www.bhv.ru
www.bhv.kiev.ua

Игорь Ануфриев

MatLab 5.3/6.x

САМОУЧИТЕЛЬ



- Основы программирования
- Использование графических возможностей пакета
- Решение специальных задач
- Интеграция с MS Word и MS Excel

Дискета содержит М-файлы примеров

Эффективное средство для решения математических задач

Игорь Ануфриев

САМОУЧИТЕЛЬ
MatLab 5.3/6.x

Санкт-Петербург
«БХВ-Петербург»
2002

УДК 681.3.06

Ануфриев И. Е.

Самоучитель MatLab 5.3/6.x. — СПб.: БХВ-Петербург, 2002. — 736 с.: ил.

ISBN 5-94157-107-0

Книга посвящена применению пакета MatLab для решения различных математических задач. Изложены основы программирования в MatLab. Подробно рассмотрены реализация численных методов, создание приложений с графическим интерфейсом пользователя, возможности специализированных модулей (ToolBox), связь MatLab с другими средами программирования, интегрирование с Word и Excel. Книга содержит большое количество последовательно усложняющихся примеров и задач.

Для исследователей и разработчиков

УДК 681.3.06

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.03.02.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 59,34.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-107-0

© Ануфриев И. Е., 2002
© Оформление, издательство "БХВ-Петербург", 2002

Содержание

Введение **1**

Часть I. Основы работы в MatLab **15**

Глава 1. Простейшие вычисления **17**

Рабочая среда MatLab 17
Арифметические вычисления 19
Простейшие вычисления 19
Форматы вывода результата вычислений 20
Использование элементарных функций 23
Встроенные элементарные функции 27
Тригонометрические, гиперболические и обратные к ним функции 28
Экспоненциальная функция, логарифмы, степенные функции 28
Функции для работы с комплексными числами 29
Округление и остаток от деления 29
Использование переменных 30
Сохранение рабочей среды 32
Просмотр переменных 34

Глава 2. Работа с массивами **36**

Основные определения и соглашения 36
Вектор-столбцы и вектор-строки 37
Ввод, сложение и вычитание векторов 37
Обращение к элементам вектора 40
Применение функций обработки данных к векторам 42
Поэлементные операции с векторами 44
Построение таблицы значений функции 47
Построение графиков функции одной переменной 50
Умножение векторов 54
Скалярное произведение 55
Векторное произведение 55
Внешнее произведение 56
Двумерные массивы, матрицы 57
Ввод матриц, простейшие операции 58

Различные способы ввода ... 58
Обращение к элементам матриц ... 59
Сложение, вычитание, умножение, транспонирование и возведение в степень ... 60
Перемножение матрицы и вектора ... 62
Решение систем линейных уравнений ... 62
Считывание и запись данных ... 63
Блочные матрицы ... 64
Конструирование блочных матриц ... 64
Выделение блоков ... 66
Удаление строк и столбцов ... 66
Заполнение матриц при помощи индексации ... 67
Создание матриц специального вида ... 68
Визуализация матриц ... 72
Поэлементные операции и встроенные функции ... 73
Поэлементные операции с матрицами ... 74
Вычисление математических функций от элементов матриц ... 75
Применение функций обработки данных к матрицам ... 76
Графики функций двух переменных ... 80

Глава 3. Высокоуровневая графика ... 83

Диаграммы и гистограммы ... 83
Представление векторных данных ... 83
Диаграммы векторных данных ... 83
Гистограммы векторных данных ... 89
Представление матричных данных ... 92
Графики функций ... 95
Графики функций одной переменной ... 96
Графики в линейном масштабе ... 96
Графики в логарифмических масштабах ... 98
Изменение свойств линий ... 99
Оформление графиков ... 101
Графики параметрических и кусочно-заданных функций ... 102
Графики функций двух переменных ... 104
Трехмерные графики функций ... 104
Контурные графики ... 109
Оформление графика ... 112
Поворот графика, изменение точки обзора ... 117
Построение параметрически заданных поверхностей и линий ... 119
Построение освещенной поверхности ... 122
Анимированные графики ... 123
Работа с несколькими графиками ... 125
Вывод графиков в отдельные окна ... 125
Вывод нескольких графиков на одни оси ... 127
Несколько графиков в одном графическом окне ... 129

Глава 4. Редактирование графиков 132

Редактирование графиков в MatLab 5.3 132
Изменение свойств осей, подписи, заголовок 132
Свойства линий 134
Дополнительные элементы оформления 135
Сохранение, экспорт и печать графиков 136
Редактирование графиков в MatLab 5.3 при помощи редактора свойств 138
Структура объектов в MatLab 138
Установка свойств объектов 141
Заголовок, подписи осей 141
Свойства линий и поверхностей 142
Свойства осей 144
Управление камерой 146
Свойства графического окна 148
Редактирование графиков в MatLab 6.*x* 148
Запуск редактора свойств 149
Свойства осей, подписи, заголовок 151
Пределы, масштаб, разметка, сетка 151
Подписи и заголовок 152
Свойства линий и поверхностей 153
Свойства линий 154
Свойства поверхностей 154
Дополнительные элементы оформления 156
Управление освещением графика 157
Изменение точки обзора 160
Сохранение, экспорт и печать 162

Глава 5. M-файлы 165

Работа в редакторе M-файлов 165
Типы M-файлов 167
Файл-программы 168
Установка путей 169
Установка путей в версии 5.3 169
Установка путей в версии 6.*x* 170
Команды для установки путей 172
Файл-функции 172
Файл-функции с одним входным аргументом 173
Файл-функции с несколькими входными аргументами 175
Файл-функции с несколькими выходными аргументами 176

Часть II. Численные методы и программирование 177

Глава 6. Вычисления в MatLab 179

Исследование функций 179

Решение уравнений ... 179
Решение произвольных уравнений ... 179
Вычисление всех корней полинома ... 183
Минимизация функций ... 184
Минимизация функции одной переменной ... 184
Минимизация функции нескольких переменных ... 185
Задание дополнительных параметров ... 187
Интегрирование функций ... 189
Вычисление определенных интегралов ... 189
Вычисление двойных интегралов ... 191
Вычисление некоторых интегралов ... 192
Интегралы, зависящие от параметра ... 192
Интегралы с переменным верхним пределом ... 193
Полиномы и интерполяция ... 194
Операции с полиномами ... 194
Умножение, деление, сложение и вычитание ... 194
Вычисление производных ... 196
Интерполирование ... 196
Приближение по методу наименьших квадратов ... 197
Интерполяция сплайнами ... 198
Интерполяция двумерных и многомерных данных ... 200
Задачи линейной алгебры ... 202
Системы уравнений, определители, обращение матриц ... 202
Системы с плохо обусловленными матрицами ... 202
Переопределенные и недоопределенные системы ... 204
Обращение матриц ... 206
Собственные числа и векторы матрицы, функции матриц ... 207
Решение дифференциальных уравнений ... 209
Схема решения задач с начальными условиями ... 209
Решение уравнений Лотка—Вольтерра ... 212
Управление процессом решения ... 214
Солверы для решения задач с начальными условиями ... 214
Задание точности вычислений ... 215
Решение граничных задач ... 219
Схема решения ... 219
Простой пример граничной задачи ... 220

Глава 7. Основы программирования в MatLab ... 223

Операторы цикла ... 223
Цикл *for* ... 223
Цикл *while*, суммирование рядов ... 231
Операторы ветвления ... 234
Условный оператор *if* ... 234
Проверка входных аргументов ... 234
Организация ветвления ... 238
Оператор *switch* ... 244

Прерывания цикла, исключительные ситуации 247
Прерывание цикла, оператор *break* 247
Обработка исключительных ситуаций, оператор *try...catch* 248
Логические выражения с массивами и числами 249
Операции отношения 249
Логические операции 251
Приоритет операций 252
Логическое индексирование 253

Глава 8. Тонкости программирования 256

Работа со строками 256
Простейшие операции со строками 256
Ввод и сцепление строк 256
Сервисные функции для работы со строками 257
Массивы строк 260
Текстовые файлы 262
Открытие файла, считывание данных и закрытие файла 262
Запись в текстовый файл 265
Запись строк 265
Форматный вывод 268
Массивы структур и массивы ячеек 271
Массивы структур 271
Создание файл-функций для работы массивами структур 275
Запись данных массивов структур в текстовый файл 276
Считывание информации из текстового файла 278
Операции с массивами структур 282
Массивы ячеек 283
Приложения с интерфейсом из командной строки 288
Простой пример, программа-калькулятор 288
Формирование и исполнение команд, функция *eval* 292
Файл-функции с переменным числом аргументов 297
Функции от функций 305
Подфункции и приватные функции 309
Подфункции 309
Приватные функции 312

Глава 9. Дескрипторная графика 313

Графические объекты 313
Свойства графических объектов 314
Функции *set* и *get*, текущие объекты 314
Свойства осей 314
Свойства линий и поверхностей 317
Указатели на объекты 319
Изменение свойств линий и осей 320
Добавление линий графиков 322

Удаление и очистка объектов ... 323
Получение информации о свойствах ... 324
Использование указателей, примеры ... 324
Задание свойств в аргументах графических функций ... 327
Расположение графических окон и осей ... 328
Управление положением графических окон ... 328
Управление положением осей ... 331
Пример работы с графикой. Исследование функций ... 334
Размещение текстовой информации ... 336
Текстовые объекты ... 336
Размещение текста в графическом окне ... 341

Часть III. Работа в среде GUIDE ... 343

Глава 10. Принципы создания приложений с GUI ... 345

Принципы создания приложений в версии 5.3 ... 345
Среда разработки приложений GUIDE ... 345
Программирование событий в версии 5.3 ... 348
Принципы создания приложений в версии 6.*x* ... 353
Среда GUIDE ... 353
Программирование событий в версии 6.*x* ... 355

Глава 11. Конструирование интерфейса в версии 5.3 ... 360

Установка свойств объектов, функция *findobj* ... 360
Работа над приложением ... 364
Программирование событий в файл-функции ... 366
Программирование элементов интерфейса ... 368
Флаги ... 368
Переключатели ... 372
Списки ... 379
Полосы скроллинга ... 383
Область ввода текста ... 384

Глава 12. Конструирование интерфейса в версии 6.*x* ... 386

Управление свойствами объектов ... 386
Установка свойств при редактировании ... 386
Программное изменение свойств ... 387
Работа над приложением ... 389
Запуск приложения ... 389
Оформление интерфейса ... 390
Программирование элементов интерфейса ... 391
Флаги и рамки ... 391
Переключатели ... 395

Списки ... 402
Полосы скроллинга ... 406
Область ввода текста ... 408

Глава 13. Диалоговые окна и меню приложения ... 410

Виды диалоговых окон ... 410
Окно подтверждения ... 410
Окна открытия файла и записи в файл ... 411
Окно с сообщением об ошибке ... 413
Меню графического окна ... 414
Создание меню в редакторе в версии 5.3 ... 414
Создание меню в редакторе в версии 6.*x* ... 417
Программирование пунктов меню в версии 5.3 ... 419
Программирование пунктов меню в версии 6.*x* ... 420
Флаги состояния и разделительные линии ... 424
Пункты меню с флагами состояния ... 424
Разделительные линии ... 427
Упорядочение меню в версии 5.3 ... 427
Контекстное меню объектов ... 430
Создание меню в версии 5.3 ... 430
Создание меню в версии 6.*x* ... 431
Связывание контекстного меню с объектом ... 431
Программирование контекстного меню в версии 5.3 ... 434
Программирование контекстного меню в версии 6.*x* ... 435

Глава 14. Программирование событий ... 437

Событие осей *ButtonDownFcn* ... 437
Размещение элементов интерфейса ... 437
Программирование приложения ... 438
События и свойства объектов в MatLab ... 443
Иерархия объектов ... 444
Объект *Root* ... 445
Объект *Figure* ... 445

Часть IV. Использование ToolBox ... 449

Глава 15. Решение задач математической физики ... 451

Простой пример ... 451
Постановка задачи ... 451
Среда *pdetool*, конструирование области ... 452
Определение уравнения и граничных условий ... 455
Решение и визуализация результата ... 457

Описание возможностей ToolBox PDE....460
Эллиптическое уравнение....460
Переменные коэффициенты и правая часть уравнения....461
Параболическое и гиперболическое уравнения....463
Пример нестационарной задачи....463
Задача на собственные значения....466
Системы дифференциальных уравнений....466
Параметры триангуляции и управление процессом решения....468
Конструирование геометрии области....469
Геометрические примитивы....469
Задание структуры области....471
Композитные материалы....472
Использование сетки....474
Использование функций ToolBox PDE....474
Задание геометрии области....474
Триангуляция....482
Граничные условия и коэффициенты уравнения....484
Солверы....486
Визуализация результата....488
Решение модельной задачи....489
Функции ToolBox PDE....492
Создание геометрических примитивов....492
Геометрия области и триангуляция....492

Глава 16. Разреженные матрицы....495

Работа с разреженными матрицами....495
Схема хранения....495
Создание разреженных матриц....497
Операции с разреженными матрицами....502
Задачи линейной алгебры....503
Факторизация матриц....504
Профайлер....507
Решение систем уравнений и исследование спектра....510

Глава 17. Оптимизация....512

ToolBox Optimization....512
Линейное и нелинейное программирование....512
Линейное программирование....512
Квадратичное программирование....514
Нелинейное программирование....515
Нелинейные задачи....517
Решение нелинейных уравнений....518
Метод наименьших квадратов....519
Подбор параметров....520
Параметры оптимизации....523

Примеры....525
Решение большой системы нелинейных уравнений....525
Пример приложения с GUI....529

Глава 18. Символические вычисления....535

Символические переменные и функции....535
Определение переменных и функций и работа с ними....535
Матрицы и векторы....537
Преобразование в числовые значения....539
Графическое представление функций....540
Упрощение и преобразование выражений....542
Решение задач....545
Задачи линейной алгебры....545
Суммирование и разложение в ряд....549
Пределы, дифференцирование и интегрирование....550
Решение уравнений и систем....556
Решение дифференциальных уравнений и систем....559

Часть V. Дополнительные возможности MatLab....563

Глава 19. Связь MatLab и MS Office....565

M-книги....565
Настройка MatLab и создание M-книги....565
Группировка ячеек....567
Управление M-книгой....569
Excel Link....571
Конфигурирование Excel....571
Обмен данными между MatLab и Excel....572
Обращение к основным функциям Excel Link....574
Функции Excel Link....576

Глава 20. Редактирование приложений с GUI версии 5.3 в версии 6.*x*....578

Пример приложения для MatLab 5.3....578
Модернизация приложения для версии 6.*x*....580
Сохранение приложения в формате FIG....580
Переход к форматам FIG и M....582

Глава 21. Повышение производительности приложений MatLab....586

Ускорение работы M-файлов....586
Поэлементные операции....586
Выделение памяти под массивы....589

Компилирование M-файлов 590
Конфигурирование MatLab Compiler 591
Компилирование файл-функций 592
Компилирование нескольких файл-функций 594
Работа с файл-программами 595
Генерация MEX-файлов 596
Простой пример, сложение двух чисел 596
Работа с комплексными переменными 600
Обмен массивами данных 602

Часть VI. Приложения 609

Приложение 1. Основные команды и функции MatLab и ToolBox 611

Управление средой, файлами и переменными 611
Получение справочной информации 611
Управление средой MatLab 612
Управление переменными 615
Манипулирование файлами и каталогами 616
Операторы и специальные символы 618
Логические операции и операторы 619
Побитовые операции 620
Логические функции 624
Программирование 627
Конструкции языка 627
Сервисные функции и переменные 628
Интерактивный ввод 631
Объектно-ориентированное программирование и преобразование типов 632
Функции даты и времени 632
Двоичные и текстовые файлы 633
Функции для работы с массивами ячеек 640
Функции для работы со структурами 644
Звуковые и графические файлы 647
Чтение, запись и преобразование звуковых данных 647
Графические файлы 649
Работа со строками 651
Обработка строк 651
Преобразования строка-число 655
Преобразование системы счисления 658
Работа с матрицами и массивами 659
Создание матриц и массивов 659
Операции с массивами 661
Математические функции 662
Специальные функции 662
Преобразование координат 667

Функции для решения задач линейной алгебры 667
Матричный анализ 668
Решение спектральных задач 670
Решение линейных уравнений, разложения и обращение матриц 671
Вычисление функций от матриц 674
Решение различных математических задач 674
Поиск корней 674
Интерполяция 676
Минимизация и оптимизация 677
Дифференцирование и конечные разности 678
Интегрирование 679
Решение дифференциальных уравнений и систем 680
Графика и визуализация данных 680
Двумерные графики 680
Трехмерные графики 683
Визуализация функции на непрямоугольной области 696
Оформление графиков 699
Управление видом графика 702

Приложение 2. Описание дискеты 711

Литература 712

Введение

Пакет MatLab был создан компанией MathWorks более десяти лет назад. Работа сотен ученых и программистов направлена на постоянное расширение его возможностей и совершенствование заложенных алгоритмов. В настоящее время MatLab является мощным и универсальным средством решения задач, возникающих в различных областях человеческой деятельности. Спектр проблем, исследование которых может быть осуществлено при помощи MatLab, охватывает: матричный анализ, обработку сигналов и изображений, задачи математической физики, оптимизационные задачи, обработку и визуализацию данных, работу с картографическими изображениями, нейронные сети, нечеткую логику и многие другие. Специализированные средства собраны в пакеты, называемые ToolBox, и могут быть выборочно установлены вместе с MatLab по желанию пользователя. В состав многих ToolBox входят приложения с графическим интерфейсом пользователя, которые обеспечивают быстрый и наглядный доступ к основным функциям. Пакет Simulink, поставляемый вместе с MatLab, предназначен для интерактивного моделирования нелинейных динамических систем, состоящих из стандартных блоков.

Обширная и удобная справочная система MatLab способна удовлетворить потребности как начинающего, так и достаточно опытного пользователя. Полная гипертекстовая информационная система (на английском языке) содержит описание встроенных функций и достаточно большое число примеров их использования. Ссылки позволяют переходить к разделам, имеющим отношение к изучаемому вопросу, что облегчает самостоятельный поиск интересующей информации и увеличивает объем знаний начинающего пользователя. Доступ из командной строки к кратким сведениям о встроенных функциях обеспечивает возможность выбора нужного варианта обращения к функциям. Инженерам и научным работникам, проводящим самостоятельные исследования, оказываются полезными прилагаемые к пакету электронные книги в формате PDF. Данные книги не только дублируют справочную систему MatLab и каждого ToolBox, но и содержат теоретические сведения и математическую базу, необходимые для осознанного использования описываемых средств. Справочная система снабжена ссылками на книги и статьи, посвященные реализованным алгоритмам в MatLab и ToolBox, что позволяет исследователю и разработчику собственных алгоритмов вникнуть в суть дела.

Огромным преимуществом MatLab является открытость кода, что дает возможность опытным пользователям разбираться в запрограммированных алгоритмах и, при необходимости, изменять их. Впрочем, разнообразие набора

функций MatLab и ToolBox допускает решение большинства задач без каких-либо предварительных модификаций.

MatLab прекрасно интегрируется с Microsoft Word и Excel. Связь MatLab и Word обеспечивает возможность написания в редакторе Word интерактивных документов, так называемых M-книг, основанных на специальном шаблоне. Пользователь, работающий с M-книгой, может запускать блоки команд MatLab непосредственно из документа Word, причем результат выполнения команд отображается в M-книге. Данное средство прекрасно подходит для создания отчетов и учебных пособий, поскольку позволяет дополнить документ примерами и результатами расчетов.

Надстройка Excel Link, поставляемая вместе с MatLab, существенно расширяет возможности Excel, обеспечивая доступ пользователя к функциям MatLab и ToolBox. Подготовка данных осуществляется непосредственно в электронных таблицах, а обращение к функциям производится либо из ячеек рабочего листа, либо в модуле, написанном на Visual Basic.

Символические вычисления в MatLab основаны на библиотеке, являющейся ядром пакета Maple. Решение уравнений и систем, интегрирование и дифференцирование, вычисление пределов, разложение в ряд и суммирование рядов, поиск решения дифференциальных уравнений и систем, упрощение выражений — вот далеко не полный перечень возможностей MatLab для проведения аналитических выкладок и расчетов. Поддерживаются вычисления с произвольной точностью. Пользователи, имеющие опыт работы в Maple, могут напрямую обращаться ко всем функциям данного пакета (кроме графических) и вызывать процедуры, написанные на встроенном языке Maple.

Информация, хранящаяся в базах данных многих популярных форматов, может быть импортирована в MatLab, нужным образом обработана и исследована при помощи функций MatLab, а затем экспортирована в какую-либо другую базу данных. Для обмена данными используются команды языка запросов SQL. Поддерживается, в частности, связь с Microsoft Access, Microsoft SQL Server, Oracle. Имеется приложение с графическим интерфейсом, которое облегчает работу пользователей, не знакомых с языком запросов SQL.

MatLab обладает хорошо развитыми возможностями визуализации двумерных и трехмерных данных. Высокоуровневые графические функции призваны сократить усилия пользователя до минимума, обеспечивая, тем не менее, получение качественных результатов. Редактор графиков помогает оформить результат требуемым образом: добавить стрелки, поясняющие надписи, задать цвета и стиль линий и поверхностей, словом, получить изображение, пригодное для помещения в отчет или статью. Полный доступ к изменению свойств отображаемых графиков дают низкоуровневые функции, но их применение подразумевает понимание принципов компьютерной графики. Создание приложений MatLab с графическим выводом требует от програм-

миста умения управлять видом графиков при помощи низкоуровневых функций и средств дескрипторной графики.

В MatLab реализованы классические численные алгоритмы решения уравнений, задач линейной алгебры, нахождения значений определенных интегралов, интерполяции, решения дифференциальных уравнений и систем. Применение базовых вычислительных возможностей требует знания основных численных методов в рамках программы технических вузов. Решение специальных задач, разумеется, невозможно без соответствующей теоретической подготовки, впрочем, сведения, изложенные в справочной системе, оказываются неоценимым подспорьем для желающих самостоятельно разобраться в обширных возможностях пакета MatLab.

Простой встроенный язык программирования позволяет легко создавать собственные алгоритмы. Простота языка программирования компенсируется огромным множеством функций MatLab и ToolBox. Данное сочетание позволяет достаточно быстро разрабатывать эффективные программы, направленные на решение практически важных задач.

Визуальная среда GUIDE предназначена для написания приложений с графическим интерфейсом пользователя. Работа в среде GUIDE достаточно проста, но предполагает владение основами программирования и дескрипторной графики. Наличие определенного навыка работы в среде GUIDE предоставляет возможность создать визуальную среду для проведения собственных исследований, что значительно облегчает работу и существенно экономит время.

MatLab является интерпретатором, т. е. каждая строка программы преобразуется в код и затем выполняется. Разумеется, интерпретирование команд существенно увеличивает время работы алгоритма, содержащего циклически повторяемые действия. Для повышения производительности вычислений в составе пакета имеется дополнительный модуль MatLab Compiler, который обеспечивает компиляцию программ, написанных на языке MatLab.

Объектно-ориентированный подход, заложенный в основу MatLab, обеспечивает современную эффективную технологию программирования. С учетом специфики решаемой задачи разработчик приложений MatLab в дополнение к существующим классам имеет возможность создавать собственные каждый со своими методами.

Программный интерфейс приложения (API) реализует связь среды MatLab с программами, написанными на C или Fortran. Библиотека программного интерфейса позволяет вызывать имеющиеся модули C или Fortran из среды или программ MatLab, обращаться к функциям MatLab из программ на C или Fortran, осуществлять обмен данными между приложениями MatLab и другими программами, создавать приложения типа клиент-сервер.

Подводя итог вышесказанному, можно сделать вывод, что начинающий пользователь MatLab может в процессе работы совершенствовать свои зна-

ния как в области моделирования и численных методов, так и программирования и визуализации данных.

Обзор возможностей MatLab представляет демонстрационная программа, для ее запуска следует набрать в командной строке, обозначаемой символом ">>", команду `demo` и нажать клавишу <Enter>. Появляется окно программы **MATLAB Demo Window**, приведенное на рис. В1.

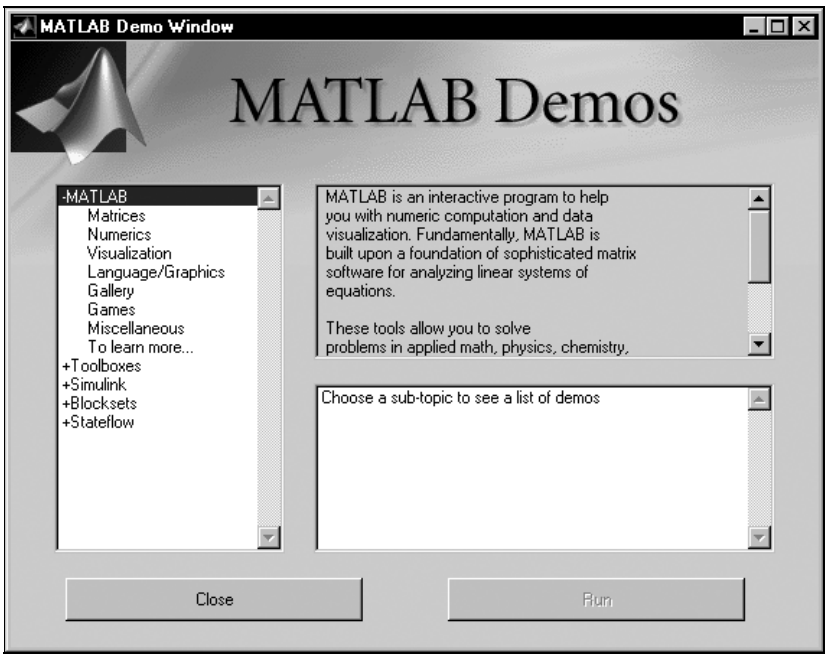


Рис. В1. Окно **MATLAB Demo Window**

Левое поле окна содержит разделы, охватываемые данной демонстрационной программой. При выборе раздела в правом нижнем окне отображаются доступные темы. Нажатие на кнопку **Run** приводит к появлению отдельного окна, предназначенного для показа возможностей MatLab, связанных с выбранной темой. К примеру, перейдите к **3-D Plots of complex functions** в разделе **Visualization** (в версии 5.3) или **Plots of complex functions** в разделе **Graphics** (в версии 6.*x*) и запустите демонстрацию, нажав на кнопку **Run 3-D Plots of...**. Интерфейс появляющегося окна **Examples of Complex Functions Plots** достаточно прост. Выберите один из предлагаемых вариантов функции комплексной переменной при помощи соответствующей кнопки в правой части окна, к примеру **Square root**. На графике отображается риманова поверхность исследуемой функции, а в области **MiniCommandWindow** — команды MatLab, приводящие к визуализации этой поверхности (см. рис. В2).

Строки, начинающиеся со знака процента, являются комментариями. Оказывается, что для изучения поведения квадратного корня из комплексного аргумента достаточно использовать всего одну функцию `cplxroot`, разумеется, с подходящими аргументами.

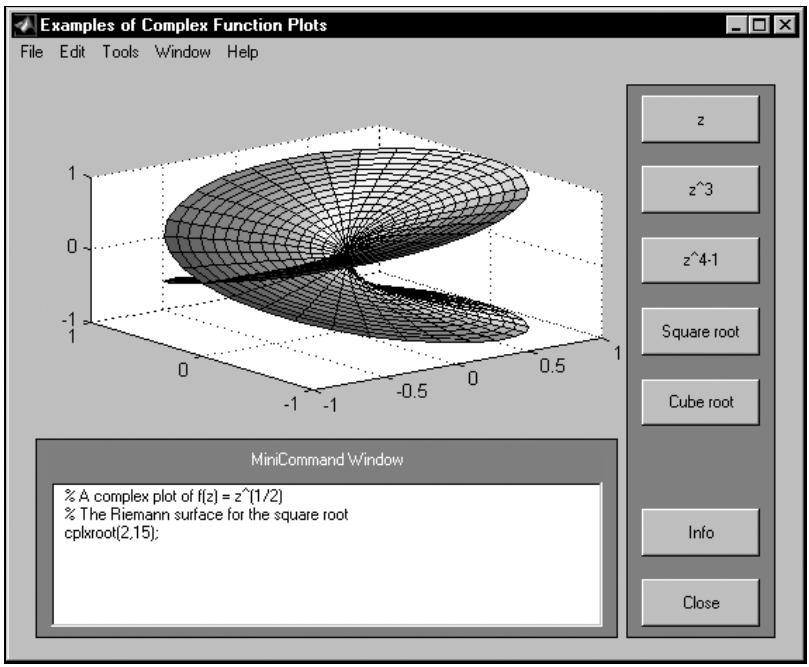


Рис. В2. Риманова поверхность квадратного корня

Двойной щелчок мышью по разделу **ToolBoxes** (со знаком плюс слева от названия) приводит к раскрытию списка подразделов. Темы каждого подраздела охватывают многие практически важные задачи, которые могут быть решены при помощи данного ToolBox. На самом деле возможности ToolBox MatLab значительно шире. Далее приведено краткое описание некоторых ToolBox. Команда `ver`, выполняемая из командной строки, выводит список всех установленных ToolBox с указанием их версий. Следует иметь в виду, что в MatLab 6.0 и 6.1 расширены возможности многих ToolBox по сравнению с версией 5.3.

Signal Processing ToolBox предназначен для исследования и обработки сигналов. Основными возможностями данного ToolBox являются:

- генерация, импорт и экспорт сигналов;
- разработка, анализ и применение фильтров с конечной и бесконечной импульсной характеристикой;

- спектральный анализ и статистическая обработка сигналов;
- моделирование линейных систем.

В состав Signal Processing ToolBox входит несколько приложений с графическим интерфейсом, предназначенных для облегчения доступа к функциям ToolBox. Данные приложения позволяют импортировать, визуализировать и исследовать сигналы, изучать спектр сигналов, интерактивно создавать фильтры с заданными характеристиками.

Image Processing ToolBox содержит большое число функций цифровой обработки и анализа изображений, в частности:

- импорт и экспорт графической информации;
- геометрические операции, например такие, как изменение размеров и поворот;
- получение статистической информации об изображении;
- анализ изображений, например нахождение границ интенсивности;
- обработка изображений: изменение контрастности, применение фильтров;
- разработка линейных фильтров;
- дискретные преобразования, в частности быстрое преобразование Фурье;
- операции над соседними элементами;
- работа с картой цветов;
- различные методы представления цветов;
- преобразование типов изображений.

В состав ToolBox Image Processing входит несколько демонстрационных приложений, охватывающих решение задач о нахождении границ изображений, фильтрации и разработки фильтров, сжатии изображения.

Функции и приложения **Statistics ToolBox** покрывают широкий спектр статистических задач и реализуют основные методы их решения. Доступно более двадцати классических распределений, для них имеются функции распределения вероятности (и обратной к ней), плотности вероятности, вычисления моментов распределений и генерации выборки из распределения. Основные классы статистических задач могут быть исследованы при помощи ToolBox Statistics, включая:

- исследование линейных моделей;
- параметрическое оценивание;
- проверка гипотез;
- планирование эксперимента;
- задачи кластерного анализа

и др.

ToolBox Statistics содержит набор функций для построения статистических графиков и приложения с графическим интерфейсом пользователя, предназначенные для изучения распределений и аппроксимации данных с использованием регрессионной модели.

ToolBox Optimization нацелен на решение основных линейных и нелинейных задач оптимизации, причем для задач с большим числом неизвестных предусмотрены весьма эффективные специальные методы. Класс задач, охватываемый данным ToolBox, включает:

- ❒ линейное и квадратичное программирование;
- ❒ минимизацию нелинейных функций при наличии нелинейных ограничений;
- ❒ подбор параметров;
- ❒ минимаксные задачи и задачи о достижении цели.

ToolBox Partial Differential Equations (PDE) создан для исследования задач математической физики, описываемых дифференциальными уравнениями и системами в частных производных. Решение задач значительно упрощается благодаря приложению с графическим интерфейсом, которое позволяет легко и наглядно осуществить все этапы решения задач методом конечных элементов — от задания области и граничных условий до визуализации результата. Приложение может быть легко настроено на определенный класс решаемых задач, из различных областей науки, например таких, как:

- ❒ теория упругости;
- ❒ электростатика и магнитостатика;
- ❒ теплопроводность;
- ❒ теория диффузии.

Нестационарные процессы отображаются при помощи анимированных графиков. В состав ToolBox PDE входят солверы для решения нелинейных задач и задач в адаптивном режиме. Возможности ToolBox PDE не ограничиваются вышеперечисленными типами задач, в частности, встроенные функции могут быть использованы для решения систем уравнений произвольной размерности.

В дополнение к вышеперечисленным средствам, следует подчеркнуть, что MatLab и соответствующие ToolBox могут с успехом применяться для интерактивного моделирования и анализа нелинейных систем, исследования устойчивости, разработки цифровых и аналоговых систем связи, передачи и хранения информации. Многие практические задачи, возникающие в области нечеткой логики и нейронных сетей, могут быть решены с использованием соответствующих ToolBox. Специальный пакет направлен на поддержку процесса сбора аналоговой и цифровой информации при помощи внешних устройств.

Разумеется, ограниченность объема книги не позволяет подробно описать все средства, которые MatLab и ToolBox предоставляют в распоряжение исследователя и инженера. Первая часть книги посвящена основам работы в MatLab. В *главе 1* описано использование переменных и вычисление арифметических выражений, изменение формата вывода чисел и основные встроенные математические функции. Название MatLab является сокращением от Matrix Laboratory, и первоначально пакет MatLab разрабатывался как средство доступа к библиотекам программ LINPACK и EISPACK, которые направлены на решение задач, связанных с матрицами и матричными вычислениями. Особенности представления данных в виде массивов, в частности матриц и векторов, дают пользователю широкие возможности по сравнению с большинством языков программирования. Большой набор специальных функций и средств унифицирует работу с массивами данных, делая ее очень эффективной. Отсутствие навыков оперирования с массивами приводит к многочисленным затруднениям, даже при решении самых простых задач. Поэтому *глава 2* книги подробно разъясняет принципы работы с матрицами и векторами, включая и основы визуализации векторных и матричных данных.

Пакет MatLab обладает чрезвычайно мощными возможностями графического представления одномерных и многомерных данных различных типов, включая и отображение функций. *Глава 3* нацелена на обучение читателя свободному владению средствами высокоуровневой графики для построения диаграмм и гистограмм, линий и поверхностей. Приведены команды, служащие для размещения и оформления графических результатов с целью получения хорошо читаемых графиков. Редактор графиков позволяет легко изменить многие свойства элементов, представленных на графике, по своему усмотрению, что существенно экономит время при подготовке различных документов, к примеру, отчетов. Отдельно описаны возможности редактирования графиков в версиях 5.3 и 6.*x*. Показано, как получить доступ ко всем свойствам графических объектов при помощи редактора свойств.

Работа из командной строки, разумеется, не очень удобна и подходит только для решения простых задач. Выход состоит в использовании M-файлов, т. е. программ и функций, содержащих нужную последовательность команд MatLab. Написание основных типов M-файлов (файл-программ и файл-функций) во встроенном редакторе разобрано в *главе 5*. M-файлы сохраняются на диске и запускаются на выполнение так же, как и другие команды и функции MatLab, что позволяет расширять набор стандартных средств MatLab, и создавать собственные пакеты программ для решения специальных задач. Более того, подавляющее большинство функций MatLab и ToolBox имеют открытый код, они запрограммированы в M-файлах, что дает опытному пользователю уникальную возможность разбираться в особенностях реализации алгоритмов и изменять их, приспосабливаясь к решению сложных специальных задач.

Вторая часть книги посвящена более сложным вопросам — применению численных методов и программированию собственных алгоритмов. Программирование в MatLab не требует специальных знаний, достаточно понимать принципы алгоритмизации. Пользователи, имеющие опыт программирования на Basic, C или Pascal, легко освоят встроенный язык программирования, основанный на минимальном наборе конструкций. Решение классических задач численными методами при помощи функций MatLab требует, в отличие от программирования, как минимум знаний, в объеме программы технических вузов. Поиск корней и минимизация функций, интегрирование и интерполирование, решение задач линейной алгебры, обыкновенных дифференциальных уравнений и систем разобрано в *главе 6*. Изложение сопровождается примерами с пояснениями.

Глава 7 содержит описание основных конструкций языка программирования MatLab, включая операторы ветвления и организации циклов. Описаны логические операции и логическое индексирование в применении к массивам. Работа со строками, текстовыми файлами и специальными типами данных — массивами ячеек и структур, продемонстрирована в *главе 8* на нескольких содержательных примерах. В этой же главе описан простейший способ организации взаимодействия программы MatLab с пользователем на основе интерфейса из командной строки. Несколько разделов *главы 8* информируют читателя о принципах написания файл-функций с переменным числом входных и выходных аргументов, поскольку подавляющее большинство функций MatLab допускают именно такое универсальное обращение к ним.

Разработка в MatLab программ, связанных с визуализацией данных, основана на управлении свойствами графических объектов прямо в ходе работы программы. Хорошо написанная программа не должна требовать от пользователя доработки графических результатов, к примеру, при помощи редактора графиков. MatLab является объектно-ориентированной системой, все графические объекты выстроены в некоторую иерархию и имеют определенные свойства. Полный доступ к свойствам всех графических объектов эффективно реализуется средствами дескрипторной графики. *Глава 9* раскрывает принципы управления свойствами графических объектов и содержит описание основных свойств. Простые примеры, приведенные в этой главе, демонстрируют основные возможности, имеющиеся в распоряжении разработчика графических программ в MatLab.

Третья часть книги предназначена для поэтапного обучения процессу создания приложений с графическим интерфейсом пользователя в среде GUIDE. Простота программирования и работы в среде GUIDE компенсируется потенциалом вычислительных и визуальных средств MatLab и ToolBox. Разработка приложений с графическим интерфейсом пользователя в среде GUIDE занимает немного времени, но существенно облегчает и ускоряет проведение исследований. *Глава 10* состоит из двух независимых разделов, каждый

из которых содержит необходимые сведения для начала работы в среде GUIDE в MatLab версии 5.3 и 6.*x*. На примере простого приложения показан процесс размещения элементов интерфейса в окне приложения и программирование событий. Следует иметь в виду, что обработка событий элементов управления требует понимания основ дескрипторной графики, которые изложены в *главе 9*. *Главы 11* и *12* являются независимыми, читателю следует выбрать из них соответствующую установленной версии MatLab. По мере чтения данных глав, читатель модернизирует приложение с графическим интерфейсом, работа над которым была начата в *главе 10*. Приложение пополняется флагами, переключателями, областями ввода и полосами скроллинга, причем читатель получает сведения об особенностях обработки событий каждого из элементов интерфейса и обеспечении согласованной работы всех элементов управления. О том, как снабдить собственное приложение диалоговыми окнами, сообщается в *главе 13*. Удобство работы с приложением во многом определяется хорошо продуманной структурой меню. В этой же главе описано конструирование, упорядочение и программирование пунктов меню. *Глава 14*, завершающая третью часть книги, содержит некоторые дополнительные сведения о программировании событий графических объектов и иерархии этих объектов в MatLab.

Четвертая часть книги целиком посвящена применению ToolBox для исследования некоторых специальных задач. *Глава 15* раскрывает перед читателем возможности ToolBox Partial Differential Equations (PDE), позволяющего решать задачи математической физики, описываемые уравнениями в частных производных, методом конечных элементов. Детально разобраны этапы решения задач в среде `pdetool`: описание геометрии области, задание уравнения и граничных условий, разбиение области сеткой, поиск приближенного решения и визуализация результата. Разобраны примеры стационарных и нестационарных задач. Следует иметь в виду, что среда `pdetool` лишь облегчает доступ к большому набору функций ToolBox PDE. Непосредственное использование данных функций в собственных программах позволяет проводить более сложные исследования, по сравнению с возможностями `pdetool`. В связи с этим, в *главе 15* приведено описание форматов представления данных, связанных с реализацией метода конечных элементов в ToolBox PDE, и разобраны примеры использования функций ToolBox.

Решение многих современных больших задач численными методами приводит к так называемым, разреженным матрицам, т. е. матрицам, содержащим достаточно много нулевых элементов. Работа с разреженными матрицами в MatLab с точки зрения пользователя происходит практически так же, как и с обычными. Разреженные матрицы принадлежат специальному классу, в котором обычные матричные операции переопределены в соответствии со спецификой разреженных матриц. *Глава 16* поясняет схему хранения, создание и операции с разреженными матрицами. Профайлер MatLab позволяет отчетливо выявить преимущества учета структуры матрицы при решении

задач линейной алгебры и матричного анализа, например таких, как факторизация матриц.

Решение различных типов линейных и нелинейных оптимизационных задач на основе функций ToolBox Optimization разобрано в *главе 17*. Эффективное использование оптимизационных алгоритмов для решения больших задач основывается на знаниях методов и, в частности, на умении работать с разреженными матрицами. Приведен пример решения большой системы нелинейных уравнений. Отдельный раздел *главы 18* посвящен написанию приложения с графическим интерфейсом пользователя для решения практически важной задачи о подборе параметров.

Исследователи, чья работа сопряжена с проведением большого количества аналитических выкладок и программированием модулей для соответствующих расчетов, несомненно, заинтересуются ToolBox Symbolic Math. Символические вычисления основаны на мощном ядре Maple, при этом пользователь имеет доступ ко всем ресурсам MatLab. *Глава 18* книги нацелена на обучение пользователя работе с символическими выражениями, включая упрощение, преобразование и вычисления с произвольной точностью. Отдельные разделы данной главы описывают технику решения задач в аналитическом виде, включая матричный анализ, суммирование, разложение в ряды, нахождение пределов функций и интегрирование, поиск решения дифференциальных уравнений и систем.

Последняя пятая часть книги охватывает несколько вопросов, которые могут быть полезны читателям с различными уровнями подготовки. *Глава 19* раскрывает возможности интегрирования MatLab с продуктами Microsoft Word и Excel. Описано создание в Word интерактивных документов (M-книг), позволяющих представлять постановку задачи, методы и результаты расчетов в наглядной форме с использованием всех возможностей мощного текстового редактора Word и среды MatLab. Второй раздел *главы 19* содержит информацию о конфигурировании Excel и организации совместной работы в MatLab и Excel. Возможен не только обмен данными между средой MatLab и таблицами Excel, но и вызов функций MatLab как из ячеек листа, так и из приложений на Visual Basic.

Пользователям, которые имеют приложения с графическим интерфейсом, созданные в версии 5.3, несомненно, окажется полезной информация о модернизации приложений в формат, принятый в новых версиях. *Глава 20* описывает процесс преобразования приложений из формата M/MAT, поддерживаемого в MatLab 5.3, в формат M/FIG, который используется в версиях, начиная с шестой. Переработка приложения достаточно проста, но требует понимания основ программирования дескрипторной графики в MatLab.

Работа пользователя MatLab не ограничена только возможностями среды и модулей ToolBox. В пакет MatLab входит компилятор MatLab Compiler и

библиотека функций MatLab API, реализующих программный интерфейс приложений. Последняя глава книги, *глава 21*, посвящена читателям, желающим повысить эффективность разрабатываемых приложений. Обычные программы, написанные на встроенном языке, интерпретируются MatLab, т. е. каждый оператор сначала преобразуется в машинный код, а затем выполняется. Эффективность многих приложений значительно увеличивается, если воспользоваться MatLab Compiler для генерации из M-файлов, так называемых, MEX-файлов. MEX-файлы вызываются из среды и других приложений MatLab точно так же, как и M-файлы, но во многих случаях работают в десятки и сотни раз быстрее. Программный интерфейс приложения позволяет использовать готовые модули на Fortran или C в приложениях MatLab. Не менее полезной оказывается возможность вызова функций MatLab из собственных программ на Fortran или C. Выбор оптимального сочетания ресурсов MatLab и собственных модулей позволяет достаточно быстро разработать средства решения многих сложных задач.

В настоящее время наиболее широко используется MatLab 5.3 и 6.0, недавно вышла версия 6.1. Пользователи, которые работали в версии 5.3, оценят более удобный интерфейс среды MatLab, появившийся, начиная с версии 6.0. Обновленная рабочая среда объединяет командное окно, историю команд и управление каталогами. Переход к справочной системе MatLab и ToolBox также осуществляется из рабочей среды. Сама гипертекстовая справочная система стала намного удобнее и полнее по сравнению с версией 5.3, в которой использовался браузер Web-страниц. К внешним отличиям версий 6.0 и 6.1 от 5.3 следует отнести более совершенный редактор и отладчик M-файлов, позволяющий, например, получать контекстный доступ к описанию команд, функций и операторов MatLab при помощи всплывающего меню. Создание приложений с графическим интерфейсом пользователя в среде GUIDE отличается в новых версиях MatLab по сравнению с 5.3, в частности используется другая схема организации файлов приложения. Среда GUIDE теперь автоматически генерирует M-файл с заголовками подфункций обработки событий объектов приложения.

Вышеперечисленные новшества касаются лишь "видимой части айсберга". Функции, реализующие некоторые численные методы (интегрирование, решение дифференциальных уравнений и задач матричного анализа), стали более эффективны. В состав пакета MatLab 6.0 вошли новые компоненты.

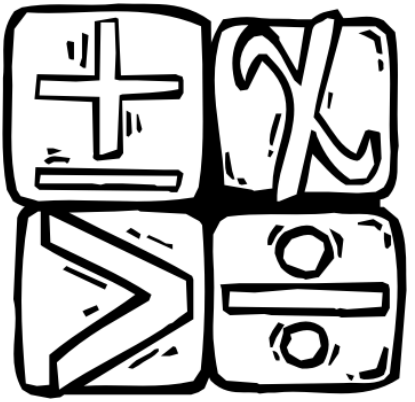
- Communications Blockset 2.0, расширяющий возможности Communications Toolbox 1.4, который входил в версию 5.3.
- Filter Design ToolBox 2.0, предоставляющий широкие возможности для разработки фильтров.
- Instrument Control ToolBox 1.0, позволяющий наладить интерфейс с внешними устройствами, подключаемыми к компьютеру через последовательный порт и универсальную шину интерфейса (GPIB).

- Simulink Performance Tools 4.0, расширяющий возможности пакета Simulink.
- Real-Time Workshop Embedded Coder, позволяющий оптимизировать С-код моделей, созданных при помощи пакета Simulink.

Представлены новые версии некоторых ToolBox. Появилось удобное средство для генерирования MEX-файлов из среды Microsoft Visual Studio.

MatLab 6.1 предлагает исследователям и разработчикам нелинейных динамических систем в среде Simulink технологию получения трехмерных анимационных моделей на основе языка моделирования виртуальной реальности VRML. В версии 6.1 улучшены некоторые математические алгоритмы, библиотека MatLab API пополнилась рядом функций для обеспечения интерфейса с модулями, написанными на языке Fortran. Несколько ToolBox в пакете MatLab 6.1 заменены обновленными версиями.

Данная книга ни в коей мере не претендует на полноту изложения. Достаточно сказать, что документация по MatLab и ToolBox весьма объемна, в частности, описание каждого из ToolBox PDE, Optimization составляет около трехсот страниц, для ToolBox Statistics — превосходит четыреста страниц. Описание ToolBox Signal Processing занимает почти восемьсот страниц. Следует иметь в виду, что информационная система позволяет не только научиться применять средства MatLab для решения различных задач, но и разобраться в особенностях реализованных методов. Огромное количество сведений, содержащихся в документации и справочной системе, оказывается полезным для исследователей и инженеров, владеющих основами работы в MatLab. Начинающий пользователь может просто запутаться в обилии информации. Предлагаемая вашему вниманию книга предназначена для тех читателей, которые хотят изучить принципы вычислений и программирования в MatLab и освоить работу в некоторых ToolBox. Углубление знаний в области решения специализированных задач потребует от читателя достаточно много самостоятельной работы. Список литературы, касающейся программирования и решения задач в MatLab, приведен в конце книги.



Часть I

Основы работы в MatLab

Глава 1. Простейшие вычисления

Глава 2. Работа с массивами

Глава 3. Высокоуровневая графика

Глава 4. Редактирование графиков

Глава 5. М-файлы

Глава 1



Простейшие вычисления

Данная глава посвящена описанию рабочей среды MatLab и вычислениям алгебраических выражений с использованием встроенных математических функций. Команды, с которых мы начнем, не очень длинные, поэтому для простоты будем работать из командной строки MatLab.

Рабочая среда MatLab

Рабочая среда MatLab 5.3 немного отличается от рабочей среды версии 6.*x*, однако принципиальных трудностей при изучении и программировании вычислений не возникает. Все вычисления в обеих версиях проводятся аналогично, но рабочая среда MatLab 6.*x* имеет более удобный интерфейс для доступа ко многим вспомогательным элементам MatLab.

При запуске MatLab 5.3 на экране появляется командное окно **MATLAB Command Window**, изображенное на рис. 1.1.

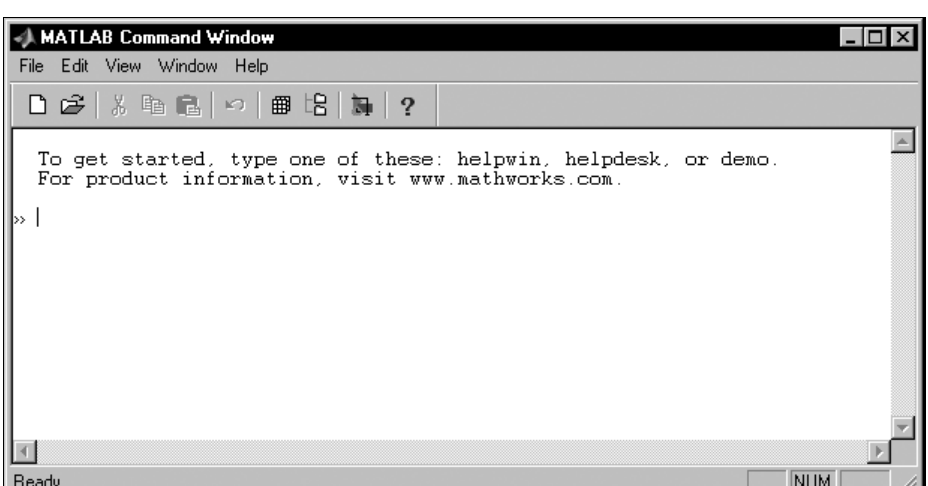


Рис. 1.1. Командное окно MatLab 5.3

Окно **MATLAB Command Window** состоит из следующих основных элементов:

- меню;
- панели с кнопками;
- рабочей области с командной строкой, в которой находится мигающий вертикальный курсор;
- строки состояния.

Все команды, описанные в этой главе, следует набирать в командной строке. Сам символ >>, обозначающий приглашение командной строки, приведенный в примерах, набирать не нужно. Для просмотра рабочей области удобно использовать полосы скроллинга или клавиши <Home>, <End> для перемещения влево или вправо и <PageUp>, <PageDown> для перемещения вверх или вниз. Про использование клавиш <↑>, <↓>, <→>, <←> будет сказано дополнительно. Если вдруг после перемещения по рабочей области командного окна пропала командная строка с мигающим курсором, просто нажмите <Enter>.

Важно запомнить, что набор любой команды или выражения должен заканчиваться нажатием на <Enter>, для того, чтобы программа MatLab выполнила эту команду или вычислила выражение.

Запуск MatLab 6.*x* приводит к открытию рабочей среды, изображенной на рис 1.2.

Замечание

Если в рабочей среде MatLab 6.*x* отсутствуют некоторые окна, приведенные на рис. 1.2, то следует в меню **View** выбрать соответствующие пункты: **Command Window**, **Command History**, **Current Directory**, **Workspace**, **Launch Pad**.

Рабочая среда MatLab 6.*x* предоставляет дополнительные удобства по сравнению с командным окном MatLab 5.3. Она содержит следующие элементы:

- меню;
- панель инструментов с кнопками и раскрывающимся списком;
- окно с вкладками **Launch Pad** и **Workspace**, из которого можно получить простой доступ к различным модулям ToolBox и к содержимому рабочей среды;
- окно с вкладками **Command History** и **Current Directory**, предназначенное для просмотра и повторного вызова ранее введенных команд, а также для установки текущего каталога;
- командное окно, работа в котором не отличается от работы в командном окне MatLab 5.3 (см. выше);
- строку состояния.

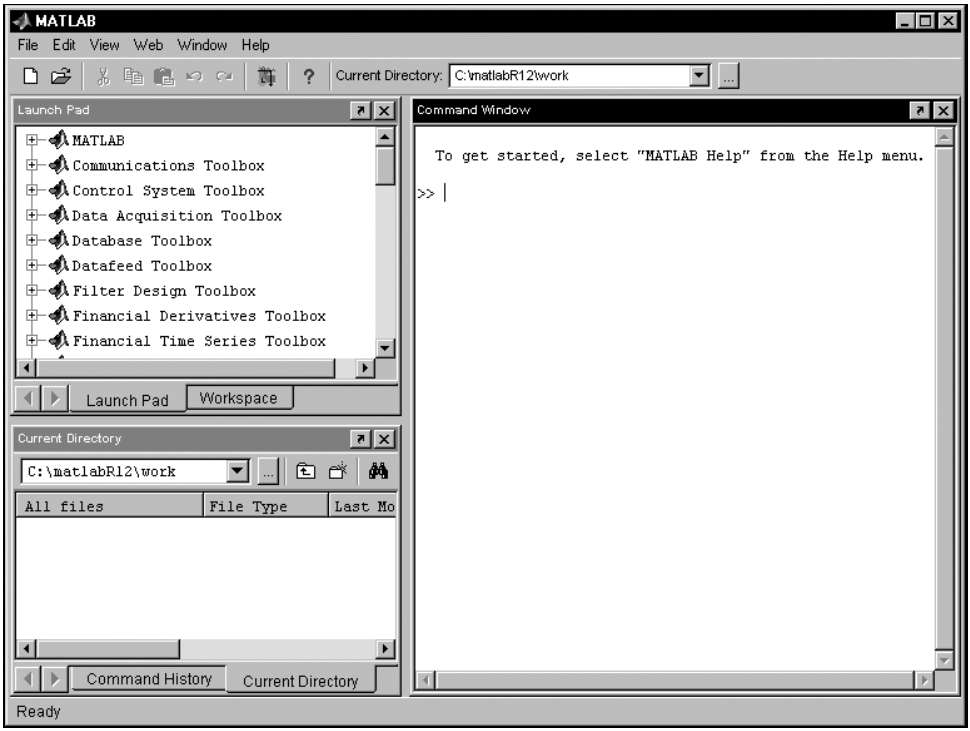


Рис. 1.2. Рабочая среда MatLab 6.*x*

При использовании MatLab 6.*x* все команды из этой главы следует набирать в командном окне.

Арифметические вычисления

Встроенные математические функции MatLab позволяют находить значения различных выражений. MatLab предоставляет возможность управления форматом вывода результата. Команды для вычисления выражений имеют вид, свойственный всем языкам программирования высокого уровня.

Простейшие вычисления

Наберите в командной строке `1+2` и нажмите <Enter>. В результате в командном окне MatLab отображается следующее:

```
>> 1 + 2
ans =
     3
>> |
```

Что сделала программа MatLab? Сначала она вычислила сумму 1+2, затем записала результат в специальную переменную `ans` и вывела ее значение, равное 3, в командное окно. Ниже ответа расположена командная строка с мигающим курсором, обозначающая, что MatLab готова к дальнейшим вычислениям. Можно набирать в командной строке новые выражения и находить их значения.

Если требуется продолжить работу с предыдущим выражением, например, вычислить `(1+2)/4.5`, то проще всего воспользоваться уже имеющимся результатом, который хранится в переменной `ans`. Наберите в командной строке `ans/4.5` (при вводе десятичных дробей используется точка) и нажмите <Enter>, получается:

```
>> ans/4.5
ans =
    0.6667
>> |
```

Предупреждение

Вид, в котором выводится результат вычислений, зависит от формата вывода, установленного в MatLab. Далее объяснено, как задать основные форматы вывода.

Форматы вывода результата вычислений

Требуемый формат вывода результата определяется пользователем из меню MatLab. Выберите в меню **File** пункт **Preferences**. На экране появится диалоговое окно **Preferences**, изображенное на рис. 1.3 (для версии 5.3), в котором следует перейти на вкладку **General**.

На панели **Numeric Format** расположены переключатели, при помощи которых устанавливается формат вывода результатов вычислений.

Диалоговое окно **Preferences** MatLab 6.*x* изображено на рис. 1.4. Для установки формата вывода следует убедиться, что в списке левой панели выбран пункт **Command Window** (как показано на рис. 1.4). Задание формата производится из раскрывающегося списка **Numeric format** панели **Text display**.

Разберем пока только наиболее часто используемые форматы. Установите переключатель **Short (default)**, если вы работаете в MatLab 5.3, или выберите **short** в раскрывающемся списке **Numeric format** в MatLab 6.*x*. Закройте диалоговое окно, нажав кнопку **OK**. Сейчас установлен короткий формат с плавающей точкой `short` для вывода результатов вычислений, при котором на экране отображаются только четыре цифры после десятичной точки. Наберите в командной строке `100/3` и нажмите <Enter>.

Результат выводится в формате `short`:

```
>> 100/3
ans =
   33.3333
```

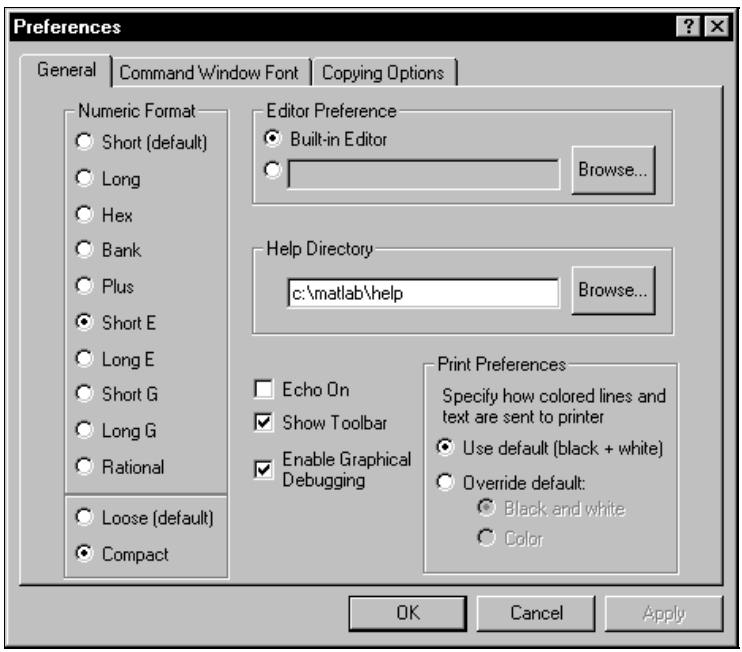


Рис. 1.3. Диалоговое окно **Preferences** MatLab 5.3

Этот формат вывода сохранится для всех последующих вычислений, если только не будет установлен другой формат. Заметьте, что в MatLab возможна ситуация, когда при отображении слишком большого или малого числа результат не укладывается в формат `short`. Вычислите 10 000/3, результат выводится в экспоненциальной форме:

```
>> 100000/3
ans =
  3.3333e+004
```

То же самое произойдет и при нахождении 1/3000:

```
>> 1/3000
ans =
  3.3333e-004
```

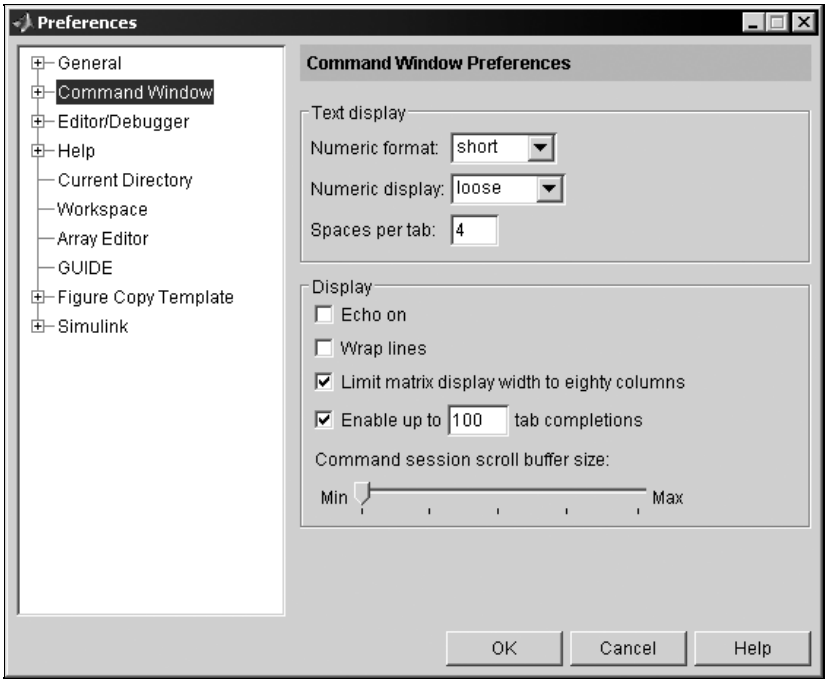


Рис. 1.4. Диалоговое окно **Preferences** MatLab 6.*x*

Однако, первоначальная установка формата сохраняется и при дальнейших вычислениях для небольших чисел вывод результата снова будет происходить в формате `short`.

В предыдущем примере MatLab вывела результат вычислений в *экспоненциальной форме*. Запись 3.3333e-004 обозначает $3.3333 \cdot 10^{-4}$ или 0.00033333. Аналогично можно набирать числа в выражениях. Например, проще набрать `10e9` или `1.0e10`, чем 1 000 000 000, а результат будет тот же самый. Пробел между цифрами и символом `e` при вводе не допускается, т. к. это приведет к сообщению об ошибке:

```
>> 10 e9
??? 10 e9
       |
Missing operator, comma, or semi-colon.
```

Если требуется получить результат вычислений более точно, то в диалоговом окне **Preferences** следует установить переключатель **Long** (для MatLab 5.3), или выбрать в раскрывающемся списке **long** (для MatLab 6.*x*). Результат будет отображаться в длинном формате с плавающей точкой `long` с четырнадцатью цифрами после десятичной точки. Форматы `short e` и `long e` предназначены для вывода результата в экспоненциальной форме с четырьмя и

пятнадцатью цифрами после десятичной точки соответственно. Информацию о форматах можно получить, набрав в командной строке команду `help` с аргументом `format`:

```
>> help format
```

В командном окне появляется описание каждого из форматов.

Задавать формат вывода можно непосредственно из командной строки при помощи команды `format`. Например, для установки длинного с плавающей точкой формата вывода результатов вычислений следует ввести команду `format long e` в командной строке:

```
>> format long e
>> 1.25/3.11
ans =
    4.019292604501608e-001
```

Обратите внимание, что команда `help format` выводит на экран название форматов прописными буквами. Однако команда, которую надо ввести, состоит из строчных букв. К этой особенности встроенной справки `help` надо привыкнуть. MatLab различает прописные и строчные буквы. Попытка набора команды прописными буквами приведет к ошибке:

```
>> FORMAT LONG E
??? FORMAT LONG
           |
Missing operator, comma, or semi-colon.
```

Для более удобного восприятия результата MatLab выводит результат вычислений через строку после вычисляемого выражения. Однако иногда бывает удобно разместить больше строк на экране, для чего следует в диалоговом окне **Preferences** установить переключатель **Compact** (MatLab 5.3) или выбрать **compact** из раскрывающегося списка (MatLab 6.*x*). Добавление пустых строк обеспечивается установкой переключателя **Loose** (MatLab 5.3) или выбором **loose** из раскрывающегося списка **Numeric display** (MatLab 6.*x*).

Замечание

Все промежуточные вычисления MatLab производит с *двойной точностью*, независимо от того, какой формат вывода установлен.

Использование элементарных функций

Предположим, что требуется вычислить значение следующего выражения:

$$e^{-2.5} \cdot (ln 11.3)^{0.3} - \sqrt{\frac{sin\, 2.45\pi + cos\, 3.78\pi}{tg 3.3}}.$$

Введите в командной строке это выражение в соответствии с правилами MatLab и нажмите <Enter>.

```
>> exp(-2.5)*log(11.3)^0.3-
↳sqrt((sin(2.45*pi)+cos(3.78*pi))/tan(3.3))
```

Ответ выводится в командное окно:

```
ans =
   -3.2105
```

При вводе выражения использованы встроенные функции MatLab для вычисления экспоненты, натурального логарифма, квадратного корня и тригонометрических функций. В следующем пункте приведены часто употребляемые встроенные математические функции. Аргументы функций заключаются в круглые скобки, имена функций набираются строчными буквами. Для ввода числа π достаточно набрать `pi` в командной строке.

Арифметические операции в MatLab выполняются в обычном порядке, свойственном большинству языков программирования:

- возведение в степень `^`;
- умножение и деление `*`, `/`;
- сложение и вычитание `+`, `-`.

Для изменения порядка выполнения арифметических операторов следует использовать круглые скобки.

Если теперь требуется вычислить значение выражения, похожего на предыдущее, например

$$e^{-2.5}\cdot(ln11.3)^{0.3}+\left(\frac{sin2.45\pi+cos3.78\pi}{tg3.3}\right)^2,$$

то необязательно снова набирать его в командной строке. Можно воспользоваться тем, что MatLab запоминает все вводимые команды. Для повторного занесения их в командную строку служат клавиши <↑>, <↓>. Вычислите данное выражение, проделав следующие шаги.

1. Нажмите клавишу <↑>, при этом в командной строке появится введенное ранее выражение.
2. Внесите в него необходимые изменения, заменив минус на плюс и квадратный корень на возведение в квадрат (для перемещения по строке с выражением служат клавиши <→>, <←>, <Home>, <End>).
3. Вычислите измененное выражение, нажав <Enter>.

Получается

```
>> exp(-2.5)*log(11.3)^0.3+((sin(2.45*pi)+cos(3.78*pi))/tan(3.3))^2
ans =
  121.2446
```

Если необходимо получить более точный результат, то следует выполнить команду `format long e`, затем нажимать клавишу <↑> до тех пор, пока в командной строке не появится требуемое выражения, и вычислить его, нажав <Enter>.

```
>> format long e
>> exp(-2.5)*log(11.3)^0.3+((sin(2.45*pi)+cos(3.78*pi))/tan(3.3))^2
ans =
    1.212446016556763e+002
```

Вывести результат последнего найденного выражения в другом формате можно без повторного вычисления. Следует изменить формат командой `short`, а затем посмотреть значение переменной `ans`, набрав ее в командной строке и нажав <Enter>:

```
>> format short
>> ans
ans =
  121.2446
```

В рабочей среде MatLab 6.*x* для вызова ранее введенных команд имеется дополнительное удобное средство — окно **Command History** с историей команд, изображенное на рис. 1.5. История команд содержит время и дату каждого сеанса работы с MatLab 6.*x*. Для активизации окна **Command History** необходимо выбрать вкладку с одноименным названием. Текущая команда в окне изображена на синем фоне (на рис. 1.5 — `format long e`). Если щелкнуть на какой-либо команде в окне левой кнопкой мыши, то данная команда становится текущей. Для ее выполнения в MatLab надо применить двойной щелчок мыши или выбрать строку с командой при помощи клавиш <↑>, <↓> и нажать клавишу <Enter>. Лишнюю команду можно убрать из окна. Для этого ее надо сделать текущей и удалить при помощи клавиши <Delete>. Можно выделить несколько идущих подряд команд при помощи комбинации клавиш <Shift>+<↑>, <Shift>+<↓> и выполнить их при помощи <Enter> или удалить клавишей <Delete>. Выделение последовательно идущих команд можно производить левой кнопкой мыши с одновременным удерживанием клавиши <Shift>. Если команды не идут одна за другой, то для их выделения следует использовать левую кнопку мыши с удерживанием клавиши <Ctrl>.

При щелчке правой кнопкой мыши по области окна **Command History** появляется всплывающее меню. Выбор пункта **Copy** приводит к копированию команды в буфер Windows. При помощи **Evaluate Selection** можно выполнить отмеченную группу команд. Для удаления текущей команды предназначен пункт **Delete Selection**, для удаления всех команд до текущей — **Delete to Selection**, для удаления всех команд — **Delete Entire History**.

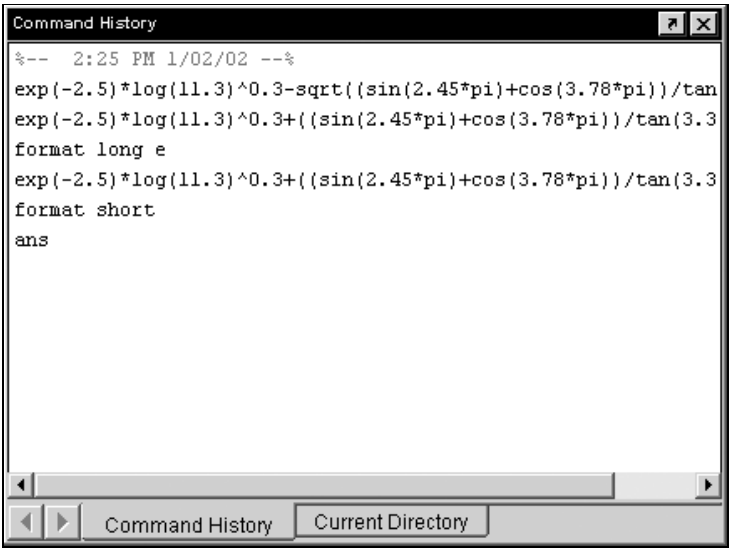


Рис. 1.5. Окно **Command History** MatLab 6.*x*

При вычислениях возможны некоторые исключительные ситуации, например деление на ноль, которые в большинстве языков программирования приводят к ошибке. При делении положительного числа на ноль в MatLab получается `Inf` (бесконечность), а при делении отрицательного числа на ноль получается `-Inf` (минус бесконечность) и выдается предупреждение:

```
>> 1/0
Warning: Divide by zero.
ans =
   Inf
```

При делении нуля на ноль получается `NaN` (не число) и также выдается предупреждение:

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

При вычислении, например $\sqrt{-1}$, никакой ошибки или предупреждения не возникает. MatLab автоматически переходит в область комплексных чисел:

```
>> sqrt(-1.0)
ans =
        0 + 1.0000i
```

При наборе комплексных чисел в командной строке MatLab можно использовать либо `i`, либо `j`, а сами числа при умножении, делении и возведении в степень необходимо заключать в круглые скобки:

```
>> (2.1+3.2i)*2 + (4.2+1.7i)^2
ans =
  18.9500 +20.6800i
```

Если не использовать скобки, то умножаться или возводиться в степень будет только мнимая часть и получится неверный результат:

```
>> 2.1+3.2i*2 + 4.2+1.7i^2
ans =
   3.4100 + 6.4000i
```

Для вычисления комплексно-сопряженного числа применяется апостроф, который следует набирать сразу за числом, без пробела:

```
>> 2-3i'
ans =
   2.0000 + 3.0000i
```

Если необходимо найти комплексно-сопряженное выражение, то исходное выражение должно быть заключено в круглые скобки:

```
>> ((3.2+1.5i)*2+4.2+7.9i)'
ans =
  10.6000 -10.9000i
```

MatLab позволяет использовать комплексные числа в качестве аргументов встроенных элементарных функций:

```
>> sin(2+3i)
ans =
   9.1545 - 4.1689i
```

Как узнать, какие встроенные элементарные функции можно использовать и как их вызывать? Наберите в командной строке команду `help elfun`, при этом в командное окно выводится список всех встроенных элементарных функций с их кратким описанием. В следующем разделе приведены часто используемые функции.

Встроенные элементарные функции

Встроенные элементарные функции MatLab включают тригонометрические, гиперболические, экспоненциальные и логарифмические функции, а также функции для работы с комплексными числами и для округления различными способами.

Тригонометрические, гиперболические и обратные к ним функции

Ниже перечислены встроенные в MatLab тригонометрические функции и обратные к ним:

- `sin`, `cos`, `tan`, `cot` — синус, косинус, тангенс и котангенс;
- `sec`, `csc` — секанс, косеканс ($sec(x) = \frac{1}{cos(x)}$, $csc(x) = \frac{1}{sin(x)}$);
- `asin`, `acos`, `atan`, `acot` — арксинус, арккосинус, арктангенс и арккотангенс;
- `asec`, `acsc` — арксеканс, арккосеканс.

> **Замечание**
>
> Аргументы тригонометрических функций должны быть выражены в радианах. Обратные тригонометрические функции возвращают результат также в радианах.

В MatLab встроены следующие гиперболические функции и обратные к ним:

- `sinh`, `cosh`, `tanh`, `coth` — гиперболические синус, косинус, тангенс и котангенс;
- `sech`, `csch` — гиперболические секанс и косеканс;
- `asinh`, `acosh`, `atanh`, `acoth` — гиперболические арксинус, арккосинус, арктангенс и арккотангенс;
- `asech`, `acsch` — гиперболические арксеканс и арккосеканс.

Экспоненциальная функция, логарифмы, степенные функции

Ниже перечислены примеры этих функций в MatLab:

- `exp` — экспоненциальная функция;
- `log` — натуральный логарифм;
- `log10` — десятичный логарифм;
- `log10` — логарифм по основанию 2;
- `pow2` — возведение числа 2 в степень;
- `sqrt` — квадратный корень;
- `nextpow2` — степень, в которую надо возвести число 2, чтобы получить ближайшее число (большее или равное аргументу), например

```
>> nextpow2(1000)
ans =
    10
```

Функции для работы с комплексными числами

К ним относятся следующие функции MatLab:

- `abs`, `angle` — модуль r и фаза φ (в радианах от $-\pi$ до π) комплексного числа $a+i\cdot b=r\cdot(cos\varphi+i\cdot sin\varphi)$;
- `complex` — конструирует комплексное число по его действительной и мнимой части:

```
>> complex(2.3, 5.8)
ans =
   2.3000 + 5.8000i;
```

- `conj` — возвращает комплексно-сопряженное число;
- `imag`, `real` — возвращает мнимую и действительную часть комплексного числа.

Округление и остаток от деления

Ниже приведены примеры использования этих функций в MatLab:

- `fix` — округление до ближайшего целого по направлению к нулю:

```
>> fix(1.8)          >> fix(-1.9)
ans =                ans =
     1                    -1
```

- `floor`, `ceil` — округление до ближайшего целого по направлению к минус бесконечности или плюс бесконечности:

```
>> floor(3.2)        >> ceil(3.2)
ans =                ans =
     3                    4
```

- `round` — округление до ближайшего целого:

```
>> round(4.1)        >> round(4.5)
ans =                ans =
     4                    5
```

- `mod` — остаток от целочисленного деления (со знаком):

```
>> mod(5,2)          >> mod(5,-2)
ans =                ans =
     1                    -1
```

- `rem` — остаток от целочисленного деления:

```
>> rem(5,2)          >> rem(5,-2)
ans =                ans =
     1                    1
```

- `sign` — возвращает знак числа.

Замечание

В MatLab имеются встроенные специальные математические функции, такие как: функция Бесселя, полиномы Лежандра и т. д. Подробную информацию с примерами *см. в приложении 1*. Команда `help specfun` выдает список специальных функций с кратким описанием. Для более подробной информации требуется набрать в командной строке `help` и имя функции.

Использование переменных

Как и во всех языках программирования, в MatLab предусмотрена возможность работы с переменными. Причем пользователь не должен заботиться о том, какие значения будет принимать переменная (комплексные, вещественные или только целые). Для того чтобы присвоить, например, переменной `z` значение 1.45, достаточно написать в командной строке `z=1.45`, при этом MatLab сразу же выведет значение `z`:

```
>> z = 1.45
z =
    1.4500
```

Здесь знак равенства используется в качестве *оператора присваивания*. Часто не очень удобно после каждого присваивания получать еще и результат. Поэтому в MatLab предусмотрена возможность завершать оператор присваивания точкой с запятой для подавления вывода результата в командное окно. Именем переменной может быть любая последовательность букв и цифр без пробела, начинающаяся с буквы. Строчные и прописные буквы различаются, например `Mz` и `mZ` являются двумя разными переменными. Количество воспринимаемых MatLab символов в имени переменной составляет 31 (такие длинные имена и не нужны).

В качестве упражнения на использование переменных найдите значение следующего выражения:

$$\frac{\frac{sin1.3\pi}{ln3.4}+\sqrt{\frac{tg2.75}{th2.75}}}{\frac{sin1.3\pi}{ln3.4}-\sqrt{\frac{tg2.75}{th2.75}}}.$$

Наберите последовательность команд, приведенную ниже (обратите внимание на точку с запятой в первых двух операторах присваивания для подавления вывода промежуточных значений на экран):

```
>> x = sin(1.3*pi)/log(3.4);
>> y = sqrt(tan(2.75)/tanh(2.75));
```

```
>> z = (x+y)/(x-y)
z =
   0.0243 - 0.9997i
```

Последний оператор присваивания не завершается точкой с запятой для того, чтобы сразу получить значение исходного выражения. Конечно, можно было бы ввести сразу всю формулу и получить тот же результат:

```
>> (sin(1.3*pi)/log(3.4)+sqrt(tan(2.75)/tanh(2.75)))/(sin(1.3*pi)/...
log(3.4)-sqrt(tan(2.75)/tanh(2.75)))
ans =
   0.0243 - 0.9997i
```

Но обратите внимание, насколько первая запись компактнее и яснее второй! Во втором варианте формула не помещалась в командном окне на одной строке, и пришлось записать ее в две строки, для чего в конце первой строки поставлены три точки.

Замечание

Для ввода длинных формул или команд в командную строку следует поставить три точки (подряд, без пробелов), нажать клавишу <Enter> и продолжить набор формулы на следующей строке. Так можно разместить выражение на нескольких строках. MatLab вычислит все выражение или выполнит команду после нажатия на <Enter> в последней строке (в которой нет трех идущих подряд точек).

MatLab запоминает значения всех переменных, определенных во время сеанса работы. Если после ввода примера, приведенного выше, были проделаны еще какие-либо вычисления, и возникла необходимость вывести значение `x`, то следует просто набрать `x` в командной строке и нажать <Enter>:

```
>> x
x =
   -0.6611
```

Переменные, определенные выше, можно использовать и в других формулах. Например, если теперь необходимо вычислить выражение

$$\left[\frac{sin1.3\pi}{ln3.4}-\sqrt{\frac{tg\,2.75}{th2.75}}\right]^{3/2},$$

то достаточно ввести следующую команду:

```
>> (x-y)^(3/2)
ans =
  -0.8139 + 0.3547i
```

Вызов функций в MatLab обладает достаточной гибкостью. Например, вычислить $e^{3.5}$ можно, вызвав функцию `exp` из командной строки:

```
>> exp(3.5)
ans =
   33.1155
```

Другой способ состоит в использовании оператора присваивания:

```
>> t = exp(3.5)
t =
   33.1155
```

Предположим, что часть вычислений с переменными выполнена, а остальные придется доделать во время следующего сеанса работы с MatLab. В этом случае понадобится сохранить переменные, определенные в рабочей среде.

Сохранение рабочей среды

Самый простой способ сохранить значения всех переменных — использовать в меню **File** пункт **Save Workspace As**. При этом появляется диалоговое окно **Save Workspace As** (в версии 5.3) или **Save Workspace Variables** (в версии 6.*x*), в котором следует указать каталог и имя файла. По умолчанию предлагается сохранить файл в подкаталоге work основного каталога MatLab. Оставьте пока этот каталог. В дальнейшем будет объяснено, как устанавливать пути к каталогам в MatLab для поиска файлов. Удобно давать файлам имена, содержащие дату работы, например work20-09-01. MatLab сохранит результаты работы в файле work20-09-01.mat. Теперь можно закрыть MatLab одним из следующих способов:

- выбрать в меню File пункт Exit MATLAB;
- нажать клавиши <Ctrl>+<Q>;
- набрать команду `Exit` в командной строке и нажать <Enter>;
- нажать на кнопку с крестиком в правом верхнем углу окна программы MatLab.

В следующем сеансе работы для восстановления значений переменных следует открыть файл work20-09-01.mat при помощи подпункта **Load Workspase** (в версии 5.3) или **Open** (в версии 6.*x*) меню **File**. Теперь все переменные, определенные в прошлом сеансе, стали доступными. Их можно использовать во вновь вводимых командах.

Сохранение и восстановление переменных рабочей среды можно выполнить и из командной строки. Для этого служат команды `save` и `load`. В конце сеанса работы с MatLab надо выполнить команду

```
>> save work20-09-01
```

Расширение можно не указывать, MatLab сохранит переменные рабочей среды в файле work20-09-01.mat. В начале следующего сеанса работы для считывания переменных следует ввести команду

```
>> load work20-09-01
```

Подробную информацию о командах `save` и `load` можно получить, набрав в командной строке `help save` или `help load`.

Замечание

Переменные в файлах с расширением mat хранятся в двоичном виде. Просмотр этих файлов в любом текстовом редакторе не даст никакой информации о переменных и их значениях.

В MatLab имеется возможность записывать исполняемые команды и результаты в текстовый файл (вести журнал работы), который потом можно легко прочитать или распечатать из текстового редактора. Для начала ведения журнала служит команда `diary`. В качестве аргумента команды `diary` следует задать имя файла, в котором будет храниться журнал работы. Набираемые далее команды и результаты их исполнения будут записываться в этот файл, например последовательность команд

```
>> diary d20-09-01.txt
>> a1 = 3;
>> a2 = 2.5;
>> a3 = a1 + a2
>> a3 =
>>     5.5000
>> save work20-09-01
>> quit
```

производит следующие действия:

1. Открывает файл d20-09-01.txt.
2. Производит вычисления.
3. Сохраняет переменные в двоичном файле work20-09-01.mat.
4. Сохраняет на диске в подкаталоге work корневого каталога MatLab журнал работы в файле d20-09-01.txt и закрывает MatLab.

Посмотрите содержимое файла d20-09-01.txt в каком-нибудь текстовом редакторе, например в стандартной программе Windows Блокнот (NotePad). В файле окажется следующий текст:

```
a1 = 3;
a2 = 2.5;
a3 = a1+a2
```

```
a3 =
    5.5000
save work20-09-01
quit
```

Запустите снова MatLab и введите команду `load work20-09-01`, или откройте файл work20-09-01.mat при помощи меню, как описано в начале этого пункта. Изучим возможность просмотра переменных, определенных в рабочей среде.

Просмотр переменных

При работе с достаточно большим количеством переменных необходимо знать, какие переменные уже использованы, а какие нет. Для этой цели служит команда `who`, выводящая в командное окно MatLab список используемых переменных:

```
>> who
Your variables are:
a1        a2        a3
```

Команда `whos` позволяет получить более подробную информацию о переменных в виде таблицы:

```
>> whos
   Name      Size            Bytes   Class
   a1        1x1             8       double array
   a2        1x1             8       double array
   a3        1x1             8       double array
Grand total is 3 elements using 24 bytes
```

Первый столбик `Name` состоит из имен используемых переменных. То, что содержится в столбике `Size`, по существу, определяется основным принципом работы MatLab. Программа MatLab *все данные представляет в виде массивов.* Переменные `a1`, `a2` и `a3` являются двумерными массивами размера один на один. Каждая из переменных занимает по восемь байтов, как указано в столбике `Bytes`. Наконец, в последнем столбике `Class` указан тип переменных — `double array`, т. е. массив, состоящий из чисел двойной точности. В строке под таблицей написано, что в итоге три элемента, т. е. переменные, занимают двадцать четыре байта. Оказывается, что представление всех данных в MatLab в виде массивов дает определенные преимущества.

Работа с массивами подробно описана в главе 2.

Для освобождения из памяти всех переменных используется команда `clear`. Если в аргументах указать список переменных (через пробел), то только они будут освобождены из памяти, например:

```
>> clear a1 a3
>> who
Your variables are:
a2
```

Начиная с версии 6.0, появилось удобное средство для просмотра переменных рабочей среды — окно **Workspace** (рис. 1.6), для перехода к которому следует активизировать одноименную закладку. Данное окно содержит таблицу, аналогичную той, что выводится командой `whos`. Двойной щелчок по строке, соответствующей каждой переменной, приводит к отображению ее содержимого в отдельном окне, что особенно полезно при работе с массивами. Панель инструментов окна **Workspace** позволяет удалить лишние переменные, сохранить и открыть рабочую среду.

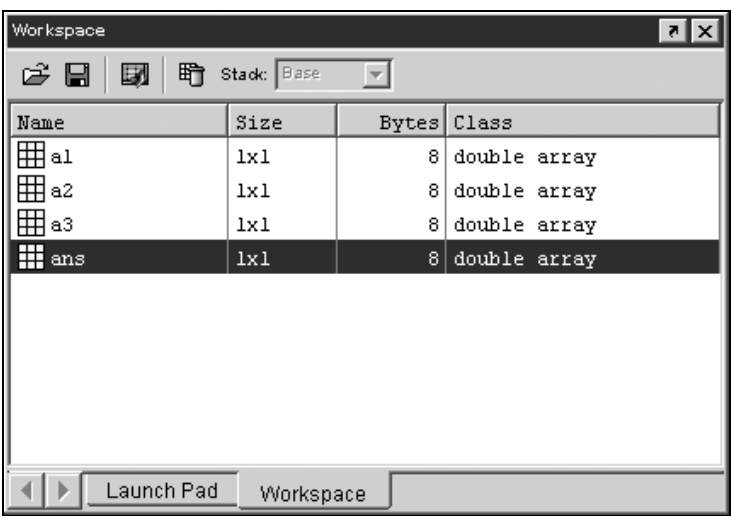


Рис. 1.6. Окно **Workspace** MatLab 6.*x*

Глава 2



Работа с массивами

Все данные MatLab представляет в виде массивов. Очень важно правильно понять, как использовать массивы. Без этого невозможна эффективная работа в MatLab, в частности построение графиков, решение задач линейной алгебры, обработки данных, статистики и многих других. В этой главе подробно описаны вычисления с векторами и матрицами. Даже если вы знакомы с каким-либо языком программирования, все равно лучше прочесть эту главу. MatLab предоставляет пользователю обширные возможности для работы с массивами данных. Следующий раздел посвящен необходимым сведениям, касающимся массивов.

Основные определения и соглашения

Что такое массив, должно быть известно каждому, кто хоть немного занимался программированием. *Массив* — упорядоченная, пронумерованная совокупность однородных данных. У массива должно быть *имя*. Массивы различаются по числу *размерностей* или *измерений*: одномерные, двумерные, многомерные. *Размером* массива называют число элементов, вдоль каждого из измерений. Доступ к элементам осуществляется при помощи *индекса*. В MatLab нумерация элементов массивов начинается с единицы. Это значит, что индексы должны быть больше или равны единице.

Важно понять, что вектор, вектор-строка, матрица или тензор являются математическими объектами, а одномерные, двумерные или многомерные массивы — способы хранения этих объектов в компьютере. Всюду дальше в книге будут использоваться слова вектор, матрица и тензор, если больший интерес представляет сам объект, чем способ его хранения. Вектор может быть записан в столбик (вектор-столбец) и в строку (вектор-строка). Вектор-столбцы и вектор-строки часто будут называться просто векторами, различие будет сделано в тех случаях, если важен способ хранения вектора в MatLab. Векторы и матрицы обозначаются курсивом, а соответствующие им массивы прямым моноширинным шрифтом, например: "вектор a содержится в массиве `a`", "запишите матрицу R в массив `R`".

Вектор-столбцы и вектор-строки

Ввод, сложение и вычитание векторов

Работу с массивами начнем с простого примера — вычисления суммы векторов

$$a = \begin{pmatrix} 1.3 \\ 5.4 \\ 6.9 \end{pmatrix} \qquad b = \begin{pmatrix} 7.1 \\ 3.5 \\ 8.2 \end{pmatrix}.$$

Для хранения векторов используйте массивы `a` и `b`. Введите массив `a` в командной строке, используя квадратные скобки и разделяя элементы вектора точкой с запятой:

```
>> a = [1.3; 5.4; 6.9]
a =
    1.3000
    5.4000
    6.9000
```

Замечание

В предыдущей главе было сказано, что точка с запятой в конце выражения используется для подавления вывода результата выражения на экран. Оказывается, что этот символ предназначен и для разделения элементов векторов.

Так как введенное выражение не завершено точкой с запятой, то MatLab автоматически вывела значение переменной `a`. Введите теперь второй вектор, подавив вывод на экран

```
>> b = [7.1; 3.5; 8.2];
```

Для нахождения суммы векторов используется знак `+`. Вычислите сумму, запишите результат в массив `c` и выведите его элементы в командное окно:

```
>> c = a + b
c =
    8.4000
    8.9000
   15.1000
```

Узнайте размерность и размер массива `a` при помощи встроенных функций `ndims` и `size`:

```
>> ndims(a)
ans =
     2
```

```
>> size(a)
ans =
     3     1
```

Итак, вектор *a* хранится в двумерном массиве `a` размерностью три на один (вектор-столбец из трех строк и одного столбца). Аналогичные операции можно проделать и для массивов `b` и `c`. В *главе 1* было замечено, что числа в MatLab представляются в виде двумерного массива один на один. Теперь должно быть понятно, почему при сложении векторов используется тот же знак плюс, что и для сложения чисел. Естественно, для нахождения разности векторов следует применять знак минус, с умножением дело обстоит несколько сложнее.

Замечание

Если размеры векторов, к которым применяется сложение или вычитание, не совпадают, то выдается сообщение об ошибке.

Особенность MatLab представлять все данные в виде массивов является очень удобной. Пусть, например, требуется вычислить значение функции `sin` сразу для всех элементов вектора *c* (который хранится в массиве `c`) и записать результат в вектор *d*. Используйте следующий оператор присваивания:

```
>> d = sin(c)
d =
    0.8546
    0.5010
    0.5712
```

Итак, встроенные в MatLab элементарные функции приспосабливаются к виду аргументов; если аргумент является массивом, то результат функции *будет массивом того же размера*, но с элементами, равными значению функции от соответствующих элементов исходного массива. Убедитесь в этом еще на одном примере. Если необходимо найти квадратный корень из элементов вектора *d* со знаком минус, то достаточно записать:

```
>> sqrt(-d)
ans =
        0 + 0.9244i
        0 + 0.7078i
        0 + 0.7558i
```

Оператор присваивания не использовался, поэтому MatLab записала ответ в стандартную переменную `ans`.

Ввод вектор-строки осуществляется в квадратных скобках, однако элементы следует разделять пробелами или запятыми. Операции сложения, вычитания и вычисление элементарных функций от вектор-строк производятся так же,

как и с вектор-столбцами, в результате получается вектор-строка того же размера, что и исходные.

```
>> s1 = [3 4 9 2]
s1 =
     3     4     9     2
>> s2 = [5 3 3 2]
s2 =
     5     3     3     2
>> s3 = s1 + s2
s3 =
     8     7    12     4
>> s4=log(s3)
s4 =
    2.0794    1.9459    2.4849    1.3863
```

Выясните, в каких массивах хранятся вектор-строки. Для этого можно использовать функции `ndims` и `size` или команду `whos`:

```
>> whos
 Name      Size          Bytes  Class
  s1        1x4             32  double array
  s2        1x4             32  double array
  s3        1x4             32  double array
  s4        1x4             32  double array
```

Итак, вектор-строки `s1`, `s2`, `s3` и `s4` содержатся в двумерных массивах размерностью один на четыре. Для определения длины векторов или вектор-строк служит встроенная функция `length`:

```
>> length(s1)
ans =
     4
```

Из нескольких вектор-столбцов можно составить один, используя квадратные скобки и разделяя исходные вектор-столбцы точкой с запятой:

```
>> v1 = [1; 2];
>> v2 = [3; 4; 5];
>> v = [v1; v2]
v =
     1
     2
     3
     4
     5
```

Для сцепления вектор-строк также применяются квадратные скобки, но сцепляемые вектор-строки отделяются пробелами или запятыми:

```
>> v1 = [1 2];
>> v2 = [3 4 5];
>> v = [v1 v2]
v =
     1     2     3     4     5
```

Обращение к элементам вектора

Доступ к элементам вектора или вектор-строки осуществляется при помощи индекса, заключаемого в круглые скобки после имени массива, в котором хранится вектор. Если среди переменных рабочей среды есть массив `v`, определенный вектор-строкой

```
>> v = [1.3 3.6 7.4 8.2 0.9];
```

то для вывода, например его четвертого элемента, используется *индексация*:

```
>> v(4)
ans =
    8.2000
```

Появление элемента массива в левой части оператора присваивания приводит к изменению в массиве

```
>> v(2) = 555
v =
    1.3000  555.0000    7.4000    8.2000    0.9000
```

Из элементов массива можно формировать новые массивы, например

```
>> u = [v(3); v(2); v(1)]
u =
    7.4000
  555.0000
    1.3000
```

Для помещения определенных элементов вектора в другой вектор в заданном порядке служит *индексация при помощи вектора*. Запись в массив `w` четвертого, второго и пятого элементов `v` производится следующим образом:

```
>> ind = [4 2 5];
>> w = v(ind)
w =
    8.2000  555.0000    0.9000
```

MatLab предоставляет удобный способ обращения к блокам последовательно расположенных элементов вектора или вектор-строки. Для этого служит *индексация при помощи знака двоеточия*. Предположим, что в массиве `w`, соответствующем вектор-строке из семи элементов, требуется заменить нулями элементы со второго по шестой. Индексация при помощи двоеточия позволяет просто и наглядно решить поставленную задачу:

```
>> w = [0.1 2.9 3.3 5.1 2.6 7.1 9.8];
>> w(2:6) = 0;
>> w
w =
    0.1000         0         0         0         0         0    9.8000
```

Присваивание `w(2:6)=0` эквивалентно последовательности команд `w(2)=0; w(3)=0; w(4)=0; w(5)=0; w(6)=0`.

Индексация при помощи двоеточия оказывается удобной при выделении части из большого объема данных в новый массив:

```
>> w = [0.1 2.9 3.3 5.1 2.6 7.1 9.8];
>> w1 = w(3:5)
w1 =
    3.3000    5.1000    2.6000
```

Составьте массив `w2`, содержащий элементы `w` кроме четвертого. Используйте двоеточие и сцепление строк:

```
>> w2 = [w(1:3) w(5:7)]
w2 =
    0.1000    2.9000    3.3000    2.6000    7.1000    9.8000
```

Элементы массива могут входить в выражения. Нахождение, например среднего геометрического из элементов массива `u`, можно проделать следующим образом:

```
>> gm = (u(1)*u(2)*u(3))^(1/3)
gm =
   17.4779
```

Конечно, этот способ не очень удобен для длинных массивов. Для того чтобы найти среднее геометрическое, необходимо набрать в формуле все элементы массива. В MatLab существует достаточно много специальных функций, облегчающих подобные вычисления.

Применение функций обработки данных к векторам

Перемножение элементов вектора или вектора-строки осуществляется при помощи функции `prod`:

```
>> z = [3; 2; 1; 4; 6; 5];
>> p = prod(z)
p =
   720
```

Зная об этой функции, несложно догадаться, как просто найти среднее квадратичное элементов вектора *z*:

```
>> gm = prod(z)^(1/length(z))
gm =
    2.9938
```

Функция `sum` предназначена для суммирования элементов вектора. Попробуйте самостоятельно вычислить среднее арифметическое элементов вектора *z*. Проверьте результат, вычислив среднее арифметическое используя встроенную функцию `mean`. Вот что должно получиться:

```
>> sum(z)/length(z)
ans =
    3.5000
>> mean(z)
ans =
    3.5000
```

Для нахождения минимума и максимума из элементов вектора служат встроенные функции `min` и `max`:

```
>> M=max(z)
M =
     6
>> m=min(z)
m =
     1
```

Часто необходимо знать не только значение минимального или максимального элемента в массиве, но и его индекс (порядковый номер). Вы уже видели, что вызов функции в MatLab достаточно универсален, более того, функции MatLab изменяют число выходных аргументов в зависимости от способа обращения к ним.

Вызовите, например, функцию `min` с двумя выходными аргументами:

```
>> [m, k] = min(z)
m =
     1
k =
     3
```

В результате переменной `m` будет присвоено значение минимального элемента массива `z`, а номер минимального элемента занесен в переменную `k`. Как же узнать, как именно можно вызывать функцию. Для этого следует набрать в командной строке `help` и имя функции. MatLab выведет в командное окно всевозможные способы обращения к функции с дополнительными пояснениями.

В число основных функций для работы с векторами входит функция упорядочения вектора по возрастанию его элементов `sort`.

```
>> r = [9.4 -2.3 -5.2 7.1 0.8 1.3];
>> R = sort(r)
R =
   -5.2000   -2.3000    0.8000    1.3000    7.1000    9.4000
```

Попробуйте упорядочить вектор по убыванию, используя эту же функцию `sort`. Правильный ответ:

```
>> R1 = -sort(-r)
R1 =
    9.4000    7.1000    1.3000    0.8000   -2.3000   -5.2000
```

Упорядочение элементов в порядке возрастания их модулей производится с привлечением вышеописанной функции `abs`:

```
>> R2 = sort(abs(r))
R2 =
    0.8000    1.3000    2.3000    5.2000    7.1000    9.4000
```

Вызов `sort` с двумя выходными аргументами приводит к образованию массива индексов соответствия элементов упорядоченного и исходного массивов:

```
>> [rs, ind] = sort(r)
rs =
   -5.2000   -2.3000    0.8000    1.3000    7.1000    9.4000
ind =
     3     2     5     6     4     1
```

Равенство `r(ind(k))=rs(k)` для `k` от 1 до `length(r)` связывает исходный массив `r`, упорядоченный `rs` и массив индексов `ind`.

Если аргументом функций `max` и `min` является вектор, состоящий из комплексных чисел, то результатом является максимальный или минимальный по модулю элемент. Функция `sort` также упорядочивает комплексный вектор по модулю, а компоненты с равными модулями располагаются в порядке возрастания фаз.

В число встроенных функций входит дискретное преобразование Фурье — `fft`, свертка — `conv`, работа со звуком — `sound` и многие другие. Подробно о них написано в *приложении 1*. Самостоятельно о функциях обработки данных можно узнать, набрав в командной строке команду `help datafun`. В последующих разделах описано применение функций обработки данных к матричным данным.

Замечание

Дополнительные функции содержатся в специализированных ToolBox. Команда `help stats` выводит список статистических функций, доступных в MatLab, если установка MatLab включает Statistics Toolbox.

