCLUSIVE OR) | |
| Any | Верно, если все элементы вектора равны нулю | |
| All | Верно, если все элементы вектора не равны нулю | |

Работа операторов поясняется приведенными ниже примерами:

```
>> A=[1 2 3];
>> B=[1 0 0];
>> and(A,B)
ans = 1      0      0
>> or(A,B)
ans = 1      1      1
>> A&B
ans = 1      0      0
>> A|B
ans = 1      1      1
>> not(A)
ans = 0      0      0
>> not(B)
ans = 0      1      1
>> ~B
ans = 0      1      1
>> xor(A,B)
ans = 0      1      1
>> any(A)
ans = 1
>> all([0 0 0])
ans = 0
>> all(B)
ans = 0
```



```
>> and('abc','012')
ans = 1      1      1
```

Обратите внимание, что аргументами логических операторов могут быть числа и строки. При аргументах-числах логический нуль соответствует числовому нулю, а любое отличное от нуля число воспринимается как логическая единица. Для строк действует уже отмеченное правило: каждый символ строки представляется своим ASCII-кодом.

В MATLAB возможны также укороченные логические операторы `&&` и `||` (укороченные И и ИЛИ). Эти операторы осуществляют укороченную проверку логических выражений. Например, в выражении `A&&B` если оператор `A` ложен, то проверка оператора `B` уже не осуществляется, поскольку результат тождественно ложен. Аналогично в `A||B` если оператор `A` истинен, то выражение не проверяется, поскольку оно заведомо истинное.

3.2. Специальные символы

3.2.1. Специальные символы

К классу операторов в системе MATLAB относятся также *специальные символы*. Они предназначены для создания самых разнообразных объектов входного языка и языка программирования системы и придания им различных форм. В табл. 3.4 представлено описание полного набора специальных символов.

Таблица 3.4. Специальные символы MATLAB

| Обозначение | Название | Категория |
|-------------|--|------------|
| : | Двоеточие | colon |
| () | Круглые скобки | paren |
| [] | Квадратные скобки | paren |
| { } | Фигурные скобки | paren |
| . | Десятичная точка | punct |
| . | Выделение поля структуры | punct |
| .. | Родительский каталог | punct |
| ... | Продолжение строки | punct |
| , | Разделитель | punct |
| ; | Точка с запятой | punct |
| % | Комментарий | punct |
| ! | Вызов команды операционной системы | punct |
| = | Присваивание | punct |
| ' | Кавычка | punct |
| .' | Транспонирование | transpose |
| ' | Транспонирование с комплексным сопряжением | ctranspose |
| [,] | Горизонтальная конкатенация | horzcat |
| [;] | Вертикальная конкатенация | vertcat |
| (), { }, . | Присваивание подмассива | subsasgn |
| (), { }, . | Ссылка на подмассив | subsref |
| b(a) | Индекс подмассива | subsindex |

Теперь рассмотрим их более подробно.

Символ `:` (двоеточие) – формирование подвекторов и подматриц из векторов и матриц.

Оператор `:` – один из наиболее часто используемых операторов в системе MATLAB. Он использует следующие правила для создания векторов:

- `j:k` – то же, что и `[j, j+1, ..., k]`;
- `j:k` – пустой вектор, если $j > k$;
- `j:i:k` – то же, что и `[j, j+i, j+2i, ..., k]`;
- `j:i:k` – пустой вектор, если $i > 0$ и $j > k$ или если $i < 0$ и $j < k$, где i, j и k – скалярные величины.

Ниже показано, как выбирать с помощью оператора `:` строки, столбцы и элементы из векторов, матриц и многомерных массивов:

- `A(:, j)` – это j -й столбец из A ;
- `A(i, :)` – это i -я строка из A ;
- `A(:, :)` – эквивалент двумерного массива (для матриц это аналогично A);
- `A(j:k)` – это $A(j), A(j+1), \dots, A(k)$;
- `A(:, j:k)` – это $A(:, j), A(:, j+1), \dots, A(:, k)$;
- `A(:, :, k)` – это k -ая страница трехмерного массива A ;
- `A(i, j, k, :)` – вектор, выделенный из четырехмерного массива A . Вектор включает элементы $A(i, j, k, 1), A(i, j, k, 2), A(i, j, k, 3)$ и т. д.;
- `A(:)` – записывает все элементы массива A в виде столбца.

Символы `()` (круглые скобки) используются для задания порядка выполнения операций в арифметических выражениях, указания последовательности аргументов функции и указания индексов элемента вектора или матрицы. Если X и V – векторы, то $X(V)$ можно представить как $[X(V(1)), X(V(2)), \dots, X(V(n))]$. Элементы вектора V должны быть целыми числами, чтобы их можно было использовать как индексы элементов массива X . Ошибка генерируется в том случае, если индекс элемента меньше единицы или больше чем `size(X)`. Такой же принцип индексирования действителен и для матриц. Если вектор V имеет m компонентов, а вектор W – n компонентов, то `A(V, W)` будет матрицей размера $m \times n$, сформированной из элементов матрицы A , индексы которой – элементы векторов V и W .

Символы `[]` (квадратные скобки) используются для формирования векторов и матриц:

- `[6.9 9.64 sqrt(-1)]` – вектор, содержащий три элемента, разделенных пробелами;
- `[6.9, 9.64, i]` – такой же вектор;
- `[1+j 2-j 3]` и `[1+j 2-j 3]` – разные векторы: первый содержит три элемента, а второй – пять;
- `[11 12 13; 21 22 23]` – матрица размера 2×3 . Точка с запятой разделяет первую и вторую строки.

Еще несколько примеров:

- `A = []` – создает пустую матрицу A ;
- `A(m, :) = []` – удаляет строку m из матрицы A ;

- $A(:, n) = []$ удаляет столбец n из матрицы A .

Символы $\{ \}$ (фигурные скобки) используются для формирования массивов ячеек. Например, `{magic(3) 6.9 'hello'}` – массив ячеек с тремя элементами.

Символ `.` (десятичная точка) используется для отделения дробной части чисел от целой. Например, `314/100`, `3.14` и `.314e1` – одно и то же число.

Кроме того, символ точки `.` используется для выделения полей структур. Например, `A.(field)` и `A(i).field`, где A – структура, означает выделение поля структуры с именем «field».

Ниже перечислено назначение остальных специальных символов MATLAB:

- `..` (родительский каталог) – переход по дереву каталогов на один уровень вверх;
- `...` (продолжение) – три или более точек в конце строки указывают на продолжение строки;
- `,` (запятая) – используется для разделения индексов элементов матрицы и аргументов функции, а также для разделения операторов языка MATLAB. При разделении операторов в строке запятая может заменяться на точку с запятой с целью запрета вывода на экран результата вычислений;
- `;` (точка с запятой) – используется внутри круглых скобок для разделения строк матриц, а также в конце операторов для запрета вывода на экран результата вычислений;
- `%` (знак процента) – используется для указания логического конца строки. Текст, находящийся после знака процента, воспринимается как комментарий и игнорируется (увы, за исключением русскоязычных комментариев, которые нередко ведут к ошибочным командам);
- `!` (восклицательный знак) – является указателем ввода команды операционной системы. Строка, следующая за ним, воспринимается как команда операционной системы;
- `=` (знак равенства) – используется для присваивания значений в арифметических выражениях;
- `'` (одиночная кавычка, апостроф) – текст в кавычках представляется как вектор символов с компонентами, являющимися ASCII-кодами символов. Кавычка внутри строки задается двумя кавычками. Например:

```
>> a='Hello''my friend'
a = Hello'my friend
```

- `'` (транспонирование с комплексным сопряжением) – транспонирование матриц и векторов, например `A'` – транспонированная матрица A . Транспонирование вектора-столбца превращает его в вектор-столбец, и наоборот. Для комплексных матриц транспонирование дополняется комплексным сопряжением. Строки транспонированной матрицы соответствуют столбцам исходной матрицы;
- `.'` (транспонирование) – транспонирование массива, например `A.'` – транспонированный массив A . Для комплексных массивов операция сопряжения не выполняется;

- `[,]` – горизонтальная конкатенация. Так, `[A, B]` – горизонтальная конкатенация (объединение) матриц `A` и `B`. `A` и `B` должны иметь одинаковое количество *строк*. `[A B]` действует аналогично. Горизонтальная конкатенация может быть применена для любого числа матриц в пределах одних скобок: `[A, B, C]`. Горизонтальная и вертикальная конкатенации могут использоваться одновременно: `[A, B; C]`;
- `[;]` – вертикальная конкатенация. Так, `[A; B]` – вертикальная конкатенация (объединение) матриц `A` и `B`. `A` и `B` должны иметь одинаковое число *столбцов*. Вертикальная конкатенация может быть применена для любого числа матриц в пределах одних скобок: `[A; B; C]`. Горизонтальная и вертикальная конкатенации могут использоваться одновременно: `[A; B, C]`;
- `()`, `{ }` – присваивание подмассива. Приведем несколько примеров:
 - `A(I)=B` – присваивает значения элементов массива `B` элементам массива `A`, которые определяются вектором индексов `I`. Массив `B` должен иметь такую же размерность, как и массив `I`, или может быть скаляром;
 - `A(I, J)=B` – присваивает значения массива `B` элементам прямоугольной подматрицы `A`, которые определяются векторами индексов `I` и `J`. Массив `B` должен иметь `LENGTH(I)` строк и `LENGTH(J)` столбцов;
 - `A{I}=B`, где `A` – массив ячеек и `I` – скаляр, помещает копию массива `B` в заданную ячейку массива `A`. Если `I` имеет более одного элемента, то появляется сообщение об ошибке.

3.2.2. Системные переменные и константы

Как отмечалось ранее, в состав объектов MATLAB входит ряд системных переменных и констант, значения которых устанавливаются системой при ее загрузке или автоматически формируются в процессе вычислений:

- `ans` – результат выполнения последней операции. Переменная `ans` создается автоматически, когда не определены выходные аргументы какого-либо оператора. Примеры неоднократно приводились выше;
- `computer` – возвращает строку с информацией о типе компьютера, на котором установлена система MATLAB;
- `[str,maxsize] = computer` – возвращает строку `str` с информацией о компьютере и целое число `maxsize`, содержащее максимально допустимое число элементов матрицы для данной версии MATLAB. Пример:

```
>> [str,maxsize] = computer
str = PCWIN
maxsize = 268435455
```

- `eps` – возвращает интервал между числом `1.0` и следующим ближайшим числом с плавающей запятой, которое воспринимается как отличное от `1.0`. Значение `eps` определяет заданный по умолчанию порог для функций `pinv` и `rank`, а также для некоторых других функций. На машинах с арифметикой с плавающей запятой `eps = 2-52`, что приблизительно составляет `2.22e-16`. Пример:

```
>> eps
ans = 2.2204e-016
```

- i или j мнимая единица (равная $\sqrt{-1}$), которая используется для задания мнимой части комплексных чисел. Символ i при задании комплексной константы можно использовать без знака умножения. В качестве мнимой единицы можно также использовать символ j . Пример:

```
>> w=3+5i
w = 3.0000 + 5.0000i
```

Примечание

Переменным i и j можно задать и иное значение, например они могут выступать в качестве индексов в циклах `for`. Однако это чревато путаницей, если внутри цикла пользователь задает выражения с комплексными числами. Используйте как индексы I и J вместо i и j .

- `Inf` – возвращает представление положительной бесконечности для машинной арифметики. Бесконечность следует из операций, подобных делению на нуль, и переполнения, которое ведет к результатам, слишком большим, чтобы их можно было представить в виде числа с плавающей запятой.

Пример:

```
>> 4/0
Warning: Divide by zero.
ans = Inf
```

- j – мнимая единица. Символ j можно использовать в качестве мнимой единицы наряду с i . Как мнимая единица (равная $\sqrt{-1}$) символ j используется для задания мнимой части комплексных чисел. Все сказанное о символе i относится и к j . Пример:

```
>> s=4-3j
s = 4.0000 - 3.0000i
```

- `NaN` – возвращает представление для нечисловых величин, например в случае операций, которые имеют неопределенные численные результаты. Пример:

```
>> s=0/0
Warning: Divide by zero.
s = NaN
```

- `pi` – число π (отношение длины окружности к ее диаметру). `pi` возвращает число с плавающей запятой, ближайшее к значению π . Выражения $4*\text{atan}(1)$ и $\text{imag}(\log(-1))$ выдают тот же результат. Пример:

```
>> pi
ans = 3.1416
```

- `realmax` – возвращает самое большое число в формате с плавающей запятой, соответствующее конкретному компьютеру. Большее значение соответствует системной переменной `Inf`. Пример:

```
>> n = realmax
n =      1.7977e+308
```

- `realmin` – возвращает наименьшее нормализованное положительное число в формате с плавающей запятой, представимое на конкретном компьютере. Любое меньшее число воспринимается как нуль. Пример:

```
>> n = realmin
n =      2.2251e-308
```

Переменные `varargin` и `varargout` позволяют использовать в функциях переменное число входных и выходных параметров:

- `varargout = foo(n)` – возвращает список выходных параметров переменной длины функции `foo`;
- `y = functionbar(varargin)` – принимает переменное число аргументов в функцию `bar`.

Переменные `varargin` и `varargout` используются только внутри `m`-файлов функции для задания произвольных аргументов функции. Эти переменные должны быть последними в списке входов или выходов, а для их обозначения могут использоваться только строчные буквы. Использование этих возможностей мы рассмотрим более подробно в уроке 11.

3.3. Работа со специальными данными

3.3.1. Поразрядная обработка данных

Ряд функций предназначен для поразрядной логической обработки данных:

- `bitand(A, B)` – возвращает поразрядное И двух неотрицательных целых аргументов `A` и `B`. Пример:

```
>> f=bitand(7,14)
f =      6
```

- `bitcmp(A, n)` – возвращает поразрядное дополнение аргумента `A` как `n`-битовое неотрицательное целое число. Пример:

```
>> g=bitcmp(6,4)
g =      9
```

- `bitor(A, B)` – возвращает поразрядное ИЛИ двух неотрицательных целых аргументов `A` и `B`. Пример:

```
>> v=bitor(12,21)
v =      29
```

- `bitmax` – возвращает максимальное целое число без знака, которое может быть представлено в формате чисел с плавающей запятой применительно к используемому компьютеру. Это значение определяется для комбинации, когда все биты установлены. На машинах с IEEE-арифметикой это значение равно $2^{53} - 1$. Пример:

```
>> bitmax
```

```
ans = 9.0072e+015
```

- `bitset(A, bit)` – устанавливает бит в позиции `bit` аргумента `A` в единичное значение. Аргумент `A` должен быть неотрицательным целым. `bit` – это номер в диапазоне между 1 и числом бит в целом числе, представленном в формате чисел с плавающей запятой;
- `bitset(A, bit, v)` – устанавливает бит в позиции `bit` равным значению `v`, которое должно быть 0 или 1. Пример:

```
>> d=bitset(12,2,1)
d =      14
```

- `bitshift(A, n)` – возвращает значение аргумента `A`, сдвинутое на `n` бит. Если `n>0`, это аналогично умножению на 2^n (левый сдвиг). Если `n<0`, это аналогично делению на 2^n (правый сдвиг). Пример:

```
>> f=bitshift(4,3)
f =      32
```

- `bitget(A, bit)` – возвращает значение бита в позиции `bit` операнда `A`. Аргумент `A` должен быть неотрицательным целым числом. `bit` – это номер между 1 и числом бит в целом числе формата с плавающей запятой. Пример:

```
>> disp(dec2bin(23))
10111
>> C = bitget(23,5:-1:1)
C =      1      0      1      1      1
```

- `bitxor(A, B)` – возвращает результат поразрядного исключающего ИЛИ для двух аргументов `A` и `B`. Оба аргумента должны быть целыми. Пример:

```
>> x=bitxor(12,31)
x =      19
```

Чтобы операнды этих функций гарантированно были целыми числами, при их задании рекомендуется использовать функции `ceil`, `fix`, `floor` и `round`.

3.3.2. Обработка множеств

Множество – первичное понятие математики, не имеющее четкого определения. Под множеством подразумевается совокупность некоторых объектов, например книг в библиотеке, людей в зале или элементов вектора. В этом разделе приводятся некоторые функции для обработки множеств, представленных векторами. Они широко используются при анализе и обработке данных.

- `intersect(a, b)` – возвращает пересечение множеств для двух векторов `a` и `b`, то есть общие элементы векторов `a` и `b`. Результирующий вектор отсортирован по возрастанию. Если входные массивы не являются векторами, то они рассматриваются как совокупность векторов-столбцов `a=a(:)` или `b=b(:)`;
- `intersect(a, b, 'rows')` – возвращает строки, общие для `a` и `b`, когда `a` и `b` представляют собой матрицы с одинаковым числом столбцов;

- `[c, ia, ib] = intersect(a,b)` – также возвращает вектор-столбец индексов `ia` и `ib`, но так, что `c = a(ia)` и `c = b(ib)` (или `c = a(ia, :)` и `c = b(ib, :)`).

Пример:

```
>> A = [1 7 2 6]; B = [7 2 3 4 6 1];
>> [c, ia, ib] = intersect(A,B)
c =      1      2      6      7
ia =     1      3      4      2
ib =     6      2      5      1
```

- `ismember(a, S)` – возвращает вектор той же длины, что и исходный `a`, содержащий логические единицы на месте тех элементов вектора `a`, которые принадлежат множеству `S`, и логические нули на месте тех элементов вектора `a`, которые не принадлежат множеству `S`;
- `ismember(A, S, 'rows')` – возвращает вектор, содержащий логические единицы там, где строки матрицы `A` являются также строками матрицы `S`, и логические нули в остальных позициях. `A` и `S` должны быть матрицами с одинаковым числом столбцов.

Пример:

```
>> set = [0 1 3 5 7 9 11 15 17 19];
>> a=[1 2 3 4 5 6 7 8];
>> k = ismember(a, set)
k =      1      0      1      0      1      0      1      0
```

- `setdiff(a,b)` – возвращает разность множеств, то есть те элементы вектора `a`, которые не содержатся в векторе `b`. Результирующий вектор сортируется по возрастанию;
- `setdiff(a,b, 'rows')` – возвращает те строки из матрицы `a`, которые не содержатся в матрице `b`. Матрицы `a` и `b` должны иметь одинаковое число столбцов;
- `[c, i] = setdiff(...)` – возвращает также вектор индексов `i`, такой, что `c = a(i)` или `c = a(i, :)`.

Если входной массив `a` является матрицей, то он расценивается как вектор-столбец `a(:)`.

Пример:

```
>> a=[2 3 5 7 8 9 10 13 20];
>> b=[1 4 5 6 8 9 4]
>> c = setdiff(a,b)
c =      2      3      7      10      13      20
```

- `setxor(a,b)` – исключающее ИЛИ для векторов `a` и `b`. Результирующий вектор отсортирован;
- `setxor(a,b, 'rows')` – возвращает строки, которые не являются пересечениями матриц `a` и `b`. Матрицы `a` и `b` должны иметь одинаковое число столбцов;
- `[c, ia, ib] = setxor(...)` – возвращает также векторы индексов `ia` и `ib` так, что `c` является отсортированной комбинацией элементов `c = a(ia)` и `c = b(ib)` или для комбинаций строк `c = a(ia, :)` и `c = b(ib, :)`.

Если входной массив a является матрицей, то он расценивается как вектор-столбец $a(:)$.

Пример:

```
>> a = [-1 0 1 Inf -Inf NaN];
>> b = [-2 pi 0 Inf];
>> c = setxor(a,b)
c =      -Inf      -2.0000      -1.0000      1.0000      3.1416      NaN
```

- `union(a,b)` – возвращает вектор объединенных значений из a и b без повторяющихся элементов. Результирующий вектор сортируется в порядке возрастания;
- `union(a,b,'rows')` – возвращает объединенные строки из a и b , не содержащие повторов (а и b – это матрицы с одинаковым числом столбцов);
- `[c,ia,ib] = union(...)` – дополнительно возвращает ia и ib – векторы индексов, такие, что $c = a(ia)$ и $c = b(ib)$ или, для объединенных строк, $c = a(ia,:)$ и $c = b(ib,:)$.

Невекторный массив a расценивается как вектор-столбец $a(:)$.

Пример:

```
>> a=[2,4,-4,9,0];b=[2,5,4];
>> [c,ia,ib]=union(a,b)
c =      -4      0      2      4      5      9
ia =      3      5      4
ib =      1      3      2
```

- `unique(a)` – возвращает значения элементов из a , не содержащие повторов. Результирующий вектор сортируется в порядке возрастания. Невекторный массив расценивается как вектор-столбец $a = a(:)$;
- `unique(a,'rows')` – возвращает уникальные строки a ;
- `[b,i,j] = unique(...)` – дополнительно возвращает i и j – векторы индексов, такие, что $b = a(i)$ и $a = b(j)$ (или $b = a(i,:)$ и $a = b(j,:)$).

Примеры:

```
>> b=[-2,3,5,4,1,-6,2,2,7]
b =      -2      3      5      4      1      -6      2      2      7
>> [c,i,j]=unique(b)
c =      -6      -2      1      2      3      4      5      7
i =      6      1      5      8      2      4      3      9
j =      2      5      7      6      3      1      4      4      8
>> a=[-2,3,5;4,1,-6;2,2,7;-2,3,5]
a =
      -2      3      5
      4      1      -6
      2      2      7
      -2      3      5
>> c=unique(a,'rows')
c =
      -2      3      5
```

```

2      2      7
4      1     -6

```

Примеры графической визуализации операций с множествами можно найти в [54].

3.3.3. Работа с функциями времени и даты

Ряд функций служит для возврата текущего времени и даты. Они перечислены ниже.

- `calendar` – возвращает матрицу размером 6×7 , содержащую календарь на текущий месяц. Календарь начинается с воскресенья (первый столбец) и завершается субботой;
- `calendar(d)` – возвращает календарь на месяц, в который попадает день, заданный аргументом `d` (дни отсчитываются от начала летоисчисления);
- `calendar(y, m)` – возвращает календарь на месяц, заданный аргументом `m`, и год, заданный аргументом `y`.

Вызов функции без присваивания результата выдает календарь на экран.

Примеры:

```

>> calendar
              Jul 2005
  S      M      Tu      W      Th      F      S
  0      0      0      0      0      1      2
  3      4      5      6      7      8      9
 10     11     12     13     14     15     16
 17     18     19     20     21     22     23
 24     25     26     27     28     29     30
 31      0      0      0      0      0      0

```

```

>> calendar(700477)
              Nov 1917
  S      M      Tu      W      Th      F      S
  0      0      0      0      1      2      3
  4      5      6      7      8      9     10
 11     12     13     14     15     16     17
 18     19     20     21     22     23     24
 25     26     27     28     29     30      0
  0      0      0      0      0      0      0

```

- `clock` – возвращает вектор из 6 элементов, содержащий текущую дату и время в десятичной форме [год месяц день час минуты секунды]. Первые пять элементов этого вектора – целые числа. Шестой элемент имеет несколько десятичных знаков после запятой. Функция `fix(clock)` округляет число секунд до целого значения. Пример:

```

>> c=clock
c =      1.0e+003 *
      2.0000   0.0070   0.0240   0.0200   0.0120   0.0148
>> fix(clock)

```

```
ans = 2000          7          24          20          12          26
```

- `cputime` – возвращает время работы процессора (в секундах), использованное системой MATLAB с момента ее запуска. Это число может выйти за рамки внутреннего представления, и тогда отсчет времени начинается заново. Пример:

```
>> +t1=cputime; w=surf(peaks(30));cputime-t1
ans = 0.2200
```

- `str = date` – возвращает строку, содержащую дату в формате дд-ммм-гггг (день-месяц-год). Пример:

```
>> d = date
d = 15-Jul-2005
```

- `datenum` – преобразует строку даты в порядковый номер даты, который отсчитывается с некоторого начального дня (01.01.00);
- `datenum(str)` – преобразует дату, заданную строкой `str`, в порядковый номер даты. Строка `string` должна иметь один из следующих форматов: 0, 1, 2, 6, 13, 14, 15 или 16, определенных для функции `datestr`;
- `datenum(Y, M, D)` – возвращает порядковый номер даты для соответствующих массивов элементов `Y`, `M` и `D` (год, месяц, день). Массивы `Y`, `M` и `D` должны иметь одинаковую размерность (при этом любые из них могут быть скалярами);
- `datenum(Y, M, D, H, MI, S)` – возвращает порядковый номер даты для соответствующих массивов элементов `Y`, `M`, `D`, `H`, `MI` и `S` (год, месяц, день, часы, минуты, секунды). Массивы `Y`, `M`, `D`, `H`, `MI` и `S` должны иметь одинаковую размерность (при этом любые из них могут быть скалярами).

Пример:

```
>> n1 = datenum('26-Nov-1998')
n1 = 730085
>> Y=[1998,2000];M=[1,12];D=23;N=datenum(Y,M,D)
N = 729778          730843
```

- `datestr(D, dateform)` – преобразует каждый элемент массива порядковых номеров даты `D` в строку. Аргумент `dateform` определяет формат результата; `dateform` может быть номером или строкой в соответствии с табл. 3.5.

Таблица 3.5. Форматы представления даты

| Dateform (номер) | Dateform (строка) | Пример |
|------------------|------------------------|-------------------|
| 0 | 'dd-mmM-yyyy HH:MM:SS' | 01-Mar-1995 03:45 |
| 1 | 'dd-mmM-yyyy' | 01-Mar-1995 |
| 2 | 'mm/dd/yy' | 03/01/95 |
| 3 | 'mmm' | Mar |
| 4 | 'm' | M |
| 5 | 'mm' | 3 |
| 6 | 'mm/dd' | 03/01 |
| 7 | 'dd' | 1 |

Таблица 3.5. Форматы представления даты (продолжение)

| Dateform (номер) | Dateform (строка) | Пример |
|------------------|-------------------|----------|
| 8 | 'ddd' | Wed |
| 9 | 'd' | W |
| 10 | 'yyyy' | 1995 |
| 11 | 'yy' | 95 |
| 12 | 'mmmyy' | Mar95 |
| 13 | 'HH:MM:SS' | 15:45:17 |

- `datevec(A)` – преобразует входные величины в массив размерности $n \times 6$, каждая строка которого представляет собой вектор $[Y, M, D, H, MI, S]$. Первые пять элементов вектора – целые числа. Массив A может состоять или из строк, удовлетворяющих формату функции `datestr`, или из скалярных величин, созданных функциями `datenum` и `now`;

- $[Y, M, D, H, MI, S] = \text{datevec}(A)$ – возвращает компоненты вектора даты как индивидуальные переменные.

Любой компонент входного вектора, который не вписывается в нормальный диапазон дат, преобразуется в следующий диапазон (так, например, несуществующая дата June 31 преобразуется в July 1). Допускаются значения нулевого месяца и нулевого дня. Например:

```
>> n = datevec('11/31/98')
n =    1998    12     1     0     0     0
>> n = datevec(710223)
n =    1944     7    10     0     0
```

- `eomday(Y, M)` – возвращает последний день года и месяца, заданных соответственно элементами массивов Y и M . Пример (нахождение високосных лет XX столетия):

```
>> y = 1900:1999;
>> E = eomday(y, 2*ones(length(y), 1));
>> y(find(E==29))
ans =
  Columns 1 through 6
    1904    1908    1912    1916    1920    1924
  Columns 7 through 12
    1928    1932    1936    1940    1944    1948
  Columns 13 through 18
    1952    1956    1960    1964    1968    1972
  Columns 19 through 24
    1976    1980    1984    1988    1992    1996
```

- `etime(t2, t1)` – возвращает длительность промежутка времени (в секундах), задаваемого векторами $t1$ и $t2$. Векторы должны удовлетворять формату, выдаваемому функцией `clock`:

```
T = [год месяц день час минуты секунды].
```

Функция работает некорректно, если в текущий промежуток времени попадут границы месяца или года, что, однако, случается крайне редко и исправ-

ляется при повторе операции. Пример (вычисляется время, затрачиваемое на быстрое преобразование Фурье с 2048 точками):

```
>> x = rand(2048,1); t = clock; fft(x); etime(clock,t); etime
(clock,t)
ans = 0.0500
```

- `now` – возвращает текущие время и дату в форме числа. Использование `rem(now, 1)` возвращает только время, а `floor(now)` – только дату. Пример:

```
>> t1 = now, t2 = rem(now,1)
t1 = 7.3009e+005
t2 = 0.6455
```

- `tic` – запускает таймер;
- `toc` – выводит время, прошедшее с момента запуска таймера;
- `t = toc` – возвращает прошедшее время в переменной `t`. Пример:

```
>> tic, surf(peaks(50)); toc
elapsed_time = 0.7600
```

- `[N, S] = weekday(D)` – возвращает день недели в виде числа `N` и в виде строки `S` для каждой даты массива `D`. Пример:

```
>> D=[728647,735730]; [N,S] = weekday(D)
N =      2          1
S = Mon          Sun
```

3.4. Встроенные элементарные функции

Элементарные функции – пожалуй, наиболее известный класс математических функций [50–54]. Функции, перечисленные ниже, сгруппированы по функциональному назначению и являются встроенными в ядро MATLAB функциями языка программирования системы. В тригонометрических функциях углы измеряются в радианах. Все функции могут использоваться с синтаксисом вида $y = \text{func}(x)$, где `func` – имя функции. Мы, однако, будем использовать более простую форму `func(x)`. Форма `[y, z, ...] = func(x, ...)` будет использоваться только в тех случаях, когда функция возвращает значения ряда переменных.

3.4.1. Алгебраические и арифметические функции

В системе MATLAB определены следующие алгебраические и арифметические функции:

- `abs(X)` – возвращает абсолютную величину для каждого числового элемента вектора `X`. Если `X` содержит комплексные числа, `abs(X)` вычисляет модуль каждого числа. Примеры:

```
abs(-5) = 5
abs(3+4i) = 5
>> abs([1 -2 i 3i 2+3i])
ans =
  1.0000 2.0000 1.0000 3.0000 3.6056
```

- `exp(X)` – возвращает экспоненту для каждого элемента X. Для комплексного числа $z = x + i*y$ функция `exp(z)` вычисляет комплексную экспоненту: $\exp(z) = \exp(x) * (\cos(y) + i*\sin(y))$. Примеры:

```
>> exp([1 2 3])
ans = 2.7183 7.3891 20.0855
>> exp(2+3i)
ans = -7.3151 + 1.0427i
```

- `factor(n)` – возвращает вектор-строку, содержащую простые множители числа n. Для массивов эта функция неприменима. Пример:

```
f = factor(221)
f = 13 17
```

- `G=gcd(A, B)` – возвращает массив, содержащий наибольшие общие делители соответствующих элементов массивов целых чисел A и B. Функция `gcd(0, 0)` возвращает значение 0, в остальных случаях возвращаемый массив G содержит положительные целые числа;

- `[G, C, D] = gcd(A, B)` – возвращает массив наибольших общих делителей G и массивов C и D, которые удовлетворяют уравнению $A(i) .* C(i) + B(i) .* D(i) = G(i)$. Они полезны для выполнения элементарных эрмитовых преобразований. Примеры:

```
>> A=[2 6 9];
>> B=[2 3 3];
>> gcd(A,B)
ans = 2 3 3
>> [G,C,D]=gcd(A,B)
G = 2 3 3
C = 0 0 0
D = 1 1 1
```

- `lcm(A, B)` – возвращает наименьшие общие кратные для соответствующих парных элементов массивов A и B. Массивы A и B должны содержать положительные целые числа и иметь одинаковую размерность (любой из них может быть скаляром). Пример:

```
>> A=[1 3 5 4];
>> B=[2 4 6 2];
>> lcm(A,B)
ans = 2 12 30 4
```

- `log(X)` – возвращает натуральный логарифм элементов массива X. Для комплексного или отрицательного z, где $z = x + y*i$, вычисляется комплексный логарифм в виде $\log(z) = \log(\text{abs}(z)) + i*\text{atan2}(y, x)$. Функция логарифма вычисляется для каждого элемента массива. Область опре-

деления функции включает комплексные и отрицательные числа, что способно привести к непредвиденным результатам при некорректном использовании. Пример:

```
>> x=[1.2 3.34 5 2.3];
>> log(X)
ans = 0.1823 1.2060 1.6094 0.8329
```

- $\log_2(X)$ – возвращает логарифм по основанию 2 элементов массива X;
- $[F, E] = \log_2(X)$ – возвращает массив действительных значений F и массив целых чисел E. Элементы массива F обычно лежат в диапазоне $0.5 \leq \text{abs}(F) < 1$. Для действительных X возвращаемые массивы F удовлетворяют уравнению вида $X = F \cdot 2^E$. Для нулевых значений X возвращаются $F = 0$ и $E = 0$. Пример:

```
>> x=[2 4.678 5;0.987 1 3];
>> [F,E] = log2(X)
F =
    0.5000 0.5847 0.6250
    0.9870 0.5000 0.7500
E =
     2     3     3
     0     1     2
```

- $\log_{10}(X)$ – возвращает логарифм по основанию 10 для каждого элемента X. Область функции включает комплексные числа, что способно привести к непредвиденным результатам при некорректном использовании. Пример:

```
>> x=[1.4 2.23 5.8 3];
>> log10(X)
ans = 0.1461 0.3483 0.7634 0.4771
```

- $\text{mod}(x, y)$ – возвращает $x \bmod y$;
- $\text{mod}(X, Y)$ – возвращает остаток от деления X на Y (то есть $X - Y \cdot \text{floor}(X./Y)$) для ненулевого Y и X в противном случае. Если операнды X и Y имеют одинаковый знак, функция $\text{mod}(X, Y)$ возвращает тот же результат, что $\text{rem}(X, Y)$. Однако (для положительных X и Y) $\text{mod}(-x, y) = \text{rem}(-x, y) + y$. Примеры:

```
>> M = mod(5,2)
M = 1
>> mod(10,4)
ans = 2
```

- $\text{pow2}(Y)$ – возвращает массив X, где каждый элемент есть 2^Y ;
- $\text{pow2}(F, E)$ – вычисляет $X = F \cdot 2^E$ для соответствующих элементов F и E. Аргументы F и E – массивы действительных и целых чисел соответственно. Пример:

```
>> d=pow2(pi/4,2)
d = 3.1416
```

- $p = \text{nextpow2}(A)$ – возвращает такой показатель степени p, что $2^p \geq \text{abs}(A)$. Эта функция эффективно применяется для выполнения быстрого

преобразования Фурье. Если A не является скалярной величиной, то `nextpow2` возвращает значение `nextpow2(length(A))`.

Пример:

```
>> x=[2 6 7 8 9 3 4 5 6 7 7 8 4 3 2 4];
>> length(x)
ans = 16
>> p = nextpow2(x)
p = 4
>> x=4;
>> p = nextpow2(x)
p = 2
>> x=45;
>> p = nextpow2(x)
p = 6
```

Функция `log(10)` уже в MATLAB 6.5 была доработана так, что теперь она возвращает значение i в выражении `log10(10^i)`, например:

```
>> log10(10^5)
ans = 5
>> log10(10^5.678)
ans = 5.6780
```

Функция `primes(n)` возвращает вектор-строку простых чисел, меньших или равных n . Пример:

```
>> p = primes(25)
p = 2 3 5 7 11 13 17 19 23
```

- `[N, D] = rat(X)` – возвращает массивы N и D , такие, что $N./D$ аппроксимирует X с точностью $1.e-6*\text{norm}(X(:), 1)$. Даже при том, что все числа с плавающей запятой – рациональные, иногда желательно аппроксимировать их дробями, у которых числитель и знаменатель являются по возможности малыми целыми числами. Функция `rat` пытается это сделать;
- `[N, D] = rat(X, tol)` – возвращает массивы N и D , такие, что $N./D$ аппроксимирует X с точностью `tol`;
- `rat(X)` без выходных параметров просто выдает на экран массив цепных дробей;
- `rats(X, strlen)` – возвращает ряд, полученный путем упрощенной рациональной аппроксимации элементов X . Аргумент `strlen` – длина возвращаемой строки. Функция возвращает знак «*», если полученное значение не может быть напечатано в строке, длина которой задана значением `strlen`. По умолчанию `strlen=13`. Тот же алгоритм аппроксимации используется в командном окне MATLAB при задании рационального формата вывода командой `format rat`. Пример:

```
>> [g, j]=rat(pi, 1e-10)
g = 312689
j = 99532
```


- `sqrt(A)` – возвращает квадратный корень каждого элемента массива `X`. Для отрицательных и комплексных элементов `X` функция `sqrt(X)` вычисляет комплексный результат. Пример:

```
>> A=[25 21.23 55.8 3];
>> sqrt(A)
ans = 5.0      4.6076      7.4699      1.7321
```

На рис. 3.1 представлены графики ряда распространенных алгебраических функций. Эти графики получены в результате исполнения следующего файла-сценария (программы):

```
syms x
subplot(2,2,1),ezplot(x^2,[-5 5]),xlabel(''),grid on
subplot(2,2,2),ezplot(exp(x),[-2 2]),xlabel(''),grid on
subplot(2,2,3),ezplot(log(x),[0 5]),grid on
subplot(2,2,4),ezplot(sqrt(x),[0 10]),grid on
```

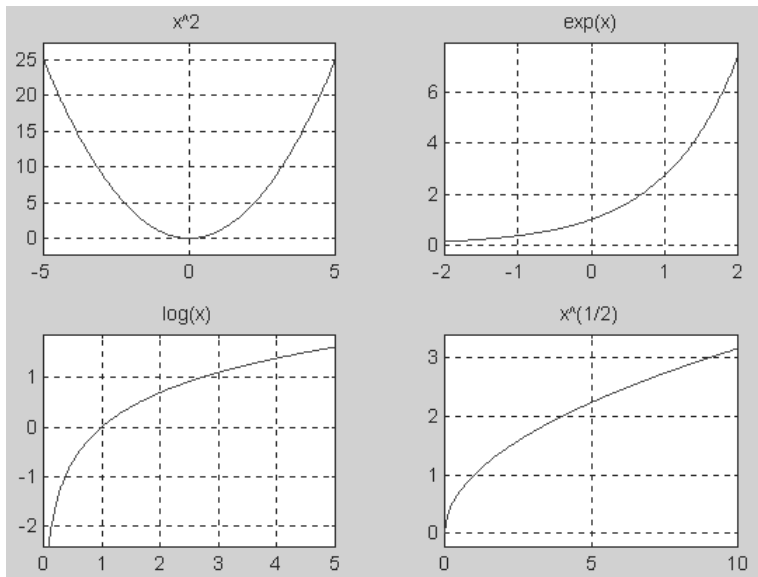


Рис. 3.1. Графики ряда алгебраических функций

Графики дают наглядное представление о поведении представленных на них функций. Обратите внимание на применение графической команды `ezplot` из пакета `Symbolic Math Toolbox` (она отличается от обычной команды `ezplot` `MATLAB` отсутствием заключения символьных переменных в апострофы `'`), команды `syms`, также входящей в пакет `Symbolic Math Toolbox` и задающей символьную переменную `x`, и несколько необычное применение команды `xlabel('')`. Эта команда с аргументом в виде пустой строки снимает вывод обозначения горизонтальной оси на двух верхних графиках. Если этого не сделать, то

символ `x` окажется наложенным на наименование функций нижних графиков, которое команда `ezplot` выводит над графиками автоматически.

3.4.2. Тригонометрические и обратные тригонометрические функции

В системе MATLAB определены следующие тригонометрические и обратные тригонометрические функции. Функции вычисляются для каждого элемента массива. Входной массив допускает комплексные значения. Напоминаем, что все углы в функциях задаются в радианах.

- `acos(X)` – возвращает арккосинус для каждого элемента X . Для действительных значений X в области $[-1, 1]$ `acos(X)` возвращает действительное значение из диапазона $[0, \pi]$, для действительных значений X вне области $[-1, 1]$ `acos(X)` возвращает комплексное число. Примеры:

```
>> Y = acos(0.5)
Y = 1.0472
>> acos([0.5 1 2])
ans = 1.0472 0 + 1.3170i
```

- `acot(X)` – возвращает арккотангенс для каждого элемента X . Пример:

```
>> Y=acot(0.1)
Y = 1.4711
```

- `acsc(X)` – возвращает арккосеканс для каждого элемента X . Пример:

```
>> Y = acsc(3)
Y = 0.3398
```

- `asec(X)` – возвращает арксеканс для каждого элемента X . Пример:

```
>> Y=asec(0.5)
Y = 0 + 1.3170i
```

- `asin(X)` – возвращает арксинус для каждого элемента X . Для действительных значений X в области $[-1, 1]$ `asin(X)` возвращает действительное число из диапазона $[-\pi/2, \pi/2]$, для действительных значений X вне области $[-1, 1]$ `asin(X)` возвращает комплексное число. Пример:

```
>> Y = asin(0.278)
Y = 0.2817
```

- `atan(X)` – возвращает арктангенс для каждого элемента X . Для действительных значений X `atan(X)` находится в области $[-\pi/2, \pi/2]$. Пример:

```
>> Y=atan(1)
Y = 0.7854
```

- `atan2(Y, X)` – возвращает массив P той же размерности что X и Y , содержащий поэлементно арктангенсы отношения вещественных частей Y и X . Мнимые части игнорируются. Элементы P находятся в интервале $[-\pi, \pi]$. Специфический квадрант определен функциями `sign(Y)` и `sign(X)`. Это

отличает полученный результат от результата $\text{atan}(Y/X)$, который ограничен интервалом $[-\pi/2, \pi/2]$. Пример:

```
>> atan2(1,2)
ans = 0.4636
```

- $\cos(X)$ – возвращает косинус для каждого элемента X . Пример:

```
>> X=[1 2 3];
>> cos(X)
ans = 0.5403 -0.4161 -0.9900
```

- $\cot(X)$ – возвращает котангенс для каждого элемента X . Пример:

```
>> Y = cot(2)
Y = -0.4577
```

- $\csc(X)$ – возвращает косеканс для каждого элемента X . Пример:

```
>> X=[2 4.678 5;0.987 1 3];
>> Y = csc(X)
Y =
    1.0998 -1.0006 -1.0428
    1.1985  1.1884  7.0862
```

- $\sec(X)$ – возвращает массив той же размерности, что и X , состоящий из секансов элементов X . Пример:

```
>> X=[pi/10 pi/3 pi/5];
>> sec(X)
ans = 1.0515  2.0000  1.2361
```

- $\sin(X)$ – возвращает синус для каждого элемента X . Пример:

```
>> X=[pi/2 pi/4 pi/6 pi];
>> sin(X)
ans = 1.0000  0.7071  0.5000  0.0000
```

- $\tan(X)$ – возвращает тангенс для каждого элемента X . Пример:

```
>> X=[0.08 0.06 1.09]
X = 0.0800  0.0600  1.0900
>> tan(X)
ans = 0.802  0.0601  1.9171
```

Следующий файл-сценарий (программа) позволяет наблюдать графики четырех тригонометрических функций (рис. 3.2):

```
% Программа построения графиков 4-х
% тригонометрических функций
syms x
subplot(2,2,1),ezplot(sin(x),[-5 5]),xlabel(''),grid on
subplot(2,2,2),ezplot(tan(x),[-5 5]),xlabel(''),grid on
subplot(2,2,3),ezplot(asin(x),[-1 1]),grid on
subplot(2,2,4),ezplot(atan(x),[-5 5]),grid on
```

Поскольку многие тригонометрические функции периодичны, появляется возможность формирования из них любопытных комбинаций, позволяющих создавать типовые тестовые сигналы, используемые при моделировании радиоэлек-

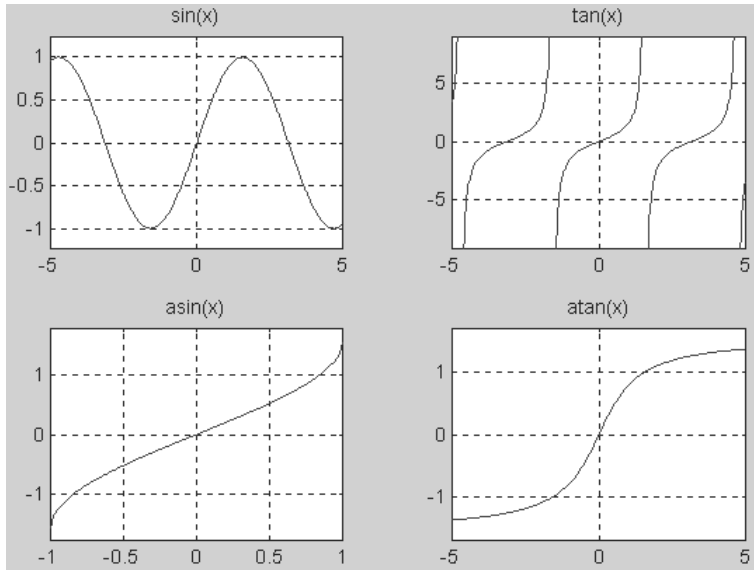


Рис. 3.2. Графики четырех тригонометрических функций

тронных устройств. Следующая программа строит графики для таких комбинаций, создающих из синусоиды три наиболее распространенных сигнала – прямоугольные, пилообразные и треугольные импульсы¹:

```
% Программа построения графиков сигналов
x=-10:0.01:10;
subplot(2,2,1),plot(x,0.8*sin(x)),...
xlabel('0.8*sin(x)')
subplot(2,2,2),plot(x,0.8*sign(sin(x))),...
xlabel('0.8*sgn(sin(x))')
subplot(2,2,3),plot(x,atan(tan(x/2))),...
xlabel('atan(tan(x/2))')
subplot(2,2,4),plot(x,asin(sin(x))),...
xlabel('asin(sin(x))')
```

Соответствующие графики представлены на рис. 3.3.

Дополнительный ряд графиков, полученных комбинациями элементарных функций, показан на рис. 3.4. Эти графики строятся следующим файлом-сценарием (программой):

```
% Программа построения графиков комбинаций
% элементарных функций
x=-10:0.01:10;
subplot(2,2,1),plot(x,sin(x).^3),xlabel('sin(x)^3')
```

¹ В пакете расширения Signal Processing Toolbox есть специальные функции для генерации таких сигналов – square и sawtooth.

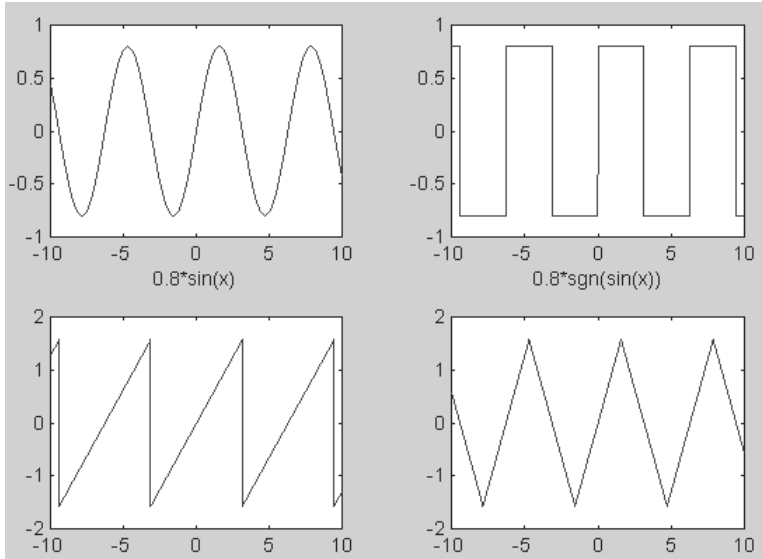


Рис. 3.3. Графики синусоиды, прямоугольных, пилообразных и треугольных колебаний

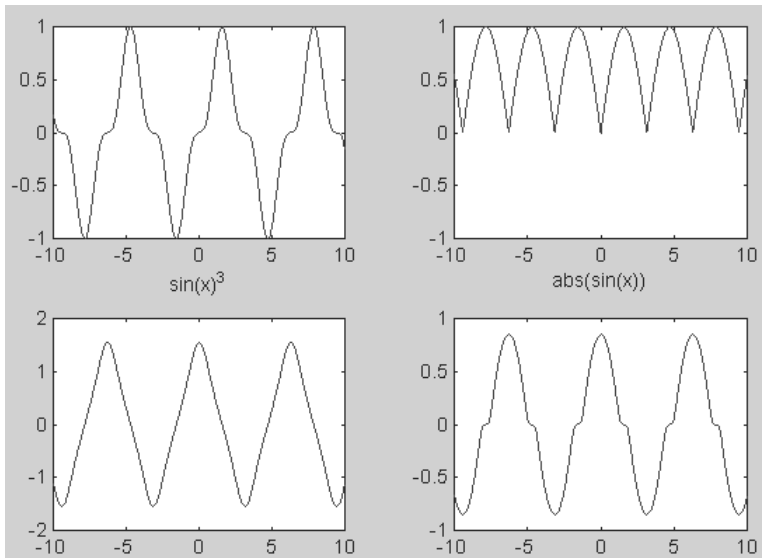


Рис. 3.4. Графики периодических сигналов без разрывов

```
subplot(2,2,2),plot(x,abs(sin(x))),xlabel('abs(sin(x))'),axis([-10
10 -1 1]),
subplot(2,2,3),plot(x,tan(cos(x))),xlabel('tan(cos(x))')
subplot(2,2,4),plot(x,csch(sec(x))),xlabel('csch(sec(x))')
```

Эти графики неплохо моделируют сигналы, получаемые при выпрямлении синусоидального напряжения (или тока) и при прохождении синусоидальных сигналов через нелинейные цепи.

3.4.3. Вычисление гиперболических и обратных гиперболических функций

Наряду с тригонометрическими функциями в математических расчетах часто используются и гиперболические функции. Ниже приводится список таких функций, определенных в системе MATLAB. Функции вычисляются для каждого элемента массива. Входной массив допускает комплексные значения. Все углы в тригонометрических функциях измеряются в радианах.

- `acosh(X)` – возвращает гиперболический арккосинус для каждого элемента X . Пример:

```
>>Y = acosh (0.7)
Y =      0 + 0.7954i
```
- `acoth(X)` – возвращает гиперболический арккотангенс для каждого элемента X . Пример:

```
>>Y = acoth (0.1)
Y = 0.1003 + 1.5708i
```
- `acsch(X)` – возвращает гиперболический арккосеканс для каждого элемента X . Пример:

```
>> Y = acsch(1)
Y =      0.8814
```
- `asech(X)` – возвращает гиперболический арксеканс для каждого элемента X . Пример:

```
>> Y = asech(4)
Y =      0 + 1.3181i
```
- `asinh(X)` – возвращает гиперболический арксинус для каждого элемента X . Пример:

```
>> Y = asinh (2.456)
Y =      1.6308
```
- `atanh(X)` – возвращает гиперболический арктангенс для каждого элемента X . Пример:

```
>> X=[0.84 0.16 1.39];
>> atanh (X)
ans = 1.2212      0.1614      0.9065 + 1.5708i
```
- `cosh(X)` – возвращает гиперболический косинус для каждого элемента X .

Пример:

```
>> X=[1 2 3];
>> cosh(X)
ans = 1.5431  3.7622  10.0677
```

- $\coth(X)$ – возвращает гиперболический котангенс для каждого элемента X. Пример:

```
>> Y = coth(3.987)
Y = 1.0007
```

- $\operatorname{csch}(x)$ – возвращает гиперболический косеканс для каждого элемента X. Пример:

```
>> X=[2 4.678 5;0.987 1 3];
>> Y = csch(X)
Y =
    0.2757    0.0186    0.0135
    0.8656    0.8509    0.0998
```

- $\operatorname{sech}(X)$ – возвращает гиперболический секанс для каждого элемента X. Пример:

```
>> X=[pi/2 pi/4 pi/6 pi];
>> sech(X)
ans = 0.3985  0.7549  0.8770  0.0863
```

- $\sinh(X)$ – возвращает гиперболический синус для каждого элемента X. Пример:

```
>> X=[pi/8 pi/7 pi/5 pi/10];
>> sinh(X)
ans = 0.4029  0.4640  0.6705  0.3194
```

- $\tanh(X)$ – возвращает гиперболический тангенс для каждого элемента X. Пример:

```
>> X=[pi/2 pi/4 pi/6 pi/10];
>> tanh(X)
ans = 0.9172  0.6558  0.4805  0.3042
```

Следующий m-файл-сценарий (программа) строит графики (рис. 3.5) ряда гиперболических функций:

```
% Программа построения графиков гиперболических функций
syms x
subplot(2,2,1),ezplot(sinh(x),[-4 4]),xlabel(''),grid on
subplot(2,2,2),ezplot(cosh(x),[-4 4]),xlabel(''),grid on
subplot(2,2,3),ezplot(tanh(x),[-4 4]),grid on
subplot(2,2,4),ezplot(sech(x),[-4 4]),grid on
```

Нетрудно заметить, что гиперболические функции, в отличие от тригонометрических, не являются периодическими. Выбранные для графического представления функции дают примеры характерных нелинейностей.

В другой программе использованы команды для построения графиков (рис. 3.6) ряда обратных гиперболических функций:

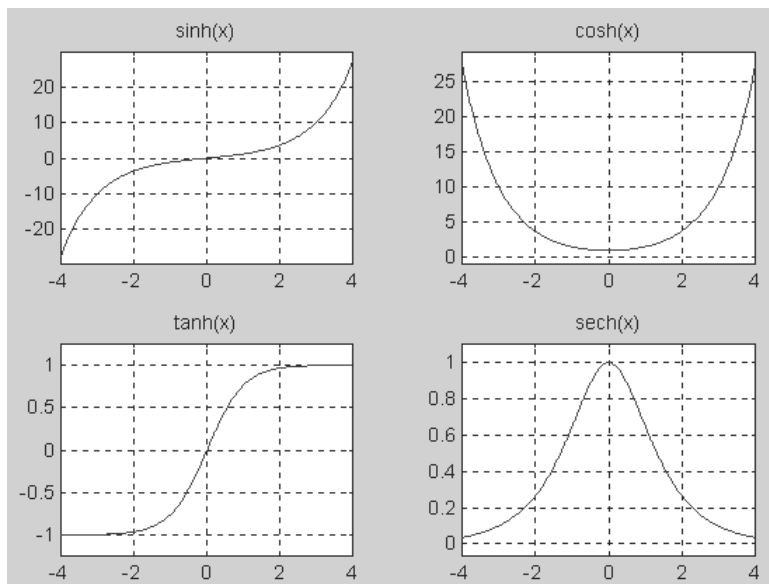


Рис. 3.5. Графики гиперболических функций

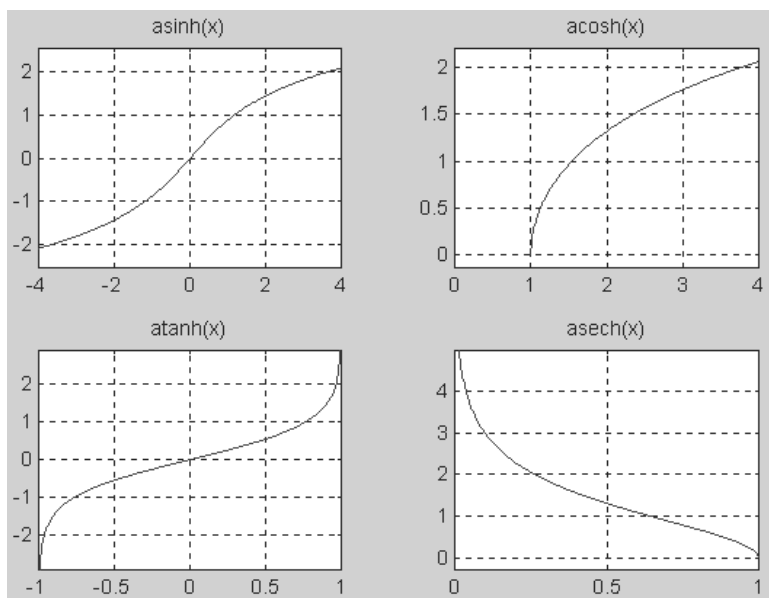


Рис. 3.6. Графики обратных гиперболических функций


```
% Программа построения графиков обратных
% гиперболических функций
syms x
subplot(2,2,1),ezplot(asinh(x),[-4 4]),xlabel(''),grid on
subplot(2,2,2),ezplot(acosh(x),[0 4]),xlabel(''),grid on
subplot(2,2,3),ezplot(atanh(x),[-1 1]),grid on
subplot(2,2,4),ezplot(asech(x),[0 1]),grid on
```

На этих графиках хорошо видны особенности данного класса функций. Такие функции, как обратный гиперболический синус и тангенс, «ценятся» за симметричный вид их графиков, дающий приближение к ряду типовых нелинейностей.

3.5. Числовые функции

3.5.1. Округление и смена знака чисел

Ряд особых функций служат для выполнения операций округления числовых данных и анализа их знака.

- `fix(A)` – возвращает массив `A` с элементами, округленными до ближайшего к нулю целого числа. Для комплексного `A` действительные и мнимые части округляются отдельно. Примеры:

```
>> A=[1/3 2/3; 4.99 5.01]
A =
    0.3333    0.6667
    4.9900    5.0100
>> fix(A)
ans =
     0     0
     4     5
```

- `floor(A)` – возвращает `A` с элементами, представляющими ближайшее меньшее или равное соответствующему элементу `A` целое число. Для комплексного `A` действительные и мнимые части преобразуются отдельно. Примеры:

```
>> A=[-1/3 2/3; 4.99 5.01]
A =
   -0.3333    0.6667
    4.9900    5.0100
>> floor(A)
ans =
    -1     0
     4     5
```

- `ceil(A)` – возвращает ближайшее большее или равное `A` целое число. Для комплексного `A` действительные и мнимые части округляются отдельно. Примеры:

```
>> a=-1.789;
>> ceil(a)
```

```
ans = -1
>> a=-1.789+i*3.908;
>> ceil(a)
ans = -1.0000 + 4.0000i
```

- $\text{rem}(X, Y)$ – возвращает $X - \text{fix}(X./Y) .* Y$, где $\text{fix}(X./Y)$ – целая часть от частного X/Y .

Если операнды X и Y имеют одинаковый знак, функция $\text{rem}(X, Y)$ возвращает тот же результат, что $\text{mod}(X, Y)$. Однако (для положительных X и Y) $\text{rem}(-x, y) = \text{mod}(-x, y) - y$. Функция rem возвращает результат, находящийся между 0 и $\text{sign}(X) * \text{abs}(Y)$. Если $Y=0$, функция rem возвращает NaN. Аргументы X и Y должны быть целыми числами. Из-за неточного представления в компьютере чисел с плавающей запятой использование вещественных (или комплексных) входных аргументов может привести к непредвиденным результатам. Пример:

```
>> X=[25 21 23 55 3];
>> Y=[4 8 23 6 4];
>> rem(X,Y)
ans = 1 5 0 1 3
```

- $\text{round}(X)$ – возвращает округленные до ближайшего целого элементы массива X . Для комплексного X действительные и мнимые части округляются отдельно.

Пример:

```
>> X=[5.675 21.6+4.897*i 2.654 55.8765];
>> round(X)
ans = 6.0000 22.0000 + 5.0000i 3.0000 56.0000
```

- $\text{sign}(X)$ – возвращает массив Y той же размерности, что и X , где каждый из элементов Y равен:

- 1, если соответствующий элемент X больше 0;
- 0, если соответствующий элемент X равен 0;
- -1, если соответствующий элемент X меньше 0.

Для ненулевых действительных и комплексных X :

$$\text{sign}(X) = X ./ \text{abs}(X)$$

Пример:

```
>> X=[-5 21 2 0 -3.7];
>> sign(X)
ans =
    -1     1     1     0    -1
```

3.5.2. Операции с комплексными числами

Комплексные числа особенно широко применяются в электро- и радиотехнике, где они являются основой *символического метода* анализа линейных цепей. Для работы с комплексными числами и данными в MATLAB используется ряд функций.

- `angle(Z)` возвращает аргумент комплексного числа в радианах для каждого элемента массива комплексных чисел `Z`. Углы находятся в диапазоне $[-\pi; +\pi]$. Для комплексного `Z` модуль и аргумент вычисляются следующим образом: `R = abs(Z)` – модуль, `theta = angle(Z)` – аргумент. При этом формула `Z = R.*exp(i*theta)` дает переход от показательной формы представления комплексного числа к алгебраической. Примеры:

```
>> Z=3+i*2
Z =      3.0000 + 2.0000i
>> theta = angle(Z)
theta = 0.5880
>> R = abs(Z)
R =      3.6056
>> Z =R.*exp(i*theta)
Z =      3.0000 + 2.0000i
```

- `imag(Z)` – возвращает мнимые части всех элементов массива `Z`. Пример:

```
>> Z=[1+i, 3+2i, 2+3i];
>> imag(Z)
ans =      1      2      3
```

- `real(Z)` – возвращает вещественные части всех элементов комплексного массива `Z`. Пример:

```
>> Z=[1+i, 3+2i, 2+3i];
>> real(Z)
ans =      1      3      2
```

- `conj(Z)` – возвращает число, комплексно-сопряженное аргументу `Z`. Если `Z` комплексное, то `conj(Z) = real(Z) - i * imag(Z)`. Пример:

```
>> conj(2+3i)
ans = 2.0000 - 3.0000i
```

3.6. Специальные математические функции

Специальные математические функции являются решениями дифференциальных уравнений специального вида или обозначениями некоторых видов интегралов. Довольно полный обзор специальных функций дается в книгах [51–54], так что ниже мы ограничимся только указанием функций системы MATLAB, реализующих их вычисление.

3.6.1. Функции Эйри

Функция Эйри формирует пару линейно независимых решений линейного дифференциального уравнения вида

$$\frac{d^2W}{dZ^2} - ZW = 0.$$

Связь между функцией Эйри и модифицированной функцией Бесселя выражается формулой

$$\text{Ai}(Z) = \left[\frac{1}{\pi} \sqrt{\frac{Z}{3}} \right] K_{1/3}(\zeta),$$

где $\zeta = \frac{2}{3} Z^{3/2}$.

- $\text{airy}(Z)$ – возвращает функцию Эйри, $\text{Ai}(Z)$, для каждого элемента комплексного массива Z ;
- $\text{airy}(k, Z)$ – возвращает различные варианты результата в зависимости от значения k :
- $k=0$ – тот же результат, что и $\text{airy}(Z)$;
- $k=1$ – производную от $\text{Ai}(Z)$;
- $k=2$ – функцию Эйри второго рода, $\text{Bi}(Z)$;
- $k=3$ – производную от $\text{Bi}(Z)$.

Пример:

```
>> D=[1, 3+2i]
D =      1.0000      3.0000 + 2.0000i
>> S=airy(D)
S =      0.1353      -0.0097 + 0.0055i
```

3.6.2. Функции Бесселя

Линейное дифференциальное уравнение второго порядка вида

$$z^2 \frac{d^2 y}{dz^2} + z \frac{dy}{dz} + (z^2 - \nu^2)y = 0,$$

где ν – неотрицательная константа, называется *уравнением Бесселя*, а его решения известны как *функции Бесселя*. Функции $J_\nu(z)$ и $J_{-\nu}(z)$ формируют фундаментальное множество решений уравнения Бесселя для неотрицательных значений ν (это так называемые *функции Бесселя первого рода*):

$$J_\nu(z) = \left(\frac{z}{2} \right)^\nu \sum_{k=0}^{\infty} \frac{\left(\frac{-z^2}{4} \right)^k}{k! \Gamma(\nu + k + 1)},$$

где для гамма-функции используется следующее представление:

$$\Gamma(a) = \int_0^{\infty} e^{-t} t^{a-1} dt.$$

Второе решение уравнения Бесселя, линейно независимое от $J_\nu(z)$, определяется как

$$Y_\nu(z) = \frac{J_\nu(z) \cos(\nu\pi) - J_{-\nu}(z)}{\sin(\nu\pi)}$$

и задает *функции Бесселя второго рода* $Y_\nu(z)$.

Функции Бесселя третьего рода (функции Ханкеля) и функция Бесселя первого и второго рода связаны следующим выражением:

$$H_v^{(1)}(z) = J_v(z) + iY_v(z),$$

$$H_v^{(2)}(z) = J_v(z) - iY_v(z),$$

где $J_v(z)$ – это `besselj`, а $Y_v(z)$ – `bessely`.

- `besselj(nu, Z)` – возвращает функцию Бесселя первого рода, $J_v(z)$, для каждого элемента комплексного массива Z . Порядок nu может не быть целым, однако должен быть вещественным. Аргумент Z может быть комплексным. Результат вещественный, если Z положительно. Если nu и Z – массивы одинакового размера, то результат имеет тот же размер. Если любая входная величина – скаляр, результат расширяется до размера другой входной величины. Если одна входная величина – вектор-строка, а другая – вектор-столбец, результат представляет собой двумерный массив значений функции.
- `bessely(nu, Z)` – возвращает функцию Бесселя второго рода, $Y_v(z)$.

При вызове в формате `[J, ierr] = besselj(nu, Z)` и `[Y, ierr] = bessely(nu, Z)` функции всегда возвращают массив с флагами ошибок:

- `ierr = 1` – запрещенные аргументы;
- `ierr = 2` – переполнение (возвращает `Inf`);
- `ierr = 3` – некоторая потеря точности при приведении аргумента;
- `ierr = 4` – недопустимая потеря точности: Z или nu слишком велики;
- `ierr = 5` – нет сходимости (возвращает `NaN`).

Примеры:

```
>> S=[2-5i, 4, 7]; T=[8, 1, 3]; g=besselj(T, S)
g =
    -0.1114 - 0.0508i    -0.0660    -0.1676
>> S=[2-5i, 4, 7]; T=[8, 1, 3]; [g, ierr]=bessely(T, S)
g =
    0.1871 - 0.0324i    0.3979    0.2681
ierr = 0      0      0
```

- `besselh(nu, K, Z)` – для $K=1$ или 2 возвращает функцию Бесселя третьего рода (функцию Ханкеля) для каждого элемента комплексного массива Z . Если nu и Z – массивы одинакового размера, то результат имеет тот же размер. Если одна из входных величин – скаляр, результат формируется по размеру другой входной величины. Если одна входная величина – вектор-строка, а другая – вектор-столбец, результат представляет собой двумерный массив значений функции.
- `besselh(nu, Z)` – использует по умолчанию $K = 1$.
- `besselh(nu, 1, Z, 1)` – масштабирует $H_v^{(1)}(z)$ с коэффициентом $\exp(-i*z)$.
- `besselh(nu, 2, Z, 1)` – масштабирует $H_v^{(2)}(z)$ с коэффициентом $\exp(+i*z)$.
- `[H, ierr] = besselh(...)` – всегда возвращает массив с флагами ошибок:
 - `ierr = 1` – запрещенные аргументы;

- `ierr = 2` – переполнение (возвращает `Inf`);
- `ierr = 3` – некоторая потеря точности при приведении аргумента;
- `ierr = 4` – недопустимая потеря точности: `Z` или `nu` слишком велики;
- `ierr = 5` – нет сходимости (возвращает `NaN`).

Пример:

```
>> D=[1, 3+2i]; F=[3, 2]; besselh(F, D)
ans = 0.0196 - 5.8215i      0.0509 - 0.0583i
```

Линейное дифференциальное уравнение второго порядка вида

$$z^2 \frac{d^2 y}{dz^2} + z \frac{dy}{dz} - (z^2 + \nu^2)y = 0,$$

где ν – неотрицательная константа, называется *модифицированным уравнением Бесселя*, а его решения известны как *модифицированные функции Бесселя* $I_\nu(z)$ и $I_{-\nu}(z)$. $K_\nu(z)$ – второе решение модифицированного уравнения Бесселя, линейно независимое от $I_\nu(z)$. $I_\nu(z)$ и $K_\nu(z)$ определяются следующим образом:

$$I_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{\left(\frac{z^2}{4}\right)^k}{k! \Gamma(\nu + k + 1)},$$

$$K_\nu(z) = \left(\frac{\pi}{2}\right) \frac{I_{-\nu}(z) - I_\nu(z)}{\sin(\nu\pi)}.$$

- `besseli(nu, Z)` – возвращает модифицированную функцию Бесселя первого рода, $I_\nu(z)$, для каждого элемента массива `Z`. Порядок `nu` может не быть целым, однако должен быть вещественным. Аргумент `Z` может быть комплексным. Результат вещественный, если `Z` положительно. Если `nu` и `Z` – массивы одинакового размера, то результат имеет тот же размер. Если любая входная величина – скаляр, результат расширяется до размера другой входной величины. Если одна входная величина – вектор-строка, а другая – вектор-столбец, результат является двумерным массивом значений функции.
- `besselk(nu, Z)` – возвращает модифицированную функцию Бесселя второго рода, $K_\nu(z)$, для каждого элемента комплексного массива `Z`.
- `besseli(nu, Z, 1)` – возвращает `besseli(nu, Z) .* exp(-Z)`.
- `besselk(nu, Z, 1)` – возвращает `besselk(nu, Z) .* exp(-Z)`.
- `[I, ierr] = besseli(...)` и `[K, ierr] = besselk(...)` всегда возвращают массив с флагами ошибок:
 - `ierr = 1` – запрещенные аргументы;
 - `ierr = 2` – переполнение (возвращает `Inf`);
 - `ierr = 3` – некоторая потеря точности при приведении аргумента;
 - `ierr = 4` – недопустимая потеря точности: `Z` или `nu` слишком велики;
 - `ierr = 5` – нет сходимости (возвращает `NaN`).

Примеры:

```
>> D=[1, 3+2i]; F=[3, 2]; K=besseli(F, D)
K =      0.0222      -1.2577 + 2.3188i
```

```
>> D=[1,3+2i];F=[3,2];[K,ierr]=besselk(F,D)
K =          7.1013          -0.0401 - 0.0285i
ierr =          0              0
```

Естественно, что возможно построение графиков специальных функций. В качестве примера рассмотрим простую программу (m-файл-сценарий), приведенную ниже:

```
x=0:0.1:10; y0=besselj(0,x);
y1=besselj(1,x);y2=besselj(2,x); y3=besselj(3,x);
plot(x,y0,'-m',x,y1,'-r',x,y2,'-.k',x,y3,':b');
legend('besselj(0,x)', 'besselj(1,x)', 'besselj(2,x)', 'besselj(3,x)');
```

Рисунок 3.7 иллюстрирует построение четырех функций Бесселя $besselj(n, x)$ для $n = 0, 1, 2$ и 3 с легендой, облегчающей идентификацию каждой кривой рисунка.

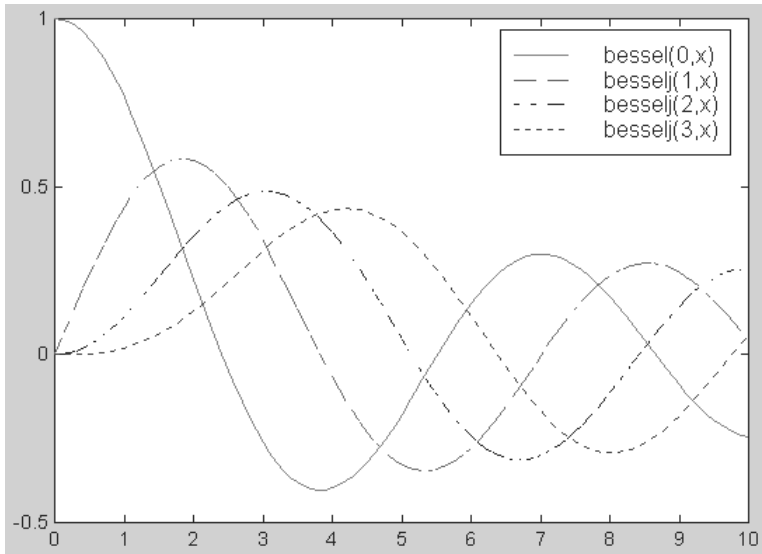


Рис. 3.7. Графики четырех функций Бесселя $besselj(n, x)$

Эти графики дают наглядное представление о поведении функций Бесселя, широко используемых при анализе поведения систем, описываемых линейными дифференциальными уравнениями второго порядка. Описание построения графиков функций Бесселя при комплексном аргументе можно найти в [33].

3.6.3. Бета-функция и ее варианты

Бета-функция определяется как

$$B(z, w) = \int_0^1 t^{z-1} (1-t)^{w-1} dt = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)},$$

где $\Gamma(z)$ – гамма-функция. Неполная бета-функция определяется по формуле

$$I_x(z, w) = \frac{1}{B(z, w)} \int_0^x t^{z-1} (1-t)^{w-1} dt.$$

- `beta(Z, W)` – возвращает бета-функцию для соответствующих элементов комплексных массивов Z и W . Массивы должны быть одинакового размера (или одна из величин может быть скаляром).
- `betainc(X, Z, W)` – возвращает неполную бета-функцию. Элементы X должны быть в закрытом интервале $[0, 1]$.
- `betaln(Z, W)` – возвращает натуральный логарифм бета-функции $\log(\text{beta}(Z, W))$, без вычисления $\text{beta}(Z, W)$. Так как сама бета-функция может принимать очень большие или очень малые значения, функция `betaln(Z, W)` иногда более полезна, так как позволяет избежать переполнения.

Пример:

```
>> format rat; beta((1:10)'.^1, 4)
ans = 1/4 1/20 1/60 1/140 1/280 1/504 1/840 1/1320 1/1980 1/2860
```

3.6.4. Эллиптические функции и интегралы

Эллиптические функции Якоби определяются интегралом

$$u = \int_0^{\phi} \frac{d\theta}{(1 - m \sin^2 \theta)^{1/2}}$$

и соотношениями:

$$\text{sn}(u) = \sin \phi,$$

$$\text{cn}(u) = \cos \phi,$$

$$\text{dn}(u) = (1 - \sin^2 \phi)^{1/2},$$

$$\text{am}(u) = \phi.$$

В некоторых случаях при определении эллиптических функций используются модули k вместо параметра m . Они связаны выражением $k^2 = m = \sin^2 \alpha$.

- `[SN, CN, DN] = ellipj(U, M)` – возвращает эллиптические функции Якоби SN , CN и DN , вычисленные для соответствующих элементов – аргумента U и параметра M . Входные величины U и M должны иметь один и тот же размер (или любая из них может быть скаляром).
- `[SN, CN, DN] = ellipj(U, M, tol)` – возвращает эллиптическую функцию Якоби, вычисленную с точностью `tol`. Значение `tol` по умолчанию – `eps`; его можно увеличить, тогда результат будет вычислен быстрее, но с меньшей точностью.

¹ Обратите внимание, что аргумент задается как вектор-столбец.

Пример:

```
>> [SN, CN, DN]=ellipj ([23, 1], [0.5, 0.2])
SN =      474/719                1224/1481
CN =     1270/1689              1457/2588
DN =      399/451                538/579
```

Полные эллиптические интегралы первого и второго рода определяются следующим образом:

$$K(m) = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}} = \int_0^{\pi/2} \frac{d\theta}{(1-m \sin^2 \theta)^{1/2}},$$

$$E(m) = \int_0^1 (1-t^2)^{-1/2} (1-mt^2)^{1/2} dt = \int_0^{\pi/2} (1-m \sin^2 \theta)^{1/2} d\theta.$$

- `ellipke(M)` – возвращает полный эллиптический интеграл первого рода для элементов M.
- `[K, E]=ellipke(M)` – возвращает полные эллиптические интегралы первого и второго рода.
- `[K, E]=ellipke(M, tol)` – возвращает эллиптические функции Якоби, вычисленные с точностью `tol`. Значение по умолчанию – `eps`; его можно увеличить, тогда результат будет вычислен быстрее, но с меньшей точностью.

Пример:

```
>> [f, e]=ellipke([0.2, 0.8])
f =      707/426                1018/451
e =      679/456                515/437
```

Для вычисления этих функций используется итерационный метод арифметико-геометрического среднего (см. детали в Reference Book по системе MATLAB).

3.6.5. Функции ошибки

Функция ошибки определяется следующим образом:

$$\operatorname{erf}(X) = \frac{2}{\sqrt{\pi}} \int_0^X e^{-t^2} dt.$$

- `erf(X)` – возвращает значение функции ошибки для каждого элемента вещественного массива X.

Дополнительная (остаточная) функция ошибки задается соотношением

$$\operatorname{erfc}(X) = \frac{2}{\sqrt{\pi}} \int_X^\infty e^{-t^2} dt = 1 - \operatorname{erf}(X).$$

- `erfc(X)` – возвращает значение остаточной функции ошибки.
- `erfcx(X)` – возвращает значение масштабированной остаточной функции ошибки. Эта функция определяется так:

$$\operatorname{erfcx}(x) = e^{x^2} \operatorname{erfc}(x).$$

- `erfinv(Y)` – возвращает значение обратной функции ошибки для каждого элемента массива Y . Элементы массива Y должны лежать в области $-1 < Y < 1$.

Примеры:

```
>> Y=[0.2,-0.3];a=erf(Y)
a =    0.2227   -0.3286
>> b=erfc(Y)
b =    0.7773    1.3286
>> c=erfcx(Y)
c =    0.8090    1.4537
>> d=erfinv(Y)
d =    0.1791   -0.2725
```

При вычислении данных функций используется аппроксимация по Чебышеву (см. детали алгоритма в Reference Book по MATLAB).

3.6.6. Интегральная показательная функция

Интегральная показательная функция определяется следующим образом:

$$E_1(x) = \int_x^{\infty} \frac{e^{-t}}{t} dt.$$

- `expint(X)` – возвращает интегральную показательную функцию для каждого элемента X . Пример:

```
>> d=expint([2,3+7i])
d =    0.0489   -0.0013  -0.0060i
```

Для вычисления этой функции используется ее разложение в ряд (см. [40]).

3.6.7. Гамма-функция и ее варианты

Гамма-функция определяется выражением

$$\Gamma(a) = \int_0^x e^{-t} t^{a-1} dt.$$

Неполная гамма-функция определяется как

$$P(x,a) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt.$$

В MATLAB заданы следующие функции этого класса:

- `gamma(A)` – возвращает гамма-функцию элементов A . Аргумент A должен быть вещественным.
- `gammainc(X,A)` – возвращает неполную гамма-функцию соответствующих элементов X и A . Аргументы X и A должны быть вещественными и иметь одинаковый размер (или любой из них может быть скалярным).
- `gammaLn(A)` – возвращает логарифмическую гамма-функцию, $\text{gammaLn}(A) = \log(\text{gamma}(A))$. Команда `gammaLn` позволяет избежать переполнения,

которое может происходить, если вычислять логарифмическую гамма-функцию непосредственно, используя `log(gamma(A))`.

Примеры:

```
>> f=[5,3];d=gamma(f)
d =      24      2
>> h=gammaln(f)
h =      3.1781      0.6931
```

Гамма-функция имеет довольно сложный график (рис. 3.8), заслуживающий построения. Это можно осуществить с помощью следующего файла-сценария:

```
syms x
ezplot(gamma(x),[-4 4])
grid on
```

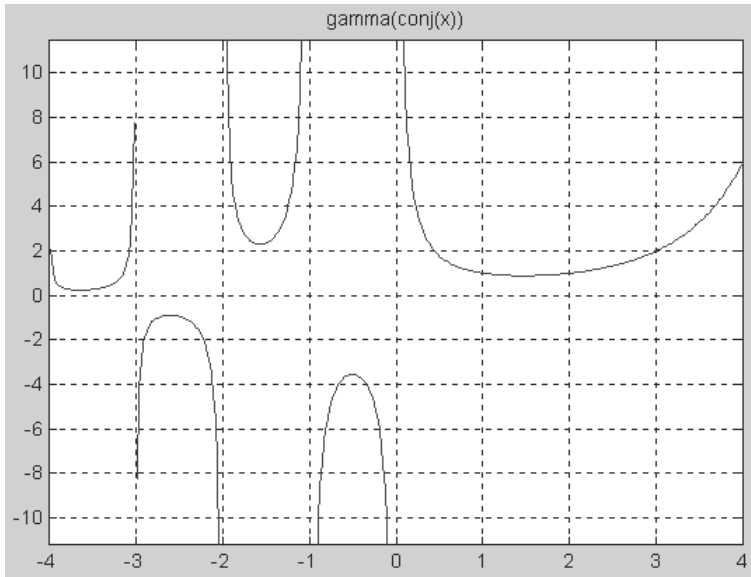


Рис. 3.8. График гамма-функции

Гамма-функция вычисляется по известному алгоритму W. J. Cody (1989 г.). Для вычисления неполной гамма-функции используются рекуррентные формулы, приведенные в [53].

3.6.8. Ортогональные полиномы Лежандра

Функция Лежандра определяется следующим образом:

$$P_n^m = (-1)^m (1-x^2)^{m/2} \frac{d^m P_n(x)}{dx^m},$$

где $P_n(x)$ – полином Лежандра степени n , рассчитываемый как

$$P_n(x) = \frac{1}{2^n n!} \left[\frac{d^n (x^2 - 1)^n}{dx} \right].$$

- `legendre (n, X)` – возвращает функции Лежандра степени n и порядков $m = 0, 1, \dots, n$, вычисленные для элементов X . Аргумент n должен быть скалярным целым числом, не превосходящим 256, а X должен содержать действительные значения в области $-1 \leq x \leq 1$. Возвращаемый массив P имеет большую размерность, чем X , и каждый элемент $P(m+1, d1, d2\dots)$ содержит связанную функцию Лежандра степени n и порядка m , вычисленную в точках $X(d1, d2\dots)$.
- `legendre (n, X, 'sch')` – возвращает *квазинормализованные* по Шмидту функции Лежандра.

Пример:

```
>> g=rand(3,2);legendre(3,g)
ans(:,:,1) =
    -0.4469    -0.0277     0.1534
    -0.0558     1.4972    -2.0306
     5.4204     0.2775     4.0079
   -10.5653   -14.9923    -2.7829
ans(:,:,2) =
    -0.4472    -0.3404     0.0538
     0.0150    -1.0567    -1.9562
     5.3514     5.7350     4.4289
   -10.7782    -7.3449    -3.4148
```

Функция вычисления полиномов Лежандра имеет дополнительную форму:

`N = legendre (n, X, 'norm')` – вычисляет полином Лежандра в полной нормализованной форме.

Формулы для вычисления функций Лежандра приведены в [53].

3.6.9. Полигамма-функция ψ

Набор специальных математических функций в системах MATLAB функционально достаточно полон, а потому пересматривается очень редко. Чести быть включенной в этот набор в версии MATLAB 6.5 удостоена полигамма-функция ψ :

- `psi (x)` – возвращает значение логарифмической производной гамма-функции (полигамма-функции) $\psi(x) = \text{digamma}(x) = d(\log(\text{gamma}(x)))/dx = \text{gamma}'(x)/\text{gamma}(x)$;
- `psi (k, x)` – возвращает значение логарифмической k -ой производной гамма-функции (полигамма-функции);
- `Y=psi (k0:k1, x)` – возвращает значение логарифмических производных гамма-функции (полигамма-функции) от $k0$ -ой до $k1$ -ой.

Примеры:

```
>> -psi(1)
```

```
ans = 0.5772
>> psi(2,2)
ans = -0.4041
>> Y=psi(1:3,1:3);
>> Y
Y =
    1.6449    0.6449    0.3949
   -2.4041   -0.4041   -0.1541
    6.4939    0.4939    0.1189
```

Операции с векторами и матрицами

| | |
|--|-----|
| 4.1. Создание матриц с заданными свойствами | 194 |
| 4.2. Операции с матрицами | 199 |
| 4.3. Создание и вычисление специальных матриц | 207 |
| 4.4. Матричные операции линейной алгебры | 213 |
| 4.5. О скорости выполнения матричных операций | 231 |

Матрицы как двумерные массивы с числовыми элементами [2, 3, 49, 55] представляют собой самые распространенные объекты языка программирования системы MATLAB. Ниже описываются основные операции с матрицами, которые выполняются как в командном режиме, так и в составе программ. По обилию матричных операторов и функций MATLAB является лидером среди массовых СКМ. Система включает в себя средства известных матричных программ LINPACK и EISPACK, в разработке которых принимал участие создатель фирмы The MathWorks, Inc., Моулер (С. Moler) [49].

4.1. Создание матриц с заданными свойствами

4.1.1. Создание единичной матрицы

Для создания *единичной матрицы* (она обычно обозначается как **E**), которая имеет единичные диагональные элементы и нулевые все остальные, служит функция `eye`:

- `eye (n)` – возвращает единичную матрицу размера $n \times n$;
- `eye (m, n)` или `eye ([m n])` – возвращают матрицу размера $m \times n$ с единицами по диагонали и нулями в остальных ячейках;
- `eye (size (A))` – возвращает единичную матрицу того же размера, что и **A**.

Единичная матрица не определена для многомерных массивов. Так, функция `y = eye ([2, 3, 4])` при попытке ее вычисления приведет к выводу сообщения об ошибке.

Пример использования функции `eye`:

```
>> t=eye(4,5)
t =
```

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |

Чаще всего применяются квадратные единичные матрицы, но последние могут быть и неквадратными, что и видно из приведенного примера.

4.1.2. Создание матрицы с единичными элементами

Для создания матриц, все (а не только диагональные) элементы которых – единицы, используется функция `ones`:

- `ones (n)` – возвращает матрицу размера $n \times n$, все элементы которой – единицы. Если n – не скаляр, то появится сообщение об ошибке;
- `ones (m, n)` или `ones ([m n])` – возвращают матрицу размера $m \times n$, состоящую из единиц;

- `ones (d1, d2, d3, ...)` или `ones ([d1 d2 d3...])` – возвращают массив из единиц с размером $d1 \times d2 \times d3 \times \dots$;
- `ones (size (A))` – возвращает массив единиц той же размерности и размера, что и A. Матрица с единичными элементами, в отличие от единичной матрицы, в MATLAB определена и для многомерных массивов.

Пример:

```
>> s=ones(3,4)
s =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

4.1.3. Создание матрицы с нулевыми элементами

Иногда нужны матрицы, все элементы которых – нули. Следующая функция обеспечивает создание таких матриц:

- `zeros (n)` – возвращает матрицу размера $n \times n$, содержащую нули. Если n – не скаляр, то появится сообщение об ошибке;
- `zeros (m, n)` или `zeros ([m n])` – возвращают матрицу размера $m \times n$, состоящую из нулей;
- `zeros (d1, d2, d3, ...)` или `zeros ([d1 d2 d3...])` – возвращают массив из нулей размера $d1 \times d2 \times d3 \times \dots$;
- `zeros (size (A))` – возвращает массив нулей того же размера и размерности, что и A.

Пример:

```
>> D=zeros(3,2)
D =
     0     0
     0     0
     0     0
```

4.1.4. Создание линейного массива равноотстоящих точек

Функция `linspace` формирует *линейный массив равноотстоящих узлов*. Это подобно оператору `:`, но дает прямой контроль над числом точек. Применяется в следующих формах:

- `linspace (a, b)` – возвращает линейный массив из 100 точек, равномерно распределенных между a и b ;
- `linspace (a, b, n)` – генерирует n точек, равномерно распределенных в интервале от a до b .

Пример:

```
>> M=linspace(4,20,14)
M =
Columns 1 through 7
 4.0000  5.2308  6.4615  7.6923  8.9231 10.1538 11.3846
Columns 8 through 14
12.6154 13.8462 15.0769 16.3077 17.5385 18.7692 20.0000
```

4.1.5. Создание вектора равноотстоящих в логарифмическом масштабе точек

Функция `logspace` генерирует вектор равноотстоящих в логарифмическом масштабе точек. Она особенно эффективна при создании вектора частот. Это логарифмический эквивалент оператора `:` и функции `linspace`:

- `logspace(a,b)` – возвращает вектор-строку из 50 равноотстоящих в логарифмическом масштабе точек между декадами 10^a и 10^b ;
- `logspace(a,b,n)` – возвращает n точек между декадами 10^a и 10^b ;
- `logspace(a,pi)` – возвращает точки в интервале между 10^a и π . Эта функция очень полезна в цифровой обработке сигналов.

Все аргументы функции `logspace` должны быть скалярными величинами.

Пример:

```
>> L=logspace(1,2,14)
L =
Columns 1 through 7
10.0000 11.9378 14.2510 17.0125 20.3092 24.2446 28.9427
Columns 8 through 14
34.5511 41.2463 49.2388 58.7802 70.1704 83.7678 100.0000
```

4.1.6. Создание массивов со случайными элементами

`p = randperm(n)` – возвращает случайные перестановки целых чисел $1:n$ в векторе-строке. Пример:

```
>> randperm(6)
ans =
     2     4     3     6     5     1
```

Функция `rand` генерирует массивы случайных чисел, значения элементов которых *равномерно* распределены в промежутке $(0,1)$:

- `rand(n)` – возвращает матрицу размера $n \times n$. Если n – не скаляр, то появится сообщение об ошибке;
- `rand(m,n)` или `rand([m n])` – возвращают матрицу размера $m \times n$;
- `rand(m,n,p,...)` или `rand([m n p...])` – возвращает многомерный массив;
- `rand(size(A))` – возвращает массив того же размера и размерности, что и A , с элементами, распределенными по равномерному закону;

- `rand` (без аргументов) – возвращает одно случайное число, которое изменяется при каждом последующем вызове и имеет равномерный закон распределения;
- `rand('state')` – возвращает вектор с 35 элементами, содержащий текущее состояние генератора случайных чисел с равномерным распределением. Для изменения состояния генератора можно применять следующие формы этой функции:
- `rand('state', s)` – устанавливает состояние в s ;
- `rand('state', 0)` – сбрасывает генератор в начальное состояние;
- `rand('state', j)` – для целых j устанавливает генератор в j -е состояние;
- `rand('state', sum(100*clock))` – каждый раз сбрасывает генератор в состояние, зависящее от времени.

Пример:

```
>> Y=rand(4,3)
Y =
    0.9501    0.8913    0.8214
    0.2311    0.7621    0.4447
    0.6068    0.4565    0.6154
    0.4860    0.0185    0.7919
```

Проверить равномерность распределения случайных чисел можно, построив большое число точек на плоскости со случайными координатами. Это делается с помощью следующих команд:

```
>> X=rand(1000,1); Y=rand(1000,1); plot(X,Y, '.')
```

Полученный при этом график показан на рис. 4.1. Нетрудно заметить, что точки довольно равномерно распределены на плоскости, так что нет оснований не доверять заданному закону распределения координат точек.

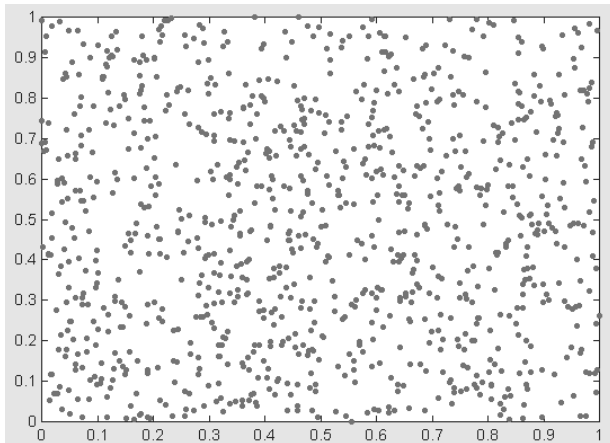


Рис. 4.1. Случайные точки
с равномерным распределением координат на плоскости

Функция `randn` генерирует массив со случайными элементами, распределенными по нормальному закону с нулевым математическим ожиданием и среднеквадратическим отклонением, равным 1:

- `randn(n)` – возвращает матрицу размера $n \times n$. Если n – не скаляр, то появится сообщение об ошибке;
- `randn(m,n)` или `randn([m n])` – возвращают матрицу размера $m \times n$;
- `randn(m,n,p,...)` или `randn([m n p...])` – возвращает массив с элементами, значения которых распределены по нормальному закону;
- `randn(size(A))` – возвращает массив того же размера, что и A , с элементами, распределенными по нормальному закону;
- `randn` (без аргументов) – возвращает одно случайное число, которое изменяется при каждом последующем вызове и имеет нормальное распределение;
- `randn('state')` – возвращает двухэлементный вектор, включающий текущее состояние нормального генератора. Для изменения состояния генератора можно применять следующие формы этой функции:
- `randn('state',s)` – устанавливает состояние в s ;
- `randn('state',0)` – сбрасывает генератор в начальное состояние;
- `randn('state',j)` – для целых j устанавливает генератор в j -е состояние;
- `randn('state',sum(100*clock))` – каждый раз сбрасывает генератор в состояние, зависящее от времени.

Пример:

```
>> Y=randn(4,3)
Y =
    -0.4326    -1.1465     0.3273
    -1.6656     1.1909     0.1746
     0.1253     1.1892    -0.1867
     0.2877    -0.0376     0.7258
```

Проверить распределение случайных чисел по нормальному закону можно, построив гистограмму распределения большого количества чисел. Например, следующие команды

```
>> Y=randn(10000,1); hist(Y,100)
```

строят гистограмму (рис. 4.2) из 100 столбцов для 10 000 случайных чисел с нормальным распределением. По рисунку видно, что огибающая гистограммы действительно близка к нормальному закону распределения.

В пакете расширения `Statistics Toolbox` можно найти множество статистических функций, в том числе для генерации случайных чисел с различными законами распределения и определения их статистических характеристик.

4.1.7. Создание массивов с логическими значениями элементов

Уже в версии MATLAB 6.5 были введены функции `true(m,n)` и `false(m,n)` для генерации массивов размера $m \times n$, содержащих соответственно логические 1 и 0. Примеры:

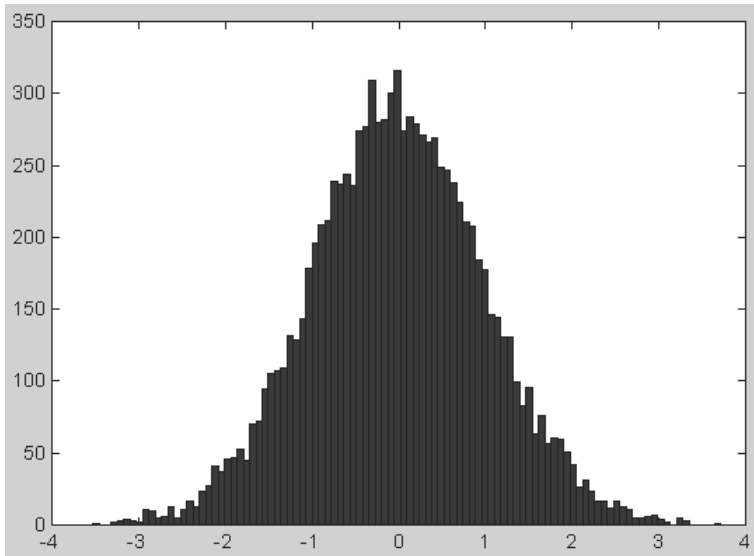


Рис. 4.2. Гистограмма для 10 000 нормально распределенных чисел в 100 интервалах

```
>> true(2,3)
ans =
     1     1     1
     1     1     1
>> false(4,2)
ans =
     0     0
     0     0
     0     0
     0     0
```

4.2. Операции с матрицами

4.2.1. Конкатенация матриц

Конкатенацией называют объединение массивов и матриц. Это реализует следующая функция:

- $C = \text{cat}(\text{dim}, A, B)$ – объединяет массивы A и B в соответствии со спецификацией размерности dim и возвращает объединенный массив; $\text{dim}=1$ – горизонтальная конкатенация, $\text{dim}=2$ – вертикальная, $\text{dim}=3$ – многомерный массив размерности 3 и т. д.;
- $C = \text{cat}(\text{dim}, A1, A2, A3, A4, \dots)$ объединяет все входные массивы ($A1, A2, A3, A4$ и т. д.) в соответствии со спецификацией размерности dim и возвращает объединенный массив;

- `cat(2, A, B)` – это то же самое, что и `[A, B]`, а `cat(1, A, B)` – то же самое, что и `[A; B]`. При записи `cat(dim, C(:))` или `cat(dim, C.field)` эта функция применима к массивам ячеек или структур, содержащим численные матрицы. Пример:

```
>> A = [2, 4; 3, 5]; B = [8, 7; 9, 0]; C = cat(1, A, B)
C =
     2     4
     3     5
     8     7
     9     0
```

4.2.2. Создание матриц с заданной диагональю

Свойства матриц сильно зависят от их *диагональных элементов*, то есть элементов, расположенных на той или иной диагонали матриц. Следующая функция MATLAB позволяет создавать специальные типы матриц с заданными диагональными элементами:

- `X = diag(v, k)` – для вектора v , состоящего из n компонентов, возвращает квадратную матрицу X порядка $n + \text{abs}(k)$ с элементами v на k -ой диагонали, при $k=0$ это главная диагональ (из левого верхнего угла матрицы в правый нижний угол), при $k>0$ – одна из диагоналей (диагональ в терминологии MATLAB – это линия, параллельная главной диагонали) выше главной диагонали, при $k<0$ – одна из нижних диагоналей. Остальные элементы матрицы – нули;
- `X = diag(v)` – помещает вектор v на главную диагональ (то же, что и в предыдущем случае при $k=0$);
- `v = diag(X, k)` – для матрицы X возвращает вектор-столбец, состоящий из элементов k -ой диагонали матрицы X ;
- `v = diag(X)` – возвращает главную диагональ матрицы X (то же, что и в предыдущем случае при $k=0$).

Примеры:

```
>> v = [2, 3]; X = diag(v, 2)
X =
     0     0     2     0
     0     0     0     3
     0     0     0     0
     0     0     0     0

>> X = [2, 5, 45, 6; 3, 5, 4, 9; 7, 9, 4, 8; 5, 66, 45, 2]; v = diag(X, 0)
v =
     2
     5
     4
     2
```

4.2.3. Перестановки элементов матриц

Для перестановок элементов матриц служат следующие функции:

- $B = \text{fliplr}(A)$ – зеркально переставляет столбцы матрицы A относительно вертикальной оси. Пример:

```
>> F=[1,2,3;5,45,3]
F =
     1     2     3
     5    45     3
>> fliplr(F)
ans =
     3     2     1
     3    45     5
```

- $B = \text{flipud}(A)$ – зеркально переставляет строки матрицы A относительно горизонтальной оси. Пример:

```
>> F=[3,2,12;6,3,2]
F =
     3     2    12
     6     3     2
>> flipud(F)
ans =
     6     3     2
     3     2    12
```

- $\text{perms}(v)$ – возвращает матрицу P , которая содержит все возможные перестановки элементов вектора v , каждая перестановка в отдельной строке. Матрица P содержит $n!$ строк и n столбцов. Пример:

```
>> v=[1 4 6]
v =
     1     4     6
>> P=perms(v)
P =
     6     4     1
     4     6     1
     6     1     4
     1     6     4
     4     1     6
     1     4     6
```

4.2.4. Вычисление произведений

Несколько простых функций служат для перемножения элементов массивов:

- $\text{prod}(A)$ – возвращает произведение элементов массива, если A – вектор, или вектор-строку, содержащую произведения элементов каждого столбца, если A – матрица;
- $\text{prod}(A, \text{dim})$ – возвращает матрицу (массив размерности два) с произведением элементов массива A по столбцам ($\text{dim}=1$), по строкам ($\text{dim}=2$), по иным размерностям в зависимости от значения скаляра dim .

Пример:

```
>> A=[1 2 3 4; 2 4 5 7; 6 8 3 4]
A =
     1     2     3     4
     2     4     5     7
     6     8     3     4
>> B=prod(A)
B =    12    64    45   112
```

- `cumprod(A)` – возвращает произведение с накоплением. Если A – вектор, `cumprod(A)` возвращает вектор, содержащий произведения с накоплением элементов вектора A . Если A – матрица, `cumprod(A)` возвращает матрицу того же размера, что и A , содержащую произведения с накоплением для каждого столбца матрицы A (первая строка без изменений, во второй строке – произведение первых двух элементов каждого столбца, в третьей – элементы второй строки матрицы-результата умножаются на элементы третьей строки матрицы входного аргумента по столбцам и т. д.);
- `cumprod(A, dim)` – возвращает произведение с накоплением элементов по строкам или столбцам матрицы в зависимости от значения скаляра `dim`. Например, `cumprod(A, 1)` дает прирост первому индексу (номеру строки), таким образом выполняя умножение по столбцам матрицы A .

Примеры:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> B = cumprod(A)
B =
     1     2     3
     4    10    18
    28    80   162
B = cumprod(A, 1)
B =
     1     2     3
     4    10    18
    28    80   162
```

- `cross(U, V)` – возвращает векторное произведение векторов U и V в трехмерном пространстве, то есть $\mathbf{W}=\mathbf{U}\times\mathbf{V}$. U и V – обязательно векторы с тремя элементами;
- `cross(U, V, dim)` – возвращает векторное произведение U и V по размерности, определенной скаляром `dim`. U и V – многомерные массивы, которые должны иметь одну и ту же размерность, причем размер векторов в каждой размерности `size(U, dim)` и `size(V, dim)` должен быть равен 3.

Пример:

```
>> a = [6 5 3]; b = [1 7 6]; c = cross(a,b)
c =     9    -33    37
```

- `kron(X, Y)` – вычисления тензорного произведения Кронекера для матриц X и Y .

Пример:

```
>> X=[1 2; 3 4]
X =
     1     2
     3     4
>> Y=eye(2)
Y =
     1     0
     0     1
>> kron(X,Y)
ans =
     1     0     2     0
     0     1     0     2
     3     0     4     0
     0     3     0     4
>> kron(Y,X)
ans =
     1     2     0     0
     3     4     0     0
     0     0     1     2
     0     0     3     4
```

4.2.5. Суммирование элементов массивов

Определены следующие функции суммирования элементов массивов:

- `sum(A)` – возвращает сумму элементов массива, если A – вектор, или вектор-строку, содержащую сумму элементов каждого столбца, если A – матрица;
- `sum(A, dim)` – возвращает сумму элементов массива по столбцам ($dim=1$), строкам ($dim=2$) или иным размерностям, в зависимости от значения скаляра dim .

Пример:

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> B = sum(A)
B =    34    34    34    34
```

- `cumsum(A)` – выполняет суммирование с накоплением. Если A – вектор, `cumsum(A)` возвращает вектор, содержащий результаты суммирования с накоплением элементов вектора A . Если A – матрица, `cumsum(A)` возвращает матрицу того же размера, что и A , содержащую суммирование с накоплением для каждого столбца матрицы A ;

- `cumsum(A, dim)` – выполняет суммирование с накоплением элементов по размерности, определенной скаляром `dim`. Например, `cumsum(A, 1)` выполняет суммирование по столбцам.

Пример:

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> B = cumsum(A)
B =
    16     2     3    13
    21    13    13    21
    30    20    19    33
    34    34    34    34
```

4.2.6. Функции формирования матриц

Для создания матриц, состоящих из других матриц, служат следующие функции:

- `repmat(A, m, n)` – возвращает матрицу `B`, состоящую из $m \times n$ копий матрицы `A` (то есть в матрице $m \times n$ каждый элемент заменяется на копию матрицы `A`);
- `repmat(A, n)` – формирует матрицу, состоящую из $n \times n$ копий матрицы `A`;
- `repmat(A, [m n])` – дает тот же результат, что и `repmat(A, m, n)`;
- `repmat(A, [m n p...])` – возвращает многомерный массив ($m \times n \times p \dots$), состоящий из копий многомерного массива или матрицы `A`;
- `repmat(A, m, n)` – когда `A` – скаляр, возвращает матрицу размера $m \times n$ со значениями элементов, заданных `A`. Это делается намного быстрее, чем `A*ones(m, n)`.

Пример:

```
F =
     3     2
    43    32

>> repmat(F, 2, 3)
ans =
     3     2     3     2     3     2
    43    32    43    32    43    32
     3     2     3     2     3     2
    43    32    43    32    43    32
```

- `reshape(A, m, n)` – возвращает матрицу `B` размерностью $m \times n$, сформированную из `A` путем последовательной выборки по столбцам. Если число элементов `A` не равно $m \times n$, то выдается сообщение об ошибке;
- `reshape(A, m, n, p, ...)` или `B = reshape(A, [m n p...])` – возвращает `N`-мерный массив с элементами из `A`, но имеющий размер $m \times n \times p \dots$. Произведение $m \times n \times p \dots$ должно быть равно значению `prod(size(A))`;

- `reshape(A, siz)` – возвращает N-мерный массив с элементами из A, но перестроенный к размеру, заданному с помощью вектора `siz`.

Пример:

```
>> F=[3,2,7,4;4,3,3,2;2,2,5,5]
F =
     3     2     7     4
     4     3     3     2
     2     2     5     5
>> reshape(F,2,6)
ans =
     3     2     3     7     5     2
     4     2     2     3     4     5
```

4.2.7. Поворот матриц

Следующая функция обеспечивает *поворот матрицы* (по расположению элементов):

- `rot90(A)` – осуществляет поворот матрицы A на 90° против часовой стрелки;
- `rot90(A, k)` – осуществляет поворот матрицы A на величину $90 * k$ градусов, где `k` – целое число.

Пример:

```
>> M=[3,2,7;3,3,2;1,1,1]
M =
     3     2     7
     3     3     2
     1     1     1
>> rot90(M)
ans =
     7     2     1
     2     3     1
     3     3     1
```

4.2.8. Выделение треугольных частей матриц

При выполнении ряда матричных вычислений возникает необходимость в выделении треугольных частей матриц. Следующие функции обеспечивают такое выделение:

- `tril(X)` – возвращает матрицу, все элементы которой выше главной диагонали X заменены нулями, неизменными остаются лишь элементы нижней треугольной части, включая элементы главной диагонали;
- `tril(X, k)` – возвращает неизменной нижнюю треугольную часть матрицы X, начиная с `k`-ой диагонали. При `k=0` это главная диагональ, при `k>0` – одна из верхних диагоналей, при `k<0` – одна из нижних диагоналей.

Пример:

```
>> M=[3,1,4;8,3,2;8,1,1]
M =
     3     1     4
     8     3     2
     8     1     1
>> tril(M)
ans =
     3     0     0
     8     3     0
     8     1     1
```

- `triu(X)` – возвращает неизменной верхнюю треугольную часть матрицы X , включая элементы главной диагонали, и заменяет нулями остальные элементы;
- `triu(X, k)` – возвращает неизменной верхнюю треугольную часть матрицы X , начиная с k -ой диагонали. При $k=0$ это главная диагональ, при $k>0$ – одна из верхних диагоналей, при $k<0$ – одна из нижних диагоналей.

Пример:

```
M =
     3     1     4
     8     3     2
     8     1     1
>> triu(M)
ans =
     3     1     4
     0     3     2
     0     0     1
```

4.2.9. Операции с пустыми матрицами

Возможно создание *пустых матриц*, например:

```
>> M=[]
M =
 []
>> whos M
  Name      Size      Bytes      Class
  M         0x0         0          double array
Grand total is 0 elements using 0 bytes
```

Из этого примера видно, что пустая матрица создается с помощью пустых квадратных скобок. С помощью функций `zeros`, `ones`, `rand` или `eye` также можно создавать пустые матрицы заданного размера. Например:

```
>> E=zeros(0,5)
E = Empty matrix: 0-by-5
```

С пустыми матрицами можно выполнять некоторые операции, например:

```
>> sum(M)
ans = 0
```

```
>> prod(M)
ans = 1
>> max(M)
ans = []
>> min(M)
ans = []
```

4.3. Создание и вычисление специальных матриц

4.3.1. Сопровождающие матрицы

Начиная с этого раздела, рассматриваются функции, относящиеся к различным *специальным матрицам*. Они довольно широко используются при решении достаточно серьезных задач матричного исчисления. В связи с тем, что назначение соответствующих функций вытекает из их наименования, мы не будем сопровождать описание вводными комментариями. Соответствующие подробные определения вы найдете в книгах [44, 45, 49]. Рекомендуем читателю построить графики, представляющие элементы этих матриц.

`companion(u)` – возвращает сопровождающую матрицу, первая строка которой равна $-u(2:n)/u(1)$, где u – вектор полиномиальных коэффициентов. Собственные значения `companion(u)` – корни многочлена. Пример: для многочлена x^3+x^2-6x-8 вектор полиномиальных коэффициентов r имеет следующий вид:

```
>> r=[1,1,-6,-8]
r = 1 1 -6 -8
>> A=companion(r) % сопровождающая матрица
A =
    -1     6     8
     1     0     0
     0     1     0
>> eig(companion(r)) % корни многочлена
ans =
    -2.0000
     2.5616
    -1.5616
```

4.3.2. Тестовые матрицы

Для выполнения ряда вычислений в области линейной алгебры создан ряд специальных матриц, именуемых *тестовыми матрицами*. Такие матрицы создаются указанными ниже функциями.

`[A, B, C, ...] = gallery('tmfun', P1, P2, ...)` – возвращает тестовые матрицы, определенные строкой `tmfun`, где `tmfun` – это имя семейства матриц, вы-

бранное из списка. P1, P2, ... – входные параметры, требуемые для конкретного семейства матриц. Число используемых параметров P1, P2, ... изменяется от матрицы к матрице. Функция `gallery` хранит более 50 различных тестовых матричных функций, полезных для тестирования численных алгоритмов и других целей (включая матрицы Коши, Чебышева, фон Неймана, Чебышева–Вандермонде, Вандермонде, Уилкинсона и т. д.). Пример:

```
>> A=gallery('dramadah',5,2)
A =
     1     1     0     1     0
     0     1     1     0     1
     0     0     1     1     0
     0     0     0     1     1
     0     0     0     0     1
```

Команда

```
>> help gallery
```

выводит формат команды `gallery` и список входящих в галерею матриц. Наиболее полное собрание тестовых матриц содержится в пакете расширения Test Matrix Toolbox [25], разработанном на факультете математики Манчестерского университета (Великобритания) еще в 1995 г.

4.3.3. Матрицы Адамара

Функция `H = hadamard(n)` – возвращает *матрицу Адамара* порядка n . Это квадратная матрица размера n , составленная из значений 1 и -1 , столбцы которой ортогональны, так что справедливо соотношение $H' * H = n * I$, где $I = \text{eye}(n, n)$ (единичная квадратная матрица размера n). Матрицы Адамара применяются в различных областях, включая комбинаторику, численный анализ, обработку сигналов. Матрица Адамара размера $n \times n$ при $n > 2$ существует, только если n делится на 4 без остатка. Алгоритм MATLAB вносит дополнительные ограничения, вычисляя матрицы Адамара только для тех n , когда или n , или $n/12$, или $n/20$ являются степенями по основанию 2. Пример:

```
>> H = hadamard(4)
H =
     1     1     1     1
     1    -1     1    -1
     1     1    -1    -1
     1    -1    -1     1
```

4.3.4. Матрицы Ганкеля

Матрицы Ганкеля относятся к симметричным матрицам с постоянными значениями на антидиагоналях. Элементы матриц Ганкеля определяются выражением

$$\begin{cases} h(i, j) = p(i + j - 1) \\ \mathbf{p} = [\mathbf{c} \ \mathbf{r}(2:\text{end})] \end{cases}.$$

Векторы \mathbf{c} и \mathbf{r} могут задаваться произвольно, а вектор \mathbf{p} определяет матрицу. Определены следующие функции для матриц Ганкеля:

- `hankel(c)` – возвращает квадратную матрицу Ганкеля, первый столбец которой совпадает с вектором \mathbf{c} и все элементы, лежащие ниже первой антидиагонали (из левого нижнего угла матрицы в правый верхний угол), равны 0;
- `hankel(c, r)` – возвращает матрицу Ганкеля, первый столбец которой совпадает с вектором \mathbf{c} , а последняя строка – с вектором \mathbf{r} . Если последний элемент вектора \mathbf{c} отличен от первого элемента вектора \mathbf{r} , то выдается предупреждение об ошибке, но предпочтение отдается последнему элементу вектора \mathbf{c} .

Примеры:

```
>> c=1:4
c =     1         2         3         4
>> r=6:10
r =     6         7         8         9         10
>> H = hankel(c,r)
Warning: Column wins anti-diagonal conflict.
H =
     1     2     3     4     7
     2     3     4     7     8
     3     4     7     8     9
     4     7     8     9    10
```

4.3.5. Матрицы Гильберта

Функция `hilb(n)` возвращает матрицу Гильберта порядка n . Матрица Гильберта является примером плохо обусловленной матрицы. Элементы матрицы Гильберта определяются как $H(i, j) = 1 / (i + j - 1)$. Пример:

```
>> H = hilb(5)
H =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
>> cond(hilb(5))
ans = 4.7661e+005
```

Значение числа обусловленности матрицы Гильберта указывает на очень плохо обусловленную матрицу.

`invhilb(n)` – возвращает матрицу, обратную матрице Гильберта порядка n ($n < 15$). Для $n > 15$ функция `invhilb(n)` возвращает приближенную матрицу. Точная обратная матрица – это матрица с очень большими целочисленными значениями. Эти целочисленные значения могут быть представлены как числа с плавающей запятой без погрешности округления до тех пор, пока порядок матрицы n не превышает 15. Пример:

```
>>H = invhilb(5).
H =
    25    -300    1050    -1400     630
   -300    4800   -18900    26880   -12600
    1050    18900    79380   -117600    56700
   -1400    26880   -117600    179200   -88200
     630   -12600    56700   -88200    44100
```

А вот результат обращения матрицы Гильберта с плавающей запятой:

```
>> inv(hilb(5))
ans =
    1.0e+005 *
    0.0002  -0.0030  0.0105  -0.0140  0.0063
   -0.0030  0.0480  -0.1890  0.2688  -0.1260
    0.0105  -0.1890  0.7938  -1.1760  0.5670
   -0.0140  0.2688  -1.1760  1.7920  -0.8820
    0.0063  -0.1260  0.5670  -0.8820  0.4410
```

4.3.6. Матрицы магического квадрата

Функция `magic(n)` возвращает *магическую матрицу* размера $n \times n$, состоящую из целых чисел от 1 до n^2 , в которой суммы элементов по строкам, столбцам и главным диагоналям равны одному и тому же числу. Порядок матрицы должен быть больше или равен 3. Пример:

```
>> M=magic(4)
M =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

4.3.7. Матрицы Паскаля

Матрицы Паскаля – это симметрические положительно определенные квадратные матрицы порядка n , составленные из элементов треугольника Паскаля. Он составлен из коэффициентов разложения бинома $(1 + w)^k$, записанных для $k = 1, 2, 3, 4$ в виде:

$$\begin{array}{c}
 1 \\
 1+w \\
 1+2w+w^2 \\
 1+3w+3w^2+w^3 \\
 1+4w+6w^2+3w^3+w^4
 \end{array}$$

Определены следующие функции для задания матриц Паскаля:

- `pascal(n)` – возвращает матрицу Паскаля порядка n , то есть симметричную положительно определенную матрицу с целочисленными элементами, взятыми из треугольника Паскаля;

- `pascal (n, 1)` – возвращает нижний треугольный фактор (до знаков столбцов) Холецкого для матрицы Паскаля. Полученная матрица, все элементы которой выше главной диагонали равны нулю, является своей обратной матрицей, то есть квадратным корнем из единичной матрицы;
- `pascal (n, 2)` – возвращает матрицу, полученную в результате транспонирования и перестановок матрицы `pascal (n, 1)`, при этом результат является кубическим корнем из единичной матрицы.

Примеры:

```
>> A=pascal(4)
A =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20

>> A=pascal(4,2)
A =
     0     0     0    -1
     0     0    -1     3
     0     1     2    -3
     1     1     1    -1
```

4.3.8. Матрицы Россера

Матрицей Россера называют матрицу порядка 8 с целочисленными элементами, собственные значения которой имеют:

- пару кратных значений;
- 3 близких собственных значения;
- нулевое собственное значение;
- малое ненулевое собственное значение.

Многие алгоритмы вычисления собственных значений спотыкаются на этой матрице. Исключением является QR-алгоритм Франсиса, усовершенствованный Уилкинсоном, входящий в MATLAB.

Пример:

```
>> R=rosser
R =
    611    196   -192    407     -8   -52   -49    29
    196    899    113   -192    -71   -43    -8   -44
   -192    113    899    196     61    49     8    52
    407   -192    196    611     8    44    59   -23
     -8    -71     61     8    411  -599   208   208
   -52   -43     49    44  -599   411   208   208
   -49    -8     8    59   208   208    99  -911
     29   -44    52   -23   208   208  -911    99

>> eig(rosser)
ans =
```



```

1.0e+003 *
-1.0200
-0.0000
 0.0001
 1.0000
 1.0000
 1.0199
 1.0200
 1.0200

```

4.3.9. Матрицы Теплица

Для задания *матриц Теплица* заданы следующие функции:

- `toeplitz(c, r)` – возвращает несимметричную матрицу Теплица, где c – ее первый столбец, а r – первая строка. Если первый элемент столбца c и первый элемент строки r различны, то выдается соответствующее предупреждение, но отдается предпочтение элементу столбца;
- `toeplitz(r)` – возвращает симметричную, или эрмитову, матрицу Теплица, однозначно определяемую вектором r .

Пример:

```

>> c=1:3;
>> r=1.5:4.0;
>> T = toeplitz(c,r)
Warning: Column wins diagonal conflict.
T =
    1.0000    2.5000    3.5000
    2.0000    1.0000    2.5000
    3.0000    2.0000    1.0000

```

4.3.10. Матрица Вандермонда

Матрица Вандермонда `vander(x)` задает преобразование вектора x и создает матрицу, размер которой определяется размером вектора, а каждый элемент j -го столбца удовлетворяет выражению $V(:, j) = x^{(n-j)}$. Пример:

```

>> x=[1 3 5 7];
>> V=vander(x)
V =
     1     1     1     1
    27     9     3     1
   125    25     5     1
   343    49     7     1

```

Обратите внимание на то, что третий столбец аналогичен исходному вектору. Матрица Вандермонда используется при решении классической задачи полиномиальной аппроксимации.

4.3.11. Матрицы Уилкинсона

Функция `wilkinson(n)` возвращает одну из тестовых матриц Уилкинсона (другие матрицы Уилкинсона можно вызвать при помощи функции `gallery`). Это симметричная трехдиагональная матрица, собственные значения которой попарно близки, но не равны друг другу. Наиболее широко используется `wilkinson(21)`, собственные значения которой (10,746) совпадают до 14-го знака после запятой (различаются с 15-го). Пример:

```
W = wilkinson(5)
```

```
W =
     2     1     0     0     0
     1     1     1     0     0
     0     1     0     1     0
     0     0     1     1     1
     0     0     0     1     2
```

Полные данные о тестовых матрицах можно найти в справочной системе MATLAB.

4.4. Матричные операции линейной алгебры

Линейная алгебра – область, в которой наиболее часто используются векторы и матрицы. Наряду с операциями общего характера, рассмотренными выше, применяются функции, решающие наиболее характерные задачи линейной алгебры.

4.4.1. Матричные функции

Весьма представлен в MATLAB набор *матричных функций*, широко используемых в линейной алгебре. Они перечислены ниже.

- `expm(X)` – возвращает e^X от матрицы X . Комплексный результат получается, если X имеет неположительные собственные значения. Функция `expm` является встроенной и использует разложение Паде. Ее вариант в виде `m`-файла располагается в файле **expm1.m**. Второй метод вычисления матричной экспоненты использует разложение Тейлора и находится в файле **expm2.m**. Метод Тейлора не рекомендуется применять как основной, так как он зачастую бывает относительно медленным и неточным. Реализация третьего способа вычисления матричной экспоненты находится в файле **expm3.m** и использует спектральное разложение матрицы A . Этот метод неудачен, если входная матрица не имеет полного набора линейно независимых собственных векторов. Пример:

```
>> S=[1,0,3;1,3,1;4,0,0]
S =
```

```

      1     0     3
      1     3     1
      4     0     0
>> a=expm(S)
a =
      31.2203         0      23.3779
      38.9659      20.0855      30.0593
      31.1705         0      23.4277

```

- `funm(X,@function)`¹ – возвращает любую функцию от квадратной матрицы X , если правильно ввести имя, составленное из латинских букв. Команды `funm(X,@exp)` `funm(X,@sqrt)` `funm(X,@log)` и `expm(X)`, `sqrtm(x)`, `logm(X)` вычисляют соответственно одинаковые функции, но используют разные алгоритмы. Однако предпочтительнее использовать `expm(X)`, `sqrtm(x)`, `logm(X)`;
- `[Y,esterr]=funm(X,@function)` – не выдает никакого сообщения, но, помимо результата вычислений, в матрице Y возвращает грубую оценку относительной погрешности результата вычислений `funm` в `esterr`. Если матрица X – действительная симметричная или комплексная эрмитова, то ее форма Шура диагональна и полученный результат может иметь высокую точность.

Примеры:

```

>> S=[1,0,3;1,3,1;4,0,0]
S =
      1     0     3
      1     3     1
      4     0     0
>> a=funm(S,@exp)
a =
      31.2203      0.0000      23.3779
      38.9659      20.0855      30.0593
      31.1705     -0.0000      23.4277

```

- `logm(X)` – возвращает логарифм матрицы. Результат получается комплексным, если X имеет отрицательные собственные значения;
- `[Y,esterr]=logm(X)` – не выдает какого-либо предупреждающего сообщения, но возвращает оценку погрешности в виде относительной невязки `norm(expm(Y)-X)/norm(X)`.

Если матрица X – действительная симметричная или комплексная эрмитова, то теми же свойствами обладает и `logm(X)`. Пример:

```

a =
      31.2203         0.0000      23.3779
      38.9659      20.0855      30.0593
      31.1705     -0.0000      23.4277

```

¹ Форма `funm(X,'function')`, как в предыдущих версиях MATLAB, по-прежнему возможна, но не рекомендуется.

```
>> logm(a)
ans =
    1.0000    0.0000    3.0000
    1.0000    3.0000    1.0000
    4.0000   -0.0000   -0.0000
```

- `sqrtm(X)` – возвращает квадратный корень из X , соответствующий неотрицательным действительным частям собственных значений X . Результат получается комплексным, если X имеет отрицательные собственные значения. Если X вырожденная, то выдает предупреждение об ошибке;
- `[Y, resnorm]=sqrtm(X)` – не выдает какого-либо предупреждающего сообщения, но возвращает оценку погрешности в виде относительной невязки по нормам Фробениуса (см. урок 11) `norm(X-Y^2, 'fro')/norm(X, 'fro')`;
- `[Y, alpha, condest]=sqrtm(X)` – с тремя выходными аргументами функция, помимо квадратного корня, возвращает также фактор стабильности (но не невязку!) и оценку числа обусловленности результирующей матрицы Y .

Пример:

```
>> S=[2,1,0;6,7,-2;3,4,0];
>> e=sqrtm(S)
e =
    1.2586    0.2334    0.0688
    1.6066    2.7006   -0.6043
    0.5969    1.1055    0.7918
```

4.4.2. Вычисление нормы и чисел обусловленности матрицы

Для понимания всего нижеизложенного материала необходимо учесть, что *нормы матриц* в MATLAB отличаются от норм векторов. *Норма вектора* x (или, точнее, его p -норма) задается выражением

$$\|x\|_p = (\sum |x_i|^p)^{1/p}$$

и вычисляется функцией `norm(x, p)`.

Пусть A – матрица. Тогда `n=norm(A)` эквивалентно `n=norm(A, 2)` и возвращает вторую норму, то есть самое большое сингулярное число матрицы A . Функция `n=norm(A,1)` возвращает первую норму, то есть самую большую из сумм абсолютных значений элементов матрицы по столбцам. Норма неопределенности `n=norm(A,inf)` возвращает самую большую из сумм абсолютных значений элементов матрицы по рядам. Норма Фробениуса (Frobenius) `norm(A, 'fro') = sqrt(sum(diag(A'A)))`.

В общем случае p -норма матрицы A вычисляется как

$$\|A\|_p = \max_x \frac{\|Ax\|_p}{\|x\|_p}.$$

Для этого вычисления, при $p = 1, 2$ и inf , служит функция `norm(A, p)`. Пример:

```
>> A=[2, 3, 1; 1, 9, 4; 2, 6, 7]
```

```
A =
```

```
     2     3     1
     1     9     4
     2     6     7
```

```
>> norm(A, 1)
```

```
ans = 18
```

Числа обусловленности матрицы определяют чувствительность решения системы линейных уравнений к погрешностям исходных данных. Следующие функции позволяют найти числа обусловленности матриц.

- `cond(X)` – возвращает число обусловленности, основанное на второй норме, то есть отношение самого большого сингулярного числа X к самому малому. Значение `cond(X)`, близкое к 1, указывает на хорошо обусловленную матрицу;
- `c = cond(X, p)` – возвращает число обусловленности матрицы, основанное на p -норме: `norm(X, p) * norm(inv(X), p)`, где p определяет способ расчета:
 - $p=1$ – число обусловленности матрицы, основанное на первой норме;
 - $p=2$ – число обусловленности матрицы, основанное на второй норме;
 - $p='fro'$ – число обусловленности матрицы, основанное на норме Фробениуса (Frobenius);
 - $p=\text{inf}$ – число обусловленности матрицы, основанное на норме неопределенности;
- `c = cond(X)` – возвращает число обусловленности матрицы, основанное на второй норме. Пример:

```
>> d=cond(hilb(4))
d = 1.5514e+004
```

- `condeig(A)` – возвращает вектор чисел обусловленности для собственных значений A . Эти числа обусловленности – обратные величины косинусов углов между левыми и правыми собственными векторами;
- `[V, D, s] = condeig(A)` – эквивалентно `[V, D] = eig(A); s = condeig(A);`.

Большие числа обусловленности означают, что матрица A близка к матрице с кратными собственными значениями. Пример:

```
>> d=condeig(rand(4))
d =
    1.0766
    1.2298
    1.5862
    1.7540
```

- `rcond(A)` – оценивает обратную величину обусловленности матрицы A по первой норме, используя оценивающий обусловленность метод LAPACK. Если A – хорошо обусловленная матрица, то `rcond(A)` около 1.00, если плохо обусловленная, то около 0.00. По сравнению с `cond`

функция `rcond` реализует более эффективный в плане затрат машинного времени, но менее достоверный метод оценки обусловленности матрицы.

Пример:

```
>> s=rcond(hilb(4))
s = 4.6461e-005
```

4.4.3. Определитель и ранг матрицы

Для нахождения *определителя (детерминанта)* и ранга матриц в MATLAB имеются следующие функции:

- `det(X)` – возвращает определитель квадратной матрицы X . Если X содержит только целые элементы, то результат – тоже целое число. Использование `det(X)=0` как теста на вырожденность матрицы действительно только для матрицы малого порядка с целыми элементами. Пример:

```
>> A=[2,3,6;1,8,4;3,6,7]
A =
     2     3     6
     1     8     4
     3     6     7
>> det(A)
ans = -29
```

Детерминант матрицы вычисляется на основе треугольного разложения методом исключения Гаусса:

```
[L,U]=lu(A); s=det(L); d=s*prod(diag(U));
```

Ранг матрицы определяется количеством сингулярных чисел, превышающих порог `tol=max(size(A))*nprn(A)*eps`. При этом используется следующий алгоритм:

```
s=svd(A); tol=max(size(A))*nprn(A)*eps; r=sum(s>tol);
```

Для вычисления ранга используется функция `rank`:

- `rank(A)` – возвращает количество сингулярных чисел, которые являются большими, чем заданный по умолчанию допуск;
- `rank(A, tol)` – возвращает количество сингулярных чисел, которые превышают `tol`.

Пример:

```
>> rank(hilb(11))
ans = 10
```

4.4.4. Определение нормы вектора

Норма вектора – скаляр, дающий представление о величине элементов вектора. Функция `norm` определяет, является ли ее аргументом (входным аргументом в терминологии MATLAB) вектор или матрица, и измеряет несколько различных типов норм векторов:

- `norm(X) = norm(X, 2)` – вторая норма возвращает наибольшее сингулярное число X , `max(svd(X))`;
- `norm(X, p)`, где p – целое положительное число, возвращает корень степени p из суммы абсолютных значений элементов вектора, возведенных в степень p . При $p = 1$ это может совпадать либо с первой нормой, либо с нормой неопределенности матриц;
- `norm(X, 'inf')` возвращает максимальное из абсолютных значений элементов вектора;
- `norm(X, '-inf')` возвращает минимальное из абсолютных значений элементов вектора.

4.4.5. Определение ортонормированного базиса матрицы

Вычисление *ортонормированного базиса матрицы* обеспечивают следующие функции:

- `V = orth(A)` – возвращает ортонормированный базис матрицы A . Столбцы V определяют то же пространство, что и столбцы матрицы A , но столбцы V ортогональны, то есть $V' * V = \text{eye}(\text{rank}(A))$. Количество столбцов матрицы V равно рангу матрицы A . Пример:

```
>> A=[2 4 6;9 8 2;12 23 43]
```

```
A =
```

```
     2     4     6
     9     8     2
    12    23    43
```

```
>> B=orth(A)
```

```
B =
```

```
  0.1453  -0.0414  -0.9885
  0.1522  -0.9863   0.0637
  0.9776   0.1597   0.1371
```

- `null(A)` возвращает ортонормированный базис для нулевого (пустого) пространства A . Пример:

```
>> null(hilb(11))
```

```
ans =
```

```
  0.0000
 -0.0000
  0.0009
 -0.0099
  0.0593
 -0.2101
  0.4606
 -0.6318
  0.5276
 -0.2453
  0.0487
```

4.4.6. Функции приведения матрицы к треугольной форме

Треугольной матрицей называется квадратная матрица A , если при $l > k$ (верхняя треугольная матрица) или при $k > l$ (нижняя треугольная матрица) следует, что элементы матрицы $A(l, k)$ равны нулю. В строго треугольной матрице нули находятся и на главной диагонали. В линейной алгебре часто используется приведение матриц к той или иной треугольной форме. Оно реализуется следующими функциями:

- `rref(A)` возвращает приведенную к треугольной форме матрицу, используя метод исключения Гаусса с частичным выбором ведущего элемента. По умолчанию принимается значение порога допустимости для незначительного элемента столбца матрицы, равное $(\max(\text{size}(A)) * \text{eps} * \text{norm}(A, \text{inf}))$;
- `[R, jrb] = rref(A)` также возвращает вектор `jrb`, так что:
 - `r = length(jrb)` может служить оценкой ранга матрицы A ;
 - `x(jrb)` – связанные переменные в системе линейных уравнений вида $Ax=b$;
 - `A(:, jrb)` – базис матрицы A ;
 - `R(1:r, jrb)` – единичная матрица размера $r \times r$;
- `[R, jrb] = rref(A, tol)` осуществляет приведение матрицы к треугольной форме, используя метод исключения Гаусса с частичным выбором ведущего элемента для заданного значения порога допустимости `tol`;
- `rrefmovie(A)` показывает пошаговое исполнение процедуры приведения матрицы к треугольной.

Примеры:

```
>> s=magic(3)
s =
     8     1     6
     3     5     7
     4     9     2

>> rref(s)
ans =
     1     0     0
     0     1     0
     0     0     1
```

4.4.7. Определение угла между двумя подпространствами

Угол между двумя подпространствами вычисляет функция `theta = subspace(A, B)`. Она возвращает угол между двумя подпространствами, натянутыми на столбцы матриц A и B . Если A и B – векторы-столбцы единичной длины, то угол вычисляется по формуле $\arccos(A' * B)$. Если некоторый физический эксперимент описыва-

ется массивом A , а вторая реализация этого эксперимента – массивом B , то $\text{subspace}(A, B)$ измеряет количество новой информации, полученной из второго эксперимента и не связанной со случайными ошибками и флуктуациями.

Пример:

```
>> H = hadamard(20); A = H(:, 2:4); B = H(:, 5:8);
>> subspace(A, B)
ans = 1.5708
```

4.4.8. Вычисление следа матрицы

След матрицы A – это сумма ее диагональных элементов. Он вычисляется функцией $\text{trace}(A)$. Пример:

```
>> a=[2, 3, 4; 5, 6, 7; 8, 9, 1]
a =
     2     3     4
     5     6     7
     8     9     1
>> trace(a)
ans = 9
```

4.4.9. Разложение Холецкого

Разложение Холецкого – известный прием матричных вычислений. Функция chol находит это разложение для действительных симметричных и комплексных эрмитовых матриц.

- $R = \text{chol}(X)$ для квадратной матрицы¹ X возвращает верхнюю треугольную матрицу R , так что $R' * R = X_{\text{new}}$. Если симметрическая матрица X_{new} , заданная верхней треугольной частью и диагональю матрицы X , не является положительно определенной матрицей, выдается сообщение об ошибке. Разложение Холецкого возможно для действительных и комплексных эрмитовых матриц²;
- $[R, p] = \text{chol}(X)$ с двумя выходными аргументами никогда не генерирует сообщение об ошибке в ходе выполнения разложения Холецкого квадратной матрицы X . Если верхняя треугольная часть и диагональ X задают положительно определенную матрицу, то $p=0$, а R совпадает с вышеописанным случаем, иначе p – положительное целое число, а R – верхняя треугольная матрица порядка $q=p-1$, такая что $R' * R = X(1:q, 1:q)$.

Пример:

¹ Положительно определенной называется действительная симметрическая матрица, все собственные значения которой положительны. Поскольку используется только верхний треугольник матрицы X , матрица X не обязательно должна быть симметрической.

² Квадратная матрица с комплексными элементами, комплексно сопряженная матрица которой может быть получена транспонированием, то есть равна транспонированной матрице ($A^* = A'$).

```
>> c=chol(pascal(4))
c =
     1     1     1     1
     0     1     2     3
     0     0     1     3
     0     0     0     1
```

4.4.10. Обращение матриц – функции *inv*, *pinv*

Обращение матриц – одна из наиболее распространенных операций матричного анализа. *Обратной* называют матрицу, получаемую в результате деления единичной матрицы E на исходную матрицу X . Таким образом, $X^{-1}=E/X$. Следующие функции обеспечивают реализацию данной операции:

- `inv(X)` возвращает матрицу, обратную квадратной матрице X . Предупреждающее сообщение выдается, если X плохо масштабирована или близка к вырожденной. Пример:

```
>> inv(rand(4,4))
ans =
     2.2631    -2.3495    -0.4696    -0.6631
    -0.7620     1.2122     1.7041    -1.2146
    -2.0408     1.4228     1.5538     1.3730
     1.3075    -0.0183    -2.5483     0.6344
```

На практике вычисление явной обратной матрицы не так уж необходимо. Чаще операцию обращения применяют при решении системы линейных уравнений вида $Ax=b$. Один из путей решения этой системы – вычисление $x=inv(A)*b$. Но лучшим с точки зрения минимизации времени расчета и повышения точности вычислений является использование оператора матричного деления $x=A\b b$. Эта операция использует метод исключения Гаусса без явного формирования обратной матрицы;

- `B=pinv(A)` возвращает матрицу, псевдообратную матрице A (псевдообращение матрицы по Муру-Пенроузу). Результатом псевдообращения матрицы по Муру-Пенроузу является матрица B того же размера, что и A' , и удовлетворяющая условиям $A*B*A=A$ и $B*A*B=B$. Вычисление основано на использовании функции `svd(A)` и приравнивании к нулю всех сингулярных чисел, меньших величины `tol`;
- `B=pinv(A,tol)` возвращает псевдообратную матрицу и отменяет заданный по умолчанию порог, равный `max(size(A))*norm(A)*eps`.

Пример:

```
>> pinv(rand(4,3))
ans =
    -1.3907    -0.0485    -0.2493     1.8640
    -0.8775     1.1636     0.6605    -0.0034
     2.0906    -0.5921    -0.2749    -0.5987
```

Обращение матриц, как и другие матричные операции, является ноу-хау разработчика системы MATLAB. Поэтому полного описания алгоритмов матричных

описаний в обычной документации нет. К тому же используется ряд алгоритмов – в зависимости от свойств исходной матрицы.

4.4.11. LU- и QR-разложение

Так называемые *LU*- и *QR*-разложения реализуются следующими матричными функциями. Функция *lu* выражает любую квадратную матрицу X как произведение двух треугольных матриц, одна из которых (возможно, с перестановками) – нижняя треугольная матрица, а другая – верхняя треугольная матрица. Иногда эту операцию называют *LR-разложением*. Для выполнения этой операции служит следующая функция:

- $[L, U] = \text{lu}(X)$ возвращает верхнюю треугольную матрицу U и психологическую нижнюю матрицу L (то есть произведение нижней треугольной матрицы и матрицы перестановок), так что $X=L*U$;
 - $[L, U, P] = \text{lu}(X)$ возвращает верхнюю треугольную матрицу U , нижнюю треугольную матрицу L и сопряженную (эрмитову) матрицу перестановок P , так что $L*U=P*X$;
 - $\text{lu}(X)$ – вызванная с одним выходным параметром, функция возвращает результат из подпрограмм DGETRF (для действительных матриц) или ZGETRF (для комплексных) известного пакета программ линейной алгебры LAPACK;
 - $\text{lu}(X, \text{thresh})$, где thresh в диапазоне $[0...1]$, управляет центрированием в разреженных матрицах (см. урок 12). Отдельная форма предыдущего случая. Центрирование происходит, если элемент столбца на диагонали меньше, чем произведение thresh и любого поддиагонального элемента. $\text{Thresh}=1$ – значение по умолчанию. $\text{Thresh}=0$ задает центрирование по диагонали. Если матрица неразреженная, выводится сообщение об ошибке.
- Пример:

```
>> f=[3,5,4;12,7,5;34,65,23]
f =
     3     5     4
    12     7     5
    34    65    23

>> [d,h]=lu(f)
d =
    0.0882    0.0461    1.0000
    0.3529    1.0000     0
    1.0000     0         0

h =
    34.0000    65.0000    23.0000
     0    -15.9412    -3.1176
     0     0         2.1144

>> d*h
ans =
    3.0000    5.0000    4.0000
```

| | | |
|---------|---------|---------|
| 12.0000 | 7.0000 | 5.0000 |
| 34.0000 | 65.0000 | 23.0000 |

qr-функция выполняет *QR-разложение* матрицы. Эта операция полезна для квадратных и треугольных матриц. Она выполняет QR-разложение, вычисляя произведение унитарной матрицы и верхней треугольной матрицы. Функция используется в следующих формах:

- $[Q, R] = \text{qr}(X)$ вычисляет верхнюю треугольную матрицу R того же размера, как и у X , и унитарную матрицу Q , так что $X=Q^*R$;
- $[Q, R, E] = \text{qr}(X)$ вычисляет матрицу перестановок E , верхнюю треугольную матрицу R с убывающими по модулю диагональными элементами и унитарную матрицу Q , так что $X^*E=Q^*R$. Матрица перестановок E выбрана так, что $\text{abs}(\text{diag}(R))$ уменьшается;
- $[Q, R] = \text{qr}(X, 0)$ и $[Q, R, E] = \text{qr}(X, 0)$ вычисляют *экономное* разложение, в котором E – вектор перестановок, так что $Q^*R=X(:, E)$. Матрица E выбрана так, что $\text{abs}(\text{diag}(R))$ уменьшается;
- $A = \text{qr}(X)$ возвращает результат из LAPACK.

Пример:

```
>> C=rand(5, 4)
```

```
C =
```

| | | | |
|--------|--------|--------|--------|
| 0.8381 | 0.5028 | 0.1934 | 0.6979 |
| 0.0196 | 0.7095 | 0.6822 | 0.3784 |
| 0.6813 | 0.4289 | 0.3028 | 0.8600 |
| 0.3795 | 0.3046 | 0.5417 | 0.8537 |
| 0.8318 | 0.1897 | 0.1509 | 0.5936 |

```
>> [Q,R]=qr(C)
```

```
Q =
```

| | | | | |
|---------|---------|---------|---------|---------|
| -0.5922 | -0.1114 | 0.5197 | 0.0743 | -0.6011 |
| -0.0139 | -0.9278 | -0.0011 | -0.3448 | 0.1420 |
| -0.4814 | -0.1173 | 0.0699 | 0.5940 | 0.6299 |
| -0.2681 | -0.1525 | -0.8268 | 0.2632 | -0.3898 |
| -0.5877 | 0.2997 | -0.2036 | -0.6734 | 0.2643 |

```
R =
```

| | | | |
|---------|---------|---------|---------|
| -1.4152 | -0.7072 | -0.5037 | -1.4103 |
| 0 | -0.7541 | -0.7274 | -0.4819 |
| 0 | 0 | -0.3577 | -0.4043 |
| 0 | 0 | 0 | 0.2573 |
| 0 | 0 | 0 | 0 |

- $[Q, R] = \text{qrdelete}(Q, R, j)$ изменяет Q и R таким образом, чтобы пересчитать QR-разложение матрицы A для случая, когда в ней удален j -ый столбец ($A(:, j)=[]$). Входные значения Q и R представляют QR-разложение матрицы A как результат действия $[Q, R] = \text{qr}(A)$. Аргумент j определяет столбец, который должен быть удален из матрицы A . Примеры:

```
>> C=rand(3, 3)
```

```
C =
```

| | | |
|--------|--------|--------|
| 0.0164 | 0.0576 | 0.7176 |
|--------|--------|--------|

```

      0.1901      0.3676      0.6927
      0.5869      0.6315      0.0841
>> [Q,R]=qr(C)
Q =
     -0.0265     -0.2416     -0.9700
     -0.3080     -0.9212      0.2378
     -0.9510      0.3051     -0.0500
R =
     -0.6171     -0.7153     -0.3123
         0      -0.1599     -0.7858
         0         0      -0.5356
>> [Q1,R1]=qrdelete(Q,R,2)
Q1 =
     -0.0265      0.7459      0.6655
     -0.3080      0.6272     -0.7153
     -0.9510     -0.2239      0.2131
R1 =
     -0.6171     -0.3123
         0      0.9510
         0         0

```

- `[Q,R]=qrinsert(Q,R,j,x)` изменяет Q и R таким образом, чтобы пересчитать разложение матрицы A для случая, когда в матрице A перед j -ым столбцом вставлен столбец x . Входные значения Q и R представляют QR-разложение матрицы A как результат действия `[Q,R]=qr(A)`. Аргумент x – вектор-столбец, который нужно вставить в матрицу A . Аргумент j определяет столбец, перед которым будет вставлен вектор x . Примеры:

```

>> C=rand(3,3)
C =
      0.1210      0.8928      0.8656
      0.4508      0.2731      0.2324
      0.7159      0.2548      0.8049
>> [Q,R]=qr(C)
Q =
     -0.1416      0.9835      0.1126
     -0.5275      0.0213     -0.8493
     -0.8377     -0.1797      0.5157
R =
     -0.8546     -0.4839     -0.9194
         0      0.8381      0.7116
         0         0      0.3152
>> x=[0.5,-0.3,0.2]; [Q2,R2]=qrinsert(Q,R,2,x')
Q2 =
     -0.1416      0.7995     -0.5838
     -0.5275     -0.5600     -0.6389
     -0.8377      0.2174      0.5010
R2 =
     -0.8546     -0.0801     -0.4839     -0.9194
         0      0.6112      0.6163      0.7369
         0         0     -0.5681     -0.2505

```

4.4.12. Вычисление собственных значений и сингулярных чисел

Во многих областях математики и прикладных наук большое значение имеют средства для вычисления *собственных значений* (собственных чисел, характеристических чисел, решений векового уравнения) матриц, принадлежащих им векторов и сингулярных чисел. В новой версии MATLAB собственные вектора нормализуются иначе, чем в предыдущих. Основной критерий: либо $V'V=I$, либо $V'BV=I$, где V – собственный вектор, I – единичная матрица. Поэтому результаты вычислений в новой версии, как правило, отличаются от результатов старых версий MATLAB.

Несимметрические матрицы могут быть плохо обусловлены при вычислении их собственных значений. Малые изменения элементов матрицы, такие как ошибки округления, могут вызвать большие изменения в собственных значениях. Масштабирование, хотя и не превращает их в симметрические, значительно повышает стабильность собственных значений. *Масштабирование* – это попытка перевести каждую плохую обусловленность собственных векторов матрицы в диагональное масштабирование. Однако масштабирование обычно не может преобразовать несимметрическую матрицу в симметрическую, а только пытается сделать (векторную) норму каждой строки равной норме соответствующего столбца. Масштабирование значительно повышает стабильность собственных значений.

- $[D, B] = \text{balance}(A)$ возвращает диагональную матрицу D , элементы которой являются степенями основания 2, и масштабированную матрицу B , такую, что $B=D \setminus A * D$, а норма каждого ряда масштабированной матрицы приближается к норме столбца с тем же номером;
- $B = \text{balance}(A)$ – возвращает масштабированную матрицу B .

Пример использования функции `balance`:

```
>> A=[1 1000 10000;0.0001 1 1000;0.000001 0.0001 1]
A =
    1.0e+004 *
         0.0001         0.1000         1.0000
         0.0000         0.0001         0.1000
         0.0000         0.0000         0.0001
>> [F,G]=balance(A)
F =
    1.0e+004 *
         3.2768         0         0
         0         0.0032         0
         0         0         0.0000
G =
         1.0000         0.9766         0.0095
         0.1024         1.0000         0.9766
         1.0486         0.1024         1.0000
```

Величина, связывающая погрешность вычисления собственных значений с погрешностью исходных данных, называется *числом обусловленности* (собственных значений) матрицы и вычисляется следующим образом:

$\text{cond}(V) = \text{norm}(V) * \text{norm}(\text{inv}(V))$

где $[V, D] = \text{eig}(A)$;

- $\text{eig}(A)$ возвращает вектор собственных значений квадратной полной или симметрической разреженной матрицы A , обычно после автоматического масштабирования, но для больших разреженных матриц (в терминологии MATLAB это просто полные матрицы со сравнительно небольшим числом нулей), а также во всех случаях, где помимо собственных значений необходимо получать и собственные вектора разреженной матрицы, вместо нее рекомендовано использовать $\text{eigs}(A)$;
- $\text{eig}(A, B)$ возвращает вектор обобщенных собственных значений квадратных матриц A и B ;
- $[V, D] = \text{eig}(A, B)$ вычисляет диагональную матрицу обобщенных собственных значений D и матрицу V , столбцы которой являются соответствующими собственными векторами (правыми собственными векторами), таким образом, что $A V = B V D$;
- $[V, D] = \text{eig}(A)$ вычисляет диагональную матрицу собственных значений D матрицы A и матрицу V , столбцы которой являются соответствующими собственными векторами (правыми собственными векторами), таким образом, что $A V = V D$.
Нужно использовать $[W, D] = \text{eig}(A')$; $\bar{W} = W'$, чтобы вычислить *левые* собственные вектора, которые соответствуют уравнению $\bar{W} * A = D * \bar{W}$;
- $[V, D] = \text{eig}(A, 'nobalance')$ находит собственные векторы и собственные значения без предварительного масштабирования. Иногда это улучшает обусловленность входной матрицы, обеспечивая большую точность вычисления собственных векторов для необычно масштабированных матриц;
- $\text{eig}(A, B, 'chol')$ возвращает вектор, содержащий обобщенные собственные значения, используя разложение матрицы B по методу Холецкого; если A – симметрическая квадратная матрица и B – симметрическая положительно определенная квадратная матрица, то $\text{eig}(A, B)$ по умолчанию работает точно так же;
- $\text{eig}(A, B, 'qz')$ не требует, чтобы матрицы были симметрическими, и возвращает вектор, содержащий обобщенные собственные значения, используя QZ-алгоритм; при явном указании этого флага QZ-алгоритм используется вместо алгоритма Холецкого даже для симметрической матрицы и симметрической положительно определенной матрицы B , так как может давать более стабильные значения, чем предыдущий метод. Для несимметрических матриц в MATLAB 6 всегда используется QZ-алгоритм и параметр 'chol' или 'qz' игнорируется;
- $[V, D] = \text{eig}(A, B)$ возвращает диагональную матрицу обобщенных собственных значений D и матрицу V , чьи столбцы являются соответствующими собственными векторами, так чтобы $A * V = B * V * D$.

Пример:

```
>> B = [3 -12 -.6 2*eps; -2 48 -1 -eps; -eps/8 eps/2 -1 10; -.5
-.5 .3 1]
```

```

B =
    3.0000   -12.0000   -0.6000    0.0000
   -2.0000    48.0000   -1.0000   -0.0000
   -0.0000    0.0000   -1.0000   10.0000
   -0.5000   -0.5000    0.3000    1.0000

>> [G,H]=eig(B)
G =
   -0.2548    0.7420   -0.4842    0.1956
    0.9670    0.0193   -0.0388    0.0276
   -0.0015   -0.6181   -0.8575    0.9780
   -0.0075   -0.2588   -0.1694   -0.0676

H =
   48.5287         0         0         0
         0    3.1873         0         0
         0         0    0.9750         0
         0         0         0   -1.6909

```

- `svd(X)` возвращает вектор сингулярных чисел. Команда `svd` выполняет сингулярное разложение матрицы X ;
- `[U, S, V] = svd(X)` вычисляет диагональную матрицу S тех же размеров, которые имеет матрица X с неотрицательными диагональными элементами в порядке их убывания, и унитарные матрицы U и V , так что $X=U*S*V'$;
- `[U, S, V] = svd(X, 0)` выполняет экономичное сингулярное разложение.

Пример:

```

>> F=[23 12;3 5;6 0]
F =
    23    12
     3     5
     6     0

>> [k,l,m]=svd(F)
k =
    0.9628   -0.0034   -0.2702
    0.1846    0.7385    0.6485
    0.1974   -0.6743    0.7116

l =
   26.9448         0
         0    4.1202
         0         0

m =
    0.8863   -0.4630
    0.4630    0.8863

```

4.4.13. Приведение матриц к форме Шура и Хессенберга

Ниже приводятся функции, обеспечивающие приведение матриц к специальным формам Шура и Хессенберга:

- `cdf2rdf` – преобразование комплексной формы Шура в действительную. Если система $[V, D]=\text{eig}(X)$ имеет комплексные собственные значения, объединенные в комплексно-сопряженные пары, то функция `cdf2rdf` преобразует систему таким образом, что матрица D принимает вещественный диагональный вид с 2×2 вещественными блоками, заменяющими первоначальные комплексные пары. Конкретные столбцы матрицы V больше не являются собственными векторами, но каждая пара векторов связана с блоком размера 2×2 в матрице D .

Пример:

```
>> A=[2 3 6;-4 0 3;1 5 -2]
A =
     2     3     6
    -4     0     3
     1     5    -2

>> [S,D]=eig(A)
S =
    0.7081 + 0.3296i    0.7081 - 0.3296i   -0.3355
   -0.3456 + 0.3688i   -0.3456 - 0.3688i   -0.5721
    0.0837 + 0.3571i    0.0837 - 0.3571i    0.7484

D =
    3.1351 + 4.0603i    0                0
     0                3.1351 - 4.0603i    0
     0                0                -6.2702

>> [S,D]=cdf2rdf(S,D)
S =
    0.7081    0.3296   -0.3355
   -0.3456    0.3688   -0.5721
    0.0837    0.3571    0.7484

D =
     3.1351    4.0603    0
    -4.0603    3.1351    0
     0         0       -6.2702
```

Функция `qz` обеспечивает приведение пары матриц к обобщенной форме Шура:

- $[AA, BB, Q, Z, V] = \text{qz}(A, B)$ возвращает, возможно, комплексные верхние треугольные матрицы AA и BB и соответствующие матрицы приведения Q и Z , такие, что $Q^*A^*Z=AA$ и $Q^*B^*Z=BB$. Функция также возвращает матрицу обобщенных собственных векторов V .

Обобщенные собственные значения могут быть найдены из следующего условия:

$$A^*V^*\text{diag}(BB) = B^*V^*\text{diag}(AA)$$

Пример:

```
>> A=[1 2 3;6 3 0;4 7 0];B=[1 1 1;0 7 4;9 4 1];
>> [aa,bb,f,g,h]=qz(A,B)
aa =
   -2.9395    0.4775    0.8751
```

```

      0      9.5462      3.5985
      0      0      3.2073
bb =
      5.5356      3.5345     -2.2935
      0      8.4826      6.7128
      0      0      0.7667
f =
     -0.0367      0.7327     -0.6796
     -0.1052     -0.6791     -0.7265
     -0.9938      0.0448      0.1020
g =
     -0.7023     -0.7050     -0.0989
      0.6867     -0.6343     -0.3552
     -0.1877      0.3174     -0.9295
h =
     -1.0000     -0.4874     -0.0561
      0.9778     -1.0000      0.6238
     -0.2673      0.4340     -1.0000

```

Функция `qz(A,B,'real')` при заданных матрицах A и B возвращает действительные треугольную матрицу VV и квазитреугольную матрицу AA с 2×2 диагональными блоками, соответствующими парам сопряженных комплексных значений. Так как матрица AA квазитреугольная, то необходимо решить проблему обобщения 2×2 для получения подлинных собственных значений. Пример:

```

>> A=[1 2 3;6 3 0;4 7 0];B=[1 1 1;0 7 4;9 4 1];
>> [aa,bb,f,g,h]=qz(A,B,'real')
aa =
     -2.9395      0.4775      0.8751
      0      9.5462      3.5985
      0      0      3.2073
bb =
      5.5356      3.5345     -2.2935
      0      8.4826      6.7128
      0      0      0.7667
f =
     -0.0367      0.7327     -0.6796
     -0.1052     -0.6791     -0.7265
     -0.9938      0.0448      0.1020
g =
     -0.7023     -0.7050     -0.0989
      0.6867     -0.6343     -0.3552
     -0.1877      0.3174     -0.9295
h =
     -1.0000     -0.4874     -0.0561
      0.9778     -1.0000      0.6238
     -0.2673      0.4340     -1.0000

```

- $T = \text{schur}(A)$ возвращает матрицу Шура T ;
- $[U, T] = \text{schur}(A)$ возвращает матрицу Шура T и унитарную матрицу U , такие, что $A = U T U'$ и $U' U = \text{eye}(\text{size}(A))$ (единичная матрица размера A);

- $[U, T] = \text{rsf2csf}(u, t)$ – преобразование результатов предыдущей функции (действительной формы Шура) в комплексную форму Шура, может использоваться только после вызова $[u, t] = \text{schur}(A)$. Комплексная форма Шура – это верхняя треугольная матрица со всеми собственными значениями на диагонали. Действительная форма Шура имеет действительные собственные значения на диагонали, а комплексные собственные значения содержатся в 2×2 блоках, расположенных вдоль диагонали. И входные, и выходные матрицы U, u и T, t представляют собой соответственно унитарные матрицы и матрицы Шура исходной матрицы A , которая удовлетворяет условиям $A=UTU'$ и $U'U=\text{eye}(\text{size}(A))$.

Примеры:

$A =$

```

1      1      1      1
-3     1     -4     1
1      0     -5     1
-1     2      3      0

```

$\gg [u, t] = \text{schur}(A)$

$u =$

```

-0.4883    -0.6416    -0.5757     0.1362
-0.5289     0.7465    -0.3986    -0.0646
-0.1403    -0.1528     0.0583    -0.9765
-0.6798    -0.0884     0.7115     0.1540

```

$t =$

```

1.2036    -2.7670    -0.8023    -0.0842
1.9478     2.3183     1.5080     2.6513
0          0          -0.6449    -2.9694
0          0          0.0000    -5.8771

```

$\gg [U, T] = \text{rsf2csf}(u, t)$

$U =$

```

-0.3226 - 0.3631i  0.4318 + 0.4771i  -0.5757  0.1362
 0.5771 - 0.3933i  0.2027 - 0.5551i  -0.3986  -0.0646
-0.0724 - 0.1044i  0.1183 + 0.1136i   0.0583  -0.9765
 0.0682 - 0.5056i  0.4532 + 0.0657i   0.7115  0.1540

```

$T =$

```

1.7610 + 2.2536i  0.5003 - 1.2897i  1.1168 + 0.5967i  1.7196 + 0.0626i
0               1.7610 - 2.2536i  0.2383 + 1.1215i  -0.4335 + 1.9717i
0               0               -0.6449          -2.9694
0               0               0               -5.8771

```

- $H = \text{hess}(A)$ находит H , верхнюю форму Хессенберга для матрицы A ;
- $[P, H] = \text{hess}(A)$ возвращает матрицу Хессенберга H и унитарную матрицу преобразований P , такую что $A=P*H*P'$ и $P'*P=\text{eye}(\text{size}(A))$.

Элементы матрицы Хессенберга, расположенные ниже первой поддиагонали, равны нулю. Если матрица симметричная или эрмитова, то матрица Хессенберга вырождается в трехдиагональную. Эта матрица имеет те же собственные значения, что и оригинал, но для их вычисления необходимо меньшее количество операций.

Пример:

$\gg f=\text{magic}(4)$

$f =$

```

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> hess(f)
ans =
    16.0000    -8.0577     8.8958     6.1595
   -11.0454    24.2131    -8.1984     2.1241
     0    -13.5058    -4.3894    -7.8918
     0         0     3.2744    -1.8237

```

4.5. О скорости выполнения матричных операций

4.5.1. О повышении скорости вычислений в старых версиях MATLAB

В литературе по устаревшим версиям системы MATLAB отмечаются факты резкого ускорения выполнения матричных операций при переходе от обычных арифметических операций с элементами матриц (например, +, -, *, /, ^) к «параллельным» поэлементным операциям (например, .+, .-, .*, ./, .^). Постараемся прояснить этот вопрос применительно к использованию новых версий системы MATLAB.

4.5.2. Ситуация со скоростью вычислений в MATLAB 7.*

В новых реализациях MATLAB благодаря применению ускорителя времени исполнения (JIT) такое увеличение стало мифическим. Покажем это на примере выполнения операции поэлементного возведения элемента матрицы A в степень элемента матрицы B, используя для этого следующие m-файлы (программы):

| | |
|---|--|
| <pre> Файл test1 A=rand(1000,1000); B=rand(1000,1000); for i=1:1000 for j=1:1000 A(i,j)=A(i,j)^B(i,j); end end </pre> | <pre> Файл test2 A=rand(1000,1000); B=rand(1000,1000); A=A.^B </pre> |
|---|--|

В этих примерах используются матрицы A и B со случайными значениями элементов, имеющие размер 1000×1000, то есть в примерах выполняется миллион операций возведения в степень. Для вычисления затрат процессорного времени можно использовать конструкцию `tic, ..., toc`, на место многоточия которой помещается команда или группа команд, время вычисления которых нужно измерить. Для наших примеров:

```
>> clear all
>> tic, test1, toc
Elapsed time is 1.062000 seconds.
>> clear all
>> tic, test2, toc
Elapsed time is 0.891000 seconds.
```

Результат в особых комментариях не нуждается – время вычисления для обоих примеров различается всего 16%.

Возьмем еще пару примеров – на этот раз на операцию деления каждого элемента квадратной матрицы A на величину $i*j$ во вложенном цикле:

Файл test3

```
A=rand(1000,1000);
for i=1:1000
    for j=1:1000
        A(i,j)=A(i,j)/(i*j);
    end
end
```

Файл test4

```
A=rand(1000,1000);
row=[1:1000]
A=A./(row'*row)
```

Результаты исполнения этих тестов оказались вообще одинаковыми:

```
>> clear all
>> tic, test3, toc
Elapsed time is 0.157000 seconds.
>> clear all
>> tic, test4, toc
Elapsed time is 0.157000 seconds.
```

Можно также отметить, что практически теряет значения и предварительное задание под большие матрицы памяти в ОЗУ. Все это говорит о тщательной оптимизации вычислений, как обычных, так и «параллельных» поэлементных в новых версиях MATLAB. В итоге главным улучшением новых реализаций MATLAB является уже не повышение скорости последних операций, а лишь их более удобная форма записи. Однако при работе с прежними версиями MATLAB ускорение «параллельных» поэлементных операций может быть куда более значительным.

Здесь уместно отметить, что в новых реализациях MATLAB (R2007a и особенно R2007b) серьезное внимание уделено поддержке 64-разрядных микропроцессоров, многопоточной работе, в частности, процессоров с Hyper Threading и многоядерных процессоров – пока в основном двухъ- и четырехъядерных (в них, кстати, технология Hyper Threading обычно не применяется). Распределение функций обслуживания графического интерфейса и вычислений по нескольким ядрам (или нескольким процессорам в многопроцессорных серверах) существенно повышает скорость вычисления матричных операций большой размерности и означает новые возможности в решении с помощью системы MATLAB сложных научно-технических задач большой размерности. В частности, MATLAB R2007* уже имеет средства для поддержания на уровне параллельных вычислений до четырех сессий одновременно.

Типы данных – массивы специального вида

| | |
|---|-----|
| 5.1. Разреженные матрицы | 234 |
| 5.2. Применение разреженных матриц | 245 |
| 5.3. Функции разреженных матриц | 250 |
| 5.4. Многомерные массивы | 255 |
| 5.5. Работа с размерностями массивов | 262 |
| 5.6. Тип данных – структуры | 264 |
| 5.7. Функции полей структур | 267 |
| 5.8. Массивы ячеек | 269 |
| 5.9. Многомерные массивы ячеек | 274 |

MATLAB обладает уникальными возможностями в выполнении матричных операций. Особое значение приобретают операции с разреженными матрицами, которые широко применяются в технике блочного имитационного математического моделирования, реализованной в расширении Simulink. Для подготовки баз данных большое значение имеют многомерные массивы. Описание этих средств MATLAB и дается в этом уроке. Все они относятся к сложным типам данных.

5.1. Разреженные матрицы

5.1.1. Роль и назначение разреженных матриц

Матрицы без элементов с нулевыми значениями называются *полными матрицами*. Матрицы, содержащие некоторое число элементов с нулевыми значениями, в MATLAB называются *разреженными матрицами*. Часто такими матрицами являются матрицы с 1–3 диагоналями, заполненными ненулевыми элементами и имеющими остальные нулевые элементы. Сильно разреженные матрицы имеют большую часть элементов с нулевыми значениями.

Строго говоря, разреженными называют те матрицы, при работе с которыми используются численные методы, учитывающие упрощение арифметических операций с нулевыми элементами (например, получение нуля при умножении на нуль или пропуск операций сложения и вычитания при использовании этих операций с нулевыми элементами матриц). Применение таких операций уменьшает время, затрачиваемое на обработку матриц и вычисления с ними.

Разреженные матрицы имеют специальную структуру для исключения хранения нулевых элементов. Например, могут храниться только ненулевые элементы в виде чисел двойной точности и их целочисленные индексы или, точнее, диапазоны индексов. Уже одно это уменьшает размеры разреженной матрицы порою во много раз. Кроме того, фактически разреженные матрицы развертываются по столбцам в одномерные массивы, что упрощает индексацию ненулевых элементов – вместо указания двух индексов произвольного элемента достаточно указать один индекс. Детали упаковки разреженных матриц большинству пользователей не важны.

Разреженные матрицы широко используются при решении прикладных задач. Например, моделирование электронных и электротехнических линейных цепей часто приводит к появлению в матричном описании топологии схем сильно разреженных матриц. Для таких матриц создан ряд функций, обеспечивающих эффективную работу с ними и устраняющих тривиальные операции с нулевыми элементами матриц [45, 49].

5.1.2. Элементарные разреженные матрицы

Вначале рассмотрим *элементарные разреженные матрицы* и относящиеся к ним функции системы MATLAB.

Функция `spdiags` расширяет возможности встроенной функции `diag`. Возможны четыре операции, различающиеся числом входных аргументов:

- $[B, d] = \text{spdiags}(A)$ извлекает все ненулевые диагонали из матрицы A размера $m \times n$. B – матрица размера $\min(m, n) \times p$, столбцы которой p являются ненулевыми диагоналями A . d – вектор длины p , целочисленные элементы которого точно определяют номера диагоналей матрицы A (положительные номера – выше главной диагонали, отрицательные – ниже);
- $B = \text{spdiags}(A, d)$ извлекает диагонали, определенные вектором d ;
- $A = \text{spdiags}(B, d, A)$ заменяет столбцами матрицы B диагонали матрицы A , определенные вектором d ;
- $A = \text{spdiags}(B, d, m, n)$ создает разреженную матрицу размера $m \times n$, размещая соответствующие столбцы матрицы B вдоль диагоналей, определяемых вектором d .

Пример:

```
>> A=[1 3 4 6 8 0 0;7 8 0 7 0 0 5; 0 0 0 0 0 9 8 ; 7 6 54 32 0 9 6];
>> d=[1 3 2 2]
>> B = spdiags(A,d)
B =
      3      6      4      4
      0      0      7      7
      0      9      0      0
      0      6      9      9
```

- $S = \text{speye}(m, n)$ возвращает разреженную матрицу размера $m \times n$ с единицами на главной диагонали и нулевыми недиагональными элементами;
- $S = \text{speye}(n)$ равносильна $\text{speye}(n, n)$.

Пример:

```
>> S = speye(4)
S =
      (1,1)      1
      (2,2)      1
      (3,3)      1
      (4,4)      1
```

Матрица $R = \text{sprand}(S)$ имеет ту же структуру, что и разреженная матрица S , но ее элементы распределены по равномерному закону:

- $R = \text{sprand}(m, n, \text{density})$ возвращает случайную разреженную матрицу размера $m \times n$, которая имеет приблизительно $\text{density} \times m \times n$ равномерно распределенных ненулевых элементов ($0 \leq \text{density} \leq 1$).
- $R = \text{sprand}(m, n, \text{density}, rc)$ в дополнение к этому имеет в числе параметров число обусловленности по отношению к операции обращения, приблизительно равное rc . Если вектор rc имеет длину lr ($lr \leq \min(m, n)$), то матрица R имеет rc в качестве своих первых lr сингулярных чисел, все другие значения равны нулю. В этом случае матрица R генерируется с помощью матриц случайных плоских вращений, которые применяются к диагональной матрице с заданными сингулярными числами. Такие матрицы играют важную роль при анализе алгебраических и топологических структур.

Пример:

```
>> d=sprand(4,3,0.6)
d =
      (1,1)      0.6614
      (2,1)      0.2844
      (4,1)      0.0648
      (3,3)      0.4692
      (4,3)      0.9883
```

- `R = sprandn(S)` возвращает матрицу со структурой разреженной матрицы `S`, но с элементами, распределенными по нормальному закону с нулевым средним и дисперсией, равной 1.
- `R = sprandn(m, n, density)` возвращает случайную разреженную матрицу размера $m \times n$, имеющую примерно $density \times m \times n$ нормально распределенных ненулевых элементов ($0 \leq density \leq 1$).
- `R = sprandn(m, n, density, rc)` в дополнение к этому имеет своим параметром число обусловленности по отношению к операции обращения, приблизительно равное `rc`. Если вектор `rc` имеет длину `lr` ($lr \leq \min(m, n)$), то матрица `R` имеет `rc` в качестве своих первых `lr` сингулярных чисел, все другие значения равны нулю. В этом случае матрица `R` генерируется с помощью матриц случайных плоских вращений, которые применяются к диагональной матрице с заданными сингулярными числами.

Пример:

```
>> f=sprandn(3,4,0.3)
f =
      (2,1)     -0.4326
      (2,2)     -1.6656
      (2,3)      0.1253
      (2,4)      0.2877
```

- `sprandsym(S)` возвращает случайную симметрическую матрицу, нижние поддиагонали и главная диагональ которой имеют ту же структуру, что и матрица `S`. Элементы результирующей матрицы распределены по нормальному закону со средним, равным 0, и дисперсией, равной 1.
- `sprandsym(n, density)` возвращает симметрическую случайную разреженную матрицу размера $n \times n$, которая имеет приблизительно $density \times n \times n$ ненулевых элементов; каждый элемент сформирован в виде суммы нормально распределенных случайных чисел ($0 \leq density \leq 1$).
- `R = sprandsym(n, density, rc)` возвращает матрицу с числом обусловленности по отношению к операции обращения, равным `rc`. Закон распределения не является равномерным; значения случайных элементов симметричны относительно 0 и находятся в пределах $[-1, 1]$. Если `rc` – вектор размера `n`, то матрица `R` имеет собственные значения, равные элементам вектора `rc`. Таким образом, если элементы вектора `rc` положительны, то матрица `R` является положительно определенной. В любом случае матрица `R` генерируется с помощью случайного вращения по Якоби диагональных

матриц с заданными собственными значениями и числом обусловленности. Такие матрицы играют важную роль при анализе алгебраических и топологических структур.

- `R = sprandsym(n, density, rc, kind)` возвращает положительно определенную матрицу. Аргумент `kind` может быть следующим:
 - `kind=1` – матрица `R` генерируется из положительно определенной диагональной матрицы с помощью случайных вращений Якоби. `R` имеет точно заданное число обусловленности;
 - `kind=2` – матрица `R` генерируется как смещенная сумма матриц внешних произведений. Число обусловленности матрицы приблизительно, но структура более компактна (по сравнению с предыдущим случаем);
 - `kind=3` – генерируется матрица `R` той же структуры, что и `S`, а число обусловленности приближенно равно $1/rc$. Значение `density` игнорируется.

Пример:

```
>> a=sprandsym(4,0.3,0.8)
a =
      (1,1)      0.9818
      (3,1)      0.0468
      (2,2)     -0.9283
      (1,3)      0.0468
      (3,3)      0.8800
      (4,4)     -0.8000
```

5.1.3. Преобразование разреженных матриц

Теперь рассмотрим функции преобразования разреженных матриц. Они представлены ниже:

- `k = find(X)` возвращает индексы вектора `x` для его ненулевых элементов. Если таких элементов нет, то `find` возвращает пустой вектор. `find(X>100)` возвращает индексы элементов вектора с `X>100`;
- `[i, j] = find(X)` возвращает индексы строки и столбца для ненулевого элемента матрицы `X`;
- `[i, j, v] = find(X)` возвращает вектор-столбец `v` ненулевых элементов матрицы `X` и индексы строки `i` и столбца `j`. Вместо `X` можно вставить `(X, операция отношения, параметр)`, и тогда индексы и вектор-столбец будут отражать элементы матрицы, удовлетворяющие данному отношению. Единственное исключение `find(x ~= 0)`. Индексы те же, что и при исполнении `find(X)`, но вектор `v` содержит только единицы.

Пример:

```
>> q=sprand(3,4,0.6)
q =
      (1,1)      0.7266
      (1,2)      0.4120
      (3,2)      0.2679
      (3,3)      0.4399
```

```

          (2,4)      0.7446
          (3,4)      0.9334
>> [i,j]=find(q)
i =
     1
     1
     3
     3
     2
     3
j =
     1
     2
     2
     3
     4
     4

```

- `full(S)` преобразует разреженную матрицу S в полную; если исходная матрица S была полной, то `full(S)` возвращает S . Пусть X – матрица размера $m \times n$ с $nz = nnz(X)$ ненулевыми элементами. Тогда `full(X)` требует такой объем памяти, чтобы хранить $m \times n$ действительных чисел, в то время как `sparse(X)` требует пространство для хранения лишь nz действительных чисел и $(nz+n)$ целых чисел – индексов. Большинству компьютеров для хранения действительного числа требуется вдвое больше пространства, чем для целого. Для таких компьютеров `sparse(X)` требует меньше пространства, чем `full(X)`, если плотность $nnz / \text{prod}(\text{size}(X)) < 2/3$. Выполнение операций над разреженными матрицами, однако, требует больше затрат времени, чем над полными, поэтому для эффективной работы с разреженными матрицами плотность расположения ненулевых элементов должна быть много меньше $2/3$.

Примеры:

```

>> q=sprand(3,4,0.6)
q =
          (1,1)      0.0129
          (1,2)      0.3840
          (2,2)      0.6831
          (3,3)      0.0928
>> d=full(q)
d =
    0.0129    0.3840         0         0
         0    0.6831         0         0
         0         0    0.0928         0

```

- `S=sparse(A)` преобразует полную матрицу в разреженную, удаляя нулевые элементы. Если матрица S уже разреженная, то `sparse(S)` возвращает S . Функция `sparse` – это встроенная функция, которая формирует матри-

цы в соответствии с правилами записи разреженных матриц, принятыми в системе MATLAB;

- `S=sparse(i, j, s, m, n, nzmax)` использует векторы `i`, `j` и `s` для того, чтобы генерировать разреженную матрицу размера $m \times n$ с ненулевыми элементами, количество которых не превышает `nzmax`. Векторы `i` и `j` задают позиции элементов и являются целочисленными, а вектор `s` определяет числовое значение элемента матрицы, которое может быть действительным или комплексным. Все элементы вектора `s`, равные нулю, игнорируются вместе с соответствующими значениями `i` и `j`. Векторы `i`, `j` и `s` должны быть одной и той же длины;
- `S = sparse(i, j, s, m, n)` использует `nzmax=length(s)`;
- `S = sparse(i, j, s)` использует `m=max(i)` и `n=max(j)`. Максимумы вычисляются раньше, чем нулевые строки столбца `S`, которые будут удалены;
- `S = sparse(m, n)` равносильно `sparse([], [], [], m, n, 0)`. Эта команда генерирует предельную разреженную матрицу, где $m \times n$ элементов нулевые.

Все встроенные в MATLAB арифметические, логические и индексные операции могут быть применены как к полным матрицам, так и к разреженным. Операции над разреженными матрицами возвращают разреженные матрицы, а операции над полными матрицами возвращают полные матрицы. В большинстве случаев операции над смешанными матрицами возвращают полные матрицы. Исключение составляют случаи, когда результат смешанной операции явно сохраняет разреженный тип. Так бывает при поэлементном умножении массивов `A.*S`, где `S` – разреженный массив. Пример:

```
>> i=[2,4,3];j=[1,3,8];s=[4,5+5i,9];t = sparse(i,j,s,5,8)
t =
      (2,1)      4.0000
      (4,3)      5.0000+ 5.0000i
      (3,8)      9.0000
```

Функция `spconvert` используется для создания разреженных матриц из простых разреженных форматов, легко производимых вне средств MATLAB:

- `S = spconvert(D)` преобразует матрицу `D` со строками, содержащими `[i, j, r]` или `[i, j, r, s]`, где `i` – индекс ряда, `j` – индекс строки, `r` – численное значение, в соответствующую разреженную матрицу. Матрица `D` может иметь `nnz` или `nnz+1` строк и три или четыре столбца. Три элемента в строке генерируют действительную матрицу, четыре элемента в строке генерируют комплексную матрицу (`s` преобразуется в мнимую часть значения элемента). Последняя строка массива `D` типа `[m n 0]` или `[m n 0 0]` может быть использована для определения `size(S)`. Команда `spconvert` может быть использована только после того, как матрица `D` загружена, или из MAT-файла, или из ASCII-файла при помощи команды `load`.