Поэлементные операции с векторами

В предыдущих разделах вектор использовался в качестве аргумента математических функций, результатом которых являлся вектор с элементами, равными значениям функции от соответствующих элементов исходного вектора. Таким образом, происходило *поэлементное* вычисление вызываемой функции. В этом разделе подробно описаны возможности поэлементной работы с векторами, которые понадобятся в дальнейшем для определения собственных функций и построения их графиков.

Введите две вектор-строки:

```
>> v1 = [2 -3 4 1];
>> v2 = [7 5 -6 9];
```

Операция `.*` (не вставляйте пробел между точкой и звездочкой!) приводит к поэлементному умножению векторов одинаковой длины. В результате получается вектор с элементами, равными произведению соответствующих элементов исходных векторов:

```
>> u = v1.*v2
u =
    14   -15   -24     9
```

При помощи `.^` осуществляется поэлементное возведение в степень:

```
>> p = v1.^2
p =
     4     9    16     1
```

Показателем степени может быть вектор той же длины, что и возводимый в степень. При этом каждый элемент первого вектора возводится в степень, равную соответствующему элементу второго вектора:

```
>> P = v1.^v2
P =
  128.0000 -243.0000    0.0002    1.0000
```

Деление соответствующих элементов векторов одинаковой длины выполняется с использованием операции ./

```
>> d = v1./v2
d =
    0.2857   -0.6000   -0.6667    0.1111
```

Обратное поэлементное деление (деление элементов второго вектора на соответствующие элементы первого) осуществляется при помощи операции .\

```
>> dinv = v1.\v2
dinv =
    3.5000   -1.6667   -1.5000    9.0000
```

Итак, точка в MatLab используется не только для ввода десятичных дробей, но и для указания того, что деление или умножение массивов одинакового размера должно быть выполнено поэлементно.

К поэлементным относятся и операции с вектором и числом. Сложение вектора и числа не приводит к сообщению об ошибке. MatLab прибавляет число к каждому элементу вектора. То же самое справедливо и для вычитания:

```
>> v = [4 6 8 10];
>> s = v + 1.2
s =
    5.2000    7.2000    9.2000   11.2000
>> s1 = 1.2 + v
s1 =
    5.2000    7.2000    9.2000   11.2000
>> r = 1.2 - v
r =
   -2.8000   -4.8000   -6.8000   -8.8000
>> r1 = v - 1.2
r1 =
    2.8000    4.8000    6.8000    8.8000
```

Умножать вектор на число можно как справа, так и слева:

```
>> v = [4 6 8 10];
>> p = v*2
```

```
p =
     8    12    16    20
>> p1 = 2*v
p1 =
     8    12    16    20
```

Делить при помощи знака / можно вектор на число:

```
>> p = v/2
p =
     2     3     4     5
```

Попытка деления числа на вектор приводит к сообщению об ошибке:

```
>> p = 2/v
??? Error using ==> /
Matrix dimensions must agree.
```

Это связано с тем, что операция / в MatLab предназначена, в частности, для решения систем линейных алгебраических уравнений. Если требуется разделить число на каждый элемент вектора и записать результат в новый вектор, то следует использовать операцию ./

```
>> w = [4 2 6];
>> d = 12./w
d =
     3     6     2
```

Все вышеописанные операции применимы как к вектор-строкам, так и к вектор-столбцам.

Разберем, как правильно транспонировать и вычислять сопряженные векторы в MatLab. Для вектор-столбца u, к примеру с тремя комплексными элементами (в частности и с вещественными), *сопряженный* к нему u^* определяется как вектор-строка из его комплексно-сопряженных элементов, а *транспонированный* u^{T} — просто как вектор-строка из его элементов, например

$$u = \begin{bmatrix} 2+3i \\ 1-2i \\ 3+2i \end{bmatrix} \quad u^* = \begin{bmatrix} 2-3i & 1+2i & 3-2i \end{bmatrix} \quad u^{\mathrm{T}} = \begin{bmatrix} 2+3i & 1-2i & 3+2i \end{bmatrix}.$$

Аналогично определяется сопряжение и транспонирование для вектор-строки, приводящее к вектор-столбцу. Ясно, что для векторов, состоящих только из действительных чисел, операции сопряжения и транспонирования совпадают.

Для нахождения сопряженного вектора в MatLab используется апостроф, а для транспонирования следует применять точку с апострофом:

```
>> u = [2+3i; 1-2i; 3+2i];
>> v = u'
v =
   2.0000 - 3.0000i   1.0000 + 2.0000i   3.0000 - 2.0000i
>> v = u.'
v =
   2.0000 + 3.0000i   1.0000 - 2.0000i   3.0000 + 2.0000i
```

Операции .' и ' над вещественными векторами приведут к одинаковым результатам. Поэлементные вычисления с массивами используются на протяжении всей книги.

Построение таблицы значений функции

Отображение функции в виде таблицы удобно, если имеется сравнительно небольшое количество значений функции. Пусть требуется вывести в командное окно таблицу значений функции

$$y(x) = \frac{\sin^2 x}{1+\cos x} + e^{-x} \cdot \ln x$$

в точках 0.2, 0.3, 0.5, 0.8, 1.3, 1.7, 2.5. Задача решается в два этапа.

1. Создайте вектор-строку *x*, содержащую координаты заданных точек.
2. Вычислите функцию $y(x)$ от каждого элемента вектора *x* и запишите полученные значения в вектор-строку *y*. Важно только сделать это правильно! Необходимо найти значения функции для каждого из элементов вектор-строки *x*, поэтому операции в выражении для функции должны выполняться *поэлементно*, как было описано в предыдущих разделах.

```
>> x = [0.2 0.3 0.5 0.8 1.3 1.7 2.5]
x =
    0.2000    0.3000    0.5000    0.8000    1.3000    1.7000    2.5000
>> y = sin(x).^2./(1+cos(x))+exp(-x).*log(x)
y =
   -1.2978   -0.8473   -0.2980    0.2030    0.8040    1.2258    1.8764
```

Обратите внимание, что при попытке использования операций возведения в степень ^, деления / и умножения * (которые не относятся к поэлементным) выводится сообщение об ошибке уже при возведении sin(x) в квадрат:

```
>> y = sin(x)^2/(1+cos(x))+exp(-x)*log(x)
??? Error using ==> ^
Matrix must be square.
```

Дело в том, что в MatLab операции * и ^ применяются для перемножения матриц соответствующих размеров и возведения квадратной матрицы в степень, о чем написано в разделах, посвященных работе с матрицами.

Таблице можно придать более удобный для чтения вид, расположив значения функции непосредственно под значениями аргумента:

```
>> x
x =
    0.2000    0.3000    0.5000    0.8000    1.3000    1.7000    2.5000
>> y
y =
   -1.2978   -0.8473   -0.2980    0.2030    0.8040    1.2258    1.8764
```

Часто требуется вывести значение функции в точках отрезка, отстоящих друг от друга на равное расстояние (шаг). Предположим, что необходимо вывести таблицу значений функции $y(x)$ на отрезке [1, 2] с шагом 0.2. Можно, конечно, ввести вектор-строку значений аргумента x = [1, 1.2, 1.4, 1.6, 1.8, 2.0] из командной строки и вычислить все значения функции так, как описано выше. Однако, если шаг будет не 0.2, а например 0.01, то предстоит большая работа по вводу вектора x.

В MatLab предусмотрено простое создание векторов, каждый элемент которых отличается от предшествующего на постоянную величину, т. е. шаг. Для ввода таких векторов служит двоеточие (не путайте с индексацией при помощи двоеточия). Следующие два оператора приводят к формированию одинаковых вектор-строк:

```
>> x = [1, 1.2, 1.4, 1.6, 1.8, 2.0]
x =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
>> x = [1:0.2:2]
x =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

Условно можно записать

```
x = [начальное значение :  шаг : конечное значение]
```

Необязательно заботиться о том, чтобы сумма предпоследнего значения и шага равнялась бы конечному значению, например, при выполнении следующего оператора присваивания

```
>> x = [1:0.2:1.9]
x =
    1.0000    1.2000    1.4000    1.6000    1.8000
```

вектор-строка заполнится до элемента, не превосходящего определенное нами конечное значение. Шаг может быть и отрицательным:

```
>> x = [1.9:-0.2:1]
x =
    1.9000    1.7000    1.5000    1.3000    1.1000
```

В случае отрицательного шага для получения непустой вектор-строки начальное значение должно быть больше конечного.

Попробуйте самостоятельно заполнить вектор-столбец элементами, начинающимися с нуля и заканчивающимися 0.5 с шагом 0.1. Для этого следует заполнить вектор-строку, а затем использовать операцию транспонирования:

```
>> x = [0:0.1:0.5]'
x =
         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
```

Обратите внимание, что элементы вектора, заполняемого при помощи двоеточия, могут быть только вещественные, поэтому для транспонирования можно использовать апостроф вместо точки с апострофом.

Шаг, равный единице, допускается не указывать при автоматическом заполнении:

```
>> x = [1:5]
x =
     1     2     3     4     5
```

Выведите теперь таблицу значений функции

$$y(x) = e^{-x}\sin 10x$$

на отрезке [0, 1] с шагом 0.05, произведя следующие действия:

1. Сформируйте вектор-строку x при помощи двоеточия.
2. Вычислите значения $y(x)$ от элементов x (не забудьте использовать поэлементное умножение).
3. Запишите результат в вектор-строку y.
4. Выведите x и y.

Результат, отображенный на экране, не очень напоминает таблицу:

```
>> x = [0:0.05:1];
>> y = exp(-x).*sin(10*x);
```

```
>> x
x =
  Columns 1 through 7
         0    0.0500    0.1000    0.1500    0.2000    0.2500    0.3000
  Columns 8 through 14
    0.3500    0.4000    0.4500    0.5000    0.5500    0.6000    0.6500
  Columns 15 through 21
    0.7000    0.7500    0.8000    0.8500    0.9000    0.9500    1.0000
>> y
y =
  Columns 1 through 7
         0    0.4560    0.7614    0.8586    0.7445    0.4661    0.1045
  Columns 8 through 14
   -0.2472   -0.5073   -0.6233   -0.5816   -0.4071   -0.1533    0.1123
  Columns 15 through 21
    0.3262    0.4431    0.4445    0.3413    0.1676   -0.0291   -0.2001
```

Вектор-строки x и y состоят из двадцати одного элемента, не помещаются на экране в одну строку и выводятся по частям. Так как x и y хранятся в двумерных массивах размерностью один на двадцать один, то выводятся по столбцам, каждый из которых состоит из одного элемента. Сначала выводятся столбцы с первого по седьмой (`Columns 1 through 7`), затем — с восьмого по четырнадцатый (`Columns 8 through 14`), и наконец — с пятнадцатого по двадцать первый (`Columns 15 through 21`). Более наглядным и удобным является графическое представление функции.

Построение графиков функции одной переменной

MatLab обладает хорошо развитыми графическими возможностями для визуализации данных. Графике в MatLab посвящена следующая глава. В настоящем разделе описано построение простейшего графика функции одной переменной на примере функции

$$y(x) = e^{-x} \sin 10x ,$$

определенной на отрезке [0, 1]. Вывод отображения функции в виде графика состоит из следующих этапов:

1. Задание вектора значений аргумента x.
2. Вычисление вектора y значений функции $y(x)$.
3. Вызов команды `plot` для построения графика.

Команды для задания вектора x и вычисления функции лучше завершать точкой с запятой для подавления вывода в командное окно их значений (после команды `plot` точку с запятой ставить необязательно, т. к. она ничего не выводит в командное окно). Не забудьте использовать поэлементное умножение `.*`

```
>> x = [0:0.05:1];
>> y = exp(-x).*sin(10*x);
>> plot(x, y)
```

После выполнения команд на экране появляется окно **Figure No. 1** с графиком функции, изображенное на рис. 2.1. Окно содержит меню, панель инструментов и область графика. Вид графического окна в версиях 5.3 и 6.x практически одинаков. В следующей главе, посвященной графике в MatLab, описаны команды, специально предназначенные для оформления графика. Сейчас нас будет интересовать сам принцип построения графиков и некоторые простейшие возможности визуализации функций.

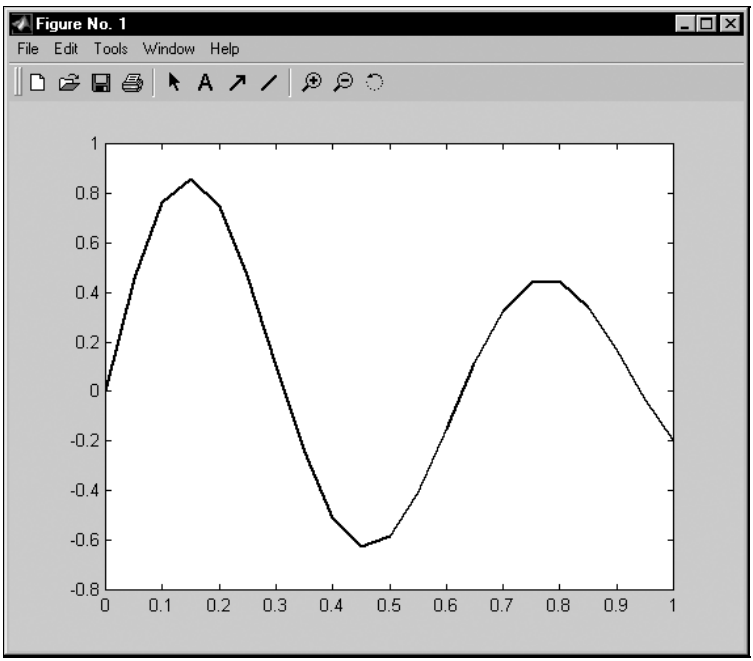


Рис. 2.1. Простейший график функции

Для построения графика функции в рабочей среде MatLab должны быть определены два вектора одинаковой размерности, например x и y. Соответствующий массив `x` содержит значения аргументов, а `y` — значения функции

от этих аргументов. Команда `plot` соединяет точки с координатами (`x(i)`, `y(i)`) прямыми линиями, автоматически масштабируя оси для оптимального расположения графика в окне. При построении графиков удобно расположить на экране основное окно MatLab и окно с графиком рядом так, чтобы они не перекрывались, как показано на рис. 2.2.

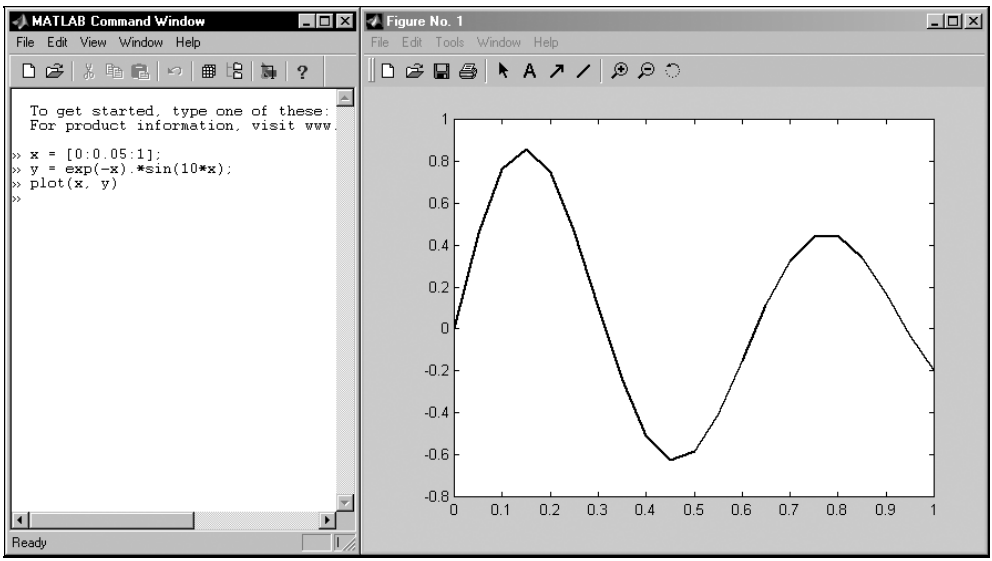


Рис. 2.2. Расположение окон

Построенный график функции не должен иметь изломов, т. к. сама функция гладкая. Для точного построения графика вычислите функцию в большем числе точек на отрезке [0, 1], т. е. задайте меньший шаг при вводе вектора *x*. Не забудьте, что можно занести в командную строку введенные ранее команды при помощи клавиш <↑>, <↓>, затем отредактировать их и выполнить, нажав <Enter>. Следующие команды:

```
>> x = [0:0.01:1];
>> y = exp(-x).*sin(10*x);
>> plot(x, y)
```

приводят к построению графика функции в виде плавной кривой, изображенной на рис. 2.3 (далее в книге почти всюду приводится только область графика, без заголовка окна, меню и панели инструментов).

Сравнение нескольких функций удобно производить, отобразив их графики на одних осях. Например, постройте на отрезке [−1, −0.3] графики функций

$$f(x) = \sin\frac{1}{x^2}, \qquad g(x) = \sin\frac{1.2}{x^2}$$

при помощи последовательности команд, приведенной ниже:

```
>> x = [-1:0.005:-0.3];
>> f = sin(x.^-2);
>> g = sin(1.2*x.^-2);
>> plot(x, f, x, g)
```

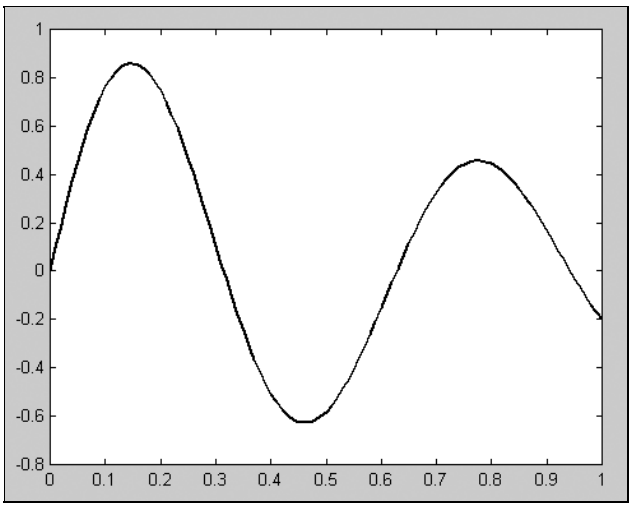


Рис. 2.3. Гладкий график функции

Получившиеся графики, приведенные на рис. 2.4, дают наглядное представление о поведении исследуемых функций.

MatLab выводит графики разным цветом. Монохромный принтер напечатает графики различными оттенками серого цвета, что не всегда удобно. Команда `plot` позволяет легко задать стиль и цвет линий, например

```
>> plot(x,f,'k-',x,g,'k:')
```

осуществляет построение первого графика сплошной черной линией, а второго — черной пунктирной (рис. 2.5). Аргументы `'k-'` и `'k:'` задают стиль и цвет первой и второй линий. Здесь `k` означает черный цвет, а дефис или двоеточие — сплошную или пунктирную линию. Визуализация данных и построение графиков подробно описаны в следующих главах. Окно с графиком можно закрыть, нажав на кнопку с крестиком в правом верхнем углу.

Построение графиков функций в MatLab требует понимания работы с векторами — необходимо уметь вводить векторы, использовать двоеточие для автоматического заполнения с заданным шагом, применять поэлементные операции для вычисления функций от вектора значений аргумента. Все эти вопросы разобраны выше. Перейдем теперь к умножению векторов.

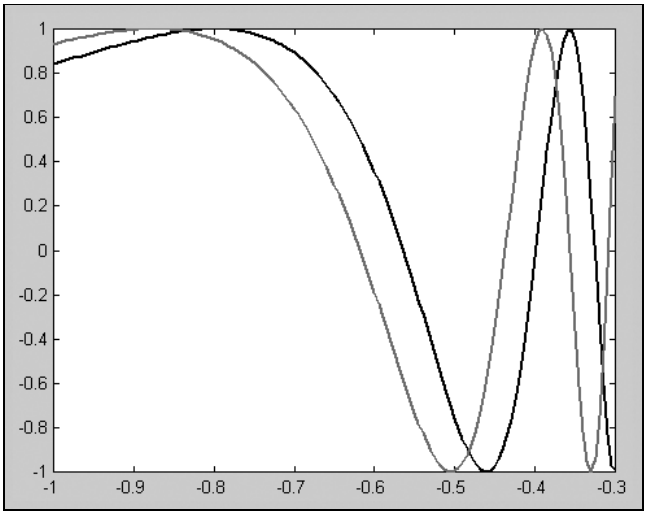


Рис. 2.4. Два графика на одних осях

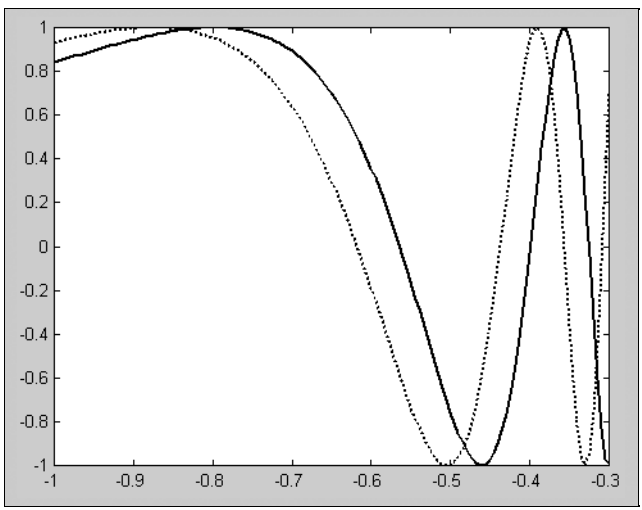


Рис. 2.5. Изменение стиля и цвета линий графиков

Умножение векторов

Вектор можно умножить на другой вектор скалярно (это произведение еще называют внутренним), векторно, или образовать так называемое внешнее произведение. Результатом скалярного произведения является число, векторного — вектор, а внешнего — матрица.

Скалярное произведение

Скалярное произведение векторов a и b длины N, состоящих из действительных чисел, определяется формулой

$$a \cdot b = \sum_{k=1}^{N} a_k b_k .$$

Следовательно, для вычисления скалярного произведения необходимо просуммировать компоненты вектора, полученного в результате поэлементного умножения a на b, т. е. надо использовать функцию `sum` и поэлементное умножение. Найдите самостоятельно скалярное произведение векторов:

$$a = \begin{bmatrix} 1.2 \\ -3.2 \\ 0.7 \end{bmatrix} \qquad b = \begin{bmatrix} 4.1 \\ 6.5 \\ -2.9 \end{bmatrix}.$$

Ниже приведена требуемая последовательность команд:

```
>> a = [1.2; -3.2; 0.7];
>> b = [4.1; 6.5; -2.9];
>> s = sum(a.*b)
s =
  -17.9100
```

Найдите длину (или, как еще говорят, модуль) вектора a

$$|a| = \sqrt{a \cdot a} .$$

Решение очевидно:

```
>> d = sqrt(sum(a.*a))
d =
    3.4886
```

Векторное произведение

Векторное произведение $a \times b$ определено только для векторов из трехмерного пространства, т. е. состоящих из трех элементов. Результатом также является вектор из трехмерного пространства. Для вычисления векторного произведения в MatLab служит функция `cross`:

```
>> a = [1.2; -3.2; 0.7];
>> b = [4.1; 6.5; -2.9];
>> c = cross(a, b)
c =
    4.7300
```

```
    6.3500
   20.9200
```

Для тренировки попробуйте вычислить $a \times b + b \times a$. Если получился вектор, состоящий из нулей, то вы все проделали правильно, т. к. для любых векторов выполняется свойство $a \times b = -b \times a$.

Смешанное произведение векторов *a*, *b*, *c* определяется по формуле $abc = a \cdot (b \times c)$. Модуль смешанного произведения векторов равен объему параллелепипеда, построенного на этих векторах так, как показано на рис. 2.6.

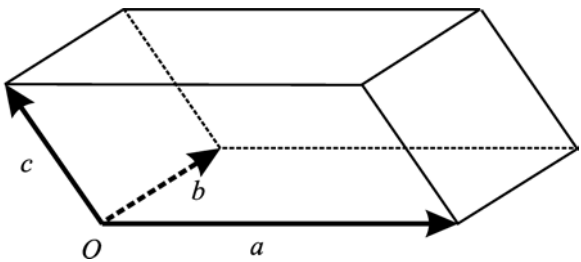


Рис. 2.6. Параллелепипед, образованный тремя векторами

Найдите объем параллелепипеда, если

$$a = \begin{bmatrix} 3.5 \\ 0 \\ 0 \end{bmatrix} \qquad b = \begin{bmatrix} 0.5 \\ 2.1 \\ 0 \end{bmatrix} \qquad c = \begin{bmatrix} -0.2 \\ -1.9 \\ 2.8 \end{bmatrix}.$$

Правильные действия таковы:

```
>> a = [3.5; 0; 0];
>> b = [0.5; 2.1; 0];
>> c = [-0.2; -1.9; 2.8];
>> V = abs(sum(a.*cross(b,c)))
V =
   20.5800
```

Внешнее произведение

Внешним произведением векторов $a = (a_j)_{j=1,\ldots,N}$, $b = (b_k)_{k=1,\ldots,M}$ называется матрица $C = (c_{jk})_{j=1,\ldots,N,\,k=1,\ldots,M}$ размера $N \times M$, элементы которой вычисляются по формуле

$$c_{jk} = a_j b_k.$$

Вектор-столбец a в MatLab представляется в виде двумерного массива размера N на один. Вектор-столбец b при транспонировании переходит в вектор-строку размера один на M. Вектор-столбец и вектор-строка есть матрицы, у которых один из размеров равен единице. Фактически, $C = ab^{\mathrm{T}}$, где умножение происходит по правилу *матричного произведения*. Для вычисления матричного произведения в MatLab используется оператор "звездочка":

```
>> a = [1; 2; 3];
>> b = [5; 6; 7];
>> C = a*b'
C =
     5     6     7
    10    12    14
    15    18    21
```

MatLab вывела в командное окно матрицу в привычном виде — по строкам. Используйте команду `whos` для просмотра переменных рабочей среды:

```
>> whos
  Name      Size          Bytes     Class
  C         3x3           72        double array
  a         3x1           24        double array
  b         3x1           24        double array
Grand total is 15 elements using 120 bytes
```

Числа, векторы и матрицы хранятся в двумерных массивах, числа — в массивах размерностью один на один, вектор-столбцы и вектор-строки содержатся в массивах, у которых одно из измерений равно единице, а для матриц выделяются двумерные массивы подходящих размеров. Именно поэтому операции и встроенные функции в MatLab приспосабливаются к виду аргументов, выдавая результат в соответствующем виде. Если вы внимательно изучили использование векторов, то читать следующие разделы о работе с матрицами не представит большого труда.

Двумерные массивы, матрицы

В этом разделе описан ввод матриц, математические операции с ними, поэлементные операции, вычисление функций от элементов матриц, чтение и запись с использованием текстового файла, простейшая визуализация матричных данных.

Ввод матриц, простейшие операции

Различные способы ввода

Вводить небольшие по размеру матрицы удобно прямо из командной строки. Введите матрицу размерностью два на три

$$A = \begin{pmatrix} 3 & 1 & -1 \\ 2 & 4 & 3 \end{pmatrix}.$$

Для хранения матрицы используйте двумерный массив с именем `A`. При вводе учтите, что матрицу A можно рассматривать как вектор-столбец из двух элементов, каждый из которых является вектор-строкой длиной три, следовательно, строки при наборе отделяются точкой с запятой:

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

Для изучения простейших операций над матрицами приведем еще несколько примеров. Рассмотрим другие способы ввода. Введите квадратную матрицу размера три так, как описано ниже:

$$B = \begin{pmatrix} 4 & 3 & -1 \\ 2 & 7 & 0 \\ -5 & 1 & 2 \end{pmatrix}.$$

Начните набирать в командной строке

```
>> B=[4 3 -1
```

Нажмите клавишу <Enter>. Обратите внимание, что MatLab ничего не вычислила. Курсор мигает на следующей строке без символа >>. Продолжите ввод матрицы построчно, нажимая в конце каждой строки <Enter>. Последнюю строку завершите закрывающей квадратной скобкой, получается:

```
2 7 0
-5 1 2]
B =
     4     3    -1
     2     7     0
    -5     1     2
```

Еще один способ ввода матриц состоит в том, что матрицу можно трактовать как вектор-строку, каждый элемент которой является вектор-столбцом.

Например, матрицу два на три

$$C = \begin{pmatrix} 3 & -1 & 7 \\ 4 & 2 & 0 \end{pmatrix}$$

можно ввести при помощи команды:

```
>> C = [[3; 4]  [-1; 2]  [7; 0]]
C =
     3    -1     7
     4     2     0
```

Посмотрите переменные рабочей среды, набрав в командной строке `whos`:

```
  A          2x3             48  double array
  B          3x3             72  double array
  C          2x3             48  double array
```

Итак, в рабочей среде содержится три матрицы, две прямоугольные и одна квадратная.

Обращение к элементам матриц

Доступ к элементам матриц осуществляется при помощи двух индексов — номеров строки и столбца, заключенных в круглые скобки, например

```
 >> C(2, 3)
ans =
      0
```

Элементы матриц могут входить в состав выражений:

```
>> C(1, 1) + C(2, 2) + C(2, 3)
ans =
      5
```

Расположение элементов матрицы в памяти компьютера определяет еще один способ обращения к ним. Матрица *A* размера `m` на `n` хранится в виде вектора длины `mn`, в котором элементы матрицы расположены один за другим построчно

```
[A(1,1) A(1,2) ... A(1,n)  ...  A(m,1) A(m,2) ... A(m,n) ].
```

Для доступа к элементам матрицы можно использовать *один индекс*, задающий порядковый номер элемента матрицы в векторе.

Матрица *C*, определенная в предыдущем разделе, содержится в векторе

```
[C(1,1) C(1,2) C(1,3) C(2,1) C(2,2) C(2,3)]
```

Индексация при помощи порядкового номера производится следующим образом:

```
>> C(1)
ans =
     3
>> C(5)
ans =
     2
```

Сложение, вычитание, умножение, транспонирование и возведение в степень

При использовании матричных операций следует помнить, что для сложения или вычитания матрицы должны быть одного размера, а при перемножении число столбцов первой матрицы обязано равняться числу строк второй матрицы. Сложение и вычитание матриц, так же как чисел и векторов, осуществляется при помощи знаков плюс и минус. Найдите сумму и разность матриц *C* и *A*, определенных выше:

```
>> S = A + C                    >> R = C-A
S =                             R =
     6     0     6                    0    -2     8
     6     6     3                    2    -2    -3
```

Следите за совпадением размерности, иначе получите сообщение об ошибке:

```
>> S = A+B
??? Error using ==> +
Matrix dimensions must agree.
```

Для умножения матриц предназначена звездочка:

```
>> P = C*B
P =
   -25     9    11
    20    26    -4
```

Умножение матрицы на число тоже осуществляется при помощи звездочки, причем умножать на число можно как справа, так и слева:

```
>> P = A*3
P =
     9     3    -3
     6    12     9
>> P = 3*A
```

```
P =
     9     3    -3
     6    12     9
```

Транспонирование матрицы, так же как и вектора, производится при помощи .', а символ ' означает комплексное сопряжение. Для вещественных матриц эти операции приводят к одинаковым результатам:

```
>> B'                          >> B.'
ans =                          ans =
     4     2    -5                   4     2    -5
     3     7     1                   3     7     1
    -1     0     2                  -1     0     2
```

Сопряжение и транспонирование матриц, содержащих комплексные числа, приведут к созданию разных матриц:

```
>> K = [1-i, 2+3i; 3-5i, 1-9i]
K =
   1.0000 - 1.0000i   2.0000 + 3.0000i
   3.0000 - 5.0000i   1.0000 - 9.0000i
>> K'
ans =
   1.0000 + 1.0000i   3.0000 + 5.0000i
   2.0000 - 3.0000i   1.0000 + 9.0000i
>> K.'
ans =
   1.0000 - 1.0000i   3.0000 - 5.0000i
   2.0000 + 3.0000i   1.0000 - 9.0000i
```

Вспомните, что при вводе вектор-строк их элементы можно разделять или пробелами, или запятыми. При вводе матрицы *K* применены запятые для более наглядного разделения комплексных чисел в строке.

Возведение квадратной матрицы в целую степень производится с использованием оператора ^:

```
>> B2 = B^2
B2 =
    27    32    -6
    22    55    -2
   -28    -6     9
```

Проверьте полученный результат, умножив матрицу саму на себя.

Убедитесь, что вы освоили простейшие операции с матрицами в MatLab. Найдите значение следующего выражения

$$(A+C)B^3(A-C)^{\mathrm{T}}.$$

Учтите *приоритет* операций, сначала выполняется транспонирование, потом возведение в степень, затем умножение, а сложение и вычитание производятся в последнюю очередь.

```
>> (A+C)*B^3*(A-C)'
ans =
        1848        1914
       10290        3612
```

Перемножение матрицы и вектора

Поскольку вектор-столбец или вектор-строка в MatLab являются матрицами, у которых один из размеров равен единице, то все вышеописанные операции применимы и для умножения матрицы на вектор, или вектор-строки на матрицу. Например, вычисление выражения

$$\begin{bmatrix}1 & 3 & -2\end{bmatrix}\begin{pmatrix}2 & 0 & 1\\ -4 & 8 & -1\\ 0 & 9 & 2\end{pmatrix}\begin{bmatrix}-8\\ 3\\ 4\end{bmatrix}$$

можно осуществить следующим образом:

```
>> a = [1 3 -2];
>> B = [2 0 1; -4 8 -1; 0 9 2];
>> c = [-8;3;4];
>> a*B*c
ans =
    74
```

В математике ничего не говорится про деление матриц и векторов, однако в MatLab символ \ используется для решения систем линейных уравнений.

Решение систем линейных уравнений

Решите небольшую систему, состоящую из трех уравнений с тремя неизвестными:

$$\begin{cases}1.2x_1+0.3x_2-0.2x_3=1.3;\\ 0.5x_1+2.1x_2+1.3x_3=3.9;\\ -0.9x_1+0.7x_2+5.6x_3=5.4.\end{cases}$$

Введите матрицу системы в массив A, для вектора правой части используйте массив b. Решите систему при помощи символ \

```
>> x = A\b
x =
    1.0000
    1.0000
    1.0000
```

Проверьте, правильный ли получился ответ, умножив *A* на *x*.

Замечание

Алгоритм решения систем линейных уравнений при помощи оператора \ в MatLab определяется структурой матрицы коэффициентов системы. В частности, MatLab исследует, является ли матрица треугольной, или может быть приведена перестановками строк и столбцов к треугольному виду, симметричная матрица или нет, квадратная или прямоугольная (MatLab умеет решать системы с прямоугольными матрицами — переопределенные или недоопределенные). Поэтому решать системы при помощи \ разумно, когда выбор алгоритма решения поручается MatLab. Если же имеется информация о свойствах матрицы системы, то разумно использовать специальные методы.

Решение систем небольшой размерности можно выполнить, введя матрицу системы и вектор правой части непосредственно из командной строки. Однако часто требуется найти решение системы, состоящей из большого числа линейных уравнений, причем матрица и вектор правой части системы хранятся в файлах. В следующем разделе на примере решения системы показано, как считать данные из текстового файла, получить результат и записать его в файл.

Считывание и запись данных

Перед нами стоит задача — решить систему линейных уравнений, матрица и вектор правой части которой хранятся в текстовых файлах matr.txt, rside.txt, и записать результат в файл sol.txt. Матрица записана в файле построчно, элементы в строке отделены пробелом, вектор правой части записан в столбик, как показано на рис. 2.7.

Подготовьте файлы с данными, например в стандартной программе Windows Блокнот (NotePad). Скопируйте файлы matr.txt, rside.txt в подкаталог work основного каталога MatLab. Для считывания из файла используйте команду load, для записи — save. Примение load и save для сохранения и считывания переменных рабочей среды описано в *главе 1*, однако при работе с файлами данных используется другой формат вызова этих команд с выходными аргументами:

```
>> A = load('matr.txt');
>> b = load('rside.txt');
```

```
>> x = A\b;
>> save 'sol.txt' x -ascii
```

файл matr.txt				файл rside.txt
3.45	0.11	⋯	-0.25	7.25
1.08	5.97	⋯	0.09	0.91
⋮		⋱	⋮	⋮
−0.41	1.06		5.84	4.23

Рис. 2.7. Файлы с матрицей и вектором правой части системы

Параметр `-ascii` означает запись в текстовом формате. После выполнения этих команд в каталоге work создастся файл sol.txt, в котором в столбик будет записано решение системы. Посмотреть содержимое файла можно, используя любой текстовый редактор. Для записи результата в файл с двойной точностью следует использовать команду `save 'sol.txt' x -ascii -double`. На рис. 2.8 приведено содержимое файла sol.txt в случае использования команды `save` с параметром `-double` и без него.

3.5756116e+000	3.5756115823410211e+000
-1.4288319e+000	-1.4288318561740618e+000
⋮	⋮
2.8680803e+000	2.8680803204263228e+000
save 'sol.txt' x -ascii	save 'sol.txt' x -ascii -double

Рис. 2.8. Содержимое файла sol.txt

Аналогично можно записать матрицу в текстовый файл. Запись, например матрицы A, хранящейся в массиве `A`, в файл matrA.txt осуществляется командой `save 'sol.txt' A -ascii`.

Блочные матрицы

Очень часто в приложениях возникают так называемые блочные матрицы, т. е. матрицы, составленные из непересекающихся подматриц (блоков). Соответствующие размеры блоков должны совпадать.

Конструирование блочных матриц

Введите матрицы

$$A=\begin{pmatrix}-1 & 4\\ -1 & 4\end{pmatrix} \qquad B=\begin{pmatrix}2 & 0\\ 0 & 5\end{pmatrix} \qquad C=\begin{pmatrix}3 & -3\\ -3 & 3\end{pmatrix} \qquad D=\begin{pmatrix}8 & 9\\ 1 & 10\end{pmatrix}$$

и создайте из них блочную матрицу

$$K = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right),$$

учитывая, что матрица K состоит из двух строк, в первой строке матрицы A и B, а во второй — C и D:

```
>> K = [A B; C D]
K =
    -1     4     2     0
    -1     4     0     5
     3    -3     8     9
    -3     3     1    10
```

Блочная матрица получена. Можно было поступить и по-другому, а именно, считать, что матрица K состоит из двух столбцов, в первом — матрицы A и C, а во втором — B и D. Как бы тогда следовало записать команду для создания блочной матрицы? Проверьте себя

```
>> K = [[A; C]   [B; D]]
```

Вот еще один хороший пример для проверки знаний о работе с массивами в MatLab. Требуется составить блочную матрицу

$$M = \left(\begin{array}{c|c} S & a \\ \hline b & 2.5 \end{array}\right),$$

где

$$S = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \qquad a = \begin{pmatrix} 4 \\ 5 \end{pmatrix} \qquad b = \begin{pmatrix} -9 & 9 \end{pmatrix}.$$

Решение этой задачи следующее:

```
>> S = [2 0; 0 3];
>> a = [4; 5];
>> b = [-9 9];
>> M = [S a; b 2.5]
M =
    2.0000         0    4.0000
         0    3.0000    5.0000
   -9.0000    9.0000    2.5000
```

Последний оператор можно заменить на эквивалентный `M = [[S; b] [a; 2.5]]`. Обратной задачей к конструированию блочных матриц является выделение блоков.

Выделение блоков

Выделение блоков матриц осуществляется индексацией при помощи двоеточия, которая уже использовалась в предыдущих разделах для выделения блоков из векторов. Введите матрицу

$$P = \begin{pmatrix} 1 & 2 & 0 & 2 \\ 4 & 10 & 12 & 5 \\ 0 & 11 & 10 & 5 \\ 9 & 2 & 3 & 5 \end{pmatrix},$$

затем выделите очерченный блок, задав номера строк и столбцов при помощи двоеточия:

```
>> P1 = P(2:3,2:3)
P1 =
    10    12
    11    10
```

Для выделения из матрицы столбца или строки (то есть массива, у которого один из размеров равен единице) следует в качестве одного из индексов использовать номер столбца или строки матрицы, а другой индекс заменить на двоеточие без указания пределов. Например, запишите вторую строку P в вектор p

```
>> p = P(2, :)
p =
     4    10    12     5
```

При выделении блока до конца матрицы можно не указывать ее размеры, а использовать элемент `end`:

```
>> p = P(2, 2:end)
p =
     10    12     5
```

Удаление строк и столбцов

В MatLab парные квадратные скобки `[]` обозначают пустой массив, который, в частности, позволяет удалять строки и столбцы матрицы. Для удаления строки следует присвоить ей пустой массив. Удалите, например, первую строку квадратной матрицы:

```
>> M = [2 0 3
       1 1 4
       6 1 3];
>> M(1,:) = [];
```

```
>> M
M =
     1     1     4
     6     1     3
```

Обратите внимание на соответствующее изменение размеров массива, которое можно проверить при помощи `size`:

```
>> size(M)
ans =
     2     3
```

Аналогичным образом удаляются и столбцы. Для удаления нескольких идущих подряд столбцов (или строк) им нужно присвоить пустой массив. Удалите второй и третий столбец в массиве `M`

```
>> M(:, 2:3) = []
M =
     1
     6
```

Индексация существенно экономит время при вводе матриц, имеющих определенную структуру.

Заполнение матриц при помощи индексации

Выше было описано несколько способов ввода матриц в MatLab, в том числе и считывание из файла. Однако бывает проще сгенерировать матрицу, чем вводить ее, особенно если она обладает простой структурой. Рассмотрим пример такой матрицы:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \end{pmatrix}.$$

Генерация матрицы T осуществляется в три этапа:

1. Создание массива `T` размера пять на пять, состоящего из нулей.
2. Заполнение первой строки единицами.
3. Заполнение части последней строки минус единицами до последнего элемента.

Соответствующие команды MatLab приведены ниже (они записаны в три колонки с целью экономии места, но набирать их надо последовательно).

Команды не завершаются точкой с запятой для вывода промежуточных результатов в командное окно с целью слежения за процессом формирования матрицы

```
>> A(1:5, 1:5) = 0
A =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

>> A(1, :) = 1
A =
     1     1     1     1     1
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

>> A(end, 3:end) = -1
A =
     1     1     1     1     1
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0    -1    -1    -1
```

Создание некоторых специальных матриц в MatLab осуществляется при помощи встроенных функций.

Создание матриц специального вида

Заполнение прямоугольной матрицы нулями производится встроенной функцией `zeros`, аргументами которой являются число строк и столбцов матрицы:

```
>> A = zeros(3, 6)
A =
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Один аргумент функции `zeros` приводит к образованию квадратной матрицы заданного размера:

```
>> A = zeros(3)
A =
     0     0     0
     0     0     0
     0     0     0
```

Единичная матрица инициализируется при помощи функции `eye`:

```
>> I = eye(4)
I =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Функция `eye` с двумя аргументами создает прямоугольную матрицу, у которой на главной диагонали стоят единицы, а остальные элементы равны нулю:

```
>> I = eye(4, 8)
I =
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
```

Матрица, состоящая из единиц, образуется в результате вызова функции `ones`:

```
>> E = ones(3, 8)
E =
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
```

Использование одного аргумента в `ones` приводит к созданию квадратной матрицы, состоящей из единиц.

MatLab предоставляет возможность заполнения матриц случайными элементами. Результатом функции `rand` является матрица чисел, распределенных случайным образом между нулем и единицей, а функции `randn` — матрица чисел, распределенных по нормальному закону:

```
>> R = rand(3, 5)
R =
    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
```

Один аргумент функций `rand` и `randn` приводит к формированию квадратных матриц:

```
>> RN = randn(3, 5)
RN =
    0.1139   -0.0956   -1.3362   -0.6918   -1.5937
    1.0668   -0.8323    0.7143    0.8580   -1.4410
    0.0593    0.2944    1.6236    1.2540    0.5711
```

Вопрос об автоматическом заполнении векторов или вектор-строк не должен поставить нас в тупик, поскольку мы знаем, что вектор или вектор-строка в MatLab являются матрицей, у которой один из размеров равен единице. Заполните вектор-строку шестью случайными числами.

Проверьте себя, выполнив следующий пример:

```
>> r = rand(1, 6)
r =
    0.7468    0.4451    0.9318    0.4660    0.4186    0.8462
```

Часто возникает необходимость создания диагональных матриц, т. е. матриц, у которых все внедиагональные элементы равны нулю. Функция `diag` формирует диагональную матрицу из вектора или вектор-строки, располагая их элементы по диагонали матрицы:

```
>> d = [1; 2; 3; 4];
>> D = diag(d)
D =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
```

Для заполнения не главной, а побочной диагонали предусмотрена возможность вызова функции `diag` с двумя аргументами. В этом случае второй аргумент означает, насколько побочная диагональ отстоит от главной, а его знак указывает на направление, плюс — вверх, минус — вниз от главной диагонали:

```
>> d = [1; 2];
>> D = diag(d, 2)
D =
     0     0     1     0
     0     0     0     2
     0     0     0     0
     0     0     0     0
>> D = diag(d, -2)
D =
     0     0     0     0
     0     0     0     0
     1     0     0     0
     0     2     0     0
```

Функция `diag` служит и для выделения диагонали матрицы в вектор, например

```
>> A = [10 1 2; 1 20 3; 2 3 30]
A =
    10     1     2
```

```
     1    20     3
     2     3    30
>> d = diag(A)
d =
    10
    20
    30
```

Проделайте следующее упражнение для того, чтобы убедиться, что вы освоили автоматическое заполнение матриц, работу с блочными матрицами и запись данных в файл. Заполните и запишите в файлы матрицы:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & -4 & 0 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & -4 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & -4 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & 0 & -4 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}, \qquad G = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}.$$

Матрицу M можно представить в виде блочной матрицы из четырех квадратных блоков размера пять, диагональные блоки формируются при помощи функции `eye`, а внедиагональные — с использованием `ones`:

```
>> M = [eye(5) 2*ones(5); 3*ones(5) -4*eye(5)];
>> save 'M.txt' M -ascii
```

Квадратная матрица G размера семь является трехдиагональной. Заполняя эту матрицу, можно использовать то обстоятельство, что G является суммой диагональной матрицы и двух матриц с шестью ненулевыми элементами над и под главной диагональю.

```
>> G = 2*eye(7)+diag(ones(1,6),1)+diag(ones(1,6),-1)
>> save 'G.txt' G -ascii
```

Посмотрите содержание полученных файлов (они должны находиться в подкаталоге work основного каталога MatLab). При больших размерах матриц бывает удобно получить ее представление в наглядном виде. В MatLab это можно проделать при помощи визуализации матриц. В следующем разделе разобраны простейшие способы визуализации матричных данных.

Визуализация матриц

Матрицы с достаточно большим количеством нулей называются *разреженными*. Часто необходимо знать, где расположены ненулевые элементы, т. е. получить так называемый *шаблон* матрицы. Для этого в MatLab служит функция `spy`. Посмотрим шаблон матрицы *G*, определенной в предыдущем разделе

```
>> spy(G)
```

После выполнения команды `spy` на экране появляется графическое окно **Figure No. 1**. На рис. 2.9 изображена часть окна без заголовка, меню и панели инструментов.

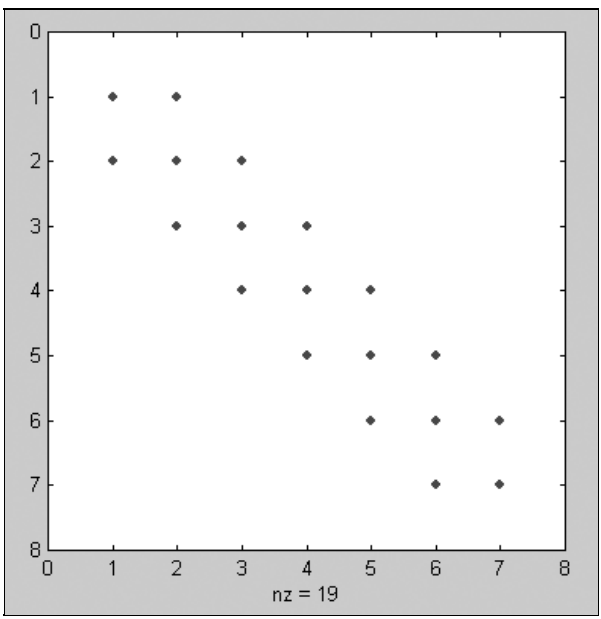


Рис. 2.9. Шаблон разреженной матрицы

На вертикальной и горизонтальной осях отложены номера строк и столбцов. Ненулевые элементы обозначены маркерами, внизу графического окна указано число ненулевых элементов (`nz=19`).

Наглядную информацию о соотношении величин элементов матрицы дает функция `imagesc`, которая интерпретирует матрицу как прямоугольное изображение. Каждый элемент матрицы представляется в виде квадратика, цвет которого соответствует величине элемента. Для того чтобы узнать соответствие цвета и величины элемента следует использовать команду `colorbar`, вы-

водящую рядом с изображением матрицы шкалу цвета. Наконец, для печати на монохромном принтере удобно получить изображение в оттенках серого цвета, используя команду `colormap(gray)`. Мы будем работать с матрицей *M*, определенной в предыдущем разделе. Набирайте команды, указанные ниже, и следите за состоянием графического окна:

```
>> imagesc(M)
>> colorbar
>> colormap(gray)
```

В результате получается наглядное представление матрицы, приведенное на рис. 2.10.

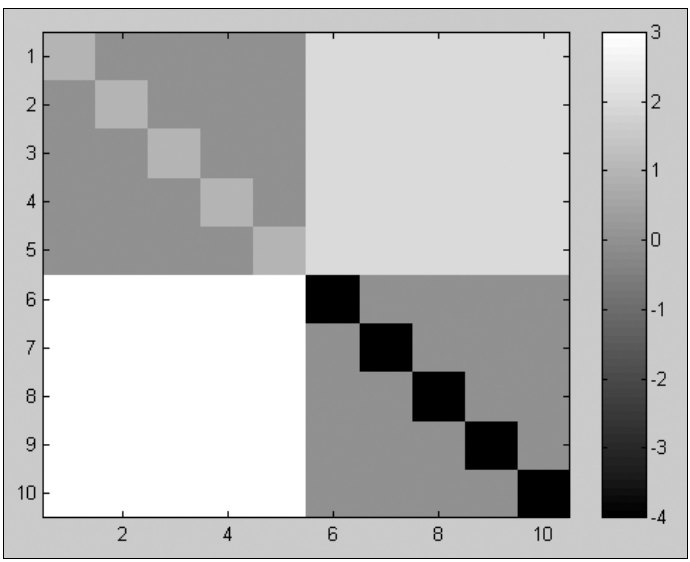


Рис. 2.10. Интерпретация матрицы как изображения

Подробно про использование графики в MatLab написано в следующих главах, однако уже сейчас вы можете визуализировать матрицы, с которыми мы будем работать при изучении поэлементных матричных операций.

Поэлементные операции и встроенные функции

Поскольку векторы и матрицы хранятся в двумерных массивах, то применение математических функций к матрицам и поэлементные операции производятся так же, как для векторов. Работа со встроенными функциями (такими, как `min`, `max`, `sum` и т. д.) имеет свои особенности в применении к матрицам.

Поэлементные операции с матрицами

Введите две матрицы

$$A = \begin{pmatrix} 2 & 5 & -1 \\ 3 & 4 & 9 \end{pmatrix} \qquad B = \begin{pmatrix} -1 & 2 & 8 \\ 7 & -3 & -5 \end{pmatrix}.$$

Умножение каждого элемента одной матрицы на соответствующий элемент другой производится при помощи оператора `.*`

```
>> C = A.*B
C =
    -2    10    -8
    21   -12   -45
```

Для деления элементов первой матрицы на соответствующие элементы второй используется `./`, а для деления элементов второй матрицы на соответствующие элементы первой служит `.\`

```
>> R1 = A./B
R1 =
   -2.0000    2.5000   -0.1250
    0.4286   -1.3333   -1.8000
>> R2 = A.\B
R2 =
   -0.5000    0.4000   -8.0000
    2.3333   -0.7500   -0.5556
```

Поэлементное возведение в степень осуществляется при помощи оператора `.^`

```
>> P = A.^2
P =
     4    25     1
     9    16    81
```

Показатель степени может быть матрицей того же размера, что и матрица, возводимая в степень. При этом элементы первой матрицы возводятся в степени, равные элементам второй матрицы:

```
>> PB = A.^B
PB =
  1.0e+003 *
    0.0005    0.0250    0.0010
    2.1870    0.0000    0.0000
```

Обратите внимание на форму вывода результата. Во-первых, выделен общий множитель 1.0e+003 для всех элементов результирующей матрицы, т. е. ответ выглядит так:

$$PB = 10^3 \cdot \begin{pmatrix} 0.0005 & 0.0250 & 0.0010 \\ 2.1870 & 0.0000 & 0.0000 \end{pmatrix} \equiv \begin{pmatrix} 0.5 & 25 & 1 \\ 2187 & 0 & 0 \end{pmatrix}.$$

Во-вторых, при возведении 4^{-3} и 9^{-5} получились нули, т. е. произошла потеря точности из-за того, что использовался формат `short`. Дело в том, что $4^{-3} \approx 0.0156$, $9^{-5} \approx 0.000016935$, а эти числа малы, по сравнению с остальными, и допустимого числа знаков формата `short` не хватает для их отображения на экране. Если мы хотим получить более точный результат поэлементного возведения в степень, то мы должны задать формат `long`, и затем вывести *PB* снова:

```
>> format long
>> PB
PB =
  1.0e+003 *
   0.00050000000000   0.02500000000000   0.00100000000000
   2.18700000000000   0.00001562500000   0.00000001693509
```

Обратите внимание, что повторного вычисления элементов матрицы *PB* не потребовалось. Вне зависимости от установленного формата вывода все вычисления производятся с двойной точностью. Перейдем теперь к вычислению математических функций от элементов матриц.

Вычисление математических функций от элементов матриц

Очень важно сразу понять, что в книгах по теории матриц формула $\cos A$, где A — квадратная матрица, означает вычисление косинуса от матрицы, которое осуществляется при помощи разложения в ряд. В MatLab имеется возможность вычисления функций от матриц.

Вычисление функций от матриц описано в разд. "Собственные числа и векторы матрицы, функции матриц" главы 6.

Запись `C = cos(A)` в MatLab приводит к вычислению косинусов от элементов массива `A` и записи их в массив `C` того же размера, что и `A`.

Нахождение, например косинусов от элементов матрицы

$$A = \begin{pmatrix} \pi/2 & -\pi/2 & 0 \\ \pi & -\pi & 2\pi \\ 0 & 2\pi & \pi/3 \end{pmatrix}$$

производится при помощи следующих команд

```
>> A = [pi/2 -pi/2 0;   pi -pi 2*pi;   0 2*pi pi/3];
>> C = cos(A)
C =
    0.0000    0.0000    1.0000
   -1.0000   -1.0000    1.0000
    1.0000    1.0000    0.5000
```

Аналогично вычисляются и другие математические функции. Использование функций обработки данных для матриц (нахождение максимума, минимума, суммы и др.) несколько отличается от их применения при работе с векторами.

Применение функций обработки данных к матрицам

Функция `sum` вычисляет сумму элементов вектора. С другой стороны, вектора в MatLab, так же как и матрицы, хранятся в двумерных массивах. Возникает вопрос: что же будет, если в качестве аргумента `sum` использовать не вектор, а матрицу. Оказывается, MatLab вычислит вектор-строку, длина которой равна числу столбцов матрицы, а каждый элемент является суммой соответствующего столбца матрицы, например:

```
>> M = [1 -2 -4
        3 -6  4
        2 -2  0];
>> s = sum(M)
s =
     6   -10     0
```

Функция `sum` по умолчанию вычисляет сумму по столбцам, изменяя первый индекс массива при фиксированном втором. Для того чтобы производить суммирование по строкам, необходимо вызвать функцию `sum` с двумя аргументами, указав место индекса, по которому следует суммировать:

```
>> s2 = sum(M, 2)
s2 =
    -5
     1
     0
```

Заметьте, что `sum(M)` и `sum(M, 1)` приводят к одинаковым результатам. Итак, функция `sum` суммирует или по строкам, или по столбцам, выдавая

результат в виде вектора или вектор-строки. Аналогично работает и функция `prod`:

```
>> p = prod(M)
p =
     6   -24     0
```

```
>> p2 = prod(M, 2)
p2 =
     8
   -72
     0
```

Функция `sort` упорядочивает элементы каждого из столбцов матрицы в порядке возрастания. Вызов `sort` со вторым аргументом, равным двум, приводит к упорядочению элементов строк:

```
>> MC = sort(M)
MC =
     1    -6    -4
     2    -2     0
     3    -2     4
```

```
>> MR = sort(M, 2)
MR =
    -4    -2     1
    -6     3     4
    -2     0     2
```

Так же как и для векторов, функция `sort` позволяет получить матрицу индексов соответствия элементов исходной и упорядоченной матриц. Для этого необходимо вызвать `sort` с двумя выходными аргументами:

```
>> [MC, Ind] = sort(M)
MC =
     1    -6    -4
     2    -2     0
     3    -2     4
Ind =
     1     2     1
     3     1     3
     2     3     2
```

Матрицы *M*, *MC*, и *Ind* связаны между собой так `MC(i,j) = M(Ind(i,j),j)`, где `i` и `j` изменяются от одного до трех.

Функции `max` и `min` вычисляют вектор-строку, содержащую максимальные или минимальные элементы в соответствующих столбцах матрицы:

```
>> mx = max(M)
mx =
     3    -2     4
```

```
>> mn = min(M)
mn =
     1    -6    -4
```

Для того чтобы узнать не только значения максимальных или минимальных элементов, но и их номера в столбцах, следует вызвать `max` или `min` так:

```
>> [mx, k] = max(M)
mx =
     3    -2     4
```

```
>> [mn, n] = min(M)
mn =
     1    -6    -4
```

```
k =                                    n =
     2     1     2                         1     2     1
```

Обратите внимание, что во втором столбце матрицы M два равных максимальных элемента — первый и третий. Всегда возвращается номер первого максимального элемента (второй элемент в вектор-строке равен единице). Точно так же работает и min в случае двух равных минимальных элементов в столбце матрицы. Функции max и min позволяют выделить максимальные или минимальные элементы из двух матриц одинаковых размеров и записать результат в новую матрицу того же размера, что и исходные:

```
>> P = [4 3 -1
       2 0 7];
>> Q = [10 0 11
       -5 3 22]
>> R = max(P, Q)
R =
    10     3    11
     2     3    22
```

Одним из аргументов может быть число. В результирующую матрицу записывается максимум из этого числа и соответствующего элемента исходной матрицы:

```
>> S = max(P, 1)
S =
     4     3     1
     2     1     7
```

Если оба аргумента функций min или max являются числами, то возвращается минимальное или максимальное из этих чисел.

Для нахождения максимума или минимума не по столбцам матрицы, а по строкам предусмотрена следующая форма вызова со вторым аргументом — пустым массивом:

```
>> mx = max(S, [], 2)
mx =
     4
     7
```

Для того чтобы дополнительно получить номера максимальных элементов в строках, используем вызов max с двумя выходными аргументами:

```
>> [mx, j] = max(S, [], 2)
mx =
     4
     7
```

```
j =
     1
     3
```

Более подробно про обработку матричных данных можно узнать из *приложения 1* или вывести список всех встроенные функции обработки данных командой `help datafun`, а затем посмотреть информацию о нужной функции, например `help max`.

Проделайте следующее упражнение, для проверки знаний о применении к матрицам математических функций и функций обработки данных. Задана квадратная матрица A размера n с элементами a_{ik}. Требуется вычислить приведенные ниже величины (нормы матрицы):

$$p = \max_{1\le i\le n}\sum_{k=1}^{n}|a_{ik}| \qquad q = \max_{1\le k\le n}\sum_{i=1}^{n}|a_{ik}| \qquad r = \left(\sum_{i,k=1}^{n}a_{ik}^2\right)^{1/2} \qquad s = \sum_{i,k=1}^{n}|a_{ik}|.$$

Создайте квадратную матрицу размера четыре, состоящую из случайных целых чисел от нуля до десяти, вычтите из нее матрицу, состоящую из пятерок, и введите необходимые выражения для вычисления норм полученной матрицы. Проверьте себя, выполнив следующие действия:

```
>> A = round(10*rand(4))-5*ones(4)
A =
     3     3    -1     3
     2    -4    -2     5
     0     1     4     5
     1    -4    -5     3
>> p = max(sum(abs(A)))
p =
    16
>> q = max(sum(abs(A),2))
q =
    13
>> q = max(sum(abs(A')))
q =
    13
>> r = sqrt(sum(sum(abs(A).^2)))
r =
   13.0384
>> s = sum(sum(abs(A)))
s =
    46
```

В заключение этой главы описан принцип использования матричных данных при построении графиков функций двух переменных в MatLab.

Графики функций двух переменных

Построение графика функции двух переменных в MatLab на прямоугольной области определения переменных включает два предварительных этапа:

1. Разбиение области определения прямоугольной сеткой.
2. Вычисление значений функции в точках пересечения линий сетки и запись их в матрицу.

Построим график функции $z(x, y) = x^2 + y^2$ на области определения в виде квадрата $x \in [0, 1]$, $y \in [0, 1]$. Необходимо разбить квадрат равномерной сеткой (например, с шагом 0.2) так, как показано на рис. 2.11, и вычислить значения функций в узлах, обозначенных точками.

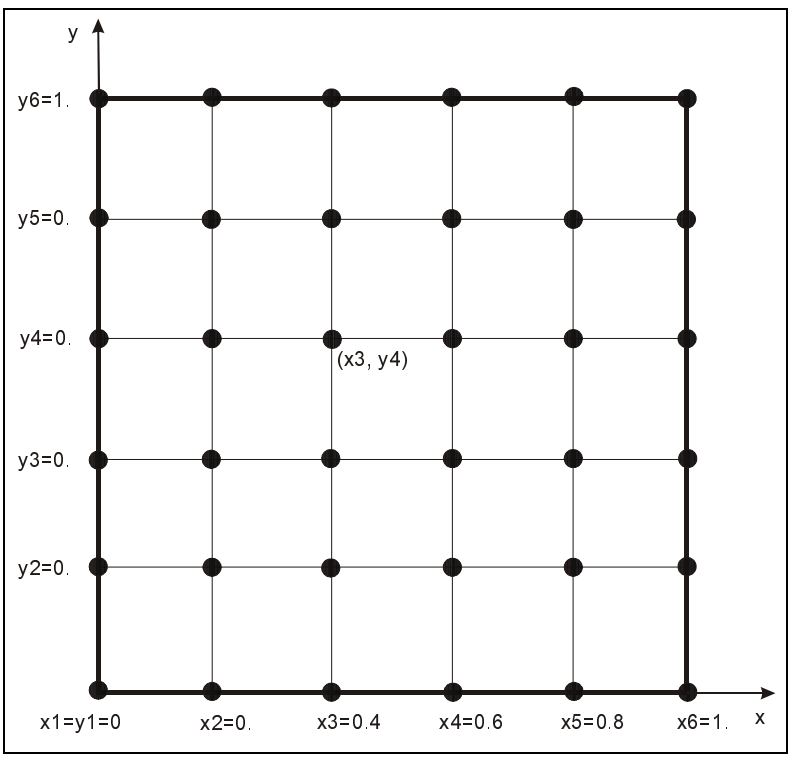


Рис. 2.11. Прямоугольная область построения графика

Удобно использовать два двумерных массива X и Y размерностью шесть на шесть для хранения информации о координатах узлов. Массив X состоит из одинаковых строк, в которых записаны координаты x1, x2,..., x6, а Y содержит одинаковые столбцы с y1, y2,..., y6. Значения функции в узлах сетки запишем в матрицу Z такой же размерности (6×6), причем для вычисления матрицы Z используем выражение для функции, но с *поэлементными* матричными операциями. Тогда, например Z(3,4) как раз будет равно значению функции $z(x, y)$ в точке (x3, y4). Для генерации массивов сетки X и Y по координатам узлов в MatLab предусмотрена функция meshgrid, для построения графика в виде каркасной поверхности — функция mesh. Следующие операторы приводят к появлению на экране окна с графиком функции, часть которого изображена на рис. 2.12 (точка с запятой в конце операторов не ставится для того, чтобы проконтролировать генерацию массивов):

```
>> [X, Y] = meshgrid(0:0.2:1,0:0.2:1)
X =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
Y =
         0         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
>> Z = X.^2 + Y.^2
Z =
         0    0.0400    0.1600    0.3600    0.6400    1.0000
    0.0400    0.0800    0.2000    0.4000    0.6800    1.0400
    0.1600    0.2000    0.3200    0.5200    0.8000    1.1600
    0.3600    0.4000    0.5200    0.7200    1.0000    1.3600
    0.6400    0.6800    0.8000    1.0000    1.2800    1.6400
    1.0000    1.0400    1.1600    1.3600    1.6400    2.0000
>> mesh(X, Y, Z)
```

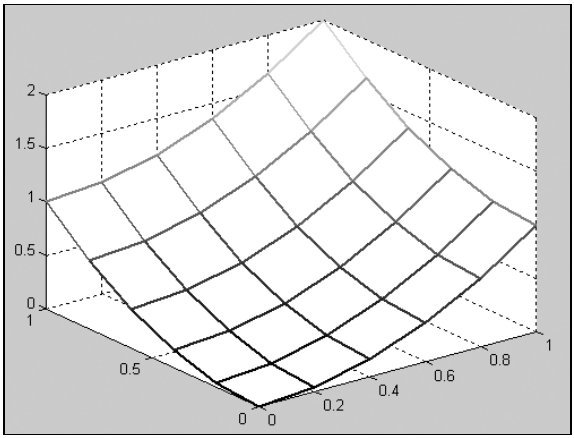


Рис. 2.12. График функции *z(x,y)*

График функции, изображенный на рис. 2.12, получился достаточно грубым из-за редкой сетки, покрывающей область изменения аргументов. Для более точного построения следует выбрать меньший шаг сетки:

```
>> [X, Y] = meshgrid(0:0.05:1, 0:0.05:1);
>> Z = X.^2 + Y.^2;
>> mesh(X, Y, Z)
```

Соответствующий график приведен на рис. 2.13.

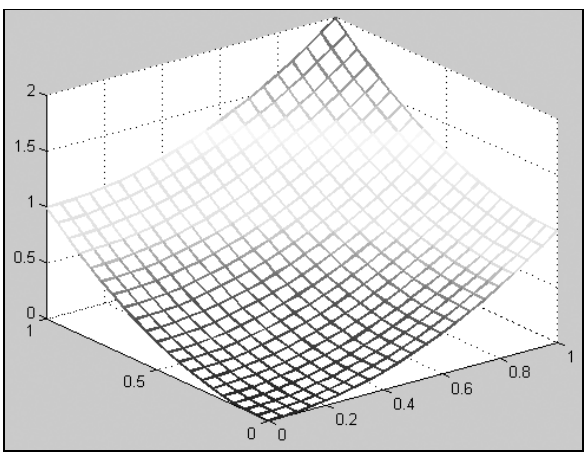


Рис. 2.13. График функции *z(x,y)* на более мелкой сетке

Следующая глава посвящена подробному описанию графических возможностей, предоставляемых MatLab для визуализации данных.

Глава 3



Высокоуровневая графика

В данной главе описаны основные возможности высокоуровневой (high-level) графики MatLab для отображения функций двух и трех переменных и визуализации векторных и матричных данных. Высокоуровневая графика позволяет пользователю получать результаты в графическом виде, прикладывая минимум усилий. Работу, связанную с масштабированием осей и подбором цветов, MatLab берет на себя. Для чтения этой главы необходимо уметь работать с массивами — векторами и матрицами, понимать, как происходят поэлементные операции с массивами. Все эти необходимые сведения были изложены в *главе 2*.

Диаграммы и гистограммы

Наглядным способом представления векторных и матричных данных являются диаграммы и гистограммы. Значение элемента вектора пропорционально высоте столбика диаграммы (в случае столбчатой диаграммы) или площади сектора диаграммы (для круговой диаграммы). Гистограммы используются для получения информации о распределении данных по заданным интервалам.

Представление векторных данных

Диаграммы векторных данных

Отображение вектора в виде столбчатой диаграммы осуществляется функцией `bar`. Запишите, например вектор-строку

$$x = \begin{bmatrix} 1.2 & 1.7 & 2.2 & 2.4 & 2.5 & 1.3 & 1.1 & 0.5 & 0.4 & 0.1 \end{bmatrix}$$

в переменную `data` и представьте ее столбчатой диаграммой, вызвав функцию `bar` с аргументом `x`:

```
>> data = [1.2 1.7 2.2 2.4 2.5 1.3 1.1 0.5 0.4 0.1];
>> bar(data)
```

На экране появится графическое окно, изображенное на рис. 3.1, содержащее столбчатую диаграмму вектор-строки (далее в книге на рисунках приво-

дится только область построения графика, а заголовок окна, меню и панель инструментов опускаются, в версиях 5.3 и 6.0 эти области выглядят одинаково). По горизонтальной оси откладывается номер элемента вектора, а по вертикальной — его значение. Аргументом функции `bar` может быть как вектор-строка, так и вектор-столбец.

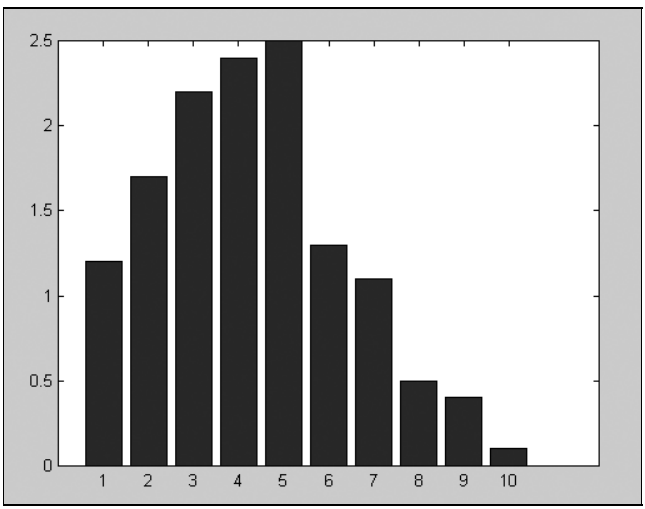


Рис. 3.1. Столбчатая диаграмма вектора

Разметку горизонтальной оси можно задать вектором с возрастающими значениями. В этом случае первый аргумент `bar` является вектором с координатами точек разметки, а второй — вектором значений, которые требуется отобразить на диаграмме. Удобно использовать этот способ построения диаграммы для отображения значений элементов векторов в зависимости не от их номера, а например от времени, если в вектор записаны результаты измерений в некоторые моменты времени. Важно, чтобы длины этих векторов совпадали, иначе будет выдано сообщение об ошибке.

```
>> time = [0.0 0.1 0.2 0.4 0.5 0.8 1.1 1.3];
>> data = [2.85 2.93 2.99 3.26 3.01 2.25 2.09 1.79];
>> bar(time, data)
```

Результат приведен на рис. 3.2. Пропущенные столбики соответствуют тем моментам времени, в которые измерения не производились.

Выбор ширины столбцов осуществляется заданием третьего дополнительного аргумента. По умолчанию ширина равна 0.8. Диаграмма без промежутков между столбиками получается, если установить ширину, равную единице. Выбор значений, больших единицы, приводит к перекрывающимся

столбикам. В качестве примера отобразите функцию $x(t) = \sin t \cdot e^t$ на отрезке $[-1, 1]$ в виде столбчатой диаграммы без промежутков, выполнив следующую последовательность операций:

```
>> t=[-1:0.1:1];
>> x=sin(t).*exp(t);
>> bar(t,x,1.0)
```

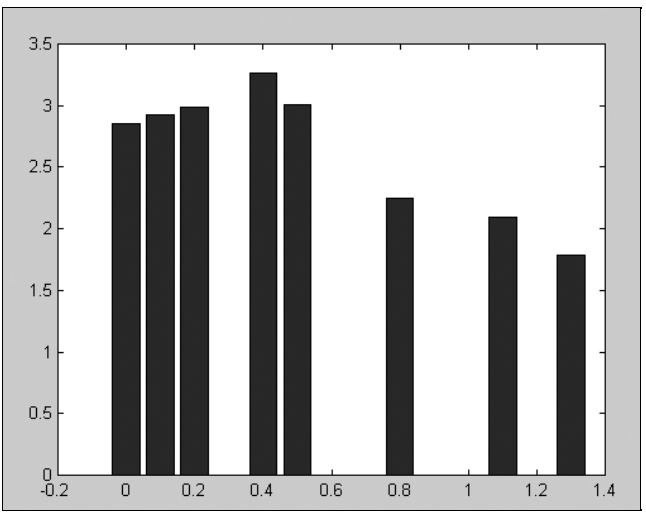


Рис. 3.2. Результаты измерений

Результат (диаграмма без промежутков между столбиками) показан на рис. 3.3.

Замечание

Функция `barh` строит горизонтальную столбчатую диаграмму, т. е. повернутую на девяносто градусов. Для построения объемных диаграмм применяется функция `bar3`. Использование `barh` и `bar3` аналогично `bar`.

Если требуется оценить вклад каждого из элементов вектора в общую сумму его элементов, то удобно построить круговую диаграмму при помощи функции `pie`, например:

```
>> data = [19.5 13.4 42.6 7.9];
>> pie(data)
```

В результате получается диаграмма, изображенная на рис. 3.4, в которой площади секторов отвечают процентному вкладу каждого из элементов вектора в общую сумму, т. е. MatLab *нормирует* значения, вычисляя `data/sum(data)`.

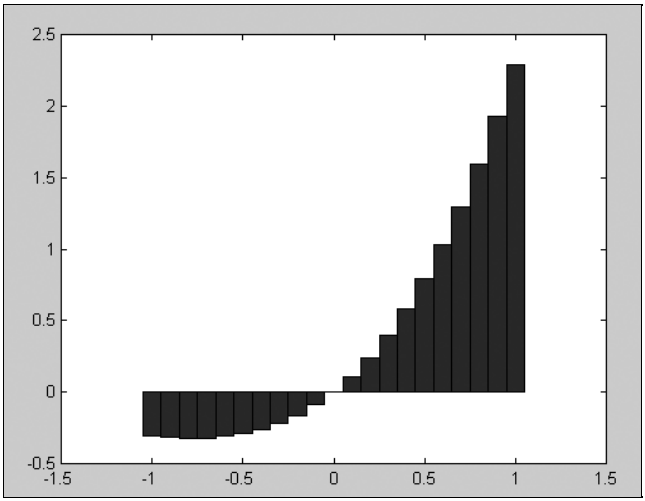


Рис. 3.3. Отображение функции в виде диаграммы

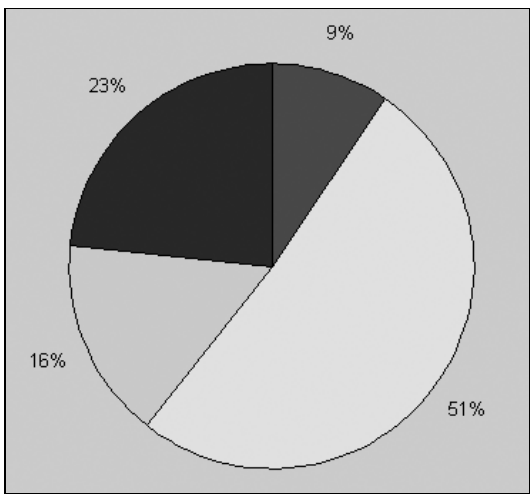


Рис. 3.4. Круговая диаграмма

Если сумма элементов вектора (аргумента `pie`) больше или равна единице, то MatLab производит нормировку и строит круг, состоящий из секторов. Если сумма меньше единицы, то нормировка не производится и получается круг с пропущенным сектором, такой, как на рис. 3.5.

Часто необходимо отодвинуть от круга диаграммы сектор, соответствующий некоторому элементу. Это можно проделать, задав вторым аргументом функции `pie` вектор, состоящий из единиц и нулей, причем единица стоит в

позиции, соответствующей номеру отделяемой части. Диаграмма с отделенным сектором (см. рис. 3.6), отвечающим значению 13.4, выводится в результате выполнения команд

```
>> data = [19.5 13.4 42.6 7.9];
>> parts = [0 1 0 0];
>> pie(data, parts)
```

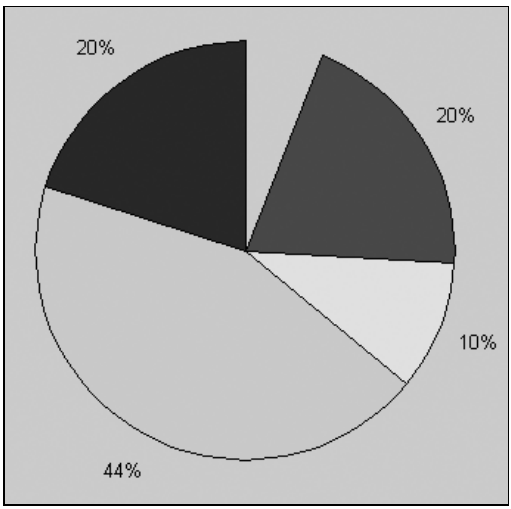


Рис. 3.5. Круговая диаграмма с выброшенным сектором

Можно отделить несколько секторов, расположив единицы во вспомогательном векторе на подходящих позициях. Важно только, чтобы размеры векторов были одинаковы.

В качестве упражнения напишите команды построения диаграммы с отделенным сектором, соответствующим максимальному значению среди элементов вектора, автоматически создав вспомогательный вектор. Используйте функции `zeros` для создания нулевого вектора, той же длины, что *x*, и `max` с двумя выходными аргументами для поиска номера максимального элемента в векторе *x*. Ниже приведена требуемая последовательность команд:

```
>> parts = zeros(size(data));
>> [mx, ind] = max(data);
>> parts(ind) = 1;
>> pie(data, parts)
```

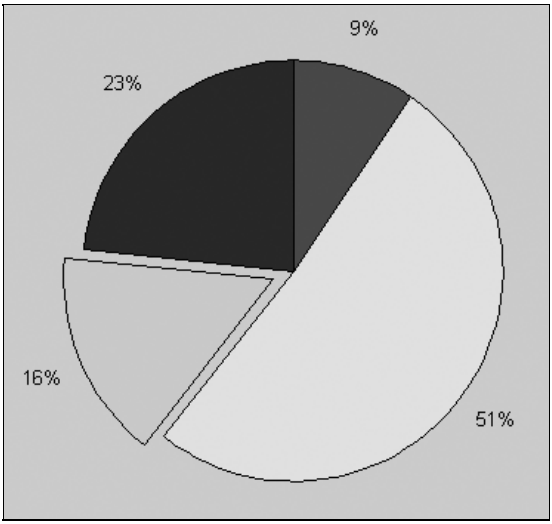


Рис. 3.6. Круговая диаграмма с отделенным сектором

Визуализация векторных данных может быть осуществлена при помощи `pie3` и `bar3`, которые строят трехмерные круговые и столбчатые диаграммы, например команды

```
>> data = [24.1 17.4 10.9];
>> parts = [1 0 0];
>> pie3(data, parts)
```

приводят к появлению трехмерной круговой диаграммы с отделенным сектором, изображенной на рис. 3.7.

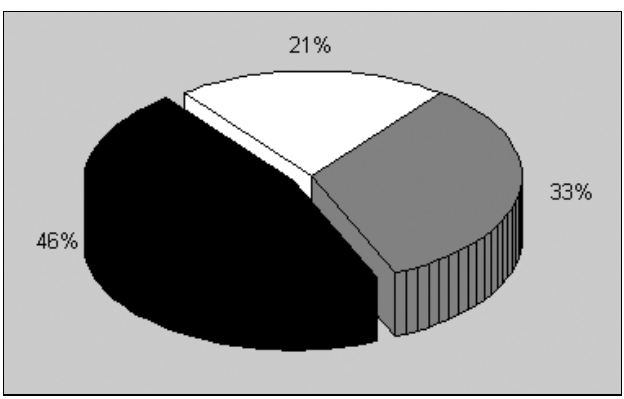


Рис. 3.7. Трехмерная круговая диаграмма

Гистограммы векторных данных

Обработка данных включает вопрос о том, сколько данных попало в тот или иной интервал. Для получения наглядного представления о распределении данных служит функция `hist`. Например, команды

```
>> data = randn(100000, 1);
>> hist(data)
```

заполняют вектор *x* числами, распределенными по нормальному закону, разбивают интервал, которому они принадлежат, на десять равных частей (по умолчанию) и строят гистограмму попадания чисел в каждый из интервалов. Получающаяся гистограмма приведена на рис. 3.8.

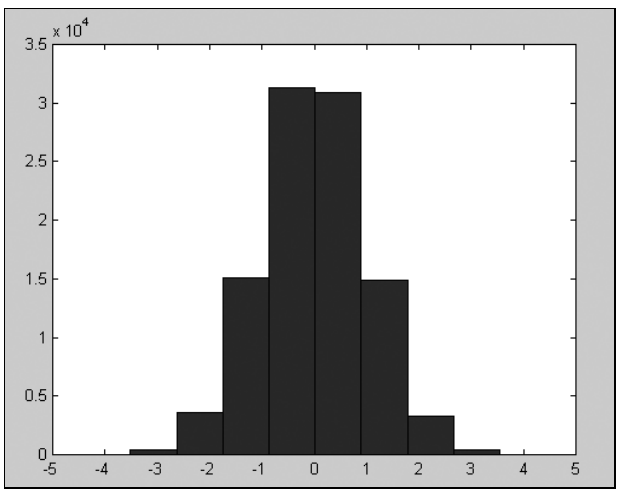


Рис. 3.8. Гистограмма распределения чисел по десяти интервалам

Замечание

Обратите внимание на масштаб вертикальной оси. Число 10^4 в левом верхнем углу значит, что значения по вертикальной оси умножаются на 10 000, т. е. по вертикальной оси отложены числа 5000, 10 000, 15 000 и т. д.

Для увеличения числа интервалов следует в качестве второго аргумента указать число интервалов, например `hist(data, 50)`. Вместо автоматического разбиения на равные интервалы можно использовать собственное, задав вторым аргументом вектор, содержащий центры интервалов. Команды

```
>> data = [0.9 1.0 1.1 1.2 1.4 2.4 3.0 3.3];
>> centers = [1.1 2.3 3.2];
>> hist(data, centers)
```

приводят к построению диаграммы, изображенной на рис. 3.9, где звездочки на горизонтальной оси отмечают центры интервалов.

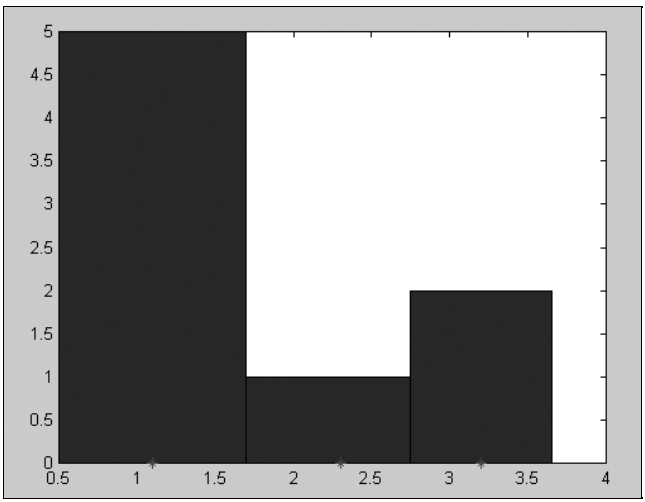


Рис. 3.9. Гистограмма распределения по интервалам, задаваемым центрами

Часто необходимо задать не центры, а границы интервалов. Для построения таких гистограмм следует использовать функцию `histc` в сочетании с вышеописанной функцией `bar`. Функция `histc` возвращает вектор, содержащий число величин, попавших в заданные интервалы. При помощи функции `bar` с дополнительным аргументом `'histc'` полученный вектор представляется в виде гистограммы.

```
 >> data = [0.9 1.0 1.1 1.2 1.4 2.4 3.0 3.3];
 >> intervals = [1.1 2.0 3.2];
 >> count = histc(x, interval)
count =
     3     2     0
 >> bar(interval, count, 'histc')
```

В результате выполнения вышеописанных команд появляется гистограмма, приведенная на рис. 3.10.

Функцию `hist` можно вызывать с одним или двумя выходными аргументами для получения массивов с информацией о распределении данных, при этом гистограмма не строится. В случае одного аргумента в него записывается вектор, содержащий распределение данных по интервалам. Следующий пример демонстрирует создание вектора `count` из пяти элементов, каждый

из которых соответствует числу элементов из data, попавших в один из пяти интервалов.

```
>> data = randn(10000, 1);
>> count = hist(data, 5)
count =
          98        1915        5398        2434         155
```

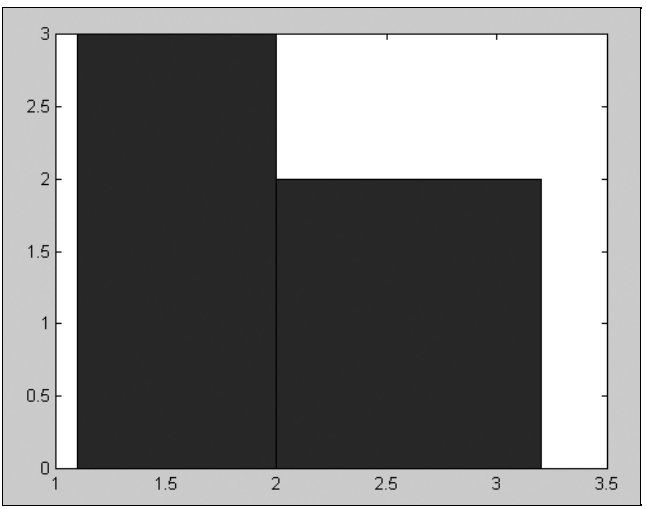


Рис. 3.10. Гистограмма распределения по интервалам, задаваемым границами

Использование hist с двумя аргументами приводит к получению дополнительного вектора с информацией о расположении интервалов:

```
>> [count, intervals] = hist(data, 5)
count =
          98        1915        5398        2434         155
intervals =
   -3.0520   -1.5614   -0.0707    1.4199    2.9106
```

Функция rose предназначена для построения угловых гистограмм (в полярных координатах). Аргументом функции rose является вектор значений в радианах. Угловые гистограммы дают наглядное представление о данных, связанных с измерениями направлений. Пусть, например, в течение суток каждый час измерялось направление ветра в градусах. Результат измерений содержится в файле winddir.dat. Для выяснения преобладающего направления используйте круговую гистограмму, считав значения из файла в вектор

data и преобразовав их в значения в радианах. Файл winddir.dat сохраните в подкаталоге work основного каталога MatLab.

```
>> data = load('winddir.dat');
>> datarad = data * pi/180;
>> rose(datarad)
```

Получающаяся круговая гистограмма изображена на рис. 3.11. Из нее следует, что преобладающее направление примерно равно 100°.

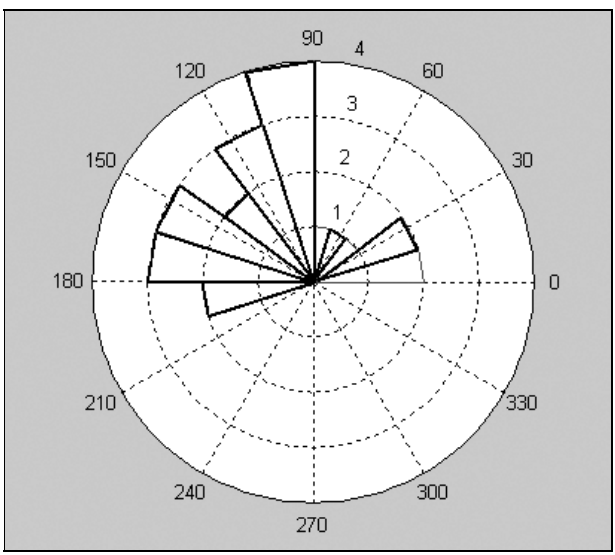


Рис. 3.11. Круговая гистограмма распределения направлений ветра

Функция `rose`, так же как и `hist`, допускает получение информации о распределении по интервалам и о границах интервалов при вызове ее с выходными аргументами. Гистограмма в этом случае не отображается.

Если производятся измерения группы величин, то результат представляет собой матрицу. Для отображения матричных данных используются те же функции, что и для векторных данных. Особенности работы с матричными данными изложены в следующем разделе

Представление матричных данных

Предположим, что в матрице *DATA*, состоящей из четырех строк и трех столбцов содержатся результаты измерений трех величин за четыре момента

времени. Для построения столбчатой диаграммы данных примените функцию `bar`, задав в качестве аргумента массив `DATA`:

```
>> DATA = [1.2 1.4 1.1
          3.7 3.5 3.1
          2.0 2.8 2.2
          4.2 4.7 4.1];
>> bar(DATA)
```

В результате появляется диаграмма сгруппированных данных, изображенная на рис. 3.12. На диаграмме расположены четыре группы данных, каждая из которых состоит из трех столбиков, соответствующих измеряемым величинам.

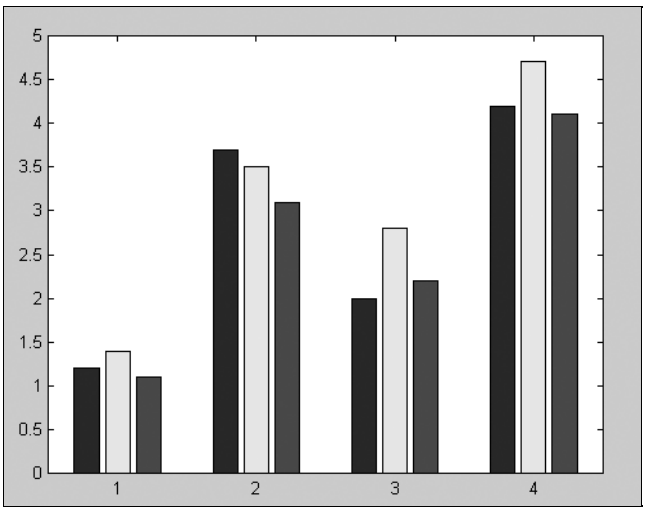


Рис. 3.12. Диаграмма сгруппированных данных

Использование аргументов функции `bar` для визуализации матричных данных не отличается от случая векторных данных, разобранного в предыдущем разделе. Например, ширина интервалов между столбцами внутри группы задается при помощи числового параметра. Диаграмма с перекрывающимися столбцами внутри группы, приведенная на рис. 3.13, получена при помощи `bar(DATA,1.7)`.

Вклад каждой из величин в общую сумму внутри группы хорошо виден на диаграмме с накоплением, в которой каждая группа отображается в виде столбика, состоящего из частей. Число частей равно числу измеряемых величин, а их высота соответствует вкладу каждой величины в сумму внутри группы. На рис. 3.14 показана диаграмма с накоплением, построенная при помощи функции `bar` с дополнительным аргументом: `bar(DATA,'stack')`.

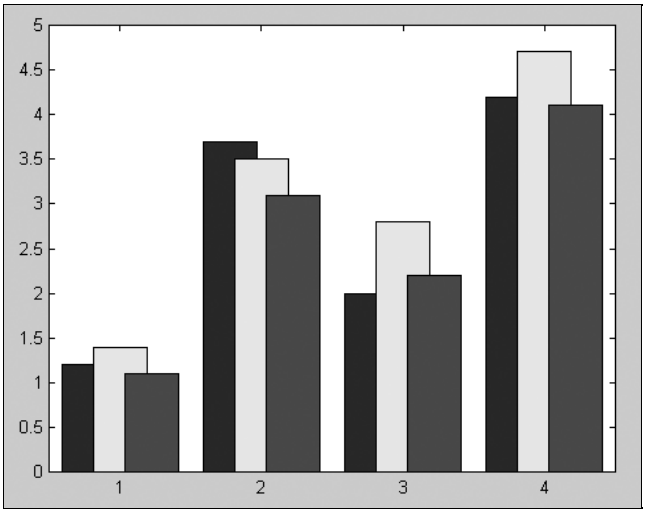


Рис. 3.13. Диаграмма сгруппированных данных с перекрывающимися столбцами

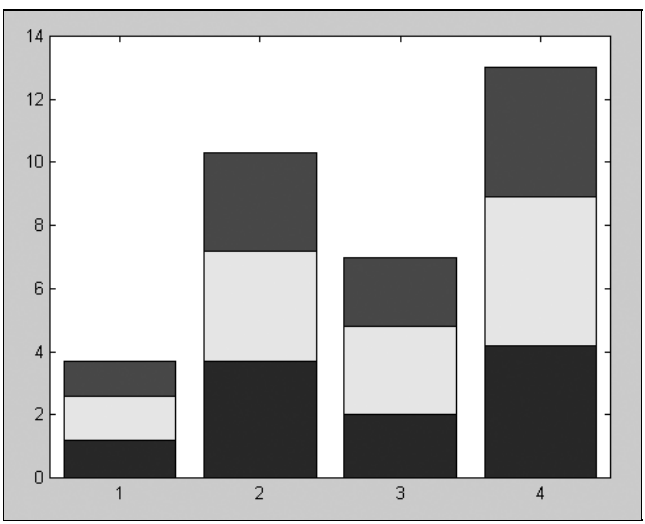


Рис. 3.14. Диаграмма с накоплением

Использование функции `barh` в случае матриц осуществляется так же, как и применение функции `bar`. Наглядная трехмерная столбцевая диаграмма, представляющая матричные данные, получается, если применить функцию `bar3`.

Проследить за изменением величин и одновременно узнать вклад значений в общую сумму позволяет функция `area`, выводящая диаграмму с областями.

Запишите в матрицу *GAIN* поквартальную прибыль от продаж трех видов продукции и проследите за изменением прибыли при помощи диаграммы с областями (см. рис. 3.15).

```
>> GAIN = [12.0 23.0 48.0
          10.6 31.5 49.0
          8.0  25.0 78.0
          9.6  29.0 61.5];
>> area(GAIN)
```

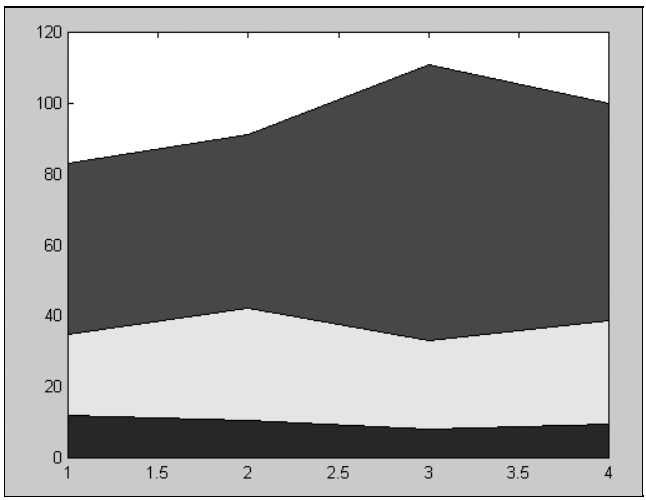


Рис. 3.15. Диаграмма с областями

Дополнительные возможности для визуализации разреженных матриц и представлении матриц в виде изображений предоставляют функции `spy` и `imagesc`.

Применение `spy` *и* `imagesc` *описано в главе 2.*

Перейдем теперь к построению графиков функций одной и двух переменных.

Графики функций

MatLab предоставляет обширные возможности для визуализации функций одной и двух переменных. Использование функций для построения графиков с минимальным набором задаваемых параметров (остальные MatLab выбирает автоматически) приводит к получению качественных графиков. В этом разделе разобрано управление основными свойствами линий на графиках, которое осуществляется при помощи дополнительных параметров графических функций.

Графики функций одной переменной

MatLab позволяет строить графики функций в линейном, логарифмическом и полулогарифмическом масштабах. Причем в одном окне можно построить графики нескольких функций, даже определенных на разных отрезках.

Графики в линейном масштабе

Построение графиков функций одной переменной в линейном масштабе осуществляется при помощи функции `plot`. В зависимости от входных аргументов функция `plot` позволяет строить один или несколько графиков, изменять цвет и стиль линий и добавлять маркеры на каждый график. Во второй главе приведен пример вывода простейшего графика функции одной переменной:

```
>> x=[0:0.05:1];
>> y = exp(-x).*sin(10*x);
>> plot(x, y)
```

Обратите внимание, что понимание поэлементных операций с векторами является необходимым условием для графического отображения функций. В случае возникновения вопросов обращайтесь к предыдущей главе.

Сравнение нескольких функций легко производить, построив графики на одних координатных осях. Постройте графики функций $f(x)=e^{-0.1x}\sin^2 x$ и $g(x)=e^{-0.2x}\sin^2 x$ на отрезке $[-2\pi, 2\pi]$. Сгенерируйте вектор-строку значений аргумента `x` и вектор-строк *f* и *g*, содержащих значения функций. Команда `plot` с двумя парами аргументов приводит к построению графика, изображенного на рис. 3.16.

```
>> x = [-2*pi:pi/20:2*pi];
>> f = exp(-0.1*x).*sin(x).^2;
>> g = exp(-0.2*x).*sin(x).^2;
>> plot(x, f, x, g)
```

Функции необязательно должны быть определены на одном и том же отрезке. В этом случае при построении графиков MatLab выбирает максимальный отрезок, содержащий остальные. Важно только в каждой паре векторов абсцисс и ординат указать соответствующие друг другу вектора, например:

```
>> x1 = [-pi:0.01:2*pi];
>> f = exp(-0.1*x1).*sin(x1).^2;
>> x2 = [-2*pi:0.01:pi];
>> g = exp(-0.2*x2).*sin(x2).^2;
>> plot(x1, f, x2 ,g)
```

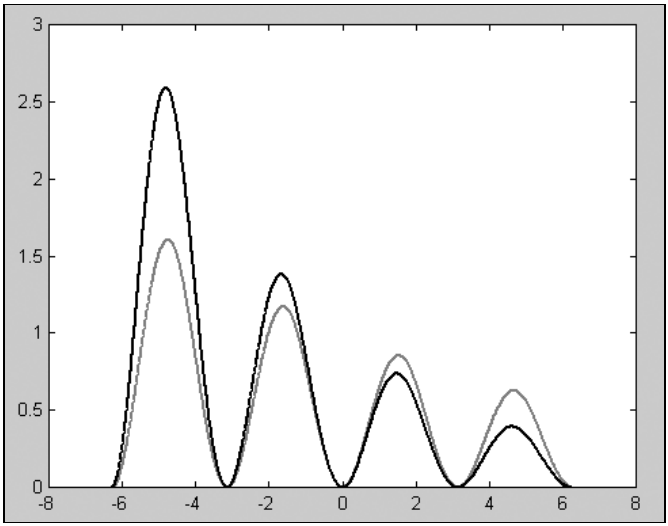


Рис. 3.16. Графики двух функций

Аналогичным образом при помощи задания в `plot` через запятую пар аргументов вида: вектор абсцисс, вектор ординат, осуществляется построение графиков произвольного числа функций.

Замечание

Использование `plot` с одним аргументом — вектором — приводит к построению "графика вектора", т. е. зависимости значений элементов вектора от их номеров. Аргументом `plot` может быть и матрица, в этом случае на одни координатные оси выводятся графики столбцов.

Иногда требуется сравнить поведение двух функций, значения которых сильно отличаются друг от друга. График функции с небольшими значениями практически сливается с осью абсцисс, и установить его вид не удается. В этой ситуации помогает функция `plotyy`, которая выводит графики в окно с двумя вертикальными осями, имеющими подходящий масштаб. Сравните, например, две функции: $c_{jk} = a_j b_k$ и $F(x) = 1000 * (x + 0.5)^{-4}$.