```
load mydata.dat
A = spconvert(mydata);
```

5.1.4. Работа с ненулевыми элементами разреженных матриц

Поскольку разреженные матрицы содержат *ненулевые элементы*, то предусмотрен ряд функций для работы с ними:

- `nnz(X)` возвращает число ненулевых элементов матрицы X . Плотность разреженной матрицы определяется по формуле $\text{nnz}(X) / \text{numel}(X)$. Пример:

```
h = sparse(hilb(10));
>> nnz(h)
ans = 100
```

- `nonzeros(A)` возвращает полный вектор-столбец ненулевых элементов матрицы A , выбирая их последовательно по столбцам. Эта функция дает только выход s , но не значения i и j из аналогичного выражения $[i, j, s] = \text{find}(A)$. Вообще, $\text{length}(s) = \text{nnz}(A) \leq \text{nzmax}(A) \leq \text{prod}(\text{size}(A))$. Пример:

```
>> g=nonzeros(sparse(hankel([1,2,8])))
g =
     1
     2
     8
     2
     8
     8
```

- `nzmax(S)` возвращает количество ячеек памяти для ненулевых элементов. Обычно функции `nnz(S)` и `nzmax(S)` дают один и тот же результат. Но если S создавалась в результате операции над разреженными матрицами, такой как умножение или LU-разложение, может быть выделено больше элементов памяти, чем требуется, и `nzmax(S)` отражает это. Если S – разреженная матрица, то `nzmax(S)` – максимальное количество ячеек для хранения ненулевых элементов. Если S – полная матрица, то `nzmax(S) = numel(S)`. Пример:

```
>> q=nzmax(sparse(hankel([1,7,23])))
q = 6
```

- `S=spalloc(m,n,nzmax)` создает массив для разреженной матрицы S размера $m \times n$ с пространством для размещения `nzmax` ненулевых элементов. Затем матрица может быть заполнена по столбцам;
- `spalloc(m,n,nzmax)` эквивалентна функции `sparse([],[],[],m,n,nzmax)`;

Пример:

```
>> S = spalloc(5,4,5);
```

- `spfun` – вычисление функции для ненулевых элементов. Функция `spfun` применяется выборочно только к ненулевым элементам разреженной матрицы, сохраняя при этом разреженность исходной матрицы.

- `f = spfun(@function, S)` вычисляет `function(S)` для ненулевых элементов матрицы `S`. Имя `function` – это имя `m`-файла или встроенной в ядро функции. `function` должна работать с матричным аргументом `S` и вычислить функцию для каждого элемента матрицы `S`.

Пример:

```
>> S=spfun(@exp, sprand(4,5,0.4))
S =
      (2,2)      1.6864
      (2,3)      2.4112
      (3,3)      2.6638
      (2,4)      1.1888
      (3,4)      1.3119
      (4,4)      2.4007
      (3,5)      1.2870
```

- `R = spones(S)` генерирует матрицу `R` той же разреженности, что и `S`, но заменяет на 1 все ненулевые элементы исходной матрицы. Пример:

```
>> S=sprand(3,2,0.3)
S =
      (3,1)      0.2987
      (1,2)      0.1991
>> spones(S)
ans =
      (3,1)      1
      (1,2)      1
```

5.1.5. Функция `spy` визуализации разреженных матриц

Визуализация разреженных матриц нередко позволяет выявить не только любопытные, но и полезные и поучительные свойства тех математических закономерностей, которые порождают такие матрицы или описываются последними. MATLAB имеет специальные средства для визуализации разреженных матриц, реализованные приведенными ниже командами:

- `spy(S)` графически отображает разреженность произвольной матрицы `S`;
- `spy(S, markersize)` графически отображает разреженность матрицы `S`, выводя маркеры в виде точек точно определенного размера `markersize`;
- `spy(S, 'LineStyle')` отображает разреженность матрицы в виде графика с точно определенным (с помощью параметра `LineStyle`) цветом линии и маркера. Параметр `LineStyle` определяется так же, как параметр команды `plot`;
- `spy(S, 'LineStyle', markersize)` использует точно определенные тип, цвет и размер графического маркера. Обычно `S` – разреженная матрица, но допустимо использование и полной матрицы, когда расположение элементов, отличных от нуля, составляет график.

Пример:

```
>>S=sparse(sprandn(20,30,0.9));spy(S,'r',6)
```

Построенный по этому примеру график показан на рис. 5.1.

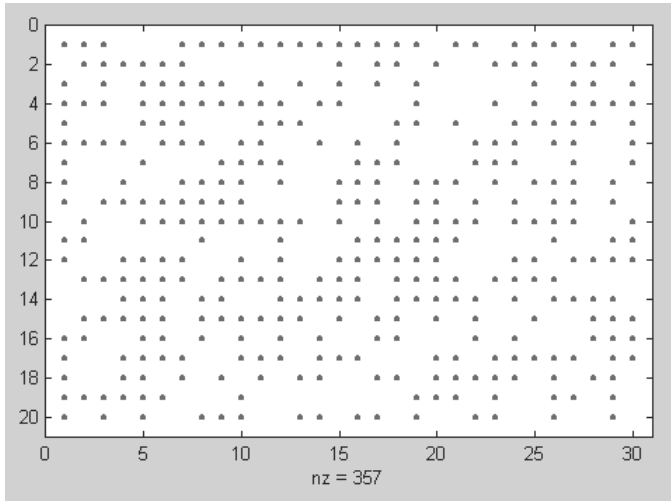


Рис. 5.1. Визуализация разреженной матрицы

5.1.6. Алгоритмы упорядочения

Упорядочение – это еще одна характерная для разреженных матриц операция. Ее алгоритм реализуется несколькими функциями:

- $p = \text{colmmd}(S)$ возвращает вектор упорядоченности столбцов разреженной матрицы S ¹. Для несимметрической матрицы S вектор упорядоченности столбцов p такой, что $S(:, p)$ будет иметь более разреженные L и U в LU -разложении, чем S . Такое упорядочение автоматически применяется при выполнении операций обращения \backslash и деления $/$, а также при решении систем линейных уравнений с разреженными матрицами. Можно использовать команду `sprands`, чтобы изменить некоторые параметры, связанные с эвристикой в алгоритме `colmmd`;
- $j = \text{colperm}(S)$ возвращает вектор перестановок j , такой что столбцы матрицы $S(:, j)$ будут упорядочены по возрастанию числа ненулевых элементов. Эту функцию полезно иногда применять перед выполнением LU -разложения. Если S – симметрическая матрица, то $j = \text{colperm}(S)$ возвращает вектор перестановок j , такой что и столбцы, и строки $S(j, j)$ упорядочены по возрастанию ненулевых элементов. Если матрица S поло-

¹ Функция `colamd` – более мощная и быстрая реализация `colmmd`.

жительно определенная, то иногда полезно применять эту функцию и перед выполнением разложения Холецкого. Пример:

```
>> S=sparse([2,3,1,4,2],[1,3,2,3,2],[4,3,5,6,7],4,5);full(S)
ans =
     0     5     0     0     0
     4     7     0     0     0
     0     0     3     0     0
     0     0     6     0     0

>> t=colperm(S)
t =
     4     5     1     2     3

>> full(S(:,t))
ans =
     0     0     0     5     0
     0     0     4     7     0
     0     0     0     0     3
     0     0     0     0     6
```

- $p = \text{dmperm}(A)$ возвращает вектор максимального соответствия p , такой что если исходная матрица A имеет полный столбцовый ранг, то $A(p, :)$ – квадратная матрица с ненулевой диагональю. Матрица $A(p, :)$ называется декомпозицией Далмейджа-Мендельсона, или DM-декомпозицией. Если A – приводимая матрица¹, линейная система $Ax=b$ может быть решена приведением A к верхней блочной треугольной форме с неприводимым диагональным блоком. Решение может быть найдено методом обратной подстановки;
- $[p, q, r] = \text{dmperm}(A)$ находит перестановку строк p и перестановку столбцов q квадратной матрицы A , такую что $A(p, q)$ – матрица в блоке верхней треугольной формы. Третий выходной аргумент r – целочисленный вектор, описывающий границы блоков. K -й блок матрицы $A(p, q)$ имеет индексы $r(k) : r(k+1)-1$;
- $[p, q, r, s] = \text{dmperm}(A)$ находит перестановки p и q и векторы индексов r и s , так что матрица $A(p, q)$ оказывается в верхней треугольной форме. Блок имеет индексы $(r(i) : r(i+1)-1, s(i) : s(i+1)-1)$.

В терминах теории графов диагональные блоки соответствуют сильным компонентам Холла графа смежности матрицы A . Примеры:

```
>> A=sparse([1,2,1,3,2],[3,2,1,1,1],[7,6,4,5,4],3,3);full(A)
ans =
     4     0     7
     4     6     0
     5     0     0

>> [p,q,r]=dmperm(A)
```

¹ Квадратная матрица A называется приводимой, если она подобна клеточной матрице, квадратные элементы которой соответствуют индукции линейного оператора A в отдельные подпространства.


```

p =
     1     2     3
q =
     3     2     1
r =
     1     2     3     4
>> full(A(p,q))
ans =
     7     0     4
     0     6     4
     0     0     5

```

- `symmmd(S)` возвращает вектор упорядоченности для симметричной положительно определенной матрицы S , так что $S(p,p)$ будет иметь более разреженное разложение Холецкого, чем S . Иногда `symmmd` хорошо работает с симметрическими неопределенными матрицами. Такое упорядочение автоматически применяется при выполнении операций \backslash и $/$, а также при решении линейных систем с разреженными матрицами.

Можно использовать команду `spparms`, чтобы изменить некоторые опции и параметры, связанные с эвристикой в алгоритме.

Алгоритм упорядочения для симметрических матриц основан на алгоритме упорядочения по разреженности столбцов. Фактически `symmmd(S)` только формирует матрицу K с такой структурой ненулевых элементов, что $K' * K$ имеет тот же график разреженности, что и S , и затем вызывает алгоритм упорядочения по разреженности столбцов для K . Пример:

```

>> B=bucky;p=symmmd(B);
>> R=B(p,p);
>> subplot(1,2,1),spy(B);subplot(1,2,2),spy(R)

```

На рис. 5.2 приводится пример применения функции `symmmd` к элементам разреженной матрицы.

- `r = symrcm(S)` возвращает вектор упорядоченности для симметричной матрицы S и называется упорядочением Катхилла-Макки. Причем форми-

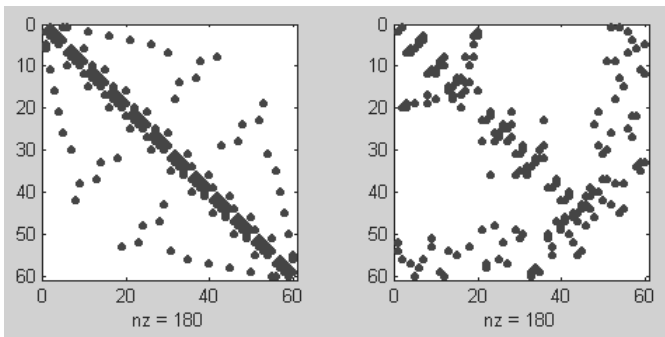


Рис. 5.2. Пример применения функции `symmmd`

руется такая перестановка r , что $S(r, r)$ будет концентрировать ненулевые элементы вблизи диагонали. Это хорошее упорядочение как перед LU-разложением, так и перед разложением Холецкого. Упорядочение применимо как для симметрических, так и для несимметрических матриц. Для вещественной симметрической разреженной матрицы S (такой, что $S=ST$) собственные значения $S(r, r)$ совпадают с собственными значениями S , но для вычисления $\text{eig}(S(r, r))$ требуется меньше времени, чем для вычисления $\text{eig}(S)$. Пример:

```
>> B=bucky;p=symrcm(B); R=B(p,p);
>> subplot(1,2,1), spy(B);subplot(1,2,2), spy(R)
```

На рис. 5.3 приведен пример концентрации ненулевых элементов разреженной матрицы вблизи главной диагонали.

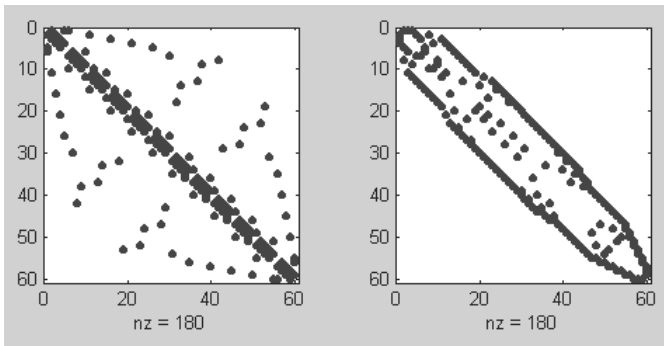


Рис. 5.3. Пример применения функции *symrcm*

5.2. Применение разреженных матриц

5.2.1. Смежные матрицы и графы

Во многих приложениях математики используются *графы*. Их можно определить как совокупность точек (узлов) со спецификацией соединений между ними. Графы можно экономно представлять с помощью разреженных смежных матриц. Эти матрицы имеют в основном нулевые элементы, но часть последних имеет единичные значения и используется для факта соединения вершин графов, что и создает те или иные фигуры.

Пример представления графа – фигуры ромба, имеющего 4 узла, – с помощью смежной матрицы A представлен на рис. 5.4.

Полное описание графа требует, кроме задания смежной матрицы, указания списка xu координат узлов, например:

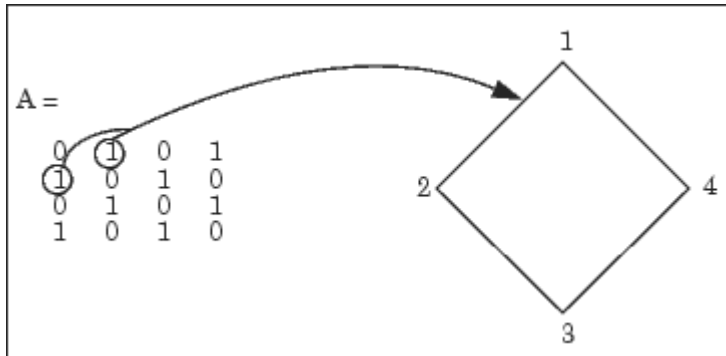


Рис. 5.4. Пример представления графа с помощью разреженной смежной матрицы

```
A=[0 1 0 1; 1 0 1 0; 0 1 0 1; 1 0 1 0]1;
xy=[1 2; 2 1; 3 3; 2 5]
```

Тогда с помощью графической функции `gplot` можно построить граф:

```
gplot(A,xy)
```

Узлы графа при этом будут построены по явно заданным координатам.

5.2.2. Пример построения фигуры *bucky*

Аналогичным описанному способом можно строить довольно сложные фигуры. К примеру, рассмотренный выше объект *bucky* описывает в виде графов молекулу C_{60} , содержащую 60 атомов сферической конфигурации. Ее можно представить матрицей B и вектором v :

```
>> [B,v]=bucky;
```

Вектор задает список *xyz*-координат для 60 точек единичной сферы. Задав в командном режиме команды:

```
>> gplot(B,v)
>> axis equal
```

можно построить граф данной молекулы. Он представлен на рис. 5.5.

Граф фигуры *bucky* представляет собой сферическую поверхность, построенную из многоугольников. Она дает наглядное представление о структуре молекулы.

5.2.3. Оцифровка узлов графа

Иногда желательно построить граф, узлы которого оцифрованы. Это несложно сделать, используя конструкцию цикла `for-end`:

```
% Программа построения графа с оцифровкой узлов
```

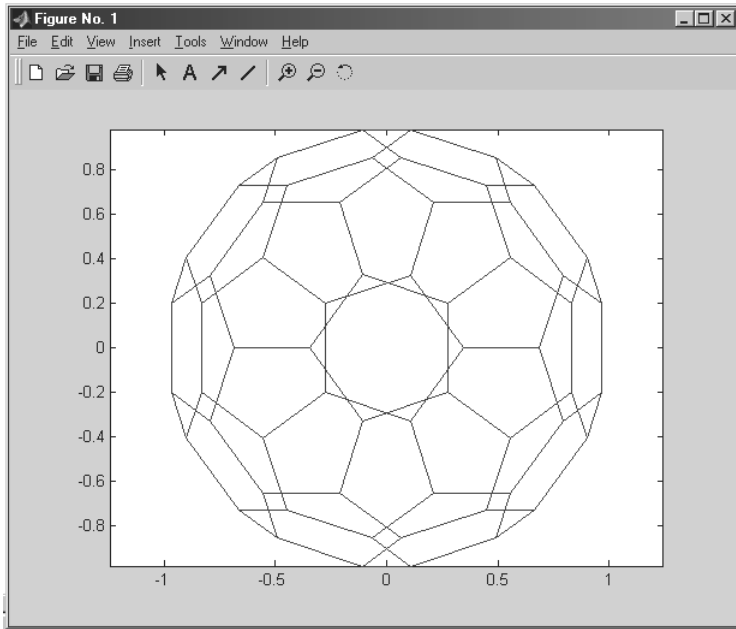


Рис. 5.5. Пример построения графа фигуры *bucky*, представленной смежной матрицей *B* и вектором координат узлов *v*

```
k = 1:30;
gplot(B(k,k),v);
axis square
cfor j = 1:30, text(v(j,1),v(j,2), int2str(j));
end
```

Здесь задана оцифровка первых 30 узлов, что дает граф половины фигуры, представленной на рис. 5.5, – см. рис. 5.6.

5.2.4. Применение разреженных матриц в аэродинамике

В описании и справке по системе MATLAB можно найти пример подобного описания для моделирования профиля воздушного потока, обтекающего крыло самолета, – рис. 5.7.

При решении этой задачи в NASA использовались 4253 треугольные ячейки сетки, для описания которых потребовалась разреженная матрица, имеющая 28 831 ненулевой элемент. При этом степень заполнения матрицы составляла всего 0,0016.

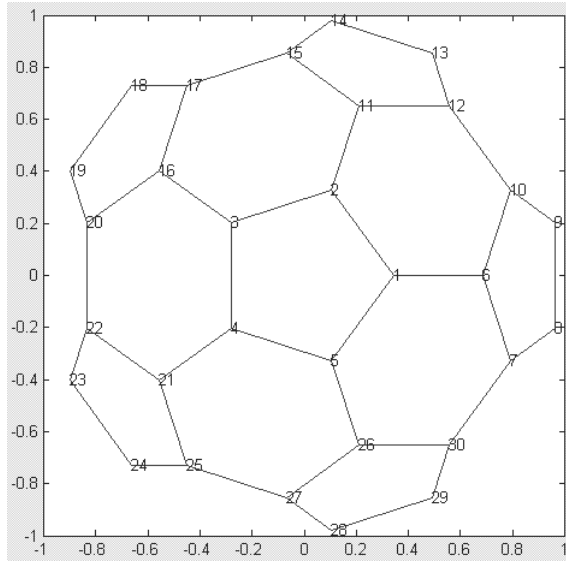


Рис. 5.6. Пример построения графа передней части фигуры *wisku* с оцифровкой узлов

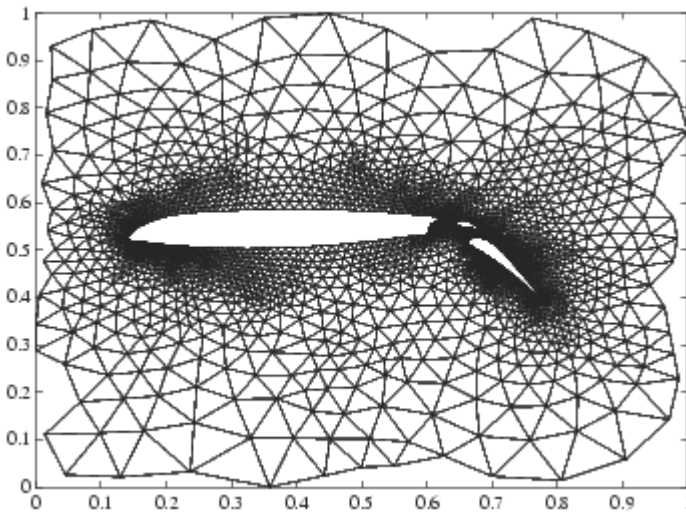


Рис. 5.7. Представление графом воздушного потока, обтекающего крыло самолета

5.2.5. Визуализация разреженных матриц, возведенных в степень

Команда

```
>> spy(B)
```

позволяет визуализировать матрицу B . Полученный с ее помощью вид матрицы уже был представлен на рис. 5.2 (слева). Любопытно посмотреть, как меняется вид разреженных матриц при математических операциях. Например, команда

```
>> spy(B^3)
```

строит вид матрицы B^3 , представленный на рис. 5.8.

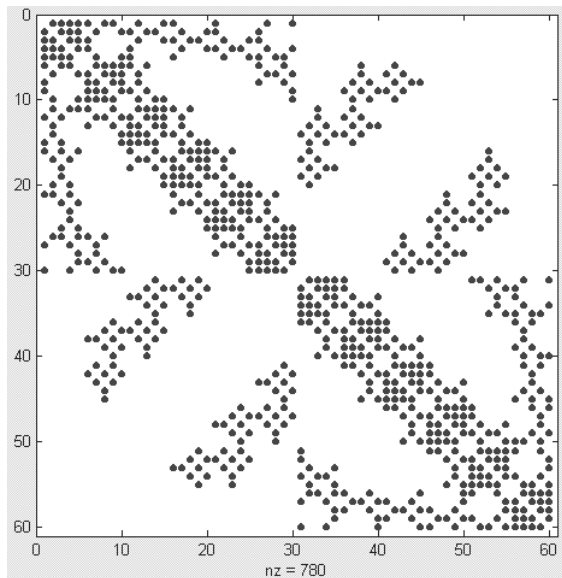


Рис. 5.8. Визуализация матрицы B^3

Другая команда

```
>> spy(B^5)
```

строит представление матрицы B^5 . Оно представлено на рис. 5.9.

Сравнение представлений матриц B^3 и B^5 с представлением матрицы B (рис. 5.2, слева) наглядно показывает, насколько меняется вид матрицы даже при таких простых преобразованиях, как возведение в целую степень.

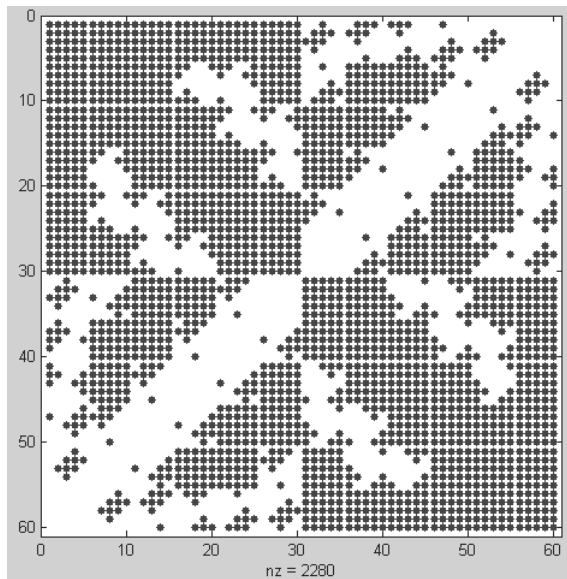


Рис. 5.9. Визуализация матрицы B^5

5.2.6. Демонстрационные примеры на визуализацию разреженных матриц

В разделе демонстрационных примеров на технику применения разреженных матриц можно найти ряд интересных примеров на их визуализацию. Например, в разделе справки **Demos** системы MATLAB 6.5 имеется пример **Sparse Matrix**, позволяющий просматривать представление различных разреженных матриц в режиме слайд-шоу. Первый кадр слайд-шоу иллюстрирует представление симметричной разреженной матрицы – рис. 5.10.

Нажимая кнопку **Next** или задав опцию **AutoPlay**, можно «вручную» или автоматически просмотреть представление для ряда других разреженных матриц.

5.3. Функции разреженных матриц

5.3.1. Норма, число обусловленности и ранг разреженной матрицы

Ниже представлены функции, позволяющие вычислять *числа обусловленности* и *ранги* для разреженных матриц.

- $c = \text{condest}(A)$ использует метод Хейджера в модификации Хаема для оценки числа обусловленности матрицы по первой норме. Вычисленное

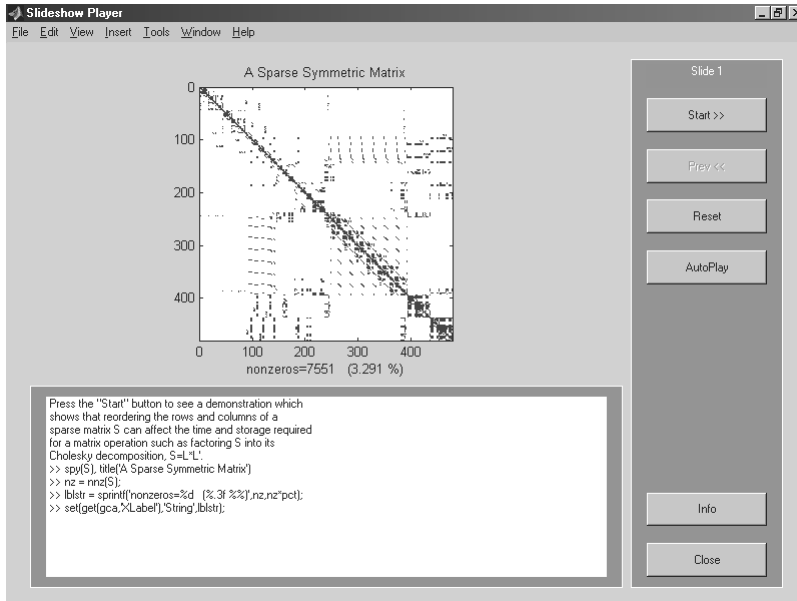


Рис. 5.10. Визуализация матрицы B^5

значение c – нижняя оценка числа обусловленности матрицы A по первой норме. Для повторяемости результатов перед выполнением функции `condst` нужно обязательно выполнить `rand('state',L)`, где L – одно и то же целое число.

- `[c, v] = condst(A)` возвращает число обусловленности и вектор v , такой что выполняется условие $\text{norm}(A*v, 1) = \text{norm}(A, 1) * \text{norm}(v, 1) / c$. Таким образом, для больших значений c вектор v является аппроксимацией нуль-вектора матрицы A .
- `nrm = normest(S)` тогда, когда из-за чрезмерного числа элементов в матрице вычисление `nrm = normest(S)` занимает слишком много времени, то `nrm = normest(S)` возвращает оценку второй нормы матрицы S . Эта функция изначально предназначена для работы с разреженными матрицами, хотя она работает корректно и с разреженными матрицами.
- `nrm = normest(S, tol)` использует относительную погрешность `tol` вместо используемого по умолчанию значения 10^{-6} .
- `[nrm, count] = normest(...)` возвращает оценку второй нормы и количество использованных операций.

Примеры:

```

>> F=wilkinson(150);
>> condst(sparse(F))
ans = 460.2219
>> normest(sparse(F))
ans = 75.2453
    
```


- `r=sprank(S)` вычисляет структурный ранг разреженной матрицы S . В терминах теории графов он известен также под следующими названиями: максимальное сечение, максимальное соответствие и максимальное совпадение. Для величины структурного ранга всегда выполняется условие $\text{sprank}(S) \geq \text{rank}(S)$, а в точной арифметике с вероятностью 1 выполняется условие $\text{sprank}(S) == \text{rank}(\text{sprandn}(S))$.

Пример:

```
>> S=[3 0 0 0 4; 5 4 0 8 0; 0 0 0 1 3];
>> r=sprank(S)
r =
    3
```

5.3.2. Функции разложения Холецкого для разреженных матриц

Теперь рассмотрим функции, реализующие разложение Холецкого для разреженных матриц.

- `cholinc(X, '0')` возвращает неполное разложение Холецкого для действительной симметрической положительно определенной разреженной матрицы¹. Результат представляет собой верхнюю треугольную матрицу.
- `R=cholinc(X, '0')` возвращает верхнюю треугольную матрицу, которая имеет такую же разреженную структуру, как и верхний треугольник действительной положительно определенной матрицы X . Результат умножения $R'*R$ соответствует X по своей разреженной структуре. Положительной определенности матрицы X недостаточно, чтобы гарантировать существование неполного разложения Холецкого, и в этом случае выдается сообщение об ошибке.
- `[R,p]=cholinc(X, '0')` никогда не выдает сообщения об ошибке в ходе разложения. Если X – положительно определенная матрица, то $p=0$ и матрица R – верхняя треугольная, в противном случае p – положительное целое число, R – верхняя треугольная матрица размера $q \times n$, где $q=p-1$. Разреженная структура матрицы R такая же, как и у верхнего треугольника размера $q \times n$ матрицы X , и произведение $R'*R$ размера $n \times n$ соответствует структуре разреженности матрицы X по ее первым q строкам и столбцам $X(1:q, :)$ и $X(:, 1:q)$.
- `R=cholinc(X, droptol)` возвращает неполное разложение Холецкого любой квадратной разреженной матрицы, используя положительный числовой параметр `droptol`. Функция `cholinc(X, droptol)` возвращает приближение к полному разложению Холецкого, вычисленному с помощью функции `chol(X)`. При меньших значениях `droptol` аппроксимация

¹ Проверить, является ли матрица разреженной, можно при помощи функции `issparse`. Она вернет 1, если матрица разреженная. Сама эта функция может применяться при использовании `pcg` или подобных методов решения линейных уравнений, когда обе части уравнения умножаются `cholinc(S)`, где S – симметрическая матрица.

улучшается, пока значение `droptol` не станет равным 0. В этом случае `cholinc` задает полное преобразование Холецкого (`chol(X)`).

- `R = cholinc(X, options)` использует структуру с тремя переменными, которые могут быть использованы в любой из комбинаций: `droptol`, `micchol`, `rdiag`. Дополнительные поля игнорируются. Если `micchol=1`, `cholinc` возвращает модифицированное разложение Холецкого. Если `rdiag=1`, то все нули на диагонали верхней треугольной матрицы заменяются квадратным корнем от произведения `droptol` и нормы соответствующего столбца матрицы $X - \sqrt{\text{droptol} * \text{norm}(X(:, j))}$. По умолчанию `rdiag=0`.
- `R = cholinc(X, droptol)` и `R = cholinc(X, options)` возвращают верхнюю треугольную матрицу `R`. Результат `R'*R` – это аппроксимация матрицы.
- `R = cholinc(X, 'inf')` возвращает разложение Холецкого в неопределенности, когда не удастся получить обычное разложение. Матрица `X` может быть действительной квадратной положительно полуопределенной.

Пример:

```
>> S = delsq(numgrid('C', 4))
S =
      (1,1)      4
      (2,1)     -1
      (1,2)     -1
      (2,2)      4
      (3,2)     -1
      (2,3)     -1
      (3,3)      4

>> R0 = cholinc(S, '0')
R0 =
      (1,1)      2.0000
      (1,2)     -0.5000
      (2,2)      1.9365
      (2,3)     -0.5164
      (3,3)      1.9322
```

5.3.3. LU-разложение разреженных матриц

Функция `luinc` осуществляет неполное LU-разложение и возвращает нижнюю треугольную матрицу, верхнюю треугольную матрицу и матрицу перестановок для разреженных матриц. Используется в следующих формах:

- `luinc(X, '0')` возвращает неполное LU-разложение уровня 0 квадратной разреженной матрицы. Треугольные факторы (множители) имеют такую же разреженность (то есть график разреженности, см. `spy`), как и матрица перестановок квадратной матрицы `X`, и их произведение имеет ту же разреженность, что и матрица перестановок `X`. Функция `luinc(X, '0')` возвращает нижнюю треугольную часть нижнего фактора (множителя) и

верхний треугольный фактор в одной и той же результирующей матрице. Вся информация о матрице перестановок теряется, но зато число ненулевых элементов результирующей матрицы равно числу ненулевых элементов матрицы X с возможностью исключения некоторых нулей из-за сокращения.

- $[L, U] = \text{luinc}(X, '0')$, где X – матрица размером $n \times n$, возвращает нижнюю треугольную матрицу L и верхнюю треугольную матрицу U . Разреженности матриц L , U и X не сравнимы, но сумма числа ненулевых элементов в матрицах L и U поддерживается равной $\text{nnz}(X) + n$, с возможностью исключения некоторых нулей в L и U из-за сокращения.
- $[L, U, P] = \text{luinc}(X, '0')$ возвращает нижнюю треугольную матрицу L , верхнюю треугольную матрицу U и матрицу перестановок P . Матрица L имеет такую же разреженную структуру, как нижняя треугольная часть перестановленной матрицы X – $\text{spones}(L) = \text{spones}(\text{tril}(P * X))$ с возможными исключениями единиц на диагонали матрицы L , где $P * X$ может быть равно 0 .
- $\text{luinc}(X, \text{droptol})$ возвращает неполное LU-разложение любой разреженной матрицы, используя порог droptol . Параметр droptol должен быть неотрицательным числом.
- $\text{luinc}(X, \text{droptol})$ возвращает приближение к полному LU-разложению, полученному с помощью функции $\text{lu}(X)$. При меньших значениях droptol аппроксимация улучшается, пока значение droptol не станет равным 0 . В этом случае имеет место полное LU-разложение.
- $\text{luinc}(X, \text{options})$ использует структуру с четырьмя переменными, которые могут быть использованы в любой из комбинаций: droptol , milu , udiag , thresh . Дополнительные поля игнорируются. Если $\text{milu} = 1$, функция luinc возвращает модифицированное неполное LU-разложение. Если $\text{udiag} = 1$, то все нули на диагонали верхней треугольной части заменяются на локальную ошибку droptol .
- $\text{luinc}(X, \text{options})$ – то же самое, что и $\text{luinc}(X, \text{droptol})$, если options содержит только параметр droptol .
- $[L, U] = \text{luinc}(X, \text{options})$ возвращает перестановку треугольной матрицы L и верхнюю треугольную матрицу U . Результат $L * U$ аппроксимирует X .
- $[L, U, P] = \text{luinc}(X, \text{options})$ возвращает нижнюю треугольную матрицу L , верхнюю треугольную матрицу U и матрицу перестановок P . Ненулевые входные элементы матрицы U удовлетворяют выражению $\text{abs}(U(i, j)) \geq \text{droptol} * \text{norm}(X(:, j))$, с возможным исключением диагональных входов, которые были сохранены, несмотря на неудовлетворение критерию.
- $[L, U, P] = \text{luinc}(X, \text{options})$ – то же самое, что и $[L, U, P] = \text{luinc}(X, \text{droptol})$, если options содержит только параметр droptol .

5.3.4. Собственные значения и сингулярные числа разреженных матриц

Применение функции `eigs` решает *проблему собственных значений*, состоящую в нахождении нетривиальных решений системы уравнений, которая может быть интерпретирована как алгебраический эквивалент системы обыкновенных дифференциальных уравнений в явной форме Коши: $A \cdot v = \lambda \cdot v$. Вычисляются только отдельные выбранные собственные значения, или собственные значения и собственные векторы.

- `[V, D] = eigs(A)` или `[V, D] = eigs('Afun', n)` возвращает отдельные собственные значения для первого входного аргумента. Этот параметр может быть как квадратной матрицей (полной или разреженной, симметрической или несимметрической, вещественной или комплексной), так и строкой, содержащей имя m-файла, который применяет линейный оператор к столбцам данной матрицы. В последнем случае второй входной аргумент `n` определяет порядок задачи.

В случае одного выходного параметра `D` – вектор, содержащий `k` собственных значений. В случае двух выходных аргументов `D` – диагональная матрица размера $k \times k$ и `V` – матрица, содержащая `k` столбцов, так что $A \cdot V = V \cdot D$ или $A \cdot V = B \cdot V \cdot D$.

- `[U, S, V] = svds(A, k)` возвращает `k` наибольших сингулярных чисел и сингулярных векторов матрицы `A`. По умолчанию `k=5`. Если `A` – матрица размера $m \times n$, то `U` – матрица размера $m \times k$ с ортонормальными столбцами, `S` – диагональная матрица размера $k \times k$, `V` – матрицы размера $n \times k$ с ортонормальными столбцами.
- `[U, S, V] = svds(A, k, 0)` возвращает `k` наименьших сингулярных чисел и сингулярных векторов.
- `s = svds(A, k, ...)` возвращает только вектор сингулярных чисел.

С рядом дополнительных примеров на операции с разреженными матрицами (с их визуализацией) можно ознакомиться по справке MATLAB по данному разделу (Sparse Matrix Operations). В нем приведена и библиография (зарубежная) по разреженным матрицам и их применению.

5.4. Многомерные массивы

5.4.1. Понятие о многомерных массивах

Многомерные массивы характеризуются размерностью более двух. Таким массивам можно дать наглядную интерпретацию. Так, матрицу (двумерный массив) можно записать на одном листе бумаги в виде строк (`rows`) и столбцов (`columns`), состоящих из элементов матрицы, – рис. 5.11.

Тогда блокнот с такими листками можно считать трехмерным массивом (рис. 5.12), полку в шкафу с блокнотами – четырехмерным массивом, шкаф со

| | column | | | |
|-----|--------|--------|--------|--------|
| row | (1, 1) | (1, 2) | (1, 3) | (1, 4) |
| | (2, 1) | (2, 2) | (2, 3) | (2, 4) |
| | (3, 1) | (3, 2) | (3, 3) | (3, 4) |
| | (4, 1) | (4, 2) | (4, 3) | (4, 4) |

Рис. 5.11. Представление
двумерного массива (матрицы)

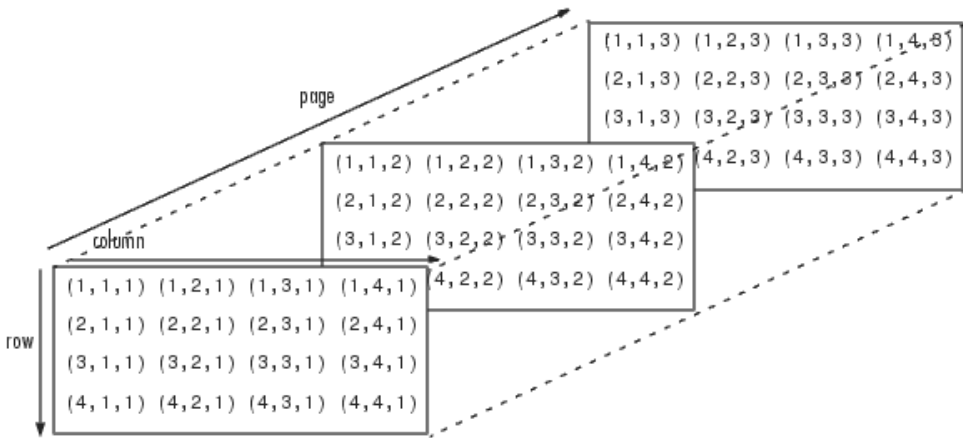


Рис. 5.12. Представление трехмерного массива,
содержащего ряд страниц (pages)

множеством полков – пятимерным массивом и т. д. В этой книге практически нигде, кроме этого раздела, мы не будем иметь дело с массивами, размерность которых выше двух, но знать о возможностях MATLAB в части задания и применения многомерных массивов все же полезно.

В нашей литературе понятия «размер» и «размерность» массивов являются почти синонимами. Однако в литературе по системе MATLAB и в данной книге они имеют явно разный смысл. Под *размерностью* массивов понимается число измерений в пространственном представлении массивов, а под *размером* – число строк и столбцов ($m \times n$) в каждой размерности массива.

С многомерными массивами могут выполняться те же операции и вычисления, что и с двумерными массивами (матрицами). В частности, это относится ко всем операциям, осуществляемым поэлементно, а также к функции `sum`, `mean`, `cross` и др.

5.4.2. Применение оператора : в многомерных массивах

Для выделения отдельных страниц многомерных массивов можно использовать оператор : (двоеточие). Подобные операции наглядно иллюстрирует рис. 5.13.

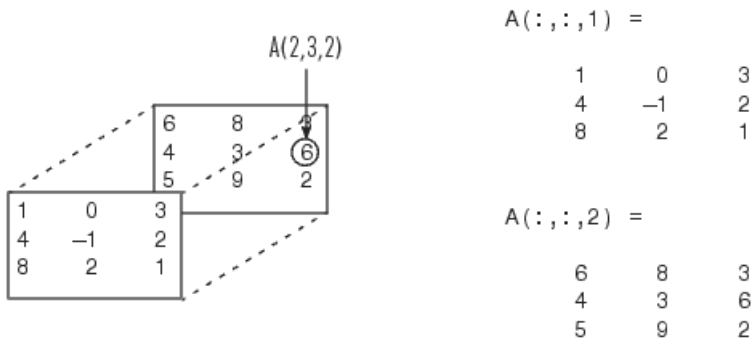


Рис. 5.13. Примеры работы с трехмерным массивом

При обычном задании массивов (с помощью символа точки с запятой ;) число строк массива получается на 1 больше, чем число символов, но массив остается двумерным. Оператор : позволяет легко выполнять операции по увеличению размерности массивов. Приведем пример на формирование трехмерного массива путем добавления новой страницы. Пусть у нас задан исходный двумерный массив М с размером 3×2:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
M =
     1     2     3
     4     5     6
     7     8     9
```

Для добавления новой страницы с тем же размером можно расширить М следующим образом:

```
>> M(:, :, 2)=[10 11 12; 13 14 15; 16 17 18]
M(:, :, 1) =
     1     2     3
     4     5     6
     7     8     9
M(:, :, 2) =
    10    11    12
    13    14    15
    16    17    18
```

Посмотрим, что теперь содержит массив М при явном его указании:

```
>> M
M(:, :, 1) =
     1     2     3
     4     5     6
     7     8     9
M(:, :, 2) =
    10    11    12
    13    14    15
    16    17    18
```

Как можно заметить, числа в выражениях $M(:, :, 1)$ и $M(:, :, 2)$ означают наличие в массиве двух страниц.

5.4.3. Удаление размерности у многомерного массива

Мы уже отмечали возможность удаления отдельных столбцов присвоением им значений пустого вектора-столбца `[]`. Этот прием нетрудно распространить на страницы и вообще размерности многомерного массива. Например, первую страницу полученного массива `M` можно удалить следующим образом:

```
>> M(:, :, 1) = []
M =
    10    11    12
    13    14    15
    16    17    18
```

Нетрудно заметить, что в этом массиве осталась только вторая страница и что размерность массива уменьшилась на 1 – он стал двумерным.

5.4.4. Доступ к отдельному элементу многомерного массива

Чтобы вызвать средний элемент сначала первой, а затем второй страницы, надо записать следующее:

```
>> M(2, 2, 1)
ans = 5
>> M(2, 2, 2)
ans = 14
```

Таким образом, в многомерных массивах используется то же правило индексации, что и в одномерных и двумерных. Произвольный элемент, например, трехмерного массива, задается как $M(i, j, k)$, где i – номер строки, j – номер столбца и k – номер страницы. Этот элемент можно вывести, а можно присвоить ему заданное значение x : $M(i, j, k) = x$.

5.4.5. Создание страниц, заполненных константами и случайными числами

Если после знака присваивания стоит численная константа, то соответствующая часть массива будет содержать элементы, содержащие данную константу. Например, создадим из массива M (см. пример выше) массив, у которого вторая страница содержит единицы:

```
>> M(:,:,2)=1
M(:,:,1) =
    10    11    12
    13    14    15
    16    17    18
M(:,:,2) =
     1     1     1
     1     1     1
     1     1     1
```

А теперь заменим первую страницу массива на страницу с нулевыми элементами:

```
>> M(:,:,1)=0
M(:,:,1) =
     0     0     0
     0     0     0
     0     0     0
M(:,:,2) =
     1     1     1
     1     1     1
     1     1     1
```

5.4.6. Функции *ones*, *zeros*, *rand* и *randn*

Функции *ones* (создание массивов с единичными элементами), *zeros* (создание массивов с нулевыми элементами) и *rand* или *randn* (создание массивов с элементами – случайными числами с равномерным и нормальным распределением) могут также использоваться для создания многомерных массивов. Примеры приводятся ниже:

```
>> E=ones(3,3,2)
E(:,:,1) =
     1     1     1
     1     1     1
     1     1     1
E(:,:,2) =
     1     1     1
     1     1     1
     1     1     1
```



```
>> Z=zeros(2,2,3)
Z(:,:,1) =
     0     0
     0     0
Z(:,:,2) =
     0     0
     0     0
Z(:,:,3) =
     0     0
     0     0

>> R=randn(3,2,2)
R(:,:,1) =
    -1.6656    -1.1465
     0.1253     1.1909
     0.2877     1.1892
R(:,:,2) =
    -0.0376    -0.1867
     0.3273     0.7258
     0.1746    -0.5883
```

Эти примеры достаточно очевидны и не требуют особых комментариев. Обратите, однако, внимание на легкость задания размеров массивов для каждой размерности. Кроме того, следует отметить, что если хотя бы одна размерность массива равна нулю, то массив будет пустым:

```
>> A=randn(3,3,3,0)
A = Empty array: 3-by-3-by-3-by-0
```

Как видно из данного примера, пустой массив возвращается с соответствующим комментарием.

5.4.7. Объединение многомерных массивов

Для создания многомерных массивов служит описанная ранее для матриц специальная функция конкатенации `cat`:

- `cat(DIM,A,B)` возвращает результат объединения двух массивов `A` и `B` вдоль размерности `DIM`;
- `cat(2,A,B)` возвращает массив `[A,B]`, объединенный по столбцам;
- `cat(1,A,B)` возвращает массив `[A;B]`, объединенный по строкам;
- `B=cat(DIM,A1,A2,...)` объединяет множество входных массивов `A1,A2,...` вдоль размерности `DIM`.

Функции `cat(DIM,C{:})` и `cat(DIM,C.FIELD)` обеспечивают объединение массива ячеек или массива записей (см. далее), содержащего числовые матрицы, в многомерный массив. Ниже приводятся примеры применения функции `cat`:

```
>> M1=[1 2;3 4]
M1 =
     1     2
     3     4
```

```

    3     4
>> M2=[5 6;7 8]
M2 =
    5     6
    7     8
>> cat(1,M1,M2)
ans =
    1     2
    3     4
    5     6
    7     8
>> cat(2,M1,M2)
ans =
    1     2     5     6
    3     4     7     8
>> M=cat(3,M1,M2)
M(:,:,1) =
    1     2
    3     4
M(:,:,2) =
    5     6
    7     8

```

5.4.8. Функция преобразования размеров многомерного массива reshape

Еще один путь создания многомерных массивов заключается в преобразовании их размеров. Для этого используется функция reshape в ряде форм записи:

```

reshape(A,m,n,p,...)   reshape(A,[m n p ...])
reshape(A,...,[ ],...) reshape(A,siz)

```

К примеру, в первых двух конструкциях эта функция возвращает многомерный массив (размера $m \times n \times p \dots$), сформированный из элементов массива A.

Рисунок 5.14 показывает пример выполнения функции reshape на примере создания массива размера 6×5 из двух массивов размера 3×5 .

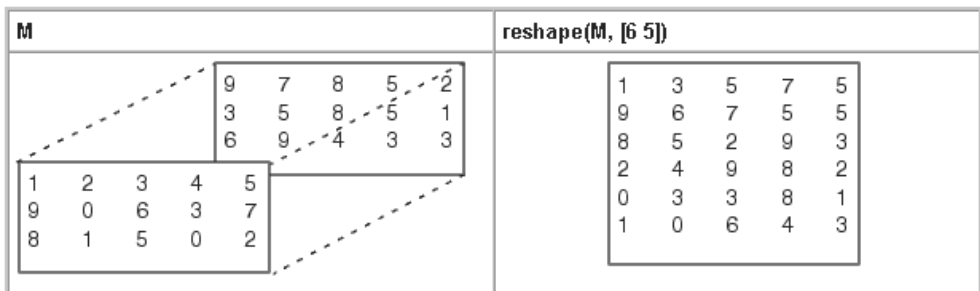


Рис. 5.14. Пример создания массива размера 6×5 из двух массивов размера 3×5

В другом примере, представленном на рис. 15.15, массив размера 6×2 создается из трех массивов размера 2×2 .

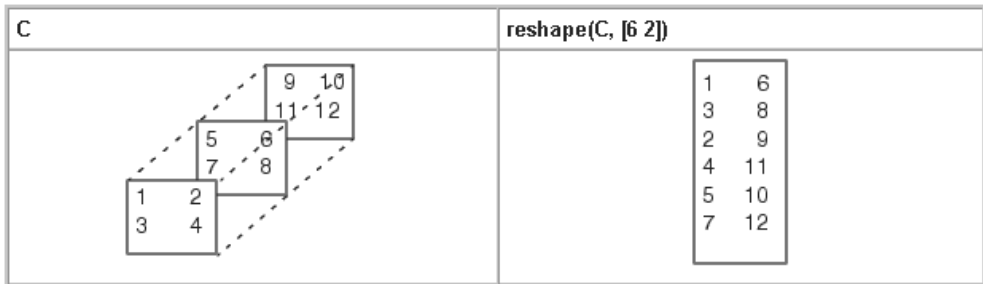


Рис. 5.15. Пример создания массива размера 6×2 из двух массивов размера 2×2

Существовавшая в прежних версиях MATLAB функция создания многомерных массивов из блоков `hermat` в описываемых версиях исключена.

5.5. Работа с размерностями массивов

5.5.1. Вычисление числа размерностей массива

Функция `ndims(A)` возвращает размерность массива `A` (если она больше или равна двум). Но если входной аргумент – массив Java, то независимо от размерности массива эта функция вернет 2. Следующий пример иллюстрирует применение функции `ndims`:

```
>> M=rand(2:3:4:5);
>> ndims(M)
ans = 4
```

5.5.2. Перестановки размерностей массивов

Если представить многомерный массив в виде страниц, то их перестановка является перестановкой размерностей массива. Для двумерного массива перестановка означает *транспонирование* – замену строк столбцами и наоборот. Следующие функции обеспечивают перестановку размерностей многомерных массивов:

- `permute(A, ORDER)` переставляет размерности массива `A` в порядке, определяемом вектором перестановок `ORDER`. Элементы вектора перестановок – числа от 1 до N , где N – размерность массива;
- `ipermute(A, ORDER)` делает то же, но в обратном порядке.

Ниже приводятся примеры применения этих функций:

```
>> A=[1 2; 3 4];
>> B=[5 6; 7 8];
>> C=[9 10;11 12];
>> D=cat(3,A,B,C)
D(:,:,1) =
     1     2
     3     4
D(:,:,2) =
     5     6
     7     8
D(:,:,3) =
     9    10
    11    12
>> size(D)
ans =
     2     2     3
>> size(permute(D,[3 2 1]))
ans =
     3     2     2
>> size(ipermute(D,[2 1 3]))
ans =
     2     2     3
```

5.5.3. Сдвиг размерностей массивов

Сдвиг размерностей реализуется функцией `shiftdim`:

- `B=shiftdim(X,N)` – сдвиг размерностей в массиве `X` на величину `N`. Если `N>0`, то сдвиг выполняется влево по кругу. Если `N<0`, сдвиг выполняется вправо, причем `N` первых размерностей становятся единичными;
- `[B,NSHIFTS]=shiftdim(X)` возвращает массив `B` с тем же числом элементов, что и у массива `X`, но с удаленными ведущими единичными размерностями. Выходной параметр `NSHIFTS` показывает число удаленных размерностей. Если `X` – скаляр, функция ничего не возвращает.

Следующий пример иллюстрирует применение функции `shiftdim`:

```
>> A=randn(1,2,3,4);
>> [B,N]=shiftdim(A)
B(:,:,1) =
    -2.1707    -1.0106     0.5077
    -0.0592     0.6145     1.6924
B(:,:,2) =
     0.5913     0.3803    -0.0195
    -0.6436    -1.0091    -0.0482
B(:,:,3) =
     0.0000     1.0950     0.4282
    -0.3179    -1.8740     0.8956
B(:,:,4) =
```

```

      0.7310      0.0403      0.5689
      0.5779      0.6771     -0.2556
N =      1

```

5.5.4. Удаление единичных размерностей

Функция `squeeze(A)` возвращает массив, в котором удалены все единичные размерности. Единичной называется размерность, в которой `size(A, dim)==1`. Но если `A` – одномерный или двумерный массив (матрица или вектор), то функция вернет тот же самый массив `A`. Следующий пример поясняет работу `squeeze`:

```

>> A=randn(1,2,1,3,1);
>> B=squeeze(A)
B =
      0.6145      1.6924     -0.6436
      0.5077      0.5913      0.3803

```

Обратите внимание на то, что пятимерный массив `A` превращается в массив с размерностью 2 и размером 2×3.

5.6. Тип данных – структуры

5.6.1. Структура записей

Структуры относятся к сложным типам данных. В предшествующих версиях MATLAB они именовались записями, что приводило к неточностям в терминологии системы MATLAB и баз данных. После того как в MATLAB были включены средства создания баз данных, этот тип данных стал именоваться структурами (*structures*). Они могут содержать разнородные данные, относящиеся к некоторому именованному объекту. Например, объект `man` (человек) может характеризоваться следующими данными:

| Поле | Значение | Комментарий |
|--------------------------|----------|--------------|
| <code>Man(I, ...)</code> | | Имя записи |
| <code>.name</code> | Иван | Имя человека |
| <code>.surname</code> | Петров | Фамилия |
| <code>.date</code> | 1956 | Год рождения |
| <code>.height</code> | 170.5 | Рост |
| <code>.weight</code> | 70.34 | Вес |

Первые два столбца представляют *схему структуры*. Как нетрудно заметить, каждая *i*-я структура состоит из ряда *полей*, имеющих *имена*, например `man(i).name`, `man(i).date` и т. д. Поля могут содержать *данные* любого типа – от пустого поля `[]` до массивов. Приведенная выше структура имеет размер 1×1. MATLAB поддерживает и массивы структур, что позволяет создавать мощные базы данных.

Поле структуры может содержать другую вложенную структуру или массив структур. Это позволяет создавать вложенные структуры и даже многомерные массивы структур.

5.6.2. Создание структур и доступ к их компонентам

Для задания структур на языке MATLAB можно использовать операторы присваивания, что иллюстрирует следующий пример:

```
>> man.name='Иван';  
>> man.surname='Петров';  
>> man.date=1956;  
>> man.height=170.5;  
>> man.weight=70.34;
```

Здесь построена базовая структура без индексного указателя. Теперь можно просмотреть полученную структуру, просто указав ее имя:

```
>> man  
man =  
    name: 'Иван'  
  surname: 'Петров'  
    date: 1956  
  height: 170.5000  
  weight: 70.3400
```

Нетрудно догадаться, что компоненты структуры можно вызывать по имени и менять их значения. При этом имя компонента состоит из имени структуры и имени поля, разделенных точкой. Это поясняют следующие примеры:

```
>> man.date  
ans =  
    1956  
>> man.date=1964  
man =  
    name: 'Иван'  
  surname: 'Петров'  
    date: 1964  
  height: 170.5000  
  weight: 70.3400
```

Примечание

В MATLAB есть старая проблема с записью символов кириллицы в командном режиме. Так, в командном режиме нельзя вводить в аргументы записей малую букву «с» русского алфавита – она ведет к переводу строки. Подобного ограничения нет при задании структур в программах, хотя и в этом случае ошибки не исключены. В новых реализациях MATLAB R2006/2007* этот недостаток устранен. Тем не менее применение символов кириллицы в идентификаторах переменных и функций недопустимо.*

Для создания массива структур вводится их *индексация*. Например, вектор структур можно создать, введя индекс в скобках после имени структуры. Так, для создания новой, второй структуры можно поступить следующим образом:

```
>> man(2).name='Петр';
>> man(2).surname='Сидоров';
>> man(2).date=1959;
>> man(2)
ans =
    name: 'Петр'
  surname: 'Сидоров'
    date: 1959
   height: [ ]
   weight: [ ]
>> man(2).surname
ans =
Сидоров
>> length(man)
ans =
    2
```

Обратите внимание на то, что не все поля данной структуры заполнены. Поэтому значением двух последних компонентов структуры 2 оказываются пустые массивы. Число структур позволяет найти функция `length` (см. последний пример).

5.6.3. Функция создания структур

Для *создания структур* используется следующая функция:

- `struct('field1',VALUES1,'field2',VALUES2,...)` возвращает созданную данной функцией структуру, содержащую указанные в параметрах поля `'fieldn'` с их значениями `'VALUESn'`. Значением может быть массив ячеек;
- `struct(OBJ)` конвертирует объект OBJ в эквивалентную структуру или массив структур. OBJ может быть объектом или массивом Java.

Пример:

```
>> S=struct('student','Иванов','grup',2,'estimate','good')
S =
    student: 'Иванов'
      grup: 2
  estimate: 'good'
```

5.6.4. Проверка имен полей и структур

Выполнение операций с полями и элементами полей выполняется по тем же правилам, что и при работе с обычными массивами. Однако существует ряд функций, осуществляющих специфические для структур операции.

Приведенные ниже функции служат для тестирования имен полей и структур записей:

- `isfields(S, 'field')` возвращает логическую 1, если 'field' является именем поля структуры S;
- `isstruct(S)` возвращает логическую 1, если S – структура, и 0 в ином случае.

Их применение на примере структуры `man` показано ниже:

```
>> isfield(man, 'name')
ans = 1
>> isfield(man, 'family')
ans = 0
>> isstruct(man)
ans = 1
>> isstruct(many)
??? Undefined function or variable 'many'.
>> isstruct('many')
ans = 0
```

5.7. Функции полей структур

5.7.1. Функция возврата имен полей

Следующая функция позволяет вывести имена полей заданной структуры:

- `fieldnames(S)` возвращает имена полей структуры S в виде массива ячеек. Пример:

```
>> fieldnames(man)
ans =
    'name'
    'surname'
    'date'
    'height'
    'weight'
```

5.7.2. Функция возврата содержимого полей структуры

В конечном счете работа со структурами сводится к выводу и использованию содержимого полей. Для возврата содержимого поля структуры S служит функция `getfield`:

- `getfield(S, 'field')` возвращает содержимое поля структуры S, что эквивалентно `S.field`;
- `getfield(S, {i, j}, 'field', {k})` эквивалентно `F=S(i, j).field(k)`.

Пример:

```
>> getfield(man(2), 'name')
ans = Петр
```


5.7.3. Функция присваивания значений полям

Для присваивания полям заданных значений используется следующая функция:

- `setfield(S, 'field', V)` возвращает структуру `S` с присвоением полю `'field'` значения `V`, что эквивалентно `S.field=V`;
- `setfield(S, {i, j}, 'field', {k}, V)` эквивалентно `S(i, j).field(k)=V`.

Пример:

```
>> setfield(man(2), 'name', 'Николай')
ans =
    name: 'Николай'
  surname: 'Сидоров'
    date: 1959
   height: [ ]
   weight: [ ]
```

5.7.4. Удаление полей

Для удаления полей структуры можно использовать следующую функцию:

- `rmfield(S, 'field')` возвращает структуру `S` с удаленным полем `'field'.S`;
- `rmfield(S, FIELDS)` возвращает структуру `S` с несколькими удаленными полями. Список удаляемых полей `FIELDS` задается в виде массива символов или строкового массива ячеек.

Пример:

```
>> rmfield(man(2), 'surname')
ans =
    name: 'Петр'
    date: 1959
   height: []
   weight: []
```

5.7.5. Применение массивов структур

Массивы структур находят самое широкое применение. Например, они используются для представления цветных изображений известного RGB. Они состоят из массивов интенсивности трех цветов – красного `r`, зеленого `g` и синего `b`. Еще более сложные структуры (но, в принципе, вполне очевидные) нужны для разработки баз данных, например о работниках предприятия, службах города, городах страны и т. д. Во всех этих случаях особенно важна возможность доступа к отдельным ячейкам структур и возможность присвоения таким структурам уникальных имен.

5.8. Массивы ячеек

5.8.1. Создание массивов ячеек

Массив ячеек – наиболее сложный тип данных в системе MATLAB. Это массив, элементами которого являются *ячейки*, содержащие любые типы массивов, включая массивы ячеек. Отличительным атрибутом массивов ячеек является задание содержимого последних в фигурных скобках `{ }`. Создавать массивы ячеек можно с помощью оператора присваивания.

Существуют два способа присваивания данных отдельным ячейкам:

- индексацией ячеек;
- индексацией содержимого.

Рассмотрим первый способ. Для этого создадим файл-сценарий с именем **се.m**:

```
A(1,1)={'КуриТЬ вредно!'};  
A(1,2)={[1 2;3 4]};  
A(2,1)={2+3i};  
A(2,2)={0:0.1:1}
```

Примечание

Уже отмечалось, что в командном режиме малая русская буква «с» в строках ведет к переводу строки ввода. Однако в файлах-сценариях, создаваемых в редакторе/отладчике М-файлов, эта недоработка не проявляется. Хотя гарантии в этом, увы, пока нет.

В этом примере задан массив ячеек с четырьмя элементами: строкой символов, матрицей, комплексным числом и одномерным массивом из 11 чисел. Теперь можно вызвать этот массив:

```
>> се  
A =  
    'КуриТЬ вредно!'    [2x2 double]  
    [2.0000+ 3.0000i]    [1x11 double]  
>> A(1,1)  
ans =  
    'КуриТЬ вредно!'  
>> A(2,1)  
ans = [2.0000+ 3.0000i]
```

Заметим, что к ячейкам такого массива можно обращаться с помощью индексирования, например в виде `A(1,1)`, `A(2,1)` и т. д.

При индексации содержимого массив ячеек задается следующим образом:

```
A{1,1}='КуриТЬ вредно!';  
A{1,2}=[1 2;3 4];  
A{2,1}=2+3i;  
A{2,2}=0:0.1:1;
```

Теперь можно ознакомиться с созданным массивом ячеек в командном режиме:

```
>> A
ans =
    'Курить вредно!'    [2x2 double]
    [2.0000+ 3.0000i]  [1x1 double]
>> A{1,1}
ans = Курить вредно!
>> A{2,1}
ans = 2.0000 + 3.0000i
```

При серьезной работе с массивами структур (записей) и массивами ячеек полезно иметь дополнительную информацию о списках значений. Для получения такой информации следует выполнить команду `help list`.

5.8.2. Создание ячеек с помощью функции `cell`

Для создания массива ячеек может использоваться функция `cell`:

- `cell(N)` создает массив ячеек из $N \times N$ пустых матриц;
- `cell(M,N)` или `cell([M,N])` создает массив ячеек из $M \times N$ пустых матриц;
- `cell(M,N,P,...)` или `cell([M N P ...])` создает массив из $M \times N \times P \times \dots$ пустых матриц;
- `cell(size(A))` создает массив ячеек из пустых матриц того же размера, что и массив `A`;
- `cell(объект Java)` автоматически преобразует объекты или массивы Java (`javaarray`) в массив ячеек, элементы которого являются объектами MATLAB.

Следующие примеры поясняют применение данной функции:

```
>> cell(2)
ans =
    [ ]    [ ]
    [ ]    [ ]
>> C=cell(2,3)
C =
    [ ]    [ ]    [ ]
    [ ]    [ ]    [ ]
>> C0=zeros(2,3)
C0 =
    0     0     0
    0     0     0
>> cell(size(C0))
ans =
    [ ]    [ ]    [ ]
    [ ]    [ ]    [ ]
```

Созданные пустые ячейки можно заполнить, используя операции присваивания:

```
>> C{1,1}=1;C{1,2}='Привет';C{2,1}='Hello';C{2,2}=[1 2; 3 4];
>> C
C =
      [      1]      'Привет'      [ ]
      'Hello'      [2x2 double]      [ ]
```

5.8.3. Визуализация массивов ячеек

Для отображения массива ячеек `C` служит команда `celldisp(C)`. Она дает рекурсивное отображение содержимого массива ячеек `C`. Например, для ранее созданного массива ячеек `A` получится следующее:

```
>> celldisp(A)
A{1,1} = Курить вредно!
A{2,1} = 2.0000 + 3.0000i
A{1,2} =
      1      2
      3      4
A{2,2} =
Columns 1 through 7
      0  0.1000  0.2000  0.3000  0.4000  0.5000  0.6000
Columns 8 through 11
      0.7000  0.8000  0.9000  1.0000
```

Для более наглядного графического представления массива ячеек может использоваться команда `cellplot`:

- `cellplot(C)` строит структуру массива ячеек `C`;
- `cellplot(C, 'legend')` строит структуру массива ячеек `C` вместе с «легендой» – шкалой стилей представления данных;
- `N=cellplot(C)` возвращает вектор дескрипторов созданных графических объектов.

На рис. 5.16 показано представление массива ячеек `A`, сформированного ранее.

Как видно на рис. 5.16, ячейки массива представлены квадратами. Векторы и матрицы с численными данными представляются массивами красного цвета с прямоугольными ячейками, при этом отображаются отдельные числа и текстовые данные. Справа от представления массива показана легенда, которая даже в монохромном изображении облегчает выделение типов компонент массива оттенками серого цвета.

5.8.4. Создание массива символьных ячеек из массива строк

Для создания из массива символов `S` строкового массива ячеек может использоваться функция `cellstr(S)`. Каждый ряд массива символов превращается в отдельную ячейку. Следующий пример поясняет применение функции `cellstr`:

```
>> S={'Привет' 'дорогой' 'друг'};
```

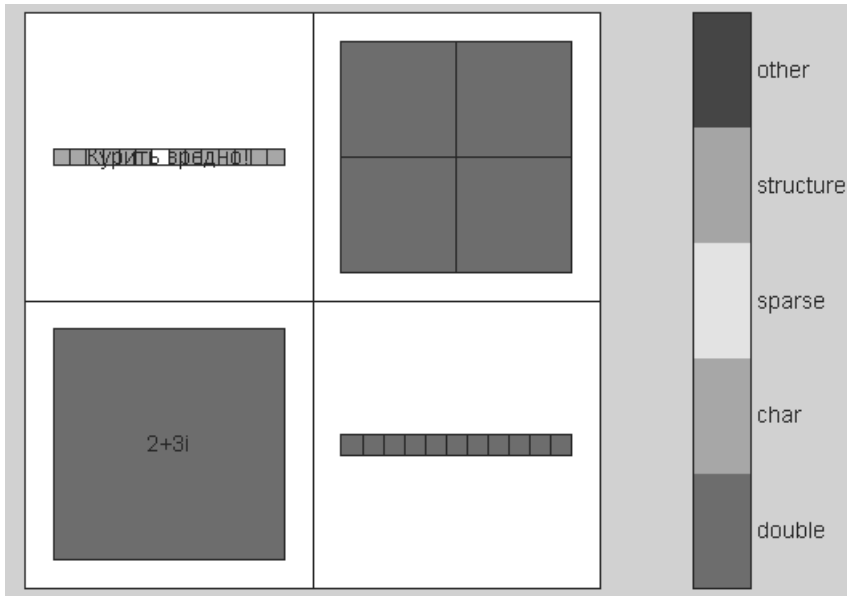


Рис. 5.16. Графическое представление массива с четырьмя ячейками

```
>> C=cellstr(S)
С = 'Привет' 'дорогой' 'друг'
```

Это еще один способ формирования массивов ячеек.

Функция `iscellstr(C)` равна 1, если ее аргумент `C` – строковый массив ячеек, и 0, если это неверно.

5.8.5. Присваивание с помощью функции `deal`

С помощью функции `deal` возможно множественное присваивание входных данных выходным:

- `[A, B, C, ...]=deal(X, Y, Z, ...)` обеспечивает последовательное присваивание входных данных выходным, то есть $A=X$, $B=Y$, $C=Z$ и т. д.;
- `[A, B, C, ...]=deal(X)` присваивает единственный вход всем выходам, то есть $A=X$, $B=X$, $C=X$ и т. д.

Возможен ряд полезных применений функции `deal`:

- `[S.FIELD]=deal(X)` присваивает всем полям `FIELD` структуры `S` значения `X`. Если `S` не существует, то нужно использовать конструкцию `[S(1:M).FIELD]=deal(X)`;
- `[X{:}]=deal(A.FIELD)` копирует поля `FIELD` структуры `A` в массив ячеек `X`. Если `X` не существует, следует использовать конструкцию `[X{1:M}]=deal(A.FIELD)`;
- `[A, B, C, ...]=deal(X{:})` копирует содержимое массива ячеек `X` в отдельные переменные `A`, `B`, `C`, ...;

- `[A, B, C, ...]=deal (S.FIELD)` копирует содержимое поля FIELD массива структур S в отдельные переменные A, B, C, ...

Следующий пример иллюстрирует применение функции deal:

```
>> [X, Y, Z]=deal (1, 2+3i, 'Привет! ')
X = 1
Y = 2.0000 + 3.0000i
Z =Привет!
>> [X Y Z]=deal ("Привет!")
X =Привет!
Y =Привет!
Z =Привет!
```

5.8.6. Тестирование имен массивов ячеек

Ввиду обилия типов данных в системе MATLAB часто возникает необходимость их *тестирования*. Для тестирования массивов ячеек может использоваться функция `iscell (C)`, которая возвращает логическое значение 1, если C – массив ячеек, и 0 в противном случае. Это поясняют следующие примеры:

```
>> t=iscell(A)
t = 1
>> B=[1 2 3];
>> iscell(B)
ans = 0
```

5.8.7. Функции преобразования типов данных

При обработке сложных данных возникает необходимость в *преобразовании* их типов. Ниже представлены такие функции, имеющие отношение к массивам ячеек:

- `num2cell (A)` преобразует массив чисел A в массив ячеек и возвращает последний. Возвращаемый массив имеет тот же размер, что и исходный массив A;
- `num2cell (A, DIM)` преобразует массив чисел A в массив ячеек, помещая в отдельные ячейки фрагменты, соответствующим разным значениям индекса вдоль измерения, указанного параметром DIM.

Примеры применения данной функции:

```
>> A=[1 2; 3 4; 5 6]
A =
     1     2
     3     4
     5     6
>> num2cell(A, 2)
ans =
    [1x2 double]
    [1x2 double]
```

```

        [1x2 double]
>> num2cell(A,[1 2])
ans = [3x2 double]

```

- `cell2struct(C,FIELDS,DIM)` преобразует массив ячеек `C` в массив структур вдоль размерности `DIM`, сохраняя размер массива `C` по этой размерности в записи структуры. Размерность 1 – столбцы. Размерность 2 – строки. Пример преобразования:

```

>> C={'Привет!',123,2+3i}
C = 'Привет!' [123] [2.0000+ 3.0000i]
>> f={'name','number','complex'};
>> S=cell2struct(C,f,2)
S =
    name: 'Привет!'
    number: 123
    complex: 2.0000+ 3.0000i

```

- `struct2cell(S)` преобразует массив структур `S` размером $m \times n$, в котором содержатся p полей, в массив ячеек, так что возвращаемый массив будет иметь размер $p \times m \times n$. Если массив записей многомерный, то возвращаемый массив будет иметь размер, равный `[p size(S)]`. Пример такого преобразования приводится ниже:

```

>> C=struct2cell(S)
C =
    'Привет!'
    [          123]
    [2.0000 + 3.0000i]

```

5.9. Многомерные массивы ячеек

5.9.1. Создание многомерных массивов ячеек

С помощью функции `cat` можно формировать *многомерные массивы ячеек*. Например, трехмерный массив `C` формируется следующим образом (m-файл с именем `ce2.m`):

```

A{1,1}='Курить вредно!';
A{1,2}=[1 2;3 4];
A{2,1}=2+3i;A{2,2}=0:0.1:1;
V{1,1}='Пить тоже вредно!';
V{1,2}=[1 2 3 4];
V{2,1}=2;
V{2,2}=2*pi;
C=cat(3,A,V);

```

Теперь можно посмотреть данный массив, имеющий две страницы:

```
>> ce2
```

```
>> C
C(:,:,1) =
    'Куриль вредно!'           [2x2 double]
    [2.0000+ 3.0000i]         [1x1 double]
C(:,:,2) =
    'Пить тоже вредно!'       [1x4 double]
    [                2]       [        6.2832]
```

Этот многомерный массив можно просмотреть с помощью команды `cellplot(C)`. Полученный результат показан на рис. 5.17, где многомерный массив отображается как стопка страниц.

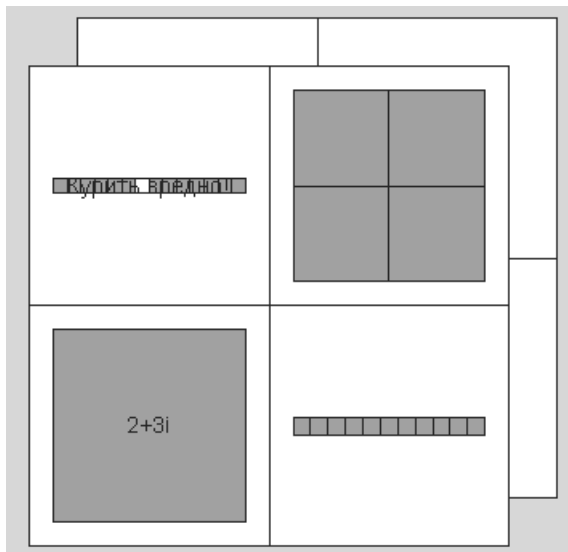


Рис. 5.17. Отображение трехмерного массива ячеек командой `cellplot`

Доступ к ячейкам многомерных массивов очевиден и поясняется следующими примерами:

```
>> C(1,1,1)
ans = 'Куриль вредно!'
>> C(1,1,2)
ans = 'Пить тоже вредно!'
```

5.9.2. Вложенные массивы ячеек

Содержимым ячейки массива ячеек может быть, в свою очередь, произвольный массив ячеек. Таким образом, возможно создание *вложенных* массивов ячеек – пожалуй, самого сложного типа данных. В следующем примере показано фор-

мирование массива ячеек A с вложенным в него массивом B (он был создан в примере выше):

```
>> clear A;
>> A(1,1)={magic(3), {'Hello!'}};
>> A(1,2)=B;
>> A
ans = {1x2 cell}    {2x2 cell}
>> A{1}
ans = [3x3 double] {1x1 cell}
>> A{2}
ans =
    'Пить тоже вредно!'    [1x4 double]
    [                2]    [        6.2832]
>> cellplot(A)
```

На рис. 5.18 показано отображение массива A с вложенным в него массивом B. В данном случае вложенный массив отображается полностью как часть массива A.

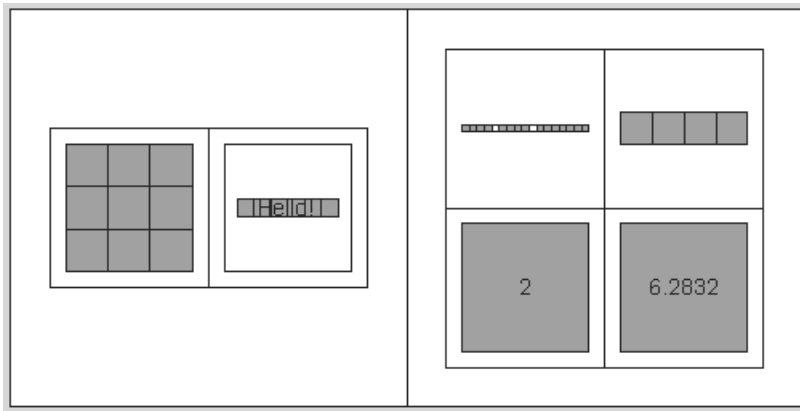


Рис. 5.18. Графическое представление массива с вложенным в него другим массивом

Программные средства обычной графики

| | |
|---|-----|
| 6.1. Графики функций и данных | 278 |
| 6.2. Визуализация в полярной системе координат | 289 |
| 6.3. Визуализация векторов | 291 |
| 6.4. Основы трехмерной графики | 293 |
| 6.5. Улучшенные средства визуализации 3D-графики | 302 |
| 6.6. Текстовое оформление графиков | 312 |
| 6.7. Форматирование графиков | 316 |
| 6.8. Цветовая окраска графиков | 327 |
| 6.9. Другие возможности графики | 336 |

Одно из достоинств системы MATLAB – обилие средств графики, начиная от команд построения простых графиков функций одной переменной в декартовой системе координат и заканчивая комбинированными и презентационными графиками с элементами анимации, а также средствами проектирования графического пользовательского интерфейса (GUI). Особое внимание в системе уделено трехмерной графике с функциональной окраской отображаемых фигур и имитацией различных световых эффектов. Этот урок посвящен описанию основных программных средств обычной графики системы MATLAB [13, 16, 46]. Показано построение графиков как командами, исполняемыми из командной строки, так и с помощью простых программ с линейной структурой. Математические основы машинной графики описаны в [66] и здесь не рассматриваются. Материал этого урока относится к любой реализации базовой системы MATLAB.

6.1. Графики функций и данных

6.1.1. Построение графиков отрезками прямых

Функции одной переменной $y(x)$ находят широкое применение в практике математических и других расчетов, а также в технике компьютерного математического моделирования. Для отображения таких функций используются *графики* в декартовой (прямоугольной) системе координат. При этом обычно строятся две оси – горизонтальная X и вертикальная Y , и задаются координаты x и y , определяющие узловые точки функции $y(x)$. Эти точки соединяются друг с другом отрезками прямых, то есть при построении графика осуществляется *линейная интерполяция* для промежуточных точек. Поскольку MATLAB – матричная система, совокупность точек $y(x)$ задается векторами X и Y одинакового размера.

Команда `plot` служит для построения графиков функций в декартовой системе координат. Эта команда имеет ряд параметров, рассматриваемых ниже.

- `plot(X, Y)` строит график функции $y(x)$, координаты точек (x, y) которой берутся из векторов одинакового размера Y и X . Если X или Y – матрица, то строится семейство графиков по данным, содержащимся в колонках матрицы.

Приведенный ниже пример иллюстрирует построение графиков двух функций – $\sin(x)$ и $\cos(x)$, значения функции которых содержатся в матрице Y , а значения аргумента x хранятся в векторе X :

```
>> x=[0 1 2 3 4 5]; Y=[sin(x);cos(x)]; plot(x,Y)
```

На рис. 6.1 показан график функций из этого примера. В данном случае отчетливо видно, что график состоит из отрезков, и если вам нужно, чтобы отображаемая функция имела вид гладкой кривой, необходимо увеличить количество узловых точек. Расположение их ординат может быть произвольным.

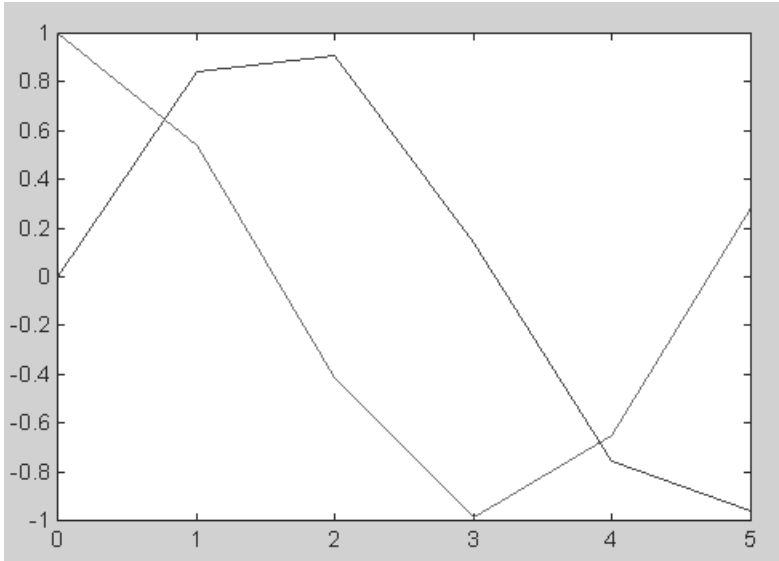


Рис. 6.1. Графики двух функций в декартовой системе координат

- `plot(Y)` строит график $y(i)$, где значения y берутся из вектора Y , а i представляет собой индекс соответствующего элемента. Если Y содержит комплексные элементы, то выполняется команда `plot(real(Y), imag(Y))`. Во всех других случаях мнимая часть данных игнорируется.

Вот пример использования команды `plot(Y)`:

```
>> x=-2*pi:0.02*pi:2*pi; y=sin(x)+i*cos(3*x); plot(y)
```

Соответствующий график показан на рис. 6.2.

- `plot(X, Y, S)` аналогична команде `plot(X, Y)`, но тип линии графика можно задавать с помощью строковой константы S . Значениями константы S могут быть следующие символы.

| Цвет линии | |
|------------|------------|
| y | Желтый |
| m | Фиолетовый |
| c | Голубой |
| r | Красный |
| g | Зеленый |
| b | Синий |
| w | Белый |
| k | Черный |
| Тип точки | |
| . | Точка |
| o | Окружность |

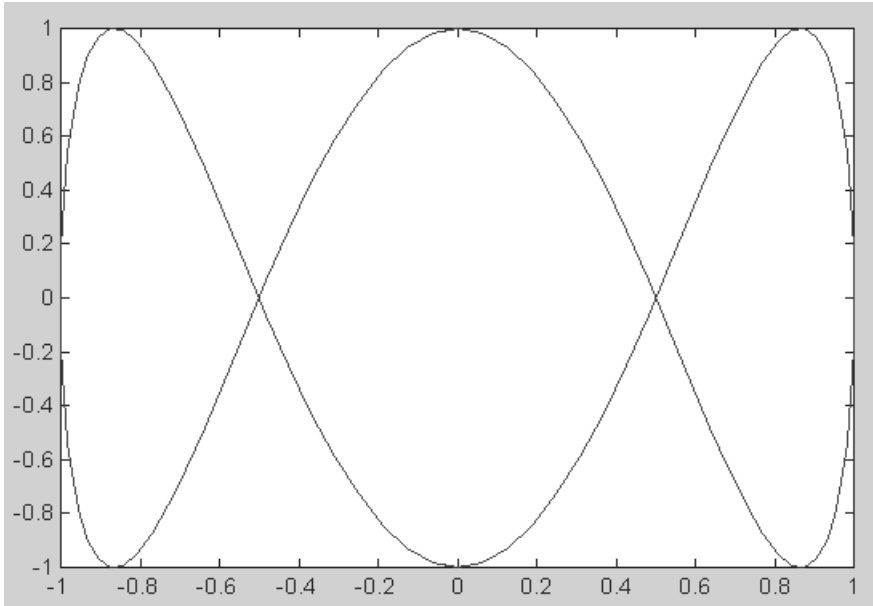


Рис. 6.2. График функции, представляющей вектор Y с комплексными элементами

| Тип точки | |
|-----------|----------------------|
| x | Крест |
| + | Плюс |
| * | Звездочка |
| s | Квадрат |
| d | Ромб |
| v | Треугольник (вниз) |
| ^ | Треугольник (вверх) |
| < | Треугольник (влево) |
| > | Треугольник (вправо) |
| p | Пятиугольник |
| h | Шестиугольник |
| Тип линии | |
| - | Сплошная |
| : | Двойной пунктир |
| -. | Штрих-пунктир |
| - | Штриховая |

Таким образом, с помощью строковой константы S можно изменять цвет линии, представлять узловые точки различными отметками (точка, окружность, крест, треугольник с разной ориентацией вершины и т. д.) и менять тип линии графика.

- `plot(X1, Y1, S1, X2, Y2, S2, X3, Y3, S3, ...)` – эта команда строит на одном графике ряд линий, представленных данными вида (X_i, Y_i, S_i) , где X_i и Y_i – векторы или матрицы, а S_i – строки. С помощью такой конструкции возможно построение, например, графика функции линией, цвет которой отличается от цвета узловых точек. Так, если надо построить график функции линией синего цвета с красными точками, то вначале надо задать построение графика с точками красного цвета (без линии), а затем графика только линии синего цвета (без точек).

При отсутствии указания на цвет линий и точек он выбирается автоматически из таблицы цветов (белый исключается). Если линий больше шести, то выбор цветов повторяется. Для монохромных систем линии выделяются стилем.

Рассмотрим пример простой программы для построения графиков трех функций с различным стилем представления каждой из них:

```
% Программа построения графиков трех функций
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x, y1, '-m', x, y2, '-.+r', x, y3, '-ok')
```

Эта программа является типичным скрипт-файлом. Графики функций при ее запуске (указанием заданного имени) показаны на рис. 6.3.

Здесь график функции y_1 строится сплошной фиолетовой линией, график y_2 строится штрих-пунктирной линией с точками в виде знака «плюс» красного цве-

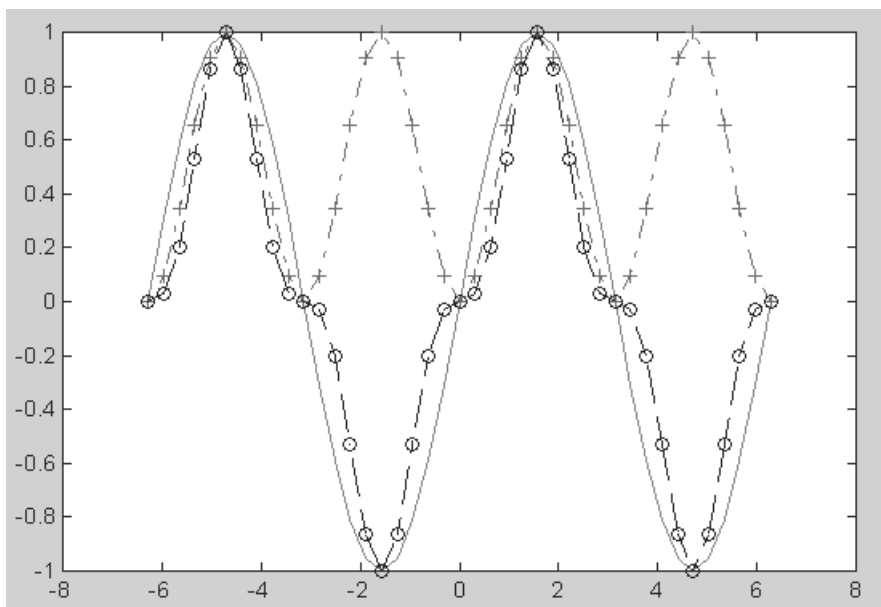


Рис. 6.3. Построение графиков трех функций на одном рисунке с разным стилем линий

та, а график y^3 строится штриховой линией с кружками черного цвета. К сожалению, на черно-белых рисунках этой книги вместо разных цветов видны разные градации серого цвета.

6.1.2. Графики в логарифмическом масштабе

Для построения графиков функций со значениями x и y , изменяющимися в широких пределах, нередко используются логарифмические масштабы. Рассмотрим команды, которые используются в таких случаях.

- `loglog(...)` – синтаксис команды аналогичен ранее рассмотренному для функции `plot(...)`. Логарифмический масштаб используется для координатных осей X и Y . Ниже дан пример применения данной команды:

```
>> x=logspace(-1,3); loglog(x,exp(x)./x); grid on
```

На рис. 6.4 представлен график функции $\exp(x)/x$ в логарифмическом масштабе. Обратите внимание на то, что командой `grid on` строится координатная сетка.

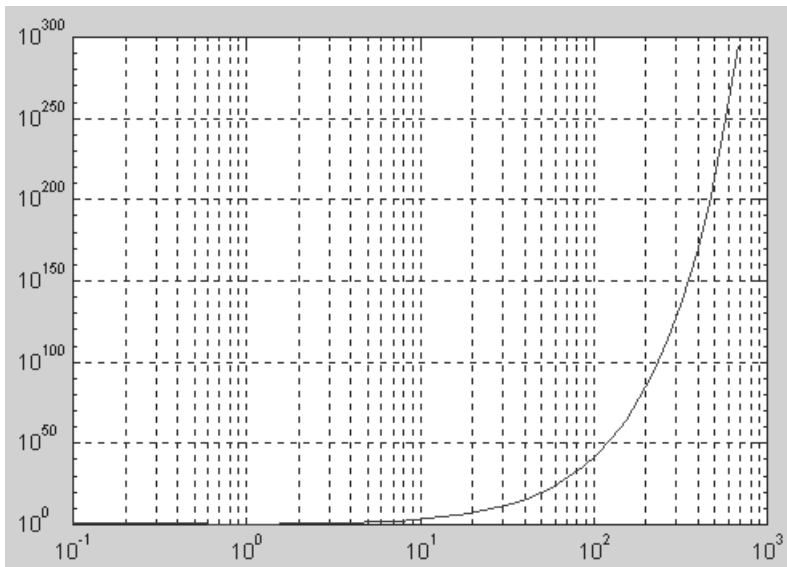


Рис. 6.4. График функции $\exp(x)/x$ в логарифмическом масштабе

Неравномерное расположение линий координатной сетки указывает на логарифмический масштаб осей.

6.1.3. Графики в полулогарифмическом масштабе

В некоторых случаях предпочтителен *полулогарифмический* масштаб графиков, когда по одной оси задается логарифмический масштаб, а по другой – линейный. Для построения графиков функций в полулогарифмическом масштабе используются следующие команды:

- `semilogx(...)` строит график функции в логарифмическом масштабе (основание 10) по оси X и линейном по оси Y ;
- `semilogy(...)` строит график функции в логарифмическом масштабе по оси Y и линейном по оси X .

Запись параметров (...) выполняется по аналогии с функцией `plot(...)`. Ниже приводится пример построения графика экспоненциальной функции:

```
>> x=0:0.5:10; semilogy(x,exp(x))
```

График функции при логарифмическом масштабе по оси Y представлен на рис. 6.5.

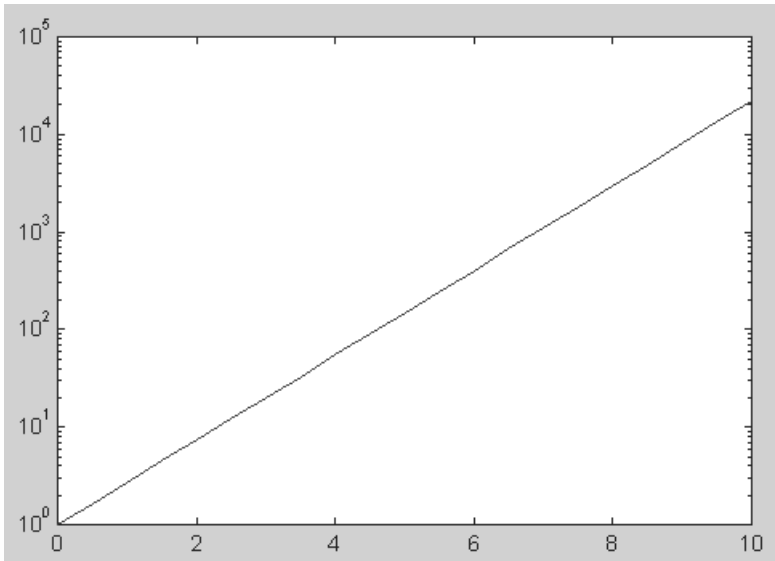


Рис. 6.5. График экспоненты в полулогарифмическом масштабе

Нетрудно заметить, что при таком масштабе график экспоненциальной функции выродился в прямую линию. Масштабной сетки теперь уже нет.

6.1.4. Столбцовые диаграммы

Столбцовые диаграммы широко используются в литературе, посвященной финансам и экономике, а также в математической литературе. Ниже представлены команды для построения таких диаграмм.

- `bar(x, Y)` строит столбцовый график элементов вектора Y (или группы столбцов для матрицы Y) со спецификацией положения столбцов, заданной значениями элементов вектора x , которые должны идти в монотонно возрастающем порядке;
- `bar(Y)` строит график значений элементов матрицы Y так же, как указано выше, но фактически для построения графика используется вектор $x=1:m$;
- `bar(x, Y, WIDTH)` или `BAR(Y, WIDTH)` – команда аналогична ранее рассмотренным, но со спецификацией ширины столбцов (при $WIDTH > 1$ столбцы перекрываются). По умолчанию задано $WIDTH = 0.8$.

Возможно применение этих команд и в следующем виде:

```
bar(..., 'Спецификация')
```

для задания спецификации графиков, например типа линий, цвета и т. д., по аналогии с командой `plot`. Спецификация `'stacked'` задает рисование всех n столбцов друг на друге.

Пример построения столбцовой диаграммы матрицы размером 12×3 приводится ниже:

```
>> subplot(2,1,1), bar(rand(12,3),'stacked'), colormap(cool)
```

На рис. 6.6 представлен полученный график.

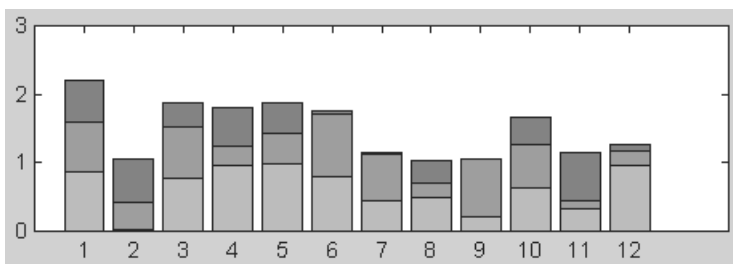


Рис. 6.6. Пример построения диаграммы с вертикальными столбцами

Помимо команды `bar(...)`, существует аналогичная ей по синтаксису команда `barh(...)`, которая строит столбцовые диаграммы с горизонтальным расположением столбцов. Пример, приведенный ниже, дает построения, показанные на рис. 6.7.

```
>> subplot(2,1,1), barh(rand(5,3),'stacked'), colormap(cool)
```

Какое именно расположение столбцов выбрать, зависит от пользователя, использующего эти команды для представления своих данных.

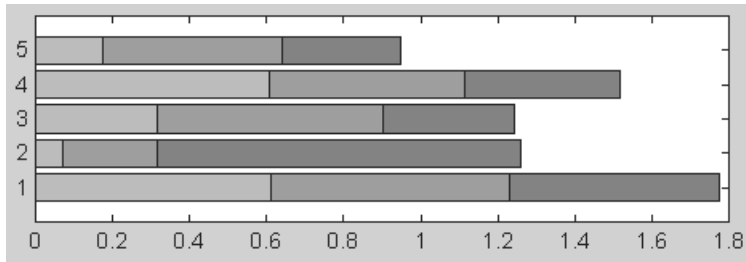


Рис. 6.7. Пример построения столбцовой диаграммы с горизонтальными столбцами

6.1.5. Гистограммы

Классическая *гистограмма* характеризует графически числа попаданий значений элементов вектора Y в M интервалов с представлением этих чисел в виде столбцовой диаграммы. Для получения данных для гистограммы служит функция `hist`, записываемая в следующем виде:

- $N = \text{hist}(Y)$ возвращает вектор чисел попаданий для 10 интервалов, выбираемых автоматически. Если Y – матрица, то выдается массив данных о числе попаданий для каждого из ее столбцов;
- $N = \text{hist}(Y, M)$ аналогична вышерассмотренной, но используется M интервалов (M – скаляр);
- $N = \text{hist}(Y, X)$ возвращает числа попаданий элементов вектора Y в интервалы, центры которых заданы элементами вектора X ;
- $[N, X] = \text{HIST}(\dots)$ возвращает числа попаданий в интервалы и данные о центрах интервалов.

Команда `hist(...)` с синтаксисом, аналогичным приведенному выше, строит график гистограммы. В следующем примере строится гистограмма для 1000 случайных чисел и выводится вектор с данными о числах их попаданий в интервалы, заданные вектором x :

```
>> x=-3:0.2:3; y=randn(1000,1);
>> hist(y,x); h=hist(y,x)
h =
Columns 1 through 12
0    0    3    7    8    9   11   23   33   43   57   55
Columns 13 through 24
70   62   83   87   93   68   70   65   41   35   27   21
Columns 25 through 31
12    5    6    3    2    1    0
```

Построенная гистограмма показана на рис. 6.8.

Нетрудно заметить, что распределение случайных чисел близко к нормальному закону. Увеличив их количество, можно наблюдать еще большее соответствие этому закону.

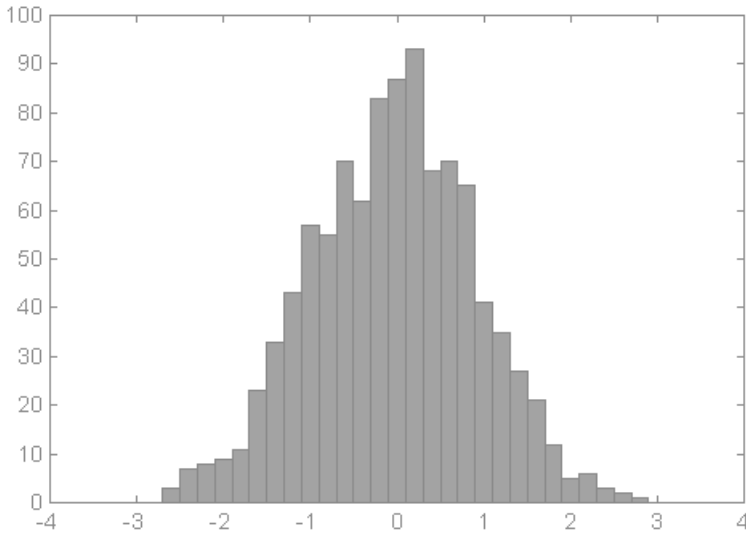


Рис. 6.8. Пример построения гистограммы

6.1.6. Лестничные графики

Лестничные графики визуально представляют собой ступеньки с огибающей, представленной функцией $y(x)$. Такие графики используются, например, для отображения процессов квантования функции $y(x)$, представленной рядом своих отсчетов. При этом в промежутках между отсчетами значения функции считаются постоянными и равными величине последнего отсчета.

Для построения лестничных графиков в системе MATLAB используются команды группы `stairs`:

- `stairs(Y)` строит лестничный график по данным вектора Y ;
- `stairs(X, Y)` строит лестничный график по данным вектора Y с координатами x переходов от ступеньки к ступеньке, заданными значениями элементов вектора X ;
- `stairs(..., S)` аналогична по действию вышеописанным командам, но строит график линиями, стиль которых задается строками S .

Следующий пример иллюстрирует построение лестничного графика:

```
>> x=0:0.25:10; stairs(x, x.^2);
```

Результат построения представлен на рис. 6.9.

Обратите внимание на то, что отсчеты берутся через равные промежутки по горизонтальной оси. Если, к примеру, отображается функция времени, то `stairs` имеет вид квантованной по времени функции.

Функция `H=stairs(X, Y)` возвращает вектор дескрипторов графических объектов. Функция

```
[XX, YY]=stairs(X, Y)
```

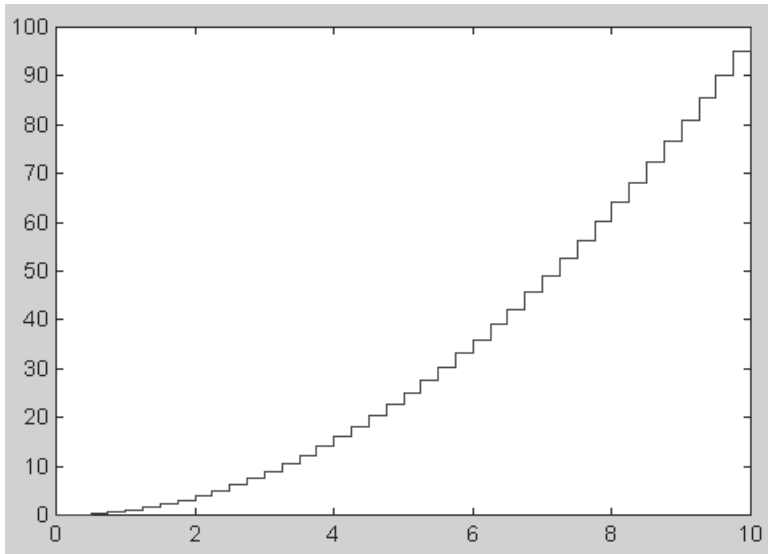


Рис. 6.9. Лестничный график функции x^2

сама по себе графика не строит, а возвращает векторы XX и YY , которые позволяют построить график с помощью команды `plot(XX, YY)`.

6.1.7. Графики с зонами погрешности

Если данные для построения функции определены с заметной погрешностью, то используют графики функций типа `errorbar` с оценкой погрешности каждой точки путем ее представления в виде буквы I , высота которой соответствует заданной погрешности представления точки. Команда `errorbar` используется в следующем виде:

- `errorbar(X, Y, L, U)` строит график значений элементов вектора Y в зависимости от данных, содержащихся в векторе X , с указанием нижней и верхней границ значений, заданных в векторах L и U ;
- `errorbar(X, Y, E)` и `errorbar(Y, E)` строит графики функции $Y(X)$ с указанием этих границ в виде $[Y-E \ Y+E]$, где E – погрешность;
- `errorbar(..., 'LineStyle')` аналогична описанным выше командам, но позволяет строить линии со спецификацией `'LineStyle'`, аналогичной спецификации, примененной в команде `plot`.

Следующий пример иллюстрирует применение команды `errorbar`:

```
>> x=-2:0.1:2; y=erf(x);  
>> e = rand(size(x))/10; errorbar(x,y,e);
```

Построенный график показан на рис. 6.10.

Функция, записываемая в виде `H=ERRORBAR(...)`, возвращает вектор дескрипторов графических объектов.

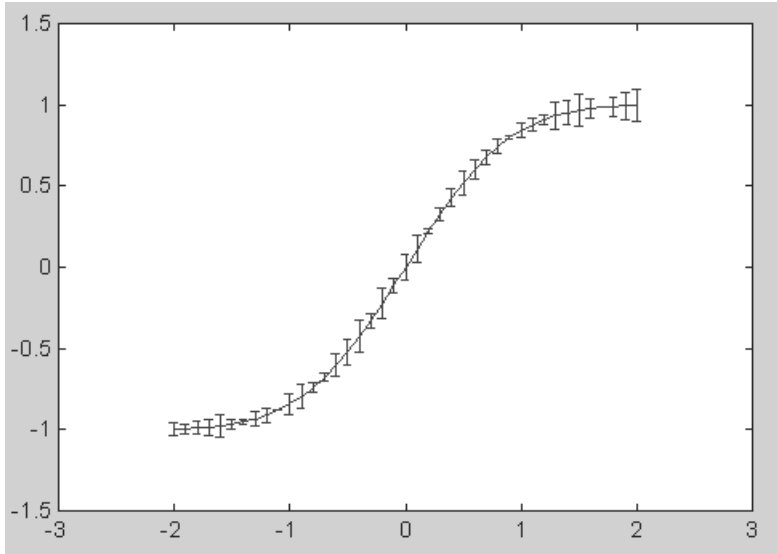


Рис. 6.10. График функции $\text{erf}(x)$ с зонами погрешности

6.1.8. Графики дискретных отсчетов функции

Еще один вид графика функции $y(x)$ — ее представление дискретными отсчетами. Этот вид графика применяется, например, при описании квантования сигналов. Каждый отсчет представляется вертикальной чертой, увенчанной кружком, причем высота черты соответствует y -координате точки.

Для построения графика подобного вида используются команды `stem(...)`:

- `stem(Y)` строит график функции с ординатами в векторе Y в виде отсчетов;
- `stem(X, Y)` строит график отсчетов с ординатами в векторе Y и абсциссами в векторе X ;
- `stem(..., 'filled')` строит график функции с закрашенными маркерами;
- `stem(..., 'LINESPEC')` дает построения, аналогичные ранее приведенным командам, но со спецификацией линий 'LINESPEC', подобной спецификации, приведенной для функции `plot`.

Следующий пример иллюстрирует применение команды `stem`:

```
>> x = 0:0.1:4; y = sin(x.^2).*exp(-x); stem(x,y)
```

Полученный для данного примера график показан на рис. 6.11.

Функция `N=STEM(...)` строит график и возвращает вектор дескрипторов графических объектов.

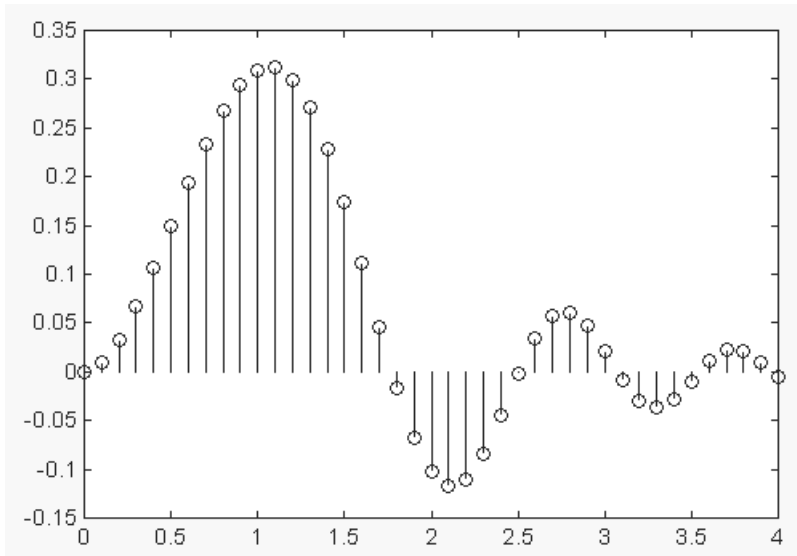


Рис. 6.11. График дискретных отсчетов функции

6.2. Визуализация в полярной системе координат

6.2.1. Графики в полярной системе координат

В *полярной системе координат* любая точка представляется как конец радиус-вектора, исходящего из начала системы координат, имеющего длину RHO и угол $THETA$. Для построения графика функции $RHO(THETA)$ используются приведенные ниже команды. Угол $THETA$ обычно меняется от 0 до $2 \cdot \pi$. Для построения графиков функций в полярной системе координат используются команды типа `polar(...)`:

- `polar(THETA, RHO)` строит график в полярной системе координат, представляющий собой положение конца радиус-вектора с длиной RHO и углом $THETA$;
- `polar(THETA, RHO, S)` аналогична предыдущей команде, но позволяет задавать стиль построения с помощью строковой константы S по аналогии с командой `plot`.

Рисунок 6.12 демонстрирует результат выполнения команд:

```
>> t=0:pi/50:2*pi; polar(t, sin(5*t))
```

Графики функций в полярных координатах могут иметь весьма разнообразный вид, порой напоминая такие объекты природы, как снежинки или кристалли-

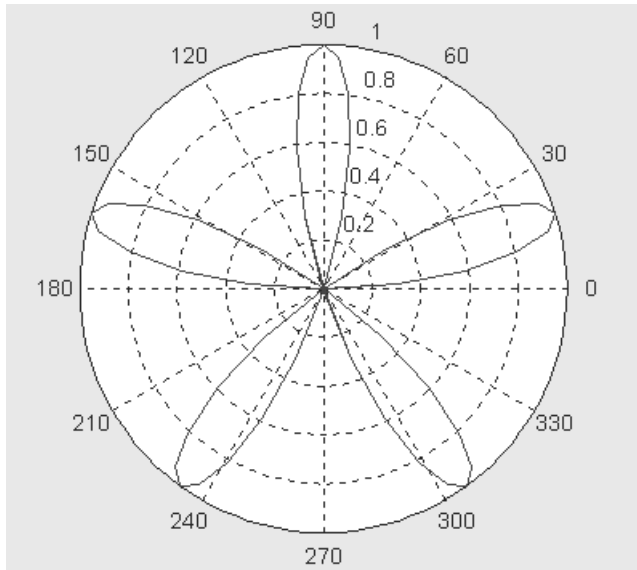


Рис. 6.12. График функции в полярной системе координат

ки льда на стекле. Вы можете сами попробовать построить несколько таких графиков – многие получают от этого удовольствие.

6.2.2. Угловые гистограммы

Угловые гистограммы находят применение в индикаторах радиолокационных станций, для отображения «роз» ветров и при построении других специальных графиков. Для этого используется ряд команд типа `rose (...)`:

- `rose (ТНЕТА)` строит угловую гистограмму для 20 интервалов по данным вектора ТНЕТА;
- `rose (ТНЕТА, N)` строит угловую гистограмму для N интервалов в пределах угла от 0 до $2 \cdot \pi$ по данным вектора ТНЕТА;
- `rose (ТНЕТА, X)` строит угловую гистограмму по данным вектора ТНЕТА со спецификацией интервалов, указанной в векторе X.

Следующий пример иллюстрирует применение команды `rose`:

```
>> rose(1:100,12)
```

На рис. 6.13 показан пример построения графика командой `rose`.

Функция `H=rose (...)` строит график и возвращает вектор дескрипторов графических объектов, а функция `[T,R]=rose (...)` сама по себе график не строит, но возвращает векторы T и R, которые нужны команде `polar (T, R)` для построения подобной гистограммы.

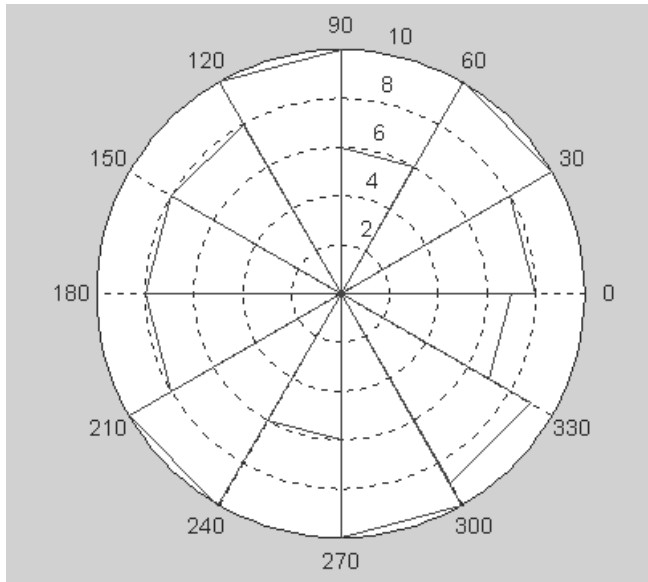


Рис. 6.13. Угловая гистограмма

6.3. Визуализация векторов

6.3.1. Графики векторов

Иногда желательно представление ряда радиус-векторов в их обычном виде, то есть в виде стрелок, исходящих из начала координат и имеющих угол и длину, определяемые действительной и мнимой частями комплексных чисел, представляющих эти векторы. Для этого служит группа команд `compass`:

- `compass(U, V)` строит графики радиус-векторов с компонентами (U, V) , представляющими действительную и мнимую части каждого из радиус-векторов;
- `compass(Z)` эквивалентно `compass(real(Z), imag(Z))`;
- `compass(U, V, LINESPEC)` и `compass(Z, LINESPEC)` аналогичны представленным выше командам, но позволяют задавать спецификацию линий построения `LINESPEC`, подобную описанной для команды `plot`.

В следующем примере показано использование команды `compass`:

```
>> z=[-1+2i, -2-3i, 2+3i, 5+2i]; compass(z)
```

Построенный в этом примере график представлен на рис. 6.14.

Функция `H=COMPASS(...)` строит график и возвращает дескрипторы графических объектов.

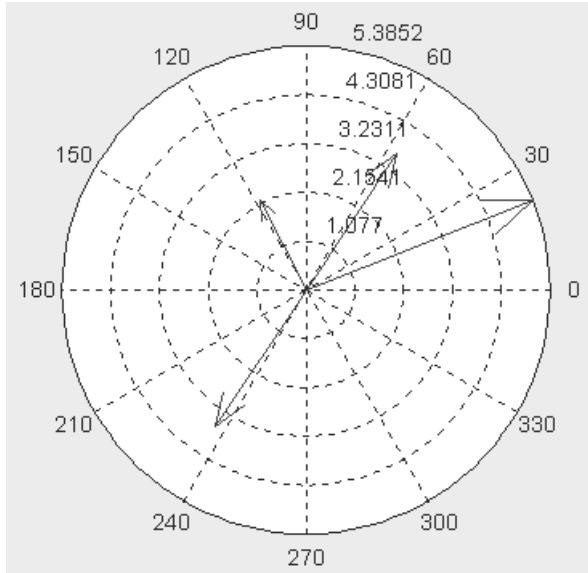


Рис. 6.14. Построение радиус-векторов

6.3.2. График проекций векторов на плоскость

Иногда полезно отображать комплексные величины вида $z = x + yi$ в виде проекции радиус-вектора на плоскость. Для этого используется семейство графических команд класса `feather`:

- `feather (U, V)` строит график проекции векторов, заданных компонентами U и V , на плоскость;
- `feather (Z)` для вектора Z с комплексными элементами дает построения, аналогичные `feather (REAL (Z) , IMAG (Z))`;
- `feather (... , S)` дает построения, описанные выше, но со спецификацией линий, заданной строковой константой S по аналогии с командой `plot`.

Пример применения команды `feather`:

```
>> x=0:0.1*pi:3*pi; y=0.05+i; z=exp(x*y); feather(z)
```

График, построенный в этом последнем примере, показан на рис. 6.15.

Функция `H=FEATHER (...)` строит график и возвращает вектор дескрипторов графических объектов.

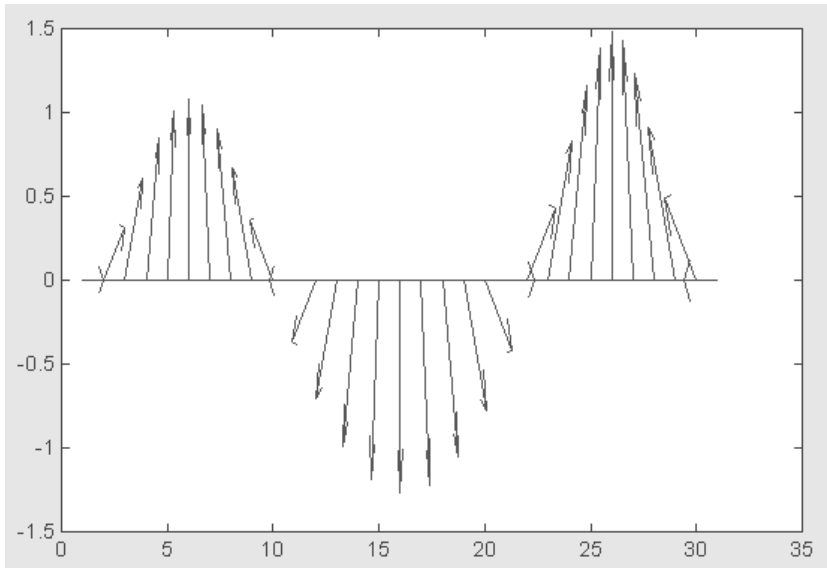


Рис. 6. 15. График, построенный командой *feather*

6.4. Основы трехмерной графики

6.4.1. Контурные графики

Контурные графики служат для представления на плоскости функции двух переменных вида $z(x,y)$ с помощью линий равного уровня. Они получаются, если трехмерная поверхность пересекается рядом плоскостей, расположенных параллельно друг другу. При этом контурный график представляет собой совокупность спроецированных на плоскость (x,y) линий пересечения поверхности $z(x,y)$ плоскостями. Типичный пример контурных графиков – обычные карты.

Для построения контурных графиков используются команды *contour*:

- *contour* (Z) строит контурный график по данным матрицы Z с автоматическим заданием диапазонов изменения x и y ;
- *contour* (X, Y, Z) строит контурный график по данным матрицы Z с указанием спецификаций для X и Y;
- *contour* (Z, N) и *contour* (X, Y, Z, N) дает построения, аналогичные ранее описанным командам, с заданием N линий равного уровня (по умолчанию N=10);
- *contour* (Z, V) и *contour* (X, Y, Z, V) строят линии равного уровня для высот, указанных значениями элементов вектора V;
- *contour* (Z, [v v]) или *contour* (X, Y, Z, [v v]) вычисляет одиночный контур для уровня v;

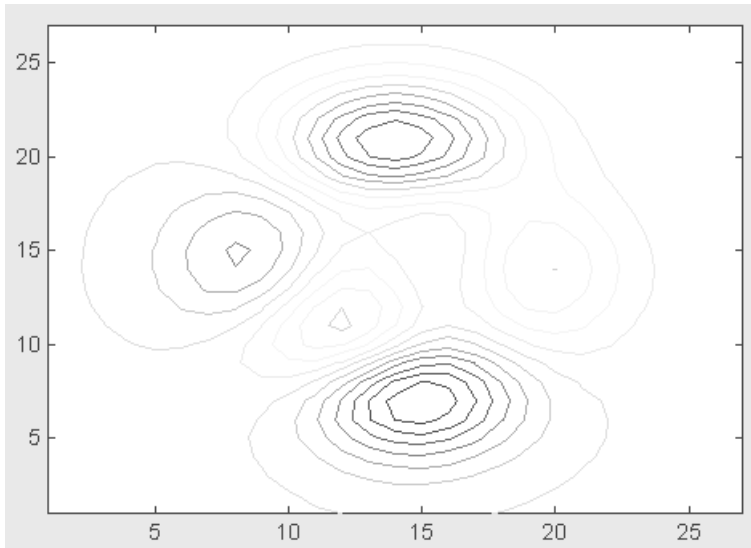


Рис. 6.16. Контурный график,
построенный с помощью команды `contour`

- `[C, H] = contour (...)` возвращает дескрипторы – матрицу **C** и вектор-столбец **H**. Они могут использоваться как входные параметры для команды `clabel`;
- `contour (... , 'LINESPEC')` позволяет использовать перечисленные выше команды с указанием спецификации линий, которыми идет построение.

Пример построения контурного графика поверхности, заданной функцией `peaks`:

```
>> z=peaks(27); contour(z,15)
```

Построенный в этом примере график показан на рис. 6.16. Заметим, что объект – функция `peaks` – задан в системе в готовом виде.

Графики этого типа часто используются в топографии для представления на листе бумаги (как говорят математики – на плоскости) объемного рельефа местности. Для оценки высот контурных линий используется их функциональная окраска.

6.4.2. Создание массивов данных для трехмерной графики

Поверхности как объекты трехмерной графики обычно описываются функцией двух переменных $z(x,y)$. Специфика построения трехмерных графиков требует не просто задания ряда значений x и y , то есть векторов x и y . Она требует определения для X и Y двумерных массивов – матриц. Для создания таких массивов слу-

жит функция `meshgrid`. В основном она используется совместно с функциями построения графиков трехмерных поверхностей. Функция `meshgrid` записывается в следующих формах:

- $[X, Y] = \text{meshgrid}(x, y)$ преобразует область, заданную векторами x и y , в массивы X и Y , которые могут быть использованы для вычисления функции двух переменных и построения трехмерных графиков. Строки выходного массива X являются копиями вектора x ; а столбцы Y – копиями вектора y ;
- $[X, Y] = \text{meshgrid}(x)$ аналогична $[X, Y] = \text{meshgrid}(x, x)$;
- $[X, Y, Z] = \text{meshgrid}(x, y, z)$ возвращает трехмерные массивы, используемые для вычисления функций трех переменных и построения трехмерных графиков.

Пример:

```
>> [X, Y] = meshgrid(1:4, 13:17)
X =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4
Y =
    13    13    13    13
    14    14    14    14
    15    15    15    15
    16    16    16    16
    17    17    17    17
```

Приведем еще один пример применения функции `meshgrid`:

```
>> [X, Y] = meshgrid(-2:.2:2, -2:.2:2);
```

Такой вызов функции позволяет задать опорную плоскость для построения трехмерной поверхности при изменении x и y от -2 до 2 с шагом $0,2$. Дополнительные примеры применения функции `meshgrid` будут приведены далее при описании соответствующих команд. Рекомендуется ознакомиться также с командами `surf` и `slice` (ломтик).

Функция `ndgrid` является многомерным аналогом функции `meshgrid`:

- $[X1, X2, X3, \dots] = \text{ndgrid}(x1, x2, x3, \dots)$ преобразует область, заданную векторами $x1, x2, x3, \dots$, в массивы $X1, X2, X3, \dots$, которые могут быть использованы для вычисления функций нескольких переменных и многомерной интерполяции. i -я размерность выходного массива Xi является копией вектора xi ;
- $[X1, X2, \dots] = \text{ndgrid}(x)$ аналогична $[X1, X2, \dots] = \text{ndgrid}(x, x, \dots)$.

Пример применения функции `ndgrid` представлен ниже:

```
[X1, X2] = ndgrid(-2:.2:2, -2:.2:2);
Z = X1 .* exp(-X1.^2 - X2.^2); mesh(Z)
```

Рекомендуем читателю опробовать действие этого примера.

6.4.3. Графики поля градиентов

Для построения *графиков полей градиента* служат команды `quiver`:

- `quiver(X,Y,U,V)` строит график поля градиентов в виде стрелок для каждой пары элементов массивов X и Y , причем элементы массивов U и V указывают направление и размер стрелок;
- `quiver(U,V)` строит векторы скорости в равнорасположенных точках на плоскости (x,y) ;
- `quiver(U,V,S)` или `quiver(X,Y,U,V,S)` автоматически масштабирует стрелки по сетке и затем вытягивает их по значению S . Используйте $S=0$, чтобы построить стрелки без автоматического масштабирования;
- `quiver(...,LINESPEC)` использует для векторов указанный тип линии. Указанные в `LINESPEC` маркеры рисуются у оснований, а не на концах векторов. Для отмены любого вида маркера используйте спецификацию `'.'`. Спецификации линий, цветов и маркеров были подробно описаны в разделе, посвященном команде `plot`;
- `quiver(...,'filled')` дает график с закрашенными маркерами;
- `H=quiver(...)` строит график и возвращает вектор дескрипторов.

Ниже представлен пример программы с применением команды `quiver`:

```
% Программа построения графика поля градиентов
x = -2:.2:2; y = -1:.2:1;
[xx,yy] = meshgrid(x,y);
zz = xx.*exp(-xx.^2-yy.^2);
[px,py] = gradient(zz,.2,.2);
quiver(x,y,px,py,2);
```

Построенный с помощью этой программы график показан на рис. 6.17.

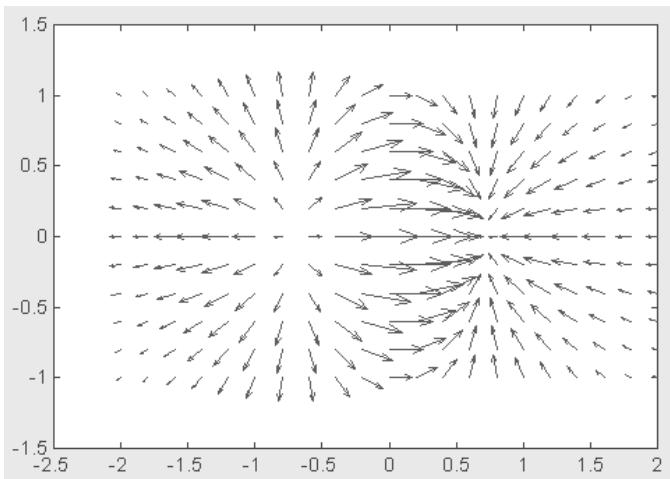


Рис. 6.17. Пример построения графика поля градиентов

Нетрудно заметить, что представление поля градиентов стрелками дает весьма наглядное представление о линиях поля, указывая области, куда эти линии впадают и откуда они исходят.

6.4.4. Графики поверхностей

Команда `plot3(...)` является аналогом команды `plot(...)`, но относится к функции двух переменных $z(x,y)$. Она строит аксонометрическое изображение трехмерных поверхностей и представлена следующими формами:

- `plot3(x, y, z)` строит массив точек, представленных векторами x , y и z , соединяя их отрезками прямых. Эта команда имеет ограниченное применение;
- `plot3(X, Y, Z)`, где X , Y и Z – три матрицы одинакового размера, строит точки с координатами $X(i, :)$, $Y(i, :)$ и $Z(i, :)$ и соединяет их отрезками прямых.

Ниже дан пример программы построения трехмерной поверхности, описываемой функцией $z(x, y) = x^2 + y^2$:

```
% Программа построения поверхности линиями
[X, Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X, Y, Z)
```

График этой поверхности показан на рис. 6.18.

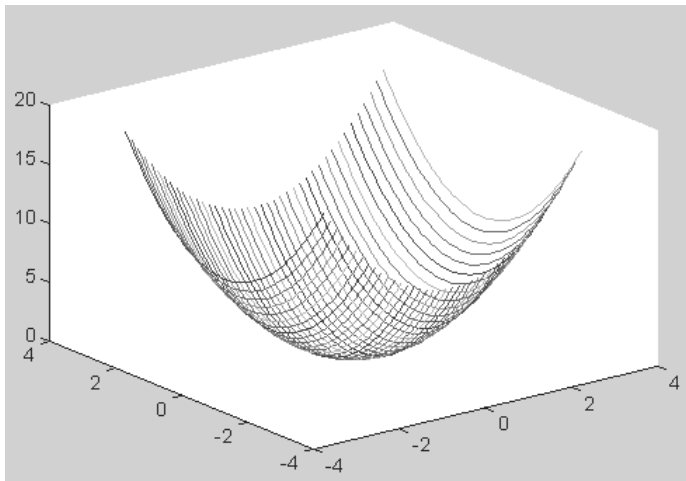


Рис. 6.18. График поверхности, построенный линиями

- `plot3(X, Y, Z, S)` обеспечивает построения, аналогичные рассмотренным ранее, но со спецификацией стиля линий и точек, соответствующей спецификации команды `plot`. Ниже дан пример применения этой команды для построения поверхности кружками:

```
% Программа построения поверхности кружками
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z,'o')
```

График поверхности, построенный кружками, показан на рис. 6.19.

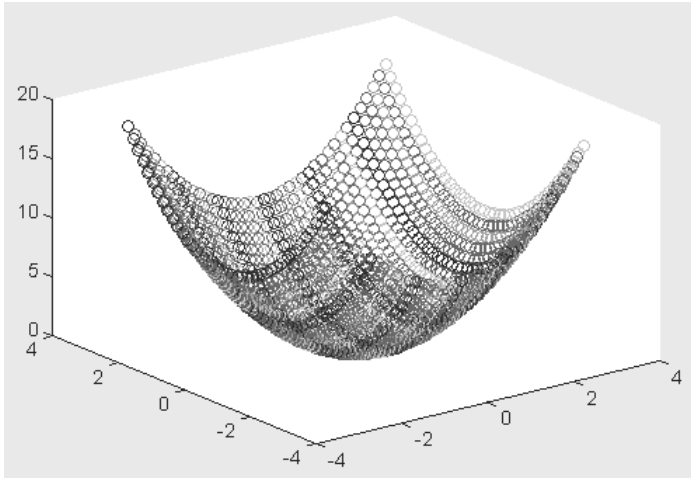


Рис. 6.19. График поверхности, построенный разноцветными кружками

- `plot3(x1,y1,z1,s1,x2,y2,z2,s2,x3,y3,z3,s3,...)` строит на одном рисунке графики нескольких функций $z_1(x_1, y_1)$, $z_2(x_2, y_2)$ и т. д. со спецификацией линий и маркеров каждой из них.

Пример применения последней команды дан ниже:

```
% Программа построения сетчатого графика функции
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z,'-k',Y,X,Z,'-k')
```

График функции, соответствующей последнему примеру, представлен на рис. 6.20.

В данном случае строятся два графика одной и той же функции с взаимно перпендикулярными образующими линиями. Поэтому график имеет вид сетки без окраски ее ячеек (напоминает проволочный каркас фигуры).

6.4.5. Сетчатые 3D-графики с окраской

Наиболее представительными и наглядными являются *сетчатые графики* поверхностей с заданной или функциональной окраской. В названии их команд присутствует слово `mesh`. Имеются три группы таких команд. Ниже приведены данные

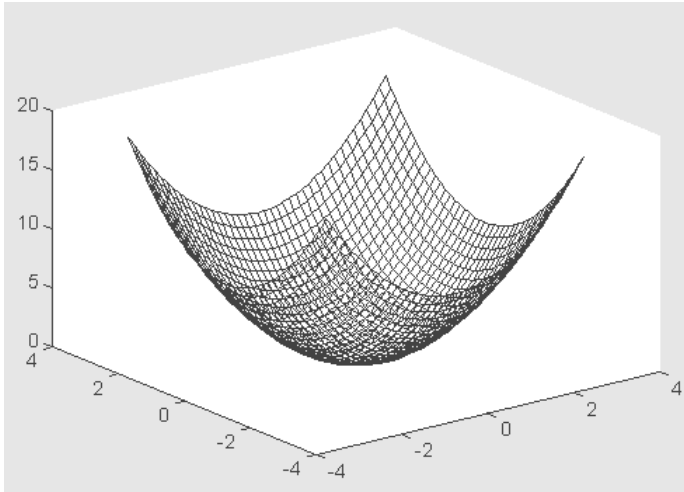


Рис. 6.20. График функции в сетчатом представлении

о наиболее полных формах этих команд. Наличие более простых форм можно уточнить, используя команду `help Имя`, где `Имя` – имя соответствующей команды.

- `mesh(X, Y, Z, C)` выводит в графическое окно сетчатую поверхность $Z(X, Y)$ с цветами узлов поверхности, заданными массивом `C`.
- `mesh(X, Y, Z)` – аналог предшествующей команды при $C=Z$. В данном случае используется функциональная окраска, при которой цвет задается высотой поверхности.

Возможны также формы команды `mesh(x, y, Z)`, `mesh(x, y, Z, C)`, `mesh(Z)` и `mesh(Z, C)`. Функция `mesh` возвращает дескриптор для объекта класса `surface`. Ниже приводится пример программы с применением команды `mesh`:

```
% Программа построения графика поверхности с окраской
[X, Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
mesh(X, Y, Z)
```

На рис. 6.21 показан график поверхности, созданной командой `mesh(X, Y, Z)`. Нетрудно заметить, что функциональная окраска линий поверхности заметно усиливает наглядность ее представления.

MATLAB имеет несколько графических функций, возвращающих *матричный образ поверхностей*. Например, функция `peaks(N)` возвращает матричный образ поверхности с рядом пиков и впадин. Такие функции удобно использовать для проверки работы графических команд трехмерной графики. Для упомянутой функции `peaks` можно привести такой пример:

```
>> z=peaks(25); mesh(z);
```

График поверхности, описываемой функцией `peaks`, представлен на рис. 6.22.

Рекомендуется ознакомиться с командами и функциями, используемыми совместно с описанными командами: `axis`, `caxis`, `colormap`, `hold`, `shading` и `view`.

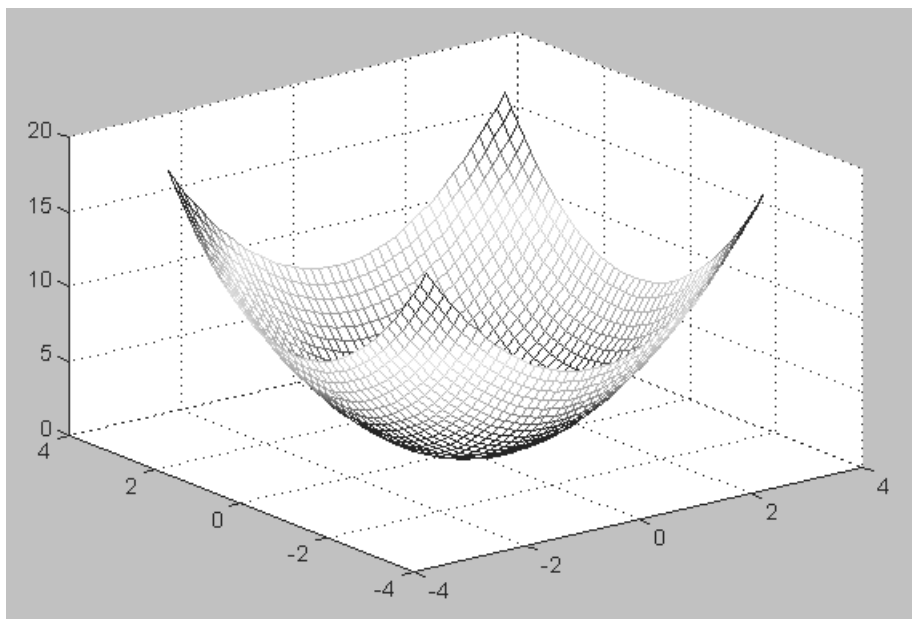


Рис. 6.21. График поверхности, созданный командой `mesh(X,Y,Z)`

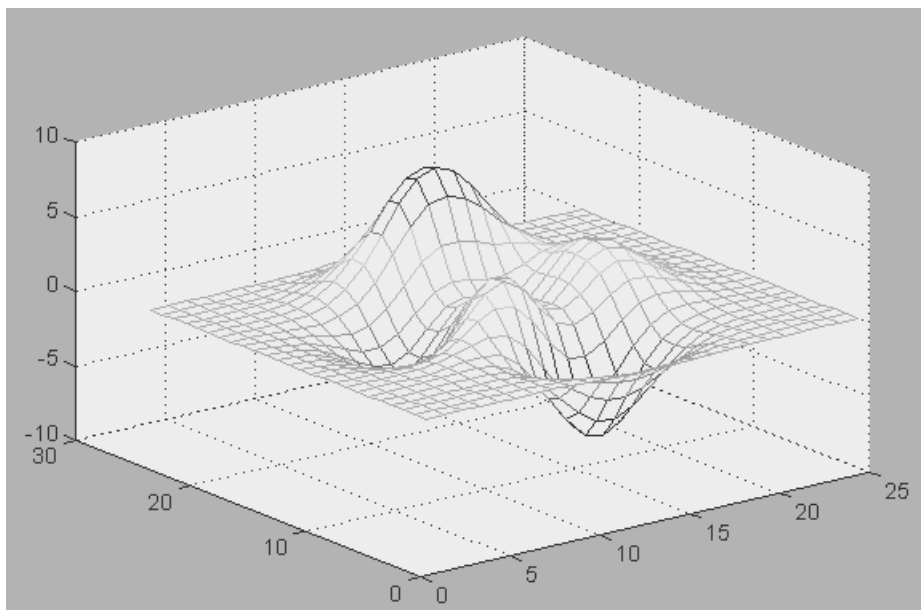


Рис. 6.22. График поверхности, описываемой функцией `peaks`

6.4.6. Сетчатые 3D-графики с проекциями

Иногда график поверхности полезно объединить с контурным графиком ее проекции на плоскость, расположенным под поверхностью. Для этого используется команда `meshc`:

- `meshc (...)` аналогична `mesh (...)`, но помимо графика поверхности дает изображение ее проекции в виде линий равного уровня (графика типа `contour`).

Ниже дан пример применения этой команды:

```
% Программа построения графика поверхности  
% и ее проекции на плоскость  
[X,Y]=meshgrid([-3:0.15:3]);  
Z=X.^2+Y.^2;  
meshc(X,Y,Z)
```

Построенный график показан на рис. 6.23.

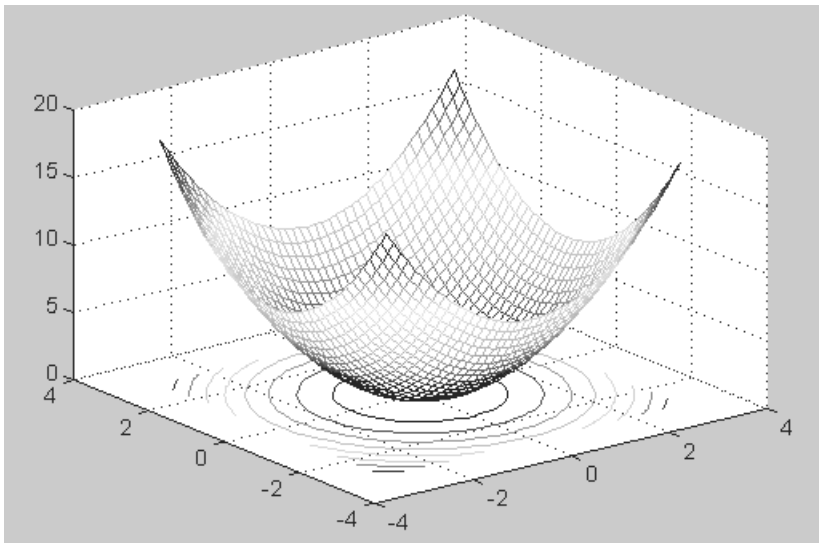


Рис. 6.23. График поверхности и ее проекции на расположенную ниже плоскость

Нетрудно заметить, что график такого типа дает наилучшее представление об особенностях поверхности.

6.4.7. Построение поверхности столбцами

Еще один тип представления поверхности, когда она строится из многочисленных столбцов, дают команды класса `meshz`:

- `meshz (...)` аналогична `mesh (...)`, но строит поверхность как бы в виде столбиков.

Следующая программа иллюстрирует применение команды `meshz`:

```
% Программа построения поверхности столбцами
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2; meshz(X,Y,Z)
```

Столбцовый график поверхности показан на рис. 6.24.

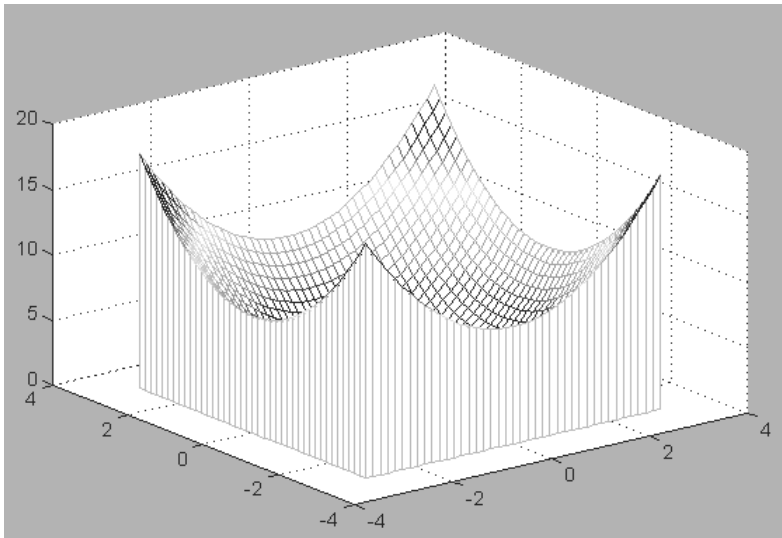


Рис. 6.24. Построение поверхности столбцами

Графики такого типа используются довольно редко. Возможно, он полезен архитекторам или скульпторам, поскольку дает неплохое объемное представление о поверхностях.

6.5. Улучшенные средства визуализации 3D-графики

6.5.1. Построение поверхности с окраской

Особенно наглядное представление о поверхностях дают сетчатые графики, использующие *функциональную закраску* ячеек. Например, цвет окраски поверхности $z(x,y)$ может быть поставлен в соответствии с высотой z поверхности с выбором для малых высот темных тонов, а для больших – светлых.

Для построения таких поверхностей используются команды класса `surf (...)`:

- `surf (X, Y, Z, C)` строит цветную параметрическую поверхность по данным матриц X , Y и Z с цветом, задаваемым массивом C ;
- `surf (X, Y, Z)` аналогична предшествующей команде, где $C=Z$, так что цвет задается высотой той или иной ячейки поверхности;
- `surf (x, y, Z)` и `surf (x, y, Z, C)` с двумя векторными аргументами x и y – векторы x и y заменяют два первых матричных аргумента и должны иметь длины $\text{length}(x)=n$ и $\text{length}(y)=m$, где $[m, n] = \text{size}(Z)$. В этом случае вершины областей поверхности представлены тройками координат $(x(j), y(i), Z(i, j))$. Заметим, что x соответствует столбцам Z , а y соответствует строкам;
- `surf (Z)` и `surf (Z, C)` используют $x = 1:n$ и $y = 1:m$. В этом случае высота Z – однозначно определенная функция, заданная геометрически прямоугольной сеткой;
- `h=surf (...)` строит поверхность и возвращает дескриптор объекта класса `surface`.

Команды `axis`, `caxis`, `colormap`, `hold`, `shading` и `view` задают координатные оси и свойства поверхности, которые могут использоваться для большей эффективности показа поверхности или фигуры.

Ниже приведен простой пример построения поверхности – параболоида:

```
% Программа построения графика параболоида с окраской  
[X,Y]=meshgrid([-3:0.15:3]);  
Z=X.^2+Y.^2;  
surf(X,Y,Z)
```

Соответствующий этому примеру график показан на рис. 6.25.

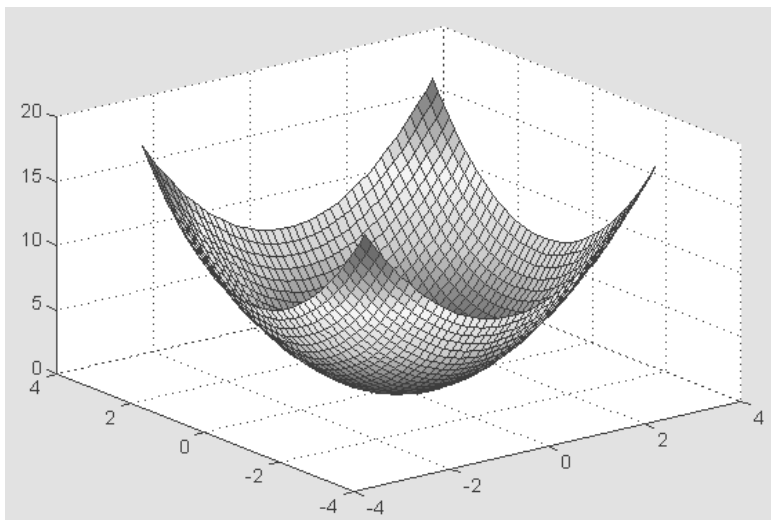


Рис. 6.25. График параболоида с функциональной окраской ячеек

Можно заметить, что благодаря функциональной окраске график поверхности гораздо более выразителен, чем при построениях без такой окраски, представленных ранее (причем даже в том случае, когда цветной график печатается в черно-белом виде).

В следующей программе используется функциональная окраска оттенками серого цвета с выводом шкалы цветовых оттенков:

```
% Программа построения параболоид  
% со шкалой цветовых оттенков  
[X,Y]=meshgrid([-3:0.1:3]);  
Z=sin(X)./(X.^2+Y.^2+0.3);  
surf(X,Y,Z); colormap(gray)  
shading interp; colorbar
```

В этом примере команда `colormap(gray)` задает окраску тонами серого цвета, а команда `shading interp` обеспечивает устранение изображения сетки и задает интерполяцию для оттенков цвета объемной поверхности. На рис. 6.26 показан вид графика, построенного при исполнении этой программы.

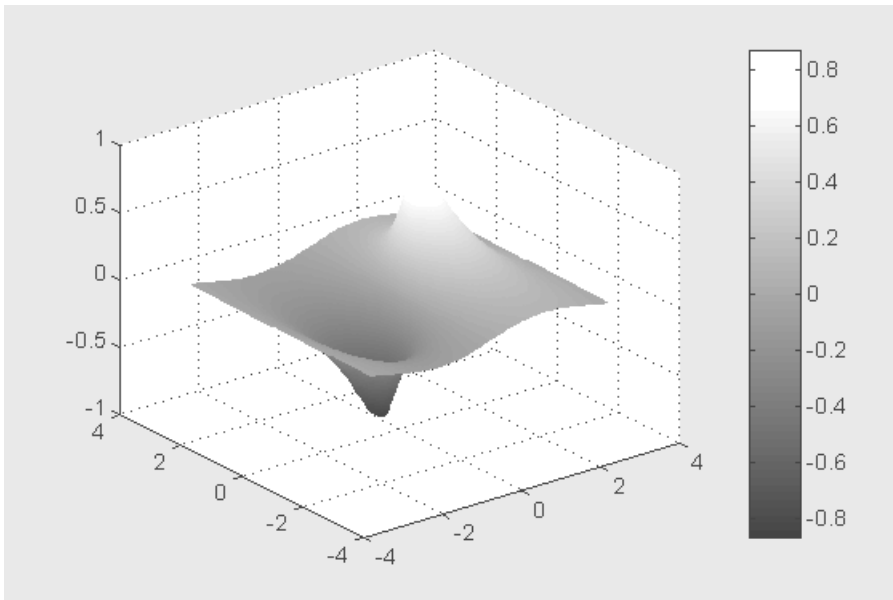


Рис. 6.26. График поверхности с функциональной окраской серым цветом

Обычно применение интерполяции для окраски придает поверхностям и фигурам более реалистичный вид, но фигуры каркасного вида дают более точные количественные данные о каждой точке.

6.5.2. Построение поверхности и ее проекции

Для повышения наглядности представления поверхностей можно использовать дополнительный график линий равного уровня, получаемый путем проецирования поверхности на опорную плоскость графика (под поверхностью). Для этого используется команда `surfz`:

- `surfz(...)` аналогична команде `surf`, но обеспечивает дополнительное построение контурного графика проекции фигуры на опорную плоскость.

Пример применения команды `surfz` приводится ниже:

```
>> [X,Y]=meshgrid([-3:0.1:3]);  
>> Z=sin(X)./(X.^2+Y.^2+0.3); surfz(X,Y,Z)
```

На рис. 6.27 показаны графики, построенные в данном примере.

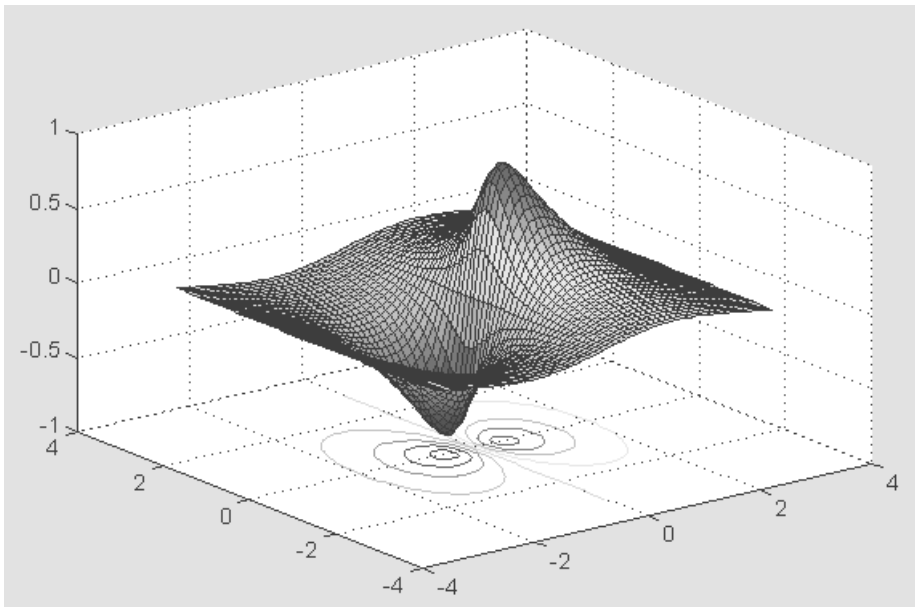


Рис. 6.27. График поверхности и ее проекции на опорную плоскость

Рассмотрим еще один пример применения команды `surfz`, на этот раз для построения поверхности, описываемой функцией `peaks` с применением интерполяции цветов и построением цветовой шкалы:

```
% Программа построения поверхности peaks  
% с построением шкалы цветовых оттенков  
[X,Y]=meshgrid([-3:0.1:3]);  
Z=peaks(X,Y);
```

```
surf(X, Y, Z)
shading interp;
colorbar
```

Рисунок 6.28 показывает график, построенный при пуске данной программы.

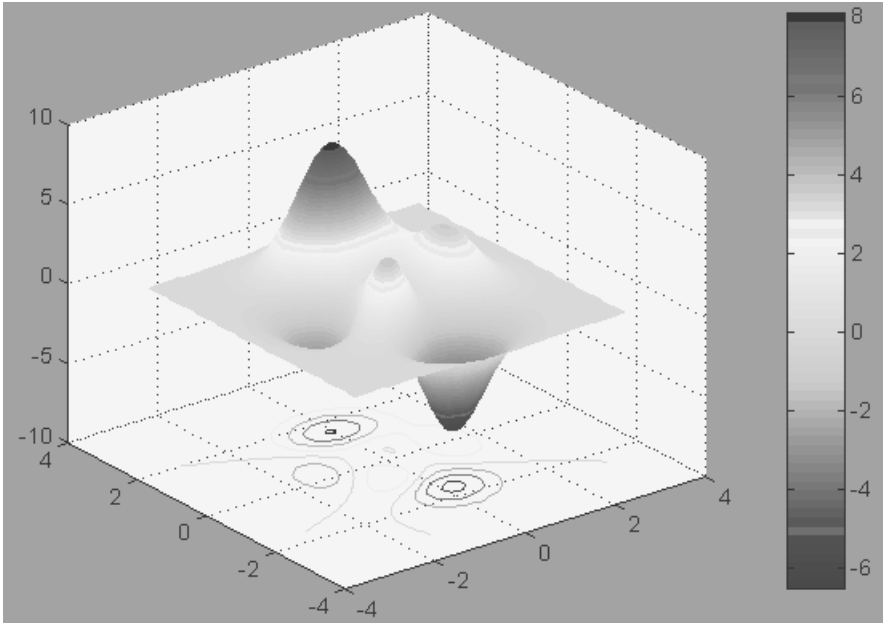


Рис. 6.28. График функции *peaks* с проекцией и шкалой цветов

И здесь нетрудно заметить, что графики сложных поверхностей с интерполяцией цветовых оттенков выглядят более реалистичными, чем графики сетчатого вида и графики без интерполяции цветов.

6.5.3. Построение освещенной поверхности

Пожалуй, наиболее реалистичный вид имеют графики поверхностей, в которых имитируется освещение от точечного источника света, расположенного в заданном месте координатной системы. Графики имитируют оптические эффекты рассеивания, отражения и зеркального отражения света. Для получения таких графиков используется команда `surf1`:

- `surf1(...)` аналогична команде `surf(...)`, но строит график поверхности с подсветкой от источника света;
- `surf1(Z, S)` или `surf1(X, Y, Z, S)` строит графики поверхности с подсветкой от источника света, положение которого в системе декартовых координат задается вектором $S = [S_x, S_y, S_z]$, а в системе сферических координат – вектором $S = [AZ, EL]$;

- `surf` (... , 'light') позволяет при построении задать цвет подсветки с помощью объекта `Light`;
- `surf` (... , 'cdata') при построении имитирует эффект отражения;
- `surf` (X, Y, Z, S, K) задает построение поверхности с параметрами, заданными вектором $K=[k_a, k_d, k_s, \text{spread}]$, где k_a – коэффициент фоновой подсветки, k_d – коэффициент диффузного отражения, k_s – коэффициент зеркального отражения и `spread` – коэффициент глянцевого отражения;
- `H=surf` (...) строит поверхность и возвращает дескрипторы поверхности и источников света.

По умолчанию вектор `S` задает углы азимута и возвышения в 45° . Используя команды `cla`, `hold on`, `view (AZ, EL)`, `surf` (...) и `hold off`, можно получить дополнительные возможности управления освещением. Надо полагаться на упорядочение точек в X, Y и Z матрицах, чтобы определить внутреннюю и внешнюю стороны параметрических поверхностей. Попробуйте транспонировать матрицы и использовать `surf` (X', Y', Z'), если вам не понравился результат работы этой команды. Для вычисления векторов нормалей поверхности `surf` требует в качестве аргументов матрицы с размером по крайней мере 3×3 .

Ниже представлен пример применения команды `surf`:

```
% Программа построения поверхности с имитацией
% ее освещенности от точечного источника
[X, Y]=meshgrid([-3:0.1:3]);
Z=sin(X) ./ (X.^2+Y.^2+0.3);
surf(X, Y, Z);
colormap(gray);
shading interp;
colorbar
```

Построенная в этом примере поверхность представлена на рис. 6.29.

Сравните этот рисунок с рис. 6.26, на котором та же поверхность построена без имитации ее освещения.

Примечание

Нетрудно заметить определенную логику в названиях графических команд. Имя команды состоит из основного слова и суффикса расширения. Например, все команды построения поверхностей имеют основное слово `surf` (сокращение от `surface` – поверхность) и суффиксы: `c` – для контурных линий поверхности, `l` – для освещенности и т. д. Это правило облегчает запоминание многочисленных команд графики.

6.5.4. Средства управления подсветкой и обзором фигур

Рекомендуется с помощью команды `help` ознакомиться с командами, задающими управление подсветкой и связанными с ней оптическими эффектами:

- `diffuse` – задание эффекта диффузионного рассеяния;

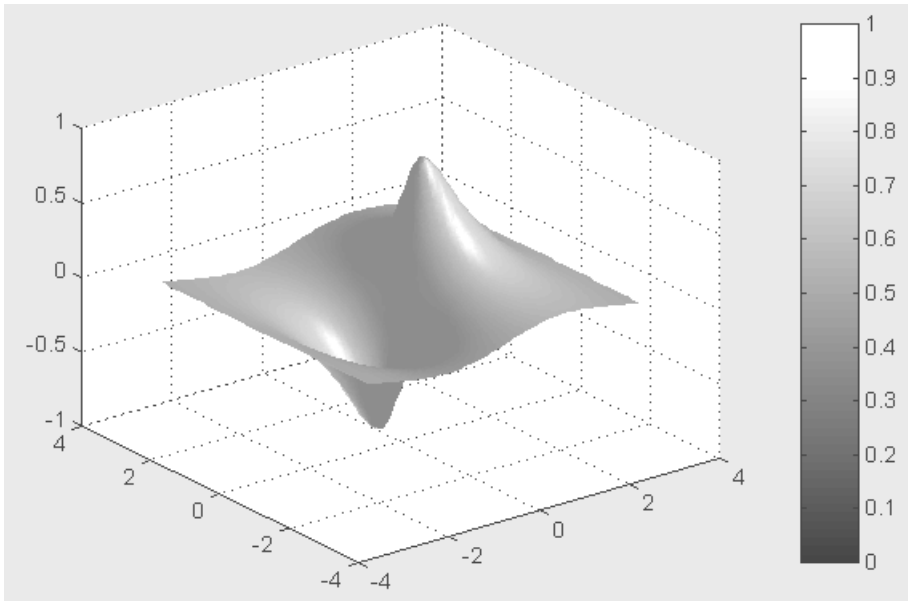


Рис. 6.29. График поверхности с имитацией ее освещения точечным источником

- `lighting` – управление подсветкой;
- `material` – имитация свойств рассеивания света различными материалами;
- `specular` – задание эффекта зеркального отражения.

Следующие три команды позволяют управлять углами просмотра, под которыми рассматривается видимая в графическом окне фигура:

- `view` – задание положения точки просмотра;
- `viewmtx` – задание и вычисление матрицы вращения;
- `rotate3d` – задание поворота трехмерной фигуры.

В ряде случаев применением этих команд можно добиться большей выразительности трехмерных объектов. Скорость построения таких графиков сильно зависит от аппаратной поддержки графики в конкретном ПК. Так, использование современных видеоадаптеров с графическим процессором и поддержкой средств OpenGL позволяет повысить скорость построения трехмерных графиков в несколько раз и добиться большей их выразительности.

6.5.5. Построение графиков функций трех переменных

Графики сечений функций трех переменных строит команда `slice` (в переводе – «ломтик»). Она используется в следующих формах.

- `slice(X, Y, Z, V, Sx, Sy, Sz)` строит плоские сечения объемной фигуры V в направлении осей x, y, z с позициями, задаваемыми векторами Sx, Sy, Sz .

Массивы X , Y , Z задают координаты для V и должны быть монотонными и трехмерными (как возвращаемые функцией `meshgrid`) с размером $M \times N \times P$. Цвет точек сечений определяется трехмерной интерполяцией в объемной фигуре V .

- `slice(X, Y, Z, V, XI, YI, ZI)` строит сечения объемной фигуры V по поверхности, определенной массивами XI , YI , ZI .
- `slice(V, Sx, Sy, Sz)` или `slice(V, XI, YI, ZI)` – подразумевается $X=1:N, Y=1:M, Z=1:P$.
- `slice(..., 'method')` – при построении задается метод интерполяции, который может быть одним из следующих: 'linear', 'cubic' или 'nearest'. По умолчанию используется линейная интерполяция – 'linear'.
- `h=slice(...)` строит сечение и возвращает дескриптор объекта класса `surface`.

```
% Программа построения графика функции трех переменных
[x, y, z] = meshgrid(-2:.2:2, -2:.25:2, -2:.16:2);
v = sin(x) .* exp(-x.^2 - y.^2 - z.^2);
slice(x, y, z, v, [-1.2 .8 2], 2, [-2 -2])
```

График, который строит эта программа, показан на рис. 6.30.

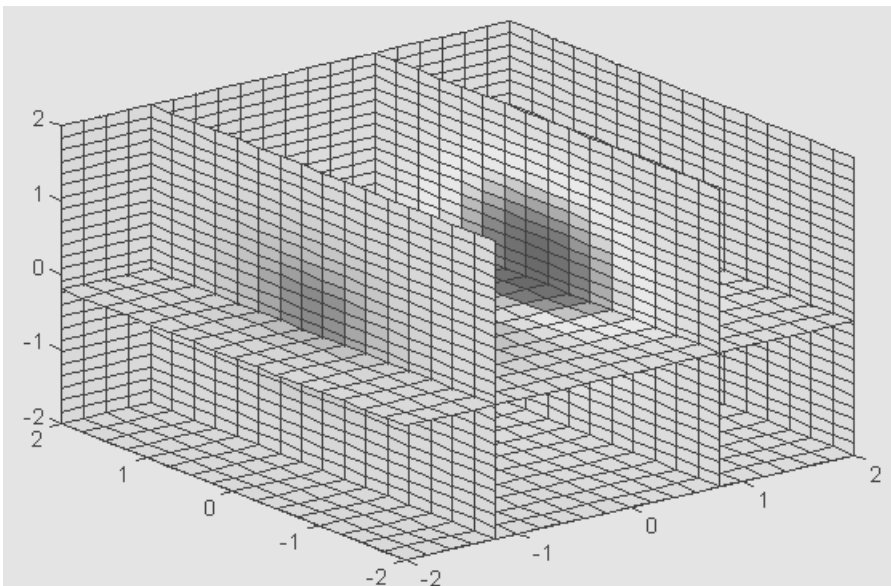


Рис. 6.30. График, показывающий сечения трехмерной поверхности

6.5.6. График трехмерной слоеной поверхности

Иногда бывают полезны графики трехмерных слоеных поверхностей, как бы состоящие из тонких пластинок – слоев. Такие поверхности строит функция `waterfall` (водопад):

- `waterfall (...)` строит поверхность как команда `mesh (...)`, но без показа ребер сетки. При ориентации графика относительно столбцов следует использовать запись `waterfall (Z')` или `waterfall (X', Y', Z')`. Пример:

```
% Программа построения графика слоеной поверхности
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
waterfall(X,Y,Z);
colormap(gray);
shading interp
```

Соответствующий график показан на рис. 6.31.

6.5.7. Трехмерные контурные графики

Трехмерный контурный график представляет собой расположенные в пространстве линии равного уровня, полученные при расслоении трехмерной фигуры рядом секущих плоскостей, расположенных параллельно опорной плоскости фигу-

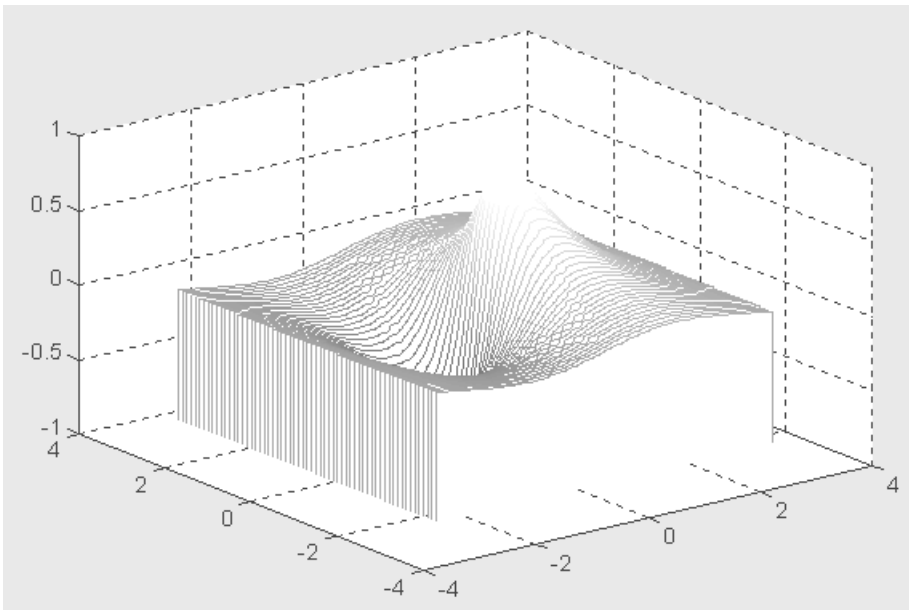


Рис. 6.31. Трехмерная слоеная поверхность

ры. При этом, в отличие от двумерного контурного графика, линии равного уровня отображаются в аксонометрии. Для получения трехмерных контурных графиков используется команда `contour3`:

- `contour3 (...)` имеет синтаксис, аналогичный команде `contour (...)`, но строит линии равного уровня в аксонометрии с использованием функциональной окраски (окраска меняется вдоль оси Z).

Полезные частные формы записи этой команды:

- `contour3 (Z)` строит контурные линии для поверхности, заданной массивом Z , без учета диапазона изменения x и y ;
- `contour3 (Z, n)` строит то же, что предыдущая команда, но с использованием n секущих плоскостей (по умолчанию $n=10$);
- `contour3 (X, Y, Z)` строит контурные линии для поверхности, заданной массивом Z , с учетом изменения x и y . Двумерные массивы X и Y создаются с помощью функции `meshgrid`;
- `contour3 (X, Y, Z, n)` строит то же, что предыдущая команда, но с использованием n секущих плоскостей.

Пример применения команды `contour3`:

```
>> contour3(peaks, 20); colormap(gray)
```

Соответствующий данному примеру график представлен на рис. 6.32. В данном случае задано построение 20 линий уровня.

С командой `contour3` связаны следующие одноименные функции (не выполняющие графических построений).

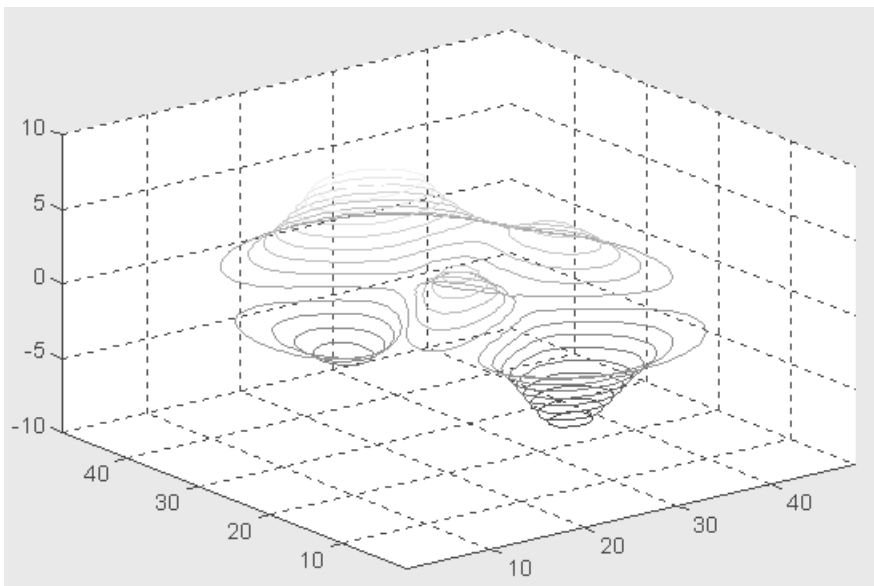


Рис. 6.32. Трехмерный контурный график для функции `peaks`

- `C=contour3 (...)` возвращает матрицу описания контурных линий `C` для использования командой `clabel`.
- `[C, H]=contour3 (...)` возвращает массив `C` и вектор-столбец `H` дескрипторов объектов `path` для каждой линии уровня. Свойство `UserData` каждого объекта содержит значение высоты для соответствующего контура.

6.6. Текстовое оформление графиков

6.6.1. Установка титульной надписи

После того как график уже построен, MATLAB позволяет выполнить его форматирование или оформление в нужном виде. Соответствующие этому средства описаны ниже. Так, для установки над графиком титульной надписи используется следующая команда:

- `title('string')` установка на двумерных и трехмерных графиках титульной надписи, заданной строковой константой `'string'`.

Пример применения этой команды будет дан в следующем разделе.

6.6.2. Установка осевых надписей

Для установки надписей возле осей x , y и z используются следующие команды:

```
xlabel('String')
ylabel('String')
zlabel('String')
```

Соответствующая надпись задается символьной константой или переменной `'String'`. Пример установки титульной надписи и надписей по осям графиков приводится ниже:

```
% Программа построения графика поверхности
% с текстовым оформлением
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surf(X,Y,Z); colorbar
colormap(gray); shading interp
xlabel('Axis X'); ylabel('Axis Y')
zlabel('Axis Z'); title('Surface graphic')
```

Построенный в этом примере график трехмерной поверхности показан на рис. 6.33.

Сравните его с графиком, показанным на рис. 6.29. Надписи делают рисунок более наглядным.

6.6.3. Ввод текста в любое место графика

Часто возникает необходимость добавления текста в определенное место графика, например для обозначения той или иной кривой графика. Для этого используется команда `text`:

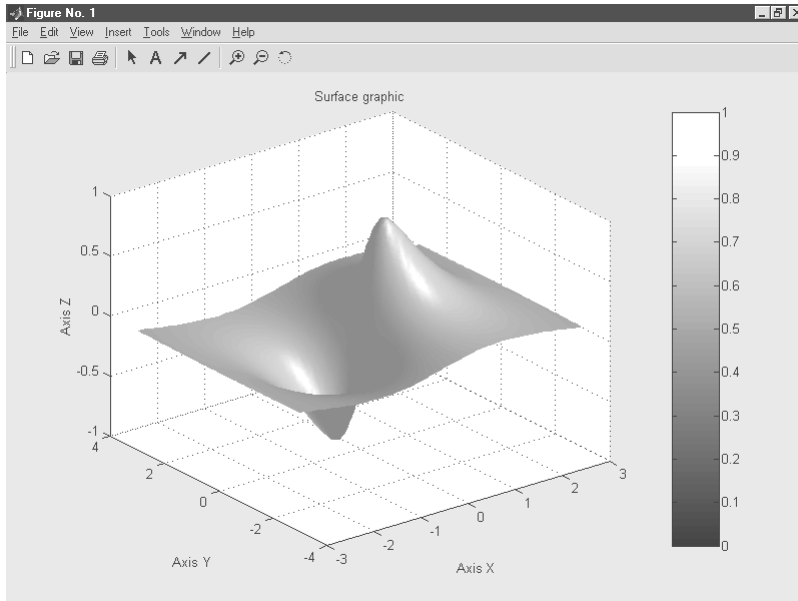


Рис. 6.33. График трехмерной поверхности с титульной надписью и надписями по координатным осям

- `text(X,Y,'string')` добавляет в двумерный график текст, заданный строковой константой `'string'`, так что начало текста расположено в точке с координатами (X,Y) . Если X и Y заданы как одномерные массивы, то надпись помещается во все позиции $[x(i), y(i)]$;
- `text(X,Y,Z,'string')` добавляет в трехмерный график текст, заданный строковой константой `'string'`, так что начало текста расположено в позиции, заданной координатами X, Y и Z .

В приведенном ниже примере надпись «График функции $\sin(x^3)$ » размещается под кривой графика в позиции $(-4, 0.7)$:

```
>> x=-10:0.1:10; plot(x, sin(x).^3)
>> text(-4,0.7,'Graphic sin(x)^3')
```

График функции с надписью у кривой показан на рис. 6.34.

Математически правильной записью была бы \sin^3x . Попробуйте ввести самостоятельно

```
>> x=-10:0.1:10;
>> plot(x, sin(x).^3)
>> text(-4,0.7,'Graphic (sin(x))^3')
```

Функция `h=text(...)` возвращает вектор-столбец `h` дескрипторов объектов класса `text`, дочерних для объектов класса `axes`. Следующий пример вычисляет дескриптор `h`:

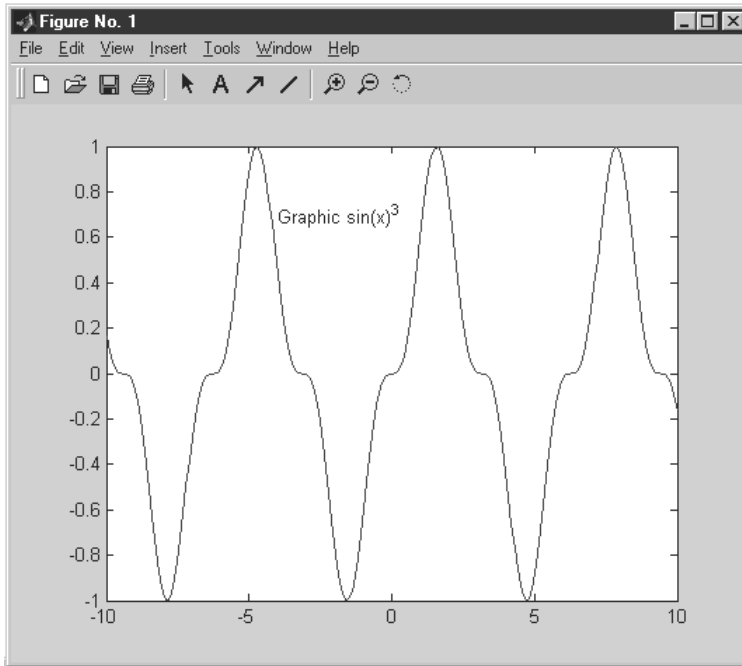


Рис. 6.34. Пример ввода надписи в поле графика функции

```
>> h=text(.25,.5,'\ite^{i\omega\tau} = cos(\omega\tau) + ... i
sin(\omega\tau)')
h = 3.0022
```

и выводит в пустом графике математическую формулу в формате TeX вида:

$$e^{j\omega t} = \cos(\omega t) + j \sin(\omega t).$$

Пары координат X,Y (или тройки X,Y,Z для трехмерных графиков) могут сопровождаться парами «имя параметра/значение параметра» для задания дополнительных свойств текста. Пары координат X,Y (или тройки X,Y,Z для трехмерных графиков) могут быть полностью опущены, при этом все свойства, в том числе и позиция текста, задаются с помощью пар «имя параметра/значение параметра», определенных по умолчанию.

Используйте функцию `get (H)`, где H – дескриптор графического объекта (в нашем случае графического объекта класса `text`), чтобы просмотреть список свойств объекта и их текущие значения. Используйте `set (H)`, чтобы просмотреть список свойств графических объектов и их допустимых значений.

6.6.4. Позиционирование текста с помощью мыши

Очень удобный способ ввода текста предоставляет команда `gtext`:

- `gtext('string')` задает выводимый на график текст в виде строковой константы 'string' и выводит на график перемещаемый мышью маркер в виде крестика. Установив маркер в нужное место, достаточно щелкнуть любой кнопкой мыши для вывода текста;
- `gtext(C)` позволяет аналогичным образом разместить многострочную надпись из массива строковых переменных `C`.

Пример применения команды `gtext`:

```
>> x=-15:0.1:15; plot(x, sin(x).^3)
>> gtext('Function sin(x)^3')
```

При исполнении этого примера вначале можно увидеть построение графика функции с большим крестом, перемещаемым мышью (рис. 6.35).

Установив перекрестие в нужное место графика, достаточно нажать любую клавишу или любую кнопку мыши, и на этом месте появится надпись (рис. 6.36).

Высокая точность позиционирования надписи и быстрота процесса делают данный способ нанесения надписей на графики одним из наиболее удобных.

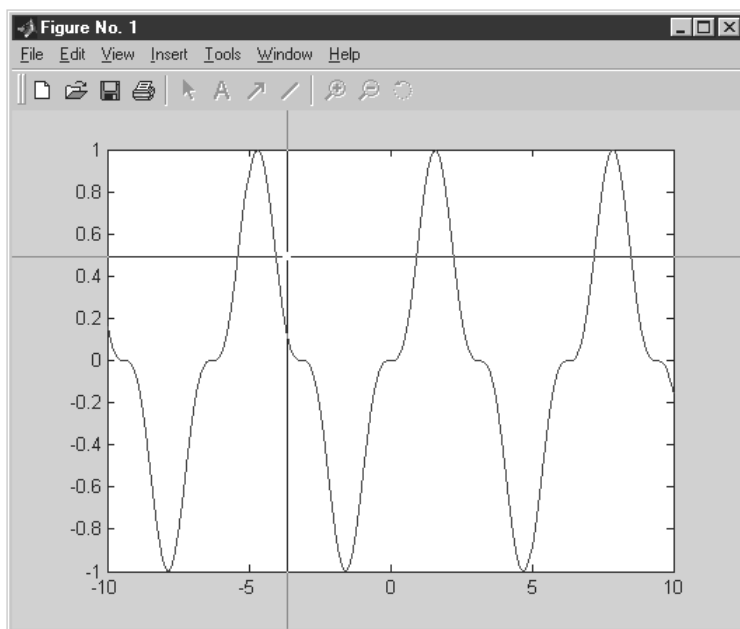


Рис. 6.35. График функции с крестообразным маркером, перемещаемым мышью

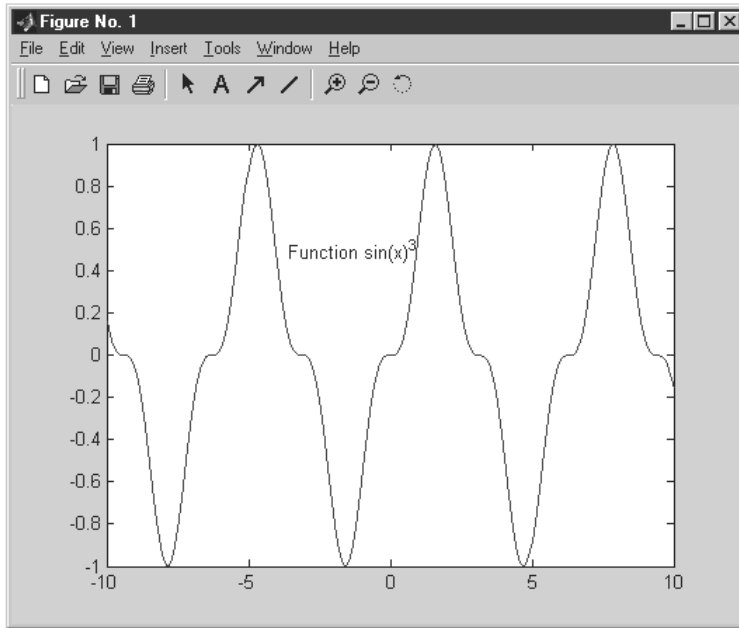


Рис. 6.36. График функции с надписью, установленной с помощью мыши

6.7. Форматирование графиков

6.7.1. Вывод пояснений и легенды

Пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, называется *легендой*. Для создания легенды используются различные варианты команды `legend`:

- `legend(string1, string2, string3, ...)` добавляет к текущему графику легенду в виде строк, указанных в списке параметров;
- `legend(h, string1, string2, string3, ...)` помещает легенду на график, содержащий объекты с дескрипторами `h`, используя заданные строки как метки для соответствующих дескрипторов;
- `legend(AX, ...)` помещает легенду в осях (объект класса `axes`) с дескриптором `AX`;
- `legend(M)` размещает легенду, используя данные из строковой матрицы `M`;
- `legend OFF` устраняет ранее выведенную легенду;
- `legend` перерисовывает текущую легенду, если таковая имеется;

- `legend(legendhandle)` перерисовывает легенду, указанную дескриптором `legendhandle`;
- `legend(..., Pos)` помещает легенду в точно определенное место, специфицированное параметром `Pos`:
 - `Pos=0` – лучшее место, выбираемое автоматически;
 - `Pos=1` – верхний правый угол;
 - `Pos=2` – верхний левый угол;
 - `Pos=3` – нижний левый угол;
 - `Pos=4` – нижний правый угол;
 - `Pos=-1` – справа от графика.

Чтобы перенести легенду, установите на нее курсор, нажмите левую кнопку мыши и перетащите легенду в необходимую позицию.

- `[leg, objh]=legend(...)` – эта функция возвращает дескриптор объекта для легенды (`leg`) и матрицу `objh`, содержащую дескрипторы объектов, из которых легенда состоит.

Команда `legend` может использоваться с двумерной и трехмерной графикой и со специальной графикой – столбцовыми и круговыми диаграммами и т. д. Двойным щелчком можно вывести легенду на редактирование.

Программа, приведенная ниже, строит график трех функций с легендой, размещенной в поле графика:

```
% Программа построения графика трех функций
% с выводом их обозначений – легендой
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x, y1, '-m', x, y2, '-.r', x, y3, '-ok')
legend('Function 1', 'Function 2', 'Function 3');
```

Полученный график представлен на рис. 6.37.

Незначительная модификация команды `legend` (применение дополнительного параметра `-1`) позволяет построить график трех функций с легендой вне поля графика. Это иллюстрирует следующая программа:

```
% Программа построения графика трех функций
% с выводом легенды вне поля графика
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x, y1, '-m', x, y2, '-.r', x, y3, '-ok')
legend('Function 1', 'Function 2', 'Function 3', -1);
```

Соответствующий график показан на рис. 6.38.

В данном случае недостатком можно считать сокращение полезной площади самого графика. Остальные варианты расположения легенды пользователю предлагается изучить самостоятельно. Следует отметить, что применение легенды придает графикам более осмысленный и профессиональный вид. При необходимости легенду можно переместить мышью в подходящее место графика.

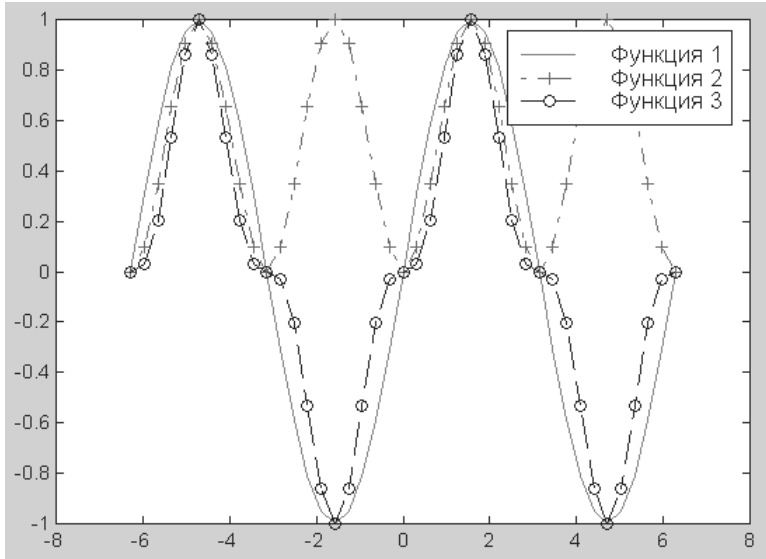


Рис. 6.37. График трех функций с легендой в поле графика

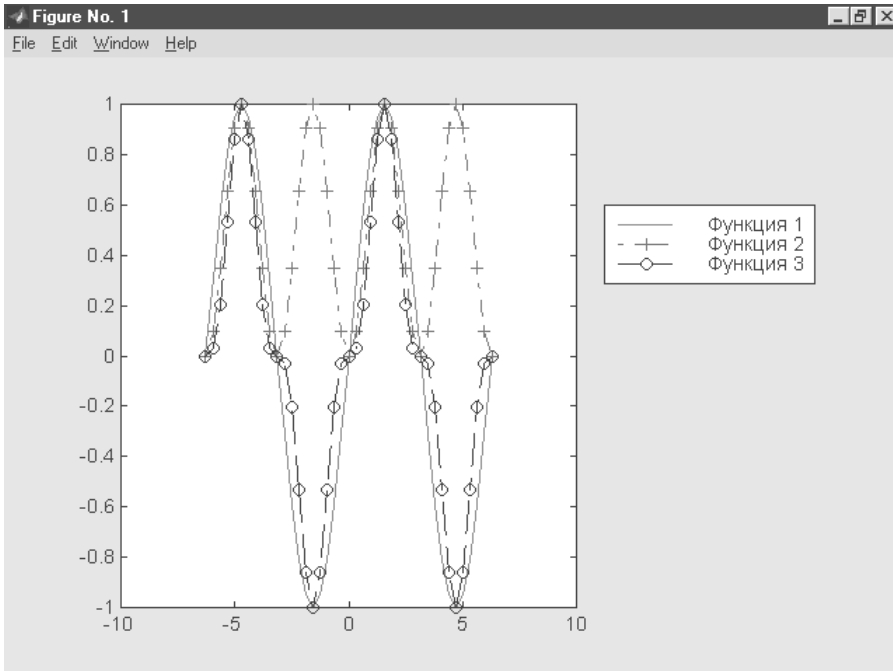


Рис. 6.38. График трех функций с легендой, расположенной вне поля графика

6.7.2. Маркировка линий уровня на контурных графиках

К сожалению, контурные графики плохо приспособлены для количественных оценок, если их линии не маркированы. В качестве маркеров используются крестики, рядом с которыми располагаются значения высот. Для маркировки контурных графиков используются команды группы `clabel`:

- `clabel(CS, H)` маркирует контурный график с данными в контурной матрице `CS` и дескрипторами объектов, заданными в массиве `H`. Метки вставляются в разрывы контурных линий и ориентируются в соответствии с направлением линий;
- `clabel(CS, H, V)` маркируются только те уровни, которые указаны в векторе `V`. По умолчанию маркируются все контуры. Позиции меток располагаются случайным образом;
- `clabel(CS, H, 'manual')` маркирует контурные графики с установкой положения маркеров с помощью мыши. Нажатие клавиши **Enter** или кнопки мыши завершает установку маркера. При отсутствии мыши для перехода от одной линии уровня к другой используется клавиша пробела, а для перемещения надписи используются клавиши перемещения курсора;
- `clabel(CS)`, `clabel(CS, V)` и `clabel(CS, 'manual')` – дополнительные возможности маркировки контурных графиков. При отсутствии аргумента `h` метки не ориентируются вдоль линий контуров; точную позицию метки отмечает значок «плюс» (далее на рис. 6.39 показан именно этот вариант).

Пример применения команды `clabel` приводится ниже:

```
% Программа построения контурного графика поверхности
% с маркированными линиями уровня
[X, Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3); C=contour(X, Y, Z, 10);
colormap(gray); clabel(C)
```

Рисунок 6.39 показывает построение контурного графика с маркированными линиями уровня, полученного при исполнении приведенной программы.

Функция `H=clabel(...)` маркирует график и возвращает дескрипторы создаваемых при маркировке объектов класса `TEXT` (и, возможно, `LINE`).

6.7.3. Управление свойствами осей графиков

Обычно графики выводятся в режиме автоматического масштабирования. Следующие команды класса `axis` меняют эту ситуацию:

- `axis([XMIN XMAX YMIN YMAX])` – установка диапазонов координат по осям x и y для текущего двумерного графика;

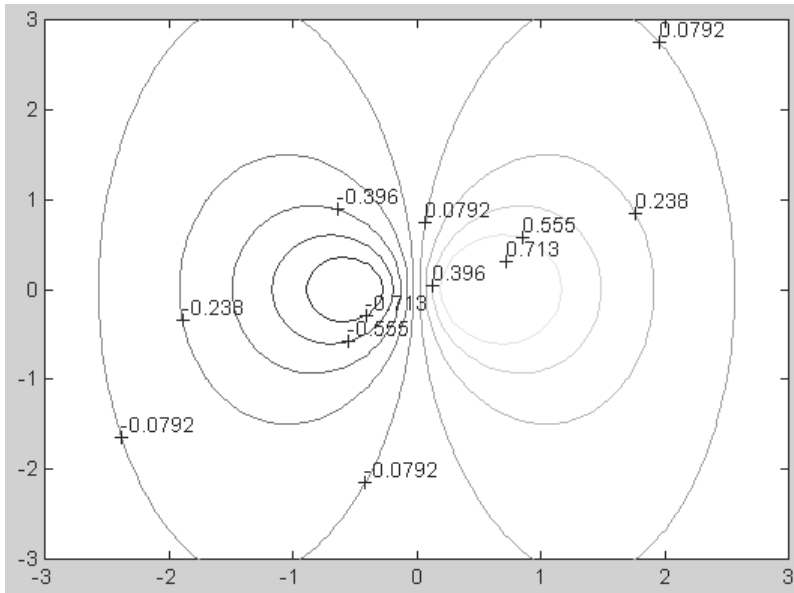


Рис. 6.39. Контурный график с маркированными линиями уровня

- `axis ([XMIN XMAX YMIN YMAX ZMIN ZMAX])` – установка диапазонов координат по осям x , y и z текущего трехмерного графика;
- `axis auto` – установка параметров осей по умолчанию;
- `axis manual` «замораживает» масштабирование в текущем состоянии, чтобы при использовании команды `hold on` следующие графики использовали те же параметры осей;
- `axis tight` устанавливает диапазоны координат по осям в соответствии с диапазонами изменения данных;
- `axis ij` задает «матричную» прямоугольную систему координат с началом координат в левом верхнем углу, ось i – вертикальная, размечаемая сверху вниз, ось j – горизонтальная и размечается слева направо;
- `axis xy` устанавливает декартову систему координат с горизонтальной осью x , размечаемой слева направо, и вертикальной осью y , размечаемой снизу вверх (начало координат размещается в нижнем левом углу);
- `axis equal` включает масштаб с одинаковым расстоянием между метками по осям x , y и z ;
- `axis image` устанавливает масштаб, при котором пиксели изображения становятся квадратами;
- `axis square` устанавливает текущие оси в виде квадрата (или куба в трехмерном случае) с одинаковым расстоянием между метками и одинаковой длиной осей;
- `axis normal` восстанавливает масштаб, отменяя установки `axis equal` и `axis square`;

- `axis vis3d` «замораживает» пропорции осей для возможности поворота трехмерных объектов;
- `axis off` убирает с осей их обозначения и маркеры;
- `axis on` восстанавливает ранее введенные обозначения осей и маркеры;
- `V=axis` возвращает вектор-строку, содержащую коэффициенты масштабирования для текущего графика. Если текущий график двумерный, то вектор имеет 4 компонента, если трехмерный – 6 компонентов.

Следующий пример иллюстрирует применение команды `axis` при построении двумерного графика функции одной переменной:

```
>> x=-5:0.1:5; plot(x, sin(x)); axis([-10 10 -1.5 1.5])
```

На рис. 6.40 показано изображение, которое строится в этом примере.

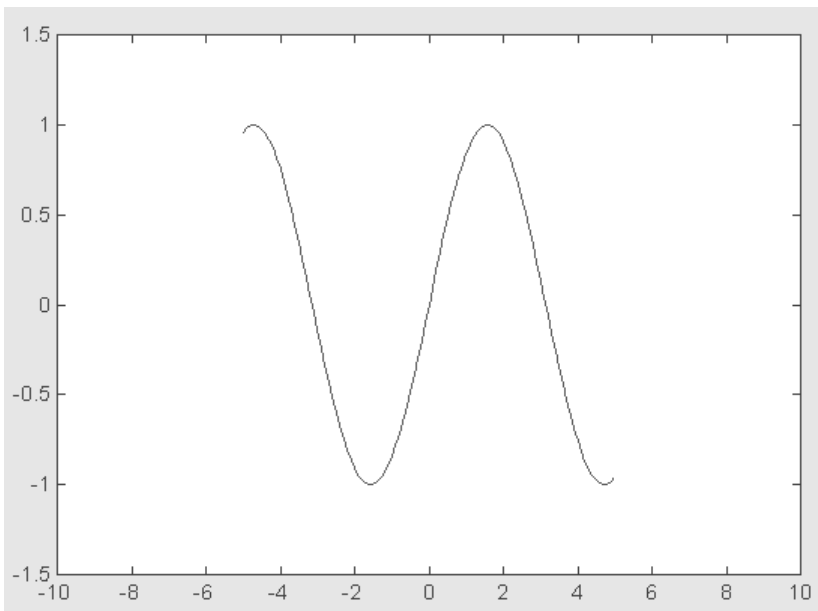


Рис. 6.40. Пример задания масштаба осей двумерного графика

Обратите внимание, что теперь масштабы осей заданы командой `axis`, а не диапазоном изменения значений x и y .

6.7.4. Включение и выключение сетки

В математической, физической и иной литературе при построении графиков в дополнение к разметке осей часто используют масштабную сетку. Команды `grid` позволяют задавать построение сетки или отменять это построение:

- `grid on` добавляет сетку к текущему графику;

- `grid off` отключает сетку;
- `grid` последовательно производит включение и отключение сетки.

Команды `grid` устанавливают свойства объектов `XGrid`, `Ygrid` и `Zgrid` для текущих осей. Ниже приведен пример из предшествующего раздела с добавлением в него команды `grid`:

```
>> x=-5:0.1:5; plot(x, sin(x));  
>> axis([-10 10 -1.5 1.5]); grid on
```

Построенный график показан на рис. 6.41.

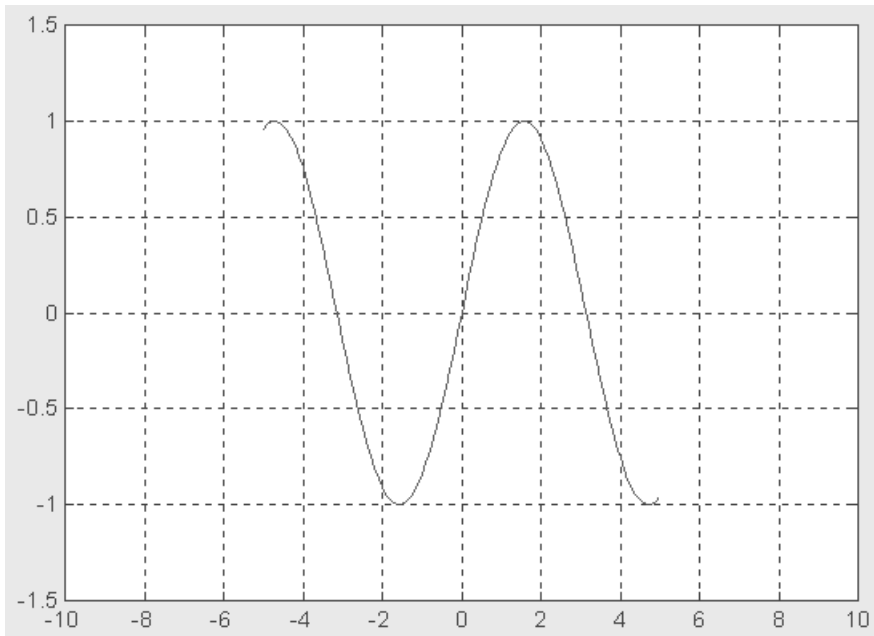


Рис. 6.41. График синусоиды с сеткой разметки

Сравните этот график с графиком на рис. 6.40, на котором сетка отсутствует. Нетрудно заметить, что наличие сетки облегчает количественную оценку координат точек графика.

6.7.5. Наложение графиков друг на друга

Во многих случаях желательно построение многих наложенных друг на друга графиков в одном и том же окне. Для этого служит команда продолжения графических построений `hold`. Она используется в следующих формах:

- `hold on` обеспечивает продолжение вывода графиков в текущее окно, что позволяет добавлять последующие графики к уже существующим;

- `hold off` отменяет режим продолжения графических построений;
- `hold` работает как переключатель, последовательно включая режим продолжения графических построений и отменяя его.

Команда `hold on` устанавливает значение `add` для свойства `NextPlot` объектов `figure` и `axes`, а `hold off` устанавливает для этого свойства значение `replace`. Рекомендуется также ознакомиться с командами `ishold`, `newplot`, `figure` и `axes`.

Приведенный ниже пример показывает, как с помощью команды `hold on` на график синусоиды накладываются еще три графика параметрически заданных функций:

```
>> x=-5:0.1:5; plot(x, sin(x)); hold on
>> plot(sin(x), cos(x)); plot(2*sin(x), cos(x))
>> plot(4*sin(x), cos(x)); hold off
```

Графики построенных функций показаны на рис. 6.42.

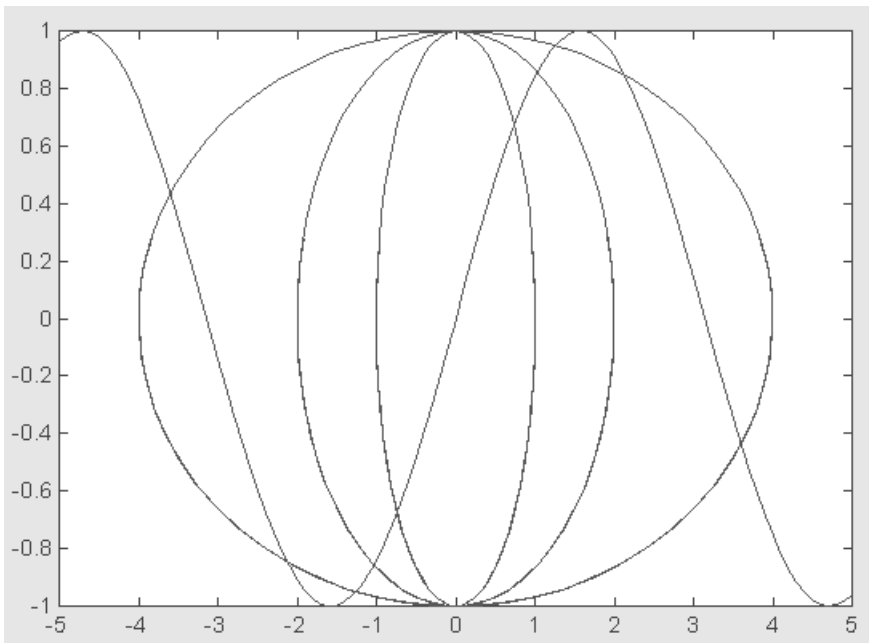


Рис. 6.42. Графики синусоиды и трех параметрических функций в одном окне

В конце приведенного фрагмента программы команда `hold off` отключает режим добавления графиков к ранее построенным графикам.

6.7.6. Разбиение графического окна

Бывает, что в одном окне надо расположить несколько координатных осей с различными графиками без наложения их друг на друга. Для этого используются команды `subplot`, применяемые перед построением графиков:

- `subplot` создает новые объекты класса `axes` (подокна);
- `subplot(m, n, p)` или `subplot(mnp)` разбивает графическое окно на $m \times n$ подокон, при этом m – число подокон по горизонтали, n – число подокон по вертикали, а p – номер подокна, в котором будет выводиться текущий график (подокна отсчитываются последовательно по строкам);
- `subplot(N)`, где N – дескриптор для объекта `axes`, дает альтернативный способ задания подокна для текущего графика;
- `subplot('position', [left bottom width height])` создает подокно с заданными нормализованными координатами (в пределах от 0.0 до 1.0);
- `subplot(111)` и `clf reset` удаляют все подокна и возвращают графическое окно в обычное состояние.

Следующая программа иллюстрирует применение команды `subplot`:

```
% Программа построения четырех графиков в разных подокнах
x=-5:0.1:5;
subplot(2,2,1),plot(x,sin(x))
subplot(2,2,2),plot(sin(5*x),cos(2*x+0.2))
subplot(2,2,3),contour(peaks)
subplot(2,2,4),surf(peaks)
```

В этом примере при пуске программы последовательно строятся четыре графика различного типа, размещаемых в разных подокнах (рис. 6.43).

Следует отметить, что для всех графиков возможна индивидуальная установка дополнительных объектов, например титульных надписей, надписей по осям и т. д.

6.7.7. Изменение масштаба графика

Для изменения масштаба двумерных графиков используются команды класса `zoom`:

- `zoom` переключает состояние режима интерактивного изменения масштаба для текущего графика;
- `zoom(FACTOR)` устанавливает масштаб в соответствии с коэффициентом `FACTOR`;
- `zoom on` включает режим интерактивного изменения масштаба для текущего графика;
- `zoom off` выключает режим интерактивного изменения масштаба для текущего графика;
- `zoom out` обеспечивает полный просмотр, то есть устанавливает стандартный масштаб графика;
- `zoom xon` или `zoom yon` включает режим изменения масштаба только по оси x или по оси y ;

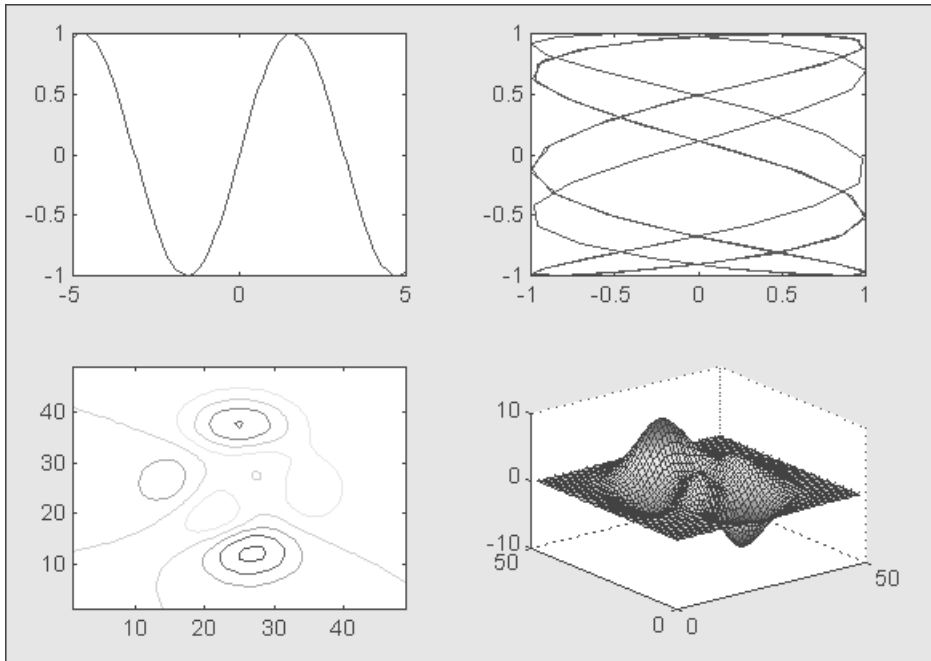


Рис. 6.43. Четыре графика различного типа, размещенных в подокнах одного окна

- `zoom reset` запоминает текущий масштаб в качестве масштаба по умолчанию для данного графика;
- `zoom(FIG,OPTION)` применяется к графику, заданному дескриптором `FIG`, при этом `OPTION` может быть любым из перечисленных выше аргументов.

Команда `zoom` позволяет управлять масштабированием графика с помощью мыши. Для этого надо подвести курсор мыши к интересующей вас области рисунка. Если команда `zoom` включена (`on`), то нажатие левой кнопки увеличивает масштаб вдвое, а правой – уменьшает вдвое. При нажатой левой кнопке мыши можно выделить пунктирным черным прямоугольником нужный участок графика – при отпускании кнопки он появится в увеличенном виде и в том масштабе, который соответствует выделяющему прямоугольнику.

Рассмотрим работу команды `zoom` на следующем примере:

```
>> x=-5:0.01:5; plot(x,sin(x.^5)./(x.^5+eps)); zoom on
```

Рисунок 6.44 показывает график функции данного примера в режиме выделения его участка с помощью мыши.

После прекращения манипуляций левой кнопкой мыши график примет вид, показанный на рис. 6.45. Теперь в полный размер графического окна будет возвращено изображение, попавшее в выделяющий прямоугольник.

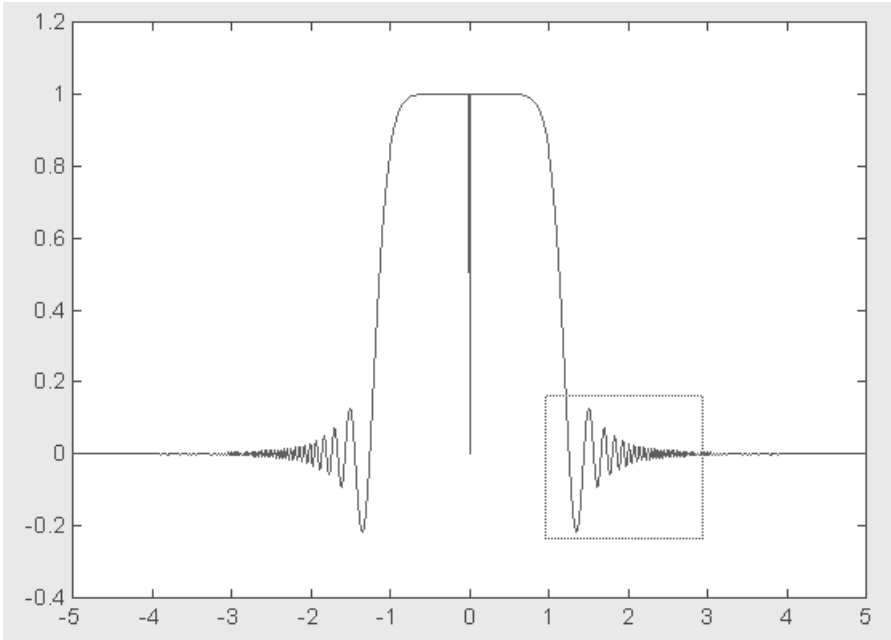


Рис. 6.44. Выделение части графика мышью при использовании команды *zoom*

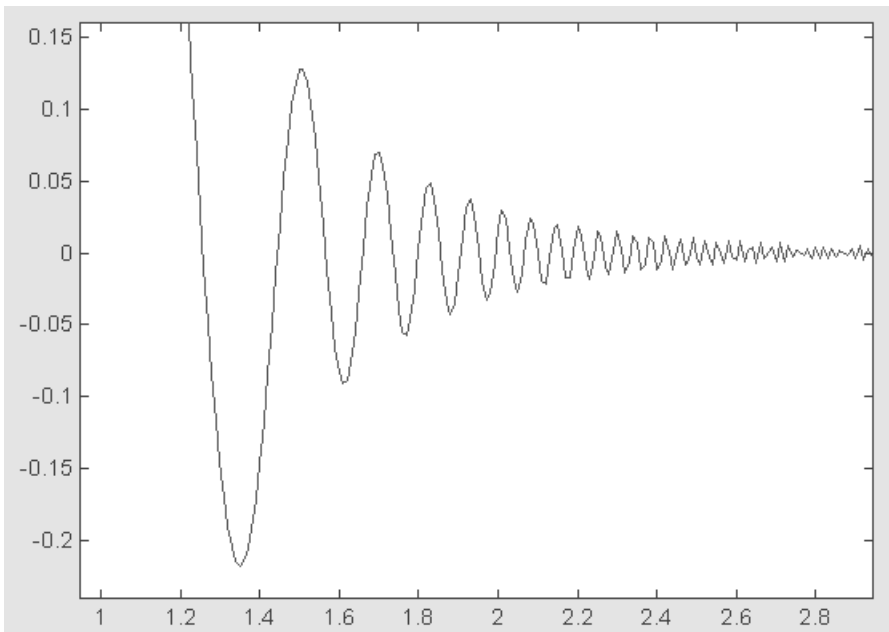


Рис. 6.45. График выделенного участка

Команда `zoom`, таким образом, выполняет функцию «лупы», позволяющей наблюдать в увеличенном виде отдельные фрагменты сложных графиков. Однако следует учитывать, что для наблюдения фрагментов графиков при высоком увеличении они должны быть заданы большим количеством точек. Иначе вид отдельных фрагментов и тем более особых точек (в нашем случае это точка при x вблизи нуля) будет существенно отличаться от истинного.

6.8. Цветовая окраска графиков

6.8.1. Установка палитры цветов

Поскольку графика MATLAB обеспечивает получение цветных изображений, в ней есть ряд команд для управления цветом и различными световыми эффектами. Среди них важное место занимает установка палитры цветов. Палитра цветов RGB задается матрицей `MAP` из трех столбцов, определяющих значения интенсивности красного (`red`), зеленого (`green`) и синего (`blue`) цветов. Их интенсивность задается в относительных единицах от 0.0 до 1.0. Например, `[0 0 0]` задает черный цвет, `[1 1 1]` – белый цвет, `[0 0 1]` – синий цвет. При изменении интенсивности цветов в указанных пределах возможно задание любого цвета. Таким образом, цвет соответствует общепринятому формату RGB.

Для установки палитры цветов служит команда `colormap`, записываемая в следующих формах:

- `colormap('default')` устанавливает палитру по умолчанию, при которой распределение цветов соответствует радуге;
 - `colormap(MAP)` устанавливает палитру RGB, заданную матрицей `MAP`;
 - `C=colormap` – функция возвращает матрицу текущей палитры цветов `C`.
- `m`-файл с именем **colormap** устанавливает свойства цветов для текущего графика.

Команда `help graph3d` наряду с прочим выводит полный список характерных палитр, используемых графической системой MATLAB:

- `hsv` – цвета радуги;
- `hot` – чередование черного, красного, желтого и белого цветов;
- `gray` – линейная палитра в оттенках серого цвета;
- `bone` – серые цвета с оттенком синего;
- `copper` – линейная палитра с оттенками меди;
- `pink` – розовые цвета с оттенками пастели;
- `white` – палитра белого цвета;
- `flag` – чередование красного, белого, синего и черного цветов;
- `lines` – палитра с чередованием цветов линий;
- `colorcube` – расширенная палитра RGB;
- `jet` – разновидность палитры HSV;
- `prism` – призматическая палитра цветов;
- `cool` – оттенки голубого и фиолетового цветов;
- `autumn` – оттенки красного и желтого цветов;

- `spring` – оттенки желтого и фиолетового цветов;
- `winter` – оттенки синего и зеленого цветов;
- `summer` – оттенки зеленого и желтого цветов.

Все эти палитры могут служить параметрами команды `colormap`, например `colormap(hsv)` фактически устанавливает то же, что и команда `colormap('default')`. Примеры применения команды `colormap` будут приведены в следующих разделах.

6.8.2. Установка соответствия между палитрой цветов и масштабом осей

При использовании функциональной окраски важное значение имеет установка соответствия между палитрой цветов и масштабом координатных осей. Так, выбор ограниченного диапазона интенсивностей цветов может привести к тому, что цветовая гамма будет блеклой и функциональная закраска не будет достигать своих целей. С помощью команды `caxis` можно обеспечить соответствие между палитрой цветов и масштабом осей:

- `caxis(V)` с помощью двухэлементного вектора `V` со списком элементов `[cmin cmax]` устанавливает диапазон используемой палитры цветов для объектов `surface` и `patch`, создаваемых такими командами, как `mesh`, `pcolor` и `surf`. Пиксели, цвета которых выходят за пределы `[cmin cmax]`, приводятся к граничным цветам диапазона;
- `caxis('manual')` устанавливает шкалу цветов по текущему интервалу параметра, задающего цвет;
- `caxis('auto')` устанавливает типовое масштабирование шкалы цветов, при котором диапазон используемых цветов соответствует диапазону изменения данных от `-Inf` до `Inf`. Линии и грани с цветами, равными `NaN`, отсекаются.

Функция `caxis` возвращает двухэлементный вектор с элементами `[cmin cmax]` для текущего светового эффекта. `m`-файл с именем `caxis` задает свойства `CLim` и `CLimMode` объекта `axes` (см. команду `help axes`).

6.8.3. Окраска поверхностей

Для окраски поверхностей используется команда `shading`, которая управляет объектами `surface` (поверхность) и `patch` (заплата), создаваемыми командами и функциями `surf`, `mesh`, `pcolor`, `fill` и `fill3`. Команда `shading` (затенение) работает с параметрами и имеет следующий вид:

- `shading flat` задает окраску ячеек или граней в зависимости от текущих данных;
- `shading interp` задает окраску с билинейной интерполяцией цветов;
- `shading faceted` – равномерная раскраска ячеек поверхности (принята по умолчанию).

Эти команды устанавливают свойства `EdgeColor` и `FaceColor` для графических объектов `surface` и `patch` в зависимости от того, какая из команд – `mesh` (сетчатая поверхность) или `surf` (затененная поверхность) – используется. Примеры применения команд `shading` уже приводились.

6.8.4. Установка палитры псевдоцветов

Довольно часто возникает необходимость представления той или иной матрицы в цветах. Для этого используют *псевдоцвета*, зависящие от содержимого ячеек. Такое представление реализуют команды класса `pcolor`:

- `pcolor(C)` задает представление матрицы `C` в псевдоцвете;
- `pcolor(X, Y, C)` задает представление матрицы `C` на сетке, формируемой векторами или матрицами `X` и `Y`.

Функция `pcolor` возвращает дескриптор объекта класса `surface`. Пример применения команды `pcolor` приводится ниже:

```
>> z=peaks(40); colormap(hsv); pcolor(z)
```

График, построенный в этом примере, показан на рис. 6.46.

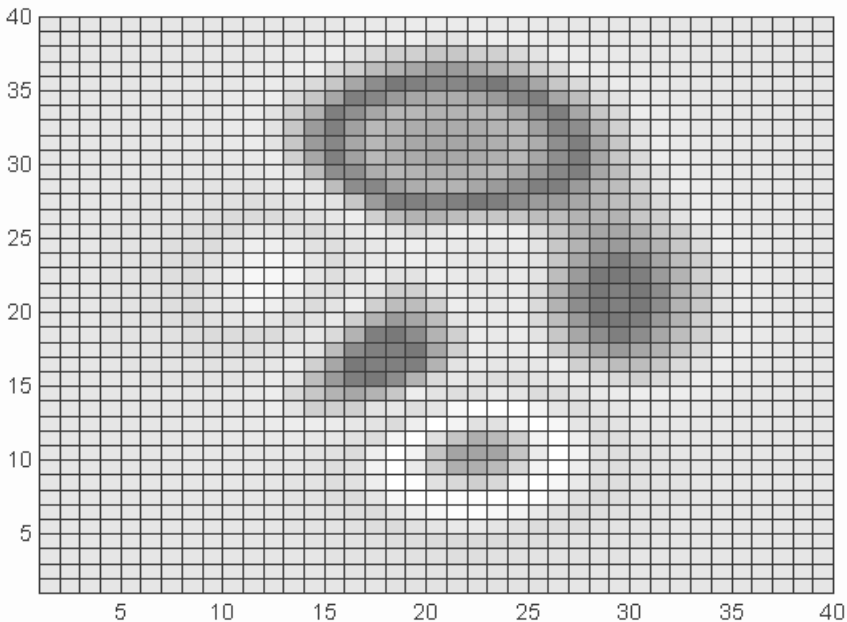


Рис. 6.46. Пример применения команды `pcolor`

Характер расцветки поверхности командой `pcolor` существенно зависит от выбора палитры цветов. В приведенном примере она задается командой `colormap`.

6.8.5. Создание закрашенного многоугольника

Для создания закрашенного пятна в виде многоугольника может использоваться команда `patch`:

- `patch(X, Y, C)` создает закрашенный многоугольник, вершины которого заданы векторами X и Y в текущей системе координат, а спецификация окраски задана вектором цветовой палитры C . Можно также задавать цвет с помощью символьной переменной `'color'` вида `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, `'w'` или `'k'`. X и Y могут быть матрицами;
- `patch(X, Y, Z, C)` создает многоугольник в трехмерной системе координат, при этом матрица Z должна иметь тот же размер, что X и Y .

Следующий пример поясняет применение команды `patch`:

```
>> X=[1 2 3 2 1]; Y=[1 2 0 5 1]; patch(X,Y,[1 0 0])
```

Построенный многоугольник показан на рис. 6.47.

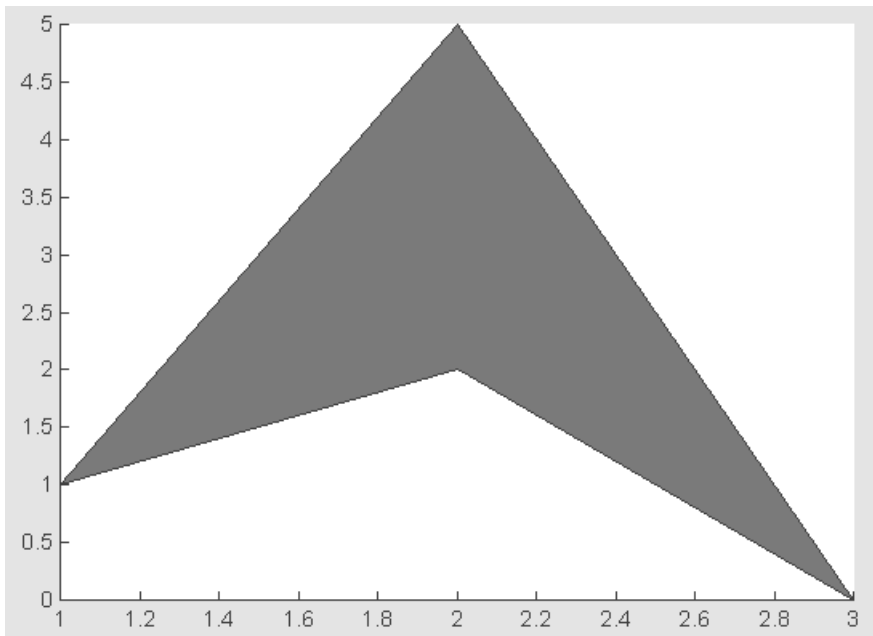


Рис. 6.47. Многоугольник, построенный командой `patch`

В данном случае многоугольник окрашен красным цветом, поскольку вектор цветов `[1 0 0]` указывает на наличие только красной составляющей цвета (другие составляющие представлены относительным уровнем 0).

6.8.6. Окраска плоских многоугольников

Для построения окрашенных в заданный цвет плоских многоугольников может использоваться команда `fill` (заполнить):

- `fill (X, Y, C)` строит закрашенный плоский многоугольник, вершины которого задаются векторами X и Y с цветом, заданным C . Многоугольник должен быть замкнутым. Для построения нескольких прямоугольников параметры команды должны быть матрицами;
- `fill (X1, Y1, C1, X2, Y2, C2, ...)` представляет собой другой способ построения нескольких закрашенных прямоугольников.

Следующий пример показывает построение четырехугольника, закрашенного синим цветом:

```
>> X=[1 2 3 2 1]; Y=[5 0.5 0 4 5]; fill(X,Y,[0 0 1])
```

Построения, реализованные этим примером, показаны на рис. 6.48.

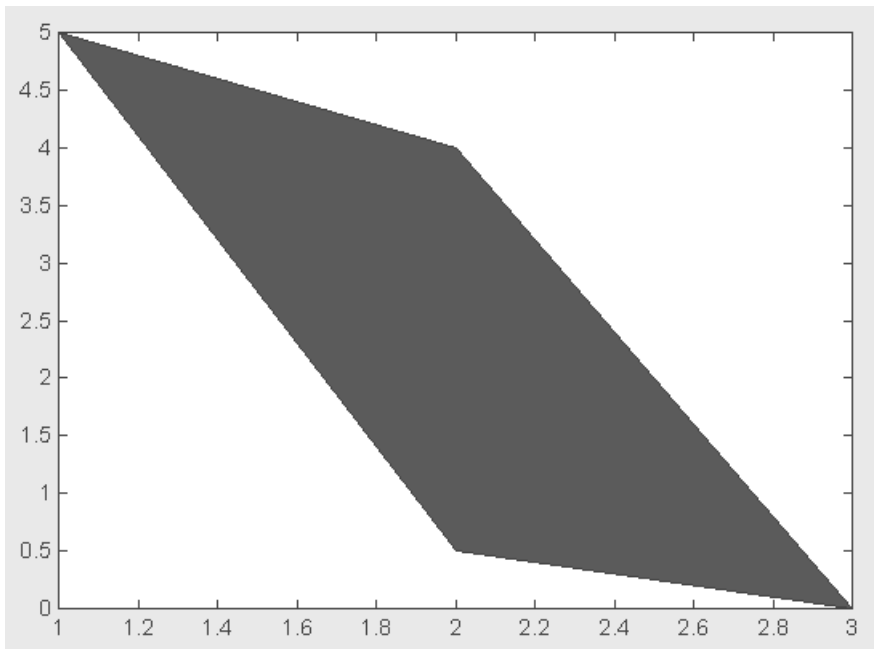


Рис. 6.48. Построение закрашенного четырехугольника на плоскости

Функция `H=fill (...)` строит график и возвращает вектор-столбец дескрипторов для созданных объектов класса `patch`, по одному дескриптору на каждый объект.

6.8.7. Вывод шкалы цветов

При использовании функциональной окраски весьма полезным является вывод *шкалы цветов* командой `colorbar`. Ее варианты перечислены ниже:

- `colorbar('vert')` выводит вертикальную шкалу цветов на текущий график;
- `colorbar('horiz')` выводит горизонтальную шкалу цветов на текущий график;
- `colorbar(N)` выводит шкалу цветов на график с дескриптором `N` с автоматическим размещением шкалы по вертикали или по горизонтали в зависимости от соотношения размеров графика;
- `colorbar` устанавливает в текущий график новую вертикальную шкалу цветов или обновляет уже имеющуюся.

Следующий пример показывает применение команды `colorbar` совместно с командой `fill3`:

```
>> fill3(rand(5,4), rand(5,4), rand(5,4), rand(5,4))  
>> colorbar('vert')
```

Более подробно функция `fill3` будет рассмотрена ниже. На рис. 6.49 показана полученная при запуске этого примера картина. Следует отметить, что поскольку многоугольники строятся со случайными значениями координат вершин, то при каждом запуске будет наблюдаться новая картина.

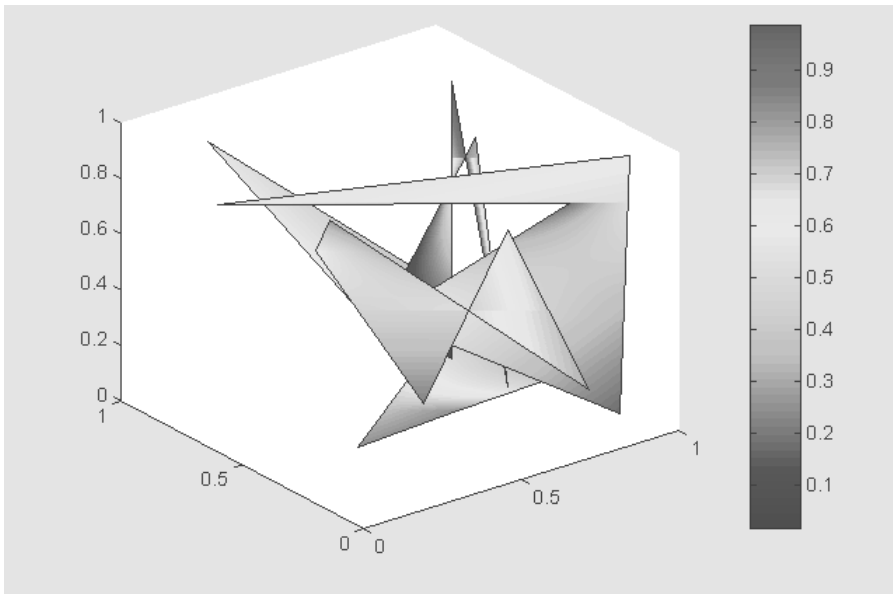


Рис. 6.49. Случайные многоугольники с функциональной окраской и вертикальной шкалой цветов

Функция `h=colorbar(...)` возвращает дескриптор для объекта `axes` со шкалой цветов.

6.8.8. Цветные плоские круговые диаграммы

Закрашенные секторы часто используются для построения *круговых диаграмм*. Для этого в MATLAB служит команда `pie`:

- `pie(X)` строит круговую диаграмму по данным нормализованного вектора $X/\text{SUM}(X)$. $\text{SUM}(X)$ – сумма элементов вектора. Если $\text{SUM}(X) \leq 1.0$, то значения в X непосредственно определяют площадь секторов;
- `pie(X, EXPLODE)` строит круговую диаграмму, у которой отрыв секторов от центра задается вектором `EXPLODE`, который должен иметь тот же размер, что и вектор данных X .

Следующий пример строит цветную круговую диаграмму с пятью секторами, причем последний сектор отделен от остальных:

```
>> x=[1 2 3 4 5]; pie(x,[0 0 0 0 2])
```

Построенная диаграмма показана на рис. 6.50.

Функция `h=pie(...)` строит график и возвращает вектор дескрипторов созданных объектов классов `patch` и `text`.

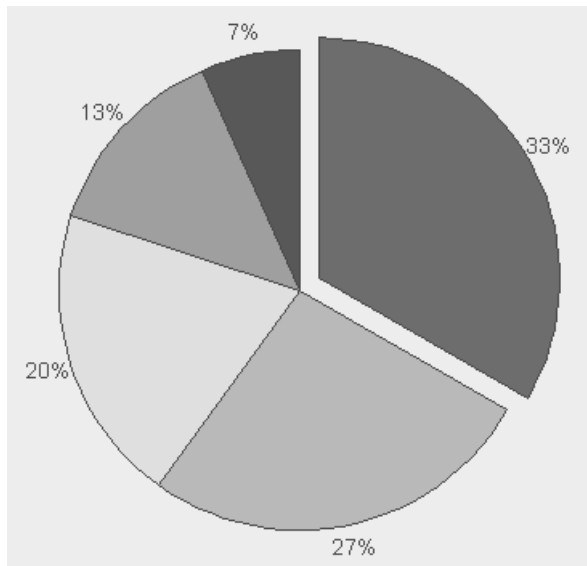


Рис. 6.50. Плоская круговая диаграмма

6.8.9. Окрашенные многоугольники в пространстве

Для закраски многоугольников, определенных в пространстве, служит команда `fill3`. Ниже представлены основные ее формы:

- `fill3(X, Y, Z, C)` строит окрашенный многоугольник в пространстве с данными вершин, хранящимися в векторах `X`, `Y` и `Z`, и цветом, заданным палитрой `C`. При построении нескольких окрашенных многоугольников параметры команды должны быть матрицами;
- `fill3(X1, Y1, Z1, C1, X2, Y2, Z2, C2, ...)` – другой вариант построения нескольких окрашенных многоугольников в пространстве;
- `fill3` – функция, возвращающая вектор-столбец дескрипторов объектов класса `patch`.

Следующий пример показывает действие команды `fill3`:

```
>> fill3(rand(5,4), rand(5,4), rand(5,4), rand(5,4))
```

На рис. 6.51 представлены построенные в этом примере окрашенные многоугольники. Поскольку координаты вершин многоугольников формируются с применением генератора случайных чисел, то наблюдаемая картина оказывается случайной и не будет повторяться при последующих запусках данного примера.

Следует обратить внимание на то, что команда `fill3` дает функциональную закраску построенных фигур.

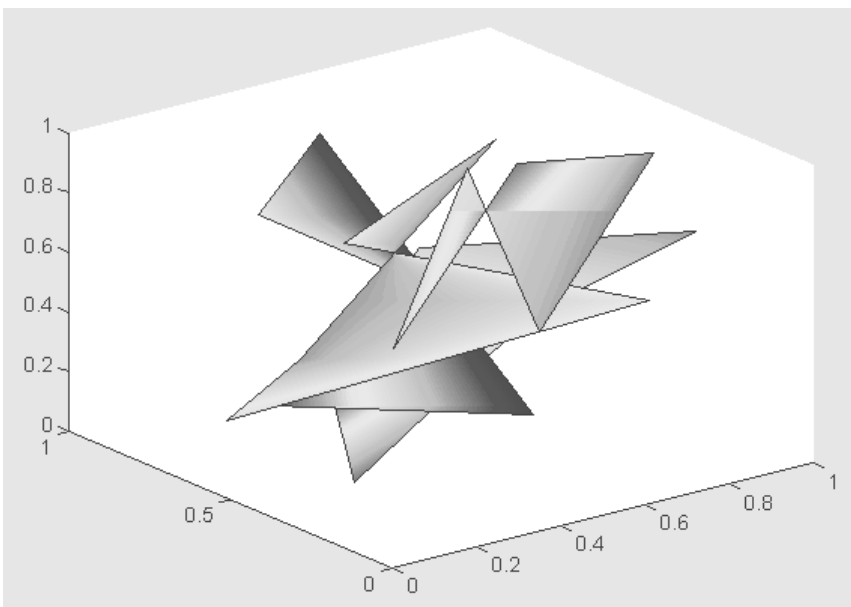


Рис. 6.51. Закрашенные многоугольники в пространстве

6.8.10. Цветные объемные круговые диаграммы

Иногда используются *объемные круговые диаграммы*. Для их построения служит команда `pie3`:

- `pie3(...)` аналогична команде `pie(...)`, но дает построение объемных секторов.

В приведенном ниже примере строится объемная диаграмма с отделением двух секторов, показанная на рис. 6.52.

```
>> X=[1 2 3 4 5]; pie3(X,[0 0 1 0 1])
```

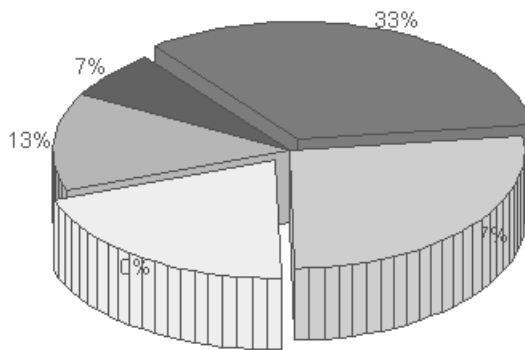


Рис. 6.52. Объемная круговая диаграмма

Функция `H=pie3(...)` строит график и возвращает вектор, содержащий дескрипторы созданных объектов классов `patch`, `surface` и `text`.

6.8.11. Другие команды управления световыми эффектами

Здесь мы только отметим некоторые дополнительные команды, связанные с управлением цветовыми палитрами:

- `colstyle` – выделение цвета и стиля графика из заданного массива;
- `rgbplot` – изображение палитры;
- `hsv2rgb` – преобразование палитры HSV в палитру RGB;
- `rgb2hsv` – преобразование палитры RGB в палитру HSV;
- `brighten` – управление яркостью;
- `contrast` – управление контрастом;
- `hidden` – управление показом невидимых линий;
- `whitebg` – управление цветом фона.

Рекомендуется ознакомиться с информацией об этих командах с помощью команды `help`.

6.9. Другие возможности графики

6.9.1. Построение цилиндра

Для построения цилиндра в виде трехмерной фигуры применяется функция `cylinder`:

- `[X, Y, Z]=cylinder(R, N)` создает массивы `X`, `Y` и `Z`, описывающие цилиндрическую поверхность с радиусом `R` и числом узловых точек `N` для последующего построения с помощью функции `surf(X, Y, Z)`;
- `[X, Y, Z]=cylinder(R)` и `[X, Y, Z]=cylinder` подобны предшествующей функции для `N=20` и `R=[1 1]`.

Пример построения объемного цилиндра:

```
>> [X, Y, Z]=cylinder(10, 30); surf(X, Y, Z, X)
```

На рис. 6.53 показан результат построения цилиндра для данного примера:

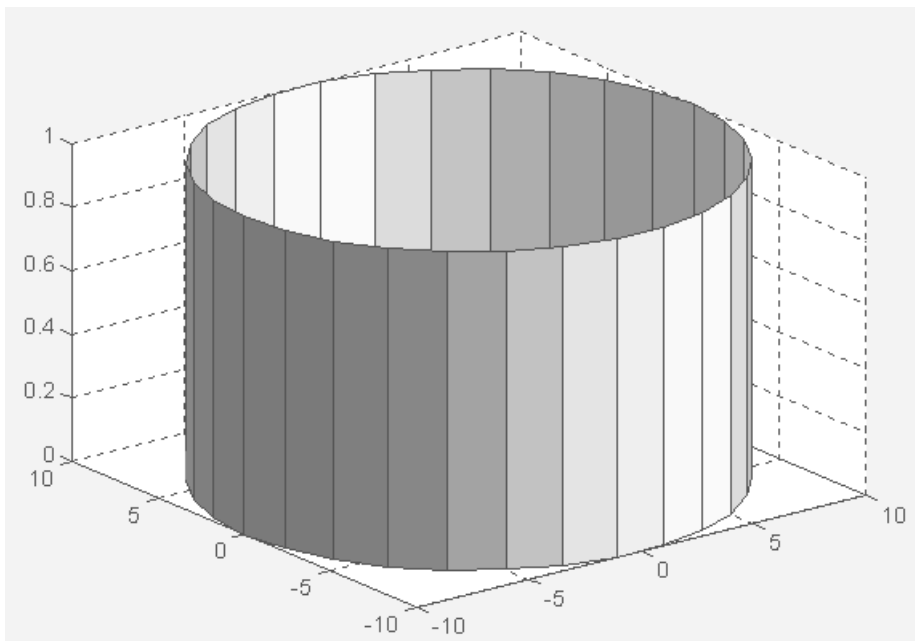


Рис. 6.53. Построение цилиндра

Естественность воспроизведения цилиндра существенно зависит от графической команды, используемой для его построения. Команда `surf` дает возможность задать функциональную окраску с цветом, определяемым вектором `X`, что делает представление цилиндра достаточно наглядным.

6.9.2. Построение сферы

Для расчета массивов X , Y и Z координат точек сферы как трехмерной фигуры используется функция `sphere`:

- `[X, Y, Z]=sphere(N)` генерирует матрицы X , Y и Z размера $(N+1) \times (N+1)$ для последующего построения сферы с помощью команд `surf(X, Y, Z)` или `surfl(X, Y, Z)`;
- `[X, Y, Z]=sphere` аналогична предшествующей функции при $N=20$.

Пример применения этой функции:

```
>> [X, Y, Z]=sphere(30); surfl(X, Y, Z)
```

На рис. 6.54 показана построенная в этом примере сфера. Хорошо видны геометрические искажения (сфера приплюснута), связанные с разными масштабами по координатным осям.

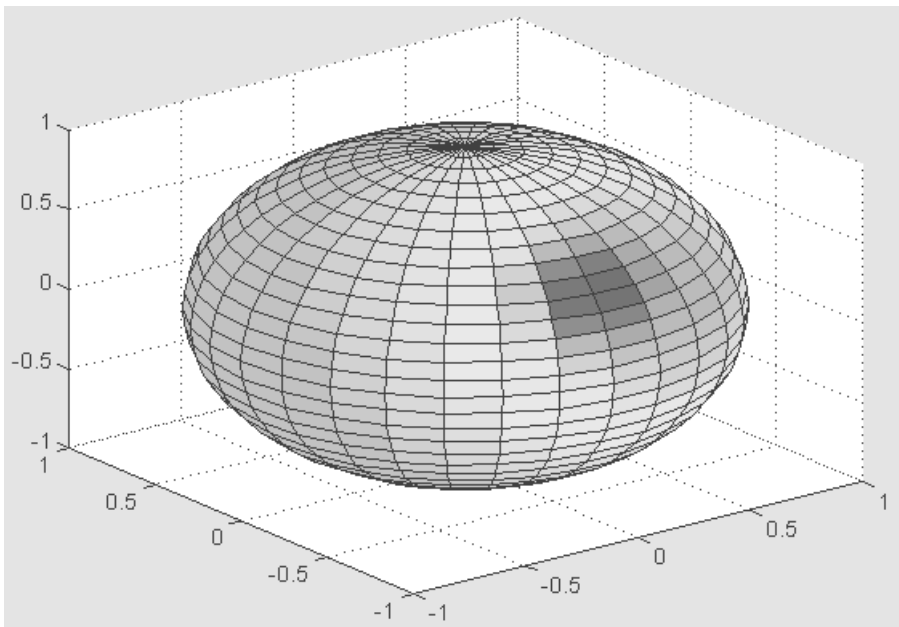


Рис. 6.54. Построение сферы

Обратите внимание на то, что именно функциональная окраска сферы придает ей довольно реалистичный вид. В данном случае цвет задается вектором Z .

6.9.3. 3D-графика с треугольными плоскостями

К числу специальных видов графики относится построение объемных фигур с помощью плоских треугольников. Для построения таких фигур в виде каркаса (без окраски и отображения плоскостей) используется команда `trimesh`:

- `trimesh(TRI, X, Y, Z, C)` – построение объемной каркасной фигуры с треугольниками, специфицированными матрицей поверхности `TRI`, каждая строка которой содержит три элемента и задает одну треугольную грань путем указания индексов, по которым координаты выбираются из векторов `X`, `Y`, `Z`. Цвета ребер задаются вектором `C`;
- `trimesh(TRI, X, Y, Z)` – построение, аналогичное предшествующему при `C=Z`, то есть с цветом ребер, зависящим от значений высоты;
- `N=trimesh(...)` строит график и возвращает дескрипторы графических объектов;
- `trimesh(..., 'param', 'value', 'param', 'value'...)` добавляет значения `'value'` для параметров `'param'`.

Следующая программа иллюстрирует применение команды `trimesh` для построения случайной объемной фигуры, параметры которой задаются с помощью генератора случайных чисел:

```
% Программа построения случайной объемной фигуры
x = rand(1,40); y = rand(1,40);
z = sin(x.*y); tri = delaunay(x,y);
trimesh(tri,x,y,z)
```

Одна из построенных фигур показана на рис. 6.55. Учтите, что при каждом пуске программы строится новая фигура.

Другая, абсолютно аналогичная по заданию входных параметров команда, – `trisurf(...)` – отличается только закраской треугольных областей, задающих трехмерную фигуру. Если в приведенном выше примере заменить функцию `trimesh` на `trisurf`, то можно получить графики, подобные приведенному на рис. 6.56.

Обратите внимание на то, что рис. 6.56 также принадлежит к множеству случайных графических построений. Поэтому возможность его буквального повторения отсутствует.

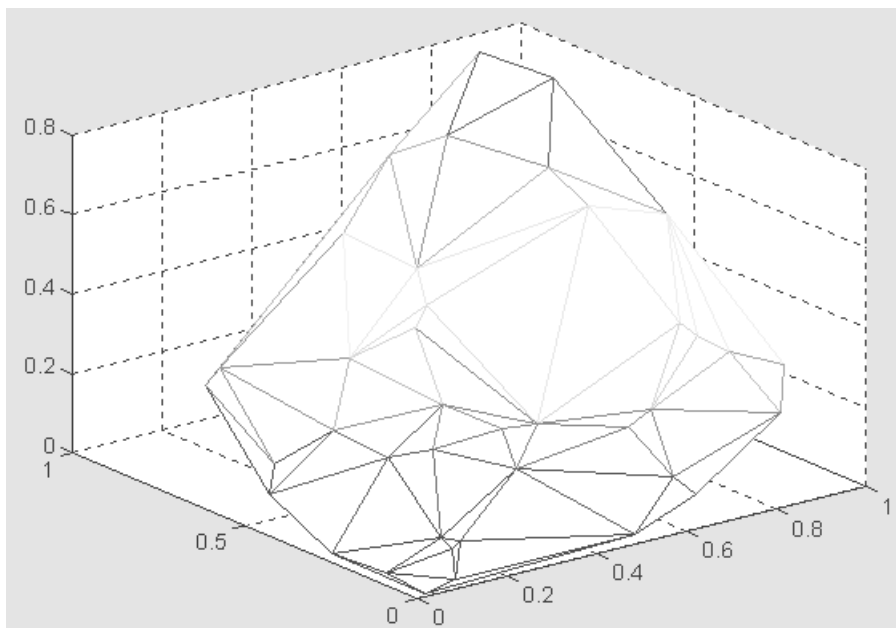


Рис. 6.55. Одна из объемных фигур, построенных командой `trimesh`

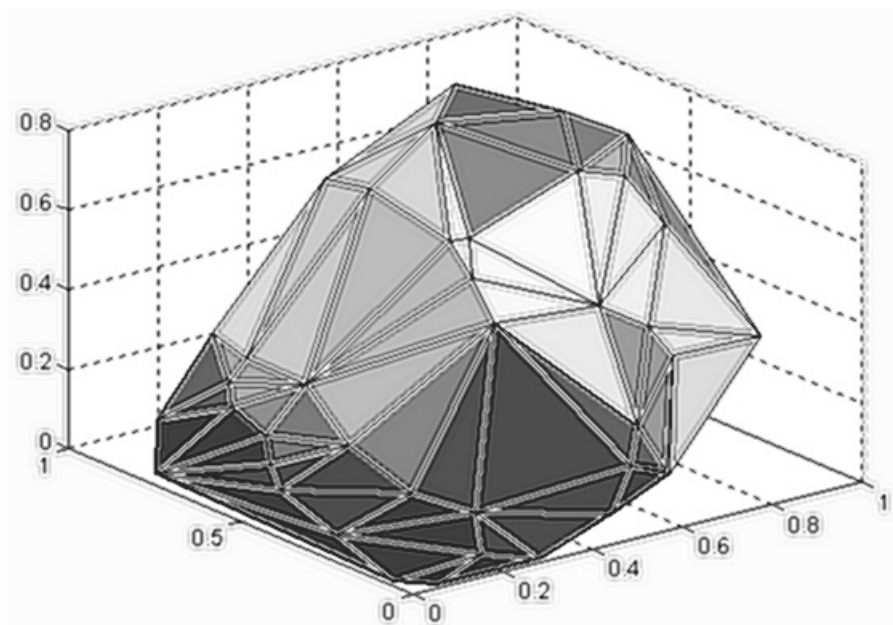


Рис. 6.56. Один из рисунков, построенных командой `trisurf`

Программные средства специальной графики

| | |
|---|-----|
| 7.1. Анимационная графика | 342 |
| 7.2. Основы дескрипторной графики | 347 |
| 7.3. Галерея трехмерной графики | 357 |
| 7.4. Графический интерфейс пользователя GUI | 362 |
| 7.5. Графическая поддержка цвета | 369 |
| 7.6. Расширенная техника визуализации вычислений | 372 |

В этом уроке мы рассмотрим некоторые виды специальной графики. Это прежде всего *анимационная* и *дескрипторная* (handle) графика [13, 16, 46]. Эти виды графики существенно расширяют степень визуализации результатов вычислений, но рассчитаны на достаточно опытного пользователя. Мы также опишем галерею графики и средства графического интерфейса пользователя GUI.

7.1. Анимационная графика

7.1.1. Движение точки на плоскости

Начнем с высокоуровневой реализации *анимации* (оживления) графических изображений. Для отображения движения точки по траектории используется команда `comet`. При этом движущаяся точка напоминает ядро кометы с длинным угасающим хвостом. Используются следующие формы представления этой команды:

- `comet (Y)` отображает движение «кометы» по траектории, заданной вектором Y ;
- `comet (X, Y)` отображает движение «кометы» по траектории, заданной парой векторов Y и X ;
- `comet (X, Y, p)` аналогична предшествующей команде, но позволяет задавать длину хвоста кометы (отрезка траектории, выделенного цветом) как $p * \text{length}(Y)$, где $\text{length}(Y)$ – размер вектора Y , а $p < 1$. По умолчанию $p = 0.1$ ¹.

Следующий пример иллюстрирует применение команды `comet`:

```
>> X=0:0.01:15; comet(X, sin(X), 0.15)
```

Стоп-кадр изображения показан на рис. 7.1.

«Хвост кометы» на черно-белом рисунке заметить трудно, поскольку он представляет собой отрезок линии с цветом, отличающимся от цвета линии основной части графика.

7.1.2. Движение точки в пространстве

Есть еще одна команда, которая позволяет наблюдать движение точки, но уже в трехмерном пространстве, – это команда `comet3`:

- `comet3 (Z)` отображает движение точки с цветным «хвостом» по трехмерной кривой, определенной массивом Z ;
- `comet3 (X, Y, Z)` отображает движение точки «кометы» по кривой в пространстве, заданной точками $[X(i), Y(i), Z(i)]$;
- `comet3 (X, Y, Z, p)` аналогична предшествующей команде с заданием длины «хвоста кометы» как $p * \text{length}(Z)$. По умолчанию параметр p равен 0.1 .

Ниже представлен пример применения команды `comet3`:

```
>> W=0:pi/500:10*pi; comet3(cos(W), sin(W)+W/10, W)
```

¹ Обратите внимание, что если вы используете лупу, как-то иначе пытаетесь изменить размер вашего рисунка или используете вкладку **Copy Figure** меню **Edit**, то график, полученный при использовании `comet` или `comet3`, исчезает.

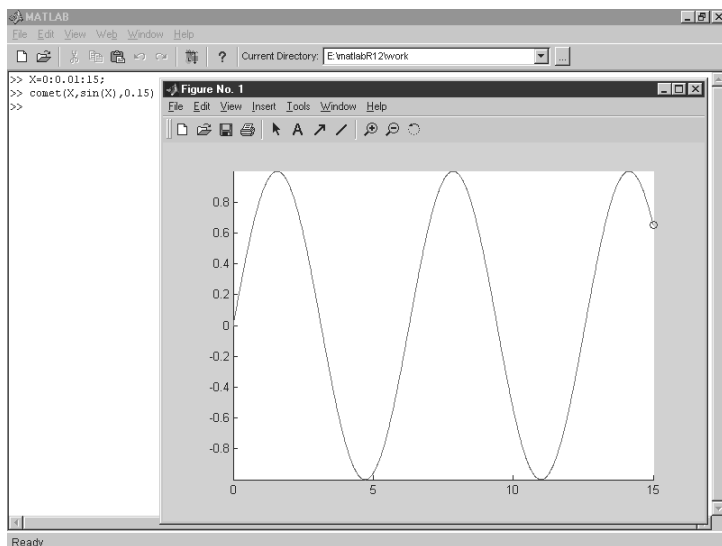


Рис. 7.1. Стоп-кадр изображения, полученный из примера использования команды `comet`

На рис. 7.2 показан стоп-кадр изображения, созданного командой `comet3`.

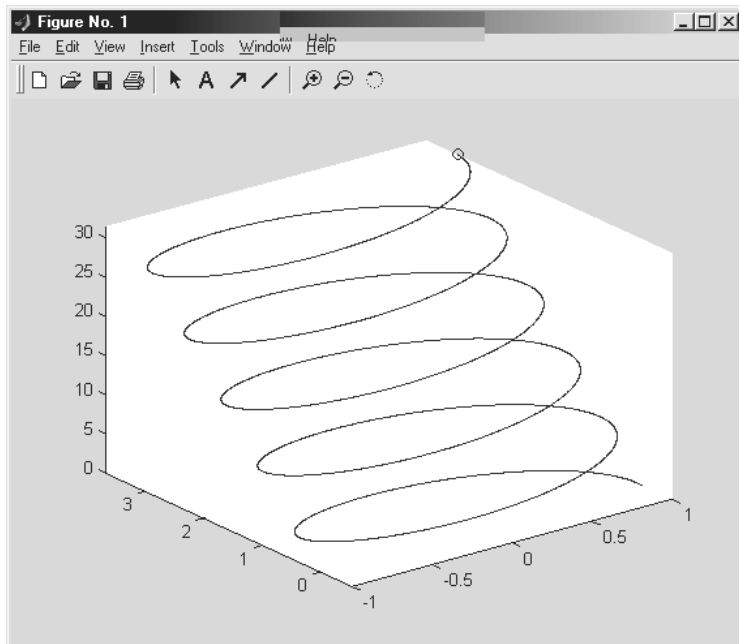


Рис. 7.2. Стоп-кадр изображения, созданного командой `comet3`

Разумеется, движение точки по заданной траектории как в двумерном, так и в трехмерном пространстве является самым простейшим примером анимации. Тем не менее эти средства существенно расширяют возможности графической визуализации при решении ряда задач динамики.

7.1.3. Основные средства анимации

Для более сложных случаев анимации возможно применение техники *мультипликации*. Она сводится к построению ряда кадров изображения, причем каждый кадр появляется на некоторое время, затем стирается и заменяется на новый кадр, несколько отличающийся от предшествующего. Если это отличие незначительно, то создается иллюзия плавного перемещения объекта.

Отметим кратко основные команды, реализующие анимацию в системе MATLAB:

- `capture` – захват видеоизображения;
- `getframe` – создание кадра для анимации;
- `moviein` – выполнение анимации;
- `rotate` – вращение фигуры;
- `frame2im` – преобразование кадра в графический образ;
- `im2frame` – преобразование графического образа в кадр.

Применение некоторых из этих команд мы рассмотрим далее на конкретных примерах.

7.1.4. Вращение фигуры – логотипа MATLAB

Рассмотрим вначале не очень сложный пример вращения сложной трехмерной поверхности – логотипа системы MATLAB, который представлен файлами **logo.m** и **logo.mat**. Ниже изображен фрагмент программы, обеспечивающий вращение этой поверхности (фигуры) относительно осей системы координат:

```
% Программа вращения логотипа системы MATLAB
if ~exist('MovieGUIFlag'), figNumber=0; end; load logo
h=surfl(L,source); colormap(M);
ax=[7 52 7 52 -.5 .8]; axis(ax); axis on;
shading interp; m=moviein(25);
for n=1:25,
    rotate(h,[0 90],15,[21 21 0]);
h=surfl(get(h,'XData'),get(h,'YData'),get(h,'ZData'),
source);
    axis(ax);    axis on;shading interp;
    m(:,n)=mvframe(figNumber,24);
end;
mvstore(figNumber,m);
```

Эта программа имеет два блока: в первом задаются исходная функция и ее образ, а во втором (с циклом `for`) выполняются создание кадров и их последова-

тельное воспроизведение, создающее эффект анимации. На рис. 7.3 показан стоп-кадр полученной анимации.

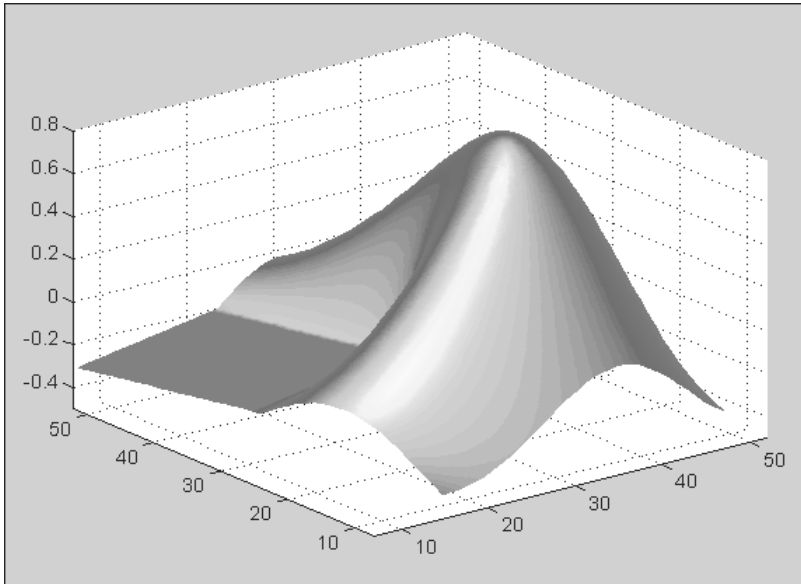


Рис. 7.3. Стоп-кадр программы, обеспечивающей вращение фигуры – логотипа MATLAB

7.1.5. Волновые колебания мембраны

Принцип мультипликации легко распространить на существенно более сложные задачи анимации. В качестве иллюстрации можно рассмотреть системный пример `vibes`, демонстрирующий волнообразные колебания тонкой пластины – мембраны. Ниже представлен переработанный файл данного примера, в котором сокращены подробные комментарии на английском языке и введены (только для пояснения) комментарии на русском языке:

```
% Волновые колебания мембраны
% Переработка файла VIBES фирмы MathWorks, Inc.
if~exist('MovieGUIFlag'); figNumber=0; end;
hlpStr= ...
    [' Это пример анимации – наблюдение колебаний '
    ' трехмерной поверхности – мембраны. '];
mvinit(figNumber,hlpStr);
% Загрузка данных функции
load vibesdat;[n,n] = size(L1);
nh = fix(n/2); x = (-nh:nh)/nh;
% Вычисление коэффициентов
```

```

clear c
for k = 1:12
    eval(['c(k) = L' num2str(k) '(24,13)/3;'])
end;
% Установка графических параметров
axis([-1 1 -1 1 -1 1]);caxis(26.9*[-1.5 1]);
colormap(hot); hold on
% Генерация кадров мультипликации
delt = 0.1; nframes = 12; M = moviein(nframes);
for k = 1:nframes,
    % Коэффициенты
    t = k*delt; s = c.*sin(sqrt(lambda)*t);
    % Амплитуды
    L = s(1)*L1 + s(2)*L2 + s(3)*L3 + s(4)*L4 + s(5)*L5 ... +
s(6)*L6 + s(7)*L7 + s(8)*L8 + s(9)*L9 + s(10)*L10 + ... s(11)*L11 +
s(12)*L12;
    % Скорость мультипликации
    s = s .* lambda;
    V = s(1)*L1 + s(2)*L2 + s(3)*L3 + s(4)*L4 + s(5)*L5...
+ s(6)*L6 + s(7)*L7 + s(8)*L8 + s(9)*L9 + s(10)*L10...
+ s(11)*L11 + s(12)*L12;
    % График поверхности; цвет задается скоростью
    V(1:nh,1:nh) = NaN*ones(nh,nh); Cla
surf(x,x,L,V); axis off
    % Создание кадров мультипликации
    M(:,k) = mvframe(figNumber,nframes);
end;
hold off
%=====
% Запись кадров мультипликации
mvstore(figNumber,M);

```

Этот пример дан с целью иллюстрации, и подробно данную программу мы описывать не будем. Ограничимся приведением заключительного кадра (рис. 7.4) анимации, показывающей колебания мембраны.

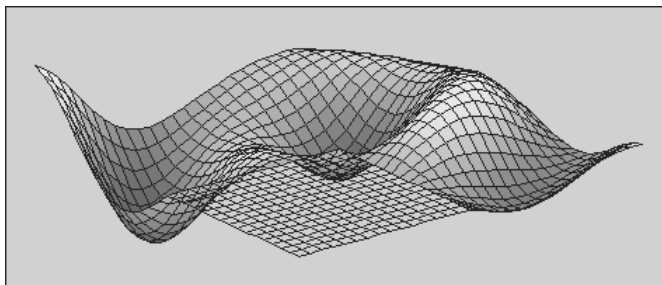


Рис. 7.4. Заключительный кадр анимации, демонстрирующей колебания мембраны

7.2. Основы дескрипторной графики

7.2.1. Объекты дескрипторной графики

Графические средства MATLAB базируются на низкоуровневой графике, которая называется *дескрипторной* (описательной), или handle-графикой. По существу, эта графика обеспечивает объектно-ориентированное программирование как всех рассмотренных выше графических команд, так и пользовательского интерфейса.

Центральным понятием дескрипторной графики является *графический объект*. Имеются следующие типы таких объектов:

- `root` (корень) – первичный объект, соответствующий экрану компьютера;
- `figure` (рисунок) – объект создания графического окна;
- `uicontrol` (элемент управления, определенный пользователем) – объект создания элемента пользовательского интерфейса;
- `axes` (оси) – объект, задающий область расположения графика в окне объекта `figure`;
- `uimenu` (определенное пользователем меню) – объект создания меню;
- `uicontextmenu` (определенное пользователем контекстное меню) – объект создания контекстного меню;
- `image` (образ) – объект создания растровой графики;
- `line` (линия) – объект создания линии;
- `patch` (заплата) – объект создания закрашенных фигур;
- `rectangle` (прямоугольник) – объект создания закрашенных прямоугольников;
- `surface` (поверхность) – объект создания поверхности;
- `text` (текст) – объект создания текстовых надписей;
- `light` (свет) – объект создания эффектов освещенности.

Объекты подчас взаимосвязаны и могут обращаться друг к другу для получения того или иного графического эффекта.

7.2.2. Создание графического окна и управление им

Прежде чем мы рассмотрим применение дескрипторной графики на реальных примерах, отметим команды и функции, которые предназначены для создания графических окон и управления ими:

- `figure` – открыть чистое графическое окно;
- `gcf` – получить дескриптор текущего графического окна `figure`;
- `clf` – очистить графическое окно;
- `shg` – показать ранее свернутое графическое окно;
- `close` (закрыть) – закрыть графическое окно;
- `refresh` (обновить) – обновить графическое окно.

Эти команды и функции достаточно очевидны, и мы не будем обсуждать их подробно. Заметим, что команды `help name` или `doc name` позволяют уточнить назначение той или иной команды или функции с обобщенным именем `name`.

7.2.3. Создание координатных осей и управление ими

Еще одна группа простых команд служит для создания координатных осей и управления ими:

- `axes` (оси) – создать оси координат;
- `box` (ящик) – построить прямоугольник вокруг рисунка;
- `cla` – убрать построения `axes`;
- `gca` – получить дескриптор графического объекта `axes`;
- `hold` – сохранить оси координат;
- `ishold` – проверка статуса `hold` (1, если оси сохранены, и 0 в противоположном случае).

Эти команды также достаточно очевидны. Заметим, что их можно использовать и в обычной (высокоуровневой) графике, например для устранения осей из уже созданного графика.

7.2.4. Пример применения объекта дескрипторной графики

Приведем пример задания и использования графического объекта. Пусть надо построить линию, проходящую через три точки с координатами (0, 1), (2, 4) и (5, –1). Для этого воспользуемся объектом `line`, который порождается одноименной графической функцией:

```
>> line([0 2 5],[1 4 -1], 'Color', 'blue')
```

На рис. 7.5 построена заданная линия с помощью дескрипторной команды `line`, которая явно не входит в высокоуровневую графику. Однако нетрудно понять, что именно эта команда составляет основу высокоуровневой команды `plot`, описанной ранее.

Особенность команды `line` заключается в явном задании всех условий построения графика: координат конкретных точек, параметра цвета `'Color'` и самого цвета `'blue'` (синий). В итоге строятся два отрезка прямой, проходящие через заданные точки и имеющие синий цвет.

7.2.5. Дескрипторы объектов

С понятием объектов дескрипторной графики связана особая характеристика объектов – *дескриптор* (описатель). Его можно понимать как некое число – своеобразный идентификатор («распознаватель») объектов. Дескриптор объектов `root` всегда равен 0, а дескриптор объектов `figure` (рисунок) – это целое число,

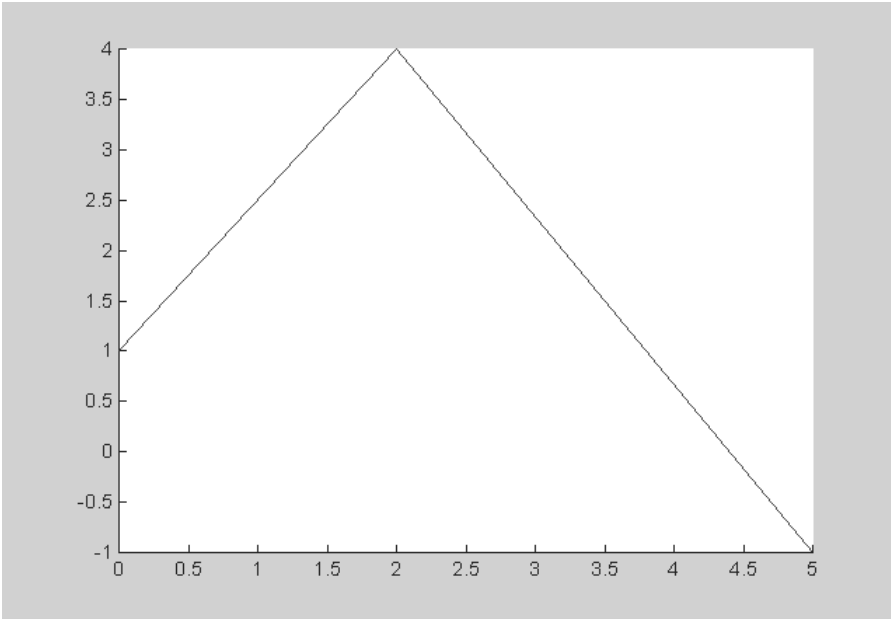


Рис. 7.5. Построение отрезков прямой объектом *line*

указывающее на номер графического окна. Дескрипторы других объектов – это числа с плавающей запятой. По значениям дескрипторов MATLAB идентифицирует объекты. Дескриптор одного такого объекта представляет собой одно число, а если объектов несколько – несколько чисел (вектор). Например, следующие команды строят пять графиков, представляющих значения элементов магической матрицы (магического квадрата), в одном окне:

```
>> A=magic(5);
>> h=plot(A)
h =
    3.0013
   101.0009
   102.0004
   103.0004
   104.0004
```

В данном случае вектор *h* содержит дескрипторы элементов графика, показанного на рис. 7.6.

Мы еще раз обращаем ваше внимание на то, что дескрипторы дают лишь внутреннее описание того или иного объекта, и ассоциировать их явно с привычными параметрами, например координатами или цветом объекта, не следует. Более того, нет никаких оснований считать их одинаковыми для разных версий MATLAB, для разных компьютерных платформ и даже для одинаковых команд, но в разных местах сессии.

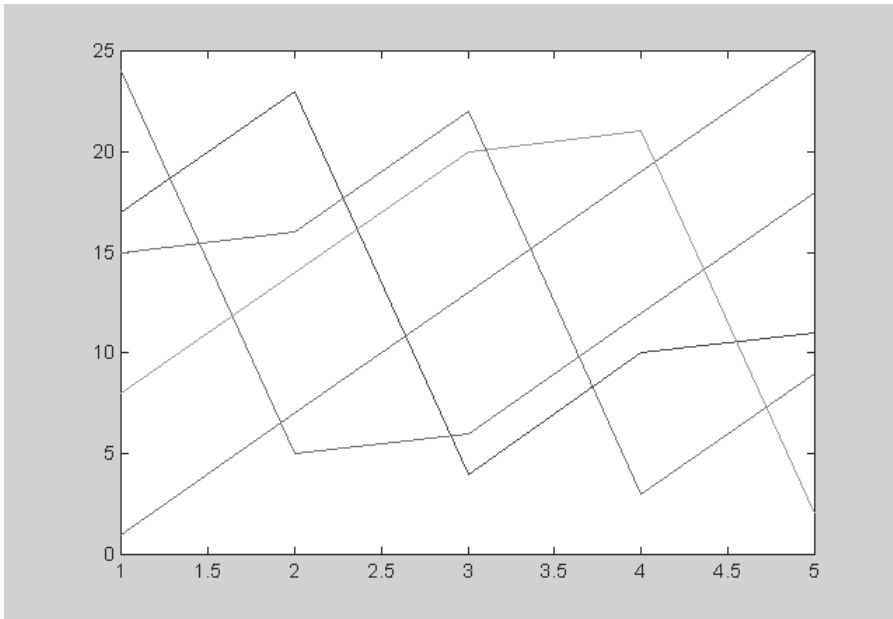


Рис. 7.6. Графики пяти функций, представляющих значения элементов магической матрицы $\text{magic}(5)$

7.2.6. Операции над графическими объектами

К графическим объектам применяется ряд операций:

- `set` – установка свойств (параметров) графического объекта;
- `get` – вывод свойств графического объекта;
- `reset` – восстановление свойств графического объекта по умолчанию;
- `delete` – удаление созданного графического объекта;
- `gco` – возвращение дескриптора текущего графического объекта;
- `gcbo` – возвращение дескриптора объекта, чья функция в данный момент выполняется;
- `gcbf` – возвращение дескриптора окна, содержащего объект, функция которого в данный момент выполняется;
- `drawnow` – выполнение очереди задержанных графических команд;
- `findobj` – нахождение объектов с заданными свойствами;
- `copyobj` – копирование объекта и порожденных им объектов.

Кроме того, имеются три утилиты, связанные с операциями над объектами:

- `closereq` – закрытие окна по запросу;
- `ishandle` – проверка дескриптора на истинность;
- `newplot` – восстановление свойств объекта, измененных `nextPlot`.

Назначение большинства этих операций достаточно очевидно. Мы остановимся на двух наиболее важных операциях, связанных с контролем и установкой свойств объектов.

7.2.7. Свойства объектов – команда *get*

Каждый объект дескрипторной графики имеет множество параметров, определяющих его свойства. Многие свойства задаются по умолчанию и определяют такие важные характеристики объекта, как его стиль, наличие и тип координатных осей и маркеров, цветовое и световое оформление и т. д. Для получения списка свойств и их значений объекта *h* используется команда *get*, например в формах:

```
get(h), get(h, 'PropertyName'), a = get(h, 'Default')...
```

Рассмотрим ее применение. Для этого вернемся к нашему примеру с построением графика из двух отрезков линии и повторим этот пример в следующем виде:

```
>> h=line([0 2 5],[1 4 -1], 'Color', 'blue')  
h = 3.0010
```

Теперь объект имеет дескриптор *h*, и его значение выведено наряду с построением графика. Команда *get(h)* выводит свойства объекта с заданным именем:

```
>> get(h)  
Color = [0 0 1]  
EraseMode = normal  
LineStyle = -  
LineWidth = [0.5]  
Marker = none  
MarkerSize = [6]  
MarkerEdgeColor = auto  
MarkerFaceColor = none  
XData = [0 2 5]  
YData = [1 4 -1]  
ZData = []  
BeingDeleted = off  
ButtonDownFcn =  
Children = []  
Clipping = on  
CreateFcn =  
DeleteFcn =  
BusyAction = queue  
HandleVisibility = on  
HitTest = on  
Interruptible = on  
Parent = [100.001]  
Selected = off  
SelectionHighlight = on  
Tag =  
Type = line
```

```

    UIContextMenu = []
    UserData = []
    Visible = on

```

Вы видите, что эта команда выводит довольно обширный список свойств графического объекта (в нашем случае линии), заданных по умолчанию. Он содержит свойства, характерные и для других графических объектов.

7.2.8. Изменение свойств объекта – команда *set*

С помощью команды `set` можно изменить отдельные свойства объекта `h` дескрипторной графики. Эта команда имеет ряд параметров, и с ними можно ознакомиться с помощью команд `help set` или `doc set`. Приведем некоторые варианты ее задания:

```

set(h, 'PropertyName', PropertyValue, ...)
set(h, a)          set(h, pn, pv...)
set(h, pn, <m-by-n cell array>)
a= set(h)  a= set(h, 'Default')
a= set(h, 'DefaultObjectTypePropertyName')
<cell array> = set(h, 'PropertyName') ...

```

С учетом значений слов, входящих в параметры (`Property` – свойство, `Name` – имя, `Value` – значение, `Default` – по умолчанию и т. д.) их смысл вполне очевиден. В связи с этим ограничимся примером – допустим, нам надо сменить цвет линии с голубого по умолчанию на красный. Для этого достаточно выполнить следующую, первую из приведенных команду:

```
>> set(h, 'Color', 'red')
```

Обратите внимание, что при этом цвет сменится на ранее построенном рисунке с дескриптором `h`. Еще раз обращаем внимание читателя на то, что все свойства, заданные командой `set` и выводимые командой `get`, доступны при использовании редактора графики.

7.2.9. Просмотр свойств

С помощью команды `set(h, 'Property')` можно вывести полный список значений того или иного свойства конкретного объекта `h`. Например, выясним, какого стиля можно выводить линии в нашем последнем примере:

```
>> set(h, 'LineStyle')
[ {-} | - | : | -. | none ]
```

Или выясним, какого вида может быть задан графический курсор (маркер):

```
>> set(h, 'Marker')
[ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram
| hexagram | {none} ]
```

Таким образом, команды `set` и `get` позволяют легко устанавливать, контролировать и менять свойства объектов дескрипторной графики без утомительного поиска нужных свойств по справке или весьма объемной технической документации. Для детального знакомства со свойствами и их параметрами надо обращаться к справке по ним.

7.2.10. Примеры дескрипторной графики

Теперь рассмотрим более сложные примеры, наглядно демонстрирующие возможности дескрипторной графики. Воспользовавшись командой **File** ⇒ **New** ⇒ **M-File** или `edit ms1.m`, создадим файл `ms1.m` следующего содержания:

```
[x,y] = meshgrid([-2:.4:2]);Z =sin(x.^2+y.^2);  
fh = figure('Position',[350 275 400 300],'Color','w');  
ah = axes('Color',[.8 .8 .8],'XTick',[-2 -1 0 1 2],...  
'YTick',[-2 -1 0 1 2]);  
sh = surface('XData',x,'YData',y,'ZData',Z,...  
'FaceColor',get(ah,'Color')+.1,...  
'EdgeColor','k','Marker','o',...  
'MarkerFaceColor',[.5 1 .85]);
```

В этом файле заданы три объекта: прямоугольник `fh` – объект класса `figure`, оси с метками `ah` – объект класса `axes` и трехмерная поверхность `sh` – объект класса `surface`. При первом запуске файла `ms1` появится плоская сетка, показанная на рис. 7.7.

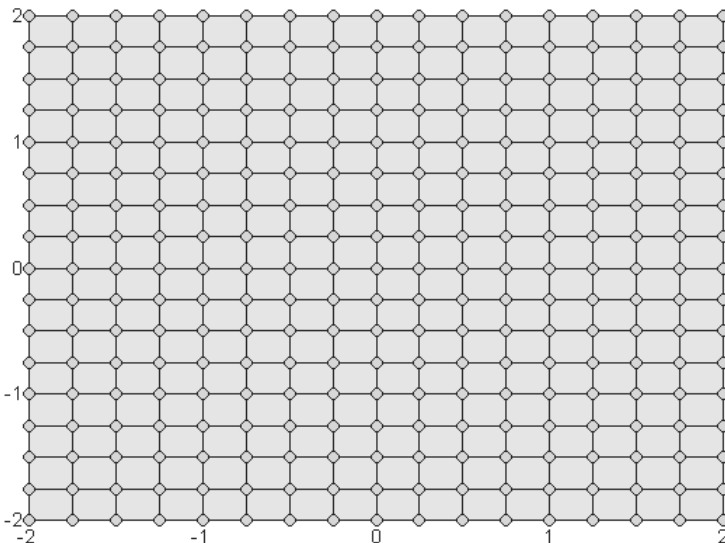


Рис. 7.7. Результат наложения объектов `fh` и `ah` друг на друга

Она является результатом наложения объектов `fh` и `ah` друг на друга. При этом объект `ah` класса `axes` явно наследует свойства объекта `fh` класса `figure`. Наследование здесь проявляется в том, что при задании свойства «цвет граней» (`FaceColor`) объекта `sh` используется осветление (добавлением константы 0.1) цвета осей, полученного при помощи функции `get(ah, color)`.

Пока ничего неожиданного в полученной двумерной фигуре нет. Но стоит исполнить команду

```
>> view(3)
```

как будет получено весьма любопытное изображение, представленное на рис. 7.8.

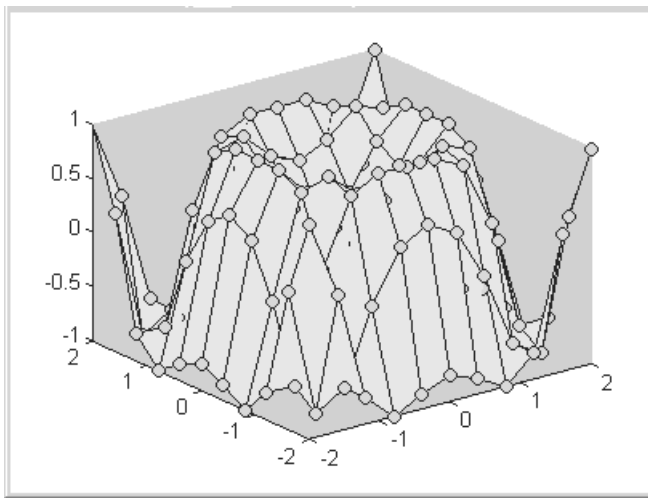


Рис. 7.8. Трехмерная поверхность, унаследовавшая узловые точки плоской фигуры

Команда `view(3)` изменяет точку обзора трехмерной поверхности. Раньше, когда параметры осей были жестко заданы (рис. 7.7), мы смотрели на поверхность строго сверху. Теперь команда `view(3)` установила точку обзора трехмерных графиков, принятую по умолчанию. Нетрудно заметить, что полученная трехмерная поверхность рис. 7.8 наследует узловые точки сетки, показанной на рис. 7.7. Таким образом, здесь в явной форме проявляется такое качество объектов, как наследование свойств, производных от родительских объектов.

Теперь создадим второй файл – **ms2.m**:

```
h(1) = axes('Position',[0 0 1 1]); sphere
h(2) = axes('Position',[0 0 .4 .6]); peaks;
h(3) = axes('Position',[0 .5 .5 .5]); sphere
h(4) = axes('Position',[.5 0 .4 .4]); sphere
h(5) = axes('Position',[.5 .5 .5 .3]); cylinder([0 0 0.5])
```

```
set(h, 'Visible', 'off')
set(gcf, 'Renderer', 'painters')1
```

Здесь задано 5 трехмерных объектов: три сферы разных размеров, поверхность peaks и цилиндр.

Еще более интересную картину мы получим, запустив файл ms2. Заново будет вычислена величина z, а затем построены изображения еще пяти фигур:

```
z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
    - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
    - 1/3*exp(-(x+1).^2 - y.^2)
```

Полученное в том же окне комбинированное изображение показано на рис. 7.9. Оно является результатом наложения новых построений трехмерных фигур на ранее построенный рис. 7.8, причем поскольку объекты axes в диаграмме иерархии объектов (см. ниже) находятся ниже объектов figure, то они строятся на том же рисунке, но так как они расположены на этой диаграмме выше объектов surface, то они находятся спереди.

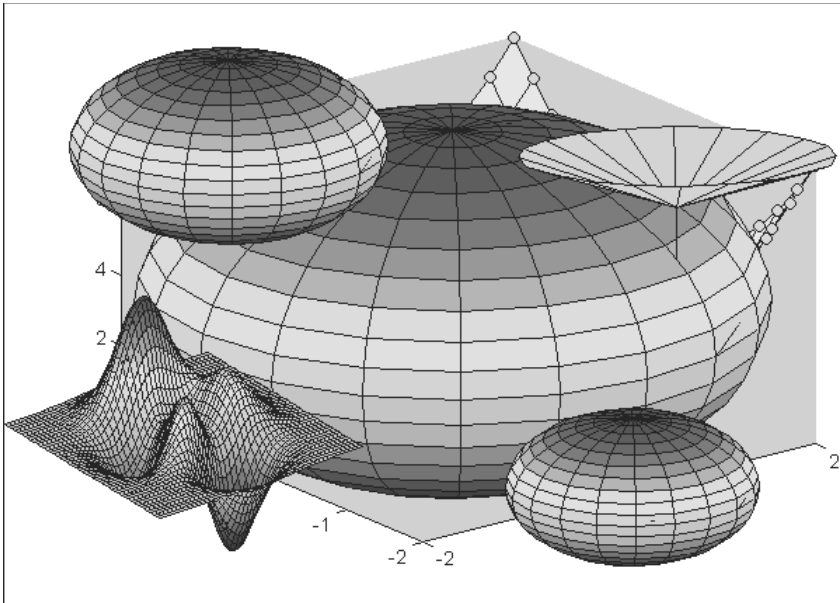


Рис. 7.9. Комбинированный рисунок, полученный при запуске файла ms2.m после запуска ms1.m

¹ Обратите внимание, что трехмерная графика в этом примере строится с рендерингом MATLAB 4 (painters). В MATLAB 5.3 по умолчанию был бы выбран рендеринг с использованием Z-буфера и на некоторых компьютерах были бы возможны искажения. В MATLAB 6.* при включенном режиме видеоадаптера TrueColor можно заменить последнюю команду на set(gcf, 'Renderer', 'opengl') или опустить ее. Но при необходимости вывода изображения на печать лучше выбрать рендеринг 'painters'.

Последовательность наложения фигур, заданных в файле `ms2`, определяется последовательностью их описания в файле. Любопытен вид цилиндра – похоже, что произошедшее с ним преобразование связано с изменением системы координат с декартовой на сферическую.

7.2.11. Иерархия объектов дескрипторной графики

Чтобы понять, какие из объектов наследуют свойства других объектов, следует рассмотреть диаграмму иерархии объектов дескрипторной графики MATLAB, представленную на рис. 7.10 сверху. Этот рисунок показывает окно справки, посвященное свойствам объектов дескрипторной графики для справки системы MATLAB 6.5. В ней (ранее этого не было) объекты диаграммы выполнены в виде гиперссылок, так что активизация любого объекта открывает страницу справки по его свойствам.

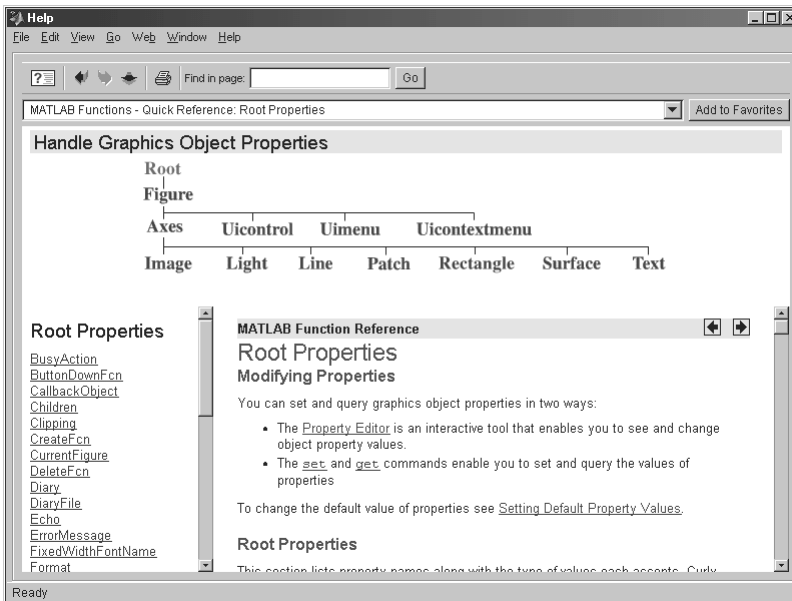


Рис. 7.10. Окно свойств дескрипторной графики MATLAB 6.5 с указанием иерархии объектов

Из приведенной диаграммы видно, что объекты `surface` расположены ниже объектов `axes`, а те, в свою очередь, расположены ниже объектов класса `figure`. Поэтому ясно, что в случае запуска файла `ms1` свойства сетки, построенной с применением объекта `axes`, будут унаследованы объектом `sh`, построенным командой `surface`.

Все объекты второго файла – `ms2` – относятся к классу `axes`. Именно поэтому они строятся поверх объектов, показанных на рис. 7.8. Координаты всех пяти трехмерных фигур (см. рис. 7.9) жестко заданы в соответствующих командах `axes`.

7.2.12. Справка по дескрипторной графике

Для детального знакомства с дескрипторной графикой можно воспользоваться справкой системы MATLAB – рис. 7.11.

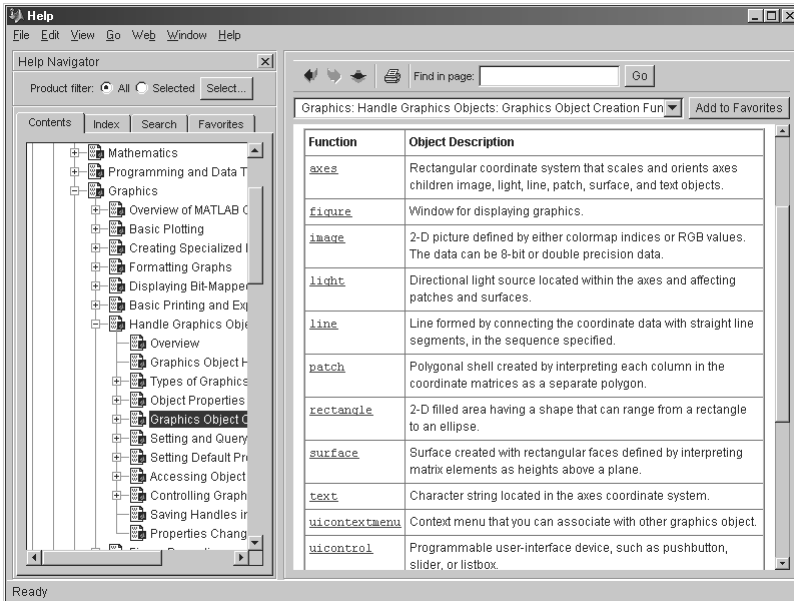


Рис. 7.11. Окно раздела справки MATLAB R2006b по дескрипторной графике

Здесь открыт раздел справки, посвященный основным объектам дескрипторной графики. Для перехода в тот или иной раздел достаточно активизировать гиперссылку соответствующего объекта. Например, на рис. 7.12 представлено полное окно справки по объекту `axis` (оси).

По каждому объекту можно найти синтаксическую форму записи команды или функции, а также относящиеся к ней опции. В справке можно найти много дополнительных примеров на применение дескрипторной графики.

7.3. Галерея трехмерной графики

7.3.1. Доступ к галерее

Для знакомства с возможностями трехмерной графики MATLAB имеет галерею (**Gallery**) в виде профессионально выполненных графических программ. Доступ к ним возможен как из режима демонстрации (команда **Examples and Demos**

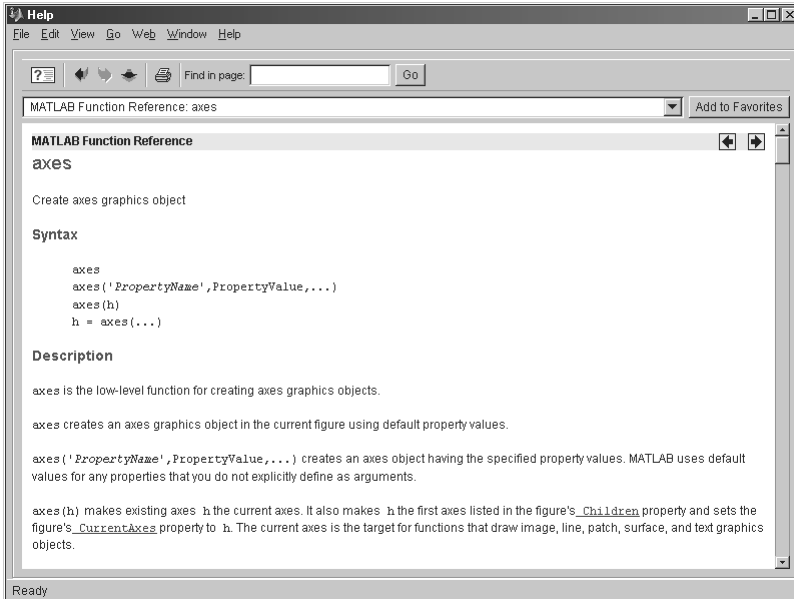


Рис. 7.12. Окно раздела справки MATLAB R2006b по объекту axis дескрипторной графики

в меню **Help** командного окна MATLAB), так и путем запуска команды из командной строки с указанием имени соответствующего файла.

Галерея представлена фигурами и файлами, список которых приведен в представленной ниже таблице.

Таблица 7.3. Данные о фигурах галереи трехмерной графики

| Имя фигуры | Файл | Наименование фигуры |
|------------|------------|--|
| knot | knot.m | Завязанный узел |
| quiver | quivdemo.m | Векторное объемное поле |
| klein 11 | klein 1.m | Объемное кольцо |
| cruller | cruller.m | Объемное кольцо Мебиуса |
| hoops | Tory4.m | Четыре объемных обруча |
| Slosh | spharm2.m | Построение фигуры, напоминающей улитку |
| modes | modes.m | Демонстрация фаз анимации трехмерной поверхности |
| logo | logo.m | Построение логотипа системы MATLAB |

Обратите внимание на то, что иногда имя файла не совпадает с именем фигуры в галерее.

7.3.2. Примеры построения фигур из галереи

Ниже приведено несколько примеров, которые дают наглядное представление о возможностях дескрипторной графики системы MATLAB.

Команда `quivdemo` выводит окно с демонстрацией построения пространственного векторного поля. Это окно показано на рис. 7.13.

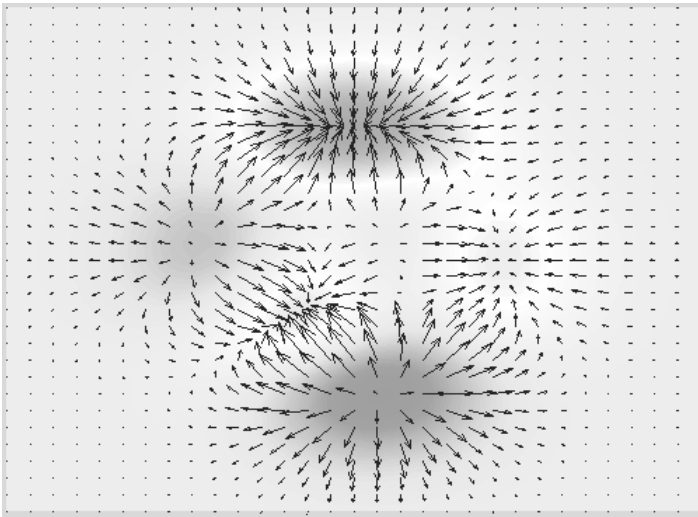


Рис. 7.13. График пространственного векторного поля

Полезно обратить внимание на то, что в этом примере сам по себе график – двумерный. Объемный вид поверхности достигается сочетанием функциональной окраски с изображением графика векторного поля с помощью стрелок.

Команда `klein1` строит график объемной ленты Мебиуса с одним перекручиванием. Вид этой фигуры показан на рис. 7.14. Этот график хорошо иллюстрирует хотя и одноцветную, но функциональную закраску фигуры с имитацией ее освещения источником света, расположенным сверху справа, и реализацией эффектов отражения света.

Команда `cruller` строит объемное кольцо Мебиуса с двойным перекручиванием. Построенная фигура показана на рис. 7.15. В данном случае используется обычная функциональная окраска с сохранением линий каркаса фигуры.

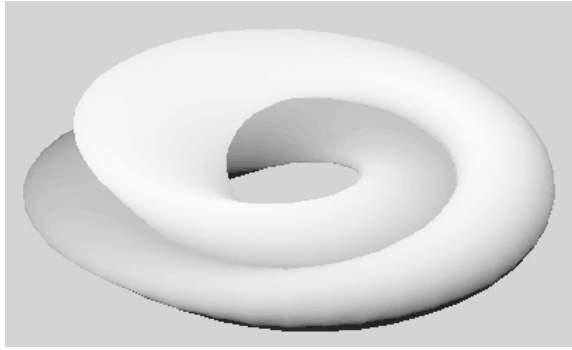


Рис. 7.14. Объемная линия Мебиуса с одним перекручиванием

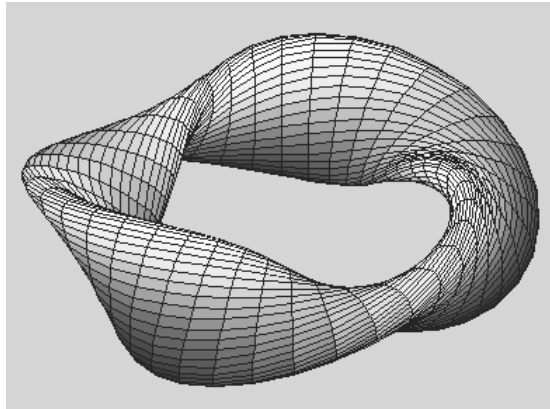


Рис. 7.15. Объемное кольцо Мебиуса

Команда `tory4` строит четыре переплетающихся друг с другом тора (объемных кольца) в пространстве (рис. 7.16). Наглядности этой картины также способствуют функциональная окраска торов и видимые линии каркаса. Обратите внимание, что невидимые линии удалены.

Любопытную фигуру, напоминающую раковину улитки, строит команда `spharm2`. Вид фигуры показан на рис. 7.17. Здесь интересно применение многоцветной функциональной окраски с использованием интерполяции по цвету, а также имитации эффектов отражения при освещении фигуры источником точечного света. Отчетливо видны зеркальные блики на поверхности фигуры.

Еще одна команда – `modes` – иллюстрирует построение фаз анимации поверхности (рис. 7.18). Она генерирует 12 фигур, отражающих положение поверхности в пространстве в различные моменты времени.

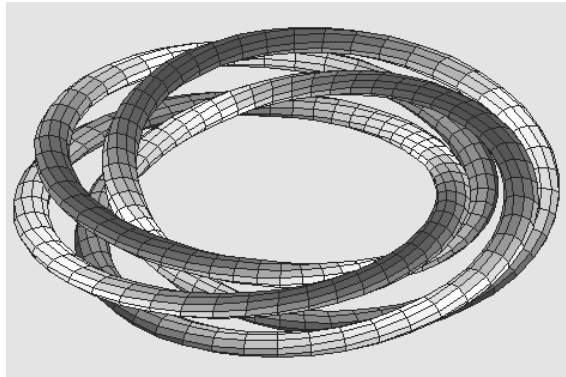


Рис. 7.16. Четыре тора в пространстве

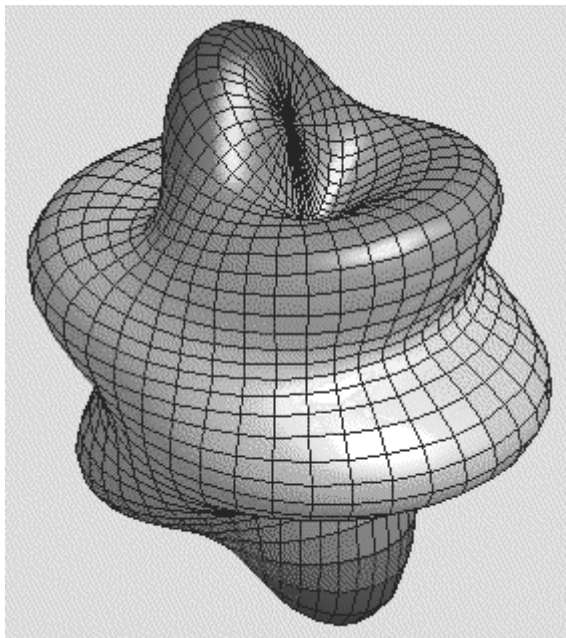


Рис. 7.17. Фигура, напоминающая улитку

В целом указанный набор программ дает хорошее представление о возможностях трехмерной графики системы MATLAB. Команда `type name`, где `name` – имя соответствующей команды, выводит полный листинг программы, реализующей построение той или иной фигуры. Знакомство со свойствами этих фигур позволяет оценить возможности средств управления световыми и цветовыми эффектами трехмерной графики и применить их в своих целях.

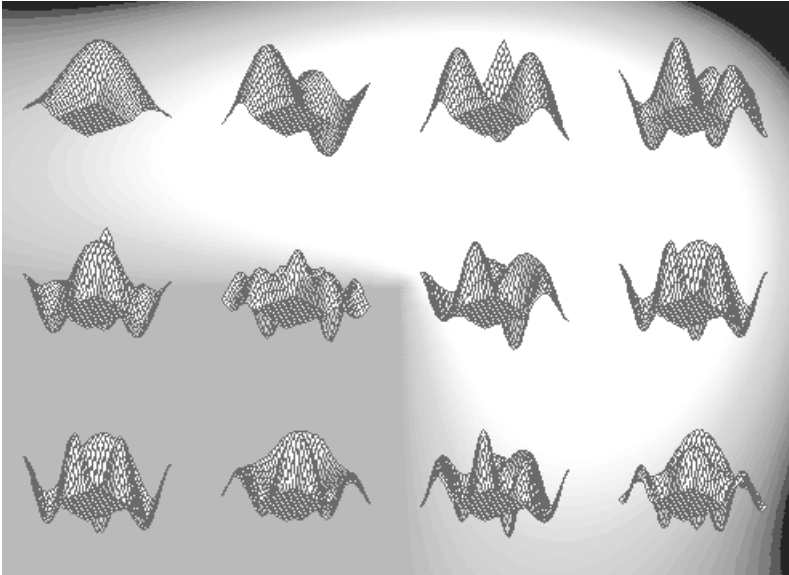


Рис. 7.18. Фазы анимации трехмерной поверхности

7.4. Графический интерфейс пользователя GUI

7.4.1. Основные команды для создания GUI

В MATLAB дескрипторная графика позволяет конструировать детали графического пользовательского интерфейса – GUI (Graphic User Interface). При этом различные функции и m-файлы вызываются из графического окна общего стандартного вида. Однако наполнение окна элементами интерфейса (кнопками, меню, слайдерами, надписями и т. д.) задается пользователем. Полный список команд и функций для проектирования пользовательского интерфейса из командной строки можно получить, выполнив команду `help uicontrols`. Позже (в уроке 12) мы рассмотрим визуально-ориентированное программирование GUI.

Ниже перечислены все команды и функции проектирования GUI в командном режиме работы. Это следующие функции:

- `uicontrol` – создание управляющего элемента;
- `uimenu` – создание пользовательского меню;
- `ginput` – графический ввод с помощью мыши.

Перечень команд и функций пользовательского интерфейса:

- `dragrect` – создание выделяющего прямоугольника с помощью мыши;
- `rbbox` – растягивание прямоугольника мышью;

- `selectmoveresize` – интерактивное выделение, перемещение и копирование объектов с помощью мыши;
- `waitforbuttonpress` – ожидание нажатия клавиши клавиатуры или кнопки мыши в окне;
- `waitfor` – прекращение выполнения программы в ожидании уничтожения заданного графического объекта или изменения его свойств;
- `uiwait` – прекращение выполнения программы в ожидании вызова функции `uiresume` или закрытия заданного графического окна;
- `uiresume` – возобновление выполнения после блокировки;
- `uisuspend` – прекращение интерактивного состояния фигуры;
- `uirestore` – возобновление интерактивного состояния фигуры.

Средства проектирования пользовательского интерфейса:

- `guide` – создание GUI;
- `align` – выравнивание положения объектов интерфейса;
- `cbedit` – изменение повторного вызова объектов;
- `menuedit` – изменение меню;
- `propedit` – изменение свойств объектов.

Средства создания диалоговых окон:

- `dialog` – создание диалогового окна;
- `axlimdlg` – ограничение размеров диалогового окна;
- `errordlg` – создание окна с сообщением об ошибке;
- `helpdlg` – создание справочного окна;
- `inputdlg` – создание окна диалога ввода;
- `listdlg` – создание окна диалога для выбора вариантов параметра из списка;
- `menu` – создание меню диалогового ввода;
- `msgbox` – создание окна сообщений;
- `questdlg` – создание окна запроса;
- `warndlg` – создание окна предупреждения;
- `uigetfile` – создание стандартного окна открытия файлов;
- `uiputfile` – создание стандартного окна записи файлов;
- `uisetcolor` – создание окна выбора цвета;
- `uisetfont` – создание окна выбора шрифта;
- `pagedlg` – создание диалогового окна параметров страницы;
- `printdlg` – создание диалогового окна печати;
- `waitbar` – создание окна с индикатором прогресса.

Создание меню:

- `makemenu` – создание структуры меню;
- `menubar` – установление типовых свойств для объекта `MenuBar`;
- `umtoggle` – изменение статуса параметра «checked» для объекта `uimenu`;
- `winmenu` – создание подменю для меню **Window**.

Создание кнопок панели инструментов и управление ими:

- `btngroup` – создание кнопки панели инструментов;

- `btnstate` – запрос статуса кнопки;
- `btnpress` – управление кнопкой;
- `btndown` – нажатие кнопки;
- `btnup` – отпускание кнопки.

Утилиты задания свойств объектов `figure/axes`:

- `clruprop` – удалить свойство объекта;
- `getuprop` – запросить свойство объекта;
- `setuprop` – установить свойство объекта.

Вспомогательные утилиты:

- `allchild` – запросить все порожденные объекты;
- `findall` – найти все объекты;
- `hidegui` – скрыть/открыть объекты GUI;
- `edtext` – интерактивное редактирование объектов `text`;
- `getstatus` – запросить свойства строки объекта `figure`;
- `setstatus` – установить свойства строки объекта `figure`;
- `popupstr` – запросить свойства строки выпадающего меню;
- `remapfig` – изменить положение объекта `figure`;
- `setptr` – установить указатель на объект `figure`;
- `getptr` – получить указатель на объект `figure`;
- `overobj` – запросить дескриптор объекта, над которым находится курсор мыши.

Таким образом, MATLAB содержит обширный набор команд и функций для создания типовых элементов пользовательского интерфейса.

7.4.2. Простой пример создания объектов GUI

Ниже представлена программа (распечатка `m`-файла с именем `ui`), которая при запуске создает 4 объекта интерфейса:

```
k1=uicontrol('Style','pushbutton',...
    'Units','normalized','Position',[.7 .5 .2 .1],...
    'String','click here');
k2=uicontrol('Style','pushbutton',...
    'Units','normalized','Position',[.6 .3 .2 .1],...
    'String','click here');
ck = uicontrol('Style', 'pushbutton', 'String', 'Clear',...
    'Position', [150 150 100 70], 'Callback', 'cla');
hprop = uicontrol('Style', 'popup',...
    'String', 'hsv|hot|cool|gray',...
    'Position', [30 320 100 50],...
    'Callback', 'setmap');
```

Первые два объекта `k1` и `k2` – это малые кнопки с надписью **click here** («щелкни здесь»). Объект `ck` – это большая кнопка `Clear` (кстати, действующая). Объект `hprop` – выпадающий список (тоже действующий, хотя и содержащий поименован-

ные позиции – пустышки). Для создания всех этих объектов используется команда `uicontrol` с соответствующими параметрами, задающими стиль (вид) объекта интерфейса, место его размещения и надпись (на кнопках). На рис. 7.19 построены все эти объекты, причем список показан в открытом состоянии.

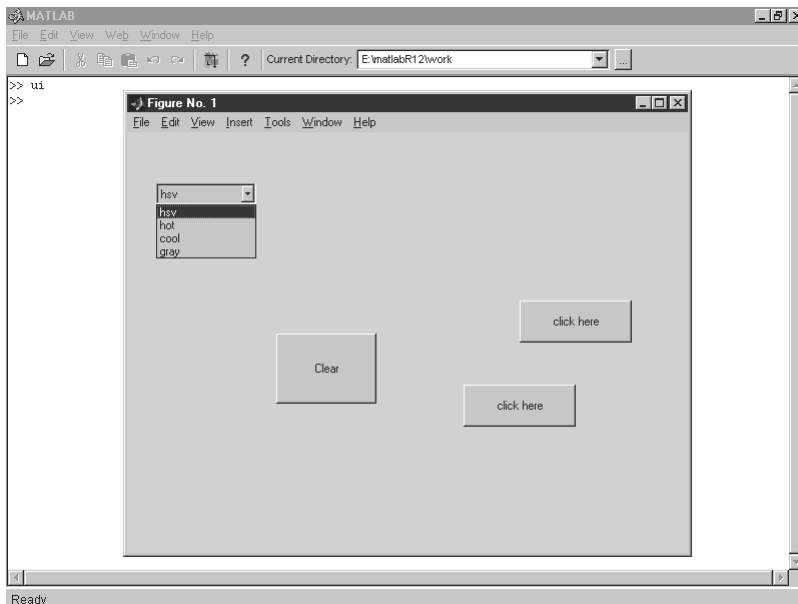


Рис. 7.19. Пример построения объектов пользовательского интерфейса

Дескрипторная графика MATLAB позволяет создавать любые детали современного пользовательского интерфейса. Более подробно с функциями создания и модификации пользовательского интерфейса и программированием задач с использованием GUI можно познакомиться в уроке 10.

7.4.3. Примеры программирования GUI

В директории TOOLBOX/MATLAB/DEMOS можно найти множество как простых, так и сложных примеров программирования GUI для решения разнообразных задач. Доступ к этим примерам довольно прост – надо в справке выйти в раздел демонстрационных примеров **Demos** и подобрать нужный пример.

Мы остановимся на паре характерных примеров, наглядно демонстрирующих возможности программирования задач, для визуализации которых используются элементы графического пользовательского интерфейса GUI. Доступ к одному из таких примеров представлен на рис. 7.20.

В правой части окна справки видно описание примера. В верхней строке его имеются имя файла с полным путем к нему из директории MATLAB и команда

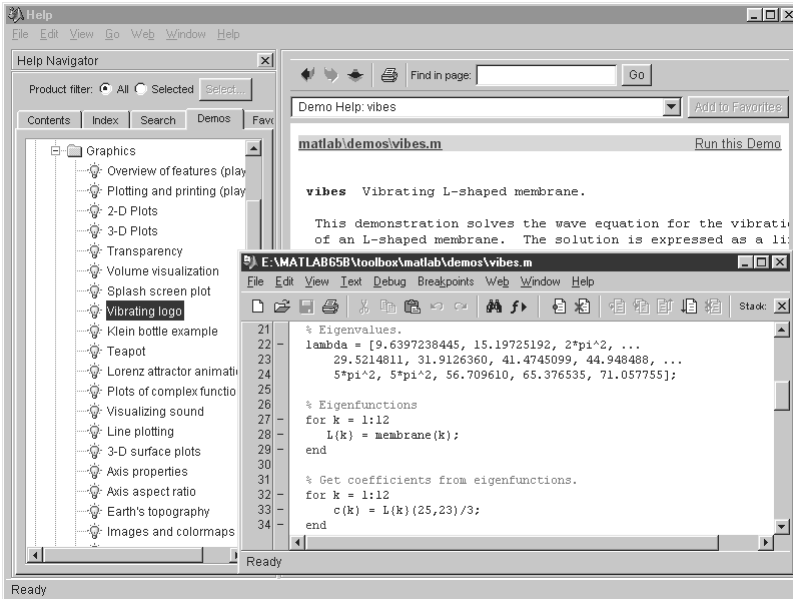


Рис. 7.20. Доступ к одному из примеров

пуска примера. Активизируя гиперссылку файла, можно открыть окно редактора и отладчика m-файлов и увидеть листинг выбранного файла в этом окне. Оно также показано на рис. 7.20.

7.4.4. Программирование анимации поверхности с разной скоростью

Файл `vibes.m` создает графическое окно GUI, в котором можно наблюдать анимацию некоторой поверхности. В этом окне строятся три кнопки **faster** (быстрее), **slower** (медленнее) и **done** (дальше) и видна меняющаяся во времени поверхность. Листинг этого m-файла с краткими комментариями представлен ниже:

```
function vibes
% Eigenvalues
lambda = [9.6397238445, 15.19725192, 2*pi^2, ...
          29.5214811, 31.9126360, 41.4745099, 44.948488, ...
          5*pi^2, 5*pi^2, 56.709610, 65.376535, 71.057755];
% Eigenfunctions
for k = 1:12
    L{k} = membrane(k);
end
% Get coefficients from eigenfunctions.
for k = 1:12
    c(k) = L{k}(25,23)/3;
```

```

end
% Set graphics parameters.
fig = figure; set(fig, 'color', 'k')
x = (-15:15)/15; h = surf(x,x,L{1});
[a,e] = view; view(a+270,e);
axis([-1 1 -1 1 -1 1]); caxis(26.9*[-1.5 1]);
colormap(hot); axis off
% Buttons
uicontrol('pos',[20 20 60 20], 'string', 'done', 'fontsize', 12, ...
    'callback', 'close(gcbf)');
uicontrol('pos',[20 40 60 20], 'string', 'slower', 'fontsize', 12, ...
    'callback', 'set(gcbf, 'userdata', sqrt(0.5)*...
    get(gcbf, 'userdata'))');
uicontrol('pos',[20 60 60 20], 'string', 'faster', 'fontsize', 12, ...
    'callback', 'set(gcbf, 'userdata', sqrt(2.0)*get(gcbf, 'userdata'))');
% Run
t = 0; dt = 0.025; set(fig, 'userdata', dt)
while ishandle(fig)
    % Coefficients
    dt = get(fig, 'userdata');    t = t + dt;    s =
c.*sin(sqrt(lambda)*t);
    % Amplitude
    A = zeros(size(L{1}));
    for k = 1:12
        A = A + s(k)*L{k};
    end
    % Velocity
    s = lambda .*s;    V = zeros(size(L{1}));
    for k = 1:12
        V = V + s(k)*L{k};
    end
    V(16:31, 1:15) = NaN;
    % Surface plot of height, colored by velocity.
    set(h, 'zdata', A, 'cdata', V);    drawnow
end;

```

Читателю несложно разобраться с этим простым примером. В случае затруднений рекомендуется познакомиться с материалом урока 11 и затем вернуться к разбору данной программы. Вид окна GUI для этого примера представлен на рис. 7.21.

Все три кнопки данного примера – действующие.

7.4.5. Программирование визуализации звукового сигнала

Теперь рассмотрим еще один пример – на визуализацию звукового сигнала (файл `xrsound.m`). В этом примере используется целый ряд элементов GUI, включая кнопки, раскрывающиеся списки и слайдер. Окно этого примера показано на рис. 7.22. Для детального знакомства с этим примером изучите файл `xrsound.m`.

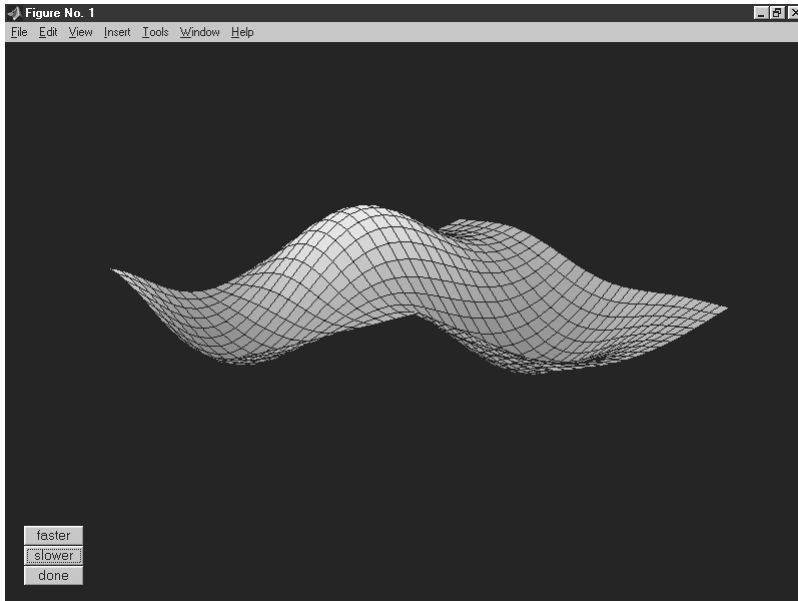


Рис. 7.21. Окно GUI примера на анимацию поверхности

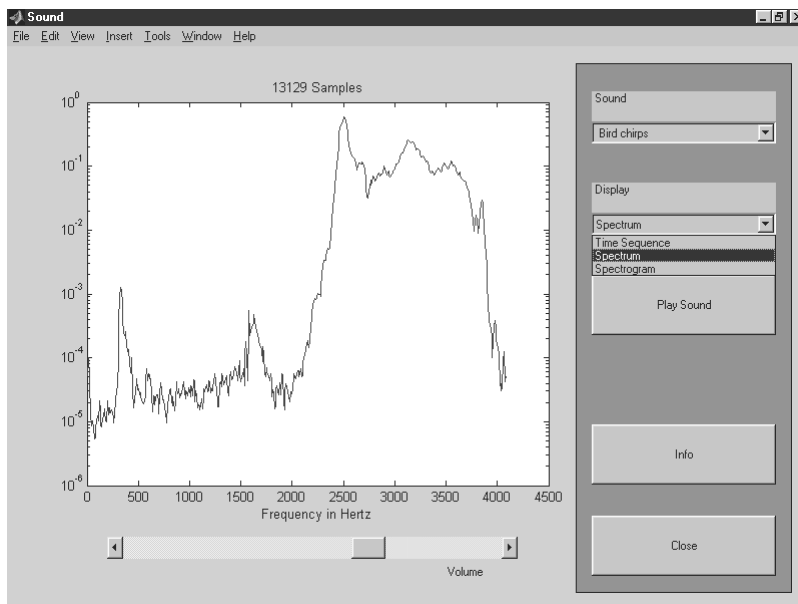


Рис. 7.22. Окно GUI примера на анимацию поверхности

Стоит еще раз обратить внимание читателей на то, что в MATLAB имеется огромное число примеров на решение самых разнообразных задач с применением техники GUI. Поэтому у достаточно настойчивого читателя опыт программирования таких задач без сомнения будет успешным.

7.5. Графическая поддержка цвета

7.5.1. Цветовые системы и OpenGL

Дескрипторная графика MATLAB обеспечивает поддержку некоторых наиболее распространенных цветовых систем: RGB (Red, Green, Blue) и HSV (hue – цветовой тон, saturation – насыщенность и value – яркость).

На уровне ядра графических операций в системах MATLAB поддерживаются довольно очевидные функции преобразования цветовых моделей:

- `RGB=hsv2rgb` (HSV) преобразует матрицу изображения HSV в матрицу изображения RGB;
- `HSV=rgb2hsv` (RGB) преобразует матрицу изображения RGB в матрицу изображения HSV.

Работа этих функций наглядна лишь при цветной графике. Поскольку иллюстрации в книге черно-белые, мы ограничимся лишь упоминанием о данных функциях преобразования.

Одной из новинок новых версий систем MATLAB является поддержка графических средств OpenGL. Эти средства обычно используются чаще всего при реализации быстрой трехмерной графики, например при осуществлении сложной функциональной окраски поверхностей и трехмерных фигур с учетом характера освещения и структуры материала (рейдеринг), при осуществлении анимации для таких объектов, при построении поверхностей из многоугольников, осуществлении эффектов прозрачности и т. д.

Средства OpenGL в MATLAB задействованы автоматически. Это значит, что они будут использованы, если видеокарта компьютера пользователя поддерживает их и если установлены соответствующие драйверы видеоадаптера. На уровне средств стандартной графики MATLAB никаких функций управления OpenGL нет. Однако дескрипторная графика такую возможность предоставляет с помощью команды

```
opengl selection_mode
```

Эта команда задает графические режимы осуществления рендеринга. Параметр `selection_mode` может принимать следующие значения:

- `autoselect` задает автоматическое применение OpenGL и вводит в работу средства OpenGL при наличии возможностей для этого;
- `neverselect` отключает автоматическое применение OpenGL;
- `advise` выводит сообщение о возможности применения OpenGL, но режим рендеринга (RenderMode) устанавливается вручную.

Команда `opengl info` выводит данные о средствах OpenGL ПК, на котором установлена система MATLAB, например (для MATLAB 7 SP2):

```
>> opengl info
Version          = 1.4.0
Vendor           = NVIDIA Corporation
Renderer        = GeForce FX 5200/AGP/SSE2
MaxTextureSize  = 4096
Visual          = 05 (RGB 32 bits(08 08 08 00) zdepth 24, Hardware
Accelerated, OpenGL, Double Buffered, Window)
Software        = false
# of Extensions = 90
Driver Bug Workarounds:
OpenGLBitmapZbufferBug    = 0
OpenGLWobbleTesselatorBug = 0
OpenGLLineSmoothingBug   = 0
OpenGLDockingBug         = 0
OpenGLClippedImageBug    = 1
OpenGLEraseModeBug       = 0
```

Возможно также управление средствами рендеринга и OpenGL на уровне средств дескрипторной графики с помощью команды `set`, например:

```
set(gcf, 'Renderer', 'OpenGL')
```

7.5.2. Управление прозрачностью графических объектов

Пожалуй, наиболее впечатляющие и внешне заметные результаты дает применение свойства прозрачности изображений (`transparency`), доступное при установке средств Open GL. Прозрачность задается свойством `alphadata`, которое представляет собой матрицу размера $m \times n$ типа `double` или `uint8` и определяет прозрачность каждого элемента. По умолчанию это значение 1, которое задает непрозрачное изображение элемента. Свойство `alphadatamapping` представляет собой метод представления карты прозрачности (возможные значения – `none`, `direct`, `scaled`, по умолчанию `none`).

Свойство прозрачности основано на представлении изображений в виде отдельных слоев, что обычно требует применения многомерных массивов. Данные о прозрачности размещаются в матрице размера $m \times n$ `AlphaData`, элементы которой должны иметь тип `double` или `uint8` (элементы типа `NaN` недопустимы). Возможности задания прозрачности поддерживаются графическими файлами с расширением PNG. В изображениях, хранящихся в этих файлах, возможна поддержка кодирования цветов с разным разрешением – вплоть до 48 бит при RGB-графике.

7.5.3. Примеры построения изображений со свойствами прозрачности

Приведем наглядный пример использования свойств прозрачности из описания графики системы MATLAB:

```
% Программа построения графика flow (течение)
[x y z v] = flow;
p=patch(isosurface(x,y,z,v,-3));
isonormals(x,y,z,v,p);
set(p,'facecolor','red','edgecolor','none');
daspect([1 1 1]);
view(3); axis tight; grid on;
camlight; lighting gouraud;
```

Здесь для построения графика дана трехмерная фигура flow (течение). Она представлена тремя массивами своих точек x , y и z и дополнительным массивом класса AlphaData – v . При первом построении свойство прозрачности отсутствует (по умолчанию) и построенная фигура будет иметь вид, представленный на рис. 7.23.

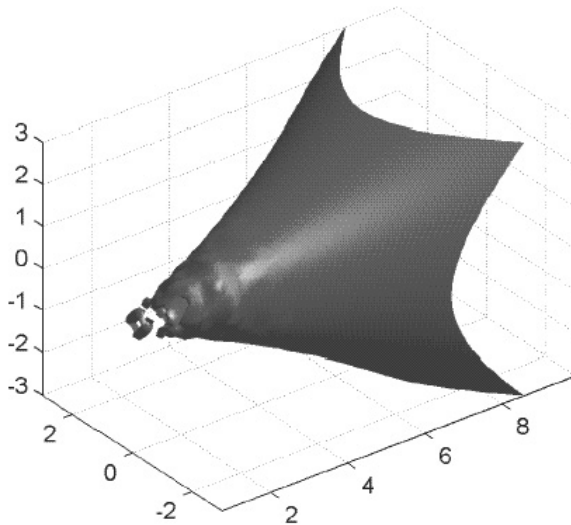


Рис. 7.23. Трехмерная фигура в обычном представлении (без свойства прозрачности)

Если исполнить команду `alpha(0.5)`, то в массиве AlphaData будут заданы элементы, обеспечивающие степень прозрачности 0.5. При этом изображение объекта будет иметь вид, представленный на рис. 7.24. Теперь на нем четко видна скрытая ранее твердая сердцевина фигуры и даже проглядывают координатные оси.

Более подробные сведения об использовании свойства прозрачности можно найти в обширной документации по графике в формате PDF. Эта документация в виде файла `graphg.pdf` с объемом свыше 12 Мб поставляется с системой MATLAB 6.

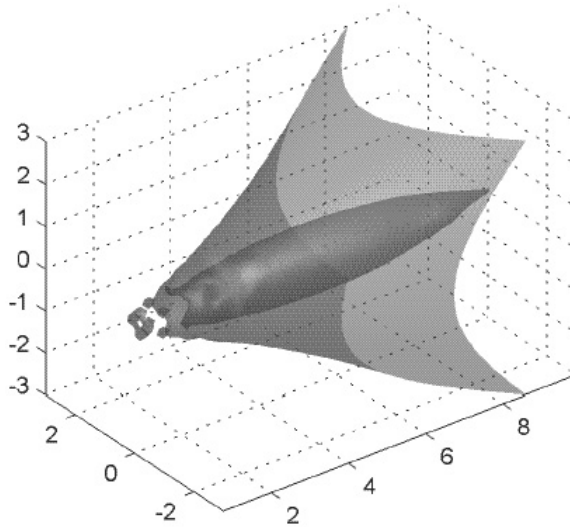


Рис. 7.24. Трехмерная фигура с установкой свойства прозрачности

7.6. Расширенная техника визуализации вычислений

Средства дескрипторной графики позволяют реализовать весьма эффектную визуализацию сложных объектов (прежде всего трехмерных) и многих физических явлений, например таких, как струи газа и жидкости, электрические разряды и т. д. Ниже представлены краткие сведения о подобных возможностях.

7.6.1. Задание *Path*-объектов

Объект *Patch* может использоваться для построения закрасенных многоугольников (полигонов). Для этого служит специальная высокоуровневая команда

```
Patch(xc, yc, [zc, ]colordata)
```

В ней *xc*, *yc* и *zc* – массивы координат. Она может быть также задана на низком уровне, например как

```
patch('XData', sin(t), 'YData', cos(t))
```

Следующая программа строит закрасенный десятиугольник с интерполяцией закраски, дающей плавные переходы цвета (рис. 7.25):

```
% Программа построения закрасенного 10-угольника
t = 0:pi/5:2*pi;
a = t(1:length(t)-1);
patch(sin(a), cos(a), 1:length(a), 'FaceColor', 'interp')
```

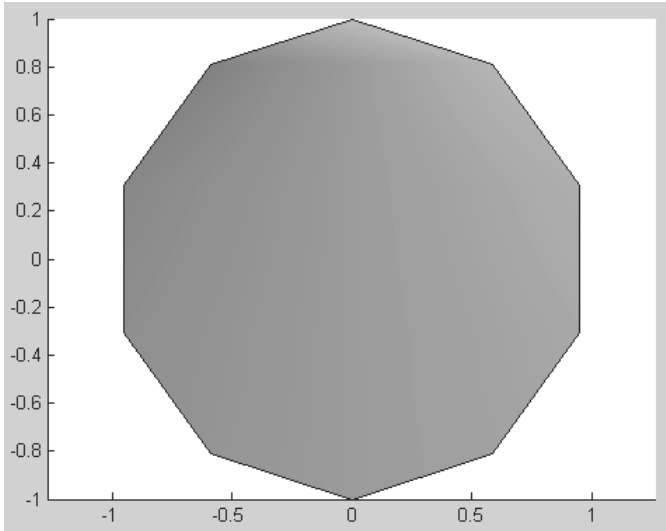


Рис. 7.25. Закрашенный многоугольник, построенный командой `patch`

```
colormap cool;  
axis equal
```

7.6.2. Построение среза черепной коробки человека

В MATLAB с помощью массивов могут быть заданы достаточно сложные объекты. Например, файл `mri` содержит массив графических данных черепной коробки человека вместе с ее «внутренностями». Название файла происходит от сокращения слов *Magnetic Resonance Imaging*. Таким образом, этот файл содержит пример задания изображения от медицинских приборов на основе применения магнитного резонанса – томографов.

Приведенный ниже пример показывает загрузку этого массива и выделение одного слоя из разрезанной горизонтальными плоскостями черепной коробки (рис. 7.26):

```
% Программа построения среза черепной коробки человека  
load mri;  
D = squeeze(D);  
image_num = 8;  
image(D(:,:,image_num))  
axis image;  
colormap(map)
```

В другом примере строятся уже несколько срезов черепной коробки (рис. 7.27):

```
% Программа построения нескольких срезов
```

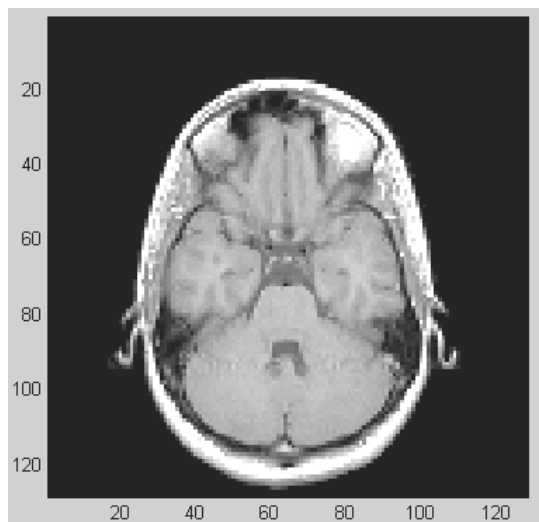


Рис. 7.26. Срез черепной коробки человека

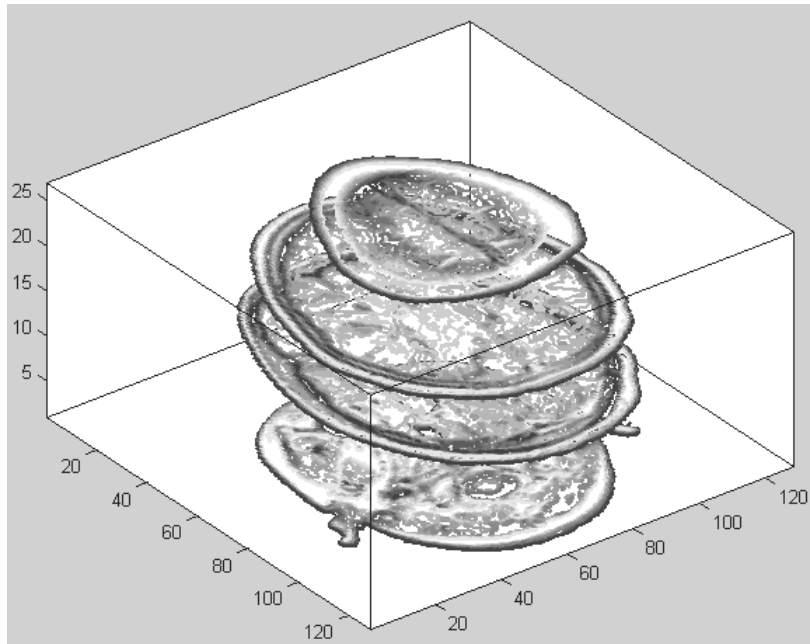


Рис. 7.27. Несколько срезов черепной коробки человека

```
% черепной коробки человека
phandles = contourslice(D, [], [], [1,12,19,27],8);
view(3);
axis tight;
daspect([1,1,.4])
set(phandles, 'LineWidth',2)
```

С деталями иного применения использованных в этих простых программных фрагментах функций можно ознакомиться по справке MATLAB, посвященной дескрипторной графике.

7.6.3. Расширенная визуализация трехмерных объектов

Рассмотренные выше приемы можно распространить на существенно расширенную визуализацию трехмерных объектов. Приведенный ниже m-файл обеспечивает сглаживание данных массива черепной коробки командой `smooth3` и построение реалистичного рисунка черепной коробки с удаленным сверху ее фрагментом:

```
% Программа построения реалистического изображения
% среза черепной коробки человека
Ds = smooth3(D);
hiso = patch(isosurface(Ds,5), 'FaceColor', [1,.75,.65],...
    'EdgeColor', 'none');
hcap = patch(isocaps(D,5), 'FaceColor', 'interp',...
    'EdgeColor', 'none');
colormap(map); view(45,30); axis tight; daspect([1,1,.4])
lightangle(45,30); set(gcf, 'Renderer', 'zbuffer');
lighting phong; isonormals(Ds,hiso);
set(hcap, 'AmbientStrength', .6)
set(hiso, 'SpecularColorReflectance', 0,...
    'SpecularExponent', 50)
```

Полученное после пуска этой программы изображение представлено на рис. 7.28.

Этот пример наглядно показывает, что средства дескрипторной графики MATLAB могут эффективно применяться в медицине и в уроках по медицине для изучения внутреннего строения органов человека.

7.6.4. Выделение части объема

С помощью функций `subvolume` и `isonormal` можно выделить часть объема и наглядно ее представить. Следующий простой фрагмент программы делает это для массива файла, представляющего черепную коробку человека (рис. 7.29):

```
% Программа построения части черепной коробки человека
load mri; D = squeeze(D);
[x,y,z,D] = subvolume(D, [60,80,nan,80,nan,nan]);
```

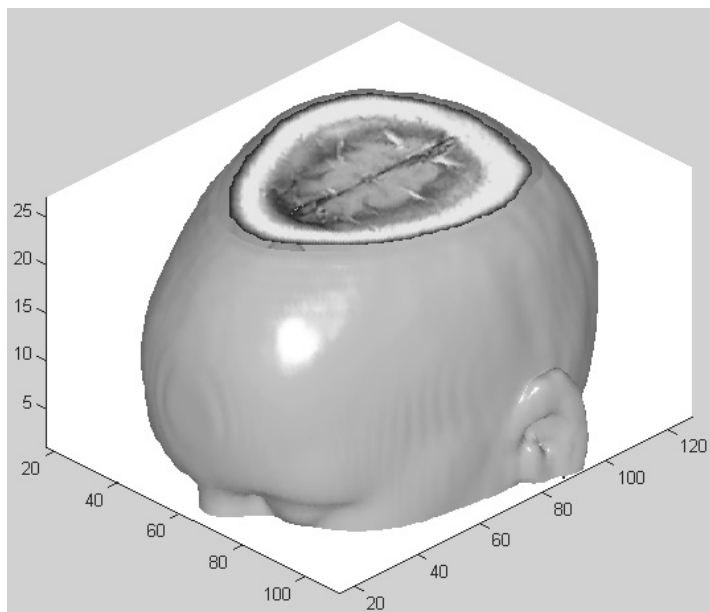


Рис. 7.28. Черепная коробка человека со срезом

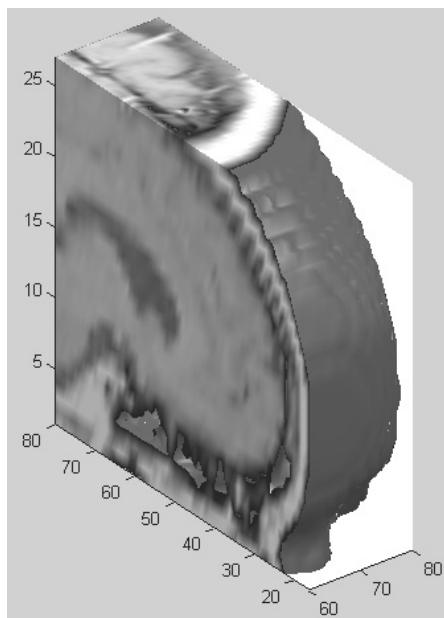


Рис. 7.29. Объемный вырез черепной коробки человека со срезами с разных сторон

```

p1 = patch(isosurface(x,y,z,D, 5),...
'FaceColor','red','EdgeColor','none');
isonormals(x,y,z,D,p1);
p2 = patch(isocaps(x,y,z,D, 5),...
'FaceColor','interp','EdgeColor','none');
view(3); axis tight; daspect([1,1,.4]); colormap(gray(100))
camlight right; camlight left; lighting gouraud

```

7.6.5. Визуализация струи в пространстве

Дескрипторная графика обеспечивает эффективную визуализацию в пространстве струй жидкостей или газов. Такие задачи сплошь и рядом встречаются в гидродинамике и аэродинамике. Наглядный пример этого дает функция `flow`, создающая массивы представления струи:

```
[x,y,z,v] = flow;
```

Выполним нормализацию масштабов по осям координат и вспомогательные операции к подготовке представления струи:

```

% Программа построения струи (часть 1)
xmin = min(x(:)); ymin = min(y(:)); zmin = min(z(:));
xmax = max(x(:)); ymax = max(y(:)); zmax = max(z(:));
hslice = surf(linspace(xmin,xmax,100),...
             linspace(ymin,ymax,100), zeros(100));
rotate(hslice, [-1,0,0], -45);
xd = get(hslice, 'XData'); yd = get(hslice, 'YData');
zd = get(hslice, 'ZData'); delete(hslice)

```

Обратите внимание на то, что объемный массив `hslice` в этом примере имеет вспомогательную роль. После получения из него массивов `xd`, `yd` и `zd` проекций массив `hslice` удаляется во избежание перегрузки памяти компьютера.

Теперь построим проекции струи на плоскости координатного ящика и диагональную плоскость:

```

% Программа построения струи (часть 2)
h = slice(x,y,z,v,xd,yd,zd);
set(h,'FaceColor','interp','EdgeColor','none','DiffuseStrength',.8)
hold on; hx = slice(x,y,z,v,xmax,[],[]);
set(hx,'FaceColor','interp','EdgeColor','none')
hy = slice(x,y,z,v,[],ymax,[]);
set(hy,'FaceColor','interp','EdgeColor','none')
hz = slice(x,y,z,v,[],[],zmin);
set(hz,'FaceColor','interp','EdgeColor','none')
daspect([1,1,1]); axis tight; box on
view(-38.5,16); camzoom(1.4); camproj perspective
Добавим эффекты светового выделения и вывода шкалы цветов:
% Программа построения струи (часть 3)
lightangle(-45,45); colormap(jet(24));
set(gcf,'Renderer','zbuffer'); colormap(flipud(jet(24)));
caxis([-5,37.4832]); colorbar('horiz')

```

Собрав воедино представленные фрагменты m-файла и пустив его на исполнение, получим рис. 7.30, представляющий собой наглядную картину строения струи.

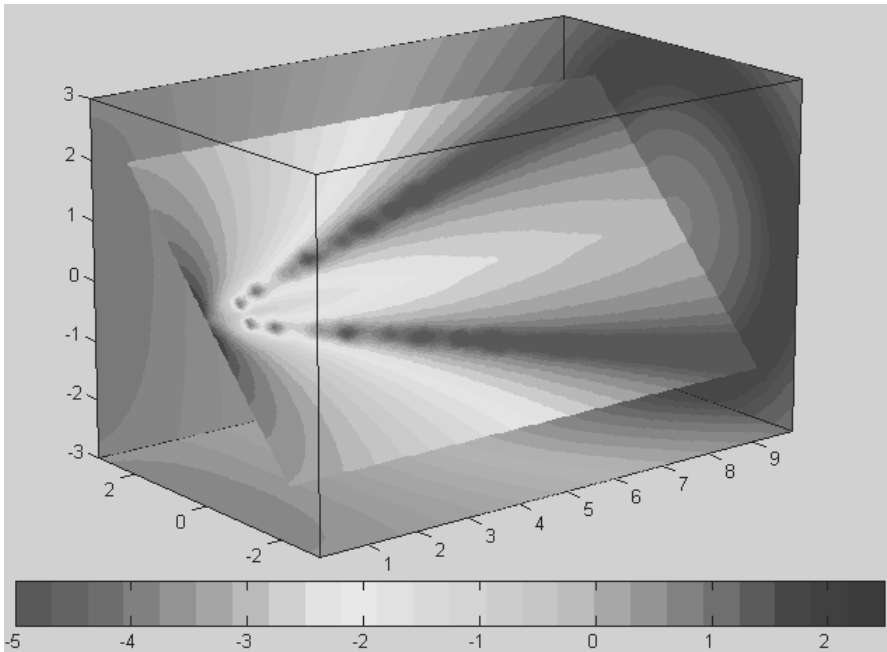


Рис. 7.30. Визуальное представление струи в пространстве

Образ струи в пространстве имеет высокую наглядность (особенно в оригинале – в виде цветного изображения на экране дисплея) и прекрасно выделяет различные особенности струи, например ее турбулентность. Однако надо помнить, что представленная программа вовсе не моделирует струю, а лишь отображает готовый результат моделирования, представленный в тестовой функции flow в некоторый момент времени.

7.6.6. Визуализация электрических разрядов

Файл wind хранит массив, представляющий развитие электрических разрядов в атмосфере. Приведенный ниже пример иллюстрирует технику визуализации этих явлений (рис. 7.31):

```
% Программа визуализации электрических разрядов
load wind; xmin = min(x(:));
xmax = max(x(:)); ymax = max(y(:)); zmin = min(z(:));
wind_speed = sqrt(u.^2 + v.^2 + w.^2);
```

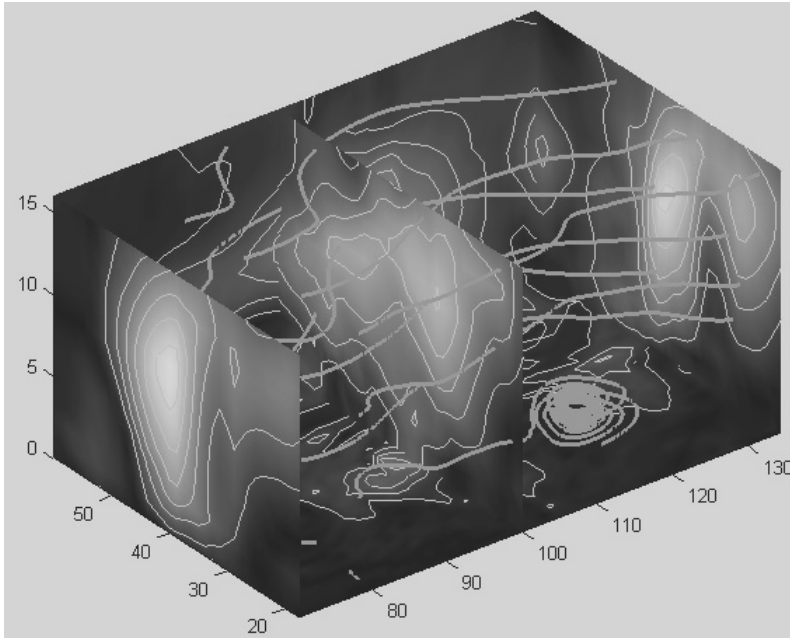


Рис. 7.31. Визуальное представление электрических разрядов

```

hsurfaces = slice(x,y,z,wind_speed,[xmin,100,xmax],ymax,zmin);
set(hsurfaces,'FaceColor','interp','EdgeColor','none');
hcont = contourslice(x,y,z,wind_speed,...
    [xmin,100,xmax],ymax,zmin);
set(hcont,'EdgeColor',[.7,.7,.7],'LineWidth',.5);
[sx,sy,sz] = meshgrid(80,20:10:50,0:5:15);
hlines = streamline(x,y,z,u,v,w,sx,sy,sz);
set(hlines,'LineWidth',2,'Color','r');
view(3); daspect([2,2,1]); axis tight

```

Приведенная выше программа критична к применяемой видеокарты версии OpenGL. Если при ее выполнении появляется сообщение о некорректной версии OpenGL, то цветовая окраска будет отличаться от оригинальной и могут пропасть эффекты прозрачности. Тем не менее и в этом случае картина разрядов будет выглядеть достаточно эффектно и реалистично.

7.6.7. Анимация явления подъема предметов вихрями

Жители Северной Америки нередко наблюдают сложные атмосферные явления, такие как вихри, торнадо и смерчи. Они нередко поднимают не только тучи пыли, но и различные предметы – от камней до автомобилей и домов. Уже упомянутый

файл `wind` хранит данные не только о развитии атмосферных разрядов электричества, но и о сопровождающих их атмосферных вихрях (типа смерча). Представленный ниже фрагмент программы дает визуализацию этих эффектов:

```
% Программа визуализации атмосферных вихрей (вариант 1)
load wind; [sx sy sz] = meshgrid(100,20:2:50,5);
verts = stream3(x,y,z,u,v,w,sx,sy,sz);
sl = streamline(verts);
view(-10.5,18); daspect([2 2 0.125]); axis tight; box on;
iverts = interpstreamspeed(x,y,z,u,v,w,verts,0.05);
set(gca,'drawmode','fast'); streamparticles(iverts,15,...
    'Animate',10, 'ParticleAlignment','on',...
    'MarkerEdgeColor','none', 'MarkerFaceColor','red', 'Marker','o');
```

В течение нескольких секунд можно наблюдать весьма поучительную анимационную картину подъема «камней», представленных кружками, – рис. 7.32.

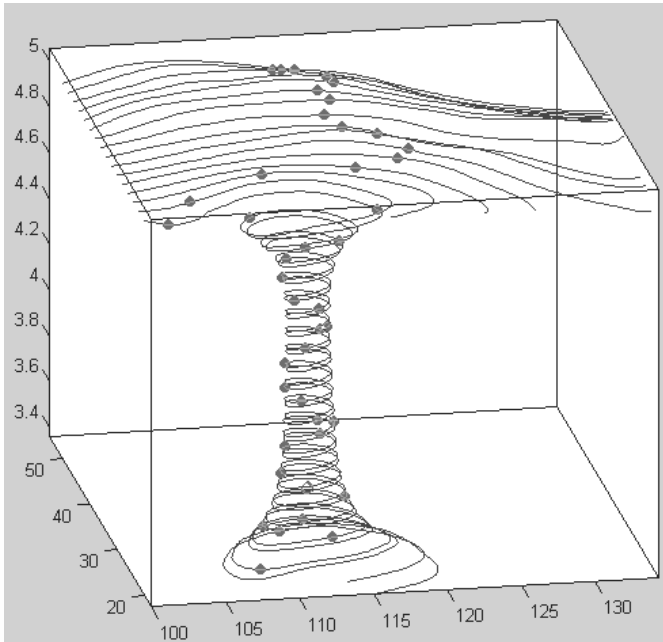


Рис. 7.32. Анимация эффекта подъема камней при развитии смерча

Еще один пример на анимацию подъема предметов вихрем (рис. 7.33) представлен следующим фрагментом программы:

```
% Программа визуализации атмосферных вихрей (вариант 2)
load wind; [sx sy sz] = meshgrid(80,20:1:55,5);
verts = stream3(x,y,z,u,v,w,sx,sy,sz);
sl = streamline(verts);
```

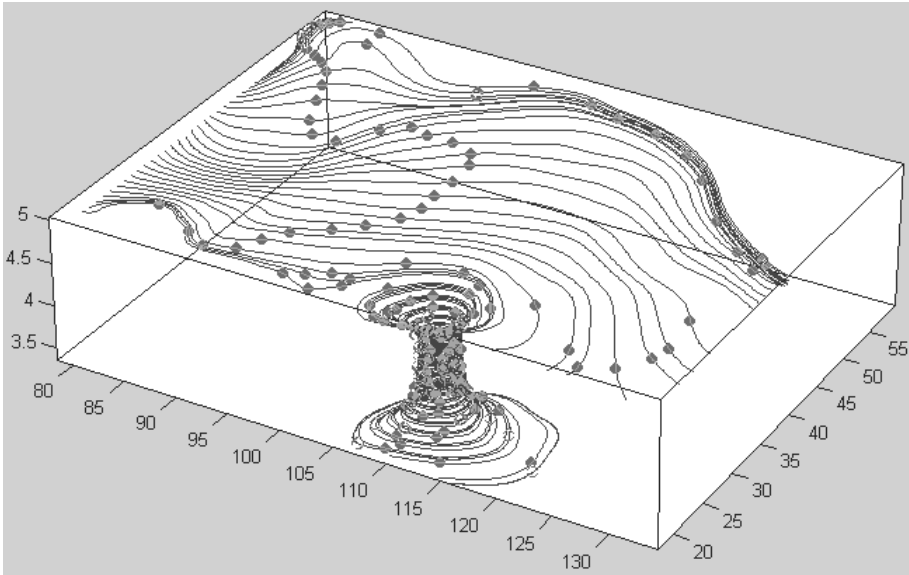


Рис. 7.33. Анимация эффекта подъема камней при развитии смерча в ограниченном пространстве

```
iverts = interpstreamspeed(x,y,z,u,v,w,verts,.025);
axis tight; view(30,30); daspect([1 1 .125]);
camproj perspective; camva(8); set(gca,'DrawMode','fast')
box on; streamparticles(iverts,35,'animate',10,...
'ParticleAlignment','on')
```

Вновь обращаем внимание читателей, что эти примеры лишь иллюстрируют (на этот раз в развитии) результаты моделирования. Само моделирование таких сложных явлений достаточно сложно и вряд ли разумно при использовании ПК.

7.6.8. Применение «конусной» графики для визуализации струй

Дети по наитию исследуют течение ручейков воды, пуская по ним бумажные кораблики. Вышедшие из детского возраста физики тоже используют подобный подход – только вместо корабликов они запускают в струи воздуха маленькие конусы – их движение позволяет исследовать движение воздушных (газовых, жидкостных) струй. Приведенный ниже фрагмент программы дает представление о применении этой техники визуализации (рис. 7.34):

```
% Программа визуализации струи с помощью конусов
load wind; wind_speed = sqrt(u.^2 + v.^2 + w.^2);
hiso = patch(isosurface(x,y,z,wind_speed,40));
isonormals(x,y,z,wind_speed,hiso);
```

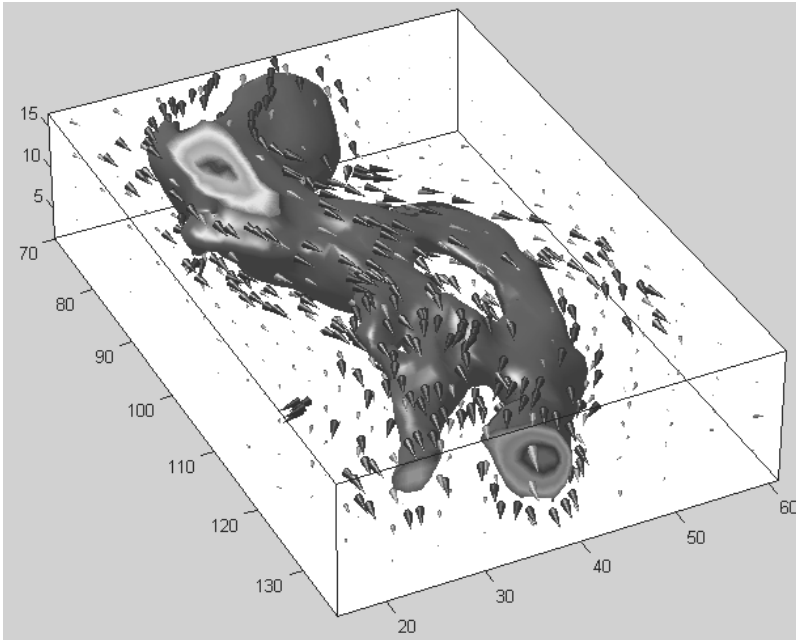


Рис. 7.34. Визуализация потоков и струй с помощью конусов

```

set(hiso,'FaceColor','red','EdgeColor','none');
hcap = patch(isocaps(x,y,z,wind_speed,40),...
    'FaceColor','interp','EdgeColor','none');
colormap hsv; daspect([1,1,1]);
[f verts] = ...
    reducepatch(isosurface(x,y,z,wind_speed,30),0.07);
h1 = ...
    coneplot(x,y,z,u,v,w,verts(:,1),verts(:,2),verts(:,3),3);
set(h1,'FaceColor','blue','EdgeColor','none');
xrange = linspace(min(x(:)),max(x(:)),10);
yrange = linspace(min(y(:)),max(y(:)),10); zrange = 3:4:15;
[cx,cy,cz] = meshgrid(xrange,yrange,zrange);
h2 = coneplot(x,y,z,u,v,w,cx,cy,cz,2);
set(h2,'FaceColor','green','EdgeColor','none');
axis tight; box on; camproj perspective;
camzoom(1.25); view(65,45);
camlight(-45,45); set(gcf,'Renderer','zbuffer');
lighting phong; set(hcap,'AmbientStrength',.6)

```

Возможно, некоторым читателям представленные выше программы могут показаться довольно сложными. Однако с позиций программирования это простейшие линейные программы, не содержащие никаких управляющих структур и входных параметров. Они реализуются в виде Script-файлов (m-файлов процедур) и могут быть исполнены как программы, так и по частям в командной строке.

Программные средства численных методов

| | |
|--|-----|
| 8.1. Решение систем линейных уравнений (СЛУ) | 384 |
| 8.2. Решение СЛУ с разреженными матрицами | 388 |
| 8.3. Вычисление корней функций | 394 |
| 8.4. Вычисление минимумов функций | 398 |
| 8.5. Аппроксимация производных | 403 |
| 8.6. Численное интегрирование | 408 |
| 8.7. Математические операции с полиномами | 411 |
| 8.8. Обыкновенные дифференциальные уравнения (ОДУ) | 416 |
| 8.9. Примеры решения дифференциальных уравнений | 422 |

В этом большом уроке описываются программные средства (в основном встроенные функции) системы MATLAB, предназначенные для реализации алгоритмов типовых численных методов решения прикладных задач [2, 3, 53–64]. Наряду с базовыми операциями решения систем линейных и нелинейных уравнений рассмотрены функции вычисления конечных разностей, численного дифференцирования, численного интегрирования, триангуляции и аппроксимации Лапласиана. Отдельные разделы посвящены работе с полиномами. Особое внимание уделено численным методам решения обыкновенных дифференциальных уравнений в среде MATLAB.

8.1. Решение систем линейных уравнений (СЛУ)

8.1.1. Элементарные средства

Решение *систем линейных уравнений (СЛУ)* относится к самой массовой области применения матричных методов, описанных в уроках 4 и 5. Как известно, обычная СЛУ имеет вид:

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n = b_1,$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n = b_2,$$

...

$$a_{n,1}x_1 + a_{n,2}x_2 + \dots + a_{n,n}x_n = b_n.$$

Здесь $a_{1,1}, a_{1,2}, \dots, a_{n,n}$ – коэффициенты, образующие матрицу A , которые могут иметь действительные или комплексные значения, x_1, x_2, \dots, x_n – неизвестные, образующие вектор X , и b_1, b_2, \dots, b_n – свободные члены (действительные или комплексные), образующие вектор B . Эта система может быть представлена в матричном виде как $AX=B$, где A – матрица коэффициентов уравнений, X – искомый вектор неизвестных и B – вектор свободных членов. В зависимости от вида матрицы A и ее характерных особенностей MATLAB позволяет реализовать различные методы решения.

Для реализации различных алгоритмов решения СЛУ и связанных с ними матричных операций применяются следующие матричные операторы: $+$, $-$, $*$, $/$, \backslash , \wedge , $\backslash\backslash$. Как отмечалось ранее, MATLAB имеет два различных типа арифметических операций – поэлементные и для массивов (векторов и матриц) в целом. Матричные арифметические операции определяются *правилами линейной алгебры*.

Арифметические операции сложения и вычитания над массивами выполняются поэлементно. Знак точки $.$ отличает операции над элементами массивов от матричных операций. Однако поскольку операции сложения и вычитания одинаковы для матрицы и элементов массива, знаки $+$ и $-$ не используются. Рассмотрим другие операторы и выполняемые ими операции:

- $*$ – матричное умножение;
- $C = A*B$ – линейное алгебраическое произведение матриц A и B :

$$C(i, j) = \sum_{k=1}^n A(i, k) * B(k, j).$$

Для случая нескаллярных A и B число столбцов матрицы A должно равняться числу строк матрицы B . Скаляр может умножаться на матрицу любого размера;

- $/$ – *правое* деление. Выражение $X=B/A$ дает решение ряда систем линейных уравнений $AX=B$, где A – матрица размера $m \times n$ и B – матрица размера $n \times k$;
- \backslash – *левое* деление. Выражение $X=B \backslash A$ дает решение ряда систем линейных уравнений $XA=B$, где A – матрица размера $m \times n$ и B – матрица размера $n \times k$. Если A – квадратная матрица, то $A \backslash B$ – примерно то же самое, что и $\text{inv}(A) * B$, в остальных случаях возможны варианты, отмеченные ниже.

Если A – матрица размера $n \times n$, а B – вектор-столбец с n компонентами или матрица с несколькими подобными столбцами, тогда $X=A \backslash B$ – решение уравнения $AX=B$, которое находится хорошо известным методом исключения Гаусса.

Если A – матрица размера $m \times n$ и $m \times n$, а B представляет собой вектор-столбец с m компонентами или матрицу с несколькими такими столбцами, тогда система оказывается недоопределенной или переопределенной и решается на основе минимизации второй нормы невязок.

Ранг k матрицы A находится на основе QR-разложения с выбором ведущего элемента. Полученное решение X будет иметь не больше чем k ненулевых компонентов на столбец. Если $k < n$, то решение, как правило, не будет совпадать с $\text{pinv}(A) * B$, которое имеет наименьшую норму невязок $\|X\|$;

- \wedge – возведение матрицы в степень. $X \wedge p$ – это X в степени p , если p – скаляр. Если p – целое число, то степень матрицы вычисляется путем умножения X на себя p раз. Если p – целое отрицательное число, то X сначала инвертируется. Для других значений p вычисляются собственные значения и собственные векторы, так что если $[V, D] = \text{eig}(X)$, то $X \wedge p = V * D . \wedge p / V$. Если X – скаляр и P – матрица, то $X \wedge P$ – это скаляр X , возведенный в матричную степень P . Если X и P – матрицы, то $X \wedge P$ становится некорректной операцией и система выдает сообщение об ошибке. Возможный вариант решения матричного уравнения с применением оператора \wedge можно представить как $X=B * A \wedge -1$;
- $'$ – транспонирование матрицы, то есть замена строк столбцами, и наоборот. Например, A' – транспонированная матрица A . Для комплексных матриц транспонирование дополняется комплексным сопряжением. Транспонирование при решении СЛУ полезно, если в матрице A переставлены местами столбцы и строки.

При записи СЛУ в матричной форме необходимо следить за правильностью записи матрицы A и вектора B . Пример (в виде m -файла):

$$A = \begin{bmatrix} 2 & 1 & 0 & 1; \\ 1 & -3 & 2 & 4; \\ -5 & 0 & -1 & -7; \\ 1 & -6 & 2 & 6; \end{bmatrix};$$

```

B=[8      9      -5      0];
X1=B/A
X2=B*A^-1
X3=B*inv(A)

```

Эта программа выдает три варианта результатов решения:

```

X1 = 3.0000    -4.0000    -1.0000     1.0000
X2 = 3.0000    -4.0000    -1.0000     1.0000
X3 = 3.0000    -4.0000    -1.0000     1.0000

```

Как и следовало ожидать, результаты оказываются одинаковыми для всех трех способов решения.

8.1.2. Решение систем линейных уравнений с ограничениями

Теперь рассмотрим функции, введенные для решения систем линейных уравнений с ограничениями, методом наименьших квадратов:

- $X = \text{lsqcov}(A, B, V)$ – возвращает вектор X решения СЛУ вида $A \cdot X = B + e$, где e – вектор шумов, который имеет ковариационную матрицу V . Реализуется метод наименьших квадратов в присутствии шумов с известной ковариацией. Прямоугольная матрица A должна быть размера $m \times n$, где $m > n$. При решении минимизируется следующее выражение: $(AX - b)' \cdot \text{inv}(V) \cdot (AX - b)$. Решение имеет вид $X = \text{inv}(A' \cdot \text{inv}(V) \cdot A) \cdot A' \cdot \text{inv}(V) \cdot B$. Алгоритм решения, однако, построен так, что операция инвертирования матрицы V не проводится;
- $[X, dX] = \text{lsqcov}(A, B, V)$ – возвращает также стандартную погрешность X , помещая ее в переменную dX . Статистическая формула для стандартной погрешности вычисления коэффициентов имеет следующий вид:

$$\text{mse} = B' \cdot (\text{inv}(V) - \text{inv}(V) \cdot A \cdot \text{inv}(A' \cdot \text{inv}(V) \cdot A) \cdot A' \cdot \text{inv}(V)) \cdot B / (m - n)$$

$$dX = \text{sqrt}(\text{diag}(\text{inv}(A' \cdot \text{inv}(V) \cdot A) \cdot \text{mse}))$$
- $X = \text{lsqnonneg}(A, B)$ – решение СЛУ $AX = B$ методом наименьших квадратов с неотрицательными ограничениями. A – действительная прямоугольная матрица, B – действительный вектор. Вектор X содержит неотрицательные элементы $X_j \geq 0$, где $j = 1, 2, \dots, n$. Критерий: минимизация второй нормы вектора $B - AX$;
- $X = \text{lsqnonneg}(A, B, X0)$ – решение СЛУ с явно заданным неотрицательным начальным значением X для итераций;
- $[X, W] = \text{lsqnonneg}(\dots)$ – вместе с решением возвращает вторую норму вектора остатков в квадрате;
- $[X, W, W1] = \text{lsqnonneg}(\dots)$ – вместе с решением возвращает вторую норму вектора остатков в квадрате и вектор остатков $W1$;
- $[X, W, W1, \text{exitflag}] = \text{lsqnonneg}(\dots)$ – вместе с решением возвращает вторую норму вектора остатков в квадрате, вектор остатков $W1$ и флаг exitflag , равный 1, если решение сходится после заданного по умолчанию числа итераций, и 0 в противном случае;

- `[X,W,W1,exitflag,output] = lsqnonneg(...)` – дополнительно возвращает число итераций в `output`;
- `[X,W,W1,exitflag,output,lambda] = lsqnonneg(...)` – дополнительно возвращает вектор `lambda`, минимизирующий произведение `lambda W1` на последнем шаге итераций решения. `lambda(i)` близко к нулю, когда `X(i)` положительно, `lambda(i)` отрицательно, когда `X(i)` равно 0;
- `X=lsqnonneg(A,B,X0,options)` или `[X,W,W1,exitflag,output,lambda] = lsqnonneg(A,B,X0,options)` – обычно используется, если при предыдущей попытке решения системы `exitflag = 1` или если необходимо изменить заданный по умолчанию порог отбора `X - tolX`, равный $10 \cdot \max(\text{size}(A)) \cdot \text{norm}(A,1) \cdot \text{eps}$ (произведению первой нормы матрицы, большего из измерений матрицы, машинной точности и 10). Параметры `options` должны быть предварительно заданы при помощи команды `optimset`. `Lsqnonneg` использует только параметры `'display'` и `'tolX'`. Поэтому наиболее часто используемая вместе с `lsqnonneg` форма записи команды – `options=optimset('tolX',tolvalue)`, где `tolvalue` – новое значение порога отбора по `X`.

Применение ограничений позволяет избежать получения отрицательных корней, хотя и ведет к появлению несколько больших погрешностей решения, чем в случае решений без ограничений. Пример:

```
>> C=[4 3;12 5;3 12];b=[1,45,41];D=lsqnonneg(C,b')
D =
    2.2242
    2.6954
```

8.1.3. Решение систем линейных уравнений с комплексными элементами

В ряде задач, например при расчете электрических цепей символическим методом, нужно решение систем линейных уравнений с комплексными элементами матрицы и вектора свободных членов. Приведем пример такого решения для системы из трех уравнений:

```
>> A=[4+i 0.24 -0.08;0.09 3 -0.15; 0.04 -0.08 4+i]
A =
    4.0000 + 1.0000i    0.2400    -0.0800
    0.0900           3.0000    -0.1500
    0.0400           -0.0800    4.0000 + 1.0000i
>> B=[8 9 20]
B =
     8     9    20
>> X=B/A
X =
    1.7764 - 0.4319i    2.9870 + 0.0020i    4.8427 - 1.2192i
```


Проверка показывает, что в среде MATLAB это решение верно:

```
>> X*A
ans =
  8.0000    9.0000 - 0.0000i   20.0000
```

Заметим, что в данном случае вектор B задан как вектор-строка, так что решает-ся по существу иная система уравнений. Далее мы приведем другой пример решения, при котором используется представление вектора B в виде вектора-столбца.