```
>> x = [0.5:0.01:3];
>> f = x.^-3;
>> F = 1000*(x+0.5).^-4;
>> plotyy(x, f ,x, F)
```

Результат приведен на рис. 3.17. При выполнении этого примера обратите внимание, что цвет графика совпадает с цветом соответствующей ему оси ординат.

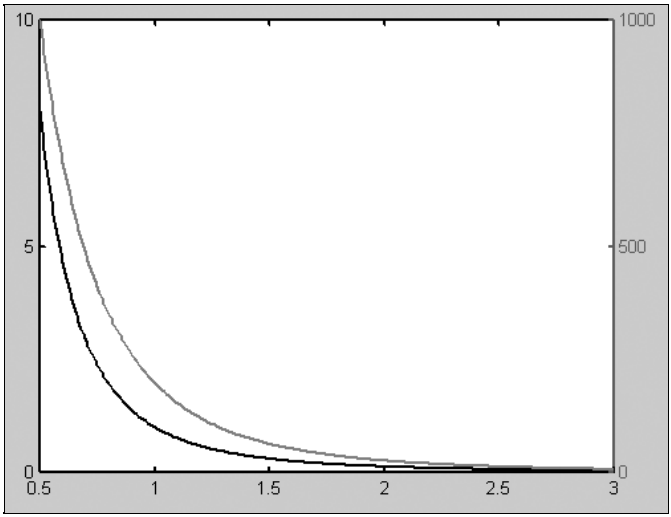


Рис. 3.17. Сравнение графиков при помощи функции `plotyy`

Функция `plot` использует линейный масштаб по обеим координатным осям. Однако MatLab предоставляет пользователю возможность строить графики функций одной переменной в логарифмическом или полулогарифмическом масштабе.

Графики в логарифмических масштабах

Для построения графиков в логарифмическом и полулогарифмическом масштабах служат следующие функции:

- `loglog` (логарифмический масштаб по обеим осям);
- `semilogx` (логарифмический масштаб только по оси абсцисс);
- `semilogy` (логарифмический масштаб только по оси ординат).

Аргументы `loglog`, `semilogx` и `semilogy` задаются в виде пары векторов значений абсцисс и ординат так же, как для функции `plot`, описанной в предыдущем разделе. Постройте, например, графики функций $f(x) = \ln 0.5x$ и $g(x) = \sin \ln x$ на отрезке [0.1, 5] в логарифмическом масштабе по оси x:

```
>> x = [0.1:0.01:10];
>> f = log(0.5*x);
>> g = sin(log(x));
>> semilogx(x, f, x ,g)
```

Получающиеся графики изображены на рис. 3.18.

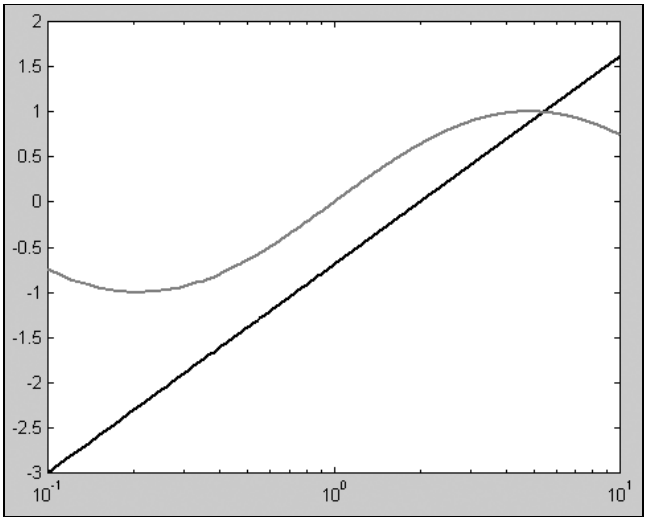


Рис. 3.18. Графики в полулогарифмической шкале

Аналогичным образом строятся графики при помощи функций `loglog` и `semilogy`.

Изменение свойств линий

Построенные графики функций должны быть максимально удобными для восприятия. Часто требуется нанести маркеры, изменить цвет линий, а при подготовке к монохромной печати — задать тип линии (сплошная, пунктирная, штрих-пунктирная и т. д.). MatLab предоставляет возможность управлять видом графиков, построенных при помощи `plot`, `loglog`, `semilogx` и `semilogy`, для чего служит дополнительный аргумент, помещаемый за каждой парой векторов. Этот аргумент заключается в апострофы и состоит из трех символов, которые определяют: цвет, тип маркера и тип линии. Используется одна, две или три позиции, в зависимости от требуемых изменений. В табл. 3.1 приведены возможные значения данного аргумента с указанием результата.

Таблица. 3.1. *Свойства линии*

Цвет		Тип маркера		Тип линии	
y	желтый	.	точка	-	сплошная
m	розовый	o	кружок	:	пунктирная
c	голубой	x	крестик	-.	штрих-пунктирная
r	красный	+	знак "плюс"	--	штриховая

Таблица. 3.1 *(окончание)*

Цвет		Тип маркера		Тип линии	
g	зеленый	*	звездочка		
b	синий	s	квадрат		
w	белый	d	ромб		
k	черный	v	треугольник вершиной вниз		
		^	треугольник вершиной вверх		
		<	треугольник вершиной влево		
		>	треугольник вершиной вправо		
		p	пятиконечная звезда		
		h	шестиконечная звезда		

Например, для построения первого графика (см. рис. 3.16) красными точечными маркерами без линии, а второго пунктирной черной линией следует использовать команду `plot(x, f, 'r.', x, g, 'k:')`. Результат приведен на рис. 3.19. Обратите внимание, что абсциссы маркеров совпадают со значениями аргумента, содержащихся в массиве `x`.

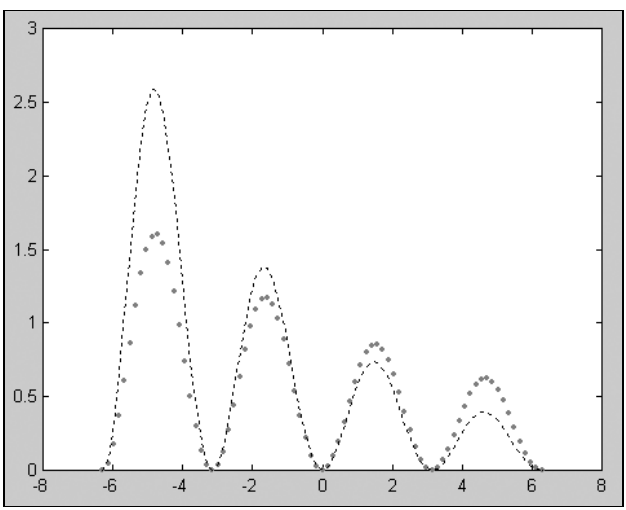


Рис. 3.19. Изменение параметров линий

Оформление графиков

Удобство использования графиков во многом зависит от дополнительных элементов оформления: координатной сетки, подписей к осям, заголовка и легенды. Сетка наносится командой `grid on`, подписи к осям размещаются при помощи `xlabel`, `ylabel`, заголовок дается командой `title`. Наличие нескольких графиков на одних осях требует помещения легенды командой `legend` с информацией о линиях. Все перечисленные команды применимы к графикам как в линейном, так и в логарифмическом и полулогарифмическом масштабах. Следующие команды выводят графики изменения суточной температуры, изображенные на рис. 3.20, которые снабжены всей необходимой информацией.

```
>> time = [0 4 7 9 10 11 12 13 13.5 14 14.5 15 16 17 18 20 22];
>> temp1 = [14 15 14 16 18 17 20 22 24 28 25 20 16 13 13 14 13];
>> temp2 = [12 13 13 14 16 18 20 20 23 25 25 20 16 12 12 11 10];
>> plot(time, temp1, 'ro-', time, temp2, 'go-')
>> grid on
>> title('Суточные температуры')
>> xlabel('Время (час.)')
>> ylabel('Температура (C)')
>> legend('10 мая', '11 мая')
```

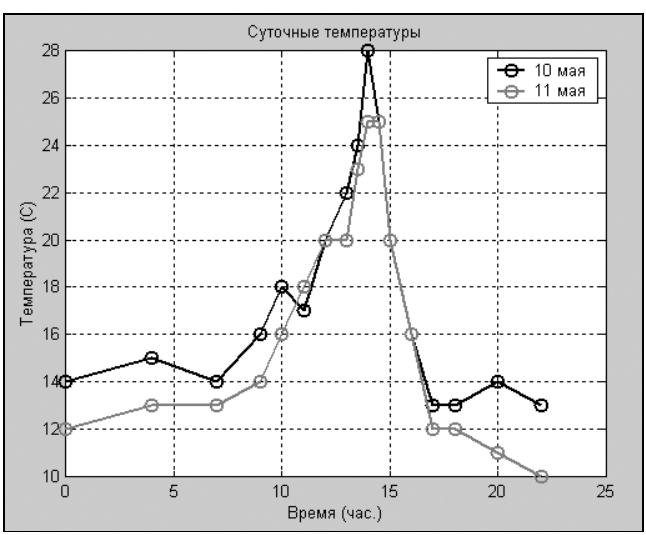


Рис. 3.20. График изменения суточной температуры

При добавлении легенды следует учесть, что порядок и количество аргументов команды `legend` должны соответствовать линиям на графике. Послед-

ним дополнительным аргументом `legend` может быть положение легенды в графическом окне:

- −1 — вне графика в правом верхнем углу графического окна;
- 0 — выбирается лучшее положение в пределах графика так, чтобы как можно меньше перекрывать сами графики;
- 1 — в верхнем правом углу графика (это положение используется по умолчанию);
- 2 — в верхнем левом углу графика;
- 3 — в нижнем левом углу графика;
- 4 — в верхнем левом углу графика.

В заголовке графика, легенде и подписях осей допускается добавление формул и изменение стилей шрифта при помощи формата TeX, подробно об этом написано далее.

Графики параметрических и кусочно-заданных функций

Для построения функций, заданных параметрически, следует сперва сгенерировать вектор значений аргумента. Затем необходимо вычислить значения функций и записать их в векторы, которые и надо использовать в качестве аргументов `plot`. График функции $x(t) = 0.5 \cdot \sin t$, $y(t) = 0.7 \cdot \cos t$ для $t \in [0, 2\pi]$ (эллипс), приведенный на рис. 3.21, получается при помощи следующих команд:

```
>> t = [0:0.01:2*pi];
>> x = 0.5*sin(t);
>> y = 0.7*cos(t);
>> plot(x, y)
```

Для того чтобы проверить свои знания о построении графиков и работе с массивами, постройте график функции, заданной кусочным образом:

$$y(x) = \begin{cases} \pi \cdot \sin x, & -2\pi \le x \le -\pi; \\ \pi - |x|, & -\pi < x < \pi; \\ \pi \cdot \sin^3 x, & \pi \le x \le 2\pi. \end{cases}$$

Сначала необходимо вычислить каждую из трех ветвей, т. е. фактически получить три пары массивов `x1` и `y1`, `x2` и `y2`, `x3` и `y3`, затем объединить значения абсцисс в вектор *x*, а значения ординат в *y* и построить график функции, задаваемой парой массивов `x` и `y`:

```
>> x1 = [-2*pi:0.01:-pi];
>> y1 = pi*sin(x1);
>> x2 = [-pi:0.01:pi];
```

```
>> y2 = pi-abs(x2);
>> x3 = [pi:0.01:2*pi];
>> y3 = pi*sin(x1).^3;
>> x = [x1 x2 x3];
>> y = [y1 y2 y3];
>> plot(x, y)
```

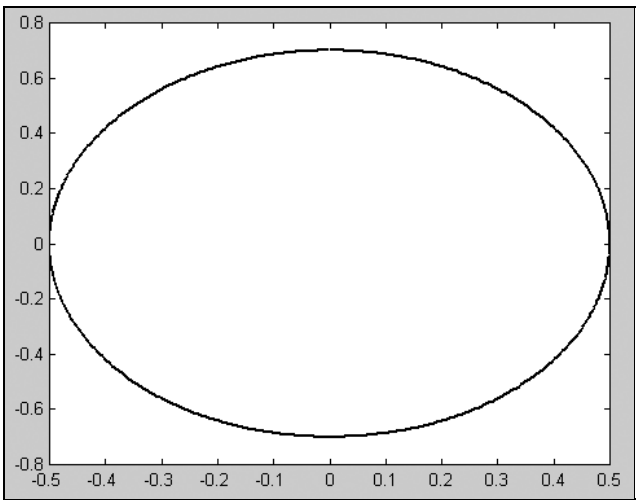


Рис. 3.21. График параметрически заданной функции

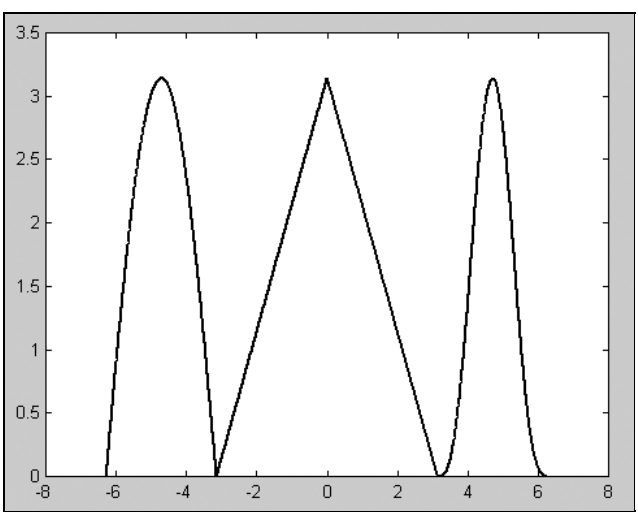


Рис. 3.22. График функции, заданной кусочным образом

При таком подходе получается график, изображенный на рис. 3.22.

Можно поступить и по-другому — построить графики трех ветвей, как три различные функции:

```
>> plot(x1, y1, x2, y2, x3, y3)
```

В этом случае график имеет более наглядный вид, т. к. каждая ветвь функции строится своим цветом (см. рис. 3.23).

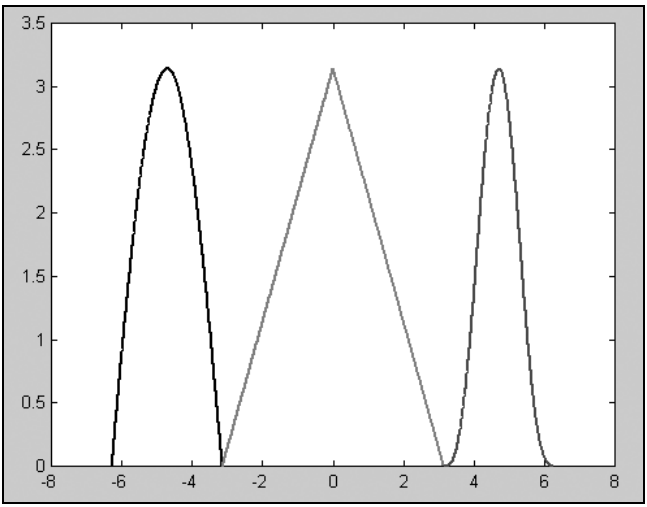


Рис. 3.23. График функции, заданной кусочным образом (разные цвета ветвей условно обозначены различной толщиной линий)

Графики функций двух переменных

MatLab предоставляет различные способы визуализации функций двух переменных — построение трехмерных графиков и линий уровня, параметрически заданных линий и поверхностей.

Трехмерные графики функций

В *главе 2* был разобран простой пример построения трехмерного графика при помощи функции `mesh`. Для отображения функции двух переменных следует:

1. Сгенерировать матрицы с координатами узлов сетки на прямоугольной области определения функции.
2. Вычислить функцию в узлах сетки и записать полученные значения в матрицу.

3. Использовать одну из графических функций MatLab.
4. Нанести на график дополнительную информацию, в частности, соответствие цветов значениям функции.

Сетка генерируется при помощи команды `meshgrid`, вызываемой с двумя аргументами. Аргументами являются векторы, элементы которых соответствуют сетке на прямоугольной области построения функции. Можно использовать один аргумент, если область построения функции — квадрат. Для вычисления функции следует использовать поэлементные операции, описанные в *главе 2*.

Изучим основные возможности, предоставляемые MatLab для визуализации функций двух переменных, на примере построения графика

$$z(x, y) = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot (1 - x^2) \cdot y \cdot (1 - y)$$

на прямоугольной области определения $x \in [-1, 1]$, $y \in [0, 1]$. Подготовьте матрицы с координатами узлов сетки и значениями функции:

```
>> [X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
>> Z = 4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
```

Для построения *каркасной поверхности*, изображенной на рис. 3.24, используется функция `mesh`, вызываемая с тремя аргументами:

```
>> mesh(X,Y,Z)
```

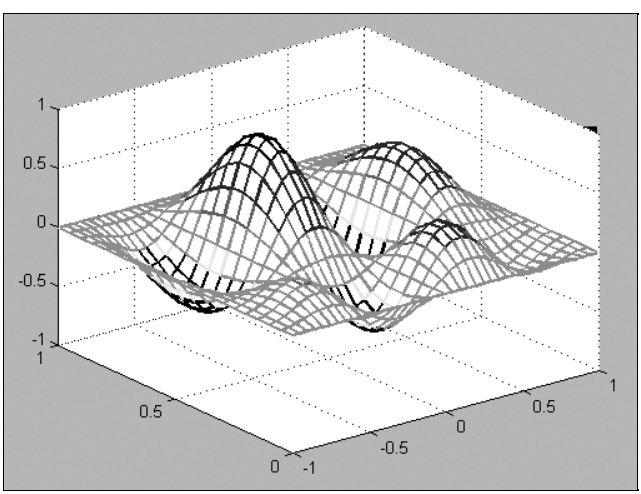


Рис. 3.24. Каркасная поверхность (`mesh`)

Цвет линий поверхности соответствует значениям функции. MatLab рисует только видимую часть поверхности. При помощи команды `hidden off`

можно сделать каркасную поверхность "прозрачной", добавив скрытую часть (см. рис. 3.25). Команда `hidden on` убирает невидимую часть поверхности, возвращая графику прежний вид.

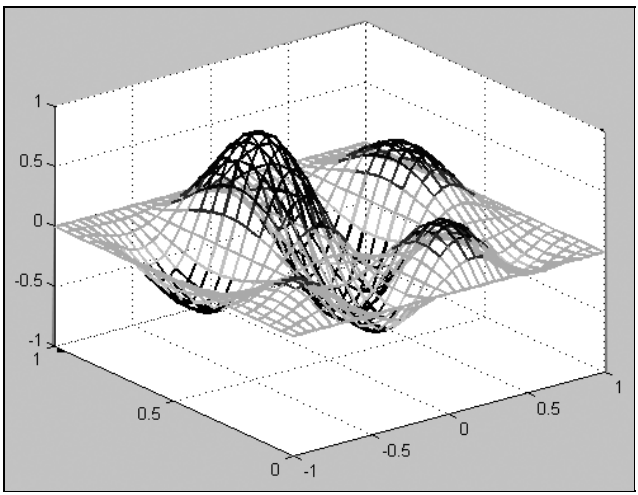


Рис. 3.25. Прозрачная каркасная поверхность (`mesh, hidden off`)

Функция `surf` строит каркасную поверхность графика функции и заливает каждую клетку поверхности определенным цветом, зависящим от значения функции в точках, соответствующих углам клетки. Команда

```
>> surf(X,Y,Z)
```

приводит к результату, изображенному на рис. 3.26.

В пределах каждой клетки цвет постоянный. Команда `shading flat` позволяет убрать каркасные линии. Для получения поверхности, плавно залитой цветом, зависящим от значений функции (рис. 3.27), предназначена команда `shading interp`.

При помощи `shading faceted` можно вернуться к виду поверхности, приведенной на рис. 3.26.

Трехмерные графики, изображенные на рис. 3.24—3.27, удобны для получения представления о форме поверхности, однако по ним трудно судить о значениях функции. В MatLab определена команда `colorbar`, которая выводит рядом с графиком столбик, устанавливающий соответствие между цветом и значением функции. Постройте при помощи `surf` график поверхности и дополните его информацией о цвете.

```
>> surf(X,Y,Z)
>> colorbar
```

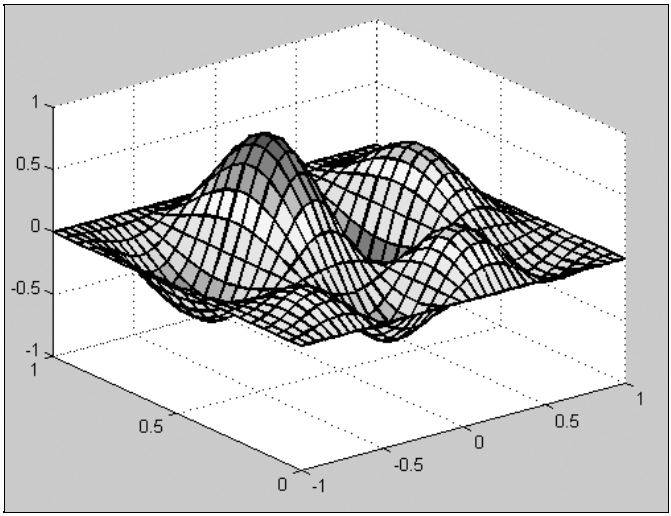


Рис. 3.26. Каркасная поверхность, залитая цветом (`surf`)

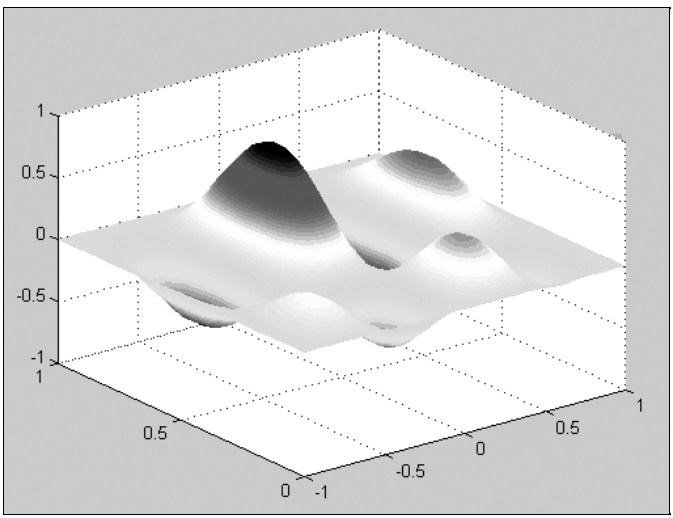


Рис. 3.27. Поверхность, залитая цветом (`surf`, `shading interp`)

Рис. 3.28 иллюстрирует получающийся результат. Окно вывода графика несколько уменьшается из-за того, что рядом размещается столбик с цветовой гаммой. Команду `colorbar` можно применять в сочетании со всеми функциями, строящими трехмерные объекты.

Пользуясь графиком, изображенным на рис. 3.28, трудно сделать вывод о значении функции в той или иной точке плоскости *xy*. Команды `meshc` или

surfc позволяют получить более точное представление о поведении функции. Эти команды строят каркасную поверхность или залитую цветом каркасную поверхность и размещают на плоскости *xy* линии уровня функции (линии постоянства значений функции):

```
>> surfc(X,Y,Z)
>> colorbar
```

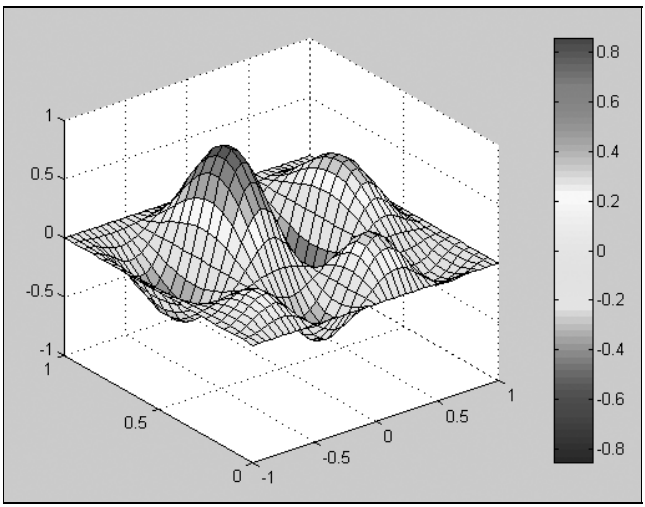


Рис. 3.28. Соответствие цвета и значений функции (colorbar)

Получающийся график приведен на рис. 3.29.

MatLab позволяет построить поверхность, состоящую из линий уровня, при помощи функции contour3. Эту функцию можно использовать так же, как и описанные выше mesh, surf, meshc и surfc с тремя аргументами. При этом число линий уровня выбирается автоматически. Имеется возможность задать четвертым аргументом в contour3 либо число линий уровня, либо вектор, элементы которого равны значениям функции, отображаемым в виде линий уровня. Задание вектора удобно, когда требуется исследовать поведение функции в некоторой области ее значений (срез функции). Постройте, например поверхность, состоящую из линий уровня, соответствующих значениям функции от 0 до 0.5 с шагом 0.01:

```
>> levels=[0:0.01:0.5];
>> contour3(X,Y,Z,levels)
>> colorbar
```

Результат приведен на рис. 3.30.

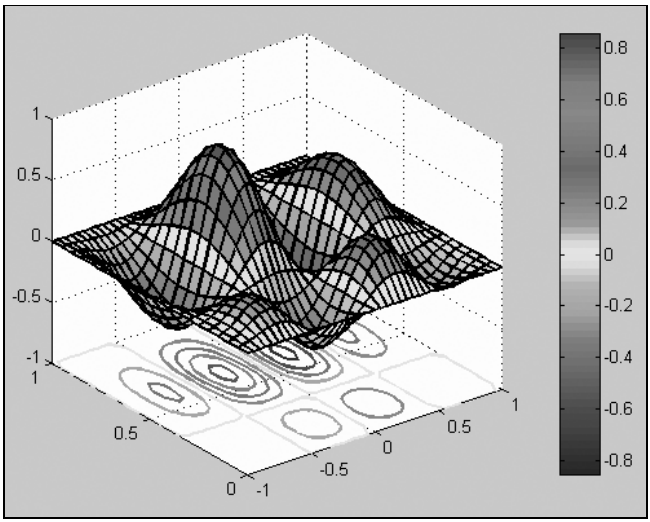


Рис. 3.29. График поверхности с линиями уровня на плоскости *xy* (`surfc`)

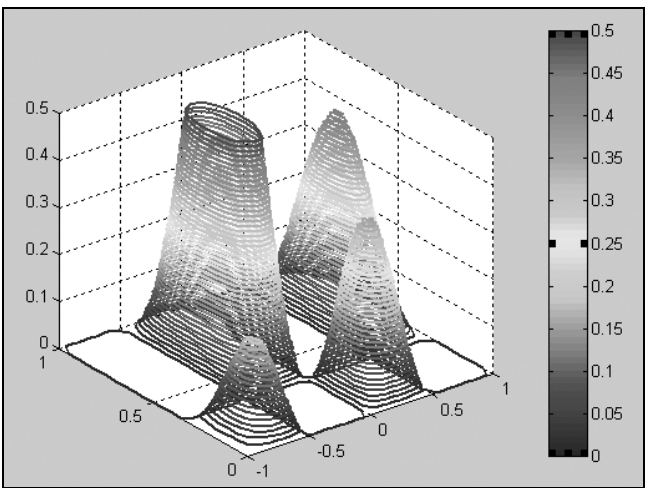


Рис. 3.30. График среза функции, состоящий из линий уровня (`contour3`)

Часто более содержательную информацию о числовых значениях дают плоские контурные графики, содержащие линии уровня исследуемой функции.

Контурные графики

MatLab предоставляет возможность получать различные типы контурных графиков при помощи функций `contour` и `contourf`. Разберем их возможно-

сти на примере функции из предыдущего раздела. Использование `contour` с тремя аргументами

```
>> contour(X,Y,Z)
```

приводит к графику, изображенному на рис. 3.31, на котором показаны линии уровня на плоскости *xy*.

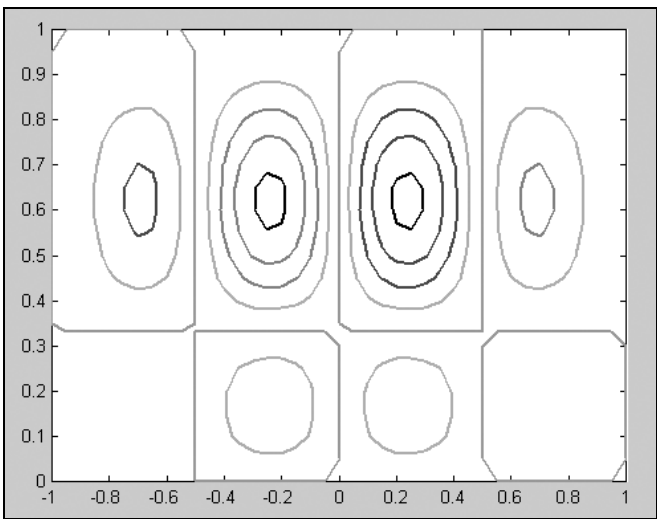


Рис. 3.31. Линии уровня функции (`contour`)

Такой график является малоинформативным, он не позволяет узнать значения функции на каждой из линий уровня. Использование команды `colorbar` также не позволит точно определить значения функции. Каждую линию уровня можно снабдить значением, которое принимает на ней исследуемая функция, при помощи определенной в MatLab функции `clabel`. Функция `clabel` вызывается с двумя аргументами: матрицей, содержащей информацию о линиях уровня и указателем на график, на котором следует нанести разметку. Нам пока ничего не говорят эти слова. Про указатели будет сказано в главах, посвященных созданию собственных приложений, а про структуру матрицы с информацией о линиях уровня можно узнать при помощи `help`. Оказывается, пользователю не нужно самому создавать аргументы `clabel`. Функция `contour`, вызванная с двумя выходными параметрами, не только строит линии уровня, но и находит требуемые для `clabel` параметры. Используйте `contour` с выходными аргументами (в массиве `CMatr` содержится информация о линиях уровня, а в массиве `h` — указатели). За-

вершите вызов `contour` точкой с запятой для подавления вывода на экран значений выходных параметров и нанесите на график сетку:

```
>> [CMatr, h] = contour(X, Y, Z);
>> clabel(CMatr, h)
>> grid on
```

Полученный график приведен на рис. 3.32.

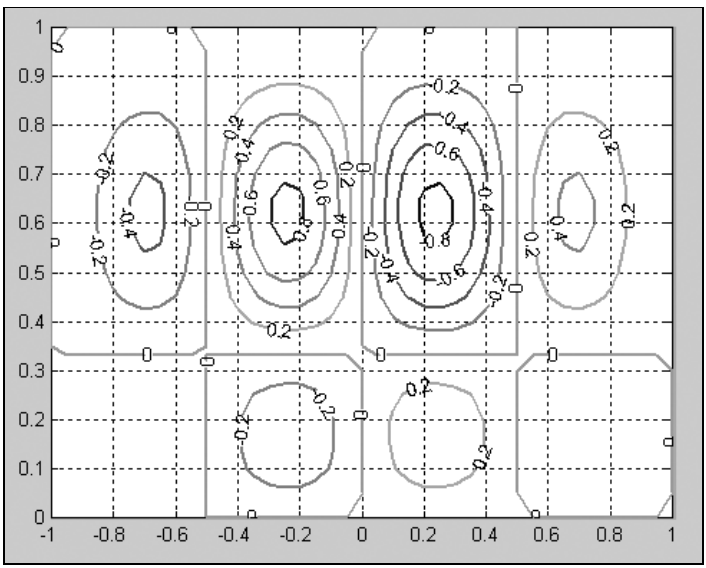


Рис. 3.32. Маркированные линии уровня (`contour`, `clabel`)

Дополнительным аргументом функции `contour` (так же, как и `contour3`, описанной выше) может быть или число линий уровня, или вектор, содержащий значения функции, для которых требуется построить линии уровня.

Наглядную информацию об изменении функции дает заливка прямоугольника на плоскости *xy* цветом, зависящим от значения функции в точках плоскости. Для построения таких графиков предназначена функция `contourf`, использование которой не отличается от применения `contour`. В следующем примере выводится график, изображенный на рис. 3.33, который состоит из двадцати линий уровня, а промежутки между ними заполнены цветами, соответствующими значениям исследуемой функции:

```
>> contourf(X, Y, Z, 20)
>> colorbar
```

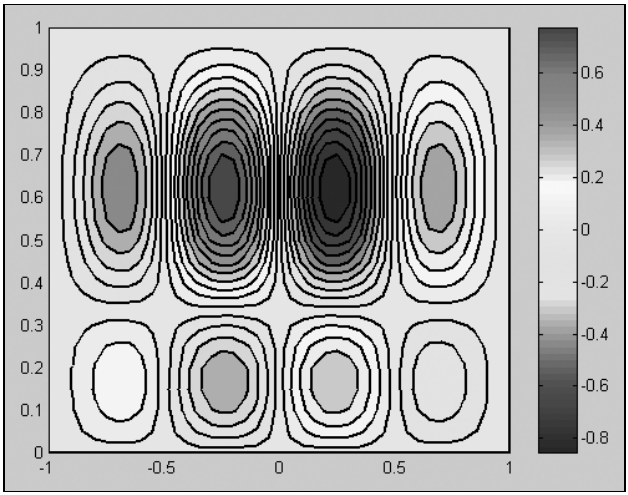


Рис. 3.33. Заливка цветом промежутков между линиями уровня (`contourf`)

Оформление графика

Простым, но эффективным способом изменения цветового оформления графика является установка цветовой палитры при помощи функции `colormap`. Следующий пример демонстрирует, как подготовить график функции для печати на монохромном принтере, используя палитру `gray`. Результат приведен на рис. 3.34.

```
>> surfc(X, Y, Z)
>> colorbar
>> colormap(gray)
>> title('График функции z(x,y)')
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

Обратите внимание, что команда `colormap(gray)` изменяет палитру графического окна, т. е. следующие графики будут выводиться в этом окне также в серых тонах. Для восстановления первоначального значения палитры следует применить команду `colormap('default')`. Цветовые палитры, доступные в MatLab, приведены в табл. 3.2.

Таблица 3.2. *Палитры цвета*

Палитра	Изменение цвета
`autumn`	плавное изменение: красный-оранжевый-желтый

Таблица 3.2 *(окончание)*

Палитра	Изменение цвета
bone	похожа на палитру gray, но с легким оттенком синего цвета
colorcube	каждый цвет изменяется от темного к яркому
cool	оттенки голубого и пурпурного цветов
copper	оттенки медного цвета
flag	циклическое изменение: красный-белый-синий-черный
gray	оттенки серого
hot	плавное изменение: черный-красный-оранжевый-желтый-белый
hsv	плавное изменение (как цвета радуги)
jet	плавное изменение: синий-голубой-зеленый-желтый-красный
pink	похожа на палитру gray, но с легким оттенком коричневого цвета
prism	циклическое изменение: красный-оранжевый-желтый-зеленый-синий-фиолетовый
spring	оттенки пурпурного и желтого
summer	оттенки зеленого и желтого
vga	палитра Windows из шестнадцати цветов
white	один белый цвет
winter	оттенки синего и зеленого

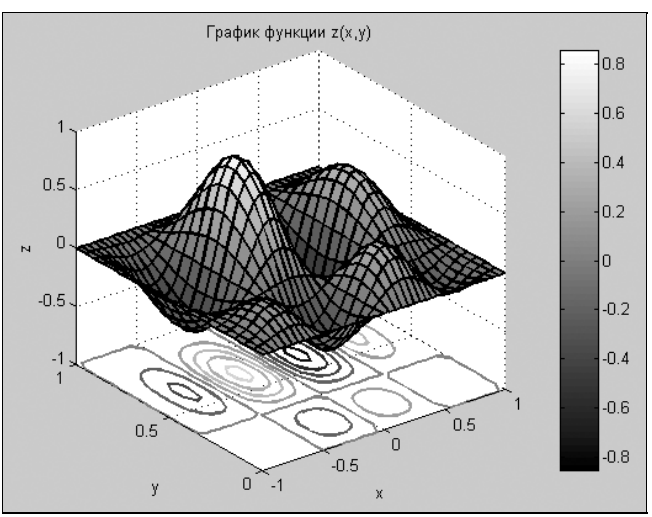


Рис. 3.34. График с нанесенными обозначениями (палитра gray)

Поэкспериментируйте самостоятельно, задавая различные палитры цвета и способы построения функций.

Добавление заголовка графика и подписей к осям осуществляется теми же командами `title`, `xlabel`, `ylabel`, что применялись при визуализации функций одной переменной. Несложно догадаться, что для вертикальной оси предназначена команда `zlabel`. Часто требуется добавить в заголовок или рядом с вертикальной осью формулу. Использование в аргументах команд некоторых математических обозначений в формате TeX позволяет добавлять формулы на график. В заголовок графика, изображенного на рис. 3.33, поместите формулу отображаемой функции

$$z = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot (1 - x^2) \cdot y \cdot (1 - y).$$

Используйте команду, которая приводит к появлению требуемого заголовка (три точки использованы для того, чтобы разместить команду в двух строках; если она целиком помещается в командной строке, то ее можно набирать без переноса):

```
>> title('4 sin(2\pi{\itx}) cos(1.5\pi{\ity}) (1-{\ity}^{2})...
{\ity} (1-{\ity})')
```

Если вам знаком TeX, то вопросов возникнуть не должно. Если же вы не работаете в TeX, то просто используйте правила набора формул и изменения свойств шрифтов, приведенные в табл. 3.3.

Таблица 3.3. *Правила набора формул и изменения свойств шрифтов*

Что требуется	Команда TeX	Результат
Выделение курсивом одного символа или текста	`{\itx}`	*x*
	`1.2{\itP}`	1.2*P*
	`{\itГиперболический}` синус	*Гиперболический* синус
Выделение жирным шрифтом одного символа или текста	Шаблон матрицы `{\bfM}`	Шаблон матрицы **M**
	`{\bfАЧХ}` фильтра	**АЧХ** фильтра
Набор символа или текста жирным курсивом	Векторы `{\bf\itx}` и `{\bf\ity}`	Векторы ***x*** и ***y***
	`{\bf\itОптимальная}` кривая	***Оптимальная*** кривая
Изменение шрифта и его размера	`{\fontname{arial}\fontsize{14}Z-функция}`	Z-функция

Таблица 3.3 *(окончание)*

Что требуется	Команда TeX	Результат
Степень, верхний индекс	`x^{2}`	x^2
	`{\itx}^{2.5}`	$x^{2.5}$
	`{\ite}^{\it-x}`	e^{-x}
Нижний индекс	`f_{5}`	f_5
	`f_{\itxx}`	f_{xx}

Замечание

Прямой шрифт текста в TeX устанавливается командой `\rm`. Для получения обычного прямого шрифта можно не указывать никаких команд.

Возможно использование греческих букв и специальных символов, например `title('Зависимость при a=\pi')` приводит к заголовку: "Зависимость при a=π". Ниже в табл. 3.4 и 3.5 приведены команды TeX для вставки некоторых прописных и строчных греческих букв и специальных символов.

Таблица 3.4. *Греческие буквы*

Команда	Символ	Команда	Символ	Команда	Символ
`\alpha`	α	`\lambda`	λ	`\chi`	χ
`\beta`	β	`\mu`	μ	`\psi`	ψ
`\gamma`	γ	`\nu`	ν	`\omega`	ω
`\delta`	δ	`\xi`	ξ	`\Gamma`	Γ
`\epsilon`	ε	`\rho`	ρ	`\Delta`	Δ
`\eta`	η	`\sigma`	σ	`\Theta`	Θ
`\theta`	θ	`\tau`	τ	`\Lambda`	Λ
`\kappa`	κ	`\phi`	φ	`\Phi`	Φ

Таблица 3.5. *Специальные символы*

Команда	Символ	Команда	Символ
`\leq`	≤	`\leftrightarrow`	↔
`\geq`	≥	`\leftarrow`	←

Таблица 3.5 *(окончание)*

Команда	Символ	Команда	Символ
\pm	±	\rightarrow	→
\propto	∞	\downarrow	↓
\partial	∂	\uparrow	↑

Текст в формате TeX можно использовать в качестве аргумента функций `title`, `xlabel`, `ylabel` при построении двумерных графиков и в тех же командах вместе с `zlabel` для трехмерных графиков.

В качестве упражнения создайте график, изображенный на рис. 3.35.

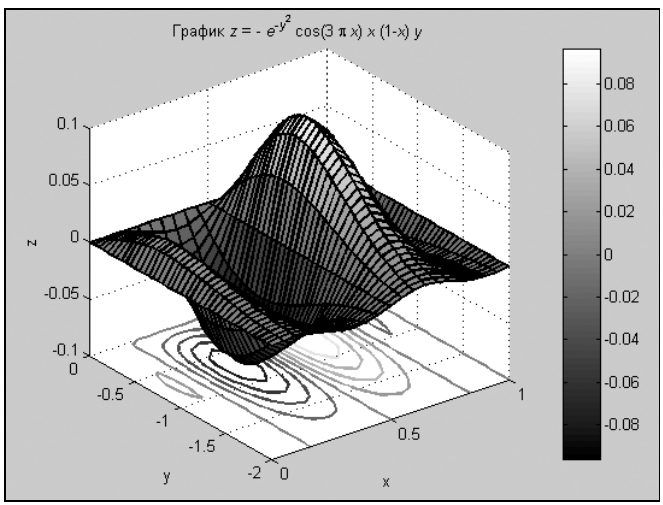


Рис. 3.35. Задание для самостоятельной работы

Ниже приведена последовательность команд, обеспечивающих требуемый результат.

```
>> [X, Y] = meshgrid(0:0.05:1, -2:0.05:0);
>> Z = -exp(-Y.^2).*cos(3*pi*X).*X.*(1-X).*Y;
>> surfc(X, Y, Z)
>> colormap(gray)
>> colorbar
>> title('График {\itz} = – {\ite}^{-{\ity}^{2}} cos(3 \pi {\itx})...
{\itx} (1-{\itx}) {\ity}')
```

```
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

Поворот графика, изменение точки обзора

Примеры, приведенные в предыдущих разделах, свидетельствуют о том, что при построении трехмерных поверхностей оси координат располагаются всегда одинаковым образом. Часть поверхности остается при этом скрытой. Для получения полной информации о поверхности ее желательно "осмотреть" со всех сторон. Положение наблюдателя за системой координат, изображенной на рис. 3.36, характеризуется двумя углами: *азимутом* (Az) и *углом возвышения* (El). Азимут отсчитывается от оси, противоположной *y*, а угол возвышения от плоскости *xy*. На рис. 3.36 положительные направления отсчета обозначены стрелками.

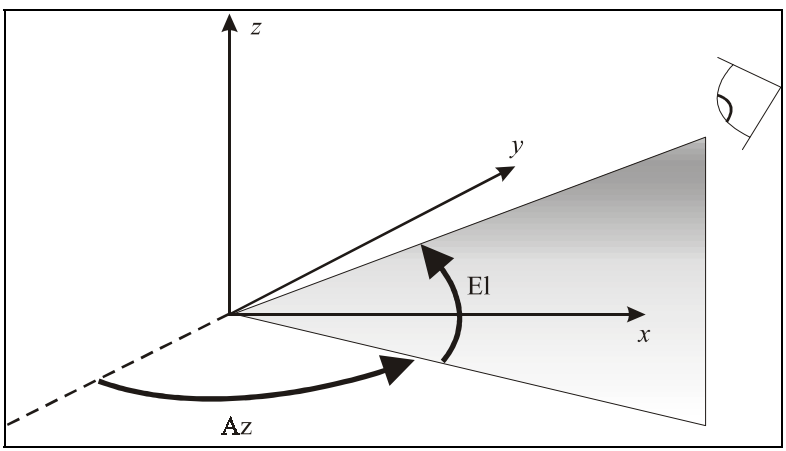


Рис. 3.36. Положение наблюдателя

Изменение положения наблюдателя относительно графика в MatLab осуществляет функция `view`. Аргументами `view` являются азимут и угол возвышения, отсчитываемые в градусах. По умолчанию Az = −37.5°, El = 30°. Для того чтобы узнать текущее положение наблюдателя, следует вызвать `view` с двумя выходными аргументами:

```
>> [Az, El] = view
Az =
  -37.5000
El =
    30
```

Положение наблюдателя задается входными аргументами `view`. Посмотрите, например, на поверхность, изображенную на рис. 3.35, возвышаясь над биссектрисой первого квадранта плоскости *xy* под углом 45°, для того, чтобы увидеть скрытую часть поверхности. Используйте команду `view(135,45)`, при этом получается график, приведенный на рис. 3.37.

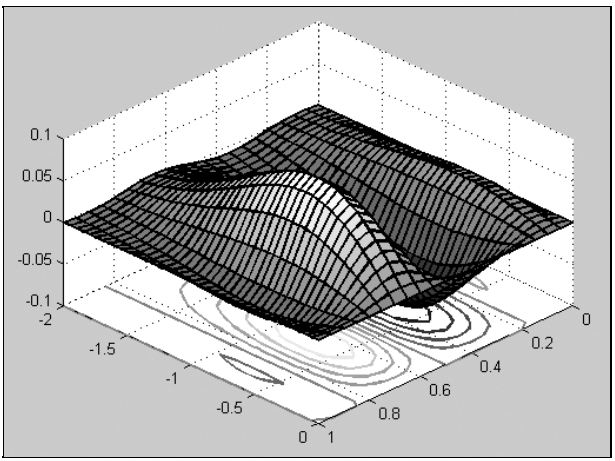


Рис. 3.37. График с точки зрения наблюдателя с Az = 135°, El = 45°

Разверните график поверхности так, как показано на рис. 3.38, чтобы посмотреть на него вдоль оси *y* со стороны плоскости *xz*.

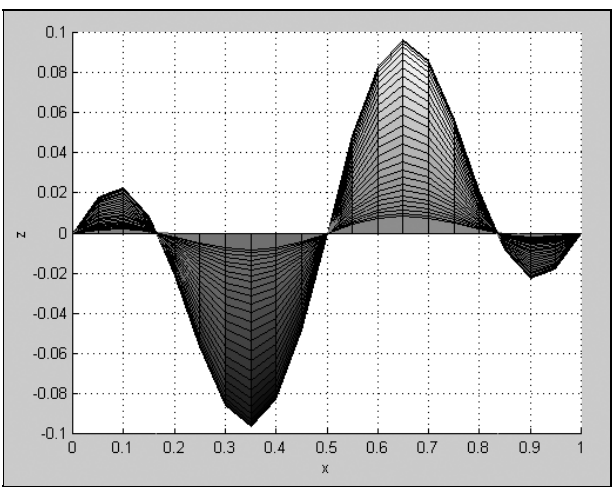


Рис. 3.38. Наблюдатель смотрит на график вдоль оси *y* со стороны плоскости *xz*

Несложно догадаться, что необходимо использовать `view(0,0)`. Потренируйтесь самостоятельно, наблюдая за графиком поверхности из различных точек.

Построение параметрически заданных поверхностей и линий

MatLab позволяет строить трехмерные линии, определенные формулами:

$$x = x(t),\ y = y(t), z = z(t),\ t \in [a,b]$$

и поверхности, задаваемые зависимостями

$$x = x(u,v),\ y = y(u,v),\ z = z(u,v),\ u \in [a,b],\ v \in [c,d].$$

Функция `plot3` визуализирует параметрически заданные линии, используя в качестве аргументов векторы, содержащие значения функций $x(t)$, $y(t)$ и $z(t)$, вычисленные для значений параметра t. Сначала следует сформировать вектор t, что проще всего сделать, используя заполнение с постоянным шагом при помощи двоеточия, а затем вычислить и записать в векторы соответствующие значения функции. Получите, например, график линии

$$x = e^{-|t-50|/50}\sin t, \qquad y = e^{-|t-50|/50}\cos t, \qquad z = t, \qquad t \in [0,100].$$

Используйте для этого следующие команды:

```
>> t = [0:0.1:100];
>> x = exp(abs(t-50)/50).*sin(t);
>> y = exp(abs(t-50)/50).*cos(t);
>> z = t;
>> plot3(x,y,z)
>> grid on
```

В результате выводится график, изображенный на рис. 3.39.

Также имеется возможность изменять тип и цвет линии, добавлять маркеры.

См. разд. "Изменение свойств линий" данной главы.

Например, `plot3(x,y,z,'r:')` рисует красную пунктирную линию.

Параметрически заданную поверхность можно построить при помощи любой из функций, предназначенных для отображения трехмерных графиков. Важно только правильно подготовить аргументы. Дело в том, что функции $x(u,v)$ и $y(u,v)$ могут быть многозначны, что надо учесть при создании матриц с информацией о расположении узлов сетки на области построения и матрицы, содержащей значения функции $z(u,v)$ в этих точках. Поставим задачу отобразить поверхность (конус), определенную зависимостями

$$x(u,v) = 0.3 \cdot u \cdot \cos v, \qquad y(u,v) = 0.3 \cdot u \cdot \sin v, \qquad z(u,v) = 0.6 \cdot u, \qquad u,v \in [-2\pi, 2\pi].$$

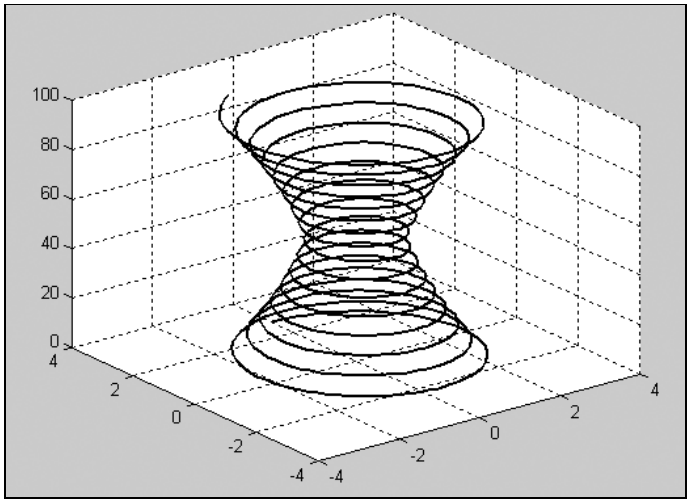


Рис. 3.39. Параметрически заданная линия (`plot3`)

Сгенерируйте при помощи двоеточия вектор-столбец и вектор-строку, содержащие значения параметров на заданном интервале (важно, что u — вектор-столбец, а v — вектор-строка!):

```
>> u = [-2*pi:0.1*pi:2*pi]';
>> v = [-2*pi:0.1*pi:2*pi];
```

Далее сформируйте матрицы X, Y, содержащие значения функций $x(u,v)$, $y(u,v)$ в точках, соответствующих значениям параметров при помощи *внешнего произведения векторов* (звездочка без точки):

```
>> X = 0.3*u*cos(v);
>> Y = 0.3*u*sin(v);
```

Матрица Z должна быть того же размера, что X и Y и, кроме того, она должна содержать значения, соответствующие значениям параметров. Если бы в функцию $z(u,v)$ входило произведение u и v, то матрицу Z можно было заполнить аналогично X и Y при помощи внешнего произведения. С другой стороны, функцию $z(u,v)$ можно представить в виде $z(u,v) = 0.6 \cdot u \cdot g(v)$, где $g(v) \equiv 1$. Поэтому для вычисления Z снова примените внешнее произведение на вектор-строку той же размерности, что V, состоящую из единиц:

```
>> Z = 0.6*u*ones(size(v));
```

Все требуемые матрицы созданы. Используйте теперь любую из описанных выше функций для построения трехмерных графиков.

Например, последовательность команд

```
>> surf(X, Y, Z)
>> colorbar
>> xlabel('\itx=0.3 \itu cos \itv')
>> ylabel('\ity=0.3 \itu sin \itv')
>> zlabel('\itz=0.6 \itu ')
```

приводит к графику, изображенному на рис. 3.40.

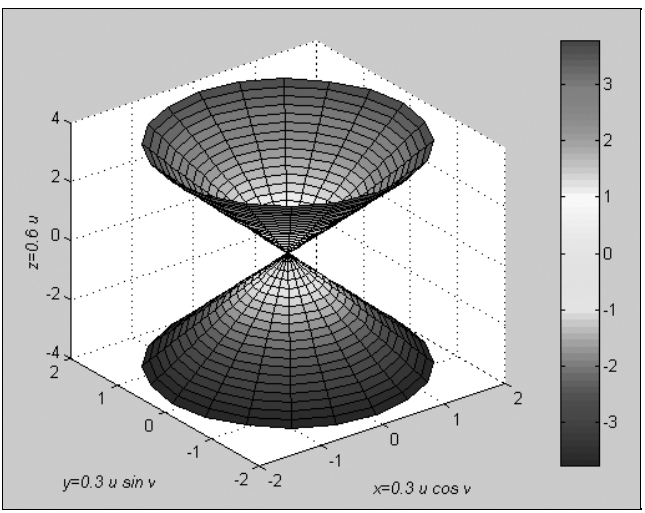


Рис. 3.40. График параметрически заданной поверхности

Постройте самостоятельно прозрачную каркасную поверхность эллипсоида, заданного соотношениями

$$x(u,v)=\cos u\cdot\cos v,\quad y(u,v)=0.7\cos u\cdot\sin v,\quad z(u,v)=0.8\cdot\sin u,\quad u,v\in[-2\pi,2\pi].$$

Требуемый результат позволяет получить следующая последовательность команд:

```
>> u = [-pi:0.1*pi:pi]';
>> v = [-pi:0.1*pi:pi];
>> X = cos(u)*cos(v);
>> Y = 0.9*cos(u)*sin(v);
>> Z = 0.8*sin(u)*ones(size(v));
>> mesh(X, Y, Z)
>> hidden off
```

Все способы построения трехмерных графиков, описанные выше, изменяют цвет поверхности в зависимости от значений функции, что приводит к поверхности, выглядящей не совсем естественно. В то же время MatLab позволяет получить очень наглядное изображение поверхности в трехмерном пространстве, освещенной с одной или нескольких сторон.

Построение освещенной поверхности

Предположим, что поверхность графика функции сделана из материала с определенными свойствами отражения и поглощения света и, кроме того, можно управлять расположением источника света. Эти две возможности вместе с поворотом графика позволяют получить естественно выглядящую поверхность, повернутую и освещенную под нужным углом. Для построения освещенной поверхности применяется функция `surfl`.

Постройте освещенную поверхность, задаваемую на прямоугольной области $x \in [-1,1]$, $y \in [0,1]$ формулой

$$z(x,y) = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot (1-x^2) \cdot y \cdot (1-y).$$

При использовании `surfl` удобно задавать цветовые палитры: `copper`, `bone`, `gray`, `pink`, в которых интенсивность цвета изменяется линейно. Для получения плавно изменяющихся оттенков следует использовать `shading interp`. Команды, приведенные ниже, приводят к получению требуемой освещенной поверхности, изображенной на рис. 3.41.

```
>> [X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
>> Z = 4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
>> surfl(X, Y, Z)
>> colormap('copper')
>> shading interp
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

По умолчанию источник света имеет азимут, больший на 45°, чем наблюдатель, и тот же угол возвышения. Дополнительным четвертым аргументом `surfl` может быть вектор-строка из двух элементов — азимута и угла возвышения источника света. Измените, например, азимут источника на $-90°$ по отношению к наблюдателю, а угол возвышения установите в ноль. Это можно проделать при помощи команд:

```
>> [Az, El] = view;
>> surfl(X, Y, Z, [Az-90, 0])
>> shading interp
```

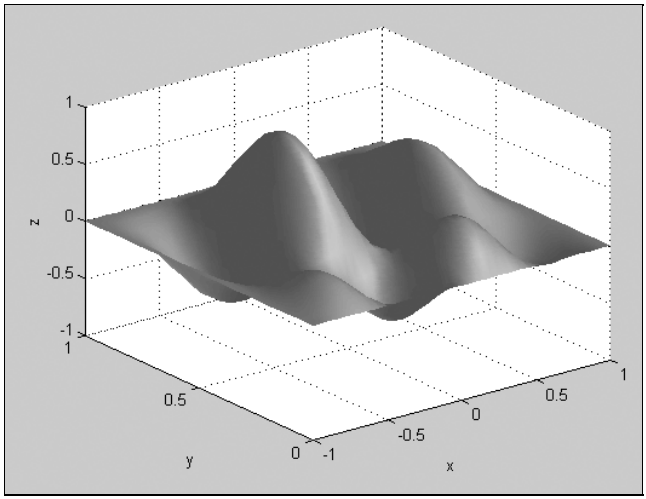


Рис. 3.41. Освещенная поверхность

В результате получается освещенная поверхность, изображенная на рис. 3.42.

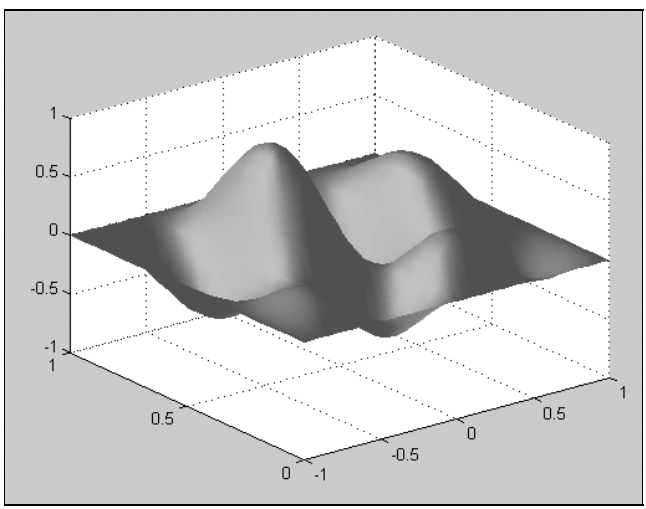


Рис. 3.42. Изменение положения источника света

Анимированные графики

При изучении движения точки на плоскости или в трехмерном пространстве полезно не только построить траекторию точки, но и следить за движе-

нием точки по траектории. MatLab предоставляет возможность получить анимированный график, на котором кружок, обозначающий точку, перемещается на плоскости или в пространстве, оставляя за собой след в виде линии — траектории движения. График похож на летящую комету с хвостом. Для построения анимированных графиков применяются функции `comet` и `comet3`. Постройте, например, траекторию движения точки в течение 10 секунд, координаты которой изменяются по закону

$$x(t) = \frac{\sin t}{t+1} \qquad y(t) = \frac{\cos t}{t+1}.$$

Действуйте точно так же, как при построении графика параметрически заданной функции, но для визуализации результата используйте `commet`:

```
>> t = [0:0.001:10];
>> x = sin(t)./(t+1);
>> y = cos(t)./(t+1);
>> comet(x, y)
```

При выполнении последней команды следите за тем, чтобы окно с графиком было поверх остальных окон. Окончательный вид траектории движения приведен на рис. 3.43.

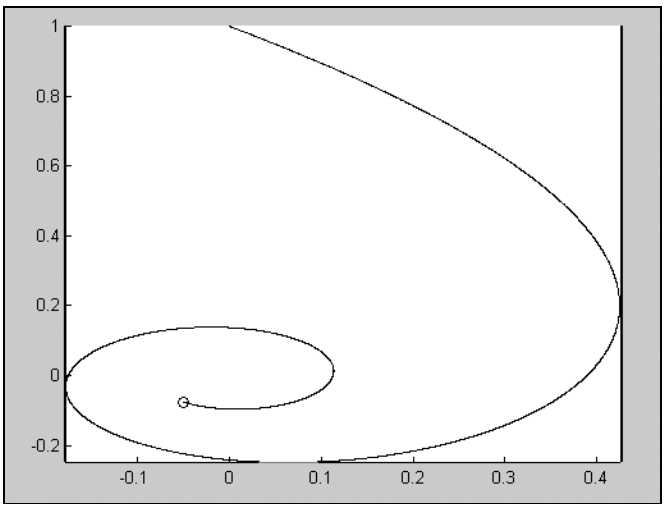


Рис. 3.43. Окончательный вид траектории движения (`commet`)

Скоростью движения кружка можно управлять, задавая различные шаги при автоматическом заполнении вектора, соответствующего времени. Использование `comet` с одним аргументом (вектором) приводит к построению динамически рисующегося графика значений элементов номера в зависимости

от их номеров. Функцию `comet` можно вызвать и с третьим дополнительным числовым параметром, который задает длину хвоста кометы. По умолчанию он равен 0.1. Обратите внимание, что при изменении размеров графического окна или при его минимизации и последующем восстановлении траектория движения пропадает. Это связано со способом, который применяет MatLab для построения графика.

Получите самостоятельно траекторию движения фиксированной точки на окружности, катящейся по прямой (циклоиду). Циклоида описывается параметрическими зависимостями $x(t) = t - \sin t,\ y(t) = 1 - \cos t$.

Для построения траектории точки, перемещающейся в пространстве, используется функция `comet3`. Пусть координаты точки в течение 100 секунд изменялись по следующему закону

$$x = e^{-|t-50|/50}\sin t, \qquad y = e^{-|t-50|/50}\cos t, \qquad z = t.$$

Отобразите траекторию движения точки применяя следующие команды:

```
>> t = [0:0.1:100];
>> x = exp(abs(t-50)/50).*sin(t);
>> y = exp(abs(t-50)/50).*cos(t);
>> z = t;
>> comet3(x, y, z)
```

Функцию `comet3` можно вызывать с четвертым числовым аргументом, который так же, как и в случае `comet` задает длину хвоста кометы.

Работа с несколькими графиками

Во всех примерах, приведенных в предыдущих разделах, графики выводились в специальное графическое окно с заголовком **Figure No. 1**. При следующем построении графика предыдущий пропадал, а новый выводился в то же самое окно. MatLab предоставляет следующие возможности работы с несколькими графиками:

- вывод каждого графика в свое окно;
- вывод нескольких графиков в одно окно (на одни координатные оси);
- отображение в пределах одного окна нескольких графиков, каждого на своих осях.

Вывод графиков в отдельные окна

Команда `figure`, определенная в MatLab, служит для создания пустого графического окна и отображения его на экране. Окно становится *текущим*, т. е. все последующие графические функции будут осуществлять построение

графиков в этом окне. Для получения нового графического окна следует снова использовать `figure`. Например, последовательность команд

```
>> [X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
>> Z = 4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
>> figure
>> mesh(X, Y, Z)
>> figure
>> surfl(X, Y, Z)
```

приводит к появлению на экране двух графических окон: **Figure No. 1**, содержащего каркасную поверхность, и **Figure No. 2** с освещенной поверхностью. Окно **Figure No. 2** является текущим, т. к. было создано последним. Команды, набираемые далее, например

```
>> colormap('copper')
>> shading interp
```

приведут к изменениям именно в этом окне. Для того чтобы сделать графическое окно **Figure No. 1** текущим, следует щелкнуть на нем мышкой, вернуться в рабочую среду MatLab и продолжать ввод команд. Команды повлекут изменения в окне **Figure No. 1**. Для очистки всего текущего окна используется команда `clf` (сокращение от clear figure), а для того, чтобы убрать только график, но оставить оси, заголовок и названия осей, следует применить `cla` (сокращение от clear axes).

Вышеописанным способом можно получить сколько угодно графических окон и вывести в них графики различных функций или визуализировать векторные и матричные данные. Однако для изменения того или иного графика придется искать его окно на экране и делать его текущим при помощи щелчка мыши. Есть более универсальный и удобный способ работы с несколькими окнами. При создании каждого нового графического окна при помощи `figure` следует вызвать ее с выходным аргументом. Этот аргумент называется в MatLab *указателем* на графическое окно. Значением выходного аргумента является число, совпадающее с номером графического окна. Для того чтобы сделать графическое окно текущим, следует вызвать `figure`, применив в качестве входного аргумента указатель на требуемое графическое окно. Разберите использование указателей на следующем примере. Требуется создать два графических окна, построить в них графики функций $f = \sin x$ и $g = \ln x$, а затем оформить их — дать заголовки и нанести сетку на второй график (см. рис. 3.44). Последовательность команд, приведенная ниже, позволяет получить желаемый результат.

```
>> sinGr = figure;
>> lnGr = figure;
>> x = [0.1:0.05:10];
```

```
>> f = sin(x);
>> g = log(x);
>> figure(sinGr)
>> plot(x, f)
>> figure(lnGr)
>> plot(x, g)
>> figure(sinGr)
>> title('\itf=sin\itx')
>> figure(lnGr)
>> title('\itg=ln\itx')
>> grid on
```

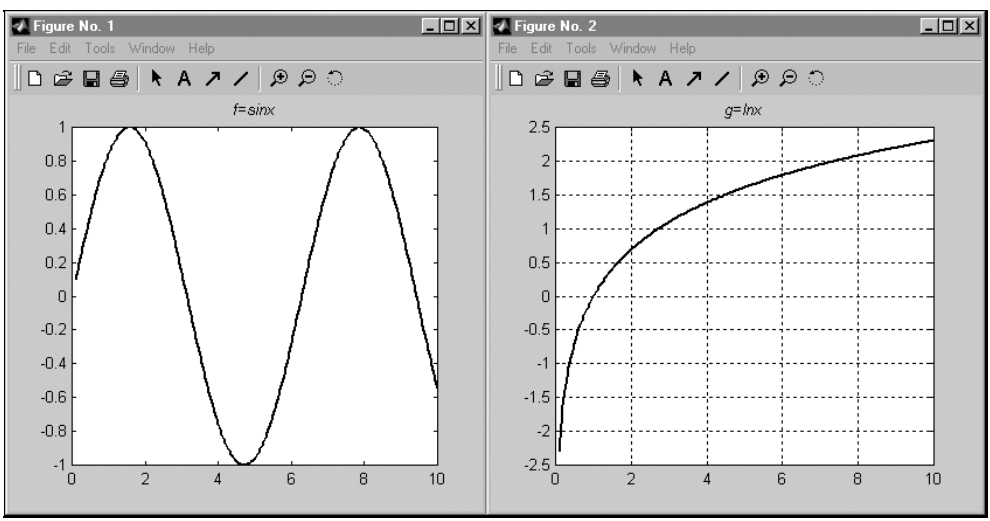


Рис. 3.44. Вывод графиков в разные окна

Для того чтобы очистить графическое окно с указателем `lnGr`, следует использовать команду `clf(lnGr)`. Удаление графика из первого окна, на которое указывает `sinGr`, производится при помощи команды `cla(sinGr)`.

Вывод нескольких графиков на одни оси

Возможность отображения нескольких графиков функций одной переменной на одних осях использовалась при изучении функций `plot`, `plotyy`, `semilogx`, `semilogy`, `loglog`. Они позволяют выводить графики нескольких функций, задавая соответствующие векторные аргументы парами, например `plot(x,f,x,g)`. Однако, при построении трехмерных графиков или различ-

ных типов графиков, объединять их на одних осях не было возможности. Для объединения графиков предназначена команда `hold on`, которую нужно задать перед построением следующего графика. В следующем примере выводится пересечение плоскости и конуса, заданного параметрически. Результат приведен на рис. 3.45.

```
>> u = [-2*pi:0.1*pi:2*pi]';
>> v = [-2*pi:0.1*pi:2*pi];
>> X = 0.3*u*cos(v);
>> Y = 0.3*u*sin(v);
>> Z = 0.6*u*ones(size(v));
>> surf(X, Y, Z)
>> [X, Y] = meshgrid(-2:0.1:2);
>> Z = 0.5*X+0.4*Y;
>> hold on
>> mesh(X, Y, Z)
>> hidden off
```

Команда `hidden off` применена для того, чтобы показать часть конуса, находящуюся под плоскостью.

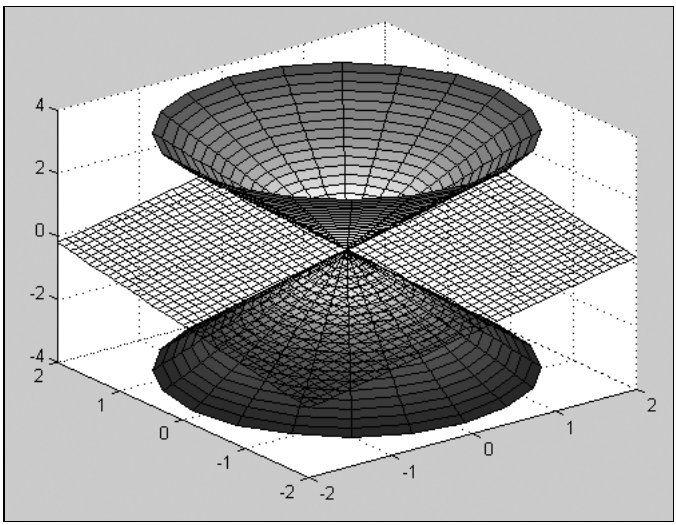


Рис. 3.45. Пересечение плоскости и конуса

Обратите внимание, что `hold on` распространяется на все последующие выводы графиков в текущее окно. Для размещения графиков в новых окнах следует выполнить команду `hold off`. Команда `hold on` может применяться

и для расположения нескольких графиков функций одной переменной, например,

```
>> plot(x, f, x, g)
```

эквивалентно последовательности

```
>> plot(x, f)
>> hold on
>> plot(x, g)
```

Несколько графиков в одном графическом окне

MatLab позволяет разбить графическое окно на несколько подграфиков со своими осями. Для этого служит команда `subplot`, которая располагает подграфики в виде матрицы и используется с тремя параметрами: `subplot(i,j,n)`. Здесь `i` и `j` — число подграфиков по вертикали и горизонтали, а `n` — номер подграфика, который надо сделать текущим. Номер отсчитывается от левого верхнего угла построчно. Например, команда `subplot(3,2,4)` предполагает наличие шести подграфиков и делает четвертый текущим, что схематично изображено на рис. 3.46.

После выполнения `subplot(3,2,4)` все графические функции будут осуществлять вывод именно в этот подграфик.

В качестве завершающего упражнения постройте графики функции

$$z(x,y) = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot (1 - x^2) \cdot y \cdot (1 - y)$$

на прямоугольной области определения $x \in [-1,1]$, $y \in [0,1]$ всеми известными способами, размещая их на отдельных подграфиках. Названия команд, применяемых для построения графиков, включите в заголовки подграфиков.

```
>> [X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
>> Z = 4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
>> subplot(3, 2, 1)
>> mesh(X, Y, Z)
>> title('mesh')
>> subplot(3, 2, 2)
>> surf(X, Y, Z)
>> title('surf')
>> subplot(3, 2, 3)
>> meshc(X, Y, Z)
```

```
>> title('meshc')>> subplot(3, 2, 4)
>> surfc(X, Y, Z)
>> title('surfc')
>> subplot(3, 2, 5)
>> contour3(X, Y, Z)
>> title('contour3')
>> subplot(3, 2, 6)
>> surfl(X, Y, Z)>> shading interp
>> title('surfl')
>> colormap(gray)
```

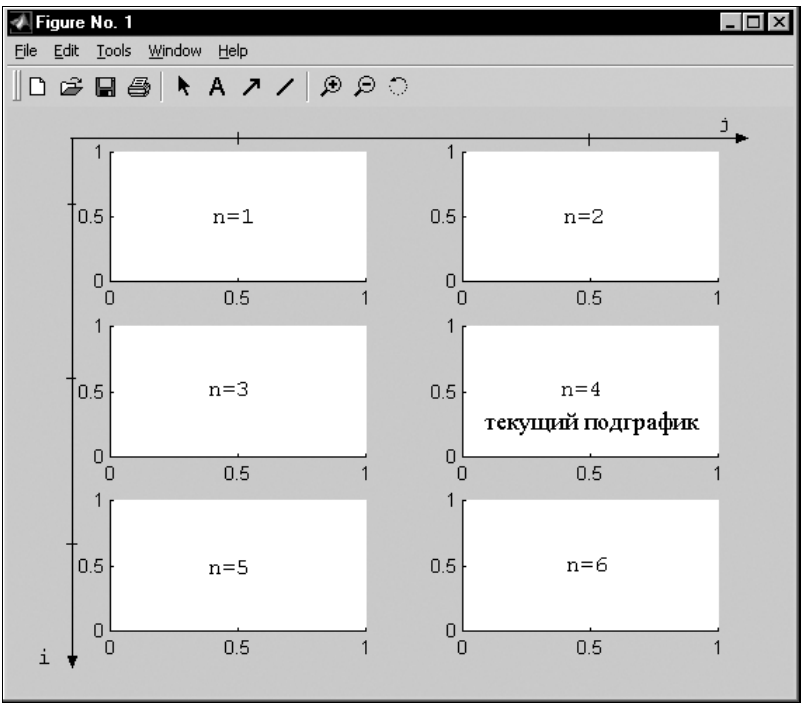


Рис. 3.46. Схема расположения подграфиков при выполнении команды `subplot (3, 2, 4)`

В результате получается графическое окно, изображенное на рис. 3.47, которое содержит шесть подграфиков, наглядно демонстрирующих способы построения трехмерных графиков в MatLab.

Обратите внимание, что последняя команда `colormap(gray)` изменяет палитру всего графического окна, а не подграфиков по отдельности.

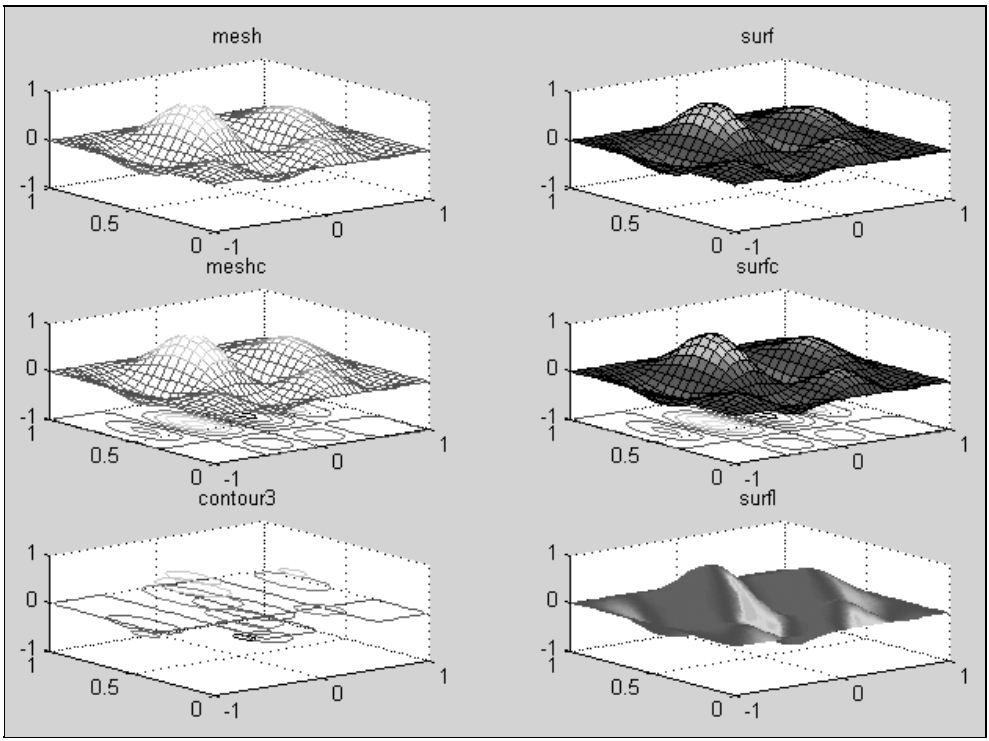


Рис. 3.47. Использование подграфиков (`subplot`)

Глава 4



Редактирование графиков

MatLab предоставляет удобное интерактивное средство для изменения вида графика, нанесения дополнительной информации, подготовки его к печати, сохранения и экспорта в различные графические форматы. Для редактирования графиков предназначены меню и панель инструментов графического окна. Использование редактора оправдано, когда требуется подготовить небольшое число графиков для печати, или для вставки в какой-либо документ. Однако при написании собственных программ в среде MatLab необходим доступ к всевозможным свойствам графических объектов, который осуществляется при помощи управляемой графики (Handle Graphics). Управляемой графике посвящена отдельная глава. Интерактивное редактирование графиков поможет понять структуру объектов MatLab и многообразие их свойств. Возможности редактирования в MatLab 6.*x* шире, чем в версии 5.3, кроме того, в этих версиях MatLab несколько отличаются и способы редактирования графиков.

Редактирование графиков в MatLab 5.3

Выведите при помощи команды `plot` на экран окно, содержащее графики двух функций, например $f = \sin(x)$ и $g = \cos(x)$. Для того чтобы начать редактирование, выберите в меню **Tools** графического окна пункт **Enable Plot Editing**. Теперь элементы графического окна (оси и линии графиков) можно выделять при помощи щелчка мыши на них, выделенный элемент помечается квадратными черными маркерами.

Изменение свойств осей, подписи, заголовок

Выделите оси графика и выберите в меню **Tools** пункт **Axes Properties**, откроется диалоговое окно **Edit Axes Properties**, изображенное на рис. 4.1. Используя это диалоговое окно, можно добавить заголовок на график, набрав его в поле **Title**, сделать подписи к осям, заполнив соответствующие поля **Label**. Результат будет тот же, что при использовании команд `title`, `xlabel`, `ylabel`. Учтите, что текст не нужно помещать в апострофы, в отли-

чие от аргументов соответствующих команд. При наборе формул допускается использование формата TeX.

См. разд. "Оформление графика" главы 3.

Пределы изменения переменных по каждой из осей координат выбираются автоматически, однако пользователь может изменить их значения, установив в поле **Limits** флажки **Manual** и введя в соответствующие поля начальное и конечное значения переменных. В поле **Tick Step** аналогично определяется шаг разметки по осям абсцисс и ординат. Изменение масштаба и направления осей достигается установкой переключателей в поле **Scale**. Возможны два вида масштабирования: линейный и логарифмический, им соответствуют переключатели **Linear** и **Log**. Для каждого из способов масштабирования выбирается либо обычное направление оси (переключатель **Normal**), либо обратное (переключатель **Reverse**). Нанесение сетки по каждой из осей осуществляется установкой соответствующего флага **Grid**.

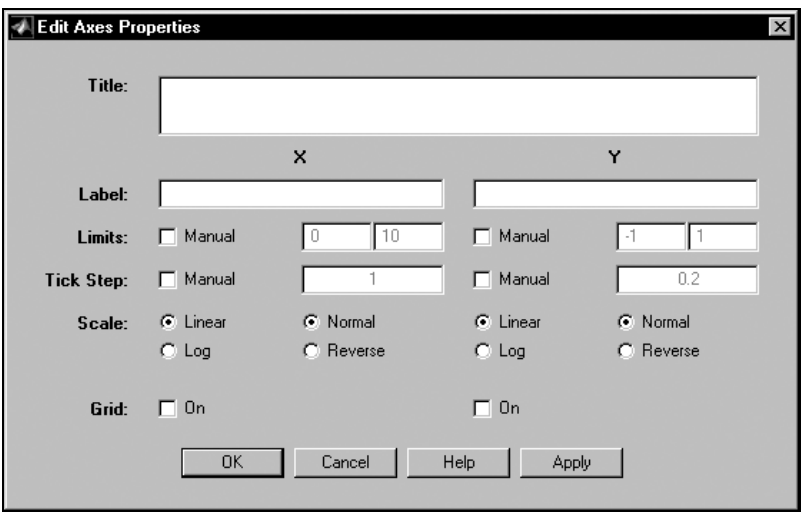


Рис. 4.1. Диалоговое окно **Edit Axes Properties**

Замечание

Подписи к осям, пределы, разметка, масштаб направление и сетка устанавливаются независимо для каждой координатной оси. Координаты линий сетки совпадают с разметкой осей.

Для изменения размеров и положения осей необходимо выделить оси и выбрать в меню **Tools** пункт **Unlock Axes Position**. Теперь для изменения размеров осей требуется навести курсор мыши на один из черных квадратиков

рамки выделения, щелкнуть на нем левой кнопкой мыши и, удерживая кнопку в нажатом положении (курсор меняет форму на двухстороннюю стрелку), переместить мышь, изменяя размеры графика. Для перемещения графика в пределах графического окна следует поместить курсор мыши внутри осей, щелкнуть на нем левой кнопкой мыши и, удерживая кнопку в нажатом положении (курсор меняет форму на четырехстороннюю стрелку), выбрать нужное положение графика. После проделанных изменений лучше выбрать в меню **Tools** пункт **Lock Axes Position** для предотвращения случайного изменения размеров и положения графика.

Увеличение или уменьшение масштаба просмотра (не построения!) производится выбором пунктов **Zoom In** или **Zoom Out** меню **Tools** и последующим щелчком левой кнопкой мыши по графику в месте изменения масштаба.

При редактировании трехмерных графиков диалоговое окно **Edit Axes Properties** содержит дополнительные поля для свойств вертикальной оси. Кроме того, можно поворачивать график при помощи мыши для просмотра его с различных сторон. Для этого следует выбрать в меню **Tools** пункт **Rotate 3D**, навести курсор мыши на график и, удерживая нажатой левую кнопку мыши, вращать его, добиваясь требуемого вида.

Свойства линий

Для изменения свойств линии графика функции выделите ее щелчком мыши и выберите в меню **Tools** пункт **Line Properties**. Открывается диалоговое окно **Edit Line Properties**, изображенное на рис. 4.2.

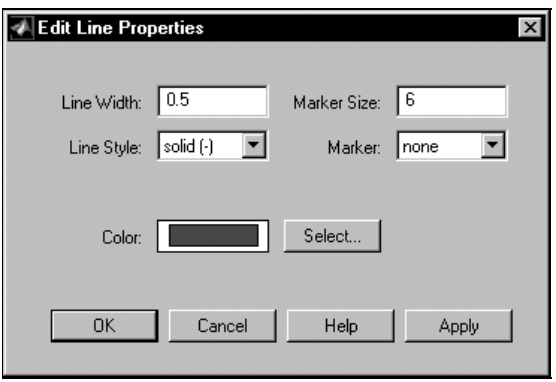


Рис. 4.2. Диалоговое окно **Edit Line Properties**

Диалоговое окно **Edit Line Properties** позволяет:

- установить ширину линии в поле **Line Width**;
- выбрать стиль — **Line Style**;

- определить размер и тип маркера в поле **Marker Size** и раскрывающемся списке **Marker**;
- задать цвет линии, нажав кнопку **Select** и выбрав цвет в появляющемся диалоговом окне **Color**.

Внесенные изменения можно применить, нажав кнопку **Apply**, при этом вид графика изменится, а диалоговое окно **Edit Line Properties** не закроется, что позволит вносить новые изменения. Для редактирования другой линии графика следует закрыть диалоговое окно **Edit Line Properties** при помощи кнопки **OK**, выделить линию при помощи щелчка мыши и снова выбрать в меню **Tools** пункт **Line Properties**.

Дополнительные элементы оформления

Размещение стрелок, линий, текста и новых осей производится при помощи пункта **Add** меню **Tools**. Для добавления стрелок или линий следует выбрать подпункт **Arrow** или **Line** и нарисовать их в области графического окна при помощи мыши, удерживая левую кнопку. Помещение текста осуществляется выбором подпункта **Text**. После щелчка мыши в области графического окна появляется заготовка, в которой можно набирать текст, причем в несколько строк. Вставка греческих букв и изменение шрифта возможны в формате TeX.

См. разд. "Оформление графика" главы 3.

Окончание набора текста завершается щелчком мыши вне текстовой области. Переход в режим редактирования производится двойным щелчком мыши по текстовой области. Изменить свойства текста можно из меню. Следует выделить текстовую область щелчком мыши и выбрать в меню **Tool** пункт **Text Edit**. Появится диалоговое окно **Edit Font Properties**, пользуясь которым легко подобрать нужный шрифт.

Добавление легенды на график осуществляется при помощи пункта **Show Legend** меню **Tools**. В правом верхнем углу появится заготовка с образцами линий и подписями к ним: **data1**, **data2**. Для изменения подписи надо перейти в режим редактирования двойным щелчком мыши по тексту подписи в области легенды и внести соответствующие поправки. Убрать легенду можно, выбрав пункт **Hide Legend** в меню **Tools**. Все дополнительные элементы оформления перемещаются внутри графического окна точно так же, как сам график.

Замечание

Часть пунктов меню дублируется кнопками на панели инструментов.

Быстрый переход к свойствам выделенного объекта осуществляется из всплывающего меню, вызываемого правой кнопкой мыши или при помощи двойного щелчка мыши, когда курсор находится на объекте.

В качестве упражнения создайте график, изображенный на рис. 4.3.

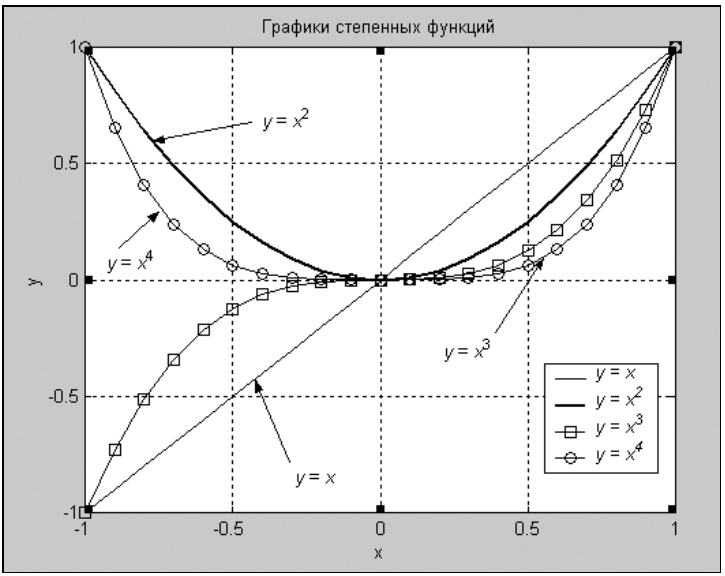


Рис. 4.3. Графики степенных функций

Сохранение, экспорт и печать графиков

MatLab использует расширение fig для файлов, содержащих графическое окно. Для сохранения графического окна используются пункты **Save** или **Save as** меню **File**. Открыть графическое окно в текущем и следующих сеансах работы с MatLab можно при помощи пункта **Open** меню **File** какого-либо графического окна или рабочей среды MatLab (при открытии из рабочей среды требуется выбрать fig в фильтре расширений диалогового окна **Open**).

Экспорт графики из MatLab возможен в различные графические форматы, в частности: EPS, AI, BMP, GIF, TIFF, JPEG. При экспорте в графический файл записывается только область графического окна без меню и панели инструментов. Для экспортирования предназначен пункт **Export** меню **File** графического окна.

Перед печатью графика следует установить свойства страницы при помощи пункта **Page Setup** меню **File**. Выбор этого пункта приводит к появлению диалогового окна **Page Setup**, изображенного на рис. 4.4.

Панели **Orientation** и **Color** не требуют комментариев. Разберем подробно возможности печати, специфичные для MatLab. Как правило, график при печати имеет другие размеры по сравнению с отображаемым на экране. Элементы управления, размещенные на панели **Size and Position**, предназна-

чены для изменения размеров графика при печати. Для сохранения размера графика, отображаемого на экране, следует установить переключатель **Match Figure Screen Size**. Можно расположить график на листе и изменить его размеры вручную. Для этого следует установить переключатель **Set Manual Position and Size** и в расположенном ниже окне с макетом страницы при помощи мыши выбрать желаемые размер и положение графика на листе.

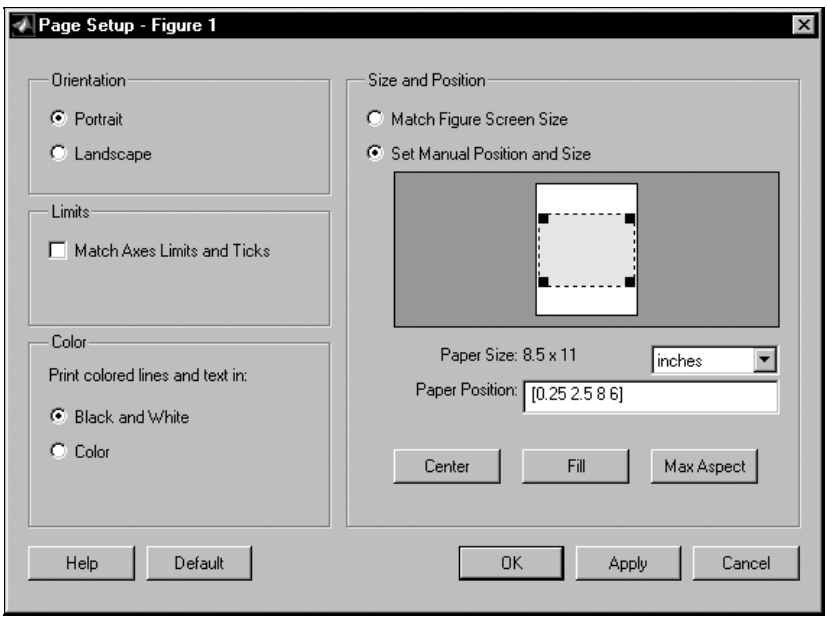


Рис. 4.4. Диалоговое окно **Page Setup**

Для размещения графика в центре используется кнопка **Center**, для растяжения графика с изменением на весь лист — **Fill**. Максимально возможное увеличение графика с сохранением пропорций осуществляется при нажатии на кнопку **Max Aspect**. Точное определение границ графика производится при помощи задания его положения на листе в строке ввода **Paper Position** в формате [граница слева, граница снизу, ширина, высота]. Раскрывающийся список **Paper Size** позволяет установить единицы измерения. Выбор опции **normalized** означает, что у левого нижнего угла страницы координаты (0, 0), а у правого верхнего — (1, 1). При печати можно автоматически изменить разметку осей и их пределы для получения хорошего вида графика. Для этого следует установить флаг **Match Axes Limits and Ticks** на панели **Limits**.

Более полную информацию можно получить, пользуясь справочной системой Plot Editor Help, доступ к которой осуществляется из пункта **Editing Plots** меню **Help** графического окна. Редактор графиков MatLab 5.3 позволя-

ет изменять только небольшую часть свойств объектов, размещенных в графическом окне. Для получения доступа ко всем свойствам следует использовать *редактор свойств*.

Редактирование графиков в MatLab 5.3 при помощи редактора свойств

В этом разделе рассмотрено редактирование графиков при помощи редактора свойств. Работа с редактором свойств подробно описана в части, посвященной разработке в MatLab собственных приложений с графическим интерфейсом пользователя. Сейчас нас будут интересовать только некоторые возможности, предоставляемые редактором свойств для интерактивного редактирования графиков.

Структура объектов в MatLab

MatLab 5.3 является *объектно-ориентированной системой*, причем все *объекты* расположены в определенной иерархической последовательности. Для редактирования графиков достаточно понимать, что на каждом графическом окне (figure) может быть расположен один или несколько графиков со своими *осями* (axes), созданные, например, при помощи команды `subplot`. Каждый график может содержать одну или несколько *линий* (line) или *поверхностей* (surface) и *текстовые объекты* (text). Вышеописанная структура иерархии объектов схематично приведена на рис. 4.5 (на самом деле она намного сложнее).

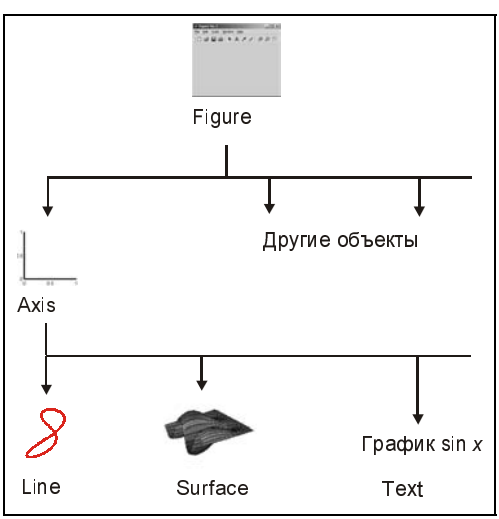


Рис. 4.5. Структура объектов MatLab (упрощенный вариант)

Выведите в графическое окно два графика. На одном постройте $\sin x$ и $\cos x$ на отрезке $[0, 6\pi]$, а на другом функцию $e^{-x^2-y^2} \cdot (x-1)^2 \cdot \sin 2\pi y$ на квадрате $x, y \in [-1, 1]$. Если вы изучили предыдущие разделы, то построение графиков не представит большого труда. Ниже приведена последовательность команд, приводящая к требуемому виду графического окна **Figure No. 1**, изображенного на рис. 4.6.

```
>> x = [0:0.1:5];
>> f = sin(pi*x);
>> g = cos(pi*x);
>> subplot(2,1,1)
>> plot(x, f, x, g)
>> [X, Y] = meshgrid(-1:0.05:1, -1:0.05:1);
>> Z = exp(-X.^2-Y.^2).*(X-1).^2.*sin(2*pi*Y);
>> subplot(2, 1, 2)
>> surf(X, Y, Z)
```

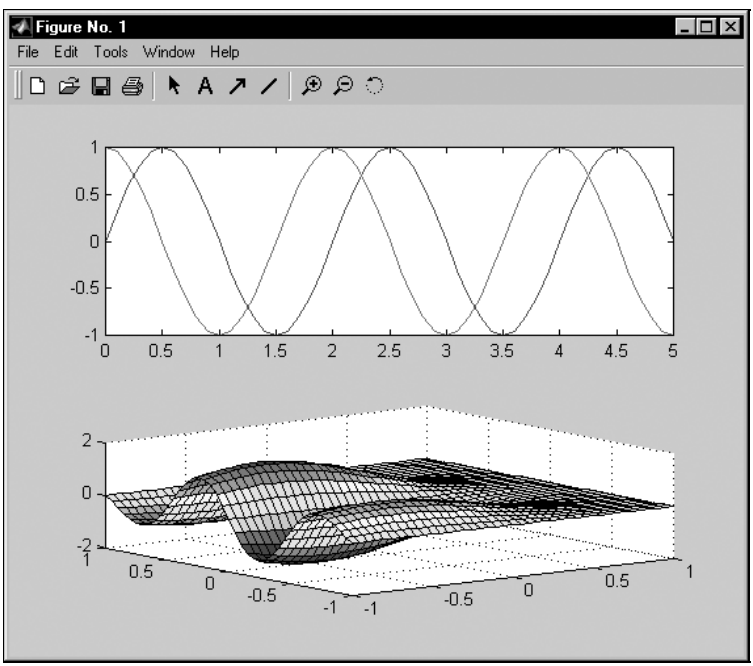


Рис. 4.6. Графическое окно для изучения возможностей редактора свойств

Редактор свойств позволяет получить доступ к свойствам всех объектов, расположенных в графическом окне. Для запуска редактора свойств выбери-

те пункт **Property Editor** меню **File** графического окна. На экране появляется диалоговое окно редактора свойств **Graphics Property Editor**, приведенное на рис. 4.7.

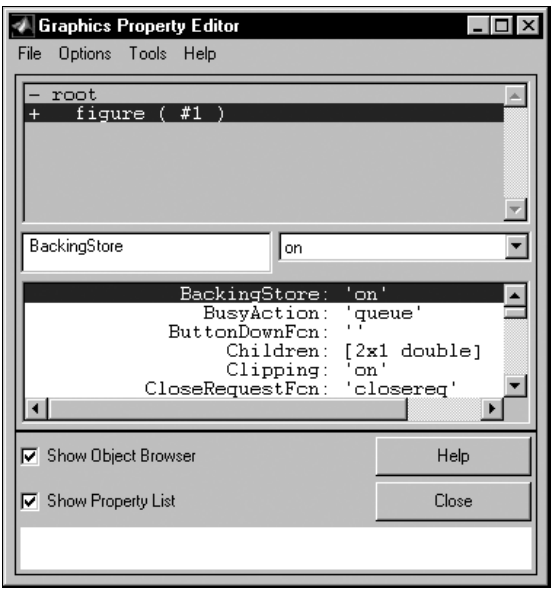


Рис. 4.7. Диалоговое окно редактора свойств MatLab 5.3

Поле со списком вверху диалогового окна редактора свойств **Graphics Property Editor** называется Object Browser (навигатор объектов), оно содержит иерархическую структуру созданных объектов. Если слева от объекта расположен знак плюс, то двойным щелчком мыши можно раскрыть список объектов, принадлежащих данному объекту. Раскройте объект `figure ( #1 )`, соответствующий графическому окну **Figure No.1**, и убедитесь, что он содержит два объекта `axes`, каждый со своими текстовыми областями, линиями или поверхностями. Интересующая нас часть иерархической структуры выглядит следующим образом:

```
-    figure        ( #1 )
-      axes
         text      (xlabel)
         text      (ylabel)
         text      (zlabel)
         text      (title)
         surface
```

```
-       axes
          text      (xlabel)
          text      (ylabel)
          text      (zlabel)
          text      (title)
          line
          line
```

Выбор любого из объектов в навигаторе приводит к отображению всех его свойств в списке Property List (список свойств), расположенном под навигатором. Левая колонка списка содержит названия свойств, правая — их значения. Список свойств на первый взгляд кажется очень большим, причем непонятно, что значит то или иное свойство объекта. Начнем изучение свойств.