8.2. Решение СЛУ с разреженными матрицами

Решение СЛУ с разреженными матрицами – хотя и не единственная, но, несомненно, одна из основных сфер применения аппарата разреженных матриц, описанного в уроке 5. Ниже приведены функции, относящиеся к этой области применения разреженных матриц. Большинство описанных методов относится к итерационным, то есть к тем, решение которых получается в ходе ряда шагов – итераций, постепенно ведущих к получению результата с заданной погрешностью или с максимальным правдоподобием. Описанные ниже функции MATLAB могут применяться и при решении обычных СЛУ (без разреженных матриц), что и демонстрируют приведенные ниже примеры.

8.2.1. Точное решение, метод наименьших квадратов и сопряженных градиентов

Начнем описание точных решений СЛУ с приведенных ниже функций.

- `lsqr(A, B)` – возвращает точное решение X СЛУ $A \cdot X = B$, если оно существует, в противном случае возвращает решение по итерационному методу наименьших квадратов. Матрица коэффициентов A должна быть прямоугольной размера $m \times n$, а вектор-столбец правых частей уравнений B должен иметь размер m . Условие $m \geq n$ может быть и необязательным. Функция `lsqr` начинает итерации от начальной оценки, по умолчанию представляющей собой вектор длиной n , состоящий из нулей. Итерации производятся или до сходимости к решению, или до появления ошибки, или до достижения максимального числа итераций (по умолчанию равного $\min(20, m, n)$ – либо 20, либо числу уравнений, либо числу неизвестных). Сходимость достигается, когда отношение вторых норм векторов $\text{norm}(B - Ax) / \text{norm}(B)$ меньше или равно погрешности метода `tol` (по умолчанию $1e-6$).
- `lsqr(A, B, tol)` – возвращает решение с заданной погрешностью (порогом отбора) `tol`.
- `lsqr(A, b, tol, maxit)` – возвращает решение при заданном максимальном числе итераций `maxit` вместо, возможно, чересчур малого числа, заданного по умолчанию.

- `lsqr(A,b,tol,maxit,M)` и `lsqr(A,b,tol,maxit,M1,M2)` – при решении используется матрица предусловий M или $M=M1*M2$, так что производится решение системы $\text{inv}(M)*A*x=\text{inv}(M)*b$ относительно x . Если $M1$ или $M2$ – пустые матрицы, то они рассматриваются как единичные матрицы, что эквивалентно отсутствию входных условий вообще.
- `lsqr(A,B,tol,maxit,M1,M2,X0)` – точно задается начальное приближение $X0$. Если $X0$ – пустая матрица, то по умолчанию используется вектор, состоящий из нулей.
- `X=lsqr(A,B,tol,maxit,M1,M2,X0)` – при наличии единственного выходного параметра возвращает решение X . Если метод `lsqr` сходится, выводится соответствующее сообщение. Если метод не сходится после максимального числа итераций или по другой причине, на экран выдаются относительный остаток $\text{norm}(B-A*X)/\text{norm}(B)$ и номер итерации, на которой метод остановлен.
- `[X,flag]=lsqr(A,X,tol,maxit,M1,M2,X0)` – возвращает решение X и флаг `flag`, описывающий сходимость метода.
- `[X,flag,relres]=lsqr(A,X,tol,maxit,M1,M2,X0)` – также возвращает относительную вторую норму вектора остатков $\text{relres}=\text{norm}(B-A*X)/\text{norm}(B)$. Если флаг `flag` равен 0, то $\text{relres} \leq \text{tol}$.
- `[X,flag,relres,iter]=bicg(A,B,tol,maxit,M1,M2,X0)` – также возвращает номер итерации, на которой был вычислен X . Значение `iter` всегда удовлетворяет условию $0 \leq \text{iter} \leq \text{maxit}$.
- `[X,flag,relres,iter,resvec]=lsqr(A,B,tol,maxit,M1,M2,X0)` – также возвращает вектор вторых норм остатков `resvec` для каждой итерации, начиная с `resvec(1)=norm(B-A*X0)`. Если флаг `flag` равен 0, то `resvec` имеет длину `iter+1` и `resvec(end) \leq \text{tol}*\text{norm}(B)`. Возможны значения `flag`, равные 0, 1, 2, 3 и 4.

Значения `flag` предоставляют следующие данные о сходимости решения:

- 0 – решение сходится при заданной точности `tol` и числе итераций не более заданного `maxit`;
- 1 – число итераций равно заданному `maxit`, но сходимость не достигнута;
- 2 – матрица предусловий M плохо обусловлена;
- 3 – процедура решения остановлена, поскольку две последовательные оценки решения оказались одинаковыми;
- 4 – одна из величин в процессе решения вышла за пределы допустимых величин чисел (разрядной сетки компьютера).

Если флаг больше нуля, то возвращается не последнее решение, а то решение, которое имеет минимальное значение отношения вторых норм векторов $\text{norm}(B-A*x)/\text{norm}(B)$.

Пример:

```
>> A=[0 0 1 2;      1 3 0 0;      0 1 0 1;      1 0 1 0];
>> B=[11;
      7;
      6;
      4];
```

Введенные в этом примере матрица A и вектор B будут использованы и в других примерах данного раздела. В приведенном ниже примере процесс итераций сходится на пятом шаге с относительным остатком (отношением вторых норм векторов невязки и свободных членов) $1.9 \cdot 10^{-13}$:

```
>> lsqr(A,B,1e-6,5)
lsqr converged at iteration 5 to a solution with relative residual
1.9e-013
ans =
1.0000
2.0000
3.0000
4.0000
```

А теперь проверим работу функции `lsqr` на решении системы из трех линейных уравнений с частично комплексными элементами:

```
>> A=[4+i 0.24 -0.08;0.09 3 -0.15; 0.04 -0.08 4+i]
A =
 4.0000 + 1.0000i    0.2400                -0.0800
 0.0900                3.0000                -0.1500
 0.0400                -0.0800                4.0000 + 1.0000i
>> B=[8 9 20]
B = 8    9    20
>> lsqr(A,B')
lsqr stopped at iteration 3 without converging to the desired
tolerance 1e-006 because the maximum number of iterations was reached.
The iterate returned (number 3) has relative residual 3.5e-017
ans =
 1.7870 - 0.4677i
 3.1839 - 0.0452i
 4.7499 - 1.1837i
>> A*X
ans =
 8.0000 - 0.0000i
 9.0000 - 0.0000i
20.0000 + 0.0000i
```

8.2.2. Двухнаправленный метод сопряженных градиентов

Решение СЛУ с разреженной матрицей возможно также известным *двухнаправленным методом сопряженных градиентов*. Он реализован указанной ниже функцией.

- `bicg(A, B)` – возвращает решение X СЛУ $A \cdot X = B$. Матрица коэффициентов A должна быть квадратной размера $n \times n$, а вектор-столбец правых частей уравнений B должен иметь длину n . Функция `bicg` начинает итерации от начальной оценки, по умолчанию представляющей собой вектор длины n ,

состоящий из нулей. Итерации производятся или до сходимости к решению, или до появления ошибки, или до достижения максимального числа итераций (по умолчанию равно $\min(20, n)$ – либо 20, либо числу уравнений). Сходимость достигается, когда относительный остаток $\text{norm}(B-A*x) / \text{norm}(B)$ меньше или равен погрешности метода (по умолчанию $1e-6$). Благодаря использованию двунаправленного метода сопряженных градиентов `bicg` сходится за меньшее число итераций, чем `lsqr` (в нашем примере быстрее на одну итерацию), но требует квадратную матрицу A , отбрасывая информацию, содержащуюся в дополнительных уравнениях, в то время как `lsqr` работает и с прямоугольной матрицей.

- `bicg(A, B, tol)` – выполняет и возвращает решение с погрешностью (порогом отбора) `tol`.
- `bicg(A, b, tol, maxit)` – выполняет и возвращает решение при заданном максимальном числе итераций `maxit`.
- `bicg(A, b, tol, maxit, M)` и `bicg(A, b, tol, maxit, M1, M2)` – при решении используется матрица предусловий M или $M=M1*M2$, так что производится решение системы $\text{inv}(M)*A*x=\text{inv}(M)*b$ относительно x . Если $M1$ или $M2$ – пустые матрицы, то они рассматриваются как единичные матрицы, что эквивалентно отсутствию входных условий вообще.
- `bicg(A, B, tol, maxit, M1, M2, X0)` – точно задается начальное приближение $X0$. Если $X0$ – пустая матрица, то по умолчанию используется вектор, состоящий из нулей.
- `X=bicg(A, B, tol, maxit, M1, M2, X0)` – при наличии единственного выходного параметра возвращает решение X . Если метод `bicg` сходится, выводится соответствующее сообщение. Если метод не сходится после максимального числа итераций или по другой причине, на экран выдаются относительный остаток $\text{norm}(B-A*X) / \text{norm}(B)$ и номер итерации, на которой метод остановлен.
- `[X, flag] = bicg(A, X, tol, maxit, M1, M2, X0)` – возвращает решение X и флаг `flag`, описывающий сходимость метода.
- `[X, flag, relres] = bicg(A, X, tol, maxit, M1, M2, X0)` – также возвращает относительную вторую норму вектора остатков $\text{relres}=\text{norm}(B-A*X) / \text{norm}(B)$. Если флаг `flag` равен 0, то $\text{relres} \leq \text{tol}$.
- `[X, flag, relres, iter] = bicg(A, B, tol, maxit, M1, M2, X0)` – также возвращает номер итерации, на которой был вычислен X . Значение `iter` всегда удовлетворяет условию $0 \leq \text{iter} \leq \text{maxit}$.
- `[X, flag, relres, iter, resvec] = bicg(A, B, tol, maxit, M1, M2, X0)` – также возвращает вектор вторых норм остатков `resvec` для каждой итерации, начиная с $\text{resvec}(1)=\text{norm}(B-A*X0)$. Если флаг `flag` равен 0, то `resvec` имеет длину `iter+1` и $\text{resvec}(\text{end}) \leq \text{tol} * \text{norm}(B)$. Возможны значения `flag`, равные 0, 1, 2, 3 и 4.

Значения `flag` предоставляют следующие данные о сходимости решения:

- 0 – решение сходится при заданной точности `tol` и числе итераций не более заданного `maxit`;

- 1 – число итераций равно заданному `maxit`, но сходимость не достигнута;
- 2 – матрица предусловий `M` плохо обусловлена;
- 3 – процедура решения остановлена, поскольку две последовательные оценки решения оказались одинаковыми;
- 4 – одна из величин в процессе решения вышла за пределы допустимых величин чисел (разрядной сетки компьютера).

Пример:

```
>> bicg(A,B)
BICG converged at iteration 4 to a solution with relative residual
2.3e-015
ans =
    1.0000
    2.0000
    3.0000
    4.0000
```

8.2.3. Устойчивый двунаправленный метод

Еще один двунаправленный метод, называемый устойчивым, реализует функция `bicgstab`:

- `bicgstab(A,B)` – возвращает решение X СЛУ $A \cdot X = B$. A – квадратная матрица. Функция `bicgstab` начинает итерации от начальной оценки, по умолчанию представляющей собой вектор длиной n , состоящий из нулей. Итерации производятся либо до сходимости метода, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость метода достигается, когда относительный остаток $\text{norm}(B - A \cdot X) / \text{norm}(B)$ меньше или равен погрешности метода (по умолчанию $1e-6$). Функция `bicgstab(...)` имеет и ряд других форм записи, аналогичных описанным для функции `bicg(...)`.

Пример, представленный выше для функции `bicg`, можно использовать и для этой функции.

8.2.4. Метод сопряженных градиентов

Итерационный метод сопряженных градиентов реализован функцией `pcg`:

- `pcg(A,B)` возвращает решение X СЛУ $A \cdot X = B$. Матрица A должна быть квадратной, симметрической и положительно определенной¹. Функция `pcg` начинает итерации от начальной оценки, представляющей собой вектор длиной n , состоящий из нулей. Итерации производятся либо до сходимости решения, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость достигается, если относительный остаток

¹ Матрица называется положительно определенной, если все ее собственные значения (характеристические числа) действительные и положительные.

$\text{norm}(b-A^*X) / \text{norm}(B)$ меньше или равен погрешности метода (по умолчанию $1e-6$). Максимальное число итераций – минимум из n и 20.

Функция `pcg(...)` имеет и ряд других форм записи, описанных для функции `bicg(...)`. Новая функция `minres` не требует, чтобы матрица A была положительно определенной. Достаточно, чтобы она была квадратной и симметрической. В отличие от `pcg`, минимизируется не относительная невязка, а абсолютная. Начиная с MATLAB 6, появилась еще одна новая функция `symmlq`, которая использует LQ-алгоритм итерационного метода сопряженных градиентов и также не требует, чтобы квадратная симметрическая матрица A была положительно определенной. С примерами применения этих функций можно познакомиться по справке.

8.2.5. Квадратичный метод сопряженных градиентов

Квадратичный метод сопряженных градиентов реализуется с помощью функции `cgs`:

- `cgs(A, B)` возвращает решение X СЛУ $A^*X=B$. A – квадратная матрица. Функция `cgs` начинает итерации от начальной оценки, по умолчанию представляющей собой вектор размера n , состоящий из нулей. Итерации производятся либо до сходимости метода, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость метода достигается, когда относительный остаток $\text{norm}(B-A^*X) / \text{norm}(B)$ меньше или равен погрешности метода (по умолчанию $1e-6$). Функция `cgs(...)` имеет и ряд других форм записи, аналогичных описанным для функции `bicg(...)`. Проверить работу этой функции можно по приведенному ранее примеру.

8.2.6. Метод минимизации обобщенной невязки

Итерационный метод минимизации обобщенной невязки также реализован в системе MATLAB. Для этого используется функция `gmres`:

- `gmres(A, B, restart)` возвращает решение X СЛУ $A^*X=B$. A – квадратная матрица. Функция `gmres` начинает итерации от начальной оценки, представляющей собой вектор размера n , состоящий из нулей. Итерации производятся либо до сходимости к решению, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость достигается, когда относительный остаток $\text{norm}(B-A^*X) / \text{norm}(B)$ меньше или равен заданной погрешности (по умолчанию $1e-6$). Максимальное число итераций – минимум из $n/\text{restart}$ и 10. Функция `gmres(...)` имеет и ряд других форм записи, аналогичных описанным для функции `bicg(...)`. Эту функцию тоже можно проверить по приведенному выше примеру.

8.2.7. Квазиминимизация невязки – функция `qmr`

Метод решения СЛУ с квазиминимизацией невязки реализует функция `qmr`:

- `qmr(A, B)` возвращает решение X СЛУ $A \cdot X = b$. Матрица A должна быть квадратной. Функция `qmr` начинает итерации от начальной оценки, представляющей собой вектор длиной n , состоящий из нулей. Итерации производятся либо до сходимости метода, либо до появления ошибки, либо до достижения максимального числа итераций. Максимальное число итераций – минимум из n и 20. Функция `qmr(...)` имеет и ряд других форм записи, аналогичных описанным для функции `bicg(...)`.

Приведенный выше пример можно использовать для проверки этой функции.

8.3. Вычисление корней функций

8.3.1. Вычисление корней функций одной переменной

Ряд функций системы MATLAB предназначен для работы с функциями (function functions – по терминологии фирменного описания системы). Под функциями понимаются как встроенные функции, например $\sin(x)$ или $\exp(x)$, так и функции пользователя, например $f(x)$, задаваемые как М-файлы-функции.

Может использоваться также класс анонимных функций, задаваемых с помощью символа `@`, например:

```
>> fe=@exp;
```

Такие функции вычисляются с помощью функции `feval`:

```
>> feval(fe,1.0)
ans = 2.7183
```

Довольно часто возникает задача решения (нахождения корней) нелинейного уравнения вида $f(x) = 0$ или $f_1(x) = f_2(x)$. Последнее, однако, можно свести к виду $f(x) = f_1(x) - f_2(x) = 0$. Таким образом, данная задача сводится к нахождению значений аргумента x функции $f(x)$ одной переменной, при котором значение функции равно нулю. Соответствующая функция MATLAB, решающая данную задачу, приведена ниже:

- `fzero(fun, x)` возвращает уточненное значение x , при котором достигается нуль функции `fun`, представленной строкой, при начальном значении аргумента x . Возвращенное значение близко к точке, где функция меняет знак, или равно NaN, если такая точка не найдена;
- `fzero(fun, [x1 x2])` возвращает значение x , при котором $fun(x) = 0$ с заданием интервала поиска с помощью вектора $x = [x1 \ x2]$, такого что знак $fun(x(1))$ отличается от знака $fun(x(2))$. Если это не так, выдает-

ся сообщение об ошибке. Вызов функции `fzero` с интервалом гарантирует, что `fzero` возвратит значение, близкое к точке, где `fun` изменяет знак;

- `fzero(fun, x, tol)` возвращает результат с заданной погрешностью `tol`;
- `fzero(fun, x, tol, trace)` выдает на экран информацию о каждой итерации;
- `fzero(fun, x, tol, trace, P1, P2, ...)` предусматривает дополнительные аргументы, передаваемые в функцию `fun(x, P1, P2, ...)`. При задании пустой матрицы для `tol` или `trace` используются значения по умолчанию, например `fzero(fun, x, [], [], P1)`.

Для функции `fzero` нуль рассматривается как точка, где график функции `fun` пересекает ось x , а не касается ее. В зависимости от формы задания функции `fzero` реализуются следующие хорошо известные численные методы поиска нуля функции: деления отрезка пополам, секущей и обратной квадратичной интерполяции.

Приведенный ниже пример показывает приближенное вычисление $\pi/2$ из решения уравнения $\cos(x)=0$ с представлением косинуса анонимной функцией:

```
>> x = fzero(@cos, [1 3])
x = 1.5708
```

8.3.2. Графическая иллюстрация поиска корней функций

В более сложных случаях настоятельно рекомендуется строить график функции $f(x)$ для приближенного определения корней и интервалов, в пределах которых они находятся. Ниже дан пример такого рода (следующий листинг представляет собой содержимое `m`-файла **fun1.m**):

```
%Функция, корни которой ищутся
function f=fun1(x)
f=0.25*x+sin(x)-1;
```

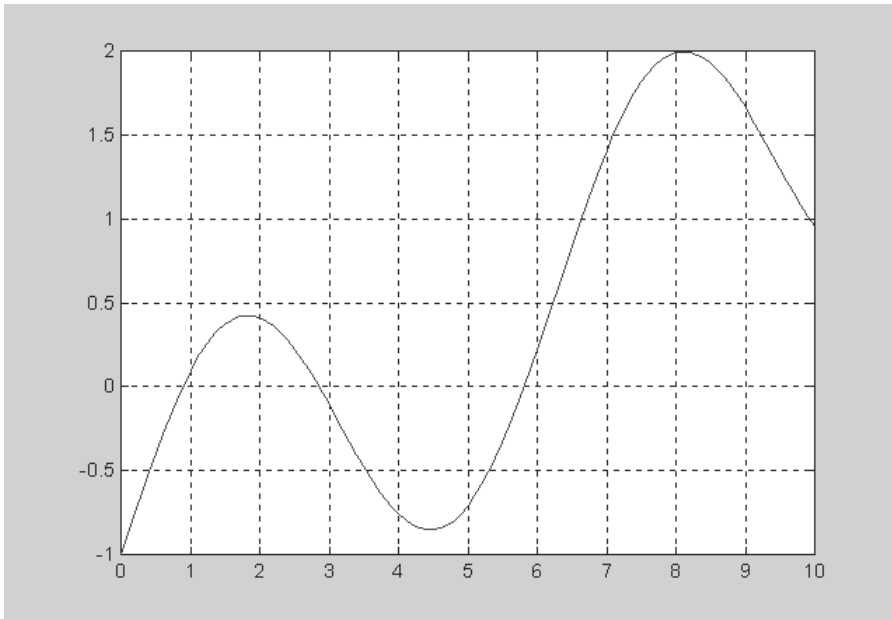
Построим график функции (рис. 8.1):

```
>> x=0:0.1:10; plot(x, fun1(x)); grid on;
```

Из него нетрудно заметить, что значения корней заключены в интервалах `[0.5 1]`, `[2 3]` и `[5 6]`. Найдем их, используя функцию `fzero`:

```
>> x1=fzero(@fun1, [0.5 1])
x1 = 0.8905
>> x2=fzero(@fun1, [2 3])
x2 = 2.8500
>> x3=fzero(@fun1, [5, 6])
x3 = 5.8128
>> x3=fzero(@fun1, 5, 0.001)
x3 = 5.8111
```

Обратите внимание на то, что корень `x3` найден двумя способами и что его значения в третьем знаке после десятичной точки отличаются в пределах заданной

Рис. 8.1. График функции $fun1(x)$

погрешности $tol=0.001$. К сожалению, сразу найти все корни функция `fzero` не в состоянии.

8.3.3. Поиск корня с помощью функций `fsolve` и `solve`

Решение таких задач возможно и с помощью функции `fsolve` из пакета Optimization Toolbox, которая решает систему нелинейных уравнений вида $f(x)=0$ методом наименьших квадратов, ищет не только точки пересечения, но и точки касания. `fsolve` имеет почти те же параметры (дополнительный параметр – задание якобиана) и почти ту же запись, что и функция `lsqnonneg`, подробно рассмотренная ранее. Пример:

```
>>fsolve('0.25*x+sin(x)-1',0:10 )
ans =
Columns 1 through 7
0.8905 0.8905 2.8500 2.8500 2.8500 5.8128 5.8128
Columns 8 through 11
5.8128 2.8500 2.8500 10.7429
```

Для решения нелинейных уравнений следует также использовать функцию `solve` из пакета Symbolic Math Toolbox. Эта функция способна выдавать результат в символьной форме, а если такого нет, то она позволяет получить решение в численном виде. Пример:

```
>> solve('0.25*x + sin(x) -1')
ans = .89048708074438001001103173059554
```

8.3.4. Решение систем нелинейных уравнений

Функция `fsolve` может использоваться и для решения систем нелинейных уравнений. В этом случае речь идет о нахождении значений двух и более переменных, удовлетворяющих решению заданной системы нелинейных уравнений. Рассмотрим следующий пример. Зададим `m`-файл с именем `myfun`, задающий систему `F` из двух уравнений, записанных как функции переменных `x(1)` и `x(2)`:

```
function F = myfun(x)
F = [2*x(1) - x(2) - log(-x(1));
     -x(1) + 2*x(2) - exp(-x(2))];
```

Тогда решение этой системы уравнений с выводом хода решения находится следующим образом:

```
>> x0 = [-5; -5];
options=optimset('Display','iter');
[x,fval] = fsolve(@myfun,x0,options)
```

| Iteration | Func-count | f(x) | Norm of step | First-order optimality | Trust-region radius |
|-----------|------------|--------------|--------------|------------------------|---------------------|
| 0 | 3 | 23579.3 | 1 | 2.31e+004 | 1 |
| 1 | 6 | 3375.08 | 1 | 3.25e+003 | 1 |
| 2 | 9 | 518.733 | 1 | 457 | 1 |
| 3 | 12 | 130.732 | 1 | 61.1 | 1 |
| 4 | 15 | 72.0289 | 1 | 16.2 | 1 |
| 5 | 18 | 12.1627 | 2.5 | 6.48 | 2.5 |
| 6 | 21 | 2.49612 | 1.68179 | 13.9 | 6.25 |
| 7 | 24 | 0.123229 | 0.190816 | 1.87 | 6.25 |
| 8 | 27 | 0.000558075 | 0.0744278 | 0.111 | 6.25 |
| 9 | 30 | 1.11275e-008 | 0.00575471 | 0.000491 | 6.25 |
| 10 | 33 | 4.39022e-018 | 2.58413e-005 | 9.75e-009 | 6.25 |

Optimization terminated: first-order optimality is less than options.TolFun.

```
x =
    -0.3786
     0.2143
fval =
    1.0e-008 *
     0.2095
    -0.0027
```

Для детального анализа решения, выходящего за рамки данного учебного курса, можно построить графики функций в системе координат `[x1,x2]`.

8.4. Вычисление минимумов функций

8.4.1. Минимизация функции одной переменной

Еще одна важная задача численных методов – поиск *минимума* функции $f(x)$ в некотором интервале изменения x – от x_1 до x_2 [59–64]. Если нужно найти *максимум* такой функции, то достаточно поставить знак «минус» перед функцией. Для решения этой задачи используется следующая функция:

```
[X,fval,exitflag,output] = fminbnd(fun,x1,x2,options, p1, p2,...)
```

- `fminbnd(fun,x1,x2)` возвращает значение x , которое является локальным минимумом функции $fun(x)$ на интервале $x_1 < x < x_2$.
- `fminbnd(fun,x1,x2,options)` сходна с описанной выше формой функции, но использует параметры `tolX`, `maxfuneval`, `maxiter`, `display` из вектора `options`, предварительно установленные при помощи команды `optimset` (смотрите описание `lsqnonneg`).
- `fminbnd(fun,x1,x2,options,P1,P2,...)` сходна с описанной выше, но передает в целевую функцию дополнительные аргументы: P_1, P_2, \dots . Если требуется использовать параметры вычислений по умолчанию, то вместо `options` перед P_1, P_2 необходимо ввести `[]` (пустой массив).
- `[x,fval] = fminbnd(...)` дополнительно возвращает значение целевой функции `fval` в точке минимума.
- `[x,fval,exitflag] = fminbnd(...)` дополнительно возвращает параметр `exitflag`, равный 1, если функция сошлась с использованием `options.tolX`, и 0, если достигнуто максимальное число итераций `options.maxiter`.

В этих представлениях используются следующие обозначения: x_1, x_2 – интервал, на котором ищется минимум функции; P_1, P_2, \dots – дополнительные, помимо x , передаваемые в функцию аргументы; `fun` – строка, содержащая название функции, которая будет минимизирована; `options` – вектор параметров вычислений.

В зависимости от формы задания функции `fminbnd` вычисление минимума выполняется известными методами золотого сечения или параболической интерполяции. Пример:

```
>> options=optimset('tolX',1.e-10);... [x]=fminbnd(@cos,3,4,options)
x =    3.1416
```

8.4.2. Минимизация функций ряда переменных симплекс-методом

Значительно сложнее задача минимизации функций нескольких переменных $f(x_1, x_2, \dots)$. При этом значения переменных представляются вектором x , причем начальные значения задаются вектором x_0 . Для минимизации функций ряда пере-

менных MATLAB обычно использует разновидности симплекс-метода Нелдера-Мида [59–64].

Этот метод является одним из лучших прямых методов минимизации функций ряда переменных, не требующим вычисления градиента или производных функции. Он сводится к построению симплекса в n -мерном пространстве, заданного $n+1$ вершиной. В двумерном пространстве симплекс является треугольником, а в трехмерном – пирамидой. На каждом шаге итераций выбирается новая точка решения внутри или вблизи симплекса. Она сравнивается с одной из вершин симплекса. Ближайшая к этой точке вершина симплекса обычно заменяется этой точкой. Таким образом, симплекс перестраивается и обычно позволяет найти новое, более точное положение точки решения. Решение повторяется, пока размеры симплекса по всем переменным не станут меньше заданной погрешности решения.

Реализующая симплекс-методы Нелдера-Мида функция записывается в виде:

- `fminsearch(fun, x0)` возвращает вектор x , который является локальным минимумом функции $fun(x)$ вблизи x_0 . x_0 может быть скаляром, вектором (отрезком при минимизации функции одной переменной) или матрицей (для функции нескольких переменных);
- `fminsearch(fun, x0, options)` аналогична описанной выше функции, но использует вектор параметров `options` точно так же, как функция `fminbnd`;
- `fminsearch(fun, x0, options, P1, P2, ...)` сходна с описанной выше функцией, но передает в минимизируемую функцию нескольких переменных `fun(x, P1, P2, ...)` ее дополнительные аргументы P_1, P_2, \dots . Если требуется использовать параметры вычислений по умолчанию, то вместо `options` перед P_1, P_2 необходимо ввести `[]`;
- `[x, fval] = fminsearch(...)` дополнительно возвращает значение целевой функции `fval` в точке минимума;
- `[x, fval, exitflag] = fminsearch(...)` дополнительно возвращает параметр `exitflag`, положительный, если процесс итераций сходится с использованием `options.tolX`, отрицательный, если итерационный процесс не сходится к полученному решению x , и 0, если превышено максимальное число итераций `options.maxiter`;
- `[x, fval, exitflag, output] = fminsearch(...)` возвращает структуру (запись) `output`;
- `output.algorithm` – использованный алгоритм;
- `output.funcCount` – число оценок целевой функции;
- `output.iterations` – число проведенных итераций.

Пользователям, имеющим опыт работы со старыми версиями MATLAB, следует учитывать различия в именах функций минимизации. Они представлены ниже:

Версия 4 и ниже

`fmin`
`fmins`
`foption`
`nls`

Версия 5 и выше

`fminbnd`
`fminsearch`
`soptimget`, `optimsetfzero`
`lsqnonneg`

8.4.3. Минимизация тестовой функции Розенброка

Классическим примером применения функции `fminsearch` является поиск минимума тестовой функции Розенброка, точка минимума которой находится в «овраге» с «плоским дном»:

$$rb(x_1, x_2, a) = 100*(x_2 - x_1^2)^2 + (a - x_1)^2.$$

Минимальное значение этой функции равно нулю и достигается в точке $[a \ a^2]$. В качестве примера уточним значения x_1 и x_2 в точке $[-1.2 \ 1]$. Зададим функцию (в файле `rb.m`):

```
% Тестовая функция Розенброка
function f=rb(x,a)
if nargin<2 a=1; end
f=100*(x(2)-x(1)^2)^2+(a-x(1))^2;
```

Теперь решим поставленную задачу:

```
>> options=optimset('tolX',1.e-6); [xmin, opt, rosexflag,
rosout]=fminsearch(@rb,[-1.2 1],options)
xmin = 1.0000 1.0000
opt = 4.1940e-014
rosexflag = 1
rosout =
    iterations: 101
    funcCount: 189
    algorithm: 'Nelder-Mead simplex direct search'
```

Для лучшего понимания сути минимизации функции нескольких переменных рекомендуется просмотреть пример минимизации этой функции, имеющийся в библиотеке демонстрационных примеров **Demos** (рис. 8.2).

8.4.4. Другие средства минимизации функций нескольких переменных

Для минимизации функций нескольких переменных можно использовать также функцию MATLAB `fminunc` и функцию `lsqnonlin` из пакета Optimization Toolbox. Первая из них позволяет использовать предварительно заданные командой `optimset` порог сходимости для значения целевой функции, вектор градиентов `options.gradobj`, матрицу Гесса, функцию умножения матрицы Гесса или график разреженности матрицы Гесса целевой функции. `lsqnonlin` реализует метод наименьших квадратов и, как правило, дает наименьшее число итераций при минимизации. Ограничимся приведением примеров их применения для минимизации функции Розенброка:

```
>> options=optimset('tolX',1e-6,'TolFun',1e-6);
>> [xmin, opt, exflag, out, grad, hessian ]=fminunc(@rb,[-1.2
1], options)
```

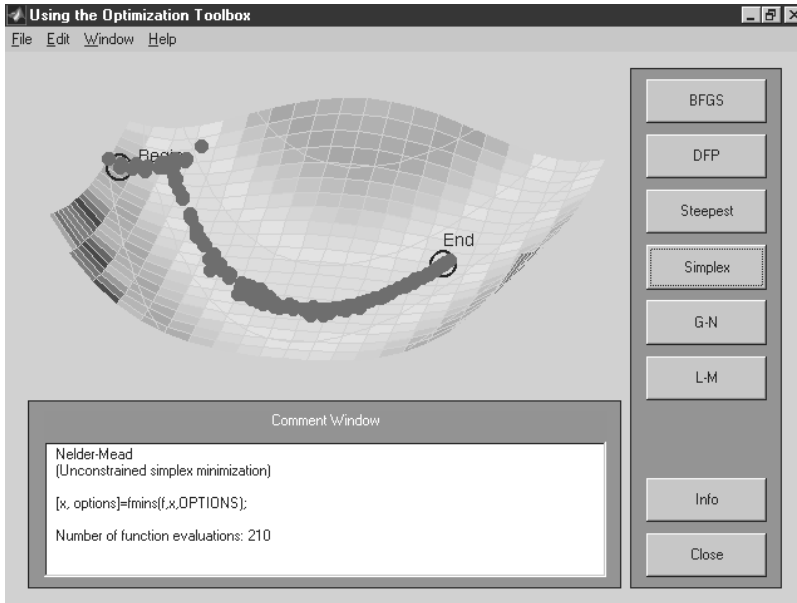


Рис. 8.2. Графическая иллюстрация минимизации функции Розенброка симплекс-методом

```
Warning: Gradient must be provided for trust-region method;
         using line-search method instead.
> In C:\MATLABR12\toolbox\optim\fminunc.m at line 211
Optimization terminated successfully:
Current search direction is a descent direction, and magnitude of
directional derivative in search direction less than
2*options.TolFun
xmin =
    1.0000    1.0000
opt =
    1.9116e-011
exflag =
         1
out =
    iterations: 26
    funcCount: 162
    stepsize: 1.2992
    firstorderopt: 5.0020e-004
    algorithm: 'medium-scale: Quasi-Newton line search'
grad =
    1.0e-003 *
    -0.5002
    -0.1888
```

```
hessian =
      820.4028    -409.5496
     -409.5496     204.7720
```

`firstorderopt` – мера оптимальности для первой нормы градиента целевой функции в найденной точке минимума:

```
>> options=optimset('tolX',1e-6, 'maxFunEvals',162);
>> [xmin, opt]=lsqnonlin(@rb,[-1.2 1],[0 1e-6],[0 1e-6],options)
Warning: Large-scale method requires at least as many equations as
variables; switching to line-search method instead. Upper and
lower bounds will be ignored.
> In C:\MATLABR12\toolbox\optim\private\lsqnoncommon.m at line 155
In C:\MATLABR12\toolbox\optim\lsqnonlin.m at line 121
Maximum number of function evaluations exceeded
Increase OPTIONS.maxFunEvals
xmin = 0.6120    0.3715
opt = 0.1446
```

Обратите внимание на то, что вопреки ожиданиям функция `lsqnonlin` к успеху не привела. Выдано сообщение о превышении лимита числа итераций, а значения `xmin` явно далеки от верных.

Можно попытаться применить функцию `fminsearch` для поиска минимума функций и с большим числом переменных. Определим в виде `m`-файла функцию трех переменных `three_var`:

```
function b = three_var(v)
x = v(1); y = v(2); z = v(3);
b = x.^2 + 2.5*sin(y) - z^2*x^2*y^2;
```

А теперь найдем минимум этой функции при разных начальных значениях переменных:

```
>> v = [-0.6 -1.2 0.135];
a = fminsearch(@three_var,v)
a = 0.0000    -1.5708    0.1803
>> three_var(a)
ans = -2.5000
>> v = [-1 -1.2 0];
>> a = fminsearch(@three_var,v)
a = 0.0000    -1.5708    0.0015
>> three_var(a)
ans = -2.5000
>> v = [-1 -1.2 0.2];
>> a = fminsearch(@three_var,v)
a = 0.0000    -1.5708    0.25
>> three_var(a)
ans = -2.5000
```

Обратите внимание на то, что минимум по первым двум переменным находится один и тот же при разных начальных значениях переменных. Однако минимум по третьей переменной существенно зависит от начальных значений переменных. Тем не менее само значение функции во всех случаях одно и то же.

А теперь приведем случай, когда решение не достигается:

```
>> v = [-1 -1.2 1];
>> a = fminsearch(@three_var,v)
Exiting: Maximum number of function evaluations has been exceeded
       - increase MaxFunEvals option.
       Current function value: -Inf
a = 1.0e+051 *
    -0.5630    -7.3469     3.8861
```

В библиотеке примеров **Demos** вы найдете дополнительные наглядные примеры применения данных функций. Новые мощные средства вычисления экстремумом функций ряда переменных имеются в пакетах расширения Optimization Toolbox и Genetic Algorithm and Direct Search Toolbox, содержащих новые мощные алгоритмы решения оптимизационных задач, в том числе гинетические и прямого поиска. Они могут использоваться, в частности, для поиска глобального экстремума сложных функций, решения задач кластеризации и др.

8.5. Аппроксимация производных

8.5.1. Аппроксимация лапласиана

Для выполнения *аппроксимации лапласиана* в MATLAB используется следующая функция:

- `del2(U)` возвращает матрицу L дискретной аппроксимации дифференциального оператора Лапласа, примененного к функции U :

$$L = \frac{\nabla^2 u}{4} = \frac{1}{4} \left(\frac{d^2 u}{dx^2} + \frac{d^2 u}{dy^2} \right).$$

Матрица L имеет тот же размер, что и матрица U , и каждый ее элемент равен разности элемента массива U и среднего значения четырех его соседних элементов (для узлов сетки во внутренней области). Для вычислений используется пятиточечная формула аппроксимации лапласиана.

Другие формы этой функции также возвращают матрицу L , но допускают дополнительные установки:

- `L = del2(U, h)` использует шаг h для установки расстояния между точками в каждом направлении (h – скалярная величина). По умолчанию $h=1$;
- `L = del2(U, hx, hy)` использует hx и hy для точного определения расстояния между точками. Если hx – скалярная величина, то задается расстояние между точками в направлении оси x , если hx – вектор, то он должен иметь размер, равный числу столбцов матрицы U , и точно определять координаты точек по оси x . Аналогично, если hy – скалярная величина, то задается расстояние между точками в направлении оси y , если hy – вектор, то он должен иметь размер, равный числу строк матрицы U , и точно определять координаты точек по оси y ;

- `L = del2(U, hx, hy, hz, ...)` – если `U` является многомерным массивом, то расстояния задаются с помощью параметров `hx`, `hy`, `hz`, ...

Пример:

```
>> [x,y] = meshgrid(-5:5,-4:4);
>> U =x.*x+y.*y
U =
    41    32    25    20    17    16    17    20    25    32    41
    34    25    18    13    10    9    10    13    18    25    34
    29    20    13    8     5     4     5     8    13    20    29
    26    17    10    5     2     1     2     5    10    17    26
    25    16    9     4     1     0     1     4     9    16    25
    26    17    10    5     2     1     2     5    10    17    26
    29    20    13    8     5     4     5     8    13    20    29
    34    25    18    13    10    9    10    13    18    25    34
    41    32    25    20    17    16    17    20    25    32    41
>> V=del2(U)
V =
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1
>> subplot(1,2,1)
>> surf1(U)
>> subplot(1,2,2)
>> surf1(V)
```

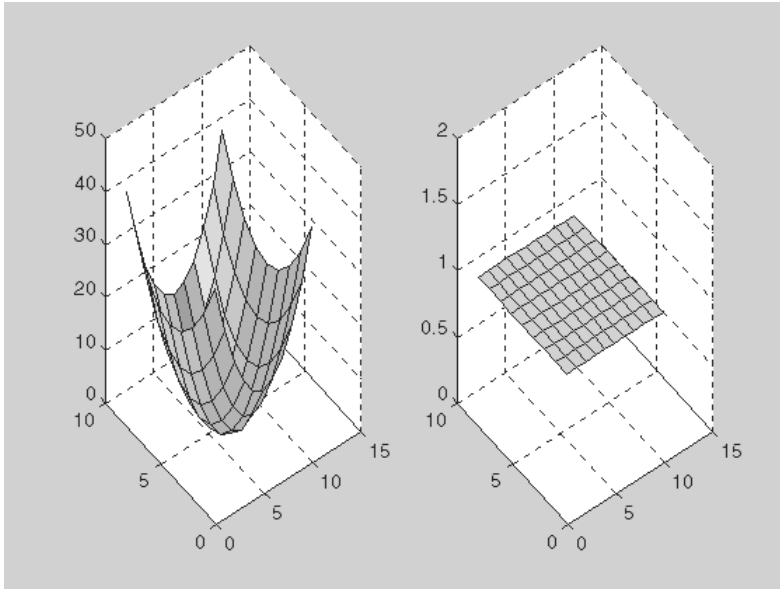
На рис. 8.3 представлены графики поверхностей `U` и `V`.

Эти графики построены завершающими командами данного примера.

8.5.2. Аппроксимация производных конечными разностями

Одним из важнейших приложений конечно-разностных методов является приближенное представление производных функций – *численное дифференцирование*. Оно реализуется следующей функцией:

- `diff(X)` возвращает конечные разности смежных элементов массива `X`. Если `X` – вектор, то `diff(X)` возвращает вектор разностей соседних элементов `[X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)]`, у которого количество элементов на единицу меньше, чем у исходного вектора `X`. Если `X` – матрица, то `diff(X)` возвращает матрицу разностей столбцов: `[X(2:m, :)-X(1:m-1, :)]`;

Рис. 8.3. Графики функций U и V

- $\text{diff}(X, n)$ возвращает конечные разности порядка n . Так, $\text{diff}(X, 2)$ – это то же самое, что и $\text{diff}(\text{diff}(X))$. При вычислениях используется рекуррентное уравнение $\text{diff}(x, n) = \text{diff}(x, n-1)$;
- $Y = \text{diff}(X, n, \text{dim})$ возвращает конечные разности для матрицы X по строкам или по столбцам, в зависимости от значения параметра dim . Если порядок n равен величине dim или превышает ее, то diff возвращает пустой массив.

Примеры:

```
>> X=[1 2 4 6 7 9 3 45 6 7]
X =
     1     2     4     6     7     9     3    45     6     7
>> size(X)
ans = 1    10
>> Y = diff(X)
Y = 1     2     2     1     2    -6    42   -39     1
>> size(Y)
ans = 1     9
>> Y = diff(X,2)
Y = 1     0    -1     1    -8    48   -81    40
>> X=magic(5)
X =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
```

```

    10    12    19    21    3
    11    18    25    2    9
>> Y = diff(X,2)
Y =
    -25    20    0     0     5
     25     5    0    -5   -25
    -5     0    0   -20    25
>> Y = diff(X,2,3)
Y =      Empty array: 5-by-5-by-0

```

Используя функцию `diff`, можно строить графики производных заданной функции. Пример этого показан ниже:

```

>> x=0:0.05:10; S=sin(X);
>> D=diff(S); plot(D/0.05)

```

Для получения приближенных численных значений производной от функции $\sin(x)$ вектор конечно-разностных значений D поделен на шаг точек по x . Как и следовало ожидать, полученный график очень близок к графику функции косинуса (рис. 8.4).

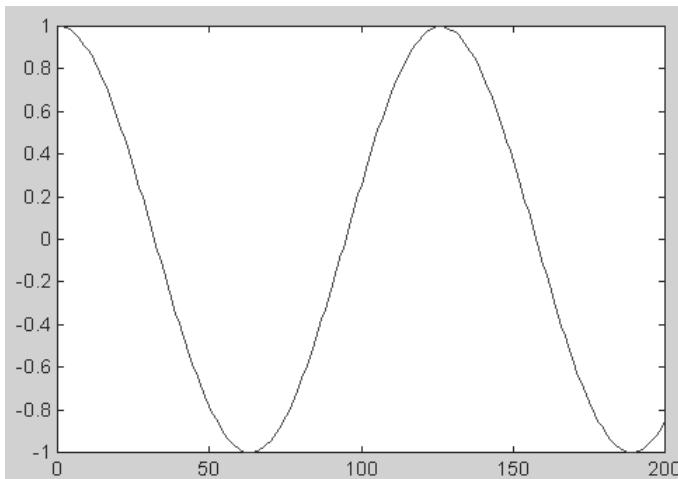


Рис. 8.4. Приближенный график производной от функции $\sin(x)$

Обратите внимание, что по оси x отложены *номера элементов* вектора X , а не истинные значения x .

Пакет расширения Symbolic Math Toolbox (одно из первых подробных его описаний дано в [5] и имеется в [10, 14, 16]) позволяет выполнять дифференцирование функций в аналитическом виде, то есть точно. Это, однако, не всегда нужно.

8.5.3. Вычисление градиента функции

Вычисление конечно-разностным методом *градиента функций* реализуется следующей функцией:

- `FX = gradient (F)` возвращает градиент функции одной переменной, заданной вектором ее значений `F`. `FX` соответствует конечным разностям в направлении `x`;
- `[FX, FY] = gradient (F)` возвращает градиент функции `F (X, Y)` двух переменных, заданной матрицей `F`, в виде массивов `FX` и `FY`. Массив `FX` соответствует конечным разностям в направлении `x` (столбцов). Массив `FY` соответствует конечным разностям в направлении `y` (строк);
- `[FX, FY, FZ, ...] = gradient (F)` возвращает ряд компонентов градиента функции нескольких переменных, заданной в виде многомерного массива `F`;
- `[...] = gradient (F, h)` использует шаг `h` для установки расстояния между точками в каждом направлении (`h` – скалярная величина). По умолчанию `h=1`;
- `[...] = gradient (F, h1, h2, ...)` – если `F` является многомерным массивом, то расстояния задаются с помощью параметров `h1, h2, h3, ...`

Пример:

```
>> F=[1 3 5 7 9 5 6 7 8]
F =      1      3      5      7      9      5      6      7      8
>> FX = gradient(F)
FX =
      Columns 1 through 7
      2.0000 2.0000 2.0000 2.0000 -1.0000      -1.5000
      1.0000
      Columns 8 through 9
      1.0000 1.0000
>> F=[1 2 3 6 7 8;1 4 5 7 9 3;5 9 5 2 5 7]
F =
      1      2      3      6      7      8
      1      4      5      7      9      3
      5      9      5      2      5      7
>> [FX, FY] = gradient(F)
FX =
      1.0000  1.0000  2.0000  2.0000  1.0000  1.0000
      3.0000  2.0000  1.5000  2.0000  -2.0000  -6.0000
      4.0000  0      -3.5000  0      2.5000  2.0000
FY =
      0      2.0000  2.0000  1.0000  2.0000  -5.0000
      2.0000  3.5000  1.0000  -2.0000  -1.0000  -0.5000
      4.0000  5.0000  0      -5.0000  -4.0000  4.0000
```

Функция `gradient` часто используется для построения графиков полей градиентов.

8.6. Численное интегрирование

Численное интегрирование традиционно является одной из важнейших сфер применения математического аппарата. В данном разделе приводятся функции для численного интегрирования различными методами. Численное интегрирование заключается в приближенном вычислении определенного интеграла вида

$$\int_a^b y(x) dx \quad (8.1)$$

одним из многочисленных численных методов [56–58]. Для выполнения аналитического интегрирования можно использовать пакет расширения Symbolic Math Toolbox [5, 10, 14, 16].

8.6.1. Интегрирование методом трапеций

Приведенные ниже функции выполняют численное интегрирование хорошо известным методом *трапеций* и методом *трапеций с накоплением*.

- `trapz(Y)` возвращает определенный интеграл, используя интегрирование методом трапеций с единичным шагом между отсчетами. Если Y – вектор, то `trapz(Y)` возвращает интеграл элементов вектора Y , если Y – матрица, то `trapz(Y)` возвращает вектор-строку, содержащую интегралы каждого столбца этой матрицы.
- `trapz(X, Y)` возвращает интеграл от функции Y по переменной X , используя метод трапеций (пределы интегрирования в этом случае задаются начальным и конечным элементами вектора X).
- `trapz(..., dim)` возвращает интеграл по строкам или по столбцам для входной матрицы, в зависимости от значения переменной `dim`.

Примеры:

```
>> y=[1,2,3,4]
y = 1      2      3      4
>> trapz(y)
ans = 7.5000
>> X=0:pi/70:pi/2; Y=cos(X); Z = trapz(Y)
Z =      22.2780
```

- `cumtrapz(Y)` возвращает численное значение определенного интеграла для функции, заданной ординатами в векторе или матрице Y с шагом интегрирования, равным единице (интегрирование методом трапеций с накоплением). В случае когда шаг отличен от единицы, но постоянен, вычисленный интеграл достаточно умножить на величину шага. Для векторов эта функция возвращает вектор, содержащий результат интегрирования с накоплением элементов вектора Y . Для матриц – возвращает матрицу того же размера, что и Y , содержащую результаты интегрирования с накоплением для каждого столбца матрицы Y .

- `cumtrapz(X, Y)` выполняет интегрирование с накоплением от Y по переменной X , используя метод трапеций. X и Y должны быть векторами одной и той же длины, или X должен быть вектором-столбцом, а Y – матрицей.
- `cumtrapz(..., dim)` выполняет интегрирование с накоплением элементов по размерности, точно определенной скаляром `dim`. Длина вектора X должна быть равна `size(Y, dim)`.

Примеры:

```
>> cumtrapz(y)
ans = 0    1.5000    4.0000    7.5000
>> Y=magic(4)
Y =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> Z = cumtrapz(Y,1)
Z =
     0         0         0         0
 10.5000    6.5000    6.5000   10.5000
 17.5000   15.5000   14.5000   20.5000
 24.0000   26.0000   25.0000   27.0000
```

8.6.2. Интегрирование методом квадратур

Метод трапеций обеспечивает невысокую точность при заданном числе шагов или дает слишком большое число шагов при вычислениях с заданной погрешностью. Приведенные ниже функции осуществляют интегрирование и двойное интегрирование, используя более точную квадратурную *формулу Симпсона* или *метод Гаусса–Лобатто*. Квадратура – численный метод нахождения площади под графиком функции $f(x)$, то есть вычисление определенного интеграла вида (8.1).

В приведенных ниже формулах подынтегральное выражение `fun` обычно задается или в прямых апострофах, или в форме `handle`-функции.

Функции `quad` и `quadl` используют два различных алгоритма квадратуры для вычисления определенного интеграла. Функция `quad` выполняет интегрирование по методу низкого порядка, используя рекурсивное правило Симпсона. Но она может быть более эффективной при негладких подынтегральных функциях или при низкой требуемой точности вычислений. Алгоритм `quad` в MATLAB 6 изменен по сравнению с предшествовавшими версиями, точность по умолчанию по сравнению с версиями 5.3x повышена в 1000 раз (с 10^{-3} до 10^{-6}).

Новая функция `quadl` (квадратура Лобатто) использует адаптивное правило квадратуры Гаусса–Лобатто очень высокого порядка. Устаревшая функция `quad8` выполняла интегрирование, используя квадратурные формулы Ньютона–Котеса 8-го порядка [53]. Достижимая точность интегрирования гладких функций в MATLAB 6 поэтому также значительно выше, чем в предшествующих версиях.

- `quad(fun, a, b)` возвращает численное значение определенного интеграла от заданной функции `@fun` на отрезке `[a b]`. Используется значительно усовершенствованный в MATLAB 6 адаптивный метод Симпсона.
- `quad(fun, a, b, tol)` возвращает численное значение определенного интеграла с заданной относительной погрешностью `tol`. По умолчанию `tol=1.e-6`. Можно также использовать вектор, состоящий из двух элементов `tol=[rel_tol abs_tol]`, чтобы точно определить комбинацию относительной и абсолютной погрешностей.
- `quad(fun, a, b, tol, trace)` возвращает численное значение определенного интеграла и при значении `trace`, не равном нулю, строит график, показывающий ход вычисления интеграла.
- `quad(fun, a, b, tol, trace, P1, P2, ...)` возвращает численное значение определенного интеграла по подынтегральной функции `fun`, использует дополнительные аргументы `P1, P2, ...`, которые напрямую передаются в подынтегральную функцию: `G=fun(X, P1, P2, ...)`.

Примеры:

```
>> quad('exp(x)-1', 0, 1, 1e-5)
ans = 0.7183
>> q = quad(@exp, 0, 2, 1e-4)
q = 6.3891
>> q = quad(@sin, 0, pi, 1e-3)
q = 2.0000
```

8.6.3. Вычисления двойных и тройных интегралов

Для вычисления двойных интегралов служит следующая функция:

- `dblquad(fun, inmin, inmax, outmin, outmax)` вычисляет и возвращает значение двойного интеграла для подынтегральной функции `fun(inner, outer)`, по умолчанию используя квадратурную функцию `quad`. `inner` – внутренняя переменная, изменяющаяся от `inmin` до `inmax`, а `outer` – внешняя переменная, изменяющаяся от `outmin` до `outmax`. Первый аргумент `@fun` – строка, описывающая подынтегральную функцию. Эта функция должна быть функцией двух переменных вида `fout=fun(inner, outer)`. Функция должна брать вектор `inner` и скаляр `outer` и возвращать вектор `fout`, который является функцией, вычисленной в `outer` и в каждом значении `inner`.
- `dblquad(fun, inmin, inmax, outmin, outmax, tol, trace)` передает в функцию `dblquad` параметры `tol` и `trace`. Смотрите справку по функции `quad` для получения информации о параметрах `tol` и `trace`.
- `dblquad(fun, inmin, inmax, outmin, outmax, tol, trace, order)` передает параметры `tol` и `trace` для функции `quad` или `quadl` в зависимости от значения строки `order`. Допустимые значения для параметра

`order` – `@quad`, `@quadl` или имя любого определенного пользователем квадратурного метода с таким же вызовом и такими же возвращаемыми параметрами, как у функций `quad` и `quadl`. (Например, при проверке старых программ можно использовать `@quad8` для большей совместимости с прежними версиями MATLAB). По умолчанию (без параметра `order`) вызывается `@quad`, поскольку подынтегральные функции могут быть негладкими.

Пример: пусть `m`-файл `integ1.m` описывает функцию $2*y*\sin(x) + x/2*\cos(y)$, тогда вычислить двойной интеграл от этой функции можно следующим образом:

```
>> result = dblquad(@integ1,pi,2*pi,0,2*pi)
result = -78.9574
```

В систему MATLAB 6.5 была введена новая функция для вычисления тройных интегралов. Она имеет четыре формы записи:

```
triplequad(fun, xmin, xmax, ymin, ymax, zmin, zmax)
triplequad(fun, xmin, xmax, ymin, ymax, zmin, zmax, tol)
triplequad(fun, xmin, xmax, ymin, ymax, zmin, zmax, tol, method)
triplequad(fun, xmin, xmax, ymin, ymax, zmin, zmax, tol, method, p1, p2, ...)
```

Параметры этой функции уже были определены выше (добавлены только пределы для новой переменной `z`). Пример:

```
>> Q = triplequad('x+y*z', 0, pi, 0, 1, -1, 1, 0.001)
Q = 9.8696
```

К сожалению, проверка данной функции на ряде тройных интегралов показала, что она не всегда обеспечивает вычисления. Возможны ситуации, когда данная функция выводит ряд сообщений об ошибках даже для берущихся интегралов.

8.7. Математические операции с полиномами

8.7.1. Определение полиномов

Полиномы (у нас их принято называть также *степенными многочленами*) – широко известный объект математических вычислений и обработки данных. В MATLAB полином записывается в виде

$$p(x) = a_1x^n + a_2x^{n-1} + \dots + a_r x + a_{n+1}.$$

В дальнейшем мы ограничимся рассмотрением полиномов с целыми положительными степенями.

Широкое применение полиномов отчасти обусловлено большими возможностями полиномов в представлении данных, а также их простотой и единообразием вычислений. В этом разделе описаны основные функции для работы с полиномами. При этом полиномы обычно задаются векторами их коэффициентов.

8.7.2. Умножение и деление полиномов

Ниже приведены функции, осуществляющие умножение и деление полиномов, или, что то же самое, свертку двух входных векторов, в которых находятся коэффициенты полиномов, и операцию, обратную свертке.

- $w = \text{conv}(u, v)$ возвращает свертку векторов u и v . Алгебраически свертка – то же самое, что и произведение полиномов, чьи коэффициенты – элементы векторов u и v . Если длина вектора u равна m , а длина вектора v – n , то вектор w имеет длину $m+n-1$, а его k -ый элемент вычисляется по формуле $w(k) = \sum_j u(j)v(k+1-j)$.

Пример:

```
>> f=[2,3,5,6];d=[7,8,3];r=conv(f,d)
r =      14      37      65      91      63      18
```

- $[q, r] = \text{deconv}(v, u)$ возвращает результат деления полинома v на полином u . Вектор q представляет собой частное от деления, а r – остаток от деления, так что выполняется соотношение $v = \text{conv}(u, q) + r$. Пример:

```
>> t=[14,37,65,91,63,18];r=[7,8,3];[w,e]=deconv(t,r)
w =      2.0000      3.0000      5.0000      6.0000
e =      1.0e-013
      0      0      0.1421      -0.1421      -0.2132      -0.1066
```

8.7.3. Вычисление полиномов

В этом разделе приведены функции вычисления коэффициентов характеристического полинома, значения полинома в точке и матричного полинома.

- $\text{poly}(A)$ для квадратной матрицы A размера $n \times n$ возвращает вектор-строку размером $n+1$, элементы которой являются коэффициентами характеристического полинома $\det(A-sI)$, где I – единичная матрица и s – оператор Лапласа. Коэффициенты упорядочены по убыванию степеней. Если вектор состоит из $n+1$ компонентов, то ему соответствует полином вида $c_1s^n + \dots + c_n s + c_{n+1}$.
- $\text{poly}(r)$ для вектора r возвращает вектор-строку p с элементами, представляющими собой коэффициенты полинома, корнями которого являются элементы вектора r . Функция $\text{roots}(p)$ является обратной, ее результаты, умноженные на целое число, дают $\text{poly}(r)$.

Примеры:

```
A =
      2      3      6
      3      8      6
      1      7      4

>> d=poly(A)
d =      1.0000     -14.0000     -1.0000     -40.0000
>> A=[3,6,8;12,23,5;11,12,32]
```

```
A =
      3      6      8
     12     23     5
     11     12    32
>> poly(A)
ans = 1.0000 -58.0000 681.0000 818.0000
```

- `polyval(p, x)` возвращает значения полинома `p`, вычисленные в точках, заданных в массиве `x`. Полином `p` – вектор, элементы которого являются коэффициентами полинома в порядке уменьшения степеней. `x` может быть матрицей или вектором. В любом случае функция `polyval` вычисляет значения полинома `p` для каждого элемента `x`.
- `[y, delta] = polyval(p, x, S)` или `[y, delta] = polyval(p, x, S, mu)` – использует структуру `S`, возвращенную функцией `polyfit`, и данные о среднем значении (`mu(1)`) и стандартном отклонении (`mu(2)`) генеральной совокупности для оценки погрешности аппроксимации (`y±delta`).

Пример:

```
>> p=[3,0,4,3];d=polyval(p,[2,6])
d =      35      675
```

- `polyvalm(p, X)` вычисляет значения полинома для матрицы. Это эквивалентно подстановке матрицы `X` в полином `p`. Полином `p` – вектор, чьи элементы являются коэффициентами полинома в порядке уменьшения степеней, а `X` – квадратная матрица. Пример:

```
>> D=pascal(5)
D =
      1      1      1      1      1
      1      2      3      4      5
      1      3      6     10     15
      1      4     10     20     35
      1      5     15     35     70
>> f=poly(d)
f =
      1.0000 -99.0000 626.0000 -626.0000 99.0000
     -1.0000
>> polyvalm(f,D)
ans =
      1.0e-006*
     -0.0003  -0.0011  -0.0038  -0.0059  -0.0162
     -0.0012  -0.0048  -0.0163  -0.0253  -0.0692
     -0.0034  -0.0131  -0.0447  -0.0696  -0.1897
     -0.0076  -0.0288  -0.0983  -0.1529  -0.4169
     -0.0145  -0.0551  -0.1883  -0.2929  -0.7984
```

Данный пример иллюстрирует также погрешности численных методов, поскольку точное решение дает нулевую матрицу.

8.7.4. Вычисление корней полинома

Приведенная ниже функция вычисляет корни (в том числе комплексные) для полинома вида $c_1s^n + \dots + c_n s + c_{n+1}$.

- `roots(c)` – возвращает вектор-столбец, чьи элементы являются корнями полинома c .

Вектор-строка c содержит коэффициенты полинома, упорядоченные по убыванию степеней. Если c имеет $n+1$ компонент, то полином, представленный этим вектором, имеет вид $c_1s^n + \dots + c_n s + c_{n+1}$.

Вначале рассмотрим простой пример – решение полиномиального уравнения $x^2 - x - 1 = 0$. Как известно, один из корней этой системы дает значение константы золотого сечения. Решение этого уравнения вполне очевидно:

```
>> p=[1 -1 -1];
>> roots(p)
ans =
    -0.6180
     1.6180
```

Полезно отметить, что пакет символьных вычислений, который поставляется отдельно, дает это решение с помощью функции `solve` в более естественном виде:

```
>> solve('x^2-x-1=0')
ans =
 1/2*5^(1/2)+1/2
 1/2-1/2*5^(1/2)
```

Нетрудно убедиться, что и в этом случае корни уравнения соответствуют ранее найденным значениям:

```
>> 1/2*5^(1/2)+1/2
ans =
     1.6180
>> 1/2-1/2*5^(1/2)
ans =
    -0.6180
```

Рассмотрим еще один пример решения полиномиального уравнения, дающий решение в виде комплексных чисел:

```
>> x=[7, 45, 12, 23]; d=roots(x)
d =
    -6.2382
    -0.0952+0.7195i
    -0.0952 -0.7195i
A=[-6.2382    -0.0952+0.7195i    -0.0952 -0.7195i];
V=Poly(A)
V=[1.0000    6.4286    1.7145    3.2859]
V*7
ans = 7.0000    45.0002    12.0015    23.0013
```

С погрешностью округления получен тот же вектор.

8.7.5. Вычисление производной полинома

Ниже приведена функция, возвращающая *производную полинома* p :

- `polyder(p)` возвращает производную полинома p ;
- `polyder(a,b)` возвращает производную от произведения полиномов a и b ;
- `[q,d]=polyder(b,a)` возвращает числитель q и знаменатель d производной от отношения полиномов b/a .

Примеры:

```
>> a=[3,5,8];b=[5,3,8];dp=polyder(a)
dp = 6 5
>> dt=polyder(a,b)
dt = 60 102 158 64
>> [q,p]=polyder(b,a)
q = 16 32 -16
p = 9 30 73 80 64
```

8.7.6. Решение полиномиальных матричных уравнений

Приведенная ниже функция вычисляет собственные значения матричного полинома.

- `[X,e]=polyeig(A0,A1,...Ap)` решает задачу собственных значений для матричного полинома степени p вида

$$(A_0 + \lambda A_1 + \dots + \lambda^p A_p)x = 0,$$

где степень полинома p – целое неотрицательное число, а A_0, A_1, \dots, A_p – входные матрицы порядка n . Выходная матрица X размера $n \times np$ содержит собственные векторы в столбцах. Вектор e размером np содержит собственные значения. Пример:

```
>> A=[1:4;5:8;9:12;13:16]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
>> B=[4:7;2:5;10:13;23:26]
B =
     4     5     6     7
     2     3     4     5
    10    11    12    13
    23    24    25    26
>> [F,a]=polyeig(A,B)
F =
     0.4373     0.0689    -0.5426    -0.7594
    -0.3372    -0.4969     0.6675    -0.1314
```

```

-0.6375      0.7870      0.2927      0.3771
 0.5374     -0.3591     -0.4176     0.5136
a =
 4.4048
 0.4425
-0.3229
-1.0000

```

8.7.7. Разложение полиномов на простые дроби

Для отношения полиномов b и a функция

```
[r, p, k] = residue(b, a)
```

возвращает вычеты, полюса и многочлен целой части отношения двух полиномов $b(s)$ и $a(s)$ в виде

$$\frac{b(s)}{a(s)} = \frac{b_1 + b_2 s^{-1} + b_3 s^{-2} + \dots + b_{m+1} s^{-m}}{a_1 + a_2 s^{-1} + a_3 s^{-2} + \dots + a_{n+1} s^{-n}}.$$

- $[b, a] = \text{residue}(r, p, k)$ выполняет обратную свертку суммы простых дробей (см. более подробное описание в справочной системе) в пару полиномов с коэффициентами в векторах b и a . Пример:

```

>> b=[4, 3, 1]; a=[1, 3, 7, 1]; [r, p, k]=residue(b, a)
r =
    1.9484 + 0.8064i
    1.9484 - 0.8064i
    0.1033
p =
   -1.4239 + 2.1305i
   -1.4239 - 2.1305i
   -0.1523
k = [ ]
>> [b1, a1]=residue(r, p, k)
b1 = 4.0000    3.0000    1.0000
a1 = 1.0000    3.0000    7.0000    1.0000

```

8.8. Обыкновенные дифференциальные уравнения (ОДУ)

8.8.1. Определение ОДУ

Анализ поведения многих систем и устройств в динамике, а также решение многих задач в теории колебаний и в поведении упругих оболочек обычно базируется на решении систем *обыкновенных дифференциальных уравнений* (ОДУ), или,

в оригинале, *ordinary differential equations* (ODEs). Их, как правило, представляют в виде системы из дифференциальных уравнений первого порядка в форме Коши [51, 52]:

$$\frac{dy}{dt} = y' = f(y, t) \quad (8.1)$$

с граничными условиями $y(t_0, t_{\text{end}}, p) = b$, где t_{end}, t_0 – начальные и конечные точки интервалов. Параметр t (независимая переменная) необязательно означает время, хотя чаще всего решение дифференциальных уравнений ищется во временной области. Система дифференциальных уравнений в форме Коши записывается аналогично (8.1), но под y в этом случае подразумевается вектор-столбец зависимых переменных. Вектор p задает начальные условия.

Для решения дифференциальных уравнений второго и высшего порядка их нужно свести к системе дифференциальных уравнений первого порядка. Как это делается, хорошо известно (см. примеры ниже).

Возможны дифференциальные уравнения, не разрешенные относительно производной:

$$F(t, y, dy/dt) = 0. \quad (8.2)$$

Уравнения (8.2) аналитически к форме (8.1) обычно привести не удастся. Однако численное решение особых трудностей не вызывает – достаточно для определения $f(y, t)$ решить (8.2) численно относительно производной при заданных y и t .

Наряду с ОДУ MATLAB может оперировать с *дифференциальными алгебраическими уравнениями* (ДАУ, или *differential-algebraic equations* – DAEs). ОДУ и ДАУ являются основой математического моделирования динамических нелинейных (и линейных) систем. Автоматическое их составление и решение реализованы в специальном расширении Simulink.

Ниже коротко описаны численные методы решения обыкновенных дифференциальных уравнений (ОДУ) и некоторые вспомогательные функции, полезные для решения систем ОДУ. Дается представление о пакете расширения, решающем дифференциальные уравнения в частных производных.

8.8.2. Решатели ОДУ

Для решения систем ОДУ в MATLAB реализованы различные численные методы. Их реализации названы *решателями* ОДУ. Подробное описание решателей можно найти книге класса textbook [49], которую можно получить из Интернета [67, 68]. Полезные материалы есть также в книгах [27–30].

Внимание!

В этом разделе обобщенное название *solver* (решатель) означает один из возможных численных методов решения ОДУ: *ode45*, *ode23*, *ode113*, *ode15s*, *ode23s*, *ode23t*, *ode23tb*, *bvp4c* или *pdepe*.

Решатели реализуют следующие методы решения систем дифференциальных уравнений:

- `ode45` – одношаговые явные методы Рунге-Кутты 4-го и 5-го порядков в модификации Дорманда и Принца. Это классический метод, рекомендуемый для начальной пробы решения. Во многих случаях он дает хорошие результаты – если система решаемых уравнений не жесткая.
- `ode23` – одношаговые явные методы Рунге-Кутты 2-го и 4-го порядков в модификации Богацки и Шампина. При умеренной жесткости системы ОДУ и низких требованиях к точности этот метод может дать выигрыш в скорости решения.
- `ode113` – многошаговый метод Адамса–Башворта–Мултона переменного порядка класса предиктор–корректор. Это адаптивный метод, который может обеспечить высокую точность решения.
- `ode15s` – многошаговый метод переменного порядка (от 1 до 5, по умолчанию 5), использующий формулы численного «дифференцирования назад». Это адаптивный метод, его стоит применять, если решатель `ode45` не обеспечивает решения и система дифференциальных уравнений жесткая.
- `ode23s` – одношаговый метод, использующий модифицированную формулу Розенброка 2-го порядка. Может обеспечить высокую скорость вычислений при низкой точности решения жесткой системы дифференциальных уравнений.
- `ode23t` – неявный метод трапеций с интерполяцией. Этот метод дает хорошие результаты при решении задач, описывающих колебательные системы с почти гармоническим выходным сигналом. При умеренно жестких системах дифференциальных уравнений может дать высокую точность решения.
- `ode23tb` – неявный метод Рунге–Кутта в начале решения и метод, использующий формулы «дифференцирования назад» 2-го порядка в последующем. Несмотря на сравнительно низкую точность, этот метод может оказаться более эффективным, чем `ode15s`.
- `bvp4c` – служит для проблемы граничных значений систем дифференциальных уравнений вида $y' = f(t, y)$, $F(y(a), y(b), p) = 0$ (полная форма системы уравнений Коши). Решаемые им задачи называют двухточечными краевыми задачами, поскольку решение ищется при задании граничных условий как в начале, так и в конце интервала решения.
- `pdere` – служит для решения систем параболических и эллиптических дифференциальных уравнений в частных производных. Этот решатель введен в ядро системы для поддержки новых графических функций OpenGL. Пакет расширения Partial Differential Equations Toolbox содержит более мощные средства для решения дифференциальных уравнений этого класса.

Все решатели могут решать системы уравнений явного вида $y' = F(t, y)$, причем для решения жестких систем уравнений рекомендуется использовать только специальные решатели `ode15s`, `ode23s`, `ode23t`, `ode23tb`.

Решатели `ode15s` и `ode23t` способны найти корни дифференциально-алгебраических уравнений $M(t)y' = F(t, y)$, где M называется матрицей массы. Решатели

ode15s, ode23s, ode23t и ode23tb могут решать уравнения неявного вида $M(t, y) y' = F(t, y)$. И наконец, все решатели, за исключением ode23s, который требует постоянства матрицы массы, и bvp4c, могут находить корни матричного уравнения вида $M(t, y) y' = F(t, y)$. ode23tb, ode23s служат для решения жестких дифференциальных уравнений, ode15s – жестких дифференциальных и дифференциально-алгебраических уравнений, ode23t – умеренно жестких дифференциальных и дифференциально-алгебраических уравнений.

8.8.3. Использование решателей систем ОДУ

В описанных далее функциях для решения систем дифференциальных уравнений приняты следующие обозначения и правила:

- `tspan` – вектор, определяющий интервал интегрирования $[t_0 \ t_{final}]$. Для получения решений в конкретные моменты времени $t_0, t_1, \dots, t_{final}$ (расположенные в порядке уменьшения или увеличения) нужно использовать `tspan = [t0 t1 ... tfinal]`;
- `y0` – вектор начальных условий;
- `options` – аргумент, создаваемый функцией `odeset` (еще одна функция – `odeget` или `bvpget` (только для `bvp4c`) – позволяет вывести параметры, установленные по умолчанию или с помощью функции `odeset /bvpset`);
- `p1, p2, ...` – произвольные параметры, передаваемые в функцию `F`;
- `T, Y` – матрица решений `Y`, где каждая строка соответствует времени, возвращенном в векторе-столбце `T`.

Перейдем к описанию синтаксиса функций для решения систем дифференциальных уравнений (под именем `solver` подразумевается любая из представленных выше функций).

- `[T, Y]=solver(@F, tspan, y0)` интегрирует систему дифференциальных уравнений вида $y' = F(t, y)$ на интервале `tspan` с начальными условиями `y0`. `@F` – дескриптор ODE-функции (можно также задавать функцию в виде 'F'). Каждая строка в массиве решений `Y` соответствует значению времени, возвращаемому в векторе-столбце `T`.
- `[T, Y]=solver(@F, tspan, y0, options)` дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента `options`, созданного функцией `odeset`. Обычно используемые параметры включают допустимое значение относительной погрешности `RelTol` (по умолчанию $1e-3$) и вектор допустимых значений абсолютной погрешности `AbsTol` (все компоненты по умолчанию равны $1e-6$).
- `[T, Y]=solver(@F, tspan, y0, options, p1, p2...)` дает решение, подобное описанному выше, передавая дополнительные параметры `p1, p2, ...` в м-файл `F` всякий раз, когда он вызывается. Используйте `options=[]`, если никакие параметры не задаются.
- `[T, Y, TE, YE, IE] = solver(@F, tspan, y0, options)` в дополнение к описанному решению содержит свойства `Events`, установленные в структуре `options` ссылкой на функции событий. Когда эти функции событий

от (t, y) равны нулю, производятся действия в зависимости от значения трех векторов `value`, `isterminal`, `direction` (их величины можно установить в `m`-файлах функций событий). Для i -й функции событий `value(i)` – значение функции, `isterminal(i)` – прекратить интеграцию при достижении функцией нулевого значения, `direction(i) = 0`, если все нули функции событий нужно вычислять (по умолчанию), `+1` – только те нули, где функция событий увеличивается, `-1` – только те нули, где функция событий уменьшается. Выходной аргумент `TE` – вектор-столбец времен, в которые происходят события (`events`), строки `YE` являются соответствующими решениями, а индексы в векторе `IE` определяют, какая из i функций событий (`event`) равна нулю в момент времени, определенный `TE`. Когда происходит вызов функции без выходных аргументов, по умолчанию вызывается выходная функция `odeplot` для построения вычисленного решения. В качестве альтернативы можно, например, установить свойство `OutputFcn` в значении `'odephas2'` или `'odephas3'` для построения двумерных или трехмерных фазовых плоскостей.

- `[T, X, Y]=solver(@model, tspan, y0, options, ut, p1, p2, ...)` использует модель SIMULINK, вызывая соответствующий решатель из нее. Пример: `[T, X, Y] = sim(solver, @model, ...)`.

Необходимая для решения система задается в виде `@F` или `@model`, характерном для `handle` (анонимных) функций. Параметры интегрирования (`options`) могут быть определены и в `m`-файле, и в командной строке с помощью команды `odeset`. Если параметр определен в обоих местах, определение в командной строке имеет приоритет.

Решатели используют в списке параметров различные параметры:

- `RelTol` – относительный порог отбора (положительный скаляр). По умолчанию `1e-3` (0.1% точность) во всех решателях; оценка ошибки на каждом шаге интеграции $e(i) \leq \max(\text{RelTol} * \text{abs}(y(i)), \text{AbsTol}(i))$;
- `AbsTol` – абсолютная точность (положительный скаляр или вектор `{1e-6}`). Скаляр вводится для всех составляющих вектора решения, а вектор указывает на компоненты вектора решения. `AbsTol` по умолчанию `1e-6` во всех решателях;
- `NormControl` – управление ошибкой в зависимости от нормы вектора решения `[on | off]`. Установите «on», чтобы $\text{norm}(e) \leq \max(\text{RelTol} * \text{norm}(y), \text{AbsTol})$. По умолчанию все решатели используют более жесткое управление по каждой из составляющих вектора решения;
- `Refine` – фактор уточнения вывода (положительное целое число) – умножает число точек вывода на этот множитель. По умолчанию всегда равен 1, кроме ODE45, где он 4. Не применяется, если `tspan > 2`;
- `OutputFcn` – дескриптор функция вывода `[function]` – имеет значение в том случае, если решатель вызывается без явного указания функции вывода, `OutputFcn` по умолчанию вызывает функцию `odeplot`. Эту установку можно поменять именно здесь;

- `OutputSel` – индексы отбора (вектор целых чисел). Установите компоненты, которые поступают в `OutputFcn`. `OutputSel` по умолчанию выводит все компоненты;
- `Stats` – установите статистику стоимости вычислений [`on` | `{off}`];
- `Jacobian` – функция матрицы Якоби [`function`|`constant matrix`]. Установите это свойство на дескриптор функции `FJas` (если `FJas(t,y)` возвращает dF/dy) или на имя постоянной матрицы dF/dy ;
- `Jpattern` – график разреженности матрицы Якоби (имя разреженной матрицы). Матрица S с $S(i, j) = 1$, если составляющая i $F(t,y)$ зависит от составляющей j величины y , и 0 в противоположном случае;
- `Vectorized` – векторизованная ODE-функция [`on` | `{off}`]. Устанавливается на «`on`», если ODE-функция `F(t,[y1 y2 ...])` возвращает вектор [`F(t,y1)` `F(t,y2)` ...];
- `Events` – [`function`] – введите дескрипторы функций событий, содержащих собственно функцию в векторе `value` и векторы `isterminal` и `direction` (см. выше);
- `Mass` – матрица массы [`constant matrix` | `function`]. Для задач $M^*y' = f(t,y)$ установите имя постоянной матрицы. Для задач с переменной M введите дескриптор функции, описывающей матрицу массы;
- `MstateDependence` – зависимость матрицы массы от y [`none` | `{weak}` | `strong`] – установите «`none`» для уравнений $M(t)^*y' = F(t,y)$. И слабая («`weak`»), и сильная («`strong`») зависимости означают $M(t,y)$, но «`weak`» приводит к неявным алгоритмам решения, использующим аппроксимации при решении алгебраических уравнений;
- `MassSingular` – матрица массы M сингулярная [`yes` | `no` | `{maybe}`] (да/нет/может быть);
- `MvPattern` – разреженность (dMv/dy), график разреженности (см. функцию `sru`) – введите имя разреженной матрицы S с $S(i,j) = 1$ для любого k , где (i,k) элемент матрицы массы $M(t,y)$ зависит от проекции j переменной y , и 0 в противном случае;
- `InitialSlope` – вектор начального уклона $yp0$ $yp0 = F(t0,y0)/M(t0,y0)$;
- `InitialStep` – предлагаемый начальный размер шага, по умолчанию каждый решатель определяет его автоматически по своему алгоритму;
- `MaxStep` – максимальный шаг, по умолчанию во всех решателях равен одной десятой интервала `tspan`;
- `BDF (Backward Differentiation Formulas)`[`on`|`{off}`] – указывает, нужно ли использовать формулы обратного дифференцирования (методы `Gear`) вместо формул численного дифференцирования, используемых в `ode15s` по умолчанию;
- `MaxOrder` – максимальный порядок `ode15S` [`1` | `2` | `3` | `4` | `{5}`].

Решатели используют в списке опций различные параметры. В приведенной ниже таблице они даны для решателей обычных (в том числе жестких) дифференциальных уравнений.

| Параметры | ode45 | ode23 | ode113 | ode15s | ode23s |
|---|-------|-------|--------|--------|--------|
| RelTol, AbsTol | + | + | + | + | + |
| OutputFcn, OutputSel, Refine, Stats | + | + | + | + | + |
| Events | + | + | + | + | + |
| MaxStep, InitialStep | + | + | + | + | + |
| Jconstant, Jacobian, Jpattern, Vectorized | - | - | - | + | + |
| Mass | - | - | - | + | + |
| MassConstant | - | - | - | + | - |
| MaxOrder, BDF | - | - | - | + | - |

Решатель `bvp4c` имеет очень небольшое число параметров, но при работе с ним можно вводить не только матрицу Якоби интегрируемой функции, но и матрицу Якоби, содержащую частные производные функции граничных условий по границам интервала и по неизвестным параметрам.

Решатели позволяют строить графики решений как в виде обычных графиков (например, временных зависимостей), так и в виде *фазовых портретов* – это параметрические графики, у которых по одной оси указывается одна из зависимостей, а по другой – ее производная. К примеру, фазовый портрет стационарного синусоидального колебания представляет собой эллипс или окружность, которые являются *предельными циклами*. Отклонение от этих простых форм указывает на наличие нелинейности. Затухающие или нарастающие колебания дают фазовые портреты в виде закручивающейся или раскручивающейся спирали. Рассмотрение более тонких деталей фазовых портретов выходит за рамки данного учебного курса.

8.9. Примеры решения дифференциальных уравнений

8.9.1. Пример на движение брошенного вверх тела

Простейшее дифференциальное уравнение $y''(t) = -g$ описывает свободное падение тела в гравитационном поле Земли. Константа гравитации $g = 9,8$. Найдем решение этого уравнения в безразмерном виде, хотя на самом деле единицами измерения будут секунды для времени и метры для высоты. Пусть высота полета будет $y_1 = y$, а скорость $y_2 = y'$. Начальные условия следует задать в виде $y_0 = [0; 10]$ – это значит, что начальная высота тела равна 0, а скорость полета в момент его бросания вверх равна 10. Решается задача на движение брошенного вверх тела без учета сопротивления воздуха.

Для решения задачи составим простой скрипт-файл с именем `demoode`:

```
y0=[0; 10] % Задание вектора начальных условий
ts=0:.2:2; % Задание сетки времен
dydt=@(t,y)[y(2);-9.8]; % Задание анонимной функции правых частей ODE
```

```
[to, yo]=ode45(dydt, ts, y0) % Решение решателем ode45
```

Остается пустить пример исполнением команды с именем файла:

```
>> demoode
y0 =
     0
    10
to =
     0
    0.2000
    0.4000
    0.6000
    0.8000
    1.0000
    1.2000
    1.4000
    1.6000
    1.8000
    2.0000
yo =
     0    10.0000
    1.8040    8.0400
    3.2160    6.0800
    4.2360    4.1200
    4.8640    2.1600
    5.1000    0.2000
    4.9440   -1.7600
    4.3960   -3.7200
    3.4560   -5.6800
    2.1240   -7.6400
    0.4000   -9.6000
```

Результат вполне очевиден – выдаются векторы-столбцы начальных условий и времени и матрица значений высот тела и скоростей (положительная скорость означает подъем тела, отрицательная – его падение). Представление результатов решения систем ODE в графическом виде будет дано в следующих примерах.

8.9.2. Примеры решения системы ОДУ Ван-дер-Поля

Покажем применение решателя ОДУ `ode15s` на ставшем классическим примере – решении нелинейного дифференциального уравнения второго порядка (уравнения Ван-дер-Поля), записанного в виде системы из двух дифференциальных уравнений:

$$\begin{aligned} y_1' &= y_2; \\ y_2' &= m*(1-y_1^2)*y_2-y_1 \end{aligned}$$

при начальных условиях

$$y_1(0) = 0; y_2(0) = 1.$$

Это уравнение описывает колебания в нелинейной системе второго порядка, например в *LC*-генераторе на электронной лампе или полевом транзисторе, и является классическим примером математического моделирования этих устройств. Поведение системы Ван-дер-Поля существенно зависит от параметра m , который задает степень влияния нелинейности на возникновение и развитие колебаний. При больших m представленная система ОДУ является жесткой. Возьмем значение $m=100$.

Перед решением нужно записать систему дифференциальных уравнений в виде ODE-функции. Для этого в главном меню выберем **File** \Rightarrow **New** \Rightarrow **M-File** и введем

```
function dydt = vdp100(t,y)
dydt = zeros(2,1); dydt(1) = y(2);
dydt(2) = 100*(1 - y(1)^2)*y(2) - y(1);
```

Сохраним данный *m*-файл-функцию.

Тогда решение решателем `ode15s` и сопровождающий его график (рис. 8.5) можно получить, используя следующие команды:

```
>> [T,Y]=ode15s(@vdp100,[0 30],[2 0]); plot(T,Y)
>> hold on; gtext('y1'), gtext('y2')
```

Последние команды позволяют с помощью мыши нанести на графики решений $y_1 = y(1)$ и $y_2 = y(2)$ помечающие их надписи.

Рассмотрим еще один пример решения уравнения Ван-дер-Поля вида $y''_1 = 2*(1-y_1^2)*y_1 - y'_1$ при $m=2$. Оно сводится к следующей системе уравнений:

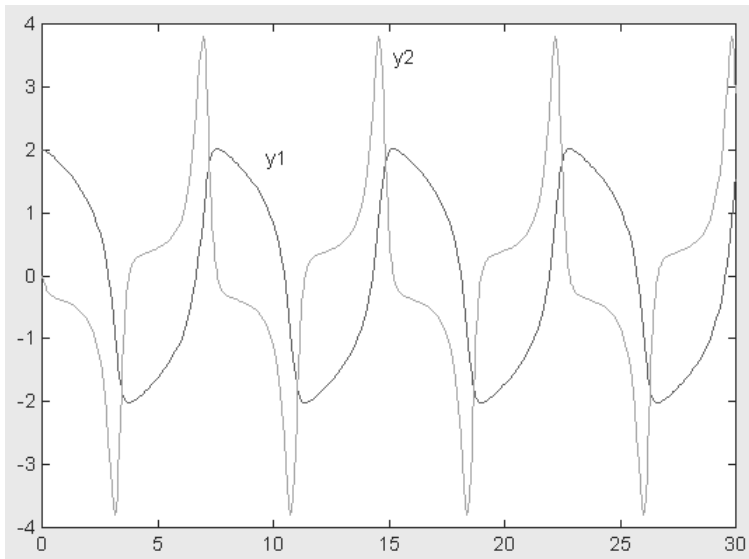


Рис. 8.5. Пример решения системы дифференциальных уравнений численным методом

$$y_1' = y_2,$$

$$y_2' = 2*(1-y_1^2)*y_1 - y_2.$$

Подготовим m-файл ode-функции **vdp.m**:

```
function [out1,out2,out3] = vdp(t,y,flag)
if nargin < 3 | isempty(flag)
    out1 = [2.*y(2).*(1-y(2).^2)-y(1); y(1)];
else
    switch(flag)
    case 'init' % Return tspan, y0 and options
        out1 = [0 20];out2 = [2; 0]; out3 = [ ];
    otherwise
        error(['Unknown request '' flag ''.']);
    end
end
```

Тогда решение системы с помощью решателя ode23 реализуется следующими командами:

```
>> [T,Y] = ode23(@vdp,[0 20],[2 0]);
>> plot(T,Y(:,1),'-',T,Y(:,2),'-.')
```

График решения для последнего примера показан на рис. 8.6. Нетрудно заметить, что в данном случае переходные процессы имеют затухающий характер и генерации периодических колебаний нет.

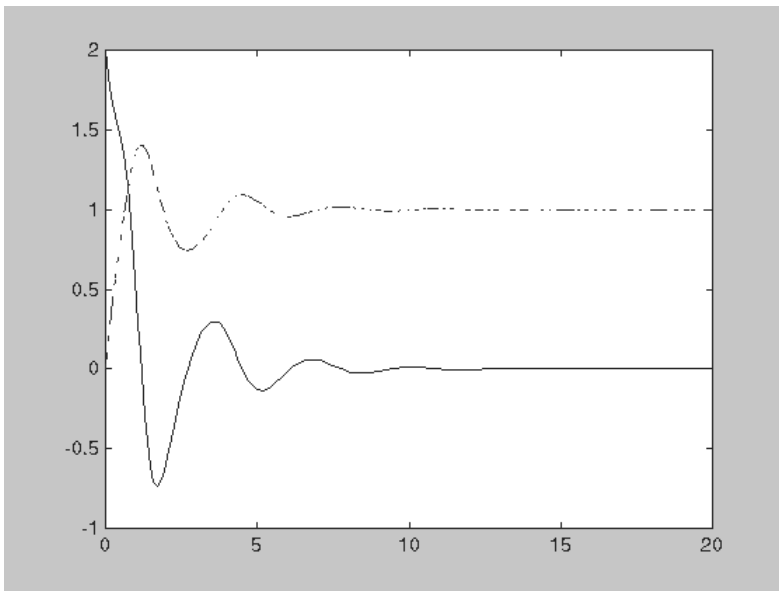


Рис. 8.6. Пример решения системы ОДУ

8.9.3. Вычисление реакции системы второго порядка на заданное воздействие

Дифференциальное уравнение второго порядка $y'' + ay' + by = e(t)$ описывает систему второго порядка, находящуюся под внешним воздействием $e(t)$. Рассмотрим численное решение этого уравнения при $a = 0,5$, $b = 5$ и $e(t) = e^{-t} \sin(2t)$ и начальных условиях 1 и 0. Для выполнения решения введем $y_1 = y$ и $y_2 = y'$. Тогда уравнение второго порядка можно свести к следующей системе из двух уравнений:

$$\left\{ \begin{array}{l} \dot{y}_1 = y_2 \\ \dot{y}_2 = -0.5 \cdot y_1 - 5 \cdot y_2 + e^{-t} \cdot \sin(2 \cdot t) \end{array} \right\} \quad \left[\begin{array}{l} y_1(0) = 1 \\ y_2(0) = 0 \end{array} \right].$$

Программа решения этой системы дифференциальных уравнений с выводом графиков входного воздействия и временных зависимостей $y_1(t)$ и $y_2(t)$ и подробными текстовыми комментариями представлена ниже (m-файл odedemo1):

```
function odedemo
% Построение графика входного воздействия (сплошная линия без маркеров)
x=0:0.1:20; plot(x,exp(-x).*sin(2.*x));hold on;
% Задание вектора начальных условий
Y0 = [1; 0];
% Вызов решателя ODE
[T, Y] = ode113(@oscil, [0 20], Y0);
% Вывод графика решения(маркеры – точки, линия – сплошная)
plot(T, Y(:,1), 'r.-')
% Вывод графика производной от решения(маркеры – точки, линия –
пунктир)
hold on; plot(T, Y(:,2), 'k.:')
% вывод титульной надписи на графике
title('Solve equation {\ity} \prime\prime+0.5{\ity} \prime+5{\ity}
= exp(-{\itt})*sin(2*{\itt})')
xlabel('\itt'); ylabel('\ity, {\ity} \prime ')
legend('input','coordinate', 'speed', 4);grid on; hold off
% Задавание правых частей системы ODE
function F = oscil(t, y)
F = [y(2); -0.5*y(2)-5*y(1)+exp(-t)*sin(2*t)];
```

Задав в командной строке

```
>> odedemo1,
```

получим результат работы программы в виде графиков, представленных на рис. 8.7. В этой программе полезно изучить вывод титульной надписи рисунка, в частности задание формулы уравнения в типичном математическом виде.

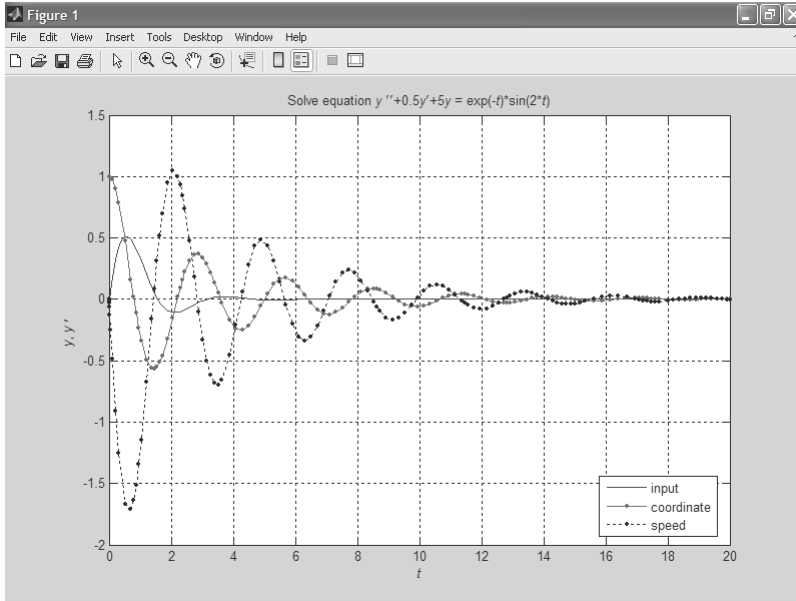


Рис. 8.7. Реакция системы второго порядка на заданное входное воздействие – затухающую синусоиду

8.9.4. Решение уравнений Лотки–Вольтерра двумя методами

Примером системы из двух дифференциальных уравнений, вызывающей проблемы в ее решении из-за жесткости, является система нелинейных дифференциальных уравнений Лотки–Вольтерра, описывающая сосуществование хищников и жертв. Пусть $y_1(t)$ – численность хищников, а $y_2(t)$ – численность жертв в момент времени t . Тогда система уравнений Лотки–Вольтерра представляется в виде:

$$\begin{cases} \dot{y}_1 = P \cdot y_1 - p \cdot y_1 y_2 \\ \dot{y}_2 = -R \cdot y_2 + r \cdot y_1 y_2 \end{cases}$$

Константа P задает увеличение числа жертв в отсутствие хищников, а R задает уменьшение числа хищников в отсутствие жертв. Вероятность поедания хищником жертвы пропорциональна произведению $y_1 y_2$, так что $py_1 y_2$ соответствует уменьшению числа жертв, а $ry_1 y_2$ соответствует росту числа пожирающих их хищников. При определенных параметрах решение системы соответствует колебательному характеру изменения числа жертв и хищников.

Даже когда временные зависимости численности хищников и жертв близки к гармоническим, решение системы Лотки–Вольтерра может вызвать трудности и потребовать выбора решателя, подходящего для того или иного набора параметров. Проведем сравнение решения данной системы при $P = 0,8$, $R = 1,0$, $p = r = 0,002$ и начальном числе жертв и хищников, равном, соответственно, 500 и 525. Ниже представлена программа, строящая фазовые портреты двух вариантов решения – с решателями `ode15s` и `ode23s` (m-файл `compode`):

```
function compode
% Сравнение решателей жестких ODE на примере уравнений Лотки-Воль-
% терра
% Задаание вектора начальных условий
Y0 = [500; 525];
% Вызов решателя ode15s
[T, Y] = ode15s(@LotVol, [0 100], Y0);
% Построение левого фазового портрета
subplot(1, 2, 1); plot(Y(:,1), Y(:,2)); title('Solver ode15s')
% Вызов решателя ode23s
[T, Y] = ode23s(@LotVol, [0 100], Y0);
% Построение правого фазового портрета
subplot(1, 2, 2); plot(Y(:,1), Y(:,2)); title('Solver ode23s')
% Задание правых частей системы ODE Лотки-Вольтерра
function F = LotVol(t, y)
F = [0.8*y(1)-0.002*y(1)*y(2); -1.0*y(2)+0.002*y(1)*y(2)];
```

Исполнение этой программы строит два фазовых портрета – слева при решателе `ode15s` и справа при решателе `ode23s`. Они представлены на рис. 8.8. Левый фазовый портрет указывает на нестабильность решения, тогда как правый дает сразу выход на предельный цикл колебаний, в ходе которого они практически повторяются. Для данного случая это говорит о преимуществе выбора решателя `ode23s`. Однако при других параметрах системы Лотки–Вольтерра это преимущество не гарантируется.

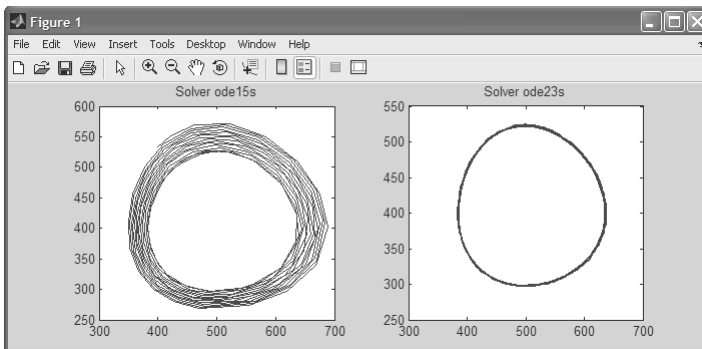


Рис. 8.8. Фазовые портреты системы Лотки–Вольтерра для решателя ODE `ode15s` (слева) и `ode23s` (справа)

Ближкий к эллипсу вид предельного цикла указывает на почти гармонический характер колебаний, а небольшие отличия от него свидетельствуют о небольшой нелинейности системы в заданных диапазонах изменения численности жертв и хищников. Разумеется, и это нельзя считать догмой, и при иных параметрах нелинейность системы может проявить себя достаточно резко.

8.9.5. Решение системы Лотки–Вольтерра с запаздывающим аргументом

Особый класс систем дифференциальных уравнений – системы с запаздывающим аргументом. Запаздывание аргумента позволяет учесть инерционность некоторых событий. Например, можно предположить, что увеличение числа хищников после поедания жертв увеличивается не сразу, а спустя некоторое время τ . В этом случае система дифференциальных уравнений Лотки–Вольтерра должна быть записана в виде:

$$\begin{cases} \dot{y}_1(t) = P \cdot y_1(t) - p \cdot y_1(t)y_2(t) \\ \dot{y}_2(t) = -R \cdot y_2(t) + r \cdot y_1(t-\tau)y_2(t-\tau) \end{cases}$$

Ниже представлена модифицированная функция LotVol2 из [28] с исправленной ошибкой (введением отсутствующей подфункции LotVolPar), которая дает построение двух фазовых портретов колебаний в системе Лотки–Вольтерра:

```
function LotVol2(P,p,R,r,tau)
% Решение системы дифференциальных уравнений с запаздыванием
options = ddeset('RelTol', 1.0e-05); % установка относительной
точности
sol = dde23(@LotVolDel,tau,@LotVolHis,[0 50],options,P,p,R,r);
plot(sol.y(1,:),sol.y(2,:),':') % вывод графика приближенного решения
% Решение системы дифференциальных уравнений без запаздывания
options = odeset('RelTol', 1.0e-05); % установка относительной
точности
[T,Y] = ode23s(@LotVolPar,[0 50],[1000 1100],options, P,p,R,r);
hold on; plot(Y(:,1),Y(:,2)) % вывод графика приближенного решения
title('Lotki-Volterra models')
legend('с запаздыванием', 'без запаздывания')
% подфункция LotVolDel для вычисления правой части системы
function F = LotVolDel(t, y, Z, P, p, R, r)
F = [P*y(1)-p*y(1)*y(2); -R*y(2)+r*Z(1)*Z(2)];
% подфункция LotVolHis для вычисления предыстории решения
function H = LotVolHis(t,P,p,R,r)
H = [1000; 1100];
% подфункция LotVolPar для вычисления правой части системы, завися-
щей от параметров
function F = LotVolPar(t, y, P, p, R, r)
F = [P*y(1)-p*y(1)*y(2); -R*y(2)+r*y(1)*y(2)];
```

Приняв $P = 0,8$, $R = 1$, $p = r = 0,001$ и $\tau = 0,1$ запустим программу командой:
 >> LotVol12(0.8, 0.001, 1, 0.001, 0.1);

Программа (функция) строит два совмещенных на одном рисунке фазовых портрета – рис. 8.9. Один для системы с запаздыванием представляет собой закручивающую спираль, а другой – для системы без запаздывания несколько искаженный круг (он строится более жирной сплошной линией).

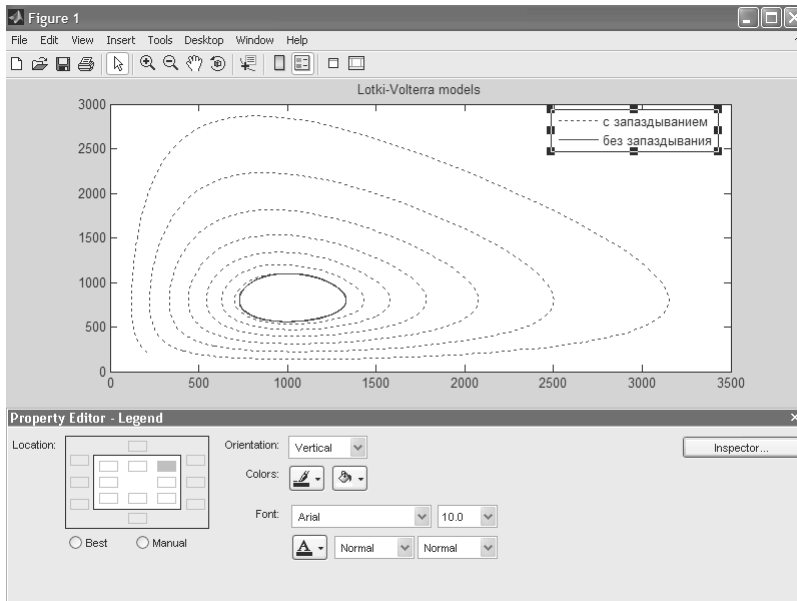


Рис. 8.9. Фазовые портреты системы Лотки–Вольтерра для случая запаздывания аргумента и без него (сплошная фигура в центре)

Отметим некоторые особенности задания и использования данной функции. Для упрощения ввода некоторые параметры (интервал времени и число жертв и хищников в начале вычислений) вписаны в листинг программы. Легенда на графике задана на русском языке, что обычно приводит к ее искажению при выводе. Для устранения этих искажений следует отформатировать вывод легенды, запустив редактор свойств легенды из контекстного меню правой клавиши мыши. Редактор свойств показан в нижней части рис. 8.9. В нем вместо шрифта по умолчанию надо из списка выбрать один из шрифтов, поддерживающих кириллицу, например Arial. Это позволит наблюдать легенду с русскоязычными надписями.

8.9.6. Решение системы дифференциальных уравнений с двухсторонними граничными условиями

Особый класс дифференциальных уравнений представляют собой дифференциальные уравнения второго и более высокого порядка, у которых надо найти параметры решения, при которых граничные условия выполняются как в начале, так и в конце интервала решения. Типичная задача такого рода – стрельба из пушки по заданной цели. Изменяя, к примеру, начальный угол полета снаряда (или его скорость), надо добиться, чтобы снаряд угодил точно в цель, находящуюся на заданной высоте и на заданном расстоянии от пушки. Нередко решение такой задачи является неоднозначным, например цель может быть поражена при двух разных начальных углах полета снаряда.

Один из методов решения подобных задач – метод пристрелки. Это итерационный метод, основанный на ряде пробных решений, которые итерационно уточняются до решения задачи с заданной погрешностью. В старых версиях MATLAB решение систем дифференциальных уравнений с двухсторонними граничными условиями было довольно сложным. Но с решателями нового поколения оно стало простым и наглядным.

Рассмотрим систему уравнений: $y' + \text{abs}(y) = 0$; $y(0) = 0$; $y(4) = -2$. Ее решение задано в файле `twobvp`. Для решения в пределах отрезка $[0; 4]$ с помощью `bvp4c` достаточно привести эту систему к виду: $y' = -\text{abs}(y)$, $y(0) = 0$; $y(4) + 2 = 0$. Затем создаем две ODE-функции `twoode` и `twobc` в разных `m`-файлах:

```
function dydx = twoode(x,y)
    dydx = [ y(2)
            -abs(y(1)) ];
```

```
function res = twobc(ya,yb)
    res = [ ya(1)
           yb(1) + 2];
```

Теперь наберите в командной строке `type twobvp` и посмотрите листинг решения. Ниже он дан в немного сокращенном и сжатом виде:

```
%TWOBVP Solve a BVP that has exactly two solutions.
% TWOBVP uses BVP4C to compute the two solutions of
%   y'' + |y| = 0
% that satisfy the boundary conditions
%   y(0) = 0, y(4) = -2
% This example illustrates how different initial guesses can lead
to different solutions.
% Jacek Kierzenka and Lawrence F. Shampine
% Copyright 1984-2002 The MathWorks, Inc.
% $Revision: 1.6.4.1 $ $Date: 2004/11/29 23:31:07 $

% One solution is obtained using an initial guess of y1(x)=1, y2(x)=0
```

```

solinit = bvpinit(linspace(0,4,5),[1 0]);
sol = bvp4c(@twoode,@twobc,solinit);
x = linspace(0,4); y1 = deval(sol,x);figure;
plot(x,y1(1,:)); xlabel('x'); ylabel('y');

% The other solution is obtained using an initial guess of y1(x)=-1,
y2(x)=0
solinit = bvpinit(linspace(0,4,5),[-1 0]);
sol = bvp4c(@twoode,@twobc,solinit); y2 = deval(sol,x);

% Plot both solutions
figure; plot(x,y1(1,:),x,y2(1,:));
xlabel('x'); ylabel('solution y');
title('A BVP with two solutions');

```

При исполнении программы `twodvp` можно наблюдать результат решения в виде графиков. Они представлены на рис. 8.10.

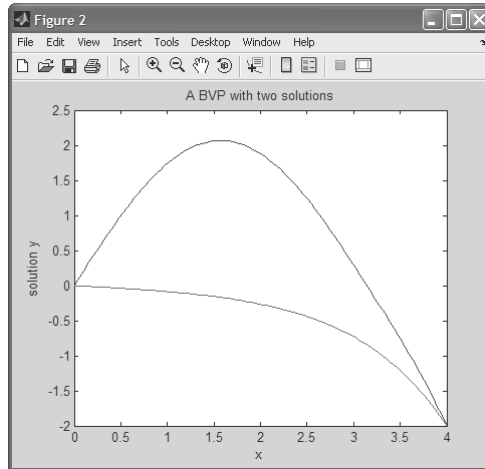


Рис. 8.10. Решения системы дифференциальных уравнений с двухсторонними граничными условиями

8.9.7. Моделирование странного аттрактора Лоренца

При решении нелинейных систем из трех и более дифференциальных уравнений обнаружены решения, порою близкие к хаотическим колебаниям. Это привело к развитию теории хаоса и беспорядка в природе [65]. К таким системам относится аттрактор Лоренца, который описывается следующей системой из трех дифференциальных уравнений:

$$\dot{y}_1 = -\sigma y_1 + \sigma y_2$$

$$\dot{y}_2 = r y_1 - y_2 - y_1 y_3.$$

$$\dot{y}_3 = y_1 y_2 - b y_3$$

Все переменные здесь и производные от них – функции времени t . Ниже представлена программа (функция) `lor`, решающая эту систему при $\sigma = 9$, $r = 38$ и $b = 13/7$ и векторе начальных условий $y_0 = [10; 20; 10]$:

```
function lor
% Задание вектора начальных условий
Y0 = [10;20;10];
% Вызов решателя ODE
[T, Y] = ode45(@oscil, [0 40], Y0);
% Вывод графика решения
subplot(2,2,1); plot(T, Y(:,1));title('y1(t)')
subplot(2,2,2); plot(T, Y(:,2));title('y2(t)')
subplot(2,2,3); plot(T, Y(:,3));title('y3(t)')
subplot(2,2,4); plot(Y(:,1),Y(:,2));title('y1(t),y2(t)')
% Вывод графика производной от решения(маркеры – точки, линия –
пунктир)
% hold on; plot(T, Y(:,2), 'k.:')
% Задание правых частей системы ODE
function F = oscil(t, y)
F = [-9*y(1)+9*y(2); 36*y(1)-y(2)-y(1)*y(3); y(1)*y(2)-(13/7)*y(3)];
```

Программа строит графики временных зависимостей всех трех переменных в интервале времени от 0 до 40. Они представлены на рис. 8.11 и наглядно отражают хаотический характер всех трех временных зависимостей, не выходящих, однако, за пределы фазового пространства. В последнем можно наблюдать два фокуса, около которых группируются «спирали» фазового портрета.

Напомним, что аттрактором в теории колебаний называют притягивающую область в фазовом пространстве. Аттрактор Лоренца, описанный выше, относится к «странным аттракторам». Причина их странности кроется в экспоненциальной неустойчивости колебаний в малых областях фазового пространства. В нашем примере их две, и в системе наблюдаются хаотические переходы от одной области притяжения к другой. В принципе, при определенных параметрах возможен переход в некоторое иное состояние, в том числе и устойчивое с прекращением колебаний или переходом их в стационарные колебания.

8.9.8. Решение жесткой алгебраически-дифференциальной системы уравнений

Рассмотрим несколько упрощенный пример из справки на решение жесткой системы дифференциальных уравнений типа HN1DAEe (проблема Робертсона):

$$y(1)' = -0.04*y(1) + 1e4*y(2)*y(3);$$

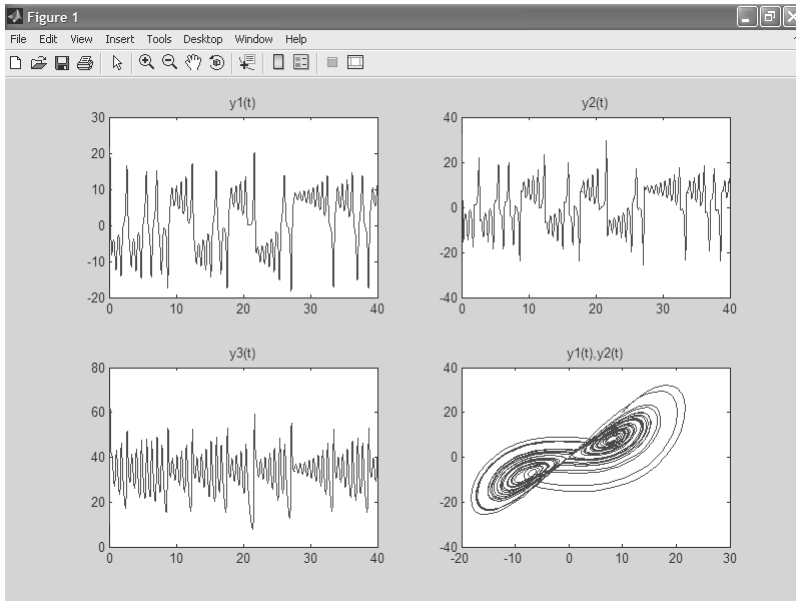


Рис. 8.11. Решения системы дифференциальных уравнений Лоренца

$$y(2)' = 0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)^2;$$

$$y(3)' = 3e7*y(2)^2.$$

Ее можно свести к алгебраически-дифференциальной системе уравнений:

$$0 = y(1)' + 0.04*y(1) - 1e4*y(2)*y(3);$$

$$0 = y(2)' - 0.04*y(1) + 1e4*y(2)*y(3) + 3e7*y(2)^2;$$

$$0 = y(1) + y(2) + y(3) - 1.$$

Составим программу (функцию) `hb1dae`, решающую данную систему:

```
function hb1dae
M = [1 0 0; 0 1 0; 0 0 0]; % Сингулярная матрица масс
y0 = [1; 0; 1e-3]; tspan = [0 4*logspace(-6,6)]; % Логарифмическая
шкала времени
options = odeset('Mass',M,'RelTol',1e-4,'AbsTol',[1e-6 1e-10 1e-6],
'Vectorized','on');
[t,y] = ode15s(@f,tspan,y0,options); y(:,2) = 1e4*y(:,2); % Решение
figure; semilogx(t,y); ylabel('1e4 * y(:,2)'); % Графическая визуа-
лизация
title('Robertson DAE problem with a Conservation Law, solved by
ODE15S');
function out = f(t,y)
out = [ -0.04*y(1,:) + 1e4*y(2,:).*y(3,:)
0.04*y(1,:) - 1e4*y(2,:).*y(3,:) - 3e7*y(2,:).^2
y(1,:) + y(2,:) + y(3,:) - 1 ];
```

Графики решения этой системы представлены на рис. 8.12. Характерно построение их в полулогарифмическом масштабе, что связано со спецификой этой задачи – процессы развиваются в очень широком диапазоне времен, и выбор полулогарифмического масштаба позволяет повысить наглядность решения. По справке можно более подробно ознакомиться с решением этой интересной задачи.

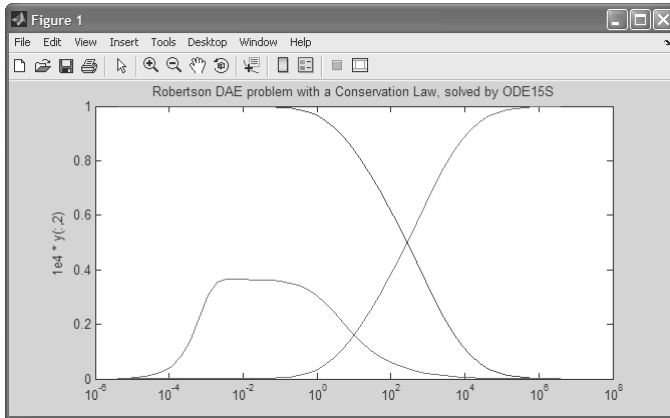


Рис. 8.12. Решение проблемы Робертсона

8.9.9. Доступ к примерам на решение дифференциальных уравнений

Можно ознакомиться со множеством интересных демонстрационных примеров на решение дифференциальных уравнений, исполнив в командной строке следующую команду:

```
>> odeexamples
```

При этом откроется окно GUI с перечнем демонстрационных примеров, представленное на рис. 8.13. С помощью списка разделов примеров, показанного в открытом виде на рис. 8.13, можно перейти в один из пяти разделов – от решения одиночных дифференциальных уравнений до решения ОДУ в частных производных.

В каждом из разделов представлен ряд примеров, список которых имеется в поле Examples of. Кнопка **Run** под этим списком запускает пример и выводит окно с результатом, как правило, графическим. На рис. 8.14 показан пример на решение системы жестких дифференциальных уравнений, описывающих процесс диффузии в химическом реакторе. Результатом решения является сложная поверхность, представляющая процесс диффузии одной фракции в другую.

Из окна редактора, нажав кнопку **View Code**, можно просмотреть листинг программы на языке MATLAB, описывающей тот или иной пример. Так, для представленного выше примера этот листинг в окне редактора программ дан на рис. 8.15.

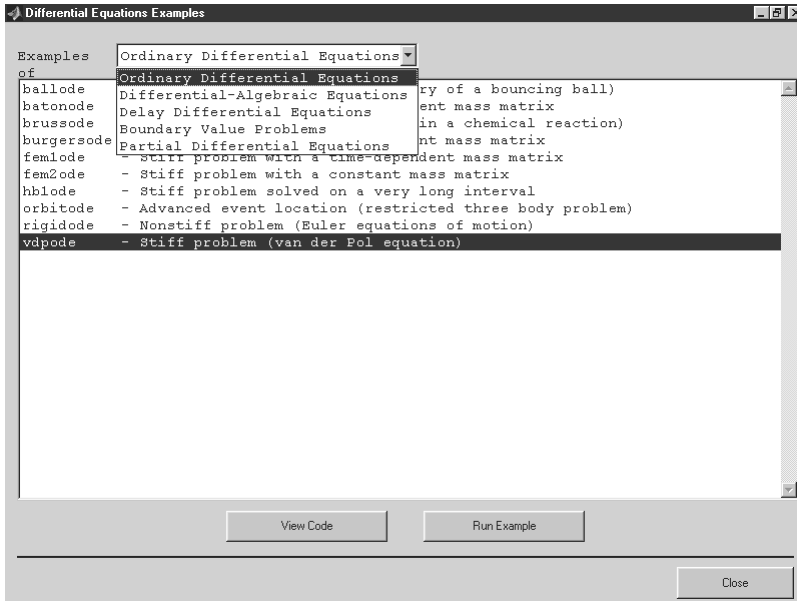


Рис. 8.13. Окно демонстрационных примеров на решение ОДУ

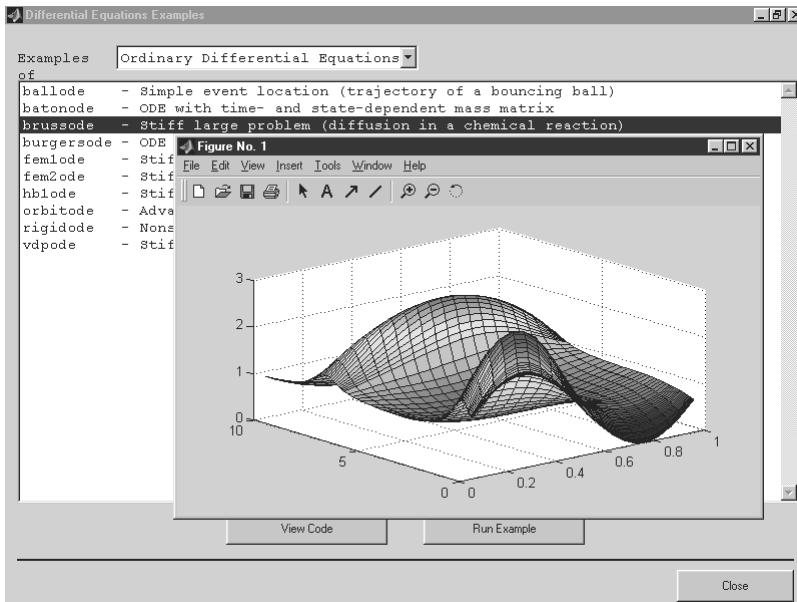


Рис. 8.14. Пример решения жестких дифференциальных уравнений, описывающих диффузию в химическом реакторе

```

E:\MATLAB65B\toolbox\matlab\demos\brussode.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: 0/20
28 % Mark W. Reichelt and Lawrence F. Shampine, 8-30-94
29 % Copyright 1984-2001 The MathWorks, Inc.
30 % $Revision: 1.17 $ $Date: 2001/04/15 12:02:56 $
31
32 if nargin<1
33     N = 20;
34 end
35
36 tspan = [0; 10];
37 y0 = [1+sin((2*pi)/(N+1))*(1:N)]; repmat(3,1,N);
38
39 options = odeset('Vectorized','on','JPattern','jpattern(N));
40
41 [t,y] = ode15s(@f,tspan,y0,options,N);
42
43 u = y(:,1:2:end);
44 x = (1:N)/(N+1);
45 figure;
46 surf(x,t,u);
47 view(-40,30);
48 xlabel('space');
49 ylabel('time');
50 zlabel('solution u');
51 title(['The Brusselator for N = ' num2str(N)]);
52
53 % -----
54
55 function dydt = f(t,y,N)
56     c = 0.02 * (N+1)^2;
57     dydt = zeros(2*N,size(y,2)); % preallocate dy/dt
58
59 % Evaluate the 2 components of the function at one edge of the grid
60 % (with edge conditions).

```

Рис. 8.15. Листинг программы для примера рис. 8.14

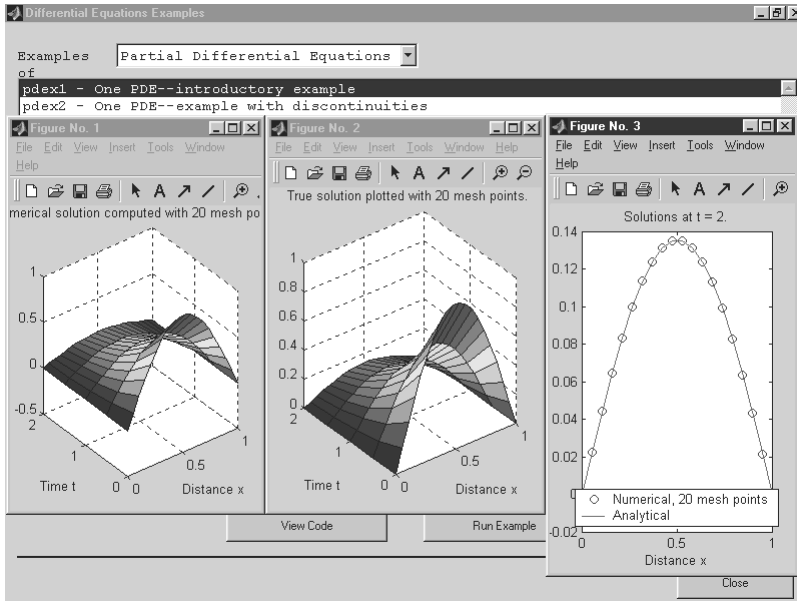
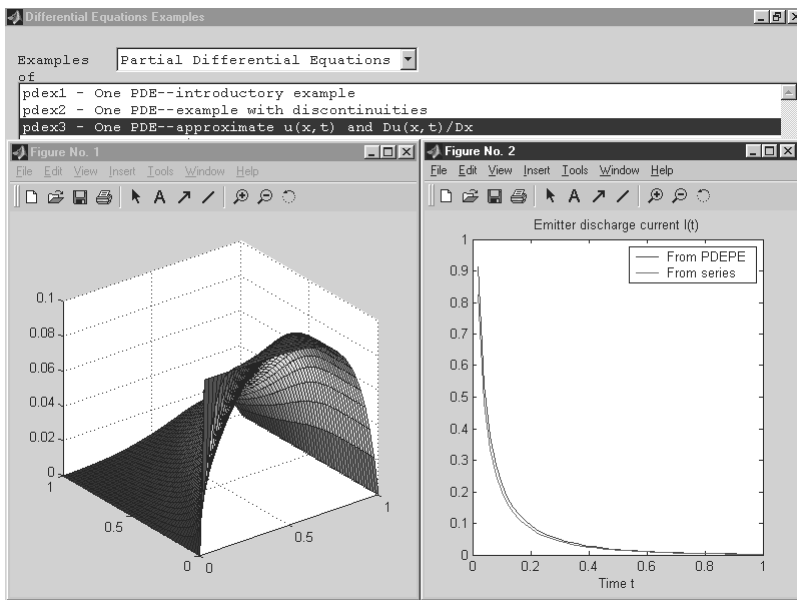
Разумеется, можно вызвать листинг программы того или иного примера, исполнив команду `type name`, где `name` – имя примера. Поскольку из-за большого размера примеров привести их в данной книге не представляется возможным, читателю настоятельно рекомендуется просмотреть листинги примеров и опробовать их в работе. Это поможет освоению данного раздела.

8.9.10. Решения дифференциальных уравнений в частных производных

Особый интерес представляют примеры на решение дифференциальных уравнений в частных производных. Они имеются в разделе `Partial Differential Equations`. Эти уравнения часто используются при моделировании поверхностей с ограничениями, например таких, как оболочки или волны. Рисунок 8.16 показывает результат выполнения одного из примеров – `pdex1`.

В этом примере результат представлен трехмерными изображениями поверхности (оболочки) и ее сечением в виде двумерного графика.

Еще один, имеющий большое прикладное значение пример – `pdex3` – демонстрирует решение важной физической проблемы из области электроники. Он дает решение во времени и пространстве для распределения тока эмиттера биполярного транзистора. Сравняются решения двумя методами – с использованием модели на основе системы ОДУ в частных производных и аппроксимации решения временным рядом. Рисунок 8.17 дает представление о визуализации решения в пространственной и временной областях.

Рис. 8.16. Выполнение примера *pdex1*Рис. 8.17. Выполнение примера *pdex3*

На рис. 8.18 представлено начало листинга программы этого примера. В нем содержится описание постановки задачи. Нетрудно заметить, что эти, как и другие, примеры применения системы MATLAB наглядно иллюстрируют полную открытость этой системы – любой пример доступен для тщательного изучения, модификации и исполнения.

```

1 function pdex3
2 %PDEX3 Example 3 for PDEPE
3 % This example illustrates the use of PDEVAL to obtain partial derivatives
4 % that are part of solving the problem. A relatively fine spatial mesh is
5 % needed to obtain accurate partial derivatives. Values are required at a
6 % relatively large number of times.
7 %
8 % This problem arises in transistor theory. It is used in [1] to illustrate
9 % solution of PDEs with series. In the form expected by PDEPE, the single PDE
10 % is
11 %
12 % [1] .* D_ [u] = D_ [ d*Du/Dx ] + [ -(eta/L)*Du/Dx ]
13 %      Dt      Dx
14 % ---      ---      -----
15 % c        u          f(x,t,u,Du/Dx)      s(x,t,u,Du/Dx)
16 %
17 % Here d and eta are physical constants. The equation is to hold on an
18 % interval 0 <= x <= L for times t >= 0. For the problem at hand, L = 1.
19 % The initial condition
20 %
21 % u(x,0) = (K*L/d)*((1 - exp(-eta*(1 - x/L)))/eta)
22 %
23 % involves another physical constant K. The left bc is u(0,t) = 0:
24 %
25 % [u] + [0] .* [ Du/Dx ] = [0]
26 %
27 % ---      ---      -----
28 % p(0,t,u)  q(0,t)  f(0,t,u,Du/Dx)  0
29 %
30 % The right bc is u(L,t) = 0:
31 %
32 % [u] + [0] .* [ Du/Dx ] = [0]
33
Ready

```

Рис. 8.18. Листинг примера pdex3

С остальными примерами нетрудно познакомиться самостоятельно. При необходимости их можно использовать в качестве заготовок для своих примеров, готовящихся на языке программирования системы MATLAB. Для решения дифференциальных уравнений с частными производными можно также использовать пакет расширения Partial Differential Equations Toolbox.

Программные средства обработки данных

| | |
|--|-----|
| 9.1. Обработка данных массивов | 442 |
| 9.2. Геометрический анализ данных | 449 |
| 9.3. Преобразование Фурье | 454 |
| 9.4. Свертка и дискретная фильтрация | 460 |
| 9.5. Интерполяция и аппроксимация данных | 465 |
| 9.6. Специальные виды интерполяции | 475 |
| 9.7. Обработка данных в графическом окне | 484 |

Обработка данных широко используется как в практике применения математических расчетов и математического моделирования, так и при проведении финансовых и экономических расчетов [51, 52, 61]. Она положена в основу многих алгоритмов обработки сигналов и изображений. Данный урок посвящен описанию программных средств обработки данных, присущих языку программирования системы MATLAB. Эти средства лежат в основе созданных пакетов расширения, решающих специфические вопросы обработки различных данных.

9.1. Обработка данных массивов

Этот урок посвящен традиционной обработке данных. В нем приведены основные функции языка программирования MATLAB для обработки данных, представленных массивами. Они широко используются для анализа данных физических, химических, экономических и иных экспериментов.

9.1.1. Нахождение максимального и минимального элементов массива

К простейшему анализу данных, содержащихся в некотором массиве, относится поиск его элементов с максимальным и минимальным значениями. Для этого определены следующие функции для нахождения минимальных и максимальных элементов массива:

- $\max(A)$ возвращает наибольший элемент, если A – вектор; или возвращает вектор-строку, содержащую максимальные элементы каждого столбца, если A – матрица;
- $\max(A, B)$ возвращает массив того же размера, что A и B , каждый элемент которого есть максимальный из соответствующих элементов этих массивов;
- $\max(A, [], \text{dim})$ возвращает наибольший элемент по столбцам или по строкам матрицы в зависимости от значения скаляра dim . Например, $\max(A, [], 1)$ возвращает максимальные элементы каждого столбца матрицы A ;
- $[C, I] = \max(A)$ – кроме максимальных значений, возвращает вектор индексов элементов с этими значениями.

Примеры:

```
>> A=magic(7)
```

```
A =
```

| | | | | | | |
|----|----|----|----|----|----|----|
| 30 | 39 | 48 | 1 | 10 | 19 | 28 |
| 38 | 47 | 7 | 9 | 18 | 27 | 29 |
| 46 | 6 | 8 | 17 | 26 | 35 | 37 |
| 5 | 14 | 16 | 25 | 34 | 36 | 45 |
| 13 | 15 | 24 | 33 | 42 | 44 | 4 |
| 21 | 23 | 32 | 41 | 43 | 3 | 12 |
| 22 | 31 | 40 | 49 | 2 | 11 | 20 |

```
>> C = max(A)
```

```

C =      46      47      48      49      43      44      45
>> C = max(A, [ ], 1)
C =      46      47      48      49      43      44      45
>> C = max(A, [ ], 2)
C =
        48
        47
        46
        45
        44
        43
        49
>> [C, I] = max(A)
C =      46      47      48      49      43      44      45
I =      3      2      1      7      6      5      4

```

Для нахождения элемента массива с минимальным значением служат подобные функции:

```
min(A)      min(A,B)      min(A, [ ], dim)      [C, I] = min(A)
```

Пример:

```

>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> [C, I] = min(A)
C =     4     2     3     1
I =     4     1     1     4

```

Работа указанных функций базируется на сравнении численных значений элементов массива A, что и обеспечивает высокую скорость выполнения операций.

9.1.2. Сортировка элементов массива

Многие операции статистической обработки данных выполняются быстрее, надежнее и точнее, если данные предварительно отсортированы. Кроме того, нередко представление данных в отсортированном виде более наглядно и ценно для их последующего применения. Ряд функций служит для выполнения сортировки элементов массива. Они представлены ниже.

- `sort(A)` в случае одномерного массива A сортирует и возвращает элементы по возрастанию их значений; в случае двумерного массива происходят сортировка и возврат элементов каждого столбца. Допустимы вещественные, комплексные и строковые элементы. Если A принимает комплексные значения, то элементы сначала сортируются по абсолютному значению, а затем, если абсолютные значения равны, – по аргументу. Если A включает NaN-элементы, `sort` помещает их в конец.

- `[B, INDEX] = sort(A)` наряду с отсортированным массивом возвращает массив индексов `INDEX`. Он имеет размер `size(A)`, с помощью этого массива можно восстановить структуру исходного массива.
- `sort(A, dim)` для матриц сортирует элементы по столбцам или по строкам в зависимости от значения переменной `dim`.

Примеры:

```
>> A=magic(5)
```

```
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

```
>> [B,INDEX] = sort(A)
```

```
B =
     4     5     1     2     3
    10     6     7     8     9
    11    12    13    14    15
    17    18    19    20    16
    23    24    25    21    22
```

```
INDEX =
     3     2     1     5     4
     4     3     2     1     5
     5     4     3     2     1
     1     5     4     3     2
     2     1     5     4     3
```

- `sortrows(A)` выполняет сортировку строк массива `A` по возрастанию и возвращает отсортированный массив, который может быть или матрицей, или вектором-столбцом.
- `sortrows(A, column)` возвращает матрицу, отсортированную по столбцам, точно указанным в векторе `column`. Например, `sortrows(A, [2 3])` сортирует строки матрицы `A` сначала по второму столбцу, и затем, если его элементы равны, по третьему.
- `[B, index] = sortrows(A)` также возвращает вектор индексов `index`. Если `A` – вектор-столбец, то `B=A(index)`. Если `A` – матрица размера $m \times n$, то `B=A(index, :)`.

Примеры:

```
>> A=[2 3 5 6 8 9; 5 7 1 2 3 5; 1 3 2 1 5 1; 5 0 8 8 4 3]
```

```
A =
     2     3     5     6     8     9
     5     7     1     2     3     5
     1     3     2     1     5     1
     5     0     8     8     4     3
```

```
>> B = sortrows(A)
```

```
B =
     1     3     2     1     5     1
```

```

      2     3     5     6     8     9
      5     0     8     8     4     3
      5     7     1     2     3     5
>> B = sortrows(A,3)
B =
      5     7     1     2     3     5
      1     3     2     1     5     1
      2     3     5     6     8     9
      5     0     8     8     4     3

```

- `cplxpair(A)` сортирует элементы по строкам или столбцам комплексного массива `A`, группируя вместе комплексно сопряженные пары. Затем найденные пары сортируются по возрастанию действительной части. Внутри пары элемент с отрицательной мнимой частью является первым. Действительные элементы следуют за комплексными парами. Заданный по умолчанию порог $100 * \text{eps}$ относительно $\text{abs}(A(i))$ определяет, какие числа являются действительными и какие элементы являются комплексно сопряженными. Если `A` – вектор, `cplxpair(A)` возвращает `A` вместе с комплексно сопряженными парами. Если `A` – матрица, `cplxpair(A)` возвращает матрицу `A` с комплексно сопряженными парами, сортированную по столбцам.
- `cplxpair(A, tol)` отменяет заданный по умолчанию порог и задает новый `tol`.
- `cplxpair(A, [], dim)` сортирует матрицу `A` по строкам или по столбцам в зависимости от значения параметра `dim`.
- `cplxpair(A, tol, dim)` сортирует матрицу `A` по строкам или по столбцам в зависимости от значения параметра `dim`, используя заданный порог `tol`.

Пример:

```

>> A=[23+12i, 34-3i, 45; 23-12i, -12, 2i; -3, 34+3i, -2i]
A =
 23.0000 + 12.0000i   34.0000 - 3.0000i   45.0000
 23.0000 - 12.0000i  -12.0000             0 + 2.0000i
 -3.0000             34.0000 + 3.0000i   0 - 2.0000i
>> cplxpair(A)
ans =
 23.0000 - 12.0000i   34.0000 - 3.0000i   0 - 2.0000i
 23.0000 + 12.0000i   34.0000 + 3.0000i   0 + 2.0000i
 -3.0000             -12.0000             45.0000

```

9.1.3. Нахождение средних и срединных значений

К элементарной статистической обработке данных в массиве обычно относят нахождение их среднего значения, медианы (срединного значения) и стандартного отклонения. Для этого определены следующие функции:

- `mean(A)` возвращает арифметическое среднее значение элементов массива, если `A` – вектор; или возвращает вектор-строку, содержащую средние значения элементов каждого столбца, если `A` – матрица. Арифметическое среднее значение есть сумма элементов массива, деленная на их число;
- `mean(A, dim)` возвращает среднее значение элементов по столбцам или по строкам матрицы в зависимости от значения скаляра `dim` (`dim = 1` по столбцам и `dim = 2` по строкам соответственно).

Примеры:

```
>> A = [1 2 6 4 8; 6 7 13 5 4; 7 9 0 8 12; 6 6 7 1 2]
A =
     1     2     6     4     8
     6     7    13     5     4
     7     9     0     8    12
     6     6     7     1     2

>> mean(A)
ans = 5.0000  6.0000  6.5000  4.5000  6.5000

>> mean(A, 2)
ans =
     4.2000
     7.0000
     7.2000
     4.4000
```

- `median(A)` возвращает медиану, если `A` – вектор; или вектор-строку медиан для каждого столбца, если `A` – матрица;
- `median(A, dim)` возвращает значения медиан для столбцов или строк матрицы в зависимости от значения скаляра `dim`.

Примеры:

```
>> A=magic(6)
A =
    35     1     6     26    19    24
     3    32     7     21    23    25
    31     9     2     22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11

>> M=median(A)
M =
    19.0000    18.5000    18.0000    19.0000    18.5000    18.0000

>> M=median(A, 2)
M =
    21.5000
    22.0000
    21.0000
    16.0000
    15.0000
    15.5000
```

9.1.4. Вычисление стандартного отклонения

Стандартное отклонение для вектора X вычисляется по формуле

$$S = \left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{1/2}.$$

Если X – матрица, то `std(X)` возвращает вектор-строку, содержащую стандартное отклонение элементов каждого столбца (обратите внимание, что оно отличается от среднеквадратического отклонения). Стандартное отклонение вычисляется следующей функцией:

- `std(X)` возвращает стандартное отклонение элементов массива;
- `std(X, flag)` возвращает то же значение, что и `std(X)`, если `flag=0`; если `flag=1`, функция `std(X, 1)` возвращает среднеквадратическое отклонение (квадратный корень из несмещенной дисперсии), вычисляемое по формуле

$$S = \left(\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{1/2};$$

- `std(X, flag, dim)` возвращает стандартное или среднеквадратическое отклонения по строкам или по столбцам матрицы X в зависимости от значения переменной `dim`.

Примеры:

```
>> X = linspace(0, 3*pi, 10)
X =
    Columns 1 through 7
    0    1.0472    2.0944    3.1416    4.1888    5.2360    6.2832
    Columns 8 through 10
    7.3304    8.3776    9.4248
>> s = std(X)
s =    3.1705
```

9.1.5. Вычисление коэффициентов корреляции

Под *корреляцией* понимается взаимосвязь некоторых величин, представленных данными – векторами или матрицами. Общепринятой мерой линейной корреляции является *коэффициент корреляции*. Его близость к единице указывает на высокую степень идентичности сопоставляемых зависимостей. Приведенная ниже функция позволяет вычислить коэффициенты корреляции для входного массива данных.

- `corrcoef(X)` возвращает матрицу коэффициентов корреляции для входной матрицы, строки которой рассматриваются как наблюдения, а столбцы – как переменные. Матрица `S=corrcoef(X)` связана с матрицей ко-

вариаций $C = \text{cov}(X)$ следующим соотношением: $S(i, j) = C(i, j) / \sqrt{C(i, i)C(j, j)}$.

- Функция $S = \text{corrcoef}(x, y)$, где x и y – векторы-столбцы, аналогична функции $\text{corrcoef}([x \ y])$.

Пример:

```
>> M=magic(5)
M =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> S=corrcoef(M)
S =
    1.0000    0.0856   -0.5455   -0.3210   -0.0238
    0.0856    1.0000   -0.0981   -0.6731   -0.3210
   -0.5455   -0.0981    1.0000   -0.0981   -0.5455
   -0.3210   -0.6731   -0.0981    1.0000    0.0856
   -0.0238   -0.3210   -0.5455    0.0856    1.0000
```

В целом корреляция данных в этом примере довольно низкая, за исключением данных, расположенных по диагонали, – здесь коэффициенты корреляции равны 1.

9.1.6. Вычисление матрицы ковариации

Приведенная ниже функция позволяет вычислить матрицу ковариации для массива данных.

- $\text{cov}(x)$ возвращает смещенную дисперсию элементов вектора x . Для матрицы, где каждая строка рассматривается как наблюдение, а каждый столбец – как переменная, $\text{cov}(x)$ возвращает матрицу ковариаций. $\text{diag}(\text{cov}(x))$ – вектор смещенных дисперсий для каждого столбца и $\sqrt{\text{diag}(\text{cov}(x))}$ – вектор стандартных отклонений.
- Функция $C = \text{cov}(x, y)$, где x и y – векторы-столбцы одинаковой длины, равносильна функции $\text{cov}([x \ y])$.

Пример:

```
>> D=[2 -3 6;3 6 -1;9 8 5];C=cov(D)
C =
    14.3333    16.3333     3.6667
    16.3333    34.3333   -10.3333
     3.6667   -10.3333    14.3333

>> diag(cov(D))
ans =
    14.3333
    34.3333
    14.3333

>> sqrt(diag(cov(D)))
```

```
ans =
    3.7859
    5.8595
    3.7859
>> std(D)
ans = 3.7859  5.8595  3.7859
```

9.2. Геометрический анализ данных

Далее мы рассмотрим функции геометрического анализа данных. Такой анализ не относится к достаточно распространенным средствам анализа данных, но для специалистов он представляет несомненный интерес.

9.2.1. Триангуляция Делоне

Пусть есть некоторое число точек. *Триангуляция Делоне* – это множество линий, соединяющих каждую точку с ее ближайшими соседними точками. *Диаграммой Вороного* называют многоугольник, вершины которого – центры окружностей, описанных вокруг треугольников Делоне.

В системе MATLAB определены функции триангуляции Делоне, триангуляции Делоне для ближайшей точки и поиска наилучшей триангуляции. Рассмотрим функции, реализующие триангуляцию Делоне.

- `TRI = delaunay(x, y)` возвращает матрицу размера $m \times 3$ множества треугольников (триангуляция Делоне), такую что ни одна из точек данных, содержащихся в векторах x и y , не попадает внутрь окружностей, проходящих через вершины треугольников. Каждая строка матрицы TRI определяет один такой треугольник и состоит из индексов векторов x и y .
- `TRI = delaunay(x, y, 'sorted')` – при расчетах предполагается, что точки векторов x и y отсортированы сначала по y , затем по x и двойные точки уже устранены.

Пример:

```
>> rand('state',0);x=rand(1,25); y=rand(1,25);
>> TRI = delaunay(x,y); trimesh(TRI,x,y,zeros(size(x)))
>> axis([0 1 0 1]); hold on; plot(x,y,'o')
```

Построенная по этому примеру диаграмма представлена на рис. 9.1.

- `dsearch(x, y, TRI, xi, yi)` возвращает индекс точки из числа содержащихся в массивах x и y , ближайшей к точке с координатами (xi, yi) , используя массив данных триангуляции TRI (триангуляция Делоне для ближайшей точки).
- `dsearch(x, y, TRI, xi, yi, S)` делает то же, используя заранее вычисленную разреженную матрицу триангуляции S : $S = \text{sparse}(TRI(:, [1\ 1\ 2\ 2\ 3\ 3]), TRI(:, [2\ 3\ 1\ 3\ 1\ 2]), 1, nxy, nxy)$, где $nxy = \text{prod}(\text{size}(x))$.
- `tsearch(x, y, TRI, xi, yi)` выполняет поиск наилучшей триангуляции, возвращает индексы строк матрицы триангуляции TRI для каждой точки

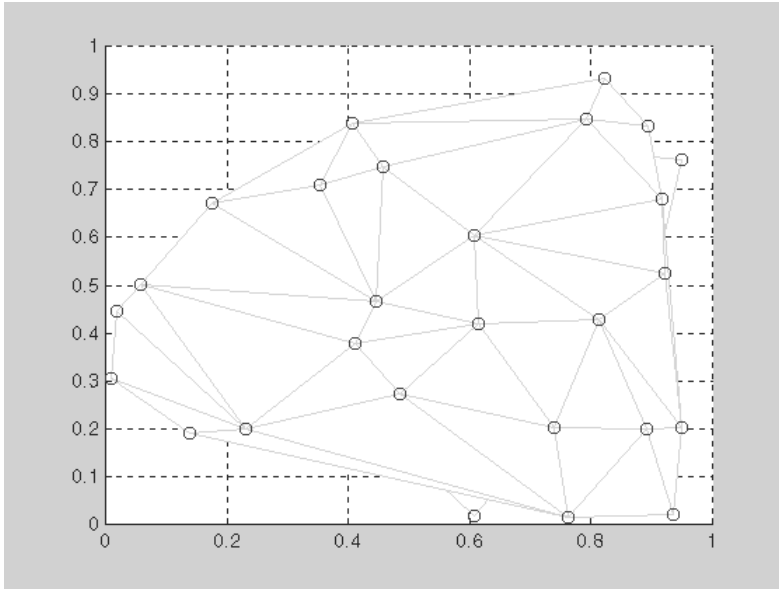


Рис. 9.1. Пример применения функции *delaunay*

с координатами (x_i, y_i) . Возвращает NaN для всех точек, находящихся вне выпуклой оболочки.

Триангуляция трехмерных и n -мерных массивов ($n \geq 2$) осуществляется при помощи функций *delaunay3* и *delaunayn* соответственно. Эти функции используют не алгоритм вычисления диаграмм Вороного, как *delaunay*, а алгоритм *qhull* Национального научно-технического и исследовательского центра визуализации и вычисления геометрических структур США.

9.2.2. Вычисление выпуклой оболочки

В системе MATLAB определена функция вычисления точек *выпуклой оболочки*:

- *convhull(x, y)* возвращает индексы тех точек, задаваемых векторами x и y , которые лежат на выпуклой оболочке;
- *convhull(x, y, TRI)* использует триангуляцию, полученную в результате применения функции триангуляции Делоне *delaunay*, вместо того чтобы вычислять ее самостоятельно.

Пример:

```
% Программа построения выпуклой оболочки
xx=-0.8:0.03:0.8;
yy = abs(sqrt(xx));
[x, y] = pol2cart(xx, yy);
k = convhull(x, y);
plot(x(k), y(k), 'r', x, y, 'g*')
```

Рисунок 9.2 иллюстрирует применение функции `convhull` для построения выпуклой оболочки.

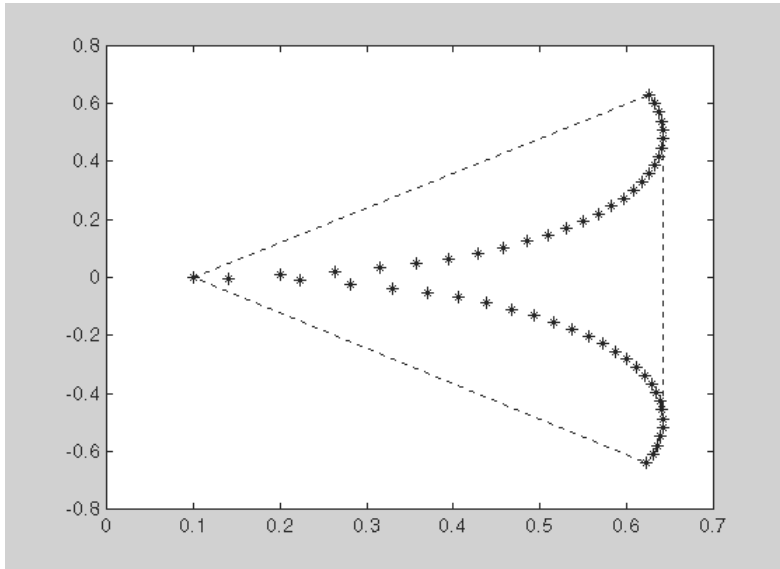


Рис. 9.2. Пример использования функции `convhull`

Функция `convhulln` вычисляет n -мерную выпуклую поверхность, основана на алгоритме `qhull`.

9.2.3. Вычисление площади полигона

В системе MATLAB определены функции, вычисляющие *площадь полигона* и анализирующие нахождение точек внутри полигона. Для вычисления площади полигона используется функция `polyarea`:

- `polyarea(X, Y)` возвращает площадь полигона, заданного вершинами, находящимися в векторах X и Y . Если X и Y – матрицы одного размера, то `polyarea` возвращает площадь полигонов, определенных столбцами X и Y .
- `polyarea(X, Y, dim)` возвращает площадь полигона, заданного столбцами или строками X и Y в зависимости от значения переменной `dim`.

Пример:

```
>> L = linspace(0, 3*pi, 10); X = sin(L)';  
>> Y = cos(L)'; A = polyarea(X, Y)  
A = 3.8971  
>> plot(X, Y, 'm')
```

Построенный по этому примеру многоугольник представлен на рис. 9.3.

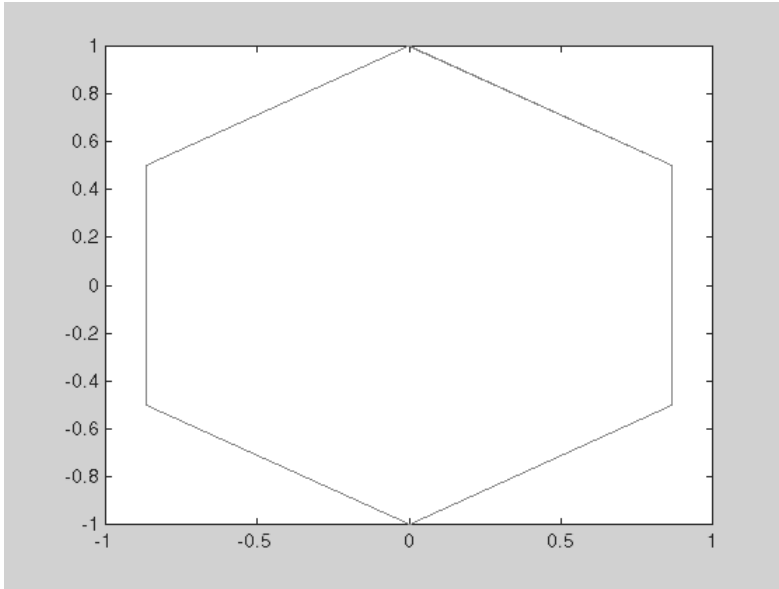


Рис. 9.3. Область многоугольника, для которого вычислена площадь

В данном примере использована функция `linspace(x1, x2, N)`, генерирующая N точек в промежутке от $x1$ до $x2$ с последующим формированием векторов X и Y для построения многоугольника в полярной системе координат.

9.2.4. Анализ попадания точек внутрь полигона

Функция `inpolygon` используется для анализа того, попадают ли заданные точки внутрь полигона:

- `IN=inpolygon(X, Y, xv, yv)` возвращает матрицу `IN` того же размера, что X и Y . Каждый элемент матрицы `IN` принимает одно из значений -1 , $0,5$ или 0 – в зависимости от того, находится ли точка с координатами $(X(p, q), Y(p, q))$ внутри полигона, вершины которого определяются векторами xv и yv :
- $IN(p, q) = 1$ – если точка $(X(p, q), Y(p, q))$ лежит внутри полигона;
- $IN(p, q) = 0.5$ – если точка $(X(p, q), Y(p, q))$ лежит на границе полигона;
- $IN(p, q) = 0$ – если точка $(X(p, q), Y(p, q))$ лежит вне полигона.

Пример:

```
% Программа построения точек и полигона
L = linspace(0, 2*pi, 8);
yv = sin(L)'; xv = cos(L)';
```

```
x = randn(100,1);
y = randn(100,1);
IN = inpolygon(x,y,xv,yv);
plot(xv,yv,'k',x(IN),y(IN),'r*',x(~IN),y(~IN),'bo')
```

Построенные в этом примере массив точек и полигон представлены на рис. 9.4.

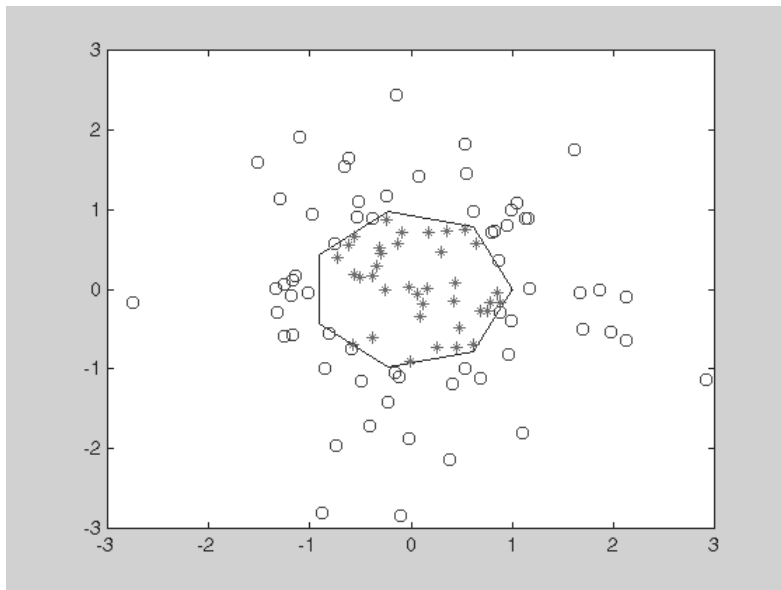


Рис. 9.4. Пример применения функции `inpolygon`

Точки, попавшие внутрь полигона, обозначены символом звездочки, а точки вне полигона обозначены кружками.

9.2.5. Построение диаграммы Вороного

Для построения *диаграммы Вороного* служит следующая команда:

- `voronoi(x,y)` строит диаграмму Вороного для точек с координатами (x,y) . Функция `voronoi(x,y,TRI)` использует триангуляцию TRI;
- `voronoi(...,'LineStyle')` строит диаграмму с заданным цветом и стилем линий;
- `[vx,vy] = voronoi(...)` возвращает вершины граней Вороного в векторах v_x и v_y , так что команда `plot(vx,vy,'-',x,y,'.')` создает диаграмму Вороного.

Пример программы для построения диаграммы Вороного:

```
% Программа построения диаграммы Вороного
rand('state',0); x = rand(1,15); y = rand(1,15);
```

```

TRI = delaunay(x,y); subplot(1,2,1),...
trimesh(TRI,x,y,zeros(size(x))); view(2),...
axis([0 1 0 1]); hold on;
plot(x,y,'o'); [vx,vy] = voronoi(x,y,TRI);
subplot(1,2,2), plot(x,y,'r+',vx,vy,'b-'),...
axis([0 1 0 1])

```

Рисунок 9.5 (слева) иллюстрирует построение треугольников Делоне.

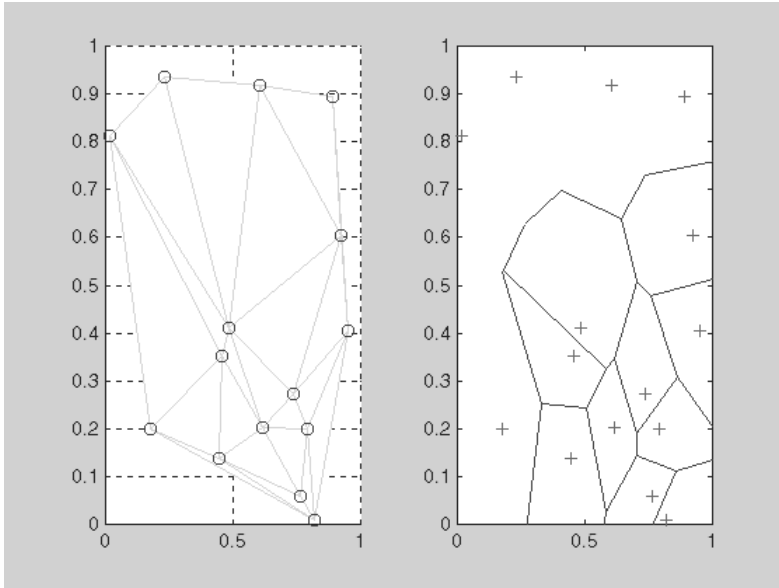


Рис. 9.5. Связь триангуляции Делоне с диаграммой Вороного

На рис. 9.5 (справа) знаками «плюс» изображены центры окружностей, проведенных вокруг треугольников Делоне.

Функция $[V,C]=\text{voronoin}(X)$ служит для построения диаграмм Вороного n -мерных данных. V – массив граней, C – массив клеток диаграмм Вороного. При $n = 2$ вершины граней Вороного возвращаются в порядке смежности, при $n > 2$ – в порядке убывания.

9.3. Преобразование Фурье

9.3.1. Основные определения

Разработка преобразований Фурье сыграла огромную роль в появлении и развитии ряда новых областей науки и техники. Достаточно отметить, что электротехника переменного тока, электрическая связь и радиосвязь базируются на спект-

ральном представлении сигналов. Ряды Фурье также можно рассматривать как приближение произвольных функций (определенные ограничения в этом известны) тригонометрическими рядами бесконечной длины. При конечной длине рядов получаются наилучшие среднеквадратические приближения.

MATLAB содержит функции для выполнения быстрых одномерного и двумерного дискретных преобразований Фурье. Для одномерного массива x с длиной N прямое и обратное преобразования Фурье реализуются по следующим формулам:

$$X(k) = \sum_{j=1}^N x(j) e^{2\pi/N(j-1)(k-1)},$$

$$x(k) = \frac{1}{N} \sum_{k=1}^N X(k) e^{-2\pi/(N(j-1)(k-1)}.$$

Прямое преобразование Фурье переводит описание сигнала (функции времени) из временной области в частотную, а обратное преобразование Фурье переводит описание сигнала из частотной области во временную. На этом основаны многочисленные методы фильтрации сигналов.

9.3.2. Одномерное прямое быстрое преобразование Фурье

В описанных ниже функциях реализован особый метод *быстрого преобразования Фурье* – Fast Fourier Transform (FFT, или БПФ), позволяющий резко уменьшить число арифметических операций в ходе приведенных выше преобразований. Он особенно эффективен, если число обрабатываемых элементов (отсчетов) составляет 2^m , где m – целое положительное число.

Для одномерного преобразования используется следующая функция:

- `fft(X)` возвращает для вектора X дискретное преобразование Фурье, по возможности используя алгоритм быстрого преобразования Фурье. Если X – матрица, функция `fft` возвращает преобразование Фурье для каждого столбца матрицы;
- `fft(X, n)` возвращает n -точечное преобразование Фурье. Если длина вектора X меньше n , то недостающие элементы заполняются нулями. Если длина X больше n , то лишние элементы удаляются. Когда X – матрица, длина столбцов корректируется аналогично;
- `fft(X, [], dim)` и `fft(X, n, dim)` применяют преобразование Фурье к одной из размерностей массива в зависимости от значения параметра `dim`.

Для иллюстрации применения преобразования Фурье создадим трехчастотный сигнал на фоне сильного шума, создаваемого генератором случайных чисел:

```
>>t=0:0.0005:1;
>>x=sin(2*pi*200*t)+.4*sin(2*pi*150*t)+.4*sin(2*pi*250*t);
>> y=x+2*randn(size(t)); plot(y(1:100), 'b')
```

Этот сигнал имеет среднюю частоту 200 рад/с и два боковых сигнала с частотами 150 и 250 рад/с, что соответствует амплитудно-модулированному сигналу с частотой модуляции 50 рад/с и глубиной модуляции 0,8 (амплитуда боковых частот составляет 0,4 от амплитуды центрального сигнала). На рис. 9.6 показан график этого сигнала (по первым 100 отсчетам из 2000). Нетрудно заметить, что из него никоим образом не видно, что полезный сигнал – амплитудно-модулированное колебание, настолько оно забито шумами.

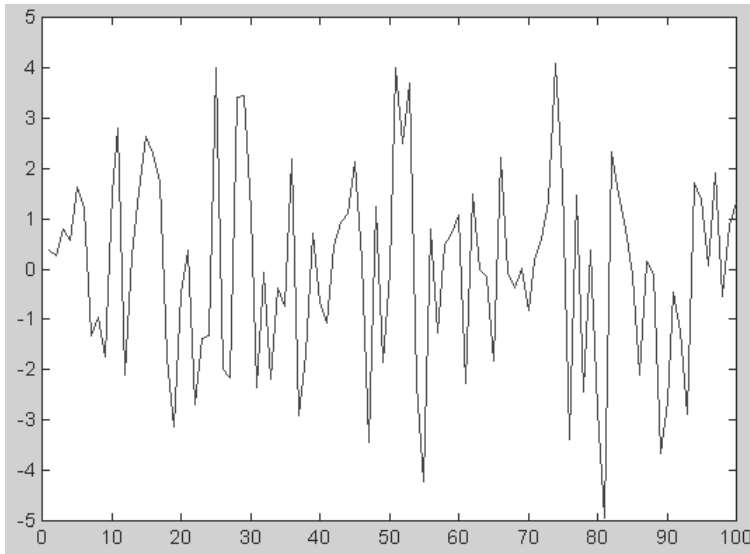


Рис. 9.6. Форма зашумленного сигнала

Теперь построим график спектральной плотности полученного сигнала с помощью прямого преобразования Фурье, по существу переводящего временное представление сигнала в частотное. Этот график (см. рис. 9.6) в области частот до 300 Гц строится следующими командами:

```
>> Y=fft(y,1024); Pyy=Y.*conj(Y)/1024;
>> f=2000*(0:150)/1024; plot(f,Pyy(1:151)),grid
```

График спектральной плотности сигнала, построенный в этом примере, представлен на рис. 9.7.

Даже беглого взгляда на рис. 9.7 достаточно, чтобы убедиться в том, что спектрограмма сигнала имеет явный пик на средней частоте амплитудно-модулированного сигнала и два боковых пика. Все эти три частотные составляющие сигнала явно выделяются на общем шумовом фоне. Таким образом, данный пример наглядно иллюстрирует технику обнаружения слабых сигналов на фоне шумов, лежащую в основе работы радиоприемных устройств.

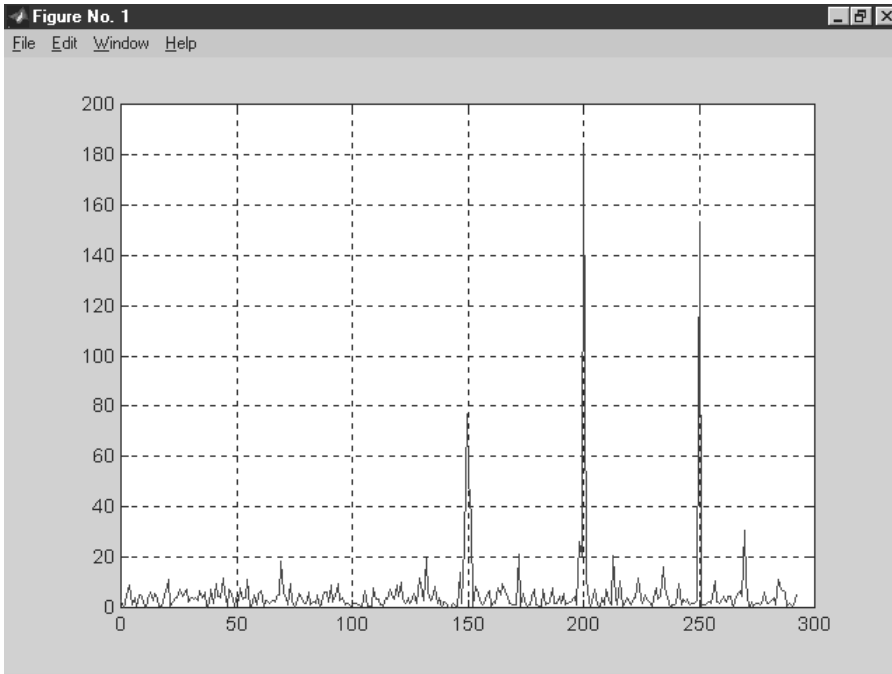


Рис. 9.7. График спектральной плотности приведенного на рис. 9.6 сигнала

9.3.3. Многомерное прямое преобразование Фурье

Для двумерного прямого преобразования Фурье используется функция `fft2`:

- `fft2(X)` возвращает для массива данных X двумерное дискретное преобразование Фурье;
- `fft2(X, m, n)` усекает массив X или дополняет его нулями, чтобы перед выполнением преобразования Фурье создать матрицу размера $m \times n$. Результат – матрица того же размера.

Для многомерного прямого преобразования Фурье также существует функция:

- `fftn(X)` возвращает результат N -мерного дискретного преобразования для массива X размерности N . Если X – вектор, то выход будет иметь ту же ориентацию;
- `fftn(X, siz)` возвращает результат дискретного преобразования для массива X с ограничением размера, заданным переменной `siz`.

9.3.4. Перегруппировка массивов

Функция $Y = \text{fftshift}(X)$ перегруппировывает выходные массивы функций `fft` и `fft2`, размещая нулевую частоту в центре спектра, что иногда более удобно. Если X – вектор, то Y – вектор с циклической перестановкой правой и левой половин исходного вектора. Если X – матрица, то Y – матрица, у которой квадранты I и III меняются местами с квадрантами II и IV. Рассмотрим следующий пример. Вначале построим график спектральной плотности мощности (рис. 9.8) при одномерном преобразовании Фурье:

```
% Программа одномерного преобразования Фурье
% и построения графика спектральной плотности мощности
rand('state',0); t=0:0.001:0.512;
x=sin(2*pi*50*t)+sin(2*pi*120*t);
y=x+2*randn(size(t))+0.3;
Y=fft(y); Pyy=Y.*conj(Y)/512;
f=1000*(0:255)/512; plot(f,Pyy(1:256)),grid
```

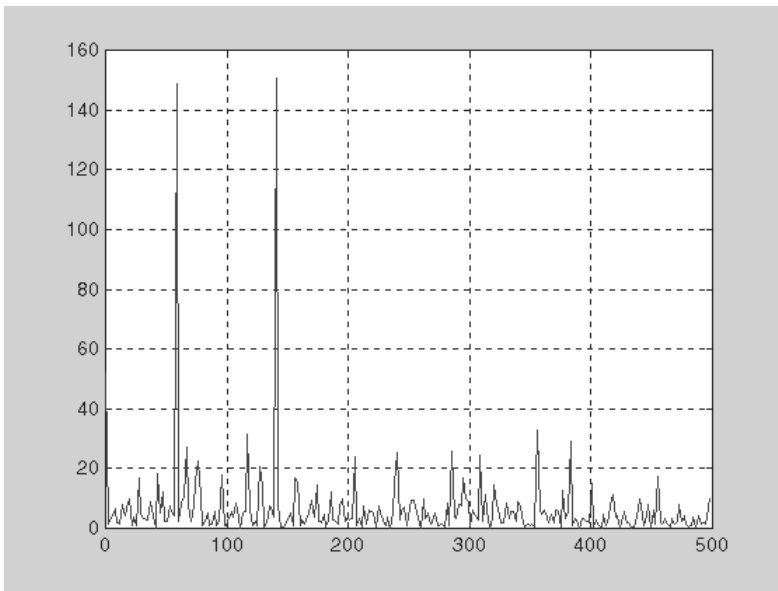


Рис. 9.8. График спектральной плотности сигнала после одномерного преобразования Фурье

Здесь мы ограничились 512 отсчетами, с тем чтобы использовать эффективный метод быстрого преобразования Фурье, при котором число отсчетов должно быть 2^N , где N – целое число. Теперь воспользуемся функцией `fftshift`:

```
>> Y=fftshift(Y); Pyy=Y.*conj(Y)/512; plot(Pyy),grid
```

Полученный при этом график представлен на рис. 9.9. Надо отметить, что этот график дает значения спектральной плотности составляющих спектра не явно от частоты, а как распределение ее значений для элементов вектора P_{yy} .

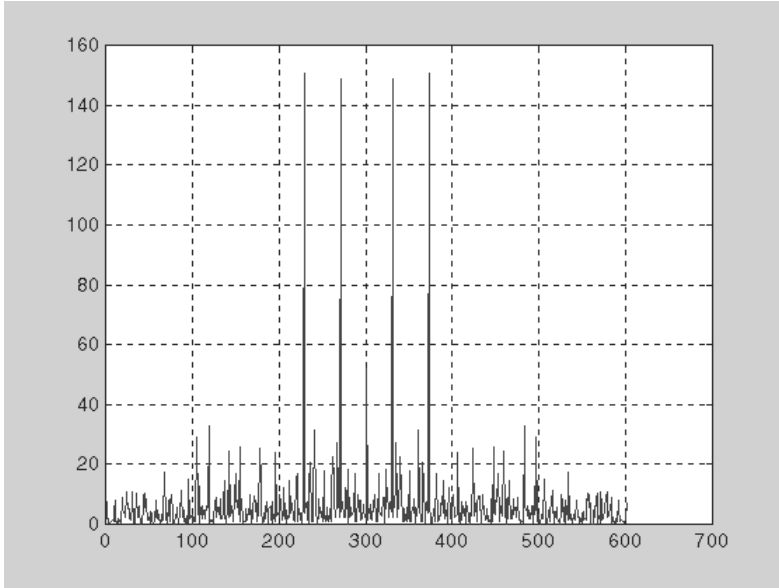


Рис. 9.9. График спектральной плотности того же сигнала после применения функции `fftshift`

9.3.5. Одномерное быстрое обратное преобразование Фурье

Возможно одномерное обратное преобразование Фурье, реализуемое следующей функцией:

- `ifft(F)` возвращает результат дискретного обратного преобразования Фурье вектора F . Если F – матрица, то `ifft` возвращает обратное преобразование Фурье для каждого столбца этой матрицы;
- `ifft(F, n)` возвращает результат n -точечного дискретного обратного преобразования Фурье вектора F ;
- `ifft(F, [], dim)` и `y = ifft(X, n, dim)` возвращают результат обратного дискретного преобразования Фурье массива F по строкам или по столбцам в зависимости от значения скаляра dim .

Для любого X результат последовательного выполнения прямого и обратного преобразований Фурье `ifft(fft(x))` равен X с точностью до погрешности округления. Если X – массив действительных чисел, `ifft(fft(x))` может иметь малые мнимые части.

Пример:

```
>> V=[1 1 1 1 0 0 0 0]; fft(V)
ans =
    Columns 1 through 4
    4.0000 1.0000 - 2.4142i    0    1.0000 - 0.4142i
    Columns 5 through 8
    0    1.0000 + 0.4142i    0    1.0000 + 2.4142i
>> ifft(fft(V))
ans =
    1    1    1    1    0    0    0    0
```

Аналогичные функции есть для двумерного и многомерного случаев:

- `ifft2(F)` производит двумерное дискретное обратное преобразование Фурье для матрицы F ;
- `ifft2(F, m, n)` производит обратное преобразование Фурье размерности $m \times n$ для матрицы F ;
- `ifftn(F)` возвращает результат N -мерного обратного дискретного преобразования Фурье для N -мерного массива F ;
- `ifftn(F, siz)` возвращает результат обратного дискретного преобразования Фурье для массива F с ограничением размера, заданным вектором `siz`. Если любой элемент `siz` меньше, чем соответствующая размерность F , то массив F будет урезан до размерности `siz`.

9.4. Свертка и дискретная фильтрация

В этом разделе рассмотрены базовые средства для проведения операций свертки и фильтрации сигналов на базе алгоритмов быстрого преобразования Фурье. Многие дополнительные операции, относящиеся к этой области обработки сигналов, можно найти в пакете прикладных программ Signal Processing Toolbox.

9.4.1. Свертка прямая и обратная

Для двух векторов x и y с длиной m и n определена операция *свертки*:

$$z(k) = \sum_{j=\max(1, k-n+1)}^{\min(k, m)} x(j)y(k-j+1).$$

В ее результате получается вектор z с длиной $(m+n-1)$. Для осуществления свертки используется функция `conv(x, y)`.

Обратная свертке функция определена как `[q, r]=deconv(z, x)`. Она фактически определяет импульсную характеристику фильтра. Если $z=\text{conv}(x, y)$, то $q=y$ и $r=0$. Если x и y – векторы с коэффициентами полиномов, то свертка эквивалентна перемножению полиномов, а обратная операция – их делению. При этом вектор q возвращает частное (фактор), а вектор r – остаток от деления полиномов.

9.4.2. Свертка двумерных массивов

Для двумерных массивов также существует функция свертки: $Z = \text{conv2}(X, Y)$ и $Z = \text{conv2}(X, Y, 'option')$.

Для двумерных массивов X и Y с размером $m_x \times n_x$ и $m_y \times n_y$, соответственно, результат двумерной свертки порождает массив размера $(m_x + m_y - 1) \times (n_x + n_y - 1)$. Во второй форме функции параметр `option` может иметь следующие значения:

- 'full' – полноразмерная свертка (используется по умолчанию);
- 'same' – центральная часть размера $m_x \times n_x$;
- 'valid' – центральная часть размера $(m_x - m_y + 1) \times (n_x - n_y + 1)$, если $(m_x \times n_x) > (m_y \times n_y)$.

Возможность изменить решение или трактовку данных с помощью параметров является свойством ряда функций системы MATLAB. Позже мы столкнемся с этой возможностью еще не раз.

9.4.3. Дискретная одномерная фильтрация

MATLAB может использоваться для моделирования работы цифровых фильтров. Для обеспечения *дискретной одномерной фильтрации* используется функция `filter` в следующих формах записи.

- `filter(B, A, X)` фильтрует одномерный массив данных X , используя дискретный фильтр, описываемый следующим конечноразностным уравнением:

$$a(1) * y(n) = b(1) * x(n) + b(2) * x(n-1) + \dots + b(nb+1) * x(n-nb) - a(2) * y(n-1) - \dots - a(na+1) * y(n-na).$$

Если $a(1)$ не равно 1, то коэффициенты уравнения нормализуются относительно $a(1)$. Когда X – матрица, функция `filter` оперирует столбцами X . Возможна фильтрация многомерного (размерности N) массива.

- `[Y, Zf]=filter(B, A, X, Zi)` выполняет фильтрацию с учетом ненулевого начального состояния фильтра Zi ; возвращает, помимо выходного сигнала Y , конечное состояние фильтра Zf .
- `filter(B, A, X, [], dim)` или `filter(B, A, X, Zi, dim)` работают в направлении размерности `dim`.

Рассмотрим типовой пример фильтрации гармонического сигнала на фоне других сигналов – файл с именем **filtdem.m** из пакета расширения Signal Processing Toolbox. На рис. 9.10 представлен кадр примера, на котором показано формирование входной совокупности сигналов в виде трех сигналов с частотами 5, 15 и 30 Гц. Показаны временная зависимость сигнала и под ней фрагмент программы, включающий команды, которые надо выполнить для этого этапа примера.

Следующий кадр (рис. 9.11) иллюстрирует конструирование фильтра с достаточно плоской вершиной амплитудно-частотной характеристики (АЧХ) и полосой частот, обеспечивающего выделение сигнала с частотой 15 Гц и подавление сигналов с частотами 5 и 30 Гц. Для формирования полосы пропускания фильтра используется функция `ellip`, а для построения АЧХ – функция `freqz` (обе – из

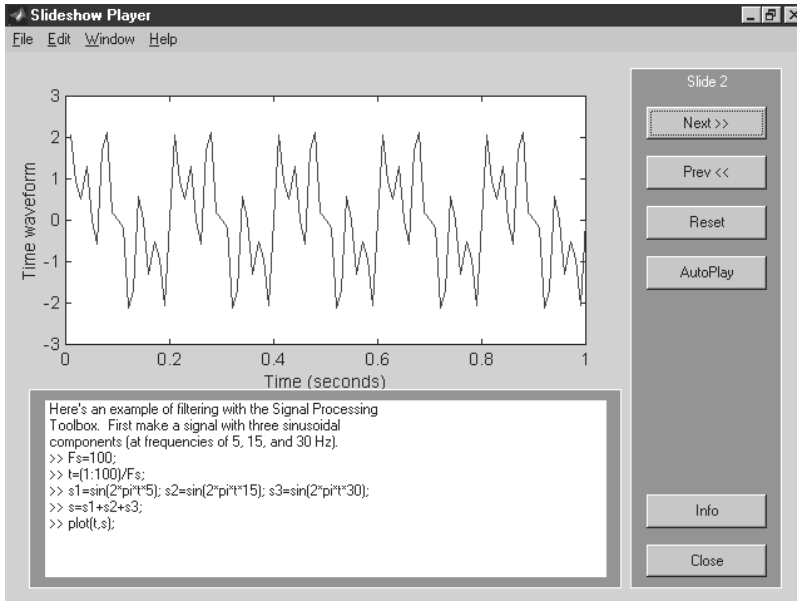


Рис. 9.10. Формирование сигнала и построение его графика

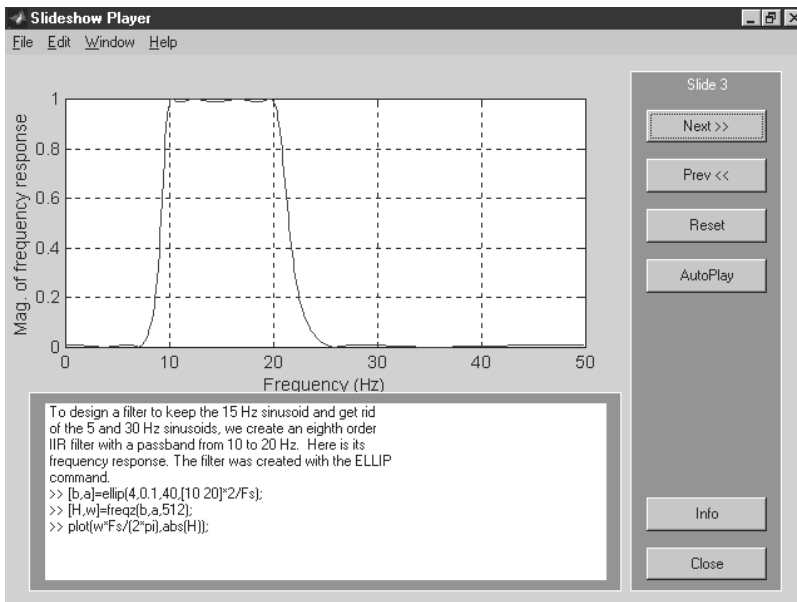


Рис. 9.11. Конструирование фильтра с заданной полосой частот и построение графика его АЧХ

пакета Signal Processing Toolbox). Это позволяет построить график АЧХ созданного фильтра.

Следующий кадр примера (рис. 9.12) иллюстрирует эффективность выделения сигнала заданной частоты (15 Гц) с помощью операции фильтрации – функции `filter`, описанной выше. Можно заметить два обстоятельства – полученный стационарный сигнал практически синусоидален, что свидетельствует о высокой степени фильтрации побочных сигналов. Однако нарастание сигнала во времени идет достаточно медленно и занимает несколько периодов частоты полезного сигнала. Характер нарастания сигнала во времени определяется переходной характеристикой фильтра.

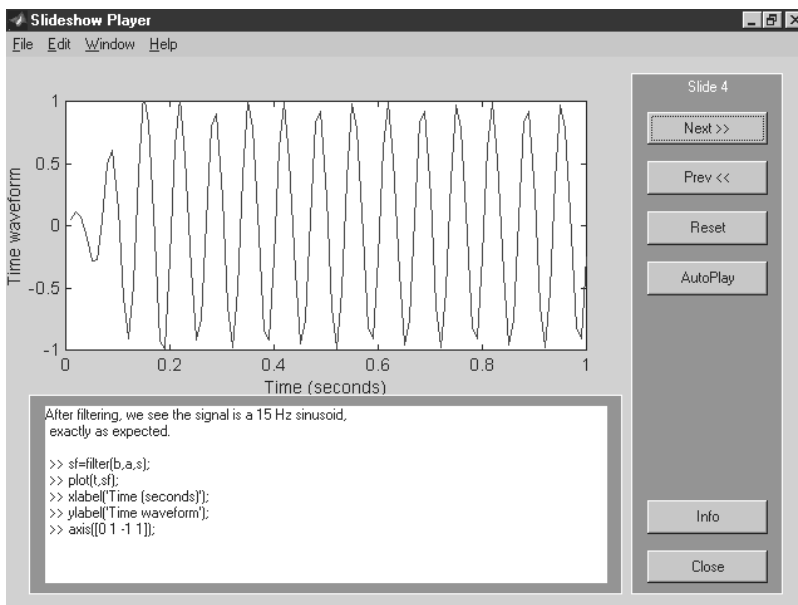


Рис. 9.12. Фильтрация и ее результат в виде временной зависимости сигнала на выходе фильтра

Заключительный кадр (рис. 9.13) показывает спектр исходного сигнала и спектр сигнала на выходе фильтра (он показан линиями другого цвета, что, к сожалению, не видно на черно-белом рисунке). Для построения спектров используется прямое преобразование Фурье – функция `fft`.

Этот пример наглядно иллюстрирует технику фильтрации. Рекомендуется просмотреть дополнительные примеры, которые есть в разделе **Demos** системы применительно к пакету расширения **Signal Processing** (если этот пакет установлен).

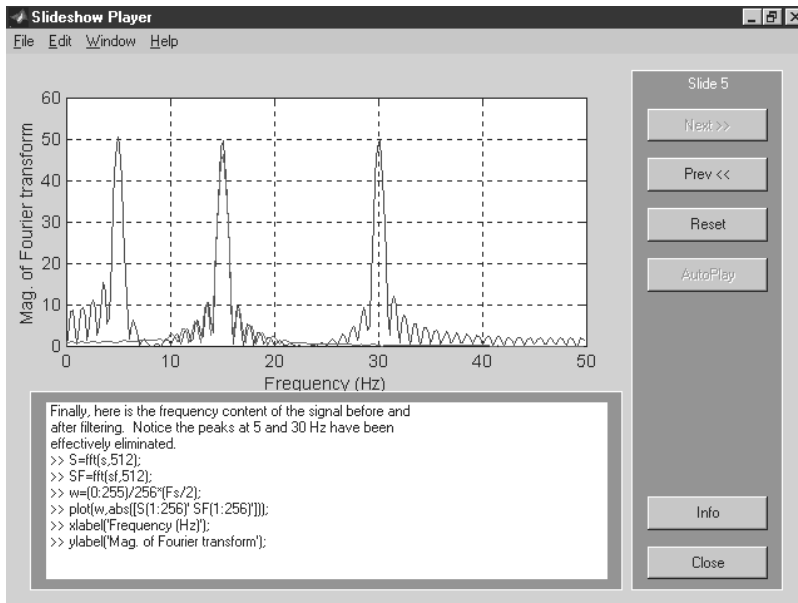


Рис. 9.13. Анализ спектров сигналов на входе и на выходе фильтра и построение их спектров

9.4.4. Двумерная фильтрация

Для осуществления двумерной фильтрации служит функция `filter2`:

- `filter2(B, X)` фильтрует данные в двумерном массиве `X`, используя дискретный фильтр, описанный матрицей `B`. Результат `Y` имеет те же размеры, что и `X`;
- `filter2(B, X, 'option')` выполняет то же, но с опцией, влияющей на размер массива `Y`:
 - 'same' – `size(Y)=size(X)` (действует по умолчанию);
 - 'valid' – `size(Y) < size(X)`, центральная часть двумерной свертки, при вычислении которой не приходится дополнять массивы нулями;
 - 'full' – `size(Y) > size(X)`, полная двумерная свертка.

9.4.5. Коррекции фазовых углов

Фазовые углы одномерных массивов испытывают разрывы при переходе через значения, кратные π . Функции `unwrap(P)` и `unwrap(P, cutoff)` устраняют этот недостаток для одномерного массива `P`, дополняя значения углов в точках разрыва значениями $\pm 2\pi$. Если `P` – двумерный массив, то данная функция применяется к столбцам. Параметр `cutoff` (по умолчанию равный π) позволяет назначить лю-

бой критический угол в точках разрыва. Функция используется при построении фазочастотных характеристик (ФЧХ) фильтров. Поскольку они строятся редко, оставим за читателем изучение практического применения данной функции.

9.5. Интерполяция и аппроксимация данных

Под аппроксимацией обычно подразумевается описание некоторой, порой не заданной явно, зависимости или совокупности представляющих ее данных с помощью другой, обычно более простой или более единообразной зависимости. Часто данные находятся в виде отдельных узловых точек, координаты которых задаются таблицей данных. График аппроксимирующей функции может не проходить через узловые точки, но приближать их с некоторой (по возможности, малой) среднеквадратической погрешностью. Это характерно для регрессии – реализации метода наименьших квадратов (МНК).

Основная задача интерполяции – оценить значение представляемой данными зависимости в промежутках между ее узловыми точками. Для этого используются подходящие функции, значения которых в узловых точках совпадают с координатами этих точек. Например, при *линейной интерполяции* зависимости $y(x)$ узловые точки просто соединяются друг с другом отрезками прямых, и считается, что искомые промежуточные точки расположены на этих отрезках.

Для повышения точности интерполяции применяют параболы (квадратичная интерполяция) или полиномы более высокой степени (полиномиальная интерполяция). Для обработки данных MATLAB использует различные функции интерполяции и аппроксимации данных. Набор таких функций вместе с несколькими вспомогательными функциями описан в этом разделе.

9.5.1. Полиномиальная регрессия

Одна из наиболее известных аппроксимаций – *полиномиальная аппроксимация*. В системе MATLAB определены функции аппроксимации данных полиномами по методу наименьших квадратов – полиномиальной регрессии. Это достаточно универсальный вид аппроксимации. Например, при степени полинома 1 мы имеем линейную регрессию, при степени полинома 2 – квадратичную и т. д.

Полиномиальную регрессию реализует функция, приведенная ниже:

- `polyfit(x, y, n)` возвращает вектор коэффициентов полинома $p(x)$ степени n , который с наименьшей среднеквадратичной погрешностью аппроксимирует функцию $y(x)$. Результатом является вектор-строка длиной $n+1$, содержащий коэффициенты полинома в порядке уменьшения степеней. Если x и y равно $n+1$, то реализуется обычная полиномиальная аппроксимация, при которой график полинома точно проходит через узловые точки с координатами (x, y) , хранящиеся в векторах x и y . В противном случае точного совпадения графика с узловыми точками не наблюдается;

- `[p, S] = polyfit(x, y, n)` возвращает коэффициенты полинома `p` и структуру `S` для использования вместе с функцией `polyval` с целью оценивания или предсказания погрешности;
- `[p, S] = polyfit(x, y, n, mu)` возвращает коэффициенты полинома `p` и структуру `S` для использования вместе с функцией `polyval` с целью оценивания или предсказания погрешности, но так, что приходится центрирование (нормирование) и масштабирование `x`, $x_{\text{norm}} = (x - \text{mu}(1)) / \text{mu}(2)$, где $\text{mu}(1) = \text{mean}(x)$ и $\text{mu}(2) = \text{std}(x)$. Центрирование и масштабирование не только улучшают свойства степенного многочлена, получаемого при помощи `polyval`, но и значительно повышают качественные характеристики самого алгоритма аппроксимации.

Пример (полиномиальная регрессия для функции $\sin(x)$):

```
>> x=(-3:0.2:3)';y=sin(x);p=polyfit(x,y,3)
p =    -0.0953    0.0000    0.8651   -0.0000
>> x=(-4:0.2:4)';y=sin(x);
>> f=polyval(p,x);plot(x,y,'o',x,f)
```

Рисунок 9.14, построенный в этом примере, дает наглядное представление о точности полиномиальной аппроксимации. Следует помнить, что она достаточно точна в небольших окрестностях от точки $x = 0$, но может иметь большие погрешности за их пределами или в промежутках между узловыми точками.

График аппроксимирующего полинома третьей степени на рис. 9.14 показан сплошной линией, а точки исходной зависимости обозначены кружками. Обратите внимание на то, что при полиномиальной регрессии узловые точки не ложатся

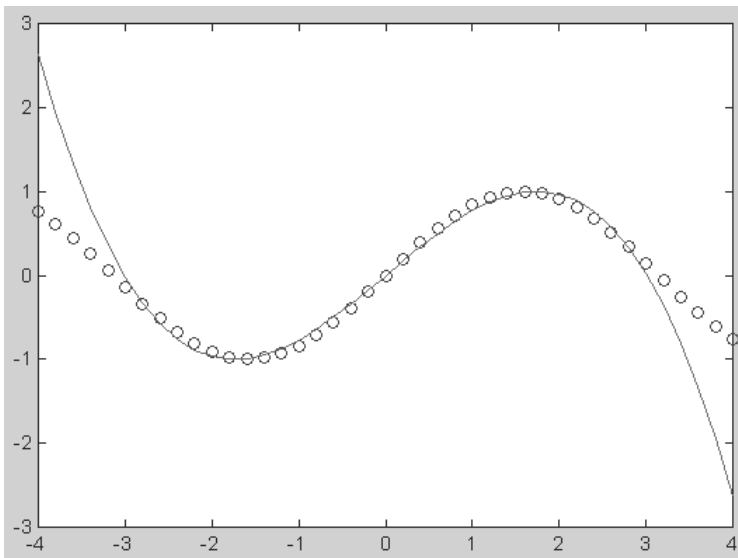


Рис. 9.14. Пример использования функции `polyfit`

точно на график полинома, поскольку их приближение к нему является наилучшим в смысле минимального среднеквадратического отклонения. Однако если степень полинома на 1 меньше числа узлов зависимости $f(x)$, то полученный полином в узловых точках дает значения, точно совпадающие (в пределах точности машинных вычислений) со значениями ординат узловых точек $f(x_i)$.

9.5.2. Фурье-интерполяция периодических функций

Под *интерполяцией* обычно подразумевают вычисление таблично заданных значений функции $f(x_i)$ в промежутках между узловыми точками с координатами x_i . Линейная, квадратичная и полиномиальная интерполяции реализуются как частные случаи полиномиальной аппроксимации. А вот для периодических (и особенно для гладких периодических) функций хорошие результаты может дать их интерполяция тригонометрическим рядом Фурье. Для этого используется следующая функция:

- `interpft(x, n)` возвращает вектор y , содержащий значения периодической функции, определенные в n равномерно расположенных точках. Если `length(x) = m` и x имеет интервал дискретизации dx , то интервал дискретизации для y составляет $dy = dx * m / n$, причем n не может быть меньше, чем m . Если X – матрица, `interpft` оперирует столбцами X , возвращая матрицу Y с таким же числом столбцов, как и у X , но с n строками. Функция `y = interpft(x, n, dim)` работает либо со строками, либо со столбцами в зависимости от значения параметра `dim`.

Пример программы интерполяции функции:

```
% Программа интерполяции функции sin(x).^3
x=0:10;
y=sin(x).^3;
x1=0:0.1:10;y1=interpft(y,101);
x2=0:0.01:10;y2=sin(x2).^3;
plot(x1,y1,'-'),hold on,plot(x,y,'o',x2,y2)
```

Рисунок 9.15 иллюстрирует эффективность данного вида интерполяции на примере функции $\sin(x) \cdot ^3$, которая представляет собой сильно искаженную синусоиду.

Исходная функция на рис. 9.15 представлена сплошной линией с кружками, а интерполирующая функция – штрихпунктирной линией.

9.5.3. Интерполяция на неравномерной сетке

Для интерполяции на неравномерной сетке используется функция, приведенная ниже.

- `ZI = griddata(x, y, z, XI, YI)` преобразует поверхность вида $z = f(x, y)$, которая определяется векторами (x, y, z) с (обычно) неравно-

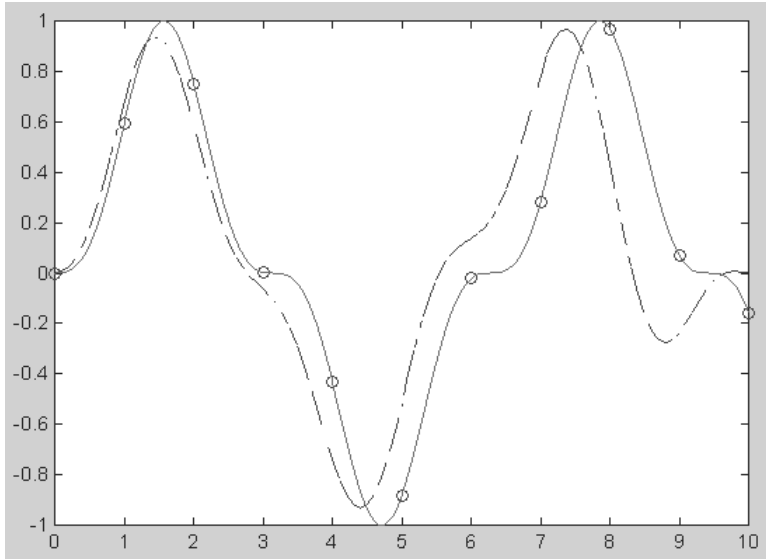


Рис. 9.15. Пример использования функции `interpft`

мерно распределенными элементами. Функция `griddata` аппроксимирует эту поверхность в точках, определенных векторами (XI , YI) в виде значений ZI . Поверхность всегда проходит через заданные точки. XI и YI обычно формируют однородную сетку (созданную с помощью функции `meshgrid`).

XI может быть вектором-строкой, в этом случае он определяет матрицу с постоянными столбцами. Точно так же YI может быть вектором-столбцом, тогда он определяет матрицу с постоянными строками.

- `[XI, YI, ZI] = griddata(x, y, z, xi, yi)` возвращает аппроксимирующую матрицу ZI , как описано выше, а также возвращает матрицы XI и YI , сформированные из вектора-столбца xi и вектора-строки yi . Последние аналогичны матрицам, возвращаемым функцией `meshgrid`.
- `[...] = griddata(..., method)` использует определенный метод интерполяции:
 - `'nearest'` – ступенчатая интерполяция;
 - `'linear'` – линейная интерполяция (принята по умолчанию);
 - `'cubic'` – кубическая интерполяция;
 - `'v4'` – метод, используемый в MATLAB 4.

Метод определяет тип аппроксимирующей поверхности. Метод `'cubic'` формирует гладкие поверхности, в то время как `'linear'` и `'nearest'` имеют разрывы первых и нулевых производных соответственно. Метод `'v4'` включен для обеспечения совместимости с версией 4 системы **MATLAB**.

Пример:

```
% Пример программы на применение функции griddata
x=rand(120,1)*4-2;
y=rand(120,1)*4-2;
z=x.*y.*exp(-x.^2-y.^2);
t=-2:0.1:2; [X,Y]=meshgrid(t,t);
Z=griddata(x,y,z,X,Y);
mesh(X,Y,Z),hold on,plot3(x,y,z,'ok')
```

Рисунок 9.16 иллюстрирует применение функции `griddata`.

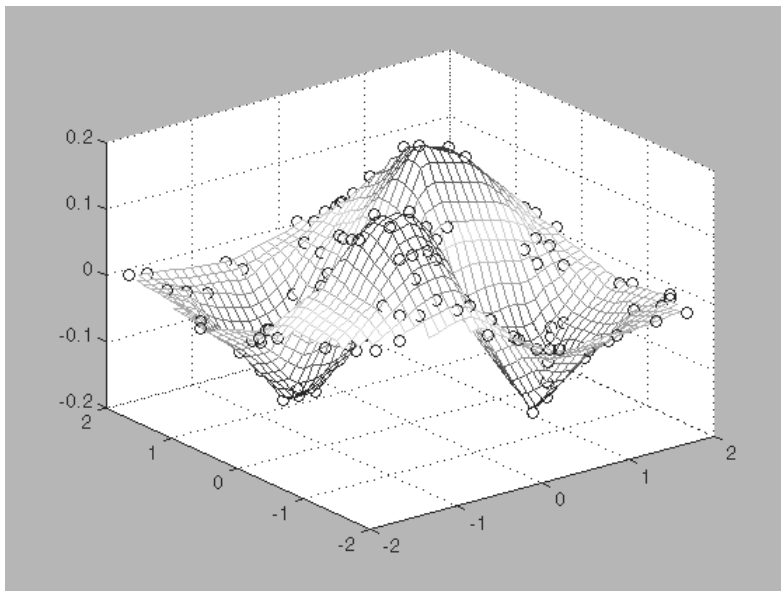


Рис. 9.16. Пример использования функции `griddata`

Функции `griddata3` и `griddatan` работают аналогично `griddata`, но для трехмерного и n -мерного случаев. Используются, в частности, при трехмерной и n -мерной триангуляции.

9.5.4. Одномерная табличная интерполяция

В ряде случаев очень удобны сплайновая интерполяция и аппроксимация таблично заданных функций. При ней промежуточные точки ищутся по отрезкам полиномов третьей степени – это *кубическая сплайновая интерполяция*. При этом обычно такие полиномы вычисляются так, чтобы не только их значения совпадали с координатами узловых точек, но также чтобы в узловых точках были непрерывны производные первого и второго порядков.

Для одномерной табличной интерполяции используется функция `interp1`:

- `yi = interp1(x, Y, xi)` возвращает вектор `yi`, содержащий элементы, соответствующие элементам `xi` и полученные интерполяцией векторов `x` и `Y`. Вектор `x` определяет точки, в которых задано значение `Y`. Если `Y` – матрица, то интерполяция выполняется для каждого столбца `Y` и `yi` имеет длину `length(xi) - by-size(Y, 2)`.
- `yi = interp1(x, Y, xi, method)` позволяет с помощью параметра `method` задать метод интерполяции:
 - `'nearest'` – ступенчатая интерполяция;
 - `'linear'` – линейная интерполяция (принята по умолчанию);
 - `'spline'` – кубическая сплайн-интерполяция;
 - `'cubic'` или `'pchip'` – интерполяция многочленами Эрмита;
 - `'v5cubic'` – кубическая интерполяция MATLAB 5;
- `yi = interp1(x, Y, xi, method, значение величин вне пределов изменения x)` позволяют отобразить особые точки на графике;
- `yi = interp1(x, Y, xi, method, 'сообщение')` позволяют изменить сообщение об особых точках на графике.

Все методы интерполяции требуют, чтобы значения `x` изменялись монотонно. Когда `x` – вектор равномерно распределенных точек, для более быстрой интерполяции лучше использовать методы `'*linear'`, `'*cubic'`, `'*nearest'` или `'*spline'`. Обратите внимание, что в данном случае наименованию метода предшествует знак звездочки.

Пример интерполяции функции косинуса):

```
% Пример интерполяции функции косинуса
x=0:10;y=cos(x); xi=0:0.1:10;
yi=interp1(x,y,xi);
plot(x,y,'x',xi,yi,'g'),hold on
yi=interp1(x,y,xi,'spline');
plot(x,y,'o',xi,yi,'m'),grid,hold off
```

Результаты интерполяции иллюстрирует рис. 9.17.

Узловые точки на рис. 9.17 обозначены кружками с наклонными крестиками. Одна из кривых соответствует линейной интерполяции, другая – сплайн-интерполяции. Нетрудно заметить, что сплайн-интерполяция в данном случае дает гораздо лучшие результаты, чем линейная интерполяция.

9.5.5. Двумерная табличная интерполяция

Двумерная интерполяция существенно сложнее, чем одномерная, рассмотренная выше, хотя смысл ее тот же – найти промежуточные точки некоторой зависимости $z(x,y)$ вблизи расположенных в пространстве узловых точек. Для *двумерной табличной интерполяции* используется функция `interp2`:

- `ZI = interp2(X, Y, Z, XI, YI)` возвращает матрицу `ZI`, содержащую значения функции в точках, заданных аргументами `XI` и `YI`, полученные путем интерполяции двумерной зависимости, заданной матрицами `X`, `Y` и `Z`. При

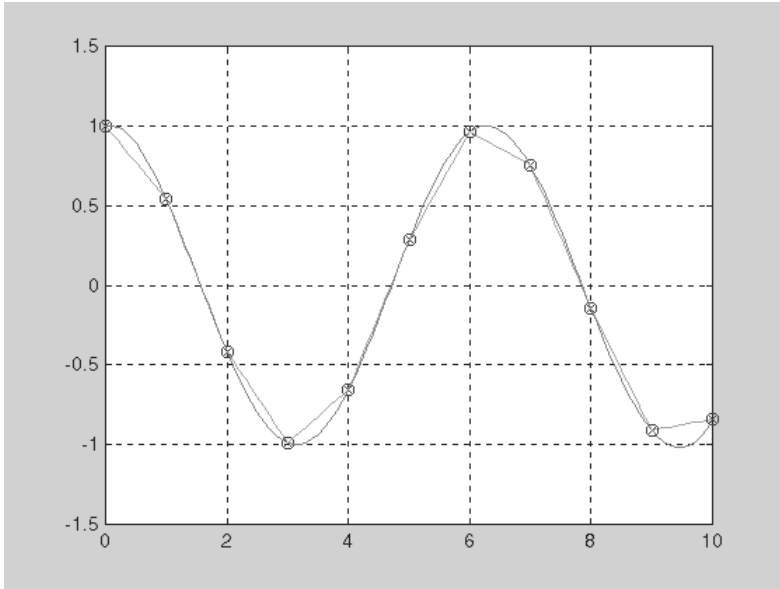


Рис. 9.17. Пример применения функции *interp1*

этом X и Y должны быть монотонными и иметь тот же формат, как если бы они были получены с помощью функции `meshgrid` (строки матрицы X являются идентичными; то же можно сказать о столбцах массива Y). Матрицы X и Y определяют точки, в которых задано значение Z . Параметры XI и YI могут быть матрицами, в этом случае `interp2` возвращает значения Z , соответствующие точкам $(XI(i, j), YI(i, j))$. В виде альтернативы можно передать в качестве параметров вектор-строку xi и вектор-столбец yi . В этом случае `interp2` представляет эти векторы так, как если бы использовалась команда `meshgrid(xi, yi)`.

- $ZI = \text{interp2}(Z, XI, YI)$ подразумевает, что $X=1:n$ и $Y=1:m$, где $[m, n]=\text{size}(Z)$.
- $ZI = \text{interp2}(Z, ntimes)$ осуществляет интерполяцию рекурсивным методом с числом шагов $ntimes$.
- $ZI = \text{interp2}(X, Y, Z, XI, YI, method)$ позволяет с помощью опции `method` задать метод интерполяции:
 - 'nearest' – интерполяция по соседним точкам;
 - 'linear' – линейная интерполяция;
 - 'cubic' – кубическая интерполяция (полиномами Эрмита);
 - 'spline' – интерполяция сплайнами.

Все методы интерполяции требуют, чтобы X и Y изменялись монотонно и имели такой же формат, как если бы они были получены с помощью функции `meshgrid`. Когда X и Y – векторы равномерно распределенных точек, для более

быстрой интерполяции лучше использовать методы `'*linear'`, `'*cubic'`, или `'*nearest'`.

Пример:

```
% Программа двумерной интерполяции
[X,Y]=meshgrid(-3:0.25:3);
Z=peaks(X/2,Y*2);
[X1,Y1]=meshgrid(-3:0.1:3);
Z1=interp2(X,Y,Z,X1,Y1);
mesh(X,Y,Z),hold on,mesh(X1,Y1,Z1+15),hold off
```

Рисунок 9.18 иллюстрирует применение функции `interp2` для двумерной интерполяции (на примере функции `peaks`) с помощью приведенной выше программы.

В данном случае поверхность снизу – двумерная линейная интерполяция, которая реализуется по умолчанию, когда не указан параметр `method`.

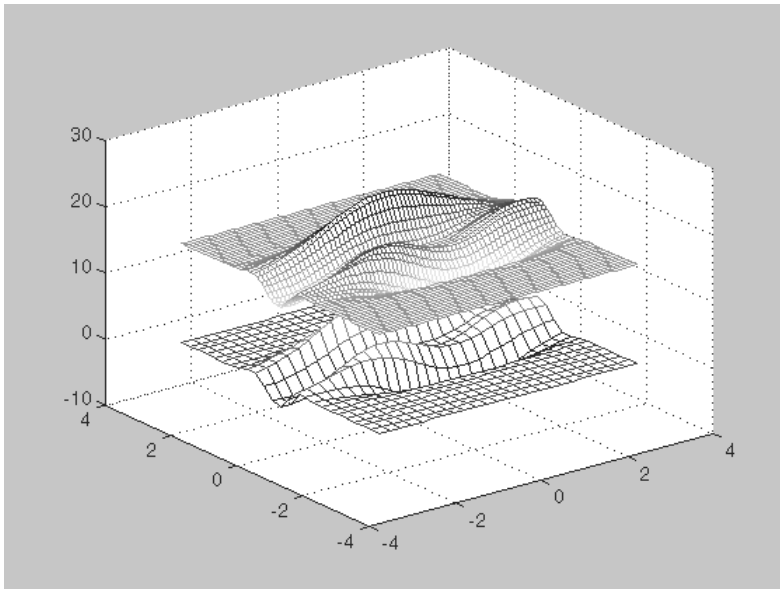


Рис. 9.18. Применение функции `interp2`

9.5.6. Трехмерная табличная интерполяция

Для трехмерной табличной интерполяции используется функция `interp3`:

- $V_I = \text{interp3}(X, Y, Z, V, X_I, Y_I, Z_I)$ интерполирует, чтобы найти V_I , значение основной трехмерной функции V в точках матриц X_I , Y_I и Z_I . Матрицы X , Y и Z определяют точки, в которых задано значение V . X_I , Y_I и Z_I

могут быть матрицами, в этом случае `interp3` возвращает значения Z , соответствующие точкам $(X_I(i, j), Y_I(i, j), Z_I(i, j))$. В качестве альтернативы можно передать векторы x_i, y_i и z_i . Векторы-аргументы, имеющие неодинаковый размер, представляются, как если бы использовалась команда `meshgrid`;

- `VI = interp3(V, XI, YI, ZI)` подразумевает $X=1:N, Y=1:M, Z=1:P$, где $[M, N, P]=\text{size}(V)$;
- `VI = interp3(V, ntimes)` осуществляет интерполяцию рекурсивным методом с числом шагов `ntimes`;
- `VI = interp3(..., method)` позволяет задать метод интерполяции:
 - `'nearest'` – ступенчатая интерполяция;
 - `'linear'` – линейная интерполяция;
 - `'cubic'` – кубическая интерполяция (полиномами Эрмита);
 - `'spline'` – интерполяция сплайнами.

Все методы интерполяции требуют, чтобы X, Y и Z изменялись монотонно и имели такой же формат, как если бы они были получены с помощью функции `meshgrid`. Когда X и Y и Z – векторы равномерно распределенных в пространстве узловых точек, для более быстрой интерполяции лучше использовать методы `'*linear', '*cubic'` или `'*nearest'`.

9.5.7. *N*-мерная табличная интерполяция

MATLAB позволяет выполнить даже n -мерную табличную интерполяцию. Для этого используется функция `interpN`:

- `VI = interpN(X1, X2, X3, ..., V, Y1, Y2, Y3, ...)` интерполирует, чтобы найти VI , значение основной многомерной функции V в точках массивов $Y1, Y2, Y3, \dots$. Функции `interpN` должно передаваться $2N+1$ аргументов, где N – размерность интерполируемой функции. Массивы $X1, X2, X3, \dots$ определяют точки, в которых задано значение V . Параметры $Y1, Y2, Y3, \dots$ могут быть матрицами, в этом случае `interpN` возвращает значения VI , соответствующие точкам $(Y1(i, j), Y2(i, j), Y3(i, j), \dots)$. В качестве альтернативы можно передать векторы $y1, y2, y3, \dots$. В этом случае `interpN` интерпретирует их, как если бы использовалась команда `ndgrid(y1, y2, y3, ...)`;
- `VI = interpN(V, Y1, Y2, Y3, ...)` подразумевает $X1=1:\text{size}(V, 1), X2=1:\text{size}(V, 2), X3=1:\text{size}(V, 3)$ и т. д.;
- `VI = interpN(V, ntimes)` осуществляет интерполяцию рекурсивным методом с числом шагов `ntimes`;
- `VI = interpN(..., method)` позволяет указать метод интерполяции:
 - `'nearest'` – ступенчатая интерполяция;
 - `'linear'` – линейная интерполяция;
 - `'cubic'` – кубическая интерполяция.

В связи с редким применением такого вида интерполяции, наглядная трактовка которой отсутствует, примеры ее использования не приводятся.

9.5.8. Интерполяция кубическим сплайном

Сплайн-интерполяция используется для представления данных отрезками полиномов невысокой степени – чаще всего третьей. При этом кубическая интерполяция обеспечивает непрерывность первой и второй производных результата интерполяции в узловых точках. Из этого вытекают следующие свойства кубической сплайн-интерполяции:

- график кусочно-полиномиальной аппроксимирующей функции проходит точно через узловые точки;
- в узловых точках нет разрывов и резких перегибов функции;
- благодаря низкой степени полиномов погрешность между узловыми точками обычно достаточно мала;
- связь между числом узловых точек и степенью полинома отсутствует;
- поскольку используется множество полиномов, появляется возможность аппроксимации функций со множеством пиков и впадин.

График интерполирующей функции при этом виде интерполяции можно формально уподобить кривой, по которой изгибается гибкая линейка, закрепленная в узловых точках. Реализуется сплайн-интерполяция следующей функцией:

- $y_i = \text{spline}(x, y, x_i)$ использует векторы x и y , содержащие аргументы функции и ее значения, и вектор x_i , задающий новые точки; для нахождения элементов вектора y_i используется кубическая сплайн-интерполяция;
- $pp = \text{spline}(x, y)$ возвращает *pp*-форму сплайна, используемую в функции `ppval` и других сплайн-функциях.

Пример:

```
% Программа сплайновой интерполяции
x=0:10; y=3*cos(x); x1=0:0.1:11;
y1=spline(x,y,x1);
plot(x,y,'o',x1,y1,'-')
```

Результат интерполяции показан на рис. 9.19.

Сплайн-интерполяция дает неплохие результаты для функций, не имеющих разрывов и резких перегибов. Особенно хорошие результаты получаются для монотонных функций. Однако при неудачном выборе узлов сплайн-интерполяция может давать большую ошибку, например выброс выгиба аппроксимирующей функции может оказаться значительным.

Ввиду важности сплайн-интерполяции и аппроксимации в обработке и представлении сложных данных в состав системы MATLAB входит пакет расширения Spline Toolbox, содержащий около 70 дополнительных функций, относящихся к реализации сплайн-интерполяции и аппроксимации, а также графического представления сплайнами их результатов. Для вызова данных об этом пакете (если он установлен) используйте команду `help splines`.

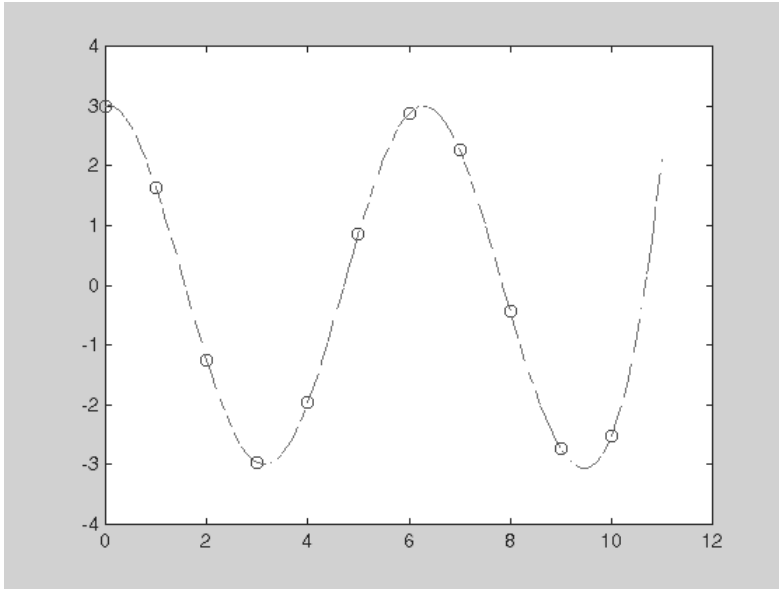


Рис. 9.19. Пример применения функции *spline*

9.6. Специальные виды интерполяции

9.6.1. Сравнение видов двумерной интерполяции поверхности

Двумерная интерполяция чаще всего выполняется для поверхностей. Возьмем, к примеру, поверхность, которую задает встроенная в MATLAB функция `peaks` (пики). Для ее построения исполним (в командной строке или в m-файле) следующие команды:

```
>> [x,y] = meshgrid(-3:1:3); z = peaks(x,y); surf(x,y,z)
```

Построенная поверхность представлена на рис. 9.20.

Очевидно, что вид поверхности получился слишком грубым, поскольку выбрано мало линий для ее представления. Увеличим число линий, задав меньший шаг:

```
>> [xi,yi] = meshgrid(-3:0.25:3);
```

и проведем построение поверхности с опцией «nearest», задающей ступенчатую интерполяцию:

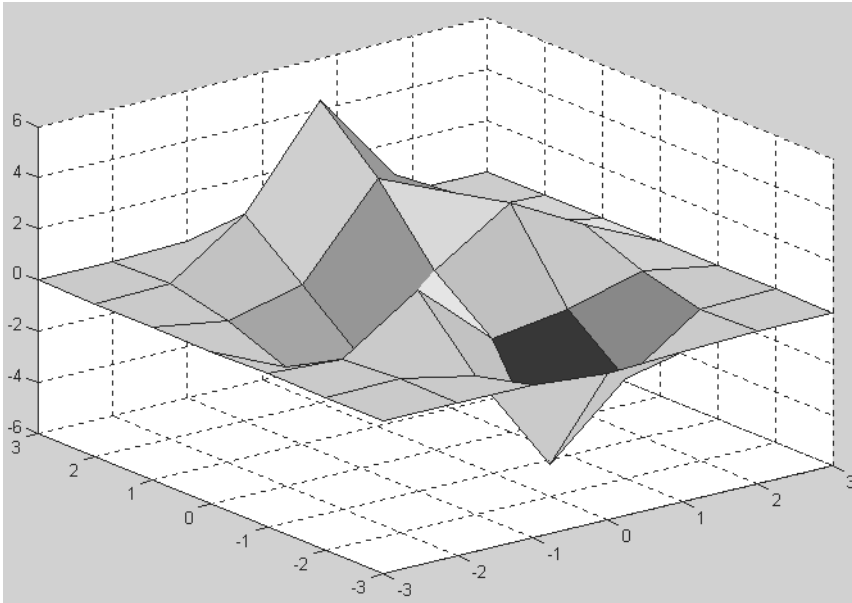


Рис. 9.20. Грубое построение поверхности *peaks*

```
>> zi1 = interp2(x,y,z,xi,yi, 'nearest');
>> surf(xi,yi,zi1)
```

Построенная для этого случая поверхность представлена на рис. 9.21.

Поверхность теперь выглядит более представительно, но состоит как бы из прямоугольных кирпичиков.

Теперь построим ту же поверхность, используя билинейную интерполяцию:

```
>> zi2 = interp2(x,y,z,xi,yi, 'bilinear ');
>> surf(xi,yi,zi2)
```

Эта поверхность представлена на рис. 9.22. Представляется, что для данного вида поверхности (пики) она выглядит более привлекательно, отображая острые пики. Однако тут явный перебор – пики выглядят неестественно остро.

Можно предположить, что заметно лучшие результаты даст сплайновая интерполяция с применением кубических сплайнов. Для ее реализации надо выполнить следующие команды:

```
>> zi3 = interp2(x,y,z,xi,yi, 'bicubic ');
>> surf(xi,yi,zi3)
```

Построенная для этого случая поверхность представлена на рис. 9.23.

Теперь уже, несмотря на небольшое число линий сетки поверхности, она выглядит вполне реалистично, что свидетельствует о высокой эффективности сплайновой двумерной интерполяции. Единственное, что сдерживает ее монопольное применение, – относительно большое время выполнения. И если на со-

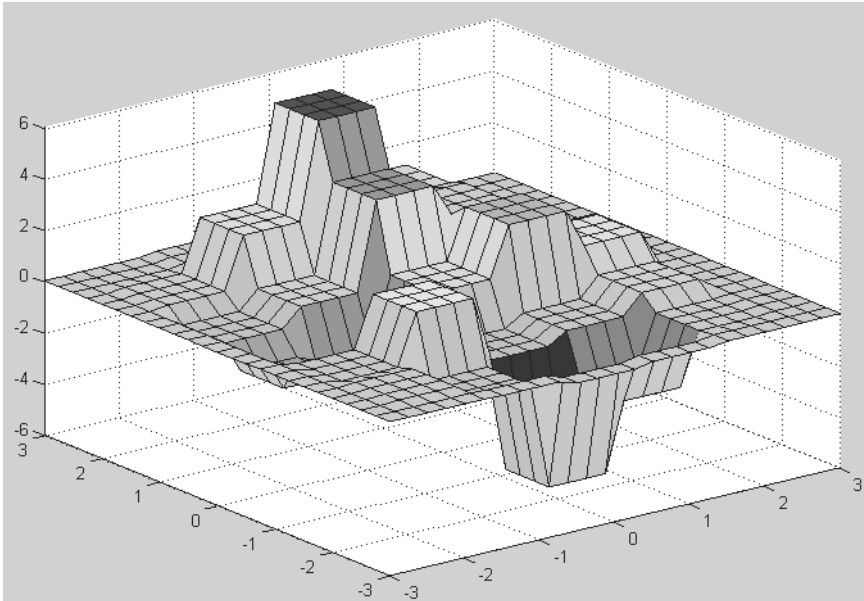


Рис. 9.21. Построение поверхности *reaks* при ступенчатой интерполяции – опция 'nearest'

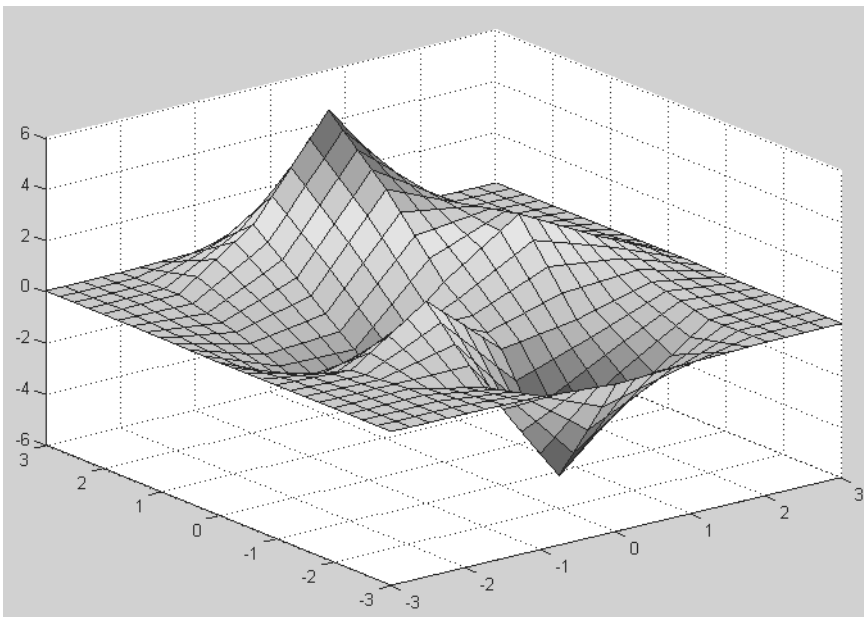


Рис. 9.22. Построение поверхности *reaks* при билинейной интерполяции – опция 'bilinear'

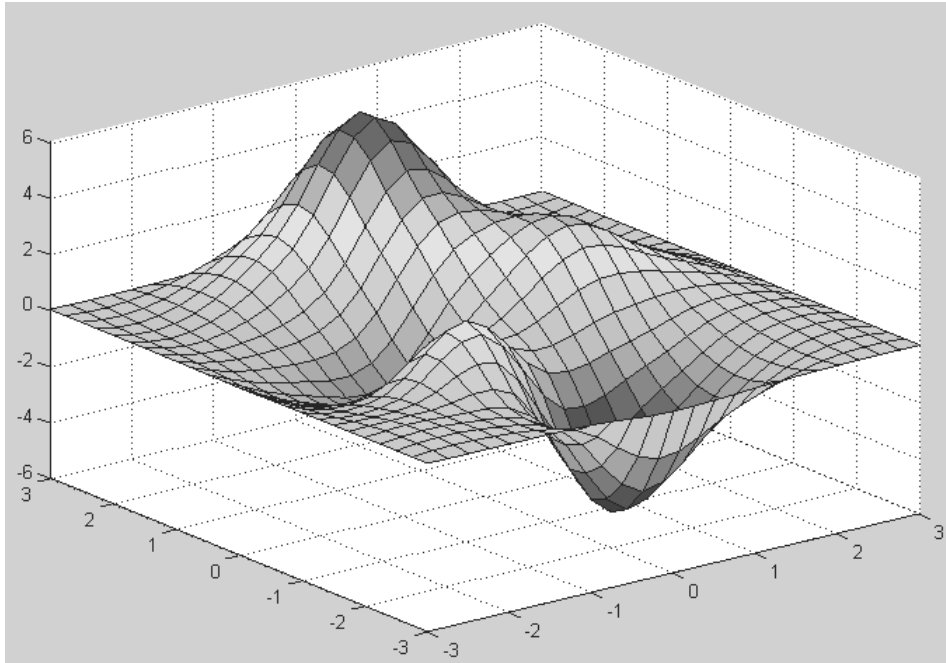


Рис. 9.23. Построение поверхности *peaks* при кубической сплайновой интерполяции – опция *'bicubic'*

временных ПК с процессорами Pentium III или IV это время мало при небольшом числе линий сетки, то в более сложных случаях, когда число линий сетки достигает десятков и сотен, это время может оказаться весьма значительным.

9.6.2. Сравнение видов интерполяции при контурных графиках

Часто поверхности представляют контурными графиками в виде линий равного уровня. Для таких графиков также полезно применение интерполяции. Представленный ниже фрагмент программы строит контурные графики для трех уже рассмотренных выше видов двумерной интерполяции:

```
subplot(1,3,1), contour (xi,yi,zi1), title ('nearest')
subplot(1,3,2), contour (xi,yi,zi2), title ('bilinear')
subplot(1,3,3), contour (xi,yi,zi3), title ('bicubic')
```

Эти графики (рис. 9.24) строятся в одном окне с титульными надписями, указывающими на вид интерполяции.

Преимущество сплайновой интерполяции в смысле качества построения здесь также вполне очевидно.

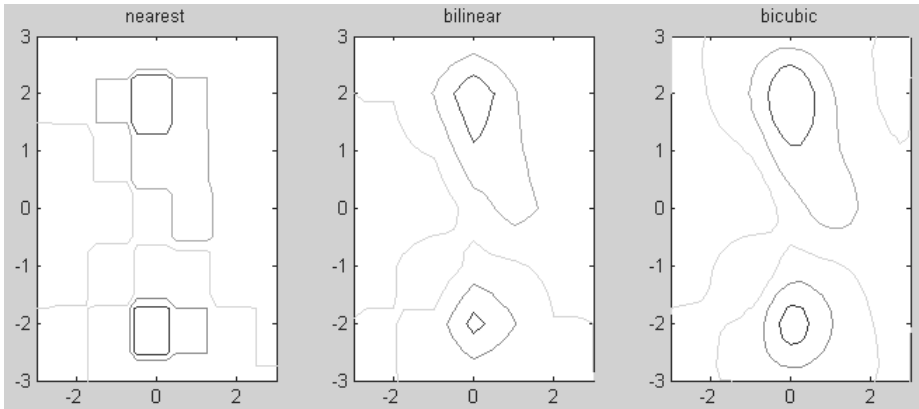


Рис. 9.24. Построение поверхности *peaks* при кубической сплайновой интерполяции – опция 'bicubic'

9.6.3. Пример многомерной интерполяции

Хороший пример многомерной интерполяции можно найти в справке MATLAB по разделу *Interpolation and Multidimensional Arrays*:

```
x1 = -2:0.2:2; x2 = -2:0.25:2; x3 = -2:0.16:2;
[X1,X2,X3] = ndgrid(x1,x2,x3);
z = X2.*exp(-X1.^2 -X2.^2 -X3.^2);
slice(X2,X1,X3,z,[-1.2 0.8 2],2,[-2 0.2])
```

Построенный этим фрагментом программы график представлен на рис. 9.25.

9.6.4. 3D-геометрический анализ и интерполяция

Рассмотренные выше в этом уроке средства геометрического анализа также можно отнести к специальным видам интерполяции и распространить на случай многомерных данных. Описание реализующих геометрический анализ функций уже было дано, так что мы переходим к 3D-геометрическому анализу и интерполяции.

Следующие команды строят точки тестовой поверхности *seamount* (рис. 9.26):

```
load seamount; plot(x,y, '.', 'markersize',12)
xlabel('Longitude'), ylabel('Latitude')
grid on
```

Теперь выполним триангуляцию Делоне, представляющую проекцию поверхности на плоскости с треугольными ячейками (рис. 9.27):

```
tri = delaunay(x,y);
hold on, triplot(tri,x,y), hold off
```

Наконец, построим поверхность в трехмерном представлении (рис. 9.28):

```
figure; hidden on; trimesh(tri,x,y,z); grid on
```

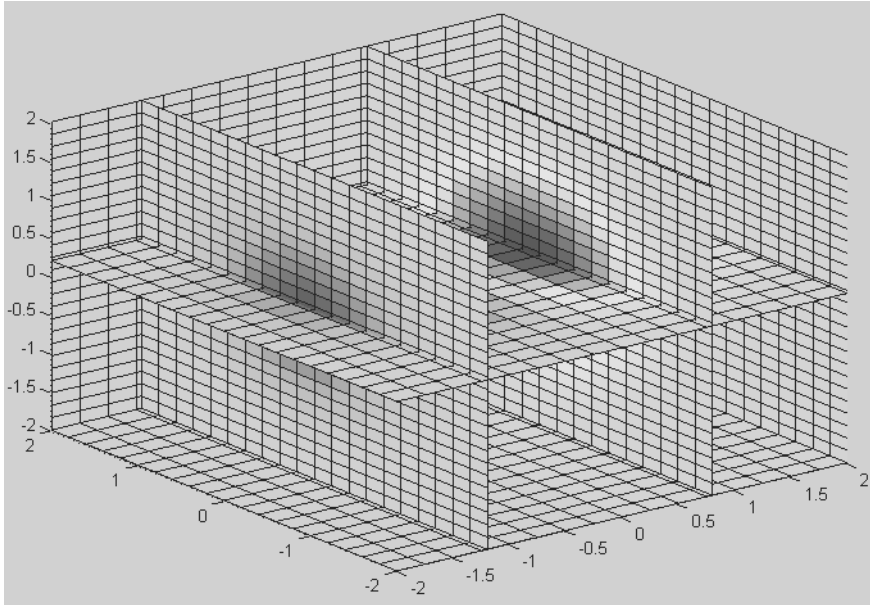


Рис. 9.25. Построение графика, представленного многомерным массивом

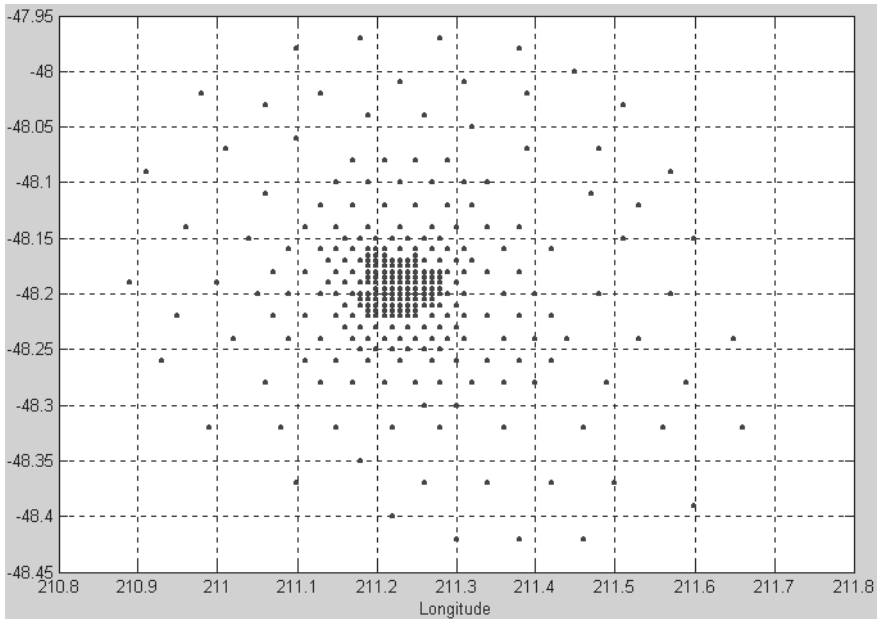


Рис. 9.26. Построение исходных точек поверхности

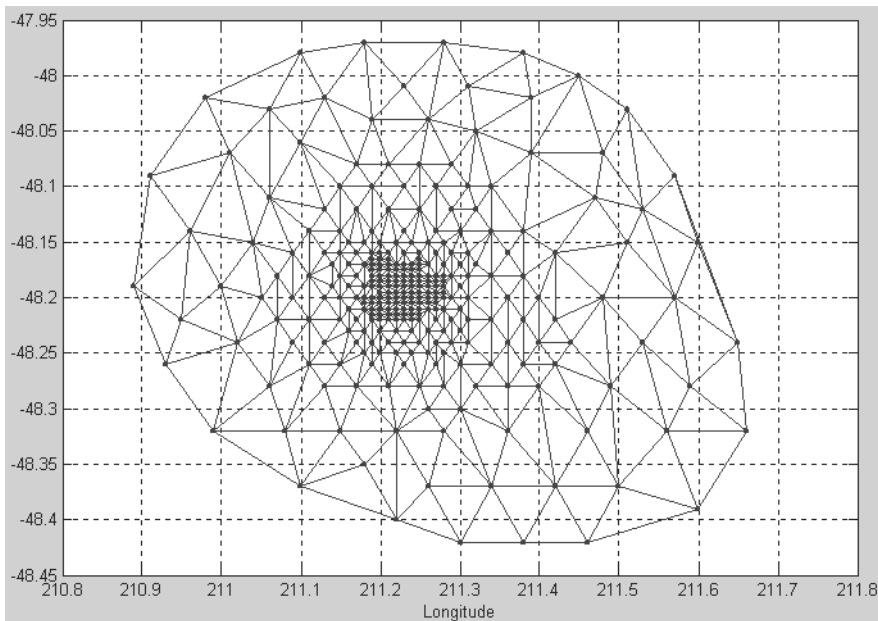


Рис. 9.27. Проведение триангуляции Делоне на плоскости

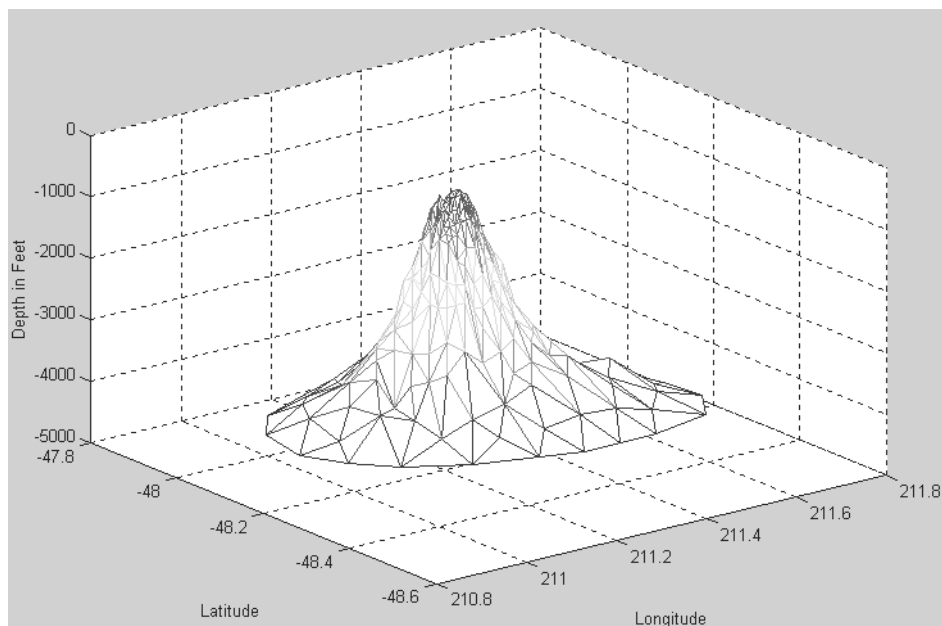


Рис. 9.28. Поверхность, построенная с применением треугольных ячеек

```
xlabel('Longitude'); ylabel('Latitude');
zlabel('Depth in Feet')
```

Несмотря на ограниченное число точек поверхности, ее вид вполне реалистичен. Теперь построим контурный график данной поверхности (рис. 9.29):

```
figure;
[xi,yi] = meshgrid(210.8:.01:211.8,-48.5:.01:-47.9);
zi = griddata(x,y,z,xi,yi,'cubic');
[c,h] = contour(xi,yi,zi,'b-');
clabel(c,h); xlabel('Longitude'), ylabel('Latitude')
```

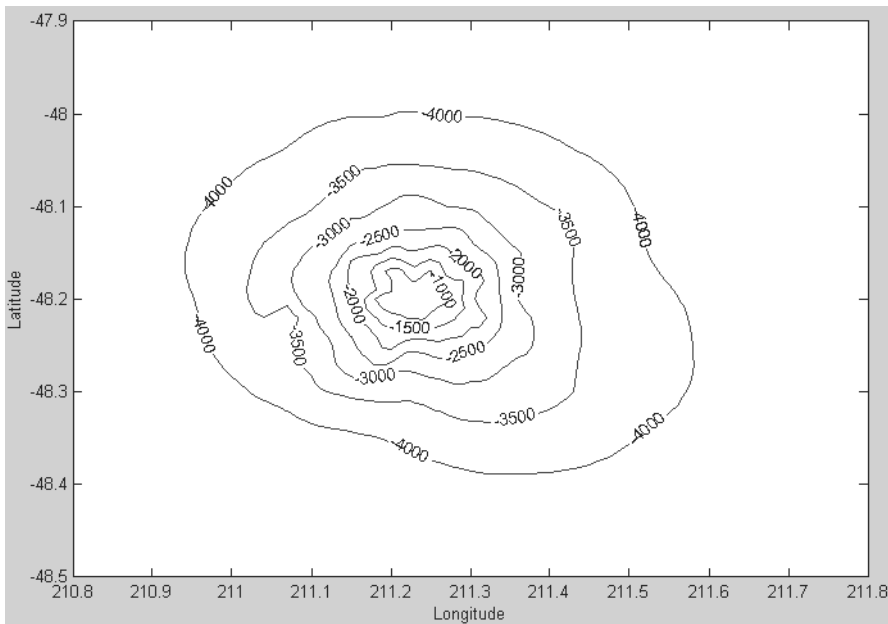


Рис. 9.29. Поверхность *seamount*, представленная контурным графиком

Как нетрудно заметить, контурный график поверхности также достаточно представителен. Разумеется, чем больше треугольных ячеек используется в ходе триангуляции Делоне, тем лучше выглядят представления поверхностей (но они дольше строятся).

9.6.5. Другие представления сложных фигур

Все представления фигур на основе геометрического анализа могут использоваться для построения 3D-фигур. Мы ограничимся двумя примерами. В первом примере для построения куба используется функция `convhulln` задания выпуклой оболочки:

```

% Программа построения куба
d = [-1 1]; [x,y,z] = meshgrid(d,d,d);
X = [x(:),y(:),z(:)]; C = convhulln (X);
figure, hold on; d = [1 2 3 1]
for i = 1: size(C,1)
    j= C(i,d);
    h(i)=patch(X(j,1),X(j,2),X(j,3),i,'FaceAlpha',0.9);
end
hold off; view(3), axis equal, axis off;
camorbit(90,-5); title('Convex hull of a cube')

```

Построенная этой программой фигура представлена на рис. 9.30.

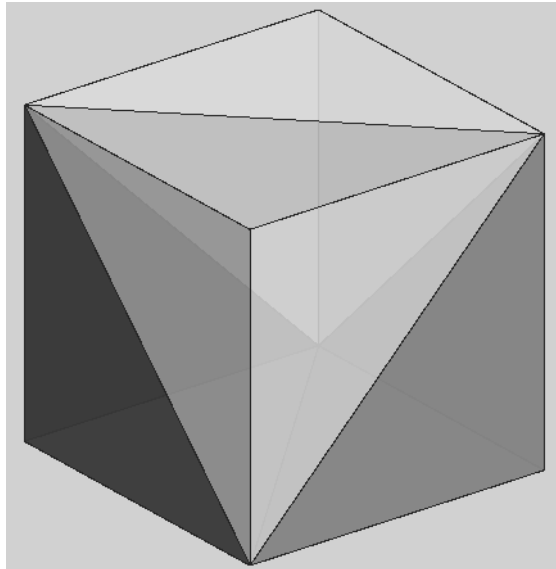


Рис. 9.30. Фигура куба,
построенная с помощью функции *convhulln*

В другом примере строится сфера на основе мозаики Делоне с эффектами ее освещения:

```

% Программа построения сферы с эффектами ее освещения
n = 5000; X = 2*rand(n,3)-1; v = sum(X.^2,2);
delta = 0.1; d = -1:delta:1;
[x0,y0,z0] = meshgrid(d,d,d); X0 = [x0(:), y0(:), z0(:)];
v0 = griddatan(X,v,X0); v0 = reshape(v0, size(x0));
p = patch(isosurface(x0,y0,z0,v0,0.6));
isonormals(x0,y0,z0,v0,p);
set(p,'FaceColor','red','EdgeColor','none');

```



```
view(3); camlight; lighting phong; axis equal  
title('Interpolated sphere from scattered data')
```

Построенная картина выглядит довольно реалистично (рис. 9.31) и хорошо отражает игру света на поверхности сферы (особенно при представлении в цвете).

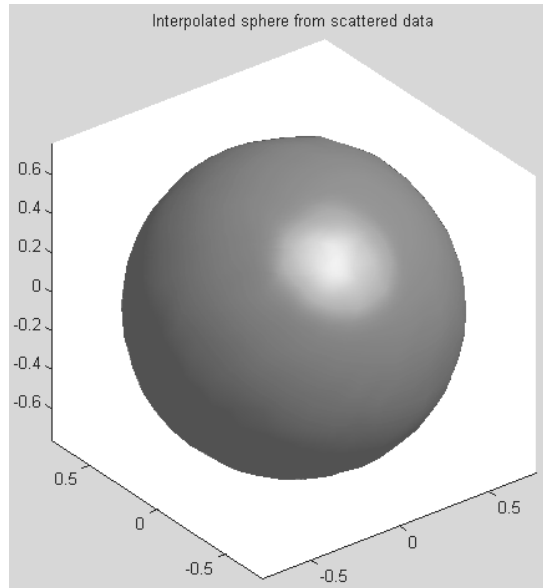


Рис. 9.31. Сфера, освещаемая источником света с бликами света на поверхности

Следует отметить, что построения, подобные представленному на рис. 9.31, выполняются довольно медленно и при ограниченных аппаратных ресурсах ПК могут даже вызвать его зависание. Не случайно подобные построения в настоящее время возлагаются на математические и графические сопроцессоры, которые имеют намного более высокие скорости работы, чем даже система MATLAB.

9.7. Обработка данных в графическом окне

9.7.1. Доступ к средствам обработки данных в графическом окне

Решение большинства задач интерполяции и аппроксимации функций и табличных данных обычно сопровождается их визуализацией. Она, как правило, заключается в построении узловых точек функции (или табличных данных) и в построении

функции аппроксимации или интерполяции. Для простых видов аппроксимации, например полиномиальной, желательно нанесение на график формулы, полученной для аппроксимации.

В MATLAB 6.* / 7.* совмещение функций аппроксимации с графической визуализацией доведено до логического конца – предусмотрена аппроксимация рядом методов точек функции, график которой построен. И все это выполняется прямо в окне редактора графики Property Editor. Для этого в позиции **Tools** графического окна имеются две новые команды:

- **Basic Fitting** – основные виды аппроксимации (регрессии);
- **Data Statistics** – статистические параметры данных.

Команда **Basic Fitting** открывает окно, дающее доступ к ряду видов аппроксимации и регрессии: сплайновой, эрмитовой и полиномиальной со степенями от 1 (линейная аппроксимация) до 10. В том числе со степенью 2 (квадратичная аппроксимация) и 3 (кубическая аппроксимация). Команда **Data Statistics** открывает окно с результатами простейшей статистической обработки данных.

9.7.2. Полиномиальная регрессия для табличных данных

Пусть некая зависимость $y(x)$ задана векторами координат ее точек:

```
>> X=[2,4,6,8,10,12,14];  
>> Y=[3.76,4.4,5.1,5.56,6,6.3,6.7];  
>> plot(X,Y,'o');
```

Рисунок 9.32 показывает пример выполнения полиномиальной регрессии (аппроксимации) для степеней полинома 1, 2 и 3. Иными словами, выполняются линейная, параболическая и кубическая регрессии.

Внимание!

При проведении полиномиальной аппроксимации надо помнить, что максимальная степень полинома на 1 меньше числа точек, то есть числа элементов в векторах X и Y.

Поясним, что же показано на рис. 9.32. В левом верхнем углу сессии MATLAB видна запись исходных векторов и команды построения заданных ими точек кружками (окно слева). Исполнив команду **Tools** \Rightarrow **Basic Fiting**, можно получить окно регрессии (оно показано справа). В этом окне птичкой отмечены три упомянутых выше вида полиномиальной регрессии. Установка птички у параметра Show equations выводит в графическом окне записи уравнений регрессии.

По команде **Tools** \Rightarrow **Data Statistics** выводится окно с рядом статистических параметров для данных, представленных векторами X и Y. Отметив птичкой тот или иной параметр в этом окне (оно показано на рис. 9.33 под окном графики), можно наблюдать соответствующие построения на графике, например вертикалей с минимальным, средним, срединным и максимальным значениями y и горизонталей с минимальным, средним, срединным и максимальным значениями x .

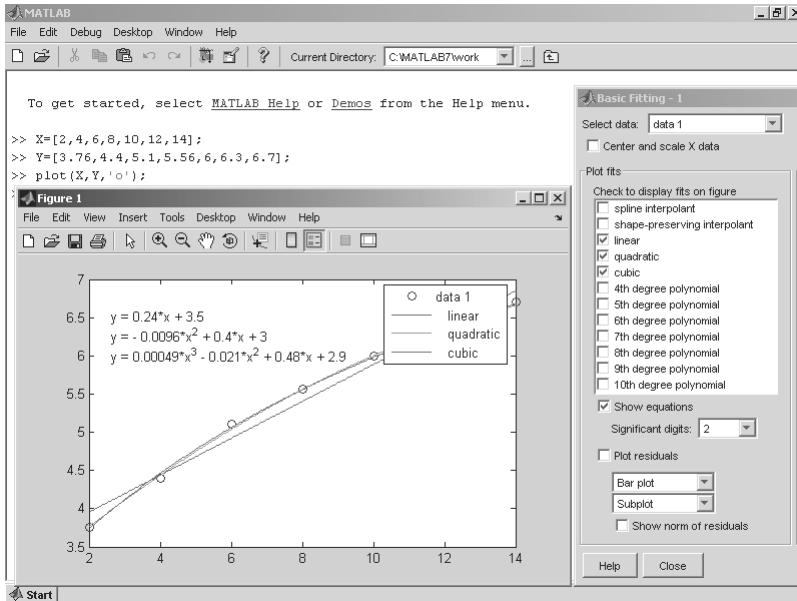


Рис. 9.32. Пример обработки табличных данных в графическом окне

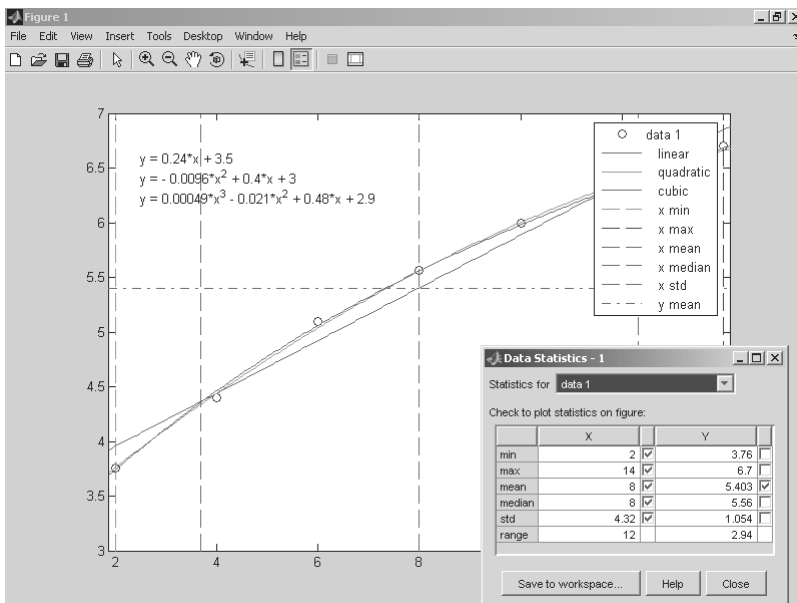


Рис. 9.33. Пример получения статистических данных о графике

9.7.3. Оценка погрешности аппроксимации

Средства обработки данных из графического окна позволяют строить столбиковый или линейчатый графики погрешностей в узловых точках и наносить на эти графики норму погрешности. Норма дает статистическую оценку среднеквадратической погрешности, и чем она меньше, тем точнее аппроксимация. Для вывода графика погрешности надо установить птичку у параметра **Plot residuals (График погрешностей)** и в меню ниже этой опции выбрать тип графика – см. рис. 9.34.

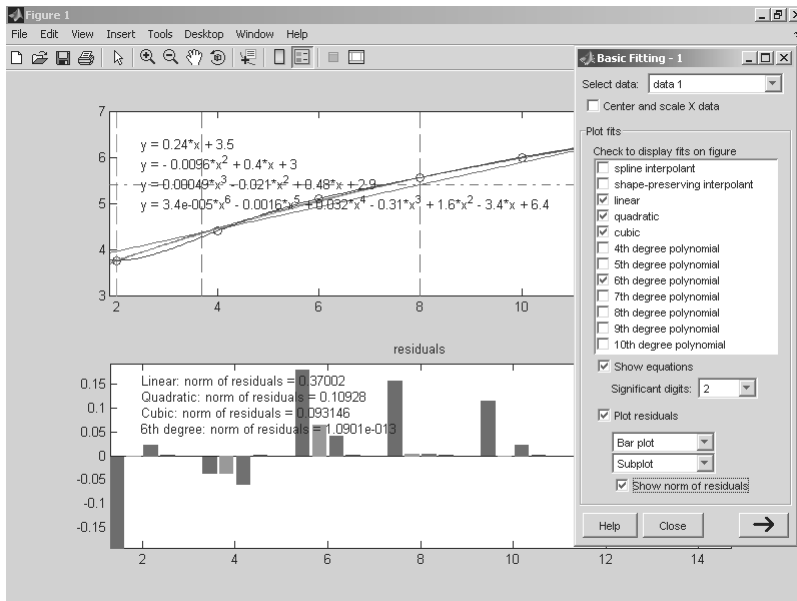


Рис. 9.34. Пример вывода данных обработки со столбцовым графиком погрешности

На рис. 9.34 приведены данные по полиномиальной аппроксимации степеней 1, 2, 3 и 6. Последний случай предельный, поскольку максимальная степень полинома должна быть на 1 меньше числа точек (их 7). В этом случае регрессия вырождается в обычную (без статистической обработки) полиномиальную аппроксимацию. При ней линия графика аппроксимирующей функции точно проходит через узловые точки, а погрешность в них равна 0 (точнее, ничтожно мала).

Рисунок 9.35 демонстрирует построение графика погрешности отрезками линий. Кроме того, опцией **Separate figure (Разделить фигуры)** задано построение графика погрешности в отдельном окне – оно расположено под графиком узловых точек и функций аппроксимации.

Таким образом, интерфейс графического окна позволяет выполнять эффективную обработку данных наиболее распространенными способами.

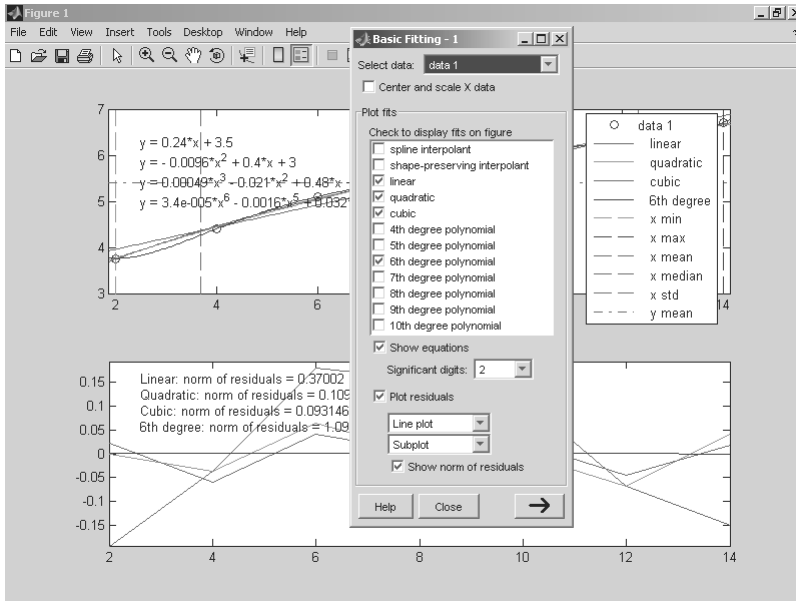


Рис. 9.35. Пример обработки табличных данных с выводом графиков погрешностей в отдельном окне

9.7.4. Расширенные возможности окна приближения кривых

На рис. 9.35 окно приближения кривых Basic Fitting представлено в упрощенном виде. В левом нижнем его углу можно заметить кнопку с жирной стрелкой \triangleright , указывающей на возможность расширения окна до двух и даже трех панелей. На рис. 9.36 показано расширенное до трех панелей окно **Basic Fitting**.

Первая панель для задания типа приближения и вывода данных о погрешности уже была описана. Вторая панель **Numerical Results (Численные результаты)** содержит список приближений, в котором можно задать выбранное приближение, например линейное, если задана позиция списка **linear**. После выбора типа приближения для него выводятся выражение для приближения, значения коэффициентов и значение нормы.

В третьей панели **Find Y=f(X)** для выбранного приближения кривой можно найти значения Y по заданным значениям X . Соответствующие точки $Y(X)$ помечаются на графике жирными ромбами – см. рис. 9.37. Пример дан для задания вектора $X=2.5:2:14$.

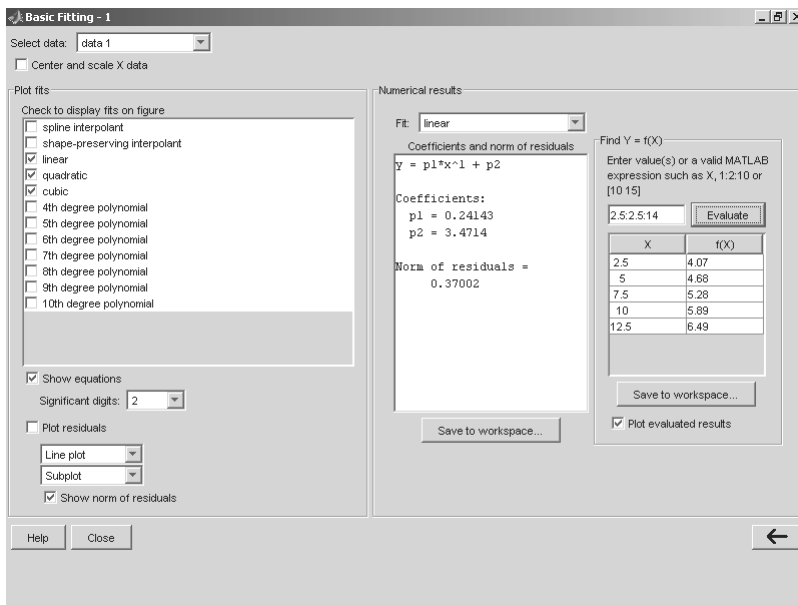


Рис. 9.36. Окно Basic Fitting с тремя панелями

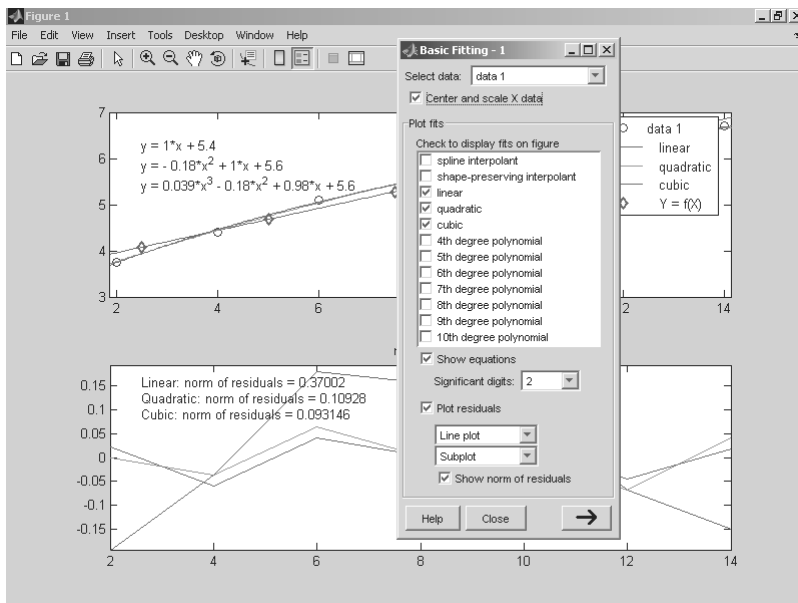


Рис. 9.37. Результаты приближения с указанием заданных точек графика линейного приближения и выводом данных о погрешности

9.7.5. Сплайновая и эрмитовая интерполяции в графическом окне

Теперь рассмотрим пример сплайновой интерполяции в графическом окне. Зададим 50 точек синусоидальной функции, как это показано в левом верхнем углу рис. 9.38.

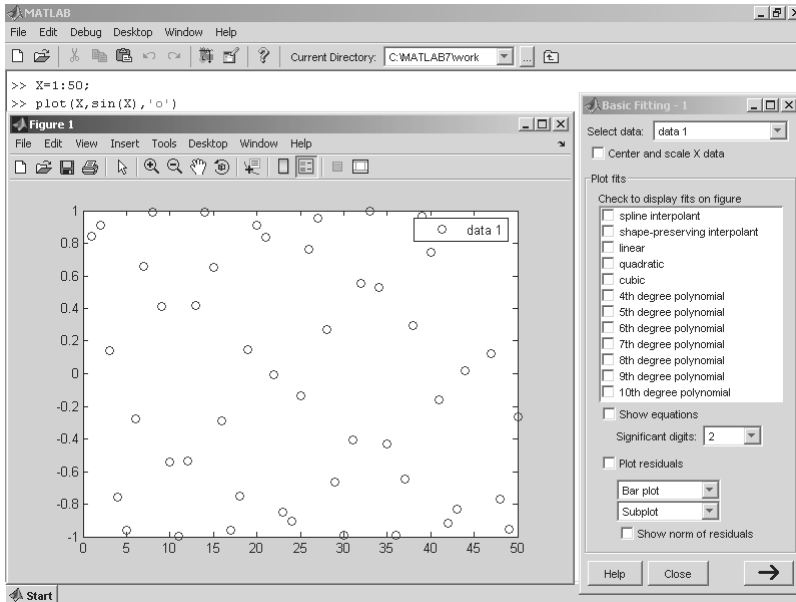


Рис. 9.38. Пример для аппроксимации синусоиды в графическом окне

Как видно по рис. 9.38, при построении исходной функции по точкам невозможно судить о ее форме. Точки оказываются разбросанными по полю рисунка, и создается впечатление (кстати, абсолютно ложное) об их случайном расположении. Попытка аппроксимации полиномом 8-ой степени не дает положительного результата – кривая проходит внутри облака точек, совершенно не представляя их.

Однако картина кардинально меняется, стоит применить сплайновую интерполяцию. На этот раз кусочная линия интерполяции (рис. 9.39) прекрасно проходит через все точки и поразительно напоминает синусоиду. Даже ее пики со значениями 1 и -1 воспроизводятся удивительно точно, причем даже в случаях, когда на них нет узловых точек.

Причина столь великолепного результата кроется в известных особенностях сплайновой интерполяции – она выполняется по трем ближайшим точкам, причем эти тройки точек постепенно перемещаются от начала точечного графика функции к ее концу. Кроме того, непрерывность первой и второй производных

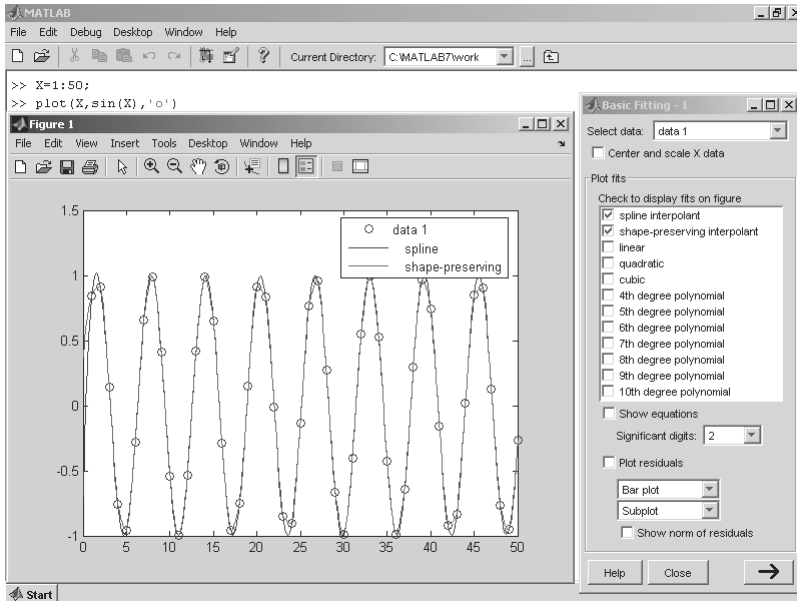


Рис. 9.39. Примеры сплайновой и эрмитовой интерполяции в графическом окне

при сплайновой интерполяции делает кривую очень плавной, что характерно и для первичной функции – синусоиды. Так что данный пример просто является удачным случаем применения сплайновой интерполяции.

MATLAB дает возможность в графическом окне использовать еще один вид многоинтервальной интерполяции на основе полиномов третьей степени Эрмита – *hermite interpolant*. В MATLAB 7 название интерполяции заменено на «*shape-preserving interpolant*». Техника интерполяции здесь та же, что и в случае сплайновой интерполяции. Это показывает рис. 9.39, где дан пример и эрмитовой интерполяции. Полиномы Эрмита имеют более гибкие линии, чем сплайны. Они точнее следуют за отдельными изгибами исходной зависимости. Это хорошо видно по рис. 9.39.

Мы не можем практически называть этот подход полноценной аппроксимацией, поскольку в данном случае нет единого выражения для аппроксимирующей функции. На каждом отрезке приближения используется кубический полином с новыми коэффициентами. Поэтому и вывода аппроксимирующей функции в поле графика не предусмотрено.

Можно сделать вывод, что сплайновая интерполяция лучше, когда нужно эффективное сглаживание быстро меняющихся от точки к точке данных и когда исходная зависимость описывается линиями, которые мы наблюдаем при построении их с помощью гибкой линейки. Эрмитова интерполяция лучше отслеживает быстрые изменения исходных данных, но имеет худшие сглаживающие свойства.

9.7.6. Графическая визуализация разложения в ряд Тейлора

Нетрудно подметить, что в средствах обработки данных в графическом окне отсутствует самая распространенная аппроксимация различных аналитических зависимостей рядом Тейлора. Но для этой аппроксимации MATLAB имеет учебное приложение `taylortool`. При его запуске этой командой появляется окно, показанное на рис. 9.40.

Работа с окном вполне очевидна – есть области для задания подлежащей приближению функции, числа членов ряда, установки сдвига a точки, относительно которой вычисляется разложение (при $a = 0$ реализуется ряд Маклорена), и пределов изменения x . В окне показываются графики исходной зависимости (сплошной линией), график разложения в ряд и формула разложения.

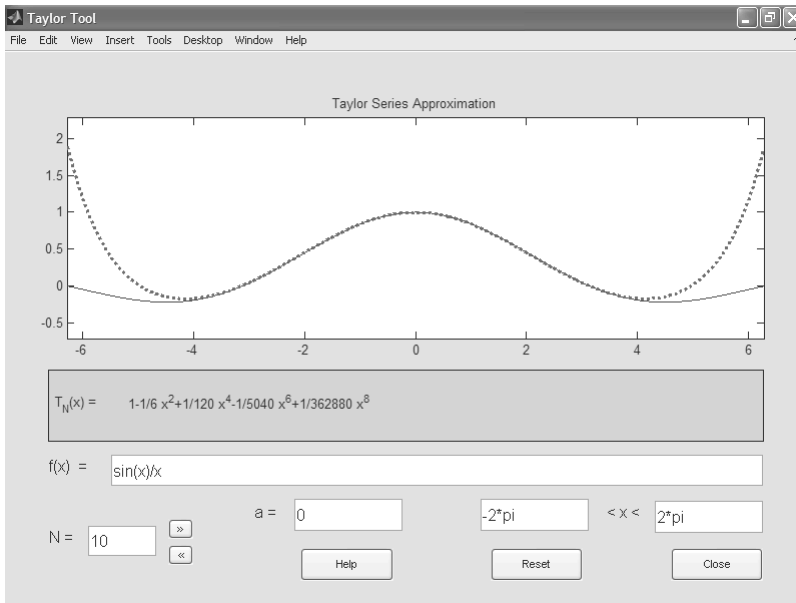


Рис. 9.40. Окно GUI аппроксимации рядом Тейлора

Работа со строками, файлами и звуками

| | |
|---|-----|
| 10.1. Обработка строковых данных | 494 |
| 10.2. Работа с файлами | 504 |
| 10.3. Работа с файлами изображений | 518 |
| 10.4. Работа со звуковыми данными | 526 |

Хотя MATLAB – прежде всего система для численных расчетов, важное значение (особенно при программировании) имеет работа с символьными данными – строками и данными файлового типа. В наш век мультимедиа большое значение имеет и работа со звуковой информацией. Всему этому и посвящен данный урок.

10.1. Обработка строковых данных

MATLAB имеет программные средства для обработки не только числовой, но и различной другой информации. Прежде всего это *строки* – цепочки из алфавитных символов. Эти средства широко применяются для создания баз данных, условных выражений, текстовых комментариев и т. д. Большие возможности есть и в обработке файлов и звуков. Этот урок посвящен возможностям обработки информации этих видов программными средствами системы MATLAB.

10.1.1. Основные функции обработки строк

В основе представления символов в строках лежит их кодирование с помощью сменных *таблиц кодов*. Такие таблицы ASCII американского кода представления информации в информационных системах ставят в однозначное соответствие каждому символу некоторый код со значением от 0 до 255.

Вектор, содержащий строку символов, в системе MATLAB задается следующим образом:

- $S = \text{'Any Characters'}$ – вектор, компонентами которого являются числовые коды, соответствующие символам.

Первые 127 чисел – это коды ASCII, представляющие буквы латинского языка, цифры и спецзнаки. Они образуют основную таблицу кодов. Вторая таблица (коды от 128 до 255) является дополнительной и может использоваться для представления символов других языков, например русского. Длина вектора S соответствует числу символов в строке, включая пробелы. Апостроф ' внутри строки символов должен вводиться как два апострофа ''.

Внимание!

Правильная работа строковых функций с дополнительной кодовой таблицей ASCII возможна, но не гарантируется для систем, не прошедших адаптацию под тот или иной язык речи людей. В частности, проблемы работы с символами кириллицы (например, перевод строки при наборе малой буквы «с» в командной строке) уже обсуждались. Поэтому примеры в этом уроке даны для строк с символами основной кодовой таблицы.

В приложениях операционных систем Windows получили распространение двухбайтовые таблицы кодирования символов – Unicode. Они позволяют кодировать до $2^{16} = 65\,536$ символов.

К основным *строковым функциям* относятся следующие.

- `char (X)` преобразует массив `X` положительных целых чисел (числовых кодов от 0 до 65 535) в массив символов системы MATLAB (причем только первые 127 кодов – английский набор ASCII, со 128 до 255 – расширенный набор ASCII) и возвращает его, на платформе Windows при значении выше 65 535 выдает предупреждение об ошибке, но возвращает русскую букву я (я повторяется также как `char(255+256n)`, где n – целые неотрицательные числа). Примеры:

```
>> X=reshape(32:127,32,3);
>> S = char(X')
S =
!>#$$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~•

>> t1='computer';
>> t2='for';
>> t3='home';
>> t4='users';
>> S = char(t1,t2,t3,t4)
S =
computer
for
home
users
```

- `char (C)` преобразует каждый элемент строкового массива ячеек в ряды массива символов, если строки массива ячеек разного размера, к ним в конце добавляются пробелы (осуществляется набивка (padding) в терминах MATLAB) так, чтобы в каждом ряду массива символов было одинаковое число.
- `char (T1, T2, T3)`, где `T` – строки, возвращает массив символов, при этом копии строк `T1, T2, T3` преобразуются в ряды массива символов добавлением, при необходимости, пробелов в конце рядов массивов символов, как описано выше.
- `char (java.lang.string)` преобразует объект класса `java.lang.string` в массив символов MATLAB.
- `char (javaarray of java.lang.string)` – единственный случай, когда выходным аргументом функции является не массив символов, а строковый массив ячеек, в который преобразуется массив строк Java.
- `double (S)` преобразует символы строки `S` в числовые коды 0–65 535 и возвращает вектор с этими числовыми кодами.
- `ischar (S)` возвращает логическую единицу, если `S` является символьной переменной, и логический нуль в противном случае.
- `deblank (str)` возвращает строку, полученную из аргумента – строки `str` с удаленными из ее конца пробелами.

- `deblank(c)` применяет функцию `deblank` к каждому элементу строкового массива ячеек `c`.

Примеры:

```
>> S = 'computer'
S =
computer
>> X = double(S)
X =
      99      111      109      112      117      116      101      114
>> ischar(S)
ans =
           1
>> c{1,1}='My ';
>> c{1,2}='home   ';
>> c{1,3}='computer ';
>> c
c =
      'My   '      'home   '      'computer   '
>> c = deblank(c)
c =
      'My'   'home' 'computer'
```

10.1.2. Операции над строками

К операциям над строками обычно относят поиск вхождений одних строк в другие, замену регистров символов, объединение строк и т. д. Следующие функции осуществляют операции над строками:

- `findstr(str1, str2)` обеспечивает поиск начальных индексов более короткой строки внутри более длинной и возвращает вектор этих индексов. Индексы указывают положение первого символа более короткой строки в более длинной строке. Пример:

```
>> str1='Example of the function is the findstr function';
>> str2='the';
>> k = findstr(str1, str2)
k =
      12      28
```

- `lower('str')` возвращает строку символов `str`, в которой символы верхнего регистра переводятся в нижний регистр, а все остальные символы остаются без изменений. Пример:

```
>> str='Example Of The Function';
>> t = lower(str)
t =
example of the function
```

- `upper('str')` возвращает строку символов `str`, в которой все символы нижнего регистра переводятся в верхний регистр, а все остальные символы остаются без изменений. Пример:

```
>> str='danger!';
>> t = upper(str)
t =
DANGER!
```

- `strcat(s1, s2, s3, ...)` выполняет горизонтальное объединение соответствующих рядов массивов символов `s1`, `s2`, `s3` и т. д., причем пробелы в конце каждого ряда отбрасываются, и возвращает объединенную строку (ряд) результирующего массива символов, пробелы добавляются заново после анализа строк в полученном массиве. Все входные массивы должны иметь одинаковое число строк (в частном случае должны быть представлены в виде одной строки символов), но если один из входных аргументов – не массив символов, а строковый массив ячеек, то любой из других входных аргументов может быть скаляром или любым массивом той же размерности и того же размера. Если входной массив состоит только из символов, то выходной массив также будет являться массивом символов. Если любой из входных массивов является строковым массивом ячеек, то функция `strcat` возвращает строковый массив ячеек, сформированный из объединенных соответствующих элементов массивов `s1`, `s2`, `s3`, при этом любой из элементов может быть скаляром и т. д. Примеры:

```
>> s1{1,1}='Home';
>> s1{1,2}='book';
>> s1
s1 =
    'Home' 'book'
>> s2{1,1}='home';
>> s2{1,2}='reading';
>> s2
s2 =
    'home' 'reading'
>> t = strcat(s1,s2)
t =
    'Homehome'    'bookreading'
>> s1=['wri']
s1 =
wri
>> s2=['ter']
s2 =
ter
>> t = strcat(s1,s2)
t =
writer
```

- `strvcat(t1,t2,t3,...)` выполняет вертикальное объединение строк `t1`, `t2`, `t3`, ... в массив символов `S` аналогично `char(t1,t2,t3,...)`. Пример:

```
>> t1=['string'];
>> t2=['concatenation'];
```

```
>> S = strvcat(t1,t2)
S =
string
concatenation
```

- `strcmp('str1','str2')` возвращает логическую единицу, если две сравниваемые строки `str1` и `str2` идентичны, и логический нуль в противном случае.
- `TF = strcmp(S,T)` возвращает строковый массив ячеек `TF`, содержащий единицы для идентичных элементов массивов `S` и `T` и нули для всех остальных, причем если один из массивов – не массив символов, а строковый массив ячеек, то перед сравнением из сравниваемых копий рядов массива символов удаляются пробелы в конце строк. Массивы `S` и `T` должны иметь одинаковый размер, или один из них может быть скалярной ячейкой. Примеры:

```
>> str1='computer';
>> str2='computer';
>> k = strcmp(str1,str2)
k =
    1

>> S{1,1}='first';
>> S{1,2}='second';
>> S
S =
    'first'    'second'

>> T{1,1}='third';
>> TF = strcmp(S,T)
TF =
    0    0

>> T{1,1}='second';
>> TF = strcmp(S,T)
TF =
    0    1
```

- `strncmp('str1','str2',n)` возвращает логическую единицу, если две сравниваемые строки `str1` и `str2` содержат `n` первых идентичных символов, и логический нуль в противном случае. Аргументы `str1` и `str2` могут быть также строковыми массивами ячеек.
- `TF = strncmp(S,T,n)` возвращает строковый массив ячеек `TF`, содержащий единицы для идентичных (до `n` символов) элементов массивов `S` и `T` и нули для всех остальных.

Примеры:

```
>> str1='computer'
str1 =
computer
>> str1='computer for me'
str1 =
computer for me
```

```
>> k = strncmp(str1, str2, 3)
k =
      1
>> k = strncmp(str1, str2, 12)
k =
      0
```

- `strjust(S)` возвращает выровненный вправо массив символов (то есть перемещает пробелы в конце рядов массива символов, если они есть, в начало тех же рядов).
- `strmatch('str', STRS)` просматривает массив символов или строковый массив ячеек `STRS` по строкам, находит строки символов, начинающиеся со строки `str`, и возвращает соответствующие индексы строк.
- `strmatch('str', STRS, 'exact')` возвращает только индексы строк символов массива `STRS`, точно совпадающих со строкой символов `str`.

Пример:

```
>> STRS{1,1}='character';
>> STRS{1,2}='array';
>> STRS{2,1}='character array';
>> STRS{2,2}='string';
>> STRS
STRS =
      'character'      'array'
      'character array' 'string'
>> i = strmatch('charac', STRS)
i =
      1
      2
>> i = strmatch('character', STRS, 'exact')
i =
      1
```

- `strrep(str1, str2, str3)` заменяет все подстроки `str2`, найденные внутри строки символов `str1`, на строку `str3`.
- `strrep(str1, str2, str3)` возвращает строковый массив ячеек, полученный в результате выполнения функции `strrep` над соответствующими рядами входных массивов символов или ячеек, если один из аргументов `str1`, `str2` или `str3` – строковый массив ячеек. В этом случае любой из аргументов может быть также скалярной ячейкой. Пример:

```
>> str1='This is a good example for me.';
>> str2='good';
>> str3='best';
>> str = strrep(str1, str2, str3)
str =
This is a best example for me.
```

- `strtok('str', delimiter)` возвращает часть текстовой строки `str`, ограниченную с ее конца разделителем `delimiter`. Символы-разделители

в начале строки игнорируются. Вектор `delimiter` содержит возможные символы-разделители.

- `strtok('str')` использует символ-разделитель по умолчанию («белое пространство»). Реальными символами-разделителями при этом являются символ табуляции (ASCII-код 9), символ возврата каретки (ASCII-код 13) и пробел (ASCII-код 32).
- `[token, rem]=strtok(...)` возвращает остаток `rem` исходной строки.

Примеры:

```
>> str='This is a good example for me.';
>> token = strtok(str)
token =
This
>> token = strtok(str, 'f')
token =
This is a good example
>> [token, rem] = strtok(str)
token =
This
rem =
is a good example for me.
```

10.1.3. Преобразование символов и строк

Теперь рассмотрим функции преобразования символов и строк.

- `int2str(X)` округляет элементы массива `X` до целых чисел и возвращает массив символов, содержащих символьные представления округленных целых чисел. Аргумент `X` может быть скаляром, вектором или матрицей.

Пример:

```
>> X=magic(3)
X =
     8     1     6
     3     5     7
     4     9     2
X=X+0.05
X =
     8.0500  1.0500  6.0500
     3.0500  5.0500  7.0500
     4.0500  9.0500  2.0500
>> str=int2str(X)
str =
     8     1     6
     3     5     7
     4     9     2
```

- `mat2str(A)` преобразует матрицу `A` в единую строку; если элемент матрицы не скаляр, то он заменяется на `[]`, при этом учитываются 15 знаков после десятичной точки.

- `mat2str(A, n)` преобразует матрицу `A` в строку, используя точность до `n` цифр после десятичной точки. Функция `eval(str)` осуществляет обратное преобразование.

Примеры:

```
>> rand('state');
>> A=rand(4,3)
A =
    0.9501    0.8913    0.8214
    0.2311    0.7621    0.4447
    0.6068    0.4565    0.6154
    0.4860    0.0185    0.7919

>> str = mat2str(A,2)
str =
[0.95 0.89 0.82;0.23 0.76 0.44;0.61 0.46 0.62;0.49 0.019 0.79]
```

- `num2str(A)` выполняет преобразование массива `A` в строку символов `str` с точностью до четырех десятичных разрядов и экспоненциальным представлением, если требуется. Обычно используется при выводе графиков совместно с `title`, `xlabel`, `ylabel` или `text`.
- `num2str(A, precision)` выполняет преобразование массива `A` в строку символов `str` с максимальной точностью, определенной аргументом `precision`. Аргумент `precision` определяет число разрядов в выходной строке.
- `num2str(A, format)` выполняет преобразование массива чисел `A`, используя заданный формат `format`. По умолчанию принимается формат, который использует четыре разряда после десятичной точки для чисел с фиксированной или плавающей точкой.

Примеры:

```
>> str = num2str(pi,7)
str =
3.141593
>> rand('state');
>> A=rand(3,5)
A =
    0.9501    0.4860    0.4565    0.4447    0.9218
    0.2311    0.8913    0.0185    0.6154    0.7382
    0.6068    0.7621    0.8214    0.7919    0.1763

>> str = num2str(A,1)
str =
1      0.5      0.5      0.4      0.9
0.2    0.9      0.02    0.6      0.7
0.6    0.8      0.8      0.8      0.2
```

- `str2double('str')` выполняет преобразование численной строки `s`, которая представлена в ASCII-символах, в число с двойной точностью. При этом `+` и `-` могут быть только в начале строки. Пример:

```
>> x = str2double('5.45+2.67i')
```

5.4500 + 2.6700i

Обратите особое внимание на последнюю функцию, поскольку именно она в MATLAB обычно обеспечивает переход от символьного представления математических выражений к их вычисленным числовым значениям.

- `str2num(s)` выполняет преобразование числового массива символов – матрицы или строки `s`, который представлен в ASCII-символах, в матрицу (массив размерности 2).

Пример:

```
>> x = str2num('5.45+2.67')
8.1200
```

Обратите особое внимание, что при этом можно вводить знаки + и – в любом месте строки. Предыдущая функция выдала бы NaN. Но фирма MathWorks рекомендует использовать `str2num` с осторожностью и, по возможности, заменять ее на `str2double`.

10.1.4. Функции преобразования систем счисления

Некоторые строковые функции служат для *преобразования систем счисления*. Ниже представлен набор этих функций.

- `bin2dec('binarystr')` возвращает десятичное число, эквивалентное строке *двоичных* символов `binarystr`. Пример:

```
>> bin2dec('101')
ans = 5
```

- `dec2bin(d)` возвращает строку двоичных символов (0 и 1), эквивалентную десятичному числу `d`. Аргумент `d` должен быть неотрицательным целым числом, меньшим чем 2^{52} .
- `dec2bin(d, n)` возвращает строку двоичных символов, содержащую по меньшей мере `n` бит.

Пример:

```
>> str = dec2bin(12)
str = 1100
```

- `dec2base(d, n)` возвращает строку символов, представляющих десятичное число `d` как число в системе счисления с основанием `n`.

Пример:

```
>> str = dec2base(1234, 16)
str = 4D2
```

- `dec2hex(d)` возвращает шестнадцатеричную строку символов, эквивалентную числу `d`. Аргумент `d` должен быть неотрицательным целым числом, меньшим чем 2^{52} .
- `str = dec2hex(d, n)` возвращает шестнадцатеричную строку, содержащую по меньшей мере `n` цифр.

Пример:

```
>> str = dec2hex(1234)
str = 4D2
```

- `base2dec(S, B)` преобразует строку символов *S*, представляющих число в системе счисления по основанию *B*, в символьное представление десятичного числа. Пример:

```
>> d = base2dec('4D2', 16)
d = 1234;
```

- `hex2dec('hex_value')` возвращает число *d*, представленное строкой шестнадцатеричных символов *hex_value*. Если аргумент *hex_value* является массивом символов, то каждая строка этого массива интерпретируется как шестнадцатеричное представление числа. Пример:

```
>> d = hex2dec('4D2')
d = 1234
```

- `hex2num('hex_value')` возвращает десятичное число *f* с удвоенной точностью, эквивалентное шестнадцатеричному числу, находящемуся в строке символов *hex_value*. Пример:

```
>> f = hex2num('4831fb52a18')
f = 6.1189e+039
```

10.1.5. Вычисление строковых выражений

Строковые выражения обычно не вычисляются, так что, к примеру, вывод строки `'2+3'` просто повторяет строку:

```
>> '2+3'
ans =
2+3
```

Однако с помощью функции `eval('строковое выражение')` строка, представляющая математическое выражение, преобразуется в вычисляемую форму и может быть вычислена:

```
>> eval('2+3')
ans = 5
>> eval('2*sin(1)')
ans = 1.6829
```

Ниже использование `eval` возвращает 12 матриц, представляющих магические квадраты чисел от 1 до 12:

```
for n = 1:12
    eval(['M' num2str(n) ' = magic(n)'])
end
```

`eval(S1, S2)` в случае ошибки в вычислении выражения *s1* оценивает выражение *s2*.

`T=evalc(S)` выполняет то же, что и функция `eval(s)`, но то, что выводится в командное окно, записывается также и в массив *T*.

Еще одна функция – `feval` (@имя_функции, x_1, x_2, \dots) – имеет важное достоинство – она позволяет передавать в вычисляемую функцию список ее аргументов. При этом вычисляемая функция задается только своим именем. Это поясняют следующие примеры:

```
>> feval(@prod,[1 2 3])
ans = 6
>> feval(@sum,[1 2 3; 4 5 6],2)
ans =
     6
    15
```

Рекомендуется применять функцию `feval` при вычислении значений функций, записанных в виде строки. `m`-файлы, содержащие функцию `feval`, корректно компилируются компилятором системы MATLAB.

Для выполнения вычислений, представленных строкой `expression`, в заданной рабочей области `ws` служит функция `evalin(ws, expression)`. Переменная `ws` может иметь два значения: `'base'` – для основной рабочей области и `'caller'` – для рабочей области вызванной функции. В приведенном ниже примере в рабочей области записаны переменные `a` и `b` и вычисляется символьное значение `'a+b'`:

```
>> a=2;b=3;
>> evalin('base','a+b')
ans = 5
```

Функция может также записываться в виде

```
[a1,a2,a3,...] = evalin(ws,expression)
```

где `a1, a2, a3, ...` – переменные, возвращающие результаты вычислений.

А функция

```
evalin(ws,expression, catch_expr)
```

позволяет проверить правильность выражения `expression` в рабочей области и сформировать сообщение, заданное в строке `catch_expr`. Например (в продолжение последнего примера):

```
>> h='Error in expression';
>> evalin('base','a+b','h');
ans = 5
>> evalin('base','a+c','h');
h =
Error in expression
```

Здесь выражение `a+c` ошибочно (переменная `c` не определена), поэтому выдается переменная `h` с ее значением в виде строки.

10.2. Работа с файлами

При обработке данных возникает необходимость хранения как исходных данных, так и результатов вычислений. Для этого обычно используются файлы. *Файлы* – это довольно распространенные объекты системы MATLAB. Последнее, в частно-

сти, относится к файлам данных. О некоторых типах файлов уже говорилось в предшествующих уроках. Ниже рассматриваются свойства файлов, которые не зависят от их типа и относятся к любым файлам.

10.2.1. Открытие и закрытие файлов

Файл обычно является некоторой совокупностью данных, объединенных одним именем. Тип файла, как правило, определяется его расширением. Мы рассматриваем файл как некое целое, хотя физически на диске он может быть представлен несколькими областями – говорят, что в этом случае файл фрагментирован.

Перед использованием любого файла он должен быть *открыт*, а по окончании использования – *закрыт*. Много файлов может быть открыто и доступно для чтения одновременно. Рассмотрим команды открытия и закрытия файлов.

Команда `open имя`, где имя должно содержать массив символов или символьную переменную, открывает файлы в зависимости от анализа параметра имя и расширения в имени `имя`:

- переменная – открывает массив, названный по имени, в редакторе массивов (**Array Editor**);
- `.mat` – открывает файл, сохраняет переменные в структуре в рабочей области;
- `.fig` – открывает его в редакторе дескрипторной графики Property Editor;
- `.m` – открывает m-файл в редакторе-отладчике;
- `.mdl` – открывает модель в Simulink;
- `.p` – открывает, если он есть, m-файл с тем же именем;
- `.html` – открывает HTML-документ в браузере помощи.

Если файлы с расширением существуют в пути MATLAB, то открывается тот файл, который возвращается командой `which имя`, если нет – то файл из файловой системы. Если файл не имеет расширения имени, то он открывается той программой, формат файлов которой был бы обнаружен функцией `which('имя файла')`. По умолчанию для всех файлов с окончаниями, отличными от вышеперечисленных, вызывается `openother`. `Open` вызывает функции `openxxx`, где `xxx` – расширение файла. Исключение – переменные рабочей области, для которых вызывается `openvar`, и рисунки, для работы с которыми вызывается `openim`. Создавая m-файлы с именем `openxxx`, пользователи могут изменять обработку файлов и добавлять новые расширения в список. Закрывать файлы, открытые при помощи `open`, нужно из редакторов, вызываемых `openxxx`.

- `[FILENAME, PATHNAME] = uigetfile(FILTERSPEC, Title)`. Открывает диалог с именем `Title` и фильтром `FILTERSPEC` (например, массивом ячеек, содержащим расширения файлов) и возвращает файл, выбранный пользователем, и путь к нему. Возвращает `FILENAME=0`, если файл не существует или если пользователь нажал на `Cancel`. `[FILENAME, PATHNAME] = uigetfile(FILTERSPEC, Title, X, Y)` размещает окно диалога в точке `X, Y` (координаты в пикселях). Пример:

```
[filename, pathname] = uigetfile('*.*;*.fig;*.mat;*.mdl',  
'All MATLAB Files (*.m, *.fig, *.mat, *.mdl)'); ...
```

- `[FILENAME, PATHNAME] = uiputfile(FILTERSPEC, TITLE)` сохраняет файл в диалоге, управляемом пользователем. Параметры аналогичны таковым в функции `uigetfile`.

Команда `uiopen` открывает диалог и, если пользователь выбрал файл с известным расширением, вызывает его, используя `open`, или если имя файла имеет неизвестное расширение, то вызывается `uigetfile`. Входными аргументами `uiopen` могут быть `matlab`, `load`, `figure`, `simulink`, `editor`. Без входных аргументов или с входным аргументом `matlab` в окне диалога предлагается выбрать `*.m`, `*.fig`, `*.mat`, `*.mdl` (если Simulink установлен), `*.cdr` (если Stateflow установлен), `*.rtw`, `*.tmf`, `*.tlc`, `*.c`, `*.h`, `*.ads`, `*.adb` (если установлен Real-Time Workshop). С аргументом `load` – `*.mat`. С аргументом `figure` предлагаются `*.fig`; `simulink` – `*.mdl`, `editor` – `*.m`, `*.mdl`, `*.cdr`, `*.rtw`, `*.tmf`, `*.tlc`, `*.c`, `*.h`, `*.ads`, `*.adb`. Пример: `uiopen figure`.

Команда `uiload` открывает файл в диалоге, управляемом пользователем, с использованием команды `load`.

Функция `uiimport` запускает **Мастер импорта (Import Wizard)**, импортирующий из файла в текущей папке или буфера обмена Windows. Она соответствует выбору **Import Data** из меню **File** или выбору **Paste Special** из меню **Edit** MATLAB.

- `uiimport (FILENAME)` запускает **Мастер импорта**, открывая файл `FILENAME`. Мастер импорта показывает данные для предварительного просмотра. В окне предварительного просмотра появляются данные и их представление в виде переменных MATLAB. Собственно данные, текст и заголовки представляются разными переменными MATLAB. Для данных ASCII вы должны удостовериться, что Мастер импорта распознал разделители столбцов. Самостоятельно он может распознать только символ табуляции, пробел, запятую или точку с запятой. Нужно щелкнуть мышью на кнопке **Next** и в следующем окне либо подтвердить выбор разделителя, сделанный Мастером, либо выбрать **Other** и ввести любой разделитель.
- `uiimport ('-file')` вначале выводит диалог выбора файла.
- `uiimport ('-pastespecial')` вначале выводит для предварительного просмотра содержимое буфера обмена Windows.
- `S = uiimport (...)` хранит результирующие переменные как поля структуры `S`.

Команда `uisave` – управляемое пользователем сохранение с Windows-диалогом.

Функция `saveas` сохраняет рисунок или модель Simulink в желаемом формате на носителе информации или на устройстве, разрешенном `print`. Функция `saveas (H, 'FILENAME')` сохраняет данные в соответствии с командой дескрипторной графики `H` в файле `FILENAME`. Формат файла определяется расширением имени `FILENAME`.

Функция `saveas (H, 'FILENAME', 'FORMAT')` выполняет то же, но с параметром `FORMAT` (формат задается тем же способом, что и расширение имени файла, и может от него отличаться). `FORMAT` имеет приоритет перед расширением имени файла. Параметры:

- 'fig' – сохранить рисунок (график) в двоичном fig-файле;
- 'm' или 'mfig' – сохранить рисунок в двоичном fig-файле и создать m-файл для его загрузки;
- 'mmap' – сохранить рисунок в m-файле как последовательность команд создания рисунка. Может не поддерживать новейших графических функций.

Примеры:

```
saveas(gcf, 'output', 'fig')
saveas(gcf, 'output', 'bmp')
```

Команда или функция `delete` удаляет файл или объект графики. `delete` имя файла удаляет файл текущей папки. Может быть использована *. Предпочтительно использование с записью в форме функции `delete('имя файла')`, когда имя файла – строка. Команда `delete(h)` удаляет графический объект с дескриптором `h`. Если этот объект – окно, то оно предварительно закрывается.

Функция `close(h)` закрывает только графические окна. Для закрытия файлов необходимо использовать команду `fclose`.

Для записи файлов на диск служит команда `save`, используемая в довольно очевидных формах:

```
save          save filename          save filename var1 var2 ...
save ... option          save('filename', ...)
```

Соответственно, для считывания файлов с диска служит команда `load`:

```
load          load filename          load filename X Y Z
load filename -ascii load filename -mat S = load(...)
```

В этих командах имя файла указывается по правилам, принятым в операционных системах класса MS-DOS. Эти команды обычно дублируются кнопками панелей инструментов и браузером файлов.

10.2.2. Операции с двоичными файлами

Двоичными, или *бинарными*, называют файлы, данные которых представляют собой машинные коды. Основные операции с такими кодами перечислены ниже.

- `fopen(filename, permission)` открывает файл с именем `filename` и параметром, определенным в `permission`, и возвращает идентификатор `fid` со значением: 0 – чтение с клавиатуры (`permission` установлено в 'r'); 1 – вывод на дисплей (`permission` установлено в 'a'); 2 – вывод сообщения об ошибке (`permission` установлен в 'a'); -1 – неудача в открытии файла с выводом сообщения `message` о типе ошибки. Идентификатор `fid` часто используется в качестве аргумента другими функциями и программами ввода-вывода. Имя файла `filename` может содержать путь к файлу.

Если открываемый для чтения файл не найден в текущем каталоге, то функция `fopen` осуществляет поиск файла по пути, указанном в MATLAB.

Параметр `permission` может принимать одно из следующих основных значений (другие см. в справочной системе):

- 'r' – открытие файла для чтения (по умолчанию);
- 'r+' – открытие файла для чтения и записи;
- 'w' – удаление содержимого существующего файла или создание нового и открытие его для записи;
- 'a' – создание и открытие нового файла или открытие существующего для записи с добавлением в конец файла.

Добавление к этой строке 'b' (подразумевается по умолчанию) предписывает системе открыть файл в двоичном режиме.

Добавление же вместо b к этой строке 't', например 'rt', в операционных системах, которые имеют различие между текстовыми и двоичными файлами, предписывает системе открыть файл в текстовом режиме. Например, во всех версиях MATLAB для Windows/MS-DOS и VMS нельзя открыть текстовый файл без параметра 'rt'. При вводе файлов с использованием fopen в текстовом режиме удаляются все символы «возврат каретки» перед символом новой строки.

- `[fid,message] = fopen(filename,permission,format)` открывает файл, как описано выше, возвращая идентификатор файла и сообщение. Кроме того, значение параметра `format` позволяет точно определить числовой формат. Возможны 8 форматов, описание которых можно найти в справочной системе. В частности, строка `format` может иметь значения 'native' (формат компьютера, на котором установлена система), 'vax', 'cray' (компьютеры VAX и Cray) и т. д.

Определенные вызовы функций `fread` или `fwrite` могут отменить числовой формат, заданный при вызове функции `fopen`.

- `fids = fopen('all')` возвращает вектор-строку, содержащую идентификаторы всех открытых файлов, не включая стандартные потоки 0, 1 и 2. Число элементов вектора равно числу открытых пользователем файлов.
- `[filename,permission,format] = fopen(fid)` возвращает полное имя файла, строку `permission` и строку `format`. При использовании недопустимых значений `fid` возвращаются пустые строки для всех выходных аргументов.
- Команда `fclose` закрывает файл. Она имеет следующие варианты.
- `status = fclose(fid)` закрывает файл, если он открыт. Возвращает статус файла `status`, равный 0, если закрытие завершилось успешно, и -1 в противном случае. Аргумент `fid` – это идентификатор, связанный с открытым файлом (см. функцию `fopen` для более подробного описания).
- `status = fclose('all')` закрывает все открытые файлы. Возвращает 0 в случае успешного завершения и -1 в противном случае.

Пример открытия и закрытия файла:

```
>> fid=fopen('c:\ex','a+')
fid = 4
>> fclose(4)
ans = 0
```

- `[A, count] = fread(fid, size, precision)` считывает двоичные данные из заданного файла и помещает их в матрицу `A`. Выходной аргумент `count` содержит число удачно считанных элементов. Значение идентификатора `fid` – это целое число, возвращенное функцией `fopen`; `size` – аргумент, определяющий количество считываемых данных. Если аргумент `size` не определен, функция `fread` считывает данные до конца файла.

Используются следующие параметры `size`:

- `n` – чтение `n` элементов в вектор-столбец;
- `inf` – чтение элементов до конца файла и помещение их в вектор-столбец, содержащий такое же количество элементов, что и в файле;
- `[m, n]` – считывает столько элементов, сколько нужно для заполнения матрицы $m \times n$.

Заполнение происходит по столбцам. Если элементов в файле мало, то матрица дополняется нулями. Если считывание достигает конца файла, не заполнив матрицу необходимого размера, то матрица дополняется нулями. Если происходит ошибка, чтение останавливается на последнем считанном значении. Параметр `precision` – строка, определяющая числовую точность считывания значений, она контролирует число считанных бит для каждого значения и интерпретирует эти биты как целое число, число с плавающей запятой или как символ.

- `[A, count] = fread(fid, size, precision, skip)` включает произвольный аргумент `skip`, определяющий число байтов, которые необходимо пропустить после каждого считывания. Это может быть полезно при извлечении данных в несмежных областях из записей фиксированной длины. Если `precision` имеет битовый формат, такой как `'bitN'` или `'ubitN'`, значение `skip` определяется в битах. Обширный список возможных значений параметра `precision` можно найти в справочной системе MATLAB.
- `count=fwrite(fid, A, precision)` записывает элементы матрицы `A` в файл, представляя их с заданной точностью. Данные записываются в файл по столбцам, выходной аргумент `count` содержит число удачно записанных элементов. Значение идентификатора `fid` – это целое число, полученное при использовании функции `fopen`. Добавляет символы «возврат каретки» перед началом новой строки.
- `count=fwrite(fid, A, precision, skip)` делает то же, но включает произвольный аргумент `skip`, определяющий число байтов, которые надо пропустить перед каждой записью. Это полезно при вставке данных в несмежные области в записях фиксированной длины. Если `precision` имеет битовый формат, такой как `'bitN'` или `'ubitN'`, значение `skip` определяется в битах.

Примеры:

```
>> fid = fopen('c:\prim', 'a+')
fid = 3
>> A=magic(7)
A =+
```

```

30    39    48    1    10    19    28
38    47    7     9    18    27    29
46    6     8    17    26    35    37
5     14    16    25    34    36    45
13    15    24    33    42    44    4
21    23    32    41    43    3     12
22    31    40    49    2     11    20

>> count = fwrite(3,A)
count = 49
>> status = fclose(3)
status = 0
>> fid = fopen('c:\prim','r')
fid = 3
>> [B,count] = fread(3,[7,7])
B =
30    39    48    1    10    19    28
38    47    7     9    18    27    29
46    6     8    17    26    35    37
5     14    16    25    34    36    45
13    15    24    33    42    44    4
21    23    32    41    43    3     12
22    31    40    49    2     11    20

count = 49

```

10.2.3. Операции над форматированными файлами

Файлы, содержащие форматированные данные, называют *форматированными файлами*. Ниже представлены функции, которые служат для работы с такими файлами.

- `line = fgetl(fid)` возвращает строку из файла с идентификатором `fid` с удалением символа конца строки. Если функция `fgetl` обнаруживает конец файла, то она возвращает значение `-1` (см. функцию `fopen` с более подробным описанием `fid`).
- `line = fgets(fid)` возвращает строку из файла с идентификатором `fid`, не удаляя символ конца строки. Если функция `fgets` обнаруживает конец файла, то она возвращает значение `-1`.
- `line = fgets(fid, nchar)` возвращает не больше чем `nchar` первых символов строки. После признака конца строки или конца файла никакие дополнительные символы не считываются (см. примеры к функции `fscanf`).
- `count = fprintf(fid, format, A, ...)` форматирует данные, содержащиеся в действительной части матрицы `A`, под контролем строки `format` и записывает их в файл с идентификатором `fid`. Функция `fprintf` возвращает число записанных байтов. Значение идентификатора `fid` – целое число, возвращаемое функцией `fopen`.

Если опустить идентификатор `fid` в списке аргументов функции `fprintf`, то вывод будет осуществляться на экран, так же как при использовании стандартного вывода (`fid=1`).

- `fprintf(format, A, ...)` – запись осуществляется на стандартное устройство – экран (но не в файл). Строка `format` определяет систему счисления, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Она может содержать обычные буквы алфавита наряду со спецификаторами, знаками выравнивания и т. д.

Функция `fprintf` ведет себя как аналогичная функция `fprintf()` языка ANSI C с некоторыми исключениями и расширениями. В табл. 10.1 описаны специальные символы, встречающиеся в строке `format`.

Таблица 10.1. Специальные символы в строках формата

| Символ | Описание |
|-----------------------------|--------------------------|
| <code>\n</code> | Новая строка |
| <code>\t</code> | Горизонтальная табуляция |
| <code>\b</code> | Возврат на один символ |
| <code>\r</code> | Возврат каретки |
| <code>\f</code> | Новая страница |
| <code>\\</code> | Обратный слэш |
| <code>\' ' или \" \"</code> | Одиночная кавычка |
| <code>%%</code> | Процент |

Для вывода числовых или символьных данных в строке формата необходимо использовать *спецификаторы*, перечисленные в табл. 10.2.

Таблица 10.2. Спецификаторы формата вывода данных

| Спецификатор | Описание |
|-----------------|---|
| <code>%c</code> | Одиночный символ |
| <code>%d</code> | Десятичная система обозначений (со знаком) |
| <code>%e</code> | Экспоненциальное представление чисел с использованием символа «e» в нижнем регистре, например 3.1415e + 00 |
| <code>%E</code> | Экспоненциальное представление чисел с использованием символа «E» в верхнем регистре, например 3.1415E + 00 |
| <code>%f</code> | Система обозначений с фиксированной точкой |
| <code>%g</code> | Наиболее компактный вариант из <code>%e</code> и <code>%f</code> . Незначащие нули не выводятся |
| <code>%G</code> | То же самое, что и <code>%g</code> , но используется верхний регистр для символа «E» |
| <code>%o</code> | Восьмеричная система обозначений (без знака) |
| <code>%s</code> | Строка символов |
| <code>%u</code> | Десятичная система обозначений (без знака) |
| <code>%x</code> | Шестнадцатеричная система обозначений с использованием символов нижнего регистра («a»...«f») |
| <code>%X</code> | Шестнадцатеричная система обозначений с использованием верхнего регистра символов («A»...«F») |

Между знаком процента и буквой в спецификатор могут быть вставлены дополнительные символы. Их значение поясняет табл. 10.3.

Таблица 10.3. Параметры спецификаторов формата

| Символ | Описание | Пример |
|---------------------|---|--------|
| Знак «минус» (-) | Выравнивание преобразованных аргументов по левому краю | %-5.2d |
| Знак «плюс» (+) | Всегда печатать знак числа (+ или -) | %+5.2d |
| Нуль (0) | Заполнение нулями вместо пробелов | %05.2d |
| Цифры | Определяет минимальное число знаков, которые будут напечатаны | %6f |
| Цифры (после точки) | Число после точки определяет количество символов, печатаемых справа от десятичной точки | %6.2f |

- $A = fscanf(fid, format)$ читает все данные из файла с идентификатором fid , преобразует их согласно значению параметра $format$ и возвращает в виде матрицы A . Значение идентификатора fid – целое число, возвращаемое функцией $fopen$. Параметр $format$ представляет собой строку, определяющую формат данных, которые необходимо прочитать.
- $[A, count] = fscanf(fid, format, size)$ считывает количество данных, определенное параметром $size$, преобразует их в соответствии с параметром $format$ и возвращает вместе с количеством успешно считанных элементов $count$. Параметр $size$ – это произвольный аргумент, определяющий количество считываемых данных. Допустимы следующие значения:
 - n – чтение n элементов в вектор-столбец;
 - inf – чтение элементов до конца файла и помещение их в вектор-столбец, содержащий такое же количество элементов, что и в файле;
 - $[m, n]$ – считывает столько элементов, сколько требуется для заполнения матрицы размера $m \times n$. Заполнение происходит по столбцам. Величина n (но не m !) может принимать значение Inf .

Строка $format$ состоит из обычных символов и (или) спецификаторов. Спецификаторы указывают тип считываемых данных и включают символ %, опцию ширины поля и символы формата. Возможные символы формата перечислены в табл. 10.4.

Таблица 10.4. Символы формата, используемые функцией $fscanf$

| Символ | Описание |
|------------|--|
| %c | Последовательность символов; параметр ширины поля определяет количество считываемых символов |
| %d | Десятичное число |
| %e, %f, %g | Число с плавающей точкой |
| %i | Целое число со знаком |
| %o | Восьмеричное число со знаком |

Таблица 10.4. Символы формата, используемые функцией `fscanf` (продолжение)

| Символ | Описание |
|--------------------|--|
| <code>%s</code> | Последовательность непробельных символов |
| <code>%u</code> | Десятичное целое число со знаком |
| <code>%x</code> | Шестнадцатеричное целое число со знаком |
| <code>[...]</code> | Последовательность символов |

Между символом `%` и символом формата допустимо вставлять следующие символы:

- звездочка (`*`) означает, что соответствующее значение не нужно сохранять в выходной матрице;
- строка цифр задает максимальную ширину поля;
- буква обозначает размер полученного объекта: `h` для короткого целого числа (например, `%hd`), `l` для длинного целого числа (например, `%ld`) или для числа с двойной точностью с плавающей запятой (например, `%lg`).

Примеры:

```
>> x = 0:pi/10:pi;y=[x;sin(x)];
>> fid = fopen('c:\sin.txt','w');
>> fprintf(fid,'%5.3f %10.6f\n',y);fclose(fid);
0.000    0.000000
0.314    0.309017
0.628    0.587785
0.942    0.809017
1.257    0.951057
1.571    1.000000
1.885    0.951057
2.199    0.809017
2.513    0.587785
2.827    0.309017
3.142    0.000000
>> fid = fopen('c:\sin.txt','r');
>> q=fscanf(fid,'%g',[2,10]);
>> q'
ans =
      0          0
    0.3140    0.3090
    0.6280    0.5878
    0.9420    0.8090
    1.2570    0.9511
    1.5710    1.0000
    1.8850    0.9511
    2.1990    0.8090
    2.5130    0.5878
    2.8270    0.3090
>> fgetl(fid)
ans = 3.142    0.000000
>> fgets(fid)
```

```
ans = -1
>> fclose(fid)
ans = 0
```

10.2.4. Позиционирование файла

При считывании и записи файлов они условно представляются в виде линейно расположенных данных, наподобие записи на непрерывной магнитной ленте. Место, с которого идет считывание в данный момент (или позиция, начиная с которой идет запись), определяется специальным *указателем*. *Файлы последовательного доступа* просматриваются строго от начала до конца, а в *файлах произвольного доступа* указатель может быть размещен в любом месте, начиная с которого ведется запись или считывание данных файла.

Таким образом, указатель обеспечивает позиционирование файлов. Имеется ряд функций позиционирования:

- `eofstat = feof(fid)` проверяет, достигнут ли конец файла с идентификатором `fid`. Возвращает 1, если указатель установлен на конец файла, и 0 в противном случае;
- `message = ferror(fid)` возвращает сведения об ошибке в виде строки `message`. Аргумент `fid` – идентификатор открытого файла (см. функцию `fopen` с ее подробным описанием);
- `message = ferror(fid, 'clear')` очищает индикатор ошибки для заданного файла;
- `[message, errnum] = ferror(...)` возвращает номер ошибки `errnum` последней операции ввода-вывода для заданного файла.

Если последняя операция ввода-вывода, выполненная для определенного значением `fid` файла, была успешной, значение `message` – это пустая строка, а `errnum` принимает значение 0.

Значение `errnum`, отличное от нуля, говорит о том, что при последней операции ввода-вывода произошла ошибка. Параметр `message` содержит строку, содержащую информацию о характере возникшей ошибки.

Пример:

```
>> fid=fopen('c:\example1','a+')
fid = 3
>> t = fread(3,[4,5])
t =
Empty matrix: 4-by-0
>> ferror(3)
ans =
Is the file open for reading? . . .
```

- `frewind(fid)` устанавливает указатель позиции в начало файла с идентификатором `fid`;
- `status = fseek(fid,offset,origin)` – устанавливает указатель в файле с идентификатором `fid` в заданную позицию – на байт, указанный параметром `offset` относительно `origin`.

Аргументы:

- `fid` – идентификатор файла, возвращенный функцией `fopen`;
- `offset` – значение, которое интерпретируется следующим образом:
- `offset>0` – изменяет позицию указателя на `offset` байт в направлении к концу файла;
- `offset=0` – не меняет позицию указателя;
- `offset<0` – изменяет позицию указателя на `offset` байт в направлении к началу файла;
- `origin` – аргумент, принимающий следующие значения:
 - `'bof'` или `-1` – начало файла;
 - `'cof'` или `0` – текущая позиция указателя в файле;
 - `'eof'` или `1` – конец файла;
- `status` – выходной аргумент. Принимает значение `0`, если операция `fseek` прошла успешно, и `-1` в противном случае. Если произошла ошибка, используйте функцию `error` для получения более подробной информации.
- `position=ftell(fid)` – возвращает позицию указателя для файла с идентификатором `fid`, полученным с помощью функции `fopen`. Выходной аргумент `position` – неотрицательное целое число, определяющее позицию указателя в байтах относительно начала файла. Если запрос был неудачным, `position` принимает значение `-1`. Используйте функцию `error` для отображения характера ошибки. Примеры:

```
>> fid=fopen('c:\example','a+')
fid = 3
>> count = fwrite(3,magic(6))
count = 36
>> ftell(3)
ans = 36
>> frewind(3);ftell(3)
ans = 0
>> fseek(3,12,0);ftell(3)
ans = 12
>> feof(3)
ans = 0
>> fclose(3)
ans = 0
```

- `s=sprintf(format,A,...)` форматирует данные в матрице `A` в формате, заданном параметром `format`, и создает из них строковую переменную `s`.
- `[s,errormsg] = sprintf(format,A,...)` аналогична ранее описанной функции, но дополнительно возвращает строку ошибки `errormsg`, если ошибка имела место, или пустую строку в противном случае. Строка `format` определяет систему счисления, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Она может содержать обычные символы наряду со спецификаторами, знаками выравнивания и т. д. Функ-

ция `fprintf` ведет себя, как и аналогичная функция `fprintf()` языка ANSI C с некоторыми исключениями и расширениями. Примеры:

```
>> sprintf('%0.5g', (1+sqrt(7))/4)
ans = 0.91144
>> sprintf('%s', 'привет')
ans =
привет
```

Функция `sscanf` аналогична функции `fscanf`, за исключением того, что она считывает данные из символьной переменной системы MATLAB, а не из файла.

- `A = sscanf(s, format)` считывает данные из символьной переменной `s`, преобразует их согласно значению `format` и создает на основе этих данных матрицу `A`. Параметр `format` определяет формат данных, которые нужно считать.
- `A = sscanf(s, format, size)` считывает количество данных, определенное параметром `size`, и преобразует их согласно строке `format`. Параметр `size` представляет собой аргумент, определяющий количество данных для чтения. Допустимы следующие значения:
 - `n` – чтение `n` элементов в вектор-столбец;
 - `inf` – чтение элементов до конца символьной переменной и помещение их в вектор-столбец, содержащий такое же количество элементов, как и в строковой переменной;
 - `[m, n]` – считывает столько элементов, сколько требуется для заполнения матрицы **размера** $m \times n$. Заполнение происходит по столбцам. Величина `n` (но не `m`!) может принимать значение `Inf`.
- `[A, count, errmsg, nextindex] = sscanf(...)` считывает данные из символьной переменной `s`, преобразует их согласно значению `format` и возвращает в матрицу `A`. Параметр `count` – выходной аргумент, который возвращает число успешно считанных элементов; `errmsg` – выходной аргумент, который возвращает строку ошибки, если ошибка произошла, и пустую строку в противном случае; `nextindex` – выходной аргумент, который содержит число, на единицу большее, чем количество символов в `s`.

Строка `format` состоит из обычных символов и спецификаторов. Спецификаторы указывают тип данных и включают в себя символ `%`, опцию ширины поля и символы формата. Пояснения можно найти в описании функции `fscanf`.

Пример:

```
>> s = '4.83 3.16 22 45';
>> [A,n,err,next] = sscanf(s,'%f')
A =
    4.8300
    3.1600
   22.0000
   45.0000
n = 4
err = ''
next = 16
```

10.2.5. Специализированные файлы

Приведенные ниже функции относятся к некоторым *специализированным файлам*.

- `M = dlmread(filename, delimiter)` считывает данные из файла `filename` с ASCII-разделителем, используя разделитель `delimiter`, в массив `M`. Используйте `'\t'`, чтобы определить в качестве разделителя символ табуляции.
- `M = dlmread(filename, delimiter, r, c)` считывает данные из файла `filename` с ASCII-разделителем, используя разделитель `delimiter`, в массив `M`, начиная со смещения `r` (по строкам) и `c` (по столбцам). Параметры `r` и `c` отсчитываются, начиная с нуля, так что `r=0, c=0` соответствует первому значению в файле.
- `M = dlmread(filename, delimiter, r, c, range)` импортирует индексированный или именованный диапазон данных с разделителями в формате ASCII. Для использования диапазона ячеек нужно определить параметр `range` в следующем виде:

```
range = [ВерхняяСтрока, ЛевыйСтолбец, НижняяСтрока, ПравыйСтолбец]
```

Аргументы функции `dlmread` следующие: `delimiter` – символ, отделяющий отдельные матричные элементы в электронной таблице формата ASCII, символ запятой (,) – разделитель по умолчанию, `r, c` – ячейка электронной таблицы, из которой берутся матричные элементы, соответствующие элементам в верхнем левом углу таблицы, `range` – вектор, определяющий диапазон ячеек электронной таблицы.

Команда `dlmwrite` преобразует матрицу MATLAB в файл с ASCII-разделителями, читаемый программами электронных таблиц:

- `dlmwrite(filename, A, delimiter)` записывает матрицу `A` в верхнюю левую ячейку электронной таблицы `filename`, используя разделитель `delimiter` для отделения элементов матрицы. Используйте `'\t'` для создания файла с элементами, разделенными табуляцией. Все элементы со значением 0 опускаются. Например, массив `[1 0 2]` появится в файле в виде `'1, , 2'` (если разделителем является запятая);
- `dlmwrite(filename, A, delimiter, r, c)` записывает матрицу `A` в файл `filename`, начиная с ячейки, определенной `r` и `c`, используя разделитель `delimiter`.
- `M = wk1read(filename)` считывает электронную таблицу Lotus123 (**WK1**) в матрицу `M`.
- `M = wk1read(filename, r, c)` считывает данные, начиная с ячейки, определенной значениями (`r, c`). Параметры `r` и `c` отсчитываются от нуля, так что `r=0, c=0` определяют первую ячейку в файле.
- `M = wk1read(filename, r, c, range)` считывает диапазон значений, определенный параметром `range`, где `range` может быть представлен в одной из следующих форм:
- вектор с четырьмя элементами, определяющий диапазон ячеек в формате `[верхняя_строка, левый_столбец, нижняя_строка, правый_столбец]`;

- диапазон ячеек, определенный строкой, например 'A1...C5';
- имя диапазона, определенное в виде строки, например 'Sales'.

`wklwrite(filename, M)` записывает значения матрицы `M` в файл `filename` электронной таблицы Lotus123 WK1.

`wklwrite(filename, M, r, c)` записывает данные, начиная с ячейки, определенной значениями `(r, c)`. Параметры `r` и `c` отсчитываются от нуля, так что `r=0`, `c=0` определяют первую ячейку в электронной таблице.

Необходимо отметить, что большинство рассмотренных выше функций редко применяются пользователями. Но они довольно широко используются в системных целях и представляют большой интерес для специалистов.

10.3. Работа с файлами изображений

10.3.1. Информация о графическом файле – *imfinfo*

Функция `info = imfinfo(filename, fmt)` возвращает структуру `info`, в полях которой содержится информация об изображении в графическом файле с именем `filename` и форматом `fmt`. Заметим, что файл с именем `filename` должен находиться в текущей директории или директории с явно указанным (по обычным правилам) путем, в противном случае будет произведен поиск файла с именем `filename` и расширением `fmt`. Отсутствие файла приведет к сообщениям об ошибках при попытке использования функций для работы с файлами.

В табл. 10.5 показаны возможные значения для аргумента `fmt`.

Таблица 10.5. Поддерживаемые графические форматы и их обозначения

| Формат | Тип файла |
|------------------|---|
| 'bmp' | Windows Bitmap (BMP) |
| 'hdf' | Hierarchical Data Format (HDF) |
| 'jpg' или 'jpeg' | Joint Photographic Experts Group (JPEG) |
| 'pcx' | Windows Paintbrush (PCX) |
| 'tif' или 'tiff' | Tagged Image File Format (TIFF) |
| 'xwd' | X Windows Dump (XWD) |

Если `filename` – TIFF- или HDF-файл, содержащий более одного изображения, то `info` представляет собой массив структур с отдельным элементом (то есть с индивидуальной структурой) для каждого изображения в файле. Например, `info(3)` будет в таком случае содержать информацию о третьем изображении в файле. Множество полей в `info` зависит от конкретного файла и его формата. Однако первые девять полей всегда одинаковы. В табл. 10.6 перечислены эти поля и описаны их значения.

Таблица 10.6. Поля информационной структуры и их значения

| Поле | Значение |
|---------------|---|
| Filename | Строка, содержащая имя файла; если файл находится не в текущей директории, строка содержит полный путь к файлу |
| FileModDate | Строка, содержащая дату последнего изменения файла |
| FileSize | Целое число, указывающее размер файла в байтах |
| Format | Строка, содержащая формат файла, заданный параметром fmt; для JPEG- и TIFF-файлов возвращается значение, состоящее из трех символов |
| FormatVersion | Строка или число, описывающее версию формата |
| Width | Целое число, указывающее ширину изображения в пикселях |
| Height | Целое число, указывающее высоту изображения в пикселях |
| BitDepth | Целое число, указывающее число битов на пиксель |
| ColorType | Строка, описывающая тип изображения: 'truecolor' для RGB-изображения, 'grayscale' для полутонового изображения или 'indexed' для изображения с индексированными цветами |

Функция `info = imfinfo(filename)` возвращает информацию о формате файла по его содержанию. В приведенном ниже примере показан вывод информации о файле `saturn.png` с изображением планеты Сатурн:

```
>> imfinfo('saturn.png')
ans =

        Filename: 'C:\Program
Files\MATLAB704\toolbox\images\imdemos\saturn.png'
        FileModDate: '13-Oct-2002 09:47:58'
        FileSize: 1166148
        Format: 'png'
        FormatVersion: []
        Width: 1200
        Height: 1500
        BitDepth: 24
        ColorType: 'truecolor'
        FormatSignature: [137 80 78 71 13 10 26 10]
        Colormap: []
        Histogram: []
        InterlaceType: 'none'
        Transparency: 'none'
        SimpleTransparencyData: []
        BackgroundColor: []
        RenderingIntent: []
        Chromaticities: []
        Gamma: []
        XResolution: []
        YResolution: []
        ResolutionUnit: []
```

```

XOffset: []
YOffset: []
OffsetUnit: []
SignificantBits: []
ImageModTime: '24 Jun 2002 19:02:45 +0000'
Title: []
Author: []
Description: 'Voyager 2 image, 1981-08-24,
             NASA catalog #PIA01364'
Copyright: []
CreationTime: []
Software: []
Disclaimer: []
Warning: []
Source: []
Comment: []
OtherText: []

```

10.3.2. Чтение изображения из файла – *imread*

Функция `A = imread(filename, fmt)` читает из файла с именем `filename` полутоновое или полноцветное изображение и создает `A`. Если исходное изображение полутоновое, то `A` – двумерный массив, если исходное изображение полноцветное, то `A` – трехмерный массив размера `m×n×3`.

Другие формы этой функции:

- `[X, map] = imread(filename, fmt)` читает из файла с именем `filename` палитровое изображение в массив `A` с цветовой картой `map`;
- `[...] = imread(filename)` пытается определить информацию о формате файла по его содержанию. Параметры `filename` и `fmt` были подробно рассмотрены в описании функции `imfinfo`;
- `[...] = imread(..., idx)` читает одно изображение из TIFF-файла. `idx` – целое число – номер изображения по порядку.

Описываемая функция имеет ряд особенностей для PNG-файлов, содержащих *прозрачные пиксели* (хотя и не всегда). Прозрачные пиксели, если они существуют, идентифицируются одним или двумя компонентами: часть данных прозрачности и альфа-канал. Часть данных прозрачности определяет прозрачные пиксели напрямую, например если часть данных прозрачности 8-битового изображения равна 0,5020, то все пиксели изображения с цветом 0,5020 будут выведены на экран как прозрачные. Заметим, что PNG-файл может содержать только один компонент – альфа-канал.

Альфа-канал представляет собой массив с таким же числом пикселей, как и исходное изображение, который определяет признак прозрачности каждого пикселя (прозрачный или непрозрачный). И наконец, последний компонент PNG-файла – это данные цвета фона, определяющие значение цвета, который «просвечивает»

из-под прозрачных пикселей. Ниже описывается поведение IPT по умолчанию при чтении PNG-изображений, содержащих или часть данных прозрачности, или альфа-канал.

- `[...] = imread(..., 'BackgroundColor', bg)` считывает изображение из PNG-файла, и пиксели прозрачности скомбинированы против определенного цвета. Форма параметра `bg` зависит от формата входного файла. Если входное изображение палитровое, то параметр `bg` должен быть целым числом порядка `[1,P]`, где `P` – длина массива цветовой карты. Если входное изображение полутоновое, то параметр `bg` должен быть целым числом порядка `[0,1]`. Если входное изображение полноцветное, то параметр `bg` должен быть трехэлементным вектором со значениями порядка `[0,1]`.
- `[A, map, alpha] = imread(...)` возвращает `A` – шаблон для указания способа, который используется для определения информации о прозрачности.
- `[A, map, alpha] = imread(filename)` или `[A, map, alpha] = imread(filename, fmt)` считывает изображение из PNG-файла, если не применяется комбинирование, и альфа-канал сохраняется отдельно от изображения.
- `[...] = imread(..., ref)` считывает одно изображение из HDF-файла. Параметр `ref` – целое число, определяющее справочное число, идентифицирующее изображение.
- `[...] = imread(..., idx)` считывает одно изображение из CUR- и ICO-файлов. Параметр `idx` – это целое число, определяющее порядок изображения в файле.

Пример применения этой функции представлен ниже (рис. 10.1):

```
>> I=imread('saturn.png');
>> imshow(I);
Warning: Image is too big to fit on screen; displaying at 25%
> In imuitools\private\initSize at 86
In imshow at 196
```

Малые файлы допустимых форматов считываются функцией `imread` без особых проблем и вводят в рабочее пространство свои массивы. Однако большие файлы могут вызвать проблемы при выводе их изображения функцией `imshow`, что и имеет место в представленном выше примере. Таблица 10.7 содержит форматы изображений, доступных для чтения функцией `imread`.

Таблица 10.7. Форматы файлов и их краткое описание

| Формат | Варианты |
|-------------|--|
| BMP | 1-битовые, 4-битовые, 8-битовые и 24-битовые несжатые изображения; 4-битовые и 8-битовые изображения со сжатием RLE |
| HDF | 8-разрядные растровые изображения, содержащие или не содержащие цветовую палитру; 24-разрядные растровые изображения |
| JPEG | Любые JPEG-изображения; JPEG-изображения с некоторыми обычно используемыми расширениями |

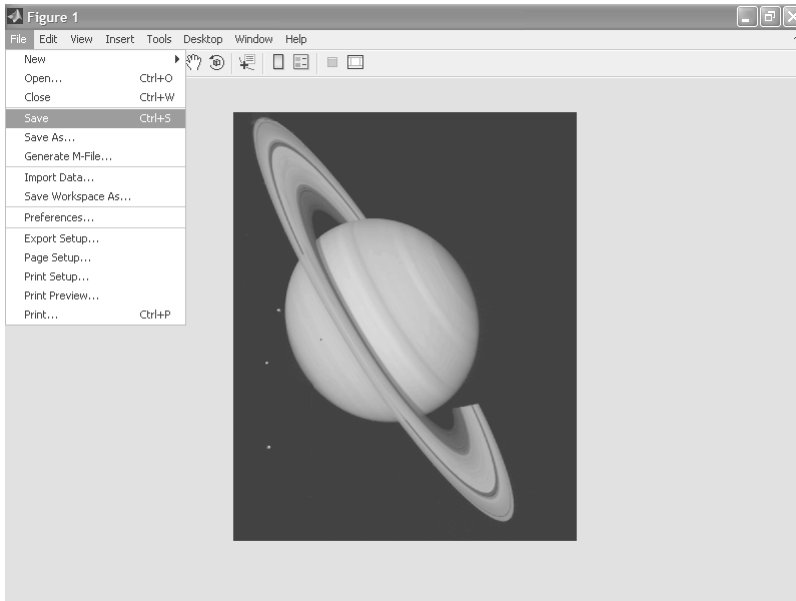


Рис. 10.1. Пример считывания и просмотра файла с большим изображением планеты Сатурн

Таблица 10.7. Форматы файлов и их краткое описание (продолжение)

| Формат | Варианты |
|--------|--|
| PCX | 1-битовые, 8-битовые и 24-битовые изображения |
| TIFF | Любые TIFF-изображения, включая 1-битовые, 8-битовые и 24-битовые несжатые изображения; 1-битовые, 8-битовые и 24-битовые изображения с <code>packbit</code> -сжатием; 1-битовые изображения со сжатием <code>CCITT</code> |
| XWD | 1-битовые и 8-битовые <code>Zpixmap</code> s; <code>XYBitmap</code> s; 1-битовые <code>XYPixmap</code> s |

10.3.3. Запись изображения в файл – `imwrite`

Для записи массива с изображением в файл служит функция `imwrite`. Она имеет следующие формы.

- `imwrite(A, filename, fmt)` записывает изображение в файл с именем `filename` в формате `fmt` из массива `A`. `A` может быть матрицей размера $M \times N$ для полутонового изображения, и массивом размера $M \times N \times 3$ для полноцветного изображения. Если `A` относится к классу `uint8` или `uint16`, то функция `imwrite` записывает фактические значения из массива в файл. Если `A` относится к классу `double`, то функция `imwrite` перемасштабирует значения в исходном массиве перед записью по формуле `uint8(round(255*A))`.

При этом числа с плавающей запятой в диапазоне [0,1] преобразуются в 8-битовые целые числа в диапазоне [0,255].

- `imwrite(X, map, filename, fmt)` записывает палитровое изображение в файл с именем `filename` в формате `fmt` из массива `X` и соответствующей цветовой карты `map`. Если `X` относится к классу `uint8` или `uint16`, то функция `imwrite` записывает фактические значения из массива в файл. Если `X` относится к классу `double`, то функция `imwrite` смещает значения в исходном массиве перед записью по формуле `uint8(X-1)`. Массив `map` должен быть цветовой картой MATLAB класса `double`, функция `imwrite` перемасштабирует исходные значения массива `map` по формуле `uint8(round(255*map))`. Заметим, что большинство графических файлов не поддерживают цветных карт с количеством ячеек больше, чем 256.
- `imwrite(..., filename)` аналогична описанным выше функциям, но формат файла определяется по расширению `filename`.

В табл. 10.8 приведены параметры, используемые при записи графических файлов функцией `imwrite`.

Таблица 10.8. Параметры, используемые при записи графических файлов

| Параметр | Значение | Значение по умолчанию |
|----------------------------------|--|---|
| Параметры для HDF-файлов | | |
| 'Compression' | Одно из следующих значений: 'none', 'rle', 'jpeg' | 'rle' |
| 'Quality' | Число между 0 и 100; параметр поддерживается для 'Compression'='jpeg'; чем больше число, тем выше качество файла (меньше искажений файла при сжатии) и тем больше его размер | 75 |
| 'WriteMode' | Одно из следующих значений: 'overwrite', 'append' | 'overwrite' |
| Параметры для JPEG-файлов | | |
| 'Quality' | Число между 0 и 100; чем больше число, тем выше качество файла (меньше искажений при сжатии файла) и тем больше его размер | 75 |
| Параметры для TIFF-файлов | | |
| 'Compression' | Одно из следующих значений: 'none', 'packbits', 'ccitt'; значение 'ccitt' допустимо только для двоичных (двухцветных) изображений | 'ccitt' для двоичных (двухцветных) изображений; 'packbits' для всех остальных |
| 'Description' | Любая строка; значение поля <code>ImageDescription</code> возвращается командой <code>imfinfo</code> | Пустая строка |
| 'Resolution' | Скалярное значение для разрешения в направлениях <code>x</code> и <code>y</code> | 72 |

Еще одна таблица – табл. 10.9 – описывает параметры для PNG-файлов.

Таблица 10.9. Параметры для PNG-файлов

| Параметр | Возможные значения Value | Значение Value по умолчанию |
|-----------------|---|---|
| 'Author' | Строка | Пустая строка Empty |
| 'Description' | Строка | Пустая строка Empty |
| 'Copyright' | Строка | Пустая строка Empty |
| 'CreationTime' | Строка | Пустая строка Empty |
| 'Software' | Строка | Пустая строка Empty |
| 'Disclaimer' | Строка | Пустая строка Empty |
| 'Warning' | Строка | Пустая строка Empty |
| 'Source' | Строка | Пустая строка Empty |
| 'Comment' | Строка | Пустая строка Empty |
| 'InterlaceType' | 'none' или 'adam7' | 'none' |
| 'BitDepth' | Скалярное значение глубины цвета. Для полутонового изображения может принимать значения 1, 2, 4, 8, или 16. Для полутонового изображения с альфа-каналом может принимать значения 8 или 16. Для палитрового изображения – 1, 2, 4, или 8. Для полноцветного изображения с или без альфа-канала – 8 или 16 | 8 bits/pixel для изображения double или uint8; 16 bits/pixel для изображения типа uint16; 1 bit/pixel для изображения logical |
| 'Transparency' | Этот параметр используется для указания прозрачности только без альфа-канала. Для палитрового изображения представляет собой Q-элементный вектор со значениями в интервале [0, 1], где Q не больше размера цветовой карты, и каждое значение указывает прозрачность, связанную с ячейкой цветовой карты. Для полутонового изображения – скаляр в интервале [0, 1]. Значение параметра указывает оттенок серого, который должен быть прозрачным. Для полноцветного изображения – трехэлементный вектор со значениями в интервале [0, 1]. Каждое значение определяет RGB-цвет, который должен быть прозрачным | Пустая ячейка или пустой вектор |

Таблица 10.9. Параметры для PNG-файлов (продолжение)

| Параметр | Возможные значения Value | Значение Value по умолчанию |
|-------------------|--|---------------------------------|
| 'Background' | Значение определяет цвет фона, используемый для композиции прозрачного пиксела. Для палитрового изображения представляет собой целое число в интервале [1,P], где P – длина цветовой карты. Для полутонового изображения – скаляр в интервале [0,1]. Для полноцветного изображения – трехэлементный вектор со значениями в интервале [0,1] | Пустая ячейка или пустой вектор |
| 'Gamma' | Неотрицательное скалярное число, указывающее файл gamma | Пустая ячейка |
| 'Chromaticities' | Восьмиэлементный вектор [wx wy gx gy bx by], определяющий опорную белую точку и основную хроматичность | Пустой вектор |
| 'Xresolution' | Скаляр, определяющий разрешение по горизонтали | Пустая ячейка |
| 'Yresolution' | Скаляр, определяющий разрешение по вертикали | Пустая ячейка |
| 'ResolutionUnit' | 'unknown' или 'meter' | Пустая строка |
| 'Alpha' | Матрица, определяющая прозрачность каждого пиксела отдельно. Количество строк и столбцов должно быть таким же, как и в массиве данных формата uint8, uint16, или double, значения находятся в диапазоне [0,1] | Пустая матрица |
| 'SignificantBits' | Скалярное число или вектор, определяющий размер исходного массива данных в битах. Значения должны находиться в пределах [1, BitDepth]. Для палитровых изображений – это трехэлементный вектор. Для полутонового изображения – скалярное число. Для полутонового изображения с альфа-каналом – двухэлементный вектор. Для полноцветного изображения – трехэлементный вектор. Для полноцветного изображения с альфа-каналом – четырехэлементный вектор | Пустая ячейка или пустой вектор |

В приведенном ниже примере считывается файл kids.tif, затем его массив I записывается в файл другого формата kids.jpg со сжатием в 20 раз, после чего проверяется тип нового файла:

```
I=imread('kids.tif'); imwrite(I,'kids.jpg','Quality',20)
iminfo('kids.jpg')
ans =

    Filename: 'kids.jpg'
    FileModDate: '31-Jul-2001 19:02:52'
```

```
FileSize: 3251
Format: 'jpg'
FormatVersion: ''
Width: 318
Height: 400
BitDepth: 8
ColorType: 'grayscale'
FormatSignature: ''
```

Здесь уместно сказать, что новый файл помещается в директорию WORK. Интересно отметить, что исходный цветной файл превратился в полутоновый типа grayscale.

В целом MATLAB содержит минимальный набор средств для работы с файлами изображений. Используя матричные операции системы MATLAB, можно организовать обработку файлов изображений. Профессионально такая обработка организована в ряде пакетов расширения MATLAB: Image Processing Toolbox, Video and Image Processing Blockset, Wavelet Toolbox и др.

10.4. Работа со звуковыми данными

Этот небольшой раздел посвящен экзотической возможности математической системы MATLAB – работе со *звуковыми данными*. Стоит напомнить, что для этого компьютер должен быть оснащен звуковой картой и звуковыми колонками. Средства поддержки звука в MATLAB имеют рудиментарный характер, но все же они есть и позволяют разнообразить выполнение некоторых примеров.

10.4.1. Функции для работы со звуками

Начиная с версии MATLAB 5.0, в системе несколько расширены средства для работы со звуком. До этого система имела единственную звуковую команду.

- `sound(Y, FS)` воспроизводит сигнал из вектора Y с частотой дискретизации FS с помощью колонок, подключенных к звуковой карте компьютера. Компоненты Y могут принимать значения в следующих пределах: $-1.0 \leq y \leq 1.0$. Для воспроизведения стереозвука на допускающих это компьютерных платформах Y должен быть матрицей размера $N \times 2$.
- `sound(Y)` функционирует аналогично, принимая частоту дискретизации по умолчанию равной 8192 Гц.
- `sound(Y, FS, BITS)` функционирует аналогично с заданием разрядности звуковой карты $BITS=8$ или $BITS=16$.

10.4.2. Функции звука в MATLAB 6.1/6.5

В версиях MATLAB 6.1/6.5 появились дополнительные команды воспроизведения звука:

- `soundsc(Y, ...)` масштабирует и воспроизводит сигнал из массива Y . По синтаксису команда аналогична `sound(Y, ...)`;

- `soundsc(Y, ..., SLIM)` аналогична предшествующей команде, но позволяет задать параметр `SLIM = [SLOW SHIGH]`, определяющий тот диапазон значений `Y`, который будет соответствовать полному динамическому диапазону звука. По умолчанию `SLIM = [MIN(Y) MAX(Y)]`;
- `beep on` или `off`, соответственно, разрешают или запрещают гудок;
- `s=beep` возвращает состояние `on|off`;
- `beep` при `s=on` издает гудок.

Кроме того, введены команды для считывания и записи файлов звукового формата **.WAV**, стандартного для операционных систем класса Windows:

- `wavwrite(Y, WAVEFILE)` записывает файл типа WAVE под именем WAVEFILE. Данные по каждому каналу в случае стерео записываются в разных столбцах массива. Величины должны быть в диапазоне $[-1;1]$;
- `wavwrite(Y, FS, WAVEFILE)` делает то же с заданием частоты дискретизации `FS` (в герцах);
- `wavwrite(Y, FS, NBITS, WAVEFILE)` делает то же с заданием числа бит на отсчет `NBITS`, причем `NBITS <= 16`;
- `Y=wavread(FILE)` считывает файл типа WAVE с именем `FILE` и возвращает данные в массиве `Y`;
- `[Y, FS, BITS]=wavread(FILE)` считывает файл типа WAVE с именем `FILE` и возвращает массив данных `Y`, частоту дискретизации `FS` (в герцах) и разрядность `BITS` кодирования звука (в битах);
- `[...]=wavread(FILE, N)` возвращает только первые `N` отсчетов из каждого канала файла;
- `[...]=wavread(FILE, [N1 N2])` возвращает только отсчеты с номерами от `N1` до `N2` из каждого канала;
- `SIZ=wavread(FILE, 'size')` возвращает объем аудиоданных в виде вектора `SIZ=[samples channels]` (`samples` – число отсчетов, `channels` – число каналов);
- `awrite` записывает файл в соответствии со звуковым форматом фирм Sun и Next; `auread` воспроизводит файлы в MATLAB 6 на Sun и в MATLAB 5 на Next.

10.4.3. Демонстрация возможностей работы со звуком

Для комплексной демонстрации возможностей работы со звуком служит файл-команда `xpsound`. Рисунок 10.2 показывает результат ее выполнения.

Эта команда выводит диалоговое окно, которое позволяет выбрать несколько видов звукового сигнала, создать для них массив данных звука и воспроизвести звук (если компьютер оснащен звуковой картой, совместимой с Sound Blaster). Кроме того, имеется возможность графически отобразить временную зависимость звукового сигнала, его частотный спектр и спектрограмму. На рис. 10.2 приведен пример временной зависимости звукового сигнала.

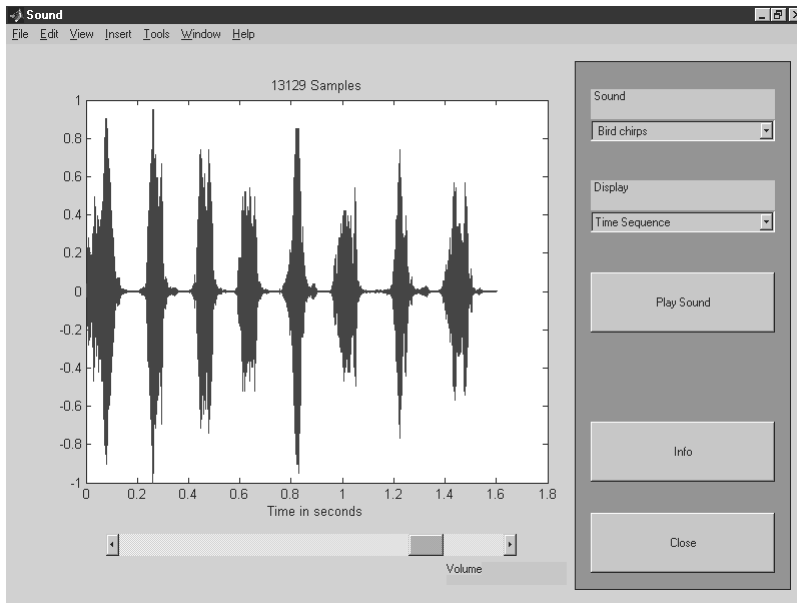


Рис. 10.2. Окно демонстрации воспроизведения звука с показом графика временной зависимости звукового сигнала

На рис. 10.3 представлен частотный спектр выбранного звукового сигнала. Он отражает относительный уровень звука в зависимости от частоты.

Еще один весьма наглядный способ представления массива данных звуковых сигналов – это показ их *спектрограммы*. Звуковой сигнал при этом делится на множество фрагментов, а спектрограмма дает представление о распределении частот спектра в разные моменты времени. Представление о том, насколько любопытной бывает спектрограмма сложного звукового сигнала, можно получить на примере рис. 10.4. Подобные спектрограммы могут быть использованы при разработке методов распознавания звуков.

Демонстрационные примеры можно просмотреть с помощью команды `type xpsound`. Вы получите доступ к более подробной информации по работе со звуком в системе MATLAB.

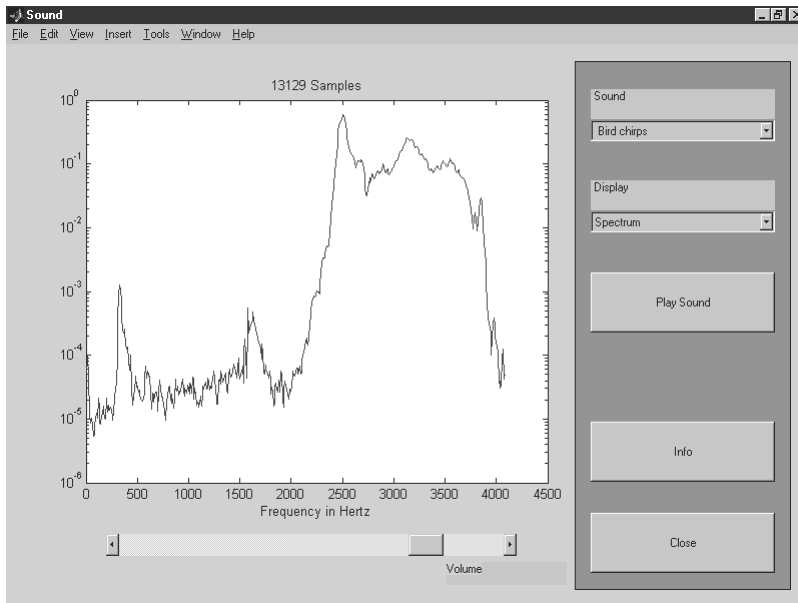


Рис. 10.3. Окно демонстрации возможности воспроизведения звука с выводом графика частотного спектра звукового сигнала

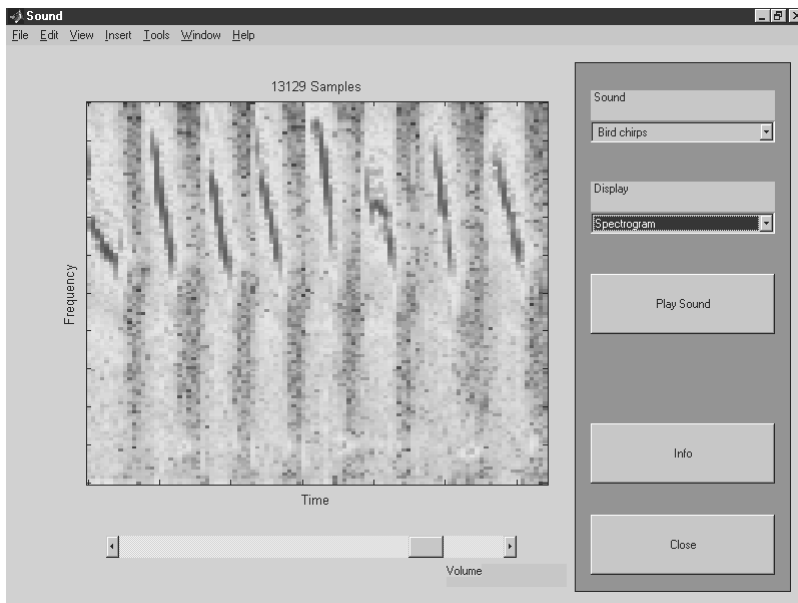


Рис. 10.4. Окно демонстрации возможности звуковоспроизведения с показом спектрограммы звукового сигнала

Типовые средства программирования

| | |
|--|-----|
| 11.1. Основные понятия программирования | 532 |
| 11.2. М-файлы сценариев и функций | 538 |
| 11.3. Обработка ошибок и комментарии | 545 |
| 11.4. Функции с переменным числом аргументов | 547 |
| 11.5. Особенности работы с m-файлами | 550 |
| 11.6. Управляющие структуры | 552 |
| 11.7. Основы объектно- ориентированного программирования | 561 |
| 11.8. Handle- и inline-функции .. | 565 |
| 11.9. Отладка программ | 567 |
| 11.10. Профилирование программ в MATLAB 7 | 575 |
| 11.11. Общение MATLAB с операционной системой | 579 |
| 11.12. Поддержка Java | 582 |
| 11.13. Компиляция MATLAB-программ | 587 |

Система MATLAB имеет один из лучших встроенных языков программирования высокого уровня четвертого поколения. Это проблемно-ориентированный на решение прежде всего математических задач язык. Практически все описанные выше операторы и функции системы MATLAB являются средствами ее языка программирования. По числу математических и графических операторов и функций язык программирования MATLAB значительно превосходит универсальные языки программирования, даже такие как BASIC, FORTRAN, PL1, C, JAVA и др. Но MATLAB содержит и все необходимые *типовые средства* программирования, такие как условные выражения, циклы, операторы ввода/вывода и др. Им и посвящен этот урок.

11.1. Основные понятия программирования

11.1.1. Назначение языка программирования MATLAB

Практически невозможно предусмотреть в одной, даже самой большой и мощной математической системе возможность решения всех задач, которые могут интересовать пользователя. *Программирование* в системе MATLAB является эффективным средством ее расширения и адаптации к решению специфических проблем. Оно реализуется с помощью *языка программирования* системы.

Большинство объектов этого языка, в частности все команды, операторы и функции, одновременно являются объектами *входного языка* общения с системой в командном режиме работы. Так что фактически мы приступили к описанию языка программирования системы MATLAB с первых строк данной книги, описав уже многие средства (прежде всего операторы и функции) входного языка MATLAB.

Так в чем же отличие входного языка от языка программирования? В основном – в способе фиксации создаваемых ими кодов. Сессии в командном режиме работы не сохраняются в памяти компьютера (ведение дневника не в счет). Хранятся только определения созданных в ходе их выполнения переменных и функций. А вот программы на языке программирования MATLAB сохраняются в виде текстовых m-файлов. При этом могут сохраняться как целые программы в виде файлов-сценариев, так и отдельные полноценные *программные модули* – функции. Кроме того, важно, что программа может менять структуру алгоритмов вычислений в зависимости от входных данных и данных, создаваемых в ходе вычислений.

С позиций программиста язык программирования системы является типичным *проблемно-ориентированным* языком программирования высокого уровня интерпретирующего типа. Точнее говоря, это даже язык *сверхвысокого* уровня, содержащий сложные операторы и функции, реализация которых на обычных языках (например, Бейсике, Паскале или Си) потребовала бы много усилий и времени. К таким функциям относятся матричные функции, функции быстрого пре-

образования Фурье (БПФ) и др., а к операторам – операторы построения разнообразных графиков, генерации матриц определенного вида и т. д.

11.1.2. Основные средства программирования

Итак, программами в системе MATLAB являются m-файлы текстового формата, содержащие запись программ в виде программных кодов. Язык программирования системы MATLAB имеет следующие средства:

- данные различного типа;
- константы и переменные;
- операторы, включая операторы математических выражений;
- встроенные команды и функции;
- функции пользователя;
- управляющие структуры;
- системные операторы и функции;
- средства расширения языка.

Коды программ в системе MATLAB пишутся на языке высокого уровня, достаточно понятном для пользователей умеренной квалификации в области программирования. Язык программирования MATLAB является типичным *интерпретатором*. Это означает, что каждая инструкция программы распознается и тут же исполняется, что облегчает обеспечение диалогового режима общения с системой. Этап компиляции всех инструкций и линковки (связями с библиотечными модулями), то есть создания полной программы, отсутствует. Высокая скорость выполнения программ обеспечена наличием заведомо откомпилированного ядра, хранящего в себе критичные к скорости выполнения инструкции.

Интерпретация означает, что MATLAB не создает исполняемых конечных программ в виде машинных кодов. Программы существуют лишь в виде m-файлов. Для выполнения программ необходима среда MATLAB, занимающая на диске много места. Однако для программ на языке MATLAB созданы компиляторы, транслирующие программы MATLAB в коды языков программирования FORTRAN, C/C++, JAVA и HTML. Это решает задачу создания исполняемых программ, первоначально разрабатываемых в среде MATLAB.

В новых версиях MATLAB можно использовать программные модули, написанные на указанных выше языках программирования в их собственной среде. Компиляторы внешних языков программирования являются вполне самостоятельными программными средствами и в данной книге не рассматриваются.

11.1.3. Основные типы данных

Обработка данных – важнейшая задача программирования. Для ее эффективного решения надо ознакомиться с основными типами данных. Структура типов данных системы MATLAB представлена на рис. 11.1. Она соответствует типовой

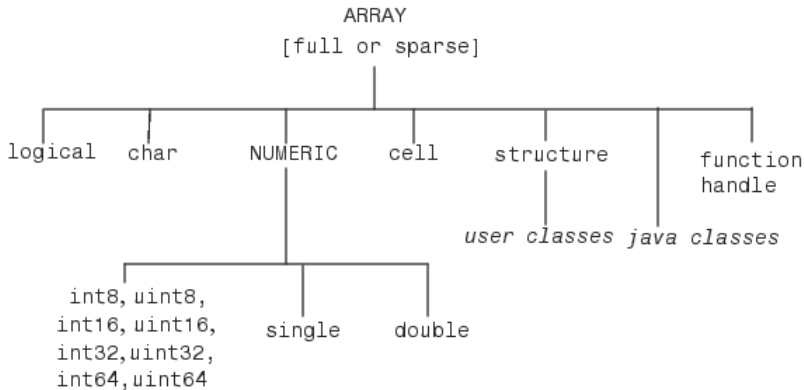


Рис. 11.1. Структура типов данных системы MATLAB

структуре данных *объекто-ориентированного программирования* и отражает характер подчиненности объектов друг другу и наследования их свойств.

Типы данных `array` и `numeric` являются *виртуальными* («кажущимися»), поскольку к ним нельзя отнести какие-либо переменные. Они служат для определения и комплектования некоторых типов данных. Таким образом, в MATLAB определены следующие основные типы данных, представляющих собой многомерные массивы:

- `single` – числовые массивы с числами одинарной точности;
- `double` – числовые массивы с числами удвоенной точности;
- `char` – строчные массивы с элементами-символами;
- `logical` – данные логического типа;
- `sparse` – наследует свойства `double`, разреженные матрицы с элементами-числами удвоенной точности;
- `cell` – массивы ячеек; ячейки, в свою очередь, тоже могут быть массивами;
- `struct` – массивы структур с полями, которые также могут содержать массивы;
- `Java classes` – объекты Java-класса;
- `function_handle` – дескрипторы функций;
- `int64`, `uint64` – массивы 64-разрядных чисел со знаком и без знаков;
- `int32`, `uint32` – массивы 32-разрядных чисел со знаком и без знаков;
- `int16`, `uint16` – массивы 16-разрядных целых чисел со знаком и без знаков;
- `int8`, `uint8` – массивы 8-разрядных целых чисел со знаками и без знаков.

Кроме того, предусмотрен еще один тип данных – `UserObject`, который относится к типам данных (объектом), определяемым пользователем. Типы данных `double`, `char` и `sparse` были рассмотрены ранее, так что в этом уроке будут детально рассмотрены оставшиеся типы. Что касается чисел класса `uint8`, то они представляют значения от 0 до 255 и занимают в памяти 1/8 часть от размера одного числа с двойной точностью. В основном этот тип данных применяется в служебных целях.

Каждому типу данных можно соотнести некоторые характерные для него операции, называемые *методами*. Дочерние типы данных, расположенные на приведенной диаграмме ниже родительских типов, наследуют от последних их методы, что является признаком *наследования объектов*. Поскольку в иерархии типов данных сверху находятся данные типа `array`, это значит, что все виды данных в MATLAB являются массивами.

Отличительной особенностью языка программирования MATLAB является отсутствие необходимости в объявлении типов переменных. Их тип автоматически устанавливается в соответствии с типом данных, используемых при присваивании переменным тех или иных значений. Это приближает язык MATLAB к естественному языку описания математических вычислений.

Другая особенность – отсутствие указания на то, откуда берется та или иная функция или оператор, заданные пользователем. Все они сохраняются на жестком диске и вызываются точно так же, как встроенные в ядро функции и операторы.

11.1.4. Виды программирования

На рынке программного обеспечения система MATLAB позиционируется как язык высокого уровня для научно-технических расчетов. Язык программирования системы MATLAB вобрал в себя все средства, необходимые для реализации различных видов программирования:

- процедурного;
- операторного;
- функционального;
- логического;
- структурного (модульного);
- объектно-ориентированного;
- визуально-ориентированного.

В основе *процедурного, операторного и функционального* типов программирования лежат процедуры, операторы и функции, используемые как основные объекты языка. Эти типы объектов присутствуют в MATLAB. *Логическое* программирование реализуется в MATLAB с помощью логических операторов и функций. Они позволяют реализовать основные идеи логического программирования, хотя на выдающуюся роль в этом классе языков программирования MATLAB не претендует.

Зато MATLAB представляет собой яркий пример плодотворности *структурного* программирования. Подавляющее большинство функций и команд языка представляют собой вполне законченные модули, обмен данными между которыми происходит через их входные параметры, хотя возможен обмен данными и через глобальные переменные. Программные модули оформлены в виде текстовых `m`-файлов, которые хранятся на диске и подключаются к программам по мере необходимости.

Важно отметить, что, в отличие от многих языков программирования, применение тех или иных модулей не требует предварительного объявления, а для создания и отладки самостоятельных модулей MATLAB имеет все необходимые средства. Подавляющее большинство команд и функций системы MATLAB поставляется в виде таких модулей. Их можно читать (с помощью как редактора MATLAB, так и любого текстового редактора), разбирать по смыслу, модифицировать и исполнять. Таким образом реализуется идея *открытого программирования*.

Объектно-ориентированное программирование также широко представлено в системе MATLAB. Оно особенно актуально при программировании задач графики. Что касается *визуально-ориентированного* программирования, то в MATLAB оно представлено в основном проектированием графического интерфейса пользователя GUI и моделированием заданных блоками устройств и систем в пакете расширения Simulink.

11.1.5. Двойственность операторов, команд и функций

Для языка системы MATLAB различие между командами (выполняемыми при вводе с клавиатуры) и программными операторами (выполняемыми из программы) является условным. И команды, и программные операторы могут выполняться как из программы, так и в режиме прямых вычислений. Под командами далее в основном понимаются средства, управляющие периферийным оборудованием, под операторами – средства, выполняющие операции с операндами (данными).

Функция преобразует одни данные в другие. Для многих функций характерен возврат значений в ответ на обращение к ним с указанием списка входных параметров – аргументов. Например, говорят, что функция $\sin(x)$ в ответ на обращение к ней возвращает значение синуса аргумента x . Поэтому функцию можно использовать в арифметических выражениях, например $2*\sin(x+1)$. Для команд, не возвращающих значения, такое применение обычно абсурдно. В данной книге все функции, возвращающие единственное значение (или один массив), записываются малыми (строчными) буквами в виде:

f_name (Список_параметров)

Для функций, возвращающих ряд значений или массивов (например, X, Y, Z,...), запись имеет следующий вид:

[X, Y, Z, ...]=f_name (Список_параметров)

Важное значение имеет *двойственность* операторов и функций. Многие операторы имеют свои аналоги в виде функций. Так, например, оператор «+» имеет аналог в виде функции sum. Команды, записанные в виде

Command argument

нередко имеют форму записи и в виде функции:

Command ('argument')

Примеры:

```
>> type('sin')
sin is a built-in function.
>> type sin
sin is a built-in function.
```

Указанная двойственность лежит в основе выбора между процедурным и функциональным типами программирования, каждый из которых имеет своих поклонников и противников и может (в той или иной мере) подходить для решения различных классов задач.

11.1.6. Некоторые ограничения

Поскольку язык программирования системы MATLAB ориентирован на структурное программирование, в нем нет номеров строк (присущих до недавнего времени Бейсику) и программных операторов безусловного перехода GO TO. Имеются лишь управляющие структуры следующих типов: условных выражений if...else...elseif...end, циклы for...end и while...end. Их форма похожа на ту, которая используется в языке Pascal (то есть область действия управляющих структур начинается их заголовком, но без слова begin, а заканчивается словом end). С позиций теории структурного программирования этих средств достаточно для решения любых задач. В MATLAB имеются также операторы-переключатели типа case, облегчающие создание программных конструкций с множественным ветвлением.

Однако в MATLAB исключены те средства, возможности которых можно реализовать уже имеющимися средствами. Зато резко увеличен набор средств для программирования решения математических задач, прежде всего сводящихся к матричным вычислениям и реализации современных численных методов.

Для ускорения вычислений на программные конструкции MATLAB также накладываются некоторые ограничения. Например, в прежних версиях MATLAB целочисленные операции с числами малой разрядности выполнялись явно быстрее, чем операции над числами высокой разрядности с плавающей запятой. Но в новых версиях это не так, в частности из-за поддержки операций с плавающей запятой математическим сопроцессором, входящим в центральные процессоры, а также вследствие применения ускорителя времени исполнения JIT. Он обеспечивает генерацию кода в момент исполнения (Just-In-Time Code Generation) и анализ типов данных в момент исполнения (Run-time Type Analysis).

Для обеспечения эффективной работы JIT-ускорителя следует учитывать ряд ограничений:

- не все операторы поддерживаются ускорителем JIT;
- если в программной строке находится хотя бы один не поддерживаемый JIT оператор, то вся строка будет интерпретироваться без ускорения;
- ускоритель не поддерживает численных данных типа Single;
- в циклах for...end управляющие переменные должны быть только целочисленными, могут применяться лишь те типы данных, которые поддерживаются ускорителем JIT, и допустимы вызовы только встроенных M-функций;

- в строке нежелательно применение более одного оператора;
- нежелательно переопределение массивов с изменением их типа;
- операции с комплексными числами не поддерживаются JIT-ускорителем.

Эти ограничения достаточно серьезны для начинающих пользователей. Однако опытные пользователи-программисты могут легко их обойти и создавать программы, обеспечивающие ускорение отдельных видов вычислений в десятки и даже в сотни раз.

11.1.7. Исполнение программных объектов

Как уже отмечалось, m-файлы сценариев (script-файлы) и функций должны иметь уникальные имена. Длина имен не ограничивается, но только первый 31 символ учитывается при идентификации имени. При исполнении программного объекта его имя сравнивается со списком имен, хранящихся в рабочей области и в директориях m-файлов. Если имя оказывается неуникальным, соответствующий программный объект не исполняется, и выводится сообщение об ошибке. Если же имя уникально, то программный объект исполняется в интерпретирующем режиме.

11.2. M-файлы сценариев и функций

11.2.1. Структура и свойства файлов-сценариев

В уроке 2 было показано, что для создания m-файлов может использоваться как встроенный редактор, так и любой текстовый редактор, поддерживающий форматы ASCII и Unicode. Подготовленный и записанный на диск m-файл становится частью системы, и его можно вызывать как из командной строки, так и из другого m-файла. Есть два типа m-файлов: файлы-сценарии и файлы-функции. Важно, что в процессе своего создания они проходят синтаксический контроль с помощью встроенного в систему MATLAB редактора/отладчика m-файлов.

Файл-сценарий, именуемый также Script-файлом, является просто записью серии команд без входных и выходных параметров. Он имеет такую структуру:

```
%Основной комментарий
%Dополнительный комментарий
Тело файла с любыми выражениями
```

Важны следующие свойства файлов-сценариев:

- они не имеют входных и выходных аргументов;
- работают с данными из рабочей области;
- в процессе выполнения не компилируются;
- представляют собой зафиксированную в виде файла последовательность операций, полностью аналогичную той, что используется в сессии.

Основным комментарием (в фирменной документации он назван H1) является первая строка текстовых комментариев, а дополнительным – последующие

строки. Основной комментарий выводится при выполнении команд `lookfor` и `help` имя_каталога. Полный комментарий выводится при выполнении команды `help` имя_файла. Рассмотрим следующий файл-сценарий:

```
%Plot with color red
%Строит график синусоиды линией красного цвета
%с выведенной масштабной сеткой в интервале [xmin,xmax]
x=xmin:0.1:xmax;
plot(x,sin(x),'r')
grid on
```

Первые три строки здесь – это комментарий, остальные – тело файла. Обратите внимание на возможность (нерекомендуемую) задания комментария на русском языке. Знак `%` в комментариях должен начинаться с первой позиции строки. В противном случае команда `help name` не будет воспринимать комментарий (иногда это может понадобиться) и возвратит сообщение вида

```
No help comments found in name.m.
```

Будем считать, что файл записан под именем `pcr`. Работа с ним представлена на рис. 11.2. Показаны подготовка к запуску файла (задание конкретных значений для `xmin` и `xmax`), запуск файла, получение рисунка (окно внизу) и вызов комментария командой `help pcr`. Командой `type pcr` можно вывести полный листинг файла.

Обратите внимание на то, что такой файл нельзя запустить без предварительной подготовки, сводящейся к заданию значений переменным `xmin` и `xmax`, ис-

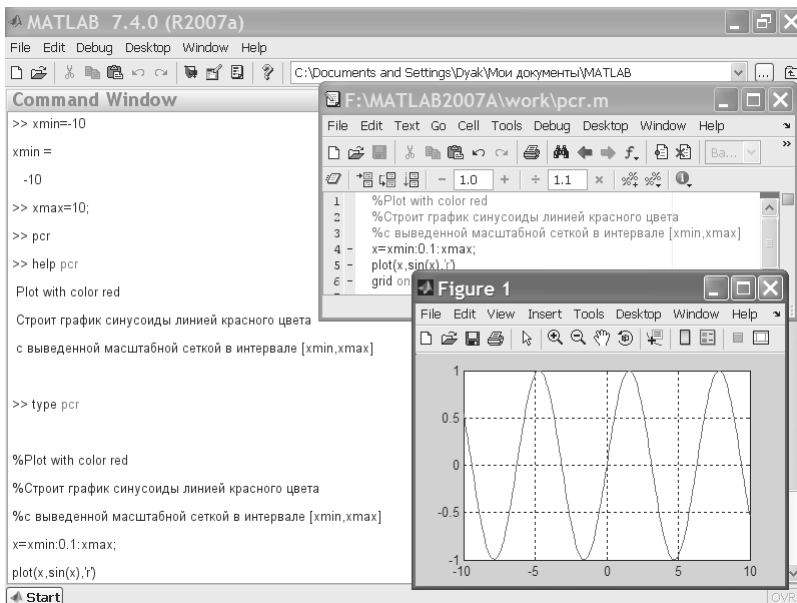


Рис. 11.2. Пример работы с файлом `pcr`

пользованным в теле файла. Это следствие первого свойства файлов-сценариев – они работают с данными из рабочей области. Переменные, используемые в файлах-сценариях, являются глобальными, то есть они действуют одинаково в командах сессии и внутри программного блока, которым является файл-сценарий. Поэтому заданные в сессии значения переменных используются и в теле файла. Имена файлов-сценариев нельзя использовать в качестве параметров функций, поскольку файлы-сценарии не возвращают значений.

11.2.2. Структура М-файла-функции

М-файл-функция является типичным полноценным объектом языка программирования системы MATLAB. Одновременно он является полноценным модулем с точки зрения структурного программирования, поскольку содержит входные и выходные параметры и использует аппарат локальных переменных. Структура такого модуля с одним выходным параметром выглядит следующим образом:

```
function var=f_name(Список_параметров)
%Основной комментарий
%Dополнительный комментарий
Тело файла с любыми выражениями
var=выражение
```

М-файл-функция имеет следующие свойства:

- он начинается с объявления `function`, после которого указываются имя переменной `var` – выходного параметра, имя самой функции и список ее входных параметров;
- функция возвращает свое значение и может использоваться в виде `name(Список_параметров)` в математических выражениях;
- все переменные, имеющиеся в теле файла-функции, являются локальными, то есть действуют только в пределах тела функции;
- файл-функция является самостоятельным программным модулем, который общается с другими модулями через свои входные и выходные параметры;
- правила вывода комментариев те же, что у файлов-сценариев;
- файл-функция служит средством расширения системы MATLAB;
- при обнаружении файла-функции он компилируется и затем исполняется, а созданные машинные коды хранятся в рабочей области системы MATLAB.

Последняя конструкция `var=выражение` вводится, если требуется, чтобы функция возвращала результат вычислений.

Приведенная форма файла-функции характерна для функции с одним выходным параметром. Если выходных параметров больше, то они указываются в квадратных скобках после слова `function`. При этом структура модуля имеет следующий вид:

```
function [var1,var2,...]=f_name(Список_параметров)
%Основной комментарий
```

```
%Дополнительный комментарий  
Тело файла с любыми выражениями  
var1=выражение  
var2=выражение  
...
```

Такая функция во многом напоминает процедуру. Ее нельзя слепо использовать непосредственно в математических выражениях, поскольку она возвращает не единственный результат, а множество результатов – по числу выходных параметров. Если функция используется как имеющая единственный выходной параметр, но имеет ряд выходных параметров, то для возврата значения будет использоваться первый из них. Это зачастую ведет к ошибкам в математических вычислениях. Поэтому, как отмечалось, данная функция используется как отдельный элемент программ вида:

```
[var1, var2, ...]=f_name(Список_параметров)
```

После его применения переменные выхода `var1`, `var2`, ... становятся определенными, и их можно использовать в последующих математических выражениях и иных сегментах программы. Если функция используется в виде `f_name(Список_параметров)`, то возвращается значение только первого выходного параметра – переменной `var1`.

11.2.3. Статус переменных в функциях

Переменные, указанные в списке параметров функции, являются *локальными* и служат для переноса значений, которые подставляются на их место при вызовах функций. Эта особенность переменных-параметров хорошо видна при разборе примера, показанного на рис. 11.3. Здесь (признаемся, что неточно) задана некоторая функция двух переменных $\text{fun}(x,y)$.

В этом примере в окне редактора создана функция `fun` двух переменных x и y , вычисляющая $z = x^2 + y^2$. Поскольку переменные x и y указаны как параметры функции `fun(x, y)`, то они являются локальными. В примере вне тела функции им заданы нулевые значения. Очевидно, что при вычислении значения `fun(2, 3)` в теле функции задается $x=2$ и $y=3$. Поэтому результат – $z=13$. Однако после выхода из тела функции переменные x и y принимают свои исходные значения, равные нулю. Так что эти переменные меняют свои значения на значения параметров функции только локально – в пределах тела функции.

А каков статус переменной z в нашем примере? Она, как и любая переменная, определенная в теле функции, также будет локальной. Изначально ее значение не определено. В теле функции переменная принимает значение $z=13$. А после возврата из функции, как нетрудно увидеть по рис. 11.3, переменная z , несмотря на ее применение в теле функции, остается неопределенной. На это указывает сообщение, отображаемое после попытки вывода значения переменной z .

Возврат из функции производится после обработки всего тела функции, то есть при достижении конца файла функции. При использовании в теле функции

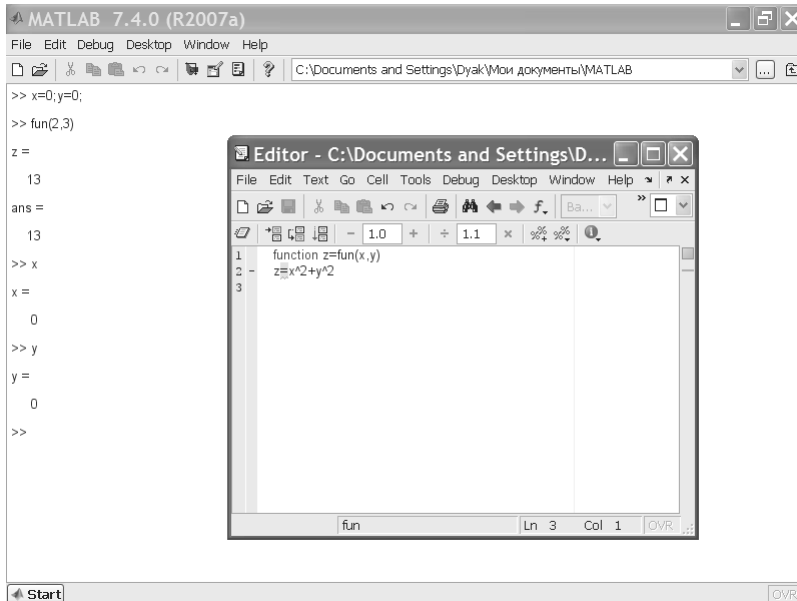


Рис. 11.3. Пример, поясняющий действие локальных и глобальных переменных при задании файла-функции

условных операторов, циклов или переключателей иногда возникает необходимость осуществить возврат функции раньше, чем будет достигнут конец файла. Для этого служит команда `return`. В любом случае результатом, возвращаемым функцией, являются значения выходных параметров (в нашем случае выходным параметром является переменная z), присвоенные им на момент возврата.

У нашей функции имеется один недостаток – вывод на индикацию значения $z=13$ из тела функции, хотя после этого z остается равным 0. Чтобы убрать побочный эффект вывода значения z , достаточно установить знак `;` после математического выражения, определяющего z . Таким образом, окончательно наша функция должна записываться следующим образом:

```
function z=fun(x, y)
z=x^2+y^2;
```

Этот пример наглядно показывает, что пропуск любого слова или даже просто оператора (вроде знака `;`) может привести к не сразу понятным побочным эффектам и даже неверной работе функции. Программирование требует особой точности и педантичности, именно поэтому далеко не все могут быть хорошими программистами.

11.2.4. Команда глобализации переменных *global*

Итак, из сказанного ясно, что переменные в файлах-сценариях являются *глобальными*, а в файлах-функциях – *локальными*. Нередко применение глобальных переменных в программных модулях может приводить к побочным эффектам. Применение локальных переменных устраняет эту возможность и отвечает требованиям структурного программирования.

Однако передача данных из модуля в модуль в этом случае происходит только через входные и выходные параметры, что требует тщательного планирования такой передачи. В жизни мы далеко не всегда едим черную икру для избранных (локальные переменные) и часто хотим отведать черного хлебушка для всех (глобальные переменные). Так и при создании файлов-функций порой желательно применение глобальных переменных. Ответственность за это должен брать на себя программист, создающий программные модули.

Команда

```
global var1 var2...
```

позволяет объявить переменные модуля-функции глобальными. Таким образом, внутри функции могут использоваться и такие переменные, если это нужно по условиям решения вашей задачи. Чтобы несколько программных модулей могли совместно использовать глобальную переменную, ее идентификатор должен быть объявлен в составе *global во всех модулях*.

11.2.5. Использование подфункций

Начиная с версии 5.0 в функции системы MATLAB можно включать *подфункции*. Они объявляются и записываются в теле основных функций и имеют идентичную им конструкцию. Не следует путать эти функции с внутренними функциями, встроенными в ядро системы MATLAB. Ниже представлен пример функции с подфункцией:

```
function [mean,stdev] = statv(x)
%STATV Interesting statistics.
%Пример функции с встроенной подфункцией
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);
%-----
function m = avg(x,n)
%Mean subfunction
m = sum(x)/n;
```

В этом примере среднее значение элементов вектора x вычисляется с помощью подфункции `avg(x, n)`, тело которой записано в теле основной функции `statv`. Пример использования функции `statv` представлен ниже:

```
>> V=[1 2 3 4 5]
V =
           1           2           3           4           5
>> [a,m]=statv(V)
a =
      3
m =
      1.4142
>> statv(V)
ans =
      3
>> help statv
STATV Interesting statistics.
```

Пример функции с встроенной подфункцией

Подфункции определены и действуют локально, то есть только в пределах `m`-файла, определяющего основную функцию. Команда `help name` выводит комментарий, относящийся только к основной функции, тогда как команда `type name` выводит весь листинг `m`-файла. Так что заданные в некотором `m`-файле подфункции нельзя использовать ни в командном режиме работы, ни в других `m`-файлах.

При обращении к функции интерпретатор системы MATLAB прежде всего просматривает `m`-файл на предмет выявления подфункций. Если они обнаружены, то задаются как локальные функции. Благодаря локальному действию подфункций их имена могут совпадать с именами основных функций системы. Если в функции и подфункциях должны использоваться общие переменные, их надо объявить глобальными как в функции, так и в ее подфункциях.

11.2.6. Частные каталоги

Для записи `m`-файлов используются каталоги, называемые *родительскими каталогами*. Они содержат группы файлов определенного функционального назначения, например по статистическим расчетам, матричным операциям, вычислению определенных классов функций и т. д.

Однако, начиная с версии MATLAB 5.0, появилась возможность в родительских каталогах создавать *частные каталоги* с именем **PRIVATE**. Расположенные в них `m`-файлы доступны только файлам родительского каталога. Файлы частных каталогов просматриваются интерпретатором системы MATLAB в первую очередь. Применение частных каталогов позволяет изменять исходные файлы, сохраняя оригиналы в родительском каталоге в неизменном виде.

Если вы решили отказаться от применения измененного файла, достаточно стереть его в частном каталоге. Такая возможность связана с тем, что интерпрета-

тор при поиске m-файла прежде всего просматривает частный каталог и интерпретирует найденный в нем файл. И только если файл не найден, ищется файл в родительском каталоге.

11.3. Обработка ошибок и комментарии

11.3.1. Вывод сообщений об ошибках

Часто в ходе вычислений возникают ошибки. Например, мы уже сталкивались с проблемой вычисления функции $\sin(x)/x$ – при $x = 0$ имеет место ошибка вида «деление на ноль». При появлении ошибки вычисления могут завершиться досрочно с выводом сообщения об ошибке. Следует, однако, отметить, что не все ошибки вызывают остановку вычислений. Некоторые сопровождаются только выдачей предупреждающей надписи.

Такие ситуации должны учитываться программистом, отмечаться как ошибочные и по возможности устраняться. Для вывода сообщения об ошибке служит команда `error('Сообщение об ошибке')`

при выполнении которой вычисления прерываются и выдается сообщение об ошибке, заданное в апострофах. Ниже дан пример вычисления функции $\text{sd}(x)=\sin(x)/x$, в котором задано сообщение об ошибке на русском языке:

```
function f=sd(x)
if x==0 error('Ошибка – деление на 0'), end
f=sin(x)/x
```

Для выявления ситуации об ошибке использован оператор условного перехода `if`, который будет описан детально несколько позднее. Результат выполнения данной функции приводится ниже:

```
>> sd(1)
f =
    0.8415
ans =
    0.8415
>> sd(0)
??? Error using ==> sd
Ошибка – деление на 0
```

Если остановка программы при появлении ошибки нежелательна, то может использоваться команда вывода предупреждающего сообщения:

```
warning('Предупреждающее сообщение')
```

Эта команда выводит стоящее в апострофах сообщение, но не препятствует дальнейшей работе программы. Признаком того, что является ошибкой, а что – предупреждением, являются символы `???` и слово `Warning` в соответствующих сообщениях.

Функция `nargchk` часто используется внутри `m`-файлов для проверки соответствия количества входных параметров (аргументов):

- `msg = nargchk(low, high, number)` возвращает сообщение об ошибке, если число `number` меньше, чем `low`, или больше, чем `high`, в противном случае возвращается пустая строка.

Пример:

```
>> msg = nargchk(4, 9, 5)
msg = [ ]
>> msg = nargchk(4, 9, 2)
msg = Not enough input arguments.
```

- `msg = nargoutchk(low, high, number)` возвращает сообщение об ошибке, если число `number` выходных параметров (выходных аргументов в терминологии MATLAB) меньше, чем `low`, или больше, чем `high`, в противном случае возвращается пустая строка.

11.3.2. Функция `lasterr` и обработка ошибок

Опытные программисты должны предусматривать ситуации с появлением ошибок. К примеру, при $x = 0$ выражение $\sin(x)/x = 0/0 = 1$ и правильным решением было бы вместо его вычисления использовать значение 1.

В данном простом примере приводится функция `sd0`, исключаяющая вычисление $\sin(x)/x$ при $x = 0$:

```
function f=sd0(x)
if x==0 f=1; else f=sin(x)/x; end
return
```

При этом вычисления пройдут корректно при любом x :

```
>> sd0(1)
ans =
      0.8415
>> sd0(0)
ans =
      1
```

Для вывода сообщения о последней произошедшей ошибке служит функция `lasterr` (см. пример ниже):

```
>> aaa
??? Undefined function or variable 'aaa'.
>> 2+3
ans =
      5
>> 1/0
Warning: Divide by zero.
ans =
      Inf
>> lasterr
ans =
Undefined function or variable 'aaa'.
```

Как нетрудно заметить, функция `lasterr` возвращает текстовое сообщение, следующее за знаками `???` сообщения об ошибке.

В общем случае программы могут содержать *обработчики ошибок*, направляющие ход вычислений в нужное русло, даже если появляется ошибка. Но для этого требуются средства индикации и обработки ошибок. Основными из них являются функции `eval` и `lasterr`. О функции `lasterr` уже говорилось, а функция

```
eval('try', 'catch')
```

в отличие от ранее рассмотренной формы имеет два входных аргумента. Один из них – это строчное выражение, которое преобразуется в исполняемую форму и выполняется при отсутствии ошибки. Если же происходит ошибка, то строка `'catch'` вызывает обращение к функции обработки ошибки.

11.3.3. Комментарии

Как отмечалось, команда `help name`, где `name` – имя `m`-файла, обеспечивает чтение первой строки с текстовым комментарием и `tex` строк с комментариями, которые следуют непосредственно за первой строкой. Комментарий, расположенный за пределами этой области, не выводится. Это позволяет создавать невыводимый программный комментарий, например:

```
Z=X+Y %Массив Z является суммой массивов X и Y
```

Пустая строка прерывает вывод комментария при исполнении команды `help name`. Команда `type name` выводит текст программы со всеми комментариями, в том числе и следующими после пустых строк. Надо быть осторожными с вводом русскоязычных комментариев – нередко это является причиной неустойчивой работы программ. С англоязычными комментариями такой проблемы нет.

Команда `help catalog`, где `catalog` – имя каталога с `m`-файлами, позволяет вывести комментарий, общий для всего каталога. Такой комментарий содержится в файле `contents.m`, который пользователь может создать самостоятельно с помощью редактора `m`-файлов. Если такого файла нет, то будет выведен список первых строк комментариев для всех `m`-файлов каталога.

11.4. Функции с переменным числом аргументов

11.4.1. Функции подсчета числа аргументов

Следующие две функции позволяют определить число входных и выходных параметров функции:

- `nargin` возвращает число входных аргументов, определенных для функции. Внутри тела `m`-файла функции `nargin` и `nargout` указывают, соответственно, количество входных или выходных аргументов, заданных пользователем. Вне тела `m`-файла функции `nargin` и `nargout` показыва-

ют, соответственно, число входных или выходных аргументов для данной функции. Отрицательное число аргументов означает, что функция имеет переменное число аргументов;

- `nargin(@fun)` возвращает число объявленных входных параметров для функции `fun`. Если функция имеет переменное число входных аргументов, возвращается `-1`;
- `nargout` возвращает число выходных параметров, определенных для функции;
- `nargout('fun')` возвращает число объявленных выходных параметров для функции `fun`.

Пусть, к примеру, мы хотим создать функцию, вычисляющую сумму квадратов до пяти аргументов `x1`, `x2`, `x3`, `x4` и `x5`. Обычный путь состоит в следующем. Создаем функцию с именем `sum2_5`:

```
function f=sum2_5(x1,x2,x3,x4,x5);
f=x1^2+x2^2+x3^2+x4^2+x5^2;
```

Теперь проверим ее в работе:

```
>> sum2_5(1,2,3,4,5)
ans =
    55
>> sum2_5(1,2)
??? Input argument 'x3' is undefined.
Error in ==> C:\MATLAB\bin\sum2_5.m
On line 2 ==> f=x1^2+x2^2+x3^2+x4^2+x5^2;
```

Итак, при наличии всех пяти аргументов функция работает корректно. Но если аргументов менее пяти, она выдает сообщение об ошибке. С помощью функции `nargin` можно создать функцию `sum2_5m`, которая работает корректно при любом числе заданных входных аргументов в пределах от 1 до 5:

```
function f=sum2m_5(x1,x2,x3,x4,x5);
n=nargin;
if n==1 f=x1^2; end
if n==2 f=x1^2+x2^2;end
if n==3 f=x1^2+x2^2+x3^2; end
if n==4 f=x1^2+x2^2+x3^2+x4^2; end
if n==5 f=x1^2+x2^2+x3^2+x4^2+x5^2;end
```

В данной функции используется условный оператор `if...end`, который будет детально описан далее. Но и без этого ясно, что благодаря применению функции `nargin` и условного оператора вычисления всякий раз идут по формуле с числом слагаемых, равным числу входных аргументов, – от одного до пяти. Это видно из приведенных ниже примеров:

```
>> sum2_5m(1)
ans = 1
>> sum2_5m(1,2)
ans = 5
>> sum2_5m(1,2,3)
ans = 14
```

```
>> sum2_5m(1,2,3,4)
ans = 30
>> sum2_5m(1,2,3,4,5)
ans = 55
>> sum2_5m(1,2,3,4,5,6)
??? Error using ==> sum2_5m
Too many input arguments.
```

Итак, при изменении числа входных параметров от 1 до 5 вычисления проходят корректно. При большем числе параметров выводится сообщение об ошибке. Это уже действует встроенная в интерпретатор MATLAB система диагностики ошибок.

11.4.2. Переменные *varargin* и *varargout*

Для упрощения записи аргументов функций их можно представить списком, который определяет специальная переменная *varargin*, являющаяся массивом ячеек. Она должна записываться строчными буквами и может включать в себя как аргументы, так и опции функций. Например, в приведенных ниже примерах

```
function myplot(x,varargin)
plot(x,varargin{:}) function [s,varargout] = mysize(x)
    nout = max(nargout,1)-1;
    s = size(x);
    for i=1:nout, varargout(i) = {s(i)}; end
```

эта переменная вбирает в себя все входные параметры и опции, начиная со второго аргумента. При обращении к данной функции

```
myplot(sin(0:.1:1), 'color', [.5 .7 .3], 'linestyle', ':')
```

varargin представляет массив ячеек размера 1×4, включающий в себя значения 'color', [.5 .7 .3], 'linestyle' и ':'.

Аналогично *varargin* переменная *varargout* объединяет любое число *выходных* параметров в массив ячеек. Эта переменная, кстати, как и *varargin*, должна быть последней в списке аргументов. Обычно эта переменная не создается при вызове функций. Приведенный ниже пример поясняет ее создание с помощью цикла:

```
function [s,varargout] = mysize(x)
    nout = max(nargout,1)-1;
    s = size(x);
    for i=1:nout,
        varargout(i) = {s(i)};
    end
```

Более подробно циклы будут рассмотрены в дальнейшем описании. В данном случае цикл использован для объединения всех параметров, начиная со второго, в значение переменной *varargout*.

Отметим еще одну полезную функцию:

- `inputname(argnum)` возвращает в тело функции название переменной рабочей области, соответствующее аргументу с номером `argnum`. Может ис-

пользоваться только внутри тела функции. Если входной аргумент не имеет никакого символического представления (например, если это выражение или функция, дающая на выходе выражение, например `a(1)`, `varargin{}`, `eval(expr)`, а не переменная), функция `inputname` возвращает пустую строку (`''`).

11.5. Особенности работы с m-файлами

11.5.1. Выполнение m-файлов-функций

M-файлы-функции могут использоваться как в командном режиме, так и вызываться из других M-файлов. При этом необходимо указывать все входные и выходные параметры. Исключением является случай, когда выходной параметр единственный – в этом варианте функция возвращает единственный результат и может использоваться в математических выражениях. При использовании глобальных переменных они должны быть объявлены во всех m-файлах, используемых в решении заданной задачи, и во всех входящих в них встроенных подфункциях.

Имена функций должны быть уникальными. Это связано с тем, что при обнаружении каждого нового имени MATLAB проверяет, относится ли это имя к переменной, подфункции в данном m-файле, частной функции в каталогах **PRIVATE** или функции в одном из каталогов пути доступа. Если последняя встречается, то будет исполнена именно эта функция. В новой версии MATLAB возможно переопределение функции, но это не рекомендуется делать подавляющему большинству пользователей системы.

Если аргумент функции используется только для вычислений и его значения не меняются, то аргумент передается ссылкой, что уменьшает затраты памяти. В других случаях аргумент передается значением. Для каждой функции выделяется своя (рабочая) область памяти, не входящая в область, предоставляемую системе MATLAB. Глобальные переменные принадлежат ряду областей памяти. При их изменении меняется содержимое всех этих областей.

При решении задач с большим объемом данных может ощущаться нехватка оперативной памяти. Признаком этого становится появление сообщения об ошибке «Out of memory». В этом случае может быть полезным применение следующих мер:

- стирание ставших ненужными данных, прежде всего больших массивов;
- увеличение размеров файла подкачки Windows;
- уменьшение размера используемых данных;
- снятие ограничений на размеры используемой памяти;
- увеличение объема физической памяти компьютера.

Чем больше емкость ОЗУ компьютера, на котором используется система MATLAB, тем меньше вероятность возникновения указанной ошибки. Опыт показывает, что даже при решении задач умеренной сложности емкость ОЗУ не должна быть менее 16–32 Мб.

11.5.2. Создание P-кодов

Когда встречается сценарий или функция в виде m-файла, то всякий раз выполняется трансляция файлов, создающая так называемые P-коды (псевдокоды). Она связана с синтаксическим контролем сценария или функции, который несколько замедляет вычисления. Временные P-коды хранятся в памяти только до использования команды `clear` или завершения сеанса работы. Кроме того, MATLAB позволяет явно создавать и хранить P-коды сценариев и функций с помощью команды `pcode`:

```
pcode имена_m-файлов
```

`pcode * .m` создает файлы p-кодов для всех m-файлов данной папки.

`pcode` с дополнительным параметром `-inplace` хранит эти файлы в тех же папках, что и исходные m-файлы.

Особенно полезно применение этой команды в том случае, когда используются сложная дескрипторная графика и средства создания GUI. В этом случае выигрыш по скорости выполнения вычислений может быть заметным. Переход к P-кодам полезен, если пользователь желает скрыть созданный им m-файл и реализованные в нем идеи и алгоритмы. Файл с P-кодами имеет расширение `.p`. Размер файла с P-кодами обычно больше, чем размер m-файла.

Рассмотрим следующий пример – создадим файл-сценарий `pp.m` следующего содержания:

```
told=cputime;  
x=-15:.0001:15;  
plot(x, sin(x))  
t=cputime-told
```

Эта программа строит график функции $\sin(x)$ по большому числу точек. Кроме того, она вычисляет время выполнения данного сценария в секундах (стоит вспомнить, что для вычисления времени операций применима и конструкция `tic, ..., toc` – см. конец урока 4). При первом пуске получим:

```
>> pp  
t = 0.4400
```

Теперь выполним создание P-кодов и вновь запустим программу:

```
>> pcode pp  
>> pp  
t = 0.3900  
>> pp  
t = 0.3300
```

Нетрудно заметить, что после преобразования в P-коды время построения графика несколько уменьшилось. Но гораздо важнее то, что теперь вы можете стереть файл `pp.m` (но оставить `pp.p`!) и снова запустить программу. Ваши слишком любопытные коллеги едва ли разберутся с тем, что записано в машинных кодах файла `pp.p`, хотя с помощью специальных программ (декомпиляторов) такая возможность в принципе реализуется.

Стоит напомнить, что, начиная с MATLAB 6.5, отдельного этапа генерации Р-кодов уже нет. JIT-ускоритель обеспечивает такую генерацию в момент исполнения того или иного оператора или функции.

11.6. Управляющие структуры

Помимо программ с *линейной структурой* (см. примеры, например, в уроке 7), инструкции которых исполняются строго по порядку, существует множество программ, структура которых *нелинейна*. При этом ветви программ могут выполняться в зависимости от определенных условий, иногда с конечным числом повторений – циклов, иногда в виде циклов, завершаемых при выполнении заданного условия. Практически любая серьезная программа имеет нелинейную структуру. Для создания таких программ необходимы специальные управляющие структуры. Они имеются в любом языке программирования и, в частности, в MATLAB.

11.6.1. Диалоговый ввод

Приведем простой пример диалоговой программы, которую легко поймут приверженцы доброго старого Бейсика:

```
% Вычисление длины окружности с диалоговым вводом радиуса
r=0;
while r>=0,
    r=input('Введите радиус окружности r=');
    if r>=0 disp(' Длина окружности l='); disp(2*pi*r), end
end
```

Эта программа служит для многократного вычисления длины окружности по вводимому пользователем значению радиуса r . Обратите внимание на то, что здесь мы впервые показываем пример организации простейшего диалога. Он реализован с помощью команды `input`:

```
r=input('Введите радиус окружности r=');
```

При выполнении этой команды вначале выводится запрос в виде строки, затем происходит остановка работы программы и ожидается ввод значения радиуса r (в общем случае числа). Ввод, как обычно, подтверждается нажатием клавиши **Enter**, после чего введенное число присваивается переменной r . Следующая строка `if r>=0 disp(' Длина окружности l='); disp(2*pi*r), end`

с помощью команды `disp` при $r>=0$ выводит надпись «Длина окружности $l=$ » и вычисленное значение длины окружности. Она представляет собой одну из наиболее простых управляющих структур типа `if...end`. В данном случае она нужна для остановки вычислений, если вводится отрицательное значение r (прием, который любят начинающие программисты).

Приведенные строки включены в управляющую структуру `while...end`. Это необходимо для циклического повторения вычислений с вводом значений r . Пока

$r \geq 0$, цикл повторяется. Но стоит задать $r < 0$, вычисление длины окружности перестает выполняться, а цикл завершается.

Если данная программа записана в виде m-файла `circ.m`, то работа с ней будет выглядеть следующим образом:

```
>> circ
Введите радиус окружности R=1
Длина окружности l=
    6.2832
Введите радиус окружности R=2
Длина окружности l=
   12.5664
Введите радиус окружности R=-1
>>
```

Итак, на примере даже простой программы мы видим пользу применения управляющих структур типа `if...end` и `while...end`, а также функций диалогового ввода `input('String')` и вывода `disp`. Обратите внимание на завершение работы программы при вводе любого отрицательного числа для радиуса окружности.

Функция `input` может использоваться и для ввода произвольных строковых выражений. При этом она задается в следующем виде:

```
input('Комментарий', 's')
```

При выполнении этой функции она останавливает вычисления и ожидает ввода строкового комментария. После ввода возвращается набранная строка. Это иллюстрирует следующий пример:

```
>> S=input('Введите выражение ', 's')
Введите выражение (Вводим) 2*sin(1)
S =
2*sin(1)
>> eval(S)
ans =
    1.6829
```

Обратите внимание на то, что функция `eval` позволяет вычислить выражение, заданное (полученное от функции `input`) в символьном виде. Вообще говоря, возможность ввода любого символьного выражения в сочетании с присущими языку программирования MATLAB управляющими структурами открывает путь к созданию диалоговых программ любой сложности.

11.6.2. Условный оператор *if...elseif...else...end*

Условный оператор `if` в общем виде записывается следующим образом:

```
if Условие
    Инструкции_1
elseif Условие
    Инструкции_2
```

```

        else
            Инструкции_3
    end
end

```

Эта конструкция допускает несколько частных вариантов. В простейшем, типа `if...end`:

```
if Условие Инструкции end
```

пока `Условие` возвращает логическое значение 1 (то есть «истина»), выполняются `Инструкции`, составляющие тело структуры `if...end`. При этом оператор `end` указывает на конец перечня инструкций. `Инструкции` в списке разделяются оператором `,` (запятая) или `;` (точка с запятой). Если `Условие` не выполняется (дает логическое значение 0, то есть «ложь»), то `Инструкции` также не выполняются.

Еще одна конструкция

```
if Условие Инструкции_1 else Инструкции_2 end
```

выполняет `Инструкции_1`, если выполняется `Условие`, или `Инструкции_2` в противном случае.

Условия записываются в виде:

```
Выражение_1 Оператор_отношения Выражение_2
```

причем в качестве `Операторов_отношения` используются следующие операторы: `==`, `<`, `>`, `<=`, `>=` или `~=`. Все эти операторы представляют собой пары символов без пробелов между ними. Действие операторов отношения, логических операторов и функций, используемых при записи условий, подробно описано в уроке 3.

Мы уже неоднократно показывали применение этой общеизвестной управляющей структуры в программных модулях. Читателю предлагается опробовать собственные варианты программ с условным оператором.

11.6.3. Циклы типа `for...end`

Циклы типа `for...end` обычно используются для организации вычислений с заданным числом повторяющихся циклов. Конструкция такого цикла имеет следующий вид:

```
for var=Выражение, Инструкция, ..., Инструкция end
```

`Выражение` чаще всего записывается в виде `s : d : e`, где `s` – начальное значение переменной цикла `var`, `d` – приращение этой переменной и `e` – конечное значение управляющей переменной, при достижении которого цикл завершается. Возможна и запись в виде `s : e` (в этом случае `d=1`). Список выполняемых в цикле инструкций завершается оператором `end`.

Следующие примеры поясняют применение цикла для получения квадратов значений переменной цикла:

```

>> for i=1:5 i^2, end;
ans = 1
ans = 4
ans = 9

```

```
ans = 16
ans = 25
>> for x=0:.25:1 x^2, end;
ans = 0
ans = 0.0625
ans = 0.2500
ans = 0.5625
ans = 1
```

Возможны вложенные циклы, например:

```
for i=1:3
    for j=1:3
        A(i,j)=i+j;
    end
end
```

В результате выполнения этого цикла (файл **for2.m**) формируется матрица A:

```
>> for2
>> A
A =
     2     3     4
     3     4     5
     4     5     6
```

Следует отметить, что формирование матриц с помощью оператора `:` (двоеточие) обычно занимает меньше места при записи и дает меньшее время исполнения, чем с помощью цикла. Однако применение цикла нередко оказывается более наглядным и понятным.

MATLAB допускает использование в качестве переменной цикла массива A размера $m \times n$. При этом цикл выполняется столько раз, сколько столбцов в массиве A, и на каждом шаге переменная `var` представляет собой вектор, соответствующий текущему столбцу массива A:

```
>> A=[1 2 3;4 5 6]
A =
     1     2     3
     4     5     6
>> for var=A; var, end
var =
     1
     4
var =
     2
     5
var =
     3
     6
```

Циклы `for` часто используются для реализации итерационных алгоритмов с заданным числом итераций. Рассмотрим классический пример – генерацию по-

следовательности чисел Фибоначчи. Первые два числа f_1 и f_2 равны 1, а последующие определяются как сумма двух предыдущих чисел, то есть $f_{i+2} = f_{i+1} + f_i$. Для вычисления цепочки из $N > 2$ чисел Фибоначчи зададим функцию:

```
function f=fib(N)
f=[1 1];
for i=1:N-2
    f(i+2)=f(i+1)+f(i);
end
```

Теперь можно вычислить цепочку чисел Фибоначчи от первого числа до N -го (N – целое число, большее 2):

```
>> fib(10)
ans = 1 1 2 3 5 8 13 21 34 55
```

11.6.4. Циклы типа *while...end*

Цикл типа *while* выполняется до тех пор, пока выполняется Условие:

```
while Условие
    Инструкции
end
```

Пример применения цикла *while* уже приводился. Досрочное завершение циклов реализуется с помощью операторов *break* или *continue*.

Цикл *while* часто используется для подготовки итерационных программ, завершающих свою работу по какому-либо условию. Ниже представлен фрагмент программы, в которой формируется фрактальное изображение, напоминающее лист папоротника (рис. 11.4).

```
function fern
shg
clf reset
set(gcf,'color','white','menubar','none', ...
    'numbertitle','off','name','Fractal Fern')
x = [.5; .5];
h = plot(x(1),x(2),'.');
darkgreen = [0 2/3 0];
set(h,'markersize',1,'color',darkgreen,'erasemode','none');
axis([-3 3 0 10])
axis off
stop = uicontrol('style','toggle','string','stop', ...
    'background','white');
drawnow
p = [ .85 .92 .99 1.00];
A1 = [ .85 .04; -.04 .85]; b1 = [0; 1.6];
A2 = [ .20 -.26; .23 .22]; b2 = [0; 1.6];
A3 = [-.15 .28; .26 .24]; b3 = [0; .44];
A4 = [ 0 0; 0 .16];
cnt = 1; tic
```

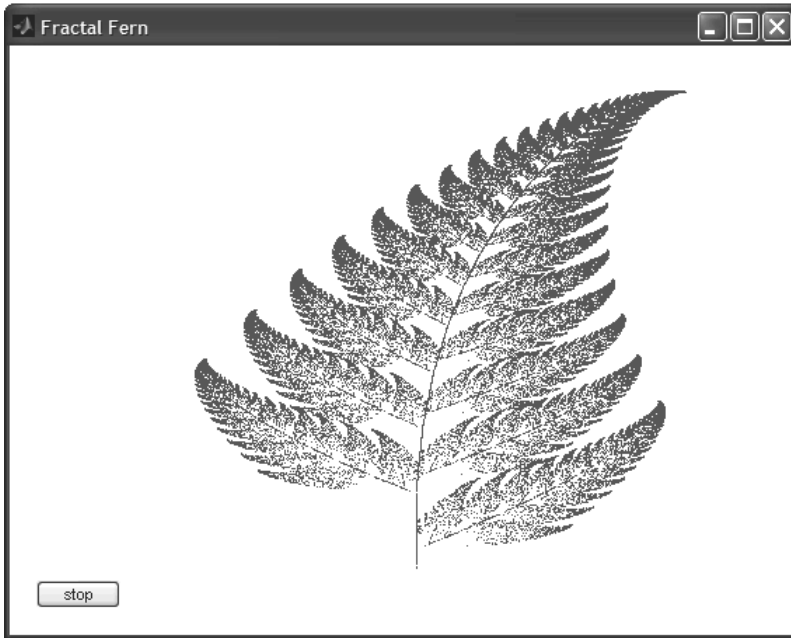


Рис. 11.4. Построение фрактала программой *fern*

```
while ~get(stop,'value')
    r = rand;
    if r < p(1)
        x = A1*x + b1;
    elseif r < p(2)
        x = A2*x + b2;
    elseif r < p(3)
        x = A3*x + b3;
    else
        x = A4*x;
    end
    set(h,'xdata',x(1),'ydata',x(2));
    drawnow
    cnt = cnt + 1;
end
t = toc; s = sprintf('%8.0f points in %6.3f seconds',cnt,t);
text(-1.5,-0.5,s,'fontweight','bold');
set(stop,'style','pushbutton','string','close','callback','close(gcf)')
```

Описание алгоритма построения фрактала выходит за рамки этой книги. Отметим лишь, что данная функция является хорошим примером реализации специальной графики. График фрактала строится непрерывно до нажатия кнопки **Stop**, завершающего работу цикла построения фрактала. При этом выводятся время построения и число точек фрактального рисунка.

11.6.5. Конструкция переключателя `switch...case...end`

Для осуществления множественного выбора (или ветвления) используется конструкция с переключателем типа `switch`:

```
switch switch_Выражение
    case case_Выражение
        Список_инструкций
    case {case_Выражение1, Case_выражение2, case_Выражение3,...}
        Список_инструкций
    ...
    otherwise,
        Список_инструкций
end
```

Если выражение после заголовка `switch` имеет значение одного из выражений `case_Выражение...`, то выполняется блок операторов `case`, в противном случае – список инструкций после оператора `otherwise`. При выполнении блока `case` исполняются те списки инструкций, для которых `case_Выражение` совпадает с `switch_Выражением`. Обратите внимание на то, что `case_Выражение` может быть числом, константой, переменной, вектором ячеек или даже строчной переменной. В последнем случае оператор `case` истинен, если функция `strcmp` (значение, выражение) возвращает логическое значение «истинно».

Поясним применение оператора `switch` на примере `m`-файла **sw1.m**:

```
switch var
case {1,2,3}
    disp('Первый квартал')
case {4,5,6}
    disp('Второй квартал')
case {7,8,9}
    disp('Третий квартал')
case {10,11,12}
    disp('Четвертый квартал')
otherwise
    disp('Ошибка в задании')
end
```

Эта программа в ответ на значения переменной `var` – номера месяца – вычисляет, к какому кварталу относится заданный месяц, и выводит соответствующее сообщение:

```
>> var=2;
>> sw1
Первый квартал
>> var=4;sw1
Второй квартал
>> var=7;sw1
Третий квартал
```

```
>> var=12;sw1
Четвертый квартал
>> var=-1;sw1
Ошибка в задании
```

11.6.6. Конструкция *try...catch...end*

В MATLAB 6.* была введена конструкция блока вывода ошибок *try...catch...end*:

```
try,
    Список инструкций
    ...,
    Список инструкций
catch,
    Список инструкций
    ...,
    Список инструкций
end
```

Эта конструкция выполняет все списки инструкций. Если в каком-то списке до оператора *catch* появляется ошибка, то выводится сообщение об ошибке, но системная переменная последней ошибки *lasterr* не меняется. В сообщениях после *catch* сообщения об ошибке не выводятся.

В следующем примере задано появление ошибки (переменная *aaa* не определена), после чего выполняется блок *try...catch...end*:

```
aaa
??? Undefined function or variable 'aaa'.
try
    2+3;
    3+4;
    2/0;
catch
    4+5;
end;
Warning: Divide by zero.
>> lasterr
ans =
Undefined function or variable 'aaa'.
```

Обратите внимание, что в конце блока команда *lasterr* выводит ее начальное значение. А в другом примере ошибка задана в блоке *try...catch...end* уже после оператора *catch*:

```
>> try
    2+3;
    3+4;
    4+5;
catch
    5/0;
end;
```

```
>> lasterr
ans =
Undefined function or variable 'aaa'.
```

Как нетрудно заметить, на этот раз ошибка в вычислении $5/0$ не выводится, а значение `lasterr` осталось тем, что было изначально.

11.6.7. Операторы *break*, *continue* и *return*

В управляющих структурах, в частности в циклах `for` и `while`, часто используются операторы, влияющие на их выполнение.

Так, оператор `break` может использоваться для досрочного прерывания выполнения цикла. Как только он встречается в программе, цикл прерывается. Пример:

```
for i=1:10 i, if i==5 break, end, end;
i = 1
i = 2
i = 3
i = 4
i = 5
```

Оператор `continue` передает управление в следующую итерацию цикла, пропуская операторы, которые записаны за ним, причем во вложенном цикле он передает управление на следующую итерацию основного цикла. Приведенный ниже пример обеспечивает подсчет числа строк в файле `magic.m`:

```
fid = fopen('magic.m','r'); count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) | strncmp(line,'% ',1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines',count));
```

25 lines

Оператор `return` обеспечивает нормальный возврат в вызывающую функцию или в режим работы с клавиатурой. Пример применения оператора `return`:

```
function d = det(A)
%DET det(A) is the determinant of A.
if isempty(A)
    d = 1;
    return
else
    ...
end
```

В данном примере, если матрица `A` пустая, будет выведено значение `1`, после чего управление будет передано в блок `else...end`.

11.6.8. Пустые матрицы в структурах *if* и *while*

В управляющих структурах *if* и *while* в качестве условия могут применяться пустые матрицы. Такое условие возвращает логическое значение *false* (0). Пример фрагмента программы с пустой матрицей:

```
A=ones(10,0,4);  
if A S1 else S0 end
```

11.6.9. Создание паузы в вычислениях

Для остановки программы используется оператор *pause*. Он используется в следующих формах:

- *pause* останавливает вычисления до нажатия любой клавиши;
- *pause* (N) останавливает вычисления на N секунд;
- *pause on* включает режим отработки пауз;
- *pause off* выключает режим отработки пауз.

Следующий пример поясняет применение команды *pause*:

```
for i=1:20;  
    x = rand(1,40);  
    y = rand(1,40);  
    z = sin(x.*y);  
    tri = delaunay(x,y);  
    trisurf(tri,x,y,z)  
    pause(1);  
end
```

Команда *pause* (1) здесь обеспечивает показ 20 рисунков – построений трехмерных поверхностей из треугольных окрашенных областей со случайными параметрами.

11.7. Основы объектно-ориентированного программирования

11.7.1. Основные понятия

Мы уже много раз упоминали различные *объекты* языка программирования системы MATLAB. Это является одним из признаков *объектно-ориентированного программирования* (ООП), причем чисто внешним. В основе объектно-ориентированного программирования лежат три основных положения.

- *Инкапсуляция* – объединение данных и программ и передача данных через входные и выходные параметры функций. В результате появляется новый элемент программирования – *объект*.

- *Наследование* – возможность создания родительских объектов и новых дочерних объектов, наследующих свойства родительских объектов. Возможно также *множественное наследование*, при котором класс наследует свойства *нескольких* родительских объектов. На наследовании основаны система задания типов данных, дескрипторная графика и многие другие приемы программирования. Примеры наследования мы уже неоднократно отмечали.
- *Полиморфизм* – присвоение некоторому действию одного имени, которое в дальнейшем используется по всей цепочке создаваемых объектов сверху донизу, причем каждый объект выполняет это действие присущим ему способом.

В дополнение к этим положениям объектно-ориентированное программирование в MATLAB допускает *агрегирование* объектов, то есть объединение частей объектов или ряда объектов в одно целое.

11.7.2. Классы объектов

Объект можно определить как некоторую структуру, принадлежащую к определенному *классу*. В MATLAB определены следующие семь основных классов объектов:

- `double` – числовые массивы с элементами-числами двойной точности;
- `sparse` – двумерные числовые или комплексные разреженные матрицы;
- `char` – массивы символов;
- `struct` – массивы структур (записей);
- `cell` – массивы ячеек;
- `javaarray` – массивы Ява;
- `function_handle` – дескрипторы функций.

С объектами этих классов мы многократно встречались, особо не оговаривая их принадлежность к объектно-ориентированному программированию. Для MATLAB вообще характерно, что никакие классы объектов (в том числе заново создаваемые) не требуют объявления. Например, создавая переменную `name='Иван'`, мы автоматически получаем объект в виде переменной `name` класса `char`. Таким образом, для переменных принадлежность к тому или иному классу определяется их значением. Является ли переменная объектом, можно определить при помощи функции `isobject` (имя переменной). Аналогичная функция `isjava` определяет, является ли переменная объектом языка Java.

Для создания новых классов объектов служат *конструкторы классов*. По существу, это `m`-файлы, имена которых совпадают с именами классов `@Имя_класса`, но без символа `@`. Этим символом помечаются поддиректории системы MATLAB, в которых имеются конструкторы классов. Множество таких директорий с примерами конструкторов классов вы найдете в поддиректориях **MATLAB\TOOLBOX**.

В качестве примера рассмотрим поддиректорию **@SYM** в директории **TOOLBOX\SYMBOLIC**. В этой поддиректории можно найти конструкторы для

более чем сотни объектов пакета символьной математики. К примеру, конструктор функции, вычисляющей арктангенс, выглядит следующим образом:

```
>> help @sym/atan.m
  ATAN Symbolic inverse tangent.
>> type @sym/atan.m
function Y = atan(X)
%ATAN Symbolic inverse tangent.
%      Copyright (c) 1993-98 by The MathWorks, Inc.
%      $Revision: 1.10 $   $Date: 1997/11/29 01:05:16 $
Y = maple('maple', 'atan', X);
```

В данном случае для конструирования нужного объекта используется функция `maple`, дающая вход в ядро системы символьной математики Maple, которое поставляется в составе системы MATLAB по лицензии фирмы MapleSoft, Inc. Этот пример, кстати, наглядно показывает, что пользователь системы MATLAB может существенно расширить число объектов класса `sym`, поскольку ядро системы Maple содержит намного больше определений, чем пакет символьной математики системы MATLAB. Для создания новых классов объектов служит функция `class`, описанная ниже.

Пакеты прикладных программ системы MATLAB позволяют разработчикам с большим успехом использовать возможности объектно-ориентированного программирования путем создания новых классов и объектов. М-файлы системы представляют собой массу наглядных примеров объектно-ориентированного программирования на языке MATLAB. Это дает основание ограничиться справочным описанием основных средств такого программирования с приведением минимума простых примеров.

11.7.3. Создание класса или объекта

Для создания класса объектов или объектов, а также для их идентификации служит функция `class`. Формы ее применения представлены ниже:

- `class(OBJ)` возвращает класс указанного объекта `OBJ`. Типы стандартных классов `double`, `sparse`, `char`, `cell`, `struct`, `function_handle` были перечислены выше. `int8` – 8-разрядный массив целых чисел со знаком; `uint8` – 8-разрядный массив целых чисел без знака; `int16` – 16-разрядный массив целых чисел со знаком; `uint16` – 16-разрядный массив целых чисел без знака; `int32` – 32-разрядный массив целых чисел со знаком; `uint32` – 32-разрядный массив целых чисел без знака; `int64` – 64-разрядный массив целых чисел со знаком; `uint64` – 64-разрядный массив целых чисел без знака; `<class_name>` – класс, определенный пользователем; `<java_class>` – имя класса Ява;
- `OBJ=class(S, 'class_name', PARENT1, PARENT2, ...)` создает объект класса `'class_name'` на базе структуры `S` и родительских объектов `PARENT1`, `PARENT2`, ... При этом создаваемый объект наследует структуру и поля родительских объектов. Объекту `OBJ` в данном случае присуще *множественное наследование*;

- `OBJ=class(struct[], 'class_name', PARENT1, PARENT2, ...)` не может иметь никаких полей, кроме унаследованных от родительских объектов. Обратите внимание на то, что эта функция обычно используется в составе `m`-файлов конструкторов классов объектов.

11.7.4. Проверка принадлежности объекта к заданному классу

Для контроля принадлежности заданного объекта к некоторому классу служит функция `isa`:

- `isa(OBJ, 'Имя_класса')` возвращает логическую 1, если `OBJ` принадлежит классу с указанным именем. Дополнительно к вышеописанным выделяет классы `numeric` и `single`. Но не обнаруживает класс `logical`. Нужно использовать функцию `islogical`, чтобы проверить принадлежность к этому классу. Примеры применения этой функции:

```
>> X=[1 2 3];
>> isa(X, 'char')
ans = 0
>> isa(X, 'double')
ans = 1
```

11.7.5. Другие функции объектно-ориентированного программирования

Для получения списка методов данного класса объектов сейчас чаще используются функции `methodview` и `methods`. Отличиями от `what` имя класса является то, что эти функции возвращают информацию также и о классах Java, но информация выводится в отдельном окне, не сообщается информация о папках, все методы из всех папок собраны вместе, и повторяющиеся имена методов удалены:

- `methodview` имя класса или `methods` имя класса `-full` в отдельном окне возвращают полное описание методов класса, включая информацию о наследовании, а для классов Java – и о подписях и атрибутах;
- `M=methods('имя класса', '-full')` возвращает ту же информацию в массиве ячеек `M`;
- `M=methods('имя класса')` возвращает массив ячеек с перечислением методов, относящихся к заданному классу объектов;
- `methods` имя класса возвращает перечень методов в отдельном окне. На-пример:

```
>> methods char
Methods for class char:
delete diff int
```

Следующие две функции могут использоваться только внутри конструкторов классов:

`inferiorto('CLASS1','CLASS2',...)` и `superiorto('CLASS1','CLASS2',...)`

Они определяют низший и высший приоритеты классов по отношению к классу конструктора.

Для дескрипторов перегружаемых функций существует функция `functions`.

`F=functions` (дескриптор функции), возвращающая массив структур `F.METHODS`, вложенный в массив `F`, при этом именем поля в массиве `F.METHODS` является имя класса, а значением поля – название метода, который вызывается тогда, когда входной аргумент принадлежит этому классу.

Дополнительно `functions` возвращает следующие структуры:

- **F.function** – строка, используемая для создания дескриптора функции (существует также отдельная функция `func2str` для получения этой информации и обратная ей функция `str2func`, превращающая строку в дескриптор функции);
- **F.type** содержит **simple** (простая), **overloaded** (перегружаемая) или **subfunction** (подфункция), то есть указывает тип функции;
- **F.default** указывает путь к тому файлу, который первый в алгоритме поиска MATLAB и не определен никаким классом;
- функция `which` имя метода находит загруженный Java-класс и все классы MATLAB, которым принадлежит данный метод;
- функция `which -all` имя метода находит все классы, которым принадлежит данный метод.

Любой оператор в системе MATLAB можно *переопределить* (то есть сделать его функцию перегружаемой) путем задания `m`-файла с новым именем в соответствующем каталоге классов. В частности, в уроке 8 отмечалось, что все арифметические операторы имеют представления в виде соответствующих функций.

11.8. Handle- и inline-функции

11.8.1. Задание handle-функции

MATLAB позволяет создавать особые объекты, называемые handle-функциями (анонимными, или дескрипторными, функциями). В этих функциях могут использоваться все объекты структурного программирования. Для конструирования handle-функции используется описатель в виде единичного символа `@`. Создадим, к примеру, handle-функцию с именем `fhsin`, вычисляющую значение синуса:

```
>> fhsin=@sin
fhsin =
    @sin
```

В том, что эта функция вовсе не обычная функция $\sin(x)$, убеждает следующий пример на неудачную попытку вычисления значения $\sin(1)$:

```
>> fhsin(1)
ans =
    @sin
```

Нетрудно заметить, что вычисления не произошло и просто выдано определение `handle`-функции.

Итак, `handle`-функция характеризуется своим именем – в нашем случае `fhsin`. Однако она не имеет списка параметров в круглых скобках (в общем случае `handle`-функция может иметь несколько аргументов). На имена `handle`-функций накладываются те же ограничения, что и на имена функций в виде `m`-файлов. С помощью команд `save` и `load` можно записывать `handle`-функции на диск и считывать их в рабочую область.

11.8.2. Вычисление и применение `handle`-функций

Для вычисления `handle`-функций служит функция

```
feval(fhandle, a1, a2, a3,...)
```

Здесь – имя `handle`-функции без описателя `@`, `a1`, `a2`, `a3`, ... – список аргументов `handle`-функции. `Handle`-функции иногда называют *анонимными функциями*.

Теперь мы можем вычислить нашу `handle`-функцию:

```
>> feval(fhsin,1)
ans = 0.8415
```

Можно также построить график `handle`-функции, например в нашем случае используя команду:

```
>> plot(feval(fhsin,0:.01:2*pi))
```

`Handle`-функция часто используется в качестве функции пользователя – см. примеры в уроке 1, помещенные в разделе функций пользователя.

11.8.3. *Inline*-функции

Как уже отмечалось в уроке 1, в `MATLAB` имеется еще один важный класс функций, способных задавать функции пользователя. Это так называемые `inline`-функции:

```
g = inline(expr)           g = inline(expr, arg1, arg2, ...)
g = inline(expr, n)
```

Здесь `expr` – выражение, `a1`, `a2`, ... – аргументы, `n` – число параметров вида `p1`, `p2`, ...

В функциях пользователя особенно удобно задавать `expr` в виде строкового выражения в апострофах. Например:

```
>> sc2=inline('sin(x).^2+cos(y).^2')
sc2 =
    Inline function:
    sc2(x,y) = sin(x).^2+cos(y).^2
```

В апострофах определения `inline` могут быть записаны любые допустимые математические выражения, что открывает простор для задания функций пользователя на основе этих выражений.

11.8.4. Преобразования *handle-* и *inline-*функций

На основе *inline-*функций можно создавать *handle-*функции. Например:

```
>> fh=@sc2;
```

Для преобразования *handle-*функции в функцию, заданную именем, используется функция преобразования `func2str`. Пример:

```
>> func2str(fhsin)
ans =
sin
```

Для обратного преобразования (функции, заданной именем, в *handle-*функцию) служит функция преобразования `str2func`, например:

```
>> str2func('cos')
ans =
@cos
```

Для дополнительного знакомства с особенностями применения *handle-*функций стоит посмотреть материалы и примеры по ним в справке MATLAB.

11.9. Отладка программ

Отладка программ – не менее серьезный этап, чем их подготовка. К сожалению, это редко учитывают начинающие программисты, ослепленные успехом работы первых простеньких программ. Однако по мере усложнения программ необходимость в средствах их отладки возрастает. Этот раздел посвящен тем средствам отладки, которые имеются в системе MATLAB.

11.9.1. Общие замечания по отладке *m-*файлов

Основным средством отладки *m-*файлов в системе MATLAB является встроенный редактор/отладчик (M-File Editor/Debugger) с современным графическим интерфейсом. Однако MATLAB предусматривает основные возможности отладки и в командном режиме. Именно они и рассматриваются в данном разделе.

Вообще говоря, отладка программ – процесс сугубо индивидуальный и творческий. Большинство пользователей средней квалификации обычно отлаживают программы, не обращаясь к специальным средствам отладки, требующим дополнительного времени для освоения и приобретения навыков использования. Если алгоритм решения задачи достаточно прост, то после нескольких модернизаций программы ее удастся довести до нужной «кондиции», то есть заставить работать корректно.

Для этого часто бывает достаточно ввести в программу режим просмотра результатов промежуточных вычислений, разблокировав их вывод снятием опера-

тора ; (точка с запятой) или введя дополнительные переменные, значения которых отражают ход вычислений. После отладки можно вновь ввести блокирующие вывод операторы и убрать указанные переменные. С помощью знака «%» можно остановить исполнение отдельных строк программы, превратив их в текстовые комментарии.

Серьезная необходимость в применении средств отладки возникает при отладке сложных программ опытными программистами, которые наверняка имеют практику работы с универсальными языками программирования с использованием развитых отладочных средств.

11.9.2. Команды отладки программ

Для перехода в командный режим отладки в *m*-файл следует включить команду `keyboard`. Ее можно запустить и в командном режиме:

```
>> keyboard
K>> type sw1

switch var
case {1,2,3}
    disp('Первый квартал')
case {4,5,6}
    disp('Второй квартал')
case {7,8,9}
    disp('Третий квартал')
case {10,11,12}
    disp('Четвертый квартал')
otherwise
    disp('Ошибка в задании')
end

K>> return
>>
```

Признаком перехода в режим отладки становится появление комбинированного символа `K>>`. Он меняется на символ `>>` после возврата командой `return` в обычный командный режим работы. То же самое происходит при использовании команды `dbquit`, также прекращающей режим отладки, но прекращающей и выполнение *m*-файла. Если команда `return` находится в самом *m*-файле, она прерывает его выполнение и передаст управление туда, откуда был вызван этот файл.

11.9.3. Вывод листинга *m*-файла с пронумерованными строками

Один из способов отладки *m*-файлов – размещение в них точек прерывания. В этих точках выполнение программы приостанавливается и можно начать анализ ее поведения, например просмотреть значения переменных. Однако в команд-

ном режиме нельзя задать установку таких точек с помощью курсора мыши (как в отладчике Windows). Поэтому необходимо иметь листинг программы с пронумерованными строками. Он создается с помощью команды `dbtype`, действие которой иллюстрирует следующий пример:

```
>> keyboard
K>> dbtype swl

1      switch var
2      case {1,2,3}
3          disp('Первый квартал')
4      case {4,5,6}
5          disp('Второй квартал')
6      case {7,8,9}
7          disp('Третий квартал')
8      case {10,11,12}
9          disp('Четвертый квартал')
10     otherwise
11         disp('Ошибка в задании')
12     end
K>>
```

11.9.4. Установка, удаление и просмотр точек прерывания

Для установки в тестируемый m-файл точек прерывания используются следующие команды:

- `dbstop in M-file at lineno` – установить точку прерывания в заданной строке;
- `dbstop in M-file at subfun` – установить точку прерывания в подфункции;
- `dbstop in M-file` – установить точку прерывания в m-файле;
- `dbstop if error` – установить точку прерывания при сообщении об ошибке, но не при ошибках внутри цикла `try...catch`;
- `dbstop if all error` – установить точку прерывания при сообщении о любой ошибке;
- `dbstop if warning` – установить точку прерывания при предупреждении;
- `dbstop if infnan` или `naninf` – установить точку прерывания при результате `Inf` или `NaN`.

Можно использовать упрощенный ввод этих команд без использования слов `in`, `at` и `if`. При установке контрольной точки она появляется в окне редактора/отладчика m-файлов. Для удаления точек прерывания используется команда `dbclear` с тем же синтаксисом, что и `dbstop`, например:

- `dbclear M-file at lineno` – удалить точку прерывания в заданной строке заданного файла.

Команда `dbstatus` выводит список установленных в данной сессии точек прерывания, например:

```
K» dbstatus
Breakpoint for C:\MATLAB\bin\demo1.m is on line 2.
Breakpoint for C:\MATLAB\bin\sd.m is on line 3.
```

Уже в системе MATLAB 6 значительно изменен синтаксис программ по сравнению с предыдущими версиями. Поэтому полезно перед отладкой старых программ выполнить команду `feature('orAndError',0)` или просто `feature('orAndError')` – для выдачи предупреждений об ошибке при выполнении тех конструкций, интерпретация которых в новой версии изменилась. `feature('orAndError',1)` выдает сообщение об ошибке вместо предупреждения об ошибке.

11.9.5. Управление исполнением m-файла

После установки точек прерывания начинается собственно процесс тестирования m-файла. Он заключается в исполнении одного или нескольких шагов программы с возможностью просмотра содержимого рабочей области, то есть значений переменных, меняющихся в ходе выполнения программы. Для пошагового выполнения программы используется команда `dbstep` в следующих формах:

- `dbstep` – выполнение очередного шага;
- `dbstep nlines` – выполнение заданного числа строк программы;
- `dbstep in` – если следующая выполняемая строка текущего m-файла является вызовом функции из другого m-файла, эта форма позволяет перейти к первой исполняемой строке *вызываемой* функции и остановиться там;
- `dbstep out` – если следующая выполняемая строка текущего m-файла является вызовом функции из другого m-файла, эта форма позволяет перейти к *вызываемой* функции и остановиться сразу же после ее выполнения.

Для перехода от одной остановки программы к другой используется команда `dbcont`.

11.9.6. Просмотр рабочей области

В точках прерывания пользователь имеет возможность просмотреть состояние рабочей области с помощью ранее описанных команд `who` и `whos`. Кроме того, для перемещения по рабочим областям стека вызванных функций вверх или вниз используются следующие команды:

- `dbdown` – перемещение в стеке вызываемых функций сверху вниз;
- `dbup` – перемещение в стеке вызываемых функций снизу вверх.

Находясь в рабочей области, можно не только просматривать значения переменных, но и менять их в ходе отладки программы. С помощью команды `dbstack` можно просматривать стек функций. Для завершения отладки используется команда `dbquit`.

В заключение еще раз обращаем внимание читателя на то, что все возможности отладки реализованы в редакторе/отладчике m-файлов, который характеризуется удобным графическим интерфейсом и средствами визуализации отладки про-

грамм. К ним относятся возможность выделения различными цветами элементов m-файла (ключевых слов, переменных, комментариев и т. д.), наглядное представление точек прерывания, простота их установки и т. д. Щелкнув мышью справа от колонки с номерами строк, вы можете установить/снять точку прерывания при помощи мыши в окне редактора-отладчика. После остановки в точке прерывания вы можете просмотреть значения всех переменных, подведя курсор мыши к символному обозначению переменной в окне редактора-отладчика.

В этом отношении некоторые описанные выше приемы отладки в командном режиме выглядят несколько архаично. Скорее всего, они ориентированы на пользователей, привыкших к командному режиму работы с системой.

11.9.7. Профилирование m-файлов

Вообще говоря, достижение работоспособности программы – лишь один из этапов ее отладки. Не менее важным вопросом является *оптимизация* программы по минимуму времени исполнения или по минимуму объема кодов. Современные компьютеры, в которых используется система MATLAB, имеют достаточные резервы памяти, так что размеры программы, как правило, не имеют особого значения. Намного важнее проблема оптимизации программы в части быстродействия.

Оценка времени исполнения отдельных частей программы называется ее *профилированием*. Для выполнения такой процедуры служит команда `profile`, имеющая ряд опций:

- `profile fun` – запуск профилирования для функции `fun`;
- `profile report` – вывод отчета о профилировании;
- `profile plot` – графическое представление результатов профилирования в виде диаграммы Парето;
- `profile filename` – профилирование файла с заданным именем и путем;
- `profile report N` – вывод отчета по профилированию заданных `N` строк;
- `profile report frac` – выводит отчет по профилированию тех строк, относительная доля выполнения которых в общем времени выполнения составляет не менее чем `frac` (от 0.0 до 1.0);
- `profile on` – включение профилирования;
- `profile off` – выключение профилирования;
- `profile reset` – выключение профилирования с уничтожением всех накопленных данных;
- `INFO = profile` – возвращает структуру со следующими полями:
- `file` – полный путь к профилируемому файлу;
- `interval` – интервалы времени в секундах;
- `count` – вектор измерений;
- `state` – состояние профилировщика: `'on'` (включен) или `'off'` (выключен).

Следует отметить, что средства профилирования MATLAB позволяют анализировать только m-файлы функций, но не сценариев. Чтобы получить профиль выполнения сценария, приходится преобразовывать его в функцию (как правило,

не имеющую входных и выходных параметров), добавляя соответствующий заголовок `function`.

Ниже приводится пример на профилирование `m`-файла `ellipj` (эллиптическая функция Якоби):

```
>> profile on
>> profile ellipj
>> ellipj([0:0.01:1],0.5);
>> profile report
Total time in "C:\MATLAB\toolbox\matlab\specfun\ellipj.m": 0.16 seconds
100% of the total time was spent on lines:
    [96 97 86]
      85:   if ~isempty(in)
0.01s,   6%   86:       phin(i,in) = 0.5 * ...
              87: (asin(c(i+1,in).*sin(rem(phin(i+1,in),2*pi)))./
a(i+1,in))
              95: m1 = find(m==1);
0.11s,  69%   96: sn(m1) = tanh(u(m1));
0.04s,  25%   97: cn(m1) = sech(u(m1));
              98: dn(m1) = sech(u(m1));
>> INFO=profile
INFO =
      file: 'C:\MATLAB\toolbox\matlab\specfun\ellipj.m'
 interval: 0.0100
  count: [98x1 double]
  state: 'off'
>> profile plot
```

Нетрудно заметить, что при профилировании выводятся номера строк программы, у которых время выполнения превосходит 0,01 с. С использованием этого интервала и оценивается время исполнения программного кода. Последняя команда выводит графическую диаграмму профилирования, показанную на рис. 11.5.

При графическом представлении профилирования по горизонтальной оси указываются номера строк, а по вертикальной – время их выполнения. Сначала показываются строки с наибольшим временем выполнения. Таким образом, программист, отлаживающий работу программы, имеет возможность наглядно оценить, где именно находятся критические по быстродействию фрагменты.

11.9.8. Создание итогового отчета

Для создания суммарного отчета о профилировании служит команда `profsum`, которая используется в нескольких формах:

- `profsumm` – вывод полного отчета о результатах профилирования `m`-файла. Выводятся данные о времени выполнения для строк, суммарное время выполнения которых составляет 95% от общего времени (если таких строк много, выводятся данные о 10 строках, выполнение которых заняло наибольшее время);

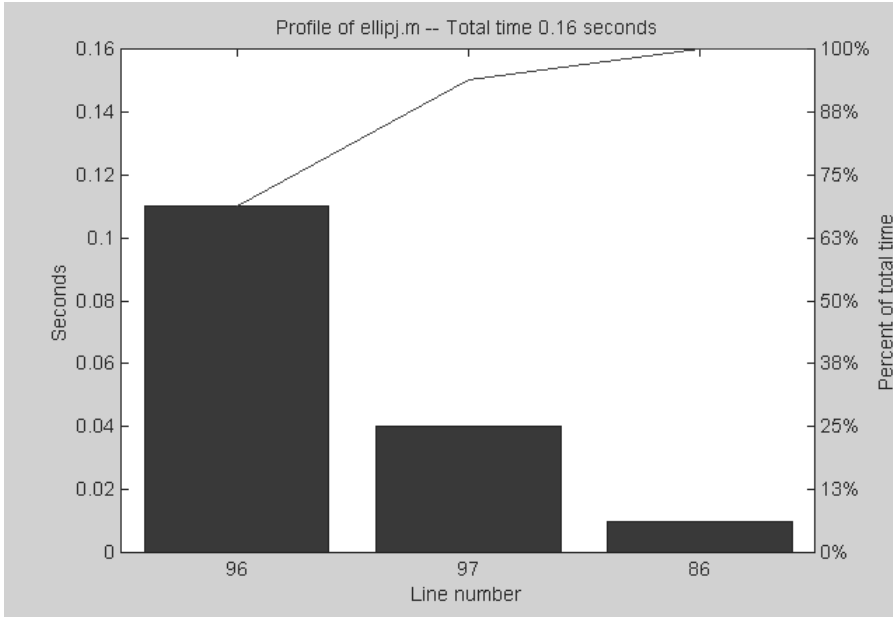


Рис. 11.5. Графическое представление результатов профилирования

- `profsumm(FRACTION)` – выводит часть отчета для строк, относительное время выполнения которых составляет `FRACTION` (от 0.0 до 1.0) от общего времени выполнения файла;
- `profsumm(N)` – выводится отчет по `N` строкам с максимальным временем выполнения;
- `profsumm(STR)` – выводит отчет только по тем строкам, в которых встречается строковое выражение `STR`;
- `profsumm(INFO)`, `profsumm(INFO, FRACTION)`, `profsumm(INFO, N)` и `profsumm(INFO, STR)` – выводят итоговый отчет по строкам, заданным массивом `INFO`.

Пример применения команды `profsumm`:

```
>> profile erfc core
>> z = erf(0:.01:100);
>> profsumm, profile done
Total time in "C:\MATLAB\toolbox\matlab\specfun\erfc core.m":
0.16 seconds
94% of the total time was spent on lines:
    [99 109 108 97 95 94 92 86 71 48]
    47: %
0.01s, 6%  48: k = find((abs(x) > xbreak) & (abs(x) <= 4.));
    49:     if ~isempty(k)
    70:         del = (y-z).*(y+z);
```

```

0.01s, 6%71: result(k) = exp(-z.*z) .* exp(-del) .* result(k)
           72:         end
           85:
0.01s, 6% 86:         y = abs(x(k));
           87:         z = 1 ./ (y .* y);
           91:         xnum = (xnum + p(i)) .* z;
0.01s, 6% 92:         xden = (xden + q(i)) .* z;
           93:         end
0.01s, 6% 94: result(k) = z .* (xnum + p(5)) ./ (xden + q(5));
0.01s, 6% 95: result(k) = (1/sqrt(pi) - result(k)) ./ y;
           96:         if jint ~= 2
0.01s, 6% 97:         z = fix(y*16)/16;
           98:         del = (y-z).*(y+z);
0.06s, 38% 99: result(k) = exp(-z.*z) .* exp(-del) .* result(k)
           100:        k = find(~isfinite(result));
           107:   if jint == 0
0.01s, 6% 108: k = find(x > xbreak);
0.01s, 6% 109: result(k) = (0.5 - result(k)) + 0.5;
           110: k = find(x < -xbreak);

```

11.9.9. Построение диаграмм Парето

Команда `profile plot` использует для построения диаграммы Парето графическую команду `pareto`. Диаграмма Парето представляет собой столбцы, расположенные в порядке убывания отображаемых значений. С другими возможностями команды `pareto` можно ознакомиться, выполнив команду `help pareto`.

- `pareto(Y, NAMES)` строит диаграмму Парето с пометкой столбцов значений вектора `Y` соответствующими именами, содержащимися в векторе `NAMES`;
- `pareto(Y, X)` строит диаграмму Парето для значений вектора `Y` в зависимости от значений вектора `X`;
- `pareti(Y)` строит диаграмму Парето для значений вектора `Y` в зависимости от индексов его элементов;
- `[H, AX]=pareto(...)` возвращает вектор дескрипторов графических объектов диаграммы `H` и их осей `AX`.

Пример построения диаграммы Парето был рассмотрен выше (см. рис. 11.5).

11.9.10. Работа с системой контроля версий

MATLAB поддерживает системы контроля версий кода Visual Source Safe фирмы Microsoft (вместе с Visual Studio), PVCS фирмы Merant (упрощенные версии этой системы бесплатно поступают с продуктами Borland), Clear Case фирмы Rational Software (в особенности на UNIX-Linux-версиях MATLAB), RCS и имеет настраиваемый пользовательский интерфейс, так что вы можете подключить свою лю-

бимую систему. Функция `smopts` выводит информацию об установленной системе контроля версий. Первоначально установленная система MATLAB реагирует следующим образом:

```
>> smopts
ans =
none
```

Для подключения PVCS или ее варианта вам нужно отредактировать m-файл `smopts.m` в папке `C:\matlabr12\toolbox\local`. Введите комментарий `%begin customization section` и введите на следующей строчке m-файла, если файл конфигурации проекта `Proj.cfg`:

```
DefaultConfigFile='c:\pvcs\pvcsproj\projmgrprj\Proj.cfg',
```

Проверим правильность нашей установки:

```
>> smopts('DefaultConfigFile')
DefaultConfigFile =
c:\pvcs\pvcsproj\projmgrprj\Proj.cfg
```

Команда `checkin(filename, 'COMMENTS', Текст комментариев, OPTION1, VALUE1, OPTION2, VALUE2.....)` включает ваши файлы в систему контроля версий. `filename` – полный путь к файлу или строковый массив ячеек, где каждая ячейка указывает путь к файлам, `Текст комментариев` – массив символов, в данной версии `option` может быть только `lock`, `value` может быть `on` (замкнута) или `off` (позволяет доступ к файлу без `checkout`).

Команда `checkout(filename, OPTION1, VALUE1, OPTION2, VALUE2.....)` извлекает файлы из системы контроля версий. `OPTION` могут быть `lock` – аналогично `checkin` – и `ревизия`, то есть указание конкретной версии файла. Команда `undocheckout(filename)` отменяет действие `checkout`, например замыкание файлов.

11.10. Профилирование программ в MATLAB 7

11.10.1. Утилита профилирования программ *Profiler* и ее запуск

В MATLAB 7 профилированию программ уделено еще большее внимание. В состав системы включена удобная в работе и мощная по своим возможностям утилита *Profiler*, запуск которой возможен рядом способов:

- из позиции **Desktop** меню – установкой птички на команде `profiler`;
- из окна командного режима исполнением команды `profile viewer`;
- из окна редактора m-файлов исполнением команды `Open Profiler` в позиции **Tools** меню.

При запуске утилиты *Profiler* открывается ее начальное окно, показанное на рис. 11.6. Поначалу это окно содержит лишь краткую информацию по утилите и

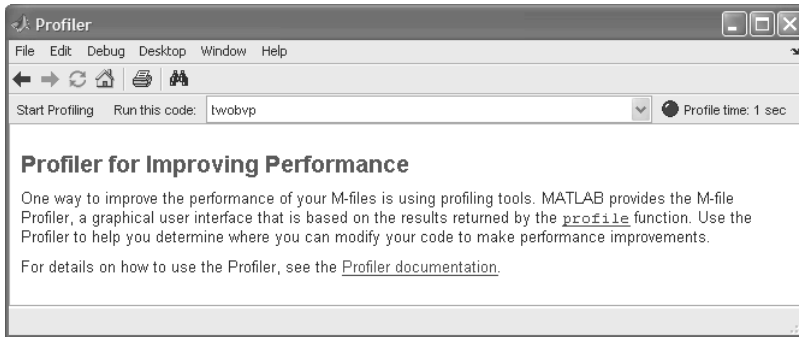


Рис. 11.6. Начальное окно утилиты профилирования Profiler

гиперссылки по применению связанной с ней функции `profile` и вызову документации из справки. Поскольку ниже рассматриваются лишь основы работы с данной утилитой, рекомендуется более детальное знакомство с ее возможностями по справке.

11.10.2. Пример профилирования программы

Для проверки программы с помощью утилиты Profiler достаточно ввести ее имя в поле над окном вывода утилиты. После этого, активизировав кнопку **Start Profiling**, можно запустить процедуру профилирования данной программы. Это сопровождается исполнением программы и подготовкой отчета по профилированию. На рис. 11.7 это демонстрируется на примере программы решения и сравнения двух систем дифференциальных уравнений `twobvp` с двухсторонними граничными условиями.

По окончании профилирования в окне профилографа появляется отчет, начало которого представлено на рис. 11.7. Он содержит несколько столбцов:

- Function name – имя исполняемой функции;
- Calls – число вызовов функции;
- Total Time – общее время исполнения, включающее время исполнения вызываемых данной функцией других функций;
- Self Time – собственное время исполнения (за исключением времени работы вызываемых функций);
- Total Time Plot (dark band = self time) – графическое представление общего времени и времени self time (темным цветом).

Данные этих столбцов дают детальную информацию о временах исполнения тех или иных функций, входящих в анализируемую программу. Это позволяет оценивать ее эффективность.

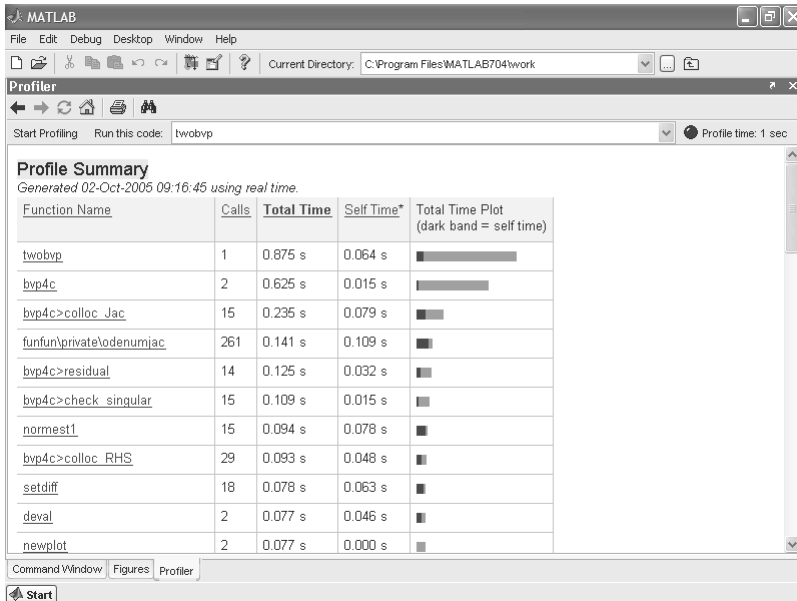


Рис. 11.7. Начало отчета по профилированию программы twobvp

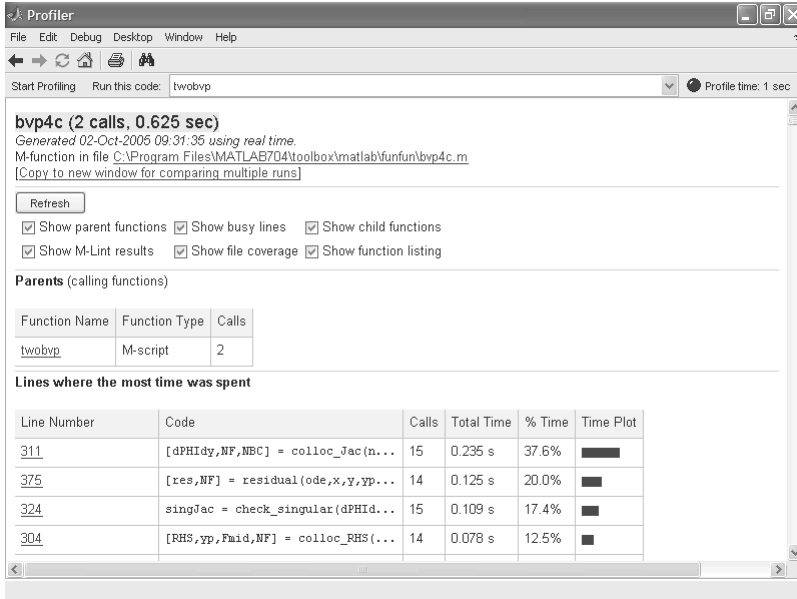
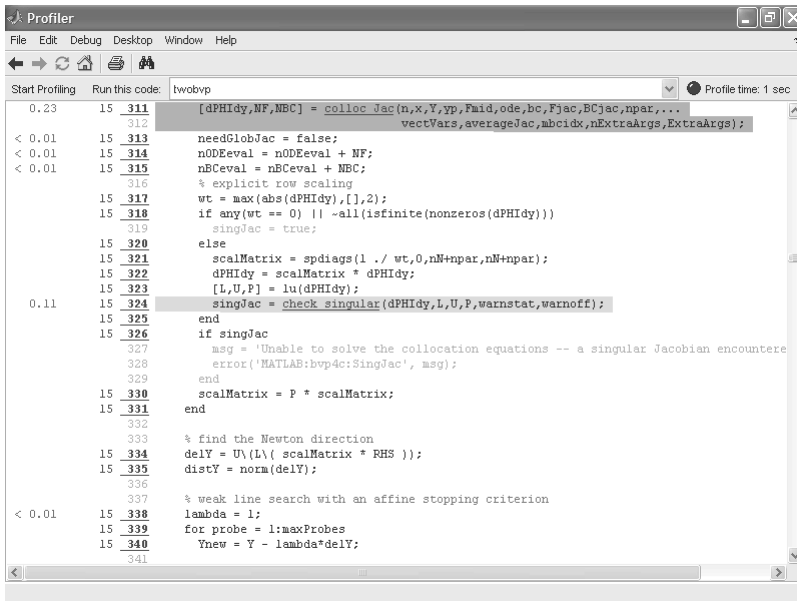
11.10.3. Профилерование избранных функций программы

Каждая входящая в программу функция в первом столбце представлена своим именем в виде гиперссылки. Активизируя ее гиперссылку, можно перейти к детальному анализу эффективности исполнения соответствующей функции. На рис. 11.8, к примеру, представлено начало такого анализа для функции `bvp4c` (вызов решателя дифференциальных уравнений).

Отчет по функции содержит детальные данные по коду, на котором основана работа той или иной функции с учетом вызова других функций. При этом в первом столбце отображаются номера строк программного кода, а далее фрагмент линстинга программного кода, число выполнений фрагмента, относительное время исполнения (в процентах) и его графическое представление.

11.10.4. Профилерование строк общего программного кода

Номера строк представлены также гиперссылками. Активизировав номер той или иной строки, можно получить детальные данные по ее выполнению. На рис. 11.9 показан пример для строки 311 и последующих строк общего кода. Тут уместно

Рис. 11.8. Отчет по функции *bvp4c*Рис. 11.9. Отчет по общим строкам программы *twobvp*, начиная со строки 311

отметить, что номера строк не совпадают с номерами строк самой программы, поскольку они даются с учетом вызовов других команд и функций.

Достаточно полный разбор данных по исполнению программы требует серьезного знания программирования, причем не только на языке MATLAB, но и на языках C, Java и др., которые включены в коды программы. Такой разбор выходит за рамки данного учебного курса. Он требует не столько обучения, сколько многолетнего опыта создания различных программ в среде MATLAB.

Интересно отметить, что в ходе исполнения той или иной программы утилита профилирования может вскрыть различные скрытые ошибки некатастрофического характера, а также выдать сообщения об ошибках даже там, где их нет. Стоит отметить, что подобные утилиты контроля программ никогда не бывают идеально точными, что ничуть не снижает их ценности. Остается отметить, что к удобству работы с утилитой профилирования относится цветная раскраска отчетов, облегчающая их анализ.

11.11. Общение MATLAB с операционной системой

11.11.1. Работа с папками

Общение системы MATLAB с операционной системой MS-DOS многим покажется рудиментарной возможностью. Так, во время написания данной книги подобное общение вообще не потребовалось. Однако это очень важно для систем, работающих в реальном масштабе времени, причем наличие наряду с командами, вводимыми знаком «!», возможности явного задания ОС (dos, unix, vms) позволяет программировать для ОС на управляющем компьютере, отличающемся от ОС пользователя MATLAB.

Так что, как говорится, из песни слов не выкинешь – MATLAB позволяет из командой строки пользоваться основными услугами старушки MS-DOS и Windows. Есть возможность общения и с другими операционными системами и даже с глобальной сетью Интернет, в том числе и с помощью собственного HTML-браузера MATLAB (браузера помощи).

Для перехода в новую папку служит команда `cd`:

- `cd wd` – переход в указанную папку `wd`;
- `cd` (или произвольное имя переменной `ad ad=cd`) – возврат строки с полным именем текущей папки;
- `cd . .` – переход к папке, родительской по отношению к текущей.

Примеры (предполагается, что MATLAB установлен на диске **E**):

```
>> cd
C:\MATLAB701\work
>> cd C:\MATLAB701\tool
?? Error using ==> cd
Cannot CD to C:\MATLAB701\tool (Name is nonexistent or not a
```



```
directory).
>> cd C:\MATLAB701\toolbox
>> cd
C:\MATLAB701\toolbox
```

Для указания пути к текущей папке может использоваться функция `pwd`:

```
>> pwd
ans =
C:\MATLAB701\toolbox
```

Для получения информации о содержимом текущей папки используется команда `dir`. Она выводит довольно длинный список имен папок.

11.11.2. Выполнение команд `!`, `dos`, `unix` и `vms`

Из командной строки MATLAB возможно выполнение команд наиболее распространенных операционных систем:

- `!` команда – выполнение заданной команды из набора операционной системы, в среде которой установлена система MATLAB;
- `dos` команда – выполнение заданной команды из набора команд MS-DOS или установленной ОС семейства Windows, в последнем случае команда выполняется в фоновом режиме.

Выведем блокнот Windows для редактирования `m`-файла.

```
dos 'notepad myfile.m'
```

или

```
[s w]=dos('notepad myfile.m')
```

`s=0`, когда команда выполнена успешно, в противном случае `s=1`, `w` содержит сообщение DOS.

- `unix` команда – выполнение заданной команды из операционной системы UNIX или UNIX-подобных систем (версии Linux);
- `vms` команда – выполнение заданной команды из операционной системы VMS (Open VMS).

11.11.3. Общение с Интернетом из командной строки

В последние годы резко возросло значение новых информационных технологий и Интернета [67, 68]. В частности, из Интернета можно получить файлы обновления текущей версии MATLAB, файлы различных примеров ее применения, можно получить документацию и справочные данные по функциям системы и даже квалифицированные ответы на свои вопросы к разработчикам системы.

Для общения с Интернетом служит команда `web`:

- `web` спецификация дает связь с Web-сервером.

Полезно отметить, что команда `web` с параметром `-browser` (например, `web http://www.mathworks.com -browser`) вызывает вместо браузера помощи MATLAB браузер HTML, установленный в ваших настройках операционной системы Windows как браузер по умолчанию.

Примеры применения команды `web`:

- `web http://www.mathworks.com` загружает Web-страницу **MathWorks Web** в браузер помощи (команда `support` сразу открывает страницу технической поддержки MATLAB);
- `web mailto:email_address` использует программу для отправки электронной почты, установленную по умолчанию в настройках операционной системы;
- все формы команды `web` могут использоваться в функциях. Например, функция `s = web('www.mathworks.com', '-browser')` запускает браузер Интернет операционной системы и выдает `s=0`, если браузер запущен, даже если браузер Интернет открывает страницу в автономном режиме (off-line) или не может ее найти, `s=1`, если браузер Интернет не был обнаружен, `s=2`, если браузер был обнаружен, но не был запущен.

Такой выход в Интернет иначе, чем экзотикой, назвать трудно, благо в Windows 98/Me/2000/NT4/XP есть куда более простые способы выхода в Интернет [67, 68] – например, браузер Internet Explorer и почтовый клиент Outlook Express. Отнесем эту возможность к числу приятных мелочей, которых в MATLAB очень много.

11.11.4. Некоторые другие команды

Есть еще несколько команд для общения с операционными системами:

- `delete name` – стирание файла с заданным именем `name` (имя записывается по правилам операционной системы);
- `getenv('name')` – возвращение значения переменной `'name'` среды окружения. Пример:

```
>> getenv('temp')
ans = C:\TEMP
```

Команда `tempdir` дает информацию о папке для хранения временных файлов:

```
>> tempdir
ans = C:\TEMP\
```

Еще одна команда – `computer` – используется в двух формах:

```
>> computer
ans = PCWIN
```

и

```
>> [C S]=computer
C = PCWIN
S = 2.1475e+009
```

Во втором случае, помимо сообщения о типе компьютера, выводится максимально возможное число элементов в массивах. Оно зависит от объема памяти и свойственных операционной системе ограничений.

Для установки типа терминала может использоваться еще одна команда – `terminal`. Возможные типы терминалов можно найти в справке по этой команде, выводимой командой `help terminal`. На этом рассмотрение команд прямого общения с операционными системами можно считать законченным.

11.12. Поддержка Java

11.12.1. Информация о средствах поддержки Java

При прямом использовании системы MATLAB не требуется знания средств реализации системы, в частности языков программирования C и Java. Однако роль Java усиливается от версии к версии системы MATLAB. Для проверки того, какая версия Java установлена на вашем ПК, можно воспользоваться командой (пример для MATLAB 7 SP2):

```
>> version -java
ans =
Java 1.5.0 with Sun Microsystems Inc. Java HotSpot(TM) Client VM
(mixed mode)
```

Для получения детальной информации о языке программирования Java и возможностях JVM можно обратиться к книгам по этому языку и к интернет-сайтам: www.javasoft.com и www.java.sun.com/jdk.

Однако для пользователей MATLAB большую часть необходимых сведений о поддержке Java системой MATLAB можно найти в справке. На рис. 11.10 показано окно справки по Java в разделе индексного указателя системы MATLAB 6.5. Кстати, в предшествующих реализациях этого раздела не было, что также указывает на то, что, начиная с реализации MATLAB 6.5, поддержке Java уделено существенно большее внимание.

Тем не менее надо помнить, что знание Java не нужно на уровне обычной работы с системой MATLAB. Основное назначение Java и JVM заключается в представлении средств для создания пользовательского интерфейса и обработки данных (например, массивов), представленных в Java-формате.

11.12.2. Java-объекты

Средства Java в MATLAB представлены как Java-объекты объектно-ориентированного языка программирования MATLAB. Для знакомства со всеми этими объектами можно вызвать окно Java-объектов, исполнив команду:

```
>> methodsview java.awt.MenuItem
```

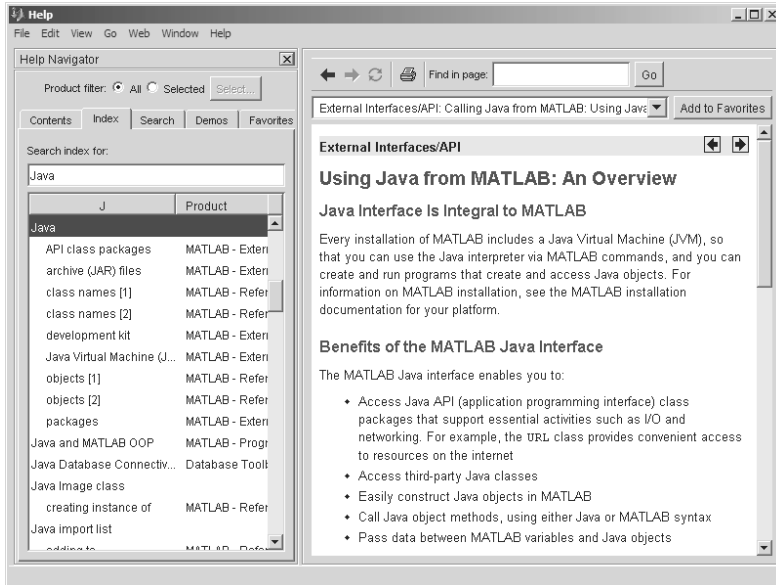


Рис. 11.10. Окно справки системы MATLAB по средствам поддержки Java

Окно со списком Java-объектов представлено на рис. 11.11. Для каждого объекта приводятся его имя, тип возвращаемого результата и форма записи аргумента.

| Qualifiers | Return Type | Name | Arguments |
|--------------|---------------------------------------|----------------------|---|
| | | MenuItem | (java.lang.String) |
| | | MenuItem | (java.lang.String, java.awt.MenuShortcut) |
| | | MenuItem | () |
| synchronized | void | addActionListener | (java.awt.event.ActionListener) |
| | void | addNotify | () |
| | void | deleteShortcut | () |
| synchronized | void | disable | () |
| | void | dispatchEvent | (java.awt.AWTEvent) |
| | void | enable | (boolean) |
| synchronized | void | enable | () |
| | boolean | equals | (java.lang.Object) |
| | javax.accessibility.AccessibleContext | getAccessibleContext | () |
| | java.lang.String | getActionCommand | () |
| | java.lang.Class | getClass | () |
| | java.awt.Font | getFont | () |
| | java.lang.String | getLabel | () |
| | java.util.EventListener[] | getListeners | (java.lang.Class) |
| | java.lang.String | getName | () |
| | java.awt.MenuContainer | getParent | () |
| | java.awt.peer.MenuComponentPeer | getPeer | () |
| | java.awt.MenuShortcut | getShortcut | () |
| | int | hashCode | () |
| | boolean | isEnabled | () |
| | void | notify | () |

Рис. 11.11. Окно Java-объектов системы MATLAB

Java-объекты в MATLAB можно создавать с использованием класса Java, например:

```
>> methodsview java.awt.MenuItem
>> f=java.awt.Frame('My Title')
f =
java.awt.Frame[frame0,0,0,0x0,invalid,hidden,layout=java.awt.
BorderLayout,resizable,title=My Title]
```

Методы объектов Java могут быть вызваны с использованием как синтаксиса Java, так и синтаксиса MATLAB. Например, в последнем случае:

```
>> setTitle(f,'new title')
>> t=getTitle(f)
t =
new title
```

Тот же пример с использованием синтаксиса Java выглядит следующим образом:

```
>> f.setTitle('modify title')
>> t=f.getTitle
t =
modify title
>> whos f
Name      Size      Bytes  Class
f         1x1       java.awt.Frame
Grand total is 1 element using 0 bytes
```

11.12.3. Специфика применения Java-объектов

Пользователь, рискнувший использовать средства Java, должен учитывать многочисленные тонкости как в форматах записи программных конструкций, так и в интерпретации их типов и результатов исполнения. Поясним это на простом примере задания строковой переменной, хранящей фразу, на языке MATLAB:

```
>> s =
Hello my friend!
>> size(s)
ans = 1      16
```

Нетрудно заметить, что строка *s* здесь интерпретируется как одномерный массив с числом элементов 16. Таким образом, каждая буква фразы трактуется как элемент массива, представленный своим кодом.

А теперь зададим строку *string* средствами Java и определим ее тип:

```
>> string=java.lang.String('Hello my friend!')
string = Hello my friend!
>> size(string)
ans = 1      1
```

Нетрудно заметить, что отдельные символы в этом случае не рассматриваются как элементы массива *string*. Однако мы все же можем выявить структуру массива:

```
>> C=char(string)
C = Hello my friend!
>> [m,n]=size(C)
m = 1
n = 16
>> whos C
  Name      Size      Bytes  Class
  C         1x16      32    char array
Grand total is 16 elements using 32 bytes
```

Итак, используя преобразование объекта класса `java.lang.string` в массив символов MATLAB, мы все же нашли число символов в строке обычными средствами MATLAB. Но, оставаясь в среде MATLAB, мы можем сделать это и средствами Java:

```
>> string.length
ans = 16
или даже так:
>> e=java.lang.StringBuffer(string)
  e = Hello my friend!
>> e.length
ans = 16
```

Следующий пример показывает задание Java-структуры многоугольника (полигона):

```
>> polygon=java.awt.Polygon([14 42 98 124],[55,12,-2,62],4)
polygon = java.awt.Polygon@9b6a46
```

Обратите внимание на дескриптор Java-объекта после знака `@`. Он не имеет жесткого значения и зависит от обстоятельств задания Java-структуры. Скорее всего, читатель, повторивший пример, получит совсем иное значение дескриптора.

Для выявления структуры Java-объекта может использоваться функция `struct(object)`, которая преобразует объект в структуру или в массив структур MATLAB с потерей информации о классе:

```
>> struct(polygon)
ans =
  npoints: 4
  xpoints: [4x1 int32]
  ypoints: [4x1 int32]
```

Следует отметить, что задание имени объекта с большой буквы (кстати, как это задано в определении класса) ведет к ошибке, поскольку сам объект, преобразуемый в структуру по правилам MATLAB (в них регистры различаются), называется `polygon`:

```
>> struct(Polygon)
??? Undefined function or variable 'Polygon'.
```

Эти примеры, и их можно привести множество, свидетельствуют о том, что при применении средств Java нужно знать этот язык программирования.

11.12.4. Java-массивы

Наиболее часто в практике программирования на языке MATLAB используются Java-массивы. Это связано как с широким применением их в справочной системе MATLAB, так и в Интернете и средствах общения с внешними программами и устройствами.

Java-массивы существенно отличаются от обычных массивов, применяемых в системе MATLAB. Прежде всего они одномерные, но возможно вложение массивов в массив без ограничения размерности. Однако для таких массивов функция `ndims` MATLAB всегда возвращает значение 2. Форма Java-массивов отличается от прямоугольной, то есть число элементов в строках (рядах) и столбцах может быть различным. В Java индексация элементов массивов идет с 0, но в MATLAB она начинается с 1 – как в обычных, так и в Java-массивах. Все это требует повышенного внимания к созданию и применению Java-массивов и делает их особыми объектами.

Приведем пример задания прямоугольного Java-массива для чисел с двойной точностью – `dblArray`:

```
>> dblArray=javaArray('java.lang.Double',3,4)
dblArray =
java.lang.Double[][]:
    []    []    []    []
    []    []    []    []
    []    []    []    []
```

Нетрудно заметить, что этот массив содержит 34 пустые ячейки. С помощью следующего фрагмента программы выполним заполнение ячеек массива:

```
>> for i=1:3
    for j=1:4
        dblArray(i, j) = java.lang.Double((i*5) + j);
    end
end
```

Этот пример демонстрирует доступ к каждой ячейке Java-массива для занесения в нее того или иного значения – в нашем случае чисел $(5*i+j)$. Теперь можно просмотреть весь массив или содержимое его отдельной ячейки:

```
>> dblArray
dblArray =
java.lang.Double[][]:
    [ 6]    [ 7]    [ 8]    [ 9]
   [11]   [12]   [13]   [14]
   [16]   [17]   [18]   [19]
>> dblArray(2,3)
ans = 13.0
```

Из сказанного можно сделать вывод, что поддержка Java представляет интерес для системных программистов, но не для обычных пользователей, использующих MATLAB по своему прямому назначению – выполнению математических расчетов.

11.13. Компиляция MATLAB-программ

11.13.1. Для чего нужна компиляция MATLAB-программ

Как уже отмечалось, язык программирования MATLAB относится к интерпретаторам, что обеспечивает диалоговый режим подготовки программ и легкую их проверку по частям – вплоть до исполнения отдельного выражения или вычисления значения отдельной функции. Пользователи интерпретаторами, например популярным по сей день Бейсиком, знают, что за это удобство приходится расплачиваться большим временем исполнения программного кода.

В этом отношении ситуация с MATLAB не так уж плоха – большая часть команд и функций его языка программирования написана на языках C и Java, и их программные коды тщательно оптимизированы и откомпилированы. Это обеспечивает малое время их исполнения и достаточно высокую скорость исполнения MATLAB-программ. Тем не менее интерпретирующий режим работы и использование ряда программных средств, написанных на интерпретаторе MATLAB, не позволяют получить предельное быстродействие при вычислениях и готовить исполняемые (exe или com) файлы, которые можно использовать без громоздкой среды MATLAB.

Однако MATLAB предусматривает возможность компиляции программ в виде m-файлов, то есть подготовки программ в виде исполняемых машинных кодов. При этом могут использоваться различные внешние компиляторы таких языков программирования, как Fortran, C и Java. Можно также использовать поставляемый с MATLAB компилятор Lcc C. Компиляция возможна только в том случае, если с MATLAB поставляется расширение MATLAB Compiler.

Возможность компиляции MATLAB-программ не стоит переоценивать. Это средство явно рассчитано на пользователя, профессионала в области программирования. Начинающим оно способно дать лишь удовлетворение от начального познания сложного процесса компиляции, который имеет множество «подводных камней» и требует хорошего знания не только системы MATLAB, но и применяемых компиляторов. Их рассмотрение выходит за рамки данной книги и выполнено лишь на начальном уровне.

11.13.2. Конфигурирование расширения MATLAB Compiler

Для конфигурирования MATLAB Compiler надо выполнить команду:

```
>> mex -setup
```

Последующий диалог выглядит следующим образом (он может различаться на разных ПК):

```
Please choose your compiler for building external interface (MEX)
```



```
files:
Would you like mex to locate installed compilers [y]/n? y
Select a compiler:
[1] Digital Visual Fortran version 6.0 in C:\Program
Files\Microsoft Visual Studio
[2] Lcc C version 2.4 in C:\MATLAB701\sys\lcc
[3] Microsoft Visual C/C++ version 6.0 in C:\Program
Files\Microsoft Visual Studio
[0] None
Compiler:
```

Итак, нужно указать на тип используемого внешнего компилятора. Например, 2 – если используется поставляемый с MATLAB компилятор Lcc C 2.4 (в сообщении указано, в какой папке находится тот или иной компилятор).

```
lease verify your choices:
Compiler: Lcc C 2.4
Location: C:\MATLAB701\sys\lcc
Are these correct?([y]/n): y
Try to update options file: C:\Documents and
Settings\Dyak\Application Data\MathWorks\MATLAB\R14\mexopts.bat
From template:
C:\MATLAB701\BIN\WIN32\mexopts\lccopts.bat
Done . . .
>>
```

11.13.3. Компиляция m-файла-функции

Файл, подлежащий компиляции, должен быть в виде m-функции. Например, можно использовать такой файл:

```
function A=cdemo(n)
A=zeros(n);
for i=1:n
    for j=1:n
        A(i,j)=1/(i^2+j^2);
    end
end
```

Можно проверить его работоспособность:

```
>> tic, A=cdemo(1000);toc
Elapsed time is 0.094000 seconds.
```

Для компиляции файла можно попытаться использовать команду

```
>> mcc cdemo
??? Unable to determine application type, since no wrapper function
was specified.
Please use the -W switch or specify application type via -m or -l.
Type 'mcc -?' for further assistance.
```

Из нее следует, что так просто компиляция не прошла и нужно использовать опцию `-m` для обычной компиляции или `-l` (латинская l – не путать с единицей).

Очень полезно просмотреть справку по компиляции, которая выводится командой `mcc -?`.

Выполнив команду

```
>> mcc -m cdemo
```

можно наблюдать успешную компиляцию:

```
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.
```

```
>>
```

Заметим, что в MATLAB 5.* / 6.* этот процесс происходит несколько иначе – надо использовать опцию в команде `mcc -x`, и успешная компиляция проходит без выдачи каких-либо сообщений.

В процессе компиляции образуются несколько файлов с именем `cdemo`, но с разными расширениями. В данном примере все они помещаются в папку `WORK` – рис. 11.12.

| Имя | Размер | Тип | Изменен |
|----------------------------|----------|---------------|------------------|
| cdemo.exe | 11 КБ | Приложение | 29.01.2005 12:02 |
| cdemo.m | 1 КБ | MATLAB M-file | 29.01.2005 7:01 |
| cdemo.ctf | 1 403 КБ | Файл "CTF" | 29.01.2005 12:02 |
| cdemo_main.c | 4 КБ | Файл "C" | 29.01.2005 12:01 |
| cdemo_mcc_component_data.c | 12 КБ | Файл "C" | 29.01.2005 12:02 |
| test1.m | 1 КБ | MATLAB M-file | 29.01.2005 5:59 |
| test2.m | | MATLAB M-file | 29.01.2005 6:00 |
| test3.m | | MATLAB M-file | 29.01.2005 6:44 |
| test4.m | | MATLAB M-file | 29.01.2005 6:43 |

Рис. 11.12. Содержимое папки `WORK`

Из созданных файлов в нашем примере самым важным представляется файл `cdemo.exe`, который является исполняемым в среде Windows и MS-DOS `exe`-файлом. Другие файлы имеют вспомогательное значение, например файлы `cdemo_main.c` и `cdemo_mcc_component_data.c` на языке C, которые можно исполнять в среде компилятора C. Эти файлы могут быть удалены. Однако файл `cdemo.m` стоит сохранить, если предполагается его модификация или перекомпиляция (она может оказаться полезной, например, для создания файла `cdemo.dll` в формате динамических библиотек Windows).

11.13.4. Исполнение откомпилированного файла

Теперь можно исполнить откомпилированный файл. Для этого используется та же команда, что и при исполнении обычного `m`-файла:

```
>> tic, A=cdemo(1000);toc
Elapsed time is 0.063000 seconds.
```

В процессе ее исполнения MATLAB отыскивает откомпилированный файл (если он есть) и исполняет его вместо исполнения m-файла. О том, что в данном случае исполнен откомпилированный файл, можно судить только по меньшему времени исполнения. Эта разница в новых версиях MATLAB 7.* /R2006*/R2007* хотя и явно заметна, но невелика. В версиях MATLAB 5.* /6.* она намного выше и достигает нескольких раз.

Визуальное программирование GUI

| | |
|---|-----|
| 12.1. Средства визуального программирования GUIDE | 592 |
| 12.2. Работа с заготовками примеров | 604 |
| 12.3. Детальная работа с инструментом GUIDE | 612 |
| 12.4. Стандартные диалоговые окна MATLAB | 642 |

Многим пользователям понравились широко применяемые в системе MATLAB приложения с графическим интерфейсом пользователя GUI (Graphic User Interface). Однако создание таких приложений на языке программирования MATLAB достаточно сложно для начинающих пользователей и даже для пользователей средней квалификации. В связи с этим в MATLAB был введен ряд средств для визуально-ориентированного программирования и проектирования приложений с GUI [28–30]. Они и описаны в данном уроке.

12.1. Средства визуального программирования GUIDE

12.1.1. Состав и назначение средств программирования GUIDE

Выше были описаны средства системы MATLAB для создания графического интерфейса пользователя GUI с помощью функций, вводимых в командной строке. Последние версии системы MATLAB приобрели специальные инструментальные средства для визуально-ориентированного программирования и проектирования приложений с GUI. Состав этих средств следующий:

- GUIDE – конструктор графического интерфейса;
- Property Inspector – инспектор свойств;
- Object Browser – браузер объекта;
- M-file Editor – редактор M-файлов;
- Component Callbacks – средство создания функций обработки событий для компонентов;
- Figure Callbacks – средства создания функций для обработки событий окон;
- Align Objects – средства выравнивания положения объектов;
- Grid and Rules – средства управления выводом сетки и линейками просмотра;
- Menu Editor – редактор меню;
- The Order Editor – средство изменения порядка активизации компонентов при нажатии клавиши **Tab**.

Эти средства дают достаточный набор средств для визуально-ориентированного программирования и проектирования GUI. Такой подход характерен для ряда современных визуально-ориентированных языков программирования, таких как Visual-BASIC, Visual-C и др., и существенно облегчает создание приложений с GUI. Это обусловлено тем, что в данном случае генерация программного кода происходит автоматически, что минимизирует усилия пользователя по программированию, а порою вообще не требует таковых.

Главным средством в визуально-ориентированном проектировании GUI является приложение GUIDE (GUI Designer). При работе с инструментом GUIDE можно создавать окна GUI путем выбора мышью нужных элементов управления и перемещения их в окно GUI. Таким образом, могут создаваться различные эле-

менты интерфейса, например кнопки, раскрывающиеся списки, линейки прокрутки и т. д. При этом возможно программирование событий, которые возникают при обращении пользователя к заданным элементам управления. Визуальное программирование совмещается с объектно-ориентированным, в частности в первом широко используется свойство наследования признаков родительских объектов их потомками.

Приложение с GUI может состоять из одного окна (основного) или нескольких окон и осуществлять вывод графической и текстовой информации как в основное окно приложения, так и в отдельные окна. MATLAB имеет ряд функций создания стандартных диалоговых окон для открытия и сохранения файлов, печати, выбора шрифта для текстовых объектов, создания окон для ввода данных и др. Эти средства (объекты) можно использовать в приложениях пользователя.

При создании приложений с GUI полезны следующие средства системы MATLAB:

- MATLAB: Creating Graphical User Interfaces;
- MATLAB: Functions – Categorical List: Creating Graphical User Interfaces;
- MATLAB: Handle Graphics Property Browser.

В справочной системе MATLAB в разделе Demos можно найти 10-минутную англоязычную демонстрацию создания приложений с графическим интерфейсом с помощью инструмента GUIDE. Однако для применения этой демонстрации на ПК нужно установить программу Macromedia Flash Player версии 5 или более поздней.

12.1.2. Открытие окна инструмента GUIDE

Для открытия окна инструмента GUIDE надо использовать команду

```
>> guide
```

При этом инструмент запускается и появляется диалоговое окно **GUIDE Quick Start** (см. рис. 12.1). Это окно имеет две вкладки:

- **Create New GUI** – создание нового приложения с GUI;
- **Open Existing GUI** – открытие уже существующего приложения с GUI.

На вкладке создания нового приложения **Create New GUI** имеется список из четырех заготовок:

- **Blank GUI** – пустое окно приложения,
- **GUI with Uicontrols** – заготовка с кнопками, переключателями и областями ввода, иллюстрирующая создание приложения с GUI вычислительного характера;
- **GUI with Axes and Menu** – заготовка с осями, меню, кнопкой и раскрывающимся списком, иллюстрирующая создание приложения с GUI графического характера;
- **Modal Question Dialog** – заготовка для модального окна, иллюстрирующего простейший диалог.

Внизу вкладки **Create New GUI** есть установка опции «**Save on startup as:**», позволяющая сразу задать имя файла, в котором будет храниться окно графиче-

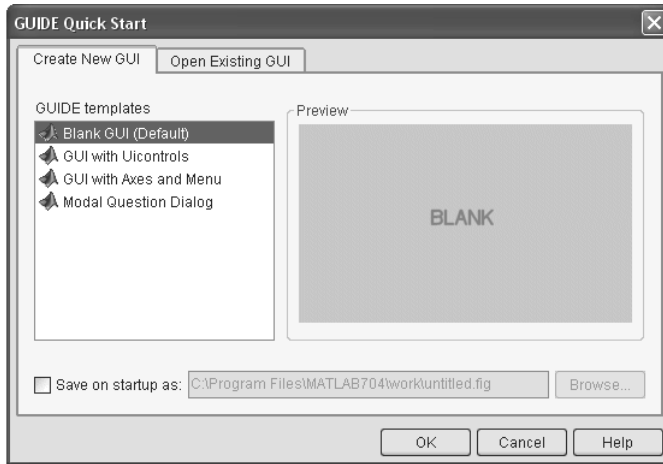


Рис. 12.1. Диалоговое окно GUIDE Quick Start

ского интерфейса при выходе из режима редактирования. Но, поскольку приложение всегда можно будет сохранить и в дальнейшем, в процессе редактирования создаваемого окна GUI, этот флаг устанавливать не обязательно.

12.1.3. Окно создания нового приложения с GUI

Выбрав мышью на вкладке **Create New GUI** заготовку **Blank GUI** и нажав клавишу **OK**, можно наблюдать инициализацию инструмента GUIDE. При этом появляется временное окошко инициализации, показанное на рис. 12.2.

По окончании инициализации появляется основное окно среды GUIDE, содержащее поле окна приложения, вертикальную панель инструментов для добавления элементов интерфейса, горизонтальную панель инструментов и обычное меню (рис. 12.3). С назначением кнопок панелей можно ознакомиться по подсказкам, которые появляются, если установить курсор мыши на нужную кнопку панели и задержать его на пару секунд. Одна из таких подсказок видна на рис. 12.3 для кнопки **Button Group**.

Окно рис. 12.3 имеет титульную строку, меню и две панели инструментов – обычную горизонтальную и вертикальную с набором объектов GUI. Горизонтальная панель инструментов с номерами кнопок показана на рис. 12.4. Эта панель позволяет работать с окном приложения с GUI, не обращаясь к его меню.

На этой панели расположены следующие кнопки:

- 1) **New** – создание нового окна с GUI;

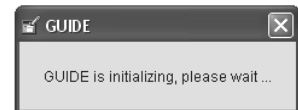


Рис. 12.2. Временное окно инициализации инструмента GUIDE

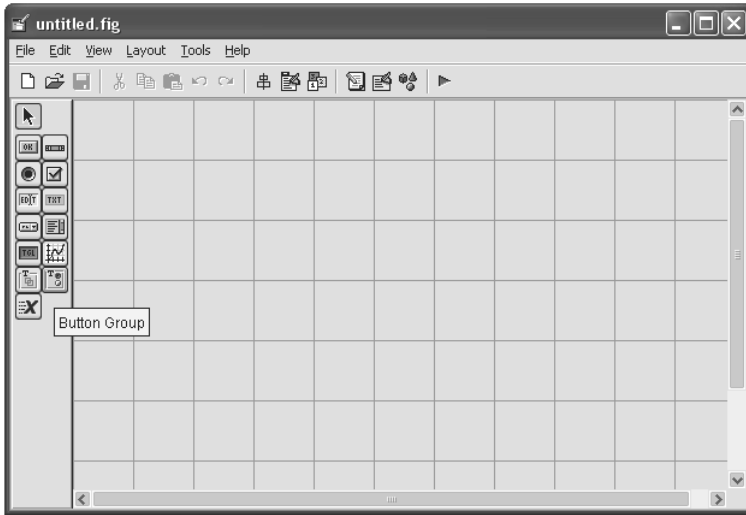


Рис. 12.3. Пустое окно приложения с GUI

Рис. 12.4. Горизонтальная панель инструментов

- 2) **Open** – открытие окна загрузки файлов с окнами GUI;
- 3) **Save** – открытие окна сохранения файла с окном GUI;
- 4) **Cut** – вырезание объекта и перенос его в буфер;
- 5) **Copy** – копирование объекта в буфер;
- 6) **Paste** – перенос объекта из буфера с сохранением его в нем;
- 7) **Undo** – отмена последней операции;
- 8) **Redo** – возврат к отмененной операции;
- 9) **Align Objects** – выравнивание объектов;
- 10) **Menu Editor** – вывод окна редактора меню;
- 11) **Tab Order Editor** – вывод окна редактора порядка обвода элементов клавишей **Tab**;
- 12) **M-file Editor** – вывод окна редактора M-файлов;
- 13) **Property Inspector** – вывод окна инспектора свойств;
- 14) **Object Browser** – вывод окна браузера объектов;
- 15) **Run** – запуск созданного приложения с GUI.



Вертикальная панель с объектами GUI и номерами кнопок показана на рис. 12.12. Она содержит следующие кнопки:

- 1) **Select** – селекция (выбор) объектов создаваемого окна GUI;
- 2) **Push Button** – обычная кнопка;
- 3) **Radio Button** – радиокнопка (беспроводная);
- 4) **Edit Text** – кнопка ввода и редактирования текста;



Рис. 12.5. Панель объектов GUI

- 5) **Pop-up Menu** – открывающийся список;
- 6) **Toggle Button** – кнопка-переключатель;
- 7) **Panel** – панель;
- 8) **ActiveX Component** – компонент ActiveX;
- 9) **Slider** – линейка прокрутки (слайдер);
- 10) **Check Box** – область задания опции (флаг);
- 11) **Static Text** – область ввода текста;
- 12) **Listbox** – список;
- 13) **Axes** – оси графика;
- 14) **Button Group** – кнопка для группы объектов.

Меню окна создания приложения с GUI (рис. 12.6) содержит следующие позиции:

- **File** – файловые операции, вызов окна **Preferences** и печать;
- **Edit** – обычные операции редактирования и работы с буфером Windows;
- **View** – просмотр деталей GUI (эта позиция на рис. 12.6 дана открытой);
- **Layout** – разметка окна;
- **Tools** – вызов инструментальных средств;
- **Help** – вызов справки.

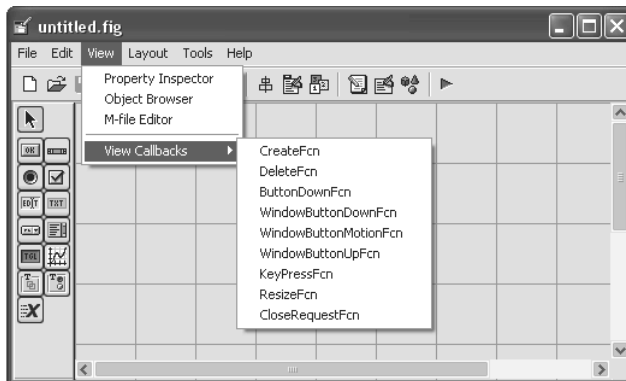


Рис. 12.6. Окно приложения с GUI с открытой позицией меню **View**

Первые две позиции меню содержат стандартные файловые операции и операции редактирования. Они будут описаны ниже в разделе 12.3. Пока же отметим, что в позиции **File** меню есть команда **Preferences...**, открывающая окно предварительной настройки окна GUI, – рис. 12.7.

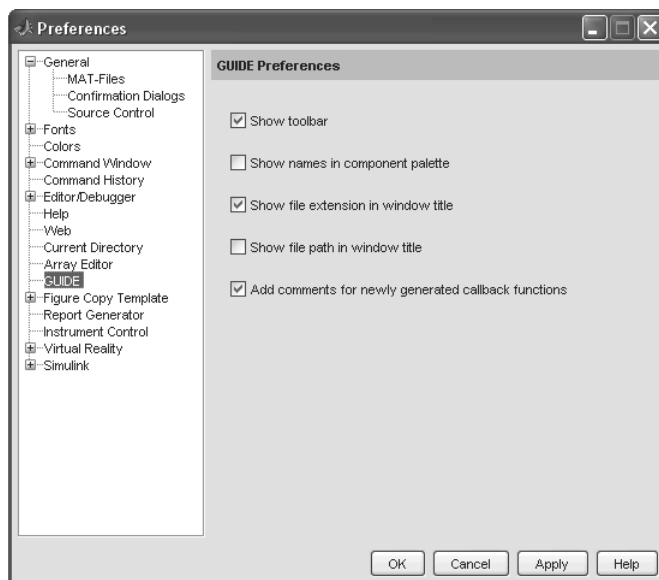


Рис. 12.7. Окно **Preferences** для инструмента **GUIDE**

Окно **Preferences** позволяет ввести следующие опции:

- **Show toolbar** – показать панель инструментов;
- **Show names is component palette** – показать имена в панели компонентов;
- **Show file extension in window title** – показать расширение файла в титульной строке окна;
- **Show path file in window title** – показать путь к файлу в титульной строке окна;
- **Add comments for newly generated callback function** – добавить комментарии в новую сгенерированную функцию обработки событий.

Следует отметить, что по сравнению с более старыми реализациями **GUIDE** в системе **MATLAB**, число опций в окне **Preferences** уменьшено. Если задать все опции, то окно нового приложения с **GUI** будет иметь вид, показанный на рис. 12.8. Обратите внимание прежде всего на действие опции **Show names is component palette** – она существенно меняет вид вертикальной панели инструментов. Инструменты в этом случае указаны со своими именами.

12.1.4. Свойства объектов **GUI**

Каждый объект (компонент) **GUI** имеет ряд свойств. Их полный набор задается таблицей из 41 свойства, представленной ниже.

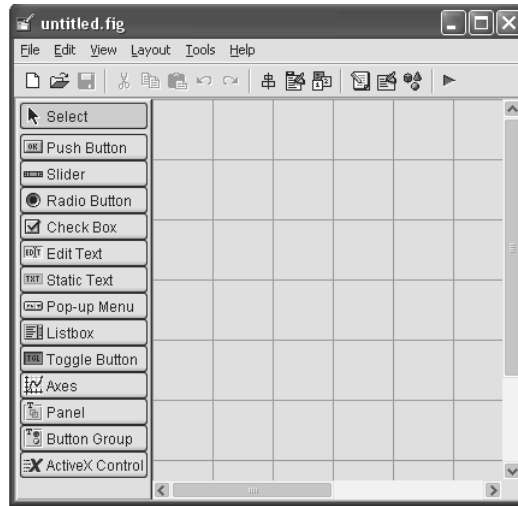


Рис. 12.8. Окно нового приложения с GUI

Таблица 12.1. Свойства объектов, их назначение и возможные значения

| Свойство | Назначение | Значения (жирным по умолчанию) |
|-----------------|---|---|
| BackgroundColor | Цвет фона (серый по умолчанию, выбор из окна выбора цветов) | Вектор [RGB] [0.753, 0.753, 0.753] |
| BeingDeleted | Признак возможности удаления | on – включено, off – выключено |
| BusyAction | Реакция на возникновение нового прерывания при обработке текущего события | queone – поставить в очередь, cancel – проигнорировать |
| ButtonDownFc | Указатель на функцию обработки нажатия правой клавиши мыши | |
| Cdata | Массив для хранения изображения, нанесенного на поверхность объекта | Тип truecolor |
| Callback | Указатель на функцию обработки события | |
| Children | Массив указателей на потомков | |
| Clipping | Признак отсекающего при выходе изображения за границы объекта | on – включено, off – выключено |
| CreateFn | Указатель на функцию обработки события «Создание объекта» | |
| DeleteFn | Указатель на функцию обработки события «Удаление объекта» | |
| Enable | Признак доступа к объекту | on – доступ есть, off – доступа нет |
| Extend | Габаритный прямоугольник | |
| FontAngle | Признак наклона букв | normal, italic, oblicue |

Таблица 12.1. Свойства объектов, их назначение и возможные значения (продолжение)

| Свойство | Назначение | Значения (жирным по умолчанию) |
|---------------------|---|---|
| FontName | Наименование шрифта | Из списка шрифтов Windows (MS Sans Serif) |
| FontSize | Размер шрифта | Список размеров шрифтов Windows (8 пунктов) |
| FontUnits | Единицы измерения шрифтов | inches, centimeters, normalized, pixels, points (пункт = 1/72 дюйма) |
| FontWeight | Толщина контура букв | normal , light, demi, bold |
| ForegroundColor | Цвет рисования | Вектор [RGB] [0.0, 0.0, 0.0] – черный |
| HandleVisibility | Признак видимости указателя обработчика событий | on – доступен, off – не доступен |
| HitTest | Признак разрешения поиска объекта по значению свойства | on – разрешен, off – не разрешен |
| HorizontalAlligment | Выравнивание надписей в поле объекта по горизонтали | left, right, center |
| Interruptible | Признак разрешения на прерывание обработчика событий | on – разрешено, off – не разрешено |
| KeyPressFcn | Указатель на функцию обработки события «Нажата кнопка» в момент, когда компонент находится в фокусе | |
| ListBoxTop | Индекс строки раскрывающегося списка, которая отображается в верхнем окне | 0 |
| Max | Максимальное значение свойства Value | 1 |
| Min | Минимальное значение свойства Value | 0 |
| Parent | Указатель на родительский объект | |
| Position | Позиция объекта в виде вектора, задающего координаты нижнего левого угла объекта, его ширину и высоту в заданных единицах | |
| Selected | Признак выбора объекта | on – выбран, off – не выбран |
| SelectionHighlight | Признак повышенной яркости для выделения элемента в объекте | on – включено, off – выключено |
| SliderStep | Вектор малых и больших перемещений ползунка слайдера | |
| String | Символьная строка, задающая надпись или значение, приписанное объекту | |
| Style | Символьная строка с типом объекта | |

Таблица 12.1. Свойства объектов, их назначение и возможные значения (продолжение)

| Свойство | Назначение | Значения (жирным по умолчанию) |
|---------------|--|--|
| Tag | Тег – символьная строка с именем объекта | |
| TooltipString | Текст всплывающей подсказки | Пустая строка |
| Type | Класс объекта | |
| UIContextMenu | Всплывающее меню объекта | |
| Units | Единицы измерения линейных величин (зависят от типа объекта) | inches, centimeters, normalized, pixels, points (пункт = 1/72 дюйма) |
| UserData | Массив данных, ассоциированный с объектом | [] – пустой массив |
| Value | Значение, характерное для данного объекта | |
| Visible | Признак видимости объекта | on – виден, off – не виден |

Этот набор свойств присущ не обязательно каждому объекту. Некоторые из свойств данного объекта могут быть для него не имеющими смысла. Для задания свойств объектов служит Инспектор свойств, работа с которым описана ниже. Именно правильный выбор свойств всех компонентов создаваемого приложения с GUI лежит в основе успеха визуально-ориентированного проектирования приложений с GUI.

12.1.5. Пример задания кнопки и работа с инспектором свойств объектов

Создание нового окна с GUI сводится к переносу объектов из панели объектов рис. 12.5 в окно GUI будущего окна с приложением пользователя. Для этого достаточно выбрать нужные объекты и с помощью мыши перенести их в поле окна с приложением пользователя.

В ограниченном объеме данной книги невозможно описать все нюансы применения всех компонентов приложений с окнами GUI. В этом нет и особого смысла – достаточно рассмотреть несколько примеров для понимания сути визуально-ориентированного проектирования окон приложений с GUI. Начнем с простого примера – добавления кнопки в заготовку окна приложения. Для этого при помощи мыши выберите кнопку **Push Button** (ее пиктограмма содержит кнопку **OK**, а имя появляется на всплывающей подсказке) и щелчком мыши поместите кнопку на заготовку окна приложения (см. рис. 12.9).

После добавления элемента интерфейса необходимо задать его имя (тег), который будет идентифицировать данный объект среди всех остальных объектов. Тег объекта понадобится для получения и установки его свойств и программирования событий, возникающих при обращении пользователя к элементу управления, например при нажатии на кнопку.

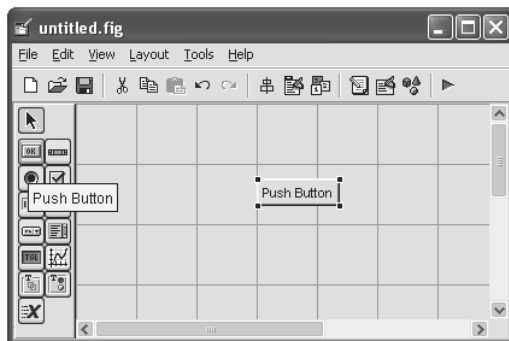


Рис. 12.9. Добавление кнопки на заготовку окна приложения

Для задания тега следует перейти к инспектору свойств. Проще всего это сделать двойным щелчком мыши по добавленной кнопке. При этом появляется окно инспектора свойств (Property Inspector), в котором отображены свойства кнопки (объекта `Uicontrol`) – рис. 12.10. Найдите в левом столбце таблицы свойство `Tag` и в области ввода справа от него измените имя по умолчанию `pushbutton1` на нужное имя и нажмите клавишу `Enter`. Всегда следует давать объектам содержательные теги.

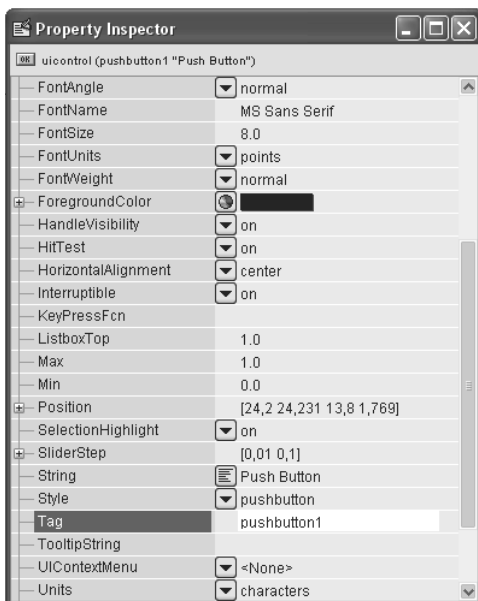


Рис. 12.10. Окно инспектора свойств кнопки

Аналогичным образом задаются другие параметры кнопки. Например, для изменения надписи на кнопке достаточно изменить значение переменной String. Многие опции и параметры в окне инспектора свойств имеют вполне очевидные названия и представлены в разделе 12.1.3. С назначением свойств легко экспериментировать, и опытный пользователь, взявшийся за создание своих приложений с GUI, получит от таких экспериментов вполне понятное удовлетворение. Следует, однако, еще раз отметить, что многие свойства могут относиться не прямо к выбранному объекту (кнопке в нашем случае), а к иным объектам – потомкам выбранного объекта.

Созданная описанным образом кнопка пока не действует. Чтобы кнопка выполняла какие-либо действия, она должна быть связана с программой обработки событий. Для каждого объекта или совокупности объектов в окне приложения с GUI такая программа, именуемая Callback, создается приложением GUIDE автоматически.

12.1.6. Вид всех компонентов и редактирование их свойств

В заключение общего рассмотрения окна создания нового GUI приведем рис. 12.11, на котором выведены все объекты GUI, кроме ActiveX. Попытка вывода объекта ActiveX приведет к выводу окна с обширным списком объектов ActiveX, которые являются действующими объектами, например календарем, часами и т. д.

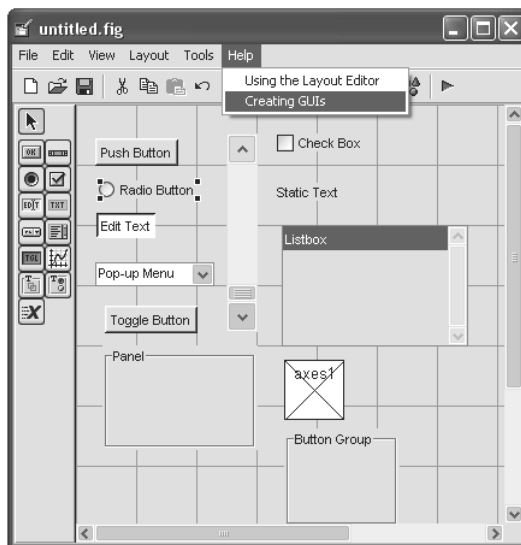


Рис. 12.11. Пример вывода всех объектов GUI, кроме ActiveX

Как видно по рис. 12.11, возможно задание практически всех типовых элементов интерфейса. С каждым элементом (компонентом) интерфейса связан свой фрагмент программы обработки событий, которая генерируется автоматически и может корректироваться пользователем. Как это делается, мы рассмотрим на ряде примеров – сначала встроенных в это приложение.

На рис. 12.12 представлено окно выбора компонентов ActiveX. Оно содержит большой список компонентов, созданных самыми различными приложениями, установленными на данном компьютере. Вид некоторых компонентов просматривается в окне просмотра этого окна.

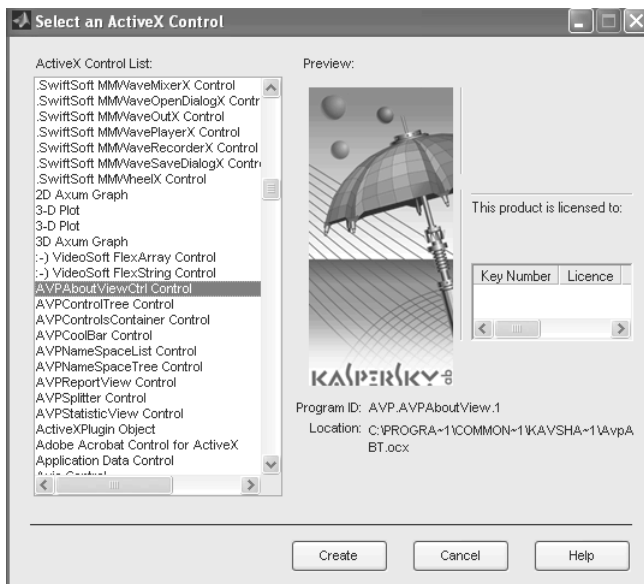


Рис. 12.12. Окно выбора компонентов класса ActiveX

Для вызова редактора свойств со свойствами каждого конкретного компонента достаточно дважды быстро щелкнуть мышью по изображению компонента. При этом появляется окно инспектора свойств данного компонента, в котором будут представлены только его свойства. Это иллюстрирует рис. 12.13, на котором показаны вывод компонента ActiveX в виде прямоугольника со значком X и окно его свойств. Нетрудно заметить, что этот компонент характеризуется всего несколькими свойствами.

Тут уместно отметить, что каждое окно приложения с GUI описывается двумя файлами с расширениями .fig и .m. Первый файл описывает фигуру – окно со всеми ее деталями, а второй – программу обработки событий. Последний файл создается и редактируется в редакторе M-файлов системы MATLAB.

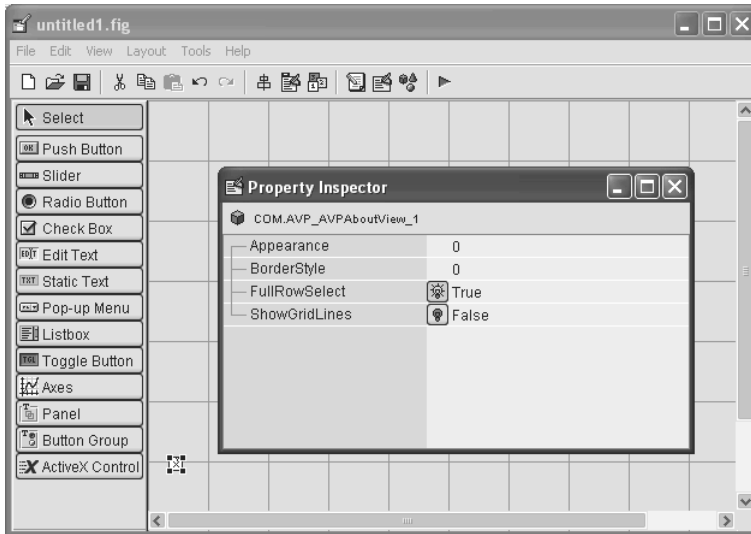


Рис. 12.13. Вывод компонента ActiveX и окна его свойств

12.2. Работа с заготовками примеров

12.2.1. Простой пример вычисления массы вещества

Рассмотрим пример GUI with Uicontrols (рис. 12.14), представляющий окно приложения с GUI, позволяющего вводить плотность и объем вещества, а также систему единиц для измерения массы, вычисляемой как произведение плотности вещества на его объем. Пример предусматривает также применение двух кнопок для вычисления массы Calculate и сброса исходных данных. В окне рис. 12.14 показан вид макета создаваемого приложения.

Обратите внимание на установку внизу вкладки **Create New GUI** опции «**Save on startup as:**» и изменение предложенного имени файла на demo1.fig. Если теперь нажать кнопку **OK**, то произойдет автоматическая подготовка программы demo1.fig, строящей заданное окно, и программы demo1.m обработки событий. На экране дисплея появится окно редактора М-файлов с программой обработки событий и окно **demo1.fig** с заготовкой окна приложения с GUI – рис. 12.15.

Приведенная на рис. 12.15 заготовка является уже готовой. Однако читатель может сам легко подготовить ее, как это было описано для примера размещения кнопки. Следует также отметить, что если приложение с GUI создается в новой директории, то MATLAB выведет окно с предложением изменения текущей директории. Если согласиться с ним, то файлы готовящегося приложения будут размещены в заданной директории.

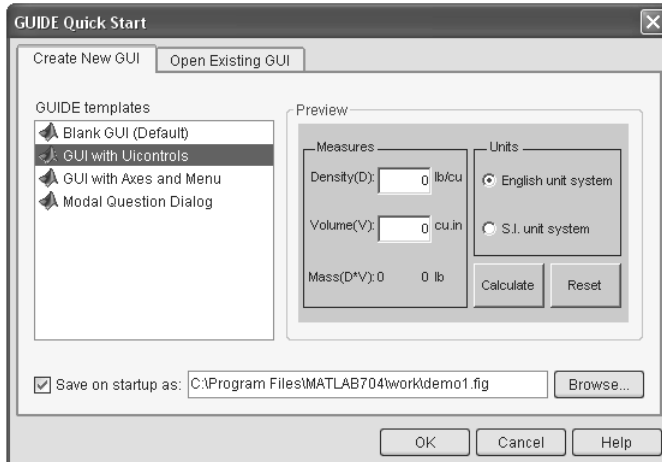


Рис. 12.14. Начало проектирования приложения с GUI для вычисления массы вещества

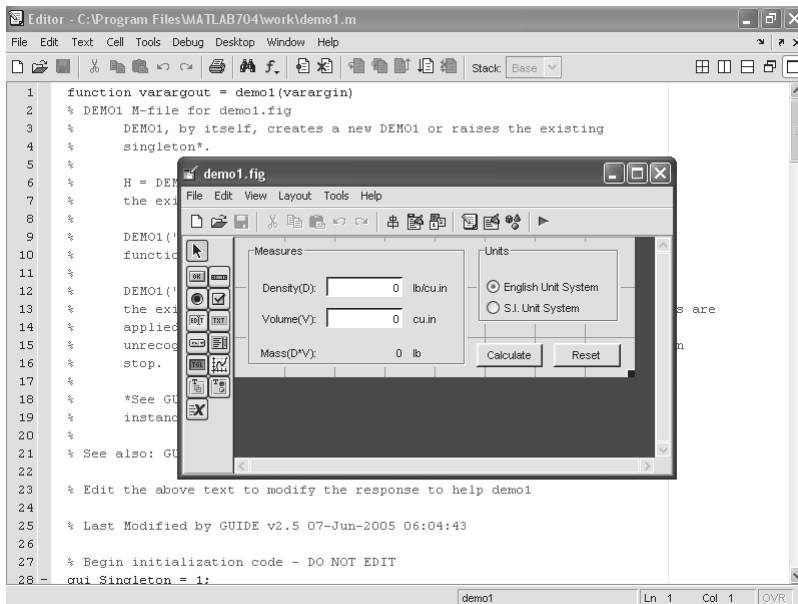


Рис. 12.15. Окна редактора M-файлов с программой обработки событий и окно **demo1.fig** с заготовкой окна приложения с GUI

Закрыв окно **demo1.fig**, можно увидеть программу в окне редактора M-файлов – рис. 12.16. Программа, даже для этого простого примера, получается довольно большой, главным образом из-за многочисленных текстовых комментариев на

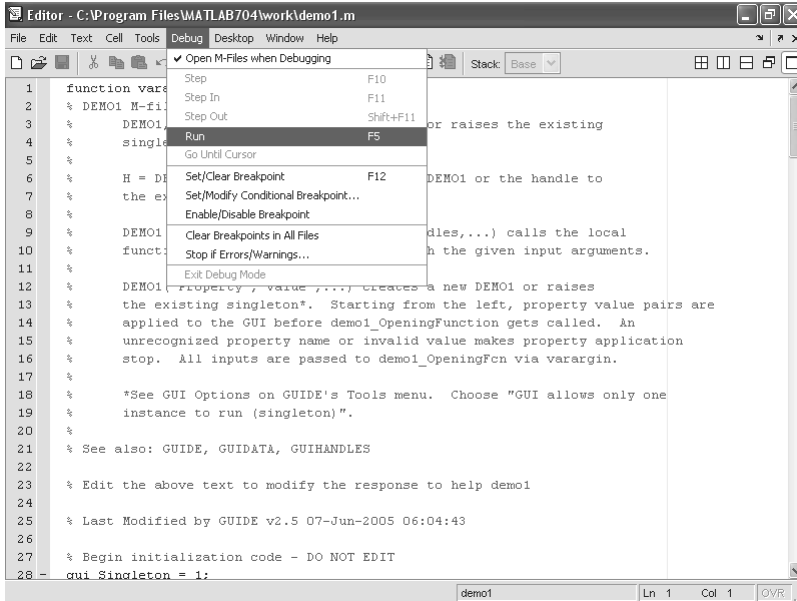


Рис. 12.16. Сгенерированная программа в окне редактора М-файлов

английском языке. Они начинаются со знаков %. Замена комментариев на русскоязычный не рекомендуется, поскольку не локализованные под Россию версии системы MATLAB нередко дают сбои при такой замене, и нормальная работа программ с подобными комментариями не гарантируется.

Взяв за основу эту программу, удалим из нее англоязычные комментарии (эта процедура делается только ради сокращения размера листинга программы, вообще же говоря, комментарии полезны, и удалять их не стоит). Листинг полученной программы можно вывести из редактора М-файлов или по команде `type`:

```
>> type demo1
function varargout = demo1(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @demo1_OpeningFcn, ...
                  'gui_OutputFcn',  @demo1_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end

function demol_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
initialize_gui(hObject, handles, false);

function varargout = demol_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function density_CreateFcn(hObject, eventdata, handles)
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,
        'defaultUicontrolBackgroundColor'));
end

function density_Callback(hObject, eventdata, handles)
density = str2double(get(hObject, 'String'));
if isnan(density)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
handles.metricdata.density = density;
guidata(hObject,handles)

function volume_CreateFcn(hObject, eventdata, handles)
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,
        'defaultUicontrolBackgroundColor'));
end

function volume_Callback(hObject, eventdata, handles)
volume = str2double(get(hObject, 'String'));
if isnan(volume)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

handles.metricdata.volume = volume;
guidata(hObject,handles)

function calculate_Callback(hObject, eventdata, handles)
mass = handles.metricdata.density * handles.metricdata.volume;
```

```

set(handles.mass, 'String', mass);
function reset_Callback(hObject, eventdata, handles)
initialize_gui(gcf, handles, true);
function unitgroup_SelectionChangeFcn(hObject, eventdata, handles)
if (hObject == handles.english)
    set(handles.text4, 'String', 'lb/cu.in');
    set(handles.text5, 'String', 'cu.in');
    set(handles.text6, 'String', 'lb');
else
    set(handles.text4, 'String', 'kg/cu.m');
    set(handles.text5, 'String', 'cu.m');
    set(handles.text6, 'String', 'kg');
end

function initialize_gui(fig_handle, handles, isreset)
if isfield(handles, 'metricdata') && ~isreset
    return;
end

handles.metricdata.density = 0;
handles.metricdata.volume = 0;
set(handles.density, 'String', handles.metricdata.density);
set(handles.volume, 'String', handles.metricdata.volume);
set(handles.mass, 'String', 0);
set(handles.unitgroup, 'SelectedObject', handles.english);
set(handles.text4, 'String', 'lb/cu.in');
set(handles.text5, 'String', 'cu.in');
set(handles.text6, 'String', 'lb');
guidata(handles.figure1, handles);

```

Заинтересованный читатель может легко разобраться в деталях этой программы. Для этого полезны убранные комментарии – даже англоязычные. Но и без них программа даже столь простого примера оказывается довольно большой. Следует отметить, что созданную программу можно запустить как указанием ее имени в командной строке MATLAB:

```
>> demo1
```

так и командой **Run** из окна редактора М-файлов (см. рис. 12.16). При запуске появляется окно GUI данной программы, показанное на рис. 12.17. Там же показан реальный пример вычисления массы вещества по заданным его плотности и объему. Для сброса данных используется кнопка **Reset**, а для вычислений по новым введенным данным – кнопка **Calculate**.

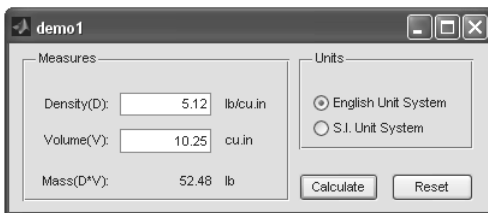


Рис. 12.17. Окно программы *demo1.m*, вычисляющей массу вещества по заданным его плотности и объему

Этот пример хорошо иллюстрирует создание простого приложения с GUI чисто вычислительного характера. Рекомендуется повторить его, создав окно приложения с GUI самостоятельно.

12.2.2. Пример на построение графиков из списка

Следующий пример из заготовок иллюстрирует создание приложения с графикой в окне GUI. Он осуществляет задание списка графиков и построение одного из выбранных из этого списка графиков нажатием кнопки **Update** – рис. 12.18.

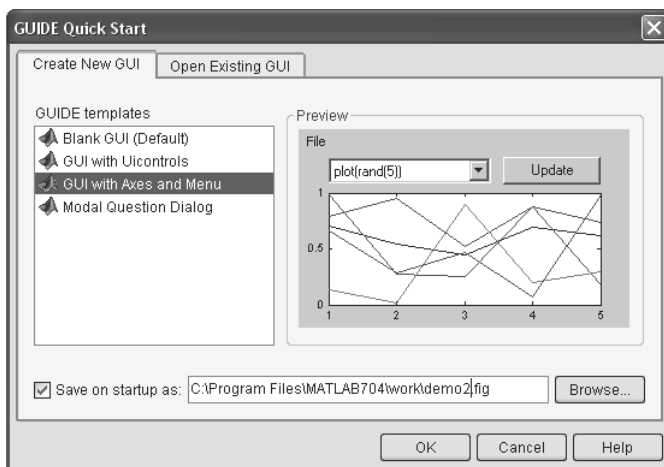


Рис. 12.18. Открытие примера GUI with Axes and Menu

Задав имя файла demo2.fig и установив опцию записи примера в файл, можно запустить генерацию программы (как в предыдущем примере) и наблюдать ее в окне редактора М-файлов. Это показано на рис. 12.19. В окне редактора М-файлов представлены сгенерированная программа на языке MATLAB и окно GUI для данного примера. В нем видны список графиков, кнопка **Update** и созданное окно для будущего графика axes1 (оно помечено перекрестием).

Как и ранее, взяв за основу сгенерированную программу и убрав из нее комментарии, следует записать программу. Ее листинг представлен ниже:

```
>> type demo2
function varargout = demo2(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',  @demo2_OpeningFcn, ...
                  'gui_OutputFcn',  @demo2_OutputFcn, ...
```

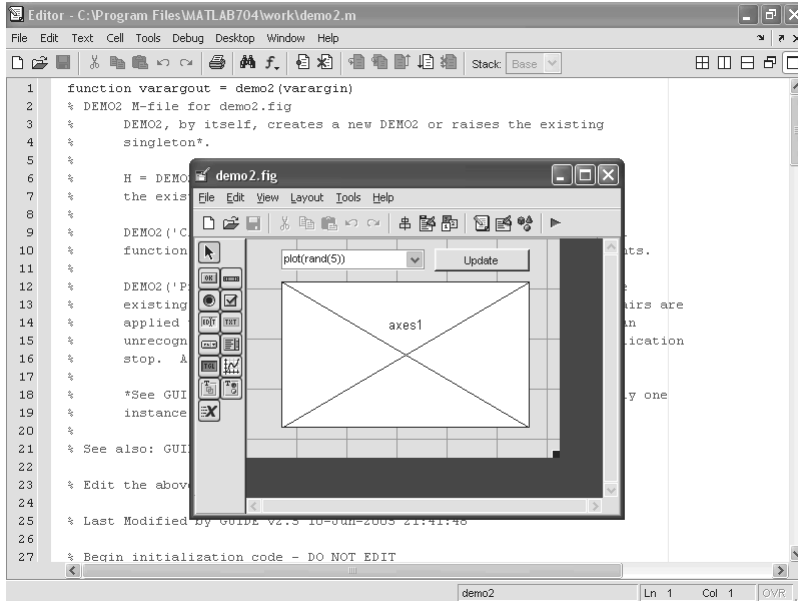


Рис. 12.19. Генерация М-файла и его листинг в окне редактора М-файлов

```

        'gui_LayoutFcn', [] , ...
        'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function demo2_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
if strcmp(get(hObject,'Visible'),'off')
    plot(rand(5));
end

function varargout = demo2_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function pushbutton1_Callback(hObject, eventdata, handles)
axes(handles.axes1);
cla;

```