Замечание

Для того чтобы навигатор объектов и список свойств отображались в редакторе свойств, должны быть установлены флаги **Show Object Browser** и **Show Property List**.

Установка свойств объектов

Заголовок, подписи осей

Выберите в навигаторе объект `text (title)` осей трехмерного графика (эти оси содержат `surface`, в отличие от осей двумерного графика, которому принадлежат два объекта `lines`). В списке свойств найдите и выделите свойство `String`. В строках ввода, расположенных между навигатором объектов и списком свойств, слева появилось название свойства, а справа — его значение. В правой строке между апострофами добавьте текст "График поверхности" (кавычки набирать не нужно). Нажмите клавишу <Enter> и переключитесь на графическое окно. Обратите внимание, что на нижний график добавился заголовок. В дальнейшем будем просто говорить, что требуется установить значение свойству объекта. Например, сейчас свойству `String` объекта `text (title)` установлено значение `'График поверхности'`. При работе с редактором свойств расположите его так, чтобы видеть все изменения, происходящие в графическом окне.

Установите самостоятельно для объектов, принадлежащих осям с графиком поверхности:

- значение `'ось x'` свойству `String` объекта `text (xlabel)`;
- значение `'ось y'` свойству `String` объекта `text (ylabel)`;
- значение `'ось z'` свойству `String` объекта `text (zlabel)`.

Аналогичным образом подпишите оси *x* и *y* двумерного графика и дайте ему заголовок. Учтите, что значение свойства `String` может содержать текст в формате TeX.

Предупреждение

Если в течение сеанса работы определена какая-либо переменная, например массив `x` и вы устанавливаете свойство `String` текстового объекта в редакторе свойств равным `'x'`, то в результате выведется не символ x, а его значение. Для того чтобы использовать имя уже определенной переменной для подписи осей, следует применить команду TeX, задающую шрифт, т. е. набрать в поле значения свойства `'{\itx}'`, или `'{\rmx}'`, `'{\bfx}'`.

Редактор свойств позволяет устанавливать все свойства объекта, определенные в MatLab, в частности для текста можно изменить:

- `FontName` — название шрифта;
- `FontSize` — размер шрифта;
- `FontUnits` — единицы измерения размера шрифта;
- `FontWeight` — тип шрифта (тонкий, обычный, полужирный, жирный);
- `FontAngle` — наклон шрифта;
- `Color` — цвет.

Поэкспериментируйте с установкой этих свойств. Часть свойств, например, `FontName`, `FontSize` приходится вводить вручную; `FontAngle`, `FontWeight` выбираются из раскрывающегося списка; для выбора цвета есть три возможности:

- задание цвета в палитре RGB тремя числами в квадратных скобках;
- использование определенных в MatLab названий `white`, `black`, и т. д.;
- выбор цвета из диалогового окна **SetColor**, которое открывается при помощи кнопки, расположенной справа от поля со значением свойства.

Свойства линий и поверхностей

Редактор свойств предоставляет больше возможностей для изменения свойств линий, чем редактор графиков. Перейдите теперь к свойствам какой-нибудь из линий верхнего графика, изображенного на рис. 4.6, выбрав объект `line` в навигаторе объектов редактора свойств. Выбранная линия выделится в графическом окне. Изменяйте значения следующих свойств и следите за состоянием графического окна:

- `Color` — цвет линии;
- `LineStyle` — стиль линии (сплошная, штриховая, пунктирная, штрихпунктирная, невидимая);

- `LineWidth` — толщина линии;
- `Marker` — тип маркера (крестик, кружок, звездочка и т. д.);
- `MarkerEdgeColor` — цвет границы маркера;
- `MarkerFaceColor` — цвет внутренности маркера;
- `MarkerSize` — размер маркера.

Все вышеперечисленные свойства, кроме `Color`, есть у поверхностей; убедитесь в этом, перейдя к свойствам поверхности. Они соответствуют линиям сетки поверхности, которая может содержать маркеры в вершинах. При изменении вида поверхностей могут понадобиться свойства, приведенные в следующем списке с пояснениями их значений.

- `EdgeColor` — цвет границ ячеек сетки, возможные значения:
  - цвет в RGB, или одно из определенных в MatLab названий цвета (`'black'`, `'white'`,);
  - `'none'`, границы не рисуются;
  - `'flat'`, цвет границ изменяется в зависимости от значения функции, образующей поверхность, причем цвет каждой границы постоянен в ячейке сетки;
  - `'interp'`, плавное изменение цвета границ.
- `EdgeLighting` — алгоритм, используемый для освещения границ ячеек сетки, возможные значения:
  - `'none'`, свет не влияет на освещенность границ;
  - `'flat'`, освещенность границ постоянна в пределах ячейки;
  - `'gouraud'`, линейное изменение освещенности границы в пределах ячейки;
  - `'prong'`, плавное освещение границ ячейки.
- `FaceColor` — цвет внутренности ячейки сетки (значения свойства аналогичны `EdgeColor`).
- `FaceLighting` — алгоритм, используемый для освещения ячеек сетки (значения свойства аналогичны `EdgeLighting`).
- `MeshStyle` — рисуемые линии сетки, возможные значения:
  - `'both'`, сетка состоит из пересекающихся линий;
  - `'row'`, сетка состоит из линий, параллельных оси *x*, что соответствует строкам матрицы, содержащей значения функций в узлах сетки на плоскости *xy*;
  - `'column'`, сетка состоит из линий, параллельных оси *y*, что соответствует столбцам матрицы, содержащей значения функций в узлах сетки на плоскости *xy*.

Свойства осей

Список свойств осей в редакторе свойств не зависит от того, двумерный или трехмерный график на них построен. Для осей, содержащих двумерный график, свойства, соответствующие вертикальной оси, повороту графика, изменению точки обзора или освещению, устанавливать нет необходимости. При редактировании графика при помощи редактора свойств оказываются полезными следующие свойства осей.

- `Box` — заключение осей в прямоугольник (для двумерных графиков) или параллелепипед (для трехмерных графиков), возможные значения:
  - `'on'`, заключать;
  - `'off'`, не заключать.
- `Color` — цвет фона графика, возможные значения:
  - `'none'`, цвет фона графика совпадает с цветом графического окна;
  - цвет в RGB, или одно из определенных в MatLab названий цвета (`'black'`, `'white'`,);
- `FontAngle`, `FontName`, `FontSize`, `FontUnits`, `FontWeight` — свойства шрифта разметки осей.
- `GridLineStyle` — тип линий сетки.
- `LineWidth` — толщина осей.
- `Projection` — тип проекции осей трехмерных графиков на экран, возможные значения:
  - `'orthographic'`, ортогональная проекция;
  - `'perspective'`, отображение осей в перспективе.
- `TickDir` — расположение отрезков разметки осей, возможные значения:
  - `'in'`, внутри графика;
  - `'out'`, снаружи графика.
- `TickLength` — длина отрезков разметки осей, значение свойства задается вектором, первый элемент которого определяет длину для двумерных графиков, а второй — для трехмерных.
- `XAxisLocation` — расположение оси *x*, возможные значения:
  - `'top'`, вверху;
  - `'bottom'`, внизу.
- `YAxisLocation` — расположение оси *y*, возможные значения:
  - `'right'`, справа;
  - `'left'`, слева.

- `XColor`, `YColor`, `ZColor` — цвет осей.
- `XDir`, `YDir`, `ZDir` — направление осей, возможные значения:
  - `'normal'`, слева направо для оси *x*, снизу вверх для оси *y* на двумерном графике, от наблюдателя для оси *y* на трехмерном графике, снизу вверх для оси *z* на трехмерном графике;
  - `'reverse'`, обратное расположение оси.
- `XGrid`, `YGrid`, `ZGrid` — сетка, перпендикулярная соответствующим осям, возможные значения:
  - `'on'`, нанести сетку;
  - `'off'`, убрать сетку.
- `XLim`, `YLim`, `ZLim` — пределы осей, значение свойства каждой оси задается вектором, первый элемент которого является начальным значением, а второй — конечным.
- `XScale`, `YScale`, `ZScale` — масштабы осей, возможные значения:
  - `'linear'`, линейный;
  - `'log'`, логарифмический.
- `XTick`, `YTick`, `ZTick` — положение отрезков разметки осей, возможные значения:
  - вектор из возрастающих значений, например `[1.2 2.5 3.5]`;
  - пустой вектор `[]` при отсутствии разметки.
- `XTickLabel`, `YTickLabel`, `ZTickLabel` — текст разметки осей, возможные значения:
  - текст в формате `'x=1.2|x=2.5|x=3.5'`, если число элементов текста разметки равно числу элементов вектора, задающего координаты отрезков разметки, то под каждым отрезком разметки помещается соответствующий текст;
  - `''`, пустая строка при отсутствии текста разметки.
- `XTickLabelMode`, `YTickLabelMode`, `ZTickLabelMode` — режим текстовой разметки осей, возможные значения:
  - `'auto'`, под отрезки разметки помещаются числа, соответствующие координатам отрезков;
  - `'manual'`, под отрезки разметки помещается текст, определенный в `XTickLabel`, `YTickLabel`, `ZTickLabel`; при установке значений свойств `XTickLabel`, `YTickLabel`, `ZTickLabel` устанавливается значение `manual` соответствующего свойства.

- `XTickMode`, `YTickMode`, `ZTickMode` — режим разметки осей, возможные значения:
  - `'auto'`, координаты отрезков разметки выбираются автоматически;
  - `'manual'`, координаты отрезков разметки определяются в `XTick`, `YTick`, `ZTick`, при установке значений свойств `XTick`, `YTick`, `ZTick` устанавливается значение `manual` соответствующего свойства.

Замечание

Значения свойств осей `Title`, `Xlabel`, `Ylabel`, `Zlabel` являются не текстом заголовка или подписей к осям, а *указателями* на соответствующие объекты. Подробнее об указателях написано в части, посвященной разработке собственных приложений в MatLab.

Часть свойств осей, начинающихся со слова `Camera` (камера), предназначены для управления обзором трехмерных поверхностей. Самый простой способ размещения наблюдателя за трехмерным объектом состоит в задании его положения двумя углами — азимутом и углом склонения.

См. раздел "Поворот графика, изменение точки обзора" главы 3.

MatLab предоставляет более развитое средство для просмотра трехмерных объектов — камеру.

Управление камерой

Изображение пространственного объекта на экране является проекцией на плоскость. Представьте, что эта проекция получена при помощи некоторой камеры, расположенной рядом с объектом так, как показано на рис. 4.8.

Для управления взаимным расположением камеры и графического объекта предназначены свойства осей, приведенные ниже.

- `CameraPosition` — положение камеры в системе координат осей, задается вектором с координатами [*x*, *y*, *z*]. Приближение камеры к объекту при фиксированном значении угла обзора `CameraViewAngle` позволяет увеличить масштаб просмотра, отдаление камеры приводит к уменьшению масштаба. Если оси отображаются в перспективе (свойство `Projection` имеет значение `perspective`), то она преобразуется соответствующим образом. Изменение положения камеры приводит к установке свойства `CameraPositionMode` в значение `manual`.
- `CameraPositionMode` — режим расположения камеры, возможные значения:
  - `'auto'`, положение камеры задается азимутом и углом склонения;
  - `'manual'`, положение камеры определяется свойством `CameraPosition`.

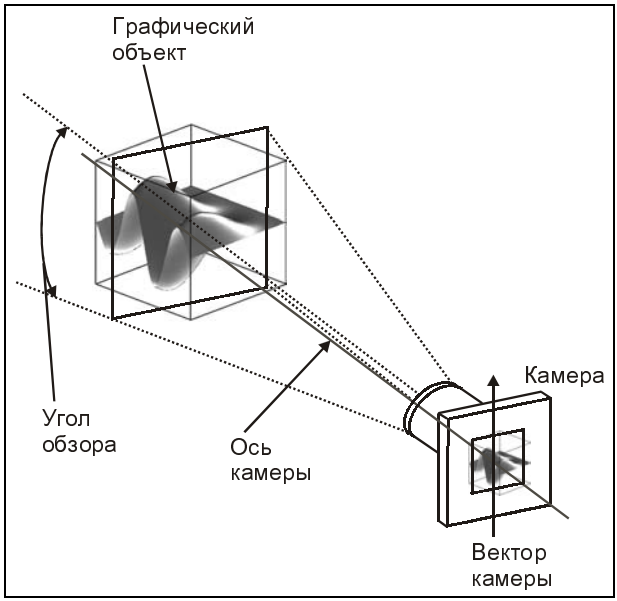


Рис. 4.8. Взаимное расположение камеры и графического объекта

- `CameraTarget` — точка с координатами [*x*, *y*, *z*], на которую направлена камера. Вместе с `CameraPositon`, свойство `CameraTarget` определяет ось камеры, приведенную на рис. 4.8.
- `CameraTargetMode` — режим позиционирования камеры, определяющий точку, на которую она направлена, возможные значения:
  - `'auto'`, камера направлена на точку, расположенную в центре параллелепипеда, образованного осями;
  - `'manual'`, точка, на которую направлена камера, задается свойством `CameraTarget`.
- `CameraUpVector` — поворот камеры вокруг оси просмотра, задается координатами [*x*, *y*, *z*] вектора камеры (см. рис. 4.8).
- `CameraUpVectorMode` — режим выбора направления вектора камеры, возможные значения:
  - `'auto'`, вектор камеры сонаправлен с осью *z*, т. е. значение свойства `CameraUpVector` равно `[0 0 1]`;
  - `'manual'`, направление вектора камеры задается свойством `CameraUpVector`.
- `CameraViewAngle` — угол обзора в градусах, больший нуля и меньший либо равный 180 °. Изменение угла обзора влияет на размер графического объекта на экране, перспектива осей не претерпевает изменений.

- `CameraViewAngleMode` — режим выбора угла обзора, возможные значения:
  - `'auto'`, автоматически подбирается минимальный угол обзора для охвата всего графического объекта;
  - `'manual'`, значение угла обзора устанавливается свойством `CameraViewAngle`.

Свойства графического окна

В предыдущих разделах, посвященных свойствам осей и поверхностей, ничего не было сказано про выбор палитры цвета. Дело в том, что палитра определяется значением свойства `Colormap` графического окна (значениями могут быть названия палитр, например `copper`, `gray` и т. д.). Для вывода графиков в различных палитрах следует использовать *разные* графические окна. Цвет фона графического окна задается свойством `Color`. При помощи редактора свойств возможно управление печатью графиков, однако проще и нагляднее установки печати производятся из меню **File** графического окна.

См. разд. "Сохранение, экспорт и печать графиков" данной главы.

Подробную информацию о назначениях свойств объектов MatLab 5.3 можно получить при помощи справки MATLAB Help Desk в формате HTML, для запуска которой следует выбрать пункт **Help Desk (HTML)** меню **Help** командного окна или любого графического окна. Открывается браузер с основным документом, содержащим разделы справочной системы. Ссылка Handle Graphics Properties указывает на документацию по свойствам объектов в MatLab. В окне с документацией приведена полная структура объектов MatLab. Для получения информации о любом объекте следует щелкнуть на его названии мышкой, при этом окно разделяется на три части (фрейма), в нижнем левом фрейме приведен полный список свойств выбранного объекта. При щелчке мыши по свойству появляется его описание в правом нижнем фрейме.

Редактирование графиков в MatLab 6.*x*

Как и версия 5.3, MatLab 6.*x* также является *объектно-ориентированной системой*, причем все *объекты* расположены в определенной иерархической последовательности. На каждом графическом окне (figure) может быть расположен один или несколько графиков со своими *осями* (axes), созданные, например, при помощи команды `subplot`. На каждом графике может быть одна или несколько *линий* (line) или *поверхностей* (surface). Кроме того, возможно добавление текста (text), стрелок (arrow) и линий (line), которые принадлежат специальному *слою примечаний* (annotationlayer) графического окна. Упрощенная

структура графических объектов была приведена на рис. 4.5. Редактор свойств позволяет получить доступ к свойствам каждого из вышеперечисленных *объектов*. В этом разделе описаны основные возможности, которые предоставляет MatLab 6.*x* для управления видом графика.

Запуск редактора свойств

Рассмотрим снова уже использованный нами ранее пример. Выведите в графическое окно два графика. На одном постройте $\sin x$ и $\cos x$ на отрезке $[0, 6\pi]$, а на другом функцию $e^{-x^2-y^2} \cdot (x-1)^2 \cdot \sin 2\pi y$ на квадрате $x, y \in [-1, 1]$. Ниже приведена последовательность команд, соответствующая требуемому виду графического окна **Figure No. 1**, изображенного на рис. 4.6.

```
>> x = [0:0.1:5];
>> f = sin(pi*x);
>> g = cos(pi*x);
>> subplot(2, 1, 1)
>> plot(x, f, x, g)
>> [X, Y] = meshgrid(-1:0.05:1, -1:0.05:1);
>> Z = exp(-X.^2-Y.^2).*(X-1).^2.*sin(2*pi*Y);
>> subplot(2, 1, 2)
>> surf(X, Y, Z)
```

Перейдите в режим редактирования графического окна, выбрав в меню **Tools** пункт **Edit Plot**. Если слева от названия пункта стоит флаг, то вы уже находитесь в режиме редактирования. Для доступа к свойствам всех объектов, расположенных на графическом окне, выберите в меню **Edit** пункт **Figure Properties**. На экране появится диалоговое окно с вкладками **Property Editor** (Редактор свойств), приведенное на рис. 4.9.

Вверху диалогового окна находится раскрывающийся список объектов **Edit Properties for**, из которого можно получить доступ к любому из объектов графического окна. Для окна, изображенного на рис. 4.6, список объектов имеет следующую древовидную структуру:

```
root : 0
    figure : 1
        axes :
            surface :
        axes :
            line :
            line :
```

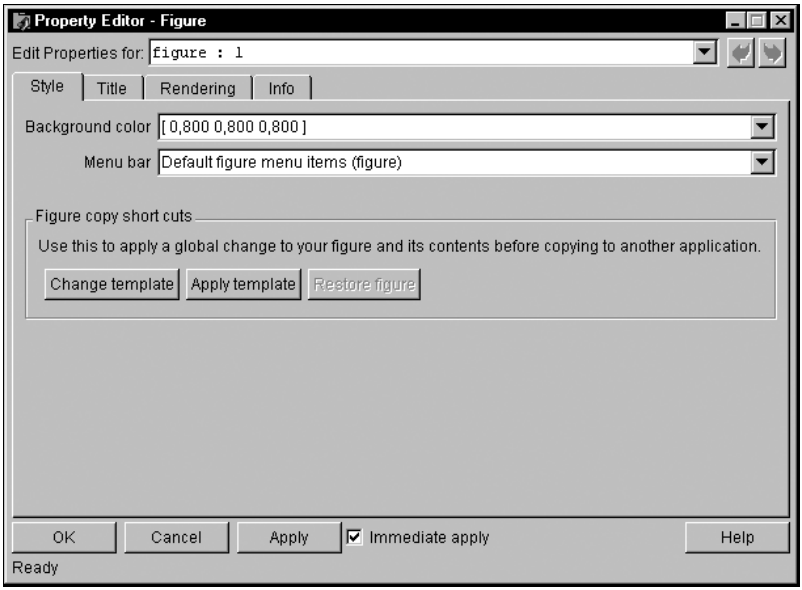


Рис. 4.9. Диалоговое окно **Property Editor**

Нас будет интересовать часть списка, начинающаяся с `figure : 1`, которая содержит информацию об объектах графического окна **Figure No. 1.** Выбор объектов в раскрывающемся списке приводит к изменению вида диалогового окна — появляются вкладки, соответствующие свойствам выбранного объекта. При выборе объекта он автоматически выделяется в графическом окне. Выделение объекта в режиме редактирования можно осуществить щелчком левой кнопкой мыши на нем в графическом окне. В диалоговом окне **Property Editor** (при условии, что оно открыто) отображаются вкладки со свойствами выделенного объекта. Выбор окна **Property Editor** с вкладкой, соответствующей требуемому объекту, производится двойным щелчком мыши по нужному объекту. В дальнейшем не уточняется способ выделения объекта, а просто говорится, что нужно перейти к свойствам объекта.

Замечание

Все открытые в MatLab графические окна (figure) помещаются в список объектов, доступ к которому производится из пункта **Figure Properties** меню **Edit** *любого* графического окна. Для каждого из графических окон указываются принадлежащие ему объекты.

Расположите графическое окно и редактор свойств на экране так, чтобы они не перекрывали друг друга. Установите флажок **Immediate apply** внизу диалогового окна, теперь все изменения будут сразу отображаться на графике, что удобно при изучении работы в редакторе свойств.

Свойства осей, подписи, заголовок

Для редактирования свойств осей требуется выбрать в раскрывающемся списке **Edit Properties for** диалогового окна **Property Editor** нужные оси. Установите, например, свойства осей верхнего графика, приведенного на рис. 4.6. Выберите в списке соответствующие оси (axes), при этом они выделятся на графике рамкой с квадратными маркерами. В нашем случае легко найти требуемые оси, т. к. они содержат две линии (line). Обратите внимание, что вкладки диалогового окна изменились, теперь они соответствуют свойствам осей.

Пределы, масштаб, разметка, сетка

При помощи элементов управления, расположенных на вкладке **Scale**, осуществляются действия (в скобках указаны названия соответствующих полей окна):

- изменение области построения графика (**Limits**);
- нанесение желаемой разметки на оси (**Ticks**) и маркировка разметки (**Labels**);
- масштабирование осей (**Scale**);
- размещение сетки на графике (**Grid**).

Все перечисленные свойства устанавливаются для каждой оси независимо. Для получения доступа к полям **Limits**, **Ticks** и **Labels** следует выключить соответствующий флажок **Auto** и ввести в поля нужные значения. Координаты разметки набираются в поле **Ticks**, текст разметки в списке **Labels** для каждой оси. Учтите, что при маркировке осей не допускается использование формата TeX. Изменение масштаба и направления осей достигается установкой переключателей в поле **Scale**. Возможны два вида масштабирования: линейный и логарифмический, им соответствуют переключатели **Linear** и **Log**. Для каждого из способов масштабирования возможно либо обычное направление оси (переключатель **Normal**), либо обратное (переключатель **Reverse**). Установка флажков **Grid** приводит к нанесению линий сетки для соответствующей оси. Координаты линий сетки совпадают с координатами разметки осей. Приведите, например, график к виду, изображенному на рис. 4.10.

Для установки требуемых свойств осей необходимо произвести следующие действия.

1. Сделать доступными поля **Limits** оси *x*, отключив флаг **Auto**, и ввести пределы построения графика 0 и 4.
2. Сделать доступным поле **Ticks** оси *x*, отключив флаг **Auto**, и ввести координаты разметки [0 1 2 3 4].

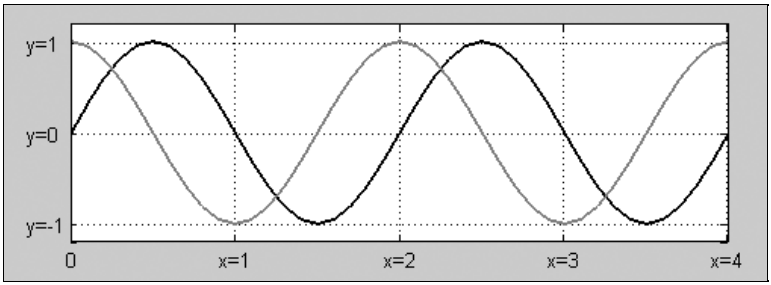


Рис. 4.10. Масштабирование, пределы и разметка осей

3. Сделать доступным поле (список) **Labels** оси *x*, отключив флаг **Auto**, и ввести в каждой строке списка обозначения разметки 0, $x = 1$, $x = 2$, $x = 3$, $x = 4$.
4. Добавить сетку по оси *x*, включив флаг **Show** в поле **Grid** оси *x*.
5. Для оси *y* аналогично задать пределы −1.2 и 1.2, координаты разметки [−1 0 1], обозначения разметки $y = -1$, $y = 0$, $y = 1$ и включить сетку.

Элементы управления, расположенные на вкладке **Style**, предоставляют дополнительные возможности оформления осей.

- Панель **Style** содержит флаг **Hide axes**, позволяющий убрать оси, разметку и сетку, оставляя только линии графика. Включенный флаг **Axis box on** означает, что оси имеют вид рамки вокруг графика, а выключенный — обычные оси. Толщина осей выбирается в раскрывающемся списке **Axis line width**.
- Задать шрифт разметки осей можно при помощи раскрывающихся списков панели **Tick Label Font**.
- Цвет осей и фона устанавливается в раскрывающихся списках панели **Color**.
- Расположение вертикальной оси справа или слева от графика и горизонтальной — сверху или снизу производится из раскрывающихся списков панели **Location**.

Подписи и заголовок

На окне с вкладкой **Labels** размещены поля для задания заголовка **Title** и подписей к осям **Xlabel**, **Ylabel** и **Zlabel**. Можно уже набрать текст в соответствующей области ввода. Но удобнее поступить по-другому. Нажатие кнопки **Edit**, расположенной справа от области ввода, меняет вид диалогового окна **Property Editor** на многостраничное диалоговое окно, которое позволяет ввести текст и установить его свойства.

Вкладка **Text** содержит две панели: **Font** с шестью раскрывающимися списками для установки свойств шрифта и **Text** с областью ввода текста и флагом **Use LaTeX Interpreter**. Установленный флаг позволяет использовать формат TeX для изменения свойств шрифта части текста (например, можно выделить переменные курсивом), набора греческих букв и установки верхних и нижних индексов. Если флаг **Use LaTeX Interpreter** отключен, то в графическое окно выведется то, что набрано в области ввода, например {\ity}, а не *y*. Аналогичным образом задаются названия осей графика. Дайте заголовок верхнему графику графического окна, как показано на рис. 4.11.

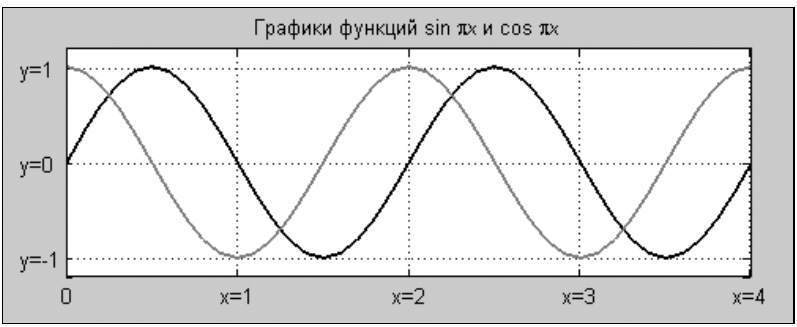


Рис. 4.11. График с заголовком

Для получения требуемого заголовка следует набрать в поле ввода **Text** строку в формате TeX: "График функций sin \pi{\itx} и cos \pi{\itx}" (кавычки вводить не надо).

Выравнивание текстовых объектов (заголовка и названия осей) производится при помощи элементов управления, расположенных на вкладке **Alignment**. Панель **Position** содержит раскрывающиеся списки для определения координаты левого нижнего угла текстового объекта. В раскрывающемся списке **Units** устанавливаются единицы измерения координат, причем **Data** означает, что координаты текста задаются в той же системе координат, что и график функции. Раскрывающиеся списки панели **Alignment** позволяют выровнять текст по вертикали и горизонтали и повернуть его под любым углом.

Работа с осями трехмерных графиков производится аналогичным образом. В качестве упражнения подпишите оси трехмерного графика графического окна, изображенного на рис. 4.6.

Свойства линий и поверхностей

Редактор свойств позволяет легко установить стиль, цвет и толщину линий и маркеров по своему усмотрению.

Свойства линий

Перейдите к свойствам какой-нибудь из линий верхнего графика, изображенного на рис. 4.10. Выберите вкладку **Style** диалогового окна **Property Editor**. На ней расположены три панели:

- **Line Properties** с раскрывающимися списками для определения стиля линии (сплошная, штриховая, пунктирная, штрих-пунктирная), толщины и цвета;
- **Marker Properties** с раскрывающимися списками для задания типа маркера, размера, цвета границ маркера и цвета маркера;
- **Example**, содержащая образец линии.

При помощи этих элементов управления легко изменить вид графика по своему усмотрению. Пункты **No line (none)** и **No marker (none)** раскрывающихся списков для выбора стилей линии и маркера означают отсутствие, соответственно, линии и маркеров на графике. Измените свойства линий и маркеров графиков, изображенных на рис. 4.10, так, чтобы получившиеся графики имели вид, приведенный на рис. 4.12.

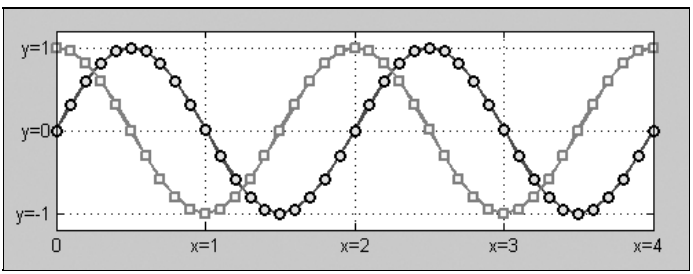


Рис. 4.12. Изменение стиля линий и маркеров

Свойства поверхностей

Перейдите к свойствам поверхности, изображенной на нижнем графике на рис. 4.5. Вкладка **Style** диалогового окна **Property Editor** позволяет задать цвет, освещенность и прозрачность поверхности. Соответствующие раскрывающиеся списки **Color**, **Lighting** и **Transparency** размещены на панели **Face Properties**. Свойства линий сетки поверхности (тип линии, толщина, цвет и т. д.) задаются из раскрывающихся списков панели **Mesh Properties**. Например, в списке **Mesh Style** указывается способ отображения матрицы, содержащей координаты точек сетки поверхности по оси *z*:

- **Rows and columns (both)** — пересекающиеся линии сетки соответствуют строкам и столбцам;

- **Row lines only** — рисуются только линии поверхности, соответствующие строкам (вдоль оси *x*);
- **Column lines only** — рисуются только линии поверхности, соответствующие столбцам (вдоль оси *y*).

На панели **Actions** имеются пять кнопок для определения вида поверхности:

- кнопка **Wireframe (h)** — рисуется каркасная модель поверхности, скрытые линии убираются (соответствует командам `mesh`, `hidden on`);
- кнопка **Wireframe** — рисуется каркасная модель поверхности, показываются скрытые линии (соответствует командам `mesh`, `hidden off`);
- кнопка **Point cloud (h)** — поверхность изображается вершинами сетки, скрытые вершины убираются;
- кнопка **Point cloud** — поверхность изображается вершинами сетки, показываются скрытые вершины;
- кнопка **Solid** — рисуется закрашенная каркасная модель поверхности (соответствует `surf`).

Имеется возможность поместить маркеры в вершины сетки поверхности. При помощи раскрывающихся списков панели **Marker Properties** можно управлять свойствами маркеров.

Выбор цветовой палитры (эквивалент команды `colormap`) осуществляется из раскрывающегося списка **Figure colormap** панели **Object Color Properties**, размещенной на вкладке **Color**.

Измените поверхность, приведенную на рис. 4.6, так, как изображено на рис. 4.13.

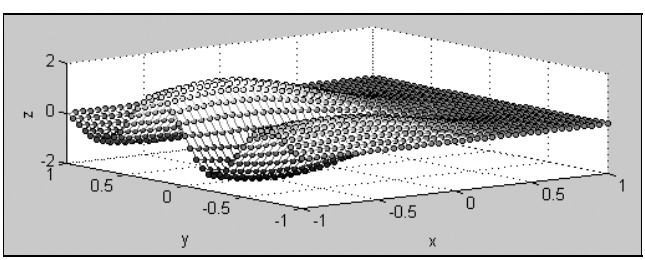


Рис. 4.13. Изменение свойств поверхности

Ниже приведены действия, приводящие к требуемому виду поверхности.

1. Измените цвет линий сетки на **Faceted CData (flat)** в списке **Color** (вкладка **Style**, панель **Mesh Properties**).
2. Определите тип маркера **Circle (o)** в списке **Style** (вкладка **Style**, панель **Marker Properties**).

3. Установите размер маркера, равный трем, в списке **Size** (вкладка **Style**, панель **Marker Properties**).
4. Задайте черный цвет **Black** границе маркера в списке **Edge color** (вкладка **Style**, панель **Marker Properties**).
5. Задайте цвет **Faceted CData (flat)** маркеру в списке **Face color** (вкладка **Style**, панель **Marker Properties**).
6. Выбрерите цветовую палитру **Gray** в списке **Figure colormap** (вкладка **Color**, панель **Object Color Properties**).

Дополнительные элементы оформления

Размещение дополнительной информации на графике осуществляется при помощи стрелок, линий, надписей и легенды. Выбор нужного объекта производится из меню **Insert** графического окна. Для того чтобы добавить стрелку (линию), следует выбрать пункт **Arrow** (**Line**) и нарисовать ее мышью при нажатой левой кнопке. Стрелки и линии, как и все объекты графического окна, можно перемещать по экрану и изменять их размеры (используя маркеры выделения границ) при помощи движения мыши, удерживая нажатой левую кнопку.

В любое место в пределах графического окна можно поместить надпись, для чего следует выбрать пункт **Text** и щелкнуть левой кнопкой мыши по графическому окну. Появится серое поле для ввода текста. При наборе текста допустимо использование формата TeX, причем текст может размещаться в несколько строк. Выход из режима набора текста осуществляется щелчком мыши вне текстовой области. Двойной щелчок левой кнопкой мыши по текстовому объекту позволяет снова перейти в режим редактирования.

Добавьте стрелки и подписи на график так, как показано на рис. 4.14.

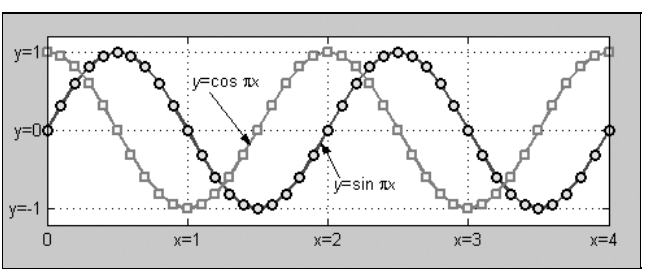


Рис. 4.14. Добавление стрелок и надписей

Выделение линии, стрелки или текстового объекта приводит к автоматическому отображению их свойств в диалоговом окне **Property Editor**. Раскройте список **Edit Properties for** и обратите внимание, что текстовые надписи,

стрелки и линии отображаются как объекты, принадлежащие специальному слову annotationlayer.

Добавьте легенду, для чего выделите оси графика и выберите пункт **Legend** меню **Insert** графического окна. В графическом окне появится заготовка для легенды. Легенда графика является объектом графического окна, для ее редактирования перейдите к свойствам легенды, например двойным щелчком мыши на ней. В окне **Property Editor** отображаются вкладки **Style** и **Text** со свойствами легенды. Панель **Style** вкладки **Style** содержит флаг **Show legend** для отображения или скрытия легенды и раскрывающийся список **Background**, позволяющий задать цвет фона легенды. Положение легенды на графике определяется из раскрывающегося списка **Position** одноименной панели. Опция **To the right of the plot (−1)** соответствует размещению легенды вне области осей справа от графика. Задание пояснений для линий легенды осуществляется в поле **Text** вкладки **Text**. Строки **data1**, **data2**, следует заменить на нужные, причем возможен ввод в формате TeX.

Трехмерные графики, построенные при помощи функций `mesh` и `surf`, полезно снабдить информацией о соответствии цвета значению функции. Выбор пункта **Colorbar** меню **Insert** приводит к размещению столбика с цветовой гаммой слева от выделенного графика (эквивалент команды `colorbar`).

Обратите внимание, что легенда и столбик с цветовой гаммой являются отдельными объектами, размещенными на графическом окне (убедиться в этом можно, раскрыв список **Edit Properties for** диалогового окна **Property Editor**).

Редактор свойств MatLab 6.*x* предоставляет также широкие возможности интерактивного управления освещенностью и видом трехмерных объектов.

Управление освещением графика

MatLab позволяет не только построить освещенную трехмерную поверхность, но и задать дополнительные источники света и их положение, определить свойства отражающей поверхности. Изучение этого вопроса начните с построения графика освещенной поверхности для функции $e^{-x^2-y^2} \cdot (x-1)^2 \cdot \sin 2\pi y$ на квадратной области определения $x, y \in [-1, 1]$ в отдельном окне при помощи команды `surfl`.

См. разд. "Построение освещенной поверхности" главы 3.

Сглаживание оттенков освещенной поверхности в *главе 3* производилось командой `shading interp`. Сейчас достаточно построить поверхность при помощи приведенных ниже команд. Остальные установки легко проделать из редактора свойств.

```
>> [X, Y] = meshgrid(-1:0.05:1, -1:0.05:1);
```

```
>> Z = exp(-X.^2-Y.^2).*(X-1).^2.*sin(2*pi*Y);
>> surfl(X, Y, Z)
```

Перейдите к пункту **Axes Properties** меню **Edit** графического окна, убедитесь, что раскрывающийся список **Edit Properties for** содержит опцию **surface**, и установите свойства поверхности, линий сетки и освещения, приводящие к плавному изменению цвета и освещенности поверхности. Выполните следующие действия.

1. Выберите цветовую палитру с большим спектром оттенков, например `gray` в раскрывающемся списке **Figure colormap** (вкладка **Color**).
2. Уберите линии сетки поверхности (опция **No line (none)**) в раскрывающемся списке **Line Style** панели **Mesh Properties** вкладки **Style**.
3. Задайте плавное изменение оттенков, выбрав **Blended CData (interp)** в раскрывающемся списке **Color** панели **Face Properties** вкладки **Style**.
4. Установите плавное изменение освещенности **Smoother rounded (phong)** в раскрывающемся списке **Lighting** панели **Face Properties** вкладки **Style**.

Получившаяся поверхность должна иметь вид, изображенный на рис. 4.15.

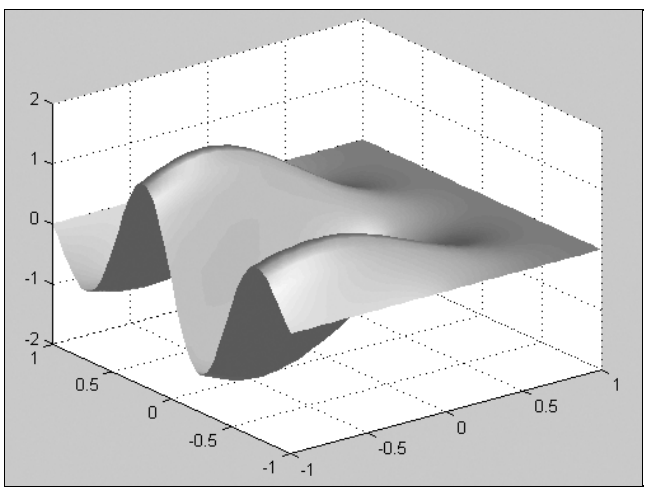


Рис. 4.15. График освещенной поверхности

Управление источниками света производится из диалогового окна, соответствующего свойствам осей. Для добавления нового источника света перейдите к вкладке **Lights** и нажмите кнопку **Add a new light**. О том, сколько источников добавлено, можно узнать из списка **Lights**. Перед изменением координат источника следует убедиться, что он выбран в этом списке, т. е. является текущим. Задание координат осуществляется при помощи строк

ввода **X position**, **Y position** и **Z position**, расположенных на панели **Position**. Можно набрать в них координаты, но проще воспользоваться кнопками со стрелочками для изменения координат текущего источника. Если флаг **Immediate apply** установлен, то вы можете одновременно наблюдать изменение освещения поверхности, расположив графическое окно и редактор свойств так, чтобы они не перекрывали друг друга. Добавьте два источника света и осветите поверхность сверху и снизу так, как показано на рис. 4.16.

Кроме изменения координат источника света возможно изменение его цвета при помощи раскрывающегося списка **Color** панели **Appearance** и помещение источника на бесконечном расстоянии от поверхности установкой флага **Place light at infinite distance** на этой же панели. Лучи света источника, помещенного на бесконечном расстоянии, идут параллельно от источника из направления, которое задано в **X position**, **Y position**, **Z position**. Если флаг **Place light at infinite distance** выключен, то лучи света равномерно распространяются от точечного источника во все стороны. При отражении от поверхности свет рассеивается, в раскрывающемся списке **Color** панели **Ambient Light** выбирается цвет рассеянного света.

Удаление текущего источника света (выбранного в списке **Lights**) производится нажатием кнопки **Delete selected light**. Обратите внимание, что удаление всех источников света не приводит к темной поверхности. Это связано с тем, что функция `surfl` при построении поверхности добавляет источник света, расположенный под углом 45° вдоль вертикальной оси в направлении против часовой стрелки от текущей точки обзора.

См. разд. "Построение освещенной поверхности" главы 3.

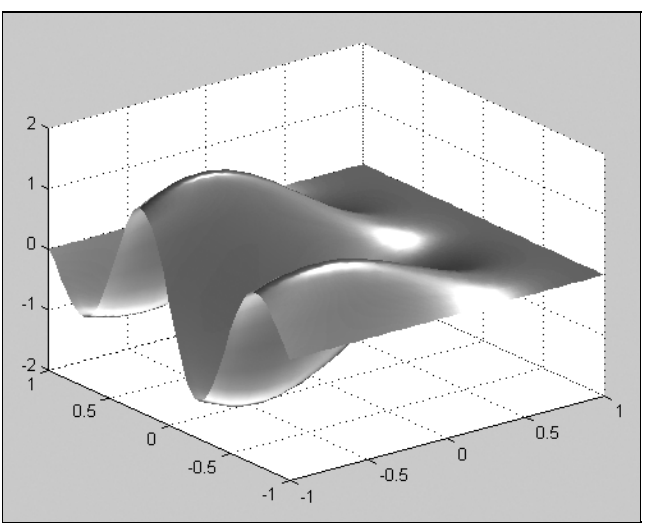


Рис. 4.16. Поверхность, освещенная с двух сторон

Изменение точки обзора

Самым простым способом изменения точки обзора является определение положения наблюдателя, которое задается двумя углами: азимутом и углом склонения.

См. разд. "Поворот графика, изменение точки обзора" главы 3.

Перейдите к свойствам осей в диалоговом окне **Property Editor**. При помощи строк ввода **Azimuth (degrees)** и **Elevation (degrees)**, расположенных на вкладке **Viewpoint**, устанавливаются значения азимута и угла склонения в градусах. Используйте кнопки со стрелками для изменения значений. Если установлен флажок **Immediate apply**, то вы можете сразу наблюдать за результатом.

Замечание

Поворачивать оси графика в любом направлении можно движением мыши (с нажатой левой кнопкой) при помощи инструмента Rotate 3D, размещенного на панели **Figure Toolbar** графического окна.

Существует более развитое средство управления видом графика по сравнению с заданием точки обзора. Когда мы смотрим на трехмерный объект, изображенный на экране компьютера, мы на самом деле видим его проекцию на экран, осуществляемую при помощи некоторой камеры (Camera в MatLab). Рис. 4.8 поясняет взаимное расположение камеры и графического объекта.

Редактор свойств позволяет интерактивно задавать все характеристики проектирования трехмерных объектов на экран при помощи элементов управления, расположенных на панели **Camera Properties** вкладки **Viewpoint** диалогового окна **Property Editor** (в котором отображены свойства осей). Панель содержит четыре поля:

- **Position** — для задания положения камеры объекта;
- **Target** — для задания положения графического объекта;
- **Up vector** — для изменения координат вектора камеры;
- **View angle** — для выбора угла обзора.

В раскрывающихся списках следует изменить **Auto** на **Manual** для получения доступа к полям координат (левое поле соответствует координате *x*, среднее — *y*, правое — *z*). Убедитесь, что флажок **Immediate apply** установлен, и, изменяя положение камеры и графического объекта, вектора камеры и угла обзора при помощи кнопок со стрелками наблюдайте за состоянием графического окна.

Оси графика по умолчанию строятся в виде ортогональной проекции, что соответствует переключателю **Orthographic** панели **Projection**. Выбор переключателя **Perspective** приводит к отображению осей в перспективе.

Быстрый доступ к свойствам камеры производится из панели инструментов **Camera Toolbar**. Добавление панели на графическое окно осуществляется выбором пункта **Camera Toolbar** из меню **View**. Панель **Camera Toolbar** состоит из нескольких групп. Группа инструментов **Camera Motion Controls**, изображенная на рис. 4.17, предназначена для управления движением камеры и расположением источника света.

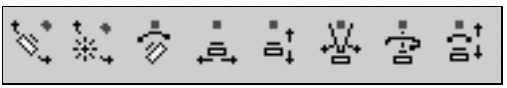


Рис. 4.17. Инструменты управления движением камеры и расположением источника света

При первом применении инструмента появляется окно с предупреждением о том, что следует установить автоматический подбор размеров осей. Это окно появляется только один раз в течение всего сеанса работы MatLab, причем при использовании инструментов управления камерой требуемые установки производятся автоматически. Чтобы избежать появления в дальнейшем данного окна следует установить в нем флаг **Do not show this dialog again**.

Пиктограмма кнопки позволяет легко определить тип движения камеры или источника света, что производится перемещением указателя мыши в пределах графического окна при нажатой левой кнопке мыши.

В группу **Camera Motion Control** входят следующие инструменты.

- **Orbit Camera** — вращение камеры вокруг основной оси (про выбор основной оси см. ниже).
- **Orbit Scene Light** — управление положением источника света, который добавляется и убирается при помощи инструмента **Toggle Scene Light**, расположенного на панели **Camera Toolbar**.
- **Pan/Tilt Camera** — перемещение графического объекта (основная ось направлена вверх).
- **Move Camera Horizontally/Vertically** — движение камеры по горизонтали или вертикали.
- **Move Camera Forward/Back** — приближение камеры к графическому объекту движением мыши вправо или вверх, отдаление — влево или вниз.
- **Zoom Camera** — увеличение угла обзора движением мыши вправо или вверх, уменьшение — налево или вниз.
- **Roll Camera** — поворот камеры вокруг ее оси вращением мыши по (или против) часовой стрелке в пределах графического окна.

- **Walk Camera** — перемещение камеры и графического объекта в направлении оси камеры вперед (назад) при передвижении мыши вверх (вниз), перемещение графического объекта направо (налево) при передвижении мыши налево (направо).

Часть инструментов управления движением камеры: **Orbit Camera**, **Pan/Tilt Camera**, **Walk Camera** требуют задания основной оси (principal axis), по отношению к которой происходит движение камеры. Основная ось направлена вверх на экране. При использовании этих инструментов становится доступной группа инструментов **Principal Axis Selector** панели **Camera Toolbar**, приведенная на рис. 4.18, причем по умолчанию основной является ось *z*. Возможно задание осей *x*, *y* или *z* в качестве основных, или установка свободного движения безотносительно какой-либо оси.

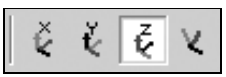


Рис. 4.18. Инструменты выбора основной оси

При помощи остальных инструментов, расположенных на панели **Camera Toolbar**, можно:

- добавить или удалить один источник света (**Toggle Scene Light**);
- установить ортогональную проекцию осей на экран (**Orthographic Projection**) или отобразить оси в перспективе (**Perspective Projection**);
- вернуть графику первоначальный вид (**Reset**);
- остановить движение графика (**Stop**), что может быть полезно, если вы задали слишком много перемещений движением мыши и MatLab долго обрабатывает изменение графического окна.

Сохранение, экспорт и печать

После сохранения графического окна при помощи пунктов **Save** или **Save as** меню **File** его можно закрыть, например, выбрав пункт **Close** меню **File** графического окна. В дальнейшем, для продолжения работы с графическим окном, следует открыть файл, содержащий окно, из пункта **Open** меню **File** рабочей среды MatLab или из одноименного пункта любого открытого графического окна.

Как уже упоминалось ранее, при помощи пкнкта **Export** меню **File** графического окна осуществляется сохранение области графического окна (без меню и панели инструментов) в файле форматов BMP, GIF, TIFF, JPEG и др.

MatLab предоставляет возможность управлять видом графика при печати. Выбор пункта **Page Setup** меню **File** приводит к появлению на экране окна многостраничного диалога **Page Setup**, изображенного на рис. 4.19.

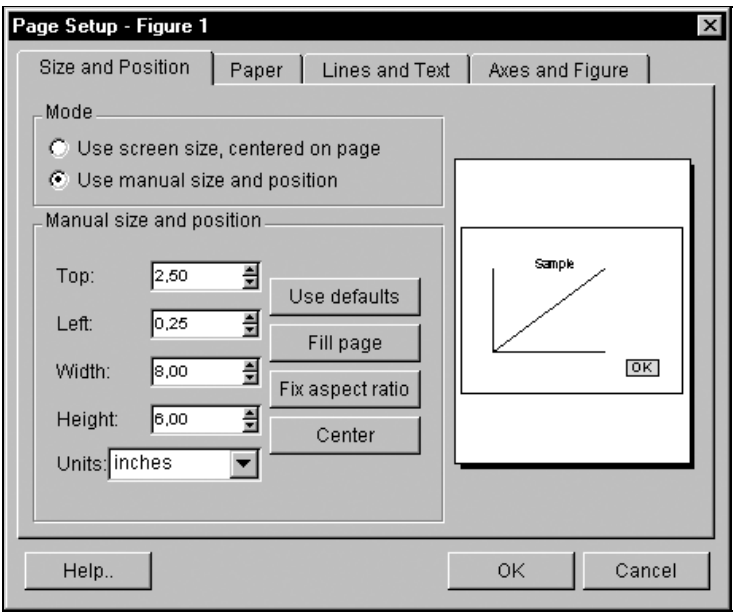


Рис. 4.19. Диалоговое окно **Page Setup** версии 6.*x*

Элементы управления, расположенные на вкладке **Size and Position**, позволяют задавать размеры графика и его положение на странице, причем эта вкладка содержит образец страницы. По умолчанию включен переключатель **Use screen size, centered on page**, что соответствует печати графика того же размера, что и на экране, в центре листа. Для расположения графика на листе по своему усмотрению установите переключатель **Use manual size and position**, при этом становятся доступными элементы управления, расположенные на панели **Manual size and position**. В полях **Top**, **Left**, **Width** и **Height** определяются координаты верхнего левого угла, ширина и высота графика. Единицы измерения выбираются в поле **Units**, причем **normalized** соответствует заданию относительных величин (высота и ширина листа считаются равными единице). Для непропорционального растяжения графика на всю область листа следует нажать кнопку **Fillpage**. Максимально возможное растяжение с сохранением пропорций происходит при нажатии кнопки **Fix aspect ratio**. Кнопка **Center** служит для помещения графика в центре листа. При помощи мыши можно изменять размеры и расположение графика в поле с образцом страницы.

Вкладка **Paper** предназначена для установки размеров и ориентации листа бумаги, причем в образце листа, размещенном на этой вкладке также возможно изменение размеров и положения графика при помощи мыши. Задание черно-белой печати или цветной печати производится из вкладки **Lines and Text**. Если вы печатаете на черно-белом принтере и хотите получить оттенки серого для цветных элементов (линий и текста), то следует установить переключатель **Color**. Для печати текста и линий черным цветом выберите **Black and White**.

При изменении размеров графика в диалоговом окне **Page Setup** MatLab автоматически подбирает пределы осей и разметку для обеспечения хорошего вида графика. Автоматическому подбору соответствует переключатель **Recompute limits and ticks**, содержащийся на вкладке **Axes and Figure**. Если вы хотите напечатать график в том виде, в котором он изображен на экране, то установите переключатель **Keep screen limits and ticks**. Как правило, нет смысла печатать серый фон графического окна, что и делается по умолчанию (включен переключатель **Force white background**). Если же требуется напечатать график на фоне графического окна, то следует установить переключатель в положение **Keep screen background color**. Флаг **Print UIControls** предназначен для того, чтобы печатать собственные элементы пользовательского интерфейса (кнопки, списки, текстовые области и т. д.), размещаемые в графическом окне. Созданию собственных приложений в MatLab посвящена третья часть книги.

Диалоговое окно для **Print Setup** установки параметров принтера доступно из пункта **Print Setup** меню **File** графического окна, а диалоговое окно печати **Print** — из пункта **Print**. После установки параметров страницы перед печатью полезно воспользоваться предварительным просмотром при помощи пункта **Print Preview** меню **File** графического окна.

Глава 5



M-файлы

Работа из командной строки MatLab затрудняется, если требуется вводить много команд и часто их изменять. Ведение дневника при помощи команды `diary` и сохранение рабочей среды незначительно облегчает работу. Самым удобным способом выполнения команд MatLab является использование *M-файлов*, в которых можно набирать команды, выполнять их все сразу или частями, сохранять в файле и использовать в дальнейшем. Для работы с M-файлами предназначен редактор M-файлов. При помощи редактора M-файлов можно создавать собственные функции и вызывать их, в том числе и из командной строки. Простейшее использование редактора M-файлов в версиях 5.3 и 6.*x* происходит практически одинаково.

Работа в редакторе M-файлов

Раскройте меню **File** основного окна MatLab и в пункте **New** выберите подпункт **M-file**. Новый файл открывается в окне редактора M-файлов, которое для версии 5.3 и 6.*x* имеет похожий вид, изображенный на рис. 5.1 и 5.2 соответственно.

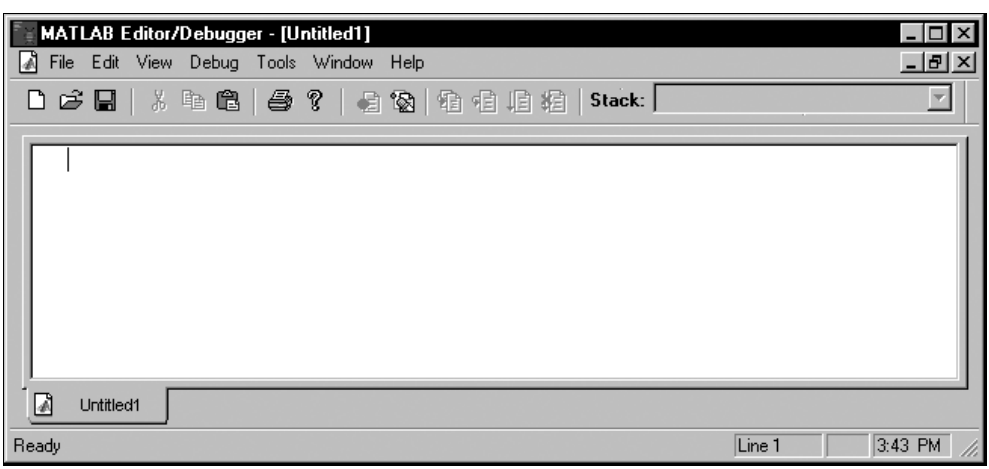


Рис. 5.1. Редактор M-файлов MatLab 5.3

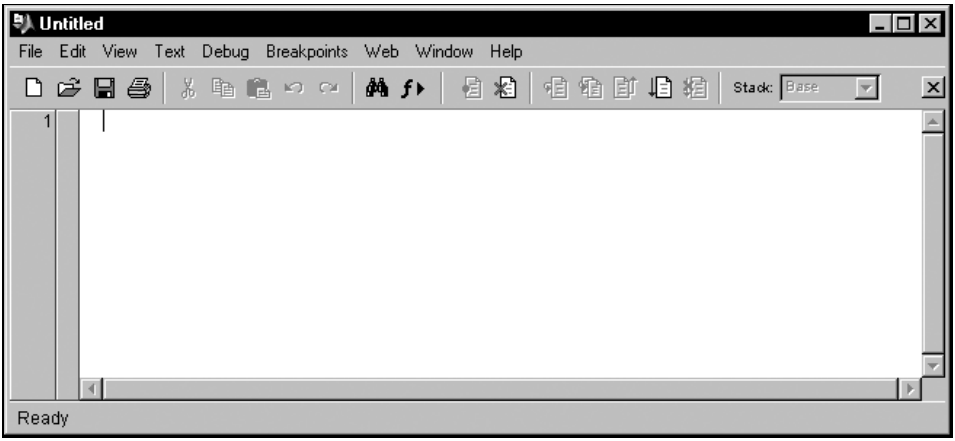


Рис. 5.2. Редактор М-файлов MatLab 6.*x*

Наберите в редакторе команды, приводящие к построению двух графиков на одном графическом окне. Необязательно набирать много команд — наша цель сейчас состоит в том, чтобы научиться выполнять команды из редактора М-файлов. Ограничьтесь командами, приведенными в листинге 5.1.

Листинг 5.1. Создание двух графиков в одном окне

```
x = [0:0.1:7];
f = exp(-x);
subplot(1, 2, 1)
plot(x, f)
g = sin(x);
subplot(1, 2, 2)
plot(x, g)
```

Сохраните теперь файл с именем mydemo.m в подкаталоге work основного каталога MatLab, выбрав пункт **Save as** меню **File** редактора. Для запуска на выполнение *всех команд*, содержащихся в файле, следует выбрать пункт **Run** в меню **Tools** (для версии 5.3) или в меню **Debug** (для версии 6.*x*). На экране появится графическое окно **Figure No.1**, содержащее графики функций. Если вы решили построить график косинуса вместо синуса, то просто измените строку `g=sin(x)` в М-файле на `g=cos(x)` и запустите все команды снова.

Замечание

Если при наборе сделана ошибка и MatLab не может распознать команду, то происходит выполнение команд до неправильно введенной, после чего выводится сообщение об ошибке в командное окно.

Команды MatLab файл-программы осуществляют вывод в командное окно. Для подавления вывода следует завершать команды точкой с запятой.

Очень удобной возможностью, предоставляемой редактором М-файлов, является *выполнение части команд*. Закройте графическое окно **Figure No.1**. Выделите при помощи мыши, удерживая левую кнопку, или клавишами со стрелками при нажатой <Shift>, первые четыре команды листинга 5.1 и выполните их из пункта **Evaluate Selection** меню **View** (в версии 5.3) или **Text** (в версии 6.*x*). Обратите внимание, что в графическое окно вывелся только один график, соответствующий выполненным командам. Запомните, что для выполнения части команд их следует выделить и нажать <F9>. Выполните оставшиеся три команды листинга 5.1 и проследите за состоянием графического окна. Потренируйтесь самостоятельно, наберите какие-либо примеры из предыдущих глав в редакторе М-файлов и запустите их.

Отдельные блоки М-файла можно снабжать *комментариями*, которые пропускаются при выполнении, но удобны при работе с М-файлом. Комментарии в MatLab начинаются со знака процента и автоматически выделяются зеленым цветом, например:

```
%построение графика sin(x) в отдельном окне
```

В редакторе М-файлов может быть одновременно открыто несколько файлов. Переход между файлами осуществляется при помощи закладок с именами файлов, расположенных внизу окна редактора.

Открытие существующего М-файла производится при помощи пункта **Open** меню **File** рабочей среды, либо редактора М-файлов. Открыть файл в редакторе можно и командой MatLab `edit` из командной строки, указав в качестве аргумента имя файла, например:

```
>> edit mydemo
```

Команда `edit` без аргумента приводит к созданию нового файла.

Все примеры, которые встречаются в следующих главах, лучше всего набирать и сохранять в М-файлах, дополняя их комментариями, и выполнять из редактора М-файлов. Применение численных методов и программирование в MatLab, описанное в следующей части книги, требует создания М-файлов.

Типы М-файлов

М-файлы в MatLab бывают двух типов: *файл-программы* (Script M-Files), содержащие последовательность команд, и *файл-функции*, (Function M-Files), в которых описываются функции, определяемые пользователем.

Файл-программы

Файл-программу (файл-процедуру) вы создали при прочтении предыдущего раздела. Все переменные, объявленные в файл-программе, становятся доступными в рабочей среде после ее выполнения. Выполните в редакторе М-файлов файл-программу, приведенную в листинге 5.1, и наберите команду `whos` в командной строке для просмотра содержимого рабочей среды. В командном окне появится описание переменных:

```
>> whos
  Name      Size            Bytes  Class
  f         1x71              568  double array
  g         1x71              568  double array
  x         1x71              568  double array
Grand total is 213 elements using 1704 bytes
```

Переменные, определенные в одной файле-программе, можно использовать в других файл-программах и в командах, выполняемых из командной строки. Выполнение команд, содержащихся в файл-программе, осуществляется двумя способами:

1. Из редактора М-файлов так, как описано выше.
2. Из командной строки или другой файл-программы, при этом в качестве команды используется имя М-файла.

Применение второго способа намного удобнее, особенно, если созданная файл-программа будет неоднократно использоваться впоследствии. Фактически, созданный М-файл становится командой, которую понимает MatLab. Закройте все графические окна и наберите в командной строке `mydemo`, появляется графическое окно, соответствующее командам файл-программы mydemo.m. После ввода команды `mydemo` MatLab производит следующие действия.

1. Проверяет, является ли введенная команда именем какой-либо из переменных, определенных в рабочей среде. Если введена переменная, то выводится ее значение.
2. Если введена не переменная, то MatLab ищет введенную команду среди встроенных функций. Если команда оказывается встроенной функцией, то происходит ее выполнение.
3. Если введена не переменная и не встроенная функция, то MatLab начинает поиск М-файла с названием команды и расширением m. Поиск начинается с *текущего каталога* (Current Directory), если М-файл в нем не найден, то MatLab просматривает каталоги, установленные в *пути поиска* (Path). Найденный М-файл выполняется в MatLab.

Если ни одно из вышеперечисленных действий не привело к успеху, то выводится сообщение в командное окно, например:

```
>> mydem
??? Undefined function or variable 'mydem'.
```

Как правило, М-файлы хранятся в каталоге пользователя. Для того чтобы система MatLab могла найти их, следует установить пути, указывающие расположение М-файлов.

Замечание

Хранить собственные М-файлы вне основного каталога MatLab следует по двум причинам. Во-первых, при переустановке MatLab файлы, которые содержатся в подкаталогах основного каталога MatLab, могут быть уничтожены. Во-вторых, при запуске MatLab все файлы подкаталога toolbox размещаются в памяти компьютера некоторым оптимальным образом так, чтобы увеличить производительность работы. Если вы записали М-файл в этот каталог, то воспользоваться им можно будет только после перезапуска MatLab.

Установка путей

В MatLab версий 5.3 и 6.*x* определяется текущий каталог и пути поиска. Установка этих свойств производится либо при помощи соответствующих диалоговых окон, несколько отличающихся в этих версиях, либо командами из командной строки, одинаковыми для обеих версий.

Установка путей в версии 5.3

Выбор текущего каталога в MatLab 5.3 производится при помощи Path Browser (навигатора путей). Запустите навигатор путей, выбрав пункт **Set Path** меню **File** рабочей среды. Появляется диалоговое окно **Path Browser** навигатора путей, изображенное на рис. 5.3.

В поле ввода **Current Directory** следует ввести полный путь к каталогу, в котором содержатся М-файлы, или выбрать его из диалогового окна **Browse for Folder**, открывающегося при нажатии на кнопку **Browse**. В поле **Files in work** отображаются М-файлы, расположенные в выбранном каталоге. Если М-файлы находятся в нескольких каталогах, следует добавить их в пути поиска, для чего в меню **Path** выберите пункт **Add to Path**. Появляется диалоговое окно **Add to Path**, приведенное на рис. 5.4.

В поле **Directory to add** укажите добавляемый путь, нажав кнопку рядом с полем и выбрав его при помощи появившегося диалогового окна **Browse for Folder**. Для установки приоритета в поиске М-файлов следует выбрать один из переключателей внизу окна. Установленный переключатель **Add to front** означает, что MatLab ищет файлы сначала в добавленном каталоге, а затем во всех остальных, переключатель **Add to back** соответствует низшему при-

оритету поиска. Для добавления пути в навигатор путей нажмите кнопку **OK**. Добавленный каталог появляется в поле **Path** диалогового окна **Path Browser**. В этом поле можно определить порядок поиска — переместить строки с путями при помощи мыши, удерживая левую кнопку.

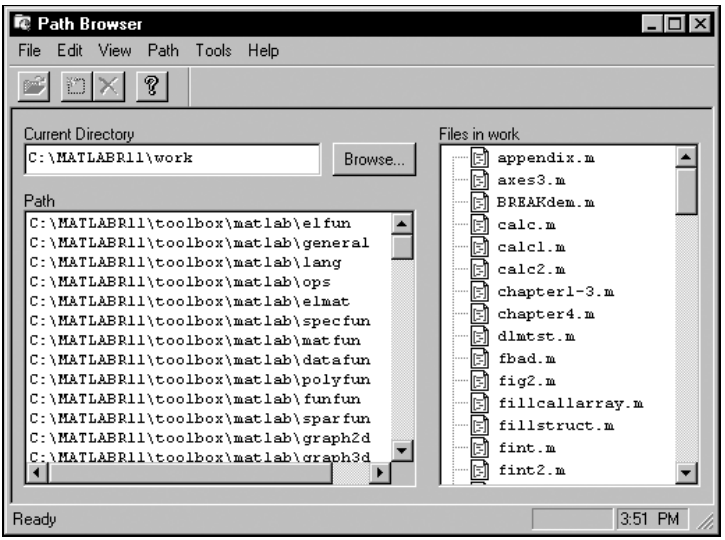


Рис. 5.3. Диалоговое окно **Path Browser** навигатора путей MatLab 5.3

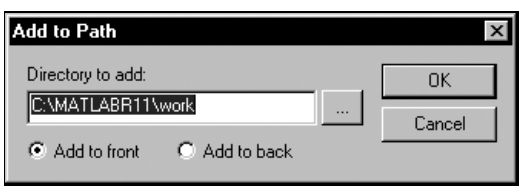


Рис. 5.4. Диалоговое окно **Add to Path** MatLab 5.3

Удаление путей осуществляется выбором пункта **Remove from Path** меню **Path**. Для сохранения установок при последующих запусках MatLab следует выбрать пункт **Save Path** меню **File** навигатора путей. Кнопки, расположенные на панели инструментов, дублируют некоторые пункты меню.

Установка путей в версии 6.*x*

Текущий каталог определяется в диалоговом окне **Current Directory** рабочей среды, изображенном на рис. 5.5. Окно присутствует в рабочей среде, если выбран пункт **Current Directory** меню **View** рабочей среды.

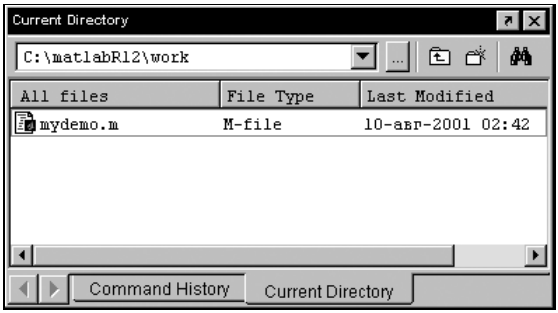


Рис. 5.5. Диалоговое окно **Current Directory** MatLab 6.*x*

Текущий каталог выбирается из списка. Если его нет в списке, то его можно добавить из диалогового окна **Browse for Folder**, вызываемого нажатием на кнопку, расположенную справа от списка. Содержимое текущего каталога отображается в таблице файлов.

Определение путей поиска производится в диалоговом окне **Set Path** навигатора путей, доступ к которому осуществляется из пункта **Set Path** меню **File** рабочей среды. Окно **Set Path** изображено на рис. 5.6.

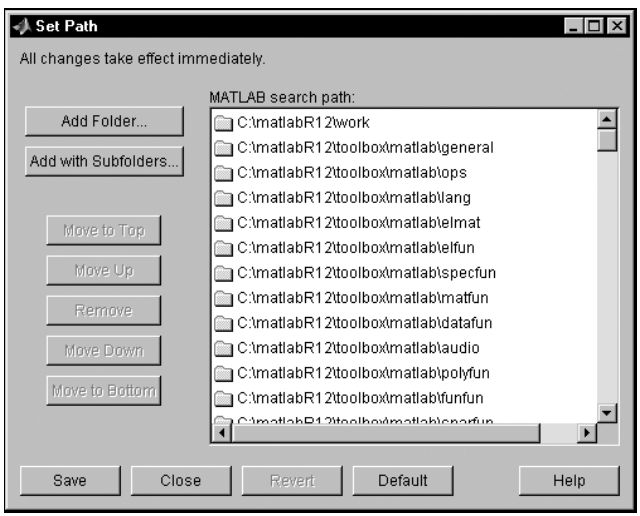


Рис. 5.6. Диалоговое окно **Set Path** MatLab 6.*x*

Для добавления каталога нажмите кнопку **Add Folder** и в появившемся диалоговом окне **Browse for Path** выберите требуемый каталог. Добавление каталога со всеми его подкаталогами осуществляется при нажатии на кнопку **Add with Subfolders**. Путь к добавленному каталогу появляется в поле

MATLAB search path. Порядок поиска соответствует расположению путей в этом поле, первым просматривается каталог, путь к которому размещен вверху списка. Порядок поиска можно изменить или вообще удалить путь к какому-либо каталогу, для чего выделите каталог в поле **MATLAB search path** и определите его положение при помощи следующих кнопок:

- **Move to Top** — поместить вверх списка;
- **Move Up** — переместить вверх на одну позицию;
- **Remove** — удалить из списка;
- **Move Down** — переместить вниз на одну позицию;
- **Move to Bottom** — поместить вниз списка.

После внесения изменений следует сохранить информацию о путях поиска, нажав кнопку **Save**. При помощи кнопки **Default** можно восстановить стандартные установки, а **Revert** предназначена для возврата к сохраненным.

Команды для установки путей

Действия по установке путей в MatLab 5.3 и 6.*x* дублируются командами, одинаковыми для обеих версий. Текущий каталог устанавливается командой `cd`, например `cd c:\users\igor`. Команда `cd`, вызванная без аргумента, выводит путь к текущему каталогу. Для установки путей служит команда `path`, вызываемая с двумя аргументами:

- `path(path,'c:\users\igor')` — добавляет каталог c:\users\igor с низшим приоритетом поиска;
- `path('c:\users\igor',path)` — добавляет каталог c:\users\igor с высшим приоритетом поиска.

Использование команды `path` без аргументов приводит к отображению на экране списка путей поиска. Удалить путь из списка можно при помощи команды `rmpath`.

`rmpath('c:\users\igor')` удаляет путь к каталогу c:\users\igor из списка путей.

Предупреждение

Не удаляйте без необходимости пути к каталогам, особенно к тем, в назначении которых вы не уверены. Удаление может привести к тому, что часть функций, определенных в MatLab, станет недоступной.

Файл-функции

Рассмотренные выше файл-программы являются последовательностью оманд MatLab, они не имеют входных и выходных аргументов. Для использования численных методов и при программировании собственных прило-

жений в MatLab необходимо уметь составлять файл-функции, которые производят необходимые действия с входными аргументами и возвращают результат в выходных аргументах. В этом разделе разобрано несколько простых примеров, позволяющих понять работу с файл-функциями. Файл-функции, так же как и файл-процедуры, создаются в редакторе М-файлов.

Файл-функции с одним входным аргументом

Предположим, что в вычислениях часто необходимо использовать функцию

$$e^{-x} \cdot \sqrt{\frac{x^2+1}{x^4+0.1}}.$$

Имеет смысл один раз написать файл-функцию, а потом вызывать его всюду, где необходимо вычисление этой функции. Откройте в редакторе М-файлов новый файл и наберите текст листинга 5.2.

Листинг 5.2. Файл-функция с одним аргументом

```
function f = myfun(x)
f = exp(-x)*sqrt((x^2+1)/(x^4+0.1));
```

Слово `function` в первой строке определяет, что данный файл содержит файл-функцию. Первая строка является *заголовком функции*, в которой размещается *имя функции* и списки входных и выходных аргументов. В примере, приведенном в листинге 5.2, имя функции `myfun`, один входной аргумент `x` и один выходной — `f`. После заголовка следует *тело функции* (оно в данном примере состоит из одной строки), где и вычисляется ее значение. Важно, что вычисленное значение записывается в `f`. Не забудьте поставить точку с запятой для предотвращения вывода лишней информации на экран.

Теперь сохраните файл в рабочем каталоге. Обратите внимание, что выбор пункта **Save** или **Save as** меню **File** приводит к появлению диалогового окна сохранения файла, в поле **File name** которого уже содержится название myfun. Не изменяйте его, сохраните файл-функцию в файле с предложенным именем!

Теперь созданную функцию можно использовать так же, как и встроенные `sin`, `cos` и другие, например из командной строки:

```
>> y = myfun(1.3)
y =
   0.2600
```

Вызов собственных функций может осуществляться из файл-программы и из другой файл-функции.

Предупреждение

Каталог, в котором содержатся файл-функции, должен быть текущим, или путь к нему должен быть добавлен в пути поиска, иначе MatLab просто не найдет функцию, или вызовет вместо нее другую с тем же именем (если она находится в каталогах, доступных для поиска).

Файл-функция, приведенная в листинге 5.2, имеет один существенный недостаток. Попытка вычисления значений функции от массива приводит к ошибке, а не к массиву значений так, как это происходит при вычислении встроенных функций.

```
>> x = [1.3 7.2];
>> y = myfun(x)
??? Error using ==> ^
Matrix must be square.
Error in ==> C:\MATLABR11\work\myfun.m
On line 2  ==> f = exp(-x)*sqrt((x^2+1)/(x^4+1));
```

Если вы изучили работу с массивами, то устранение этого недостатка не вызовет затруднений. Необходимо просто при вычислении значения функции использовать поэлементные операции.

См. разд. "Поэлементные операции с векторами" главы 2.

Измените тело функции, как указано в листинге 5.3 (не забудьте сохранить изменения в файле myfun.m).

Листинг 5.3. Файл-функция, работающая с массивом значений

```
function f = myfun(x)
f = exp(-x).*sqrt((x.^2+1)./(x.^4+1));
```

Теперь аргументом функции `myfun` может быть как число, так и вектор или матрица значений, например:

```
>> x = [1.3 7.2];
>> y = myfun(x)
y =
    0.2600    0.0001
```

Переменная `y`, в которую записывается результат вызова функции `myfun`, автоматически становится вектором нужного размера.

Постройте график функции `myfun` на отрезке [0, 4] из командной строки или при помощи файл-программы:

```
x = [0:0.5:4];
y = myfun(x);
plot(x, y)
```

MatLab предоставляет еще одну возможность работы с файл-функциями — использование их в качестве аргументов некоторых команд. Например, для построения графика служит специальная функция `fplot`, заменяющая последовательность команд, приведенную выше. При вызове `fplot` имя функции, график которой требуется построить, заключается в апострофы, пределы построения указываются в вектор-строке из двух элементов

```
fplot('myfun',[0 4])
```

Постройте графики `myfun` при помощи `plot` и `fplot` на одних осях, при помощи `hold on`, так, как показано на рис. 5.7. Обратите внимание, что график, построенный при помощи `fplot`, более точно отражает поведение функции, т. к. `fplot` сама подбирает шаг аргумента, уменьшая его на участках быстрого изменения отображаемой функции.

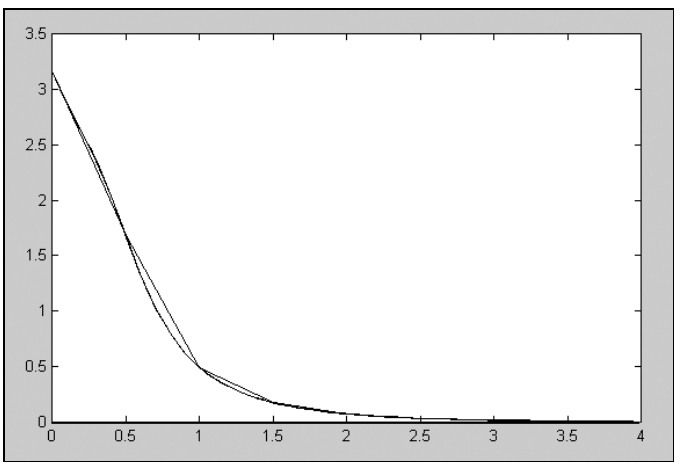


Рис. 5.7. Сравнение `plot` и `fplot`

Файл-функции с несколькими входными аргументами

Написание файл-функций с несколькими входными аргументами практически не отличается от случая с одним аргументом. Все входные аргументы размещаются в списке через запятую. Например, листинг 5.4 содержит файл-функцию, вычисляющую длину радиус-вектора точки трехмерного пространства $\sqrt{x^2 + y^2 + z^2}$.

Листинг 5.4. Файл-функция с несколькими аргументами

```
function r = radius3(x, y, z)
r = sqrt(x.^2 + y.^2 + z.^2);
```

Для вычисления длины радиус-вектора теперь можно использовать функцию `radius3`, например:

```
>> R = radius3(1, 1, 1)
R =
    1.732
```

Кроме функций с несколькими аргументами, MatLab позволяет создавать функции, возвращающие несколько значений, т. е. имеющие несколько выходных аргументов.

Файл-функции с несколькими выходными аргументами

Файл-функции с несколькими выходными аргументами удобны при вычислении функций, возвращающих несколько значений (в математике они называются *вектор-функции*). Выходные аргументы добавляются через запятую в список выходных аргументов, а сам список заключается в квадратные скобки. Хорошим примером является функция, переводящая время, заданное в секундах, в часы, минуты и секунды. Данная файл-функция приведена в листинге 5.5.

Листинг 5.5. Функция перевода секунд в часы, минуты и секунды

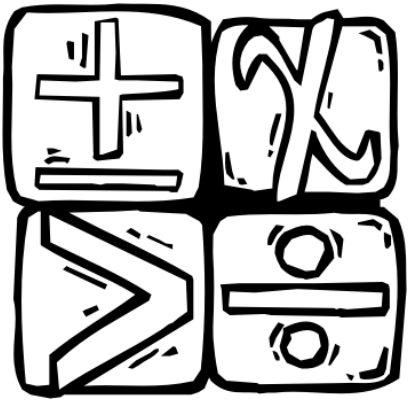
```
function [hour, minute, second] = hms(sec)
hour = floor(sec/3600);
minute = floor((sec-hour*3600)/60);
second = sec-hour*3600-minute*60;
```

При вызове файл-функций с несколькими выходными аргументами результат следует записывать в вектор соответствующей длины:

```
>> [H, M, S] = hms(10000)
H =
     2
M =
    46
S =
    40
```

Если список выходных аргументов пуст, т. е. заголовок выглядит так: `function myfun(a,b)` или `function []=myfun(a,b)`, то файл-функция не будет возвращать никаких значений. Такие функции тоже иногда оказываются полезными.

Предусмотрена также возможность создавать файл-функции, которые сами приспосабливаются к числу входных и выходных аргументов. Большинство встроенных функций работают именно таким образом. Подробно о создании файл-функций и файл-программ написано в части, посвященной программированию собственных приложений.



Часть II

Численные методы и программирование

Глава 6. Вычисления в MatLab

Глава 7. Основы программирования в MatLab

Глава 8. Тонкости программирования

Глава 9. Дескрипторная графика

Глава 6



Вычисления в MatLab

MatLab обладает большим набором встроенных функций, реализующих различные численные методы. Нахождение корней уравнений, интегрирование, интерполирование, решение обыкновенных дифференциальных уравнений, решение задач линейной алгебры, работа с разреженными матрицами — вот далеко не полный перечень возможностей, предоставляемых MatLab. В этой главе разобрано решение типовых задач на применение численных методов.

Исследование функций

Решение уравнений, нахождение максимума или минимума функции одной или нескольких переменных осуществляется вызовом встроенных функций MatLab. Число аргументов встроенных функций может быть различным, в зависимости от требуемого вида результата. Для работы с ними необходимо предварительно написать собственную файл-функцию для вычисления исследуемой функции.

См. разд."Файл-функции" главы 5.

Решение уравнений

Нахождение корней произвольных уравнений осуществляет встроенная функция `fzero`, для определения всех корней полиномов применяется `roots`.

Решение произвольных уравнений

Встроенная функция `fzero` позволяет приближенно вычислить корень уравнения по заданному начальному приближению. В самом простом варианте `fzero` вызывается с двумя входными и одним выходным аргументом `x = fzero('myf', x0)`, где `myf` — имя файл-функции, вычисляющей левую часть уравнения; `x0` — начальное приближение к корню; `x` — найденное приближенное значение корня. Решите, например, на отрезке [−5, 5] уравнение:

$$\sin x - x^2 \cos x = 0 .$$

Перед нахождением корней полезно построить график функции, входящей в левую часть уравнения. Конечно, построить график можно при помощи `plot`, но все равно понадобится написать файл-функцию, поэтому имеет

смысл воспользоваться `fplot`, которая к тому же позволяет получить более точный график по сравнению с `plot`.

Написанию собственных файл-функций посвящен разд. "Файл-функции" главы 5.

В листинге 6.1 приведен текст требуемой файл-функции.

Листинг 6.1. Файл-функция, вычисляющая левую часть уравнения

```
function y = myf(x)
y = sin(x)-x.^2.*cos(x);
```

Теперь постройте график `myf`, используя `fplot`, и нанесите сетку.

```
fplot('myf', [-5 5])
grid on
```

Из графика `myf`, изображенного на рис. 6.1 (пояснения на графике нанесены средствами MatLab), видно, что функция на этом отрезке имеет четыре корня. Один корень равен нулю, в чем несложно убедиться, подставив $x = 0$ в уравнение.

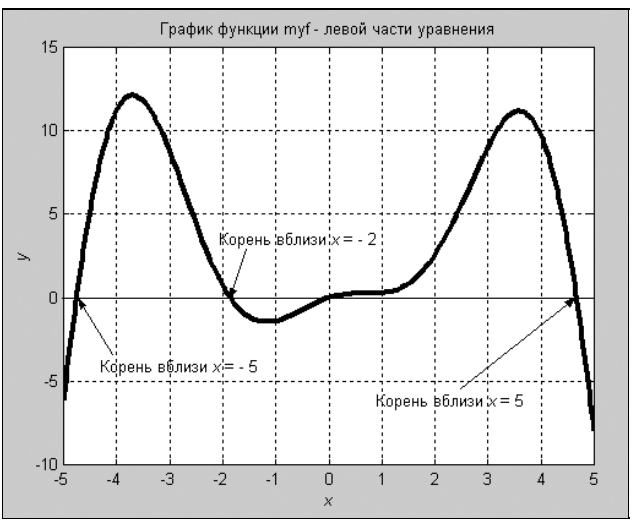


Рис. 6.1. График `myf` — левой части уравнения

Уточните значение корня, расположенного вблизи $x = -5$, при помощи `fzero`:

```
>> x1 = fzero('myf', -5)
Zero found in the interval: [-4.7172, -5.2].
```

```
x1 =
   -4.7566
```

Итак, приближенное значение корня равно -4.7566. Проверьте ответ, вычислив значение функции `myf` в точке `x1`

```
>> myf(x1)
ans =
  2.6645e-015
```

Конечно то, что значение функции близко к нулю, вообще говоря, не означает, что приближенное значение корня расположено достаточно близко к его точному значению. Заданию точности вычислений посвящен отдельный раздел данной главы.

Найдите самостоятельно корни `x2` и `x3`, расположенные около -2 и -5.

Для того чтобы увидеть больше значащих цифр корня `x1`, следует установить формат `long` и вывести `x1` еще раз (точность вычислений не зависит от формата вывода результата!).

```
>> format long
>> x1
x1 =
  -4.75655940570290
```

Возникает вопрос, сколько в ответе точных значащих цифр, т. е. *с какой точностью* найдено решение. Использование `fzero` с двумя аргументами приводит к нахождению корня с точностью `eps`, где `eps` — встроенная функция MatLab, задающая точность вычислений, значение которой можно посмотреть так же, как и значения других переменных:

```
>> eps
ans =
    2.220446049250313e-016
```

Итак, корни уравнения были найдены с точностью ± 2 в шестнадцатом знаке после десятичной точки, т. е. практически с максимально возможной точностью. Проверьте работу `fzero`, вычислив корень `myf`, расположенный вблизи нуля, там, где точное значение корня равно нулю.

```
>>  x4 = fzero('myf', -0.1)
Zero found in the interval: [0.028, -0.19051].
x4 =
   -1.242386505963434e-022
```

Функция `fzero` действительно гарантирует, что точность решения не меньше `eps`.

Важной особенностью `fzero` является то, что она вычисляет только те корни, в которых функция *меняет знак*, а не касается оси абсцисс. Найти корень уравнения $x^2 = 0$ при помощи `fzero` не удается:

```
>>  x4=fzero('myf',-0.1)
Exiting fzero: aborting search for an interval containing a sign change
    because NaN or Inf function value encountered during search
(Function value at 1.372960e+154 is Inf)
Check function or try again with a different starting value.
x4 =
   NaN
```

В данном примере `fzero` пыталась найти промежуток, на границах которого у функции `myf` были бы различные значения, что гарантировало бы существование корня на этом промежутке. Такой промежуток, естественно, определить не удалось, и `fzero` вывела сообщение об ошибке.

Вместо начального приближения вторым параметром `fzero` можно задать интервал, на котором следует найти корень:

```
>>  x2 = fzero('myf', [-3 -1])
Zero found in the interval: [-3, -1].
x2 =
  -1.85392745969615
```

На границах указываемого интервала функция должна принимать значения разных знаков, иначе выведется сообщение об ошибке!

Получить приближенное значение корня и значение функции в этой точке позволяет вызов `fzero` с двумя выходными аргументами:

```
>> [x2, f] = fzero('myf', [-3 -1])
Zero found in the interval: [-3, -1].
x2 =
  -1.85392745969615
f =
   -2.220446049250313e-016
```

В качестве исследуемой функции может выступать и встроенная математическая функция, например

```
>> fzero('sin', [2 4])
Zero found in the interval: [2, 4].
ans =
   3.14159265358979
```

Вычисление всех корней полинома

Полином в MatLab задается вектором его коэффициентов, например для определения полинома $p = x^7 + 3.2x^5 - 5.2x^4 + 0.5x^2 + x - 3$ следует использовать команду

```
>> p = [1 0 3.2 -5.2 0 0.5 1 -3];
```

Число элементов вектора, т. е. число коэффициентов полинома, всегда на единицу больше его степени, нулевые коэффициенты должны содержаться в векторе.

Функция `polyval` предназначена для вычисления значения полинома от некоторого аргумента:

```
>> polyval(p,1)
ans =
   -2.5000
```

Аргумент может быть матрицей или вектором, в этом случае производится поэлементное вычисление значений полинома и результат представляет матрицу или вектор того же размера, что и аргумент.

Нахождение сразу всех корней полиномов осуществляется при помощи функции `roots`, в качестве аргумента которой указывается вектор с коэффициентами полинома. Функция `roots` возвращает вектор корней полинома, в том числе и комплексных:

```
>> r = roots(p)
r =
  -0.5668 + 2.0698i
  -0.5668 - 2.0698i
   1.2149
   0.5898 + 0.6435i
   0.5898 - 0.6435i
  -0.6305 + 0.5534i
  -0.6305 - 0.5534i
```

Число корней полинома, как известно, совпадает со степенью полинома. Убедитесь в правильности работы `roots`, вычислив значение полинома от вектора его корней:

```
>> polyval(p,r)
ans =
  1.0e-012 *
  -0.1008 + 0.0899i
  -0.1008 - 0.0899i
  -0.0666
```

```
0.0027 - 0.0018i
0.0027 + 0.0018i
0.0102 - 0.0053i
0.0102 + 0.0053i
```

Обратите внимание, что в верхней строке результата содержится общий множитель `1.0e-012`, на который следует помножить каждое число получившегося вектора.

Минимизация функций

Встроенные функции MatLab позволяют минимизировать функции одной или нескольких переменных. Результатом является *локальный минимум*, т. е. точка, в окрестности которой исследуемая функция имеет бóльшие значения по сравнению со значением локального минимума.

Минимизация функции одной переменной

Поиск локального минимума функции одной переменной на некотором отрезке осуществляется при помощи `fminbnd`, использование которой схоже с `fzero`. Найдите локальные минимумы функции $e^{-x}\sin 3\pi x$ на отрезке [0, 2]. Требуется предварительно создать соответствующую файл-функцию, назвав ее, к примеру `ftest`.

Написанию собственных файл-функций посвящен разд. "Файл-функции" главы 5.

Перед нахождением локальных минимумов постройте график исследуемой функции командой `fplot`. Из графика, приведенного на рис. 6.2, видно, что исследуемая функция имеет три локальных минимума. Вычислите значение `x`, при котором достигается второй локальный минимум, задав первым аргументом `fminbnd` имя файл-функции, а вторым и третьим — границы отрезка, на котором ищется локальный минимум:

```
>> x2 = fminbnd('ftest', 0.7, 2.0)
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-004
x2 =
    1.1554
```

Итак, второй локальный минимум достигается при $x = 1.1554$. Сообщение, которое вывела `fminbnd`, означает, что решение найдено с точностью 10^{-4}.

Установка точности и дополнительных параметров минимизации описана в разд. "Задание дополнительных параметров" данной главы.

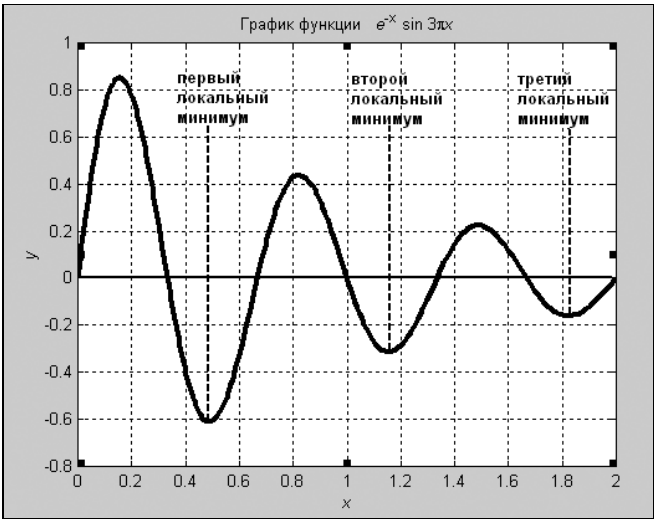


Рис. 6.2. Расположение локальных минимумов

Для одновременного вычисления значения функции в точке минимума следует вызвать `fminbnd` с двумя аргументами:

```
>> [x2, f] = fminbnd('ftest', 0.7, 2.0)
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-004
x2 =
    1.1554
f =
   -0.3132
```

Найдите самостоятельно остальные локальные минимумы и максимумы. Для нахождения локального максимума нет специальной функции, очевидно, что следует искать минимум функции с обратным знаком.

Минимизация функции нескольких переменных

Минимизация функции нескольких переменных является более сложной задачей, чем одной переменной, однако MatLab успешно справляется с ней при помощи `fminsearch`, которая вычисляет локальный минимум по заданному начальному приближению. Разберем использование `fminsearch` на примере нахождения локального минимума функции $f(x, y) = \sin \pi x \cdot \sin \pi y$.

Сначала получите представление о поведении функции, построив ее линии уровня при помощи следующих команд:

```
>> [X,Y] = meshgrid(0:0.01:2);
>> Z = sin(pi*X).*sin(pi*Y);
>> [CMatr, h]= contour(X,Y,Z,[-0.96, -0.9, -0.8, -0.5, -0.1, 0.1 ,0.5,...
  0.8,  0.9, 0.96]);
>> clabel(CMatr, h)
>> colormap(gray)
```

Отображение линий уровня функции двух переменных описано в разд. "Контурные графики" главы 3.

На получившемся графике, приведенном на рис. 6.3, видно расположение локальных минимумов и максимумов.

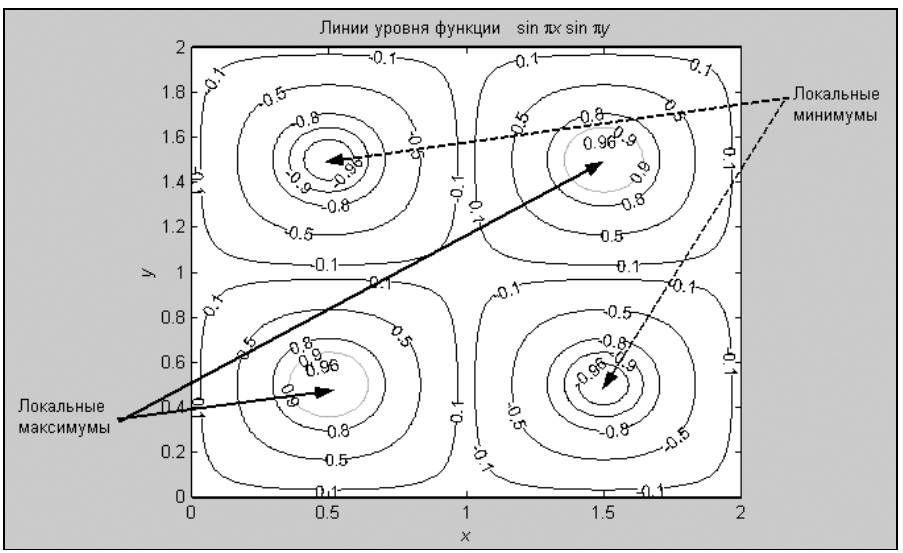


Рис. 6.3. Расположение локальных максимумов и минимумов

Перед применением `fminsearch` необходимо создать файл-функцию, вычисляющую значения искомой функции, причем аргументом файл-функции должен быть вектор, первый элемент которого соответствует переменной *x*, а второй — *y*. Текст требуемой файл-функции `ftest2` приведен в листинге 6.2.

Листинг 6.2. Файл-функция `ftest2`

```
function f = ftest2(argvec)
x = argvec(1);
```

```
y = argvec(2);
f = sin(pi*x).*sin(pi*y);
```

Теперь для нахождения локального минимума вызовите `fminsearch` с двумя входными аргументами — именем файл-функции и начальным приближением и выходным аргументом — вектором с координатами искомой точки минимума:

```
>> M = fminsearch('ftest2', [1.4, 0.6])
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-004
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of
1.000000e-004
M =
    1.5000    0.5000
```

Решение найдено с точностью 10^{-4}, как по значениям x и y, так и по значению функции. Для получения не только вектора с координатами точки минимума, но и значения функции следует вызвать `fminsearch` с двумя выходными аргументами:

```
>> [M, f] = fminsearch('ftest2', [1.4, 0.6])
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-004
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of
1.000000e-004
M =
    1.5000    0.5000
f =
   -1.0000
```

Задание дополнительных параметров

Функции `fzero`, `fminbnd` и `fminsearch` позволяют задать дополнительный параметр `options`, контролирующий вычислительный процесс. Значение `options` следует предварительно сформировать при помощи функции `optimset` в соответствии с характером требуемого контроля. Задание точности 10^{-9} нахождения минимума функции одной переменной осуществляется при помощи следующих команд (имеет смысл использовать формат `long` при повышении точности вычислений):

```
>> format long
>> options = optimset('TolX', 1.0e-09);
>> x2 = fminbnd('ftest', 0.7, 2.0, options)
```

```
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-009
x2 =
   1.15545071659568
```

Сравните полученный результат со значением, полученным `fminbnd` без аргумента `options`. Аналогичным образом точность задается при нахождении корней и минимизации функции нескольких переменных.

В общем случае аргументы `optimset` задаются попарно

```
options = optimset(...,вид контроля, значение,...)
```

причем возможные сочетания параметров `вид контроля` и `значение` приведены в табл. 6.1.

Таблица 6.1. *Параметры* `optimset`

Вид контроля	Значение	Результат
`'Display'`	`'off'`	Информация о вычислительном процессе не выводится
	`'iter'`	Выводится информация о каждом шаге вычислительного процесса
	`'final'`	Выводится информация только о завершении вычислительного процесса (используется по умолчанию)
`'MaxFunEvals'`	Целое число	Максимальное количество вызовов исследуемой функции
`'MaxIter'`	Целое число	Максимальное количество итераций вычислительного процесса
`'TolFun'`	Положительное вещественное число	Точность по функции для останова вычислений
`'TolX'`	Положительное вещественное число	Точность по аргументу функции для останова вычислений

Ограничивать количество вызовов функций и число итераций имеет смысл, если есть опасение, что получить решение не удастся из-за расхождения вычислительного процесса.

Пользователям, имеющим представление о численных методах, полезны сведения о ходе вычислений, выводимые на экран при значении аргумента `'Display'` равном `'iter'`. Последовательность команд

```
>> options = optimset('Display', 'iter','TolX', 1.0e-09);
>> x2 = fminbnd('ftest', 0.7, 2.0, options)
```

приводит к появлению в командном окне кроме результата еще и таблицы, каждая строка которой соответствует одной итерации и содержит информацию о том, какой раз вызывалась исследуемая функция, текущее приближение и значение функции от него, метод, применяемый при данной итерации.

```
Func-count     x          f(x)          Procedure
    1        1.19656    -0.290321        initial
    2        1.50344     0.222246        golden
    3        1.00689    -0.0237026       golden
    ...
    11       1.15545    -0.313158        parabolic
```

Интегрирование функций

В этом разделе описано приближенное вычисление определенных интегралов и двойных определенных интегралов с заданной точностью. Пользователь имеет возможность выбирать наиболее подходящий метод численного интегрирования в зависимости от свойств подынтегральной функции.

Вычисление определенных интегралов

Начнем с примера нахождения значения определенного интеграла

$$\int_{-1}^{1} e^{-x} \sin x \, dx .$$

Первым шагом является создание файл-функции, вычисляющей подынтегральное выражение. Ее текст приведен в листинге 6.3.

Листинг 6.3. Файл-функция, вычисляющая подынтегральное выражение

```
function f = fint(x)
f = exp(-x).*sin(x);
```

Теперь для вычисления интеграла вызовите `quad`, задав первым аргументом имя файл-функции `fint`, а вторым и третьим — нижний и верхний пределы интегрирования. В качестве выходного аргумента можно указать имя переменной, в которую следует записать найденное значение:

```
>> I = quad('fint', -1, 1)
I =
  -0.66349010601463
```

Функция quad вычисляет приближенное значение интеграла с точностью 10^{-3}. Для повышения точности вычислений следует задать дополнительный четвертый аргумент:

```
>> I = quad('fint', -1, 1, 1.0e-06)
I =
  -0.66349366575256
```

Реализованный в quad алгоритм основан на квадратурной формуле Симпсона с автоматическим подбором шага интегрирования для достижения требуемой относительной погрешности. Процесс выбора узлов интегрирования можно наблюдать в графическом окне, если задать пятым параметром функции число, не равное нулю. Окончательный вид графического окна приведен на рис. 6.4 (для наблюдения за выбором узлов интегрирования следите, чтобы графическое окно не перекрывалось другими окнами, расположенными на экране).

```
>> I = quad('fint', -1, 1, 1.0e-06, 10)
I =
  -0.66349366575256
```

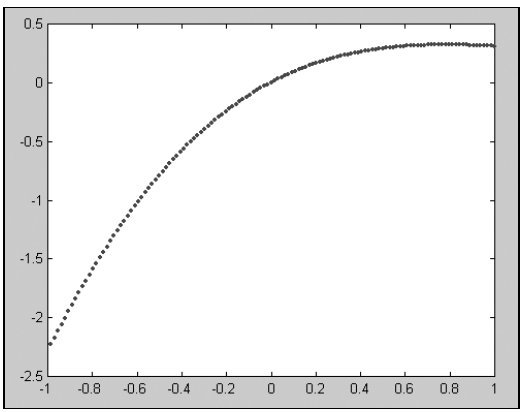


Рис. 6.4. Узлы квадратурной формулы (quad)

Интегрирование достаточно гладких функций имеет смысл производить при помощи quad8, основанной на более точных квадратурных формулах Ньютона—Котеса, так же с автоматическим подбором шага. Этот алгоритм требует меньше вычислений по сравнению с вышеописанным при *одной и той же точности вычислений* и, следовательно, экономит время вычислений. Использование quad8 не отличается от quad. В предыдущем примере замените quad на quad8. Обратите внимание, что узлов интегрирования потребовалось значительно меньше (см. рис. 6.5).

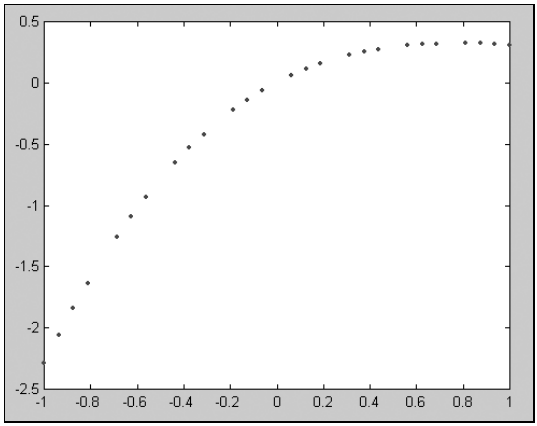


Рис. 6.5. Узлы квадратурной формулы (`quad8`)

При использовании `quad` и `quad8` следует иметь в виду, что автоматический подбор шага интегрирования происходит не более десяти раз. Если достигнут максимально возможный уровень, то выводится предупреждение о том, что функция может иметь особенность, но вычисления продолжаются. Как правило, это происходит, если подынтегральная функция имеет бесконечные производные в точках отрезка интегрирования. Например, при интегрировании $\sqrt{x}$ по отрезку [0, 1] выдается сообщение:

```
Warning: Recursion level limit reached in quad. Singularity likely.
```

К ошибке приводит вычисление в MatLab 5.3 интегралов от функций, имеющих интегрируемую особенность, например $1/\sqrt{x}$ в нуле. Начиная с версии 6.0, алгоритм, реализованный в `quad`, несколько улучшен, в частности, для интегрирования функций с особенностями. Кроме того, в версиях 6.0 и 6.1 вместо устаревшей `quad8` следует использовать `quadl`, которая находит значение определенного интеграла по квадратурным формулам Гаусса—Лобатто.

Вычисление двойных интегралов

В MatLab определена встроенная функция `dblquad` для приближенного вычисления двойных интегралов. Как и в случае вычисления определенных интегралов следует написать файл-функцию для вычисления подынтегрального выражения. Найдите значение интеграла

$$\int_0^1 \int_{-\pi}^{\pi} \left(e^x \sin y + e^{-x} \cos y \right) dx dy .$$

Файл-функция `fint2` должна содержать два входных аргумента x и y, ее текст приведен в листинге 6.4.

Листинг 6.4. Файл-функция, вычисляющая подынтегральную функцию

```
function f = fint2(x, y)
f = exp(x).*sin(y).^2+exp(-x).*cos(y).^2;
```

Функция `dblquad` имеет семь аргументов, при ее вызове необходимо учесть, что первыми задаются пределы внутреннего интеграла по x, а вторыми — внешнего по y:

```
>> dblquad('fint2', -pi, pi, 0, 1)
ans =
   23.0977
```

Дополнительным шестым параметром можно задать точность вычисления интеграла. При вычислении двойного интеграла `dblquad` использует `quad`. Если седьмым аргументом указать `'quad8'`, то вычисления будут основаны на квадратурных формулах Ньютона—Котеса и для достаточно гладких функций интеграл будет вычислен быстрее:

```
>> dblquad('fint2', -pi, pi, 0, 1, 1.0e-012, 'quad8')
ans =
  23.09747871451549
```

Вычисление некоторых интегралов

В данном разделе разобрано несколько примеров: интегрирование параметрически заданной функции и вычисление интеграла с переменным верхним пределом.

Интегралы, зависящие от параметра

Функции `quad` и `quad8` позволяют вычислять интегралы, зависящие от параметра. Аргументами файл-функции, вычисляющей подынтегральное выражение, должна быть не только переменная интегрирования, но и все параметры. Значения параметров указываются через запятую, начиная с шестого аргумента `quad` или `quad8`. Вычислите интеграл

$$\int_{-1}^{1}\left(p_1 x^2 + p_2 \sin x\right)dx$$

при значениях параметров $p_1 = 22.5$, $p_2 = -5.9$ по квадратурным формулам Ньютона—Котеса с автоматическим выбором шага с точностью 10^{-5}.

Текст файл-функции `fparam`, зависящей от *mpex* аргументов, которая вычисляет значение подынтегральной функции, приведен в листинге 6.5.

Листинг 6.5. Параметрически заданная функция

```
function z = fparam(x, Par1, Par2)
z = Par1.*x.^2+Par2.*sin(x);
```

Найдите значение интеграла при помощи `quad`, использование `quad8` производится аналогично.

```
>> q = quad('fparam', -1, 1, 1.0e-05, 1, 22.5, -5.9)
q =
  15.00000005742834
```

Пятый параметр, равный единице, поставлен для наблюдения за процессом интегрирования в графическом окне, появляющемся на экране. В графическом окне отображается выбор узлов интегрирования. Если пропустить этот параметр, то появится сообщение об ошибке, т. к. файл-функция `fparam` предполагает наличие трех входных аргументов. Итак, при вычислении интеграла, зависящего от параметров, их следует указывать, начиная с шестого аргумента `quad` или `quad8`.

Интегралы с переменным верхним пределом

Интеграл с переменным верхним пределом представляет собой некоторую функцию, например

$$F(y) = \int_0^y e^x (\sin x - \cos x) dx .$$

Для вычисления такого интеграла придется написать две файл-функции: `fint` для, подынтегральной функции и `fy`, которая находит значения интеграла для каждого значения `z`. Тексты требуемых файл-функций приведены в листингах 6.6, 6.7.

Листинг 6.6. Файл-функция, вычисляющая значение подынтегральной функции

```
function f = fint(x)
f = exp(x).*(sin(x)+cos(x));
```

Листинг 6.7. Файл-функция, вычисляющая значение интеграла

```
function f = Fy(y)
f = quad8('fint', 0, y, 1.0e-06);
```

Теперь можно вычислить интеграл при любом значении верхнего предела, задав его в качестве аргумента `Fy`. Построение графика зависимости интеграла от верхнего предела осуществляется при помощи `fplot`

```
>> fplot('Fy', [0, pi])
```

Полиномы и интерполяция

Полиномы в MatLab представляются в виде вектора коэффициентов.

Задание полинома, нахождение всех его корней и вычисление значений описано в разд. "Вычисление всех корней полинома" данной главы.

Текущий раздел посвящен операциям с полиномами, определенными в MatLab, и решению задачи интерполяции при построении полинома, наилучшим образом приближающего двух- и трехмерные табличные данные.

Операции с полиномами

Умножение, деление, сложение и вычитание

Умножение двух полиномов осуществляется при помощи `conv`. Например, для вычисления произведения $s(x)$ полиномов

$$p(x) = x^5 + x^3 + 1 \qquad q(x) = x^2 + 2x + 3$$

следует создать два вектора их коэффициентов и использовать их в качестве аргументов `conv`:

```
>> p = [1 0 1 0 0 1];
>> q = [1 2 3];
>> s = conv(p, q)
s =
     1     2     4     2     3     1     2     3
```

В результате получается полином седьмой степени, соответствующий вектору *s*

$$s(x) = x^7 + 2x^6 + 4x^5 + 2x^4 + 3x^3 + x^2 + 2x + 3\,.$$

Встроенная функция `deconv` осуществляет деление полиномов с остатком. Она вызывается с двумя выходными аргументами — частным и остатком от деления:

```
>> [d, r] = deconv(p, q)
d =
     1    -2     2     2
r =
     0     0     0     0   -10    -5
```

Размер вектора, содержащего коэффициенты остатка, равен максимальному из размеров векторов, соответствующих делимому полиному и его делителю.

Для сложения и вычитания полиномов в MatLab нет специальной функции. В то же время, использование знака плюс для нахождения суммы полиномов разной степени приведет к ошибке, т. к. нельзя складывать векторы разных размеров. Если вы изучили работу с массивами и создание файл-функций, то написание собственной функции для нахождения суммы полиномов не представляет большого труда. Алгоритм сложения полиномов, которым соответствуют векторы p и q, достаточно прост, требуется:

1. Выбрать максимальный размер из двух векторов.
2. Соответствующим образом преобразовать каждый из двух векторов к максимальному размеру.
3. Сложить новые векторы.

В листинге 6.8 приведен текст файл-функции `ploysum`, находящей значение суммы полиномов.

Листинг 6.8. Файл-функция `polysum`, вычисляющая сумму полиномов

```
function s = polysum(p, q)
% нахождение наибольшей из длин входных векторов
maxlen = max(length(p), length(q));
% создание вспомогательных векторов длины maxlen,
% имеющих нулевые элементы
p1 = zeros(1, maxlen);
q1 = zeros(1, maxlen);
% преобразование исходных векторов к векторам одинакового
% размера maxlen
p1(maxlen-length(p)+1:maxlen) = p;
q1(maxlen-length(q)+1:maxlen) = q;
% вычисление коэффициентов полинома, являющегося суммой исходных
% полиномов
s = p1 + q1;
```

Теперь для нахождения суммы и разности полиномов следует использовать `polysum`.

```
>> s = polysum(p, q)
s =
     1     0     1     1     2     4
>> d = polysum(p, -q)
d =
     1     0     1    -1    -2    -2
```

Вычисление производных

Встроенная функция `plolyder` предназначена для вычисления производной не только от полинома, но и от произведения и частного двух полиномов. Вызов `polyder` с одним аргументом — вектором, соответствующим полиному, приводит к вычислению вектора коэффициентов производной полинома:

```
>> p = [1 0 1 0 0 1];
>> p1 = polyder(p)
p1 =
     5     0     3     0     0
```

Для вычисления производной от произведения полиномов следует использовать `polyder` с двумя входными аргументами:

```
>> p = [1 0 1 0 0 1];
>> q = [1 2 3];
>> pq1 = polyder(p, q)
pq1 =
     7    12    20     8     9     2     2
```

Если необходимо найти производную отношения двух полиномов в виде дроби, числитель и знаменатель которой так же являются полиномами, то следует вызвать `polyder` с двумя выходными аргументами:

```
>> [n, d] = polyder(p, q)
n =
     3     8    16     4     9    -2    -2
d =
     1     4    10    12     9
```

Первый аргумент `n` результата содержит коэффициенты числителя, а второй `d` — знаменателя получающегося отношения полиномов.

Интерполирование

Табличные данные очень часто удобно интерпретировать как некоторую функцию, в частности полиномиальную или *сплайн* (непрерывную, гладкую функцию, которая на отрезках области определения равна полиномам определенной степени). Возникает задача о построении полиномиальной или кусочно-полиномиальной функции, сплайна, для приближения некоторых исходных данных. MatLab имеет встроенные функции для приближения сплайнами как одномерных, так и многомерных данных. Самым простым способом интерполирования, предлагаемым MatLab, является приближение полиномом методом наименьших квадратов.

Приближение по методу наименьших квадратов

Построение полинома заданной степени, который приближает функцию одной переменной, заданную таблицей значений, производится при помощи `polyfit`. Пусть исследуемая функция дана в виде табл. 6.2.

Таблица 6.2. *Функция, заданная таблицей*

x_i	0.1	0.2	0.4	0.5	0.6	0.8	1.2
y_i	–3.5	–4.8	–2.1	0.2	0.9	2.3	3.7

Приблизьте ее полиномами четвертой, пятой и шестой степени и выведите график, отражающий характер приближений. Для решения этой задачи создайте файл-программу `LSInterp`, текст которой приведен в листинге 6.9.

Листинг 6.9. Приближение полиномами методом наименьших квадратов

```
x = [0.1 0.2 0.4 0.5 0.6 0.8 1.2];
y = [-3.5 -4.8 -2.1 0.2 0.9 2.3 3.7];
% вывод графика табличной функции маркерами
plot(x, y, 'ko')
% вычисление коэффициентов полиномов разных степеней,
% приближающих табличную функцию по методу наименьших
% квадратов
p4 = polyfit(x, y, 4);
p5 = polyfit(x, y, 5);
p6 = polyfit(x, y, 6);
% построение графиков полиномов
t = [0.1:0.01:1.2];
P4 = polyval(p4, t);
P5 = polyval(p5, t);
P6 = polyval(p6, t);
hold on
plot(t, P4, 'k-', t, P5, 'k:', t, P6, 'k-.')
legend('табличные данные', 'n=4', 'n=5', 'n=6',0)
title('Приближение табличной функции полиномами степени n')
```

В результате выполнения файл-функции `LSInterp` получается график, изображенный на рис. 6.6. Обратите внимание, что приближение методом наименьших квадратов дает хороший результат не всегда, кроме того, при увеличении степени возможно ухудшение приближения, как происходит, например при n, равном шести в нашем случае. Тем не менее, данная ин-

терполяция используется, например, для построения полиномиальной регрессии при моделировании данных. Обычно при интерполяции таблично заданной функции применяются сплайны для получения плавного перехода от одного значения к другому.

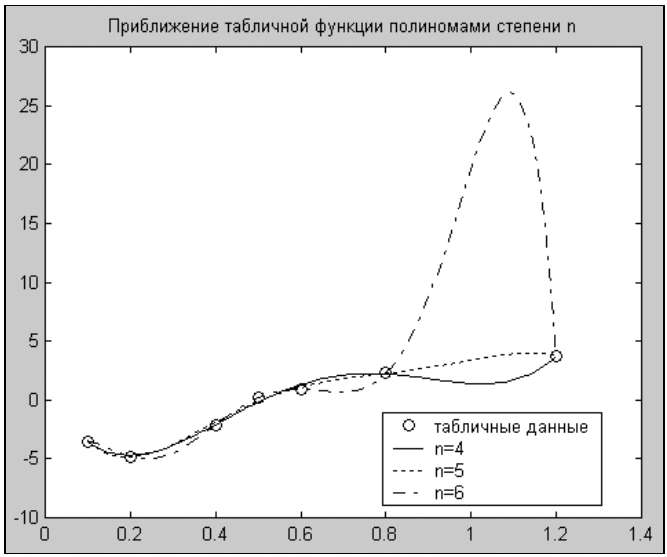


Рис. 6.6. Приближение полиномами по методу наименьших квадратов (`polyfit`)

Интерполяция сплайнами

Самым простым способом является интерполяция данных, при которой значение в каждой промежуточной точке принимается равным ближайшему значению, заданному в таблице (так называемая *интерполяция по соседним элементам*). *Линейная интерполяция* приводит к соединению соседних точек отрезками прямых согласно соответствующим табличным данным. Для получения более гладкой функции следует применять *интерполяцию кубическими сплайнами*. Все эти способы реализованы встроенной функцией `interp1`, для использования которой следует задать координаты абсцисс промежуточных точек, в которых вычисляются значения интерполянта, и способ интерполирования:

- `'nearest'` — интерполяция по соседним элементам;
- `'linear'` — линейная интерполяция;
- `'spline'` — интерполяция кубическими сплайнами.

Выходным аргументом `interp1` является вектор значений интерполянта. Текст файл-программы `interp1dem` для сравнения различных способов интерполирования приведен в листинге 6.10.

Листинг 6.10. Файл-программа `interp1dem`

```
% задание табличной функции
x = [0.1 0.2 0.4 0.5 0.6 0.8 1.2];
y = [-3.5 -4.8 -2.1 0.2 0.9 2.3 3.7];
% вывод графика табличной функции маркерами
plot(x, y, 'ko')
% задание промежуточных точек для интерполирования
xi = [x(1):0.01:x(length(x))];
ynear = interp1(x,y,xi,'nearest');
yline = interp1(x,y,xi,'linear');
yspline = interp1(x,y,xi,'spline');
hold on
plot(xi, ynear, 'k', xi, yline, 'k:', xi, yspline, 'k-.')
title('Различные способы интерполяции функций')
xlabel('\itx')
ylabel('\ity')
legend('табличная функция', 'по соседним элементам (nearest)', ...
       'линейная (linear)', 'кубические сплайны (spline)',4)
```

Выполнение `interp1dem` приводит к появлению графика, изображенного на рис. 6.7, который наглядно демонстрирует способы интерполяции.

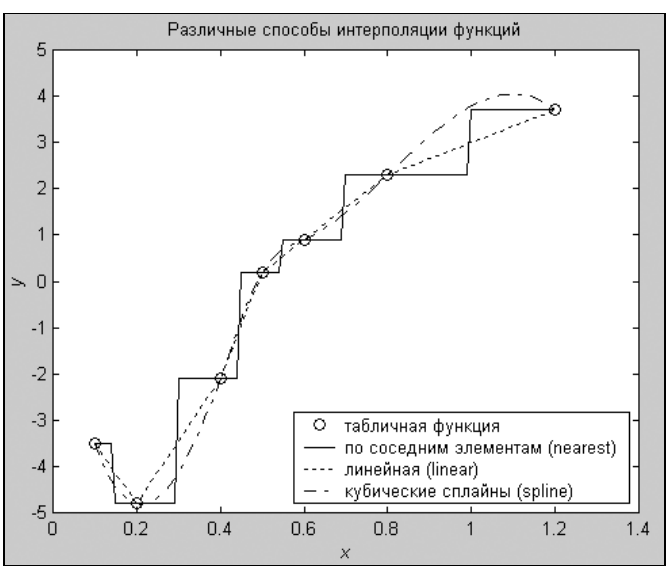


Рис. 6.7. Различные способы интерполяции функций (`interp1`)

Замечание

Для ускорения вычислений в случае *равноотстоящих точек*, соответствующих данным, используются, соответственно, аргументы `'*nearest'`, `'*linear'` или `'*spline'`.

MatLab позволяет интерполировать не только одномерные, но также двумерные и многомерные данные.

Интерполяция двумерных и многомерных данных

Интерполяция двумерных данных связана с построением функции двух переменных, приближающей заданные в точках (x_i, y_i) значения z_i. Для интерполирования двумерных данных следует задать промежуточные узлы командой `meshgrid` и воспользоваться `interp2`, которая реализует один из способов интерполирования, в зависимости от значения последнего аргумента:

- `'nearest'` — интерполяция по соседним элементам;
- `'bilinear'` — билинейная интерполяция;
- `'bicubic'` — интерполяция бикубическими сплайнами.

Сравнение вышеперечисленных способов интерполяции может быть осуществлено при помощи файл-программы `interp2dem`, текст которой приведен в листинге 6.11. Для избежания утомительного ввода таблицы двумерных данных, они генерируются при помощи некоторой функции двух переменных.

Листинг 6.11. Файл-программа `interp2dem`

```
% генерирование значений табличной функции
[X, Y] = meshgrid(0:0.2:1);
Z = sin(3*pi*X).*sin(3*pi*Y).*exp(-X.^2-Y.^2);
% визуализация табличной функции
subplot(2, 2, 1)
surf(X, Y, Z)
title('табличная функция')
% создание сетки для промежуточных значений
[Xi, Yi] = meshgrid(0:0.02:1);
% интерполяция по соседним значениям
ZiNear = interp2(X, Y, Z, Xi, Yi, 'nearest');
% билинейная интерполяция
ZiBiLin = interp2(X, Y, Z, Xi, Yi, 'bilinear');
% бикубическая интерполяция
ZiBiCub = interp2(X, Y, Z, Xi, Yi, 'bicubic');
% вывод результатов
subplot(2, 2, 2)
```

```
surf(Xi, Yi, ZiNear)
title('по соседним значениям (near)')
subplot(2, 2, 3)
surf(Xi, Yi, ZiBiLin)
title('билинейная (bilinear)')
subplot(2, 2, 4)
surf(Xi, Yi, ZiBiCub)
title('бикубическая (bicubic)')
```

На рис. 6.8 приведены графики, полученные при помощи `interp2dem`. Графики отражают поведение интерполирующих функций, полученных различными способами.

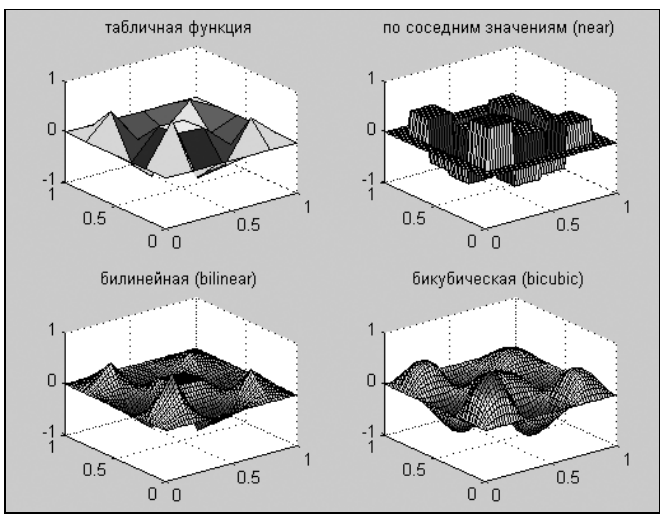


Рис. 6.8. Различные способы интерполяции двумерных данных (`interp2`)

Для интерполяции трехмерных данных служит функция `interp3`, для многомерных — `interpn`. Создание многомерных сеток осуществляется функцией `ndgrid`. Многомерное интерполирование производится аналогично двумерному, подробнее о нем можно узнать из справочной системы по MatLab.

В этом разделе разобраны самые простые способы приближения табличных функций. Интерполированию посвящен Spline ToolBox, содержащий функции для решения следующих задач:

- представление сплайнов в кусочной форме и B-форме;
- преобразование сплайнов в B-форме в кусочное представление;
- интерполяция и сглаживание при помощи кубических сплайнов;

- вычисление производных, интегралов и отображений от сплайнов;
- оптимальное расположение узлов сплайна.

Задачи линейной алгебры

Поскольку название MatLab является сокращением от Matrix Laboratory (матричная лаборатория), то естественно предположить, что возможности MatLab для задач линейной алгебры достаточно широки. Действительно, встроенные функции MatLab позволяют вычислять нормы матриц и векторов, обращать матрицы, решать системы линейных уравнений, в том числе переопределенные и недоопределенные (с прямоугольными матрицами), находить собственные числа и векторы, факторизовать матрицы, вычислять функции матриц. В приложениях очень часто встречаются разреженные матрицы (содержащие большое число нулевых элементов). Для эффективного решения задач с разреженными матрицами существуют специальные алгоритмы, многие из которых реализованы в MatLab. Для того чтобы осознанно использовать богатые возможности, предоставляемые MatLab для решения задач линейной алгебры с разреженными матрицами, необходимо, как минимум, обладать знаниями в объеме программы технического вуза.

Разреженным матрицам посвящена глава 17.

Данный раздел описывает простейшие возможности, которые предоставляет MatLab для решения систем линейных уравнений, вычисления определителей, нахождения собственных чисел и векторов.

Системы уравнений, определители, обращение матриц

Знак обратной косой черты \ предназначен для решения систем линейных алгебраических уравнений, слева от него записывается матрица системы, а справа — вектор правой части. MatLab сама подбирает наиболее эффективный метод и решает систему.

Решение системы уравнений рассмотрено в качестве примера в разд. "Решение систем линейных уравнений" главы 2.

Использование знака \ является самым простым способом решения, однако, даже при таком простом подходе возможны существенные затруднения.

Системы с плохо обусловленными матрицами

Рассмотрим простой пример: требуется найти решение следующей системы:

$$\begin{cases} 2x_1 + 3x_2 + 3x_3 = 8; \\ 4x_1 + 2x_2 + 3x_3 = 7; \\ 6x_1 + 5x_2 + 6x_3 = 7. \end{cases}$$

Введите матрицу и вектор из командной строки и примените обратную косую черту:

```
>> A = [2 3 3
       4 2 3
       6 5 6];
>> b = [8; 7; 7];;
>> x = A\b
```

В командное окно выводится предупреждение о том, что матрица вырождена или плохо обусловлена, и решение `x`:

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.306145e-017.
x =
  1.0e+016 *
    0.9007
    1.8014
   -2.4019
```

Полученное решение неверно, в чем несложно убедиться проверкой, умножив A на x. Дело в том, что матрица A является вырожденной, ее определитель равен нулю. Для вычисления определителя предназначена встроенная функция `det`:

```
>> det(A)
ans =
     0
```

Однако, "очень маленькое" значение определителя еще не означает, что при решении системы возникнут трудности. Например, система

$$\begin{cases} 1.0 \cdot 10^{-40} x_1 + 2.0 \cdot 10^{-41} = 1; \\ 2.0 \cdot 10^{-41} x_1 + 3.0 \cdot 10^{-40} = 2. \end{cases}$$

решается точно, никаких сообщений не выводится, хотя определитель матрицы системы равен $2.96 \cdot 10^{-80}$ (убедитесь в этом, решив систему и вычислив определитель!).

Возможность нахождения решения системы линейных алгебраических уравнений определяется *числом обусловленности* матрицы. Если оно сравнимо с точностью вычислений (MatLab производит вычисления с двойной точностью, удерживая шестнадцать значащих цифр), то ответ, скорее всего, получится неверным. Перед решением системы имеет смысл вычислить число обусловленности матрицы системы при помощи встроенной функции `cond`,

задав аргументом матрицу. Например, для первой системы функция `cond` выдает следующий результат:

```
>> cond(A)
ans =
  2.7526e+016
```

Число обусловленности матрицы второй системы равно 3.0808 и система решается правильно.

Замечание

На самом деле число обусловленности матрицы первой системы равно бесконечности. MatLab использует численные методы для нахождения числа обусловленности, которые в применении к плохо обусловленным матрицам могут давать неверный результат. Но все равно, большое значение числа обусловленности сигнализирует о том, что результат решения системы может быть неправильным.

MatLab позволяет решать системы с прямоугольными матрицами, так называемые *переопределенные* системы, в которых уравнений больше неизвестных, и *недоопределенные* системы с числом уравнений меньшим числа неизвестных.

Переопределенные и недоопределенные системы

Переопределенные системы встречаются в задачах подбора нескольких параметров с целью удовлетворения соотношениям, число которых больше, чем число параметров.

Одна из таких задач — приближение полиномом табличной функции методом наименьших квадратов — была рассмотрена в разд. "Приближение по методу наименьших квадратов" данной главы.

Рассмотрим задачу о подборе параметров a, b некоторого физического закона

$$y = a \cdot e^{-t} + b \cdot t$$

по результатам измерения величины y в моменты времени, приведенные в табл. 6.3.

Таблица 6.3. *Результаты измерений физической величины*

t_i	0	0.1	0.2	0.3	0.4	0.5
y_i	4.25	3.95	3.64	3.41	3.21	3.04

Для нахождения параметров потребуем соответствия измерений физическому закону, т. е. выполнение шести равенств вида:

$$y_i = a \cdot e^{-t_i} + b \cdot t_i .$$

Эти равенства являются не чем иным, как переопределенной системой из шести линейных алгебраических уравнений с двумя неизвестными a и b, матрица A и вектор правой части Y системы имеют вид

$$A = \begin{bmatrix} e^{-t_1} & t_1 \\ e^{-t_2} & t_2 \\ \vdots & \\ e^{-t_6} & t_6 \end{bmatrix}, \qquad Y = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_6 \end{bmatrix}.$$

Для нахождения неизвестных следует решить эту систему в MatLab при помощи знака обратной косой черты. Напишите самостоятельно файл-программу, решающую данную систему и строящую маркерами график исходных данных и линией график физического закона с получившимися параметрами. Физический закон лучше оформить в виде файл-функции от трех аргументов — переменной t и параметров a, b. Тексты файл-программы `fit` и файл-функции `law` содержатся в листингах 6.13, 6.14.

Листинг 6.13. Файл-функция `law` для физического закона

```
function y = law(t, a, b);
y = a.*exp(-t) + b.*t;
```

Листинг 6.14. Файл-программа `fit` подбора параметров

```
t = [0; 0.1; 0.2; 0.3; 0.4; 0.5];
y = [4.25; 3.95; 3.64; 3.41; 3.21; 3.04];
A = [exp(-t)  t];
x = A\y;
a = x(1)
b = x(2)
T = [0:0.01:0.5];
F = law(T, a, b);
plot(t, y, 's', T, F)
```

Обратите внимание, что в результате работы файл-программы `fit` получаются не только графики, изображенные на рис. 6.9. В командное окно выводятся также найденные значения параметров:

```
>> a =
    4.2478
b =
    0.9070
```

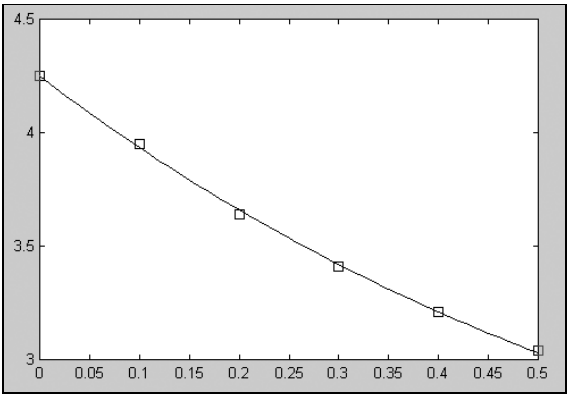


Рис. 6.9. Подбор параметров физического закона

Внесите самостоятельно в файл-программу `fit` некоторые изменения. Пусть табличные данные считываются из текстового файла, а на график наносится вся необходимая информация.

Перейдем теперь к недоопределенным системам, число неизвестных в которых больше числа уравнений. В самом простом случае недоопределенная система состоит из одного уравнения с двумя неизвестными, которое имеет бесконечно много решений. При нахождении решения недоопределенной системы MatLab ищет *базисное решение*, содержащее как можно больше нулей. Базисное решение тоже может быть не единственным. Решите в качестве упражнения систему

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 2; \\ 3x_1 + 4x_2 + 5x_3 = 2 \end{cases}$$

и произведите проверку, умножив матрицу системы на найденное решение.

Обращение матриц

Обратной к квадратной *невырожденной* матрице A называется такая матрица A^{-1}, которая при умножении на A справа и слева дает в результате единичную матрицу. Встроенная функция `inv` обращает матрицу, ее входным аргументом является исходная матрица, а выходным — обратная. Введите квадратную матрицу размерностью три самостоятельно, убедитесь в ее невырожденности при помощи `det` и обратите, а затем проверьте результат умножением справа и слева полученной матрицы на исходную. В принципе, решение систем линейных уравнений с квадратной матрицей может осуществляться умножением обратной матрицы на вектор правой части системы. Однако такой способ требует больше времени и памяти, к тому же он может дать бóльшую погрешность решения. Поэтому для решения систем следует применять знак обратной косой черты.

Встроенная функция MatLab `pinv` позволяет находить *псевдообратную* матрицу к исходной *прямоугольной*.

```
>> A = [1 2;   3 4;   5 6];
>> P = pinv(A);
>> P*A
ans =
    1.0000    0.0000
   -0.0000    1.0000
```

Собственные числа и векторы матрицы, функции матриц

Собственные числа λ_i и *собственные векторы* u_i квадратной матрицы A удовлетворяют равенствам $A \cdot u_i = \lambda_i u_i$. Функция `eig` с входным аргументом матрицей и выходным — вектором записывает в него все собственные числа матрицы:

```
>> A = [2 3; 3 5];
>> lam = eig(A)
lam =
    0.1459
    6.8541
```

Для одновременного вычисления всех собственных векторов и чисел следует вызвать `eig` с двумя выходными аргументами.

```
>> [U, Lam] = eig(A);
```

Первый выходной аргумент `U` является матрицей, составленной из собственных векторов. Для доступа, например к первому собственному вектору, следует использовать индексацию при помощи двоеточия

```
>> u1 = U(:,1);
```

Вторым выходным аргументом `Lam` возвращается диагональная матрица, содержащая собственные числа исходной матрицы.

```
>> Lam
Lam =
    0.1459         0
         0    6.8541
```

Возведение квадратной матрицы в любую целую степень осуществляется при помощи знака `^`, например:

```
>> B = A^2
B =
    13    21
    21    34
```

Обратите внимание, что при возведении матрицы в целую положительную степень происходит *матричное* умножение матрицы на саму себя столько раз, каков показатель степени, в отличие от поэлементного умножения при помощи оператора `.^`. Для отрицательных степеней вычисляется степень обратной матрицы. Возможно использование дробных степеней. Если требуется извлечь корень из квадратной матрицы, то лучше применить встроенную функцию `sqrtm`. Матричные экспонента и логарифм вычисляются при помощи функций `expm` и `logm`.

Пользователь может найти значение любой функции от матрицы. Для этого следует создать собственную файл-функцию, а затем использовать встроенную функцию `funm`, задав первым аргументом матрицу, а вторым имя файл-функции в апострофах. Вычислите, например $e^A \cdot \sin A$. Текст соответствующей файл-функции `matrf` приведен в листинге 6.15. При создании файл-функции обязательно использование поэлементных операций!

Листинг 6.15. Файл-функция `matrf` для вычисления функции матрицы

```
function B = matrf(A)
B = exp(A).*sin(A);
```

Теперь при помощи `funm` найдите значение заданной функции от матрицы *B*:

```
>> B = funm(A, 'matrf')
B =
  141.6829  228.9756
  228.9756  370.6585
```

Для контроля точности вычислений функции от матрицы лучше использовать вызов `funm` с двумя выходными аргументами. Во второй аргумент помещается информация о точности вычислений. Встроенные математические функции MatLab (`sin`, `cos` и др.) можно вычислять от матриц аналогичным образом, например

```
B = funm(A,'sin').
```

Замечание

Предпочтительнее производить вычисления функций от матриц с использованием встроенных матричных функций, т. е. `expm` вместо `exp` и т. д., для чего следует разбить вычисления на несколько этапов. Во встроенных матричных функциях реализованы специальные алгоритмы, работающие точнее и надежнее, чем общий алгоритм `funm`.

Функции матриц имеют широкую область применения, в частности решение системы линейных дифференциальных уравнений с начальным условием

$$\frac{dX}{dt} = A \cdot X, \qquad X(0) = X_0$$

находится по формуле $X(t) = X_0 \cdot e^{tA}$. Решению дифференциальных уравнений и систем посвящен следующий раздел.

Решение дифференциальных уравнений

Данный раздел посвящен описанию простейших возможностей, предоставляемых MatLab, для численного решения обыкновенных дифференциальных уравнений произвольного порядка и систем с начальными условиями, т. е. *задачи Коши*. Для решения предназначены встроенные функции MatLab, в вычислительной математике их называют *солверы*. MatLab имеет достаточно большой набор солверов, реализующих различные методы решения краевых задач.

Схема решения задач с начальными условиями

Задача Коши для дифференциального уравнения состоит в нахождении функции, удовлетворяющей дифференциальному уравнению произвольного порядка

$$y^{(n)} = f(t, y, y', \ldots, y^{(n-1)})$$

и начальным условиям при $t = t_0$

$$y(t_0) = u_0, \qquad y'(t_0) = u_1, \ \ldots, \qquad y^{(n-1)}(t_0) = u_{n-1}.$$

Задачи такого вида можно достаточно просто решать в MatLab. Схема решения состоит из следующих этапов:

1. Приведение дифференциального уравнения к системе дифференциальных уравнений первого порядка.
2. Написание специальной файл-функции для системы уравнений.
3. Вызов подходящего солвера.
4. Визуализация результата.

Разберем решение дифференциальных уравнений на примере задачи о колебаниях под воздействием внешней силы в среде, оказывающей сопротивление колебаниям. Уравнение, описывающее движение, имеет вид

$$y'' + 2y' + 10y = \sin t\,.$$

Пример носит демонстрационный характер, поэтому размерности физических величин указываться не будут.

Предположим, что координата точки в начальный момент времени равнялась единице, а скорость — нулю. Тогда соответствующие начальные условия выглядят так

$$y(0)=1\,,\qquad y'(0)=0\,.$$

Теперь исходную задачу надо привести к системе дифференциальных уравнений. Для этого вводят столько вспомогательных функций, каков порядок уравнения. В данном случае необходимы две вспомогательные функции y_1 и y_2, определяемые формулами

$$y_1=y\,,\qquad y_2=y'\,.$$

Несложно догадаться, что система дифференциальных уравнений с начальными условиями, требуемая для дальнейшей работы, такова

$$\begin{cases} y_1' = y_2 \\ y_2' = -2y_2 - 10y_1 + \sin t \end{cases} \qquad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Второй этап состоит в написании файл-функции для системы дифференциальных уравнений. Файл-функция должна иметь два входных аргумента: переменную *t*, по которой производится дифференцирование, и вектор, размер которого равен числу неизвестных функций системы. Число и порядок аргументов фиксированы, даже если *t* явно не входит в систему. Выходным аргументом файл-функции является вектор правой части системы. Текст файл-функции `oscil` для разбираемого примера приведен в листинге 6.16.

Листинг 6.16. Файл-функция `oscil`, соответствующая системе дифференциальных уравнений

```
function F = oscil(t, y)
F = [y(2);   -2*y(2)-10*y(1)+sin(t)];
```

Решите задачу, используя, например солвер `ode45`. Входными аргументами солверов в самом простом случае являются: имя файл-функции в апострофах, вектор с начальным и конечным значением времени наблюдения за колебаниями и вектор начальных условий. Выходных аргументов два: вектор, содержащий значения времени, и матрица значений неизвестных функций в соответствующие моменты времени. Значения функций расположены по столбцам матрицы, в первом столбце — значения первой функции, во втором — второй и т. д. В силу проделанных замен $y_1=y$, $y_2=y'$, *первый столбец матрицы* содержит как раз значения неизвестной функции

y(*t*), *входящей в исходное дифференциальное уравнение*, а остальные столбцы — значения ее производных. Как правило, размеры матрицы и вектора достаточно велики, поэтому лучше сразу отобразить результат на графике. Применение солвера для нахождения решения при $t \le 15$ и визуализация результата продемонстрированы на примере файл-программы `solvdem`, приведенной в листинге 6.17.

Листинг 6.17. Файл-программа `solvdem` для решения дифференциального уравнения

```
% формирование вектора начальных условий
Y0 = [1; 0];
% вызов солвера от файл-функции oscil, начального и конечного
% момента времени и вектора начальных условий
[T, Y] = ode45('oscil', [0 15], Y0);
% вывод графика решения исходного дифференциального уравнения
plot(T, Y(:,1), 'r')
% вывод графика производной от решения исходного
% дифференциального уравнения
hold on
plot(T, Y(:,2), 'k--')
% вывод пояснений на график
title('Решение {\ity} \prime\prime+2{\ity} \prime+10{\ity}=sin{\itt}')
xlabel('\itt')
ylabel('{\ity},  {\ity} \prime ')
legend('координата', 'скорость', 4)
grid on
hold off
```

В результате выполнения файл-программы `solvdem` на экран выводятся графики, изображенные на рис. 6.10, которые отражают поведение координаты точки и ее скорости в зависимости от времени. Из графика видно, что приближенное решение и его производная удовлетворяют начальным условиям, колебание происходит в установившемся режиме, начиная с $t = 5$.

Решение системы дифференциальных уравнений с начальными условиями, соответствующими исходной задаче, было получено при помощи солвера `ode45`, который использует метод Рунге—Кутта четвертого порядка. Кроме солвера `ode45`, MatLab имеет еще ряд солверов. При выборе солвера для решения задачи необходимо учитывать свойства системы дифференциальных уравнений, иначе можно получить неточный результат, или затратить слишком много времени на решение. В следующем разделе на примере ре-

шения системы уравнений Лотка—Вольтерра демонстрируется важность соответствия солвера решаемой задаче.

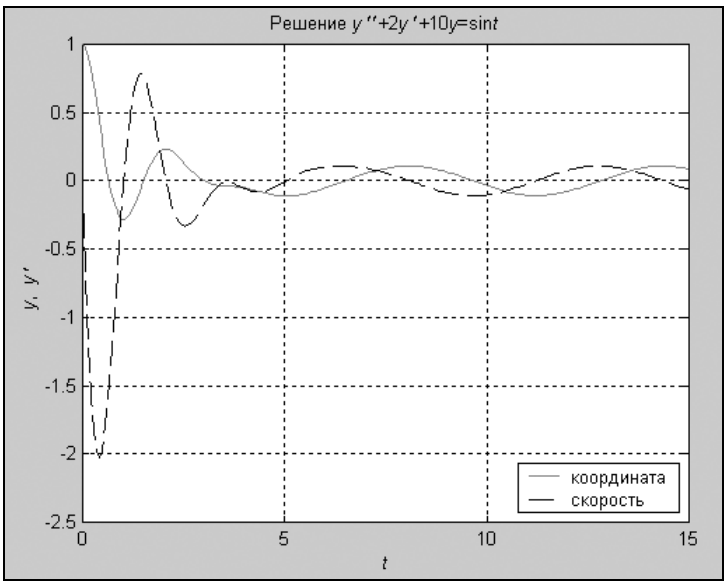


Рис. 6.10. Решение дифференциального уравнения

Решение уравнений Лотка—Вольтерра

В качестве объекта исследования возьмем модель Лотка—Вольтерра борьбы за существование. Обозначим: $y_1(t)$ — число жертв, $y_2(t)$ — число хищников. Число хищников и жертв в течение времени t изменяется по закону

$$\begin{cases} y_1' = P \cdot y_1 - p \cdot y_1 y_2; \\ y_2' = -R \cdot y_2 + r \cdot y_1 y_2, \end{cases}$$

где P — увеличение числа жертв в отсутствие хищников; R — уменьшение числа хищников в отсутствие жертв. Вероятность поедания хищником жертвы пропорциональна их числу $y_1 y_2$, при этом слагаемое $-p \cdot y_1 y_2$ соответствует вымиранию жертв, а $r \cdot y_1 y_2$ — появлению хищников. Решением этой системы на плоскости $y_1 y_2$ является *замкнутая кривая*.

Возьмите для примера $P = 3$, $R = 2$, $p = r = 1$, считайте, что в начальный момент времени было три жертвы и четыре хищника. Решите самостоятельно солвером `ode45` эту систему дифференциальных уравнений с начальными условиями для $t \le 100$, а затем используйте `ose23s` (задание его аргументов производится аналогично). Выведите в графическое окно на разные графики

решения, полученные при помощи ode45 и ode23, и сравните их. Листинги 6.18, 6.19 содержат тексты необходимых файл-функции и файл-программы.

Решением должна быть замкнутая кривая, а вычисления по умолчанию в ode45 и ode23s происходят при одной точности. Графики приближенных решений, приведенные на рис. 6.11, сильно отличаются друг от друга. Приближенное решение, полученное солвером ode23s, намного точнее, чем в случае ode45. Дело в том, что уравнения Лотка—Вольтерра являются примером так называемых *жестких систем*, для решения которых следует использовать специально приспособленные солверы. В справке по MatLab приведен еще один пример жесткой задачи — уравнение Ван-дер-Поля с большим значением параметра, для которой солвер ode45 не может найти решения.

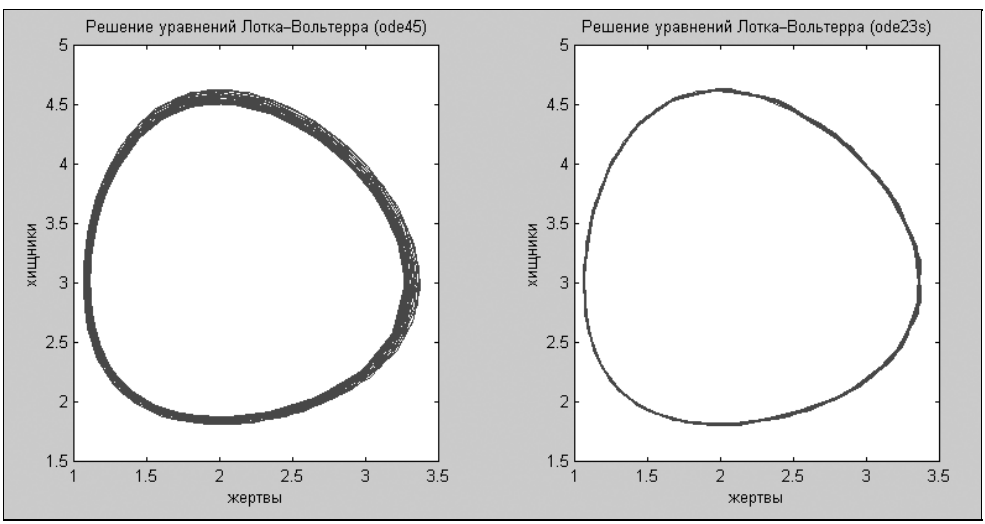


Рис. 6.11. Сравнение солверов ode45 и ode23s

Итак, при решении в MatLab дифференциальных уравнений и систем с начальными условиями следует правильно выбирать солвер в зависимости от свойств исследуемой задачи. В следующем разделе приведен список солверов, доступных в MatLab, с краткими описаниями областей применения.

Листинг 6.18. Файл-функция, соответствующая системе Лотка—Вольтерра

```
function F = LotVol(t, y)
F = [3*y(1)-y(1)*y(2);  -2*y(2)+y(1)*y(2)];
```

Листинг 6.19. Файл-программа для сравнения ode45 и ode23s

```
% формирование вектора начальных условий
Y0 = [3; 4];
% вызов солвера ode45
[T, Y] = ode45('LotVol', [0 100], Y0);
% вывод графика решения исходного дифференциального уравнения
% в виде параметрической кривой
subplot(1, 2, 1)
plot(Y(:,1), Y(:,2))
% вывод пояснений на график
title('Решение уравнений Лотка—Вольтерра (ode45)')
xlabel('жертвы')
ylabel('хищники')
%вызов солвера ode23s
[T, Y] = ode23s('LotVol', [0 100], Y0);
% вывод графика решения исходного дифференциального уравнения
% в виде параметрической кривой
subplot(1, 2, 2)
plot(Y(:,1), Y(:,2))
% вывод пояснений на график
title('Решение уравнений Лотка—Вольтерра (ode23s)')
xlabel('жертвы')
ylabel('хищники')
```

Управление процессом решения

Эффективное решение дифференциальных уравнений невозможно без понимания основных вопросов, связанных с численными методами. Солверы MatLab не являются "черными ящиками". Пользователю необходимо выбрать подходящий солвер, в зависимости от свойств решаемой задачи, и произвести необходимые установки, обеспечивающие получение приближенного решения с требуемыми свойствами, например с заданной точностью.

Солверы для решения задач с начальными условиями

В данном разделе описана стратегия применения солверов MatLab для решения задач для обыкновенных дифференциальных уравнений или систем с начальными условиями. Читатели, имеющие представление о численных методах решения дифференциальных уравнений, могут воспользоваться описанием алгоритмов солверов, которые приведены в справке по MatLab.

Очень часто солвер ode45 дает вполне хорошие результаты, им стоит воспользоваться в первую очередь. Он основан на формулах Рунге—Кутта четвертого и пятого порядка точности. Солвер ode23 также основан на формулах Рунге—Кутта, но уже более низкого порядка точности. Имеет смысл применять ode23 в задачах, содержащих небольшую жесткость, когда требуется получить решение с невысокой степенью точности. Если же требуется получить решение нежесткой задачи с высокой точностью, то наилучший результат даст ode113, основанный на методе переменного порядка Адамса—Бэшфорта—Милтона. Солвер ode113 оказывается особенно эффективным для нежестких систем дифференциальных уравнений, правые части которых вычисляются по сложным формулам. Все солверы пытаются найти решение с относительной точностью 10^{-3}. Попробуйте увеличить точность вычислений так, как описано в следующем разделе.

Если все попытки применения ode45, ode23, ode113 не приводят к успеху, то возможно, что решаемая система является жесткой. Для решения жестких систем подходит солвер ode15s, основанный на многошаговом методе Гира, который допускает изменение порядка. Если требуется решить жесткую задачу с невысокой точностью, то хороший результат может дать солвер ode23s, реализующий одношаговый метод Розенброка второго порядка.

Простейшее использование вышеперечисленных солверов производится так же, как и ode45.

См. разд. "Схема решения задач с начальными условиями" данной главы.

При решении практических задач важно контролировать вычисления. Все солверы допускают задание ряда параметров, позволяющих повысить эффективность вычислений в зависимости от решаемой задачи. В частности, при решении жестких задач задание якобиана системы позволяет увеличить быстродействие вычислений. Одной из важнейших характеристик приближенного решения является его *точность*.

Задание точности вычислений

Точность или погрешность вычислений оказывают существенное влияние на качество полученного приближенного решения. Для управления работой солверов ode45, ode23, ode113, ode15s, ode23s, ode23t и ode23tb, в частности для задания точности, используется дополнительный параметр options, который следует сформировать функцией odeset. Использование odeset сходно с применением optimset при управлении процессом минимизации

```
options = odeset(...,вид контроля, значение,...)
```

Установка параметров минимизации описана в разд. "Задание дополнительных параметров" данной главы.

Полный список возможных параметров приведен в справочной системе MatLab. Его рассмотрение выходит за рамки данной книги. Следует иметь в виду, что неоправданное изменение многих параметров может повлечь уменьшение эффективности солвера или получение неверных результатов. Убедиться в том, что заданных по умолчанию значений, в частности относительной погрешности 10^{-3}, не всегда достаточно для получения хорошего приближения можно на следующем простом примере.

Решите систему дифференциальных уравнений

$$\begin{cases} y_1' = y_2; \\ y_2' = -1/t^2 \end{cases}$$

на отрезке $[a, 100]$ при начальных условиях $y_1(a) = \ln a$, $y_2(a) = 1/a$, взяв $a = 0.001$.

Легко проверить, что точным решением этой системы является $y_1 = \ln t$, $y_2 = 1/t$.

Напишите самостоятельно файл-функцию и файл-процедуру для решения данной системы солвером `ode45`. Расположите на одном графике точное и приближенное решение. Результат, приведенный на рис. 6.12, является довольно неожиданным для погрешности 10^{-3} (используемой по умолчанию). Применение других солверов не улучшает ситуацию.

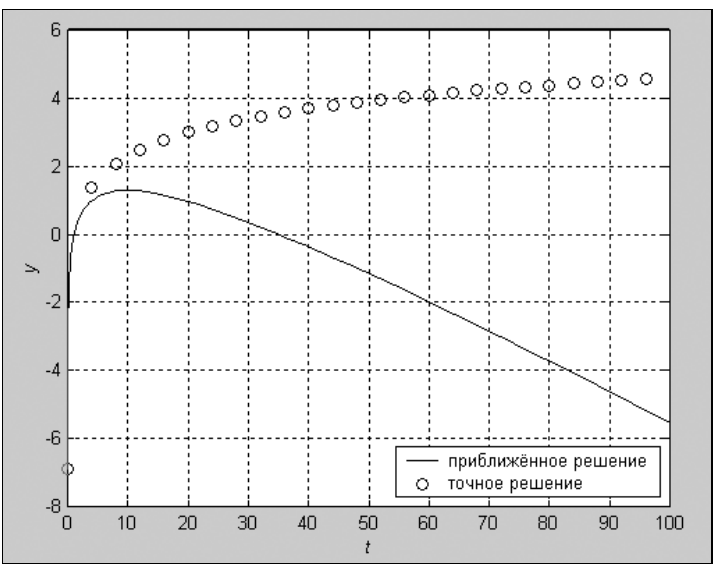


Рис. 6.12. Сравнение приближенного решения с точным (погрешность по умолчанию)

Выход состоит в уменьшении относительной погрешности вычислений при помощи формирования `options` с использованием `odeset` и включении `options` дополнительным четвертым аргументом в солвер. Для задания относительной погрешности служит аргумент `'RelTol'`, например

```
options = odeset('RelTol', 1.0e-04)
```

Дополните созданную файл-программу вызовами `ode45`, предваряя каждое обращение к солверу установкой точности. Не забывайте включать `options` в список аргументов солвера! При возникновении затруднений обратитесь к листингу 6.20. Выведите графики приближенных решений для погрешностей 10^{-3}, 10^{-4}, 10^{-6} так, как показано на рис. 6.13.

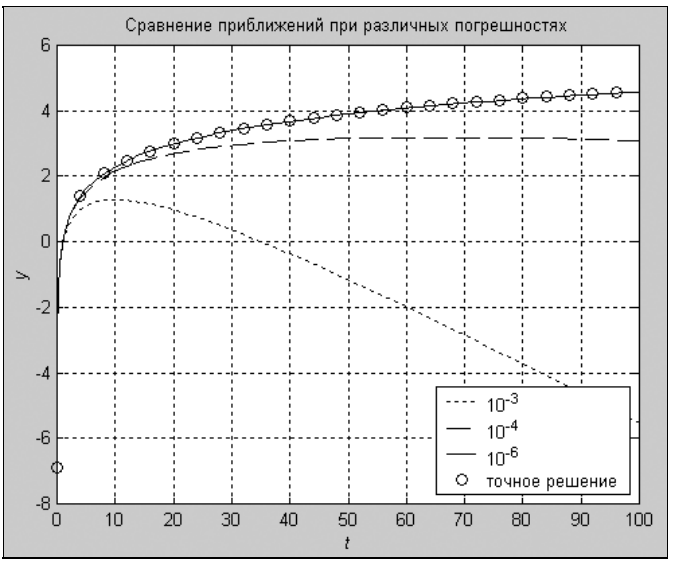


Рис. 6.13. Сравнение приближенного решения с точным при различных погрешностях

Только точность 10^{-6} обеспечивает получение приближенного решения, график которого почти совпадает с графиком точного решения.

Листинг 6.20. Файл-программа для исследования влияния погрешности

```
a = 0.001;
% задание начальных условий
Y0 = [log(a); 1/a];
% вызов солвера ode45 при различных погрешностях
% и вывод графиков приближенных решений
```

```
% точность 0.001
options = odeset('RelTol', 1.0e-3);
[T, Y] = ode45('rsbad', [a 100], Y0, options);
plot(T, Y(:,1), 'k:')
% точность 0.0001
options = odeset('RelTol', 1.0e-4);
[T, Y] = ode45('rsbad', [a 100], Y0, options);
hold on
plot(T, Y(:,1), 'k--')
% точность 0.000001
options = odeset('RelTol', 1.0e-6);
[T, Y] = ode45('rsbad', [a 100], Y0, options);
hold on
plot(T, Y(:,1), 'k-')

% вывод графика точного решения
t = [a:4:100];
y = log(t);
hold on
plot(t, y, 'ko')
% нанесение информации на график
xlabel('\itt')
ylabel('\ity')
title('Сравнение приближений при различных погрешностях')
legend('10^{-3}', '10^{-4}', '10^{-6}', 'точное решение', 4)
grid on
hold off
```

Возможности MatLab не исчерпываются решением задачи Коши для системы дифференциальных уравнений. Задачи математической физики, описываемые уравнениями в частных производных, могут быть эффективно решены при помощи Partial Differential Equation ToolBox. Данный ToolBox содержит приложение с графическим интерфейсом, работе с которым посвящена отдельная глава.

См. главу 16.

Следующий раздел освещает новые возможности решения *граничных задач* для обыкновенных дифференциальных уравнений, появившиеся в MatLab начиная с версии 6.0.

Решение граничных задач

В данном разделе на простых примерах иллюстрируются возможности, *появившиеся в версии 6.0*, для решения граничных задач для обыкновенных дифференциальных уравнений при помощи солвера `bvp4c`.

Схема решения

Рассмотрим решение граничных задач общего вида для обыкновенного дифференциального уравнения второго порядка. Требуется решить уравнение

$$y'' = f(x, y, y')$$

на отрезке $[a, b]$, причем решение должно удовлетворять следующим условиям на границах отрезка

$$\alpha \cdot y(a) + \beta \cdot y'(a) = A, \qquad \gamma \cdot y(b) + \delta \cdot y'(b) = B.$$

Здесь α, β, γ, δ, A, B — заданные числа.

Решение этой задачи состоит из следующих этапов.

1. Преобразование дифференциального уравнения второго порядка к системе двух уравнений первого порядка.
2. Написание файл-функции для вычисления правой части системы.
3. Написание файл-функции, определяющей граничные условия.
4. Формирование начального приближения при помощи специальной функции `bvpinit`.
5. Вызов солвера `bvp4c` для решения граничной задачи.
6. Визуализация результата.

Первые два этапа выполняются практически так же, как и при решении задачи Коши.

Решению задачи Коши посвящен разд. "Решение дифференциальных уравнений" данной главы.

Введение вспомогательных функций $y_1(x)$ и $y_2(x)$ приводит к формированию системы уравнений первого порядка относительно них:

$$\begin{cases} y_1' = y_2; \\ y_2' = f(x, y_1, y_2). \end{cases}$$

Файл-функция правой части системы зависит от x и вектора y, состоящего из двух компонент, $y(1)$ соответствует y_1, а $y(2)$ — y_2. Файл-функция правой части составляется так же, как при решении задачи Коши.

Граничные условия также требуется записать для вспомогательных функций так, чтобы в *правых частях стояли нули*:

$$\alpha \cdot y_1(a) + \beta \cdot y_2(a) - A = 0\,, \qquad \gamma \cdot y_1(b) + \delta \cdot y_2(b) - B = 0\,.$$

Файл-функция, описывающая граничные условия, зависит от двух аргументов — векторов *ya* и *yb* и имеет следующую структуру (см. листинг 6.21):

```
function g = bound(ya, yb)
g = [alpha*ya(1) +  beta*ya(2)-A;    yamma*yb(1) + delta*yb(2)];
```

Вместо `alpha`, `beta`, `gamma`, `delta`, `A` и `B` следует подставить заданные числа.

Выбор начального приближения может оказать влияние на решение, выдаваемое солвером `bvp4c`. MatLab находит приближенное решение граничных задач методом конечных разностей, т. е. получающееся решение есть *вектор значений* неизвестных функций в точках отрезка (в узлах сетки). Аргументами функции `bvpinit`, предназначенной для задания начального приближения, являются вектор с координатами узлов сетки на $[a, b]$ и вектор из двух элементов, содержащий постоянное начальное приближение для функций y_1, y_2. Заданная сетка может быть изменена солвером в процессе решения для обеспечения требуемой точности. Вызов `bvpinit` выглядит следующим образом:

```
initsol = bvpinit(вектор сетки, вектор постоянных значений функций)
```

После определения начального приближения вызывается солвер `bvp4c`, входными аргументами которого являются имена файл-функций правой части системы и граничных условий, начальное приближение и, при необходимости, дополнительные параметры для управления вычислительным процессом. Дополнительные параметры формируются при помощи `bvpset`. Выходным аргументом является *структура*, содержащая информацию о сетке, выбранной MatLab, значения неизвестных функций и их производных. Представление данных в виде структуры не было описано выше.

Работа со структурами подробно описана в главе 8.

В следующем разделе показывается, как использовать структуру для визуализации решения граничных задач.

Простой пример граничной задачи

Решите граничную задачу для обыкновенного дифференциального уравнения второго порядка

$$y'' = -\sin x\,, \qquad y(0) = 0\,, \qquad y'(11\pi/2) + y(11\pi/2) = -1\,.$$

и сравните полученное решение с точным — $y = \sin x$.

Система дифференциальных уравнений первого порядка, соответствующая исходному уравнению, и граничные условия для нее находятся просто:

$$\begin{cases} y_1' = y_2; \\ y_2' = -\sin x, \end{cases} \qquad y_1(0) = 0\,, \qquad y_2(11\pi/2) + y_1(11\pi/2) + 1 = 0\,.$$

Написать файл-функцию `rside` для системы уравнений также не представляет большого труда, ее текст приведен в листинге 6.21.

Листинг 6.21. Файл-функция `rside` для системы уравнений

```
function f = rside(x, y)
f = [y(2);    -sin(x)];
```

Как было указано в предыдущем разделе, файл-функция для граничных условий имеет два входных аргумента. Каждый аргумент является вектором значений неизвестных функций y_1 и y_2 в начальной и конечной точках промежутка. Поэтому файл-функция граничных условий должна быть такой, как в листинге 6.22.

Листинг 6.22. Файл-функция `bound` граничных условий

```
function f = bound(ya, yb)
f = [ya(1);    yb(2)+yb(1)+1]
```

Решение граничной задачи оформите в виде файл-программы, в которой необходимо задать начальное приближение при помощи `bvpinit`, вызвать солвер `bvp4c`, получить результат и вывести его на одном графике с точным решением для сравнения. Текст этой файл-программы содержится в листинге 6.23.

Листинг 6.23. Файл-программа для решения граничной задачи

```
% РЕШЕНИЕ ГРАНИЧНОЙ ЗАДАЧИ
% формирование начального приближения
% начальное приближение y1 = 1, y2 = 0
initsol = bvpinit([0:pi/2:11*pi/2], [1 0]);
% вызов солвера
sol = bvp4c('rside', 'bound', initsol);
% использование полей x и y структуры sol для построения решения
% в sol.x содержатся координаты сетки
% в sol.y матрица
```

```
% sol.y(1,:) соответствует значениям функции y1 в sol.x(:)
% sol.y(2,:) соответствует значениям функции y2 в sol.x(:)
plot(sol.x, sol.y(1,:), 'k.')
% вывод графика точного решения для сравнения
x = [0:pi/30:11*pi/2];
hold on
plot(x, sin(x), 'b-')
hold off
```

Следует обратить внимание на то, как работать с полученным результатом, хранящимся в структуре `sol`. Структура `sol` содержит *поля*, доступ к которым осуществляется при помощи размещения имени поля после имени структуры через точку. Проводя аналогию с выходными аргументами солверов для решения задачи Коши (см. листинг 6.17), можно сказать, что вектор T теперь соответствует полю `sol.x`, а матрица Y с двумя столбцами со значениями неизвестных функций y_1 и y_2 полю `sol.y`.

Получающиеся графики приближенного и точного решений исходной задачи приведены на рис. 6.14, они свидетельствуют о том, что для простых задач солвер `bvp4c` дает хорошие результаты.

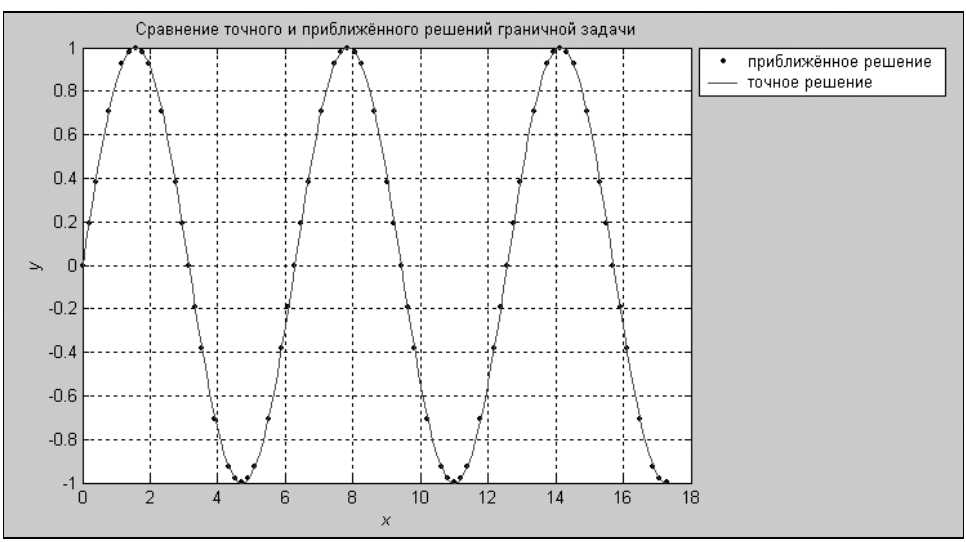


Рис. 6.14. Сравнение приближенного и точного решений граничной задачи

Глава 7



Основы программирования в MatLab

Файл-функции и файл-программы, используемые в предыдущей главе, являются самыми простыми примерами программ. Все команды MatLab, содержащиеся в них, выполняются *последовательно*. Для решения многих более серьезных задач требуется писать программы, в которых действия повторяются циклически, в зависимости от некоторых условий выполняются различные части программы.

В данной главе описаны основные операторы, задающие последовательность выполнения команд MatLab. Операторы можно использовать как в файл-процедурах, так и в файл-функциях, что позволяет создавать функции со сложной разветвленной структурой.

Операторы цикла

Выполнение схожих повторяющихся действий в MatLab осуществляется при помощи операторов циклов `for` и `while`. Цикл `for` предназначен для выполнения заданного числа повторяющихся действий, а `while` — для действий, число которых заранее не известно, но известно условие продолжения цикла.

Цикл *for*

Самое простое использование `for` осуществляется следующим образом:

```
for count = start:step:final
     команды MatLab
end
```

Здесь `count` — *переменная* цикла, `start` — ее начальное значение, `final` — конечное значение, а `step` — шаг, на который увеличивается `count` при каждом следующем заходе в цикл. Цикл заканчивается как только значение `count` становится больше `final`. Переменная цикла может принимать не только целые, но и вещественные значения любого знака. Разберем применение цикла `for` на некоторых характерных примерах.

Пусть требуется вывести семейство кривых для $x \in [0, 2\pi]$, которое задано функцией, зависящей от параметра $y(x,a) = e^{-ax}\sin x$, для значений пара-

метра от −0.1 до 0.1. Можно, конечно, последовательно вычислять $y(x, a)$ и строить ее графики для различных значений a от −0.1 до 0.1, но гораздо удобнее использовать цикл `for`. Наберите текст файл-процедуры, приведенный в листинге 7.1, в редакторе M-файлов, сохраните в файле FORdem1.m и запустите его на выполнение (или из редактора M-файлов, или из командной строки, набрав в ней команду `FORdem1` и нажав <Enter>).

Листинг 7.1. Файл-программа `FORdem1` для вывода семейства кривых

```
figure
x = [0:pi/30:2*pi];
for a = -0.1:0.02:0.1
   y = exp(-a*x).*sin(x);
   hold on
   plot(x, y)
end
```

Замечание

Редактор M-файлов автоматически предлагает расположить операторы внутри цикла с отступом от левого края. Используйте эту возможность для удобства работы с текстом программы.

В результате выполнения `FORdem1` появится графическое окно, изображенное на рис. 7.1, которое содержит требуемое семейство кривых.

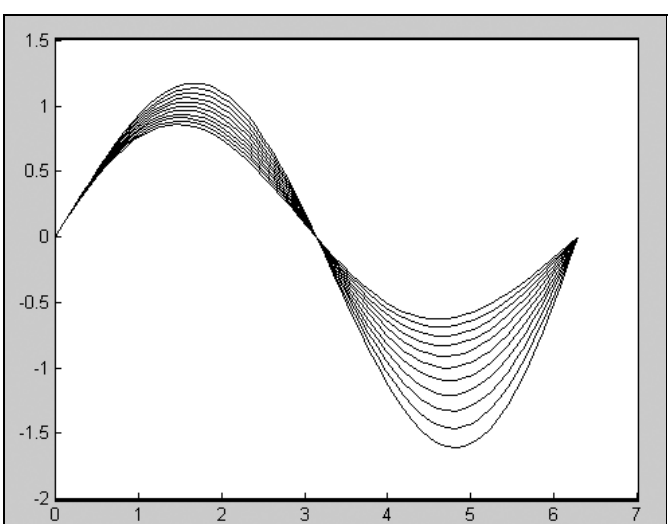


Рис. 7.1. Семейство кривых

Напишите файл-программу для вычисления суммы

$$S = \sum_{k=1}^{10} \frac{1}{k!}.$$

Алгоритм вычисления суммы использует *накопление* результата, т. е. сначала сумма равна нулю, затем в переменную `k` заносится единица, вычисляется $1/k!$ (то есть 1/1!), добавляется к `S` и результат *снова заносится* в `S`. Далее `k` увеличивается на единицу, и процесс продолжается, пока последним слагаемым не станет 1/10!. Файл-программа `FORdem2`, приведенная в листинге 7.2, вычисляет искомую сумму.

Листинг 7.2. Файл-программа `FORdem2` для вычисления суммы

```
% ФАЙЛ-ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ СУММЫ
%    1/1!+1/2!+...+1/10!

% обнуление S для накопления суммы
S = 0;
% накопление суммы в цикле
for k = 1:10
   S = S + 1/factorial(k);
end
% вывод результата в командное окно
```

Замечание

Если шаг цикла равен 1, то его можно не указывать.

Наберите файл-программу в редакторе M-файлов, сохраните в текущем каталоге в файле FORdem2.m и выполните ее. Результат отображается в командном окне, т. к. в последней строке файл-программы содержится `S` без точки с запятой для вывода значения переменной `S`

```
S =
    1.7183
```

Обратите внимание, что остальные строки файл-программы, которые могли бы повлечь вывод на экран промежуточных значений, завершаются точкой с запятой для подавления вывода в командное окно.

Первые две строки с комментариями не случайно отделены пустой строкой от остального текста программы. Именно они выводятся на экран, когда

пользователь при помощи команды `help` из командной строки получает информацию о том, что делает `FORdem2`

```
>> help FORdem2
 ФАЙЛ-ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ СУММЫ
    1/1!+1/2!+...+1/10!
```

При написании файл-программ и файл-функций, предназначенных для дальнейшей работы с ними, не пренебрегайте комментариями!

Важно понять, что все переменные, использующиеся в файл-программе, становятся доступными в рабочей среде. Они являются, так называемыми, *глобальными переменными*. Например, для получения значения `k` после выполнения `FORdem2` следует просто набрать `k` в командной строке и нажать <Enter>. Результат очевиден, т. к. последний раз цикл `for` выполнялся как раз для `k`, равного десяти. С другой стороны, в файл-программе могут использоваться все переменные, введенные в рабочей среде.

Поставим задачу вычислить сумму, похожую на предыдущую, но зависящую еще от переменной x:

$$S(x) = \sum_{k=1}^{10} \frac{x^k}{k!}.$$

Для вычисления данной суммы в файл-программе `FORdem`, приведенной в листинге 7.2, требуется изменить строку внутри цикла `for` на

```
S = S + x.^k/factorial(k);
```

Перед запуском программы следует определить переменную `x` в командной строке. Вычисление, например $S(1.5)$ производится из командной строки при помощи следующих команд:

```
>> x = 1.5;
>> FORdem2
S =
    3.4817
```

В качестве `x` может быть вектор или матрица, поскольку в файл-программе `FORdem2` при накоплении суммы использованы поэлементные операции.

Перед запуском `FORdem2` нужно обязательно присвоить переменной `x` некоторое значение, а для вычисления суммы, например из пятнадцати слагаемых, придется внести изменения в текст файл-программы. Гораздо лучше написать универсальную файл-функцию, у которой в качестве входных аргументов будут значение x и верхнего предела суммы, а выходным — значение суммы $S(x)$. Использование операторов, в частности циклов, в файл-функциях производится так же, как и в файл-программах. Файл-функция `sumN`, вычисляющая $S(x)$, приведена в листинге 7.3.

Листинг 7.3. Файл-функция sumN для вычисления суммы

```
function s = sumN(x, N)
% ФАЙЛ-ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ СУММЫ
%     x/1!+x^2/2!+...+x^N/N!
% использование:  S = sum(x, N)

% обнуление S для накопления суммы
s = 0;
% накопление суммы в цикле
for m  = 1:1:N
   s = s + x.^m/factorial(m);
end
```

Об использовании функции `sumN` пользователь может узнать, набрав в командной строке `help sumN`. В командное окно выведутся первые три строки с комментариями, отделенные от текста файл-функции пустой строкой. Ясно, что для $x = 1.5$ и $N = 10$ функция `sumN` даст тот же результат, что и `FORdem2`:

```
>> S = sumN(1.5, 10)
S =
    3.4817
```

Обратите внимание, что внутренние переменные файл-функции не являются глобальными (`m` в файл-функции `sumN`). Попытка просмотра значения переменной `m` из командной строки приводит к сообщению о том, что `m` не определена. Если в рабочей среде имеется *глобальная* переменная с тем же именем `m`, определенная из командной строки или в файл-программе, то она никак не связана с *локальной* переменной `m` в файл-функции. Как правило, лучше оформлять собственные алгоритмы в виде файл-функций для того, чтобы переменные, используемые в алгоритме, не портили значения одноименных глобальных переменных рабочей среды. Впрочем, при необходимости, файл-функция может использовать глобальные переменные.

Подробно про объявление глобальных переменных в файл-функциях написано в главе 8.

Циклы `for` могут быть *вложены* друг в друга, при этом переменные вложенных циклов должны быть *разными*. Вложенные циклы удобны для заполнения матриц. Например, во встроенной справке MatLab по оператору `for` содержится пример заполнения матрицы Гильберта при помощи вложенных циклов. Элементы матрицы Гильберта порядка n определяются формулами $a_{i,j} = 1/(i+j-1)$, для $i, j = 1,2,\ldots,n$.

Скопируйте пример в новый M-файл при помощи буфера обмена Windows, сохраните с именем HILdem.m и продолжите работу с ним, как с файл-программой.

Замечание

В данной книге описываются только основные возможности MatLab, изучение которых поможет при самостоятельной работе. Не пренебрегайте возможностью запускать примеры, приведенные во встроенной справке по MatLab, и изменять их по своему усмотрению, добиваясь желаемого результата.

Перед инициализацией матрицы следует задать ее размер. Для вывода матрицы на экран достаточно добавить в конце файл-программы имя массива, содержащего матрицу. Следует снабдить части файл-программы комментариями. После небольших изменений в скопированном примере файл-программа `HILdem` выглядит так, как показано в листинге 7.4.

Листинг 7.4. Файл-программа `HILdem` для нахождения матрицы Гильберта

```
% Задание размера матрицы
n = 4;
% Инициализация матрицы и заполнение ее нулями
a = zeros(n, n)
% Вычисление матрицы Гильберта порядка n
for i = 1:n
    for j = 1:n
        a(i,j) = 1/(i+j -1);
    end
end
% Вывод матрицы Гильберта на экран
a
```

Запуск `HILdem` приводит к выводу в командное окно матрицы Гильберта четвертого порядка и появлению в рабочей среде новой глобальной переменной — массива `a` размера четыре на четыре, содержащего элементы матрицы Гильберта.

Замечание

Перед заполнением матриц или векторов следует сначала создать их и заполнить нулями командой `zeros` для увеличения скорости алгоритма. Команда `a=zeros(n,n)` выполняется быстрее, чем последовательность `a(i,j)=0` для всех `i` и `j` от единицы до `n`.

Поставим теперь задачу — исследовать границы спектра матрицы Гильберта для различных порядков матрицы (от первого до некоторого *N*-го) и отобразить результат в виде графиков значений максимального и минимального собственных чисел в зависимости от размера матрицы. Для решения этой задачи требуется *N* раз выполнить операторы файл-программы `HILdem`, приведенной в листинге 7.4, изменяя `n` от единицы до *N*. После вычисления матрицы Гильберта порядка *n* необходимо:

1. Найти ее спектр при помощи встроенной функции `eig`.
2. Определить границы спектра, т. е. максимальный и минимальный элемент вектора — аргумента `eig`.
3. Запомнить границы в элементах с индексом `n` некоторых вспомогательных векторов.

В конце следует визуализировать результат.

Операциям с матрицами, в частности нахождению спектра, посвящен разд. "Задачи линейной алгебры" главы 6.

Напишите файл-функцию `HilSpect`, которая выводит в графическое окно на разные графики зависимости максимального и минимального собственных чисел матрицы Гильберта от порядка матрицы (для минимального собственного числа используйте логарифмическую шкалу по оси ординат). Входным аргументом файл-функции `HilSpect` должен быть максимальный порядок исследуемых матриц, выходные аргументы в данном случае не нужны. При возникновении затруднений воспользуйтесь листингом 7.5.

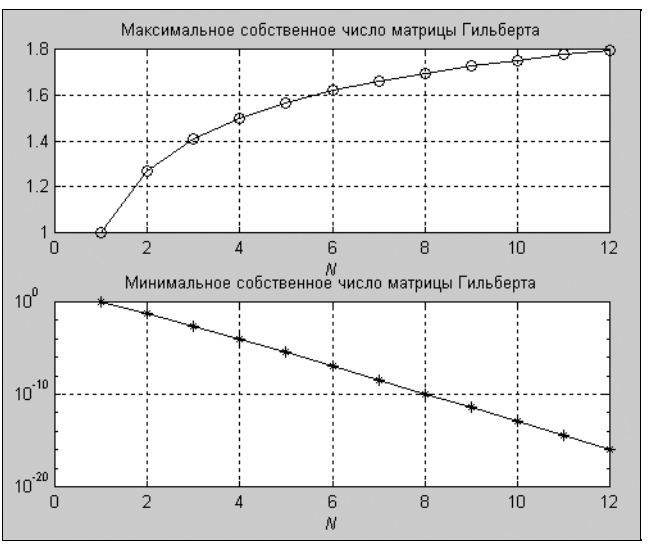


Рис. 7.2. Границы спектра матрицы Гильберта

Исследуйте границы спектра матрицы Гильберта до двенадцатого порядка. Графическое окно, появляющееся в результате вызова `HilSpect(12)`, должно выглядеть так, как показано на рис. 7.2.

Замечание

При решении практических задач, требующих много вычислений, становится актуальной проблема *временных затрат* компьютера. Оказывается, что заполнение матриц лучше осуществлять при помощи индексации, или встроенных функций MatLab, если структура матрицы позволяет это. Например, для получения матрицы Гильберта предназначена функция `hilb`. Увеличению производительности программ пользователя посвящена *глава 21*.

Листинг 7.5. Файл-функция `HilSpect` для исследования спектра матрицы Гильберта

```
function HilSpect(N)
% Исследование границ спектра матрицы Гильберта
% использование: HilSpect(N), N — максимальный порядок

% инициализация массивов для границ спектра
Lmax = zeros(1, N);
Lmin = zeros(1, N);
% вычисления для матриц от первого порядка до N-го
for n =1:N
   % заполнение матрицы Гильберта, вместо вложенных циклов
   % можно использовать встроенную функцию A = hilb(n);
   A = zeros(n);
   for k = 1:n
      for j = 1:n
         A(k,j) = 1/(k+j-1);
      end
   end
   % вычисление спектра и его границ
   Lambda = eig(A);
   Lmax(n) = max(Lambda);
   Lmin(n) = min(Lambda);
end
% вывод зависимостей границ спектра от порядка матрицы
figure;
subplot(2, 1, 1)
```

```
plot(Lmax, 'ko-')
title('Максимальное собственное число матрицы Гильберта')
xlabel('\itN')
grid on
subplot(2, 1, 2)
semilogy(Lmin, 'k*-')
title('Минимальное собственное число матрицы Гильберта')
xlabel('\itN')
grid on
```

Цикл `for` оказывается полезным при выполнении повторяющихся похожих действий в том случае, когда их число заранее определено. Обойти это ограничение позволяет более гибкий цикл `while`, применение которого разобрано в следующем разделе.

Цикл *while*, суммирование рядов

Рассмотрим пример на вычисление суммы, похожий на пример из предыдущего раздела. Требуется найти сумму ряда для заданного x (разложение в ряд $\sin x$):

$$S(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Конечно, до бесконечности суммировать не удастся, но можно накапливать сумму, пока слагаемые являются не слишком маленькими, скажем больше 10^{-10} (по модулю). Циклом `for` здесь не обойтись, т. к. заранее неизвестно значение k, обеспечивающее малость текущего слагаемого. Выход состоит в применении цикла `while`, который работает, пока выполняется *условие цикла*:

```
while условие цикла
     команды MatLab
end
```

В данном примере условием цикла является то, что текущее слагаемое $x^k / k!$ больше 10^{-10}. Для записи условия в форме, понятной MatLab, следует использовать знак больше (>). Текст файл-функции `mysin`, вычисляющей сумму ряда приведен в листинге 7.6.

Замечание

Конечно, малость слагаемого — понятие относительное, слагаемое может быть, скажем порядка 10^{-10}, но и сама сумма того же порядка. В этом случае нельзя останавливать суммирование. Пока не будем обращать на это внимания — нашей задачей является изучение работы с циклами.

Листинг 7.6. Файл-функция `mysin`, вычисляющая синус разложением в ряд

```
function s = mysin(x)
% Вычисление синуса разложением в ряд
% Использование: y = mysin(x), -pi<x<pi
s = 0;
k = 0;
while   abs( x.^(2*k+1)/factorial(2*k+1) ) > 1.0e-10
   s = s + (-1)^k*x.^(2*k+1)/factorial(2*k+1);
   k = k + 1;
end
```

Обратите внимание, что у цикла `while`, в отличие от `for`, нет переменной цикла, поэтому пришлось до начала цикла `k` присвоить ноль, а внутри цикла увеличивать `k` на единицу.

Сравните теперь результат, построив графики функций `mysin` и `sin` на отрезке $[-\pi,\pi]$ на одних осях, например при помощи `fplot` (команды можно задать из командной строки):

```
>> fplot('mysin', [-pi, pi])
>> hold on
>> fplot('sin', [-pi, pi], 'k.')
```

Получающиеся графики изображены на рис. 7.3, они свидетельствуют о правильной работе файл-функции `mysin`.

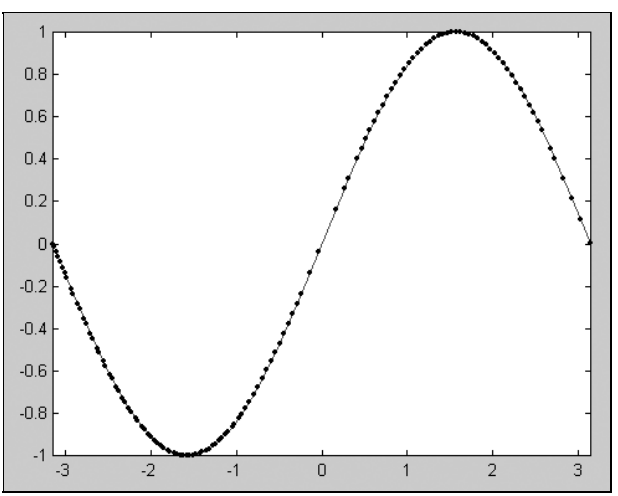


Рис. 7.3. Сравнение `mysin` и `sin`

Условие цикла `while` может содержать не только знак `>`. Для задания условия выполнения цикла допустимы также другие *операции отношения*, приведенные в табл. 7.1.

Таблица 7.1. *Операции отношения*

Обозначение	Операция отношения
==	Равенство
<	Меньше
<=	Меньше или равно
>=	Больше или равно
~=	Не равно

Задание более сложных условий производится с применением *логических операторов*. Например, условие $-1 \le x < 2$ состоит в одновременном выполнении неравенств $x \ge -1$ и $x < 2$, и записывается при помощи логического оператора `and`

```
and(x >= -1 , x < 2)
```

или эквивалентным образом с символом `&`

```
(x >= -1) & (x < 2)
```

Логические операторы и примеры их использования приведены в табл. 7.2.

Таблица 7.2. *Логические операторы*

Оператор	Условие	Запись в MatLab	Эквивалентная запись
Логическое "и"	$x < 3$ и $k = 4$	`and (x < 3, k==4)`	`(x<3) & (k == 4)`
Логическое "или"	$x = 1, 2$	`or (x == 1, x == 2)`	`(x == 1) | (x == 2)`
Отрицание "не"	$a \neq 1.9$	`not (a == 1.9)`	`~(a == 1.9)`

При вычислении суммы бесконечного ряда имеет смысл ограничить число слагаемых. Если ряд расходится из-за того, что его члены не стремятся к нулю, то условие на малость текущего слагаемого может никогда не выполниться и программа зациклится. Выполните суммирование, ограничив число слагаемых. Добавьте в условие цикла `while` файл-функции `mysin` (см. листинг 7.6) ограничение на малость слагаемого:

```
( abs( x.^(2*k+1)/factorial(2*k+1) ) > 1.0e-10 ) & ( k <= 100000)
```

или в эквивалентной форме:

```
and( abs( x.^(2*k+1)/factorial(2*k+1) ) > 1.0e-10 ,  k <= 100000)
```

Замечание

Для задания порядка выполнения логических операций следует использовать круглые скобки, например записи `(x==1)|(x==2)&(y==3)` и `(x==1)||((x==2)&(y==3))` *не эквивалентны* в MatLab, в отличие от многих языков программирования.

Подробнее про логические операторы и логические операции и про возможность применения их к массивам написано в разд. "Логические выражения с массивами и числами" данной главы.

Организация повторяющихся действий в виде циклов делает программу простой и понятной, однако часто требуется выполнить тот или иной блок команд MatLab в зависимости от некоторых условий, т. е. использовать *ветвление* алгоритма.

Операторы ветвления

Условный оператор `if` и оператор переключения `switch` позволяют создать гибкий разветвляющийся алгоритм выполнения команд, в котором при выполнении определенных условий работает соответствующий блок операторов или команд MatLab. Практически во всех языках программирования имеются аналогичные операторы. Использование операторов ветвления описано достаточно подробно, поскольку они понадобятся при написании приложений с графическим интерфейсом пользователя.

Условный оператор *if*

Оператор `if` может применяться в простом виде, для выполнения блока команд при удовлетворении некоторого условия, или в конструкции `if-elseif-else` для написания разветвляющихся алгоритмов.

Проверка входных аргументов

Начнем с простейшего примера — файл-функции для вычисления выражения

$$\sqrt{x^2-1}\,.$$

Создание файл-функции не должно вызвать затруднений. Она работает для любых значений x, причем для $-1<x<1$ результат является комплексным числом. Предположим, что вычисления происходят в области действительных чисел и требуется вывести предупреждение о том, что результат являет-

ся комплексным числом. Перед вычислением функции следует произвести проверку значения аргумента x, и вывести в командное окно предупреждение, если модуль x не превосходит единицы. Здесь уже не обойтись без условного оператора if, применение которого в самом простом случае выглядит так:

```
if условие
     команды MatLab
end
```

Если условие верно, то выполняются команды MatLab, размещенные между if и end, а если условие неверно, то происходит переход к командам, расположенным после end. При записи условия используются операции отношения, приведенные в табл. 7.1 предыдущего раздела.

Файл-функция, проверяющая значение аргумента, приведена в листинге 7.7. Команда warning служит для вывода предупреждения в командное окно.

Листинг 7.7. Файл-функция Rfun, проверяющая значение аргумента

```
function f = Rfun(x);
% вычисляет sqrt(x^2-1)
% выводит предупреждение, если результат комплексный
% использование y = Rfun(x)

% проверка аргумента
if abs(x) < 1
   warning('результат комплексный')
end
% вычисление функции
f = sqrt(x^2-1);
```

Теперь вызов Rfun от аргумента, меньшего единицы по модулю, приведет к выводу в командное окно предупреждения:

```
>> y = Rfun(0.2)
результат комплексный
y =
                0 + 0.979795897113271i
```

Файл-функция Rfun только предупреждает о том, что ее значение комплексное, все вычисления с ней продолжаются. Если же комплексный результат означает ошибку вычислений, то следует прекратить выполнение функции, используя error вместо warning. Напишите файл-функцию root2,

которая по коэффициентам квадратного уравнения находит только вещественные его корни, а для комплексных выдает ошибку. Текст файл-функции `root2` приведен в листинге 7.8.

Листинг 7.8. Файл-функция `root2`, находящая вещественные корни

```
function [x1, x2] = root2(a, b, c)
% возвращает вещественные корни квадратного
% уравнения  ax^2 + bx + c = 0
% если корни комплексные, то выдается сообщение об ошибке
% использование  [x1, x2] = root2(a, b, c)

D = b^2-4*a*c;   % вычисление дискриминанта
% проверка на наличие вещественных корней
if D < 0
   error('комплексные корни')
end
% вычисление корней
x1 = (-b + sqrt(D))/2;
x2 = (-b - sqrt(D))/2;
```

При решении квадратного уравнения, корни которого комплексны, `root2` остановит вычисления и выведет соответствующее предупреждение в командное окно:

```
>> [x1, x2] = root2(1, 0, 1)
??? Error using ==> root2
комплексные корни
```

При составлении файл-функций следует предусмотреть еще один вид контроля — проверку количества входных и выходных параметров. Если пользователь вызовет функцию `root2` с двумя входными параметрами, то получит сообщение об ошибке при выполнении того оператора файл-функции, который содержит неопределенный параметр. В случае вызова функции `root2` с одним выходным аргументом или без аргументов будет вычислен только первый корень квадратного уравнения, что также введет пользователя в заблуждение. Лучше заранее предупредить пользователя о характере ошибки и прекратить работу файл-функции. Кроме того, следует учесть, что файл-функция `root2` не может принимать массивы в качестве входных аргументов. Если даже использовать поэлементные операции при вычислениях, то дискриминант уравнения `D` будет массивом, а что такое `D` > 0 для массива — пока неизвестно.

Логические выражения подробно описаны в разд. "Логические выражения с массивами и числами" данной главы.

Дополните функцию `root2` вышеописанными видами контроля, предотвращающими неправильное ее использование. Встроенные функции `nargin` и `nargout` возвращают число входных и выходных аргументов соответственно. Для проверки, являются ли входные аргументы числами, следует сначала найти размеры соответствующих переменных при помощи `size`, а затем проверить их на равенство единице. Вывод текста в командное окно в ходе выполнения файл-программы или файл-функции осуществляется оператором `disp`, сам текст указывается в апострофах: `disp('текст')`. Листинг 7.9 содержит правильно запрограммированную файл-функцию, при использовании которой не должно возникнуть сложностей.

Листинг 7.9. Файл-функция `root2`, предотвращающая неправильное ее использование

```
function [x1, x2] = root2(a, b, c)
% возвращает вещественные корни квадратного
% уравнения  ax^2 + bx + c = 0
% использование  [x1, x2] = root2(a, b, c)

% проверка числа входных аргументов
if (nargin < 3)
   error('задайте три коэффициента квадратного уравнения')
end

% проверка, являются ли входные аргументы числами
[Na, Ma] = size(a);
[Nb, Mb] = size(b);
[Nc, Mc] = size(c);
% если хотя бы один из размеров входных аргументов не равен единице,
% то выводится сообщение об ошибке и файл-функция прекращает работу
if (Na~=1)|(Ma~=1)|(Nb~=1)|(Mb~=1)|(Nc~=1)|(Mc~=1)
   error('входные аргументы должны быть числами')
end

D = b^2-4*a*c;   % вычисление дискриминанта
% проверка на наличие комплексных корней, если корни комплексные,
% то выводится сообщение об ошибке и файл-функция прекращает работу
```

```
if D < 0
   error('комплексные корни')
end
% вычисление корней
x1 = (-b + sqrt(D))/2;
x2 = (-b - sqrt(D))/2;
% если root2 вызвана с одним выходным аргументом,
% то выводится предупреждение
if nargout ~= 2
   warning('это только один корень')
   disp('для получения двух корней используйте')
   disp('[x1,x2]=root2(a,b,c)')
end
```

Только что вы создали файл-функцию в том же стиле, в котором написаны многие стандартные функции MatLab. Функция `hadamard`, использующаяся для получения матрицы Адамара, является файл-функцией, содержащейся в файле hadamard.m подкаталога toolbox\matlab\elmat основного каталога MatLab. Ее содержание можно посмотреть в любом текстовом редакторе (не стоит только вносить изменения!). При наличии определенного опыта программирования и желания писать собственные вычислительные программы большую пользу может принести самостоятельное изучение стандартных файл-функций, расположенных в подкаталогах toolbox. Большинство из них имеют *открытый код*, что позволяет понять принципы эффективного программирования в MatLab. Другие функции являются *встроенными*. Соответствующие M-файлы содержат только комментарии, в которых указана информация об использовании функции. Например, функции `cos` соответствует M-файл cos.m подкаталога toolboxes\matlab\elfun основного каталога MatLab. Файл cos.m не содержит операторов, последняя строка комментариев `% Build-in function` указывает на то, что `cos` является встроенной функцией.

Организация ветвления

В общем виде оператор ветвления представляет конструкцию `if-elseif-else`, работу которой хорошо поясняет пример файл-функции `ifdem`, приведенной в листинге 7.10.

Листинг 7.10. Файл-функция `ifdem`, демонстрирующая работу `if-elseif-else`

```
function ifdem(a)
% Пример использования структуры if-elseif-else
```

```
if (a == 0)
   disp('a равно нулю')
elseif a == 1
   disp('a равно единице')
elseif a == 2
   disp('a равно двум')
elseif a >= 3
   disp('a больше или равно трем')
else
   disp('a меньше трех, но не ноль, не единица и не двойка')
end
```

В зависимости от выполнения того или иного условия работает соответствующая ветвь программы, если все условия неверны, то выполняются команды, размещенные после `else`. Вызовы функции `ifdem` с различными аргументами позволяют убедиться в вышесказанном:

```
>> ifdem(1)
a равно единице
>> ifdem(1.2)
a меньше трех, но не ноль, не единица и не двойка
>> ifdem(2)
a равно двум
>> ifdem(3)
a больше или равно трем
>> ifdem(-1)
a меньше трех, но не ноль, не единица и не двойка
```

Ветвей может быть сколько угодно (добавьте несколько `elseif` с похожими условиями в `ifdem`), или только две, например:

```
if (a == 0)
   disp('a равно нулю')
else
   disp('a не равно нулю')
end
```

В случае двух ветвей используется завершающее `else`, а `elseif` пропускается. Оператор `if` должен заканчиваться `end`.

Файл-функция `ifdem` хорошо демонстрирует работу оператора `if`, но на практике оказывается бесполезной. Действительно полезный пример — вычисление кусочно-заданной функции

$$y(x)=\begin{cases}\sin x, & x<-\pi;\\ x/\pi, & -\pi\le x<\pi;\\ -\cos x, & x\ge\pi.\end{cases}$$

которое реализуется файл-функцией `pwfun`. Ее текст приведен в листинге 7.11. Обратите внимание на следующие моменты.

- Число ветвей `if-elseif-else` равно трем.
- Во второй ветви достаточно только проверить, что $x<\pi$, условие $x\ge-\pi$ *уже выполнено* (иначе бы отработала первая ветвь в `if-elseif-else`, и оператор `if` закончил работу).
- В последней ветви нет смысла проверять никакие условия, она работает, если все предыдущие условия неверны, что как раз соответствует $x\ge\pi$.

Листинг 7.11. Файл-функция `pwfun`, вычисляет кусочно-заданную функцию

```
function y = pwfun(x)
% вычисляет кусочно-линейную функцию
%         sin(x)-1,    если x<-pi
% y(x)=          x,    если -pi<=x<0
%        pi*cos(x), если x>=0
% использование y = pwfun(x), x — число;
if x < -pi
   y = sin(x) -1;
elseif x < pi % проверка x > -pi не нужна!
   y = x/pi;
else          % здесь x > pi
   y = -cos(x);
end
```

Для построения графика кусочно-заданной функции `pwfun` следует воспользоваться командой `fplot('pwfun', [-3*pi, 3*pi])`, которая приводит к построению графика, изображенного на рис. 7.4.

Построение графика `pwfun` не случайно осуществлялось при помощи `fplot`. Функцией `plot` воспользоваться не удастся, т. к. требуется предварительно вычислить вектор значений функции от вектора аргументов, а `pwfun` не уме-

ет работать *с входным аргументом-вектором*. Убедиться в этом можно, построив график pwfun командами:

```
x = [-3*pi:0.1:3*pi];
y = pwfun(x);
plot(x,y)
```

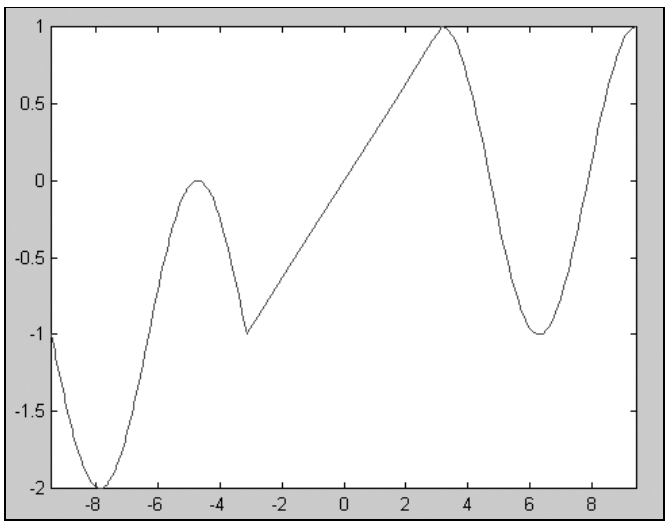


Рис. 7.4. График кусочно-заданной функции

Никакой ошибки при выполнении файл-функции не возникает, однако график строится неправильно. Дело в том, что *вектор x* входит в условия оператора if. Операции отношения (<, <=, >, >=, ~=) могут *специальным образом* применяться и к векторам, о чем подробно написано ниже, а при обычном применении в данном примере не дают ожидаемого результата. Выход состоит в применении следующего алгоритма вычисления кусочно-заданной функции, для реализации которого достаточно понимания вышеописанного материала.

1. Проверка числа входных аргументов, если число входных аргументов не равно единице, то завершение работы файл-функции с сообщением об ошибке (выход по ошибке).
2. Проверка, является ли входной аргумент вектором или числом, один из размеров входного аргумента должен быть равен единице. Если это условие не выполняется, то выход по ошибке.
3. Нахождение длины входного аргумента.
4. Создание вектора выходного аргумента того же размера, что и входной аргумент, и заполнение его нулями.

5. Перебор всех элементов входного вектора с использованием цикла `for`, вычисление от них значений кусочно-заданной функции и запись в соответствующие элементы выходного вектора.

Попытайтесь составить файл-функцию самостоятельно. В случае возникновения затруднений руководствуйтесь листингом 7.12.

Листинг 7.12. Файл-функция `pwfun1` (входной аргумент — число или вектор)