```

popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        plot(rand(5));
    case 2
        plot(sin(1:0.01:212.99));
    case 3
        bar(1:.5:10);
    case 4
        plot(membrane);
    case 5
        surf(peaks);
end

function FileMenu_Callback(hObject, eventdata, handles)

function OpenMenuItem_Callback(hObject, eventdata, handles)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

function PrintMenuItem_Callback(hObject, eventdata, handles)
printdlg(handles.figure1)

function CloseMenuItem_Callback(hObject, eventdata, handles)
selection = questdlg(['Close ' get(handles.figure1, 'Name') '?'], ...
                    ['Close ' get(handles.figure1, 'Name')
                    '...'], ...
                    'Yes', 'No', 'Yes');
if strcmp(selection, 'No')
    return;
end
delete(handles.figure1)

function popupmenu1_Callback(hObject, eventdata, handles)

function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject, 'BackgroundColor', 'white');
else
    set(hObject, 'BackgroundColor', get(0, ...
        'defaultUicontrolBackgroundColor'));
end

set(hObject, 'String', {'plot(rand(5))', 'plot(sin(1:0.01:25))', ...
    'bar(1:.5:10)', 'plot(membrane)', 'surf(peaks)'});

```

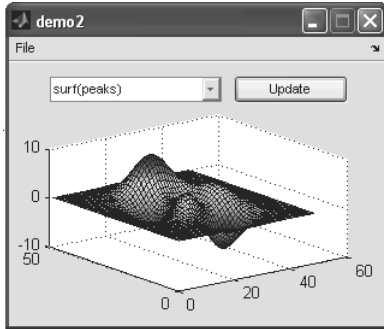



Рис. 12.20. Окно программы с GUI для построения графика из списка графиков

В этой программе несложно разобраться, и ее можно взять в качестве основы для подобных своих программ. Запустив программу из редактора М-файлов или из командной строки, можно наблюдать ее окно, представленное на рис. 12.20 при выборе последнего графика из списка, – построение поверхности *peaks*.

Третий простой пример на закрытие окна с помощью диалогового окна с двумя кнопками читатель может разобрать и опробовать самостоятельно. Заметим, что эти примеры весьма подробно описаны в справке, но для знакомства с ними нужно некоторое знание английского языка.

12.3. Детальная работа с инструментом GUIDE

12.3.1. Установка опций окна компонентов

Работа по созданию нового приложения с GUI обычно начинается с вывода окна Blank GUI инструмента GUIDE – рис. 12.3. Рекомендуется представить его в виде, показанном на рис. 12.21 с вертикальной инструментальной панелью, содержащей имена компонентов. Для этого в окне **Preferences** (рис. 12.7) следует поставить птичку (флажок) у опции **Show name in component palette**. Для вывода этого окна используется команда **Preferences...** в позиции **File** меню.

Прежде всего отметим, что в окне нового приложения можно использовать контекстное меню правой клавиши мыши – на рис. 12.21 оно показано для пока пустого окна приложения. В этом меню имеется подборка команд, доступных для того или иного объекта – в нашем случае окна нового приложения.

Рекомендуется сразу уточнить действие некоторых глобальных опций, от активности которых зависят выполняемые при конструировании приложений с GUI действия. Для вывода окна опций можно исполнить команду **GUI Options...** в контекстном меню правой клавиши мыши или в позиции **Tools** меню окна нового приложения. Появится окно **GUI Options**, показанное на рис. 12.22 внутри окна нового приложения.

Это окно имеет два раскрывающихся списка и набор опций, задаваемых флажками. Список **Resize behavior (Размеры)** окна служит для выбора поведения окна при попытке пользователя изменить его размер. Возможны следующие варианты:

- **Non-resizable** – нельзя менять размеры;
- **Proportional** – возможно изменение размеров с сохранением пропорциональности;

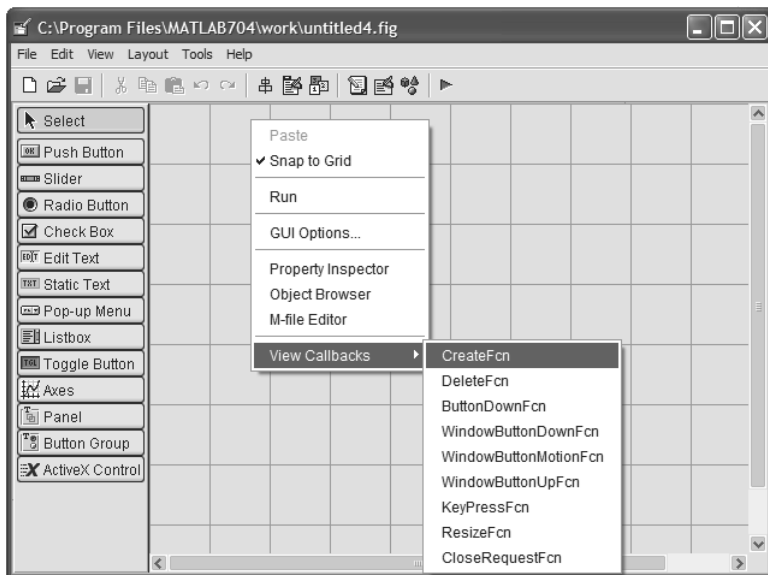


Рис. 12.21. Окно создаваемого приложения с GUI с контекстным меню правой клавиши мыши

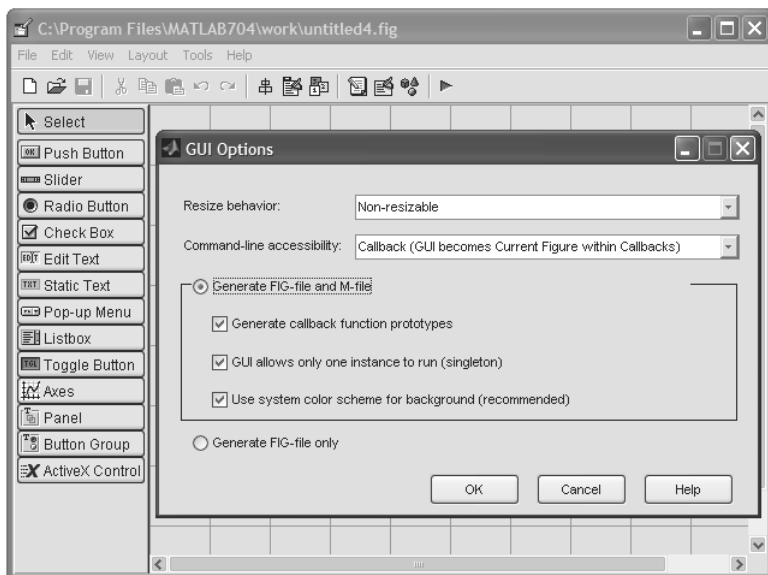


Рис. 12.22. Окно GUI Options внутри окна нового приложения

- **User-specified** – допускается изменение окна пользователем, но без переконфигурации компонентов.

Другой список **Command-line accessibility (Доступ к командной строке)** задает разграничение доступа на текущую фигуру gcf между функциями GUI и командной строкой. В этом списке заданы следующие варианты:

- **Callback** – функция обработки событий получает указатель на текущее событие;
- **Off** – навсегда отключает указатель на текущее событие;
- **On** – всегда получает указатель на текущее событие из командной строки;
- **Other** – задает установки, назначенные в инспекторе свойств Property Inspector.

Далее в описанном окне имеются опции, относящиеся к созданию файлов.

Внизу окна имеется опция

- **Generate FIG-file only** – генерация только файлов с расширением FIG.

Если она задействована (жирная точка в кружке), то остальные опции будут недоступны и генерироваться будет только FIG-файл окна приложения с GUI. Если же опция не задействована, то будет включена опция

- **Generate FIG-file and M-file** – генерация FIG-файла и M-файла.

Она, в свою очередь, открывает доступ к трем очевидным опциям:

- **Generate callback function prototype** – генерировать прототипы callback функций обработки событий;
- **GUI allows only on instance to run (singleton)** – запускать только одно приложение с GUI;
- **Use system color scheme for background (recommended)** – использовать для фона цветовую схему системы (рекомендуется).

12.3.2. Работа с меню File

В позиции **File** меню инструмента GUIDE расположены средства для работы с файлами. В связи с известностью этих средств ограничимся перечислением позиций этого меню вместе с их назначением:

- **New** – открытие нового окна приложения с GUI;
- **Open** – открытие окна загрузки ранее созданного окна приложения;
- **Close** – закрытие текущего окна приложения;
- **Save** – запись создаваемого приложения с его именем;
- **Save As** – вызов окна записи создаваемого приложения с изменением имени;
- **Export...** – вызов окна экспорта создаваемого приложения;
- **Preferences ...** – вызов окна свойств GUIDE;
- **Print...** – вызов окна печати окна создаваемого приложения с GUI.

Стоит отметить, что при печати печатается окно GUI-интерфейса с рамкой и со всеми введенными в него объектами (компонентами).

12.3.3. Ввод компонентов и их редактирование

Как уже отмечалось, ввод компонентов в окно приложения с GUI осуществляется вначале выбором компонента помещением на пиктограмму компонента курсора мыши и нажатием на левую клавишу мыши. Затем надо поместить курсор мыши в нужное место окна приложения и быстро снова нажать на левую клавишу мыши. Если включен действующий по умолчанию режим **Dnap to Grid** в позиции **Layout** меню, то при приближении изображения компонента к сетке изображение захватывается ближайшей линией сетки и располагается точно по ней. Это облегчает создание геометрически точного расположения объектов.

Изображение компонентов можно перемещать по полю окна компонентов и масштабировать – менять в размерах. Для этого надо выделить объект, поместив на его изображение курсор мыши. Он превращается в крестик из стрелок. Нажав левую кнопку мыши, можно выделить объект – он помещается в прямоугольную рамку с черными квадратиками по углам (рис. 12.23). Если не отпустить левую клавишу мыши и начать ее перемещать по столу (коврику), то объект начнет перемещаться.

Если после выделения изображения компонента зацепиться курсором мыши за один из уголков выделения, то при перемещении мыши с нажатой левой клавишей можно наблюдать изменение размера области компонента в виде рамки из тонких черных линий, уменьшающейся или увеличивающейся в размерах (рис. 12.24).

Отпустив левую кнопку, можно наблюдать объект с измененными размерами – рис. 12.25 (в данном случае объект растянут по обеим осям).

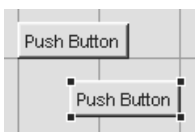


Рис. 12.23. Две кнопки: сверху – закрепленная, снизу – выделенная и перемещаемая

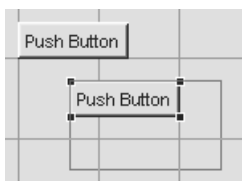


Рис. 12.24. Изменение размера области изображения

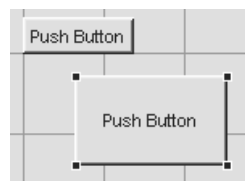


Рис. 12.25. Объект (кнопка внизу) после растяжения

Работу с объектом в окне приложения с GUI облегчает контекстное меню правой клавиши мыши. Так, для введенной и растянутой кнопки оно показано на рис. 12.26. Это меню содержит те команды, которые доступны в данный момент редактирования данного объекта – кнопки.

Для детального изучения и изменения свойств выделенного объекта следует использовать Инспектор свойств. Для его вывода можно использовать команду *Property Inspector* в контекстном меню рис. 12.25. На рис. 12.27 показаны ре-

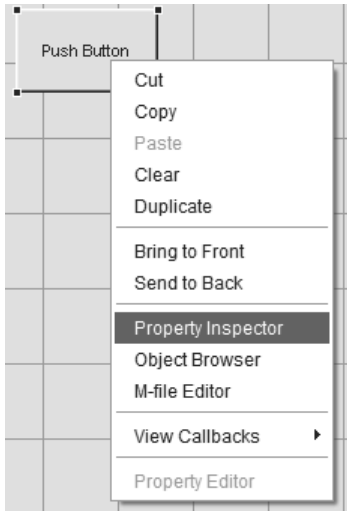


Рис. 12.26. Контекстное меню правой клавиши мыши для кнопки

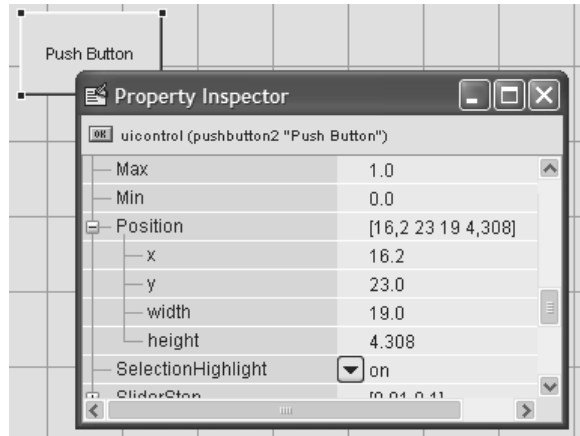


Рис. 12.27. Определение позиции, ширины и высоты кнопки с помощью Инспектора свойств

дактируемая кнопка и окно Инспектора свойств со свойством `Position`, определяющим позицию (место расположения или координаты точки привязки x и y) кнопки, ширину и высоту ее изображения. Напомним, что таким образом можно уточнить множество других свойств кнопки, например ее имя, цвет фона и т. д.

Для дальнейшего описания возможностей редактирования окна приложения с GUI пора отметить команды позиции **Edit (Редактирование)** меню **GUIDE**:

- **Undo** – отмена последней операции;
- **Redo** – восстановление последней отмененной операции;
- **Cut** – вырезка объекта и помещение его в буфер Windows;
- **Copy** – копирование объекта в буфер Windows и сохранение его в окне приложения;
- **Paste** – вызов объекта из буфера и помещение его в окно приложения по месту расположения курсора мыши;
- **Clear** – удаление объекта без размещения его в буфере;
- **Select All** – выделение всех объектов в окне приложения с GUI;
- **Duplicate** – дублирование объекта.

Эти действия вполне очевидны и в подробном описании не нуждаются. Но некоторые тонкости редактирования стоит отметить. Например, ошибочно введенный объект можно удалить, выделив его, несколькими способами, например с помощью:

- нажатия клавиши **Del**;
- команды **Clear** в позиции **Edit** меню;
- команды **Cut** в позиции **Edit** меню;
- команд **Cut** и **Clear** в контекстном меню правой клавиши мыши.

Иногда возникает необходимость в размещении двух и более компонентов в окне приложения с GUI. Это также возможно следующими способами:

- вводом нужного компонента нужное число раз;
- копированием образца в буфер командой **Copy** и применением команды **Paste** в позиции **Edit** меню или из панели инструментов;
- применением команды **Duplicate** из позиции меню **Edit**.

Возможно выделение группы объектов щелчками левой клавиши мыши при нажатой клавише **Ctrl**. Другой способ выделения заключается в создании рамки, в которой находятся нужные объекты. Для этого курсор мыши помещается в подходящее место, задающее один угол рамки, а затем при нажатой левой клавиши перемещением мыши создается рамка – рис. 12.28. Отпустив левую клавишу мыши, можно наблюдать выделение всех размещенных в рамке объектов. Заметим, что выделяются только те объекты, которые полностью попали в выделяющую рамку.

Теперь они ведут себя как единая группа объектов. Их можно переместить при нажатой левой клавише в любое другое место, однако на старом месте объекты исчезнут. Если к группе объектов применить команду **Duplicate**, то наряду с основной группой появится новая группа объектов, которую можно перенести в нужное место при сохранении исходной группы объектов. Это и иллюстрирует рис. 12.29.

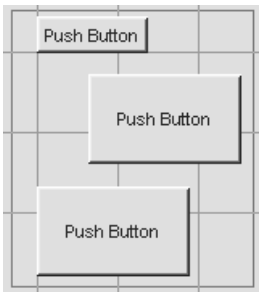


Рис. 12.28. Выделение ряда объектов рамкой

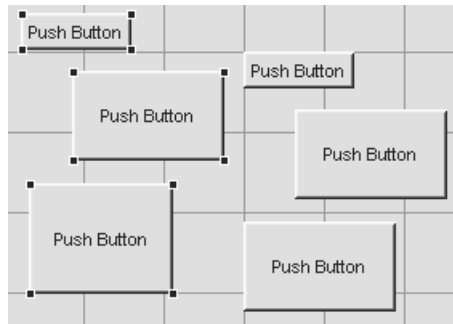


Рис. 12.29. Пример дублирования группы объектов

В заключение этого раздела стоит еще раз отметить, что редактирование объектов полнее всего осуществляется с помощью Инспектора свойств. Например, для изменения цвета кнопки надо активизировать кружок с разноцветной круговой диаграммой в свойстве **BackgroundColor**. Это приведет к выводу стандартного окна выбора цветов, показанного на рис. 12.30 вместе с окном Инспектора свойств.

Начиная с MATLAB 7.0, из набора компонентов исключен компонент **Frame**, и вместо него включен новый компонент **Panel**, свойства **BorderType** и **BorderWidth** позволяют менять форму и ширину его границ. А свойство **TitlePosition** позволяет по-разному ориентировать титульную надпись на панели – рис. 12.31. Надпись можно устанавливать сверху и снизу панели, слева, справа и по центру линии надписи.

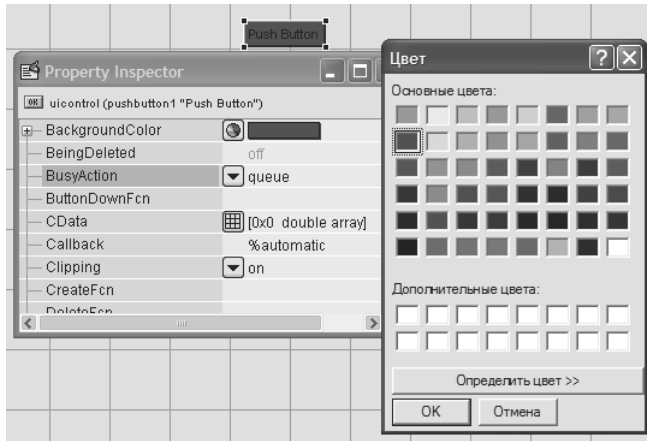


Рис. 12.30. Пример изменения цвета объекта с помощью Инспектора свойств и окна выбора цветов

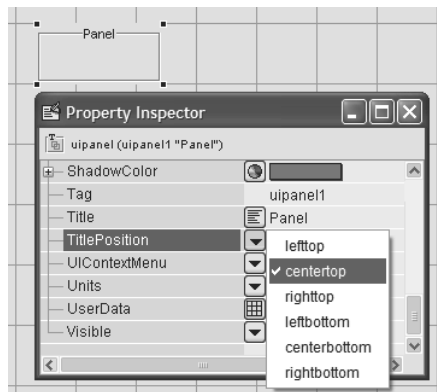


Рис. 12.31. Установка панели и меню ее свойства TitlePosition

С другими возможностями изменения свойств объекта читатель может ознакомиться самостоятельно. Напомним, что действие тех или иных свойств объектов представлено в сводной табл. 12.1, имеющейся в разделе 12.1.4 этого урока.

12.3.4. Средства обзора приложения

Позиция **View** в меню большинства программ предназначена для изменения вида интерфейса путем добавления или устранения тех или иных его деталей. Но в инструменте GUIDE в позиции **View** меню расположены средства обзора приложения:

- **Property Inspector** – вывод окна Инспектора свойств (рис. 12.27);
- **Object Browser** – вывод окна Браузера объектов;
- **M-file Editor** – вывод окна редактора М-файлов;
- **View Callbacks** – вывод подменю с перечнем записываемых функций обработки событий.

Вывод окон Инспектора свойств и редактора М-файлов уже неоднократно описывался. А вот о Браузере объектов стоит поговорить. Команда выводит окно этого браузера, показанное на рис. 12.32, для четырех объектов. Если какие-то объекты выделены, то они будут выделены и в окне Браузера объектов.

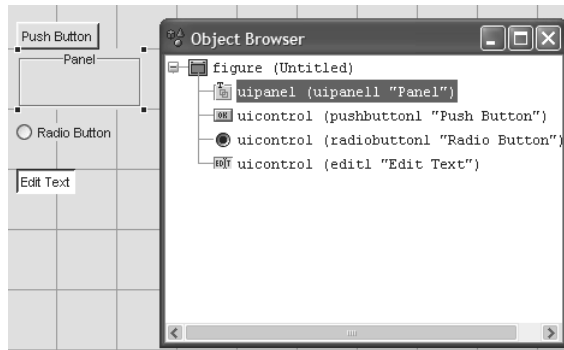


Рис. 12.32. Четыре объекта в окне приложения и окно Браузера объектов

Браузер объектов выводит дерево объектов для текущего окна приложения с GUI-интерфейсом. Работа с браузером вполне очевидна и сводится к открытию или закрытию тех или иных ветвей дерева объектов. С Браузера объектов можно запускать Инспектор свойств. Браузер объектов удобен при проектировании достаточно сложных приложений с большим числом компонентов.

12.3.5. Операции разметки объектов

В позиции **Layout** меню расположены средства разметки наложенных друг на друга объектов:

- **Snap to grid** – опция привязки объектов к сетке;
- **Bring to Front** – перенести выделенный объект или группу объектов на передний план;
- **Send to Back** – перенести выделенный объект или группу объектов на задний план;
- **Bring to Forward** – перенести выделенный объект или группу объектов на один шаг вперед;
- **Send Backward** – перенести выделенный объект или группу объектов на один шаг назад.

Опция **Snap to grid** уже обсуждалась – будучи включенной, она обеспечивает точную привязку перемещаемых мышью объектов к линиям сетки, как только эти объекты приближаются к ним достаточно близко. Остальные команды относятся к объектам, наложенным друг на друга, например из-за дублирования объектов. Поясним работу команды **Bring to Front** (рис. 12.33). Слева показаны три наложенных друг на друга объекта. Виден полностью только первый объект Check Box, который расположен на переднем плане.



Рис. 12.33. Три наложенных объекта слева и результат перемещения объекта на передний план справа

Пусть надо переместить на передний план объект Radio Button, заслоненный объектом Check Box. Для этого, уцепившись мышью за видимый краешек объекта Radio Button, выделим его щелчком левой клавиши мыши. Исполнив команду **Bring to Front**, можно вывести выделенный объект на передний план. Результат этого показан на рис. 12.33 справа. С остальными командами этой группы несложно разобраться самостоятельно.

12.3.6. Операции позиции Tools меню

Ряд специальных операций редактирования приложения с GUI и его запуска на исполнение имеется в позиции **Tools** меню:

- **Run** – запуск приложения с GUI;
- **Align Objects...** – вывод окна выравнивания объектов;
- **Grid and Rules...** – вывод окна, управляющего выводом сетки и мерных линий;
- **Menu Editor...** – вывод окна редактора меню;
- **Tab Order Editor...** – вывод окна изменения порядка следования компонентов при нажатии клавиши **Tab**;
- **GUI Options...** – вывод окна установки опций GUI (рис. 12.22).

Команда **Run**, хотя и является первой в позиции **Tools** меню, обычно выполняется последней после завершения редактирования окна приложения с GUI. Команда **Align Objects** автоматизирует процесс выравнивания объектов. Она полезна при редактировании сложных приложений с большим числом компонентов GUI. Эта команда выводит окно **Align Objects**. Оно показано на рис. 12.34 справа от четырех объектов, разбросанных по горизонтали.

Окно **Align Objects** имеет две части для выравнивания объектов по вертикали **Vertical** и по горизонтали **Horizontal**. Кнопки групп **Align** показывают, как будет осуществлено выравнивание. Кнопка **OFF** служит для отказа от выравнивания. Кнопки групп **Distribute** указывают на тип измерения расстояния между кнопка-

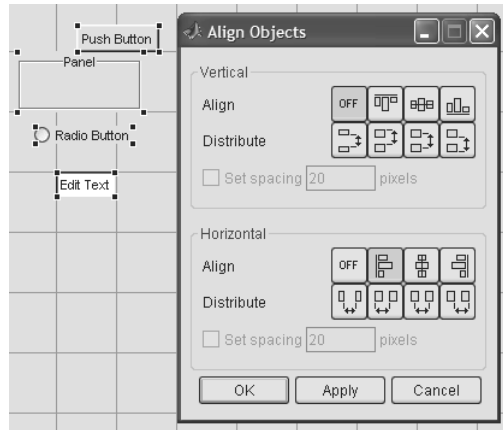


Рис. 12.34. Окно *Align Objects* с выбором режима выравнивания

ми после выравнивания. Опции **Set spacing** позволяют задать расстояние в пикселях. Кнопка **Apply** позволяет применить выравнивание, а **OK** – окончательно завершить его. Как обычно, кнопка **Cancel** служит для отказа от применения окна выравнивания.

Рисунок 12.35 показывает расположение кнопок рис. 12.34 после осуществления выравнивания по горизонтали – кнопки прижимаются к вертикальной черте и начинаются с одного и того же расстояния по горизонтали. Линия выравнивания задается началом кнопки, расположенной левее всех кнопок.

С остальными типами выравнивания нетрудно ознакомиться самостоятельно, тем более что на кнопках отчетливо нарисован тип выравнивания.

Команда **Grid and Rules...** служит для управления выводом сетки и мерных линеек. Вид окна этой команды представлен на рис. 12.36. На нем же представлено окно приложения, в котором удалена сетка и введен вывод мерных линеек – вертикальной линейки и горизонтальной.

Окно **Grid and Rules** имеет следующие опции:

- **Show rules** – показать мерные линейки;
- **Show guides** – показать окно инструмента создания GUI;
- **Show grid** – показать сетку;
- **Snap to grid** – задать привязку к линиям сетки.

При выводе мерных линеек возможен еще один тип выравнивания объектов – по вертикальной или горизонтальной линии, создаваемой курсором мыши при наведении его на вертикальную или горизонтальную линейку. В этом случае курсор превращается в двухстороннюю стрелку и появляется опорная линия, пере-

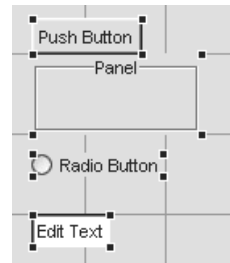


Рис. 12.35. Пример выравнивания четырех объектов по горизонтали

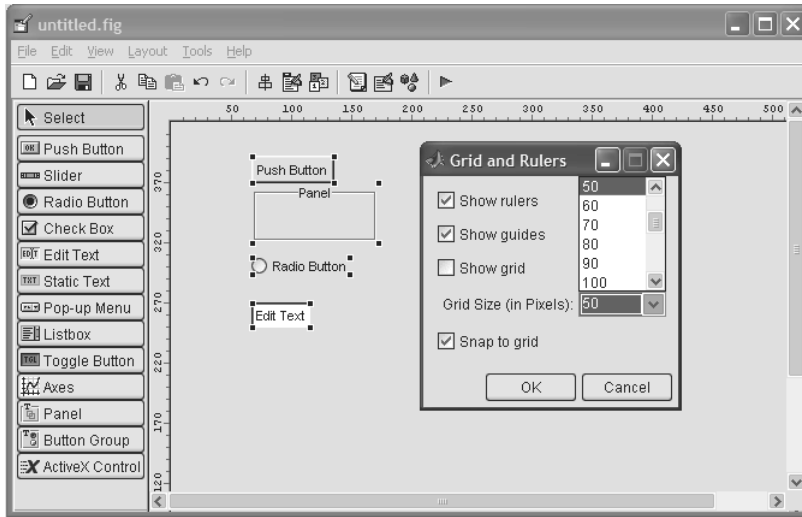


Рис. 12.36. Вид окна приложения с мерными линейками и окна Grid and Rules

мещаемая мышью при нажатой левой клавише. Пример выравнивания трех кнопок по горизонтальной линии показан на рис. 12.37.



Рис. 12.37. Пример выравнивания трех кнопок по горизонтальной линии, положение которой задается курсором мыши

12.3.7. Конструирование меню окна приложения с GUI

Приложения под Windows имеют меню двух основных типов:

- **Menu Bar** – обычное меню в верхней части окна приложения;
- **Context Menus** – контекстное меню правой клавиши мыши.

Оба этих типа меню можно создавать для приложения с GUI. Для этого инструмент GUIDE имеет специальный редактор меню. Он вводится командой **Menu Editor...** в позиции **Tools** меню GUIDE. При этом выводится окно редактора меню, показанное на рис. 12.38. Это окно имеет две вкладки с указанными выше названиями двух типов меню.

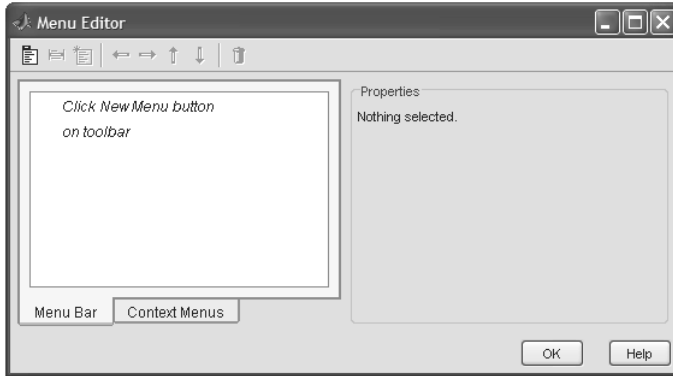


Рис. 12.38. Окно редактора меню перед созданием меню

Это окно делится на две части – в левой имеется пока пустое окно дерева меню, а в правой – окно свойств меню. В верхней части окна имеется панель инструментов. Первые три кнопки (слева направо) имеют следующее назначение:

- **New Menu** – новое меню;
- **New Menu Item** – новая позиция меню;
- **New Context Menu** – новое контекстное меню.

Следующие четыре кнопки панели инструментов (со стрелками) служат для перемещения по дереву команд (позиций) меню, а последняя – для стирания выделенной позиции. С помощью кнопок панели инструментов можно создавать многоуровневые меню, а с помощью подокна свойств – назначать им имена и действия.

Рассмотрим пример построения обычного меню. Начать надо с того, что обдумать структуру меню. Пусть оно имеет три позиции и соответствует следующей схеме:

| График функции | Толщина линии | Построение |
|------------------|---------------|------------|
| Одной переменной | Тонкая | |
| Двух переменных | Средняя | |
| Полярный | Толстая | |

Начать процесс создания меню надо с нажатия кнопки **New Menu** – рис. 12.39. В подокне слева появится одна позиция меню с надписью *Untitled 1*. В правом окне появляются свойства этой позиции. Свойство **Label** задает имя позиции, а свойство **Tag** – имя указателя. Пока они имеют имена также *Untitled 1*.

Рекомендуется вначале создать структуру меню. В нашем случае надо еще дважды нажать кнопку **New Menu** – будет построена заготовка меню из трех позиций. Вернувшись в первую позицию, по дереву меню надо трижды нажать кнопку – будут построены три позиции более низкого уровня для первой позиции меню. Затем то же надо сделать для второй позиции меню. После этого надо заме-

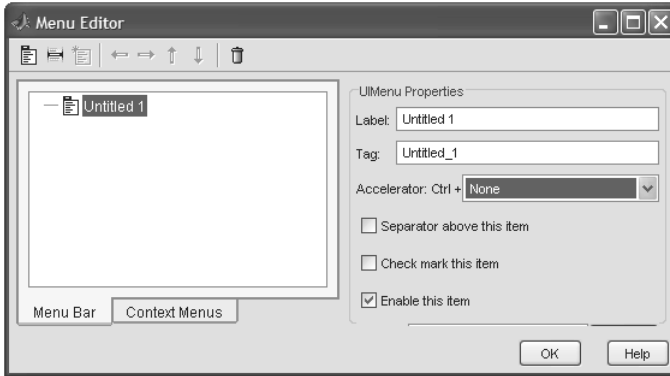


Рис. 12.39. Начало построения меню

нить названия свойств с Untitled N ($N=1,2,3,\dots$) на нужные по смыслу. Например, свойства Label надо изменить в соответствии с выбранными для построения меню именами позиций. На рис. 12.40 показана созданная заготовка нашего меню.

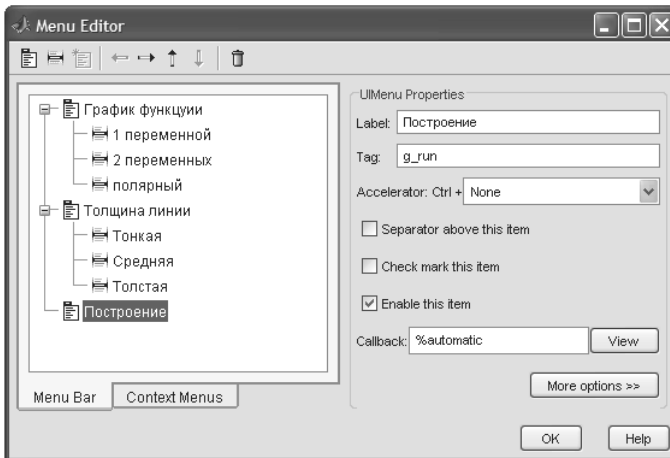
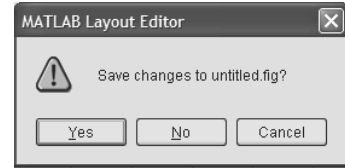


Рис. 12.40. Конец создания заготовки меню

Построение меню завершается нажатием кнопки **OK**. При этом пока ничего не происходит. Чтобы увидеть созданное меню, надо учесть, что оно относится к текущему создаваемому окну приложения с GUI. В нашем случае это окно пустое, но оно уже имеет свой набор свойств. Для наблюдения созданного пустого окна, но уже с нашим меню, достаточно попытаться закрыть окно создаваемого приложения. Появится окно, показанное на рис. 12.41, с предупреждением о необходимости записи файлов, представляющих окно создаваемого приложения.

Рис. 12.41. Окно с предупреждением о необходимости записи файлов приложения



Нажав клавишу **Yes** этого окна, надо в появившемся стандартном окне **Save As** ввести имя FIG-файла, например menu. После записи файла автоматически произойдет переход в окно редактора M-файлов, и в нем появится заготовка M-файла окна с GUI – рис. 12.42. Помимо функции, описывающей структуру окна с меню, в этом окне будут представлены функции обработки событий для каждой позиции меню и каждого указателя.

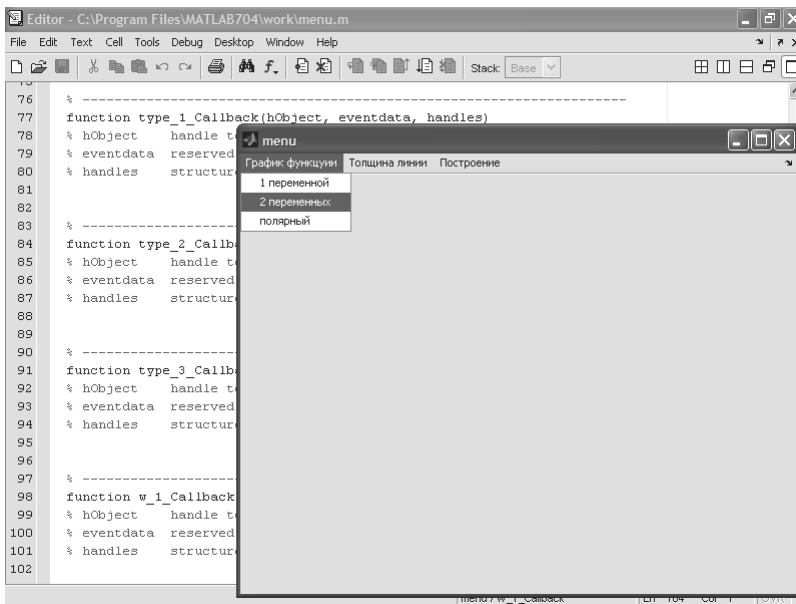


Рис. 12.42. Окно редактора M-файлов и пустое окно приложения с созданным меню

Поскольку функции обработки событий пока не созданы, то меню будет «пустышкой», то есть пока ничего не выполнять. Однако оно уже будет содержать заданные позиции как первого, так и второго уровня. К примеру, на рис. 12.42 показано окно созданного приложения после исполнения команды **Run** в позиции **Debug** окна редактора M-файлов. Окно имеет также стандартное меню, появляющееся при активизации кнопки в начале титульной строки, – рис. 6.43.

Разумеется, на этом процесс создания меню не завершается. Для каждой позиции меню можно вызвать Инспектор свойств и отредактировать свойства, например изменить размеры объекта, цвет основы, тип надписей и т. д. Это может потре-

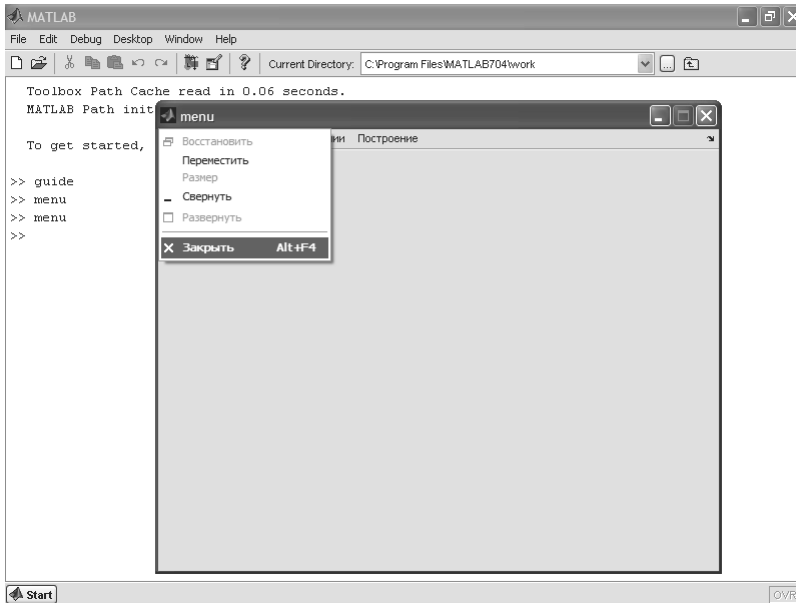


Рис. 12.43. Окно со стандартным меню, вызываемым активизацией кнопки в начале титульной строки

боваться, например, для изменения надписей при обнаружении неточностей в них – внимательный читатель может обнаружить, к примеру, лишнее «у» в слове «функций» в первой позиции меню.

Возможно редактирование уже созданного окна приложения с меню. Для этого надо вывести окно **GUIDE** и в режиме создания нового окна приложения загрузить файл созданного окна. После этого можно приступить к редактированию и пополнению окна приложения. Рисунок 12.44 поясняет это, показывает устранение ошибки в надписи первой позиции меню и вывод свойств этой позиции после нажатия кнопки **More Options**.

Чтобы сделать меню действующим, надо ввести команды для функций обработки событий – в данном случае исполнения заданных позиций созданного меню. На рис. 12.45 представлено окно редактора М-файлов для окна созданного приложения с меню (файл menu.m). В данном случае функции обработки событий представлены в строках 81, 82 программы (построение графика синусоиды $\sin(x)$ в диапазоне изменения переменной x от 0 до 12 с шагом 0,1) в строке 89 (построение фигуры peaks) и в строках 97 и 98 (построение графика функции в полярной системе координат). Команда **Save and Run** в позиции **Debug** редактора М-файлов позволяет вывести окно приложения и убедиться в работоспособности «озвученной» команды меню. Так, на рис. 12.45 показано действие первой команды позиции График функции меню приложения.

На рис. 12.46 и 12.47 представлены действия второй и третьей команд позиции График функции меню созданного приложения. Они строят соответственно гра-

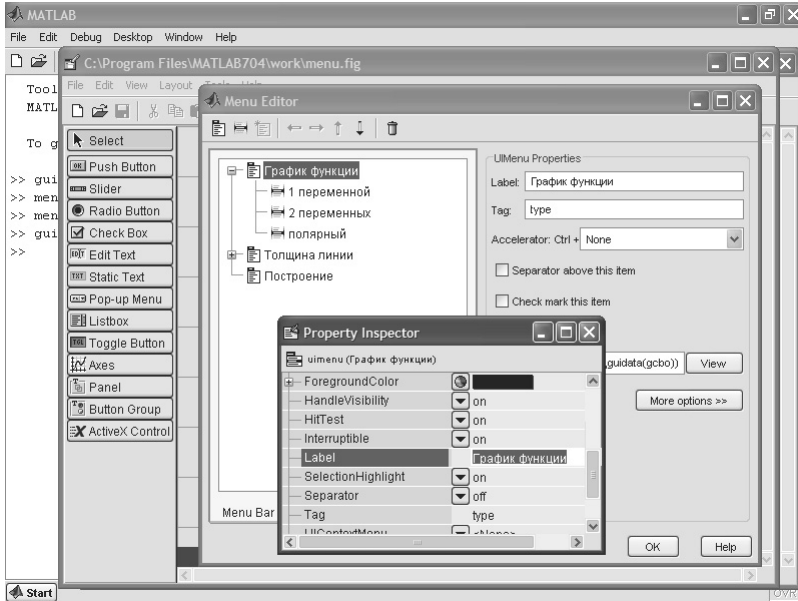


Рис. 12.44. Пример редактирования ранее созданного окна приложения с GUI

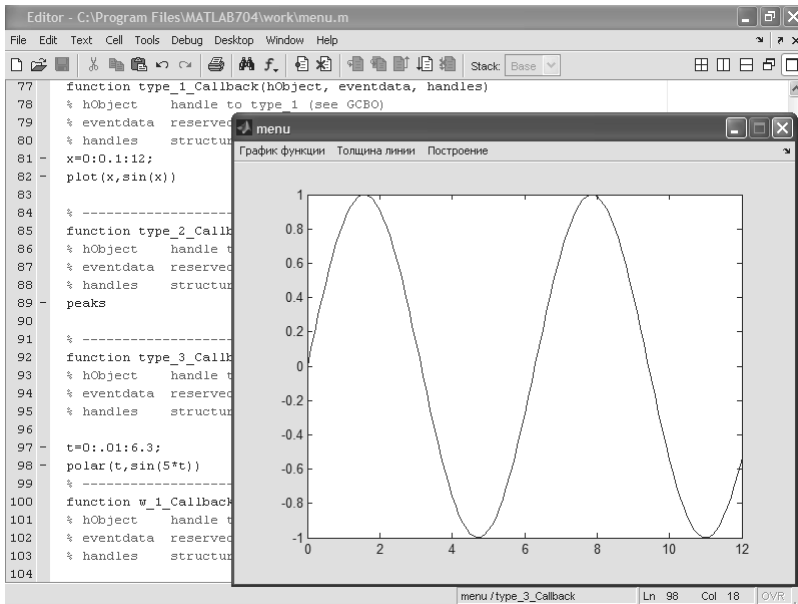


Рис. 12.45. Задание функций обработки для первой позиции График функции меню и действие первой команды – построение графика функции одной переменной

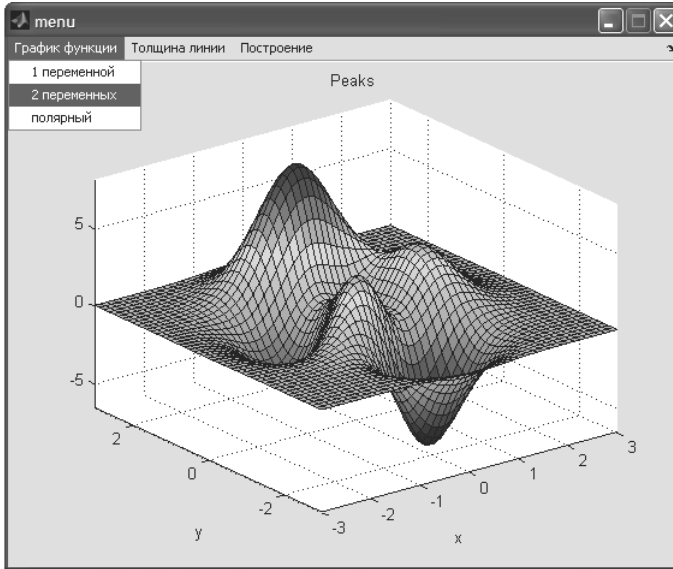


Рис. 12.46. Действие второй команды – построение графика поверхности peaks

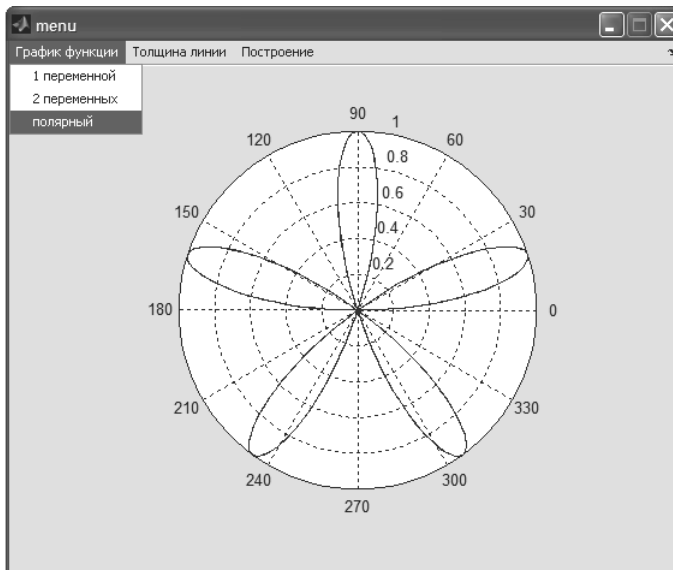


Рис. 12.47. Действие третьей команды – построение графика в полярной системе координат

фик поверхности `peaks` из набора тестовых поверхностей MATLAB и график функции в полярной системе координат.

Остальные позиции меню остаются «без работы», поскольку приведенный пример носит чисто познавательный характер. Читателю рекомендуется поэкспериментировать с построением своих приложений с меню и убедиться в том, что «не боги горшки лепят».

12.3.8. Конструирование контекстного меню окна приложения с GUI

Создание контекстного меню производится аналогично созданию обычного меню. Однако одна тонкость есть – контекстное меню всегда относится к какому-то конкретному объекту, и поэтому после создания окна приложения с объектами и контекстного меню надо обязательно указать, к какому именно объекту это меню относится. Иначе, вопреки ожиданиям, такого меню в окне приложения попросту не будет. Наличие контекстного меню у объекта задается из списка свойства `ContextMenu`. Вы можете создать несколько контекстных меню и приписать их к соответствующим объектам. Контекстное меню может не работать при смене объектов в окне приложения.

Рассмотрим простой пример: с помощью контекстного меню для объекта `Axes` зададим построение одного из трех видов графиков. Начнем с создания окна приложения с объектом `Axes` – рис. 12.48. Объект представлен расширенным прямоугольником с перекрестием и именем `Axes1`.

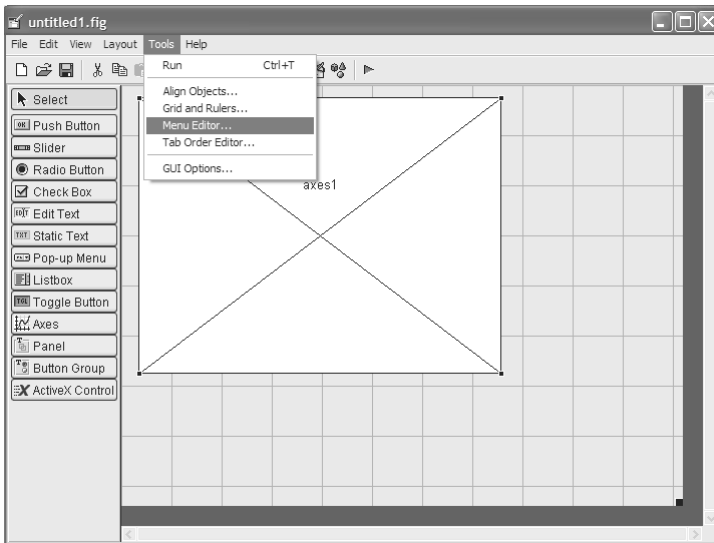


Рис. 12.48. Построение окна приложения с объектом `Axes1`

Теперь командой **Menu Editor...** в позиции **Tools** откроем окно редактора меню. Оно показано на рис. 12.49 с открытой вкладкой **Context Menu**. Сообщение в левом подокне указывает на необходимость нажатия кнопки **New Context Menu**, выделенной на рис. 12.49 и снабженной биркой-подсказкой.

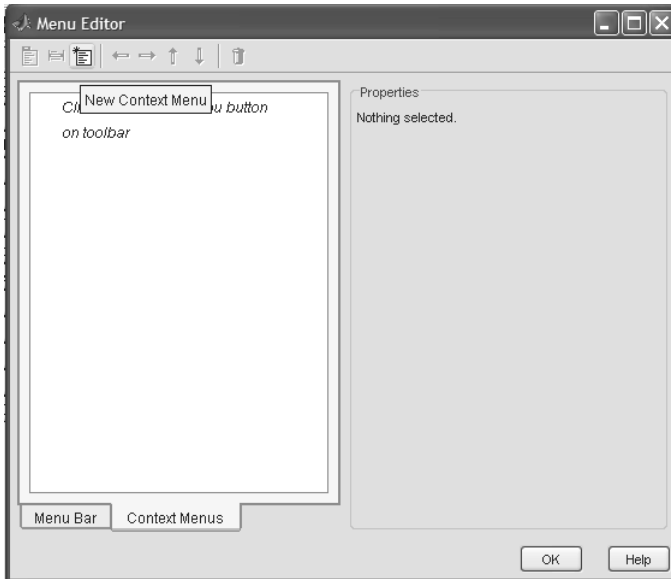


Рис. 12.49. Окно редактора меню
в начале создания контекстного меню

Активизировав кнопку **New Context Menu** и затем трижды нажав кнопку, можно получить структуру контекстного меню с тремя позициями – рис. 12.50. Имена позиций **Label** и теги **Tag** вместо имеющихся по умолчанию **UntitledN** надо заменить на свои, что также показано на рис. 12.50. Введенные надписи очевидны из этого рисунка. Завершим создание меню нажатием клавиши **OK**. Пока к видимым результатам это не приводит.

Теперь вернемся в окно редактора приложения. Щелкнув дважды по объекту **Axes1**, выведем окно Инспектора свойств и для свойства **UIContextMenu** и обнаружим, что кроме позиции по умолчанию **None** есть новая позиция **cmenu**, соответствующая имени созданного нами ранее контекстного меню. Установка свойства на созданное контекстное меню дана на рис. 12.51.

Закрыв окно редактора приложения и записав его с именем **contmenu**, выйдем в редактор **M**-файлов, что показано на рис. 12.52. Здесь представлено начало текста автоматически сгенерированной программы данного приложения. Одновременно выводится и окно самого приложения, показанное на рис. 12.52 внутри окна редактора **M**-файлов. При установке курсора в область будущего графика

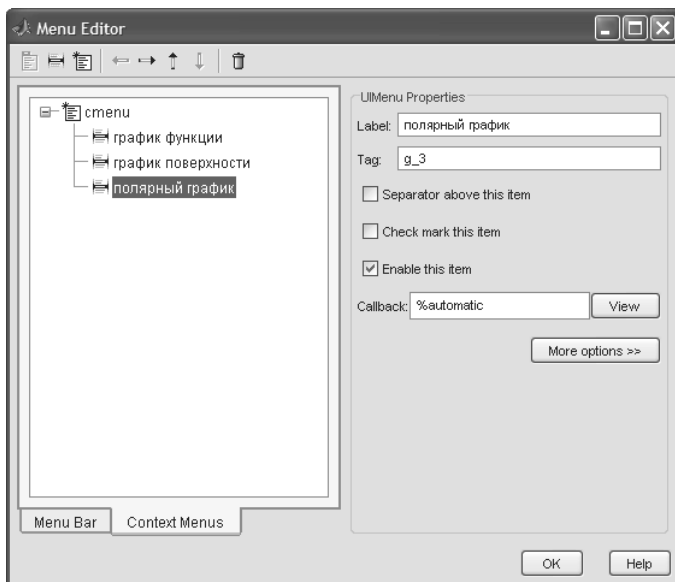


Рис. 12.50. Подготовленное меню

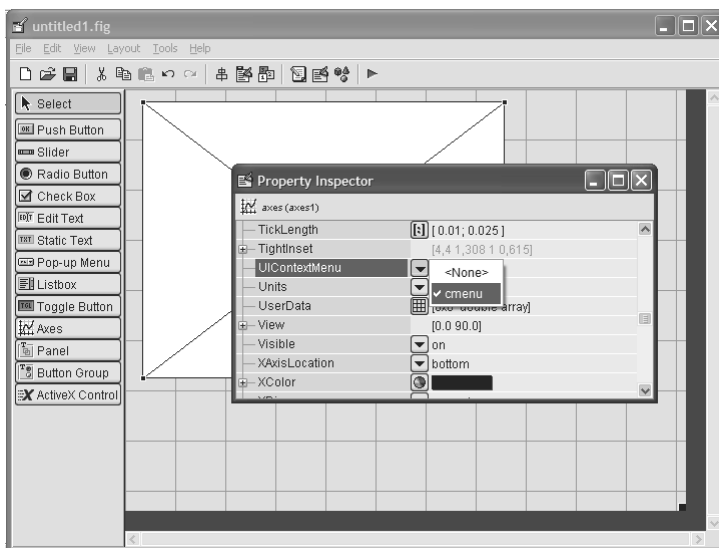


Рис. 12.51. Установка свойства UIContextMenu на созданное контекстное меню

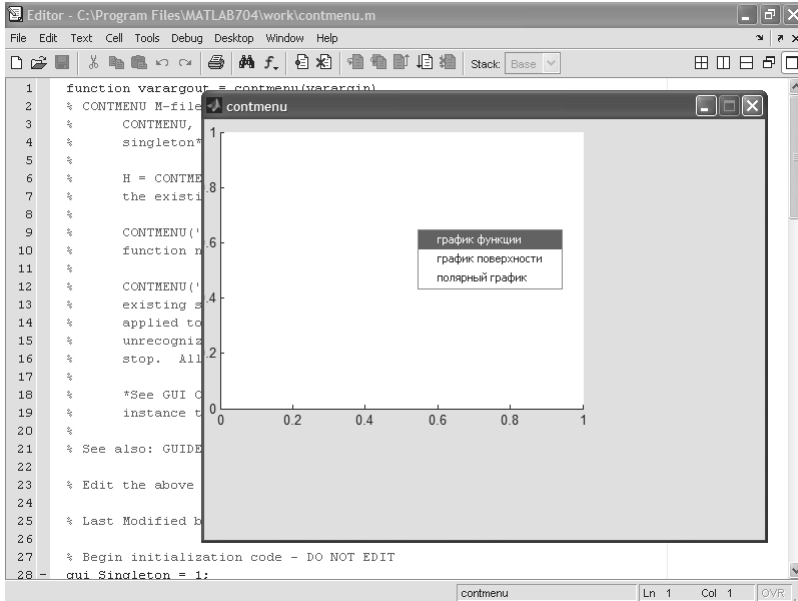


Рис. 12.52. Окно редактора приложения и окно самого приложения в нем

и нажатии на правую клавишу мыши можно увидеть появление созданного контекстного меню.

Однако пока это меню полноценно не работает, хотя можно активизировать любую из его трех позиций и наблюдать выделение той или иной позиции цветом. Чтобы заставить меню выполнять нужные функции, осталось задать функции обработки событий – активизации той или иной позиции контекстного меню. Как это делается, показано на рис. 12.53. Реакция на активизацию первой позиции меню задана в строках 81 и 82. Она задается вводом команд построения двух графиков – функции $\cos(x)$ и функции $\sin(x)/x$. Реакция на активизацию второй позиции меню задается в строке 89 построением трехмерной фигуры `srharm2` из галереи фигур трехмерной графики MATLAB. И наконец, реакция на активизацию третьей позиции контекстного меню задается в строках 96 и 97 построением графика в полярной системе координат. Рисунок 12.53 демонстрирует выбор первой позиции меню и построение графика функций одной переменной.

Примеры выполнения команд второй и третьей позиций созданного контекстного меню показаны на рис. 12.54 и 12.55 соответственно. Для доступа к контекстному меню в этом случае приходится перезапускать приложение, поскольку характер объекта на экране меняется. Читателю рекомендуется продумать усложнение программы, устраняющее этот недостаток. Например, можно ввести дополнительную кнопку, стирающую изображение перед построением нового изображения.

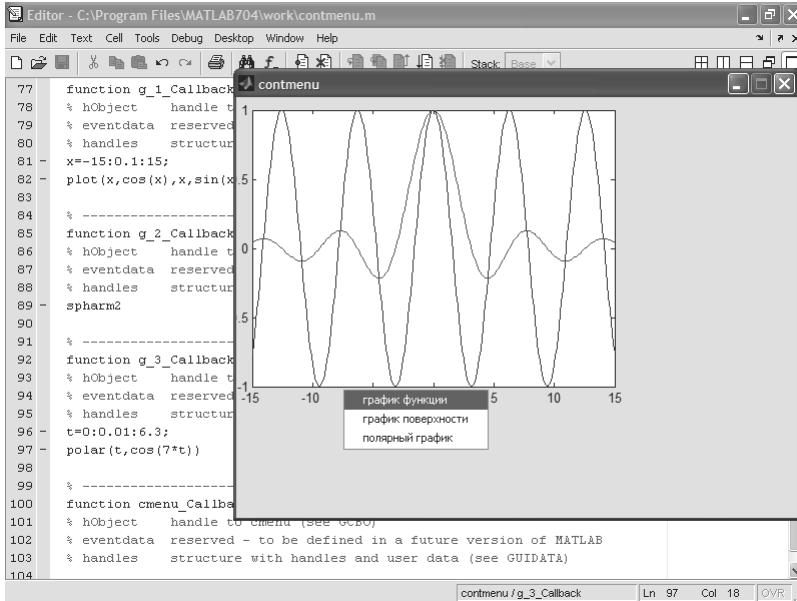


Рис. 12.53. Задание функций обработки событий – активизации позиций контекстного меню и пример выполнения приложения при активизации первой позиции

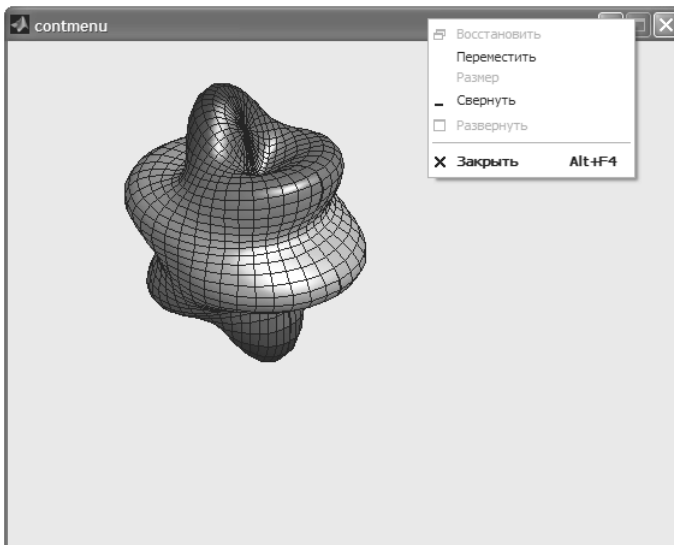


Рис. 12.54. Пример выполнения команд второй позиции контекстного меню

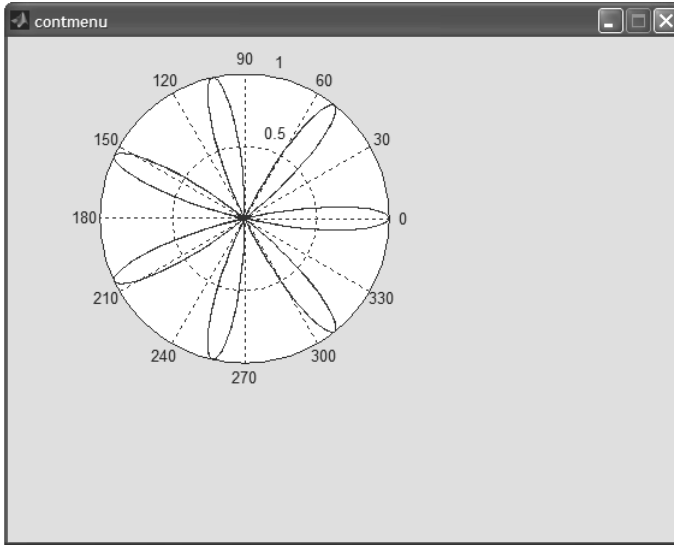


Рис. 12.55. Пример выполнения команд третьей позиции меню

Разумеется, с помощью инспектора свойств можно значительно разнообразить как вид контекстного меню, так и его возможности.

12.3.9. Применение рамки и группы кнопок

К новым в версии MATLAB 7 компонентам GUI относятся рамки и группы кнопок. О рамках уже говорилось. А группы кнопок являются хорошей альтернативой меню, нередко их применение существенно упрощает программы. Рассмотрим применение этих средств на примере построения графиков трех функций: $\sin(x)$, $\sin(x)^3$ и $\sin(x)^5$ с переключением с помощью группы из трех кнопок.

Заготовка окна приложения с GUI для сформулированной задачи представлена на рис. 12.56. В окно приложения выведены рамка, объект построения графика `Axes1` и группа кнопок, в которой размещены три радиокнопки. Помещение объекта `Axes1` внутри рамки позволяет использовать элементы ее оформления, например русскоязычные надписи, размещаемые различным образом (в нашем случае сверху и по центру). С помощью Инспектора свойств (его окно показано на рис. 12.56) названия радиокнопок типа `UntitledN` изменены на имена функций, графики которых будут строиться при активизации той или иной радиокнопки.

Теперь проверим организацию созданного окна приложения. Для этого в позиции **View** меню исполним команду `Object Browser`. Его окно, показанное на рис. 12.57, иллюстрирует состав и подчинение объектов создаваемого приложения. В нем указаны также имена компонентов, измененные с помощью Инспектора свойств.

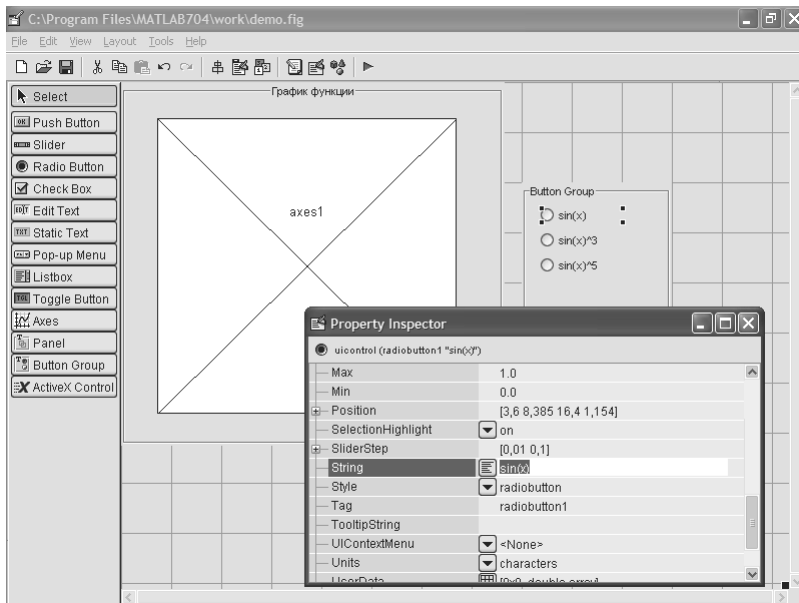


Рис. 12.56. Создание приложения demo с помощью инструмента GUIDE

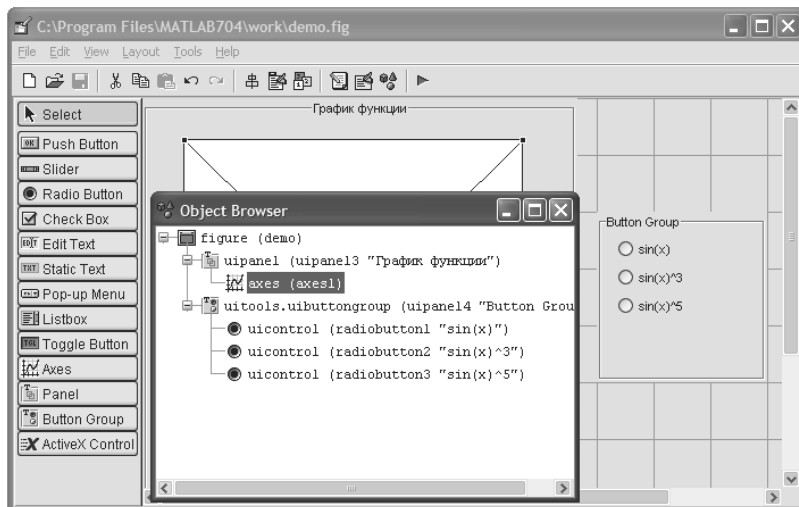


Рис. 12.57. Контроль организации создаваемого окна приложения demo

Чтобы меню было работоспособным, нужно создать для каждой радиокнопки функцию обработки ее действия при активизации кнопки мышью. Для этого, выделив первую кнопку, исполним команду `Callback` в позиции **Callback View** меню – рис. 12.58.

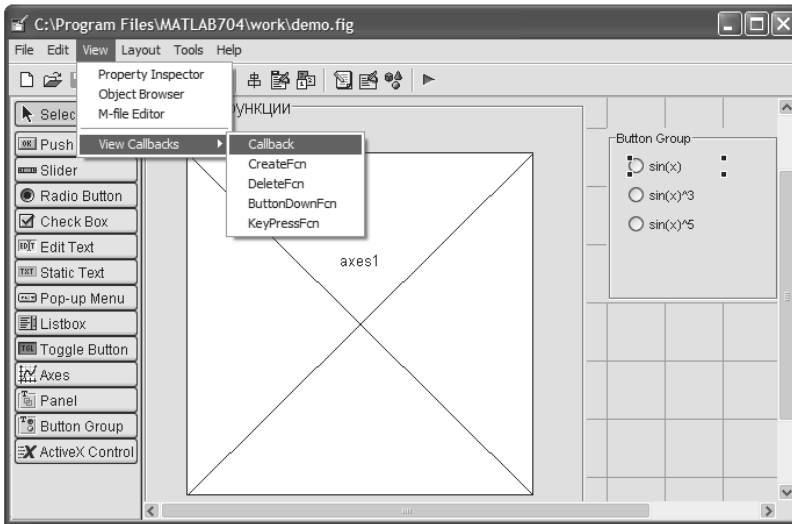


Рис. 12.58. Подготовка к исполнению приложения с GUI

При исполнении этой команды открывается редактор М-файлов с автоматически созданной программой. Отывается место задания Callback-функции первой кнопки, и заголовок функции выделяется. Естественно, что при первом пуске фрагментов программы Callback нет, и пользователь должен их создать. На рис. 12.59 показана часть программы с удаленными англоязычными комментариями и введенными фрагментами для работы Callback-функций. Первый фрагмент в строках 25–30 строит график функции $\sin(x)$, второй фрагмент в строках 32–37 строит график функции $\sin(x)^3$, а третий фрагмент в строках 39–44 строит график функции $\sin(x)^5$. Построение графика синуса при активной первой радиокнопке также представлено на рис. 12.59.

Теперь рассмотрим альтернативные варианты запуска приложения demo. Можно испытать созданное приложение в работе, исполнив команду **Run** в позиции **Tools** меню инструмента GUIDE – рис. 12.60.

Еще один вариант запуска – указание имени приложения demo прямо в командной строке системы MATLAB. Этот вариант представлен на рис. 12.61. На этот раз активизирована вторая кнопка и строится график функции $\sin(x)^3$.

Рисунок 12.62 вновь относится к запуску из редактора М-файлов. Только тут показано начало автоматически сгенерированной программы, из которой удалены комментарии. На этот раз график показан для активной третьей радиокнопки и соответствует графику функции $\sin(x)^5$. Таким образом, выше представлены все три варианта построения графика.

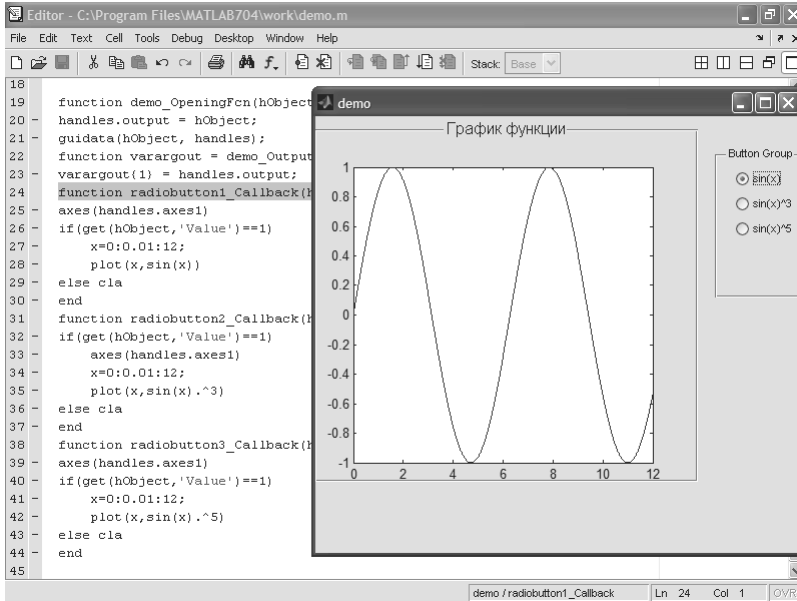


Рис. 12.59. Окно редактора М-файлов с отредактированной программой и результат пуска программы при активизации первой кнопки

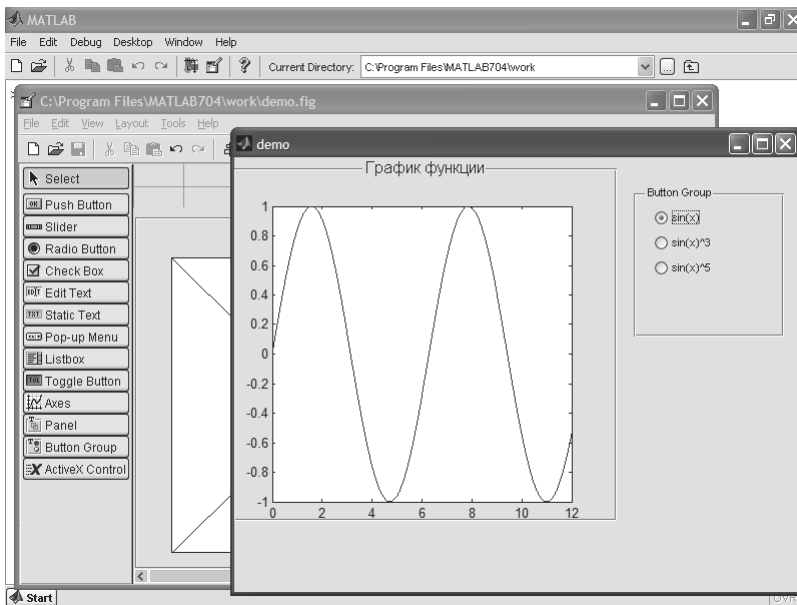


Рис. 12.60. Запуск приложения demo из окна инструмента GUIDE

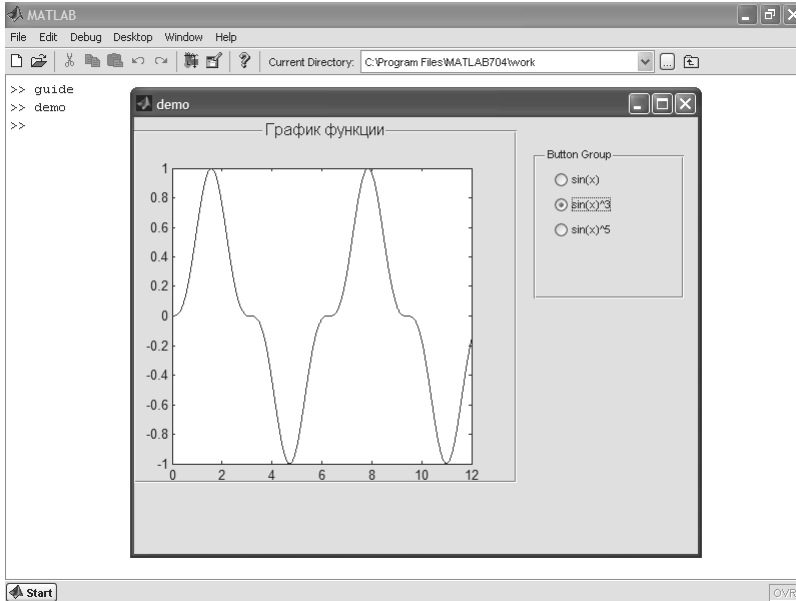


Рис. 12.61. Запуск приложения demo из командной строки MATLAB

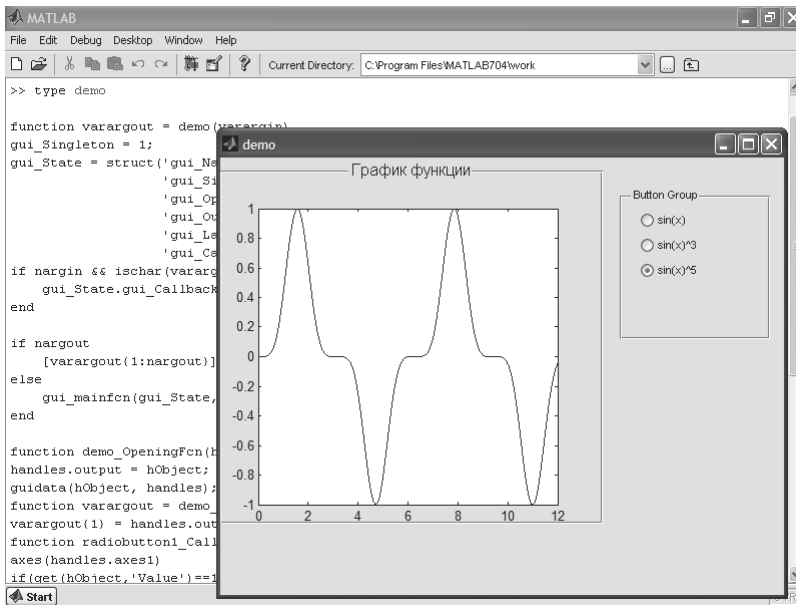


Рис. 12.62. Запуск приложения demo из редактора М-файлов и построение третьего графика

12.3.10. Интерпретация программы приложения

Итак, выше мы на множестве примеров убедились в высокой эффективности автоматической генерации программ приложений с GUI. Разумеется, были использованы достаточно простые познавательные и учебные примеры, демонстрирующие возможности визуально-ориентированного программирования в системе MATLAB простых приложений с GUI. Для более продвинутой работы пользователь должен представлять структуру генерируемых программ и уметь их редактировать и модернизировать под свои цели. Пока мы осуществили два варианта редактирования программ – удаление англоязычных комментариев и введение функций обработки событий Callback. Это очень важные, но, увы, далеко не единственные моменты успешного визуально-ориентированного программирования.

Для дальнейшего «разбора полетов» выведем файл программы demo, исполнив в командной строке MATLAB команду

```
>> type demol
```

Она выводит листинг программы, который мы рассмотрим по частям. Программу можно изучить также с помощью редактора М-файлов, загрузив нужный файл (в нашем случае demo).

Программа начинается с определения следующей главной функции с именем demo:

```
function varargout = demo(varargin)
```

Эта функция имеет один входной аргумент в виде массива `varargin` и один выходной аргумент `varargout`. Устраненный англоязычный комментарий показывает, что функцию можно вызывать в нескольких формах:

- DEMO – обычный запуск приложения без параметров;
- H = DEMO – возвращает дескриптор (указатель) приложения по завершении его работы;
- DEMO('CALLBACK', hObject, eventData, handles, ...) – вызывает локальную функцию с именем CALLBACK в DEMO.M и передает ей список входных аргументов;
- DEMO('Property', 'Value', ...) – создает новое приложение DEMO или продолжает выполнение существующего приложения. Перед стартом приложения вызывается функция `demo_OpeningFunction`, которой передаются пары параметров «свойство-значение». Если параметр `property` или его значение недопустимы, то приложение не запускается. Все входные параметры функции `demo_OpeningFcn` передаются через массив `varargin`.

Кроме того, в комментарии отмечается, что опциях GUI выбрано выполнение единственного приложения (`singleton`) и рекомендуется дополнить комментарий своими данными, если это необходимо. Однако, как уже отмечалось, вводить русскоязычный комментарий не рекомендуется, поскольку это чревато нестабильностью, мягко говоря, программ с такими комментариями.

Следующий фрагмент программы задает начало и окончание инициализации приложения:

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @demo_OpeningFcn, ...
                  'gui_OutputFcn',  @demo_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

Ничего, кроме комментариев (уже устранившихся), в этой части программы менять нельзя. Следующий фрагмент задает открывающие функции, работающие перед открытием окна приложения:

```
function demo_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);

function varargout = demo_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

Далее идет задание функций обработки событий – нажатий радиокнопок в нашем примере demo. Эту важнейшую часть программы мы уже детально обсуждали. Так что ограничимся приведением соответствующих листингов функций обработки:

```
function radiobutton1_Callback(hObject, eventdata, handles)
if (get(hObject, 'Value')==1)
    x=0:0.01:12;
    plot(x, sin(x))
else cla
end

function radiobutton2_Callback(hObject, eventdata, handles)
if (get(hObject, 'Value')==1)
    x=0:0.01:12;
    plot(x, sin(x).^3)
else cla
end

function radiobutton3_Callback(hObject, eventdata, handles)
```

```
if (get(hObject, 'Value')==1)
    x=0:0.01:12;
    plot(x, sin(x).^5)
else cla
end
```

В этих функциях содержательная часть задается уже пользователем и в нашем случае сводится к заданию построения трех графиков.

12.3.11. Несколько советов по созданию приложений с GUI

Многие системы компьютерной математики изначально создавались как системы, позволяющие решать математические задачи вообще без программирования. К MATLAB это относится в меньшей степени, чем к другим системам, хотя и в нем подавляющее большинство задач может решаться с минимальными затратами времени на программирование. Тем не менее MATLAB продвигается на рынок как мощный язык программирования для технических вычислений.

В связи с этим читатель должен понимать, что все описанное выше в части создания приложений с GUI является лишь началом достаточно сложного процесса освоения визуально-ориентированного программирования реальных задач в среде MATLAB. Многие задачи могут успешно решаться и без применения средств GUI или с применением средств объектно-ориентированного и процедурного программирования, уже вошедших в состав системы MATLAB.

Недостатком визуально-ориентированного создания приложений с GUI являются громоздкость программных кодов и неполная их оптимизация. При создании приложений обычным способом можно получить гораздо более компактные коды, хотя бывает и обратная ситуация. Кроме того, особенно при проектировании сложных приложений, детали созданных кодов могут быть не вполне ясными, а серьезный пользователь часто не может мириться с применением не вполне понятных ему кодов и фрагментов программ.

В связи с этим не рекомендуется сразу готовить сложные приложения с окнами, содержащими большой набор интерфейсных элементов. Следует постепенно вводить новые элементы интерфейса и внимательно следить за постепенным усложнением получаемой программы. При этом лучше не оставлять непонятными даже «мелкие» ее детали.

Надо внимательно следить за правильностью использования директорий файловой системы. Если задается новая директория, то GUIDE выводит окно с предупреждением о смене директории, показанное на рис. 12.63. Возможно изменение текущей директории MATLAB или добавление новой директории. Рекомендуется сохранять исходные коды, созданные GUIDE автоматически, со всеми входящими в них комментариями, и отдельно готовить файлы без таких комментариев. В сложных случаях комментарии, даже англоязычные, могут оказаться очень полезными. Следует также стараться максимально использовать как возможности

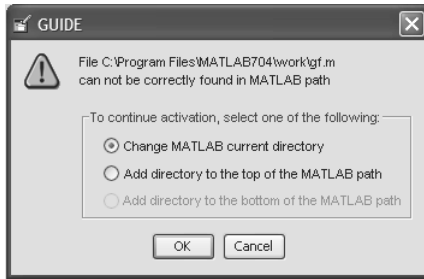


Рис. 12.63. Окно с предупреждением о смене директории

редактирования окна приложения с GUI средствами инструмента GUIDE, так и возможности контроля работы приложений различными способами.

В целом надо помнить, что ни одна книга не заменит практического опыта программирования в системе MATLAB. Про-

фессиональный уровень программирования в этой системе требует длительных и упорных навыков программирования. Впрочем, это справедливо и при программировании на любом языке программирования.

12.4. Стандартные диалоговые окна MATLAB

12.4.1. Набор диалоговых окон

В MATLAB входят около полутора десятков стандартных окон, которые пользователь может использовать при проектировании своих приложений с GUI. Кроме того, назначение и вид этих окон полезно знать, поскольку они встречаются повсеместно при работе в среде MATLAB.

Ниже в табл. 12.2 представлен список диалоговых окон с их названиями и назначением.

Таблица 12.2. Названия и назначения стандартных диалоговых окон

| Имя окна | Назначение диалогового окна |
|------------|--|
| Dialog | Диалоговое окно общего назначения |
| Errordlg | Диалоговое окно для вывода сообщения об ошибке |
| Helpdlg | Диалоговое окно справки |
| inputdlg | Диалоговое окно ввода |
| Listdlg | Диалоговое окно списка |
| Msgbox | Диалоговое окно сообщения |
| Pagedlg | Диалоговое окно страницы |
| Printdlg | Диалоговое окно печати |
| Questdlg | Диалоговое окно запроса |
| Uigetfile | Диалоговое окно запроса имени файла при считывании |
| Uigetpref | Диалоговое окно предпочтений |
| Uiputfile | Диалоговое окно запроса имени файла при записи |
| Uisave | Диалоговое окно записи рабочего пространства |
| Uisetcolor | Диалоговое окно установки цвета |
| Uisetfont | Диалоговое окно установки шрифта |
| Waitbar | Окно индикации прогресса |
| Warning | Диалоговое окно с предупреждением |

Эти диалоговые окна имеют свои интерфейсные элементы и вызываются их указанием по имени. Возможны различные формы записи вызовов диалоговых окон из командной строки MATLAB или из М-файлов. Вид окна определяется назначением его свойств.

12.4.2. Справка по диалоговым окнам и их свойства

Команда

```
>> help name
```

где name – имя диалогового окна, выводит справку по заданному диалоговому окну с указанием формы обращения к функции и значением свойств окна. Например, для диалогового окна общего назначения справка выдается в виде:

```
>> help dialog
```

```
DIALOG Create dialog figure.
  H = DIALOG(...) returns a handle to a dialog box and is
  basically a wrapper function for the FIGURE command. In
  addition,
  it sets the figure properties that are recommended for dialog
  boxes.
  These properties and their corresponding values are:
```

| | |
|---------------------|--|
| 'BackingStore' | – 'off' |
| 'ButtonDownFcn' | – 'if isempty(allchild(gcf)), close(gcf), end' |
| 'Colormap' | – [] |
| 'Color' | – DefaultUiControlBackgroundColor |
| 'DockControls' | – 'off' |
| 'HandleVisibility' | – 'callback' |
| 'IntegerHandle' | – 'off' |
| 'InvertHardcopy' | – 'off' |
| 'MenuBar' | – 'none' |
| 'NumberTitle' | – 'off' |
| 'PaperPositionMode' | – 'auto' |
| 'Resize' | – 'off' |
| 'Visible' | – 'on' |
| 'WindowStyle' | – 'modal' |

Any parameter from the figure command is valid for this command.

See also figure, uiwait, uiresume

Reference page in Help browser

```
doc dialog
```

Для доступа к полноценной справке по каждому диалоговому окну нужно использовать команду

```
>> doc name
```

В справке можно найти подробные данные о форме записи функции вызова окна и возможных значениях ее свойств.

12.4.3. Работа с простыми диалоговыми окнами

Ввиду простоты доступа к информации о диалоговых окнах рассматривать их все детально не имеет особого смысла. Хотя справки по этим окнам даны на английском языке, они довольно лаконичны и понятны. В связи с этим рассмотрим лишь несколько примеров работы с готовыми простыми диалоговыми окнами.

Диалоговое окно общего назначения вызывается командой

```
h=dialog('Свойство1', 'Значение1',...)
```

При этом выводится пустое окно и устанавливается его описатель-указатель.

Диалоговое окно для вывода сообщения об ошибке служит для организации обработки ошибок в программе и вывода сообщений об ошибках. Оно имеет следующие формы вызова функции:

```
errordlg          errordlg('errorstring')
errordlg('errorstring','dlgname')
errordlg('errorstring','dlgname','on')
h = errordlg(...)
```

К примеру, команда

```
errordlg('Переменная не определена','Ошибка')
```

выводит диалоговое окно с именем «Ошибка» и сообщением об ошибке «Переменная не определена» – рис. 12.64.

Диалоговое окно справки вызывается следующими командами:

```
helpdlg          helpdlg('helpstring')
helpdlg('helpstring','dlgname') h = helpdlg(...)
```

Так, команда

```
>> helpdlg('sin')
```

вызывает появление окна справки, показанного на рис. 12.65. Вопреки ожиданиям окно не дает вызова справки по функции синуса. Чтобы это сделать, надо задать программу обработки нажатия клавиши **ОК** данного окна.

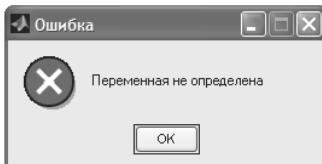


Рис. 12.64. Окно с сообщением об ошибке



Рис. 12.65. Окно справки

Подобным образом используется диалоговое окно сообщения:

```
msgbox(message)          msgbox(message,title)
msgbox(message,title,'icon')
```

```
msgbox(message,title,'custom',iconData,iconCmap)
msgbox(...,'createMode') h = msgbox(...)
```

Пример вывода сообщения (рис. 12.66):

```
>> msgbox('Не забудьте вставить диск')
```

Диалоговое окно с предупреждением задается функцией

```
h = warndlg('warningstring','dlgname')
```

Ее параметрами являются предупреждающее сообщение `warningstring` и титульная надпись `dlgname`. Например, команда

```
>> h = warndlg('Деление на 0 запрещено!','Предупреждение')
```

выводит окно, показанное на рис. 12.67. Обработка возникшей ситуации может быть выполнена анализом значения дескриптора-указателя `h`.

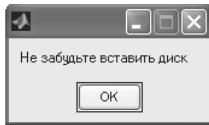


Рис. 12.66.
Пример вывода
сообщения

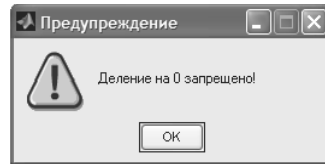


Рис. 12.67. Окно
с предупреждением

12.4.4. Диалоговые окна множественного типа

Ряд диалоговых окон можно отнести к множественному типу. Такие окна позволяют различать несколько операций или вводить данные из списка. Так, диалоговое окно запроса задается функцией `questdlg`, имеющей следующие формы записи:

```
button = questdlg('qstring')
button = questdlg('qstring','title')
button = questdlg('qstring','title','default')
button = questdlg('qstring','title','str1','str2','default')
button = questdlg('qstring','title','str1','str2','str3','d')
```

Функция строит диалоговое окно и возвращает значение одной нажатой кнопки из ряда кнопок. Например, команда

```
>> button = questdlg('Нажать?', 'Запрос')
```

выводит диалоговое окно, показанное на рис. 12.68, и при нажатии на кнопку **Yes** выводит в командной строке

```
button =
Yes
```



Рис. 12.68. Окно
с запросом

При нажатии на кнопки **No** и **Cancel** выводится значение переменной `button`, соответственно, `No` и `Cancel`. Анализируя значения `button`, нетрудно организовать фрагменты программ, выполняющих нужные действия.

Диалоговое окно ввода одного или нескольких данных (параметров) вводится следующими командами:

```
answer = inputdlg(prompt)
answer = inputdlg(prompt, dlg_title)
answer = inputdlg(prompt,dlg_title, num_lines)
answer = inputdlg(prompt, dlg_title, num_lines, defAns)
answer = inputdlg(prompt,dlg_title,num_lines,defAns,Resize)
```

Здесь `prompt` – подсказка, `dlg_title` – титульная надпись в окне, `num_lines` – число линий (вводимых параметров), `defAns` – значение по умолчанию и `Resize` – указание (on или off) на изменение размера окна. Пример ввода двух параметров `a` и `b` представлен ниже (см. также рис. 12.69):

```
>> ab = inputdlg({'Input a','Input b'}, 'Ввод данных')
ab =
    "2"
    "5"
```

Диалоговое окно выбора из списка задается функцией

```
[Selection,ok] = listdlg('ListString',S,...)
```

Следующий пример создает окно (рис. 12.70) со списком текущей директории:

```
>> d = dir; str = {d.name};
    [s,v] = listdlg('PromptString','Select a file:',...
                  'SelectionMode','single',...
                  'ListString',str)
```

```
s = 4
v = 1
```

В этом примере параметр `s` определяет номер выделенной позиции списка. При этом учитываются позиция с точкой (это корневая директория) и позиция с двумя точками (вышестоящая директория). Другая переменная `v` принимает значение 1, если пользователь сделал выбор, и 0, если выбор не был сделан. Сам выбор, как обычно, осуществляется курсором мыши и кратким нажатием левой клавиши – в результате указанная позиция списка выделяется. Фиксируется выбор нажатием клавиши **Enter** или кнопки **OK** окна выбора из списка. Следующий пример показывает вывод имени файла, который был выбран:

```
>> str(s)
ans = 'contmenu.fig'
```

Если выбор не сделан, то параметр `s` принимает значение пустого списка.

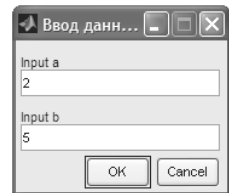


Рис. 12.69. Окно ввода двух параметров

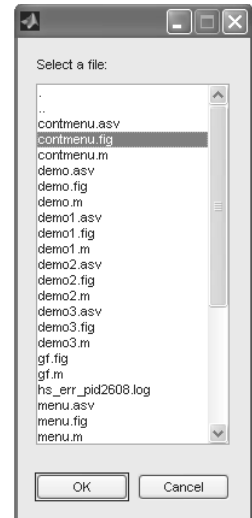


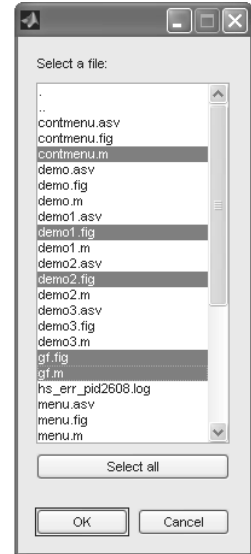
Рис. 12.70. Окно выбора из списка

Рис. 12.71. Окно выбора из списка
в режиме множественного выбора

В этом примере полезно отметить, что задана опция **SelectMode (Режим выбора)** со значением `single` – выбор одной позиции списка. Между тем по умолчанию задается значение этой опции `multiply` (множественный выбор). Для обеспечения такого выбора надо либо назначить свойству `SelectMode` значение `multiply`, либо просто исключить это назначение, как в следующей команде (рис. 12.71):

```
>> d = dir;    str = {d.name};
    [s,v]=listdlg('PromptString','Select a file:',...
    'ListString',str)
s = 5    10    13    18    19
v = 1
```

В этом случае множественный выбор делается щелчком мыши по нужным позициям списка при нажатой клавише **Ctrl**. Можно выделить все позиции с помощью новой кнопки **Select All**, появившейся в окне выбора в этом случае. Переменная `s` возвращает список чисел, задающих номера выделенных позиций списка. Напомним, что команда `help listdlg` выводит достаточно полную информацию об этой интересной и полезной функции, в частности набор всех ее свойств.



12.4.5. Диалоговые окна файловых операций

Практически все серьезные приложения имеют средства для работы с файлами. Диалоговое окно запроса имени файла при считывании (рис. 12.72) выводится при исполнении функции `uigetfile`. Она имеет ряд форм записи:

```
uigetfile
uigetfile('FilterSpec')
uigetfile('FilterSpec','DialogTitle')
uigetfile('FilterSpec','DialogTitle','DefaultName')
uigetfile(...,'Location',[x y])
uigetfile(...,'MultiSelect',selectmode)
[FileName,PathName] = uigetfile(...)
[FileName,PathName,FilterIndex] = uigetfile(...)
```

К примеру, функция

```
>> uigetfile('','Открытие файла')
```

выводит стандартное окно загрузки файла, показанное на рис. 12.72, в котором задана титульная надпись «Открытие файла» и отсутствует задание имени файла (если в промежутке между двумя апострофами поставить надпись, то она появит-

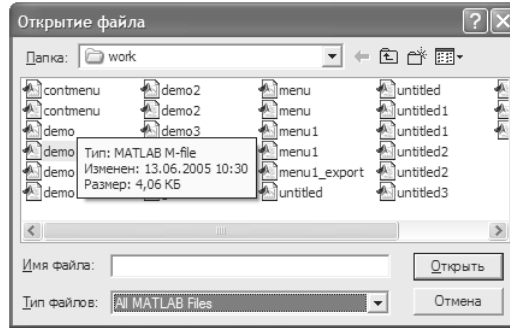


Рис. 12.72. Окно загрузки файла

ся в области задания имени файла). В окне действует всплывающая подсказка, открывающаяся список всех типов файлов, доступных к загрузке в среде MATLAB, и прочие характерные возможности этого стандартного окна.

После выбора файла и нажатия клавиши **ОК** функция возвращает переменную `ans` с именем выбранного файла. Если пользователь отказался от загрузки файла, то переменная `ans` принимает нулевое значение.

Другое диалоговое окно запроса имени файла при записи выводит функция

```

uiputfile
uiputfile('FilterSpec')
uiputfile('FilterSpec','DialogTitle')
uiputfile('FilterSpec','DialogTitle','DefaultName')
uiputfile(...,'Location',[x y])
[FileName,PathName] = uiputfile(...)
[FileName,PathName,FilterIndex] = uiputfile(...)

```

Например, команда

```
>> uiputfile('','Запись файла')
```

выводит диалоговое окно, показанное на рис. 12.73. Это хорошо знакомое окно стандартной команды **Save As**. Только в титул окна помещена надпись «Запись

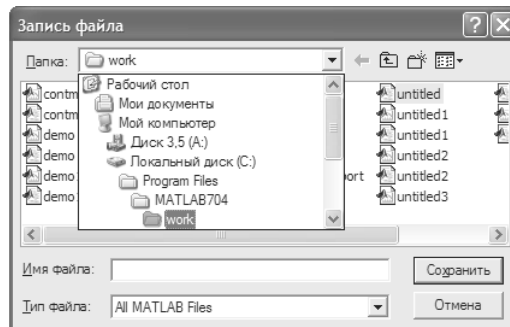


Рис. 12.73. Диалоговое окно записи файлов с заданным именем

файла». Окно обладает всеми возможностями окна **Save As**, например выводом дерева файловой системы компьютера, переходом на более высокий уровень файловой системы, ярлыком создания новой папки и т. д. Значения переменной `ans` в данном примере те же, что и для функции загрузки файлов.

12.4.6. Диалоговые окна установки цвета и шрифтов

Для вывода диалогового окна установки цвета служит функция

```
c = uisetcolor(h_or_c, 'DialogTitle')
```

Ее первый параметр задает спецификацию цвета по умолчанию, а второй – титульную строку окна. Например, команда

```
>> c = uisetcolor([1 1 1], 'Окно выбора цвета')
```

выводит стандартное диалоговое окно (рис. 12.74) с палитрой-матрицей 48 цветов с выделенным белым цветом, поскольку набор цветов `[1 1 1]` в формате RGB задает белый цвет. Квадратик с этим цветом в матрице цветов выделен – он находится в правом нижнем углу матрицы цветов. Мышью можно выбрать любой из других цветов из матрицы цветов. Нажатие клавиши **ОК** после этого выведет массив `[r g b]`, соответствующий выбранному цвету.

Кнопка **Определить цвет** расширяет начальное окно, и оно принимает вид, показанный на рис. 12.75. В правой части его имеются палитра более тонких цветов и линейка для плавной регулировки их интенсивности. Это позволяет выбрать и запомнить в палитре **Дополнительные цвета** набор дополнительных цветов. Для этого служит кнопка **Добавить в набор**.

Диалоговое окно установки шрифта также используется во многих приложениях под Windows.

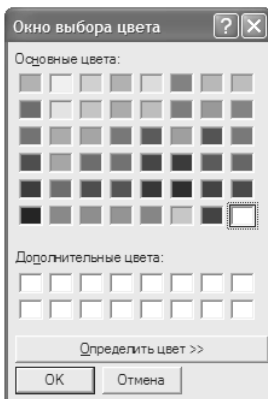


Рис. 12.74. Начальное окно выбора цветов

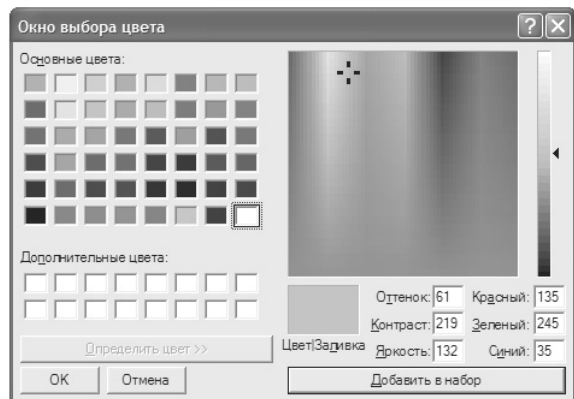


Рис. 12.75. Расширенное окно выбора цветов

12.4.7. Диалоговые окна параметров страницы и печати

Ряд диалоговых окон предназначен для настройки и осуществления печати принтером, подключенным к компьютеру. В отличие от большинства программ под Windows, MATLAB не использует встроенные в Windows окна указанного типа и имеет свои окна. Более того, доступные для создания GUI окна заметно отличаются от тех, которые выводятся соответствующими командами позиции **File** меню.

Рассмотрение их начнем с функций вывода диалогового окна установки параметров страницы. Для этого служит функция

```
dlg = pagesetupdlg(fig)
```

Она выводит довольно большое окно, показанное на рис. 12.75. В левой части окно содержит дощечки параметров, зависящие от выбранной вкладки, а в правой части – пример печати графика. На рис. 12.76 окно представлено с открытой вкладкой **Size and Position (Размер и позиция)**. Установки на этой вкладке вполне очевидны – можно использовать автоматическое размещение печатаемого объекта с центрированием его на странице или ручное с установкой верхних и нижних полей, ширины и высоты изображения. Есть возможность выбора единиц измерения размеров в дюймах, сантиметрах и пикселях.

На вкладке **Paper (Бумага)**, представленной на рис. 12.77, задаются установки бумаги (ширина и высота бумажного листа) и ориентации изображения относительно бумаги – портретная, ландшафтная и с поворотом. Расположение иллюстрируется изображением листа бумаги с загнутым одним из углов.

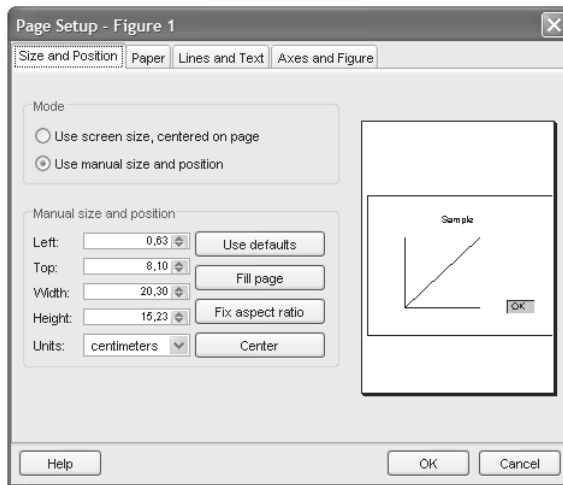


Рис. 12.76. Окно установки параметров страницы с открытой вкладкой **Size and Position**

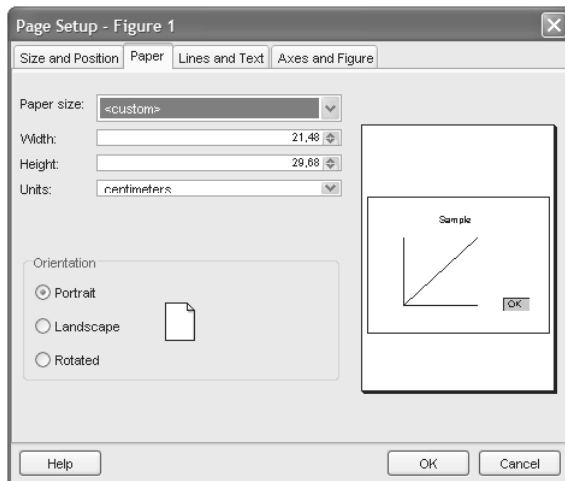


Рис. 12.77. Окно установки параметров страницы с открытой вкладкой **Paper**

Вкладка **Lines and Text (Линии и текст)**, показанная на рис. 12.78, содержит лишь две опции для установки типа печати по цвету. Можно задать черно-белую или цветную печать.

Последняя вкладка **Axes and Figure (Оси и фигура)** содержит ряд установок, относящихся к объектам **Axes** и **Figure**. Эта вкладка представлена на рис. 12.79. Установки на этой вкладке позволяют задать ограничения при печати осей, пе-

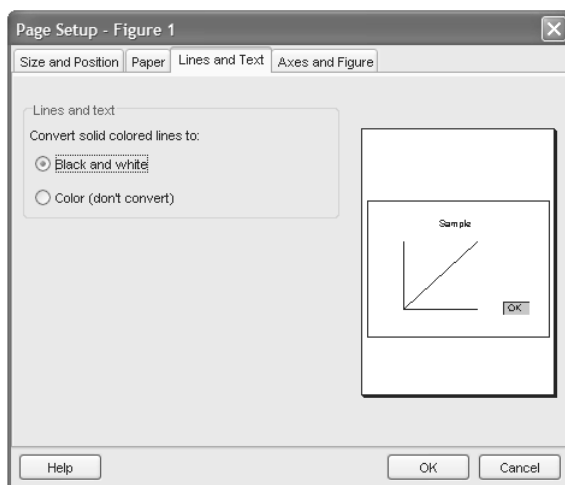


Рис. 12.78. Окно установки параметров страницы с открытой вкладкой **Lines and Text**

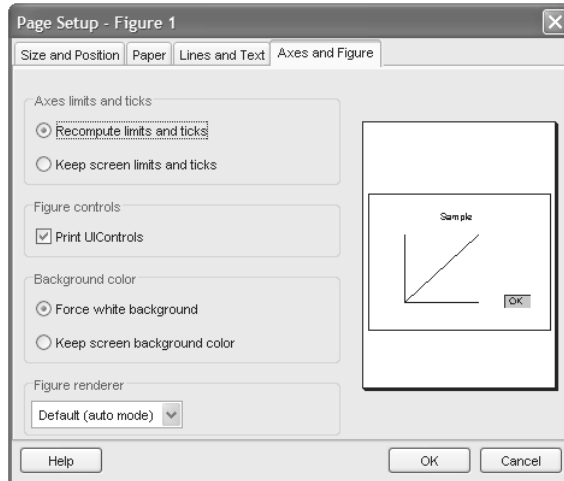


Рис. 12.79. Окно установки параметров страницы с открытой вкладкой **Axes и Figure**

чать UIControl, цвет основы печатаемой страницы и тип рендеринга (цветовой окраски) для объектов с функциональной окраской. В последнем случае установка осуществляется из списка **Figure Render** с позициями: **Default (auto mode)**, **Painter**, **ZBuffer** и **OpenGL**.

В наследство от прежних версий системе MATLAB досталась функция `pagedlg`. Она выводит более скромное окно установки параметров страницы, показанное на рис. 12.80. Эту функцию пока можно использовать, но она уже относится к числу нерекондуемых.

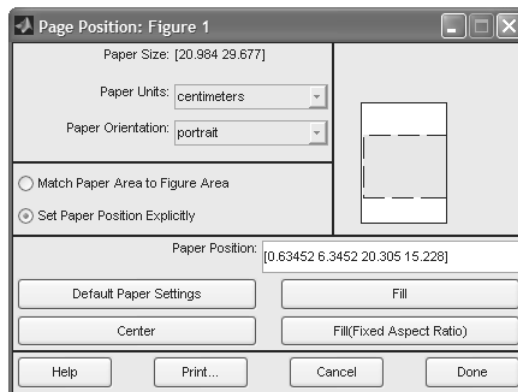


Рис. 12.80. Окно установки параметров страницы функции `pagedlg`

Рис. 12.81. Окно установки параметров страницы команды **Page Setup...** в позиции **File** меню системы MATLAB

Для сравнения на рис. 12.81 показано окно установки параметров страницы, которое выводит команда **Page Setup...** в позиции **File** меню окна системы MATLAB. Это окно отличается от представленных выше окон. Такое разнообразие окон установки параметров страницы печати говорит о том, что представленные в этих окнах интерфейсные средства находятся в развитии и можно ожидать нюансов в их представлении в текущих версиях системы MATLAB.

Диалоговое окно печати выводится функцией `printdlg`, которая имеет четыре формы записи:

```
printdlg
printdlg('-crossplatform', fig)
printdlg(fig)
printdlg('-setup', fig)
```

При использовании функции `printdlg` без параметров выводятся окно печати и окно фигуры – см. рис. 12.82. Параметр `fig` позволяет идентифицировать печатаемую фигуру.

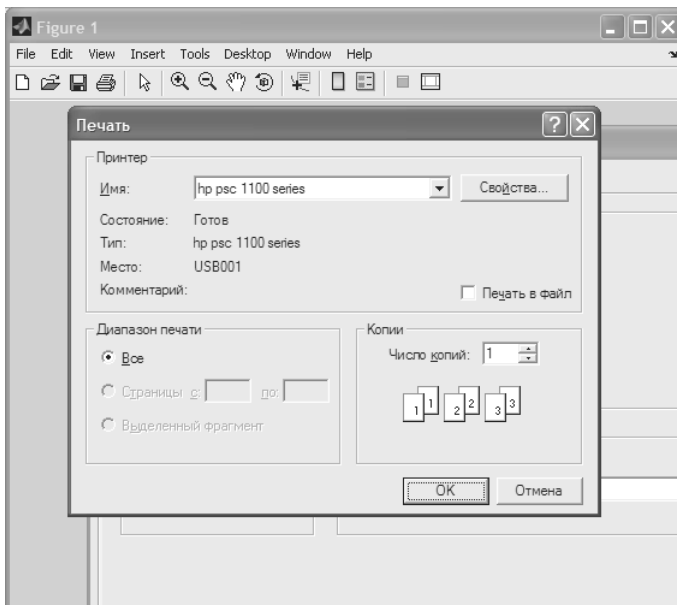
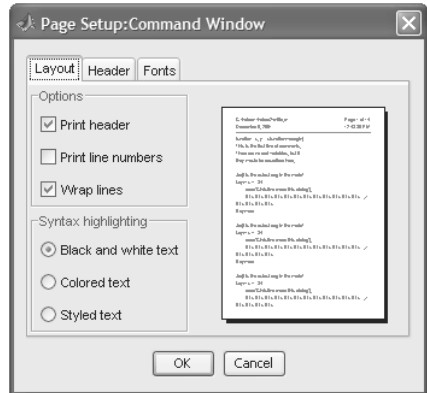


Рис. 12.82. Окна, выводимые функцией печати `printdlg`

Параметр `'-crossplatform'` позволяет вывести диалоговое окно печати на кросс-платформах Windows. Это окно аналогично окну печати (рис. 12.83), которое выводится командой **Print...** в позиции **File** меню окна системы MATLAB. На рис. 12.83 это окно показано с открытой вкладкой **General (Общие установки)**. На ней из списка задаются тип принтера для печати, пределы номеров печатаемых страниц и число печатаемых копий.

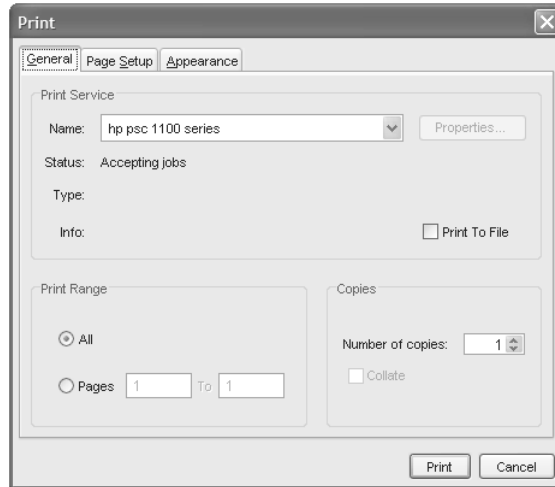


Рис. 12.83. Диалоговое окно печати в среде Windows с открытой вкладкой **General**

Вкладка **Page Setup (Установки страницы)**, показанная на рис. 12.84, содержит уже обсужденные установки. В связи с этим детальное описание этих установок не требуется.

Вкладка **Appearance (Вид – рис. 12.85)** позволяет установить цветной и монохромный вид печати, качество печати (**Draft** – черновое, **Normal** – нормальное, **High** – высокое), одностороннюю и двухстороннюю печать и другие атрибуты.

12.4.8. Другие диалоговые окна

Окно возвращения предпочтений задается следующей функцией:

```
value = uigetpref(group,pref,title,question,pref_choices)
```

С ней нетрудно разобраться с помощью следующего примера:

```
>> [selectedButton,dlgShown]=uigetpref('mygraphics',... % Group
'savefigurebeforeclosing',... % Preference
'Закрытие окна фигуры',... % Window title
{'Вы хотите записать вашу фигуру после закрытия'
''
```

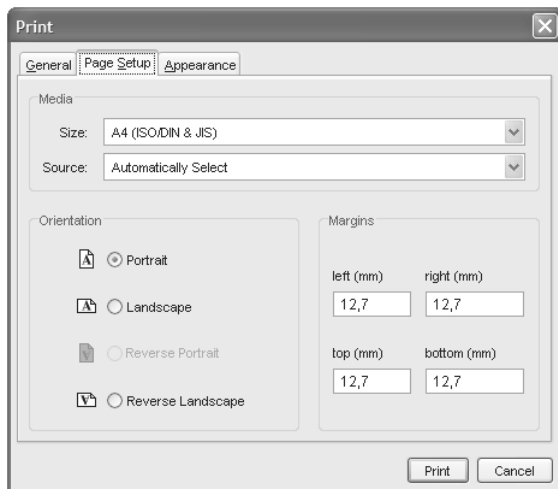


Рис. 12.84. Диалоговое окно печати в среде Windows с открытой вкладкой **Page Setup**

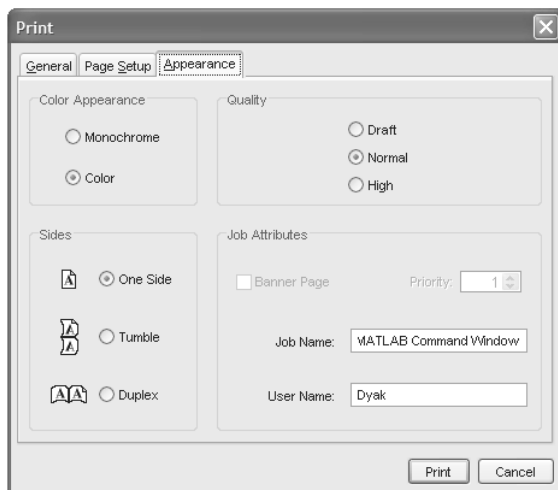


Рис. 12.85. Диалоговое окно печати в среде Windows с открытой вкладкой **Appearance**

```
'Вы можете записать вашу фигуру, нажав ''hgsave(gcf)''',...
{'always','never';'Да','Нет'},...      % Values and button strings
'ExtraOptions','Отмена',...           % Additional button
'DefaultButton','Отмена',...         % Default choice
'HelpString','Справка',...           % String for Help button
'HelpFcn','doc(''closereq'');'       % Callback
```

```
selectedButton =
always
dlgShown =
1
```

При его исполнении строится окно, показанное на рис. 12.86, и возвращаются значения переменных `selectedButton` и `dlgShown` (0 – при отказе от выбора и 1 – при сделанном выборе).

Функция `uisave` без аргументов вызывает окно файловой системы для записи в файл рабочего пространства текущего исполняемого приложения. Это окно показано на рис. 12.87.

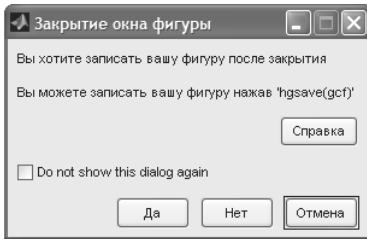


Рис. 12.86. Диалоговое окно предпочтений

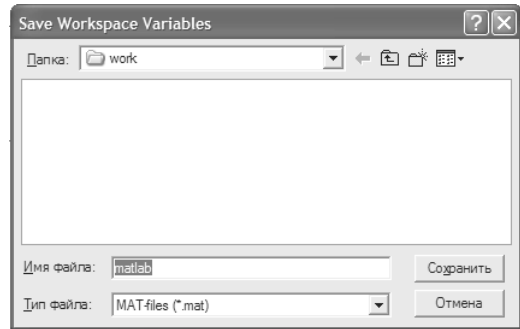


Рис. 12.87. Окно для записи в файл рабочего пространства

Функция

```
h = uicontrol('Style','Text','Position',pos);
```

позволяет точно задать границы текста `Text`, задав в параметре `pos` координаты левого нижнего и правого верхнего углов области текста. Это иллюстрирует приведенный ниже пример:

```
pos = [100 100 300 300];
h = uicontrol('Style','Text','Position',pos);552222
string = {'Для функции uicontrol ','можно корректно задать границы.'};
[outstring,newpos] = textwrap(h,string);
pos(4) = newpos(4);
set(h,'String',outstring,'Position',[pos(1),pos(2),pos(3),pos(4)])
```

Здесь использована еще одна полезная функция выравнивания текста `textwrap`. Текстовая область задана дескриптором `h`. Строящееся этим примером окно представлено на рис. 12.88.

Следующая функция строит индикатор прогресса

```
h = waitbar(x,'title')
waitbar(x,'title','CreateCancelBtn','button_callback')
```

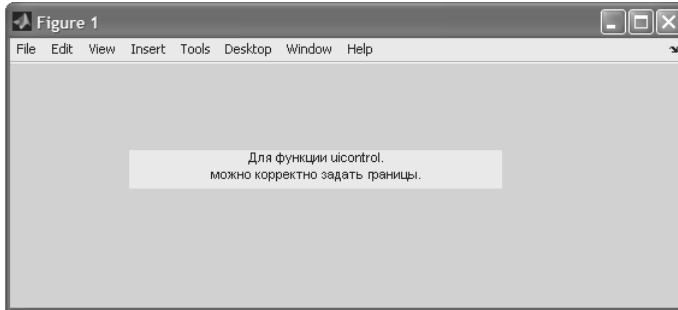


Рис. 12.88. Построение в окне текстового сообщения в заданных границах

```
waitbar(...,property_name,property_value,...)
waitbar(x)    waitbar(x,h)    waitbar(x,h,'updated title')
```

В представленном ниже примере строится индикатор прогресса, и с помощью цикла в нем задается бегущая слева направо красная полоса (рис. 12.89):

```
h = waitbar(0,'Please wait...');
for i=1:1000, waitbar(i/1000), end
close(h)
```

В этом примере число циклов (оно равно 1000) задает скорость движения полосы индикатора прогресса. Она, разумеется, зависит и от быстродействия ПК. Команда `close(h)` закрывает окно индикатора, так что индикатор рис. 12.89 является временным окном.

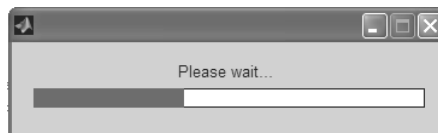


Рис. 12.89. Индикатор прогресса

Обзор расширений MATLAB

| | |
|--|-----|
| 13.1. Состав расширений MATLAB | 660 |
| 13.2. Примеры работы с Simulink | 662 |
| 13.3. Пакеты математических вычислений | 670 |
| 13.4. Пакеты анализа и синтеза систем управления | 680 |
| 13.5. Пакет идентификации систем | 687 |
| 13.6. Пакеты для обработки сигналов и изображений | 689 |
| 13.7. Прочие пакеты прикладных программ | 699 |
| 13.8. Пакеты расширения MATLAB 6.5 | 704 |
| 13.9. Новейшие пакеты расширения MATLAB 7+Simulink 6 | 709 |

Новые версии MATLAB имеют свыше 70 пакетов расширения (в новейшей версии MATLAB R2007b их число достигло 82), поставляемых только корпорацией MathWorks. В этом уроке обзорно описаны лишь наиболее важные из них и те, которые имеют очевидные перспективы для широкого применения. Некоторые узко специальные пакеты (например, ориентированные на применение специализированных интегральных микросхем или датчиков) в обзоре не отражены. Помимо фирменных пакетов расширения корпорации MathWorks, имеются сотни пакетов расширения других фирм. Многие из них доступны в Интернете.

13.1. Состав расширений MATLAB

13.1.1. Классификация расширений системы MATLAB+Simulink

Система MATLAB поставляется совместно с главным расширением Simulink, обеспечивающим визуально-ориентированную подготовку имитационных моделей систем различного назначения и выполнение их моделирования. По существу, это расширение является неотъемлемой частью системы MATLAB+Simulink, полная структура компонентов которой представлена на рис.13.1.

В соответствии с фирменными наименованиями расширения системы MATLAB входят в большой «инструментальный ящик» Toolbox, а расширения Simulink – в блок расширений Blockset. Версии расширений различны для разных версий системы MATLAB+Simulink. Представленный на рис. 13.1 набор расширений является типовым и достаточно полным. Однако для различных поставок системы MATLAB+Simulink он может существенно отличаться от приведенного.

13.1.2. Главный пакет расширения Simulink 5/6

Главное расширение системы MATLAB 6.5/7 – Simulink 5/6 служит для имитационного моделирования моделей, состоящих из графических блоков с заданными свойствами (параметрами) [9, 16, 33]. Компоненты моделей, в свою очередь, являются графическими блоками и моделями, которые содержатся в ряде библиотек и с помощью мыши могут переноситься в основное окно и соединяться друг с другом необходимыми связями. В состав моделей могут включаться источники сигналов различного вида, виртуальные регистрирующие приборы, графические средства анимации.

Для начала работы с Simulink достаточно активизировать кнопку **Simulink** в панели инструментов окна MATLAB. Откроется окно браузера библиотеки Simulink Library Browser, показанное на рис. 13.2 внутри окна системы MATLAB.

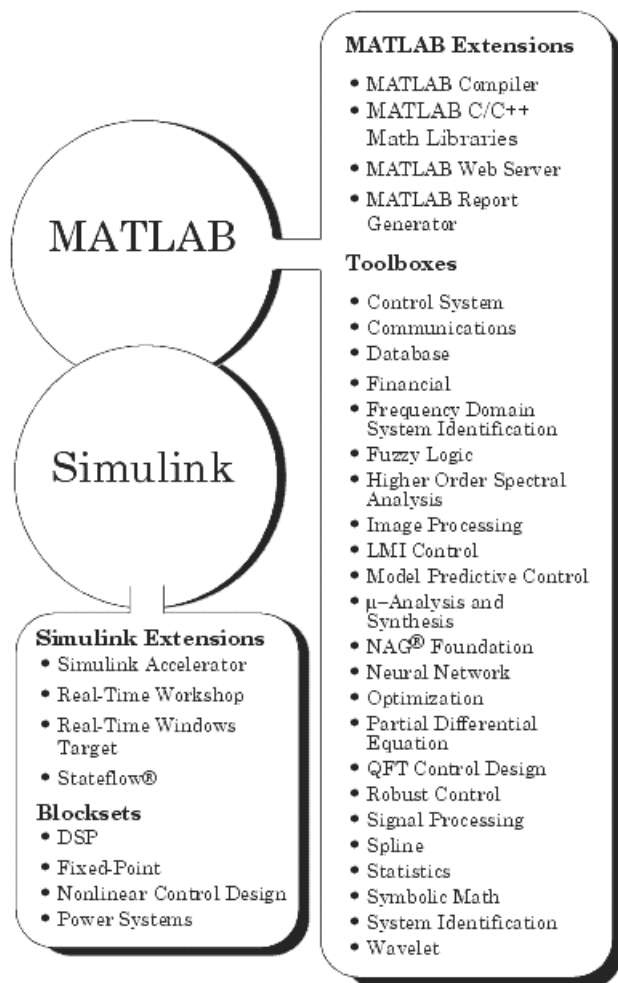


Рис. 13.1. Структура системы MATLAB+Simulink

Пакет Simulink основан на визуально-ориентированном программировании решаемых задач моделирования. Построение блочных схем (моделей) выполняется путем переноса мышью блоков из библиотеки компонентов в окно редактирования создаваемой пользователем модели и соединением блоков линиями. Затем модель запускается на выполнение (кнопкой в виде темного треугольника в панели инструментов). Интерфейс, набор блоков и организация доступа к ним в новейшей версии Simulink 7 ничем не отличаются от Simulink 6.6, входящего в MATLAB R2007a.

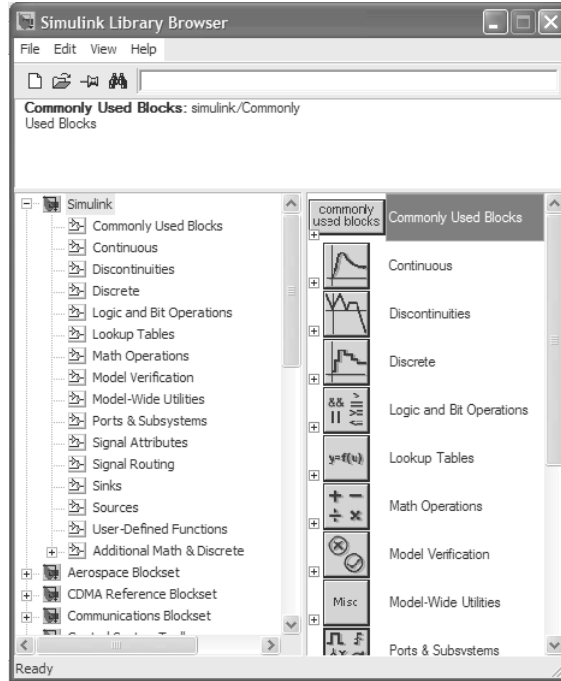


Рис. 13.2. Вызов окна библиотеки блоков Simulink (версия 6)

13.2. Примеры работы с Simulink

13.2.1. Пример моделирования системы Ван-дер-Поля

С помощью кнопки **Open** в панели инструментов MATLAB или Simulink можно вызвать обычное для Windows-приложений окно загрузки файлов и загрузить одну из моделей множества демонстрационных примеров на моделирование с помощью Simulink. На рис. 13.3 представлено окно выполнения контрольного примера на решение дифференциального уравнения Ван-дер-Поля. Это хорошо известное дифференциальное уравнение второго порядка описывает колебания в нелинейном осцилляторе второго порядка, например построенного на электронной лампе.

Двойной щелчок мышью на блоке модели выводит окно со списком его параметров, которые пользователь может менять. Запуск имитации обеспечивает математическое моделирование построенной модели с наглядным визуальным представлением результатов.

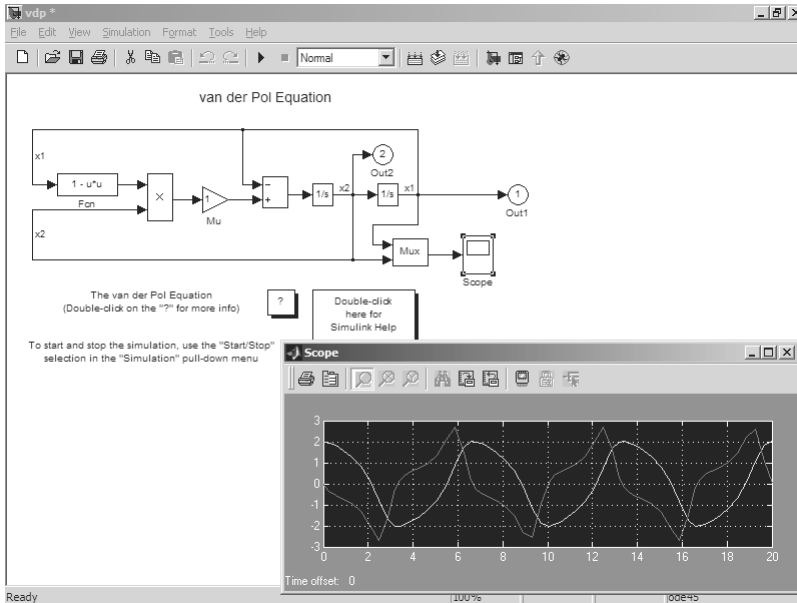


Рис. 13.3. Пример моделирования нелинейной системы Ван-дер-Поля

Одной из замечательных особенностей Simulink является возможность создания моделей из блоков, содержащих встроенные подмодели (субблоки). Рисунок 13.4 дает пример на использование этой возможности, входящий в число демонстрационных примеров Simulink.

В этом примере в основной модели используются два субблока для выполнения операций вычисления абсолютного значения сигнала Abs и задания его ограничения Saturation. Субблоки представлены над схемой основной модели. Они отличаются от основной модели только наличием портов ввода In и вывода Out. С их помощью субблоки подключаются к основной модели. Справа показаны осциллограммы работы модели после ее запуска. Работа модели вполне очевидна.

13.2.2. Nonlinear Control Design Blockset

Пакет Nonlinear Control Design (NCD) Blockset в MATLAB 6.* реализует метод динамической оптимизации для проектирования систем управления. Этот инструмент, разработанный для использования совместно с пакетом Simulink, автоматически настраивает системные параметры, основываясь на определенных пользователем ограничениях на временные характеристики.

Рисунок 13.5 показывает пример работы с этим пакетом. В данном случае вычисляются параметры интегрирующе-дифференцирующего устройства (PID-регулятора), переходная характеристика которого имеет колебательный характер и перед оптимизацией явно выходит за пределы заданных ограничений. При пуске

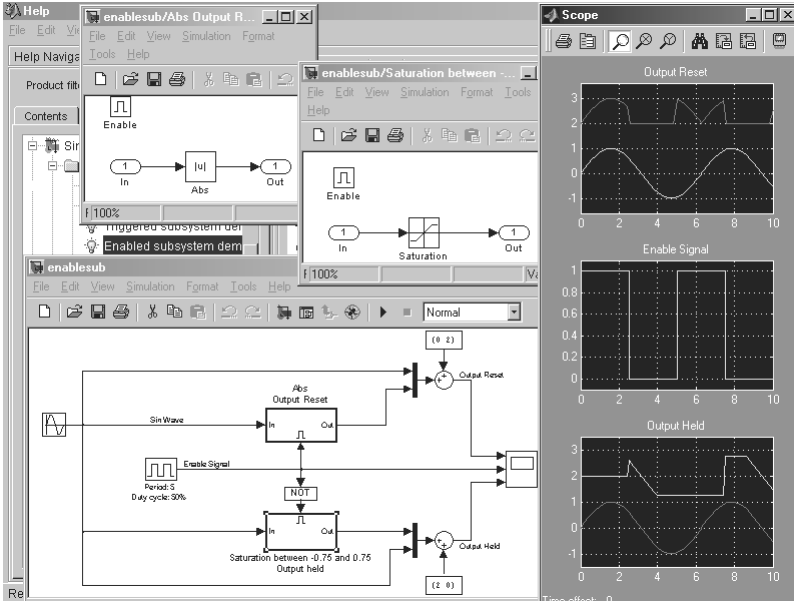


Рис. 13.4. Пример моделирования для модели с субблоками

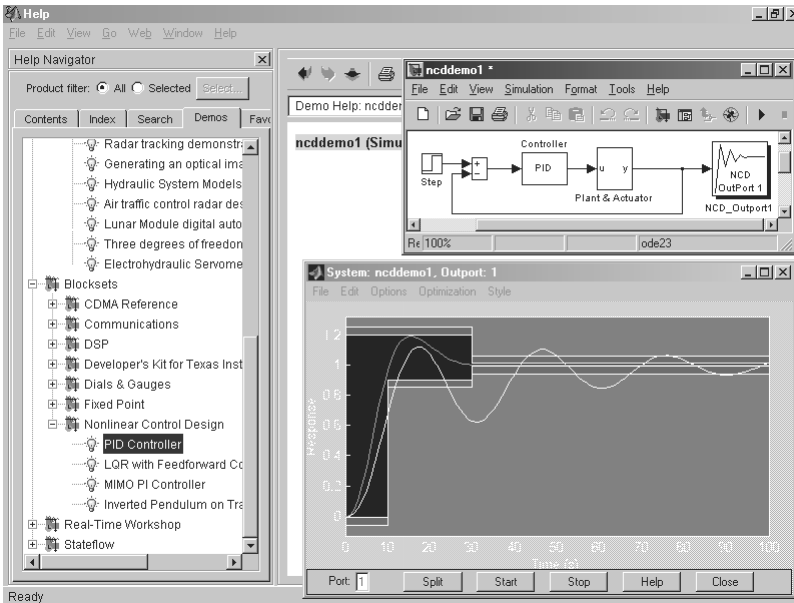


Рис. 13.5. Пример моделирования PID-регулятора с помощью пакетов Simulink и NCD

модели параметры PID-регулятора автоматически меняются так, чтобы его переходная характеристика попала в заданные области ограничений. Исходная и оптимизированная переходные характеристики представлены на рис. 13.5.

Пакет NCD использует перенос объектов мышью для изменения временных ограничений прямо на графиках, что позволяет легко настраивать переменные и указывать неопределенные параметры, обеспечивает интерактивную оптимизацию, реализует моделирование методом Монте-Карло, поддерживает проектирование SISO- (один вход – один выход) и MIMO-систем управления, позволяет моделировать подавление помех, слежение и другие типы откликов, поддерживает проблемы повторяющегося параметра и задачи управления системами с запаздыванием, позволяет осуществлять выбор между удовлетворенными и недостижимыми ограничениями. В MATLAB 7.* пакет переименован в Simulink Response Optimization.

13.2.3. Digital Signal Processing (DSP) Blockset

Digital Signal Processing (DSP) – пакет прикладных программ для проектирования устройств, использующих процессоры цифровой обработки сигналов [12, 15, 18, 36]. Это прежде всего высокоэффективные цифровые фильтры с заданной или адаптируемой к параметрам сигналов частотной характеристикой (АЧХ). В частности, это могут быть фильтры с конечной и бесконечной импульсной характеристиками.

В качестве примера применения пакета DSP на рис. 13.6 представлен пример осуществления короткого (именуемого также оконным) быстрого преобразования Фурье. Этот вид преобразований позволяет анализировать нестационарные сигналы за счет вычисления спектра в разных временных окнах. Благодаря этому можно строить сложные спектры в области «частота–время», которые обеспечивают детальный анализ особенностей сигналов, недоступный для обычного быстрого преобразования Фурье.

Альтернативой короткому быстрому преобразованию Фурье в наше время являются вейвлет-преобразования, которые предполагают разложение произвольного сигнала по новому базису «коротких волн» [41]. Он представлен совокупностью коротких волновых пакетов – вейвлетов, которые способны перемещаться по времени и масштабироваться (сжиматься и расширяться по оси времени t или по оси x). Ограниченный набор средств вейвлет-преобразований включен и в пакет DSP.

Рисунок 13.7 показывает технику осуществления дискретных диадических быстрых вейвлет-преобразований на основе алгоритма Малла, реализованного в частотной области с помощью банков вейвлет-фильтров. Этот пример иллюстрирует прохождение явно нестационарного сигнала (типа «визг»), представляющего собой амплитудно- и частотно-модулированную синусоиду с шумовой компонентой, через фильтры, задающие вначале прямое, а затем обратное вейв-

лет-преобразование. В результате этого сигнал восстанавливается абсолютно точно. На это указывает полное соответствие графиков сигналов на входе и выходе, а также нулевое значение погрешности для любого значения времени. Таким образом, в данном случае осуществляется разложение сигнала и затем его точная реставрация.

Куда более обширной реализации вейвлет-преобразований посвящен пакет расширения Wavelet Toolbox [41].

13.2.4. Пакет расширения *Fixed-Point Blockset*

Этот специальный пакет ориентирован на моделирование цифровых систем управления и цифровых фильтров в составе пакета Simulink. Специальный набор компонентов позволяет быстро переключаться между вычислениями с фиксированной и плавающей запятой (точкой). Можно указывать 8-, 16- или 32-битовую длину слова. Пакет обладает рядом полезных свойств:

- применение беззнаковой или двоичной арифметики;
- выбор пользователем положения двоичной точки;
- автоматическая установка положения двоичной точки;
- просмотр максимального и минимального диапазонов сигнала модели;
- переключение между вычислениями с фиксированной и плавающей точкой;
- коррекция переполнения;
- наличие ключевых компонентов для операций с фиксированной точкой, включая сложение, вычитание и умножение, запаздывание, суммирование;
- логические операторы, одно- и двумерные справочные таблицы.

Основное назначение этого пакета – создание оптимизированных фильтров и устройств, в максимальной степени использующих аппаратные возможности ПК.

13.2.5. Пакет расширения *Stateflow*

Stateflow – пакет моделирования событийно-управляемых систем, основанный на теории конечных автоматов. Этот пакет предназначен для использования вместе с пакетом моделирования динамических систем Simulink и придает ему качественно новые свойства – возможность моделирования событийно-управляемых систем и систем с переменной структурой. В любую Simulink-модель можно вставить Stateflow-диаграмму (или SF-диаграмму), которая будет отражать поведение компонентов объекта (или системы) моделирования.

SF-диаграмма является анимационной. По ее выделяющимся цветом блокам и связям можно проследить все стадии работы моделируемой системы или устройства и поставить его работу в зависимость от тех или иных событий. Рисунок 13.8 иллюстрирует моделирование поведения автомобиля при возникновении чрезвычайного обстоятельства на дороге. Под моделью автомобиля видна SF-диаграмма (точнее, один кадр ее работы).

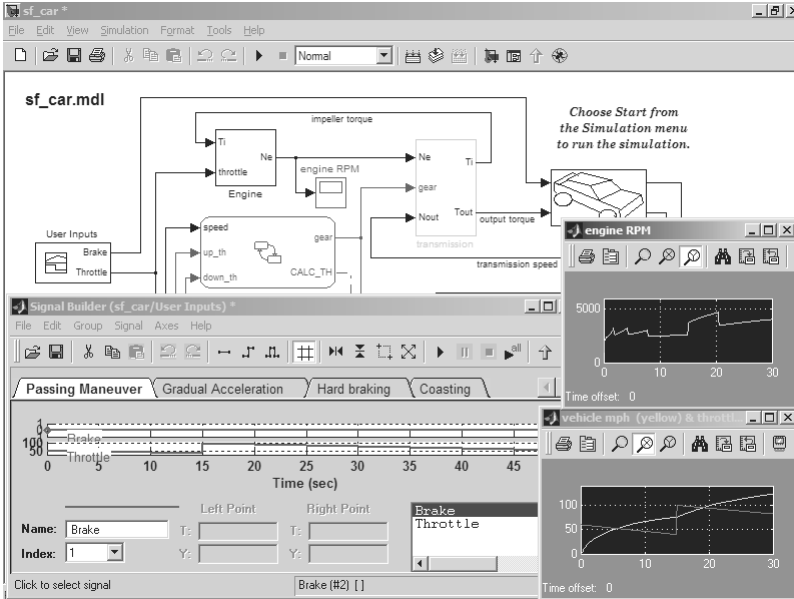


Рис. 13.8. Моделирование поведения автомобиля с применением SF-диаграммы

Пакет Stateflow значительно расширяет возможности моделирования сложных систем, содержащих специальные типы данных, например динамически изменяющиеся очереди. Кстати, построение такой очереди видно сверху SF-диаграммы, приведенной на рис. 13.8.

Для создания SF-диаграмм пакет имеет удобный и простой редактор, а также средства пользовательского интерфейса. К сожалению, в освоении этот пакет довольно сложен, и даже простые SF-диаграммы требуют для их создания много времени. Отчасти это компенсируется большим числом демонстрационных примеров по этому пакету.

13.2.6. Пакет расширения SimPower System

SimPower System (ранее это был Power System Blockset) – новый пакет моделирования мощных энергетических (в основном электротехнических) систем, таких как линии передачи, силовые ключи, регуляторы напряжения и тока, устройства управления электродвигателями различного типа и нагревательными системами, ветряные электростанции и др. Пакет органично связан с расширением Simulink, имеет свою библиотеку компонентов и позволяет проектировать и моделировать мощные устройства на уровне их функциональных и даже принципиальных электрических схем – см. пример на рис. 13.9.

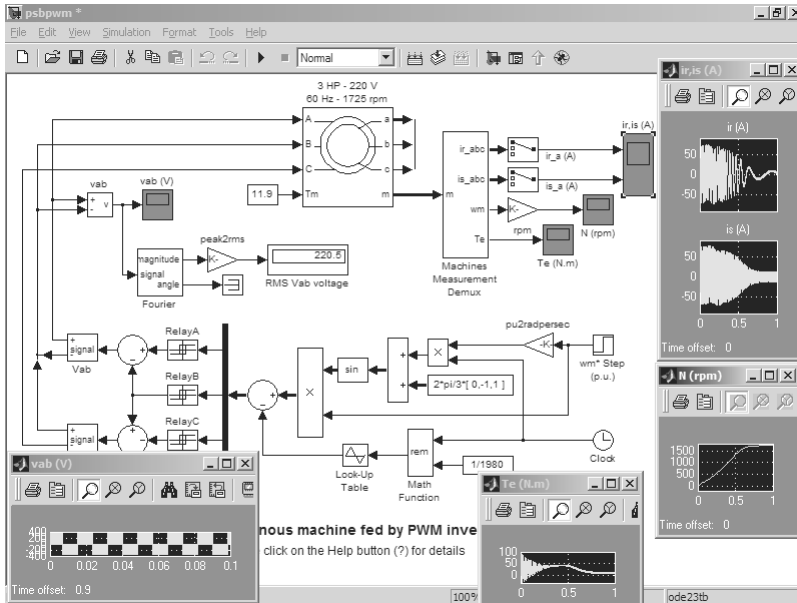


Рис. 13.9. Пример моделирования асинхронной электрической машины

Этот пакет обеспечивает моделирование широкого спектра энергетических систем и устройств – начиная с анализа простейших электрических цепей и заканчивая моделированием сложных преобразовательных устройств и даже целых электрических систем. Результаты моделирования отображаются разнообразными виртуальными измерительными приборами, такими как графопостроители, осциллографы и др.

13.2.7. Report Generator для MATLAB и Simulink

Генераторы отчетов – средство, введенное еще в MATLAB 5.3.1, дает информацию о работе системы MATLAB и пакета расширения Simulink. Это средство очень полезно при отладке сложных вычислительных алгоритмов или при моделировании сложных систем. Генераторы отчетов запускаются командой Report. Отчеты могут быть представлены в виде программ и редактироваться – см. пример на рис. 13.10, где показано окно редактирования отчета работы Simulink.

Генераторы отчетов могут запускать входящие в отчеты команды и фрагменты программ и позволяют проконтролировать поведение сложных вычислений.

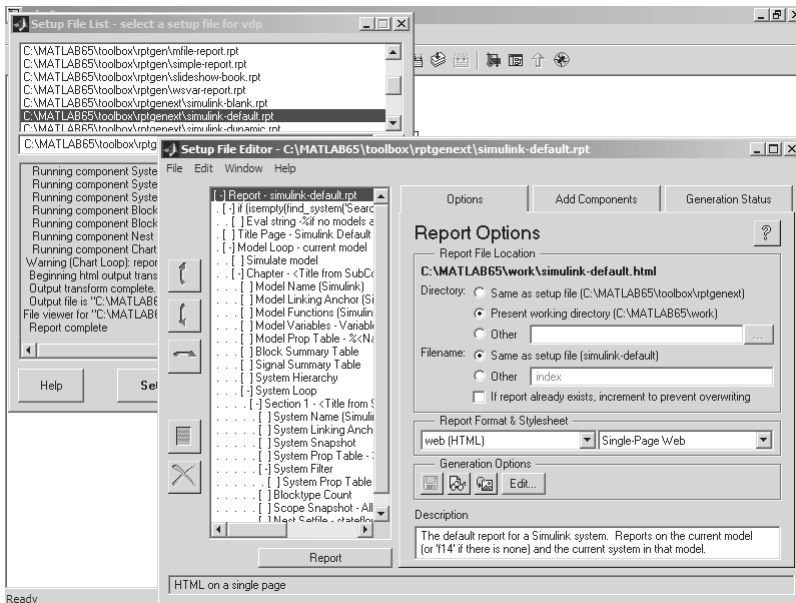


Рис. 13.10. Пример работы с отчетом

13.2.8. Real Time Windows Target и WorkShop

Подключающаяся к Simulink мощная подсистема имитационного моделирования в реальном масштабе времени (при наличии дополнительных аппаратных средств в виде плат расширения компьютера), представленная пакетами расширения Real Time Windows Target и WorkShop, – мощное средство управления реальными объектами и системами. Кроме того, эти расширения позволяют создавать исполняемые коды моделей.

Достоинством такого моделирования является его математическая и физическая наглядность. В компонентах моделей Simulink можно задавать не только фиксированные параметры, но и математические соотношения, описывающие поведение моделей. Однако надо учитывать, что данные расширения являются очень дорогими и нужны лишь при использовании MATLAB в качестве управляющей программы для внешних устройств, например таких, как роботы, самоходные средства и др.

13.3. Пакеты математических вычислений

В MATLAB входит множество пакетов расширения, существенно усиливающих математические возможности системы, повышающих скорость, эффективность и точность математических вычислений.

13.3.1. Symbolic Math Toolbox

Пакет прикладных программ Symbolic Math Toolbox дает системе MATLAB принципиально новые возможности решения задач в символьном (аналитическом) виде, включая реализацию точной арифметики произвольной разрядности. Пакет базируется на применении ядра символьной математики одной из самых мощных систем компьютерной алгебры – Maple. Обеспечивает выполнение символьного дифференцирования и интегрирования, вычисление сумм и произведений, разложение в ряды Тейлора и Маклорена, операции со степенными многочленами (полиномами), вычисление корней полиномов, решение в аналитическом виде нелинейных уравнений, всевозможные символьные преобразования, подстановки и многое другое. Имеет команды прямого доступа к ядру системы.

Для работы с пакетом надо задать неопределенные символьные переменные. Они объявляются с помощью команды `syms`, например

```
>> syms x n
```

или заключением в апострофы. Ниже представлены примеры на вычисление производной, неопределенного интеграла и разложения в ряд Тейлора (Маклорена):

```
>> diff(n*sin(x), x)
ans = n*cos(x)
>> int(x^n, x)
ans = x^(n+1)/(n+1)
>> taylor(sin(x), 6)
ans = x-1/6*x^3+1/120*x^5
```

К сожалению, символьные выражения вводятся и выводятся в строчном виде, далеко от различных версий пакета расширения Symbolic Math Toolbox, их можно найти в [5, 10, 14, 16].

Пакет позволяет готовить процедуры с синтаксисом языка программирования системы Maple и устанавливать их в системе MATLAB. По возможностям символьной математики пакет все же сильно уступает специализированным системам компьютерной алгебры, таким как новейшие версии Maple и Mathematica.

13.3.2. NAG Foundation Toolbox

NAG Foundation Toolbox – одна из самых мощных библиотек математических функций, созданная специальной группой The Numerical Algorithms Group, Ltd. Пакет содержит сотни новых функций. Названия функций и синтаксис их вызова заимствованы из известной библиотеки NAG Foundation Library. Вследствие этого опытные пользователи NAG Fortran могут без затруднений работать с пакетом NAG в MATLAB. Библиотека NAG Foundation предоставляет свои функции в виде объектных кодов и соответствующих m-файлов для их вызова. Пользователь может легко модифицировать эти MEX-файлы на уровне исходного кода.

Пакет обеспечивает следующие возможности:

- корни многочленов и модифицированный метод Лагерра;

- вычисление суммы ряда: дискретное и эрмитово-дискретное преобразования Фурье;
- обыкновенные дифференциальные уравнения: методы Адамса и Рунге-Кутты;
- уравнения в частных производных;
- интерполяция;
- вычисление собственных значений и векторов, сингулярных чисел, поддержка комплексных и действительных матриц;
- аппроксимация кривых и поверхностей: полиномы, кубические сплайны, полиномы Чебышева;
- минимизация и максимизация функций: линейное и квадратичное программирование, экстремумы функций нескольких переменных;
- разложение матриц;
- решение систем линейных уравнений;
- линейные уравнения (LAPACK);
- статистические расчеты, включая описательную статистику и распределения вероятностей;
- корреляционный и регрессионный анализ: линейные, многомерные и обобщенные линейные модели;
- многомерные методы: главных компонент, ортогональные вращения;
- генерация случайных чисел: нормальное распределение, распределения Пуассона, Вейбулла и Коши;
- непараметрические статистики: Фридмана, Крускала-Уоллиса, Манна-Уитни;
- временные ряды: одномерные и многомерные;
- аппроксимации специальных функций: интегральная экспонента, гамма-функция, функции Бесселя и Ганкеля.

Наконец, этот пакет позволяет пользователю создавать программы на языке Fortran, которые динамически линкуются с MATLAB.

13.3.3. Spline Toolbox

Это пакет прикладных программ для работы со сплайнами. Поддерживает одномерную, двумерную и многомерную сплайн-интерполяцию и аппроксимацию. Обеспечивает представление и отображение сложных данных и поддержку графики. Пакет позволяет выполнять интерполяцию, аппроксимацию и преобразование сплайнов из В-формы в кусочно-полиномиальную, интерполяцию кубическими сплайнами и сглаживание, выполнение операций над сплайнами: вычисление производной, интеграла и отображение поверхностей (см. пример на рис. 13.11).

Пакет Spline оснащен программами работы с В-сплайнами, описанными в работе «A Practical Guide to Splines» Карлом Дебуром, создателем сплайнов и автором пакета Spline. Функции пакета в сочетании с языком MATLAB и подробным

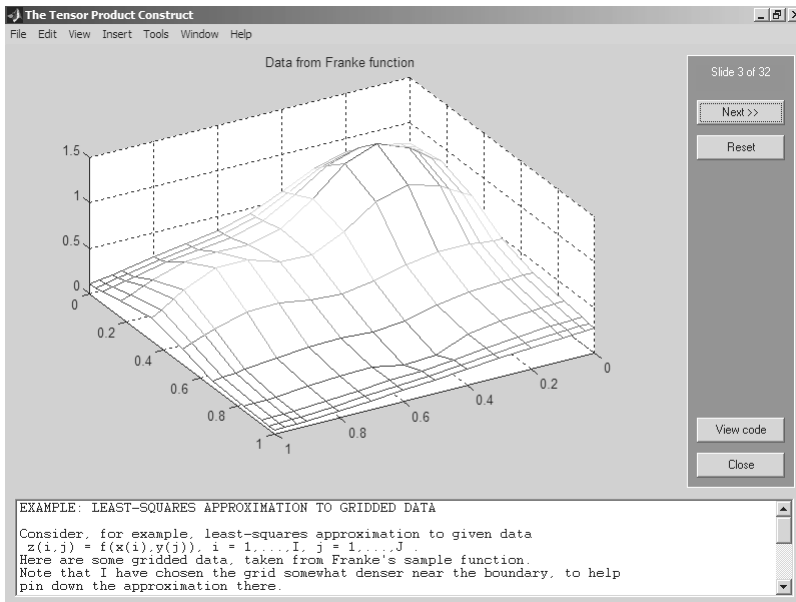


Рис. 13.11. Пример применения сплайновой интерполяции для поверхностей

руководством пользователя облегчают понимание сплайнов и их эффективное применение к решению разнообразных задач.

В пакет включены программы для работы с двумя наиболее широко распространенными формами представления сплайнов: В-формой и кусочно-полиномиальной формой. В-форма удобна на этапе построения сплайнов, в то время как кусочно-полиномиальная форма более эффективна во время постоянной работы со сплайном. Пакет включает функции для создания, отображения, интерполяции, аппроксимации и обработки сплайнов в В-форме и в виде отрезков полиномов.

13.3.4. Statistics Toolbox

Пакет прикладных программ по статистике, резко расширяющий возможности системы MATLAB в области реализации статистических вычислений и статистической обработки данных. Содержит весьма представительный набор средств генерации случайных чисел, векторов, матриц и массивов с различными законами распределения, а также множество статистических функций. Следует отметить, что наиболее распространенные статистические функции входят в состав ядра системы MATLAB (в том числе функции генерации случайных данных с равномерным и нормальным распределением). Основные возможности пакета:

- описательная статистика;
- распределения вероятностей;
- оценка параметров и аппроксимация;
- проверка гипотез;
- множественная регрессия;
- интерактивная пошаговая регрессия;
- моделирование Монте-Карло;
- аппроксимация на интервалах;
- статистическое управление процессами;
- планирование эксперимента;
- моделирование поверхности отклика;
- аппроксимация нелинейной модели;
- анализ главных компонент;
- статистические графики;
- графический интерфейс пользователя.

Пакет включает 20 различных распределений вероятностей, включая t (Стьюдента), F и Хи-квадрат. Подбор параметров, графическое отображение распределений и способ вычисления лучших аппроксимаций предоставляются для всех типов распределений. Предусмотрено множество интерактивных инструментов для динамической визуализации и анализа данных. Имеются специализированные интерфейсы для моделирования поверхности отклика, визуализации распределений, генерации случайных чисел и линий уровня.

13.3.5. Optimization Toolbox

Пакет прикладных задач для решения оптимизационных задач и систем нелинейных уравнений. Поддерживает основные методы оптимизации функций ряда переменных:

- безусловная оптимизация нелинейных функций;
- метод наименьших квадратов и нелинейная интерполяция;
- решение нелинейных уравнений;
- линейное программирование;
- квадратичное программирование (рис. 13.12);
- условная минимизация нелинейных функций;
- метод минимакса;
- многокритериальная оптимизация.

Разнообразные примеры (рис. 13.12 – лишь один из них) демонстрируют эффективное применение функций пакета. С их помощью также можно сравнить, как одна и та же задача решается разными методами.

Новый пакет Genetic Algorithm and Direct Search Toolbox дает инструментальные средства для решения оптимизационных задач с применением генетических алгоритмов и новых алгоритмов прямого поиска, позволяющих уверенно отыскивать глобальные и локальные минимумы функций ряда переменных, решать задачи кластеризации и т. д.

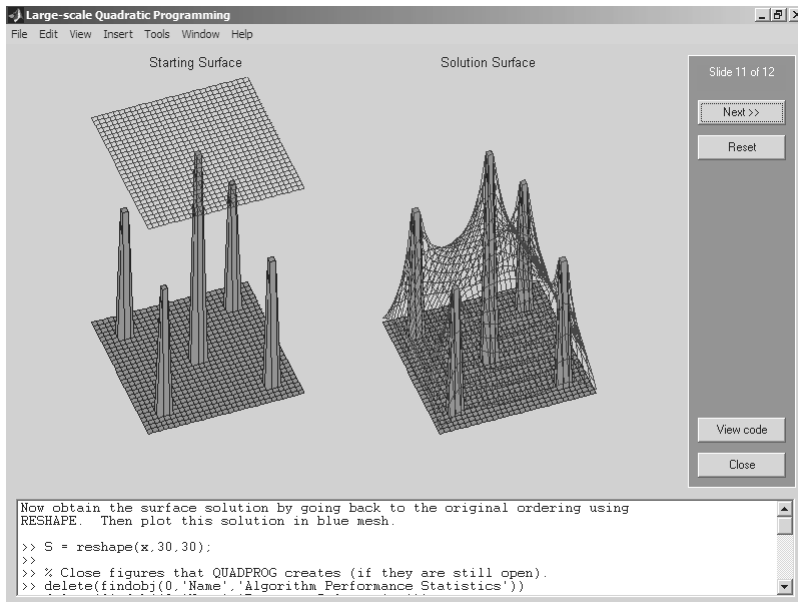


Рис. 13.12. Пример оптимизации поверхности методом квадратичного программирования

13.3.6. Partial Differential Equations Toolbox

Весьма важный пакет прикладных программ, содержащий множество функций для решения систем дифференциальных уравнений в частных производных. Дает эффективные средства для решения таких систем уравнений, в том числе жестких. В пакете используется метод конечных элементов. Команды и графический интерфейс пакета могут быть использованы для математического моделирования уравнений в частных производных применительно к широкому классу инженерных и научных приложений, включая задачи сопротивления материалов, расчеты электромагнитных устройств, задачи тепломассопереноса и диффузии. Основные возможности пакета:

- полноценный графический интерфейс для обработки уравнений с частными производными второго порядка;
- автоматический и адаптивный выбор сетки;
- задание граничных условий: Дирихле, Неймана и смешанных;
- гибкая постановка задачи с использованием синтаксиса MATLAB;
- полностью автоматическое сеточное разбиение и выбор величины конечных элементов;
- нелинейные и адаптивные расчетные схемы;
- возможность визуализации полей различных параметров и функций решения, демонстрация принятого разбиения и анимационные эффекты.

Пакет интуитивно следует шести шагам решения PDE с помощью метода конечных элементов. Эти шаги и соответствующие режимы пакета таковы: определение геометрии (режим рисования), задание граничных условий (режим граничных условий), выбор коэффициентов, определяющих задачу (режим PDE), дискретизация конечных элементов (режим сетки), задание начальных условий и решение уравнений (режим решения), последующая обработка решения (режим графика).

В разделе демонстрационных примеров **Demos** справки системы MATLAB имеется раздел, посвященный применению пакета PDE в командном режиме работы. При его активизации открывается окно **GUI** (рис. 13.13) с восемью примерами применения этого пакета (см. также описание двух этих примеров в уроке 8).

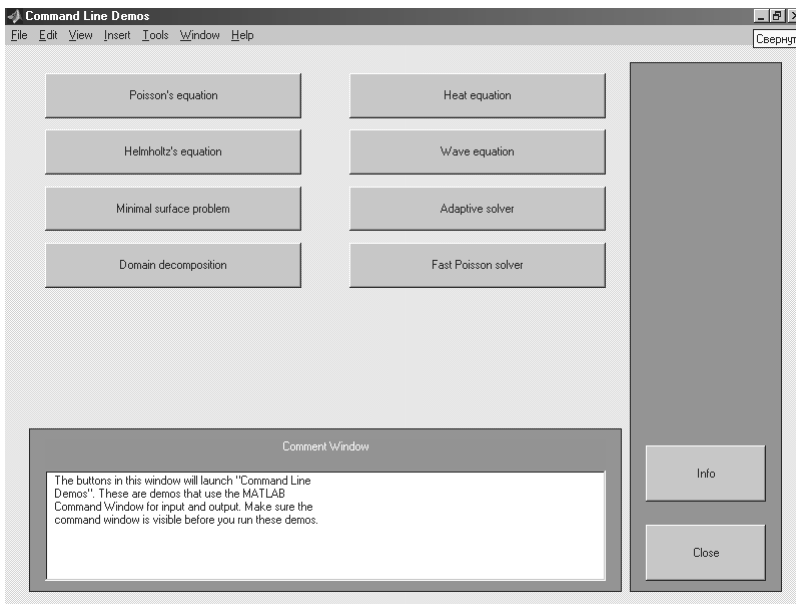


Рис. 13.13. Окно демонстрационных примеров пакета PDE

Активизация мышью кнопки одного из примеров запускает его в работу из командной строки. Нажимая любую клавишу, можно наблюдать отдельные стадии выполнения примера и просматривать фрагменты реализующей вычисления программы в окне командного режима MATLAB – рис. 13.14.

В этом примере можно наблюдать анимационную картину изменения поверхности типа «шляпа», реализованную средствами пакета PDE.

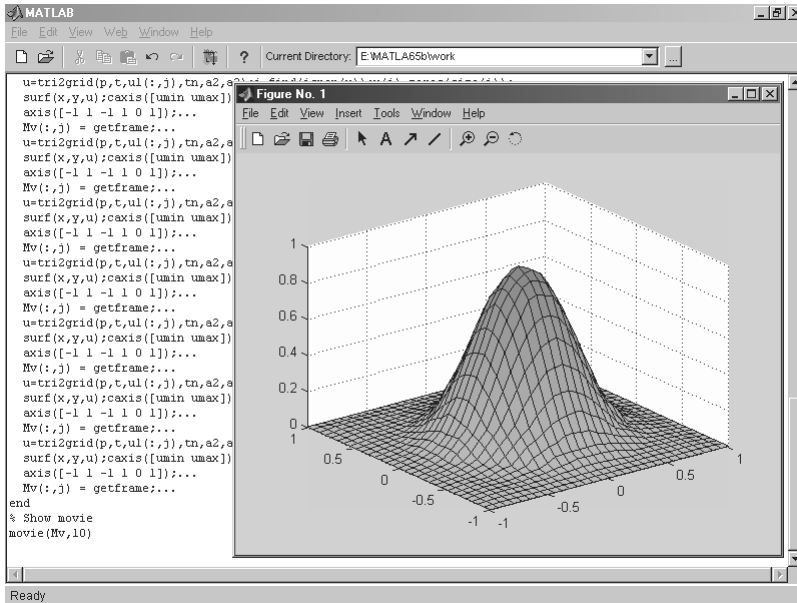


Рис. 13.14. Пример на анимацию поверхности типа «шляпа» (Head equation) пакета PDE

13.3.7. Fuzzy Logic Toolbox

Пакет прикладных программ Fuzzy Logic относится к теории нечетких (размытых) множеств [10, 43]. Это новое перспективное направление математической логики. Пакетом обеспечивается поддержка современных методов нечеткой кластеризации и построения адаптивных нечетких нейронных сетей [10, 42]. Графические средства пакета позволяют интерактивно отслеживать особенности поведения системы. Основные возможности пакета:

- определение переменных, нечетких правил и функций принадлежности;
- интерактивный просмотр нечеткого логического вывода;
- современные методы: адаптивный нечеткий вывод с использованием нейронных сетей, нечеткая кластеризация;
- интерактивное динамическое моделирование в Simulink;
- генерация переносимого С-кода с помощью Real-Time Workshop.

Рисунок 13.15 показывает применение пакета Fuzzy Logic для получения мультипликационного фильма. Моделируется поведение упругого шарика, перекатывающегося по доске, установленной на клине. Когда шарик оказывается

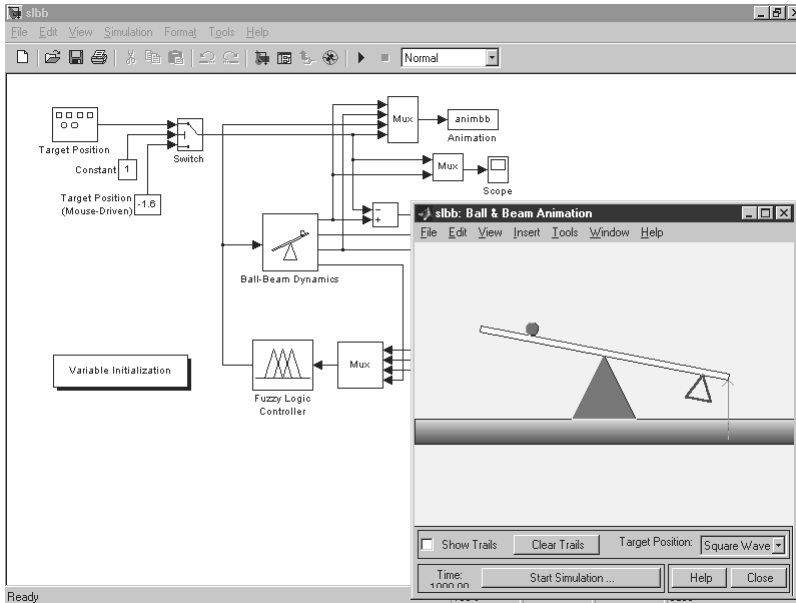


Рис. 13.15. Пример работы с пакетом нечетких множеств

у опоры, положение системы становится неопределенным и применение правил нечеткой логики позволяет смоделировать корректный выход из такого положения.

Этот пример наглядно показывает отличия в поведении модели при учете нечеткой логики и без такого учета.

13.3.8. Neural Networks Toolbox

Пакет прикладных программ, содержащих средства для построения нейронных сетей, базирующихся на поведении математического аналога нейрона [10, 42]. Пакет обеспечивает эффективную поддержку проектирования, обучения и моделирования множества известных сетевых парадигм, от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для исследования и применения нейронных сетей к таким задачам, как обработка сигналов, нелинейное управление и финансовое моделирование. Обеспечена возможность генерации переносимого С-кода с помощью Real Time Workshop.

В пакет включены более полутора десятков известных типов сетей и обучающих правил, позволяющих пользователю выбирать наиболее подходящую для конкрет-

ного приложения или исследовательской задачи парадигму. Для каждого типа архитектуры и обучающих правил имеются функции инициализации, обучения, адаптации, создания и моделирования, демонстрации и пример приложения сети.

Для управляемых сетей можно выбрать прямую или рекуррентную архитектуру, используя множество обучающих правил и методов проектирования, таких как персептрон, обратное распространение, обратное распространение Левенберга, сети с радиальным базисом и рекуррентные сети. Вы можете легко изменять любые архитектуры, обучающие правила или переходные функции, добавлять новые, и все это без написания хоть единой строки на С или ФОРТРАН.

На рис. 13.16 показан пример на построение нейронных сетей различного вида, отличающихся их передаточной характеристикой. Меню в нижнем левом углу окна этого примера задает выбор типа сети. С помощью ползунковых регуляторов над этим меню можно менять параметры передаточной характеристики.

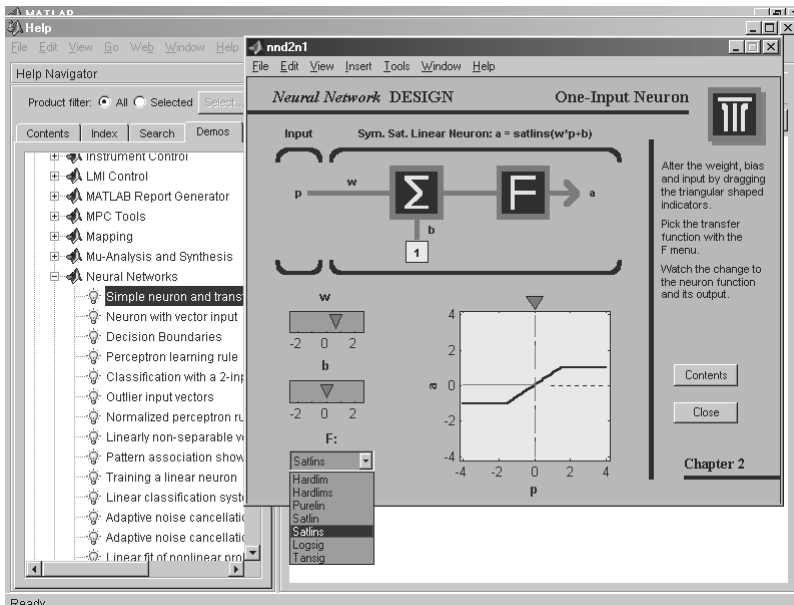


Рис. 13.16. Пример работы с пакетом нейронных сетей

В новых реализациях MATLAB этот пакет существенно переработан и дополнен. Это относится, кстати, и к демонстрационным примерам пакета по нейронным сетям.

13.4. Пакеты анализа и синтеза систем управления

13.4.1. Control System Toolbox

Пакет Control System предназначен для моделирования, анализа и проектирования систем автоматического управления – как непрерывных, так и дискретных [11, 14, 17, 35]. Функции пакета реализуют традиционные методы передаточных функций и современные методы пространства состояний. Частотные и временные отклики, диаграммы расположения нулей и полюсов могут быть быстро вычислены и отображены на экране. В пакете реализованы:

- полный набор средств для анализа ММО-систем (множество входов – множество выходов);
- временные характеристики: передаточная и переходная функции, реакция на произвольное воздействие;
- частотные характеристики: диаграммы Боде, Николса, Найквиста и др.;
- разработка обратных связей;
- проектирование LQR/LQE-регуляторов;
- характеристики моделей: управляемость, наблюдаемость, понижение порядка моделей;
- поддержка систем с запаздыванием.

Дополнительные функции позволяют конструировать более сложные модели. Временной отклик может быть рассчитан для импульсного входа, единичного скачка или произвольного входного сигнала. Имеются также функции для анализа сингулярных чисел.

Интерактивная среда для сравнения временного и частотного отклика систем предоставляет пользователю графические управляющие элементы для одновременного отображения откликов и переключения между ними. Можно вычислять различные характеристики откликов, такие как время разгона и время регулирования.

Возможности и интерфейс пакета можно проиллюстрировать на одном из примеров – управления двигателем постоянного тока (Control DC Motor) – рис. 13.17.

Нажимая клавишу **Next** окна интерфейса данного примера, можно просмотреть различные варианты блок-схем систем управления двигателем постоянного тока и соответствующие их характеристики. На рис. 13.18 показан пример просмотра одной из таких блок-схем системы управления, под которой представлены ее параметры.

На рис. 13.19 представлен конечный кадр данного примера. Он демонстрирует временные зависимости основных параметров системы управления при динамическом изменении нагрузки двигателя.

Пакет Control System содержит средства для выбора параметров обратной связи. Реализованы различные методы анализа систем: анализ особых точек, определение коэффициента усиления и затухания, линейно-квадратичное регулирова-

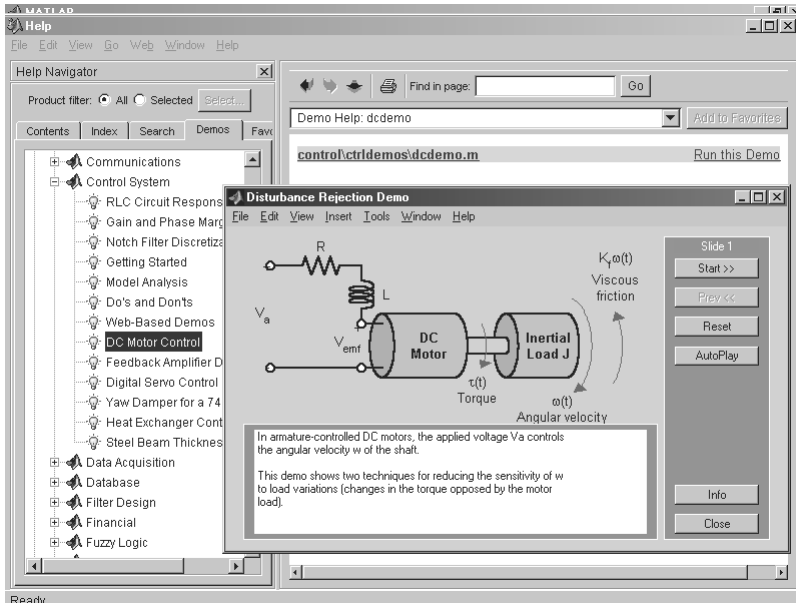


Рис. 13.17. Пример моделирования системы управления двигателем постоянного тока

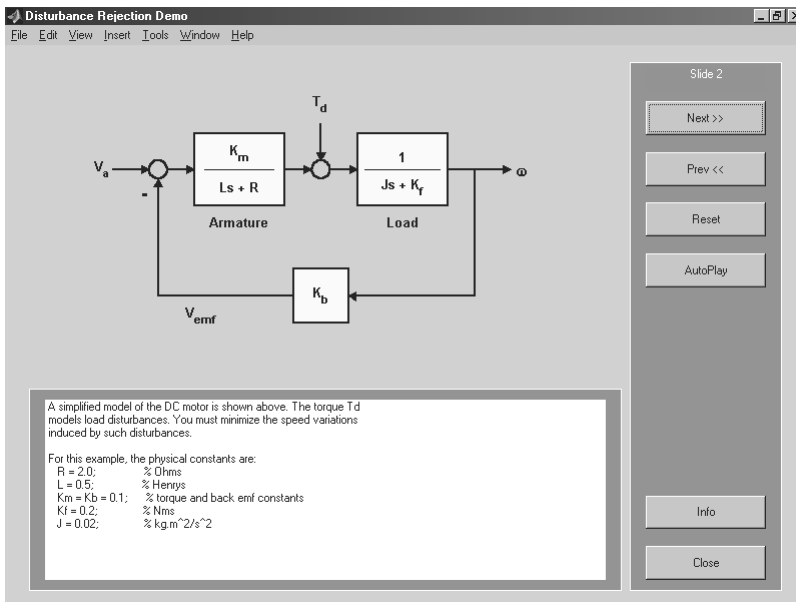


Рис. 13.18. Блок-схема одной из систем управления двигателем постоянного тока

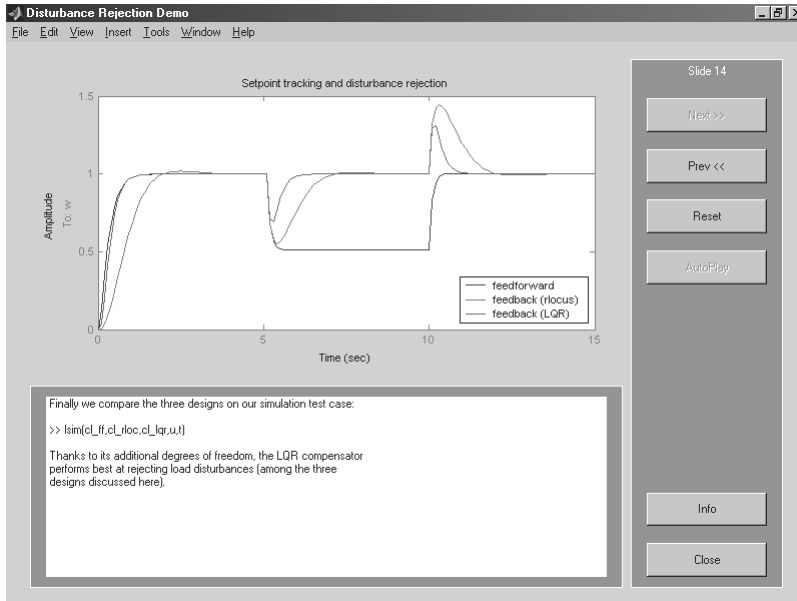


Рис. 13.19. Конечный кадр моделирования системы управления двигателем постоянного тока

ние и др. Пакет Control System включает большое количество алгоритмов для проектирования и анализа систем управления. Кроме того, он обладает настраиваемым окружением и позволяет создавать свои собственные m-файлы.

13.4.2. Robust Control Toolbox

Пакет Robust Control включает средства для проектирования и анализа многопараметрических устойчивых (робастных) систем управления. Это системы с ошибками моделирования, динамика которых известна не полностью или параметры которых могут изменяться в ходе моделирования. Мощные алгоритмы пакета позволяют выполнять сложные вычисления с учетом изменения множества параметров. Панель демонстрационных примеров этого пакета, выполняемых в командном режиме, представлена на рис. 13.20. Несколько примеров даны и на основе использования средств GUI этого пакета.

Возможности пакета Robust Control:

- синтез LQG-регуляторов на основе минимизации равномерной и интегральной нормы (рис. 13.21);
- многопараметрический частотный отклик;
- построение модели пространства состояний;
- преобразование моделей на основе сингулярных чисел;
- понижение порядка модели;
- спектральная факторизация.

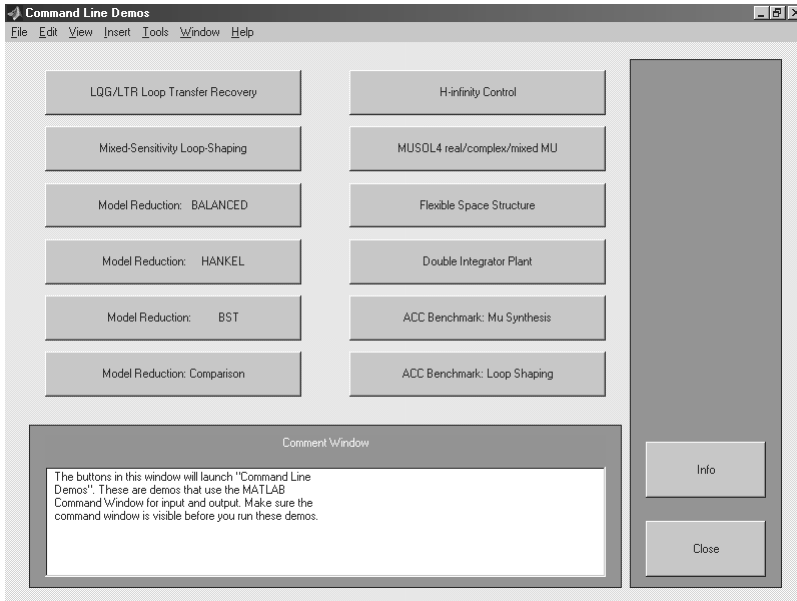


Рис. 13.20. Панель демонстрационных примеров пакета Robust Control

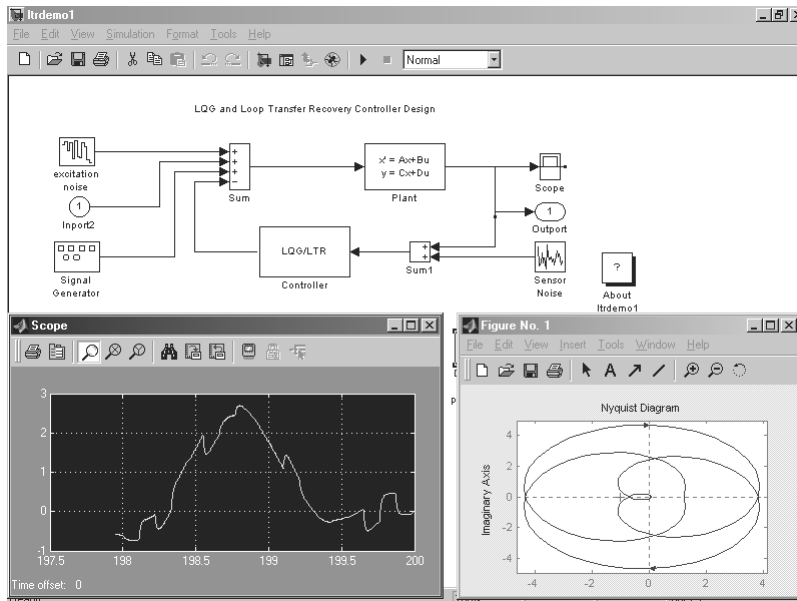


Рис. 13.21. GUI пакета Robust Control и средства контроля за работой системы

Пакет Robust Control базируется на функциях пакета Control System, одновременно предоставляя усовершенствованный набор алгоритмов для проектирования систем управления. Пакет обеспечивает переход между современной теорией управления и практическими приложениями. Он имеет множество функций, реализующих современные методы проектирования и анализа многопараметрических робастных регуляторов.

Проявления неопределенностей, нарушающих устойчивость систем, многообразны – шумы и возмущения в сигналах, неточность модели передаточной функции, немоделируемая нелинейная динамика. Пакет Robust Control позволяет оценить многопараметрическую границу устойчивости при различных неопределенностях. Среди используемых методов: алгоритм Перрона, анализ особенностей передаточных функций и др.

Пакет Robust Control обеспечивает различные методы проектирования обратных связей, среди которых: LQR, LQG, LQG/LTR и др. Необходимость понижения порядка модели возникает в нескольких случаях: понижение порядка объекта, понижение порядка регулятора, моделирование больших систем. Качественная процедура понижения порядка модели должна быть численно устойчива. Процедуры, включенные в пакет Robust Control, успешно справляются с этой задачей.

13.4.3. Model Predictive Control Toolbox

Пакет Model Predictive Control содержит полный набор средств для реализации стратегии предиктивного (упреждающего) управления. Эта стратегия была разработана для решения практических задач управления сложными многоканальными процессами при наличии ограничений на переменные состояния и управление. Методы предикативного управления используются в химической промышленности и для управления другими непрерывными процессами. Пакет обеспечивает:

- моделирование, идентификацию и диагностику систем;
- поддержку MISO (много входов – один выход), MIMO, переходных характеристик, моделей пространства состояний;
- системный анализ;
- конвертирование моделей в различные формы представления (пространство состояний, передаточные функции);
- предоставление учебников и демонстрационных примеров.

Предиктивный подход к задачам управления использует явную линейную динамическую модель объекта для прогнозирования влияния будущих изменений управляющих переменных на поведение объекта. Проблема оптимизации формулируется в виде задачи квадратичного программирования с ограничениями, решаемой на каждом такте моделирования заново. Пакет позволяет создавать и тестировать регуляторы как для простых, так и для сложных объектов.

Пакет содержит более полусотни специализированных функций для проектирования, анализа и моделирования динамических систем с использованием предикативного управления. Он поддерживает следующие типы систем: импульсные, непрерывные и дискретные по времени, пространство состояний. Обрабатываются

ся различные виды возмущений. Кроме того, в модель могут быть явно включены ограничения на входные и выходные переменные.

Средства моделирования позволяют осуществлять слежение и стабилизацию. Средства анализа включают вычисление полюсов замкнутого контура, частотного отклика, другие характеристики системы управления. Для идентификации модели в пакете имеются функции взаимодействия с пакетом System Identification Toolbox. Пакет также включает две функции Simulink, позволяющие тестировать нелинейные модели.

13.4.4. Communications Toolbox

Пакет прикладных программ для построения и моделирования разнообразных телекоммуникационных устройств: цифровых линий связи, модемов, преобразователей сигналов и др. [11, 14, 17]. Предшествующие версии этого пакета имели богатейший набор моделей самых различных устройств связи и телекоммуникаций. Он содержал ряд интересных примеров моделирования коммуникационных средств, например модема, работающего по протоколу v34, модулятора для обеспечения однополосной модуляции и др.

В настоящее время пакет, похоже, находится в далеко незавершенной стадии пересмотра. Прежде всего надо отметить, что теперь появились два варианта пакета с названием Communication – один в наборе Toolbox и другой в наборе Blockset. При этом последний существенно расширен и в него включены многочисленные библиотеки коммуникационных устройств. В связи с этим мы воздержимся от более подробного описания этого пакета (точнее, пакетов). Заинтересованный читатель может просмотреть их возможности в той реализации, которая ему доступна.

13.4.5. μ -Analysis and Synthesis

Пакет μ -Analysis and Synthesis содержит функции для проектирования устойчивых систем управления. Пакет использует оптимизацию в равномерной норме и сингулярный параметр μ . В этот пакет включен графический интерфейс для упрощения операций с блоками при проектировании оптимальных регуляторов.

Свойства пакета:

- проектирование регуляторов, оптимальных в равномерной и интегральной норме;
- оценка действительного и комплексного сингулярного параметра μ ;
- D-K-итерации для приближенного μ -синтеза;
- графический интерфейс для анализа отклика замкнутого контура;
- средства понижения порядка модели;
- непосредственное связывание отдельных блоков больших систем.

Модель пространства состояний может быть создана и проанализирована на основе системных матриц. Пакет поддерживает работу с непрерывными и дискретными моделями. Пакет обладает полноценным графическим интерфейсом,

включающим в себя: возможность устанавливать диапазон вводимых данных, специальное окно для редактирования свойств D-К-итераций и графическое представление частотных характеристик. Имеет функции для матричного сложения, умножения, различных преобразований и других операций над матрицами. Обеспечивает возможность понижения порядка моделей.

13.4.6. Quantitative Feedback Theory Toolbox

Пакет содержит функции для создания робастных (устойчивых) систем с обратной связью. QFT (количественная теория обратных связей) – инженерный метод, использующий частотное представление моделей для удовлетворения различных требований к качеству при наличии неопределенных характеристик объекта. В основе метода лежит наблюдение, что обратная связь необходима в тех случаях, когда некоторые характеристики объекта не определены и/или на его вход подаются неизвестные возмущения. Возможности пакета:

- оценка частотных границ неопределенности, присущей обратной связи;
- графический интерфейс пользователя, позволяющий оптимизировать процесс нахождения требуемых параметров обратной связи;
- функции для определения влияния различных блоков, вводимых в модель (мультиплексоров, сумматоров, петель обратной связи) при наличии неопределенностей;
- поддержка моделирования аналоговых и цифровых контуров обратной связи, каскадов и многоконтурных схем;
- разрешение неопределенности в параметрах объекта с использованием параметрических и непараметрических моделей или комбинации этих типов моделей.

Теория обратных связей является естественным продолжением классического частотного подхода к проектированию. Ее основная цель – проектирование простых регуляторов небольшого порядка с минимальной шириной полосы пропускания, удовлетворяющих качественным характеристикам при наличии неопределенностей.

Пакет позволяет вычислять различные параметры обратных связей, фильтров, проводить тестирование регуляторов как в непрерывном, так и в дискретном пространстве. Имеет удобный графический интерфейс, позволяющий создавать простые регуляторы, удовлетворяющие требованиям пользователя.

QFT позволяет проектировать регуляторы, удовлетворяющие различным требованиям, несмотря на изменения параметров модели. Измеряемые данные могут быть непосредственно использованы для проектирования регуляторов, без необходимости идентификации сложного отклика системы.

13.4.7. LMI Control Toolbox

Пакет LMI (Linear Matrix Inequality) Control обеспечивает интегрированную среду для постановки и решения задач линейного программирования. Предназначенный первоначально для проектирования систем управления, пакет позволяет

решать любые задачи линейного программирования практически в любой сфере деятельности, где такие задачи возникают. Основные возможности пакета:

- решение задач линейного программирования: задачи совместности ограничений, минимизация линейных целей при наличии линейных ограничений, минимизация собственных значений;
- исследование задач линейного программирования;
- графический редактор задач линейного программирования;
- задание ограничений в символьном виде;
- многокритериальное проектирование регуляторов;
- проверка устойчивости: квадратичная устойчивость линейных систем, устойчивость по Ляпунову, проверка критерия Попова для нелинейных систем.

Пакет LMI Control содержит современные симплексные алгоритмы для решения задач линейного программирования. Использует структурное представление линейных ограничений, что повышает эффективность и минимизирует требования к памяти. Пакет имеет специализированные средства для анализа и проектирования систем управления на основе линейного программирования.

С помощью решателей задач линейного программирования можно легко выполнять проверку устойчивости динамических систем и систем с нелинейными компонентами. Ранее этот вид анализа считался слишком сложным для реализации. Пакет позволяет даже такое комбинирование критериев, которое ранее считалось слишком сложным и разрешимым лишь с помощью эвристических подходов.

Пакет является мощным средством для решения выпуклых задач оптимизации, возникающих в таких областях, как управление, идентификация, фильтрация, структурное проектирование, теория графов, интерполяция и линейная алгебра. Пакет LMI Control включает два вида графического интерфейса пользователя: редактор задачи линейного программирования (LMI Editor) и интерфейс Magshape. LMI Editor позволяет задавать ограничения в символьном виде, а Magshape обеспечивает пользователя удобными средствами работы с пакетом.

13.5. Пакет идентификации систем

Пакет System Identification содержит средства для создания математических моделей динамических систем на основе наблюдаемых входных и выходных данных [10]. Он имеет гибкий графический интерфейс, помогающий организовать данные и создавать модели. Методы идентификации, входящие в пакет, применимы для решения широкого класса задач, от проектирования систем управления и обработки сигналов до анализа временных рядов и вибрации. Основные свойства пакета:

- простой и гибкий интерфейс;
- предварительная обработка данных, включая предварительную фильтрацию, удаление трендов и смещений;
- выбор диапазона данных для анализа;
- методы авторегрессии;
- анализ отклика во временной и частотной областях;
- отображение нулей и полюсов передаточной функции системы;

- анализ невязок при тестировании модели;
- построение сложных диаграмм, таких как диаграмма Найквиста и др.

Графический интерфейс упрощает предварительную обработку данных, а также диалоговый процесс идентификации модели. Возможна также работа с пакетом в командном режиме (рис. 13.22) и с применением расширения Simulink. Операции загрузки и сохранения данных, выбора диапазона, удаления смещений и трендов выполняются с минимальными усилиями и находятся в главном меню.

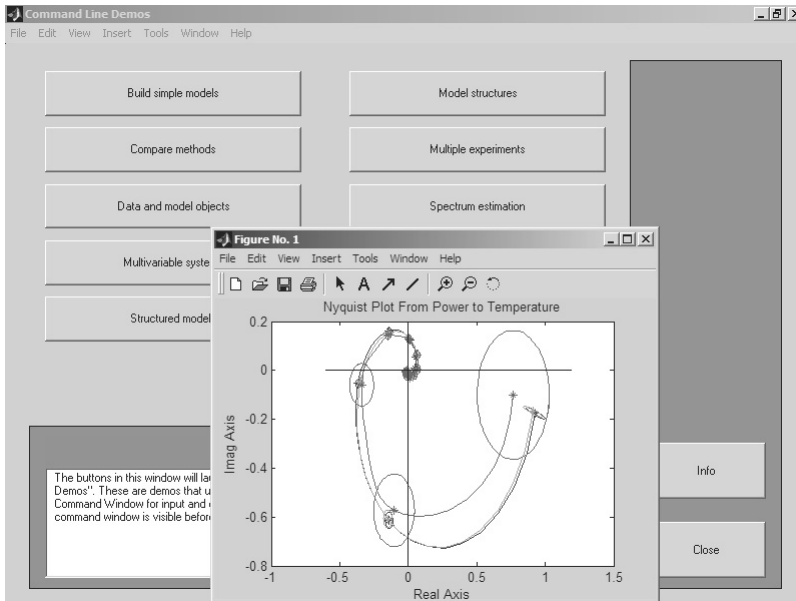


Рис. 13.22. Пример работы с пакетом идентификации систем в командном режиме

Представление данных и идентифицированных моделей организовано графически таким образом, что в процессе интерактивной идентификации пользователь легко может вернуться к предыдущему шагу работы. Для новичков существует возможность просматривать следующие возможные шаги. Специалисту графические средства позволяют отыскать любую из ранее полученных моделей и оценить ее качество в сравнении с другими моделями.

Начав с измерения выхода и входа, можно создать параметрическую модель системы, описывающую ее поведение в динамике. Пакет поддерживает все традиционные структуры моделей, включая авторегрессию, структуру Бокса–Дженкинса и др. Он поддерживает линейные модели пространства состояний, которые могут быть определены как в дискретном, так и в непрерывном пространствах. Эти модели могут включать произвольное число входов и выходов.

В пакет включены функции, которые можно использовать как тестовые данные для идентифицированных моделей. Идентификация линейных моделей широко используется при проектировании систем управления, когда требуется создать модель объекта. В задачах обработки сигналов модели могут быть использованы для адаптивной обработки сигналов. Методы идентификации успешно применяются и для финансовых приложений.

13.6. Пакеты для обработки сигналов и изображений

13.6.1. *Signal Processing Toolbox*

Пакет Signal Processing обеспечивает чрезвычайно обширные возможности создания программ обработки сигналов для современных научных и технических приложений. В пакете используются разнообразная техника фильтрации и новейшие алгоритмы спектрального анализа [12, 15, 18, 36]. Пакет содержит модули для разработки линейных систем и анализа временных рядов. Пакет будет полезен, в частности, в таких областях, как обработка аудио- и видеoinформации, телекоммуникации, геофизика, задачи управления в реальном режиме времени, экономика, финансы и медицина. Основные свойства пакета:

- моделирование сигналов и линейных систем;
- проектирование, анализ и реализация цифровых и аналоговых фильтров;
- быстрое преобразование Фурье, дискретное косинусное и другие преобразования;
- оценка спектров и статистическая обработка сигналов;
- параметрическая обработка временных рядов;
- генерация сигналов различной формы.

Пакет Signal Processing – идеальная оболочка для анализа и обработки сигналов. В нем используются проверенные практикой алгоритмы, выбранные по критериям максимальной эффективности и надежности. Пакет содержит широкий спектр алгоритмов для представления сигналов и линейных моделей. Этот набор позволяет пользователю достаточно гибко подходить к созданию сценария обработки сигналов. Пакет включает алгоритмы для преобразования модели из одного представления в другое.

Рисунок 13.23 показывает вид окна справки системы MATLAB 6.5 с полным перечнем демонстрационных примеров новейшей версии пакета Signal Processing. В правой части окна видно окно одного из примеров на конструирование полосового фильтра трех типов. Представлены АЧХ этих фильтров. Этот рисунок дает представление о типичном GUI данного пакета.

Пакет Signal Processing включает полный набор методов для создания цифровых фильтров с разнообразными характеристиками. Он позволяет быстро разрабатывать фильтры верхних и нижних частот, полосовые пропускающие и задер-

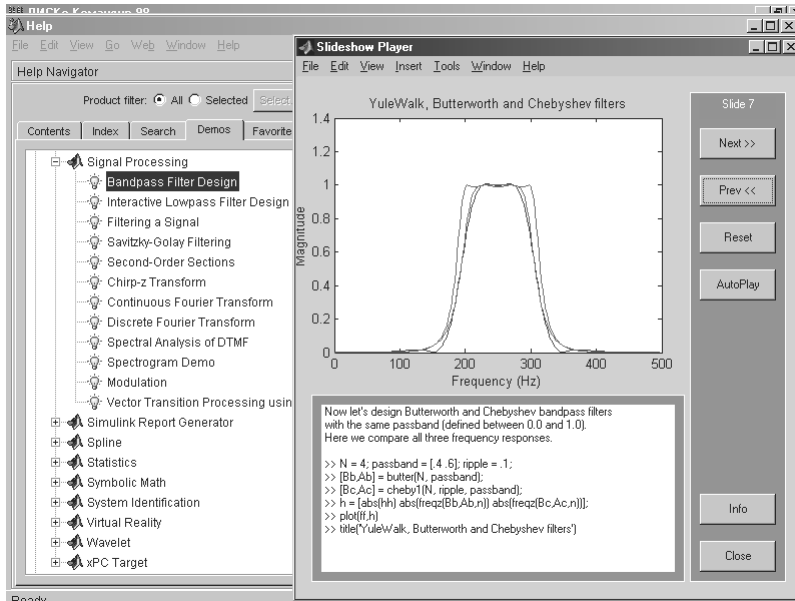


Рис. 13.23. Окно справки с перечнем примеров пакета *Signal Processing* и окном GUI примера на конструирование полосового фильтра

живающие фильтры, многополосные фильтры, в том числе фильтры Чебышева, Юла-Уолкера, эллиптические и др.

Графический интерфейс позволяет проектировать фильтры, задавая требования к ним в режиме переноса объектов мышью. В пакет включены следующие новые методы проектирования фильтров:

- обобщенный метод Чебышева для создания фильтров с нелинейной фазовой характеристикой, комплексными коэффициентами или произвольным откликом. Алгоритм разработан Макленаном и Карамом в 1995 г.;
- метод наименьших квадратов с ограничениями позволяет пользователю явно контролировать максимальную ошибку (сглаживание);
- метод расчета минимального порядка фильтра с окном Кайзера;
- обобщенный метод Баттерворта для проектирования низкочастотных фильтров с максимально однородными полосами пропускания и затухания.

Основанный на оптимальном алгоритме быстрого преобразования Фурье, пакет *Signal Processing* обладает непревзойденными характеристиками для частотного анализа и спектральных оценок. Пакет включает функции для вычисления дискретного преобразования Фурье, дискретного косинусного преобразования, преобразования Гильберта и других преобразований, часто применяемых для анализа, кодирования и фильтрации. В пакете реализованы такие методы спектрального анализа, как метод Вельха, метод максимальной энтропии и др.

Новый графический интерфейс позволяет просматривать и визуально оценивать характеристики сигналов, проектировать и применять фильтры, производить спектральный анализ (рис. 13.24), исследуя влияние различных методов и их параметров на получаемый результат. Графический интерфейс особенно полезен для визуализации временных рядов, спектров, временных и частотных характеристик, расположения нулей и полюсов передаточных функций систем.

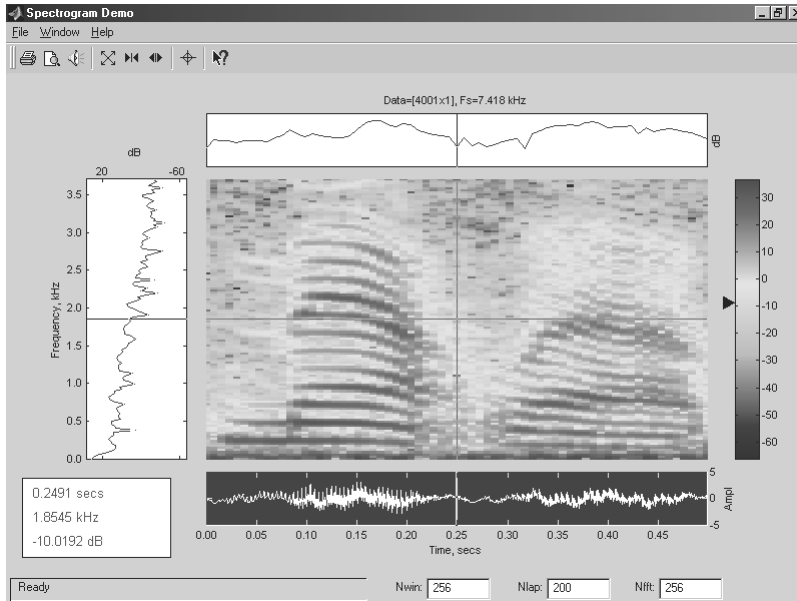


Рис. 13.24. Пример выполнения спектрального анализа в области частота–время с помощью пакета Signal Processing

Пакет Signal Processing является основой для решения многих других задач. Например, комбинируя его с пакетом Image Processing, можно обрабатывать и анализировать двумерные сигналы и изображения. В паре с пакетом System Identification пакет Signal Processing позволяет выполнять параметрическое моделирование систем во временной области. В сочетании с пакетами Neural Network и Fuzzy Logic может быть создано множество средств для обработки данных или выделения классификационных характеристик. Средство генерации сигналов позволяет создавать импульсные сигналы различной формы.

13.6.2. Image Processing Toolbox

Пакет Image Processing (версия 2 в MATLAB 6.0 и 3 в MATLAB 6.1/6.5) предоставляет ученым, инженерам и даже художникам широкий спектр средств для цифровой обработки и анализа изображений [12, 15, 19, 36]. Будучи тесно связан-

ным со средой разработки приложений MATLAB, пакет Image Processing Toolbox освобождает вас от выполнения длительных операций кодирования и отладки алгоритмов, позволяя сосредоточить усилия на решении основной научной или практической задачи. Основные свойства пакета:

- восстановление и выделение деталей изображений;
- фильтрация сигналов изображения (рис. 13.25);
- работа с выделенным участком изображения;
- анализ изображения;
- линейная фильтрация;
- преобразование изображений;
- геометрические преобразования;
- увеличение контрастности важных деталей;
- бинарные преобразования;
- обработка изображений и статистика;
- цветовые преобразования;
- изменение палитры;
- преобразование типов изображений.

Пакет Image Processing дает широкие возможности для создания и анализа графических изображений в среде MATLAB. Этот пакет обеспечивает чрезвычай-

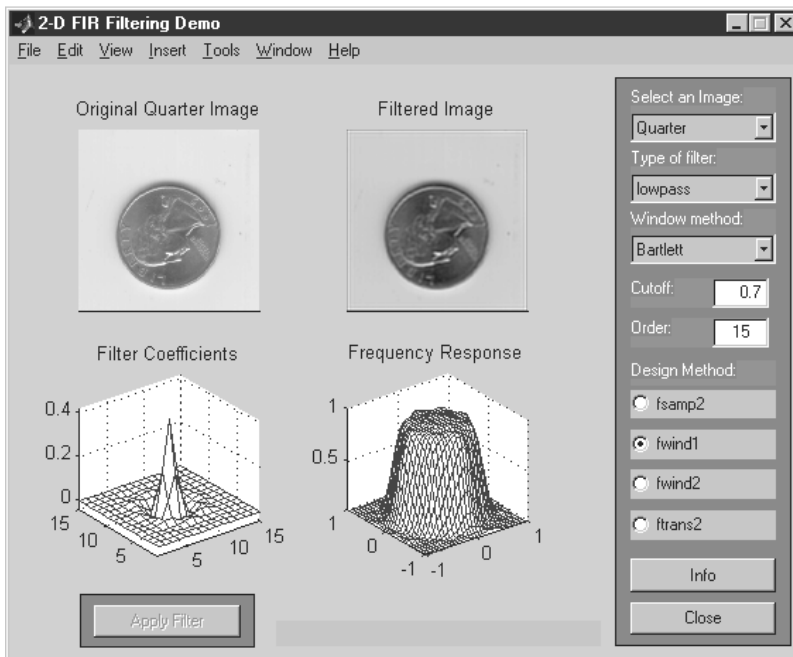


Рис. 13.25. Пример фильтрации сигналов изображения монеты и построение изображения по отфильтрованному сигналу с помощью пакета Image Processing

но гибкий интерфейс, позволяющий манипулировать изображениями (рис. 13.26), интерактивно разрабатывать графические картины, визуализировать наборы данных и аннотировать результаты для технических описаний, докладов и публикаций.

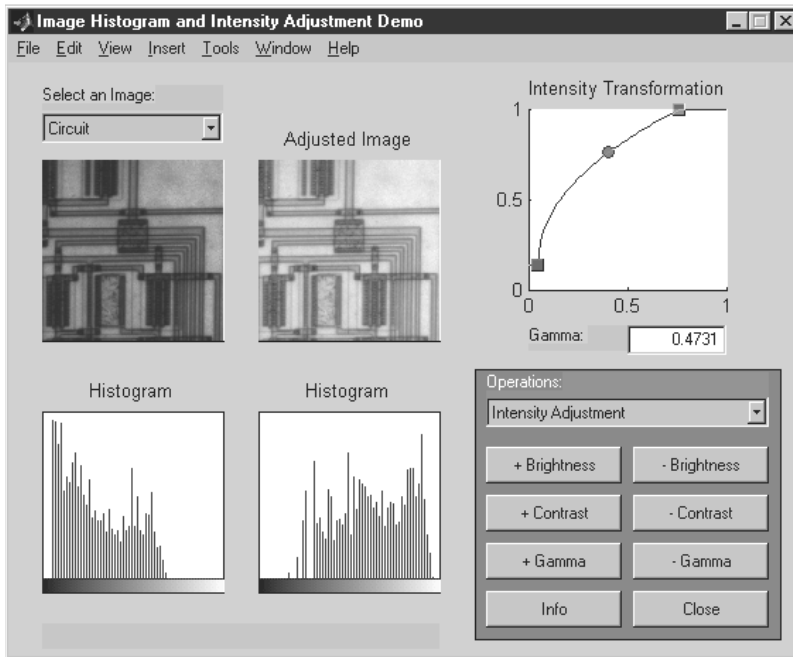


Рис. 13.26. Пример проведения гамма-коррекции изображения интегральной микросхемы с помощью пакета *Image Processing*

Гибкость, соединение алгоритмов пакета с такой особенностью MATLAB, как матрично-векторное описание, делает пакет очень удачно приспособленным для решения практически любых задач по разработке и представлению графики. На рис. 13.27 представлен пример решения еще одной распространенной задачи обработки сигналов изображений – очистка их от шума.

В MATLAB входят специально разработанные процедуры, позволяющие повысить эффективность графической оболочки. Можно отметить, в частности, такие особенности:

- интерактивная отладка при разработке графики;
- профилировщик для оптимизации времени выполнения алгоритма;
- средства построения интерактивного графического интерфейса пользователя (GUI Builder) для ускорения разработки GUI-шаблонов, позволяющие настраивать его под задачи пользователя.

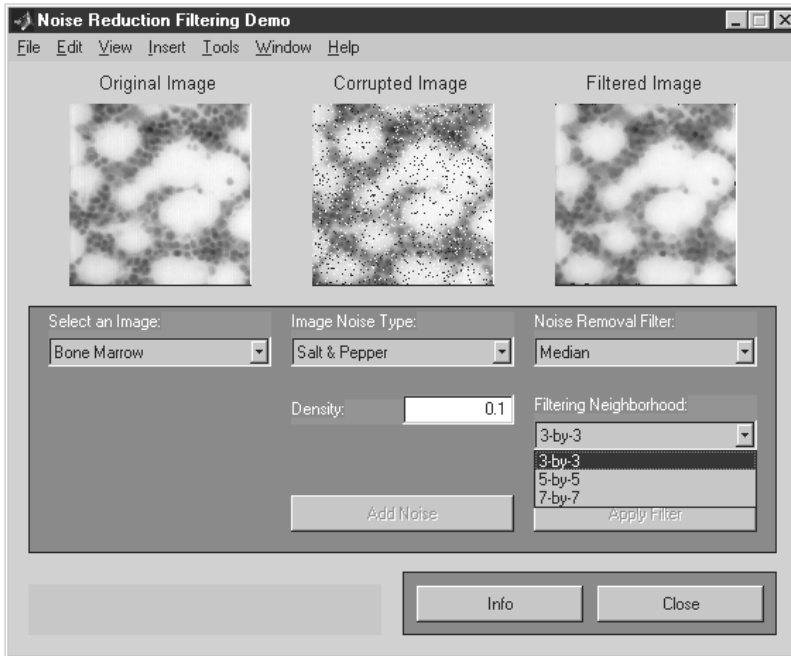


Рис. 13.27. Пример очистки изображения от шума с помощью пакета *Image Processing*

Этот пакет позволяет пользователю тратить значительно меньше времени и сил на создание стандартных графических изображений и, таким образом, сконцентрировать усилия на важных деталях и особенностях изображений.

MATLAB и пакет *Image Processing* максимально приспособлены для развития, внедрения новых идей и методов пользователя. Для этого имеется набор сопрягаемых пакетов, направленных на решение всевозможных специфических задач и задач в нетрадиционной постановке. Примером может служить задача изменения свойств изображения в некоторой произвольно выделенной области – рис. 13.28.

Пакет *Image Processing* в настоящее время интенсивно используется в более чем 4000 компаниях и университетах по всему миру. При этом имеется очень широкий круг задач, которые пользователи решают с помощью данного пакета, например космические исследования, военные разработки, астрономия, медицина, биология, робототехника, материаловедение, генетика и т. д. Версия пакета *Image Processing* 3.1 в MATLAB 6.1/6.5 существенно расширена в основном за счет расширения числа функций расширенных морфологических операций, обработки пространственных изображений, целочисленной арифметики, фильтрации изображений, регистрации изображений и др.

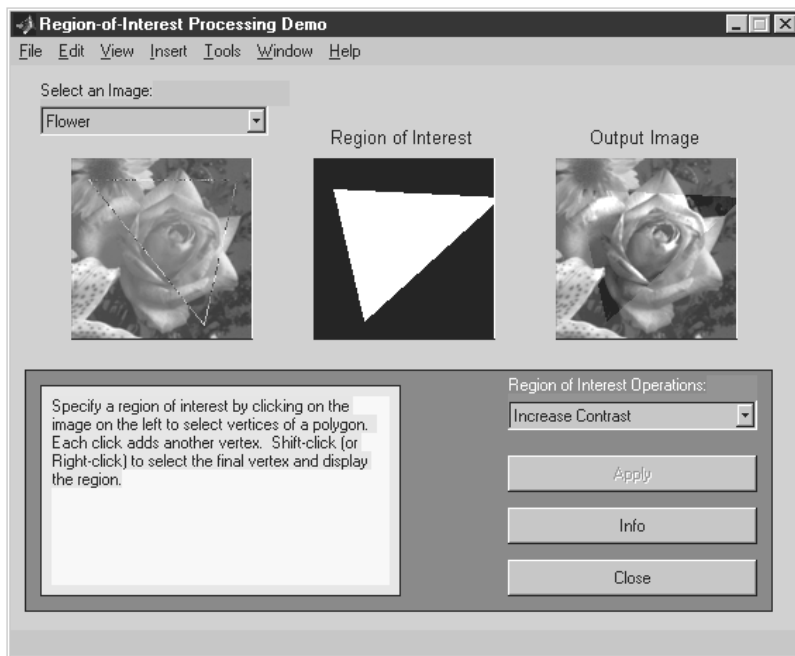


Рис. 13.28. Пример повышения контрастности изображения в заданной (треугольной) области с помощью пакета Image Processing

13.6.3. Wavelet Toolbox

Пакет Wavelet Toolbox предоставляет пользователю полный набор программ для исследования многомерных нестационарных явлений с помощью вейвлетов (коротких волновых пакетов) [15, 18, 41]. Новейшие методы пакета Wavelet расширяют возможности пользователя в тех областях, где обычно применяется техника Фурье-разложения. Пакет может быть полезен для таких приложений, как обработка речи и аудиосигналов, телекоммуникации, геофизика, финансы и медицина. Вейвлет-преобразования уже нашли широкое практическое применение – в стандартах сжатия видеосигналов MP4 и JPEG 2000, в графических системах Corel Draw 9/10 и др. Выпускаются даже интегральные микросхемы для выполнения вейвлет-преобразований.

Основные свойства пакета Wavelet Toolbox следующие:

- усовершенствованный графический пользовательский интерфейс и набор команд для анализа, синтеза, фильтрации сигналов и изображений;
- преобразование многомерных непрерывных сигналов;
- дискретное преобразование сигналов;
- декомпозиция и анализ сигналов и изображений;

- широкий выбор базисных функций, включая коррекцию граничных эффектов;
- пакетная обработка сигналов и изображений;
- анализ пакетов сигналов, основанный на энтропии;
- фильтрация с возможностью установления жестких и нежестких порогов;
- оптимальное сжатие сигналов.

Доступ к средствам пакета возможен как из командной строки, так и из раздела **Demos** справки. В первом случае достаточно в командной строке набрать команду `wavemenu`, которая выводит окно средств пакета – рис. 13.29.

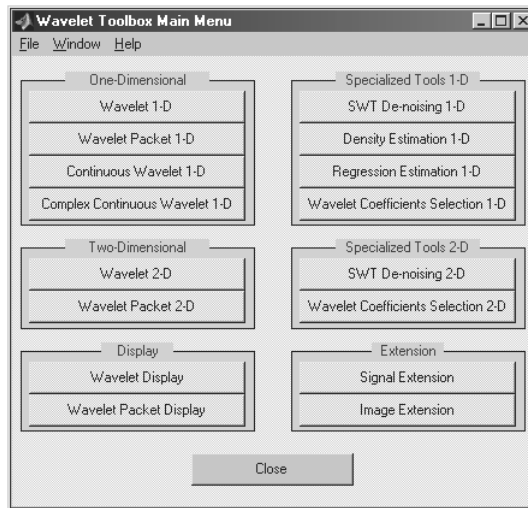


Рис. 13.29. Окно средств пакета *Wavelet Toolbox*

В этом окне представлены основные разделы вейвлет-преобразований для вейвлетов различного типа, например непрерывных, дискретных и пакетных. Нажатие той или иной кнопки вызывает появление окна GUI соответствующего раздела, в котором можно выбрать тип сигнала для вейвлет-преобразований и в широких пределах менять набор средств для таких преобразований.

На рис. 13.30 представлено применение техники непрерывных вейвлет-преобразований для получения спектрограммы сигнала, представляющего собой последовательность двух гармонических колебаний со скачком частоты. Полученная спектрограмма хорошо выявляет периодичность каждой части сигнала и наличие скачка частоты. Его легко обнаружить как по вейвлет-спектрограмме, так и по поведению вейвлет-коэффициентов и линиям локальных максимумов (*Local Maxima Lines*). Обнаружение локальных особенностей сигналов и функций – одна из наиболее распространенных задач вейвлет-анализа.

Пример, представленный на рис. 13.31, хорошо поясняет суть вейвлет-преобразований. Здесь задан сложный сигнал, характерный для электронных схем. Этот сигнал представляется совокупностью аппроксимирующих (грубых) вейв-

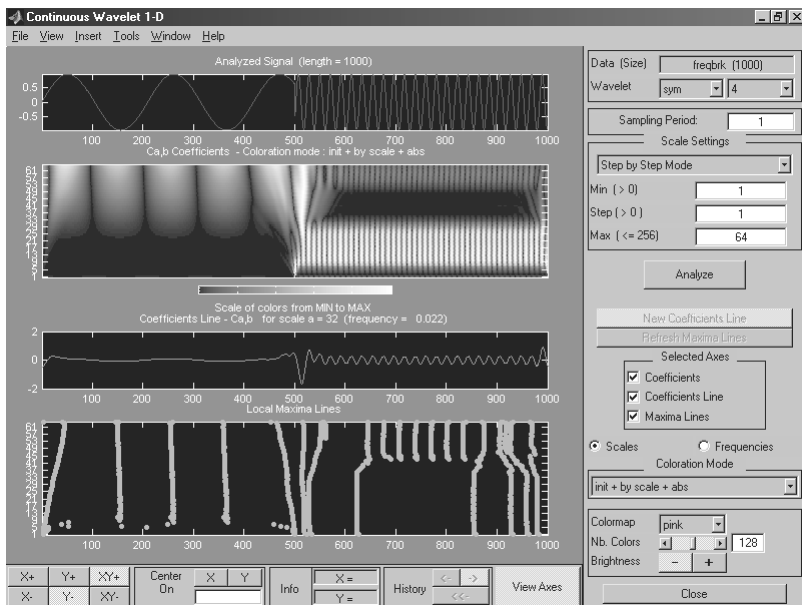


Рис. 13.30. Вейвлет-анализ и спектрограммы сигнала в виде синусоиды со скачком частоты

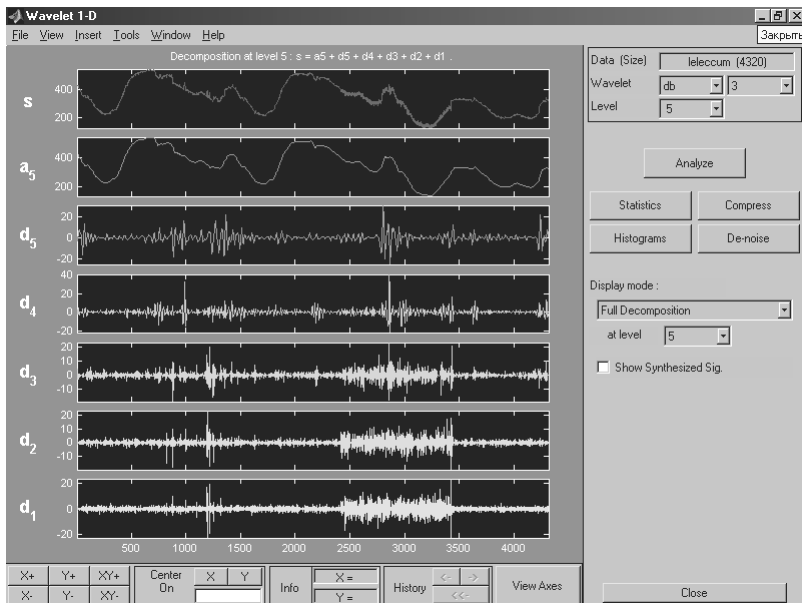


Рис. 13.31. Вейвлет-декомпозиция и реставрация сложного сигнала с применением вейвлетов Добеши

лет-коэффициентов a_i и детализирующих (точных) d_i , где i – номер коэффициентов. Использовано представление с помощью вейлетов Добеши $db3$ при уровне декомпозиции и реставрации сигнала $L=5$.

Вейлеты Добеши представляют один из многих типов ортогональных вейвлетов, которые теоретически обеспечивают точную реставрацию сигналов после прямого и обратного вейвлет-преобразования сигнала. При этом ограничение числа вейвлет-коэффициентов по существу означает процесс фильтрации сигнала, сопровождаемый его компрессией. Фильтрация сигнала позволяет, в частности, очищать сигналы от шума. При этом вейвлет-фильтрация и очистка от шума более эффективны, чем обычная фильтрация и очистка от шума на основе применения преобразований Фурье – как обычных (включая дискретные и быстрые), так и оконных.

Пользуясь пакетом, можно анализировать такие особенности сигналов и временных рядов, которые упускают другие методы анализа сигналов, то есть тренды, выбросы, разрывы в производных высоких порядках. Пакет позволяет сжимать и фильтровать сигналы без явных потерь, даже в тех случаях, когда нужно сохранить и высоко-, и низкочастотные компоненты сигнала. Имеются алгоритмы сжатия и фильтрации и для пакетной обработки сигналов. Программы сжатия выделяют минимальное число коэффициентов, представляющих исходную информацию наиболее точно, что очень важно для последующих стадий работы системы сжатия.

Рисунок 13.32 показывает технику математической обработки изображений. Здесь выполняется экстраполяция области изображения, размещенного в квадрате, в области за пределами этого квадрата.

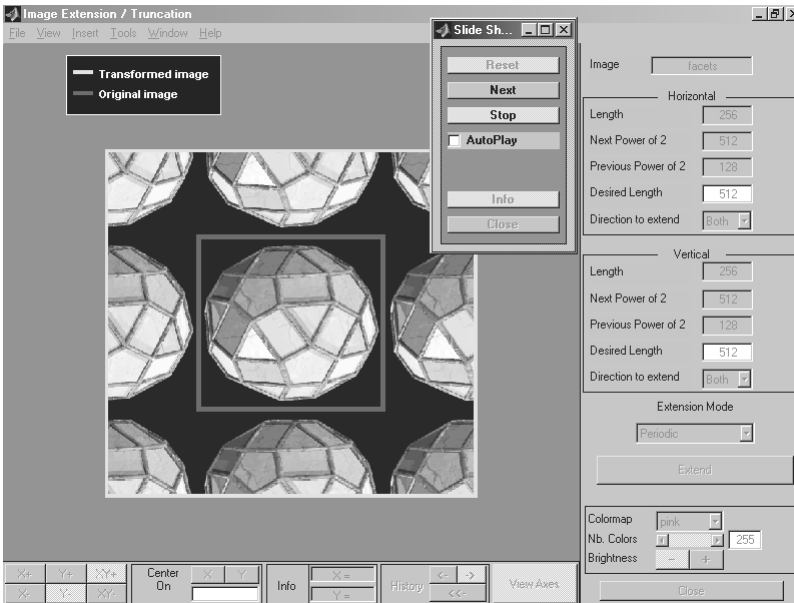


Рис. 13.32. Вейвлет-экстраполяция двумерного сигнала, представляющего изображение

В пакет включены следующие базисные наборы вейвлетов: ортогональные Добеши, биортогональные, Хаара, «Мексиканская шляпа», Майера и др. Вы также можете добавить в пакет свои собственные базисы. Обширное руководство пользователя объясняет принципы работы с методами пакета, сопровождая их многочисленными примерами и полноценным разделом ссылок.

13.7. Прочие пакеты прикладных программ

13.7.1. *Financial Toolbox*

Довольно актуальный для нашего периода рыночных реформ пакет прикладных программ по финансово-экономическим расчетам *Financial Toolbox* содержит множество функций по расчету сложных процентов, операций по банковским вкладам, вычисления прибыли и многому другому [37]. К сожалению, из-за многочисленных (хотя, в общем-то, не слишком принципиальных) различий в финансово-экономических формулах его применение в наших условиях не всегда разумно – есть множество отечественных программ для таких расчетов – например, «Бухгалтерия 1С». Но если вы хотите подключиться к базам данных агентств финансовых новостей – Bloomberg, IDC через пакет *Datafeed Toolbox MATLAB*, то, конечно, обязательно пользуйтесь и финансовыми пакетами расширения *MATLAB*.

Пакет *Financial* является основой для решения в *MATLAB* множества финансовых задач, от простых вычислений до полномасштабных распределенных приложений. Пакет *Financial* может быть использован для расчета процентных ставок и прибыли, анализа производных доходов и депозитов, оптимизации портфеля инвестиций. Основные возможности пакета:

- обработка данных;
- дисперсионный анализ эффективности портфеля инвестиций;
- анализ временных рядов;
- расчет доходности ценных бумаг и оценка курсов;
- статистический анализ;
- анализ чувствительности рынка;
- калькуляция ежегодного дохода и расчет денежных потоков;
- методы начисления износа и амортизационных отчислений.

Пакет *Financial* содержит алгоритмы, которые позволяют анализировать портфель инвестиций, динамику и экономические коэффициенты чувствительности. В частности, при определении эффективности инвестиций функции пакета позволяют сформировать портфель, удовлетворяющий классической задаче Г. Марковица. Пользователь может комбинировать алгоритмы пакета для вычисления коэффициентов Шарпе и ставок дохода. Анализ динамики и экономических коэффициентов чувствительности позволяет пользователю определить позиции для стредл-делок, хеджирования и сделок с фиксированными ставками.

Пакет Financial обеспечивает также обширные возможности для представления и презентации данных и результатов в виде традиционных для экономической и финансовой сфер деятельности графиков и диаграмм. Денежные средства могут по желанию пользователя отображаться в десятичном, банковском и процентном форматах.

13.7.2. Mapping Toolbox

Пакет Mapping предоставляет графический и командный интерфейс для анализа географических данных, отображения карт и доступа к внешним источникам данных по географии. Кроме того, пакет пригоден для работы с множеством широко известных атласов. Эти средства в комбинации с MATLAB предоставляют пользователям все условия для продуктивной работы с научными географическими данными. Основные возможности пакета:

- визуализация, обработка и анализ графических и научных данных;
- более 60 проекций карт (прямые и инверсные);
- проектирование и отображение векторных, матричных и составных карт;
- графический интерфейс для построения и обработки карт и данных;
- глобальные и региональные атласы данных и сопряжение с ответственными данными высокого разрешения;
- функции географической статистики и навигации;
- трехмерное представление карт со встроенными средствами подсветки и затенения;
- конвертеры для популярных форматов географических данных: DCW, TIGER, ETOPO5.

Пакет Mapping включает более 60 наиболее широко известных проекций, включая цилиндрическую, псевдоцилиндрическую, коническую, поликоническую и псевдоконическую, азимутальную и псевдоазимутальную. Возможны прямые и обратные проекции, а также нестандартные виды проекции, задаваемые пользователем.

В пакете Mapping *картой* называется любая переменная или множество переменных, отражающих или назначающих численное значение географической точке или области. Пакет позволяет работать с векторными, матричными и смешанными картами данных. Мощный графический интерфейс обеспечивает интерактивную работу с картами, например возможность подвести указатель к объекту и, щелкнув на нем, получить информацию. Графический интерфейс MAPTOOL – полная среда разработки приложений для работы с картами. Рисунок 13.33 показывает применение пакета Mapping Toolbox для построения анимированных орбит спутника Земли на фоне ее карты.

Наиболее широко известные атласы мира, Соединенных Штатов, астрономические атласы входят в состав пакета. Географическая структура данных упрощает извлечение и обработку данных из атласов и карт. Географическая структура данных и функции взаимодействия с внешними географическими данными форматов Digital Chart of the World (DCW), TIGER, TBASE и ETOPO5 объединены



Рис. 13.33. Пример построения анимационных орбит спутника Земли на фоне ее карты (пакет Mapping)

вместе, чтобы обеспечить мощный и гибкий инструмент для доступа к уже существующим и будущим географическим базам данных.

Тщательный анализ географических данных часто требует математических методов, работающих в сферической системе координат. Пакет Mapping снабжен подмножеством географических, статистических и навигационных функций для анализа географических данных. Функции навигации дают широкие возможности для выполнения задач перемещения, таких как позиционирование и планирование маршрутов. Пакет впервые описан в [19].

13.7.3. Data Acquisition Toolbox и Instrument Control Toolbox

Data Acquisition Toolbox – пакет расширения, относящийся к области сбора данных через блоки, подключаемые к внутренней шине компьютера. В нем сосредоточены средства для создания функциональных генераторов (рис. 13.34), виртуальных осциллографов (рис. 13.35), анализаторов спектра – словом, приборов, широко используемых в исследовательских целях для получения данных. Они поддерживаются соответствующей вычислительной базой.

Новый блок Instrument Control Toolbox позволяет подключать приборы и устройства с последовательным интерфейсом и с интерфейсами Канал общего пользования и VXI.

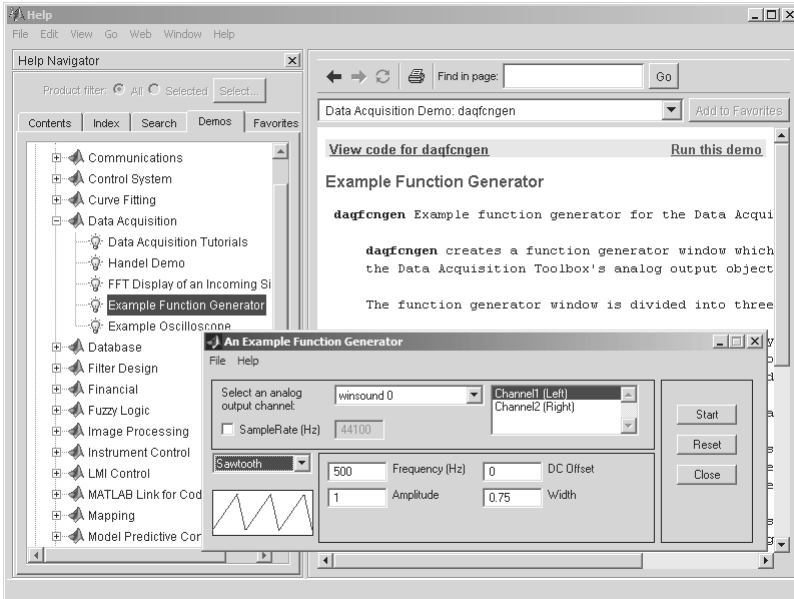


Рис. 13.34. Пример построения функционального генератора

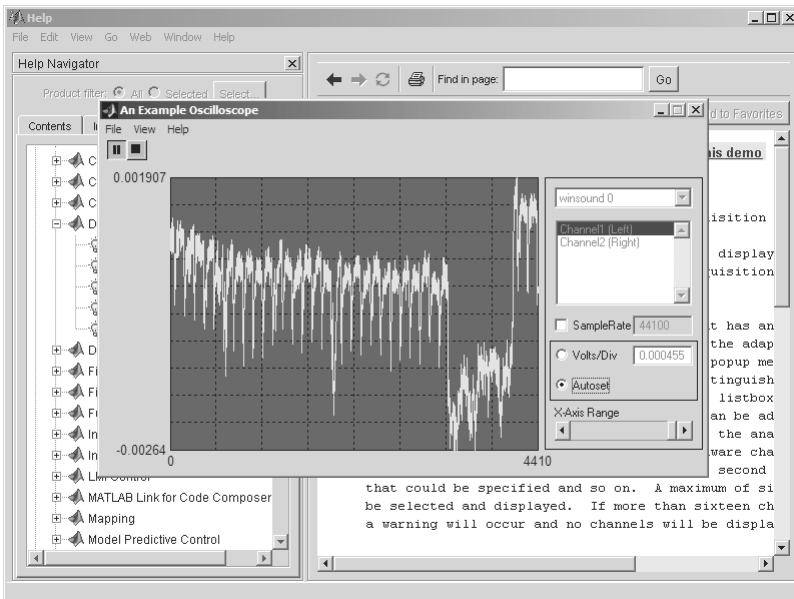


Рис. 13.35. Пример построения осциллографа и наблюдения зашумленного сигнала

13.7.4. Database toolbox

Более чем в 100 раз повышена скорость работы Database toolbox, при помощи которого осуществляется обмен информацией с целым рядом систем управления базами данных через драйверы ODBC или JDBC:

- Oracle 7.3.3;
- Access 95 или 97 Microsoft;
- Microsoft SQL Server 6.5 или 7.0;
- Sybase Adaptive Server 11;
- Sybase (бывший Watcom) SQL Server Anywhere 5.0;
- IBM DB2 Universal 5.0;
- Informix 7.2.2;
- Computer Associates Ingres (все версии).

Все данные предварительно преобразуются в массив ячеек. Визуальный конструктор (Visual Query Builder) (только при соединении через ODBC-драйверы) позволяет составлять сколь угодно сложные запросы на диалектах языка SQL этих баз данных даже без знания SQL. В одном сеансе может быть открыто много неоднородных баз данных.

13.7.5. Excel Link

Пакет расширения Excel Link позволяет использовать широко известный табличный процессор Microsoft Excel 97 как процессор ввода-вывода MATLAB. Для этого достаточно установить в Excel как add-in функцию поставляемый MathWorks файл exclink.xla. В Excel нужно набрать **Сервис** ⇒ **Надстройки** ⇒ **Обзор**, выбрать файл в каталоге \matlabr12\toolbox\exlink и установить его.

Теперь при каждом запуске Excel появится командное окно MATLAB, а панель управления Excel дополнится кнопками **getmatrix**, **putmatrix**, **evalstring**. Для закрытия MATLAB из Excel достаточно набрать =MLClose() в любой ячейке Excel. Для открытия после выполнения этой команды нужно либо щелкнуть мышью на одной из кнопок **getmatrix**, **putmatrix**, **evalstring**, либо набрать в Excel **Сервис** ⇒ **Макрос** ⇒ **Выполнить matlabinit**. Выделив мышью диапазон ячеек Excel, вы можете щелкнуть на **getmatrix** и набрать имя переменной MATLAB. Матрица появится в Excel. Заполнив числами диапазон ячеек Excel, вы можете выделить этот диапазон, щелкнуть на **putmatrix** и ввести имя переменной MATLAB. Работа, таким образом, интуитивно понятна. В отличие от MATLAB, Excel Link не чувствителен к регистру, так что ввод I и i или J и j равноценны. Достаточно полно пакет описан в [28].

13.7.6. Virtual Reality Toolbox

Пакет виртуальной реальности доступен, начиная с версии MATLAB 6.1. Позволяет осуществлять трехмерную анимацию и мультипликацию, в том числе моделей Simulink. Язык программирования – VRML – язык моделирования виртуаль-

ной реальности (Virtual Reality Modeling Language). Просмотр анимации возможен с любого компьютера, оснащенного браузером с поддержкой VRML. Этот пакет подтверждает, что математика – наука о количественных соотношениях и пространственных формах любых действительных или виртуальных миров.

13.7.7. MATLAB Compiler

Компилятор для программ, создаваемых на языке программирования системы MATLAB. Транслирует коды этих программ в программы на языке Си++. Применение компилятора обеспечивает возможность создания исполняемых кодов (полностью законченных программ), время выполнения которых для программ с большим числом циклических операций уменьшается в несколько раз. Помимо него, вы можете использовать и другие внешние компиляторы Си++.

13.8. Пакеты расширения MATLAB 6.5

13.8.1. Curve Fitting Toolbox

Математический пакет для подгонки кривых под данные (аппроксимации) Curve Fitting Toolbox расширяет возможности базовой системы по подгонке (подбору) кривых в графическом окне. Работа с пакетом (вызов командой `cftool`) и предусматривает четыре основных шага:

- импорт данных;
- проведение подгонки;
- анализ результатов подгонки;
- запись сессии на диск.

Рисунок 13.36 показывает выполнение квадратичной регрессии для тестовых данных `pop`. Здесь представлено начало подгонки кривых под заданные данные с применением окна подгонки **Fitting** (оно видно на фоне основного окна **Curve Fitting Toolbox**). Для начала подгонки и выбора первой аппроксимирующей функции в окне подгонки надо нажать клавишу **New Fit**. Данные о полученной функции аппроксимации представлены в поле результатов `Result`.

Для одного и того же набора данных возможно проведение нескольких видов подгонки (например, выполнение регрессии полиномами от первой до девятой степени). Нужные виды подгонки выбираются из списка, вводятся нажатием кнопки **Apply** и фиксируются в списке нажатием кнопки **Copy Fit**. Действие этих и других кнопок окна подгонки **Fitting** очевидно и фиксируется на графике точек исходных данных. По завершении подгонки можно получить графики кривых и точек исходных данных – рис. 13.37.

Для проведения анализа выполненной подгонки достаточно в окне **Fitting** нажать кнопку `Analysis...` Появится окно анализа (рис. 13.38), в котором надо задать виды анализа – на рис. 13.38 установлены все виды анализа, включающие вычисление первой и второй производной и интеграла от функций подгонки.

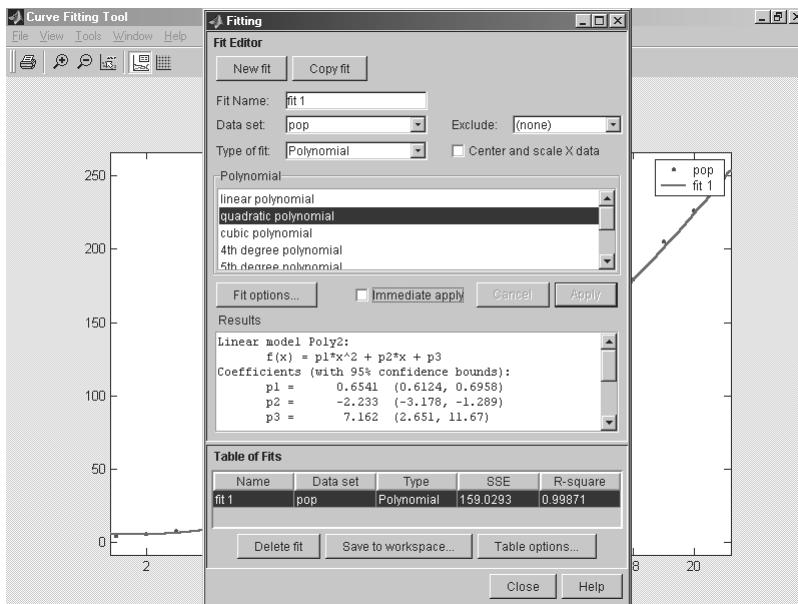


Рис. 13.36. Пример выполнения квадратичной регрессии для данных популяции pop

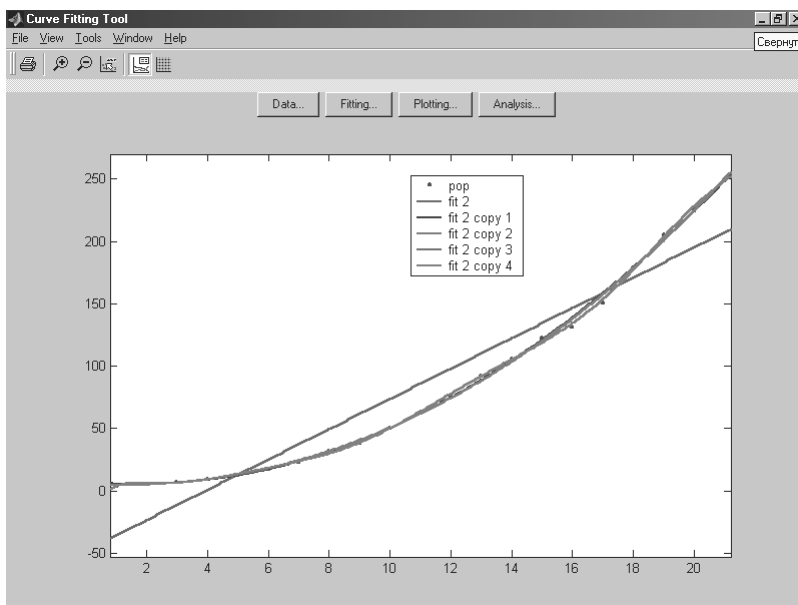


Рис. 13.37. Пример построения графиков кривых подгонки и исходных точек

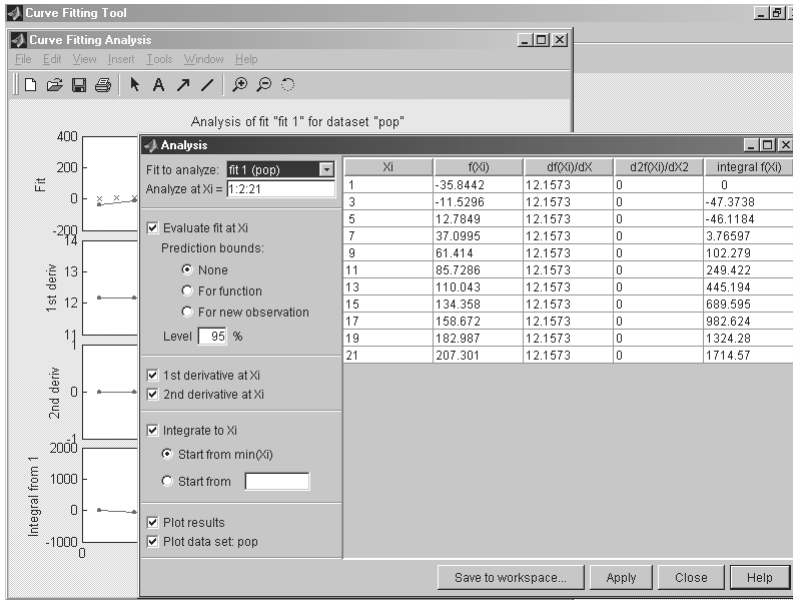


Рис. 13.38. Пример выполнения анализа по результатам подгонки

Пакет Curve Fitting Toolbox может помочь в решении ряда типовых задач приближения данных. Однако его возможности назвать выдающимися пока трудно – во многом они повторяют возможности подгонки данных в графическом окне.

13.8.2. Instrument Control Toolbox

Инструментальный пакет, предназначенный для работы с внешними устройствами. Обеспечивает:

- поддержку графического инструментального интерфейса GPIB (включая IEEE-488 и HP-IB);
- поддержку стандарта VISA и последовательных интерфейсов (RS-232, RS-422 and RS-485);
- поддержку данных бинарного и текстового форматов;
- управление внешними инструментами с помощью средств MATLAB;
- поддержку стандартных команд для управления инструментами SCPI (Standard Commands for Programmable Instruments).

Взаимодействие MATLAB с Instrument Control Toolbox и внешними устройствами (например, осциллографом) представлено на рис. 13.39.

Итак, пакет Instrument Control Toolbox предназначен для создания программно-управляемых от системы MATLAB инструментальных систем и комплексов. Для их реализации нужны специальные GPIB-драйверы и GPIB-контроллеры.

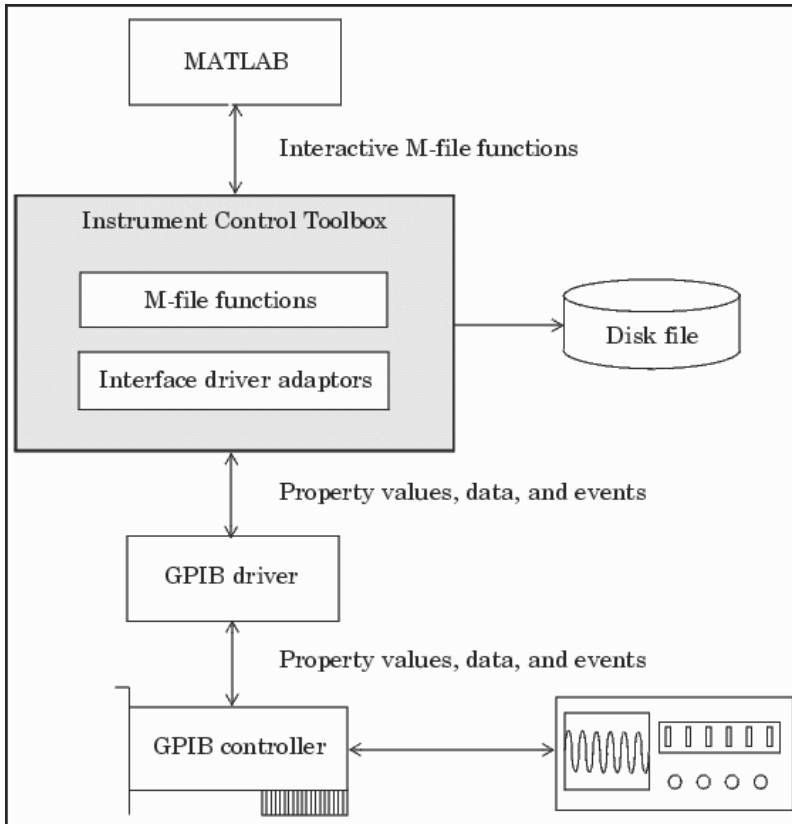


Рис. 13.39. Взаимодействие MATLAB с Instrument Control Toolbox и внешними устройствами

В пакет заложены и основы средств для поддержки новейших измерительных приборов с интерфейсом универсальной последовательной шины USB.

13.8.3. Developer's Kit for Texas Instruments DSP

Еще один новый пакет расширения для поддержки аппаратных средств – на этот раз знаменитых своей мощностью и популярностью цифровых сигнальных процессоров (DSP) корпорации Texas Instruments. Эти программируемые процессоры находят широкое применение в обработке различных сигналов (в том числе звуковых и видео) в реальном масштабе времени. Developer's Kit for Texas Instruments DSP имеет набор программных утилит и функций для программирования DSP и их испытания. Он весьма полезен специалистам, занятым разработ-

кой средств связи, телевидения, телекоммуникаций и иных устройств и систем, использующих современную технику обработки сигналов.

13.8.4. Dials & Gauges Blockset

Новый пакет для разработки сложных моделей устройств и систем с повышенной степенью визуализации. Наиболее характерными устройствами такого рода могут быть:

- приборные доски автомобилей и самолетов;
- диспетчерские пульта электростанций и аэропортов;
- пульта управления коммуникационным и медицинским оборудованием и т. д.

Этот пакет предназначен для работы с рядом других пакетов расширения системы MATLAB 6.5+Simulink 5:

- Data Acquisition Toolbox;
- DSP Blockset;
- Fixed-Point Blockset;
- Control ToolboxTool;
- Power System Blockset;
- Real-Time Windows Target;
- Real-Time WorkshopTool;
- StateflowTool;
- Virtual Reality ToolboxTool;
- xPC Target.

Отличительная черта этого пакета – повышенная реалистичность отображения блоков, из которых строятся модели устройств и систем. Например, регистрирующий напряжение или ток блок представляется уже не простым прямоугольником, а реалистично выглядящим вольтметром или амперметром с четкой шкалой и движущейся стрелкой. В версии MATLAB 6.5 SP1 пакет исчез.

13.8.5. Mechanical System Blockset

Новый набор средств для моделирования механических систем. Содержит специфические для таких систем средства подготовки и запуска модели. Примеры применения пакета содержат модели, изображения которых напоминают конструкции механических объектов, например робота с механической рукой или двигателя внутреннего сгорания. Обширное описание пакета имеется в формате PDF-файлов.

13.9. Новейшие пакеты расширения MATLAB 7+Simulink 6

13.9.1. Назначение и возможности пакета *Bioinformatics Toolbox*

Новейший пакет Bioinformatics Toolbox, включенный в состав системы MATLAB 7.*, представляет собой обширный набор инструментальных программных средств для работы в области биологии и геномной инженерии. В него входит множество функций, полезных для специалистов и ученых в этой области и существенно расширяющих их возможности в проведении научных исследований и преподавании данных областей науки.

Пакет Bioinformatics Toolbox 2.0 может работать в среде системы MATLAB Version 7.0.1 (реализация 14 с Service pack 1 на CD) и более поздних систем. Для полноценной работы с пакетом необходим пакет расширения Statistics Toolbox 5.0.1. Пакет Bioinformatics Toolbox имеет обширную справку и документацию в трех файлах формата PDF с общим объемом более 600 страниц.

Функции данного пакета позволяют его пользователям создавать новые алгоритмы в области обработки биологической информации и геномной инженерии. Функции пакета тщательно апробированы и избавляют пользователя от трудоемкой работы по их созданию самостоятельно.

Пакет имеет следующие возможности:

- подключение к Интернету для получения данных и доступа к базам данных в области биологии и генетики, считывание и конвертирование данных – рис. 13.40;
- анализ геномных цепочек – определяются основные статистические характеристики данных из области биологической информатики и геномной инженерии. Это включает образцы и последовательности, использующие скрытую модель Маркова (HMM – Hidden Markov Model);
- филогенетический (Phylogenetic) анализ – создание и управление деревом данных;
- анализ данных микроскопических множеств и их представление;
- данные массовой спектрометрии – анализ и улучшение «сырых» массовых данных спектрометрии (рис. 13.41);
- статистическое изучение – классификация и идентификация особенностей в наборах данных с помощью статистических инструментов изучения;

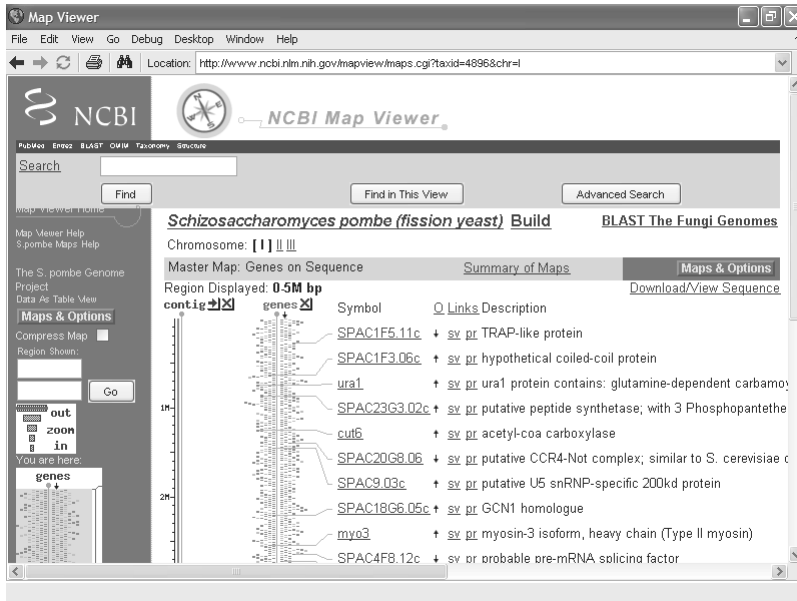


Рис. 13.40. Доступ к национальному центру биологической информации через Интернет

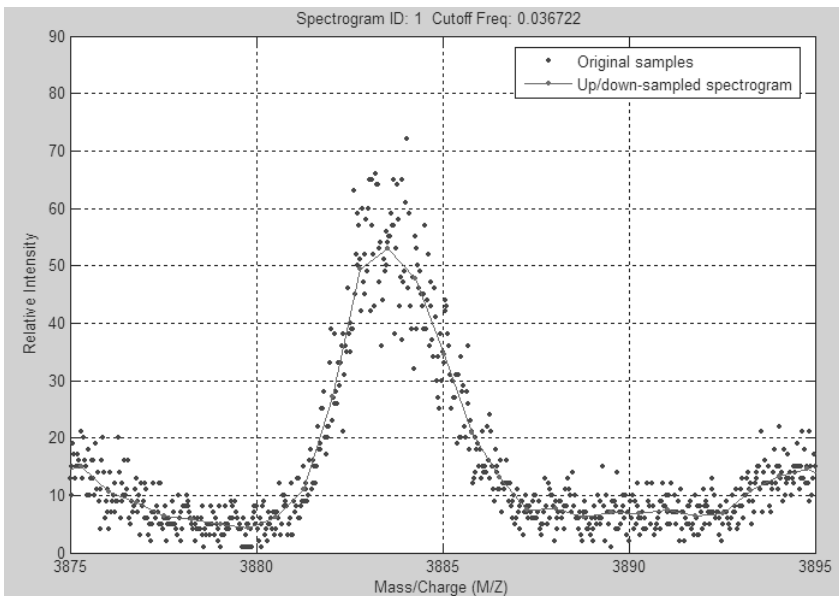


Рис. 13.41. Пример обработки спектроскопических данных

- программирование интерфейса – используется специальное программное обеспечение (BioPerl и BioJava), входящее в среду системы MATLAB.

Пакет Bioinformatics Toolbox предназначен прежде всего для ученых и инженеров в области биологии и геномной инженерии. Ученым он позволяет существенно расширить их инструментарий в области биологических и геномных исследований. Инженерам он поможет в проектировании сложнейших приборов геномной инженерии. Исключительно велика роль пакета в образовании, где недостаток средств на приобретение реальных сложных и дорогих физико-биологических приборов и систем часто удается преодолеть применением методов математического моделирования, лежащих в основе данного пакета расширения. Так что его пользователями могут быть преподаватели, аспиранты и студенты университетов и вузов.

13.9.2. Пакет расширения Genetic Algorithm and Direct Search Toolbox

Пакет по генетическим алгоритмам и алгоритмам прямого поиска (Genetic Algorithm and Direct Search Toolbox) впервые (версия 1.0) появился как дополнительный пакет расширения к программной системе MATLAB реализации 13 с Service pack 1. Версия данного пакета – версия 1.02 – не имеет существенных отличий от первоначального варианта и рассчитана на работу в среде системы MATLAB Version 7.0.1 и 7.04 (MATLAB 7 SP 2) и выше.

Пакет включает в себя функции, существенно дополняющие возможности пакета расширения оптимизации Optimization Toolbox и позволяющие значительно расширить круг решаемых оптимизационных задач. Он позволяет находить точки экстремума скалярных функций многих переменных, используя два принципиально различных типа алгоритмов:

- 1) генетические алгоритмы;
- 2) алгоритмы прямого поиска.

И те, и другие алгоритмы относятся к алгоритмам прямого поиска функции, поскольку для их работы не требуется знание градиента функции и производных более высокого порядка. При этом первая группа алгоритмов может быть отнесена к классу алгоритмов случайного поиска, вторая – к классу детерминированных алгоритмов. Для этих алгоритмов предусмотрена реализация в виде функций командной строки и утилит с GUI-интерфейсом.

При реализации генетических алгоритмов используется некоторое множество объектов (особей) – популяция. Оно меняется в ходе решения задачи, например поиска глобального минимума многоэкстремальной функции. При этом рождается новая популяция допустимых решений путем выбора лучших представителей предыдущего поколения, скрещивая их и получая множество новых особей. Это новое поколение содержит более высокие соотношения характеристик, которыми обладают хорошие члены предыдущего поколения. Таким образом, из поколения в поколение хорошие характеристики распространяются по всей популяции. Скрещивание наиболее приспособленных особей приводит к тому, что исследуют-

ся наиболее перспективные участки пространства поиска. В конечном итоге популяция будет сходиться к оптимальному решению задачи.

Итак, при размножении особей происходит слияние двух родительских половых клеток, и их ДНК взаимодействуют, образуя ДНК потомка. Основной способ взаимодействия – *кроссовер (crossover)*, или *скрещивание*. При кроссовере ДНК предков делятся на две части, а затем обмениваются своими половинками.

При наследовании возможны *мутации*, в результате которых могут измениться некоторые гены в половых клетках одного из родителей. Измененные гены передаются потомку и придают ему новые свойства. Если эти новые свойства полезны, они, скорее всего, сохранятся в данном виде. При этом произойдет скачкообразное повышение приспособленности вида.

Эти идеи и положены в основу реализации генетических алгоритмов пакета Genetic Algorithm and Direct Search Toolbox – рис. 13.42. По скорости определения оптимума целевой функции генетические алгоритмы на несколько порядков превосходят случайный поиск. Причина этому заключается в том, что большинство систем имеют довольно независимые подсистемы. Вследствие чего при обмене генетическим материалом от каждого из родителей берутся гены, соответствующие наиболее удачному варианту определенной подсистемы (неудачные варианты постепенно погибают). Генетический алгоритм позволяет накапливать удачные решения для таких систем в целом.

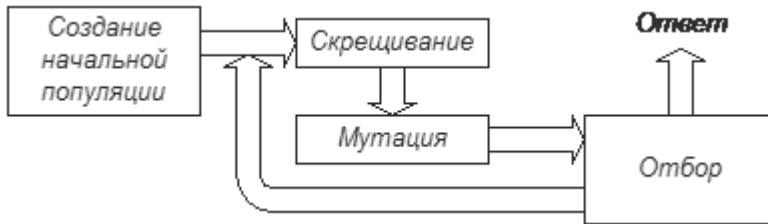


Рис. 13.42. Обобщенная схема генетического алгоритма

В качестве примера применения генетических алгоритмов можно привести нахождение глобального минимума многоэкстремальной функции. Рисунок 13.43 иллюстрирует исходный набор (популяцию) точек с произвольными координатами, выбранными вдали от глобального минимума.

В процессе выполнения алгоритма набор точек меняется от итерации к итерации, причем выбор осуществляется в пользу точек, дающих скорейший поиск минимума. На рис. 13.44 показано расположение точек на 60, 80, 96 и 100 итерациях. Хорошо видно, что точки не только смещаются к глобальному минимуму, но и сгущаются вплоть до почти полного совпадения.

Другая группа алгоритмов прямого поиска обеспечивает поиск по так называемому образцу. Она является обобщением алгоритма симплекс-метода Нелдера–Мида. При этом симплекс заменяется более общим объектом – образцом.

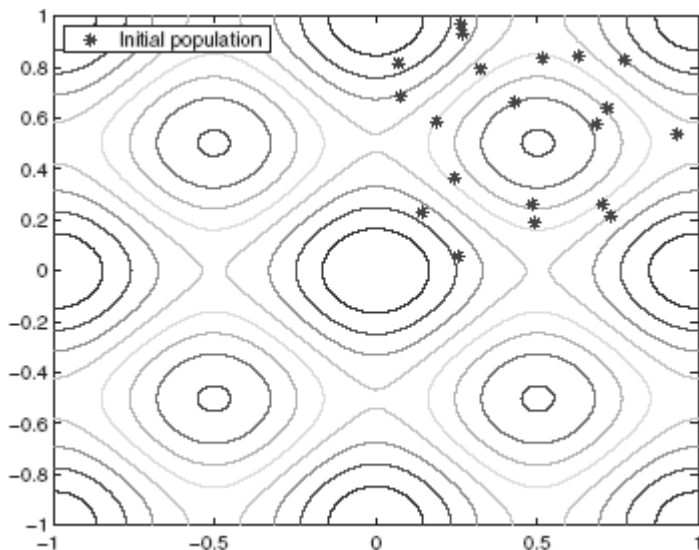


Рис. 13.43. Пример инициализации при генетическом алгоритме поиска глобального минимума многоэкстремальной функции двух переменных

13.9.3. Пакет расширения *Video and Image Processing Blockset*

Новейший пакет расширения Video and Image Processing Blockset 1.0 для Simulink 6 впервые появился в версии MATLAB 7 Service Pack 1. В последней версии Video and Image Processing Blockset 1.1 добавлены 10 новых блоков библиотеки и 3 новых демонстрационных примера.

Пакет Video and Image Processing Blockset 1.0.1 обеспечивает:

- создание и применение 2D-фильтров (FIR, медианного и свертки), 2D-преобразования (FFT, DCT, Hough) и геометрические преобразования (вращение, перемещение, разрезка изображений и др.);
- выполнение операций анализа изображений с применением усовершенствованных алгоритмов выделения кромок изображения, пороговых и морфологических преобразований, статистической обработки изображений;
- применение стандартных цветных изображений и видеопотоков и их обработку с использованием различных цветовых преобразований;
- моделирование и автоматическую генерацию кодов на языке C с использованием данных с плавающей и фиксированной точкой;
- оценка статуса видеопотоков в реальном масштабе времени и их просмотр с помощью соответствующих блоков.

Пакет Video and Image Processing Blockset обеспечивает эффективную поддержку как средств цветной графики, так и видеосредств, обеспечивающих работу

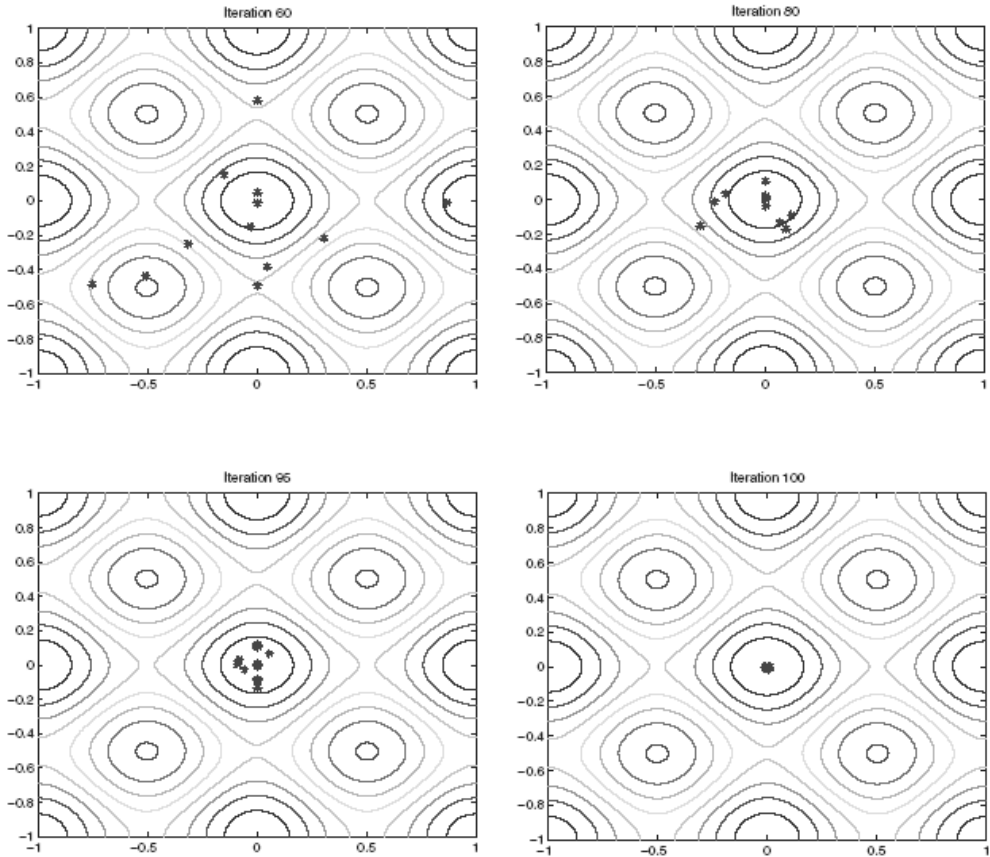


Рис. 13.44. Расположение «особей» генетического алгоритма в ходе поиска глобального минимума многоэкстремальной функции двух переменных

с видеофайлами. Он имеет собственные базовые примитивы и существенно усовершенствованные алгоритмы для обработки и представления изображений и видеофильмов. Это открывает обширные возможности применения данного пакета в художественной и научной графике, авиации, автомобилестроении, медицине, образовании и других областях.

Рисунок 13.45 иллюстрирует применение пакета для создания динамического кадра в кадре – средства, широко используемого в современных телевизорах и компьютерах.

На другом рисунке (рис. 14.46) показан пример построения панорамного изображения. Достаточно подробное описание пакета Video and Image Processing Blockset впервые представлено в книге [19].

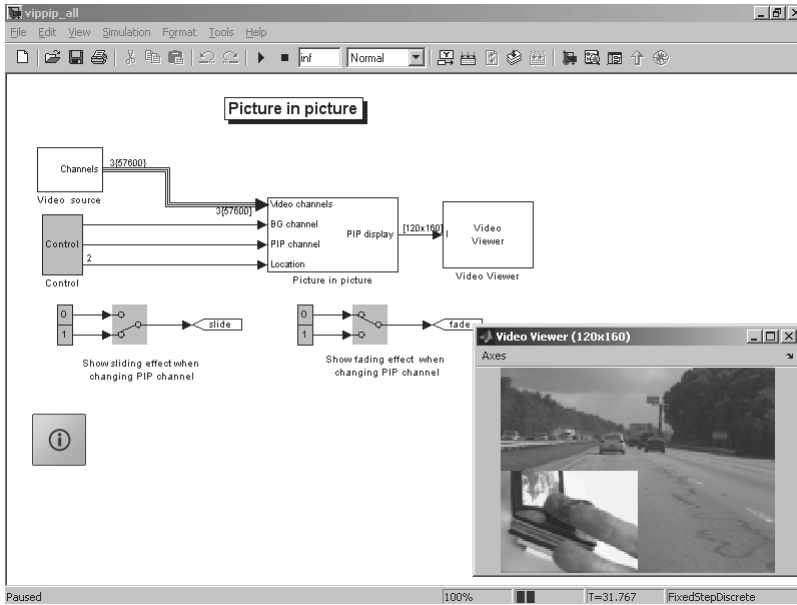


Рис. 13.45. Пример получения кадра в кадре

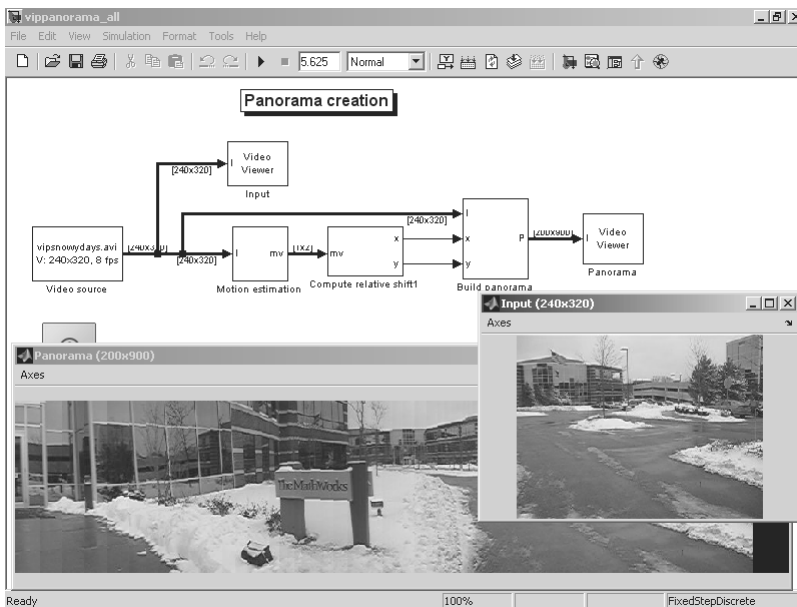


Рис. 13.46. Пример получения панорамного изображения

Стыковка MATLAB с измерительными приборами

| | |
|---|-----|
| 14.1. Работа измерительных приборов с системой MATLAB .. | 718 |
| 14.2. Стыковка компьютера с цифровым осциллографом ... | 721 |
| 14.3. Управление генераторами произвольных сигналов от системы MATLAB | 735 |
| 14.4. Применение MATLAB при совместной работе генератора и цифрового осциллографа | 739 |
| 14.5. Встраивание MATLAB в осциллографы, построенные на основе платформы ПК | 74 |

Этот урок имеет особенное значение. Впервые в литературе по системе MATLAB в нем описана прямая стыковка этой системы с современными и новейшими цифровыми измерительными приборами корпорации Tektronix – осциллографами серии TDS 1000B/2000B и генераторами произвольных функций (сигналов) серии AFG3000 [69–75].

14.1. Работа измерительных приборов с системой MATLAB

14.1.1. Современные измерительные приборы

В настоящее время широкое применение находят как отдельные измерительные приборы, так и измерительные системы, решающие задачи проведения и массовых, и уникальных измерений. Типичная измерительная система (рис. 14.1) состоит из генератора сигналов и регистратора, подключенных по тому или иному интерфейсу к персональному компьютеру. Генератор сигналов иногда объединяется с регистратором, например в анализаторах сигналов и источниках сигналов.

Наличие компьютера в системе позволяет управлять измерительными приборами извне (в том числе из Интернета [67, 68]) и автоматизировать многие виды

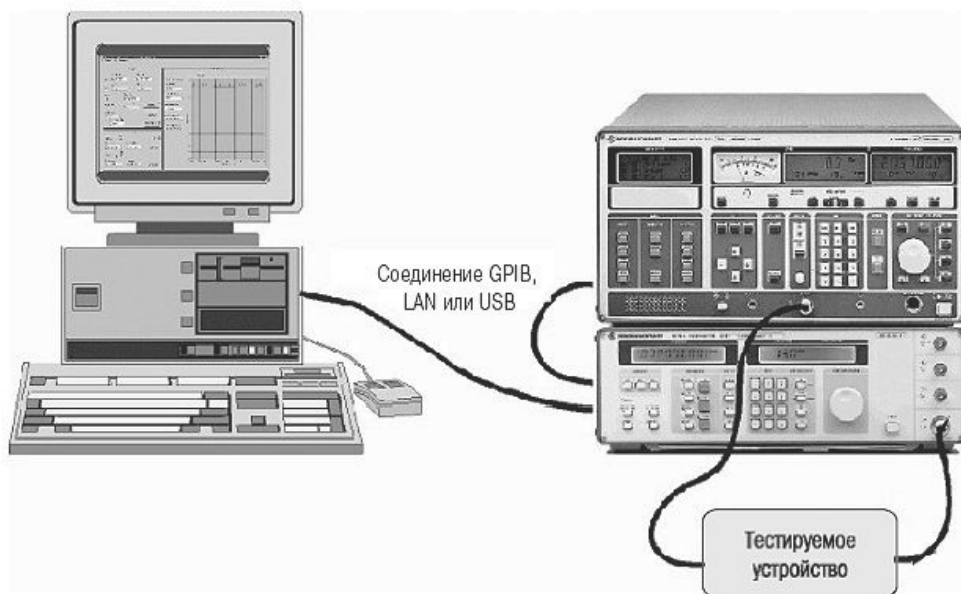


Рис. 14.1. Типичная измерительная система

измерений, например построение амплитудно-частотных характеристик устройств по многим точкам, построение спектров и спектрограмм и т. д.

Большую роль играет обработка сигналов, полученных от регистраторов, например цифровых осциллографов. Особенно большие возможности в обработке сигналов дают современные версии систем компьютерной математики (СКМ), математические и графические средства которых чрезвычайно разнообразны и обширны. К таким СКМ относятся Mathcad, MATLAB и др. Нередко применение компьютера позволяет выполнять те виды измерений, которые отсутствуют у применяемых в измерительных системах приборов или дополняют возможные измерения. Немаловажную роль имеет развитая система памяти (дисковой и на флэш-картах) современных ПК, позволяющая хранить и классифицировать большие массивы данных измерений, а также использовать возможности локальных сетей и Интернета.

Ниже рассматривается подключение к ПК и применение двух наиболее важных классов измерительных приборов – современных цифровых осциллографов и генераторов сигналов (функций) произвольной формы.

14.1.2. Порты для подключения измерительных приборов к компьютеру

Для подключения к современному компьютеру измерительных приборов (генераторов, осциллографов и др.) служат *порты* – совокупность аппаратных и программных средств, необходимых для обеспечения совместной работы компьютера с периферийным оборудованием – в нашем случае измерительными приборами. В состав портов входят разъемы, которые устанавливаются как на компьютере, так и на периферийных устройствах. Для аппаратного подключения последних к компьютеру используются специальные соединительные кабели.

Хотя в последнее время достигнут большой прогресс в создании беспроводных средств связи ПК с периферийным оборудованием (системы Bluetooth, WiFi и т. д.), подавляющее большинство приборов используют порты с проводной связью:

- принтерный порт LPT с 8-разрядной шиной передачи данных;
- последовательный COM-порт со скоростью передачи данных до 128/256 Кбит/с;
- порт универсальной последовательной шины USB;
- приборный порт GPIB;
- порт LAN для подключения к локальной сети.

В общем случае за каждым портом LPT или COM закрепляется его номер (начиная с 0), адрес Add и номер прерывания IRQ. В современных ПК используется специальный метод Plug and Play, обеспечивающий автоматическую расстановку этих параметров для портов и подключенных к ПК периферийных устройств, например генераторов, осциллографов и анализаторов. Как правило, эти устройства должны подключаться до включения компьютера и распознаются им после включения.

Порты LPT и COM, хотя и применяются по сей день, являются морально устаревшими и могут отсутствовать в ПК новейших разработок, особенно в портативных (ноутбуках). В меньшей мере устарел приборный интерфейс, реализованный портом GPIB. Но он используется при соединении приборов друг с другом.

Наиболее перспективным является порт универсальной последовательной шины USB. Стандарт USB разработали семь компаний: Compaq, Digital Equipment, IBM, Intel, Microsoft, NEC и Northern Telecom. Первые ПК с портами USB появились летом 1996 года. В настоящее время используются два стандарта шин USB.

Стандарт USB 1.1 имеет следующие технические характеристики:

- высокая скорость обмена – 12 Мбит/с;
- максимальная длина кабеля для высокой скорости обмена – 3 м;
- низкая скорость обмена – 1,5 Мбит/с;
- максимальная длина кабеля для низкой скорости обмена – 5 м;
- максимальное количество подключенных устройств (включая размножители) – 127;
- возможно подключение устройств с различными скоростями обмена;
- напряжение питания для периферийных устройств – 5 В;
- максимальный ток потребления на одно устройство – 500 мА.

USB 2.0 отличается от USB 1.1 только большей скоростью и небольшими изменениями в протоколе передачи данных для режима Hi-speed (480 Мбит/с). Существуют три скорости работы устройств USB 2.0:

- Low-speed 10–1500 Кбит/с (используется для интерактивных устройств: клавиатуры, мыши, джойстики);
- Full-speed 0,5–12 Мбит/с (аудио/видеоустройства, измерительные приборы);
- Hi-speed 25–480 Мбит/с (видеоустройства, устройства хранения информации).

В действительности скорость USB 2.0 в 480 Мбит/с на практике не реализуется. Это можно объяснить достаточно большими задержками шины USB между запросом на передачу данных и собственно началом передачи. К примеру, другая шина FireWire хотя и обеспечивает максимальную скорость в 400 Мбит/с, что на 80 Мбит/с меньше, чем у USB, в реальности позволяет достигать больших скоростей обмена данными с жесткими дисками и другими устройствами хранения информации.

Тем не менее шина USB интенсивно развивается. Так, вариант шины USB OTG обеспечивает легкое соединение периферийных USB-устройств друг с другом без необходимости подключения к ПК. Новейшая USB wireless позволяет организовать беспроводную связь с высокой скоростью передачи информации (до 480 Мбит/с на расстоянии 3 м и до 110 Мбит/с на расстоянии 10 м). В разработке находится спецификация USB 3.0, у которой теоретическая пиковая пропускная способность составит 4,8 Гбит/с.

Пока, однако, новые стандарты USB – дело будущего. Ниже описаны примеры совместной работы компьютера с измерительными приборами, имеющими порты

USB1.1 или USB2.0. Такими портами оснащены многие современные приборы корпорации Tektronix, явно лидирующей на рынке цифровых осциллографов и генераторов сигналов произвольной формы.

14.2. Стыковка компьютера с цифровым осциллографом

14.2.1. Современные цифровые осциллографы с USB-портом

Рассмотрим стыковку с компьютером наиболее массовых цифровых запоминающих осциллографов TDS 1000B/2000B корпорации Tektronix – рис. 14.2. Стоимость этих 2–4-канальных приборов – в пределах от 1 до 2,5 тыс. долл., так что они могут быть отнесены к разряду бюджетных приборов.



Рис. 14.2. Внешний вид цифровых запоминающих осциллографов TDS 1000B/2000B корпорации Tektronix

Основные технические характеристики осциллографов серии TDS 1000B/2000B представлены в таблице.

| Модель | Число каналов | Полоса частот | Частота выборки | Экран дисплея |
|----------|---------------|---------------|-----------------|---------------|
| TDS1001B | 2 | 40 МГц | 0,5 Гвыб/с | Монохромный |
| TDS1002B | 2 | 60 МГц | 1 Гвыб/с | Монохромный |
| TDS1012B | 2 | 100 МГц | 1 Гвыб/с | Монохромный |
| TDS2002B | 2 | 60 МГц | 1 Гвыб/с | Цветной |
| TDS2004B | 4 | 60 МГц | 1 Гвыб/с | Цветной |
| TDS2012B | 3 | 100 МГц | 1 Гвыб/с | Цветной |
| TDS2014B | 4 | 100 МГц | 1 Гвыб/с | Цветной |
| TDS2022B | 2 | 200 МГц | 2 Гвыб/с | Цветной |
| TDS2024B | 4 | 200 МГц | 2 Гвыб/с | Цветной |

Полоса частот приборов от 40 до 200 МГц и время нарастания до 1,8 нс открывают возможности исследования и отладки огромного числа схем на различных полупроводниковых и иных приборах и микросхемах. Похожие по внешнему виду и возможностям осциллографы TPS 1000/2000 имеют входы с гальванической развязкой и возможность питания от аккумуляторной батареи. Они очень удобны для исследования и тестирования источников электропитания и различных устройств энергетики и промышленной электроники. Приборы имеют возможность проведения курсорных измерений, 11 автоматических измерений и встроенный цифровой частотомер.

14.2.2. Применение пакета расширения MATLAB – Instrument Control Toolbox

Осциллографы указанного типа позволяют создавать файлы специального текстового формата .CSV. Но поддержка этого формата файлов системой MATLAB прямо не обеспечивается. Однако ее обеспечивает специальный пакет расширения Instrument Control Toolbox системы MATLAB, введенный в ее последние версии (см. урок 13). При этом обеспечивается поддержка виртуальных инструментов стандартной архитектуры VISA (Virtual Instrument Standard Architecture).

К сожалению, примеры применения этого пакета в его описании даны применительно к более старым моделям осциллографов, подключаемым к ПК через медленные порты – коммутационный RS-232 (COM) и приборный – GPIB. Поддержка соединения через USB-порт хотя и обеспечена пакетом Instrument Control Toolbox, но описана очень кратко и без реальных примеров применения. Это не удивительно, поскольку пакет Instrument Control Toolbox был создан до появления массовых осциллографов с USB-интерфейсом, в частности TDS1000B/2000B.

Сразу же отметим, что предполагается, что осциллограф серии TDS1000B/2000B подключен кабелем к порту USB и на ПК установлена СКМ MATLAB с пакетом расширения Instrument Control Toolbox. Практически использовалась версия СКМ MATLAB R2006b. Должна быть установлена и программа VISATek, поставляемая с осциллографами.

Пакет расширения Instrument Control Toolbox предоставляет для разработки программ стыковки осциллографа с системой MATLAB следующие основные функции:

- `instrhwinfo` – возвращает информацию о подключенном к ПК устройстве;
- `visa` – конструирование VISA-объекта;
- `fopen` – подключение VISA-объекта к прибору;
- `query` – запись или чтение форматированных данных с прибора;
- `fprintf` – запись текста в прибор;
- `fclose` – отключает связь с прибором;
- `binblockread` – чтение поблочно данных с прибора.

Для детального знакомства с каждой из этих функций достаточно в командном окне MATLAB исполнить команду

```
>> insthelp name
```

Здесь `name` – имя функции.

Перед проектированием программ надо убедиться в том, что на ПК установлена программа TekVISA. Для этого необходимо воспользоваться следующей командой:

```
>> tekvisainfo=instrhwinfo('visa','tek')
```

```
tekvisainfo =  
    AdaptorDllName: [1x67 char]  
    AdaptorDllVersion: 'Version 2.4.1'  
    AdaptorName: 'TEK'  
    AvailableChassis: []  
    AvailableSerialPorts: {'ASRL1'}  
    InstalledBoardIds: []  
    ObjectConstructorName: {'visa('tek', 'ASRL1::INSTR');'}  
    SerialPorts: {'ASRL1'}  
    VendorDllName: 'visa32.dll'  
    VendorDriverDescription: 'Tektronix VISA Driver'  
    VendorDriverVersion: 3
```

14.2.3. Идентификация осциллографа

Эта информация показывает на то, что изначально предполагается работа прибора с COM-портом ASRL1. Чтобы работать с портом USB, надо создать программу на языке системы MATLAB, определив при этом описание осциллографа. Для этого можно воспользоваться поставляемой с прибором программой OpenChoice [73]. На рис. 14.3 показан момент регистрации осциллографа TDS-2024B в окне этой программы. Для получения списка объектов надо активизировать кнопку Refresh. В данном случае осциллограф является одним из трех подключенных к ПК USB-объектов. Выделив нужный объект, его можно идентифицировать, активизируя кнопку **Identify?**. Имя объекта появится под списком объектов. Завершается идентификация активизацией кнопки **OK**.



Рис. 14.3. Начало регистрации подключенных к ПК приборов

Далее следует выяснить имя VISA-устройства, которым является применяемый осциллограф. Для этого надо открыть окно **Preferences** программы OpenChoice Desktop и активизировать (мышью) кнопку **VISA**. Появится окно **OpenChoice Instrument Manager** со списком доступных для регистрации приборов. Выделив осциллограф, надо нажать кнопку **Свойства**. Это приведет к появлению окна TDS2024B с данными о приборе – рис. 14.4. В нем указаны имя VISA-устройства и обычное имя прибора.

14.2.4. MATLAB-программы для работы с цифровыми осциллографами

На этом потребность в программе OpenChoice завершается, и ее можно закрыть и в дальнейшем использовать только тогда, когда нужны именно ее возможности – например, для получения на экране дисплея ПК точной копии экрана осциллографа. Все нужное для активизации осциллографа берет программа, созданная в среде MATLAB. Она должна начинаться с создания объекта типа USB-VISA, исполнив команду:

```
>> vu = visa('tek','USB0::1689::874::C010511::INSTR');
```

В ней первый параметр в прямых апострофах указывает на тип объекта – осциллограф фирмы Tektronix, а второй параметр – имя VISA-устройства, определение которого было описано выше. Это имя содержит указание на порт USB,

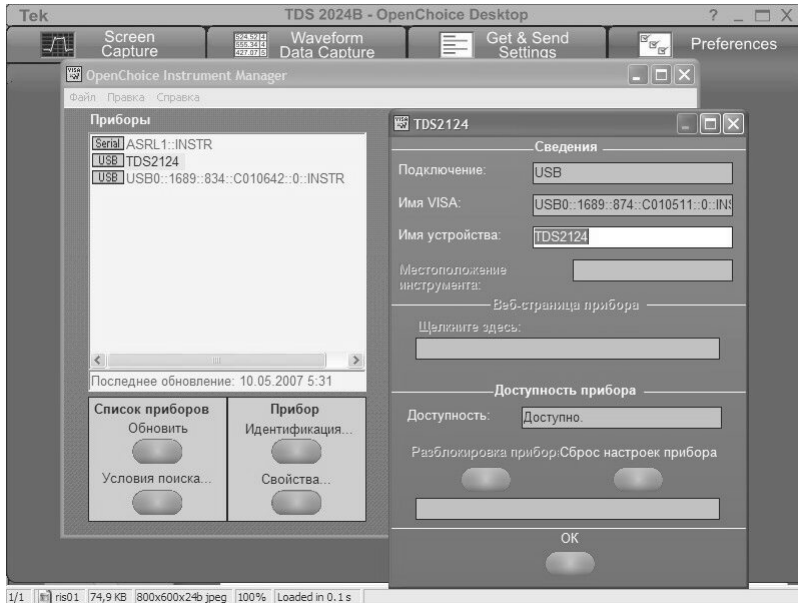


Рис. 14.4. Определение имени VISA-устройства для осциллографа TDS2024B

идентификационные номера устройства и его серийный номер. Важно обеспечить полную точность указания этих данных. Выполнение команды должно пройти гладко и закончиться приглашением MATLAB к дальнейшей работе в виде знака «>>». При этом происходит активизация VISA-объекта осциллографа. Исполнив команду `vu`, можно получить данные о созданном объекте.

Для детального знакомства с объектом `vu` можно использовать команды вызова окон инспектора объекта и обзора методов, используемых в этом программном объекте:

```
>> inspect(vu);
>> methodsview(vu);
```

Эти окна показаны на фоне окна сессии MATLAB на рис. 14.5. В них содержится детальная информация о созданном VISA-объекте – в нашем случае осциллографе TDS2024B.

Теперь создадим программу (M-файл) на языке системы MATLAB, которая обеспечивает активизацию осциллографа и передачу данных с памяти канала CH1 осциллографа в рабочую область (память) системы MATLAB с построением осциллограммы (см. рис. 14.6) в графическом окне системы MATLAB.

Для создания программы используется редактор M-файлов системы MATLAB. По завершении ввода программы надо записать файл с заданным именем, например `osc.m`. Эта программа представлена ниже и использует команды и функции пакета расширения Instrument Control Toolbox:

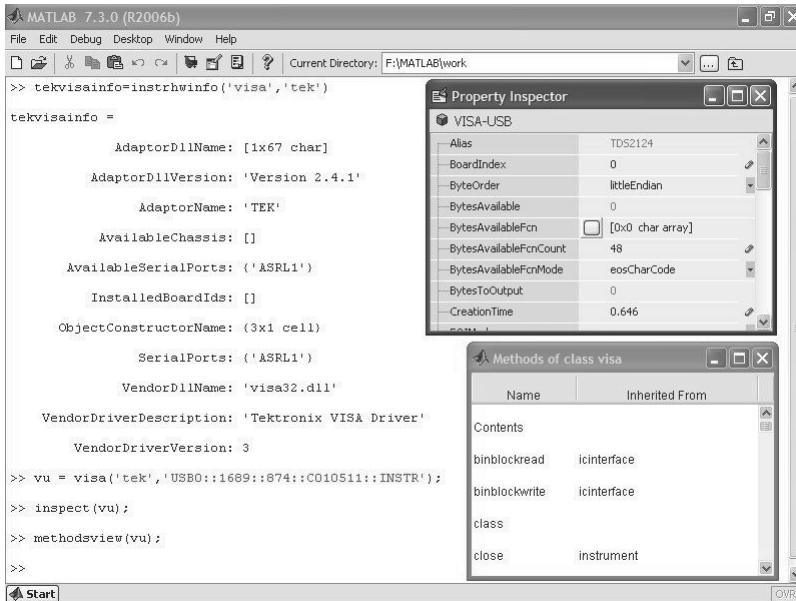


Рис. 14.5. Информация о VISA-объекте (осциллографе) в окнах системы MATLAB

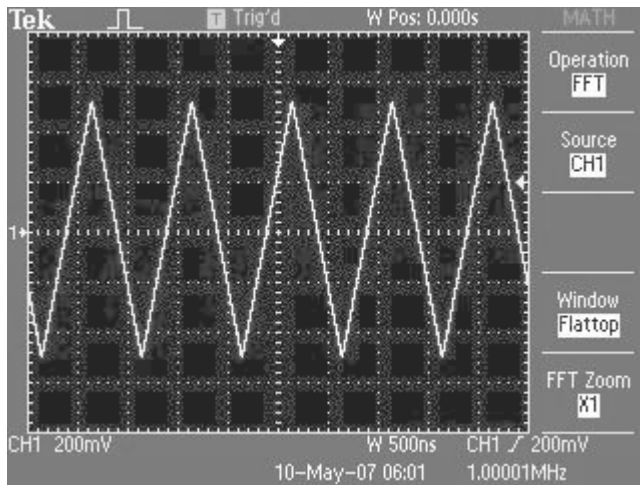


Рис. 14.6. Реальная осциллограмма треугольного сигнала

%Программа обеспечивает передачу данных с осциллографов
 %в рабочее пространство (память) системы MATLAB, создание
 %массивов xdata и ydata данных осциллограммы канала CH1
 %и определение параметров, нужных для построения графика

```

%осциллограммы в графическом окне системы MATLAB.
%Создание VISA-объекта
vu = visa('tek','USB0::1689::874::C010511::INSTR');
fopen(vu); %Открытие объекта vu
%Считывание данных с канала CH1 и определение длины записи
id=query(vu,'*IDN?');
fprintf(vu,'DATA:SOURCE CH1');
L=query(vu,'HORIZONTAL:RECORDLENGTH?','%s\n','%d');
fclose(vu); %Закрытие объекта
vu.InputBufferSize = L; %Задание длины входного буфера
fopen(vu) %Открытие объекта vu
%Считывание данных построения осциллограмм
fprintf(vu,'CURVE?')
data=binblockread(vu,'schar');
ymult = str2num(query(vu,'WFMP:YMULT?')); %Масштаб CH1
yoff = str2num(query(vu,'WFMP:YOFF?')); %Сдвиг CH1
xmult = str2num(query(vu,'WFMP:XINCR?')); %Масштаб по оси X
xoff = str2num(query(vu,'WFMP:PT_OFF?')); %Сдвиг по оси X
xzero = str2num(query(vu,'WFMP:XZERO?')); %Нуль на оси X
%Реконструкция данных для построения графики осциллограммы
ydata = ymult*(data - yoff); %Координаты точек по оси Y
xdata = xmult*((0:length(data)-1)-xoff)+xzero; %то же по оси X
%Построение осциллограммы в графическом окне MATLAB
plot(xdata,ydata)
title('Scaled Waveform Data'); ylabel('Amplitude (V)');
xlabel('Time (s)')
fclose(vu) %Закрытие объекта vu
Fs = 1/xmult; %Вычисление частоты отсчетов
NFFT = 1024; %Задание числа гармоник FFT

```

При исполнении данной программы (командой `osc` в окне командного режима MATLAB) осциллограф активизируется, и создается ряд массивов, которые видны в окне рабочего пространства системы MATLAB, показанном на рис. 14.7 слева. Справа виден график, построенный по полученным от осциллографа данным. Сравнение его с реальной осциллограммой (рис. 14.6) указывает на их полную идентичность. Из массивов наиболее важными являются `ydata` (значения координат точек осциллограмм по вертикали) и `xdata` (координаты точек по горизонтали). Важны также значения переменных масштаба и смещения по вертикальной и горизонтальной осям, положения нуля на горизонтальной оси, частота отсчетов `Fs` и число гармоник `NFFT`. Они обеспечивают реконструкцию полученных от осциллографа данных, что и позволяет строить рисунок осциллограммы в графическом окне MATLAB.

Вполне возможно считывание данных автоматических измерений осциллографа. Например, для считывания двойной амплитуды сигнала, представленного осциллограммой, перед последней строкой приведенной выше программы достаточно вставить фрагмент:

```

%Считывание данных измерения — двойной амплитуды
fprintf(vu,'MEASU:IMM:SOU CH1');

```

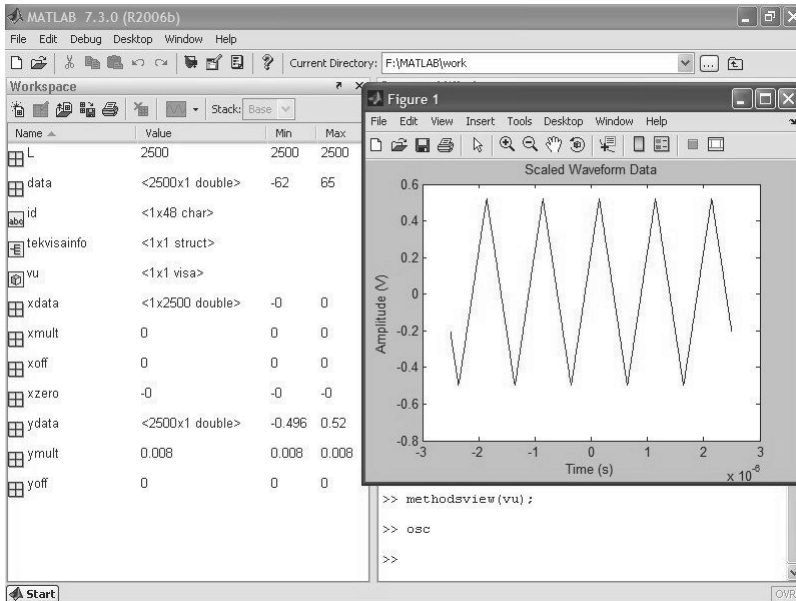


Рис. 14.7. Данные и график осциллограммы в системе MATLAB

```
fprintf(vu, 'MEASU:IMM:TYP PK2 ');
pk2pk = query(vu, 'MEASU:IMM:VAL?');
```

Тогда исполнение команды `osc` даст вывод значения двойной амплитуды:

```
>> osc
pk2pk =
1.0320000648E0
```

В данном случае на вход осциллографа был подан синусоидальный сигнал от генератора AFG3101 с двойной амплитудой 1 В.

14.2.5. Спектральный анализ осциллограмм в MATLAB

С полученными от осциллографа данными можно выполнять любые операции, которые предусмотрены в системе MATLAB и десятках пакетов расширения этой мощной системы компьютерной математики. Покажем это на весьма важных примерах проведения спектрального анализа полученной осциллограммы различными методами, которые не реализованы в самом приборе и позволяют расширить его возможности.

К примеру, осциллографы TDS1000B/2000B не предусматривают возможности проведения спектрального анализа в линейном масштабе (задан только логарифмический). Ниже представлена программа (M-файл) `спес_1`, выполняющая

вычисление и построение графика спектра с линейным масштабом для сигнала, отсчеты которого хранятся в векторе `ydata`:

```
%Вычисление и построение спектра в линейном масштабе
Y = fft(ydata,NFFT)/L;           %Задание БПФ
f = Fs/2.*linspace(0,1,NFFT/2); %Создание вектора частот
plot(f,2*abs(Y(1:NFFT/2)))      %Построение графика спектра
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('Y(f) |')
```

Для проведения спектрального анализа сигнала, осциллограмма которого имеется на экране осциллографа, надо вначале исполнить команду `osc` (данные от осциллографа вводятся в MATLAB) и после просмотра графика сигнала выполнить команду `ssec_1`. График будет заменен спектрограммой.

На рис. 14.8 показан пример импорта осциллограммы прямоугольного импульса – сигнала от генератора AFG3101 корпорации Tektronix. Масштаб по горизонтали выбран так, чтобы с одной стороны было представлено большое число периодов сигнала, а с другой стороны была видна форма импульсов.

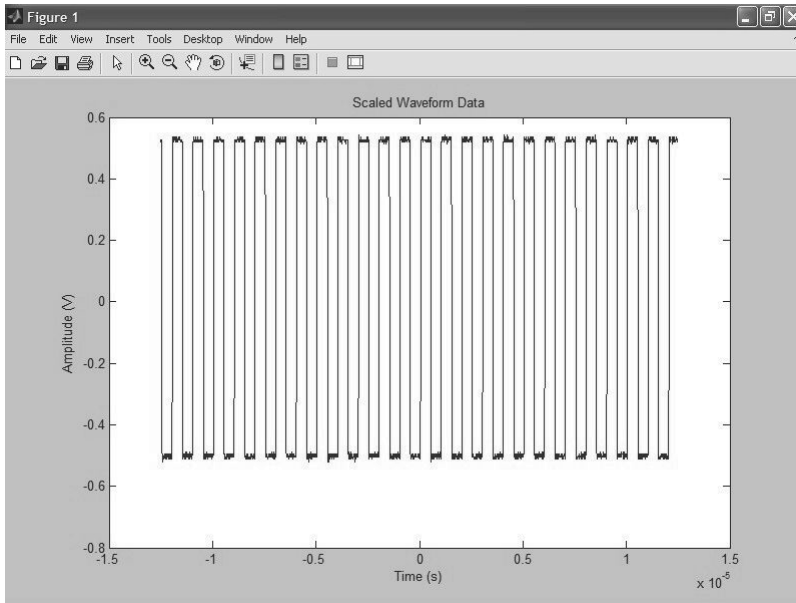


Рис. 14.8. Пример импорта прямоугольного импульса и построения его графика

На рис. 14.9 показан спектр прямоугольных импульсов с коэффициентом заполнения 10% и амплитудой 1 В. Он четко представляет гармоники спектра. В частности, отчетливо видно, что спектр имеет только нечетные гармоники, амплитуда которых убывает по мере увеличения номера гармоники.

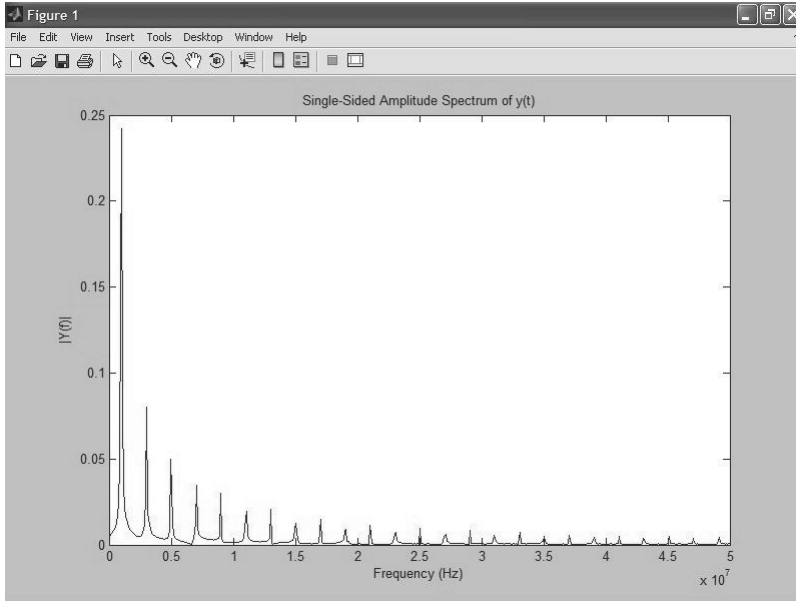


Рис. 14.9. Спектр прямоугольного импульса (рис. 14.8)

литуда которых убывает как $1/k$, где k – номер гармоники. Это полностью соответствует теоретическим представлениям о спектре меандра. Спектр характеризуется очень малым уровнем шума.

На рис. 14.10 показана осциллограмма синусоидального сигнала с частотой 1 МГц, засоренная шумом (сигнал получен также от генератора AFG3101). Масштаб по горизонтали выбран так, что осциллограмма выглядит просто как шумовая дорожка – никаких признаков наличия синусоидального сигнала не наблюдается.

На рис. 14.11 показан спектр сигнала, который показан на рис. 14.10. Весьма отчетливо видна единственная спектральная линия с пиком на частоте 1 МГц. Таким образом, в данном случае отчетливо выделен сигнал синусоидальной формы. О его синусоидальности говорит практически полное отсутствие других гармоник.

После того как данные сигнала осциллографа помещены в рабочее пространство (память) системы MATLAB командой `osc`, над ними можно проводить операции как с помощью программных модулей (см. примеры выше), так и командами, вводимыми в командном окне. Например, следующие команды обеспечивают получение спектрограммы (периодограммы) с окном Блэкмана–Харриса:

```
>> w = blackmanharris(2500);
>> periodogram(ydata,w,2500, Fs);
```

Для прямоугольного импульса с частотой 1 МГц и коэффициентом заполнения 5% периодограмма показана на рис. 14.12. В данном случае вычисляется спектр мощности сигнала в логарифмическом масштабе, что дает очень широкий

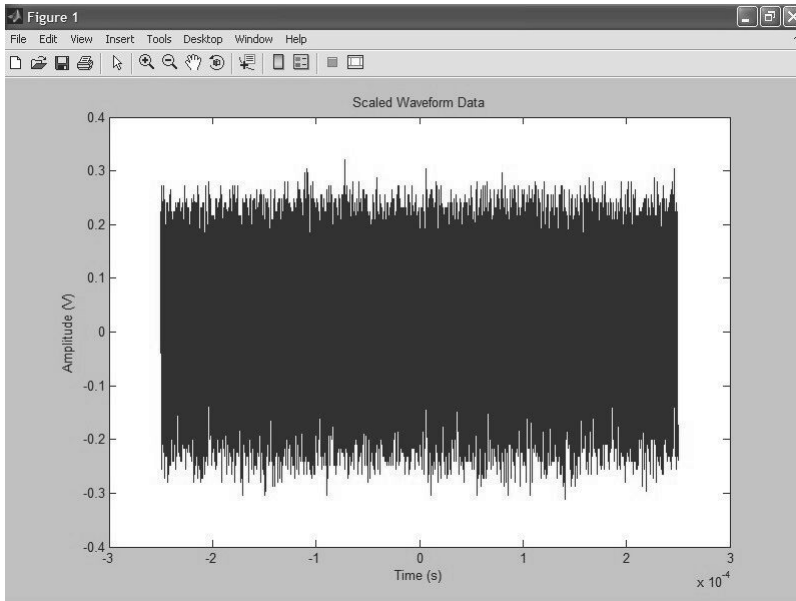


Рис. 14.10. Осциллограмма зашумленной синусоиды при большой длительности развертки

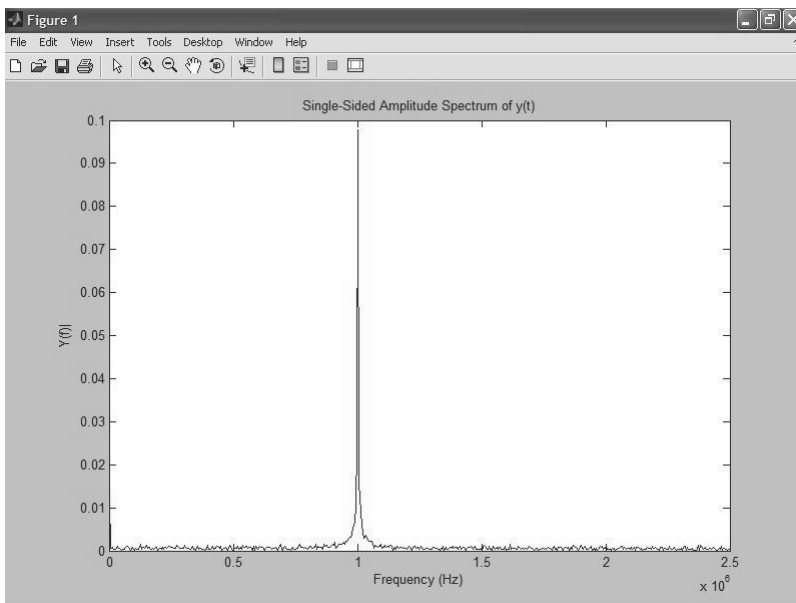


Рис. 14.11. Осциллограмма сигнала, показанного на рис. 14.10

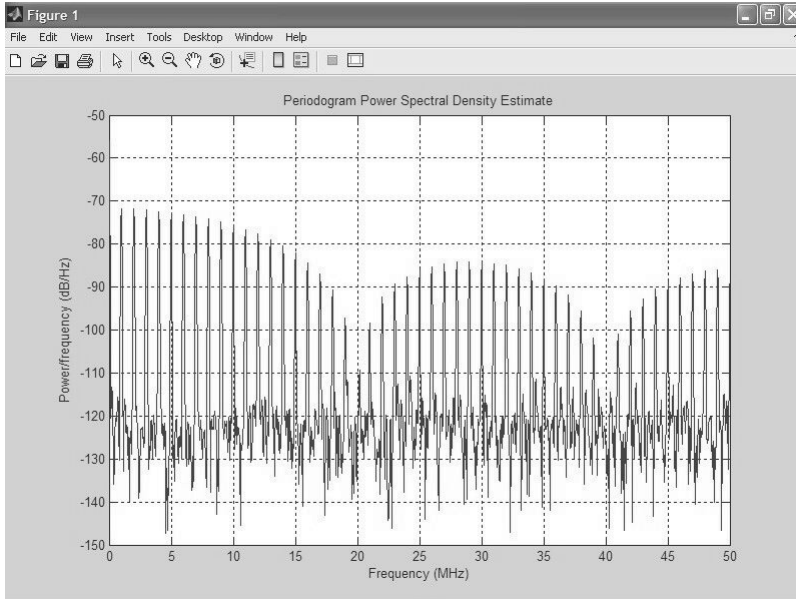


Рис. 14.12. MATLAB-периодограмма прямоугольного импульса

динамический диапазон периодограммы, в который входят и шумовые компоненты. Окно Блэкмана–Харриса эффективно выделяет гармоники спектра и подавляет шумовые компоненты. Уровень собственных боковых лепестков у этого окна ослаблен более чем на 100 дБ.

14.2.6. Построение спектрограмм осциллограмм в MATLAB

К сожалению, временное положение компонент сигнала обычный спектральный Фурье-анализ не выявляет. Для наглядной иллюстрации этого зададим (с помощью генератора AFG3101) сигнал в виде пачки из 10 периодов синусоидального зашумленного сигнала. После исполнения команды `osc` получим данные осциллограммы в рабочем пространстве MATLAB. Для получения осциллограммы и спектра в данном случае воспользуемся мощным средством пакета расширения Signal Processing Toolbox – инструментом анализа сигналов, фильтров и спектров SPTool. Запустив его командой `sptool`, можно из его окна загрузить массив `ydata` и наблюдать как сам сигнал, так и его спектр (см. рис. 14.13) при разных установках и разных видах спектрального анализа.

Высокая спектральная линия на спектрограмме отчетливо видна и говорит о наличии синусоидального сигнала с частотой 1 МГц. Однако о местоположении сигнала во времени и о его длительности спектрограмма не дает никаких

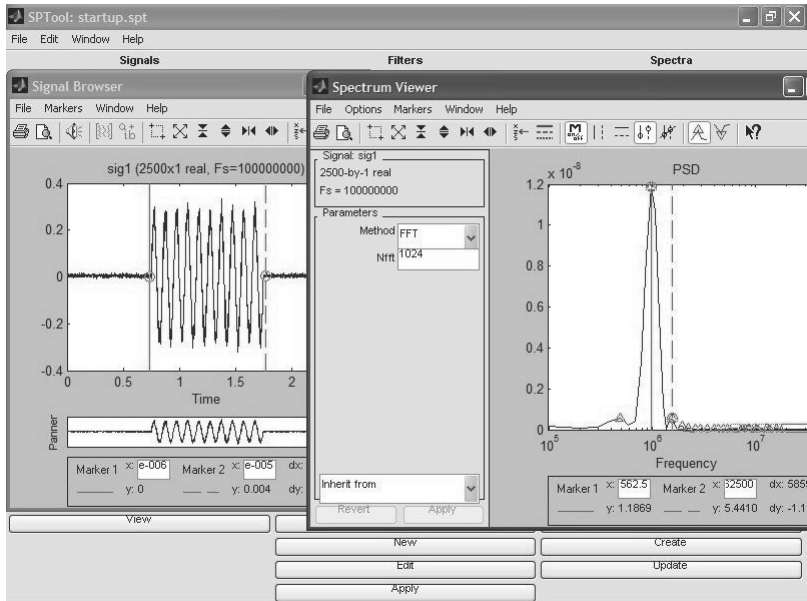


Рис. 14.13. Пример просмотра радиоимпульса и построения его спектра

намеков, хотя в этом средстве можно проводить спектральный анализ многими методами.

Функция `specgram` обеспечивает выполнение скользящего оконного БПФ и построение спектрограммы в плоскости частота–время с разбивкой времени на ряд участков, размер которой задается размером скользящего окна и длительностью сигнала. Интенсивность спектральных составляющих определяется цветом прямоугольников, из которых состоит спектрограмма. Например, для сигнала рис. 14.13 исполнение команды

```
>> specgram(ydata, 128, Fs)
```

создает спектрограмму, показанную на рис. 14.14. На ней среди шумовых компонент (хаотично разбросанные прямоугольники разного цвета) отчетливо выделяется область времени, в которой расположена компонента сигнала в виде пачки синусоид. Хорошо видно, что эта область занимает отрезок времени от 7,5 до 17,5 мкс, то есть местоположение основной компоненты сигнала и ее длительность четко определяются и совпадают с положением пачки синусоид на рис. 14.13. В указанной области снизу отчетливо видна сплошная темно-коричневая линия синусоидальной составляющей с частотой 1 МГц. На синусоидальность ее указывает отсутствие высших гармоник.

Поскольку ширина данной области равна 10 мкс, то из спектрограммы ясно, что компонента сигнала является пачкой из 10 синусоид! Спектрограммы со скользящим временным окном, таким образом, отчетливо выделяют особенности

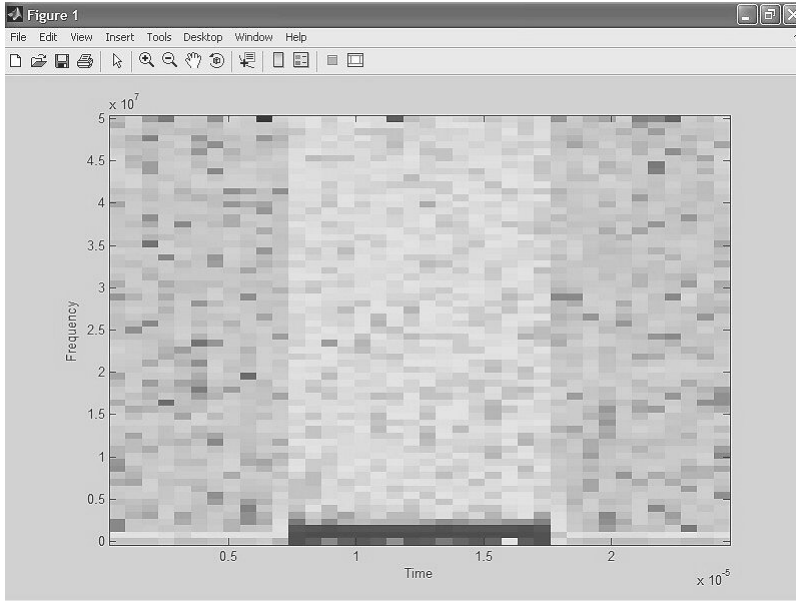


Рис. 14.14. Спектрограмма радиоимпульса

сигнала во временной области и позволяют оценивать параметры сигнала (начало появления его компонент, их длительность, временное положение), которые невозможно оценить обычным преобразованием Фурье. В некоторых случаях, как в приведенном примере, возможно даже выявление формы сигнала.

Приведенные примеры из области спектрального анализа демонстрируют лишь малую часть весьма обширных средств системы MATLAB, расширяющих возможности осциллографов. Так, для проведения спектрального анализа MATLAB имеет целый ряд функций, например оконного спектрального анализа с почти 20 видами окон. Для сравнения отметим, что спектральный анализ с помощью осциллографов TDS1000B/2000B возможен только при трех окнах. Есть даже функции спектрального анализа на основе новейших вейвлет-преобразований и проектирования фильтров. Возможности математической обработки сигналов и осциллограмм практически не ограничены.

Впрочем, нельзя не отметить и серьезное ограничение описанного подхода – обрабатываются только отдельные фрагменты сигналов, которые задаются осциллограммой, представляющей сигнал в определенном промежутке времени. Это значит, что работа в реальном масштабе времени не обеспечивается.

14.3. Управление генераторами произвольных сигналов от системы MATLAB

14.3.1. От множества генераторов к одному генератору произвольных сигналов

При проведении измерения в широком диапазоне частот еще недавно требовалось множество генераторов синусоидальных и импульсных сигналов. Ныне на смену многочисленным громоздким и тяжелым генераторам звуковых, ультразвуковых, высоких и сверхвысоких частот, а также различным типам импульсных и функциональных генераторов приходят многофункциональные генераторы произвольных функций и сигналов, например серий AFG3000, AWG5000, AWG7000 и др. корпорации Tektronix. Подобные приборы выпускает и ряд других фирм, например Agilent Technologies и др. Помимо цифрового синтеза более десятка сигналов стандартных форм (синусоидальной, прямоугольной, треугольной, пилообразной и т. д.), такие приборы позволяют синтезировать сигналы произвольной формы, заданной таблицей значений, математическими выражениями или графиками.

Возможности, параметры и особенности работы с бюджетными генераторами Tektronix AFG 3000 (рис. 14.15) достаточно полно описаны в [74, 75]. С помощью программы ArbExpress обеспечиваются связь приборов с персональным компью-

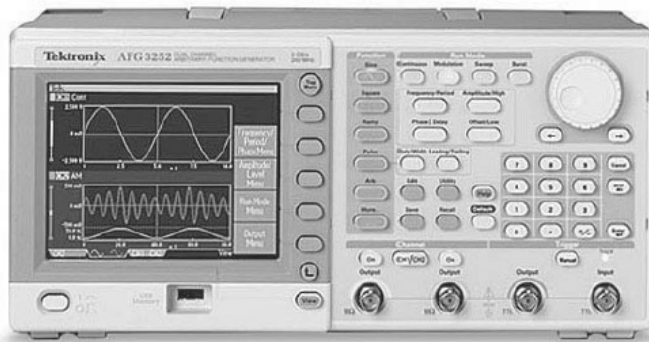


Рис. 14.15. Внешний вид генератора произвольных функций серии AFG 3000 корпорации Tektronix

тером, управление с ПК всеми их функциями и возможность задания сигналов, описываемых их графиками и математическими выражениями. Для обеспечения последней возможности программа ArbExpress имеет простой редактор математических выражений.

Однако возможности этого редактора намного уступают возможностям графического и формульного интерфейса пользователя современных СКМ. Последние позволяют создавать сигналы на основе специальных математических функций, решений алгебраических и дифференциальных уравнений, новых математических базисов (например, вейвлетов) и др. В связи с этим возникает актуальная задача синтеза сигналов с помощью существующих СКМ, таких как Excel, Mathcad, MATLAB, Mathematica, Maple и др. Это уже реализовано в новейших генераторах произвольных сигналов AWG5000 и AWG7000 класса HiFi корпорации Tektronix. А программа ArbExpress обеспечивает такую возможность и для куда более доступных (почти бюджетных), но более низкочастотных (до 240 МГц) генераторов AFG3000.

14.3.2. Управление генераторами серии AFG3000 от системы MATLAB

Опишем программирование форм сигналов для генераторов AFG3000 с помощью матричной СКМ MATLAB. Для обеспечения работы ArbExpress с СКМ MATLAB надо включить генератор AFG3000 и после загрузки микропрограммного обеспечения нажать кнопку меню **Сервис**. Необходимо записать идентификационный номер USB-порта, через который генератор подключается к компьютеру. Далее нужно переключить генератор в режим генерации произвольных функций (активизацией кнопки **Произвольн.** режимов работы) и загрузить программы ArbExpress и MATLAB.

Для обеспечения связи между программами ArbExpress имеет в каталоге Program Files\Tektronix\ArbExpress\tools\Matlab набор из нескольких функций, заданных в виде файлов с расширением .p. Их назначение можно найти в фирменном описании программы. На интернет-сайте корпорации Tektronix выложена доступная для загрузки несколько расширенная версия комплекта функций для совместной работы программ ArbExpress и MATLAB R2006b.

Ниже представлен конкретный пример программы на языке MATLAB, позволяющей задавать любую функциональную зависимость (в нашем случае синтез периода прямоугольного импульса по его первым 4 нечетным гармоникам с номерами 1, 3, 5 и 7). Эта программа вводится с помощью редактора M-файлов MATLAB [8] и сохраняется под каким-либо именем, например sample2.

```
Программа sample2 на языке MATLAB
echo off %Открытие сессии работы с генератором
s=NewSession('USB0::0x0699::0x0342::C010642::INSTR',...
'usb');
[status,idn]=query(s, '*idn?');
status=write(s, 'Output1:State On');
```

```

%Создание 1000 точек заданной функции
i = [1:1000]; w=2*pi.*i./1000;
Data = sin(w)+sin(3.*w)./3+sin(5.*w)./5+sin(7.*w)./7;
plot(i,Data); %Построение графика сигнала
%Преобразование данных в содержимое памяти генератора
TransferWfm(s, 'example.wfm', Data, 1000);
%Закрытие сессии работы с генератором
CloseSession(s);

```

Важно в функции открытия новой сессии `NewSession` правильно указать идентификационный номер USB (или LAN, GPIB) порта и обеспечить точное написание имен функций с учетом регистровой чувствительности новых реализаций MATLAB. Далее следует обеспечить активизацию подключения генератора к ПК, его идентификацию и соединение с компьютером с помощью окна программы `ArbExpress File Transfer and Control` (рис. 14.16). В подокне `ArbList` этого окна должна присутствовать ветвь с именем применяемого генератора (в нашем случае это `AFG3101`). Проверьте управление генератором с помощью программы `ArbExpress` – установка или снятие птички у опции **Output On** должно вызывать зажигание или потухание индикатора `Output` над входом генератора. После установки соединения активизацией кнопки **Connect** окно можно закрыть.

Теперь можно приступить к запуску программы `sample2` в среде MATLAB (использована реализация MATLAB R2006b, предоставленная автору разработчиком системы – корпорацией MathWorks). Важно перед этим установить текущую

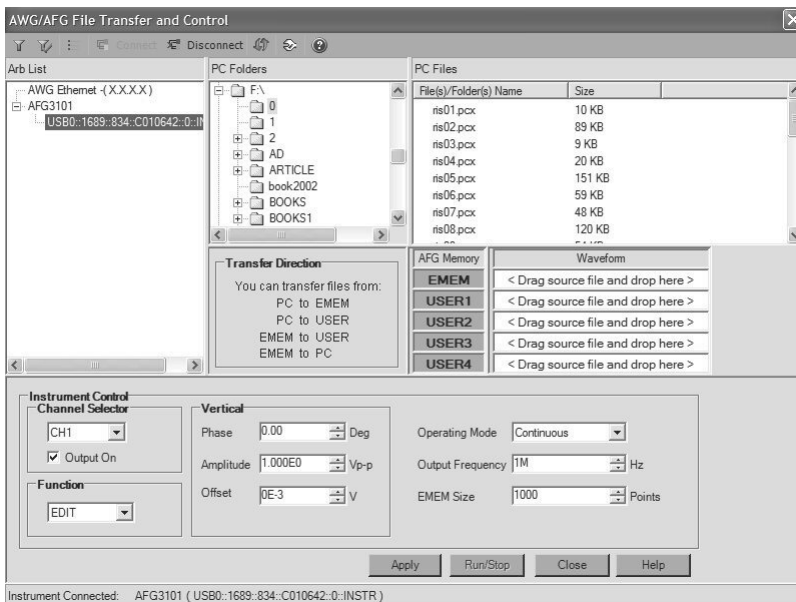


Рис. 14.16. Окно *File Transfer and Control* программы *ArbExpress*

директорию (окно **Current Directory**) на директорию, в которой хранятся файлы интерфейса программ, – см. рис. 14.17. При запуске программа выводит окно графика заданного сигнала, пересылает данные сигнала во внутреннюю память генератора и заканчивается выводом приглашения `>>` в окне командного режима работы MATLAB.

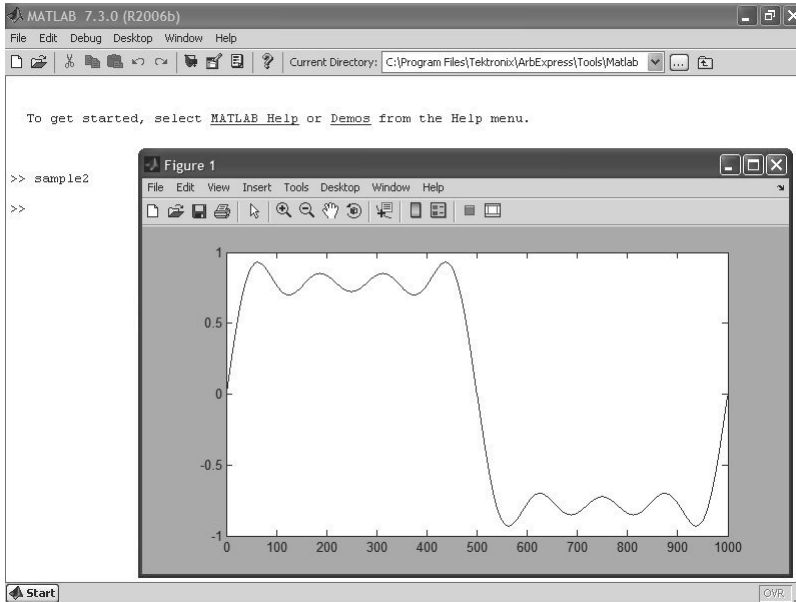


Рис. 14.17. Запуск программы `simple2` в окне программы MATLAB и вывод графика заданного сигнала

По окончании работы программы на экране генератора появляется график заданного сигнала – рис. 14.18. Его сравнение с графиком рис. 14.17 демонстрирует их полную идентичность. Таким образом, задача задания сигнала в СКМ MATLAB и загрузки его в память генератора AFG 3101 полностью решена. Окно **File Transfer and Control** можно использовать для переноса файлов (перетаскиванием мышью) из памяти генератора (основной и USER1,2,3,4) в компьютер и наоборот. Можно менять параметры сигнала в нижней части окна, с фиксацией изменений нажатием кнопки **Apply (Применить)**.



Рис. 14.18. Снимок экрана генератора AFG 3101 с графиком загруженного сигнала

14.4. Применение MATLAB при совместной работе генератора и цифрового осциллографа

На рис. 14.19 показана реальная осциллограмма сигнала с выхода генератора AFG3101, полученная с помощью цифрового осциллографа TDS2124B. Осциллограмма иллюстрирует полную идентичность представленным на рис. 14.17 и 4.18 сигналом, а также возможность получения непрерывного сигнала заданной формы. Рисунок 14.19 демонстрирует проведение осциллографом пяти (из возможных 11) автоматических измерений сигнала.

Теперь можно построить спектр этого сигнала. Используя меню **Math** осциллографа, зададим построение FFT (БПФ) при заданном по умолчанию прямоугольном окне и использовании усреднения по 16 осциллограммам для уменьшения влияния шума. Представленная на рис. 14.20 спектрограмма радует четким выделением всех четырех гармоник сигнала (1, 3, 5 и 7) и эффективным подавлением шума.

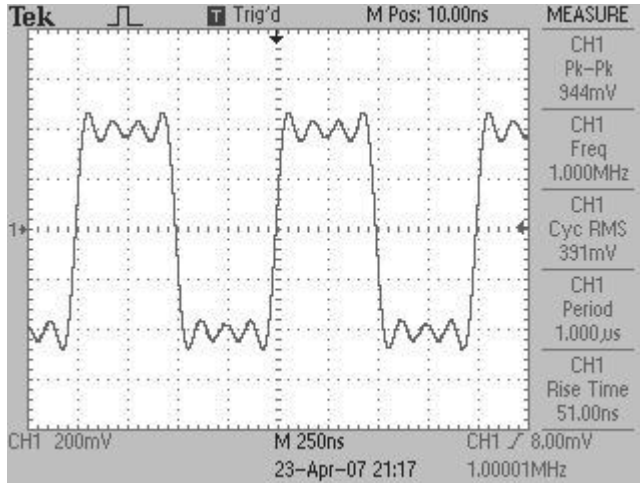


Рис. 14.19. Осциллограмма сигнала с выхода генератора AFG3101, отображаемая на экране цифрового осциллографа TDS2024B

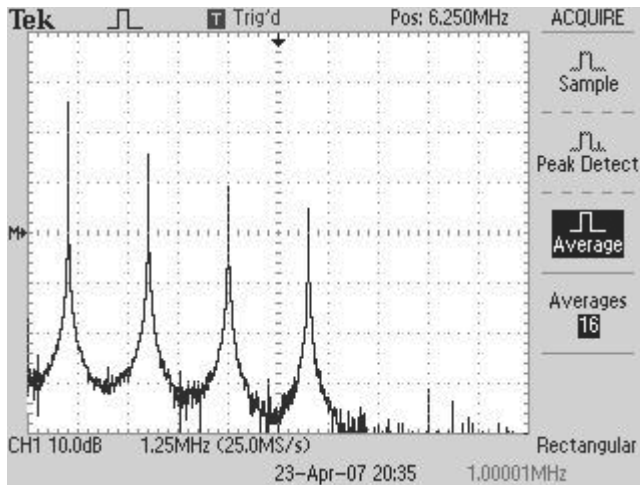


Рис. 14.20. Спектр заданного сигнала при прямоугольном окне, полученный с помощью цифрового осциллографа TDS2024B

Получение достаточно качественной картины спектров у массовых (бюджетных) осциллографов серии TDS-1000B/2000B кажется довольно неожиданным. Спектр простых сигналов при использовании других окон (см. рис. 14.21, например) у них выглядит почти как срисованный с учебников по спектральному анализу. Этому удивляться не стоит – недорогие приборы этих серий являются но-

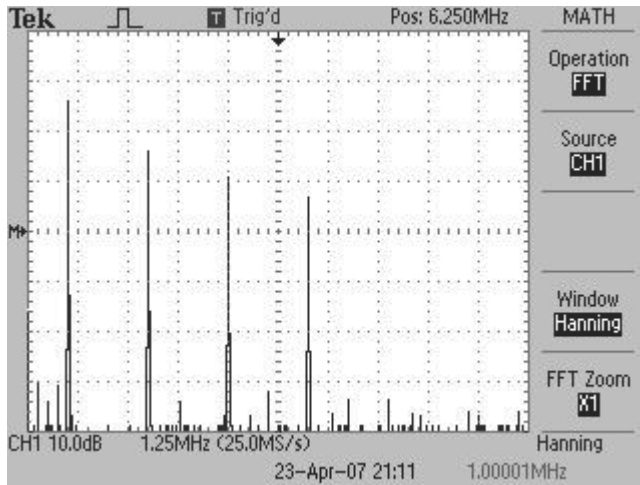


Рис. 14.21. Спектр заданного сигнала, полученный с помощью цифрового осциллографа TDS2024B, при использовании окна Хэннинга

вейшей разработкой корпорации Tektronix, и она постаралась включить в них лучшие алгоритмы обработки сигналов для получения спектров.

14.5. Встраивание MATLAB в осциллографы, построенные на основе платформы ПК

Наиболее совершенные цифровые осциллографы строятся на основе платформы ПК. Таковы, например, осциллографы корпорации Tektronix серий 5000, 6000, 7000 (рис. 14.22) и 70000. Некоторые из них относятся к категории анализаторов телекоммуникационных сигналов. В таких осциллографах имеются системная плата ПК, жесткий диск и внешние накопители, в том числе на основе карт флэш-памяти.

В подобные осциллографы система MATLAB может быть установлена прямо на их жесткий диск. При этом она становится частью общего программного обеспечения приборов и открывает обширные возможности в обработке сложных, например телекоммуникационных, сигналов. На рис. 14.23 показан пример построения трехмерной глазковой диаграммы осциллографом серии TDS7000 с помощью загруженной на его жесткий диск системы MATLAB с расширением Signal Processing Toolbox. Прямая работа системы MATLAB с измерительными приборами в этом случае также обеспечивается пакетом расширения Instrument Control Toolbox. Поставляемые с приборами корпорации Tektronix программные средства OpenChoice и TekVisa тоже обеспечивают программный интерфейс профессио-

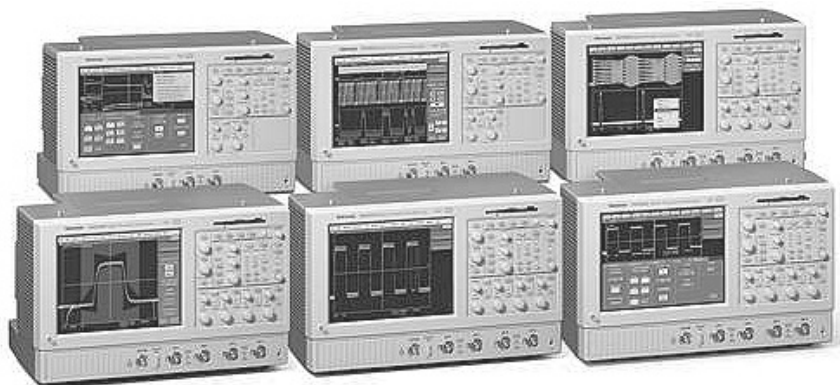


Рис. 14.22. Серия 7000 цифровых осциллографов корпорации Tektronix

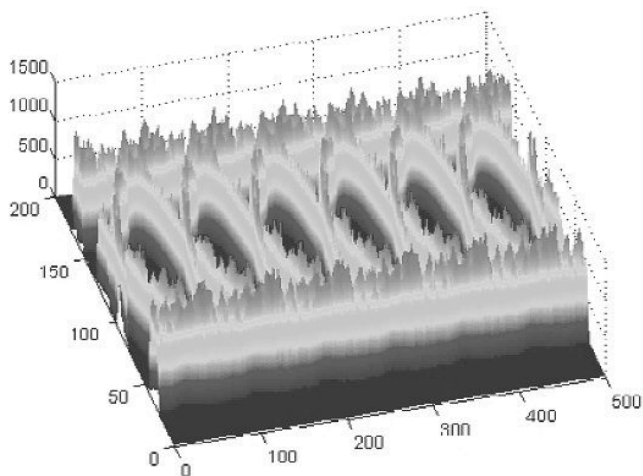


Рис. 14.23. Трехмерная глазковая диаграмма, полученная осциллографом серии TDS7000 и системой компьютерной математики MATLAB

нального уровня с системами компьютерной математики Excel, MATLAB и Mathcad и даже с текстовым процессором Word.

Выше мы рассмотрели лишь наиболее принципиальные вопросы применения компьютерной математики в измерительных приборах. Уже сейчас набор их средств огромен и включает в себя эффективные средства измерений и графической визуализации сигналов, построение специальных (в том числе трехмерных и динамических) спектрограмм и т. д. К сожалению, стоимость таких измерительных комплексов велика (доходит до \$ 100 000 и более), что ограничивает их массовое применение.

Список литературы

1. Дьяконов В. П. Компьютерная математика. Теория и практика. – М.: Нолидж, 2000.
2. Гантмахер Ф. Теория матриц. – М.: Наука, Физматлит, 1988.
3. Фадеев А. К., Фадеева В. Н. Вычислительные методы линейной алгебры. Изд. 3-е, стереотипное. – СПб.: Лань, 2002.
4. Дьяконов В.П. Справочник по применению системы PC MATLAB. – М.: Наука, Физматлит, 1993.
5. Дьяконов В. П., Абраменкова И. В. MATLAB 5.0/5.3. Система символьной математики. – М.: Нолидж, 1999.
6. Дьяконов В. П., Абраменкова И. В., Круглов В. В. MATLAB 5 с пакетами расширений. – М.: Нолидж, 2001.
7. Дьяконов В.П. MATLAB: Учебный курс. – СПб.: ПИТЕР, 2001.
8. Дьяконов В.П. MATLAB 6: Учебный курс. – СПб.: ПИТЕР, 2001.
9. Дьяконов В. П. Simulink 4: Специальный справочник. – СПб.: ПИТЕР, 2002.
10. Дьяконов В. П., Круглов В. В. Математические пакеты расширения MATLAB: Специальный справочник. – СПб.: ПИТЕР, 2001.
11. Дьяконов В. П., Круглов В. В. MATLAB. Анализ, идентификация и моделирование систем: Специальный справочник. – СПб.: ПИТЕР, 2002.
12. Дьяконов В. П., Абраменкова И. В. MATLAB. Обработка сигналов и изображений: Специальный справочник. – СПб.: ПИТЕР, 2002.
13. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5 в. Основы применения: Полное руководство пользователя. – М.: Солон-Р, 2002.
14. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5 в в математике и моделировании: Полное руководство пользователя. – М.: Солон-Р, 2002.
15. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5 в. Обработка сигналов и изображения: Полное руководство пользователя. – М.: Солон-Р, 2004.
16. Дьяконов В. П. MATLAB 6.5 SP1/7 + Simulink 5/6 в. Основы применения. – М.: Солон-Р, 2005.
17. Дьяконов В. П. MATLAB 6.5 SP1/7 + Simulink 5/6 в в математике и моделировании. – М.: Солон-Р, 2005.
18. Дьяконов В. П. MATLAB 6.5 SP1/7 + Simulink 5/6 в. Обработка сигналов и проектирование фильтров. – М.: Солон-Р, 2005.
19. Дьяконов В. П. MATLAB 6.5 SP1/7 + Simulink 5/6 в. Работа с изображениями и видеопотоками. – М.: Солон-Р, 2005.
20. Дьяконов В. П., Круглов В. В. MATLAB 6.5 SP1 7/7 SP1/7 SP2 + Simulink 5/6 в. Инструменты искусственного интеллекта и биоинформатики. – М.: Солон-ПРЕСС, 2006 .
21. Дьяконов В. П. VisSim+Mathcad+MATLAB. Визуальное математическое моделирование. – М.: Солон-Пресс, 2004.
22. Потемкин В. Г. Система MATLAB: Справочное пособие. – М.: Диалог-МИФИ, 1997.

23. Потемкин В. Г. Система инженерных и научных расчетов MATLAB 5.x. В 2 т. – М.: ДИАЛОГ-МИФИ, 1999.
24. Потемкин В. Г. Инструментальные средства MATLAB 5.x. – М.: ДИАЛОГ-МИФИ, 2000.
25. Потемкин В. Г. Вычисления в среде MATLAB. – М.: ДИАЛОГ-МИФИ, 2004.
26. Лазарев Ю. Ф. MATLAB 5.x. (серия «Библиотека студента»). – Киев: Издательская группа BHV, 2000.
27. Ануфриев. И. MATLAB 5.3/6.x: Самоучитель. – СПб.: БХВ-Петербург, 2002.
28. Ануфриев И. Е., Смирнов А. Б., Смирнова Е. Н. MATLAB 7. – СПб.: БХВ-Петербург, 2005.
29. Кетков Ю., Кетков А., Шульц М. MATLAB 6.x: программирование численных методов. – СПб.: БХВ-Петербург, 2004.
30. Кетков Ю. Л., Кетков А. Ю., Шульц М. М. MATLAB 7. Программирование, численные методы. – СПб.: БХВ-Петербург, 2005.
31. Чен К., Джиблин П., Ирвинг А. MATLAB в математических исследованиях. – М.: Мир, 2001.
32. Гулятьев, А. Визуальное моделирование в среде MATLAB: Учебный курс. – СПб.: Питер, 2000.
33. Черных И. В. SIMULINK. Среда создания инженерных приложений. – М.: ДИАЛОГ-МИФИ, 2004.
34. Лазарев Ю. Моделирование процессов и систем в MATLAB: Учебный курс. – СПб.: Питер; Киев: Изд. группа BHV, 2005.
35. Медведев В. С., Потемкин В. Г. Control System Toolbox. MATLAB 5 для студентов. – М.: ДИАЛОГ-МИФИ, 2004.
36. Рудаков П. И., Сафонов В. И. Обработка сигналов и изображений. MATLAB 5.x / Под общ. ред. В. Г. Потемкина. – М.: ДИАЛОГ-МИФИ, 2000.
37. Лавров К. Н., Цыплякова Т. П. Финансовая аналитика. MATLAB 6 / Под общ. ред. В. Г. Потемкина. – М.: Диалог-МИФИ, 2001.
38. Мартынов Н. Н., Иванов А. П. MATLAB 5.x. Вычисления, визуализация, программирование. – М.: КУДИЦ-ОБРАЗ, 2000.
39. Герман-Галкин С. Г. Компьютерное моделирование полупроводниковых систем в MATLAB 6.0. – СПб.: Корона, 2001.
40. Говорухин В., Цибулин В. Компьютер в математических исследованиях: Учебный курс. – СПб.: ПИТЕР, 2001.
41. Дьяконов В. П. Вейвлеты. От теории к практике. Изд. 2-е, доп. и перераб. – М.: СОЛОН-Пресс, 2004.
42. Медведев В. С., Потемкин В. Г. Нейронные сети. MATLAB 6. – М.: ДИАЛОГ-МИФИ, 2002.
43. Леоненков А. В. Нечеткое моделирование в среде MATLAB и fuzzyTECH. – СПб.: БХВ-Петербург, 2003.
44. MATLAB. The Language of Technical Computing. Getting Started with MATLAB. The Math Works, Inc. USA, 2000.

45. MATLAB. The Language of Technical Computing. Using MATLAB. The Math Works, Inc. USA, 2000.
46. MATLAB. The Language of Technical Computing. Using MATLAB Graphics. The Math Works, Inc. USA, 2000.
47. MATLAB. The Language of Technical Computing. External Interfaces. The Math Works, Inc. USA, 2000.
48. Simulink. Model-Based and System-Based Design. Using Simulink. The Math Works, Inc. USA, 2002.
49. Numerical Computing with MATLAB (text book) The Math Works, Inc. (www.mathworks.com/moler).
50. Математический энциклопедический словарь / Под ред. Ю. В. Прохорова. – М.: Советская энциклопедия, 1988.
51. Бронштейн И. Н., Семендяев К. А. Справочник по математике для инженеров и учащихся втузов. – М.: Наука, Физматлит, 1980.
52. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. – М.: Наука, 1973.
53. Справочник по специальным функциям с формулами, графиками и математическими таблицами / Под ред. М. Абрамовица и И. Стиган. – М.: Наука, Физматлит, 1979.
54. Ильин В. А., Позняк Э. Г. Основы математического анализа. В 2 ч. Изд-е 6-е. – М.: Физматлит, 2001.
55. Ильин В. А., Позняк Э. Г. Линейная алгебра. – М.: Физматлит, 2001.
56. Бабенко К. И. Основы численного анализа. – М.: Наука, Физматлит, 1986.
57. Марчук Г. И. Методы вычислительной математики. – М.: Наука, Физматлит, 1989.
58. Бахвалов Н.С., Жидков Н.П., Кобельков Г. М. Численные методы. – М.: Наука, Физматлит, 1987.
59. Трауб Дж. Итерационные методы решения уравнений. – М.: Мир, 1985.
60. Дэннис Дж., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений / Пер. с англ., под ред. Ю. Г. Евтушенко. – М.: Мир, 1988.
61. Иванов В. В. Методы вычислений на ЭВМ: Справочное пособие. – Киев: Наукова думка, 1986.
62. Дэннис Дж., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений / Пер. с англ., под ред. Ю. Г. Евтушенко. – М.: Мир, 1988.
63. Аоки М. Введение в методы оптимизации. – М.: Наука, 1977.
64. Банди Б. Методы оптимизации: Вводный курс. – М.: Радио и связь, 1988.
65. Шредер М. Фракталы, хаос, степенные законы. Миниатюры из бесконечного ряда. – Ижевск: НИЦ «Регулярная и хаотическая механика», 2001.
66. Роджерс Д., Адамс Дж. Математические основы машинной графики. – М.: Мир, 2001.
67. Новые информационные технологии: Учебное пособие / Под ред. В. П. Дьяконова. – М.: СОЛОН-Пресс, 2005.

68. Дьяконов В. П. Internet: Настольная книга пользователя. Изд. 5-е, перераб. и доп. – М.: СОЛОН-Пресс, 2005.
69. Дьяконов В. П. Современная осциллография и осциллографы. – М.: СОЛОН-Пресс, 2005.
70. Афонский А. А., Дьяконов В. П. Измерительные приборы и массовые электронные измерения / Под ред. В. П. Дьяконова. – М.: СОЛОН-Пресс, 2007.
71. Дьяконов В. П. Совместная работа генераторов произвольных функций Tektronix AFG3000 с осциллографами TDS1000B/2000B // Контрольно-измерительные приборы и системы. – 2007. – № 3.
72. Дьяконов В. П. Современная лаборатория разработчика электронных схем // Схемотехника. – 2007. – № 7, 8.
73. Дьяконов В. П. Работа цифровых осциллографов TDS1000B/2000B с системой компьютерной математики MATLAB // Схемотехника. – 2007. – № 7, 8.
74. Дьяконов В. П. Многофункциональные генераторы Tektronix AFG3000. Контрольно-измерительные приборы и системы // Схемотехника. – 2006. – № 6; 2007. – № 1.
75. Дьяконов В. П. Управление генераторами произвольных функций Tektronix AFG3000 с помощью программы ArbExpress // Контрольно-измерительные приборы и системы. – 2007. – № 2.

Предметный указатель

A

abs, функция, 168, 182
acos, функция, 173
acosh, функция, 177
acot, функция, 173
acoth, функция, 177
acsch, функция, 177
airy, функция, 183
alphadata, свойство прозрачности, 370
angle, функция, 182
ans, переменная, 53
ans, результат последней операции, 159
asec, функция, 173
asech, функция, 177
asin, функция, 173
asinh, функция, 177
atan, функция, 173
atan2, функция, 173
atanh, функция, 177
axis, функция, 319

B

balance, функция, 225
bar, функция, 284
barh, функция, 284
beer, функция или команда, 527
bench, тест на быстродействие, 79
besselh, функция, 184
besseli, функция, 185
besselj, функция Бесселя J_n , 184
besselk, функция, 185
bessely, функция Бесселя Y_n , 184
beta, бета-функция, 187

betainc, неполная бета-функция, 187
betaln, натуральный логарифм бета-функции, 187
bigc, функция, 390
bigstab, функция, 392
bin2dec, функция строковая, 502
bitand, функция, 161
bitget, функция, 162
bitmax, функция, 161
bitor, функция, 161
bitset, функция, 162
bitshift, функция, 162
break, оператор прерывания циклов, 560

C

calendar, функция календаря, 165
cat, функция, 199, 260
caxis, функция, 328
cd, команда, 579
cdf2rdf, функция, 228
ceil, функция, 180
cell, функция, 270
cell2struct, функция, 274
cellplot, команда, 271
cellstr, функция, 271
cgs, функция, 393
char, функция символьная, 495
checkin, команда, 575
checkout, команда, 575
chol, функция, 220
cholinc, функция, 252
clabel, функция, 319
clc, команда очистки основного окна, 49

Clear Command Window, команда, 100
clear, команда, 59
clock, функция времени, 165
close, функция, 507
csmopts, функция, 575
colmmd, функция, 242
colorbar, функция, 332
colormap, команда, 327
colormap, функция, 304
colperm, функция, 242
comet, команда, 342
comet3, команда, 342
comran, функция, 207
compass, функция, 290
computer, команда, 581
computer, функция, 159
continue, оператор продолжения, 560
Communications Toolbox, пакет по
 средствам телекоммуникации, 685
cond, функция, 216
condeig, функция, 216
condest, функция, 250
conj, функция, 182
contour, функция, 292
contour3, функция, 310
Control System Toolbox, пакет по
 системам контроля, 680
convhull, функция, 450
convhulln, функция, 451
corrcoef, функция, 447
cos, функция, 174
cosh, функция, 177
cot, функция, 174
coth, функция, 178
cov, функция, 448
cplxraic, функция, 445
cputime, функция, 166
Cray, чтение файлов компьютеров
 Cray, 508
cross, функция, 202
cruller, команда, 359
csc, функция, 174

csch, функция, 178
cumprod, функция, 202
cumsum, суммирование элементов
 массивов с накоплением, 203
cumtrapz, функция, 408
Curve Fitting Toolbox, пакет подгонки
 кривых, 704
cylinder, функция, 336

D

Data Acquisition, пакет сбора данных,
 701
datenum, функция, 166
datevec, функция, 167
dbclean, команда, 569
dbcont, команда, 570
dbdown, команда, 570
dblquad, функция, 410
dbstep, команда, 570
dbstop, команда, 569
dbststus, команда, 569
dbtype, команда, 569
dbup, команда, 570
deal, функция, 272
deblank, функция строковая, 495
dec2bin, функция строковая, 502
dec2hex, функция строковая, 502
deconv, функция, 412
del2, функция, 403
delaunay, функция, 449
delaunay3, функция, 450
delaunayn, функция, 450
delete, функция, 507
delete, команда, 581
demo, команда, 139
det, функция, 217
diag, функция, 200
Dials & Gauges Blockset, пакет
 расширенной визуализации, 708
diary, команда подготовки
 дневника, 47

diag, команда, 71
diff, функция, 404
Digital Signal Processing Blockset,
пакет по цифровой обработке
сигналов, 665
dir, команда, 580
dlmread, функция, 517
dlmwrite, функция, 517
dmperm, функция, 243
double, функция строковая, 495
Developer's Kit for TI DSP, пакет
для работы с сигнальными
процессорами, 707

E

e2pi, пример вычислений, 80
echo, команда включения/
выключения вывода, 49
echo, команда отключения вывода
m-файлов, 50
edit, команда, 101
eigs, функция, 255
ellipj, функция, 187
ellipke, функция, 188
eomday, функция, 167
eps, погрешность, 159
erf, функция ошибки, 188
erfc, дополнительная функция
ошибки, 188
erfc, функция, 188
erfinv, функция, 189
errorbar, функция, 286
etime, функция, 167
eval, функция строковая, 503
exit, команда, 74
exp, функция, 169
expint, интегральная показательная
функция, 189
expm, функция, 213
Extended Symbolic Math, пакет
символьных вычислений, 671

eye, функция, 194
ezplot, функция, 172

F

factor, функция, 169
fclose, команда, 508
feather, функция, 291
feature, команда, 570
feof, функция, 514
ferror, функция, 514
feval, функция строковая, 504
fft, функция, 455
fft2, функция, 457
fftn, функция, 457
fftshift, функция, 458
fgets, функция, 510
fieldnames, функция, 267
fill, функция, 331
fill3, функция, 334
filter, функция, 461
filter2, функция, 464
Financial Toolbox, пакет финансовых
расчетов, 699
find, функция, 237
findstr, функция строковая, 496
fix, функция, 180
Fixed-Point Blockset, пакет
расширения, 667
fliplr, функция, 201
floor, функция, 180
flow, массив визуализации струи, 377
fminbnd, функция, 398
fminsearch, функция, 399
format, команда, 56
fipud, функция, 201
fplot, функция, 85
fprintf, функция, 510
fread, функция, 509
frewind, команда, 514
fscanf, функция, 512
fseek, функция, 514

fsolve, функция, 396
 ftell, функция, 515
 full, функция, 238
 func2str, функция, 565
 functions, функция, 565
 funm, функция, 214
 Fuzzy Logic Toolbox, пакет нечеткой
 логики, 677
 fwrite, функция, 509
 fzero, функция, 394

G

gallery, функция, 207
 gamma, гамма-функция, 189
 gammainc, неполная
 гамма-функция, 189
 gammaln, логарифм
 гамма-функции, 189
 gcd, функция, 169
 get, функция, 314, 351
 getenv, команда, 581
 getfield, функция, 267
 global, объявление глобальных
 переменных, 543
 gmres, функция, 393
 gradient, функция, 407
 grid on/off, команда, 85
 grid, функция, 321
 griddata3, функция, 469
 griddatan, функция, 469
 gtext, функция, 315

H

hadamard, функция, 208
 Handle Graphics, дескрипторная
 графика, 82
 handle, графика, 347
 handle-функция, 64
 hankel, функция, 209
 help, команда вызова справки, 74

help elfun, вывод списка элементарных
 функций, 61
 help ops, вывод списка всех
 операторов, 61
 help specfun, вывод списка
 специальных функций, 61
 hess, функция, 230
 hex2dec, функция строковая, 503
 hex2num, функция строковая, 503
 hilb, функция, 209
 hist, функция, 285
 hold, команда, 322
 home, команда возврата курсора, 49
 HSV, цветовая система, 369
 hsvVrgb, функция, 369

I

i, мнимая единица, 160
 ifft, функция, 459
 ifft2, функция, 460
 ifftn, функция, 460
 Instrument Control Toolbox, пакет
 сбора данных, 701
 imag, функция, 182
 Image Processing Toolbox, пакет
 обработки изображений, 691
 iminfo, информация о графическом
 файле, 518
 Import Data, пункт меню File, 506
 imread, считывание с графического
 файла, 520
 imwrite, запись в файл
 изображения, 522
 Inf, бесконечность, 160
 inpolygon, функция, 452
 inputname, функция, 549
 Instrument Control Toolbox, пакет
 для работы с внешними
 устройствами, 706
 int2str, функция строковая, 500
 interp2, функция, 470

interp3, функция, 472
interp, функция, 473
intersect, функция, 162
inv, функция, 221
invhilb, функция, 209
ipermute, функция, 262
iscell, функция, 273
iscellstr, функция, 272
ischar, функция строковая, 495
isfields, функция, 267
isjava, функция, 562
ismember, функция, 163
isobject, функция, 562
isstruct, функция, 267

J

j, мнимая единица, 160

K

K>>, признак отладки программ, 568
keyboard, команда, 568
klein1, команда, 359
knot, построение фигуры-узла, 81
kron, функция, 203

L

LAPACK, пакет линейной алгебры, 222
lasterr, функция, 546
lcm, функция, 169
legend, функция, 316
legendre, функция Лежандра, 191
line, функция, 348
linspace, функция, 195
LMI Control Toolbox, пакет расширения, 686
load, команда считывания рабочей области, 47

load, команда, 72
log, функция, 169
log10, функция, 170
log2, функция, 170
loglog, функция, 282
logm, функция, 214
logspace, функция, 196
lookfor, команда, 77
lower, функция строковая, 496
LQ- и QR-разложения матриц, 222
lscov, функция, 386
lsqnonneg, функция, 386
lsqr, функция, 388
lu, функция, 222
luinc, функция, 253

M

m-файл как функция пользователя, 63
M-файл, функция
 простой пример, 542
 статус переменных, 543
M-файл-функция, 540
magic, функция, 68, 210
Mapping Toolbox, пакет картографии, 700
mat2str, функция строковая, 500
MATLAB
 Compiler, компилятор, 704
 браузер рабочей области, 93
 взаимодействие с ОС, 579
 как суперкалькулятор, 50
 открытость, 44
 панель инструментов, 92
 прямое выполнение команд ОС, 580
 расширяемость, 44
 типовая графика, 278
MATLAB матричная лаборатория, 35
MATLAB 6.0
 браузер файловой системы, 96
 вращение графиков мышью, 117
 меню основное, 97

панель Camera окна графики, 117
 редактор матриц, 95
 редактор/отладчик m-файлов, 101
 matlabrc, файл (команда) начального запуса, 46
 max, функция, 442
 mean, функция, 446
 Mechanical System Blockset, пакет моделирования в механике, 708
 median, функция, 446
 mesh, функция, 298
 meshc, функция, 301
 meshgrid, функция, 294
 meshz, функция, 301
 methods, функция, 564
 methodsview, функция, 564
 Microsoft Excel 97, процессор ввода-вывода, 703
 min, функция, 443
 minres, функция, 393
 mod, функция, 170
 Model Predictive Control Toolbox, пакет расширения, 684
 modes, команда, 360
 more on/off, включение/выключение постраничного вывода, 50
 mtg, массив черепной коробки человека, 373
 Mu-Analysis and Synthesis, пакет расширения, 685

N

NAG Foundation, пакет NAG алгоритмов, 671
 NaN, нечисловой результат, 160
 NaN, указатель неопределенности, 65
 nargchk, функция, 546
 nargin, функция, 547
 nargout, функция, 548
 nargoutchk, функция, 546
 NCD, пакет оптимизации нелинейных систем, 663

ndgrid, функция, 295
 ndims, функция, 262
 Neural Networks Toolbox, пакет по нейронным сетям, 678
 New file, кнопка, 93
 nextrow2, функция, 170
 nnz, функция, 240
 nonzeros, функция, 240
 norm, функция, 218
 normest, функция, 251
 null, функция, 218
 num2cell, функция, 273
 num2str, функция строковая, 501
 nzmax, функция, 240

O

ones, функция, 194
 Open file, кнопка, 93
 OpenGL
 поддержка, 369
 примеры применения, 370
 прозрачность, 370
 средства, 369
 openxxx, 505
 Optimization Toolbox, пакет оптимизации, 674
 orth, функция, 218

P

pack, дефрагментация рабочей области, 70
 pareto, команда, 574
 Partial Differential Equations, пакет расширения, 675
 pascal, функция, 210
 Paste Special, пункт меню Edit, 506
 patch, функция, 330
 Path Browser, кнопка, 96
 pcg, функция, 392
 pcolor, функция, 329
 peaks, функция, 293, 299

perm, функция, 201
permission, параметр, 507
permute, функция, 262
pi, число π , 76, 160
pie, функция, 333
pie3, функция, 335
pinv, функция, 221
plot, функция, 278
plot3, функция, 296
PNG, тип графических файлов, 520
polar, функция, 288
poly, функция, 412
polyarea, функция, 451
polyder, функция, 415
polyeig, функция, 415
polyfit, функция, 465
polyval, функция, 413
polyvalm, функция, 413
row2, функция, 170
Power System Blockset, пакет энергетических систем, 668
primes, функция, 171
prod, функция, 201
profile, команда, 571
profsumm, команда, 572
pwd, функция, 580

Q

qhull, алгоритм, 450, 451
qmr, функция, 394
qr, функция, 223
QR-разложение, 385
qrdelete, функция, 223
qrinsert, функция, 224
quad, функция, 410
quad8, функция, 409
Quantitative Feedback Theory Toolbox, пакет расширения, 686
quit, команда, 74
quiver, функция, 295
qz, функция, 228

R

rand, функция случайных чисел с равномерным распределением, 196
randn, функция, 198
randperm, функция, 196
rank, функция, 217
rat, rats, представление в виде цепной дроби, 171
rcond, функция, 216
Real Time Windows, пакет работы в реальном времени, 670
real, функция, 182
realmax, переменная, 160
realmin, переменная, 161
Redo, команда, 93
rem, функция, 181
rformat, функция, 204
Report Generator, генератор отчетов, 669
reshape, функция, 204, 261
residue, функция, 416
return, команда, 568
return, оператор возврата, 560
rgb2hsv, функция, 369
rmfield, функция, 268
Robust Control Toolbox, пакет расширения, 682
roots, функция, 414
rose, функция, 289
rot90, функция, 205
round, функция, 181
rref, функция, 219
rrefmovie, функция, 219
rsf2csf, функция, 230
Run, команда, 101

S

save, команда записи сессии, 47
Save As, команда, 101
save
команда, 71

- ключи, 71
 - saveas, функция, 506
 - schur, функция, 229
 - sec, функция, 174
 - sech, функция, 178
 - semilog, функция, 283
 - Set Patch, команда, 98
 - set, команда, 352
 - setdiff, функция, 163
 - setfield, функция, 268
 - setxor, функция, 163
 - SF-диаграмма, 667
 - shading interp, команда, 304
 - shading, команда, 328
 - shiftdim, функция, 263
 - sign, функция, 181
 - Simulink, пакет блочного моделирования систем, 660
 - sin, функция, 174
 - sinh, функция, 178
 - slice, функция, 308
 - solve, функция, 396
 - solver, функция, 420
 - sort, функция, 443
 - sortrows, функция, 444
 - sound, команда, 526
 - soundsc, команда, 526
 - spalloc, функция, 240
 - sparse, функция, 238
 - spconvert, функция, 239
 - spdiags, функция, 234
 - spreye, функция, 235
 - spfuns, функция, 240
 - spharm2, команда, 360
 - sphere, функция, 337
 - Spline Toolbox, пакет по сплайнам, 672
 - spline, функция, 474
 - spones, функция, 241
 - spparms, команда, 244
 - sprand, функция, 235
 - sprandn, функция, 236
 - sprandsym, функция, 236
 - sprang, функция, 252
 - sprintf, функция, 515
 - spru, функция, 241
 - SQL, обмен данными с СУБД, 703
 - sqrtm, функция, 215
 - squeeze, функция, 264
 - sscanf, функция, 516
 - stairs, функция, 285
 - Stateflow, пакет событийного моделирования, 667
 - Statistics Toolbox, пакет статистики, 673
 - std, функция, 447
 - stem, функция, 287
 - str2double, функция строковая, 501
 - str2func, функция, 565
 - str2num, функция строковая, 502
 - strcat, функция строковая, 497
 - strcmp, функция строковая, 498
 - strjust, функция строковая, 499
 - strcmp, функция строковая, 499
 - strtok, функция строковая, 499
 - struct, функция, 266
 - struct2cell, функция, 274
 - strvcat, функция строковая, 497
 - subplot, функция, 324
 - subspace, функция, 219
 - sum, функция, 203
 - surf, функция, 302
 - surfc, функция, 305
 - surf1, функция, 306
 - svd, функция, 227
 - svds, функция, 255
 - symmlq, функция, 393
 - symmmd, функция, 244
 - symrcm, функция, 244
 - System Identification Toolbox, пакет идентификации систем, 687
- T**
- Tab, клавиша, подсказка, 64
 - tan, функция, 174
 - tanh, функция, 178

tempdir, команда, 581
 terminal, команда, 582
 text, функция, 312
 title, функция, 312
 toeplitz, функция, 212
 toyu4, команда, 360
 trace, функция, 220
 trapz, функция, 408
 tril, функция, 205
 trimesh, функция, 338
 trisurf, функция, 338
 triu, функция, 206
 type name, вывод листинга файла
 name, 81

U

uigetfile, функция, 505
 uiimport, функция, 506
 uiopen, команда, 506
 uiputfile, функция, 506
 Undo, команда, 93
 undockcheckout, команда, 575
 union, функция, 164
 unique, функция, 164
 unwar, функция, 464
 upper, функция строковая, 496

V

varargin, системная переменная, 161
 varargout, системная
 переменная, 161
 VAX, чтение файлов компьютера
 VAX, 508
 voronoi, функция, 453
 voronoin, функция, 454

W

Warning, указатель
 предупреждений, 65

waterfall, функция, 310
 Wavelet Toolbox, пакет
 вейвлет-преобразований, 695
 wavread, команда, 527
 wavwrite, команда, 527
 web, команда, 580
 what, функция, 564
 which, функция, 565
 who, команда, 95
 whos, команда, 95
 wilkinson, функция, 213
 wk1read, функция, 517
 Workspace Browser, кнопка, 93

X, Y, Z

xlabel, функция, 312
 ylabel, функция, 312
 zeros, функция, 195
 zlabel, функция, 312
 zoom, команда, 324

Символы

(), операторы ввода скобок, 157
 -, унарный минус и знак
 вычитания, 56
 ., оператор точка, 158
 ... (многоточие), 53
 ./, оператор поэлементного,
 деления, 62
 :, оператор, 157
 ;, оператор задания
 последовательностей, 62
 @, оператор задания
 handle-функции, 64
 [], операторы задания
 массивов, 157
 { }, операторы задания массивов
 ячеек, 158
 3D-геометрический анализ, 479

А

- Анимация, волновые колебания мембраны, 345
- Адреса для переписки, 40
- Альфаканал (прозрачности), 520
- Анализ попадания точек в полигон, 452
- Анимация, 342
 - вихрей (смерчей), 379
 - команды, 344
 - логотипа MATLAB, 344
 - принцип, 344
- Аппаратные требования для установки, 44
- Аппроксимация, 465
- Аппроксимация производных конечно-разностная, 404
- АФХ. См. Амплитудно-фазовая характеристика, годограф
- АЧХ. См. Амплитудно-частотная характеристика

В

- Ввод диалоговый input, 552
- Вектор
 - норма, 217
 - точек в логарифмическом масштабе, 196
- Векторные операции простые примеры, 50
- Визуализация
 - разреженных матриц в целой степени, 249
 - расширенная, 375
 - струй конусной графикой, 381
 - электрических разрядов, 378
- Виртуальная реальность VRML, 703
- Вывод
 - предупреждающих сообщений, 545

- результатов промежуточных вычислений, 567
- сообщений об ошибках, 545
- списка значений свойств объекта, 352
- Выделение
 - части графика мышью, 325
 - части объема, 375
- Вызов списка демонстрационных примеров, 78
- Вычисление
 - градиента функции, 407
 - корней функции одной переменной, 394
 - площади полигона, 451
 - точек выпуклой оболочки, 450
 - тройного интеграла в MATLAB 6.5, 411

Г

- Галерея трехмерной графики, 357
- Гамма-функция, 189
- Гистограмма, 285
- Гистограммы угловые, 289
- Граф
 - задание, 245
 - молекулы (bucky), 246
 - смежности
 - сильные компоненты Холла, 243
 - фигуры ромба, 245
- График
 - 3D-типа с функциональной окраской, 304
 - в полярной системе координат, 289
 - вывод легенды, 127
 - выделенного мышью участка, 325
 - вырезки из черепной коробки человека, 375
 - гамма-функции, 190
 - гистограммы, 285
 - движения «кометы», 342

- движения «кометы»
 - в пространстве, 342
 - двух функций, 278
 - диаграммы столбцовой, 284
 - диаграммы столбцовой
 - горизонтальной, 284
 - дискретный, 288
 - комбинированный в одном окне, 324
 - комплексной функции, 279
 - контурный с маркировкой
 - линий, 319
 - линий поверхности, 294, 297
 - многоугольника окрашенного, 330
 - многоугольников
 - в пространстве, 333
 - многоугольников со шкалой
 - цветов, 332
 - нанесение надписи, 125
 - нескольких срезов черепной коробки, 373
 - нескольких функций, 84
 - окрашенных многоугольников
 - в пространстве, 334
 - освещенной поверхности, 307
 - поверхности peaks, 299
 - поверхности с кружками, 297
 - поверхности сетчатый, 298
 - поверхности сетчатый, цветной, 299
 - поверхности слоеный, 310
 - поверхности со шкалой
 - оттенков, 304
 - поверхности столбцовый, 302
 - поверхности цветной, 303
 - поверхности цветной
 - с проекцией, 305
 - погрешности аппроксимации, 487
 - поля градиентов, 296
 - проекции векторов
 - на плоскость, 292
 - производной функции, 406
 - радиус-векторов, 291
 - разреза черепной коробки человека, 373
 - с наложением ряда кривых, 323
 - с областями ошибок, 287
 - сечения поверхности, 309
 - синусоиды, 83
 - случайных чисел, 197
 - спектральной плотности
 - зашумленного сигнала, 456
 - сферы, 337
 - точек с нормальным
 - распределением, 198
 - трех функций, 281
 - угловой гистограммы, 290
 - фигуры из треугольных ячеек, 338
 - функции $\exp(x)/x$, 282
 - цветной поверхности со шкалой
 - цветов, 305
 - цветной фигуры из треугольных ячеек, 338
 - цилиндра, 336
 - четырёхугольника
 - закрашенного, 331
 - экспоненциальной функции, 283
- Графика
- выделение, 119
 - дескрипторная, 347
 - дескрипторная трехмерная, 372
 - иерархия объектов, 356
 - координатные оси и управление ими, 348
 - палитры цветов, 327
 - пример создания кнопки, 364
 - пространственного векторного поля, 359
 - специальная, 342
 - файлы построения трехмерных фигур, 358
- Графики
- алгебраических функций, 172
 - в декартовой системе
 - координат, 278

гиперболических функций, 178
 изменение масштаба, 127
 комбинаций тригонометрических функций, 174
 контурные, 292
 лестничные, 285
 нескольких функций одной переменной, 84
 обратных гиперболических функций, 178
 поверхностей (3D-графики), 86
 поля градиентов, 295
 трехмерные контурные, 310
 тригонометрических функций, 174
 функций Бесселя, 186
 функций одной переменной, 83
 Графиков 3D-анимация, 129
 Графическая «лупа», 126
 Графические форматы, 518
 Графов теория
 максимальное сечение,
 максимальное соответствие, 252

Д

Данные
 виртуальные array и numeric, 534
 задаваемые пользователем
 UserObject, 534
 многомерные массивы, 534
 структура типов, 533
 Деление
 массивов левое, 385
 массивов правое, 385
 Дескриптор, 82
 объекта класса surface, 329
 Дескрипторы
 графических объектов, 348
 Диагональ
 в терминологии MATLAB, 200
 Диаграмма
 Вороного, 449, 453

круговая, 333
 круговая объемная, 335
 Парето, 574
 профилирования M-файла, 572
 столбцовая, 284
 цветная плоская круговая, 333

З

завершение работы, 74
 задание
 строк, 494
 Записи, 264
 структура, 264
 запуск MATLAB, 46

И

Идентификатор, имя объекта, 59
 Изменение свойств объектов, 352
 Изображение черепной коробки,
 срезанной сверху, 375
 Интегрирование численное, 408
 Интерполяция, 465
 n-мерная табличная, 473
 двумерная табличная, 470
 на неравномерной сетке
 griddata, 467
 одномерная табличная interp1, 470
 периодических функций на основе
 БПФ interpft, 467
 поверхностей двумерная, 475
 при контурных графиках
 поверхностей, 478
 сплайновая, 469
 сплайновая в графическом
 окне, 490
 сплайновая кубическая spline, 474
 таблично заданных функций, 467
 трехмерная табличная, 472
 эрмитовая в графическом
 окне, 491

Исполнение программных объектов, 538

К

Кавычка внутри строки, 158

Карта прозрачности, 370

Кнопки

Cut, Copy и Paste панели инструментов, 93

панели инструментов редактора/отладчика m-файлов, 103

Команды строчного редактора, 49

Комментарии, 58

Комментарий программный, 547

Компиляторы для MATLAB, 533

Константы, 57

символьные, 58

числовые, 57

Контроль числа переменных функции, 547

Корреляция данных, 447

Кронекера тензорное произведение, 203

Л

Лапласиана аппроксимация, 403

Лента Мебиуса, 359

Линейная алгебра, 213

М

Маркировка линий контурных графиков, 319

Массив равноотстоящих узлов, 195

Массива

двумерного транспонирование, 262

размерность, 256

Массива

размер, 256

расширение, 257

число строк, 257

Массивы многомерные, 255

Массивы многомерные

вычисление числа

размерностей, 262

доступ к элементам, 258

заполнение страниц, 259

объединение (конкатенация), 260

перестановки размерностей, 262

применение функций ones, zeros, rand и randn, 259

сдвиг размерностей, 263

создание и применение, 257

удаление единичных размерностей, 264

Массивы ячеек, 269

вложенные, 275

графическая визуализация, 271

многомерные, 274

присваивание, 272

присваивание данных, 269

создание из строк, 271

создание функцией, 270

тестирование имен, 273

Мастер импорта, 506

Математика, определение, 704

Математическое выражение, 55

Матриц

вычисление ранга, 385

конкатенация (объединение), 199

линейное умножение, 384

объединение (конкатенация), 69

ортонормированный базис, 218

поэлементное сложение и вычитание, 384

приведение к треугольной форме, 219

разреженных алгоритмы упорядочения, 242

разреженных визуализация, 241

разреженных ранг sprank, 252

- разреженных сингулярные числа, 255
- разреженных собственные значения, 255
- разреженных числа обусловленности, 250
- транспонирование, 69
- угол между подпространствами, 219
- Матрица
 - Адамара, 208
 - Ганкеля, 209
 - Гильберта, 209
 - единичная, 194
 - квадратный корень, 211
 - кубический корень, 211
 - ковариации, 448
 - магического квадрата, 210
 - обратная, 221
 - Паскаля, 210
 - психологическая, 222
 - с единичными элементами, 194
 - с заданной диагональю, 200
 - с нулевыми элементами, 195
 - Теплица, 212
 - трехдиагональная, 230
 - Уилкинсона, 213
 - унитарная, 229
 - эрмитова, 214
- Матрицы, 194
 - LR-разложение, 222
 - LU-разложение неполное, 253
 - QR-разложение, 223
 - возведение в степень, 385
 - масштабирование, 225
 - норма, 215
 - обращение, 221
 - определитель, 217
 - особенности задания, 66
 - пустые [], 206
 - разложение Холецкого, 220
 - разреженные, 234
 - разреженные в аэродинамике, 247
 - ранг, 217
 - сингулярные числа, 225
 - след trace, 220
 - смежные, 245
 - собственные векторы, 226
 - собственные значения, 225
 - собственные значения обобщенные, 228
 - сопровождающие, 207
 - специальные, 207
 - тестовые, 207
 - удаление столбцов и строк, 70
 - форма Шура действительная, 230
 - форма Шура комплексная, 230
 - формы Шура и Хессенберга, 227
 - числа обусловленности, 216
 - число обусловленности, 225
- Меню
 - Edit, 99
 - Edit окна графики, 115
 - File, 97
 - File окна графики, 85
 - Tools окна графики, 117
 - контекстное правой клавиши мыши, 93
 - правой клавиши мыши, 88
- Метод
 - Гаусса решения СЛУ, 385
 - двунаправленный сопряженных градиентов, 390
 - интегрирования Лобатто, 409
 - интегрирования Симпсона, 409
 - исключения Гаусса, 217
 - итерационный сопряженных градиентов, 392
 - квадратичный сопряженных градиентов, 393
 - квазиминимизации невязки, 394
 - минимизации обобщенной невязки, 393
 - устойчивый двунаправленный, 392
- Методы, 535

- Минимизации функций, 398
- Массивы многомерные
удаление размерности, 258
- МНК, метод наименьших
квадратов, 465
- Модули программные, 532
- Н**
- Неполная гамма-функция, 189
- Норма
вектора, 215, 217
матрицы, 215
- Нумерация строк программы, 101
- О**
- Обработка табличных данных
в графическом окне, 485
- Обработка данных в графическом
окне, 485
- Обработка данных, 442
- Обращение к функции
пользователя, 64
- Объекты
дескрипторной графики, 347
типа Patch, 372
- ОДУ, обыкновенные
дифференциальные уравнения, 416
- Окно
графики, 115
графическое, 118
графическое и управление им, 347
основное, 46
свойств графики, 124
средств пакета Wavelet Toolbox, 696
- Окраска поверхностей, 328
- ООП
агрегирование, 562
инкапсуляция, 561
конструкторы классов, 562
контроль отношения объекта
к классу isa, 564
методы объектов функции methods,
methodsviw, what, 564
наследование, 562
объектов классы, 562
полиформизм, 562
создание классов, функция
class, 563
- Операнды, данные для операторов, 60
- Оператор, 257
матричного деления, 221
определение, 60
создания паузы pause, 561
транспонирования, 158
- Операторы
арифметические, 152
арифметические +, -, *, ./ и ^, 60
блока вывода ошибок
try...catch...end, 559
конкатенации, 159
логические, 155
матричные, 384
множественного выбора swith-case-
otherwisw-end, 558
особенности при комплексных
операндах, 154
отношения, 153
специальные, 158
условные if-elseif-else-end, 553
цикла for-end, 554
цикла while...end, 556
- Операции
арифметические с векторами и
матрицами, 68
матричные, 194
над графическими объектами, 350
с двоичными файлами, 507
с форматированными файлами, 510
со строками, 494, 496
- Определение
команд и операций, 97
параметр, 97

- Опции решателей ОДУ, 421
- Особенности
- М-файлов функций, 550
 - простых вычислений, 50
- Оцифровка узлов графа, 246
- Ошибка переполнения памяти, 550
- Ошибок диагностика, 64
- И**
- Панель инструментов
- редактора-отладчика m-файлов, 103
- Параметры
- PNG-файлов, 524
 - спецификаторов формата, 512
 - функции входные, 102
- Параметры
- интегрирования при решении ОДУ, 420
 - решателей ОДУ, 420
- Переменные, 58
- индексированные, 43
 - локальные, 102
 - присваивание значений, 59
 - системные, 57
- Переход в командный режим отладки программ, 568
- Подпапки m-файлов, 46
- Подсказка, клавиша Tab, 64
- Подфункции в M-файлах, 543
- Поиск максимального и минимального элементов в массиве, 442
- Полином степенной многочлен, 411
- Полином ортогональный
- Лежандра, 190
- Полиномов
- вычисление, 412
 - производных, 415
 - вычисление корней, 414
 - матричных собственные значения, 415
 - разложение на простые дроби, 416
 - умножение и деление, 412
- Поля информационной структуры, 518
- Построение
- легенды, 317
 - надписей титульной и по осям, 312
 - надписи в заданном месте
 - графика, 313
 - надписи с указанием места мышью, 315
- Построение легенды вне графика, 317
- Представление
- двумерного массива, 255
 - действия оператора «, 257
 - трехмерного массива, 255
- Преобразование типов данных, 273
- Фурье быстрое прямое, 455
 - Фурье прямое многомерное, 457
- Преобразования
- Фурье, 454
 - Фурье быстрые обратные, 459
- Применение массивов структур, 268
- разреженных матриц, 234
- Пример
- вейвлет-декомпозиции и реставрации сигнала, 696
 - вейвлет-экстраполяции изображения, 698
 - вертикального объединения строк, 497
 - визуализации вложенных массивов, 276
 - визуализации массива ячеек, 271
 - вложения массивов, 275
 - выдачи времени, 165
 - выдачи календаря, 165
 - выравнивания строк, 499
 - вырезания из строки, 500

- вычисления градиента, 407
- вычисления двойного интеграла, 411
- вычисления площади многоугольника, 451
- вычисления производной полинома, 415
- вычисления собственных значений матричного полинома, 415
- гамма-коррекции изображения, 692
- генерации случайных чисел, 197
- графика контурного трехмерного, 311
- графика лестничного, 286
- двумерной интерполяции, 472
- деления полиномов, 412
- диадических вейвлет-преобразований, 665
- доступа к ячейкам многомерного массива, 275
- задания и вывода массива ячеек, 269
- замены части строки, 499
- заполнения пустого массива ячеек, 270
- изменения изображения в заданной области, 694
- индексации в массиве ячеек, 269
- интегрирования методом трапеций, 409
- интегрирования с помощью функции quad, 410
- интегрирования функции методом трапеций, 408
- интерполяции периодической функции, 467
- интерполяции функции косинуса, 470
- квадратичной регрессии в Curve Fitting, 704
- контурного графика, 293
- минимизации симплекс-методом, 400
- минимизации функции, 398
- минимизации функции Розенброка, 400
- многомерной интерполяции, 479
- нахождение корней по полиному, 414
- непрерывного вейвлет-анализа сигнала, 696
- объединения строк, 497
- открытия и закрытия файла, 508
- оценки времени БПФ, 168
- оценки времени работы процессора, 166
- поиска максимального элемента в массиве, 442
- поиска минимального элемента в массиве, 443
- поиска среднего в массиве, 446
- получения информации о файле, 519
- построения спектрограммы звука, 528
- построения выпуклой оболочки, 450
- построения диаграммы Вороного, 453
- построения изображения струи, 377
- построения куба, 482
- построения окрашенного многоугольника, 372
- построения орбит спутников на карте земли, 700
- построения осциллографа, 701
- построения сферы с эффектами освещения, 483
- преобразований дат, 166
- преобразования строки в вычисляемое выражение, 501, 502
- присваивания для массива ячеек, 273
- проверки структур, 267
- просмотра 2-страничного массива, 274

- профилирования М-файла, 572
 - работы с файлами изображений, 525
 - работы со звуком, 527
 - расчета попадания точек
 - в полигон, 452
 - реализации фильтрации на основе БПФ, 461
 - решения ОДУ Ван-дер-Поля, 423
 - решения системы ОДУ, 424
 - свертки полиномов, 412
 - создания пустого массива ячеек, 270
 - создания трехмерного массива ячеек, 274
 - создания отчета, 573
 - спектрального анализа в Signal Processing, 691
 - спектрального анализа зашумленного сигнала, 455
 - строкового преобразования чисел, 502
 - фильтрации изображения, 692
 - численного дифференцирования, 405
 - Примеры
 - арифметических операций, 152
 - демонстрационные на разреженные матрицы, 250
 - демонстрационные пакета Signal Processing, 689
 - дескрипторной графики, 353
 - нахождения полинома по его корням, 412
 - операций со строками, 496
 - преобразования кодов в символы, 495
 - применения логических операторов, 155
 - применения операторов отношения, 154
 - работы с бинарными файлами, 509
 - сравнения строк, 498
 - Примеры демонстрационные, список, 78
 - Приоритет выполнения операций, 153
 - Программ
 - задание точек контроля, 569
 - листинг, 569
 - отладка, 567
 - Программирование
 - визуально-ориентированное, 536
 - некоторые ограничения, 537
 - объектно-ориентированное, 536
 - объектно-ориентированное (ООП), инкапсуляция, 561
 - структурное, 535
 - Программирования, виды, 535
 - Программы, пошаговое выполнение, 570
 - Просмотр
 - рабочей области, 95
 - содержимого матрицы, 95
 - Профилирование М-файлов, 571
 - Процессоры Intel, Pentium и AMD Athlon, 44
- Р**
- Рабочая область, 70
 - Разбиение графического окна, 324
 - Размерность и размер векторов и матриц, 42
 - Регрессия, 465
 - полиномиальная, 465
 - Режим командный, 46
 - Решатели ОДУ, 417
 - Решение
 - нелинейного уравнения с визуализацией, 395
 - систем ОДУ численное, 419
 - СЛУ, 221
 - СЛУ с разреженными матрицами, 388
 - СЛУ элементарное, 385

С

Свертка

- векторов, 412
- двумерных массивов, 461
- обратная `conv`, 460
- прямая `conv`, 460

Свойства

- М-файла функции, 540
- графических объектов, 351
- файла-сценария, 538
- файла-функции, 541

Сессия

- сеанс работы, 47
- форма представления, 52

Символы

- специальные, 511
- формата, 512

Симплекс-метод Нелдера-Мида, 398

Системные переменные и константы, 159

СКМ, MATLAB матричная лаборатория, 43

СЛУ

- недоопределенные и переопределенные, 385
- системы линейных уравнений, 384
- точное решение, 388

Создание

- Р-кодов, 551
- итогового отчета, 572

Соответствие операторов и функций, 153

Сортировка элементов массивов, 443

Состав системы

- MATLAB+Simulink, 660

Специальные символы, 156

Спецификаторы, 511

Справка

- дополнительные команды, 77

- о версии Java, 78
- о каталогах файлов, 78
- о компьютере, 77
- о текущей версии MATLAB, 78
- о файлах, 77
- о фирме MathWorks, 77
- по ключевому слову, 77
- по конкретному объекту, 75
- по определенной группе объектов, 76
- по функциям MATLAB, 137
- справочная система MATLAB, 74

Средства

- поддержки звука, 526
- языка программирования MATLAB, 533

Строчный редактор, 49

Структура

- М-файла функции с одним выходом, 540
- М-файла функции с рядом выходов, 540
- файла-сценария, 538

Структуры, 264

- возврат имен полей, 267
- возврат содержимого полей, 267
- индексация, 266
- присваивание полям значений, 268
- проверка имен, 267
- проверка имен полей, 266
- создание, 266
- создание схем, 265
- удаление полей, 268
- управляющие, 552

Т

Таблица кодов, 494

Тип линий графиков, 279

Триангуляция Делоне, 449

У

- Управление
 - подсветкой и обзором фигур, 307
 - свойствами осей графиков, 319
 - цветовыми палитрами и эффектами, 335
- Условия с пустыми матрицами, 561
- Установка сетки на графике, 322
- Установка масштаба осей
 - 2D-графика, 321

Ф

- Файл
 - сценарий, 102
 - сценарий (Script-файл), 538
 - функция, 102
- Файловая система, 45
- Файлы, 504
 - бинарные, 45
 - допустимые символы, 511
 - наборов инструментов, пакетов расширения Toolbox, 46
 - открытие и закрытие, 505
 - специализированные, 517
 - сценарии и функции, 102
 - текстового формата, 45
 - указатель позиции, 514
 - форматы, 521, 522
- Форма Коши для ОДУ, 417
- Формат представления даты, 166
- Форматирование
 - 2D-графиков, 118
 - 3D-графиков дополнительное, 128
 - графиков нескольких функций, 123
 - графиков программное, 127
 - линий графика, 119
 - маркеров опорных точек, 121
 - осей графиков, 124
- Форматирования, панель Samega, 128
- Форматы файлов, 521

Функции

- true и false MATLAB 6.5, 198
- арифметические, 152
- арифметические
 - и алгебраические, 168
- Бесселя, 183
- Бесселя модифицированные, 185
- времени и даты, 165
- выделения треугольных частей матриц, 205
- вычисления произведений элементов массивов, 201
- вычисления строковых выражений, 503
- гиперболические, 177
- двойственность с операторами, 536
- интегрирования квадратурными методами, 409
- комплексного аргумента, 56, 181
- логические, 155
- матричные, 213
- новые поддержки звука
 - в MATLAB 6.1/6.5, 526
- обработки множеств, 162
- обработки строк, 494, 495
- обратные гиперболические, 177
- обратные тригонометрические, 173
- округления, 180
- отношения, 153
- перестановки элементов матриц, 201
- поворота матрицы, 205
- поддержки звука, 526
- поразрядной обработки, 161
- построения элементов пользовательского интерфейса, 362
- представления аргументов списком, 549
- преобразования разреженных матриц, 237
- преобразования систем счисления, 502

работы с ненулевыми элементами разреженных матриц, 240
решения СЛУ, 386
синтаксис записи, 536
специальные математические, 182
статистики элементов массива, 445
суммирования элементов массивов, 203
тригонометрические, 173
формирования матриц, 204
численного интегрирования, 408
элементарные, 168
эллиптические интегралы 1-го и 2-го рода, 188
Якоби эллиптические, 187
Функции Лежандра полунормализованные по Шмидту, 191
Функций улучшение в MATLAB 6.5, 171
Функция inline задания функции пользователя, 63
psi MATLAB 6.5, 191
бета и ее варианты, 186
дополнительная ошибки, 188
интегральная показательная, 189
Лежандра, 190
минимизации функции нескольких переменных, 398
определение, 61

перегруппировки при спектральном анализе, 458
Эйри, 182
Функция ошибок, 188
ФЧХ. См. Фазочастотная характеристика

Ц

Цветовые выделения в программах, 102

Ч

Частные каталоги М-файлов, 544
Числа как объект системы MATLAB, 55
комплексные, 56
основные типы, 55
случайные с нормальным распределением, 198
Численные методы, 384

Я

Язык входной MATLAB, 532
интерпретирующий, 533
проблемно-ориентированный MATLAB, 532
программирования MATLAB, 532
Язык программирования, 43

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслать 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

MATLAB

Полный самоучитель

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru

Корректор *Синяева Г. И.*

Верстка *Чаннова А. А.*

Дизайн обложки *Мовчан А. Г.*

Подписано в печать 29.12.2011. Формат 70×100 ¹/₁₆.

Гарнитура «Петербург». Печать офсетная.

Усл. печ. л. 72. Тираж 100 экз.

№

Издательство ДМК Пресс

Web-сайт издательства: www.dmk-press.ru

Internet-магазин: www.alians-kniga.ru

Дьяконов В. П.

MATLAB

Полный самоучитель

Самоучитель по массовой матричной системе MATLAB, занимающей лидирующее место в области численных научно-технических вычислений, расчетов и моделирования. Основное внимание уделено описанию основ применения и языка программирования базовой системы MATLAB, реализации численных методов вычислений и визуально-ориентированному проектированию графического интерфейса пользователя (GUI). Описаны особенности интерфейса MATLAB, операторы, функции и средства программирования. Приведены сотни примеров применения MATLAB в учебных, научно-технических и математических вычислениях и расчетах. Для студентов, преподавателей и аспирантов университетов и вузов различного профиля, инженеров и научных работников.



Internet-магазин: www.aliants-kniga.ru

Книга – почтой:

Россия, 123242, Москва, а/я 20

е-mail: book@aliants-kniga.ru

Оптовая продажа: «Альянс-книга»

Тел./факс: (095) 258-9195

е-mail: books@aliants-kniga.ru

978-5-94074-652-2



9 785940 746522