```
function y = pwfun1(x)
% вычисляет кусочно-линейную функцию
%        sin(x)-1,  если x<-pi
% y(x)=  x,         если -pi<=x<0
%        pi*cos(x), если x>=0
% использование y = pwfun(x), x — число или вектор;

% проверяем количество входных аргументов, если аргумент не один,
% то выходим из файл-функции по ошибке
if nargin ~= 1
   error('должен быть один входной аргумент')
end
% проверяем, является ли входной аргумент вектором или числом
[Nx, Mx] = size(x);
% если оба размера входного аргумента не равны единице,
% то выходим по ошибке
if (Nx~=1) & (Mx~=1)
   error('аргументом функции может быть вектор или число')
end

Lx = length(x); % находим длину вектора
% инициализируем выходной аргумент — вектор y
% y должен быть того же размера, что x
y=zeros(size(x));
% перебираем все элементы вектора x в цикле
for i=1:Lx
% вычисляем функцию в зависимости от значения x(i)
   if x(i) < -pi
      y(i) = sin(x(i))-1;
   elseif x(i) < pi % проверка x > -pi не нужна!
      y(i) = x(i)/pi;
```

```
   else          % здесь x > pi
      y(i) = -cos(x(i));
   end
end
```

Входными аргументами файл-функции `pwfun2` могут быть как число, так и вектор, причем если входной аргумент является вектор-строкой (вектор-столбцом), то результат тоже вектор-строка (вектор-столбец).

В качестве завершающего упражнения попытайтесь улучшить `pwfun2` так, чтобы ее входным аргументом могла быть и матрица. Учтите, что вектор в MatLab, так же как и матрица, является двумерным массивом, у которого один из размеров равен единице. Очевидно, что для перебора элементов входного аргумента придется использовать вложенные циклы `for`. При появлении вопросов посмотрите листинг 7.13.

Листинг 7.13. Файл-функция `pwfun2` (входной аргумент — двумерный массив)

```
function y = pwfun2(x)
% вычисляет кусочно-линейную функцию
%        sin(x)-1,  если x<-pi
% y(x)=  x,         если -pi<=x<0
%        pi*cos(x), если x>=0
% использование y = pwfun(x), x — число;

% проверяем количество входных аргументов, если аргумент не один,
% то выходим из файл-функции по ошибке
if nargin ~= 1
   error('должен быть один входной аргумент')
end

% нахождение размеров входного аргумента
[Nx, Mx] = size(x);

% инициализируем выходной аргумент
% двумерный массив y того же размера, что x
y=zeros(size(x));

% перебираем все элементы массива x в двойном цикле
for i=1:Nx
```

```
   for j=1:Mx
      % вычисляем функцию в зависимости от значения x(i,j)
      if x(i,j) < -pi
         y(i,j) = sin(x(i,j))-1;
      elseif x(i,j) < pi % проверка x > -pi не нужна!
         y(i,j) = x(i,j)/pi;
      else          % здесь x > pi
         y(i,j) = -cos(x(i,j));
      end
   end
end
```

Оператор *switch*

Предположим, что при работе с функцией двух аргументов

$$e^{-|x\cdot y|}\sin\pi x\cdot\cos\pi x^2$$

приходится визуализировать ее четырьмя различными способами: каркасной поверхностью, сплошной поверхностью, выводить линии уровня, строить освещенную поверхность. Удобно создать файл-функцию, один из входных аргументов которой `vis` будет определять способ визуализации. Если `vis` равен единице, то строится каркасная поверхность, для `vis`, равного двум, — сплошная и т. д. Можно, конечно, использовать оператор `if` в полном виде `if-elseif-else`, который в зависимости от значения `vis` выполняет нужную ветвь программы, выводящую соответствующий график. Однако оператор переключения `switch` позволяет написать более наглядную программу. Применение `switch` поясняет следующий фрагмент:

```
switch a
  case -1
     disp('a = -1')
  case 0
     disp('a = 0')
  case 1
     disp('a = 1')
  case (2, 3, 4)
     disp('a равно 2 или 3 или 4')
  otherwise
     disp('a не равно -1, 0, 1')
end
```

Каждая ветвь определяется оператором `case`, переход в нее выполняется тогда, когда переменная оператора `swith` (в данном примере `a`) принимает значение, указанное после `case`, или одно из значений из списка `case`. После выполнения какой-либо из ветвей происходит выход из `switch`, при этом значения, заданные в других `case`, уже *не проверяются*. Если подходящих значений для `a` не нашлось, то выполняется ветвь программы, соответствующая `otherwise`.

Оператор `switch` как нельзя лучше подходит для решения поставленной задачи о выводе различных графиков исследуемой функции. Попытайтесь написать файл-функцию самостоятельно. Входными аргументами являются границы построения исследуемой функции по каждой из переменных `xmin`, `xmax`, `ymin`, `ymax` и способ построения графика, определяемый `vis`. Все пять входных аргументов должны быть числами, причем `xmin` меньше `xmax` и `ymin` меньше `ymax` — не забудьте сделать соответствующую проверку! Выходные аргументы в данном случае не требуются. Текст файл-функции `myplot3D` для решения поставленной задачи приведен в листинге 7.14.

Листинг 7.14. Файл-функция `myplot3D`

```
function myplot3D(xmin, xmax, ymin, ymax, vis)
% строит график поверхности функции
%   -|x*y|                        2
% e        * sin(pi*x) * cos(pi*x )
% на области    xmin <= x <= xmax    ymin <= y <= ymax
% использование: myplot3D(xmin, xmax, ymin, ymax, vis)
% vis = 1 — каркасная поверхность
% vis = 2 — залитая поверхность
% vis = 3 — линии уровня
% vis = 4 — освещенная поверхность

% проверяем число входных аргументов,
% если число входных аргументов не равно пяти, то выходим по ошибке
if nargin ~= 5
   error('Задайте xmin, xmax, ymin, ymax, vis')
end

% проверяем число выходных аргументов, если файл-функция вызвана
% с выходными аргументами, то выходим по ошибке
if nargout > 0
   error('Функция myplot3D не имеет выходных аргументов')
end
```

```
% находим максимальный из размеров входных аргументов
M = max([size(xmin) size(xmax) size(ymin) size(ymax) size(vis)]);
% если хотя бы один из размеров входных аргументов не равен единице,
% то выходим по ошибке
if M~=1
   error('входные аргументы должны быть числами')
end
% проверяем границы построения, если нижняя граница больше или равна
% верхней, то выходим по ошибке
if (xmin >= xmax) | (ymin >= ymax)
   error('нижняя граница должна быть меньше верхней')
end
% вычисление шагов по x и y для построения графика поверхности
dx = (xmax-xmin) /40;
dy = (ymax-ymin) /40;
% генерация сетки
[X,Y] = meshgrid([xmin:dx:xmax], [ymin:dy:ymax]);
% вычисление функции
Z = exp(-abs(X.*Y).*sin(pi*X).*cos(pi*X.^2));
% определение способа построения в зависимости от vis
switch vis
case 1  % каркасная поверхность
   figure
   mesh(X, Y, Z)
case 2  % залитая поверхность
   figure
   surf(X, Y, Z)
case 3 % линии уровня функции
   figure
   contour(X, Y, Z)
case 4 % освещенная поверхность
   figure
   surfl(X, Y, Z)
   colormap(copper)
   shading interp
otherwise % непредусмотренная ситуация
   disp('vis может быть 1, 2, 3 или 4')
end
```

Оператор `switch` удобно применять тогда, когда есть соответствие между *дискретными* значениями некоторой переменной и последующими действиями. Для определения ветви программы в зависимости от выполнения более сложных условий, например `a > 0`, приходится использовать оператор `if`.

Прерывания цикла, исключительные ситуации

Хорошо написанная программа предотвращает ошибочные действия, которые приводят к досрочному ее завершению. Проверка входных и выходных аргументов файл-функции является одним из способов контроля. MatLab предоставляет в распоряжение программиста еще два средства: прерывание цикла и обработку исключительных ситуаций.

Прерывание цикла, оператор *break*

При организации циклических вычислений следует позаботиться о том, чтобы внутри цикла не возникало ошибок. Например, пусть дан массив `x`, состоящий из целых чисел, и требуется сформировать новый массив `y` по правилу `y(i)=x(i)/x(i+1)`. Очевидно, что задача может быть решена при помощи цикла `for`. Но если один из элементов исходного массива равен нулю, то при делении получится `Inf` и последующие вычисления могут оказаться бесполезны. Предотвратить подобную ситуацию можно выходом из цикла, если текущее значение `x(i)` равно нулю. Следующий фрагмент программы демонстрирует использование оператора `break` для прерывания цикла:

```
...
y = zeros(length(x) -1)
for i = 1:length(x) -1
   if x(i) == 0
      break
   end
   y(i) = x(i+1)/x(i);
end
...
```

При выполнении условия `x(i) == 0` оператор `break` заканчивает цикл и происходит выполнение операторов, которые расположены в строках, следующих за `end`. Оператор `break` можно использовать и с циклом `while`. В случае вложенных циклов `break` осуществляет выход из внутреннего цикла.

Обработка исключительных ситуаций, оператор *try...catch*

Часть некорректных математических операций в MatLab, в отличие от многих языков программирования, не приводит к завершению работы программы. При делении на ноль получается бесконечность `Inf`, деление ноля на ноль приводит к `NaN`, сумма бесконечности и числа имеет результатом бесконечность. Однако есть ошибки, приводящие к окончанию работы программы. К таким ошибкам относится, например, работа с несуществующими файлами. Предположим, что в процессе выполнения программы следует считать в переменную данные из файла, преобразовать их и отобразить в виде круговой диаграммы, а, затем, продолжить некоторые вычисления, которые не связаны со считанными данными. Последовательность операторов, соответствующая требуемым действиям, приведена в листинге 7.15.

Листинг 7.15. Пример работы с файлом

```
A = load('my.dat');
pie(A)
X = [1 2 -1 -2];
X=X.^2
```

Если MatLab обнаруживает файл my.dat в путях поиска и считывает данные из него, то фрагмент программы из листинга 7.15 работает успешно.

Схема поиска файлов, которую применяет MatLab, описана в разд. "Установка путей" главы 5.

Однако если файл найти не удалось, или при чтении из него возникли ошибки, то MatLab выведет сообщение в командное окно и закончит выполнение программы. Выходом из подобных ситуаций является конструкция `try...catch`, позволяющая обойти исключительные ситуации, которые приводят к ошибке, и предпринять некоторые действия в случае их возникновения. Схема использования `try...catch` выглядит следующим образом:

```
try
   % операторы, выполнение которых
   % может привести к ошибке
catch
   % операторы, которые следует выполнить
   % при возникновении ошибки в блоке
   % между try и catch
end
```

Фрагмент программы, приведенный в листинге 7.15, лучше оформить с использованием `try...catch` так, как это сделано в листинге 7.16.

Листинг 7.16. Обработка ошибки при чтении данных из файла

```
try
   A = load('my.dat');
   pie(A)
catch
   disp(' не могу найти файл my.dat ')
end
X = [1; 2; -1; -2];
X = X.^2
```

Теперь при отсутствии нужного файла my.dat программа выдаст сообщение об этом и продолжит работу:

```
>> не могу найти файл my.dat
X =
     1     4     1     4
```

Логические выражения с массивами и числами

MatLab является средой, ориентированной на вычисления с матричными данными. Универсальным средством обработки матричных данных служат логические операции. Они позволяют просто и наглядно записывать алгоритмы, реализация которых в других языках достаточно громоздка. В предыдущих разделах использовались логические выражения вида `a > 0`, `(a == 1) & (b > 2)`, где `a`, `b` — числа. Поскольку MatLab представляет числа массивами размера один на один, то естественно ожидать, что и массивы могут входить в логические выражения. В данном разделе описано расширение логических операций и операций отношения на случай массивов. Разумеется, все, что касается логических операций и операций отношения для массивов, справедливо и для чисел. Объяснено применение логического индексирования при работе с массивами, которое существенно облегчает обработку данных.

Операции отношения

Результатом операций отношения может быть или "истина", или "ложь". При `x` равном двум, условие `x>=2` оказывается истинным. "Истине" в MatLab соответствует логическая единица, "ложь" обозначается логическим нулем. Арифметические переменные могут использоваться в одном выражении с

логическими, в отличие от многих языков программирования. Например, выражение `a+(b>c)` не является ошибочным, в чем легко убедиться при помощи команд:

```
>> a = 2;
>> b = 3;
>> c =  a < b
c =
    1
```

Условие `a < b` выполняется, т. е. является "истиной". Результат операции отношения "меньше" равен единице. Использование операции "больше или равно" даст в ответе ноль.

Операции отношения применимы к массивам одинакового размера. Происходит *поэлементное* сравнение и результатом является массив того же размера, что и исходные, состоящий из нулей и единиц. Единицы соответствуют тем элементам, для которых условие выполняется, а ноль означает невыполнение условия, например:

```
>> A = [1 2 3 4;  5 6 7 8;  9 10 11 12];
>> B = [3 2 3 3;  5 6 2 2;  4 10 11 11];
>> C = A == B
C =
     0     1     1     0
     1     1     0     0
     0     1     1     0
>> D = A > B
D =
     0     0     0     1
     0     0     1     1
     1     0     0     1
```

Замечание

Так же как и в поэлементных арифметических операциях, допустимо использование числа в качестве одного из аргументов операций отношения. В этом случае происходит сравнение каждого элемента массива с числом, результатом является массив того же размера, что и исходный. Попытка сравнения двух массивов разных размеров приводит к ошибке.

При сравнении двумерных массивов больших размеров удобно использовать команду `spy` для визуализации результата, которая приводит к появлению графического окна с шаблоном матрицы. Элементы результирующей матри-

цы отображаются точками, что наглядно показывает, для каких элементов выполняется проверяемое условие.

Построение шаблона матрицы командой `spy` *описано в разд. "Визуализация матриц" главы 2.*

Логические операции

Конструирование условий операторов `if` и `while` осуществляется с помощью логических операций `and`, `or` и `not`, или в эквивалентном виде с использованием `&`, `|` и `~`. Применение логических операций `and` (`&`), `or` (`|`) и `not` (`~`) к массивам приводит к поэлементному их выполнению над элементами массивов, результатом является массив того же размера, что и исходные, причем:

- `and` дает единицу, если оба соответствующих элемента массива не равны нулю, если хотя бы один из них ноль, то результатом будет также ноль;
- `or` дает единицу, если хотя бы один элемент не равен нулю;
- `not` применяется к одному массиву, если элемент массива не ноль, то соответствующий элемент результирующего массива равен нулю, если элемент исходного массива ноль, то — единице.

Замечание

Одним из элементов логических операций может быть число. В этом случае происходит поэлементное выполнение логической операции для каждого элемента массива и числа. Выполнение операций `and` и `or` над массивами различных размеров недопустимо.

Следующие примеры демонстрируют использование логических операций:

```
>> A = [1 3;  -1 0];
>> B = [0 4;   8 8];
>> C = A&B
C =
     0     1
     1     0
>> D = A|B
D =
     1     1
     1     1
>> E = ~A
E =
     0     0
     0          1
```

Кроме операций `and` и `or` в MatLab определена функция `xor`, выполняющая операцию "исключающее или". Функция `xor` имеет два входных аргумента — массивы одинакового размера. Если один из элементов входного массива не равен нулю, а второй равен, то `xor` записывает единицу на соответствующее место выходного массива. Во всех остальных случаях (ноль и ноль, не ноль и не ноль) `xor` записывает ноль. Аргументами `xor` могут быть массив и число и, конечно, два числа.

Проверка на наличие нулевых элементов в векторе осуществляется функцией `all`, которая возвращает единицу, если среди элементов вектора нет нулей, и ноль в противном случае:

```
>> v = [1 2 0];
>> q = all(v)
q =
     0
```

Аргументом функции `all` может быть и матрица. В этом случае `all` проверяет каждый столбец матрицы на наличие нулей и записывает результат в вектор, каждый элемент которого соответствует столбцу исходной матрицы:

```
>> M = [9 3 -1; 0 2 2];
>> p = all(M)
p =
     0     1     1
```

Исследование на наличие ненулевых элементов производится функцией `any`, которая возвращает единицу, если во входном векторе есть хотя бы один ненулевой элемент. Функция `any`, так же как и `all`, работает с матрицами по столбцам.

Приоритет операций

Поскольку логические и арифметические операции могут входить в одно выражение, то возникает вопрос о том, в какой последовательности они выполняются. Например, в выражении `A + B.^2 > C` сначала выполняется поэлементное возведение в степень, затем сложение и, наконец, сравнение значения суммы `A + B.^2` и `C`.

Приоритет операций отражен в следующем списке (в порядке выполнения):

1. Отрицание.
2. Транспонирование, возведение в степень (в том числе поэлементное), знак плюс или минус перед числом.
3. Умножение и деление (в том числе поэлементное).
4. Сложение и вычитание.

5. Операции отношения.
6. Логическое "и", логическое "или".

Операции, размещенные в каждом пункте списка, имеют одинаковый приоритет. Обратите внимание, что логическое "и" обладает тем же приоритетом, что и логическое "или". Для изменения порядка операций следует применять круглые скобки. Они также полезны для придания выражению легко читаемого вида.

Сначала выполняются операции над аргументами `and`, `or` и `not`, если использовать их вместо `&`, `|` и `~`, например два выражения: `and(A,B)+C` и `A&B+C` *не эквивалентны*!

Логическое индексирование

MatLab предоставляет пользователю удобное средство работы с данными — логическое индексирование. Выделение части массива, элементы которого удовлетворяют определенному условию, производится функцией `find`. В самом простом случае, при вызове функции `find` с одним выходным аргументом — вектором, она возвращает вектор номеров ненулевых элементов, например:

```
>> a = [2 3 0 -1 0 3];
>> u = find(a)
u =
     1     2     4     6
```

Несложно добиться того, чтобы найти номера всех элементов вектора, удовлетворяющих какому-либо условию, скажем меньших единицы и больших либо равных минус единице. Следует сначала применить соответствующие операции логического отношения к исследуемому вектору и получить вектор из нулей и единиц, который затем следует задать входным аргументом `find`:

```
>> b = (a < 1) & (a >= -1)
b =
     0     0     1     1     1     0
>> u = find(b)
u =
     3     4     5
```

Промежуточный вектор `b` можно не использовать, получающееся при этом выражение выглядит более наглядно `u = find((a < 1) & (a >= -1))`.

Индексация вектором позволяет сформировать новый вектор из выбранных элементов исходного:

```
>> anew = a(find((a < 1) & (a >= -1)))
```

```
anew =
     0    -1     0
```

Индексация вектором описана в разд. "Обращение к элементам вектора" главы 2.

Функция `find`, вызванная с одним выходным аргументом вектором от входного аргумента матрицы, возвращает номера ненулевых элементов матрицы в порядке, соответствующем хранению матрицы в MatLab в виде длинного вектора.

Расположение элементов матрицы в памяти описано в разд. "Доступ к элементам матриц" главы 2.

Замена элементов матрицы, удовлетворяющих определенному условию, на другие значения, легко реализуется при помощи `find`. Замените самостоятельно элементы квадратной матрицы случайных чисел из интервала (0, 1), отличающиеся от среднего значения всех элементов на 10%, на среднее значение. Проверьте себя:

```
>> A = rand(5);
>> [m, n] = size(A);
>> s = sum(sum(A))/(m*n);
>> A(find(A>s*1.1)) = s;
```

Определение индексов элементов матрицы, удовлетворяющих заданному условию, осуществляется вызовом функции `find` с двумя аргументами — векторами. В первый вектор записываются номера строк, содержащих требуемые элементы, а во второй — номера столбцов исходной матрицы.

Альтернативным способом выделения элементов из матрицы в вектор является индексация при помощи индексной логической матрицы — логической матрицы того же размера, что и исходная. Индексная матрица должна содержать единицы в позициях, соответствующих выделяемым элементам исходной матрицы. Однако использовать обычную матрицу из нулей и единиц не удастся, появится сообщение об ошибке. Ее следует преобразовать в индексную логическую матрицу при помощи функции `logical` так, как проделано в следующем примере:

```
>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
>> B = [0 1 1; 1 0 0]
B =
     0     1     1
     1     0     0
>> C = A(logical(B))
```

```
C =
     4
     2
     3
```

Некоторые индексные логические матрицы вычисляются по исходной матрице при помощи специальных функций MatLab, перечисленных ниже. Каждая функция для входной матрицы находит индексную логическую матрицу из нулей и единиц, соответствующую проверяемому условию.

- `isfinite` — единицы соответствуют числам, нули — `Inf`, `NaN`.
- `isinf` — единицы соответствуют `+Inf`, `-Inf`, нули — числам.
- `isnan` — единицы соответствуют `NaN`, нули — числам.
- `isprime` — единицы соответствуют простым числам, нули — остальным.
- `isreal` — единицы соответствуют вещественным числам, нули — комплексным.

Глава 8



Тонкости программирования

Простейшие способы организации последовательности выполнения команд и операторов MatLab изложены в предыдущей главе. Данная глава посвящена описанию возможностей MatLab, предназначенных для программирования более сложных алгоритмов. Разобрано создание файл-функций с переменным числом входных и выходных аргументов, использование массивов-структур и массивов-ячеек для хранения и обработки данных, операции со строками, использование файлов и некоторые другие вопросы, тесно связанные с эффективным программированием собственных алгоритмов.

Работа со строками

Операции со строками являются важным элементом программирования в MatLab. Наряду с обычными действиями, допустимыми со строками в большинстве языков программирования высокого уровня, MatLab предоставляет программисту возможность сформировать команду MatLab в виде строковой переменной, а затем выполнить ее.

Простейшие операции со строками

Ввод и сцепление строк

Строки в MatLab хранятся в массиве символов. Присвойте строковой переменной `str` значение 'Hello, word!' (набирать следует в апострофах). Не завершайте оператор присваивания точкой с запятой, для того, чтобы сразу увидеть значение `str`.

```
>> str = 'Hello, World!'
str =
Hello, World!
```

Посмотрите содержимое рабочей среды с помощью команды `whos`:

```
>> whos
  Name      Size          Bytes  Class
  str       1x13             26  char array
Grand total is 13 elements using 26 bytes
```

Переменная `str` является одномерным массивом символов (`char array`) из тринадцати элементов, занимающим в памяти двадцать шесть байтов, каждый символ кодируется двумя байтами. Поскольку строки хранятся в одномерных массивах, то к ним применимы некоторые операции с вектор-строками, в частности, сцепление.

Работа с векторами описана в разд. "Вектор-столбцы и вектор-строки" главы 2.

Индексация позволяет получить доступ к символам строковой переменной, причем возможно как задание числового индекса, так и выделение в отдельную строку нескольких идущих подряд символов при помощи двоеточия. В качестве упражнения переставьте слова в переменной `str`, т. е. получите строку 'World, Hello!'. Если вы изучили работу с массивами, то решение не представит большого труда.

```
>> strnew = [str(8:12) str(6:7) str(1:5) str(13)]
strnew =
World, Hello!
```

Составление одной строки из нескольких может быть произведено и при помощи функции `strcat`, аргументами которой являются строки, подлежащие сцеплению. Обратите внимание, что функция `strcat` игнорирует пробелы в конце строк и результат получается не такой, как при использовании векторного сцепления в предыдущем примере:

```
>> str1 = str(8:12);
>> str2 = str(6:7);
>> str3 = str(1:5);
>> str4 = str(13);
>> newstr = strcat(str1, str2, str3, str4)
newstr =
World,Hello!
```

Сервисные функции для работы со строками

MatLab имеет ряд сервисных функций для облегчения работы со строками. Поиск подстроки в строке осуществляет `findstr`, которая возвращает массив с начальными позициями вхождений:

```
>> str = 'MatLab хранит строки в виде вектор-строк';
>> substr = 'строк';
>> pos = findstr(str, substr)
pos =
    15    36
```

Функция `findstr` полагает, что подстрока с образцом для поиска содержится во входном аргументе меньшего размера, а строка, в которой производит-

ся поиск, — в аргументе большего размера, поэтому необязательно заботиться о порядке входных аргументов:

```
pos = findstr(substr, str)
pos =
    15    36
```

Замечание

Символы русского текста могут неправильно отображаться в некоторых приложениях в Windows, в частности в нелокализованной версии MatLab. Разрешение проблемы состоит в следовании рекомендациям, которые дают эффект для других программ, в частности Adobe Photoshop. Данные советы, связанные с изменением кодовой страницы, легко найти в соответствующих пособиях или в Интернете при помощи любого поискового сервера, набрав в строке запроса, например "русские символы Photoshop".

Сравнение двух строк выполняется функцией `strcmp`, которая возвращает логическую единицу для равных строк и ноль в противном случае:

```
>> str1 = 'abc123def098gh';
>> str2 = 'abc123def098gh';
>> rez = strcmp(str1, str2)
rez =
     1
```

Функция `strncmp` позволяет установить совпадение первых нескольких символов в двух строках, она используется так же, как `strcmp`, только в качестве третьего параметра следует указать число сравниваемых символов от начала строк:

```
>> str1 = 'Hello, World!';
>> str2 = 'Hello, Igor!';
>> rez = strncmp(str1, str2, 5)
rez =
     1
```

Функция `strrep` заменяет все встречающиеся подстроки на другие, первый ее входной аргумент является строкой, в которой следует произвести изменения, второй — подстрокой, подлежащей замене, а третий — подстрокой с образцом замены. Замените, например, '5.3' на '6.0' в предложении 'Matlab 5.3 обладает большими возможностями'. Используйте `strrep` так, как указано ниже:

```
>> str = 'Matlab 5.3 обладает большими возможностями';
>> newstr = strrep(str, '5.3', '6.0')
newstr =
Matlab 6.0 обладает большими возможностями
```

Преобразование всех прописных букв в строчные производит функция `upper`. Функция `lower` осуществляет обратное преобразование. Строка, требующая преобразования, задается входным аргументом данных функций. Функция `ischar` проверяет, является ли входной аргумент строкой или нет, возвращая логическую единицу или ноль соответственно.

Другие функции описаны в следующих разделах, посвященных написанию простых программ с интерфейсом пользователя из командной строки рабочей среды MatLab. Полный список сервисных функций обработки строк приведен в приложении.

Перечисленные выше функции значительно облегчают труд программиста, избавляя его от необходимости самостоятельного программирования алгоритмов поиска и замены. Заметьте, что изложенных в предыдущей главе сведений вполне достаточно для написания собственных файл-функций, выполняющих аналогичные действия. Действительно, поскольку строки хранятся в одномерном массиве символов, то перебором всех элементов строки в цикле можно осуществить поиск и замену одних подстрок на другие. В качестве упражнения напишите файл-функцию `strnumpos`, которая находит позиции всех цифр, входящих в строку, и возвращает результат в виде вектора. Примените цикл `for` для перебора всех символов строки и `if` для проверки на цифру. Воспользуйтесь листингом 8.1 в случае возникновения затруднений.

Действительно незаменимой и очень полезной функцией является `eval`, которая позволяет *выполнить команду MatLab, записанную в строке*. В *разд. "Приложения с интерфейсом из командной строки" данной главы* разобрано формирование и выполнение команд при помощи `eval`, которые реализуют действия, заданные пользователем.

Листинг 8.1. Файл-функция `strnumpos` для поиска цифр в строке

```
function pos = strnumpos(str)
% Функция strnumpos возвращает позиции цифр в строке
% использование:  pos = strnumpos(str)
%                 strnumpos(str)

% проверка входных и выходных аргументов
% допускается не более одного выходного аргумента
if nargout > 1
   error('Функция имеет не более одного выходного аргумента')
end
% должен быть один входной аргумент
if nargin ~= 1
```

```
   error('Должен быть один входной аргумент')
end
% входной аргумент должен быть строкой
if ~ischar(str)
   error('Входной аргумент должен быть строкой')
end

% вычисляем длину строки
slen = length(str);
% обнуляем счетчик числа цифр в строке
digits = 0;
% перебираем в цикле все символы строки
for k = 1:slen
   % проверяем, является ли текущий символ str(k) цифрой
   if (str(k) == '0') | (str(k) == '1') | (str(k) == '2') |...
      (str(k) == '3') | (str(k) == '4') | (str(k) == '5') |...
      (str(k) == '6') | (str(k) == '7') | (str(k) == '8') |...
      (str(k) == '9')
      % текущий символ str(k) — цифра, поэтому
      % увеличиваем счетчик числа цифр на единицу
      digits = digits + 1;
      % добавляем позицию цифры в массив pos
      pos(digits) = k;
   end
end
```

Массивы строк

Удобной возможностью организации строковых переменных являются массивы строк, которые формируются так же, как обычные вектор-столбцы, но в качестве их элементов выступают строки *одинаковой длины*. Создайте, например, массив `names` из строк 'Иван', 'Олег', 'Петр':

```
>> names = ['Иван'; 'Олег'; 'Петр']
names =
Иван
Олег
Петр
```

Доступ к строкам в массиве осуществляется при помощи индексации:

```
>> names(1,:)
```

```
ans =
Иван
```

Обратите внимание, что использование строк разной длины недопустимо:

```
>> surnames = ['Иванов'; 'Васильев'; 'Петров']
???  All rows in the bracketed expression must have the same
number of columns.
```

Очевидно, что `names` является двумерным массивом (матрицей), состоящим из символов, поэтому возникает ограничение на одинаковую длину составляющих строк. Короткие строки следует дополнить пробелами до максимальной из длин строк, входящих в массив. Функция `char` позволяет просто решить эту задачу:

```
>> surnames = char('Иванов', 'Васильев', 'Петров')
surnames =
Иванов
Васильев
Петров
```

Первая и последняя строки дополнились справа двумя пробелами, и длина всех строк стала одинаковой. Для удаления лишних пробелов при доступе к строкам массива предназначена функция `deblank`:

```
>> surn1 = deblank(surnames(1, :))
surn1 =
Иванов
```

Аргументами `char` могут быть и массивы строк, что позволяет добавлять строки в начало или конец существующих массивов строк, например:

```
>> surnames = char(surnames, 'Сидоров')
surnames =
Иванов
Васильев
Петров
Сидоров
```

Поиск в массиве строк производится функцией `strmatch`, входными аргументами которой являются образец подстроки для поиска и массив строк, а выходным — номера строк массива:

```
>> mas = char('Март', 'Апрель', 'Май');
>> ind = strmatch('Ма', mas)
ind =
     1
     3
```

Задание дополнительного третьего аргумента `'exact'` означает поиск подстроки целиком, вхождение в различные контексты не учитывается.

Наличие большого числа строк приводит к необходимости хранения их в текстовых файлах. Перед началом обработки следует считать информацию из файла, затем обработать ее требуемым образом и, наконец, записать полученный результат в текстовый файл.

Текстовые файлы

Работа с файлами состоит из трех этапов: открытие файла, считывание или запись информации, закрытие файла. Следующие разделы посвящены описанию команд MatLab, реализующих вышеперечисленные действия, и демонстрации их использования на некоторых простых примерах.

Открытие файла, считывание данных и закрытие файла

Команда `fopen` предназначена для открытия существующего или создания нового файла. Имя файла указывается в апострофах первым входным аргументом. Второй аргумент задает способ доступа к файлу, он может принимать следующие значения (символ `t` указывает на то, что файл текстовый):

- `'rt'` — открываемый текстовый файл предназначен только для чтения;
- `'rt+'` — открываемый текстовый файл предназначен для чтения и записи;
- `'wt'` — создаваемый пустой текстовый файл предназначен только для записи;
- `'wt+'` — создаваемый пустой текстовый файл предназначен для записи и чтения.

Выходными аргументами `fopen` являются *идентификатор*, присвоенный файлу, и строковая переменная с сообщением о результате открытия. Если файл открыть не удалось, то идентификатор становится равным минус единице. Ошибки часто возникают из-за того, что MatLab не может найти требуемый файл. Всегда лучше указывать полное имя файла, при задании только имени и расширения MatLab производит поиск в текущем каталоге.

Считывание строк из открытого текстового файла производится командой `fgetl`, входным аргументом которой является идентификатор файла, присвоенный ему при открытии, а выходным — строковая переменная. Каждое обращение к `fgetl` позволяет последовательно считывать строки от начала до конца файла. Контроль за достижением конца файла осуществляется функцией `feof` с входным аргументом — идентификатором файла, `feof` возвра-

щает единицу, если в файле нет больше строк, и ноль — в противном случае. По окончании работы необходимо закрыть файл командой `fclose`, указав в качестве входного аргумента идентификатор файла.

Файл-функция `myview` (см. листинг 8.2) демонстрирует открытие текстового файла, занесение содержимого в массив строк и вывод их на экран. Вызов файл-функции с входным аргументом — именем любого существующего файла, заключенным в апострофы, приводит к отображению содержимого файла в командном окне. Если M-файл с файл-функцией `myview` хранится в текущем каталоге MatLab, то команда `myview('myview.m')` выводит листинг самой файл-функции.

Листинг 8.2. Файл-функция `myview` для просмотра содержимого файла

```
function myview(filename)
% Функция выводит содержимое текстового файла на экран
% использование myview('имя файла')

% проверка аргументов
if nargout ~= 0
   error('Функция не имеет выходных аргументов');
end
if nargin ~= 1
   error('Функция вызывается с одним входным аргументом');
end
if ~ischar(filename)
   error('Входной аргумент функции является строкой');
end

% Открытие текстового файла для считывания (аргумент 'rt'),
% имя файла хранится в filename,
% идентификатор файла записывается в F,
% строка с информацией о возможных ошибках в mes
[F, mes] = fopen(filename, 'rt');

% Если файл успешно открылся, то идентификатор не равен минус единице
if F ~= -1
   MAS = ''; % сначала массив состоит только из пустой строки
   % Последовательное считывание из файла строки до тех пор,
   % пока не достигнут конец файла
   while feof(F) == 0
```

```
      % считывание строки
      line = fgetl(F);
      % добавление считанной строки в массив строк
      MAS = char(MAS, line);
   end
   % закрытие файла
   fclose(F);
   % вывод массива строк в командное окно
   disp(MAS)
else
   % В эту ветвь программа заходит, если при открытии файла
   % возникли ошибки; происходит информирование об ошибке
   % и вывод в командное окно сообщения, выданного fopen
   disp('ОШИБКА при открытии файла')
   disp(mes)
end
```

В качестве упражнения напишите файл-функцию, которая ищет заданную подстроку в текстовом файле и возвращает вектор с номерами строк файла, содержащих подстроку. Файл-функция должна иметь два входных аргумента — имя файла и подстроку и один выходной — массив с номерами строк. Используйте функцию `findstr`, описанную выше, для поиска подстроки в строке. Попытайтесь обойтись без считывания содержимого текстового файла в массив строк для экономии памяти (текстовый файл может быть достаточно большим). Основной блок алгоритма приведен в листинге 8.3, дополните его проверкой входных и выходных аргументов.

Листинг 8.3. Основной блок алгоритма поиска номеров строк, содержащих подстроку

```
function rows = mysearch(filename, substring)
% Файл-функция mysearch ищет подстроку в текстовом файле
% и возвращает номера строк, содержащих данную подстроку
% Использование rows = mysearch(filename, substring)

% Здесь добавьте проверку параметров

% Открытие файла
[F, mes] = fopen(filename, 'rt+');
% Проверка, успешно ли открыт файл
if F ~= -1
```

```
   count = 0; % обнуление счетчика строк
   find = 0;  % обнуление счетчика строк, содержащих подстроку
   % Последовательная обработка строк
   while feof(F) == 0
      line = fgetl(F); % считывание текущей строки
      count = count + 1; % увеличение счетчика строк
      % Сравнение длин строки файла и подстроки,
      % поиск имеет смысл в строках, которые длиннее подстроки
      if length(line) >= length(substring)
         % определение позиций вхождения подстроки
         pos = findstr(substring, line);
         % проверка, что массив вхождений pos не пуст
         if length(pos) > 0
            find = find + 1; % увеличение счетчика найденных строк
            % Занесение номера найденной строки в выходной массив
            rows(find) = count;
         end
      end
   end
   fclose(F); % закрытие файла
else
   % обработка ошибки при открытии файла
   disp('ОШИБКА при открытии файла');
   disp(mes)
end
```

Запись в текстовый файл

Символы текстового файла образуют строки со словами, предложениями или числами. Запись текстовых строк достаточно проста, а вот для занесения в текстовый файл чисел приходится прибегать к специальным *форматам*. Вывод информации в текстовый файл производится при помощи функции `fprintf`. В следующих двух разделах демонстрируется использование `fprintf` на примере создания файла, содержащего таблицу значений функции с заголовком и шапкой.

Запись строк

Добавление строки в текстовый файл осуществляется при помощи `fprint`, вызванной с двумя входными аргументами, — идентификатором файла и строкой с текстом, например команда

```
fprintf(F, 'Строка добавлена командой fprintf. ')
```

записывает соответствующую строку в файл с идентификатором F, присвоенным ему при открытии. Последующая команда fprintf выводит заданную строку *сразу за предыдущей*, а не на новой строке:

```
fprintf(F, 'Еще строка.')
```

Для вывода текста с новой строки следует добавить символ перевода строки \n в начало новой строки после апострофа:

```
fprintf(F, '\n Этот текст с новой строки.')
```

В результате выполнения трех вышеперечисленных команд содержимое текстового файла станет следующим:

```
Строка добавлена командой fprintf. Еще строка.
Этот текст с новой строки.
```

Символ перевода строки \n можно разместить в конце строки, после которой текст должен начинаться с новой строки, например последовательность команд

```
fprintf(F, 'Строка добавлена командой fprintf. ')
fprintf(F, 'Еще строка.\n')
fprintf(F, 'Этот текст с новой строки.')
```

приводит к аналогичному результату.

Конечно, вторым аргументом fprintf может быть не только строка, заключенная в апострофы, но и строковая переменная:

```
str = 'Этот текст добавляется в файл.'
fprintf(F, str)
```

Подумайте, как в данном случае указать команде fprintf, что следующий вывод должен осуществляться с новой строки. Очевидно, что решение вопроса заключается в сцеплении строк либо при помощи квадратных скобок, либо с использованием strcat:

```
str = 'Этот текст запишется в файл, а следующий — с новой строки';
fprintf(F, [str '\n']); или fprintf(F, strcat(str, '\n'));
```

Создайте файл-программу, текст которой приведен в листинге 8.4, выполните ее и посмотрите содержимое файла example.txt, например при помощи редактора М-файлов, установив в раскрывающемся списке **Files of type:** диалогового окна **Open** фильтр **All Files (*.*)** для отображения списка всех файлов.

Листинг 8.4. Файл-программа, демонстрирующая вывод строк в текстовый файл

```
[F, mes] = fopen('example.txt', 'w');
fprintf(F, 'Эта строка добавлена командой fprintf');
```

```
fprintf(F, 'Еще строка\n')
fprintf(F, 'Новая строка')
fclose(F);
```

Начните работу над программой, выводящей таблицу значений функции `sin`. Напишите файл-функцию `sintable`, записывающую в файл название и шапку таблицы, приведенные ниже.

```
ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ sin(x)
------------------
|  x    |    y     |
------------------
```

Поставленная задача решается при помощи четырех вызовов `fprintf` (см. листинг 8.5).

Листинг 8.5. Файл-функция `sintable` для записи названия и шапки таблицы в файл

```
function sintable(filename)
% Файл-функция для вывода таблицы sin(x) в файл
% Использование funtable(filename)

% Добавьте проверку входных и выходных параметров

% Открытие нового файла для записи
[F, mes] = fopen('table.dat', 'w');
% Печать в файл заголовка таблицы
fprintf(F, 'ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ sin(x)\n');
% Печать в файл шапки таблицы
fprintf(F, '------------------\n');
fprintf(F, '|  x    |    y     |\n');
fprintf(F, '------------------\n');
% Закрытие файла
fclose(F);
```

Итак, вывод текста в файл не представляет большого труда. Занесение чисел или значений переменных требует привлечения форматного вывода. Основные сведения, касающиеся форматного вывода, изложены в следующем разделе.

Форматный вывод

Задание формата вывода значений переменных в командное окно описано в *главе 1*. Например, форматы `short`, `long`, `short e`, `long e` задаются командой `format` или из меню рабочей среды. Функция `fprintf` допускает гораздо более гибкое управление видом записи чисел в файл. Схема использования `fprintf` при работе с числовыми переменными такова:

```
fprintf(идентификатор, 'формат', список переменных)
```

Здесь первый аргумент, как и в случае вывода строк, является идентификатором файла, второй — строка со специальными символами, которые определяют вид записи значений переменных из списка, заданного третьим аргументом. В списке может быть одна или несколько переменных, в том числе и массивов.

Замечание

Идентификатор файла может быть опущен, в этом случае производится вывод в командное окно.

Разберем применение форматного вывода на простом примере. Требуется записать значения переменных `x = pi/4` и `y = sin(x)` в файл в формате с плавающей точкой, оставляя четыре цифры после десятичной точки для `x` и шесть цифр — для `y`. Создайте файл-программу, приведенную в листинге 8.6, и выполните ее.

Листинг 8.6. Файл-программа, демонстрирующая форматный вывод в файл

```
[F, mes] = fopen('twonum.txt', 'w');
x = pi/4;
y = sin(x);
fprintf(F, '%7.4f%11.8f', x, y);
fclose(F);
```

Содержимое файла twonum.txt составляют два числа — значения переменных `x` и `y` с требуемым числом цифр. Обратите внимание на второй аргумент команды `fprintf`. Последовательность `%7.4f` задает формат вывода переменной `x`, которая расположена на первом месте в списке вывода. Знак процента указывает на начало формата, цифра 7 обозначает, что всего под значение переменной `x` отводится семь позиций, цифра 4 после разделителя-точки обеспечивает точность отображения результата — четыре цифры после десятичной точки. *Спецификатор* `f` указывает на то, что следует вывести число в формате с плавающей точкой. Аналогичным образом работает формат `%11.8f` для вывода `y`. Соответствие форматов и получаемого резуль-

тата приведено на схеме, изображенной на рис. 8.1, каждая позиция подчеркнута.

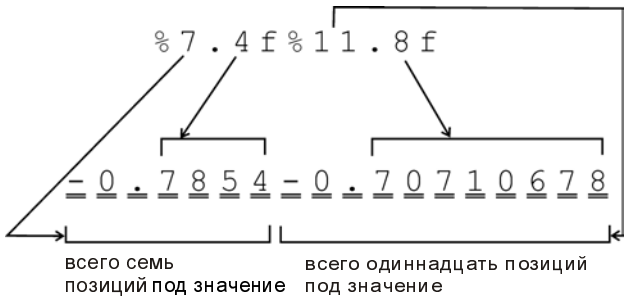


Рис. 8.1. Схема соответствия форматов вывода и результата

При использовании форматного вывода следует учесть, что число форматов, начинающихся со знака процента, должно равняться количеству элементов в списке вывода. Форматы можно разделять пробелами, которые запишутся в текстовый файл между соответствующими значениями. Более того, в строке форматов допустим текст. Внесите изменения в команду `fprintf` файл-программы, приведенной в листинге 8.6. Дополните запись в файл некоторыми пояснениями так, как приведено ниже

```
fprintf(F, 'x = %7.4f  y = %11.8f', x, y);
```

и выполните файл-программу. Содержимое файла twonum.txt выглядит теперь следующим образом:

```
x = -0.7854  y = -0.70710678
```

Завершение строки с форматами символом `\n` приводит к последующему выводу данных с новой строки. Кроме `f`, допустимы и другие спецификаторы форматов, в частности, спецификатор `e` означает вывод в экспоненциальной форме.

Полный список спецификаторов приведен в приложении 1.

После знака процента может размещаться флаг, позволяющий задать некоторые дополнительные параметры отображения чисел. Флаг может принимать следующие значения:

- знак плюс, для отображения знака положительных чисел;
- знак минус, означающий выравнивание числа по левому краю в отведенном для него поле (по умолчанию число выравнивается по правому краю);
- цифра ноль, предназначенная для заполнения оставшихся позиций слева от числа нулями.

Команды, приведенные ниже, демонстрируют использование флага:

```
a = 0.56;
b = 1.1;
c = 1.22
fprintf(F, 'a = %+7.3f  b = %-11.1f   c = %07.2f\n', a, b, c);
```

Дополните ими файл-программу, приведенную в листинге 8.6, и посмотрите результат в twonum.txt:

```
a =  +0.560  b = 1.1           c = 0001.22
```

Полезной особенностью `fprintf` является то, что список ввода может быть матрицей. В этом случае форматы применяются *по столбцам к каждому элементу столбца матрицы*. Команды

```
[F, mes] = fopen('randmatr.txt', 'w');
R = rand(3);
disp(R)
fprintf(F, '| %7.4f  | %7.4f  | %7.4f |\n', R');
fclose(F);
```

выводят квадратную матрицу R размерностью три из случайных чисел в командное окно и файл, столбцы в файле разделены вертикальными линиями. Обратите внимание, что аргументом команды `fprintf` является транспонированная матрица R', т. к. `fprint` работает с матрицей по столбцам. Матрица, отображенная в командном окне, совпадает с матрицей, записанной в файл.

Информации о форматах, приведенной выше, вполне достаточно для завершения работы над файл-функцией `sintable`, предназначенной для вывода таблицы значения функции `sin` в файл. В случае возникновения затруднений обратитесь к листингу 8.7, в котором приведена часть файл-функции, отвечающая за вывод таблицы (разумеется, данный блок должен предшествовать закрытию файла командой `fclose`). Операторы, осуществляющие запись в файл названия и шапки таблицы, были приведены в листинге 8.5.

Листинг 8.7. Блок операторов для вывода таблицы значений функции

```
% Создание вектора значений аргумента
x = [0:pi/2:2*pi];
% Конструирование матрицы, первая строка которой содержит
% значение аргумента, а вторая — значения функции sin
M = [x; sin(x)];
% форматный вывод элементов матрицы
fprintf(F, '|%7.3f|%10.4f|\n', M);
```

В результате работы файл-функции `sintable` создается файл, содержимое которого приведено ниже:

```
ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ sin(x)
------------------
|  x    |    y     |
------------------
|  0.000|    0.0000|
|  1.571|    1.0000|
|  3.142|    0.0000|
|  4.712|   -1.0000|
|  6.283|   -0.0000|
```

Массивы структур и массивы ячеек

Обычные массивы удобны при работе с однородными данными — числами или строками. Эффективное оперирование различными типами данных позволяют осуществить *массивы ячеек*. Информация может быть представлена в виде таблицы с полями, содержащими однотипные элементы, в этом случае наилучшим выбором является использование *массивов структур*.

Массивы структур

Предположим, что успеваемость группы студентов по шести предметам представлена в табл. 8.1.

Таблица 8.1. *Успеваемость студентов группы 201*

N	Фамилия	Имя	Год рождения	Оценки по предметам					
				I	II	III	IV	V	VI
1	Алексеев	Иван	1980	5	4	4	5	5	4
2	Васильев	Сергей	1981	3	4	4	3	5	4
3	Кашин	Павел	1979	4	3	4	4	5	4
4	Серова	Наталья	1981	4	3	3	5	4	5
5	Терехова	Ольга	1980	5	5	5	5	4	5

Для хранения информации удобно использовать массив структур, каждый элемент которого является *структурой* с одинаковым набором *полей*, содер-

жащих соответствующее значение. Информация о каждом студенте заключена в структуре со следующими полями:

- фамилия (`Family`), содержит строку с фамилией;
- имя (`Name`), содержит строку с именем;
- год рождения (`Year`), содержит число;
- оценки (`Marks`), содержит массив из шести элементов с оценками.

Схема, демонстрирующая организацию данных табл. 8.1 в виде массива структур, приведена на рис. 8.2.

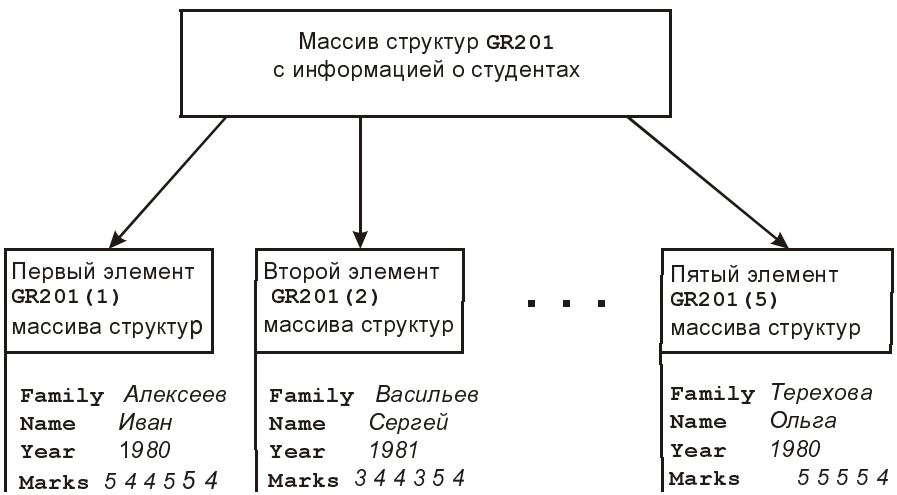


Рис. 8.2. Организация таблицы в виде массива структур

При использовании массивов структур необходимо придерживаться двух правил:

- доступ к структурам, входящим в массив, осуществляется при помощи индексации;
- поле отделяется от имени структуры при помощи точки.

Допускаются два способа заполнения массива структур — операторами присваивания для всех полей каждой структуры массива, или функцией `struct`, позволяющей занести значения сразу во все поля структуры. Операторы присваивания, заполняющие массив структур данными из табл. 8.1, приведены в листинге 8.8, применение `struct` описано ниже.

Листинг 8.8. Заполнение массива структур при помощи операторов присваивания

```
% Заполнение первой структуры массива
GR201(1).Family = 'Алексеев';
GR201(1).Name = 'Иван';
GR201(1).Year = 1980;
GR201(1).Marks = [5 4 4 5 5 4];
% Заполнение второй структуры массива
GR201(2).Family = 'Васильев';
GR201(2).Name = 'Сергей';
GR201(2).Year = 1981;
GR201(2).Marks = [3 4 4 3 5 4];
% Заполнение третьей структуры массива
GR201(3).Family = 'Кашин';
GR201(3).Name = 'Павел';
GR201(3).Year = 1979;
GR201(3).Marks = [4 3 4 4 5 4];
% Заполнение четвертой структуры массива
GR201(4).Family = 'Серова';
GR201(4).Name = 'Наталья';
GR201(4).Year = 1981;
GR201(4).Marks = [4 3 3 5 4 5];
% Заполнение пятой структуры массива
GR201(5).Family = 'Терехова';
GR201(5).Name = 'Ольга';
GR201(5).Year = 1980;
GR201(5).Marks = [5 5 5 5 4 5];
```

Заполните массив структур `GR201`, создав файл-программу в соответствии с листингом 8.8. Посмотрите значение переменной `GR201` из командной строки:

```
>> GR201
GR201 =
1x5 struct array with fields:
    Family
    Name
    Year
    Marks
```

В отличие от переменных, содержащих числа, массивы или строки, задание имени массива структур не приводит к отображению в командном окне значений полей. Выводится только информация о размере структуры и названиях полей. Просмотр содержимого полей какой-либо структуры требует задания ее индекса:

```
>> GR201(3)
ans =
    Family: 'Кашин'
      Name: 'Павел'
      Year: 1979
     Marks: [4 3 4 4 5 4]
```

Указание поля структуры позволяет получить доступ к его значению:

```
>> GR201(2).Year
ans =
        1981
```

Функция `size`, примененная к массиву структур, возвращает размеры массива, в данном случае один на пять:

```
>> size(GR201)
ans =
     1     5
```

Замечание

Массивы структур могут быть и двумерными, тогда обращение к какой-либо структуре осуществляется при помощи двух индексов.

Дополните файл-программу, приведенную в листинге 8.8, операторами, выводящими значения полей структур. Используйте цикл `for` для перебора всех элементов массива `GR201` (см. листинг 8.9).

Листинг 8.9. Вывод значений полей структуры в командное окно

```
Len = max(size(GR201));
for k = 1:Len
   disp(GR201(k))
end
```

Альтернативным вариантом занесения информации в поля структуры является применение функции `struct`, выходным аргументом которой является

структура, подлежащая заполнению, а входными — пары 'название поля'-значение, указываемые через запятую так, как приведено ниже.

```
GR201(1) = struct('Family', 'Алексеев', 'Name', 'Иван',...
   'Year', 1980, 'Marks', [5 4 4 5 5 4]);
```

Дополнительные функции для работы с полями структур приведены в приложении 1.

Создание файл-функций для работы массивами структур

Обработка данных, содержащихся в массивах структур, требует написания собственных файл-функций. Массив структур передается в качестве аргумента файл-функции, доступ к содержимому полей осуществляется при помощи названий полей в структуре так, как описано в предыдущем разделе. Операции, применяемые к значениям полей, должны соответствовать содержащимся в них данным. Создайте файл-функцию groupprogr, отображающую среднюю успеваемость группы студентов по каждому курсу (см. табл. 8.1). Результат представьте в виде столбцевой диаграммы, число столбцов которой равно числу курсов. Алгоритм решения достаточно прост. Следует определить число студентов, используя size для нахождения размера массива структуры, и количество курсов, т. е. длину вектора, хранящегося в поле Marks какой-либо структуры. Далее перебором по всем курсам и студентам при помощи двух вложенных циклов for найдите среднее арифметическое из оценок студентов за каждый курс. Запишите результат в вектор-строку и получите столбцевую диаграмму, используя функцию bar. Текст файл-функции groupprogr приведен в листинге 8.10.

Листинг 8.10. Файл-функция groupprogr для определения успеваемости группы

```
function meanmarks = groupprog(GROUP);
% Функция вычисляет средний балл студентов по каждому предмету
% и выводит результат в виде столбцевой диаграммы.
% Возвращает массив, каждый элемент которого равен
% среднему баллу по предмету с соответствующим номером
% использование meanmark = groupprog(GROUP)
%    GROUP — массив структур с полями
%       Family (строка), Name (строка), Year (число),
%       Marks (вектор-строка с отметками)

% нахождение числа студентов в группе
N = max(size(GROUP));
```

```
% Определение количества курсов по информации для
% первого студента
Courses = length(GROUP(1).Marks);
% Инициализация массива meanmarks
meanmarks = zeros(1, Courses);
% Перебор курсов и вычисление средней успеваемости
for course = 1:Courses
   % Суммирование баллов, полученных каждым из студентов по
   % курсу с номером course
   for student = 1:N
      meanmarks(course) = meanmarks(course) +...
         GROUP(student).Marks(course);
   end
   % Нахождение среднего арифметического
   meanmarks(course) = meanmarks(course) / N;
end
% Построение столбцевой диаграммы
bar(meanmarks);
```

Запись данных массивов структур в текстовый файл

Работа с большими объемами данных, содержащихся в массивах структур, значительно облегчается при использовании текстовых файлов для хранения и считывания информации. Запись информации из массива структур в текстовый файл требует применения форматного вывода функцией `fprintf`.

Описание форматного вывода приведено в разд. "Форматный вывод" данной главы.

Список вывода `fprintf` состоит из названий полей, значения которых необходимо записать в текстовый файл, а форматы соответствуют типам данных, хранящимся в полях. Напишите файл-функцию `writegroup`, которая реализует построчный вывод значений всех полей структур массива с информацией об успеваемости группы студентов. Имя текстового файла и массив структур являются входными аргументами `writegroup`. Содержимое файла должно иметь организацию, схожую с табл. 8.1, например такую, как приведена в листинге 8.11. Установите фиксированное число отводимых позиций под вывод строк и чисел и выравнивание в области вывода по левому краю при помощи флага.

Листинг 8.11. Содержимое текстового файла с информацией о группе

```
Фамилия        Имя         Год   Оценки
Алексеев       Иван        1980  5  4  4  5  5  4
Васильев       Сергей      1981  3  4  4  3  5  4
Кашин          Павел       1979  4  3  4  4  5  4
Серова         Наталья     1981  4  3  3  5  4  5
Терехова       Ольга       1980  5  5  5  5  4  5
```

Воспользуйтесь листингом 8.12 в случае возникновения затруднений.

Листинг 8.12. Файл-функция `writegroup` для записи информации о группе в файл

```
function writegroup(filename, GROUP)
% Файл-функция для записи таблицы с успеваемостью группы
% студентов в текстовый файл.
% Использование  writegroup(filename, GROUP)
%    filename — имя файла
%    group — массив структур с полями
%       Family (строка), Name (строка), Year (число),
%       Marks (вектор-строка с шестью отметками)

% Нахождение числа студентов в группе
N = max(size(GROUP));
% Открытие файла с именем filename для записи
F = fopen(filename, 'w');
% Запись шапки таблицы
fprintf(F, '%-14s %-10s %-4s  %-6s\n',...
      'Фамилия', 'Имя', 'Год' , 'Оценки');
% Запись в файл содержимого полей каждой структуры в строку
for s = 1:N
   fprintf(F, '%-14s %-10s %4.0f...
     %2.0f %2.0f %2.0f %2.0f %2.0f %2.0f\n',...
     GROUP(s).Family, GROUP(s).Name, GROUP(s).Year, GROUP(s).Marks);
end
% Закрытие файла
fclose(F);
```

Конечно, эффективная работа с данными невозможна без умения считывать их из файла. Следующий раздел посвящен получению информации из текстового файла, имеющего определенный формат.

Считывание информации из текстового файла

Функция `fscanf` позволяет *последовательно* считывать данные, хранящиеся в текстовом файле и записывать их в переменные подходящих типов. Условно можно считать, что `fscanf` осуществляет обратное действие по отношению к `fprintf`, а именно, считывание в заданном формате. Содержимое текстового файла составляют такие элементы, как текст и числа. Текст всегда считывается в строковые переменные, а числа можно занести как в строковые, так и числовые переменные. Вызов функции `fscanf` производится с тремя входными аргументами — идентификатором файла, строкой с форматом и числом считываемых в данном формате объектов и одним выходным аргументом, в который записывается результат.

```
a = fscanf(идентификатор, 'формат', число считываемых элементов)
```

Для считывания строки предусмотрен формат `%s`, для целых чисел — `%d`, а для вещественных — `%g`. Необходимо следить за соответствием формата и данных, хранящихся в файле. Работу с функцией `fscanf` проще всего понять на нескольких простых примерах. Пусть, например, в файле student1.txt, состоящем из одной строки, содержится информация о студенте:

```
Александров 1990 учащийся 201 4.5
```

Файл-программа, приведенная в листинге 8.13, записывает `'Александров'` в строковую переменную `Family`, целое число (год) 1990 в переменную `Year`, `'учащийся'` в строковую переменную `Status`, целое число (номер группы) 201 в `Group`, вещественное число (средний балл) 4.5 в `MeanMark`. Считывание сопровождается выводом в командное окно для контроля.

Листинг 8.13. Поэлементное считывание

```
F = fopen('student1.txt', 'r');
Family = fscanf(F, '%s', 1)
Year = fscanf(F, '%d', 1)
Status = fscanf(F, '%s', 1)
Group = fscanf(F, '%s', 1)
MeanMark = fscanf(F, '%g', 1)
fclose(F);
```

Разобранный выше пример демонстрирует самый простой вариант использования `fscanf` — поэлементное считывание, при котором каждый вызов `fscanf` заносит в переменную соответствующее значение. Замените команды с `fscanf` (см. листинг 8.13) на одну

```
str = fscanf(F, '%s', 5)
```

и посмотрите содержимое `str`. В данном случае вся информация интерпретируется как текстовая и заносится в одну строковую переменную:

```
str =
Александров1990учащийся2014.5
```

Допустимо не указывать число считываемых объектов и вызывать функцию `fscanf` только с двумя входными аргументами. Если при этом используется формат `%s`, то все содержимое считается в строковую переменную так же, как показано выше. Числовые форматы `%d` и `%g` позволяют записать содержимое файла, состоящего из чисел в вектор. Считывание чисел продолжается до тех пор, пока не будет достигнут конец файла или не встретится текст. Пусть, например, в файле res.dat хранится следующая информация (необязательно в одну строку):

```
1.2274 1.4998
-2.0337 (результаты измерений)
```

Функция `fscanf` (листинг 8.14) заносит числовые значения в вектор `vect`, состоящий из трех элементов, и отображает его содержимое в командном окне. Для последующего считывания строки следует применить `fscanf` с форматом `%s`.

Листинг 8.14. Считывание чисел в вектор

```
F = fopen('res.dat', 'r');
vect = fscanf(F, '%g')
fclose(F);
```

Если числовая информация, представленная в файле, обладает матричной структурой, то задание вектора, содержащего размеры матрицы, в качестве третьего аргумента `fscanf`, позволяет записать информацию в матрицу. Создайте файл с матрицей размера три на четыре, приведенный в листинге 8.15.

Листинг 8.15

```
1.4  5.2 0.4 -1.1
-2.1 3.6 7.1  0.8
2.0  2.9 8.3 -0.1
```

Используйте команду `M = fscanf(F, '%g', [3 4])` для заполнения матрицы данными из файла и выведите ее содержимое в командное окно. Матрица считалась из файла в заданном виде.

Расположение матрицы в виде таблицы в файле необязательно, ее элементы могут быть записаны в строку, столбец или произвольным образом. Способ формирования матрицы задается вектором, указанным в списке входных параметров `fscanf`, а сами элементы считываются последовательно и помещаются в нужные позиции. Для того чтобы убедиться в этом, считайте из файла (см. листинг 8.15) матрицу с помощью команд

```
N = fscanf(F, '%g', [4 3])
K = fscanf(F, '%g', [6 2])
P = fscanf(F, '%g', [12 1])
R = fscanf(F, '%g', [2 6])
```

и посмотрите результат, отобразив содержимое матриц в командном окне. Важно только, чтобы в файле имелось достаточно элементов для формирования матрицы.

Информация, хранящаяся в файле, может представлять собой таблицу определенной структуры, например такой, как в листинге 8.11. Все столбцы имеют одинаковое назначение и содержат элементы одного типа. Использование массивов структур значительно облегчает работу с подобными файлами. Напишите самостоятельно файл-функцию `readgroup` для считывания данных в массив структур с полями `Family`, `Name` (строковые переменные), `Year` (числовая переменная), `Marks` (вектор-строка из целых чисел). Имя текстового файла задается во входном аргументе `readgroup`, а выходным аргументом является массив структур. Реализуйте следующий алгоритм в файл-функции.

1. Первая строка является шапкой таблицы, ее не следует заносить в структуру. Используйте функцию `fgetl` для считывания первой строки.
2. Примените поэлементное считывание в подходящих форматах для заполнения структур массива информацией, содержащейся в каждой строке файла, начиная со второй.
3. Используйте счетчик для обращения к структурам массива, который увеличивается на единицу перед работой с новой структурой.
4. Считывание производите в цикле `while`, пока не будет достигнут конец файла.
5. При считывании оценок в вектор-строку задайте требуемый размер (один на шесть) в третьем аргументе `fscanf`.

Доступ к полям структур массива описан в разд. "Массивы структур" данной главы.

Листинг 8.16 содержит возможный вариант файл-функции `readgroup`.

Листинг 8.16. Файл-функция `readgroup` для считывания информации из файла в массив структур

```
function GROUP = readgroup(filename)
% Файл-функция для считывания таблицы с успеваемостью группы
% студентов из текстового файла в массив структур.
% Использование  writegroup(filename, GROUP)
%    filename — имя файла
%    group — массив структур с полями
%       Family (строка), Name (строка), Year (число),
%       Marks (вектор-строка с шестью отметками)

% Открытие текстового файла с именем filename для записи
F = fopen(filename, 'rt');

% Считывание первой строки с шапкой таблицы
if feof(F) == 0
   line = fgetl(F);
end

% Обнуление счетчика студентов, каждая строка файла
% содержит информацию об одном студенте
count = 0;

% Последовательное считывание строк (начиная со второй)
% и распределение информации, содержащейся в ней,
% по полям структур массива GROUP
while feof(F) == 0
   count = count + 1; % увеличение счетчика студентов
   % Считывание строки с фамилией
   GROUP(count).Family = fscanf(F, '%s', 1);
   % Считывание строки с именем
   GROUP(count).Name = fscanf(F, '%s', 1);
   % Считывание года рождения
   GROUP(count).Year = fscanf(F, '%d', 1);
   % Считывание массива с оценками в вектор-столбец
```

```
   GROUP(count).Marks = fscanf(F, '%d', [1, 6]);
end
% Закрытие файла
fclose(F);
```

Операции с массивами структур

Добавление нового поля во все структуры массива осуществляется при помощи оператора присваивания, в левой части которого задается название нового поля, а в правой части — его значение для данной структуры массива. Дополните первую структуру массива `GR201` (см. листинг 8.8) полем `NBook`, содержащим номер зачетной книжки студента:

```
GR201(1).NBook = 127001;
```

Проверьте, что поле `NBook` добавилось *во все структуры* массива. Наберите `GR201` в командной строке, и нажмите <Enter> для получения информации о массиве структур. Значение 127001 занесено в поле `NBook` только первой структуры, поле `NBook` остальных структур пока содержат пустые массивы, например:

```
>> GR201(2).NBook
ans =
     []
```

Заполнение поля `NBook` остальных структур производится аналогичными операторами присваивания.

MatLab имеет ряд встроенных функций для облегчения работы с массивами структур. Функция `rmfield` предназначена для удаления какого-либо поля из всех структур массива. Первым входным аргументом является имя массива, а вторым — название поля, заключенное в апострофы. Преобразованная структура возвращается в выходном аргументе `rmfield`. Удалите, например, поле `Year` из структур массива `GR201` (см. листинг 8.8) и запишите результат в массив `g201`:

```
g201 = rmfield(GR201, 'Year')
```

Вторым аргументом функции `rmfield` может быть массив строк, содержащий названия удаляемых полей.

Формирование массива строк описано в разд. "Массивы строк" данной главы.

Удобную возможность для получения названия всех полей структуры предоставляет функция `fieldnames`, входным аргументом которой является массив структур, а выходным — *массив ячеек*, содержащий строки с названиями полей. Подробно про массивы ячеек написано ниже, однако доступ к элементам массива ячеек достаточно прост и осуществляется при помощи

индексации так, как указано в следующем примере (название второго поля записано в строковую переменную `str`):

```
>> GRFields = fieldnames(GR201)
GRFields =
    'Family'
    'Name'
    'Year'
    'Marks'
    'NBook'
>> str = GRFields(2,:)
str =
    'Name'
```

Строковые переменные с названием имени поля используются в качестве входных аргументов функций `getfield` и `setfield`, предназначенных, соответственно, для получения и установки значения поля структуры. Примеры применения `getfield` и `setfield` приведены ниже.

```
>> Name1 = getfield(GR201(1), 'Name');
>> GR201(2) = setfield(GR201(1), 'Name', 'Алексей');
```

При написании собственных файл-функций для работы со структурами используйте `isstruct` для проверки, является ли переменная структурой или нет. Входным аргументом `isstruct` задается переменная, подлежащая проверке, а результатом является либо "истина" (единица), либо "ложь" (ноль). Перед обращением к полю полезно убедиться в том, что оно определено в структуре при помощи функции `isfield`. Первый входной аргумент `isfield` является именем структуры, а второй — строковой переменной с названием поля.

Массивы ячеек

Кроме числовых массивов, массивов строк и структур, в MatLab определен еще один тип переменных — *массив ячеек*, который хорошо приспособлен для хранения разнородных данных. В данном разделе приведены только основные сведения, касающиеся массивов ячеек, которые понадобятся, в частности, при создании функций с переменным числом аргументов.

Массив ячеек состоит из ячеек или контейнеров, каждый из которых может содержать данные различных типов. Массив ячеек является более удобной организацией данных по сравнению с массивом структур в том случае, когда не представляется возможным выделить однотипные поля, или когда требуется упростить доступ к отдельным типам данных. В качестве примера рассмотрим обработку информации о результатах экспериментов, представленных в виде прямоугольных матриц размера два на три. Каждый эксперимент

производил студент, его фамилия, имя и номер группы хранятся в структуре. Эксперименты происходили при заданных преподавателем значениях параметров (рис. 8.3). Обратите внимание, что содержимое контейнеров действительно разнородно, даже структуры имеют отличающиеся поля, а в ячейку (4,3) занесена запись о невыполнении эксперимента.

Заполнение массива ячеек осуществляется поэлементно, причем для доступа к отдельным контейнерам применяется индексация, индексы заключаются в фигурные скобки. Способ присваивания значений определяется типом данных, например, при внесении структуры следует отделить поле от ячейки при помощи точки. Листинг 8.17 содержит файл-программу, заносящую информацию (см. рис. 8.3) в контейнеры массива ячеек `EXPER`.

ячейка 1,1	ячейка 1,2	ячейка 1,3
доц. Петров Е.А.	доц. Гришин С.Е.	асс.Зинин К.О.
ячейка 2,1	**ячейка 2,2**	**ячейка 2,3**
2.2 0.3 1.7	вариант N2	2.0 0.2 1.4
ячейка 3,1	**ячейка 3,2**	**ячейка 3,3**
Family Иванов **Name** Алексей **Group** 201	**Family** Сергеев **Name** Антон **Group** 202	**Family** Пашин **Name** Антон **Info** Вечерн. ф-т
ячейка 4,1	**ячейка 4,2**	**ячейка 4,3**
–1.33 0.35 1.74 0.99 0.98 0.78	–1.43 0.24 1.88 0.90 0.91 0.59	рез. не получен

Рис. 8.3. Содержимое ячеек

Замечание

Если имя создаваемого массива ячеек занято в рабочей среде под обычный массив, то перед поэлементным заполнением массива ячеек следует применить команду `clear` для очистки данного массива, например `clear EXPER`, иначе MatLab выдаст сообщение об ошибке.

Листинг 8.17. Заполнение массива ячеек

```
% ЗАПОЛНЕНИЕ ПЕРВОГО СТОЛБЦА МАССИВА EXPER
% Занесение строки с информацией о преподавателе в ячейку (1,1)
EXPER{1,1} = 'доц. Петров Е.А.';
% Занесение вектора параметров в ячейку (2,1)
EXPER{2,1} = [2.2 0.3 1.7];
% Занесение структуры с информацией о студенте в ячейку (3,1)
EXPER{3,1}.Family = 'Иванов';
EXPER{3,1}.Name = 'Алексей';
EXPER{3,1}.Group = 201;
% Занесение результатов эксперимента в ячейку (4,1)
EXPER{4,1} = [-1.33 0.35 1.74; 0.99 0.98 0.78];

% ЗАПОЛНЕНИЕ ВТОРОГО СТОЛБЦА МАССИВА EXPER
% Занесение строки с информацией о преподавателе в ячейку (1,2)
EXPER{1,2} = 'доц. Гришин С.Е.';
% Занесение вектора параметров в ячейку (2,2)
EXPER{2,2} = 'Вариант N2';
% Занесение структуры с информацией о студенте в ячейку (3,2)
EXPER{3,2}.Family = 'Сергеев';
EXPER{3,2}.Name = 'Антон';
EXPER{3,2}.Group = 202;
% Занесение результатов эксперимента в ячейку (4,2)
EXPER{4,2} = [-1.43 0.24 1.88; 0.90 0.91 0.59];

% ЗАПОЛНЕНИЕ ТРЕТЬЕГО СТОЛБЦА МАССИВА EXPER
% Занесение строки с информацией о преподавателе в ячейку (1,3)
EXPER{1,3} = 'асс. Зинин К.О.';
% Занесение вектора параметров в ячейку (2,3)
EXPER{2,3} = [2.3 0.3 1.5];
% Занесение структуры с информацией о студенте в ячейку (3,3)
EXPER{3,3}.Family = 'Пашин';
EXPER{3,3}.Name = 'Антон';
EXPER{3,3}.Info = 'Вечерн. ф-т';
% Занесение результатов эксперимента в ячейку (4,3)
EXPER{4,3} = 'рез. не получен';
```

В результате выполнения команд, приведенных в листинге 8.17, в рабочей среде создается массив ячеек EXPER. Просмотр содержимого массива EXPER из командной строки приводит к отображению информации в сжатом виде:

```
>> EXPER
EXPER =
    'доц. Петров Е.А.'    'доц. Гришин С.Е.'    'асс. Зинин К.О.'
          [1x3 double]          'Вариант N2'          [1x3  double]
          [1x1 struct]          [1x1 struct]          [1x1  struct]
          [2x3 double]          [2x3 double]          [1x20 char  ]
```

Функция celldisp с входным аргументом — именем массива ячеек выводит значения всех контейнеров в командное окно. Более наглядным способом представления больших массивов ячеек является отображение схемы массива в графическом окне при помощи cellplot. Команда cellplot(EXPER) приводит к появлению графического окна, изображенного на рис. 8.4. Массивы чисел или текстовые строки обозначаются макетами таблиц с соответствующим числом строк и столбцов, а записи — квадратами.

Способ заполнения массива ячеек, описанный выше, является одним из двух возможных в MatLab. Второй способ состоит в том, что обращение к ячейке происходит при помощи обычной индексации в круглых скобках, а в правой части оператора присваивания помещается ячейка, содержимое которой заключается в фигурные скобки. Например, заполнение ячейки (1,1) массива EXPER (листинг 8.17) может выглядеть следующим образом EXPER(1, 1) = {'доц. Петров Е.А.'}.

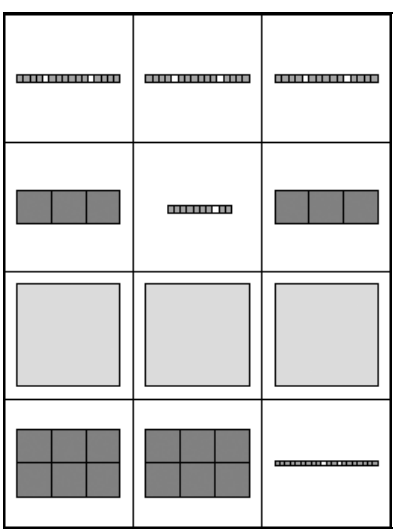


Рис. 8.4. Схема массива ячеек (celldisp)

Функция `size` находит размер массива ячеек:

```
>> size(EXPER)
ans =
     4     3
```

Замечание

Массив ячеек может представлять собой не только матрицу с контейнерами, но и вектор. Для доступа к содержимому контейнеров в этом случае применяется один индекс в фигурных скобках. Функция `length` выводит число ячеек.

Доступ к элементам контейнера массива ячеек осуществляется при помощи обращения к содержимому контейнера (индексы размещаются в фигурных скобках) в зависимости от типа хранящихся в нем данных. Напишите файл-функцию `students`, возвращающую массив строк с именами студентов, участвовавших в выполнении эксперимента. Входным аргументом файл-функции `students` (листинг 8.18) должен быть массив ячеек, имеющий такую же схему расположения данных, как и `EXPER`, только число столбцов может быть произвольным. Структуры с информацией о студентах размещаются в третьей строке массива ячеек. Число столбцов массива структур определите при помощи `size`. Используйте цикл `for` для перебора контейнеров со структурами. Строки с фамилиями студентов добавляйте в массив строк при помощи `char`.

Листинг 8.18. Файл-функция `students`, выделяющая фамилии студентов из массива ячеек

```
function strmas = students(CELLMAS)
% Файл-функция формирует массив строк с фамилиями
% студентов, участвующих в эксперименте.
% Использование strmas = students(CELLMAS)

% Определение размеров массива ячеек
SizeMas = size(CELLMAS);
% Нахождение числа студентов(информация о деятельности
% каждого студента хранится в столбце)
NStudents = SizeMas(2);
if NStudents >= 1
% Если число студентов больше или равно единице, то
% считываем из поля Family третьей ячейки первого столбца
% фамилию студента и заносим в массив строк strmas
   strmas = CELLMAS{3, 1}.Family;
end
```

```
% Продолжаем считывание по всем оставшимся столбцам,
% начиная со второго
for k = 2:NStudents
   % Считываем фамилию из поля Family третьей ячейки k-го столбца
   fam = CELLMAS{3, k}.Family;
   % Добавляем фамилию в массив строк
   strmas = char(strmas, fam);
end
```

Результатом работы файл-функции `students` является массив строк с фамилиями студентов, участвовавших в проведении эксперимента:

```
>> st = students(EXPER)
st =
Иванов
Сергеев
Пашин
```

Дополните файл-программу `students`, приведенную в листинге 8.18, проверкой, является ли входной аргумент массивом ячеек. Примените функцию `iscell`, которая возвращает "истину" (единицу), если ее входной аргумент — массив ячеек, и "ложь" (ноль) — в противном случае.

Приложения с интерфейсом из командной строки

Хорошо написанная программа позволяет пользователю не только производить вычисления, но и управлять процессом в ходе работы программы. Одним из способов взаимодействия пользователя с приложением является организация интерфейса при помощи командной строки. Перед выполнением некоторых действий программа приостанавливает работу и выдает запрос в командное окно. Ответ пользователя определяет дальнейший ход выполнения программы. Данный раздел посвящен написанию приложений с таким достаточно примитивным интерфейсом, позволяющим пользователю управлять работой программы.

Простой пример, программа-калькулятор

Хорошим демонстрационным примером приложения с интерфейсом из командной строки является программа-калькулятор, вычисляющая результат арифметических операций. Пользователь вводит число, затем выбирает арифметическое действие, вводит второе число и получает результат. Для написания такой программы требуется умение организовать вывод текста на

экран и получать ответ от пользователя. Вывод текста в командное окно осуществляется командой `disp`.

См. разд. "Проверка входных аргументов" главы 7.

Применение `disp` допускает только одностороннюю связь программы с пользователем. Интерактивное взаимодействие достигается функцией `input`. Выходной аргумент `input` является значением, введенным пользователем с клавиатуры в командную строку в ответ на запрос. Запросом является строка, указанная в качестве входного аргумента, например:

```
n  =  input('Введите значение n = ');
```

При выполнении данной команды в командное окно выводится сообщение `"Введите значение n="` и выполнение программы приостанавливается до тех пор, пока пользователь не введет число с клавиатуры и не нажмет <Enter>. После выполнения `input` переменной `n` присвоится введенное значение. Ошибочный ввод не числа, а, например символа, приведет к сообщению об ошибке и повторному появлению запроса на ввод. Функция `input` допускает ввод строк в строковую переменную, указанную в качестве выходного аргумента, причем вторым дополнительным входным аргументом следует указать `'s'`, например:

```
name = input('Введите имя ', 's' );
```

На запрос `input` следует ввести строку, не заключая ее в апострофы, строка записывается в переменную `name`.

Напишите файл-функцию, реализующую следующий алгоритм.

1. Считывание первого числа в числовую переменную (примените `input`).
2. Получение знака арифметической операции (`+`, `-`, `*` или `/`) и занесение его в строковую переменную (примените `input` со вторым аргументом `'s'`).
3. Считывание второго числа в числовую переменную.
4. Выполнение арифметического действия в зависимости от введенного знака операции (используйте оператор переключения `switch`).
5. Вывод результата в командное окно.

Программу оформите в виде файл-функции без входных и выходных аргументов. Текст требуемой файл-функции приведен в листинге 8.19.

Листинг 8.19. Файл-функция `calc`

```
function calc()
% Калькулятор с интерфейсом командной строки


% Считывание в числовую переменную числа, введенного пользователем
a = input('Введите первое число ');
```

```
% Считывание в строковую переменную операции, введенной пользователем
oper = input('Введите арифметическую операцию (+,-,*,/) ', 's');
% Считывание в числовую переменную числа, введенного пользователем
b = input('Введите второе число ');
% Вычисление результата арифметической операции
switch oper
case '+'
   a + b
case '-'
   a – b
case '*'
   a * b
case '/'
   a / b
otherwise
   error('Неизвестная арифметическая операция')
end
```

Работа с файл-функцией `calc` не требует никаких дополнительных пояснений для пользователя, ему достаточно следовать инструкциям, выводимым в командное окно, и результат будет получен. Файл-функция `calc` достаточно хорошо защищена от неправильного использования. Команда `input` без дополнительного аргумента `'s'` проверяет, что введено число и повторяет запрос в случае неправильного ввода. Ошибка при выборе арифметической операции обрабатывается ветвью `otherwise` оператора `switch`, которая выводит сообщение о неправильно введенной операции. Результат пользователь получает в стандартной переменной `ans`. Результаты работы файл-функции `calc` приведены ниже:

```
>> calc
Введите первое число 1.2
Введите арифметическую операцию (+,-,*,/) +
Введите второе число 3.1
ans =
    4.3000
```

Попробуйте изменить файл-функцию `calc` так, чтобы ответ имел более наглядный вид. Например, если производится сложение 1.2 и 3.1, то в командное окно выводится "1.2 + 3.1 = 4.3". Задача состоит в формировании строки с результатом (из введенных чисел, знака арифметической операции, знака "=", ответа) и выводе ее в командное окно. Обратите внимание, что строка результата формируется как из строк, например '+' и '=', так и из чи-

сел 1.2, 3.1, 4.3, которые хранятся в переменных. Очевидно, что для получения требуемой строки следует применить сцепление. Перед сцеплением необходимо преобразовать значение переменной в строку при помощи функции `num2str`, входной аргумент которой является числом или переменной с числовым значением, а выходной — строковой переменной, соответствующей данному числовому значению.

Вышеописанная модификация вида результата требует некоторых изменений в тексте файл-функции `calc`, приведенном в листинге 8.19. Операторы, реализующие ввод пользователя, остаются без изменений. Попытайтесь самостоятельно модернизировать файл-функцию `calc`, в случае возникновения вопросов обратитесь к листингу 8.20, в котором приведены операторы, обрабатывающие ввод пользователя и выдающие результат. Диалог файл-функции `calc` теперь выглядит следующим образом:

```
>> calc1
Введите первое число 1.2
Введите арифметическую операцию (+,-,*,/) +
Введите второе число 3.1
1.2+3.1=4.3
```

Листинг 8.20. Улучшение вида результата

```
function calc1()
% Калькулятор с интерфейсом командной строки

% Считывание в числовую переменную числа, введенного пользователем
a = input('Введите первое число ');
% Считывание в строковую переменную операции, введенной пользователем
oper = input('Введите арифметическую операцию (+,-,*,/) ', 's');
% Считывание в числовую переменную числа, введенного пользователем
b = input('Введите второе число ');
% Вычисление результата арифметической операции
switch oper
case '+'
   res = a + b;
case '-'
   res = a – b;
case '*'
   res = a * b;
case '/'
   res = a / b;
```

```
otherwise
   error('Неизвестная арифметическая операция')
end
% Преобразование чисел в строки
stra = num2str(a);
strb = num2str(b);
strres = num2str(res);
% Сцепление строк
str = strcat(stra, oper, strb, '=', strres);
% Вывод строки с результатом в командное окно
disp(str)
```

Вышеописанные примеры содержат оператор переключения `switch`, который производит вычисления в зависимости от знака арифметической операции, выбранного пользователем. Заметьте, что пользователь вводит числа и арифметическую операцию, которые входят в арифметическое выражение, понятное MatLab. Рациональнее создать строку с арифметическим выражением, выполнить ее, как команду MatLab и вывести результат, тем самым избежав перебора. Такой подход позволяет дополнительно использовать возведение в степень, ввод числа π, и придает программе более наглядный вид. Формирование команд в строке и их выполнение осуществляет функция `eval`, описанная в следующем разделе.

Формирование и исполнение команд, функция *eval*

Встроенная функция `eval` позволяет выполнить строку, которая содержит команду MatLab. Строка с командой является входным аргументом `eval`, например:

```
>> str = '1+2';
>> eval(str)
ans =
     3
```

Перед выполнением команду можно сформировать из нескольких строк:

```
>> str1 = 'y = sin(';
>> str2 = '3/2*pi';
>> str3 = ')';
>> str = strcat(str1, str2, str3);
>> eval(str)
y =
    -1
```

Используйте функцию `eval` в файл-программе `calc1`, приведенной в листинге 8.20, для формирования и вычисления результата арифметического выражения без перебора арифметических операций, которое было реализовано оператором `switch`. Файл-функция `calc2`, приведенная в листинге 8.21, решает поставленную задачу.

Листинг 8.21. Файл-функция `calc2`, демонстрирующая использование `eval`

```
function calc2()
% Калькулятор с интерфейсом командной строки

% Считывание в строковую переменную операции, введенной пользователем
a = input('Введите первое число ');
% Считывание в строковую переменную операции, введенной пользователем
oper = input('Введите арифметическую операцию (+,-,*,/) ', 's');
% Считывание в строковую переменную операции, введенной пользователем
b = input('Введите второе число ');
% Преобразование чисел в строки
stra = num2str(a);
strb = num2str(b);
% Формирование строки с арифметическим выражением (строка завершается
% точкой с запятой для подавления вывода промежуточных результатов
% в командное окно
str = strcat('res =', stra, oper, strb, ';');
% Вычисление арифметического выражения,
% его результат хранится в переменной res
eval(str);
% Формирование строки с результатом в виде '1.2+3.1=4.3'
strres = num2str(res);
% Сцепление строк
str = strcat(stra, oper, strb, '=', strres);
% Вывод строки с результатом в командное окно
disp(str)
```

Интерфейс пользователя, предоставляемый файл-функцией `calc2`, точно такой же, как в `calc1`, текст которой приведен в листинге 8.20. Формирование и выполнение команд MatLab при помощи `eval` позволило сделать алгоритм калькулятора более компактным и универсальным.

В качестве завершающего упражнения на использование строк и функции `eval` напишите программу для построения графиков функции одной пере-

менной с интерфейсом пользователя из командной строки. Работа с программой должна выглядеть следующим образом.

1. Запрос функции, график которой требуется построить. Пользователь задает формулу функции без учета поэлементных операций, т. е., например `exp(x)*sin(x)`, а не `exp(x).*sin(x)`.
2. Запрос границ отрезка. Пользователь вводит левую и правую границы.
3. Запрос цвета, стиля линий и маркеров. Пользователь выбирает номер цвета, стиля и маркера из предлагаемых списков (ограничьте выбор несколькими возможностями для уменьшения объема программы).
4. Вывод графика функции в графическое окно.
5. Запрос на продолжение работы с программой. Пользователь вводит `'y'` или `'n'`.

Запрограммируйте алгоритм в файл-функции без входных и выходных аргументов. При помощи оператора `if` проверьте, что пользователь вызвал файл-функцию без аргументов. Оформите весь диалог внутри цикла `while`, который работает, пока некоторая строковая переменная `flag` равна `'y'`. До цикла присвойте `flag` значение `'y'`. В конце цикла получите ответ от пользователя о продолжении работы при помощи `input` и занесите ответ в переменную `flag`. Запишите функцию, введенную пользователем, в строковую переменную, а пределы построения графика — в числовые переменные. Замену обычных операций на поэлементные выполните при помощи `strrep`. Вывод списка возможных цветов, типов линий и маркеров осуществите функцией `disp`. Используйте операторы переключения `switch` для формирования строки с командой `plot` в зависимости от номера цвета и стиля линии, и типа маркера, выбранных пользователем из списка. Возможный вариант текста файл-функции для визуализации функций с интерфейсом командной строки приведен в листинге 8.22.

Заметьте, что при некорректном вводе формулы для отображаемой функции программа `myplot` прекратит работу. Предположим, что пользователь ввел `ln(x)` вместо встроенной `log(x)`. Ошибка возникнет при выполнении строки с командой `eval` при вычислении вектора *y* значений функции. Улучшите самостоятельно файл-функцию `myplot`, заключив блоки с возможными источниками ошибок в конструкцию `try...catch`, которая предназначена для обработки исключительных ситуаций.

Листинг 8.22. Пример файл-функции `myplot` с интерфейсом командной строки

```
function myplot()
% Построение графиков функций,
% диалог с пользователем из командной строки
```

```
% Проверка входных и выходных аргументов
if nargout ~= 0
   error('Функция не имеет выходных аргументов')
end
if nargin ~= 0
   error('Функция не имеет входных аргументов')
end

disp('ПРОГРАММА ДЛЯ ПОСТРОЕНИЯ ГРАФИКОВ')
disp('ФУНКЦИЙ ОДНОЙ ПЕРЕМЕННОЙ')
flag = 'y'
while flag == 'y'
   % Запрос функции
   strfun = input('Введите функцию, например exp(x)*sin(x) ', 's');
   % Запрос левой границы отрезка
   left = input('Введите левую границу отрезка, например -1.2 ');
   % Запрос правой границы отрезка
   right = input('Введите правую границу отрезка, например 1.3 ');
   % Запрос типа линии
   disp('Выберите тип линии:')
   disp('         сплошная         1')
   disp('         пунктирная       2')
   disp('         штриховая        3')
   linetype = input('Введите 1, 2, или 3 ');
   % Запрос цвета линии
   disp('Выберите цвет линии:')
   disp('красная   1')
   disp('синяя     2')
   disp('черная    3')
   linecolor = input('Введите 1, 2, или 3 ');
   % Запрос маркера
   disp('Выберите тип маркера:')
   disp('без маркера   0')
   disp('звездочка     1')
   disp('кружок        2')
   marker = input('Введите 0, 1, или 2 ');
   % Обработка ввода пользователя, формирование строки
   % с командой plot и ее выполнение.
   % Замена арифметических операций на поэлементные
```

```
strfun = strrep(strfun, '*', '.*');
strfun = strrep(strfun, '/', './');
strfun = strrep(strfun, '^', '.^');

% Генерация вектора значений аргумента
x = [left : (right-left)/30 : right];
% Генерирование строки для вычисления вектора значений функции
% строка заканчивается точкой с запятой для подавления вывода
% промежуточных результатов на экран
strhelp = strcat('y=', strfun, ';');
% Вычисление вектора значений функции
eval(strhelp); % значения функции теперь хранятся в векторе y
% Формирование строки с командой plot
strplot = 'plot(x, y,';
% Добавление информации о типе линии
switch linetype
   case 1
      strplot = strcat(strplot, '''-');
   case 2
      strplot = strcat(strplot, ''':');
   case 3
      strplot = strcat(strplot, '''--');
   otherwise
      error('Неизвестный тип линии')
end
% Добавление информации о цвете линии
switch linecolor
case 1
   strplot = strcat(strplot, 'r');
case 2
   strplot = strcat(strplot, 'b');
case 3
   strplot = strcat(strplot, 'k');
otherwise
   error('Неизвестный цвет линии')
end
% Добавление информации о маркере
switch marker
```

```
   case 0
      strplot = strcat(strplot, '''');
   case 1
      strplot = strcat(strplot, '*''');
   case 2
      strplot = strcat(strplot, 'o''');
   otherwise
      error('Неизвестный тип маркера')
   end
   % Завершение генерации строки с командой plot
   strplot = strcat(strplot, ')')
   % Выполнение команды plot
   eval(strplot)
   % Запрос на продолжение работы
   flag = input('Продолжить работу? (y — да, n — нет)', 's');
end
```

Обработку ввода пользователя можно производить в бесконечном цикле, задав в качестве условия цикла `while` единицу, а выход из цикла производить оператором `break`.

Файл-функции с переменным числом аргументов

Большинство стандартных функций MatLab допускают обращение к ним с различным числом входных и выходных аргументов, например функция для нахождения минимума `fminbnd`, которую можно вызвать с одним или двумя выходными аргументами и с тремя или более входными. Обратитесь к встроенной справке по `fminbnd` при помощи `help` с именем функции для того, чтобы узнать о всевозможных вариантах вызова. В данном разделе описано создание собственных файл-функций, предназначенных для работы с переменным числом аргументов. Знание основных сведений, касающихся массивов ячеек, необходимо для написания универсальных файл-функций.

См. разд. "Массивы ячеек" данной главы.

Разберем сначала принцип организации файл-функции с переменным числом входных аргументов на следующем примере. На плоскости задано *произвольное* количество кругов (координатами центров и радиусами `x1,y1,R1`, `x2,y2,R1` и т. д.) и точка с координатами `(px,py)`. Требуется определить, лежит ли точка внутри какого-либо круга, или нет (см. рис. 8.5).

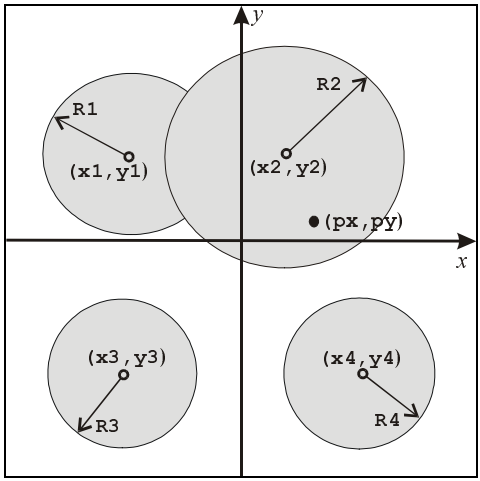


Рис. 8.5. Расположение кругов и точки

Напишем файл-функцию `point`, входными аргументами которой будут координаты точки и нескольких вектор-строк из трех элементов, задающих положение кругов, а выходным — число, единица или ноль в зависимости от попадания точки в один из кругов. Для случая, изображенного на рис. 8.5, вызов функции `point` будет выглядеть следующим образом:

```
f = point(px,py,[x1 y1 R1],[x2 y2 R2],[x3 y3 R3], [x3 y3 R3])
```

Первые два входных аргумента являются обязательными, а число векторов-строк соответствует числу кругов и может быть различным. MatLab предлагает простой способ решения проблемы. Все входные аргументы упаковываются в специальный массив (вектор) ячеек `varargin`, каждый аргумент занимает ровно одну ячейку так, как показано на рис. 8.6.

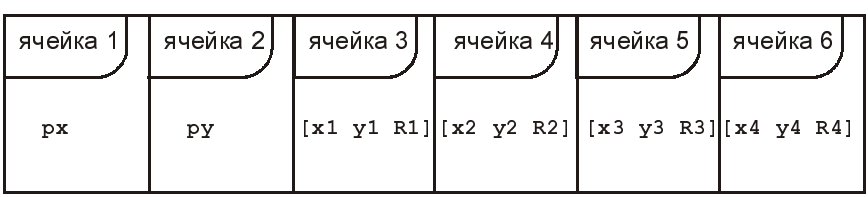


Рис. 8.6. Хранение аргументов в `varargin`

Массив `varargin` является единственным входным аргументом файл-функции, ее заголовок выглядит так:

```
function where = point(varargin)
```

Доступ к входным аргументам, т. е. ячейкам, производится при помощи заключения индекса в фигурные скобки и последующим обращении с содержимым в зависимости от типа хранимых данных. Например, ячейка `varargin{1}` содержит абсциссу точки, `varargin{2}` — ординату, `varargin{5}(3)` — радиус третьего круга и т. д.

Напишите файл-функцию `point`, придерживаясь следующего алгоритма.

1. Нахождение длины массива `varargin` при помощи `length`.
2. Проверка длины `varargin`, если она меньше трех, то задано недостаточно аргументов — выход по ошибке (используйте `error`).
3. Извлечение из `varargin` координат точки в переменные.
4. Извлечение из `varargin` координат центров кругов и радиусов в три вектора подходящей длины (примените цикл `for`).
5. Перебор всех кругов, вычисление расстояния от центра круга до точки и сравнение его с радиусом. Если найден хотя бы один круг, в который попала заданная точка, то выходному аргументу следует присвоить значение, равное единице, и прекратить перебор выходом из цикла командой `break`.

Листинг 8.23 содержит текст требуемой файл-функции, снабженный необходимыми комментариями.

Листинг 8.23. Файл-функция `point` с переменным числом входных аргументов

```
function where = point(varargin)
% Файл-функция определяет попадание точки с заданными
% координатами (px, py) в круги с центрами
% в (x1,y1), (x2, y2) и т. д. и радиусами R1, R2 и т. д.
% Возвращает
%        1 — в случае попадания
%        0 — в случае непопадания
% Использование where = point(px,py,[x1,y1,R1],[x2,y2,R2],...)

% Проверка числа входных аргументов (числа ячеек varargin)
if length(varargin) < 3
   error('Недостаточно входных аргументов')
end
% Выделение координат точки из первых двух ячеек
Xpoint = varargin{1};
Ypoint = varargin{2};
% Нахождение числа заданных кругов
```

```
% (число ячеек varargin без первых двух)
Ncircle = length(varargin) — 2;
% Извлечение координат центров и радиусов кругов
for i = 1:Ncircle
   Xcircle(i) = varargin{i+2}(1);
   Ycircle(i) = varargin{i+2}(2);
   Rcircle(i) = varargin{i+2}(3);
end
% Полагаем where=0, т. е. пока нет ни одного нужного круга
where = 0;
% Перебор кругов в цикле
for i = 1:Ncircle
   % Вычисление расстояния от точки до центра текущего круга
   dist = sqrt((Xpoint-Xcircle(i))^2+(Ypoint-Ycircle(i))^2);
   % Сравнение расстояния с радиусом круга
   if  dist <= Rcircle(i)
      where = 1;  % Требуемый круг найден
      break       % Дальше проверять нет смысла
   end
end
```

Дополните файл-функцию `point` исследованием правильности задания входных аргументов. Следует убедиться, что первые две ячейки массива `varargin` содержат вещественные числа, а остальные — векторы длиной, равной трем, также состоящие из вещественных чисел (листинг 8.24). Используйте функции `isnumeric` (проверяет, является ли ее входной аргумент числовым массивом) и `isreal` (проверяет, является ли ее входной аргумент вещественным массивом или массивом строк).

Листинг 8.24. Проверка входных аргументов файл-функции `points`

```
if ~isnumeric(varargin{1}) | ~isreal(varargin{1}) |...
      max(size(varargin{1}) ~= 1)
   error('Аргумент N1 должен быть вещественным числом')
end
if ~isnumeric(varargin{2}) | ~isreal(varargin{2}) |...
      max(size(varargin{2}) ~= 1)
   error('Аргумент N2 должен быть вещественным числом')
end
for i = 3:length(varargin)
```

```
   if ~isnumeric(varargin{i}) | ~isreal(varargin{i}) |...
         min(size(varargin{i})) ~= 1 | length(varargin{i}) ~=3 |...
         varargin{i}(3) < 0
      str1 = 'Аргумент N';
      str2 = num2str(i);
      str3 = ' долж. быть вещ. вектором длиной 3 с третьим эл-том >= 0';
      strerror = strcat(str1, str2, str3);
      error(strerror)
   end
end
```

Операторы, приведенные в листинге 8.24, следует поместить после проверки длины массива ячеек `varargin`. Теперь файл-функция `point` защищена от неправильного использования, например:

```
>> point(1,1,[0,3,-1])
??? Error using ==> point
Аргумент N3 долж. быть вещ. вектором длиной 3 с третьим эл-том >= 0
```

Усовершенствуйте файл-функцию `point`, добавив операторы, выводящие в графическое окно заданную точку и круги для получения наглядного результата. Очевидно, что требуется применить команды `plot` для построения параметрически заданных функций (окружностей) и заданной точки.

Визуализация параметрически заданных функций одной переменной описана в разд. "Графики параметрических и кусочно-заданных функций" главы 3.

Блок операторов, осуществляющих отображение данных в графическое окно, приведен в листинге 8.25. Данный блок следует разместить после извлечения параметров из массива ячеек `varargin`.

Листинг 8.25. Вывод данных в графическое окно

```
% Отображение данных в графическое окно
figure; % Создание окна
% Построение окружностей
t = [0:pi/20:2*pi]; % задание вектора параметра
for i = 1:Ncircle
   % Вычисление векторов, соответствующих параметрически
   % заданным функциям, которые определяют окружности
   x = Rcircle(i)*cos(t) + Xcircle(i);
   y = Rcircle(i)*sin(t) + Ycircle(i);
   plot(x,y) % построение окружности
   hold on
end
```

```
% Вывод точки красным маркером
plot(Xpoint,Ypoint, 'or')
hold off
axis square % сохранение одинакового масштаба осей
```

Работа файл-функции `point` сопровождается теперь появлением графического окна с расположением кругов и точки, например вызов

```
f = point(1,1,[0 3 3], [2 -1 3.3], [-1 -2 2 ]);
```

приводит к появлению графика, изображенного на рис. 8.7.

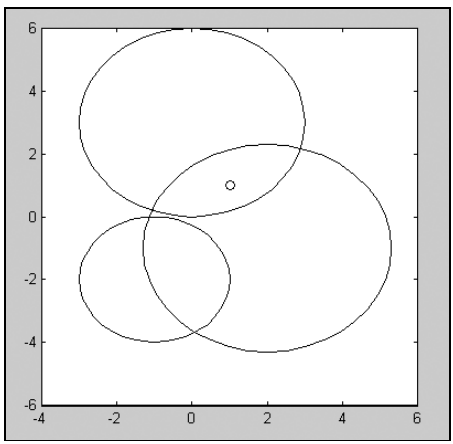


Рис. 8.7. Графическое окно с расположением кругов и точки

Список входных параметров в заголовке файл-функции может содержать комбинацию обязательных и произвольных аргументов. Например, заголовок файл-функции `point` может иметь такой вид:

```
function where = point (px, py, varargin)
```

В данном случае в массив ячеек упаковываются векторы из трех элементов с координатами центров и радиусами кругов, начиная с *первой ячейки* массива, а параметры `px` и `py` передают координаты заданной точки. Массив `varargin` всегда указывается последним!

Обратимся теперь к написанию файл-функций с переменным числом выходных аргументов. Продолжите работу с файл-программой `point`. Задача теперь состоит в том, чтобы `point` допускала следующие варианты обращения к ней.

- `f = point(px,py,[x1 y1 R1],[x2 y2 R2],...)`. В `f` записывается ноль или единица, в зависимости от принадлежности точки какому-либо кругу.

- `[f,Nc] = point(px,py,[x1 y1 R1],[x2 y2 R2],...)`. В `f` записывается ноль или единица, а в `Nc` — число кругов, которым принадлежит точка.
- `[f,Nc,Num] = point(px,py,[x1 y1 R1],[x2 y2 R2],...)`. В `f` записывается ноль или единица, в `Nc` — число кругов, которым принадлежит точка, в массив `Num` — номера этих кругов в списке входных аргументов.

Произвольное число выходных аргументов возвращается файл-функцией в специальном массиве ячеек `varargout`, для чего следует предусмотреть операторы, записывающие выходные аргументы в соответствующие ячейки данного массива. Заголовок файл-функции с переменным числом входных и выходных аргументов в М-файле выглядит следующим образом:

```
function [varargout] = myfun(varargin)
```

Аргументы, указание которых обязательно, могут быть помещены в списки входных и выходных аргументов до `varargin` и `varargout` соответственно. Например, выходной аргумент функции `point`, принимающий значение ноль или единица в зависимости от попадания точки в круги, указывается всегда, поэтому имеет смысл выделить его в списке выходных аргументов. Ниже приведен заголовок файл-функции `point`, которая допускает три варианта вызова, перечисленные выше.

```
function [where, varargout] = point(varargin)
```

Внесите необходимые дополнения в текст файл-функции `point` (см. листинги 8.23—8.25).

1. Измените заголовок, предусмотрев один фиксированный выходной аргумент `where` и произвольное число дополнительных при помощи `varargout`.
2. Перебирайте все круги в цикле `for`, запоминая число кругов и их номера, содержащих заданную точку (оператор `break` уже не нужен).
3. Запишите результаты в соответствующие ячейки массива в зависимости от числа выходных аргументов, с которыми была вызвана файл-функция (используйте `nargout` для определения их числа и оператор `switch` для заполнения требуемых ячеек).

Обратитесь к листингу 8.26 в случае возникновения вопросов. Листинг не содержит проверки числа и типа входных параметров и операторов вывода исходных данных в графическое окно. Данные блоки остаются без изменений (см. листинги 8.24—8.25).

Листинг 8.26. Файл-функция `points` с переменным числом входных аргументов

```
function [where, varargout] = point(varargin)
% Файл-функция определяет попадание точки с заданными
```

```
% координатами (px, py) в круги с центрами
% в (x1,y1), (x2, y2) и т. д. и радиусами R1, R2 и т. д.
% Использование
%   where = point(px,py,[x1,y1,R1],[x2,y2,R2],...)
%        where равно 1, если точка попала в какой-либо круг,
%                    0, в противном случае
%   [where, NC] = point(px,py,[x1,y1,R1],[x2,y2,R2],...)
%        NC равно числу кругов, содержащих точку
%   [where, NC, Nums] = point(px,py,[x1,y1,R1],[x2,y2,R2],...)
%        В вектор Nums записываются номера кругов, содержащих точку

% Выделение координат точки из первых двух ячеек
Xpoint = varargin{1};
Ypoint = varargin{2};
% Нахождение числа заданных кругов
% (число ячеек varargin без первых двух)
Ncircle = length(varargin) — 2;
% Извлечение координат центров и радиусов кругов
for i = 1:Ncircle
   Xcircle(i) = varargin{i+2}(1);
   Ycircle(i) = varargin{i+2}(2);
   Rcircle(i) = varargin{i+2}(3);
end
% Сначала where=0, т. е. пока нет ни одного нужного круга
where = 0;
% Сначала число кругов, содержащих точку, равно нулю
NC = 0;
% Перебор кругов в цикле
for i = 1:Ncircle
   % Вычисление расстояния от точки до центра текущего круга
   dist = sqrt((Xpoint-Xcircle(i))^2+(Ypoint-Ycircle(i))^2);
   % Сравнение расстояния с радиусом круга
   if  dist <= Rcircle(i)
      where = 1;  % Требуемый круг найден
      % Увеличение числа найденных кругов на единицу
      NC = NC + 1;
      % Сохранение номера круга в массиве Nums
      Nums(NC) = i;
   end
end
```

```
% Запись полученных результатов в выходной массив ячеек в зависимости
% от числа выходных аргументов, с которыми была вызвана point
switch nargout
case(2)
   % Было указано два выходных аргумента, следовательно, надо записать
   % только число кругов в первую ячейку varargout
   varargout{1} = NC
case(3)
   % Было указано три выходных аргумента, следовательно, надо записать
   % число кругов и массив с их номерами в первые две ячейки varargout
   varargout{1} = NC;
   varargout{2} = Nums;
end
```

Итак, написание файл-функции с переменным числом входных и выходных аргументов не представляет большого труда, но требует понимания работы с массивами ячеек. В качестве упражнения измените файл-функцию `point` так, чтобы она содержала два обязательных входных аргумента — координаты точки. Заголовок `point` должен выглядеть следующим образом:

```
function [where, varargout] = point(px, py, varargin)
```

Следующий раздел посвящен написанию файл-функций, входными аргументами которых, наряду с обычными переменными, являются другие файл-функции.

Функции от функций

Предположим, что для исследования функций требуется запрограммировать собственный алгоритм, который должен оперировать с достаточно большим набором функций. Алгоритм естественно оформить в виде файл-функции, но тогда при работе с новой функцией придется внести изменения в M-файл. MatLab предоставляет возможность написания файл-функций, входными аргументами которых являются другие файл-функции.

Функции `fminsearch`, `fzero`, `quad`, *описанные в главе 6, подразумевают именно такой способ вызова.*

Имя файл-функции передается в строковой переменой, а ее вычисление производится при помощи команды `feval`. Например, синус можно вычислить обычным способом, вызвав `sin(x)`, или используя `feval` с входными аргументами — названием `'sin'`, и аргументом `x`:

```
>> x = pi/2;
>> feval('sin', x)
```

```
ans =
     1
```

Применение `feval` разберем на следующем простом примере. Пусть алгоритм исследования функций является обычным методом половинного деления для нахождения корня уравнения $f(x)=0$. В качестве $f(x)$ может выступать как математическая функция, определенная в MatLab, так и заданная пользователем в M-файле. Алгоритм метода половинного деления очень простой.

1. Определяется отрезок, на границах которого $f(x)$ принимает значения разных знаков.
2. Производится деление отрезка пополам и из получившихся половин выбирается та, на границах которой знаки $f(x)$ различны (если в центре отрезка функция равна нулю, то корень найден и вычисления останавливаются).
3. Процесс продолжается до тех пор, пока длина получившегося отрезка не станет меньше заданной точности вычислений.
4. За приближение к корню принимается любая из границ отрезка.

Напишите файл-функцию `half`, реализующую метод половинного деления. Обращение к `half` должно выглядеть следующим образом:

```
r = half('myf', a, b, e)
```

Здесь `myf` — имя исследуемой функции; `a` и `b` — границы первоначального отрезка; `e` — точность вычислений.

Листинг 8.27. Файл-функция `half` для решения уравнений методом половинного деления

```
function root = half(fname, left, right, epsilon)
% Файл-функция находит корень уравнения f(x)=0
% методом половинного деления
% Использование
% root = half(fname, left, right, epsilon)
%      fname — имя файл-функции, вычисляющей f(x)
%      left, right — левая и правая границы отрезка
%      epsilon — точность вычислений

% Проверка значений функции на границах отрезка
if feval(fname, left)*feval(fname, right) > 0
   error('Одинаковые знаки функции на границах отрезка')
end
```

```
% Деление отрезка пополам
while (right – left) > epsilon
   center = (right + left)/2; % вычисление середины
   % Проверка на равенство f(x) нулю в center
   if feval(fname, center) == 0
      break    % найден точный корень, дальше делить нет смысла
   end
   % Выбор нужной половины отрезка, на границах которой
   % f(x) принимает значения разных знаков
   if feval(fname, left)*feval(fname, center) < 0
      right = center;
   else
      left = center;
   end
end
% Приближенное значение корня равно координате любой границы
% последнего полученного отрезка
root = center;
```

Теперь в качестве исследуемой функции может выступать как встроенная математическая функция MatLab, так и функция с одним входным аргументом, описанная пользователем в M-файле, например:

```
>> r = half('sin', 3, 3.5, 1.0e-4)
r =
    3.1415
```

Путь к M-файлу, в котором описана исследуемая функция, должен быть установлен в MatLab при помощи навигатора путей или соответствующей команды.

Задание путей поиска описано в разд. "Установка путей" главы 5.

В общем случае все входные аргументы исследуемой функции задаются в списке аргументов `feval` через запятую после имени функции, например, следующие два вызова некоторой функции `myfun` эквивалентны:

```
[a, b] = myfun(x, y, z)
[a, b] = feval('myfun', x, y, z)
```

Улучшите интерфейс файл-функции `half` (листинг 8.27) так, чтобы точность вычислений была необязательным параметром. Если точность не задана, то по умолчанию она полагается 10^{-3}. Предусмотрите обращение к `half` с одним или двумя выходными параметрами. Во второй дополнительный пара-

метр записывается значение функции в найденном приближенном значении корня. Очевидно, что задача сводится к написанию файл-функции с переменным числом аргументов.

См. разд. "Файл-функции с переменным числом аргументов" данной главы.

Листинг 8.28 содержит текст требуемой файл-функции. Обратите внимание, что в списках входных и выходных переменных присутствуют как обязательные параметры, так и дополнительные, передающиеся при помощи `varargin` и `varargout`.

Листинг 8.28. Файл-функция `half` с переменным числом аргументов

```
function [root, varargout] = half(fname, left, right, varargin)
% Файл-функция находит корень уравнения f(x)=0
% методом половинного деления
% Использование
% root = half(fname, left, right, epsilon)
%      fname — имя файл-функции, вычисляющей f(x)
%      left, right — левая и правая границы отрезка, на
%                    котором находится корень
%      epsilon — точность вычислений, если не задана, то
%                по умолчанию 1.0e-03
% [root, Fun] = half(fname, left, right, epsilon)
%      Fun = f(root)

% Если число входных аргументов равно четырем, то последний
% аргумент содержит точность вычислений, а если трем, то точность
% устанавливается по умолчанию 1.0e-03
switch nargin
case(4)
   epsilon = varargin{1};
case(3)
   epsilon = 1.0e-03;
otherwise
   error('Может быть три или четыре входных аргумента')
end
% Проверка значений функции на границах отрезка
if feval(fname, left)*feval(fname, right) > 0
   error('Одинаковые знаки функции на границах отрезка')
end
```

```
% Деление отрезка пополам
while (right – left) > epsilon
   center = (right + left)/2; % вычисление середины отрезка
   % проверка на равенство f(x) нулю в середине отрезка
   if feval(fname, center) == 0
      break % найден точный корень, дальше делить нет смысла
   end
   % Выбор нужной половины отрезка, на границах которой
   % f(x) принимает значения разных знаков
   if feval(fname, left)*feval(fname, center) < 0
      right = center;
   else
      left = center;
   end
end
% Приближенное значение корня равно координате любой границы
% последнего полученного отрезка
root = center;
if nargout == 2
   varargout{1} = feval(fname, root);
end
```

Теперь файл-функция `half` стала более универсальной, например, возможно такое обращение к ней:

```
>> [r, f] = half1('sin', 3, 3.5)
r =
    3.1416
f =
 -8.9089e-006
```

Подфункции и приватные функции

Оформление алгоритма в одной файл-функции не всегда удобно. Некоторые стандартные часто повторяющиеся действия следует оформить в виде отдельных функций, связанных с основным алгоритмом. MatLab предоставляет два способа решения этой задачи — *подфункции* и *приватные функции*.

Подфункции

Вызов функции, например из командной строки, приводит к поиску соответствующего M-файла в путях поиска MatLab. Когда требуемый M-файл

найден, начинается последовательное выполнение операторов, содержащихся в теле функции. Предположим, что в файл-функции `simple`, хранящейся в файле simple.m, часто приходится вычислять некоторое выражение. Конечно, можно простым копированием строк добавить соответствующие операторы (листинг 8.29).

Листинг 8.29. Файл-функция `simple` в файле simple.m

```
function simple;
x = 1.1;
y = 2.1;
f1 = x^3 – 2*y^3 + 3*(x^2+y^2)- x*y + 9
x = 3.1;
y = 4.2;
f2 = x^3 – 2*y^3 + 3*(x^2+y^2)- x*y + 9
x = -2.8;
y = 0.7;
f3 = x^3 – 2*y^3 + 3*(x^2+y^2)- x*y + 9
```

Проще и нагляднее определить вычисляемое выражение в подфункции `f` с двумя входными и одним выходным аргументом и разместить ее в том же M-файле simple.m (листинг 8.30).

Листинг 8.30. Файл-функция `simple` с подфункцией `f` в файле simple.m

```
function simple;
% Основная функция
f1 = f(1.1,2.1)
f2 = f(3.1,4.2)
f3 = f(-2.8,0.7)

function z = f(x,y)
% Подфункция
z = x^3 – 2*y^3 + 3*(x^2+y^2)- x*y + 9;
```

Первая функция `simple` является *основной функцией* в simple.m, именно ее операторы выполняются, если пользователь вызывает `simple`, например, из командной строки. Каждое обращение к подфункции `f` в основной функции приводит к переходу к размещенным в подфункции операторам и последующему возврату в основную функцию.

Файл-функция может содержать одну или несколько подфункций со своими входными и выходными параметрами, но основная функция может быть только одна. Основная функция обменивается информацией с подфункциями при помощи входных и выходных параметров. Переменные, определенные в подфункциях и в основной функции, являются локальными, они доступны только в пределах функции. Листинг 8.31 содержит пример файл-функции с подфункцией, приводящий к ошибке!

Листинг 8.31. Недопустимое использование локальных параметров

```
function simple;
ALPHA = 5.3;
BETA = 9.1;
f1 = f(1.1,2.1)
function z = f(x,y)
z = x^3 — 2*y^3 + 3*(x^2+y^2)- x*y + 9 + ALPHA*BETA;
```

Один из возможных вариантов использования переменных, которые являются общими для всех функций M-файла, состоит в объявлении данных переменных в начале основной функции и подфункции как глобальных, при помощи `global` со списком имен переменных, разделяемых пробелом (листинг 8.32).

Листинг 8.32. Объявление глобальных переменных

```
function simple;
global ALPHA BETA
ALPHA = 5.3;
BETA = 9.1;
f1 = f(1.1,2.1)
function z = f(x,y)
global ALPHA BETA
z = x^3 — 2*y^3 + 3*(x^2+y^2)- x*y + 9 + ALPHA*BETA;
```

Следует иметь в виду, что лучшим способом обмена переменными между основной функцией и подфункциями является передача их в параметрах подфункции. Значение глобальных переменных может быть случайно изменено в рабочей среде или при вызове другой файл-программы или файл-функции, которая использует одноименные глобальные переменные.

Приватные функции

Определенная структура каталога с пользовательскими файл-функциями позволяет задать некоторые вспомогательные функции, которые используются только файл-функциями, содержащимися в M-файлах данного каталога, а для файл-функций из других каталогов являются недоступными. Для этого следует создать подкаталог с именем private и разместить в нем вспомогательные файл-функции. Рис. 8.8 поясняет доступ к приватным функциям.

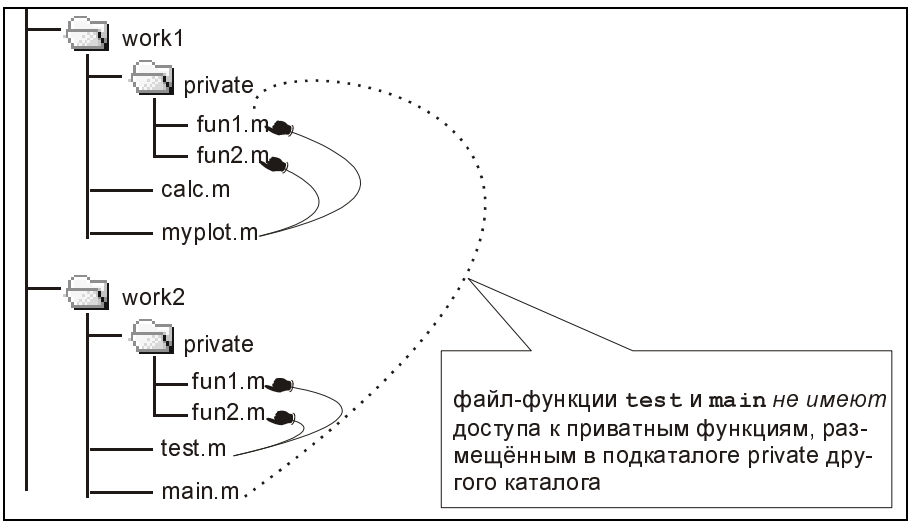


Рис. 8.8. Схема доступа к приватным функциям

Глава 9



Дескрипторная графика

При визуализации результата бывает необходимо управлять видом получающихся графиков. Интерактивное редактирование графиков при помощи редактора графиков или редактора свойств, описанное в *главе 4*, не подходит при разработке собственных программ. Окончательный вид графических результатов должен формироваться программой, а не пользователем после завершения работы программы. MatLab предоставляет в распоряжение программиста так называемую *дескрипторную графику* (Handle Graphics), основанную на низкоуровневых графических функциях. Дескрипторная графика, в отличие от высокоуровневой графики, которой посвящена *глава 3*, позволяет получить доступ к свойствам всех графических объектов при помощи соответствующих функций. Использование этих функций в собственной программе приводит к получению требуемого результата. Применение дескрипторной графики в полном объеме требует знания основ объектно-ориентированного программирования. Назначение данной главы состоит в описании тех средств, которые являются существенными при разработке приложений с графическим интерфейсом, предназначенных для решения задач исследовательского характера.

Графические объекты

Поскольку MatLab является объектно-ориентированной системой, то все элементы (графическое окно, оси, линии, поверхности, текстовые области, линии и т. д.) являются *объектами*. Объекты представлены иерархической структурой, упрощенный вид которой был приведен на рис. 4.5. Каждый объект имеет свойства, изменение их значений позволяет добиться требуемого вида объекта. Четвертая глава посвящена изменению свойств объектов при помощи редактора графиков или редактора свойств. Задание значения свойства в редакторе состоит из двух этапов:

1. Выбор нужного объекта из иерархического списка или указание его щелчком мыши, при этом объект становится текущим.
2. Установка нужных свойств при помощи элементов управления диалогового окна, соответствующего текущему объекту.

При написании собственных программ, осуществляющих отображение результатов в графическом виде, необходимо знать доступные свойства каждого объекта и уметь изменять их значения в программе.

Свойства графических объектов

Встроенные функции MatLab позволяют получить и установить значения свойств любого из объектов по своему усмотрению. Применение данных функций состоит, по существу, из тех же этапов, что и задание значений в редакторе графиков или свойств, а именно — выбор объекта и установка значения свойства.

Функции *set* и *get*, текущие объекты

Постройте график синуса на отрезке [0, 10], используя `plot`. Команды, которые понадобятся при чтении данного раздела, достаточно короткие, поэтому можно задавать их из командной строки; писать файл-программу или файл-функцию в М-файле нет смысла. Итак, операторы

```
>> x = [0:0.1:10];
>> y = sin(x);
>> plot(x, y)
```

приводят к появлению графика функции. Графические элементы MatLab являются объектами, в данном случае результатом работы `plot` явилось создание трех графических объектов: графического окна **Figure No. 1**, осей и линии — графика синуса. Манипулирование свойствами объектов в MatLab производится функциями `get` и `set`. Функция `get` предназначена для получения значений свойств, а `set` для установки новых значений. При этом функциям `get` и `set` следует указать, с каким из существующих объектов ведется работа. Имеется три стандартных функции, которые могут быть использованы в качестве входного аргумента `get` и `set`:

- `gcf` — текущее графическое окно;
- `gca` — текущие оси;
- `gco` — текущий графический объект.

Обратите внимание, что использование `gcf`, `gca`, `gco` осуществляет доступ к свойствам *текущего* окна, осей или объекта. В данном случае открыто только одно графическое окно, оно и является текущим. Единственные оси в текущем графическом окне также являются текущими. Про использование `gco` для определения текущего объекта сказано ниже.

Свойства осей

Получите свойства осей, которые содержат графики функций, построенных в предыдущем разделе, для чего выполните команду:

```
>> get(gca)
```

В командное окно выводится таблица свойств и их значений. Свойств достаточно много, среди них есть очевидные, а назначение некоторых на пер-

вый взгляд кажется непонятным. В табл. 9.1 и 9.2 содержатся простейшие свойства осей, которые обычно применяются при создании приложений. Функция `get` допускает обращение к ней с двумя аргументами, вторым аргументом является название свойства, значение которого требуется получить. Например команда

```
>> fn = get(gca, 'FontName')
```

записывает название шрифта, используемого в текущих осях, в строковую переменную `fn` и выводит ее значение на экран:

```
fn =
Helvetica
```

Таблица 9.1. *Свойства, отвечающие за общий вид осей*

Название свойства	Описание	Значения
`Box`	Заключение осей в прямоугольную рамку	`on` или `off`
`Color`	Цвет фона осей	Вектор из трех элементов, задающий цвет в формате RGB, например [1 1 1], или один из определенных цветов: `r`, `g` и т. д. *(см. приложение 1)*
`FontAngle`	Наклон шрифта разметки осей	`normal` или `italic`
`FontName`	Название шрифта	Строка с названием шрифта, например `Courier`
`FontSize`	Размер шрифта	Целое число
`FontWeight`	Толщина шрифта	`normal`, `bold`, `light`, или `demi`
`GridLineStyle`	Стиль линий сетки	`-`, `--`, `:`, `-.` или `none`
`LineWidth`	Ширина линий	Значение в пунктах (1 пункт = 1/72 дюйма)
`Visible`	Отображение осей	`on` (по умолчанию оси видны), `off`

Таблица 9.2. *Свойства каждой из осей (на примере оси X)*

Название свойства	Описание	Значения
`XColor`	Цвет оси	Вектор из трех элементов, задающий цвет в формате RGB, например [1 1 1], или один из определенных цветов: `r`, `g` и т. д. *(см. приложение 1)*

Таблица 9.2 *(окончание)*

Название свойства	Описание	Значения
`XDir`	Направление оси	`normal` или `reverse` (обратное)
`XGrid`	Сетка, перпендикулярная оси	`on` или `off`
`XAxisLocation`	Расположение оси	`top` или `bottom` (`right` или `left` для оси Y)
`XLim`	Пределы изменения переменной	Вектор из двух компонентов, равных пределам изменения переменной, например [–1.5 2.3]
`XScale`	Масштаб оси	`linear` или `log`
`XTick`	Координаты разметки оси	Вектор с координатами разметки, например [0 1 3 5]
`XTickLabel`	Разметка оси	Вектор ячеек с названиями разметки (число ячеек равно длине вектора с координатами разметки), например `{'zero';'one';'three';'five'}`

Команда `set` позволяет установить каждому свойству текущих осей любое из допустимых значений. Первым аргументом задается `gca`, а вторым и третьим — пара: `Свойство, значение`. Приведите оси с графиком синуса (см. предыдущий раздел) к виду, указанному на рис. 9.1.

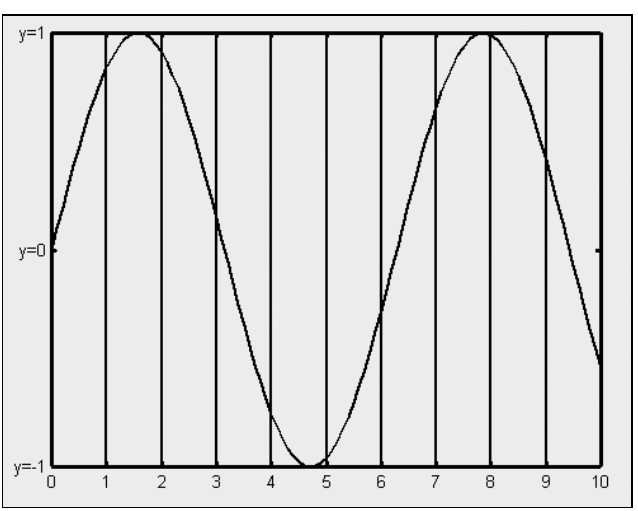


Рис. 9.1. Изменение свойств осей

Используйте следующие команды для заключения осей в рамку, задания толщины линий осей, координат и обозначений разметки, нанесения линий сетки и установки цвета (цвет фона осей совпадает с цветом графического окна):

```
>> set(gca, 'Box', 'on')
>> set(gca, 'LineWidth', 2)
>> set(gca, 'YTick', [-1 0 1])
>> set(gca, 'YTickLabel', {'y=-1';'y=0';'y=1'})
>> set(gca, 'XGrid', 'on')
>> set(gca, 'GridLines', '-')
>> set(gca, 'Color', [0.8 0.8 0.8])
```

Часть названий свойств заканчивается словом `Mode`, например `YTickMode`. Такие свойства могут иметь только два значения — `'auto'` (устанавливаемое по умолчанию) или `'manual'`, причем `'auto'` обеспечивает автоматический подбор значения соответствующего свойства, в данном случае `YTick`. Задание вектора в `YTick` приводит к смене значения `YTickMode` с `'auto'` на `'manual'`. Всегда можно отменить проделанные изменения свойства `YTick`, установив `YTickMode` в `'auto'`. Вышесказанное справедливо для всех свойств, имеющих сопутствующее свойство, которое заканчивается на `Mode`.

Команда `set` позволяет получить доступ к свойствам из собственной файл-программы или файл-функции и изменить вид графика по своему усмотрению. Названия свойств и их значения практически не отличаются от названий соответствующих элементов управления диалогового окна редактора графиков (в версии 6.*x*) или редактора свойств (в версии 5.3), описанных в *главе 4*. Например, свойства, начинающиеся со слова `Camera`, отвечают за вид трехмерных графиков.

См. разд. "Управление камерой" главы 4.

Среди названий свойств осей, полученных при помощи `get(gca)`, есть `'Title'`, `'Xlabel'`, `'Ylabel'`, `'Zlabel'`. Обратите внимание, что их значения *не являются* текстовыми строками. Они содержат указатели на соответствующие текстовые объекты, об использовании указателей подробно написано ниже.

Свойства линий и поверхностей

Стандартная функция `gca`, задаваемая в качестве аргумента `set` и `get`, позволяет получить доступ к свойствам текущих осей. Для обращения к текущей линии или поверхности на графике в MatLab нет специальной встроенной функции. Сделайте линию *текущим объектом* при помощи щелчка мыши по ней в графическом окне, затем выведите таблицу свойств и их значений в командное окно, используя `gco`:

```
>> get(gco)
```

Многие из этих свойств вы изменяли в редакторе при прочтении *главы 4*. Табл. 9.3 содержит наиболее часто употребляемые свойства линии.

Таблица 9.3. *Свойства линий*

Название свойства	Описание	Значения
`Color`	Цвет	Вектор из трех элементов, задающий цвет в формате RGB, например [1 1 1], или один из определенных цветов: `r`, `g` и т. д. *(см. приложение 1)*
`LineStyle`	Стиль линии графика	`-`, `--`, `:`, `-.` или `none`
`LineWidth`	Толщина линии	Значение в пунктах
`Marker`	Тип маркера	Одно из стандартных обозначений, например `o`, `s` (см. табл. 3.1)
`MarkerEdgeColor`	Цвет границы маркера	Такие же, как у `Color`
`MarkerFaceColor`	Цвет маркера	Такие же, как у `Color`
`MarkerSize`	Размер маркера	Значение в пунктах

Приведите график синуса (см. рис. 9.1) к виду, изображенному на рис. 9.2, используя функции `set` и `gco`. Очевидно, что требуется выполнить следующие команды:

```
>> set(gco, 'Color', 'k')
>> set(gco, 'LineWidth', 2)
>> set(gco, 'Marker', 'o')
>> set(gco, 'MarkerFaceColor', 'w')
>> set(gco, 'MarkerSize', 8)
```

Свойства поверхностей изменяются аналогичным образом. Создайте график поверхности какой-либо функции двух переменных, сделайте поверхность текущей при помощи щелчка мыши и выведите таблицу со свойствами и их значениями в командное окно. Изучите самостоятельно возможные свойства.

Обратитесь к разд. "Свойства линий и поверхностей" главы 4.

Функция `gco` указывает на текущий объект, выбранный пользователем щелчком мыши. Данным объектом может быть не только линия или поверхность, но и оси, и графическое окно. Убедитесь в этом, щелкая по объектам и выводя их свойства при помощи `get` и `gco`.

Программирование собственных алгоритмов, связанных с визуализацией данных, как правило, предполагает наличие нескольких графиков, т. е. имеется ряд объектов (графические окна, оси, линии, поверхности), свойства которых необходимо изменять в ходе выполнения программы. Объект может быть текущим в данный момент, если он только что был создан или пользователь выбрал его щелчком мыши. Однако часто в ходе выполнения программы необходимо установить некоторые свойства нужного объекта. Эта задача легко решается при помощи указателей.

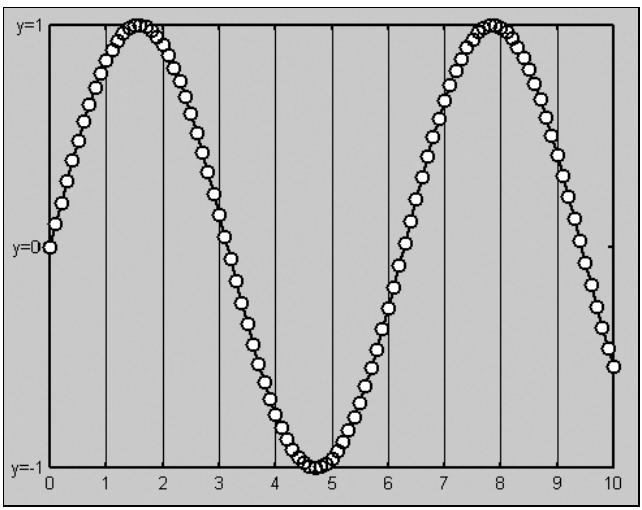


Рис. 9.2. Изменение свойств линии

Указатели на объекты

Создание любого графического объекта в MatLab сопровождается появлением числового *указателя* на него, таким образом, каждый объект уникальным образом идентифицируется в среде MatLab. Функции `gcf`, `gca` и `gco` как раз возвращают указатели на текущее окно, оси и объект. Целесообразнее всего при создании графических объектов записывать их указатели в переменные, которые будут использоваться впоследствии для обращения к нужным объектам. Вызов функций `figure`, `axes`, `plot`, `mesh`, `surfl` и т. д. с выходным аргументом приводит к присвоению ему указателя на соответствующее графическое окно, оси, линию графика или поверхность. Причем, если `plot` осуществляет построение нескольких линий (задано несколько пар векторов значений аргумента и функции), то выходной аргумент является вектором с элементами — указателями на линии графика.

Изменение свойств линий и осей

Оформите последовательность команд для построения графиков двух функций, приведенную в листинге 9.1, в виде файл-процедуры в M-файле. Названия переменных, содержащих указатели, начинаются с символа `H`.

Листинг 9.1. Сохранение указателей на графические объекты

```
% Формирование векторов со значениями аргумента и функций
t = [0:0.1:7];
x = sin(t);
y = cos(t);
% Создание графического окна и запись указателя на него в HFig
HFig = figure;
% Создание осей в текущем графическом окне
% и запись указателя на них в HAx
HAx = axes;
% Построение линий графиков на текущих осях и запись
% указателей на линии в вектор HLines
% HLines(1) содержит указатель на первую линию (sin(t))
% HLines(2) содержит указатель на вторую линию (cos(t))
HLines = plot(t,x,t,y);
```

Операторы приводят к появлению графического окна с графиками двух функций — синуса и косинуса, изображенного на рис. 9.3. На первый взгляд, тот же результат получается и при использовании `plot(t,x,t,y)` без предварительного создания окна, осей и получения указателей.

Предположим, однако, что после команд, содержащихся в листинге 9.1, в программе расположен блок операторов, которые производят вывод некоторых графиков на новые оси других графических окон. Может возникнуть необходимость вернуться к графическому окну с графиками синуса и косинуса и внести некоторые изменения в вид графиков, например, нанести сетку вдоль оси *y*, изменить стиль линий, т. е. установить новые свойства осей и линий. Указатели, записанные в переменные `HAx` и `HLines`, позволяют легко добиться желаемого результата, приведенного на рис. 9.4.

Для этого следует использовать функцию `set` с указателем на объект и парой `'Свойство', 'значение'` (см. листинг 9.2).

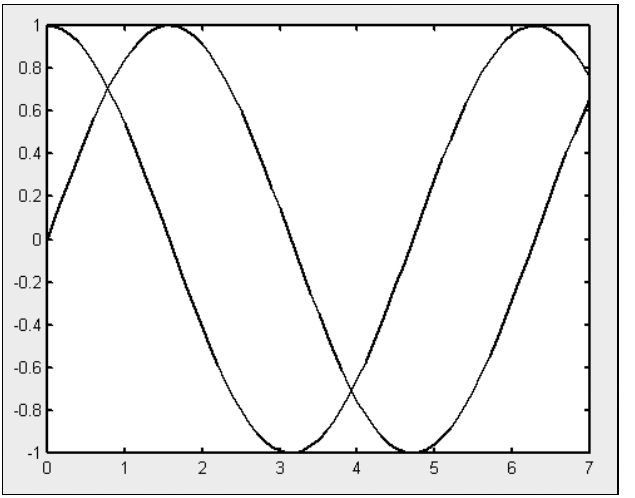


Рис. 9.3. Исходный вид графиков

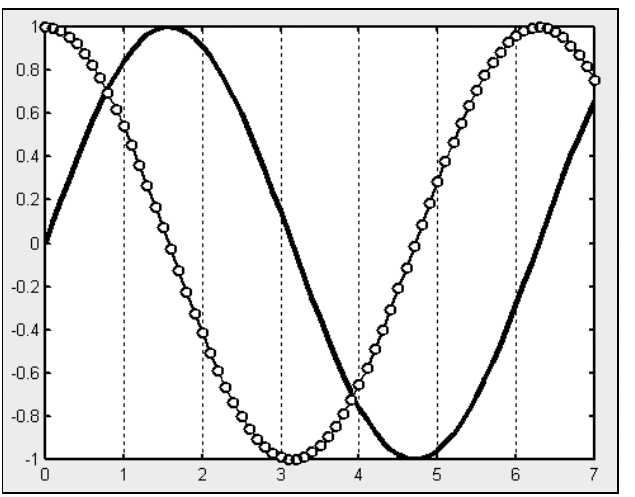


Рис. 9.4. Измененные графики (`set` и указатели на объекты)

Листинг 9.2. Изменение свойств линий и осей при помощи указателей

```
% Установка свойства XGrid осей с указателем HAx в 'on'
set(HAx,'XGrid','on')
% Задание цвета первой линии (графика синуса) с указателем HLines(1)
set(HLines(1), 'Color', 'k')
```

```
% Задание толщины первой линии (графика синуса) с указателем HLines(1)
set(HLines(1), 'LineWidth', 3)
% Задание цвета второй линии (графика косинуса) с указателем HLines(1)
set(HLines(2), 'Color', 'k')
% Задание типа маркера второй линии (графика косинуса)
% с указателем HLines(2)
set(HLines(2), 'Marker', 'o')
% Задание цвета маркеров второй линии (графика косинуса)
% с указателем HLines(2)
set(HLines(2), 'MarkerFaceColor', 'w')
% Задание цвета границ маркеров второй линии (графика косинуса)
% с указателем HLines(2)
set(HLines(2), 'MarkerEdgeColor', 'k')
```

Примененный подход позволяет получить в дальнейшем свойства любого из созданных графических объектов и делать нужный объект текущим для дополнительных изменений, например добавления графиков на оси.

Добавление линий графиков

Команда `plot` осуществляет вывод графика на текущие оси, поэтому следует сначала сделать требуемые оси текущими, задав аргументом функции `axes` указатель на них, выполнить `hold on` (для добавления линии), а затем использовать `plot`, желательно с выходным аргументом, для сохранения указателя на новую линию и установить ее свойства при помощи `set`. Добавьте, например, на оси окна, приведенного на рис. 9.4, график `log(1+t)`, изобразив его черной пунктирной линией толщиной в три пункта так, как показано на рис. 9.5.

Очевидно, что файл-программу, осуществляющую построение графиков синуса и косинуса и изменение их свойств (см. листинги 9.1 и 9.2), следует дополнить последовательностью команд, указанной в листинге 9.3.

Листинг 9.3. Добавление линии графика на заданные оси

```
% Заполнение вектора значений функции
z = log(t+1);
% Установка текущих осей (с графиками синуса и косинуса)
axes(HAx);
% Команда hold on нужна для того, чтобы plot добавила линию графика,
% а не вывела график в отдельном графическом окне
hold on
```

```
% Добавление линии графика log(t+1) и сохранение указателя
% на нее в переменной HLog
HLog = plot(t,z);
% Установка цвета линии графика log(t+1)
set(HLog, 'Color', 'k');
% Установка стиля линии графика log(t+1)
set(HLog, 'LineStyle', '--');
% Установка толщины линии графика log(t+1)
set(HLog, 'LineWidth', 3);
```

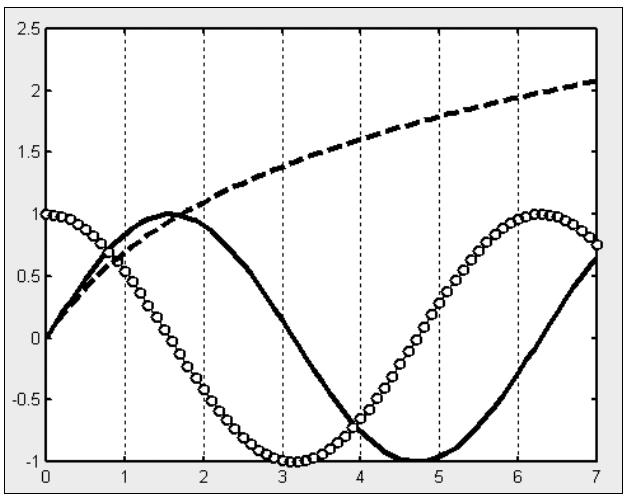


Рис. 9.5. Добавление линии графика на заданные оси

Удаление и очистка объектов

Идентификация объекта указателем позволяет не только устанавливать его свойства, но и удалять его командой `delete`, входным аргументом которой является соответствующий указатель. Например, команда

```
delete(HLines(1))
```

убирает линию графика синуса с осей графического окна, изображенного на рис. 9.5. Следует иметь в виду, что удаление осей повлечет исчезновение всех графических объектов (линий и поверхностей), принадлежащих данным осям. Аналогично, удаление графического окна приводит к тому, что пропадают все объекты, размещенные в нем. Альтернативным способом удаления линий и поверхностей является очистка осей. Следует сделать оси

текущими, а затем использовать команду очистки текущих осей `cla`, например следующие операторы очищают оси с указателем `Hax`:

```
axes(HAx)
cla
```

Очистка текущего графического окна производится командой `clf`.

Получение информации о свойствах

Графические объекты имеют достаточно большой набор свойств, каждое со своими допустимыми значениями. Полная информация о свойствах графических объектов и их значениях содержится в справочной системе по MatLab. В версии 5.3 следует перейти по ссылке **Handle Graphics Properties** в гипертекстовой справочной системе Help Desk (HTML). При работе в версии 6.*x* необходимо выбрать в содержании (в левой колонке окна **Help** раздел MATLAB и перейти по ссылке **Handle Graphics Property Browser** в правом окне. Имеется и другой способ, предоставляемый функциями `get` и `set`.

Функция `get`, как было описано выше, позволяет вывести установленные значения всех свойств или только одного свойства. Список возможных значений конкретного свойства может быть получен при помощи `set`, первый аргумент которой является указателем на графический объект, а второй — названием нужного свойства данного объекта. Например, все обозначения для маркеров, определенные в MatLab, выводятся в командное окно следующим образом:

```
>> set(HLog, 'Marker')
[ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram |
hexagram | {none} ]
```

Вызов `set` с одним аргументом — указателем на объект — приводит к появлению таблицы с названиями свойств и всеми допустимыми значениями.

Использование указателей, примеры

Наличие большого числа графиков требует умения оперировать их свойствами. Эффективной оказывается запись указателей на однотипные графические элементы в числовой массив. Разберем обработку графических объектов на следующем примере. Требуется написать программу, предназначенную для исследования различных математических функций с интерфейсом из командной строки. Пользователь задает в командной строке формулы для вычисления функций одной переменной `x` (в соответствии с правилами MatLab), а программа выводит графики функций в графическое окно, автоматически выделяя маркерами график последней построенной функции, а жирной линией — график функции, имеющей максимальное значение сре-

ди всех введенных функций. По завершении работы (пользователь ввел `end` на запрос программы) графическое окно закрывается.

Предусмотрите возможность очистки осей, которая происходит, если пользователь вводит `new`. Оформите программу в виде файл-функции с входными аргументами — границами области определения функций. Вначале постройте графическое окно и оси в нем, сохраните указатели на данные объекты. Запрос на ввод организуйте в цикле `while` при помощи команды `input`. Обработку ввода пользователя произведите с использованием `switch`. Запоминайте максимальное значение каждой функции, добавляя его в некоторый массив. Максимальное значение проще всего найти, применяя `max` к вектору значений функции. Указатели на линии графиков записывайте в массив. Листинг 9.4 содержит текст файл-функции `maxfun`, в которой реализован вышеописанный алгоритм.

Листинг 9.4. Файл-функция `maxfun`, использующая указатели на объекты

```
function maxfun(a,b)
% Файл-функция с интерфейсом из командной строки для
% исследования функций на максимум на отрезке [a,b]
% Использование: maxfun(a,b)

% Создание графического окна (оно становится текущим)
HF = figure;
% Создание осей в текущем графическом окне
HAx = axes;
% Задание вектора значений аргумента
x = [a:(b-a)/30:b];
str = ''; % инициализация строки запроса ввода пользователя
funcount = 0; % инициализация счетчика введенных функций
maximums = [];% инициализация массива с максимумами функций
hFuns = []; % инициализация массива указателей на линии графиков
% Обработка ввода пользователя в бесконечном цикле
while 1
   str = input('Введите функцию, или new, или end: ', 's');
   switch str
   case 'new'  % Пользователь задал очистку осей
      axes(HAx); % оси с указателем HAx стали текущими
      cla % очистка текущих осей
   case('end') % Пользователь завершает работу с программой
      break % выход из цикла
```

```
otherwise  % Пользователь ввел новую функцию
   funcount = funcount + 1; % увеличение счетчика функций
   % Формирование команды для вычисления массива значений
   eval(strcat('y =', str, ';'));
   % Оси HAx должны быть текущими для вывода графика
   axes(HAx)
   hold on % график следует добавить на оси
   % Построение графика функции, введенной пользователем,
   % и добавление указателя на него в массив указателей
   HFuns(funcount) = plot(x,y)
   % Установка требуемых свойств линии графика новой функции
   set(HFuns(funcount), 'Marker', 'o')
   set(HFuns(funcount), 'MarkerEdgeColor', 'k')
   set(HFuns(funcount), 'MarkerFaceColor', 'w')
   set(HFuns(funcount), 'LineWidth', 1)
   set(HFuns(funcount), 'Color', 'k')
   % Вычисление максимума новой функции и добавление
   % его значения в массив, содержащий максимумы
   maximums(funcount) = max(y);
   if funcount > 1 % пользователь ввел две или более функций
      % Удаление маркеров с графика предыдущей функции
      set(HFuns(funcount-1), 'Marker', 'none')
      % Поиск функции с максимальным значением среди ранее
      % введенных, в oldmax записывается значение,
      % а в N — номер требуемой функции
      [oldmax, N] = max(maximums(1:funcount-1));
      % График найденной функции должен отображаться жирной линией
      set(HFuns(N), 'LineWidth', 3)
      % Сравнение максимального значения новой функции с
      % максимальным из значений предыдущих функций
      if maximums(funcount) > oldmax
         % Новая функция принимает самое большое значение среди
         % всех функций, введенных пользователем, поэтому
         % график последней введенной функции должен рисоваться
         % жирной линией
         set(HFuns(funcount), 'LineWidth', 3)
         % Линия графика функции, которая ранее имела максимальное
         % значение, теперь не должна быть жирной
```

```
            set(HFuns(N), 'LineWidth', 1)
          end
       end
    end
end
% Выход из цикла while производится, когда пользователь ввел end
delete(HF) % удаление графического окна с экрана
```

Задание свойств в аргументах графических функций

Применение указателей на графические объекты полезно при изменении свойств объекта в ходе работы программы. Создание объекта с определенными постоянными свойствами может быть осуществлено при помощи высокоуровневых функций. В качестве дополнительных входных аргументов задаются пары `Свойство, значение`. Например, последовательность команд, приведенная в листинге 9.5, обеспечивает получение графика, изображенного на рис. 9.4, который был получен установкой свойств линий и осей функцией `set` с указателями на данные графические объекты (см. листинги 9.1 и 9.2).

Листинг 9.5. Задание свойств в аргументах графических функций

```
t = [0:0.1:7];
x = sin(t);
y = cos(t);
axes('XGrid','on')
hold on
plot(t,x,'Color','k','LineWidth',3)
plot(t,y,'Color','k','Marker','o','MarkerFaceColor','w',...
   'MarkerFaceColor','w','MarkerEdgeColor','k');
```

Обратите внимание, что подобное использование свойств возможно только в том случае, если `plot` строит одну линию графика функции, т. е. задана одна пара векторов со значениями аргумента и функции.

Разумеется, если сохранить указатели на создаваемые вышеописанным способом линии и оси, то их можно использовать впоследствии для изменения свойств этих объектов. Аналогичным образом задаются свойства графического окна (в аргументах `figure`) и поверхностей (в аргументах `surf`). Следующий раздел посвящен созданию графических окон и осей с заданными размерами и положением на экране.

Расположение графических окон и осей

Вывод результатов в графической форме требует предварительного создания графических окон и осей с заданными размерами и положением. Хорошо написанная программа не должна быть привязана к какому-либо разрешению, установленному на мониторе. Следующие разделы посвящены описанию тех свойств графических окон и осей, которые оказываются полезными при организации вывода графических результатов в различные окна и оси.

Управление положением графических окон

Свойство `Position` графического окна отвечает за положение окна на экране и его размер. Значением `Position` является вектор из четырех элементов, имеющий следующий формат:

```
[left bottom width height]
```

где `left` задает расстояние от левого края монитора до левого края области окна без учета толщины рамки, `bottom` означает расстояние от нижнего края монитора до нижнего края области окна, также без учета толщины рамки, а `width` и `heigh` определяют, соответственно, ширину и высоту области окна.

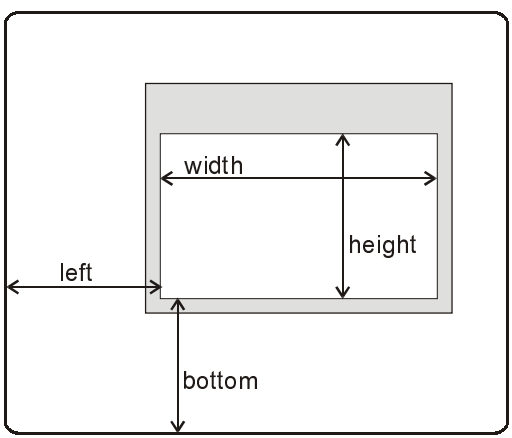


Рис. 9.6. Положение графического окна на экране

На рис. 9.6 приведена схема расположения графического окна на экране монитора, рабочая область окна изображена белым цветом. Графическое окно белого цвета позволяет убедиться в том, что рамки действительно присутствуют. Толщина левой и нижней рамок составляет пять пикселов.

Представление графических результатов в собственных программах часто не требует отображения в графическом окне его меню. Установка свойства

MenuBar в 'none' приводит к скрытию меню и, следовательно, увеличению области окна. Создайте графическое окно без меню командой

```
figure('Color', 'w', 'MenuBar', 'none')
```

и получите его размеры, указав входными аргументами get указатель на текущее окно (используйте gcf) и 'Position':

```
get(gcf, 'Position')
ans =
   232   258   560   420
```

Единицами измерения по умолчанию являются пикселы. Про установку других единиц написано ниже. Максимизируйте теперь графическое окно при помощи кнопки, расположенной в верхнем правом углу окна (рамки окна при этом не отображаются на экране) и выведите значение свойства Position еще раз (числа в данном примере получены на мониторе с разрешением 1024×768; если ваш монитор настроен на другой режим, то результаты могут отличаться):

```
>> get(gcf, 'Position')
ans =
           1           1        1024         749
```

Итак, сейчас ширина рабочей области совпадает с шириной экрана, а 768−−749=19 пикселов отводится под заголовок графического окна **Figure No. 1**, помещенный вверху окна на синем фоне. Несложные подсчеты позволяют найти требуемые ширину и высоту графических окон, располагающихся на экране заданным образом. Однако для того, чтобы обеспечить правильную работу программы вне зависимости от установленного разрешения монитора, следует предварительно получить его. Разрешение монитора хранится в виде вектора из четырех элементов в свойстве ScreenSize объекта с указателем, равным нулю (объекта Root).

Положение объекта Root *в иерархии объектов MatLab и его свойства приведены в главе 14.*

```
>> get(0,'ScreenSize')
ans =
           1           1        1024         768
```

Единицы измерения размера экрана зависят от значения свойства Units объекта Root, по умолчанию установлены пикселы. Очевидно, что графическое окно без меню и рамки, занимающее весь экран, появляется в результате выполнения команд, приведенных в листинге 9.6.

Листинг 9.6. Расположение графического окна на весь экран

```
% Нахождение размеров экрана
SCRsize = get(0, 'ScreenSize')
% Область окна начинается от левого и нижнего края экрана,
% рамка не нужна
left = SCRsize(1);
bottom = SCRsize(2);
% Ширина области окна равна ширине экрана
width = SCRsize(3);
% Высота окна вычисляется с учетом ширины заголовка окна
height = SCRsize(4)-19;
% Создание окна без меню, рабочая область и заголовок растянуты
% на весь экран, границы не отображаются
figure('Position', [left bottom width height], 'Menu', 'none')
```

Создайте и расположите на экране два графических окна без строки меню, делящих экран на равные части по вертикали. Оформите результат в виде файл-функции, возвращающей вектор указателей на графические окна. При вычислении размеров и положения окон учтите ширину рамки и заголовка окна.

Листинг 9.7. Файл-функция `fig2` для создания двух графических окон

```
function HFigs = fig2
% Создание двух графических окон равной высоты, делящих экран
% на две части по вертикали. Возвращает вектор указателей на окна
% Использование HFigs = fig2
% Нахождение размеров экрана
SCRsize = get(0, 'ScreenSize');
% Выделение ширины и высоты экрана
SCRwidth = SCRsize(3);
SCRheight = SCRsize(4);
% Ширина, высота и отступ слева одинаковы для обоих окон
width = SCRwidth-5-3;
height = (SCRheight-19-5-3-19-5-3)/2;
left = 5;
```

```
% Отступ снизу для нижнего графического окна равен 5 пикселам
bottom2 = 5;
% Вычисление отступа снизу для верхнего окна с учетом толщины
% рамок и заголовка нижнего окна и толщины верхней рамки нижнего
% окна
bottom1 = 5+height+19+5+3;
% Создание окон без меню, рамки отображаются
HFigs(1) = figure('Position', [left bottom1 width height],...
   'Menu', 'none', 'Color', 'w');
HFigs(2) = figure('Position', [left bottom2 width height],...
   'Menu', 'none', 'Color', 'w');
```

Аналогичным образом размещается на экране любое число графических окон с заданными размерами и положением. Следующим этапом является создание подходящих осей в пределах графических окон.

Управление положением осей

Оси графиков являются объектами, принадлежащими графическим окнам, как показано на рис. 4.5. Каждое графическое окно может содержать одну или несколько осей. Команда `subplot` предлагает самый простой способ организации осей — в виде матрицы.

Применение `subplot` *описано в разд. "Несколько графиков в одном графическом окне" главы 3.*

Более универсальным подходом является создание осей при помощи функции `axes` с указанием их размеров и положения. Создайте новое графическое окно без меню и оси, заключенные в рамку, сохраните указатели на данные объекты и получите значение свойства `Position` осей:

```
>> HF = figure('Menu', 'none', 'Color', 'w');
>> HAx = axes('Box', 'on');
>> get(HAx, 'Position')
ans =
    0.1300    0.1100    0.7750    0.8150
```

Положение осей задается вектором из четырех элементов

```
[left bottom width height],
```

где `left` равно расстоянию от левой границы рабочей области графического окна до осей, `bottom` — от нижней границы рабочей области графического окна до осей, а `width` и `height`, соответственно, определяют ширину и высоту осей (см. рис. 9.7 и 9.8 для случая двумерных и трехмерных осей).

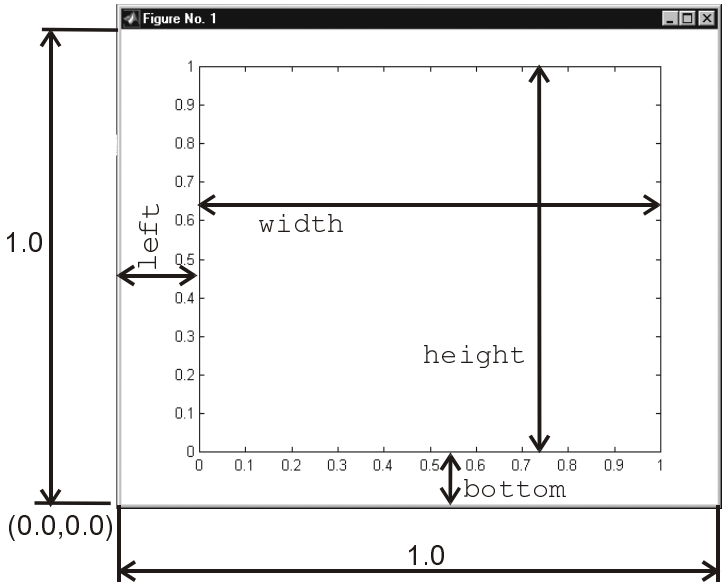


Рис. 9.7. Расположение двумерных осей в графическом окне

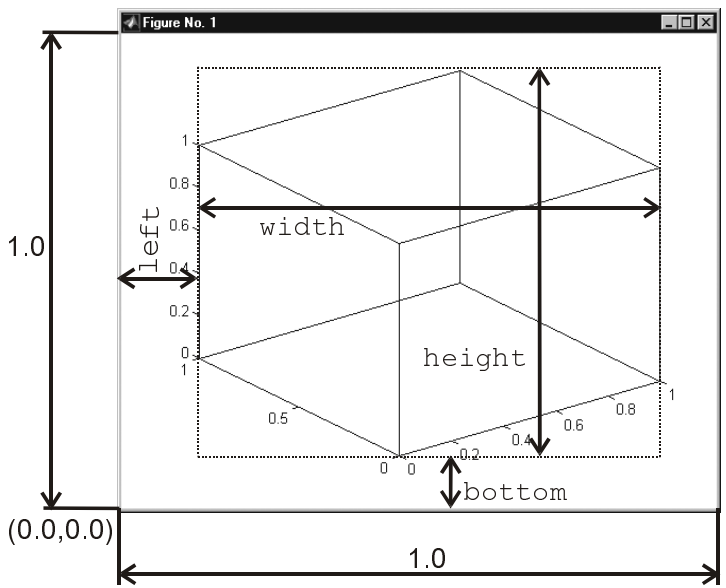


Рис. 9.8. Расположение трехмерных осей в графическом окне

По умолчанию используются нормализованные единицы для измерения расстояний. Ширина и высота рабочей области графического окна прини-

маются равными единице, а начало координат помещается в левый нижний угол области окна. Единицы измерения устанавливаются свойством `Units` осей. Нормализованные единицы `normalized` наиболее удобны при управлении расположением осей. Напишите файл-функцию `axes3`, которая создает графическое окно и три пары осей на нем так, как изображено на рис. 9.9, и возвращает указатели на созданные оси. Входным аргументом `axes3` должен быть указатель на графическое окно.

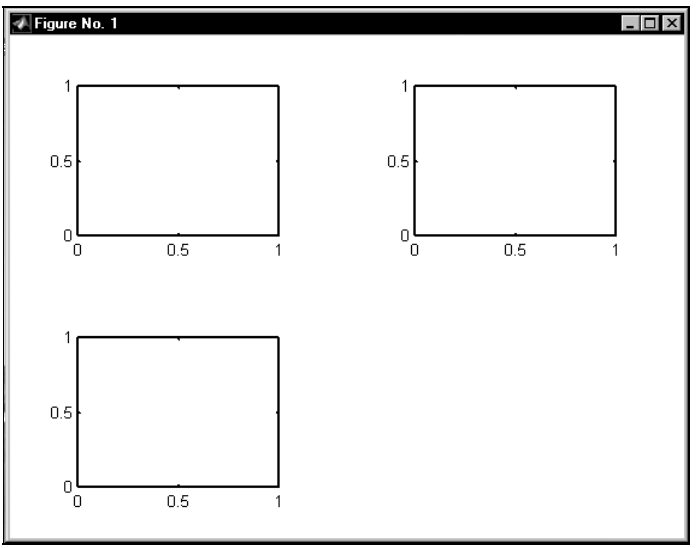


Рис. 9.9. Пример расположения осей

Текст файл-функции `axes3` приведен в листинге 9.8.

Листинг 9.8. Файл-функция `axes3`

```
function HAx = axes3(HF)
% Создание осей в графическом окне с указателем HF.
% Возвращает вектор указателей на оси.
% Использование HAx = axes3(HF)
% Окно с указателем HF становится текущим
figure(HF)
% Создание осей в текущем окне
HAx(1) = axes('Position', [0.1 0.6 0.3 0.3], 'Box', 'on');
HAx(2) = axes('Position', [0.6 0.6 0.3 0.3], 'Box', 'on');
HAx(3) = axes('Position', [0.1 0.1 0.3 0.3], 'Box', 'on');
```

Файл-функции `axes3` и `fig2` (см. листинг 9.7) позволяют легко получить два графических окна, разделяющих экран монитора по вертикали на равные части, каждое из окон содержит по три оси:

```
>> HFigs = fig2;
>> HAxTop = axes3(HFigs(1));
>> HAxBot = axes3(HFigs(2));
```

Пример работы с графикой. Исследование функций

Используйте файл-функцию `axes3` (см. листинг 9.8) при решении следующей задачи. Требуется написать программу для исследования поведения функций вблизи корня и локального минимума. Программа должна выводить в одном графическом окне три графика — график функции на заданном интервале и графики этой функции вблизи локального максимума и корня. Программу оформите в виде файл-функции `zeroandmin`. Имя исследуемой функции (описанной в М-файле или встроенной функции MatLab) и границы интервала задаются во входных аргументах `zeroandmin`. Построение графиков функций осуществите при помощи `fplot`, которая сама подбирает вектор значений аргумента, учитывающий особенности поведения функции.

Применение `fplot` и функций MatLab, предназначенных для нахождения локальных минимумов и нулей, описано в разд. "Исследование функций" главы 6.

Листинг 9.9. Файл-функция `zeroandmin`, предназначенная для исследования функций

```
function zeroandmin(funname, a, b);
% Исследование поведения функций вблизи корня и
% локального минимума на отрезке [a,b]
% Использование zeroandmin(funname, a, b);

% Формирование параметров функций fzero и fminbnd
% Подавление вывода в командное окно информации о ходе вычислений
options = optimset('Display', 'off');
% Поиск нуля функции на [a,b]
zero = fzero(funname, [a b], options);
% Поиск локального минимума функции на [a,b]
minvalue = fminbnd(funname, a, b, options);
% Создание окна
HF = figure('Menu', 'none', 'Color', 'w');
% Использование axes3 для построения трех осей
HAx = axes3(HF);
```

```
% Задание delta для вывода графиков вблизи корня и локального минимума
delta = (b-a)/30;
axes(HAx(1)) % оси с указателем HAx(1) стали текущими
% Вывод графика функции вблизи ее минимума на верхние левые оси
fplot(funname, [minvalue-delta  minvalue+delta])
axes(HAx(2)) % оси с указателем HAx(2) стали текущими
% Вывод графика функции вблизи ее нуля на верхние правые оси
fplot(funname, [zero-delta zero+delta])
axes(HAx(3)) % оси с указателем HAx(3) стали текущими
% Вывод графика функции на [a,b] на нижние оси
fplot(funname, [a b])
```

Теперь исследовать поведение функции вблизи корня и локального минимума и одновременно вывести график всей функции не представляет большого труда, например вызов

```
>> zeroandmin('sin', -pi, pi)
```

приводит к появлению графического окна, изображенного на рис. 9.10, с графиком синуса на отрезке $[-\pi, \pi]$ и графиков, построенных в более крупном масштабе вблизи точек, которые представляют интерес для пользователя.

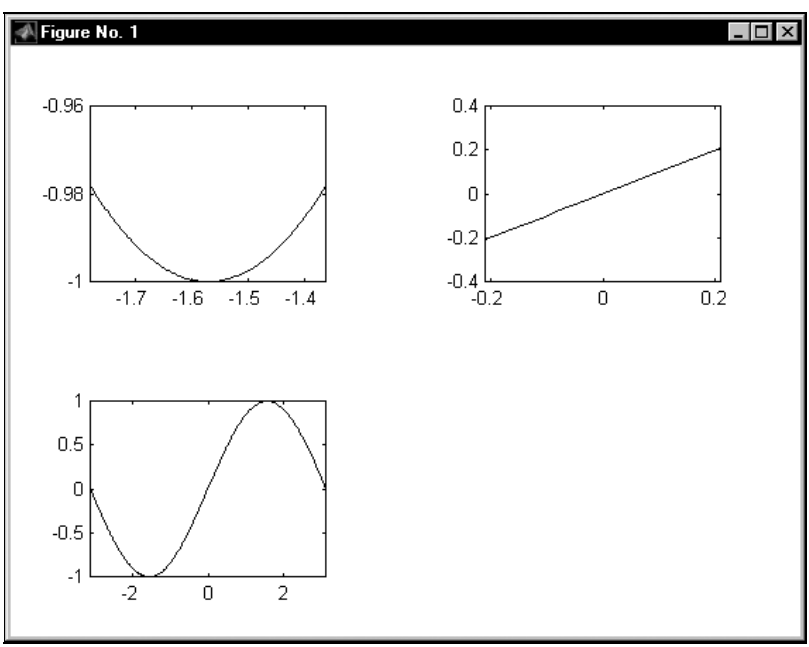


Рис. 9.10. Исследование функции `sin(x)` при помощи `zeroandmin`

Размещение текстовой информации

Результаты, представленные в графической форме, следует снабдить соответствующими пояснениями, которые облегчают чтение графиков. Добавление заголовков на текущие оси осуществляется командой `title` так, как описано *в разд. "Оформление графиков" главы 3*. Разберем более общий подход, позволяющий изменять заголовки в ходе выполнения программы и размещать текст в произвольном месте графического окна.

Текстовые объекты

Команда `title` создает текстовый объект и размещает его вверху текущих осей, выравнивая по центру, причем, `title` возвращает в выходном аргументе указатель на текстовый объект. Следовательно, можно управлять свойствами полученного объекта для достижения желаемого результата. Команда

```
HTxt = title('График функции sin({\itx})')
```

не только добавляет заголовок на текущие оси, но и позволяет впоследствии изменять его при помощи установки свойств текстового объекта с указателем `HTxt`. Например, в любом месте программы можно изменить цвет заголовка, использовав `set`:

```
set(HTxt, 'Color', 'r')
```

Свойства шрифта текстовых объектов с указанием их возможных значений приведены в табл. 9.4.

Таблица 9.4. *Общие свойства шрифта*

Название свойства	Описание	Значения
`Color`	Цвет шрифта	Вектор из трех элементов, задающий цвет в формате RGB, например [1 1 1], или один из определенных цветов: `r`, `g` и т. д. *(см. приложение 1)*
`FontAngle`	Наклон шрифта	`normal` — прямой (по умолчанию), `italic` — курсив
`FontName`	Название шрифта	Строка с названием шрифта, установленного на компьютере, например `Courier`
`FontSize`	Размер шрифта	Целое число
`FontWeight`	Толщина шрифта	`normal` (по умолчанию), `bold`, `light` или `demi`
`Interpreter`	Использование формата TeX для отображения греческих букв и символов	`tex` — использовать формат TeX (по умолчанию), `none` — не использовать

Команда `title` использует функцию `text`, которая создает текстовый объект, принадлежащий текущим осям. Создание и размещение текстовых объектов *в пределах текущих осей* может быть выполнено при помощи функции `text`. Входными аргументами `text` в самом простом случае являются координаты, определяющие положение текста, и строка, а в качестве необязательного выходного аргумента задается указатель на созданный текстовый объект, например

```
HTxt = text(x,y,'точка с координатами (x,y)')
```

Более общее обращение к `text` позволяет управлять свойствами текстового объекта:

```
HTxt = text('Свойство',значение, 'Свойство',значение,...)
```

MatLab предоставляет обширные возможности для управления положением текстовых объектов в пределах осей (табл. 9.5). Следует иметь в виду, что координаты текстового объекта могут быть выражены в различных единицах измерения в зависимости от требуемого результата. Координаты отсчитываются от левого нижнего угла осей и могут быть абсолютные `inches`, `centimeters`, `points`, `pixels` или нормализованные `normalized` (верхний правый угол имеет координаты (1.0, 1.0)). Кроме того, можно использовать *координатную систему осей* для вывода текста в нужную позицию — свойство `Units` текстового объекта должно иметь значение `data` (именно оно стоит по умолчанию). Данный способ является наиболее подходящим, если требуется вывести текст рядом с определенной точкой графика. Команды, приведенные в листинге 9.10, строят график функции и размещают пояснения в заданной точке с координатами `(Xtext,Ytext)`.

Листинг 9.10. Вывод текстового объекта в точку с заданными координатами

```
% Создание графического окна и осей
HF = figure('Menu', 'none', 'Color', 'w');
HAx = axes;
% Генерация векторов значений аргумента и функции
x = [-2:0.01:3];
y = exp(-x.^2);
plot(x,y)
% Задание координат положения текстового объекта
Xtext = 1.17;
Ytext = exp(-Xtext.^2);
```

```
% Создание и отображение текстового объекта
HTxt = text(Xtext, Ytext, '\leftarrow Функция {\ity} = {\ite}^{-x^2}');
```

Результат выполнения файл-программы, приведенной в листинге 9.10, изображен на рис. 9.11.

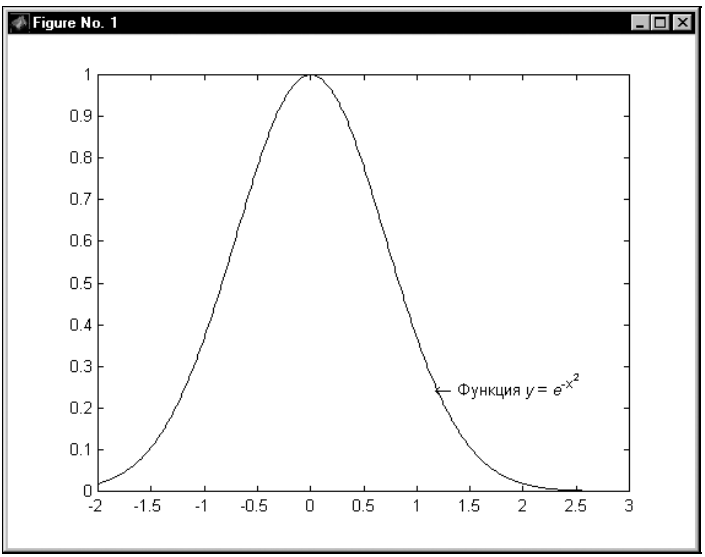


Рис. 9.11. Помещение текста в заданную позицию

Итак, положение текстового объекта задается координатами в любой из вышеперечисленных систем единиц. Свойства `HorizontalAlignment` и `VerticalAlignment` предназначены для указания способа выравнивания области текстового объекта относительно заданного положения.

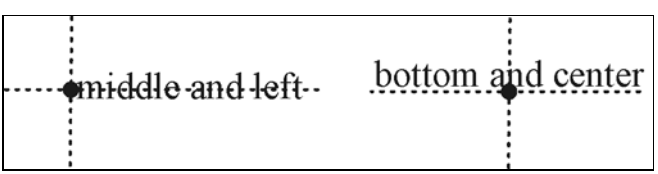


Рис. 9.12. Выравнивание области текстового объекта

Некоторые примеры выравнивания приведены на рис. 9.12. Полную информацию о значениях данных свойств можно получить при помощи справочной системы MatLab.

Таблица 9.5. *Свойства, отвечающие за расположение текста*

Название свойства	Описание	Значения
`HorizontalAlignment`	Выравнивание по горизонтали относительно положения, заданного `Position`	`left` (по умолчанию), `center`, `right`
`Position`	Положение текста в пределах осей	Вектор из двух или трех элементов, определяющий расположение текстового объекта, единицы измерения задаются свойством `Units`
`Rotation`	Угол поворота текста	Значение в градусах
`String`	Текст	Строка в апострофах или строковая переменная или массив ячеек, состоящий из строк, для получения многострочного текста. Может содержать текст в формате TeX
`Units`	Единицы измерения	`inches`, `centimeters`, `points`, `pixels`, `normalized`, `data` (по умолчанию)
`VerticalAlignment`	Выравнивание по вертикали относительно положения, заданного `Position`	`top`, `cap`, `middle` (по умолчанию), `baseline`, `bottom`
`Visible`	Отображение текстового объекта	`on` (по умолчанию), `off`

Усовершенствуйте файл-функцию `zeroandmin` (см. листинг 9.9) так, чтобы в окне с графиком функции указывалось положение корня и локального максимума, а верхние графики в увеличенном масштабе снабдите соответствующими заголовками. Очевидно, что для задания положения текстовых объектов требуется знать не только абсциссу минимума, но и ординату. Используйте обращение к функции `fminbnd`, позволяющее одновременно получить требуемое значение функции в точке локального минимума.

Листинг 9.11. Изменение файл-функции `zeroandmin`

```
...
% Поиск локального минимума функции на [a,b]
```

```
% Получение значения функции и аргумента локального минимума
[minvalue, fmin] = fminbnd(funname, a, b, options);
...
% Заголовки и пояснения
axes(HAx(1)) % левые верхние оси текущие
title('Лок. минимум') % добавление заголовка
axes(HAx(2)) % правые верхние оси текущие
title('Ноль функции') % добавление заголовка
axes(HAx(3)) % нижние оси текущие
% Создание текстового объекта, расположенного в нуле функции
HTxtZer = text('String', '\leftarrow ноль', 'Position' , [zero 0]);
% Создание текстового объекта, расположенного в лок. минимуме функции
HTxtMin = text('String', '\leftarrow лок.мин.',...
   'Position' , [minvalue fmin], 'Rotation', 90);
```

Графики, которые теперь выводит файл-функция `zeroandmin`, более наглядны (рис. 9.13).

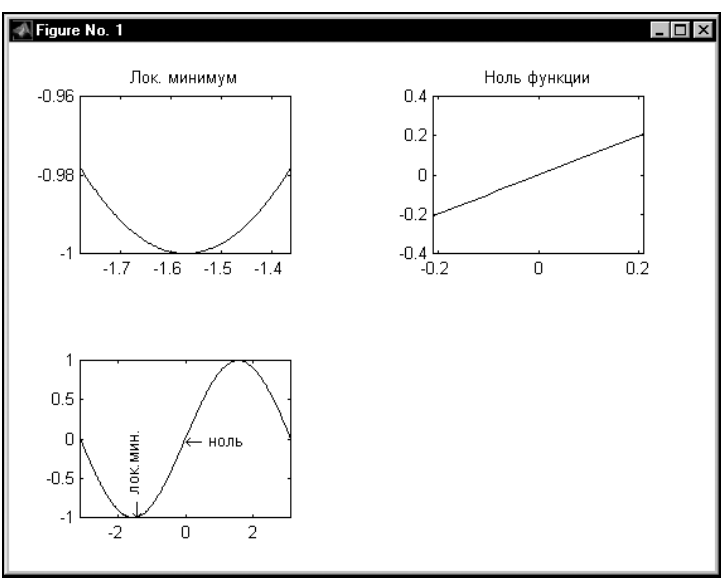


Рис. 9.13. Результат работы усовершенствованной `zeroandmin`

В графическом окне, приведенном на рис. 9.13, имеется свободное место внизу справа, которое можно использовать для вывода численных результатов. Возникает вопрос, как поместить текст в произвольное место графического окна.

Размещение текста в графическом окне

Для вывода текста в любое место графического окна, а не только в пределах осей, используется следующий прием. Создаются оси, равные по размеру графическому окну, их свойство `Visible` устанавливается в `off`. Данные оси существуют как объект, но не отображаются в графическом окне. Текстовый объект, принадлежащий невидимым осям, воспринимается как текст, размещенный в графическом окне. Многострочный текст формируется в массиве ячеек, каждая ячейка которого содержит строку, и указывается в качестве значения свойства `String` текстового объекта.

Завершите работу над файл-функцией `zeroandmin`, снабдив ее операторами, которые выводят в правый нижний угол графического окна значения переменных `fmin`, `minvalue` и `zero` в три строки (см. листинги 9.9 и 9.11). Требуемый для этого блок приведен в листинге 9.12.

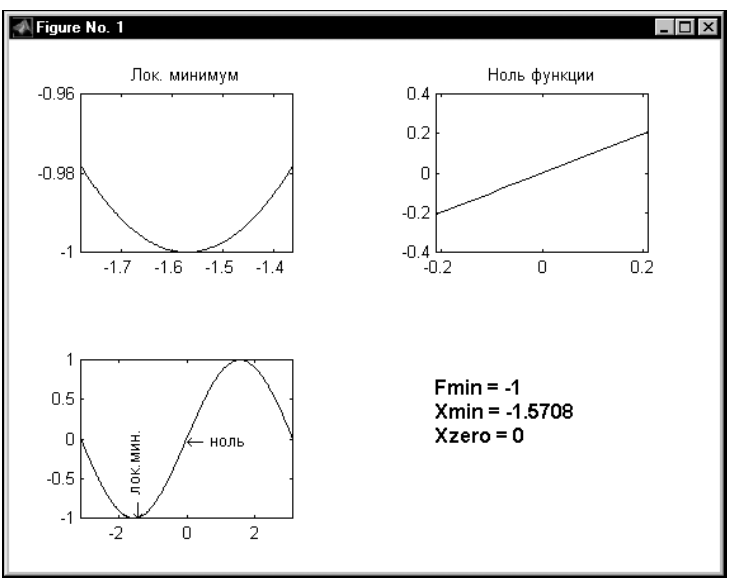


Рис. 9.14. Вывод текста в графическое окно

Листинг 9.12. Вывод текста в графическое окно

```
% Формирование массива ячеек строк
strmas{1} = ['Fmin = ', num2str(fmin)];
strmas{2} = ['Xmin = ', num2str(minvalue)];
strmas{3} = ['Xzero = ', num2str(zero)];
```

```
% Создание вспомогательных невидимых осей
% таких же размеров, как графическое окно
HHelpAx = axes;
set(HHelpAx, 'Position', [0 0 1 1], 'Visible', 'off')
% Вывод текста во вспомогательные оси
HInfo = text(0.6, 0.3, strmas);
% Установка свойств шрифта
set(HInfo, 'FontSize', 12, 'FontWeight', 'bold')
```

Окончательный вариант файл-функции `zeroandmin` позволяет получить не только графическую информацию о поведении функции вблизи интересующих нас точек, но и соответствующие числовые значения (рис. 9.14).



Часть III

Работа в среде GUIDE

Глава 10. Принципы создания приложений с GUI

Глава 11. Конструирование интерфейса в версии 5.3

Глава 12. Конструирование интерфейса в версии 6.x

Глава 13. Диалоговые окна и меню приложения

Глава 14. Программирование событий

Глава 10



Принципы создания приложений с GUI

Приложения MatLab являются графическими окнами, содержащими элементы управления (кнопки, списки, переключатели, флаги, полосы скроллинга, области ввода, меню), а также оси и текстовые области для вывода результатов работы. Создание приложений включает следующие основные этапы — расположение нужных элементов интерфейса в пределах графического окна и определение действий (команд MatLab), которые выполняются при обращении пользователя к данным объектам, например при нажатии кнопки. Процесс работы над приложением допускает постепенное добавление элементов в графическое окно, запуск и тестирование приложения и возврат в режим редактирования. Конечным результатом является программа с графическим интерфейсом пользователя (GUI), содержащаяся в нескольких файлах, запуск которой производится указанием ее имени в командной строке MatLab или в другом приложении.

Принципы создания приложений в версии 5.3

Данный раздел посвящен основным принципам создания приложений в MatLab 5.3. Описана среда GUIDE разработки приложений и основы программирования событий от элементов интерфейса.

Среда разработки приложений GUIDE

Размещение элементов интерфейса в пределах графического окна и задание связанных с ними команд производится в специальной среде программирования, переход в которую осуществляется выполнением `guide` в командной строке. На экране появляется два окна **Guide Control Panel** (панель управления) и **Figure No. 1** (заготовка для окна приложения). Удобно расположить данные окна так, как показано на рис. 10.1.

Панель управления содержит следующие основные элементы:

- инструменты **Giude Tools**;
- список графических окон **Figure List** (в данном случае список состоит из единственного окна **#1**);

- панель **New Object Pallete** для добавления элементов интерфейса (кнопок, списков, осей и т. д.) в окно приложения.

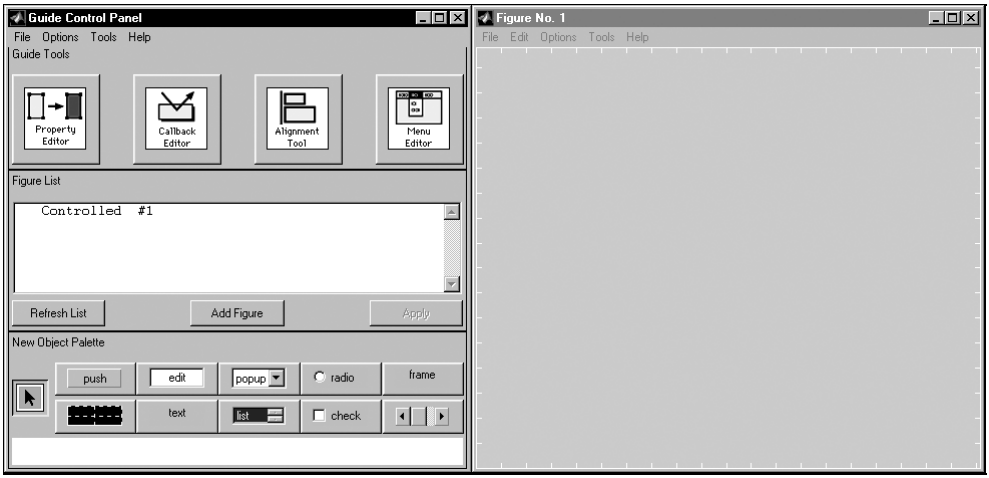


Рис. 10.1. Расположение панели управления и заготовки окна приложения на экране

Панель управления **New Object Pallete**, изображенная на рис. 10.2, позволяет разместить в окне приложения любой из десяти элементов управления. Достаточно нажать соответствующую кнопку и щелкнуть мышью по области окна приложения **Figure No. 1**. В указанное место окна приложения добавится выбранный элемент управления. Можно поступить и по-другому, а именно, нарисовать выбранный элемент в окне приложения, удерживая левую кнопку мыши. Текущий элемент интерфейса выделяется в окне приложения, его можно перемещать по области окна и менять размеры движением мыши, удерживая нажатой левую кнопку. Точная установка размеров и положения размещенных объектов производится при помощи меню **Alignment Tools** (Инструменты выравнивания).

Кнопка со стрелкой на панели **New Object Pallete** служит для перехода в режим выделения элементов окна приложения щелчком мыши. Выделенный текущий элемент можно удалить из окна приложения, нажав <Delete>. Потренируйтесь самостоятельно, добавляя различные элементы управления в окно приложения, например, получите окно, изображенное на рис. 10.3.

Сейчас приложение находится в режиме редактирования, о чем свидетельствует строка **Controlled #1** в списке **Figure List**. Для запуска приложения выполните следующие действия:

1. Выберите соответствующую строку в списке **Figure List** (в данном случае она единственная — **Controlled #1**), строка изменяется на → **Active #1**.

2. Нажмите кнопку **Apply**, расположенную под списком, появляется диалоговое окно **Save Figure**, в котором пока можно нажать кнопку **No**.

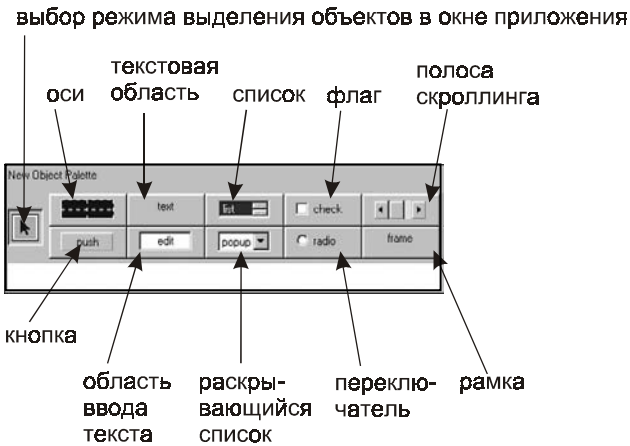


Рис. 10.2. Состав панели **New Object Pallete**

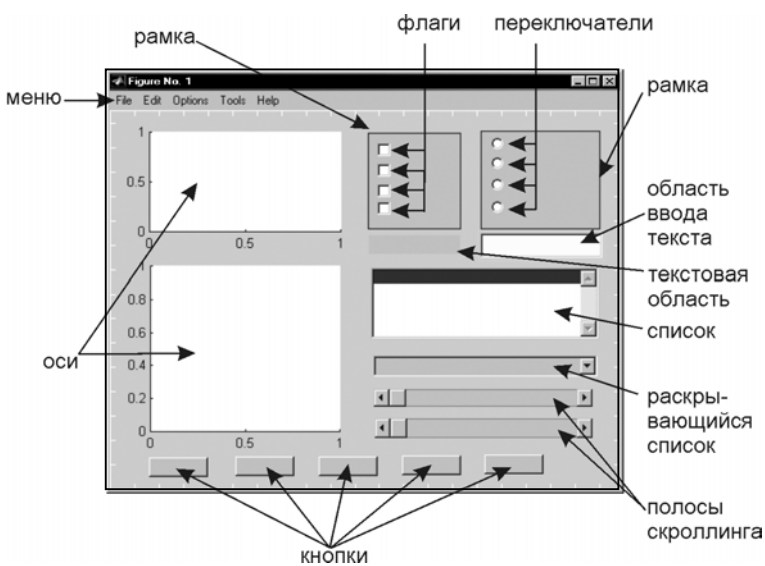


Рис. 10.3. Пример окна приложения с элементами управления

Приложение запущено (исчезли серые отрезки, направленные внутрь графического окна). Теперь можно нажимать на кнопки, перемещать полосы скроллинга, устанавливать флаги и выбирать переключатели, вводить с кла-

виатуры текст в область ввода. Разумеется, ничего полезного только что созданное приложение не делает. Мало разместить элементы интерфейса в окно приложения, следует позаботиться о том, чтобы каждый элемент выполнял нужные функции при обращении к нему пользователя. Например, при нажатии на кнопку производятся вычисления и строятся графики полученных результатов, переключатели позволяют установить цвет линий, полосы скроллинга изменяют пределы осей, в области ввода пользователь указывает некоторые параметры, управляющие ходом вычислений.

Переход в режим редактирования осуществляется так:

1. Выберите соответствующую строку в списке **Figure List** (в данном случае она единственная **Active #1**), строка изменяется на → **Controlled #1**.
2. Нажмите кнопку **Apply**, расположенную под списком.

> **Замечание**
>
> Смена режимов редактирования и запуска может также производиться двойным щелчком мыши по выделенной строке в списке **Figure List**, соответствующей окну приложения.

Приложение находится в режиме редактирования (появились серые отрезки, направленные внутрь графического окна). Можно добавлять и удалять элементы интерфейса, изменять их размеры. Следующий важный этап состоит в наполнении созданной диалоговой оболочки содержанием для получения приложения, выполняющего нужные действия. Дальнейшая работа над приложением, окно которого изображено на рис. 10.3, является слишком сложной для начинающего программиста. Закройте окно приложения при помощи кнопки с крестиком в верхнем правом углу (следует выбрать **No** в появляющемся диалоговом окне **Save Figure**), если в списке **Figure List** осталась строка с номером окна приложения, нажмите кнопку **Refresh List**. Следующий раздел посвящен изучению основ программирования элементов управления. Начните с простого примера, а затем последовательно добавляйте различные элементы интерфейса, выполняющие нужные действия так, как описано в *главах 11* и *13*.

Программирование событий в версии 5.3

Задача состоит в создании приложения, окно которого содержит оси и две кнопки. Пользователь нажимает на кнопку и получает график некоторой функции. Очистка осей производится нажатием на другую кнопку. Работа над приложением происходит в среде GUIDE, переход в которую осуществляется двумя способами:

- командой `guide` из командной строки;
- выбором пункта **Show GUI Layout Tool** меню **File** рабочей среды.

На экране должны присутствовать: панель управления **Guide Control Panel** и заготовка для окна приложения **Figure No. 1** так, как изображено на рис. 10.1. Если окна нет, то его следует добавить, нажав кнопку **Add Figure** на панели управления. Убедитесь в том, что приложение находится в режиме редактирования.

Добавьте оси и кнопку в окно приложения при помощи панели управления (рис. 10.4). Теперь следует указать, что при нажатии на кнопку выводится график функции.

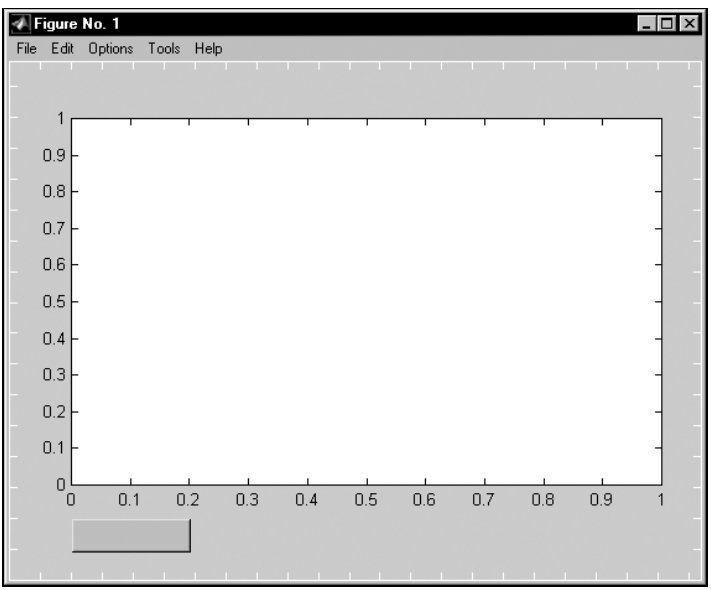


Рис. 10.4. Расположение кнопки и осей в окне приложения

Связь элемента управления с последовательностью команд производится в редакторе вызовов **Guide Callback Editor**. Удобно предварительно сделать нужный элемент текущим в окне приложения (щелкните мышью по только что добавленной кнопке). Активизируйте диалоговое окно редактора вызовов, нажав кнопку **Callback Editor** на панели управления **Guide Control Panel** (рис. 10.5) и установите в нем флаг **Show Object Browser**. Диалоговое окно редактора вызовов **Guide Callback Editor**, изображенное на рис. 10.6, состоит из нескольких частей:

- навигатора объектов с иерархической структурой всех элементов интерфейса, расположенных в окне приложения (они являются объектами MatLab);

- раскрывающегося списка, предназначенного для выбора типа события, в котором в данном случае выбрано **Callback** — пользователь нажал кнопку в окне работающего приложения;
- области для ввода операторов и команд MatLab, которые должны выполняться при нажатии кнопки;
- флага **Show Object Browser**, который отвечает за отображение навигатора объектов в окне **Guide Callback Editor**.

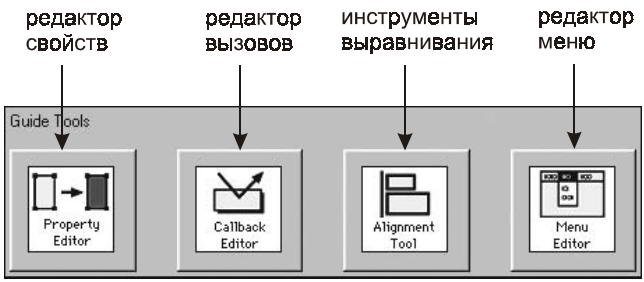


Рис. 10.5. Содержимое панели **Guide Tools**

Навигатор объектов служит для выбора нужного элемента интерфейса, в нем приведены все объекты, имеющиеся в приложении. Убедитесь, что текущим является объект `uicontrol (pushbutton ) Pushbutton1`, т. е. добавленная кнопка (см. рис. 10.6). `Pushbutton1` является именем или тегом кнопки, которое MatLab присвоила ей по умолчанию.

При нажатии пользователем данной кнопки должен строиться график функции. Поместите операторы листинга 10.1 в область для ввода операторов. В дальнейшем будет сказано, что требуется *запрограммировать или обработать событие* `Callback` кнопки `Pushbutton1`.

Листинг 10.1. Обработка события `Callback` кнопки

```
x = [-2:0.2:2];
y = exp(-x.^2);
plot(x,y)
```

Нажмите кнопку **Apply** в окне редактора вызовов. Событие `Callback` кнопки `Pushbutton1` запрограммировано. Редактор вызовов закрывать необязательно. Теперь можно запустить приложение и убедиться, что оно работает.

Запуск приложения из панели управления описан в предыдущем разделе.

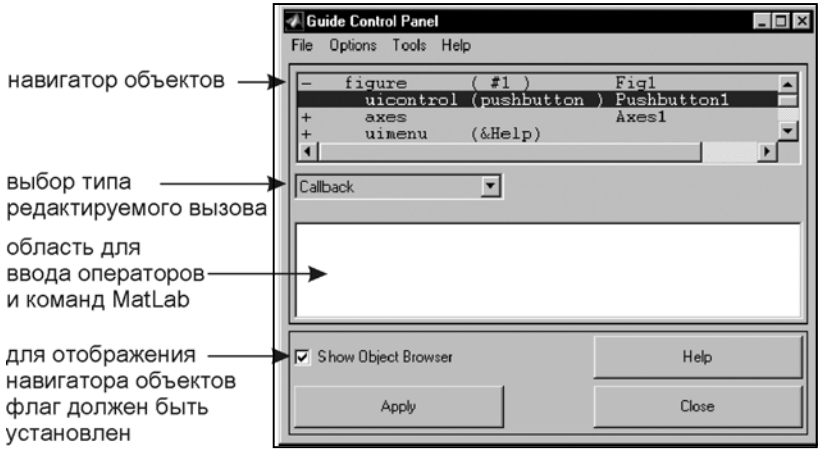


Рис. 10.6. Диалоговое окно **Guide Callback Editor** редактора вызовов

Переход в режим запуска приложения сопровождается появлением диалогового окна **Save Figure**. Имеет смысл сохранить приложение. Нажмите **OK**, в диалоговом окне сохранения создайте для него отдельную папку, например MyFirstGUI, и сохраните приложение в файле mygui.m. Убедитесь, что в раскрывающемся списке **Save as type** выбрано **M-files(*.m)**. Теперь приложение сохранено и запущено. Нажатие на кнопку приводит к построению графика функции на осях в окне приложения (рис. 10.7).

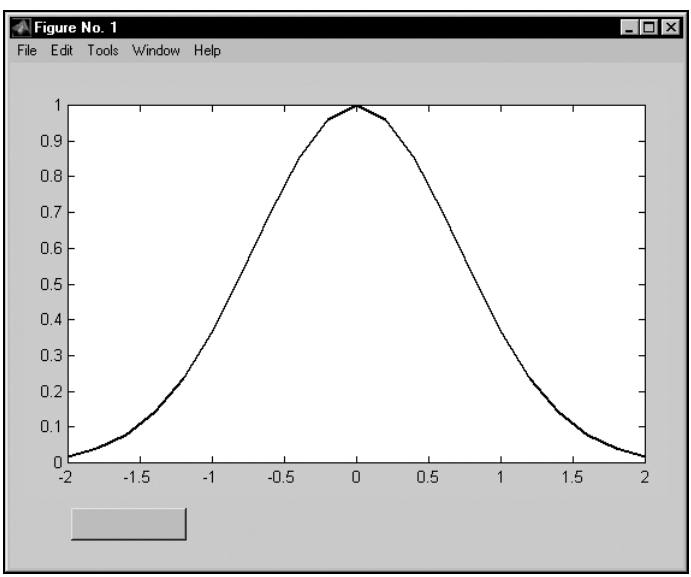


Рис. 10.7. Работающее приложение `mygui`

Замечание

В MatLab 5.3 приложению с графическим интерфейсом соответствует два файла с расширениями m и mat.

Перейдите в режим редактирования при помощи панели управления и продолжите работу над созданием приложения. Обратите внимание, что хотя приложение и находится в режиме редактирования, но график функции присутствует на осях. При повторном запуске сразу же появится график вне зависимости от действий пользователя. Дело в том, что команда `plot` создает графический объект — линию, информация о которой сохраняется в файлах приложения.

Добавьте в режиме редактирования еще одну кнопку так, как показано на рис. 10.8. Запрограммируйте событие `Callback` данной кнопки, указав единственный оператор `cla` очистки текущих осей. Программирование событий описано выше, на примере первой добавленной кнопки.

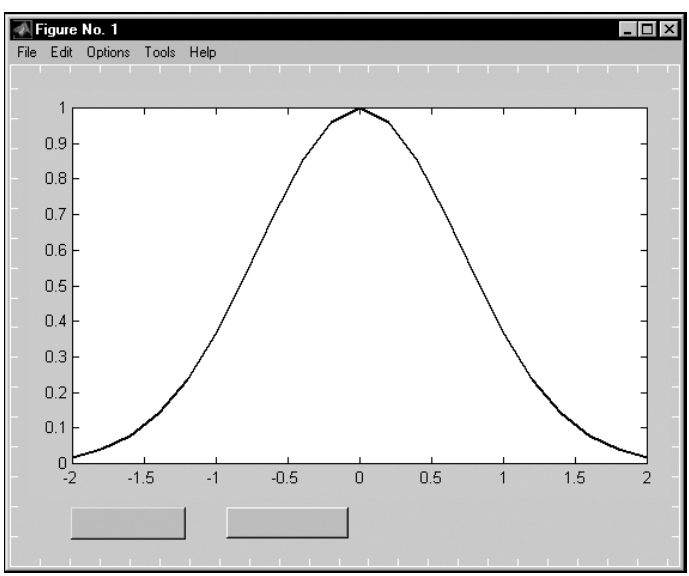


Рис. 10.8. Окно приложения с двумя кнопками

Замечание

Переход к программированию события может быть осуществлен либо выбором нужного объекта из иерархического списка навигатора объектов редактора вызовов, либо выделением объекта в окне приложения (в режиме редактирования) с последующим возвратом в редактор вызовов.

Снова запустите приложение. Нажатие на левую кнопку приводит к построению графика, а при помощи правой кнопки пользователь может удалить график.

Удобство использования приложения определяется понятным интерфейсом. Дальнейшая работа над приложением `mygui` описана в *главе 11*.

Принципы создания приложений в версии 6.*x*

В данном разделе рассмотрены основные принципы создания приложений в MatLab 6.*x*. Описана среда GUIDE разработки приложений и работа с файл-функцией приложения для программирования событий от элементов интерфейса.

Среда GUIDE

Перейдите в среду GUIDE, выполнив `guide` в командной строке. Появляется редактор окна приложения, заголовок которого **untitled.fig** означает, что в нем открыт новый файл, увеличьте немного размеры окна так, как изображено на рис. 10.9.

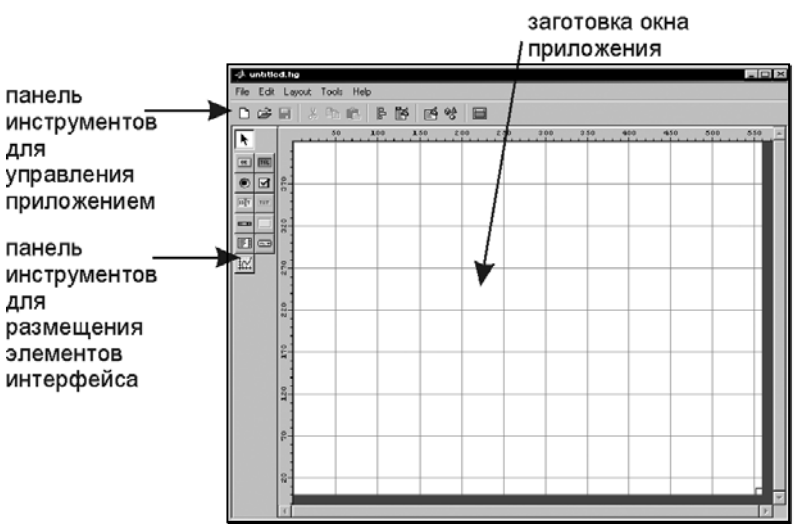


Рис. 10.9. Редактор приложения в версии 6.*x*

Редактор приложения содержит:

- строку меню;

- панель инструментов управления приложением;
- заготовку окна приложения с нанесенной сеткой;
- вертикальную и горизонтальную линейки;
- панель инструментов для добавления элементов интерфейса на окно приложения.

Редактор приложения MatLab 6.*x* позволяет разместить различные элементы интерфейса (рис. 10.10). Требуется нажать соответствующую кнопку на панели инструментов и поместить выбранный объект щелчком мыши в требуемое место заготовки окна приложения. Другой способ состоит в задании прямоугольной области объекта перемещением мыши по области заготовки окна с удержанием левой кнопки. Размер и положение добавленных объектов изменяются при помощи мыши. Перед изменением размера следует выбрать режим выделения объектов и сделать объект текущим, щелкнув по нему клавишей мыши.

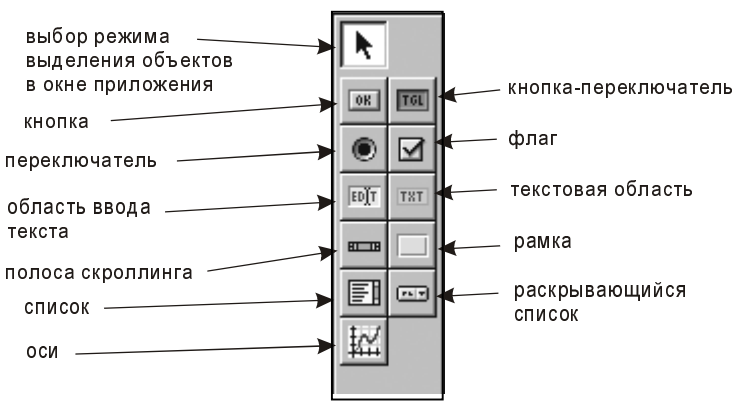


Рис. 10.10. Панель инструментов для добавления элементов интерфейса

Потренируйтесь располагать и изменять размеры элементов интерфейса, например, получите окно приложения, которое похоже на окно, изображенное на рис. 10.3.

Приложение в данный момент находится в режиме редактирования. Любой объект можно удалить с окна при помощи <Delete>, предварительно выделив его. Запуск приложения производится при помощи кнопки **Activate Figure** (рис. 10.11). Появляется диалоговое окно **GUIDE**, которое сообщает о необходимости сохранить приложение. Нажмите **Yes** и сохраните приложение в файле с расширением fig.

Приложение запускается в отдельном окне с заголовком **Untitled**. Пользователь может нажимать на кнопки, устанавливать флаги, переключатели, об-

ращаться к спискам. Разумеется, при этом ничего полезного не происходит. Более того, при нажатии на кнопки в командное окно MatLab выводится сообщение следующего содержания:

```
pushbutton1 Callback not implemented yet.
```

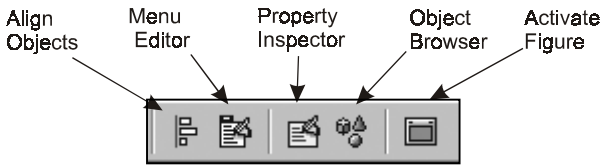


Рис. 10.11. Панель инструментов управления приложением

Данное сообщение говорит о том, что обработка события, которое происходит при нажатии на кнопку, не запрограммирована. Недостаточно разместить элементы интерфейса в окне приложения, следует позаботиться о том, чтобы каждый элемент выполнял нужные функции при обращении к нему пользователя. Например, при нажатии на кнопку производятся вычисления и строятся графики полученных результатов, переключатели позволяют установить цвет линий, полоса скроллинга изменяет толщину линии, в области ввода пользователь указывает некоторые параметры, управляющие ходом вычислений.

Программирование интерфейса, изображенного на рис. 10.3, является слишком сложным для начинающего программиста. Закройте окно приложения и редактор приложений. Следующий раздел посвящен изучению основ программирования элементов управления на простом примере.

Программирование интерфейса более подробно описано в главе 12.

Программирование событий в версии 6.*x*

Цель данного раздела состоит в объяснении принципа программирования событий в версии 6.*x*. Приложение в MatLab 6.*x* хранится в двух файлах с расширениями fig и m, первый из них содержит информацию о размещенных в окне приложения объектах, а второй является M-файлом с основной функцией и подфункциями. Добавление элемента интерфейса из редактора приложения приводит к автоматическому созданию соответствующей подфункции. Данную подфункцию следует наполнить содержимым — операторами, которые выполняют обработку события, возникающего при обращении пользователя к элементу интерфейса.

Программирование подфункций описано в разд. "Подфункции" главы 8.

Начните с создания простого приложения, окно которого содержит оси и две кнопки, предназначенные для построения графика и очистки осей.

Перейдите в среду создания приложения командой `guide`. Выберите в меню **Tools** редактора приложений пункт **Application Options**, появляется диалоговое окно **GUIDE Application Options**. Данное окно позволяет устанавливать некоторые общие свойства создаваемого приложения. Выберите в раскрывающемся списке **Command-line accessibility** строку **On (recomended for plotting windows)** и нажмите **OK**. Приложение, выводящее графики на оси, которые расположены в пределах окна приложения, требуют установки этой опции для корректной работы.

Расположите на форме оси кнопку так, как показано на рис. 10.12. На кнопке автоматически размещается надпись **Push Button**. Следующий этап очень важный. Кнопка является элементом интерфейса, ей следует дать имя, которое уникальным образом идентифицировало бы ее среди всех объектов окна приложения.

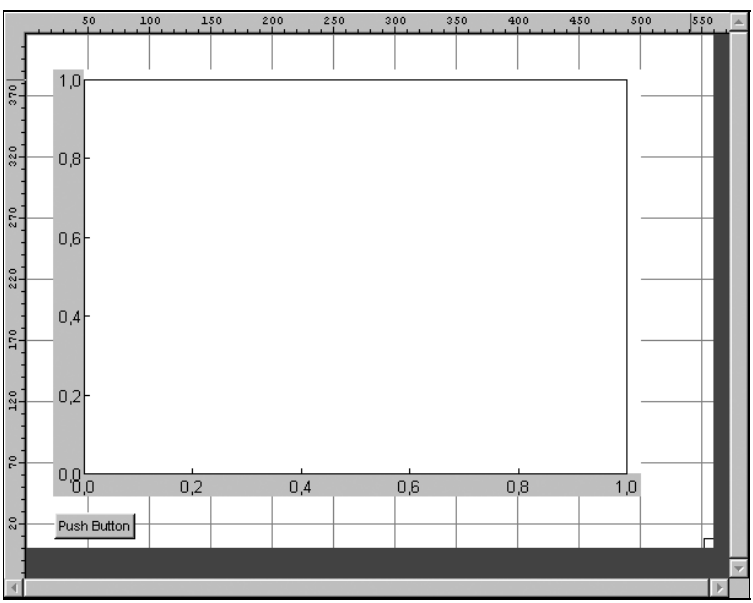


Рис. 10.12. Расположение кнопки и осей в окне приложения

Выделите кнопку **Push Button** и вызовите редактор свойств **Property Inspector** при помощи панели инструментов управления приложением (см. рис. 10.11). Появляется окно редактора свойств, приведенное на рис. 10.13, в котором содержится таблица названий свойств кнопки и их значений. Занесите в свойство `Tag` значение `btnPlot`, щелкните мышью по

строке справа от названия свойства, наберите требуемое значение и нажмите <Enter>. В дальнейшем будет говориться, что некоторому объекту или элементу управления следует дать имя. Например, `btnPlot` теперь является именем кнопки **Push Button**. Удобно задавать имена, часть которых определяет тип элемента управления (`btn` соответствует `button` — кнопке). Аналогичным образом дайте осям имя `axMain`.

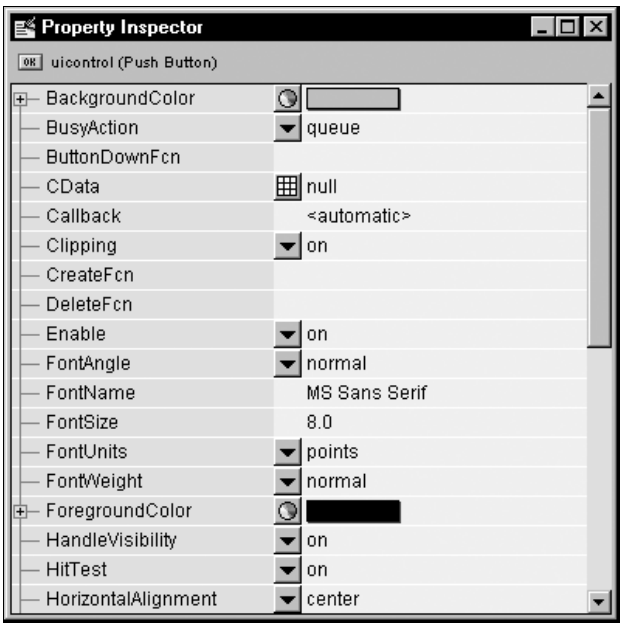


Рис. 10.13. Редактор свойств в версии 6.*x*

Выберите в меню **File** редактора приложения пункт **Save as**, создайте папку MyFirstGui и сохраните приложение в файле mygui.fig. Обратите внимание, что открылся редактор M-файлов, содержащий файл mygui.m. Данный файл имеет структуру, схематично представленную в листинге 10.2.

Листинг 10.2. Структура M-файла приложения с графическим интерфейсом

```
function varargout = mygui(varargin)
% Операторы инициализации приложения
%| ABOUT CALLBACKS:
% Краткая информация о программировании событий
% --------------------------------------------------------------------
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
```

```
% Подфункция обработки события Callback кнопки с именем btnPlot
disp('btnPlot Callback not implemented yet.')
```

Приложение `mygui` содержит одну кнопку **Press Button**. Когда пользователь нажимает на **Press Button** в работающем приложении, то происходит событие `Callback` данного элемента управления. Вызывается подфункция `btnPlot_Callback`. Сейчас она содержит оператор вывода в командное окно текста с предупреждением о том, что событие `Callback` пока не определено. Имя подфункции образовано названием кнопки и события. Очень важно задавать имена объектам в свойстве `Tag` *сразу после их добавления на окно приложения* в редакторе приложений, иначе генерируемая подфункция получит имя, которое сохранится при последующем изменении значения `Tag` и повлечет ошибки при выполнении приложения.

Завершающий этап состоит в программировании действий, которые выполняются при нажатии пользователем на кнопку **Press Button**. Измените функцию обработки события нажатия на **Press Button** в соответствии с листингом 10.3.

Листинг 10.3. Обработка события `Callback` кнопки с именем `btnPlot`

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
x = [-2:0.2:2];
y = exp(-x.^2);
plot(x,y)
```

Сохраните файл mygui.m в редакторе М-файлов и запустите приложение из редактора приложений, нажав кнопку **Activate Figure** (см. рис. 10.11). Нажатие на **Press Button** в запущенном приложении приводит к отображению графика функции на осях. Закройте окно приложения при помощи кнопки с крестиком в правом верхнем углу и продолжите работу над `mygui` в редакторе приложений.

Добавьте кнопку так, как показано на рис. 10.14, задайте ей имя `btnClear` в редакторе свойств. Быстрый доступ к свойствам выделенного объекта в редакторе приложений производится из пункта **Inspect Properties** всплывающего меню при нажатии правой кнопки мыши на объекте. Перейдите к подфункции обработки события `Callback` добавленной кнопки, для чего следует выбрать пункт **Edit Callback** всплывающего меню. Выбор данного пункта делает активным редактор М-файлов и выделяет заголовок соответствующей подфункции `btnClear_Callback`. Разместите единственный оператор очистки осей `cla` в подфункции (листинг 10.4).

Листинг 10.4. Обработка события кнопки с именем `btnClear`

```
function varargout = btnClear_Callback(h, eventdata, handles, varargin)
cla
```

Запустите приложение и убедитесь, что нажатие на левую кнопку приводит к отображению графика функции, а правая служит для очистки осей.

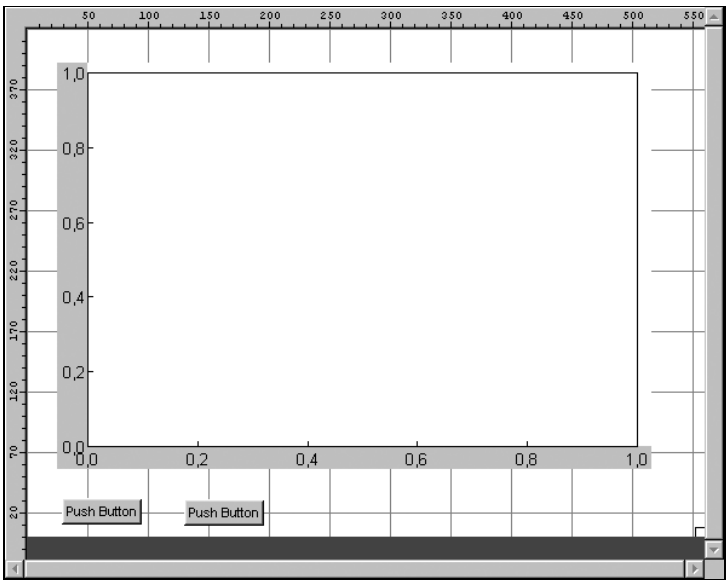


Рис. 10.14. Окно приложения с двумя кнопками

Следует позаботиться о том, чтобы интерфейс приложения был очевиден для пользователя. Дальнейшая работа над приложением `mygui` в MatLab 6.*x* описана в *главе 12*.

Глава 11



Конструирование интерфейса в версии 5.3

Данная глава посвящена работе над приложением `mygui`, которая была начата в предыдущей главе. Описан процесс создания приложения, предназначенного для визуализации данных и интерактивного управления видом получающихся графиков. Объяснено, как размещать различные элементы интерфейса в окне приложения и программировать их события.

Установка свойств объектов, функция *findobj*

Продолжим работу над приложением `mygui`, окно которого изображено на рис. 10.8. Очевидно, что следует надписать кнопки, например: **Построить** и **Очистить**. Кнопки являются графическими объектами с определенными свойствами, среди которых имеется и свойство, отвечающее за надпись на кнопке. Доступ к свойствам объектов, размещенных в окне приложения, производится из редактора свойств **Graphics Propetry Editor**, активизируемого кнопкой **Propetry Editor** панели управления.

Применение редактора свойств для изменения вида графиков описано в разд. "Редактирование графиков в MatLab 5.3" главы 4.

Сейчас редактор понадобится для установки свойств объектов, осуществляющих интерфейс приложения. Сделайте левую кнопку приложения `mygui` текущей и вызовите редактор свойств. Установите свойство `String` левой кнопки в значение `Построить`, а `Tag` измените со значения, данного по умолчанию `Pushbutton1`, на `btnPlot` (рис. 11.1). Значения свойств заключаются в апострофы. Очень важно понять, что значение свойства `String` соответствует надписи на кнопке, а `Tag` — имени или тегу кнопки, как объекта. Имена объектов используются для изменения их свойств в ходе работы приложения при выполнении блоков обработки событий от других элементов интерфейса.

Перейдите теперь к свойствам правой кнопки и установите `String` в `Очистить`, а `Tag` измените со значения, данного по умолчанию `Pushbutton2`, на `btnClear`.

Замечание

Переход к свойствам объекта может быть осуществлен либо выбором его из иерархического списка навигатора объектов редактора свойств, либо выделением объекта в окне приложения (в режиме редактирования) с последующим возвратом в редактор свойств.

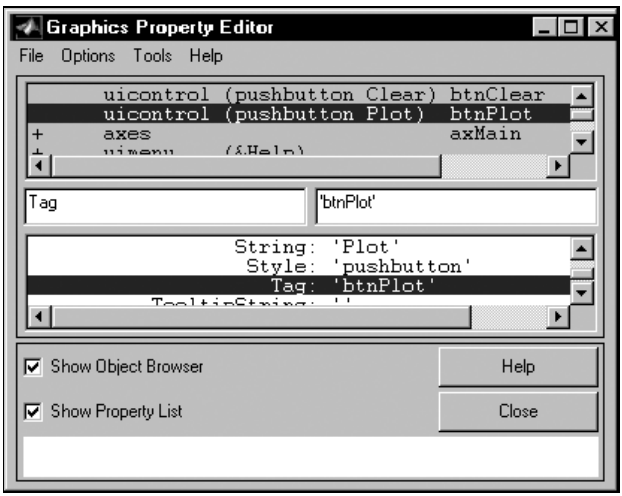


Рис. 11.1. Изменение свойств элементов интерфейса

Обратитесь к свойствам осей и измените значение их свойства `Tag` на `axMain`. Удобно давать объектам имена, которые состоят из двух частей, первая указывает на тип объекта, а вторая — на его назначение. Запустите приложение, кнопки теперь снабжены надписями, которые определяются соответствующими значениями свойств `String` (рис. 11.2).

Большинство свойств объектов можно устанавливать программно прямо в ходе работы приложения для обеспечения согласованного поведения элементов управления. Усовершенствуйте приложение `mygui` следующим образом. Пусть при запуске доступной является только кнопка **Построить**, при нажатии на кнопку **Построить** выводится график и она становится недоступной, зато пользователь может нажать кнопку **Очистить** для очистки осей, и тогда эта кнопка становится недоступной, а кнопка **Построить** — доступной. Итак, всегда доступна только одна из кнопок в зависимости от состояния осей.

Решение поставленной задачи требует привлечения свойства `Enable`. Свойство `Enable` объекта отвечает за возможность доступа к нему пользователем, значение `on` разрешает доступ, а `off`, соответственно, запрещает. Установка значений свойствам объектов производится при помощи функции `set`.

См. разд."Свойства графических объектов" главы 9.

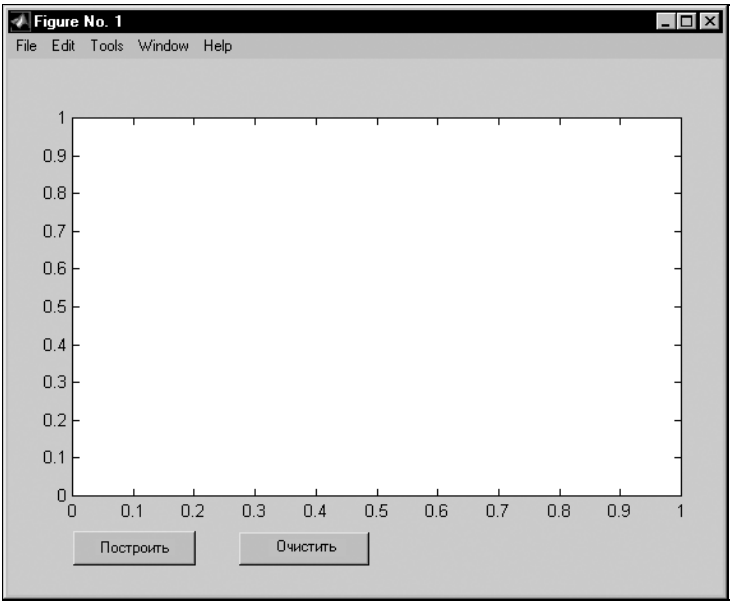


Рис. 11.2. Вид работающего приложения `mygui` с надписями на кнопках (нажата кнопка **Clear**)

Функция `set` вызывается с тремя входными аргументами — указателем на объект, названием свойства и его значением, последние два аргумента заключаются в апострофы. Основная проблема состоит в том, что свойства одного объекта должны изменяться в блоке операторов обработки события `Callback` другого объекта. Следовательно, должна иметься возможность поиска указателя на любой существующий объект. В MatLab определена функция `findobj`, которая как раз и производит требуемую операцию. Входными аргументами `findobj` являются название свойства и значение, а выходным — указатель на объект с подходящим значением свойства. Результат записывается в вектор, если существует несколько объектов, у которых некоторое свойство (например, цвет) имеет одинаковые значения. Какое же свойство и значение единственным образом характеризует объект?

При создании объекта и добавлении его в окно приложения следует давать каждому объекту уникальные имена или теги (свойство `Tag`). В рассматриваемом примере кнопка с надписью **Построить** имеет имя `btnPlot`, кнопка **Очистить** — `btnClear`, оси — `axMain`. Поиск указателя, например, на кнопку **Построить** выглядит так:

```
HbtnPlot = findobj('Tag', 'btnPlot');
```

Изменение свойств текущего объекта во время обработки его события также производится функцией `set`, но искать указатель на объект, *событие кото-*

рого обрабатывается в текущий момент времени, необязательно — он возвращается функцией `gcbo`.

Разрешение и запрещение доступа к кнопкам **Построить** и **Очистить** требуют внесения дополнений в обработку событий `Callback`. Перейдите в режим редактирования приложения так, чтобы *оси не содержали* графика функции (для этого достаточно нажать кнопку **Очистить** в работающем приложении перед сменой режима). В блок обработки события `Callback` кнопки **Построить** добавьте при помощи редактора вызовов следующее:

- поиск указателя на кнопку **Очистить**;
- установку свойства `Enable` кнопки **Очистить** в значение `on` (после вывода графика следует разрешить доступ к **Очистить**);
- установку свойства `Enable` кнопки **Построить** в значение `off` (после вывода графика следует запретить доступ к **Построить**).

Аналогичные изменения произведите в обработке события `Callback` кнопки **Очистить**, а именно:

- поиск указателя на кнопку **Построить**;
- установку свойства `Enable` кнопки **Построить** в значение `on` (после очистки осей следует разрешить доступ к **Построить**);
- установку свойства `Enable` кнопки **Очистить** в значение `off` (после очистки осей доступ к этой кнопке следует запретить).

События `Callback`, возникающие при нажатии кнопок, теперь должны быть запрограммированы так, как показано в листингах 11.1 и 11.2.

Листинг 11.1. Обработка события `Callback` кнопки Построить

```
x = [-2:0.2:2];
y = exp(-x.^2);
plot(x,y)
HbtnClear = findobj('Tag', 'btnClear');
set(HbtnClear, 'Enable', 'on')
set(gcbo, 'Enable', 'off')
```

Листинг 11.2. Обработка события `Callback` кнопки Очистить

```
cla
HbtnPlot = findobj('Tag', 'btnPlot');
set(HbtnPlot, 'Enable', 'on')
set(gcbo, 'Enable', 'off')
```

Запустите приложение mygui и убедитесь, что всегда доступной является только одна из кнопок **Построить** или **Очистить**, что является хорошей подсказкой для пользователя о возможных действиях. Сохраните приложение mygui так, как было описано в предыдущей главе, и закройте панель управления и окно приложения. Следующий раздел посвящен запуску приложения из командной строки и переходу в режим редактирования.

Работа над приложением

Запуск приложения осуществляется не только из панели управления. Возникает вопрос, как работать с уже созданным приложением с графическим интерфейсом и, возможно, вносить в него требуемые изменения. Для запуска приложения достаточно в качестве команды задать его имя в командной строке:

```
>> mygui
```

Появляется окно приложения, обращение к элементам интерфейса окна приводит к соответствующим действиям.

Замечание

Каталог с приложением должен содержаться в путях поиска MatLab или являться текущим. Установка путей поиска описана в *главе 5*.

Очень часто сразу не удается написать законченное приложение, и необходимое усовершенствование проявляется только в ходе работы с приложением. В любой момент можно перейти в режим редактирования двумя способами:

- задать guide в командной строке, что приводит к отображению панели управления и переводу запущенного приложения в режим редактирования;
- выбрать в меню **File** командного окна MatLab пункт **Show GUI Layout Tool** (появляется панель управления) и перейти в режим редактирования, изменив в окне **Figure List** Active на Controlled.

Перейдите в режим редактирования приложения mygui любым из перечисленных способов и продолжите работу над ним. Добавьте в название окна приложения **Figure No. 1** слова "Визуализация функций" и уберите строку меню так, чтобы работающее приложение mygui имело вид, изображенный на рис. 11.3.

Заголовок окна задается свойством Name графического окна (объект figure в иерархическом списке), а за наличие или отсутствие строки "Figure No." и номера отвечает свойство NumberTitle. Скрыть меню в окне приложения позволяет свойство MenuBar. Итак, для придания окну приложения требуе-

мого вида выполните следующие действия в редакторе свойств. Установите свойству `Name` объекта `figure` значение `Визуализация функций`, свойству `NumberTitle` — значение `off` и свойству `MenuBar` — `none`. Обратите внимание, что в режиме редактирования установка `MenuBar` в `none` не дает эффекта, строка меню все равно отображается в окне приложения. Убедитесь, что в работающем приложении строка меню действительно отсутствует. Дело в том, что меню окна приложения в режиме редактирования дублирует элементы интерфейса панели управления и предназначено именно для редактирования приложения.

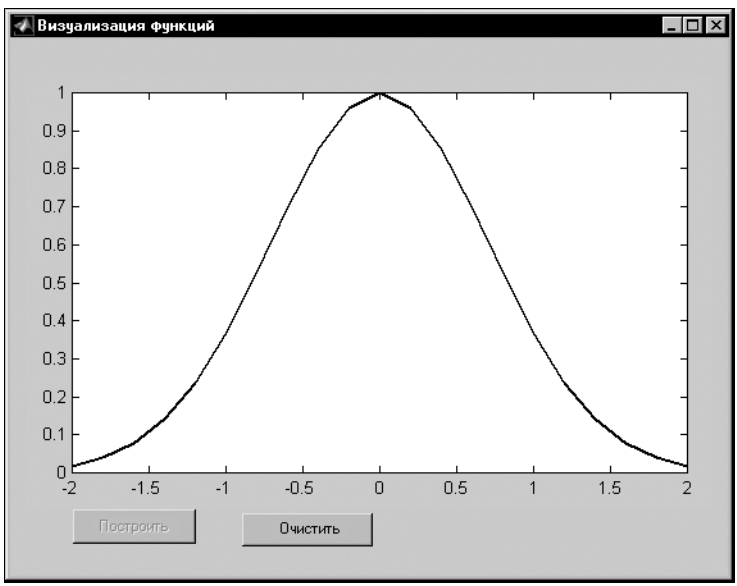


Рис. 11.3. Приложение без строки меню и с названием окна

Обобщая вышеизложенные сведения можно сказать, что процесс программирования приложения с графическим интерфейсом включает в себя следующие этапы:

- размещение элементов интерфейса в окне приложения;
- программирование событий данных элементов;
- сохранение приложения;
- запуск и работа с приложением;
- переход в режим редактирования для дальнейшего усовершенствования приложения.

Следует отметить недостатки изложенного выше способа программирования интерфейса приложения. Наличие большого числа элементов интерфейса приводит к необходимости обрабатывать достаточно много событий. Кроме того, поведение элементов интерфейса может быть согласованным (например кнопки **Построить** и **Очистить** в приложении `mygui`). Редактор вызовов предназначен для программирования каждого события в отдельной области ввода, поэтому одновременно посмотреть на операторы, обрабатывающие события от других приложений не удается. Наилучшим вариантом является программирование событий в файл-функции. Изучению данного подхода посвящен следующий раздел.

Программирование событий в файл-функции

Оформление обработки событий в отдельной файл-функции основано на создании дополнительной файл-функции с одним входным аргументом. В редакторе вызовов вместо блока операторов обработки события размещается вызов файл-функции с входным аргументом, который определяет блок выполняемых операторов. Данные операторы совпадают с использовавшимися ранее в редакторе вызовов для обработки события. Ветвление в файл-функции производится при помощи оператора `switch`.

См. разд. "Оператор switch" главы 7.

В редакторе М-файлов создайте файл-функцию `myguiprog` (листинг 11.3) и сохраните ее в файле myguiprog.m в том же каталоге, что и само приложение. Используйте копирование через буфер обмена из редактора вызовов.

Листинг 11.3. Файл-функция `myguiprog` обработки событий приложения `mygui`

```
function myguiprog(event)
% Файл-функция для обработки событий приложения mygui

% Определение события
switch event
case 'pressPlot' % Пользователь нажал кнопку Построить
   % Построение графика
   x = [-2:0.2:2];
   y = exp(-x.^2);
   plot(x,y)
```

```
   % Поиск указателя на кнопку Очистить
   HbtnClear = findobj('Tag', 'btnClear');
   % Разрешение доступа к кнопке Очистить
   set(HbtnClear, 'Enable', 'on')
   % Запрещение доступа к кнопке Построить (она — текущий объект)
   set(gcbo, 'Enable', 'off')
case 'pressClear'
   % Пользователь нажал кнопку Очистить
   cla % очистка осей
   % Поиск указателя на кнопку Построить
   HbtnPlot = findobj('Tag', 'btnPlot');
   % Разрешение доступа к кнопке Построить
   set(HbtnPlot, 'Enable', 'on')
   % Запрещение доступа к кнопке Очистить (она — текущий объект)
   set(gcbo, 'Enable', 'off')
end
```

Входным аргументом файл-функции `myguiprog` является строковая переменная `event`, значение которой определяет нужный блок `case`, содержащий обработку события. Замените в редакторе вызовов операторы обработки события `Callback` кнопки **Построить** на вызов

```
myguiprog('pressPlot'),
```

а для кнопки **Очистить** используйте

```
myguiprog('pressClear')
```

Не забудьте нажать **Apply** в редакторе вызовов после проделанных изменений. Запустите приложение `mygui` и убедитесь, что оно работает так же, как и раньше. Теперь при необходимости обработки других событий следует добавить соответствующий блок операторов в файл-функцию `myguiprog`, сохранить ее, и поместить вызов `myguiprog` в окно редактора вызовов для данного события. Использование строковой переменной `event` в качестве входного аргумента `myguiprog` позволяет задавать интуитивно понятные значения, например `'pressPlot'`, не пренебрегайте данной возможностью!

Следующие разделы посвящены программированию основных элементов графического интерфейса. Последовательно модернизируйте приложение `mygui`, размещайте новые элементы в окно приложения, изменяйте файл-функцию `myguiprog` и добавляйте вызов `myguiprog` с нужным аргументом в редакторе вызовов.

Программирование элементов интерфейса

Флаги

Флаги позволяют произвести одну или несколько установок, определяющих ход работы приложения. Усовершенствуйте приложение `mygui`, снабдив окно приложения двумя флагами с названиями **сетка по x** и **сетка по y**. Если пользователь нажимает кнопку **Построить**, то на оси наносится сетка по выбранным координатам. Нажатие на **Очистить** должно приводить не только к исчезновению графика функции, но и скрытию сетки.

Окно приложения должно содержать два флага. Обычно несколько элементов управления со схожим назначением группируются и помещаются внутри рамки. Порядок расположения элементов в окне приложения определяется в MatLab последовательностью их создания. Нанесите рамку на окно приложения при помощи кнопки **frame**, расположенной на панели **New Object Pallete** (см. рис. 10.2), предварительно изменив размеры осей. Добавьте на рамку флаги (рис. 11.4).

Разместите поясняющие подписи рядом с флагами и дайте им имена. Задайте свойству `Tag` верхнего флага значение `chbxGridX`, а свойству `String`, отвечающему за подпись флага, значение `сетка по x`. Аналогичным образом определите свойства нижнего флага, установите свойство `Tag` в `chbxGridY`, и `String` в `сетка по y`. Если текст не помещается рядом с флагом, увеличьте ширину области флага при помощи мыши, удерживая нажатой левую кнопку.

Осталось сделать так, чтобы при нажатии пользователем кнопки **Построить** происходило отображение линий сетки в зависимости от установленных флагов, а нажатие на **Очистить** приводило к скрытию сетки. Блок обработки события `Callback` кнопки **Построить** следует дополнить проверкой состояния флагов. Свойство флага `Value` принимает значение логической единицы при включении флага пользователем, и, соответственно, равно нулю, если флаг выключен. Перед проверкой следует найти указатели на флаги при помощи `findobj`, используя в качестве аргументов свойство `Tag` и его значение для нужного флага. Состояние флагов должно определять значение свойств `XGrid` и `YGrid` осей.

Измените в файл-функции `myguiprog` (см. листинг 11.3) блок `case` обработки нажатия кнопки **Построить** в соответствии с листингом 11.4, а в блок для **Очистить** добавьте `grid off`. Не забудьте сохранить файл-функцию!

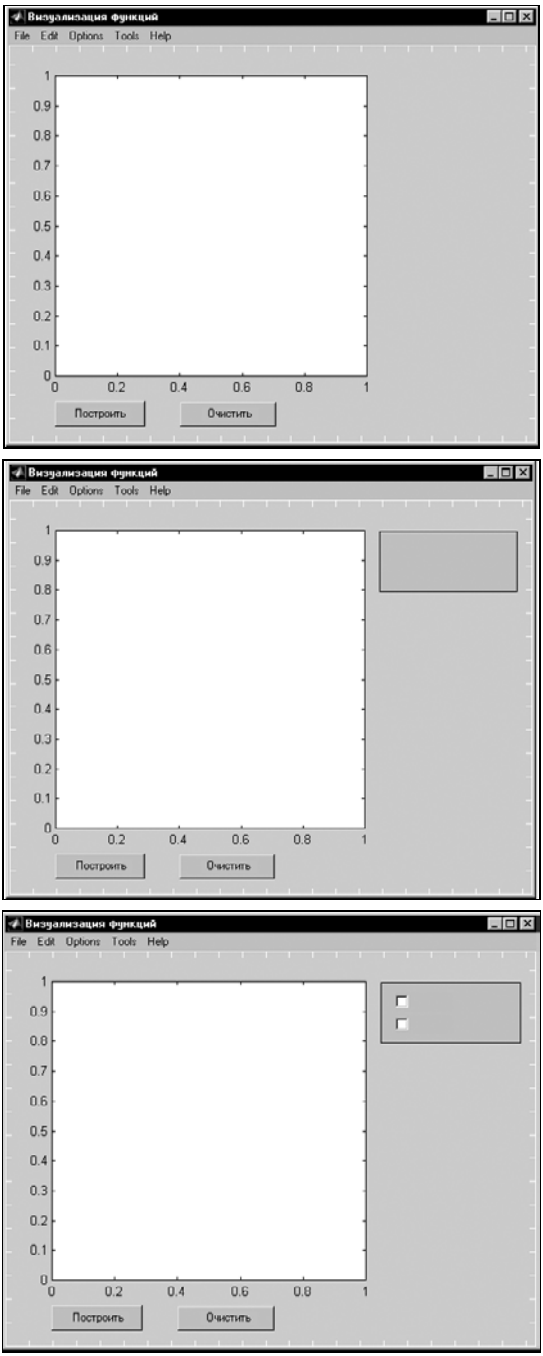


Рис. 11.4. Добавление флагов в рамке

Листинг 11.4. Обработка события кнопки Построить с учетом состояния флагов

```
case 'pressPlot' % Пользователь нажал кнопку Построить
   % Построение графика
   x = [-2:0.2:2];
   y = exp(-x.^2);
   plot(x,y);
   % Поиск указателя на флаг с подписью 'сетка по x'
   HchbxGridX = findobj('Tag', 'chbxGridX');
   % Проверка состояния флага
   if get(HchbxGridX, 'Value')
      % Флаг включен, добавляется сетка по x на текущие оси
      set(gca, 'XGrid', 'on')
   else
      % Флаг выключен, сетка по x скрывается
      set(gca, 'XGrid', 'off')
   end
   % Поиск указателя на флаг с подписью 'сетка по y'
   HchbxGridY = findobj('Tag', 'chbxGridY');
   % Проверка состояния флага
   if get(HchbxGridY, 'Value')
      % Флаг включен, добавляется сетка по y на текущие оси
      set(gca, 'YGrid', 'on')
   else
      % Флаг выключен, сетка по y скрывается
      set(gca, 'YGrid', 'off')
   end
   % Поиск указателя на кнопку Очистить
   hClear = findobj('Tag', 'btnClear');
   % Разрешение доступа к кнопке Очистить
   set(hClear, 'Enable', 'on')
   % Запрещение доступа к кнопке Построить (она — текущий объект)
   set(gcbo, 'Enable', 'off')
```

Запустите приложение `mygui` и убедитесь, что установка флагов влияет на отображение сетки при нажатии на кнопку **Построить** (рис. 11.5), а кнопка **Очистить** убирает не только график функции, но и сетку.

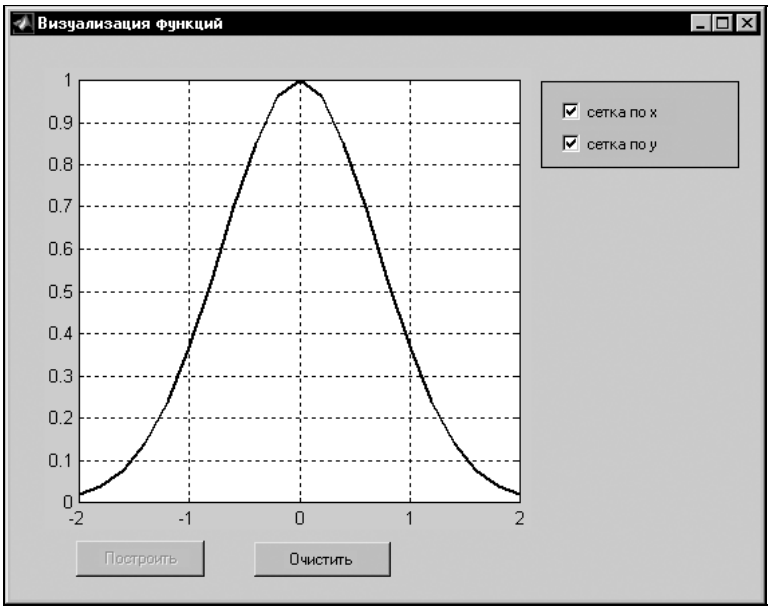


Рис. 11.5. Задание сетки флагами

Смена состояния флагов сетки не приводит к немедленным изменениям на графике. Пользователь должен перестроить график, нажимая последовательно **Очистить** и **Построить**. Для немедленного реагирования приложения на состояние флагов следует определить их события `Callback`. Программирование данных событий заключается в проверке состояния флага и отображении или скрытии соответствующих линий сетки. Используйте `gcbo` вместо `findobj` для определения указателя на флаг, т. к. требуется найти указатель на объект, событие которого обрабатывается в текущий момент времени. Внесите два блока `case`, приведенные в листинге 11.5, в файл-функцию `myguiprog` и в редакторе вызовов укажите в обработке `Callback` флагов соответствующие обращения:

```
myguiprog('pressGridX')
myguiprog('pressGridY')
```

Листинг 11.5. Обработка событий `Callback` флагов сетки

```
case 'pressGridX'
   % Проверка состояния флага
   if get(gcbo, 'Value')
      % Флаг включен, добавляется сетка по x на текущие оси
      set(gca, 'XGrid', 'on')
```

```
   else
      % Флаг выключен, сетка по x скрывается
      set(gca, 'XGrid', 'off')
   end
case 'pressGridY'
   % Проверка состояния флага
   if get(gcbo, 'Value')
      % Флаг включен, добавляется сетка по x на текущие оси
      set(gca, 'YGrid', 'on')
   else
      % Флаг выключен, сетка по x скрывается
      set(gca, 'YGrid', 'off')
   end
```

Внесенные дополнения позволяют пользователю наносить и убирать сетку по каждой координате при помощи флагов без перестроения графика функции.

Флаги предоставляют возможность выбора одной или сразу нескольких опций. Одновременный выбор *только одной* опции осуществляется при помощи переключателей.

Переключатели

Переключатели обычно группируются по их предназначению, и пользователь может выбрать только одну опцию. Всегда установлен только один переключатель из группы. Обработка событий переключателя должна влиять на состояние остальных переключателей всей группы. Модернизируйте интерфейс приложения `mygui`, предоставьте пользователю возможность выбирать тип маркера (кружок, квадрат или отсутствие маркера).

Добавьте в окно приложения новую рамку и нанесите на нее три переключателя, установите свойствам `Tag` значения `rbMarkCirc`, `rbMarkSq`, `rbMarkNone`, а `String` — `маркеры-круги`, `маркеры-квадраты`, `без маркеров` соответственно (рис. 11.6).

Состояние переключателя определяется его свойством `Value`, если `Value` равно единице, то переключатель установлен (включен), если нулю — выключен. Задайте в редакторе свойств значение, равное единице, свойству `Value` переключателя **без маркеров**, он будет установлен при запуске программы. Теперь придется дополнить файл-функцию `myguiprog` соответствующими блоками обработки события `CallBack` каждого переключателя и занести в редакторе вызовов подходящие обращения к `myguiprog`.

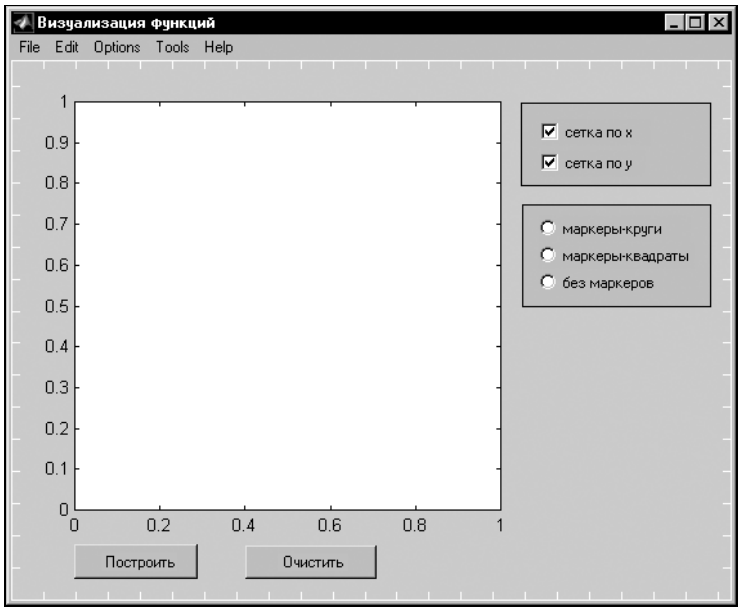


Рис. 11.6. Добавление группы переключателей

Предположим, что пользователь установил один из переключателей. Происходит обращение к соответствующему блоку в `myguiprog`, который:

- изменяет тип маркеров линии;
- выключает два остальных переключателя.

Второе действие программируется достаточно просто — следует найти указатели на другие переключатели, используя значения их свойств `Tag` и `findobj`, и занести в `Value` ноль. Изменение типа маркеров линии не представляет труда, если известен указатель на линию. Достаточно обратиться к свойству линии `Marker`. Указатель на линию возвращает `plot` в выходном аргументе, его следует записать в некоторую переменную, например `Hline`. Использовать `Hline` придется в других блоках `myguiprog`, обрабатывающих событие `Callback` переключателей. Очень важно понять, что переменная `Hline` инициализируется при вызове файл-функции с аргументом `'pressPlot'`, при этом выполняется соответствующий блок и файл-функция *заканчивает работу*. Все переменные, определенные в файл-функции, являются локальными и по окончании работы файл-функции они *не определены*. Пользователь выбирает один из переключателей, происходит *новый вызов* `myguiprog` и выполнение соответствующего блока операторов — использование неопределенной переменной `Hline` в качестве указателя на линию не приведет к требуемому результату.

Самым простым способом решения проблемы о хранении указателей на объекты является объявление соответствующих переменных глобальными в начале файл-функции.

Внесите необходимые изменения в текст файл-функции `myguiprog` (листинг 11.6):

1. Объявите `Hline` глобальной переменной.
2. Сохраните указатель на линию при построении графика командой `plot`.
3. Добавьте блоки обработки событий `Callback` переключателей.

Сохраните файл-функцию и поместите в редакторе вызовов соответствующие обращения к `myguiprog` для переключателей.

Листинг 11.6. Обработка событий переключателей в `myguiprog`

```
function myguiprog(event)
% Файл-функция для обработки событий приложения mygui
global Hline % объявление указателя на линию глобальной переменной
switch event
case 'pressPlot' % пользователь нажал кнопку Построить
    ...
    Hline = plot(x,y); % сохранение указателя на линию
    ...
case 'pressClear'
    ...
case 'pressGridX'
    ...
case 'pressGridY'
    ...
case 'pressMarkCirc'
   % Установлен переключатель 'маркеры-круги'
   set(Hline, 'Marker', 'o'); % задание типа маркера линии
   % Поиск указателей на остальные переключатели и занесение
   % нуля в Value — переключатели должны быть выключены
   HrbMarkSq = findobj('Tag', 'rbMarkSq');
   set(HrbMarkSq, 'Value', 0)
   HrbMarkNone = findobj('Tag', 'rbMarkNone');
   set(HrbMarkNone, 'Value', 0)
case 'pressMarkSq'
   % Установлен переключатель 'маркеры-квадраты'
   set(Hline, 'Marker', 's'); % задание типа маркера линии
```

```
    % Поиск указателей на остальные переключатели и занесение
    % нуля в Value — переключатели должны быть выключены
    HrbMarkCirc = findobj('Tag', 'rbMarkCirc');
    set(HrbMarkCirc, 'Value', 0)
    HrbMarkNone = findobj('Tag', 'rbMarkNone');
    set(HrbMarkNone, 'Value', 0)
case 'pressMarkNone'
    % Установлен переключатель 'без маркеров'
    set(Hline, 'Marker', 'none'); % задание типа маркера линии
    HrbMarkCirc = findobj('Tag', 'rbMarkCirc');
    % Поиск указателей на остальные переключатели и занесение
    % нуля в Value — переключатели должны быть выключены
    set(HrbMarkCirc, 'Value', 0)
    HrbMarkSq = findobj('Tag', 'rbMarkSq');
    set(HrbMarkSq, 'Value', 0)
end
```

Запустите приложение `mygui`, постройте график функции, нажав **Построить**, и убедитесь в том, что возможна установка только одного из переключателей и она приводит к появлению соответствующих маркеров на графике функции. Интерфейс `mygui` имеет ряд недостатков.

- Если пользователь использует кнопку **Очистить** для очистки осей, а затем устанавливает переключатель, то производится обращение к несуществующему объекту линии (сообщение об ошибке выводится в командное окно).
- Нажатие на кнопку **Построить** приводит к построению линии без маркеров вне зависимости от установленного переключателя.
- Повторный щелчок по области переключателя приводит к его выключению, но всегда один из переключателей должен быть установлен.

Конечно, первый недостаток является существенным — приложение должно работать без ошибок! Проще всего запретить доступ к переключателям, если нет линии на графике, и разрешить после появления линии. Очевидно, что следует внести изменения в соответствующие блоки `myguiprog`, обрабатывающие события `Callback` кнопок. Нажатие на **Построить** должно открывать доступ к группе переключателей, а очистка осей кнопкой **Очистить** — запрещать доступ. Итак, следует найти указатели на переключатели и установить их свойство `Enabled` в нужное значение `on` или `off`.

Обратите внимание, что требуется выполнить, по существу, одинаковые действия объектов. Можно найти указатель на каждый переключатель, используя уникальное значение свойства `Tag` и `findobj`, затем занести в `Enable` значение

off. Существует более простой способ поиска объектов одного класса и одновременного изменения их свойств. Объекты, реализующие элементы интерфейса, имеют свойство Style, значение которого определяет, является ли объект кнопкой (pushbutton), флагом (checkbox), переключателем (radiobutton) и т. д. Следует найти указатели на объекты со значением Style, равным radiobutton. Функция findobj запишет результат в вектор из указателей. Установить свойству Enable значение off сразу для всех переключателей позволяет функция set, первым аргументом которой является вектор указателей на нужные объекты. Произведите необходимые изменения в блоках case, которые соответствуют нажатию на кнопки **Построить** и **Очистить**. Обратитесь к листингу 11.7, содержащему требуемые операторы.

Листинг 11.7. Разрешение и запрещение доступа к группе переключателей

```
switch event
case 'pressPlot' % пользователь нажал кнопку Построить
   ...
   % Поиск вектора указателей на переключатели
   HRadioBtns = findobj('Style', 'radiobutton')
   % Разрешение доступа ко всем переключателям
   set(HRadioBtns, 'Enable', 'on')
case 'pressClear'
   ...
   % Поиск вектора указателей на переключатели
   HRadioBtns = findobj('Style', 'radiobutton');
   % Запрещение доступа ко всем переключателям
   set(HRadioBtns, 'Enable', 'off')
```

Первый недостаток интерфейса приложения mygui устранен. Теперь необходимо сделать так, чтобы при построении графика тип маркера отвечал установленному переключателю. Очевидно, что после вывода графика следует найти переключатель со значением Value, равным единице, и установить соответствующий тип маркера (листинг 11.8).

Листинг 11.8. Изменение маркеров при построении графика

```
case 'pressPlot' % пользователь нажал кнопку Построить
   ...
   Hline = plot(x,y);
   % Поиск указателя на переключатель 'маркеры-квадраты'
```

```
HrbMarkSq = findobj('Tag', 'rbMarkSq');
if get(HrbMarkSq, 'Value')
   % Переключатель 'маркеры-квадраты' установлен
   set(Hline, 'Marker', 's') % изменение маркеров на квадраты
end
% Поиск указателя на переключатель 'маркеры-круги'
HrbMarkCirc = findobj('Tag', 'rbMarkCirc');
if get(HrbMarkCirc, 'Value')
% Переключатель 'маркеры-круги' установлен
   set(Hline, 'Marker', 'o') % изменение маркеров на круги
end
```

Сохраните изменения в `myguiprog` и запустите приложение `mygui`. Теперь тип маркеров определяется установленным переключателем при построении графика.

Замечание

Программирование событий в отдельной файл-функции позволяет вносить изменения в блоки обработки событий при запущенном приложении. Важно только сохранять файл-функцию с проделанными изменениями перед продолжением тестирования приложения. Разумеется, добавление элементов интерфейса и работа в редакторе вызовов и свойств может осуществляться только в режиме редактирования.

Осталась нерешенной одна проблема. При повторном нажатии на переключатель он выключается, но всегда должен быть установлен единственный переключатель. Данный недостаток устраняется с привлечением еще одного возможного значения `inactive` свойства `Enable`. Переключатель со значением `inactive` является *неактивным*, он выглядит в работающем приложении как доступный переключатель (со значением `on`), но попытка изменить состояние данного переключателя не приводит к успеху. Усовершенствуйте обработку событий согласно следующему алгоритму.

1. Свойство `Enabled` переключателя, событие которого обрабатывается, должно иметь значение `inactive`, а для остальных двух значение `on`. Если не задать `on` для других переключателей, то в результате все они станут неактивными.
2. При нажатии на кнопку **Построить** свойству `Enable` всех переключателей присваивается значение `on`, а затем определяется установленный в данный момент переключатель и в его свойство `Enable` заносится значение `inactive`.

Дополните соответствующие блоки `myguiprog` необходимыми операторами (листинг 11.9). Обратите внимание, что в блоке `case` обработки нажатия кнопки **Построить** использована форма вызова `findobj` с первым аргумен-

том — вектором указателей на все найденные переключатели. Происходит *сужение поиска* объектов с `Value`, равным единице, только среди объектов с указателями, хранящимися в первом аргументе — векторе.

Листинг 11.9. Предотвращение выключения переключателя повторным нажатием

```
switch event
case 'pressPlot' % пользователь нажал кнопку Построить
   ...
   % Поиск указателя на включенный переключатель
   % среди всех переключателей
   HTurn = findobj(HRadioBtns, 'Value', 1);
   % Включенный переключатель должен быть неактивным,
   % для того чтобы пользователь не выключил его
   set(HTurn, 'Enable', 'inactive')
case 'pressClear'
   ...
case 'pressGridX'
   ...
case 'pressGridY'
   ...
case 'pressMarkCirc'
   ...
   % Текущий переключатель должен стать неактивным
   set(gcbo, 'Enable', 'inactive')
   % Установка Enable остальных переключателей в on
   set(HrbMarkSq, 'Enable', 'on')
   set(HrbMarkNone, 'Enable', 'on')
case 'pressMarkSq'
   ...
   % Текущий переключатель должен стать неактивным
   set(gcbo, 'Enable', 'inactive')
   % Установка Enable остальных переключателей в on
   set(HrbMarkCirc, 'Enable', 'on')
   set(HrbMarkNone, 'Enable', 'on')
case 'pressMarkNone'
   ...
   % Текущий переключатель должен стать неактивным
   set(gcbo, 'Enable', 'inactive')
```

```
    set(HrbMarkSq, 'Enable', 'on')
    % Установка Enable остальных переключателей в on
    set(HrbMarkCirc, 'Enable', 'on')
end
```

Правильная обработка переключателей (см. листинги 11.6—11.9) требует предусмотреть все ситуации, которые могут возникнуть при взаимодействии приложения с пользователем. Наличие взаимно несвязанных групп переключателей не позволяет использовать групповой поиск указателей по свойству `Style` для последующего изменения свойств. Проще всего использовать имя, определяемое свойством `Tag`, и искать указатель на каждый переключатель.

Раскрывающиеся списки реализуют альтернативный способ выбора пользователем только одной из предлагаемых опций.

Списки

Модернизируйте интерфейс приложения `mygui`, предоставьте пользователю возможность выбора цвета линии графика из раскрывающегося списка (синий, красный, зеленый). Перейдите в режим редактирования и добавьте при помощи панели управления раскрывающийся список (рис. 11.7). В редакторе свойств установите свойство `Tag` в значение `'pmColor'`.

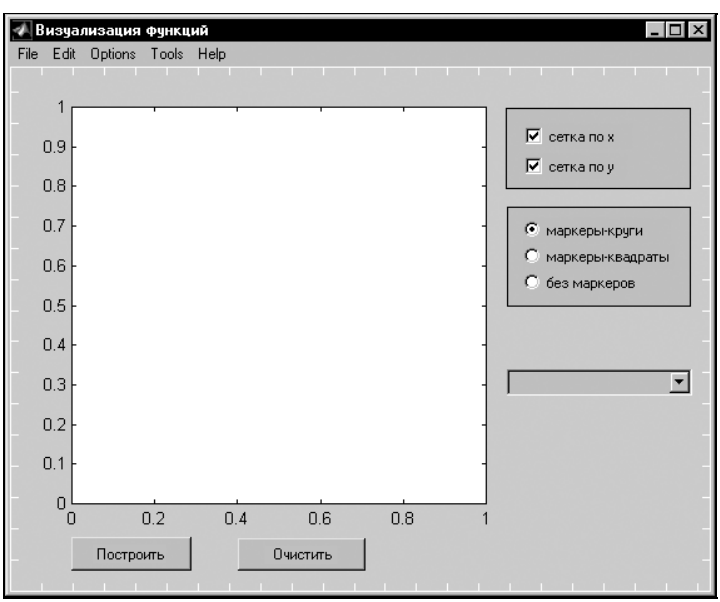


Рис. 11.7. Добавление раскрывающегося списка

Элементами раскрывающегося списка являются строки. При создании списка следует занести в его свойство `String` массив ячеек, содержащий нужные строки.

Организация массивов ячеек описана в разд. "Массивы ячеек" главы 8.

Возникает вопрос, как занести массив ячеек в область ввода значения свойства в редакторе свойств. Если раскрывающийся список содержит несколько строк, то их можно набрать, используя в качестве разделителя вертикальную черту:

```
'синий|красный|зеленый'
```

Нажмите <Enter>, введенная строка преобразовалась в глобальный массив `towork`. *Закройте редактор свойств* для сохранения введенной информации в соответствующем файле приложения. Запустите `mygui` и убедитесь, что раскрывающийся список содержит требуемые строки. Выбор различных строк пока не приводит к изменению цвета линии, — требуется запрограммировать событие `Callback` раскрывающегося списка.

Замечание

Раскрывающийся список может состоять из достаточно большого набора строк, вводить их все в редакторе свойств не очень удобно. Альтернативным вариантом является создание глобального массива ячеек в командном окне `str = {'синий'; 'красный'; 'зеленый'}`. Имя массива ячеек `str` используется в качестве значения свойства `String` раскрывающегося списка, заключать имя массива в апострофы не надо! Происходит автоматическое преобразование значения `str` в массив `towork`, для сохранения информации в файлах приложения следует закрыть редактор свойств.

Обработка события `Callback` раскрывающегося списка состоит в определении выбора пользователя и соответствующем изменении цвета линии. Свойство списка `Value` содержит номер выбранной строки (строки списка нумеруются, начиная с единицы). Дополните файл-программу `myguiprog` блоком `case` обработки выбора пользователя. Используйте оператор `switch` внутри данного блока для установки цвета линии в зависимости от номера выбранной строки списка (листинг 11.10).

Листинг 11.10. Обработка выбора пользователя из раскрывающегося списка

```
switch event
   ...
case 'pressColor'
   % Пользователь произвел выбор из раскрывающегося списка
   % Определение номера выбранной строки
   Num = get(gcbo, 'Value');
```

```
switch Num
case 1
   % Выбран синий цвет (первая строка списка)
   set(Hline, 'Color', 'b'); % установка синего цвета линии
case 2
   % Выбран зеленый цвет (вторая строка списка)
   set(Hline, 'Color', 'r'); % установка красного цвета линии
case 3
   % Выбран красный цвет (третья строка списка)
   set(Hline, 'Color', 'g'); % установка зеленого цвета линии
end
...
```

Внесите обращение `myguiprog('pressColor')` в редакторе вызовов для события `Callback` раскрывающегося списка. Запустите приложение, постройте график, нажав на **Построить**, и убедитесь в том, что раскрывающийся список позволяет изменять цвет линии графика функции. Несложно заметить, что интерфейс `mygui` имеет ряд недостатков.

- Повторное построение графика не учитывает текущий выбор цвета в раскрывающемся списке.
- Выбор цвета при отсутствии линии на графике приводит к ошибке (`Hline` указывает на несуществующий объект).
- Рядом со списком требуется разместить текст, поясняющий назначение списка.

Устраните первый недостаток, поместите в блоке `case` обработки нажатия кнопки **Построить** задание цвета построенной линии в зависимости от состояния раскрывающегося списка (листинг 11.11).

Листинг 11.11. Построение линии графика с учетом состояния списка

```
switch event
case 'pressPlot' % пользователь нажал кнопку Построить
   ...
   HpmColor = findobj('Tag', 'pmColor'); % поиск указателя на список
   Num = get(HpmColor, 'Value'); % определение номера выбранной строки
   switch Num
   case 1
      set(Hline, 'Color', 'b');
   case 2
      set(Hline, 'Color', 'r');
```

```
   case 3
      set(Hline, 'Color', 'g');
   end
```

Изменение цвета линии при отсутствии графика лишено смысла, поэтому следует запретить доступ пользователя к раскрывающемуся списку, и, напротив, разрешить при построении графика. Внесите необходимые дополнения в блоки `case`, соответствующие нажатию на кнопки, используйте свойство списка `Enable` (листинг 11.12).

Листинг 11.12. Разрешение и запрещение доступа к раскрывающемуся списку

```
case 'pressPlot' % пользователь нажал кнопку Построить
   ...
   set(HpmColor, 'Enable', 'on') % разрешение доступа к списку
case 'pressClear' % пользователь нажал кнопку Очистить
   ...
   % Поиск указателя на раскрывающийся список
   HpmColor = findobj('Tag', 'pmColor');
   set(HpmColor, 'Enable', 'off') % запрещение доступа к списку
...
```

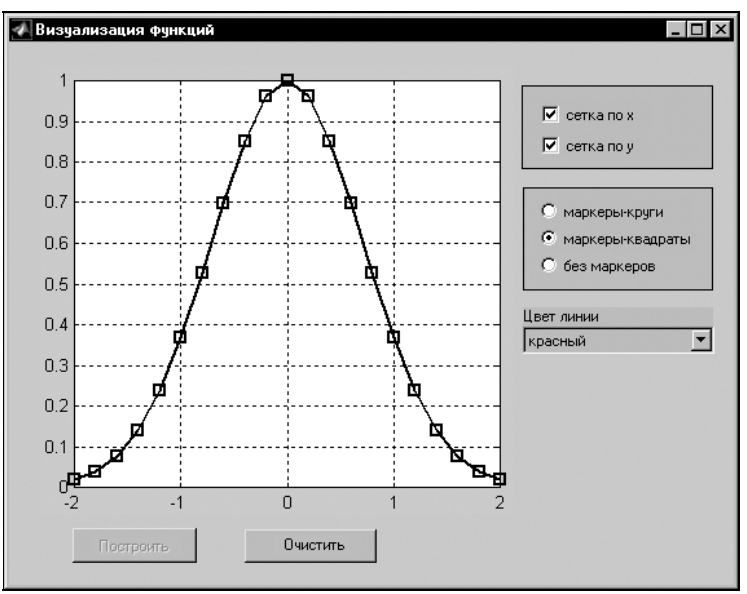


Рис. 11.8. Добавление текста

Многие элементы интерфейса, в частности раскрывающиеся списки, следует сопровождать поясняющим текстом. Перейдите в режим редактирования и при помощи панели управления разместите текстовую область над списком. Установите в редакторе свойств `String` в значение `Цвет линии`, а `HorizontalAlignment` в значение `left` для добавленного объекта. Теперь работающее приложение имеет более наглядный интерфейс (рис. 11.8).

Программирование событий обычных списков производится практически аналогично. Отличие состоит в том, что в обычных списках одновременно могут быть выделены несколько элементов. Свойство `Value` содержит вектор номеров выбранных элементов. Разрешение выбора нескольких элементов определяется значениями свойств `Max` и `Min` списка. Если разность `Max-Min` больше единицы, то пользователь может выделить несколько строк.

Полосы скроллинга

Усовершенствуйте интерфейс приложения `mygui`, предоставив пользователю возможность устанавливать ширину линии при помощи полосы скроллинга. Добавьте полосу скроллинга в окно приложения и снабдите ее текстовым пояснением "Толщина линии", так же, как и раскрывающийся список (рис. 11.9).

Задайте название `scrWidth` в свойстве `Tag` полосы. Теперь следует определить соответствие между положением бегунка полосы и числовым значением свойства `Value`.

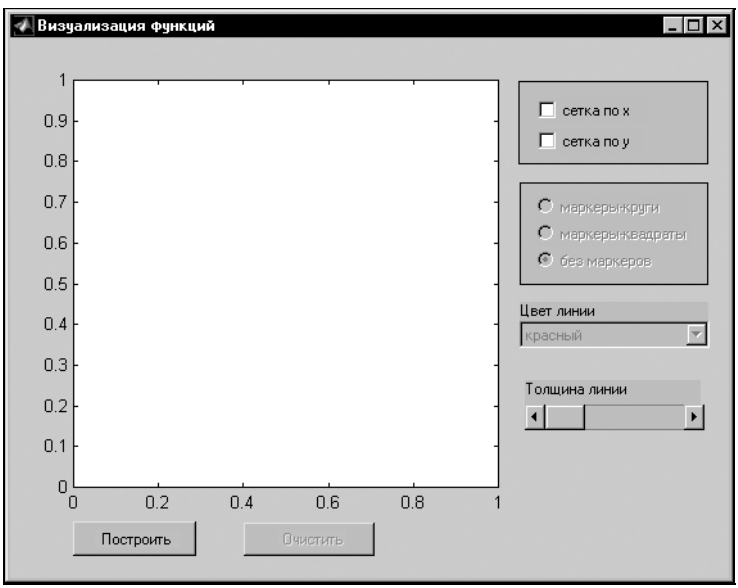


Рис. 11.9. Добавление полосы скроллинга

Проделайте следующие установки из редактора свойств.

1. В `Max` занесите десять, а в `Min` — единицу. Свойства `Max` и `Min` полосы скроллинга отвечают за границы значений, записываемых в `Value`, при перемещении бегунка.
2. Определите начальное положение, записав в `Value` единицу.
3. Обратитесь к свойству `SliderStep`. Его значением является вектор из двух компонентов, первый из которых определяет относительное изменение `Value` при нажатии на кнопки со стрелками полосы скроллинга, а второй — при перетаскивании бегунка мышью. Установите [0.1 0.2], кнопки теперь изменяют `Value` на десять процентов, а щелчок мыши справа или слева от бегунка — на двадцать.

Осталось запрограммировать событие `Callback` полосы скроллинга с именем `scrWidth`, которое состоит в задании ширины линии, равной округленному значению `Value`. Снабдите файл-программу `myguiprog` блоком `case`, приведенным в листинге 11.13, и добавьте в редакторе графиков вызов `myguiprog('pressWidth')`.

Листинг 11.13. Обработка события `Callback` полосы скроллинга

```
case 'pressWidth'
   HscrWidth = findobj('Tag', 'scrWidth');
   width = get(HscrWidth, 'Value');
   set(Hline, 'LineWidth', round(width))
```

Запустите `mygui` и убедитесь, что полоса скроллинга позволяет легко изменять толщину линии построенного графика. Устраните самостоятельно некоторые недочеты интерфейса. Полоса скроллинга должна быть недоступной после очистки осей кнопкой **Очистить**, построение графика при помощи кнопки **Построить** произведите с учетом установленной ширины линии. Данные недостатки устраняются внесением соответствующих изменений в обработку событий `Callback` перечисленных кнопок так же, как и в предыдущих разделах.

Область ввода текста

Обычные текстовые области, использовавшиеся на протяжении предыдущих разделов, позволяют лишь вывести некоторый текст в окно приложения. Обмен текстовой информацией между пользователем и приложением осуществляется при помощи областей ввода текста. Предоставьте пользователю возможность размещать заголовок на графике. Текст заголовка пользователь вводит в соответствующей строке.

Добавьте в окно приложения область ввода текста и снабдите ее пояснением **Заголовок** в текстовой области, расположенной выше (рис. 11.10).

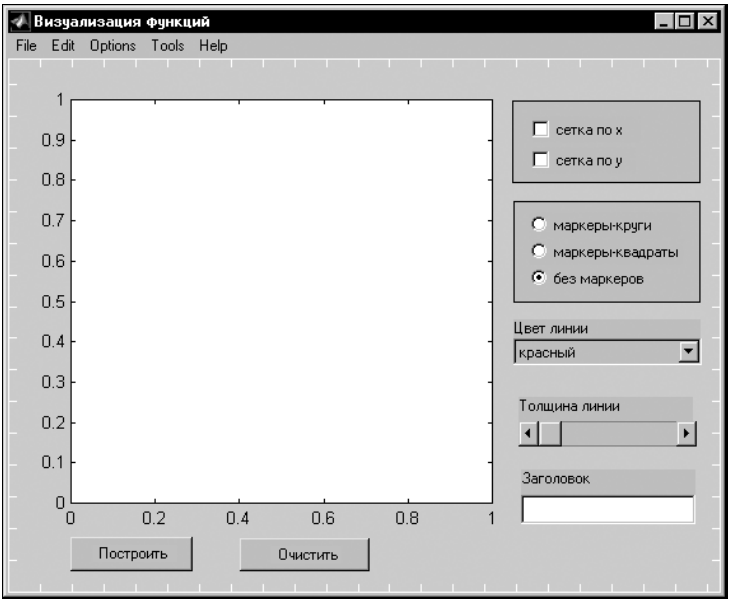


Рис. 11.10. Добавление области ввода текста

Установите значение `editTitle` свойству `Tag` области ввода. Теперь следует добавить в `myguiprog` блок `case` обработки ввода текста (листинг 11.14), и поместить обращение `myguiprog('pressTitle')` в редакторе вызовов на событие `Callback` области ввода.

Листинг 11.14. Обработка события `Callback` области ввода текста

```
case 'pressTitle'
   txt = get(gcbo, 'String');
   title(txt)
```

Запустите приложение `mygui` и нанесите заголовок на график, набрав его в области ввода и нажав клавишу <Enter>. Дополните обработку события кнопки **Очистить** очисткой заголовка при помощи команды `title('')`.

Вышеописанный пример демонстрирует использование области ввода, состоящей из одной строки. Разность значений свойств `Max` и `Min` области ввода определяет, позволяет ли данная область ввод многострочного текста. Если разность больше единицы, то заносимый пользователем текст записывается в массив ячеек текстовых строк, который хранится в свойстве `String`.

Глава 12



Конструирование интерфейса в версии 6.*x*

Данная глава посвящена продолжению работы с графическим интерфейсом пользователя, которая была начата *в главе 10* над приложением `mygui`, в версии MatLab 6.*x*. Описан процесс создания приложения, предназначенного для визуализации данных и интерактивного управления видом получающихся графиков. Объяснено, как размещать элементы интерфейса в окне приложения и программировать их события в M-файле, связанном с приложением.

Управление свойствами объектов

Разработка приложения сопряжена с изменением свойств объектов, которые они получают по умолчанию при размещении их на заготовке окна. Некоторые из свойств, например надпись на кнопке или ее размер, устанавливаются при создании объекта в режиме редактирования. Другие свойства могут изменяться программно в работающем приложении.

Установка свойств при редактировании

Продолжите работу над приложением `mygui`, окно которого было изображено на рис. 10.14. Очевидно, что следует надписать кнопки, например **Построить** и **Очистить**. Кнопки являются графическими объектами с определенными свойствами, среди которых имеется свойство, отвечающее за надпись на кнопке. Сделайте левую кнопку приложения `mygui` текущей и вызовите редактор свойств Property Inspector. Установите свойство `String` левой кнопки в значение `Построить`. Важно понять, что значение свойства `String` соответствует надписи на кнопке, а `Tag` — имени или тегу кнопки, как объекта. Имена объектов используются для изменения их свойств в ходе работы приложения при выполнении блоков обработки событий от других элементов интерфейса. Перейдите теперь к свойствам правой кнопки и установите `String` в `Очистить`.

Замечание

Доступ к редактору свойств выделенного объекта производится либо из панели инструментов управления приложением, либо из меню **Tools** редактора приложений, либо при помощи пункта **Inspect Properties** всплывающего меню.

Значение свойства `String` сразу отображается на кнопке приложения, находящегося в режиме редактирования. Запустите приложение, кнопки теперь снабжены надписями, которые определяются соответствующими значениями свойств `String`. Аналогичным образом устанавливаются и другие свойства объектов.

Программное изменение свойств

Большинство свойств объектов можно устанавливать программно прямо в ходе работы приложения для обеспечения согласованного поведения элементов управления. Усовершенствуйте приложение `mygui` следующим образом. Пусть при запуске доступной является только кнопка **Построить**, при нажатии на кнопку **Построить** выводится график и она становится недоступной, зато пользователь может нажать кнопку **Очистить** для очистки осей, и наоборот. Итак, всегда доступна только одна из кнопок в зависимости от состояния осей.

Решение поставленной задачи требует привлечения свойства `Enable`. Свойство `Enable` объекта отвечает за возможность доступа к нему пользователем, значение `on` разрешает доступ, а `off`, соответственно, запрещает. Установка значений свойствам объектов в программе производится при помощи функции `set`.

См. разд. "Свойства графических объектов" главы 9.

Функция `set` вызывается с тремя входными аргументами — указателем на объект, названием свойства и его значением, последние два аргумента заключаются в апострофы. Заметьте, что свойства одного объекта должны изменяться в блоке операторов обработки события `Callback` другого объекта. Следовательно, должна иметься возможность доступа к указателю на любой существующий объект. Аргументы `h` и `handles` подфункций, которые обрабатывают события элементов управления, содержат требуемые указатели. В `h` хранится указатель на тот объект, событие которого обрабатывается в данный момент, а `handles` является структурой указателей. Поля структуры совпадают со значениями свойств `Tag` существующих элементов интерфейса. Например, `handles.btnPlot` является указателем на кнопку **Построить** с именем `btnPlot`.

Доступ к **Очистить** должен быть запрещен в начале работы приложения, пока пользователь не нажмет **Построить**. Установите в редакторе свойств для кнопки **Очистить** `Enable` в `off`, используйте кнопку со стрелкой в строке со значением свойства. Остальные изменения значения `Enable` кнопок должны происходить в ходе работы приложения. Для разрешения и запрещения доступа к кнопкам нужно внести дополнения в обработку их событий `Callback`.

В подфункцию обработки события `Callback` кнопки **Построить** добавьте при помощи редактора вызовов:

- установку свойства `Enable` кнопки **Очистить** в значение `on` (после вывода графика следует разрешить доступ к **Очистить**);
- установку свойства `Enable` кнопки **Построить** в значение `off` (после вывода графика следует запретить доступ к **Построить**);

Аналогичные изменения произведите в обработке события `Callback` кнопки **Очистить**, а именно:

- установку свойства `Enable` кнопки **Построить** в значение `on` (после очистки осей следует разрешить доступ к **Построить**);
- установку свойства `Enable` кнопки **Очистить** в значение `off` (после очистки осей следует запретить доступ к кнопке);

Подфункции `btnPlot_Callback` и `btnClear_Callback` должны быть запрограммированы так, как показано на листинге 12.1.

Листинг 12.1. Обработка событий `Callback` кнопок `btnPlot` и `btnClear`

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
% Построение графика функции
x = [-2:0.2:2];
y = exp(-x.^2);
plot(x,y)
% Кнопка Построить должна стать недоступной после вывода графика
set(h, 'Enable', 'off')
% Кнопка Очистить должна стать доступной
set(handles.btnClear, 'Enable', 'on')

function varargout = btnClear_Callback(h, eventdata, handles, varargin)
cla % очистка осей
% Кнопка Очистить должна стать недоступной после очистки осей
set(h, 'Enable', 'off')
% Кнопка Построить должна стать доступной
set(handles.btnPlot, 'Enable', 'on')
```

Сохраните изменения в редакторе М-файлов. Запустите приложение `mygui` и убедитесь, что всегда доступной является только одна из кнопок **Построить** или **Очистить**, что является хорошей подсказкой для пользователя о возможных действиях. Закройте окно приложения и редактор приложений. Следующий раздел посвящен запуску приложения из командной строки и переходу в режим редактирования.

Работа над приложением

Запуск приложения

Запуск приложения осуществляется не только из редактора приложений. Возникает вопрос, как работать с уже созданным приложением с графическим интерфейсом и, возможно, вносить в него требуемые изменения. Для запуска приложения достаточно в качестве команды задать его имя в командной строке

```
>> mygui
```

Появляется окно приложения, обращение к элементам интерфейса окна приводит к соответствующим действиям.

Замечание

Каталог с приложением должен содержаться в путях поиска MatLab или являться текущими. Установка путей поиска описана в *главе 5*.

Очень часто сразу не удается написать законченное приложение, и необходимое усовершенствование проявляется только в ходе работы с приложением. В любой момент можно продолжить редактирование двумя способами:

- задать `guide` в командной строке, что приводит к отображению редактора приложений с заготовкой для нового приложения untitled1.fig и открыть соответствующий файл с расширением fig, выбрав пункт **Open...** в меню **File** редактора (закройте ненужный редактор приложений с untitled1.fig);
- указать имя приложения через пробел после команды `guide`, например `guide mygui`, появляется редактор приложений, в котором открыто `mygui`.

Перейдите в режим редактирования приложения `mygui` любым из перечисленных способов и продолжите работу над ним. Измените название окна приложения на "Визуализация функций" так, чтобы работающее приложение `mygui` имело вид, изображенный на рис. 11.3.

Заголовок окна задается свойством `Name` графического окна. Сделайте графическое окно текущим, щелкнув мышью по области заготовки окна в редакторе приложений. Установите в редакторе свойств `Name` в `'Визуализация функций'`. Запустите `mygui`, появляется окно **GUIDE** с предупреждением о том, что продолжение повлечет сохранение приложения. Установите флаг **Don't tell me again** для подавления вывода данного предупреждения при дальнейшей работе и нажмите **OK**. Убедитесь, что приложение имеет нужный заголовок.

Обобщая вышеизложенные сведения, можно сказать, что процесс программирования приложения с графическим интерфейсом включает в себя этапы:

- размещение элементов интерфейса в окне приложения;
- программирование событий данных элементов в подфункциях М-файла приложения;

- сохранение приложения;
- запуск и работа с приложением;
- переход в режим редактирования для дальнейшего усовершенствования приложения.

Желательно располагать элементы интерфейса в порядке, обеспечивающем удобную работу пользователя с приложением. Приложение хорошо выглядит, если однотипные элементы, например кнопки, флаги и т. д., определенным образом выровнены в окне приложения. MatLab 6.*x* предоставляет разработчику приложений простые способы выравнивания добавляемых объектов — сетку и линейку.

Оформление интерфейса

Редактор приложений содержит заготовку окна приложения с нанесенной сеткой и вертикальную и горизонтальную линейки (см. рис. 10.9). Линейка позволяет размещать элементы интерфейса в позиции с любыми координатами в пикселах, отсчитываемыми от левого нижнего угла заготовки окна приложения. Следует выделить объект щелчком мыши и перемещать его по окну, следя за указателями его положения на линейке. Происходит непрерывное движение объекта, что не всегда удобно.

Часто требуется, чтобы небольшое перемещение мыши вызывало изменение положения объекта на некоторый фиксированный шаг. Сетка редактора приложений позволяет осуществить такое дискретное движение. Выбор пункта **Grid and Rulers** меню **Layout** приводит к появлению диалогового окна **Grid and Rulers**, изображенного на рис. 12.1.

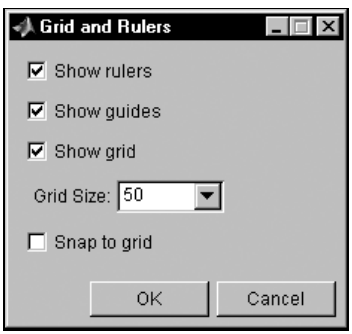


Рис. 12.1. Диалоговое окно **Grid and Rulers**

Флаги **Show rulers** и **Show grid** соответствуют отображению линеек и сетки в редакторе приложений, а раскрывающийся список **Grid Size** позволяет вы-

брать размер ячеек сетки. Минимально допустимый размер десять пикселов позволяет достаточно точно располагать элементы управления в окне приложений. Привязка перемещения к линиям сетки происходит при установленном флаге **Snap to grid**. Привязка разрешает разместить объект и изменить его размеры только при условии прохождения границы объекта по линиям сетки. Выбор мелкого шага сетки в сочетании с привязкой предоставляет разработчику возможность быстро оформить приложение. Плавно изменять положение выделенного объекта можно при помощи клавиш со стрелками. Одновременное удержание <Ctrl> приводит к перемещению с учетом привязки к сетке.

Размещение объектов в ряд по вертикали или горизонтали требует предварительного определения некоторой вспомогательной линии. Можно, конечно, использовать линии сетки, но проще добавить горизонтальные или вертикальные линии выравнивания. Следует навести курсор мыши на соответствующую линейку (курсор меняет форму на двустороннюю стрелку) и, удерживая левую кнопку мыши, вытащить на форму синюю линию. Передвижение объектов к линии выравнивания вызывает автоматическое расположение их границ на данной линии. Сами линии выравнивания можно убирать с окна приложения, перетаскивая их мышью обратно на линейку. Флаг **Show giudes** окна **Grid and Rules** отвечает за отображение линий выравнивания в области окна приложения.

Замечание

Привязка действует даже при выключенных флагах **Show rulers**, **Show guides** и **Show grid**.

Размер окна приложения изменяется при помощи перетаскивания мышью за квадратик, расположенный в правом нижнем углу заготовки окна приложения.

Отредактируйте вид приложения `mygui`, используя вышеописанные возможности. Следующие разделы посвящены размещению и программированию основных элементов интерфейса.

Программирование элементов интерфейса

Флаги и рамки

Флаги позволяют произвести одну или несколько установок, определяющих ход работы приложения. Продолжите работу над `mygui`, предоставьте пользователю возможность наносить линии сетки на график. Окно приложения должно содержать два флага с названиями **сетка по x** и **сетка по y**. Если пользователь нажимает кнопку **Построить**, то на оси наносится сетка по выбранным координатам. Нажатие на **Очистить** должно приводить не только к исчезновению графика функции, но и скрытию сетки.

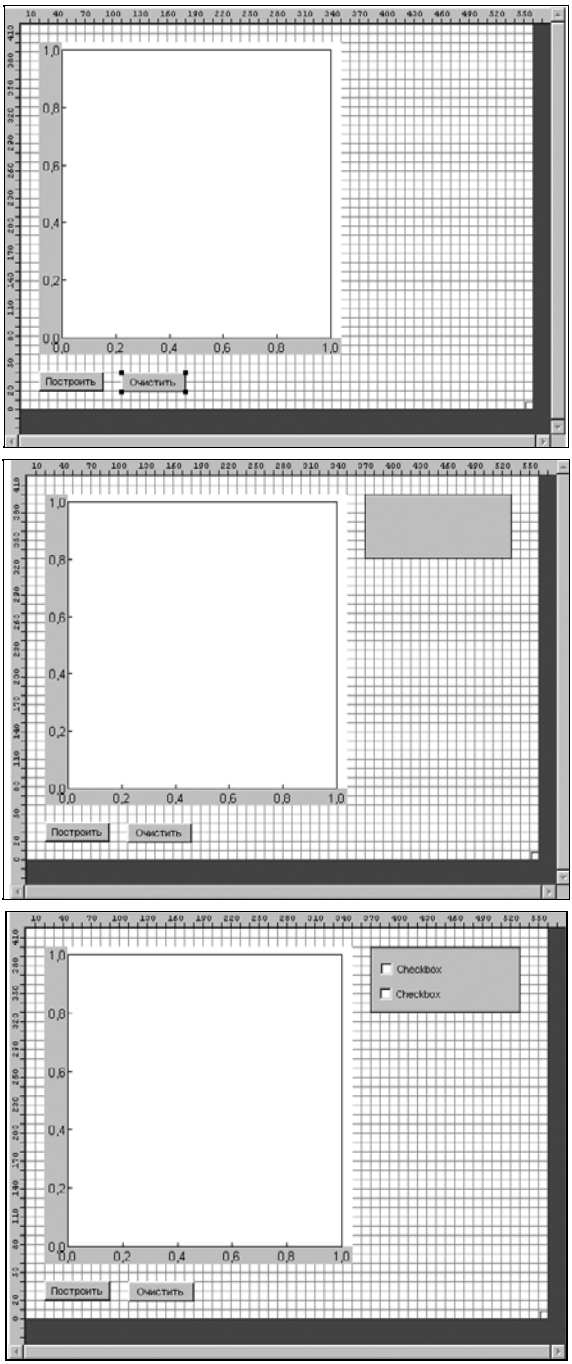


Рис. 12.2. Добавление флагов в рамке

Обычно несколько элементов управления со схожим назначением группируются и помещаются внутри рамки. Измените размеры осей, освободив справа место для рамки. Нанесите рамку на окно приложения при помощи соответствующей кнопки (см. рис. 10.10). В рамку добавьте два флага так, как показано на рис. 12.2.

Разместите поясняющие подписи рядом с флагами и дайте им имена. Задайте свойству `Tag` верхнего флага значение `chbxGridX`, а свойству `String`, отвечающему за подпись флага, значение `сетка по x`. Аналогичным образом определите свойства нижнего флага, установите свойство `Tag` в `chbxGridY`, и `String` в `сетка по y`. Если текст не помещается рядом с флагом, увеличьте ширину области флага при помощи мыши, удерживая нажатой левую кнопку. Сохраните приложение в редакторе приложений для автоматического создания в редакторе M-файлов заготовок для подфункций обработки события добавленных объектов.

Осталось сделать так, чтобы при нажатии пользователем кнопки **Построить** происходило отображение линий сетки в зависимости от установленных флагов, а нажатие на **Очистить** приводило к скрытию сетки. Блок обработки события `Callback` кнопки **Построить** следует дополнить проверкой состояния флагов. Свойство флага `Value` принимает значение логической единицы при включении флага пользователем, и, соответственно, равно нулю, если флаг выключен. Указатели на флаги содержатся в полях `chbxGridX` и `chbxGridY` структуры `handles`. Состояние флагов определяет значение свойств `XGrid` и `YGrid` осей.

Произведите необходимые изменения в подфункции обработки события `Callback` кнопки **Построить** с именем `btnPlot` (листинг 12.2).

Листинг 12.2. Обработка события кнопки `btnPlot` с учетом состояния флагов

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
% Построение графика функции
x = [-2:0.2:2];
y = exp(-x.^2);
plot(x,y)
% Проверка флага сетка по x
if get(handles.chbxGridX, 'Value')
   % Флаг включен, следует добавить линии сетки
   set(gca, 'XGrid', 'on')
else
   % Флаг выключен, следует убрать линии сетки
   set(gca, 'XGrid', 'off')
end
```

```
% Проверка флага сетка по x
if get(handles.chbxGridY, 'Value')
   % Флаг включен, следует добавить линии сетки
   set(gca, 'YGrid', 'on')
else
   % Флаг выключен, следует убрать линии сетки
   set(gca, 'YGrid', 'off')
end
% Кнопка Построить должна стать недоступной после вывода графика
set(h, 'Enable', 'off')
% Кнопка Очистить должна стать доступной
set(handles.btnClear, 'Enable', 'on')
```

Запустите приложение `mygui` и убедитесь, что установка флагов влияет на отображение сетки при нажатии на кнопку **Построить**.

Смена состояния флагов сетки не приводит к немедленным изменениям на графике. Пользователь должен перестроить график, нажимая последовательно кнопки **Очистить** и **Построить**. Для немедленного реагирования приложения на состояние флагов следует определить их события `Callback`. Программирование данных событий заключается в проверке состояния флага и отображении или скрытии соответствующих линий сетки.

Сделайте текущим флаг **сетка по x** в редакторе приложений и выберите опцию **Edit Callback** во всплывающем меню данного объекта. Запрограммируйте событие `Callback` флага. Используйте аргумент `h` соответствующих подфункций, содержащий указатель на объект, событие которого обрабатывается в текущий момент времени. Аналогичным образом обработайте событие `Callback` второго флага **сетка по y**.

Листинг 12.3. Обработка событий `Callback` флагов сетки

```
function varargout = chbxGridX_Callback(h, eventdata, handles, varargin)
% Проверка флага сетка по x
if get(h, 'Value')
   % Флаг включен, следует добавить линии сетки
   set(gca, 'XGrid', 'on')
else
   % Флаг выключен, следует убрать линии сетки
   set(gca, 'XGrid', 'off')
end
```

```
function varargout = chbxGridY_Callback(h, eventdata, handles, varargin)
% Проверка флага сетка по x
if get(h, 'Value')
   % Флаг включен, следует добавить линии сетки
   set(gca, 'YGrid', 'on')
else
   % Флаг выключен, следует убрать линии сетки
   set(gca, 'YGrid', 'off')
end
```

Внесенные дополнения позволяют пользователю наносить и убирать сетку по каждой координате при помощи флагов без перестроения графика функции.

Флаги предоставляют пользователю возможность выбора одной или сразу нескольких опций. Одновременный выбор *только одной* опции осуществляется при помощи переключателей.

Переключатели

Переключатели обычно группируются по их предназначению, и пользователь может выбрать только одну опцию. Всегда установлен единственный переключатель из группы. Обработка событий переключателя должна влиять на состояние остальных переключателей всей группы. Модернизируйте интерфейс приложения `mygui`, предоставьте пользователю возможность выбирать тип маркера (кружок, квадрат или отсутствие маркера).

Добавьте в окно приложения новую рамку и нанесите на нее три переключателя, установите свойствам `Tag` значения `rbMarkCirc`, `rbMarkSq`, `rbMarkNone`, а `String` — `маркеры-круги`, `маркеры-квадраты`, `без маркеров`, соответственно (рис. 12.3).

Состояние переключателя определяется его свойством `Value`, если `Value` равно единице, то переключатель включен, ноль — нет. Задайте в редакторе свойств значение единица свойству `Value` переключателя с надписью **без маркеров**, он будет включен при запуске программы. Значение свойства `Value` в версии MatLab 6.*x* устанавливается следующим образом. Выделите переключатель и перейдите к его свойствам. В редакторе свойств нажмите кнопку в строке с `Value`. Появляется окно **Value**, изображенное на рис. 12.4.

Выделите при помощи мыши строку со значением 0.0 и перейдите в режим редактирования значения двойным щелчком мыши. Измените 0.0 на единицу и нажмите **OK**. Обратите внимание, что в редакторе свойств значение `Value` изменилось на единицу, и включился переключатель **без маркеров** на

окне приложения в редакторе приложений. Вышеописанным образом устанавливаются значения `Value` в редакторе свойств. Дальнейшее управление значением `Value` переключателей должно осуществляться программно в ходе работы приложения `mygui`.

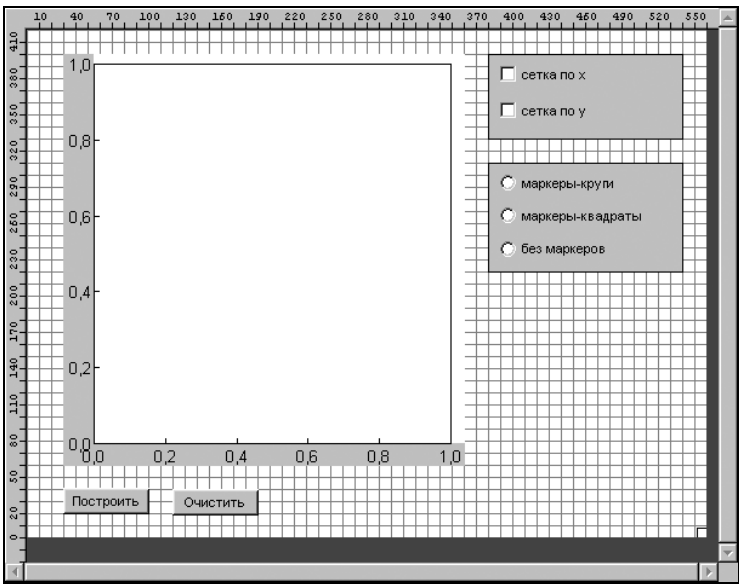


Рис. 12.3. Добавление группы переключателей

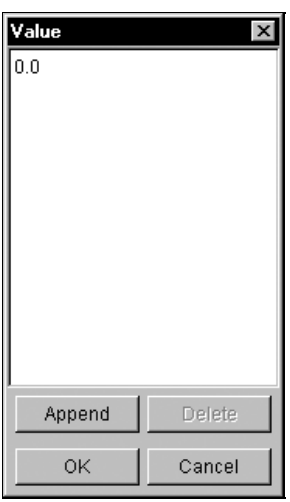


Рис. 12.4. Окно **Value** для установки значения

Предположим, что пользователь установил один из переключателей. Происходит обращение к соответствующей подфункции обработки события `Callback` переключателя, которая:

- изменяет тип маркеров линии;
- выключает два остальных переключателя.

Второе действие программируется достаточно просто — следует занести в `Value` значение ноль при помощи функции `set` и указателей на нужные переключатели, хранящихся в структуре `handles`. Изменение типа маркеров линии не представляет труда, если известен указатель на линию, то достаточно обратиться к свойству линии `Marker`. Указатель на линию возвращает `plot` в выходном аргументе, его следует записать в некоторую переменную, например `Hline`. Использовать указатель на линию придется в других подфункциях, обрабатывающих событие `Callback` переключателей. Очень важно понять, что переменная `Hline` инициализируется при вызове подфункции `btnPlot_Callback`, выполняются операторы подфункции и она *заканчивает работу*. Все переменные, определенные в подфункции, являются локальными и по окончании работы подфункции *не определены*.

Обмен данных между подфункциями проще всего осуществить при помощи структуры `handles`. Подфункция, передающая данные, должна содержать запись данных в новое поле и сохранение структуры командой `guidata`. Входной аргумент — структура `handles` всех подфункций теперь содержит добавленное поле, в которое занесено соответствующее значение. Например, в некоторой подфункции можно сохранить массив в поле `dat1` структуры `handles`

```
handles.dat1 = [1.2 3.2 0.1];
guidata(gcbo, handles)
```

а затем использовать его в другой подфункции

```
max(handles.dat1)
```

Аналогичным образом сохраняются указатели на объекты, создаваемые в пределах подфункции. Внесите необходимые изменения в подфункцию `btnPlot_Callback` и запрограммируйте обработку событий `Callback` переключателей в подфункциях `rbMarkCirc_Callback`, `rbMarkSq_Callback`, `rbMarkNone_Callback` (листинг 12.4):

- сохраните указатель на линию в поле `line` структуры `handles` при построении графика командой `plot`;
- добавьте блоки обработки событий `Callback` переключателей, каждый из которых устанавливает остальные два переключателя в положение "выключено" и наносит на линию соответствующие маркеры.

Листинг 12.4. Обработка событий переключателей в myguiprog

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
% Построение графика функции
x = [-2:0.2:2];
y = exp(-x.^2);
% Запись указателя в поле line структуры handles
handles.line = plot(x,y);
% Сохранение структуры handles для использования в других подфункциях
guidata(gcbo, handles);
...

function varargout = rbMarkCirc_Callback(h, eventdata, handles, varargin)
% Выбран переключатель маркеры-круги
set(handles.line, 'Marker', 'o') % размещение маркеров-кругов на линии
% Установка остальных переключателей в положение выключено
set(handles.rbMarkSq, 'Value', 0)
set(handles.rbMarkNone, 'Value', 0)

function varargout = rbMarkSq_Callback(h, eventdata, handles, varargin)
% Выбран переключатель маркеры-квадраты
set(handles.line, 'Marker', 's') % размещение маркеров-квадратов на линии
% Установка остальных переключателей в положение выключено
set(handles.rbMarkCirc, 'Value', 0)
set(handles.rbMarkNone, 'Value', 0)

function varargout = rbMarkNone_Callback(h, eventdata, handles, varargin)
% Выбран переключатель без маркеров
set(handles.line, 'Marker', 'none') % удаление маркеров с линии
% Установка остальных переключателей в положение выключено
set(handles.rbMarkCirc, 'Value', 0)
set(handles.rbMarkSq, 'Value', 0)
```

Запустите приложение `mygui`, отобразите график функции, нажав кнопку **Построить**, и убедитесь в том, что возможна установка только одного из переключателей и она приводит к появлению соответствующих маркеров на графике функции. Однако пока еще интерфейс `mygui` имеет ряд недостатков.

- Если пользователь использует кнопку **Очистить** для очистки осей, а затем устанавливает переключатель, то производится обращение к несущест-

вующему объекту линии (сообщение об ошибке выводится в командное окно).

- Нажатие на кнопку **Построить** приводит к получению линии без маркеров вне зависимости от установленного переключателя.
- Повторный щелчок по области переключателя приводит к его выключению, но всегда один из переключателей должен быть установлен.

Конечно, первый недостаток является существенным — приложение должно работать без ошибок! Проще всего запретить доступ к переключателям, если нет линии на графике и разрешить после ее появления. Очевидно, что следует внести изменения в соответствующие подфункции `myguiprog`, обрабатывающие события `Callback` кнопок. Нажатие на **Построить** должно открывать доступ к группе переключателей, а очистка осей кнопкой **Очистить** запрещать доступ. Итак, следует найти указатели на переключатели и установить их свойство `Enabled` в нужное значение `on` или `off`. При запуске приложения все переключатели должны быть недоступны, т. к. пользователь еще не построил график функции.

Редактор свойств позволяет одновременно установить значение общих свойств, например `Enable`, целой группы объектов. Выделите щелчком мыши один из переключателей, а остальные добавляйте в группу щелчком мыши, удерживая нажатой клавишу <Ctrl>. В результате должны быть выделены все три переключателя. Теперь перейдите в редактор свойств при помощи всплывающего меню. Вверху окна редактора свойств размещена надпись **Multiply objects selected**, означающая, что проделываемые установки произойдут для свойств сразу всех выделенных объектов. Установите `Enable` в `off`, при запуске приложения `mygui` переключатели недоступны. Осталось дополнить подфункции обработки события `Callback` кнопок **Построить** и **Очистить** для программного управления свойством `Enable`. Обратитесь к листингу 12.5, содержащему требуемые операторы.

Листинг 12.5. Разрешение и запрещение доступа к группе переключателей

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
...
% Все переключатели должны стать доступными после появления графика
set(handles.rbMarkCirc, 'Enable', 'on')
set(handles.rbMarkSq, 'Enable', 'on')
set(handles.rbMarkNone, 'Enable', 'on')
% --------------------------------------------------------------------
function varargout = btnClear_Callback(h, eventdata, handles, varargin)
...
```

```
% Все переключатели должны стать недоступными после очистки осей
set(handles.rbMarkCirc, 'Enable', 'off')
set(handles.rbMarkSq, 'Enable', 'off')
set(handles.rbMarkNone, 'Enable', 'off')
```

Первый недостаток интерфейса приложения `mygui` устранен. Теперь необходимо сделать так, чтобы при построении графика тип маркера отвечал установленному переключателю. Очевидно, что после вывода графика следует найти переключатель со значением `Value`, равным единице, и установить соответствующий тип маркера (листинг 12.6).

Листинг 12.6. Изменение маркеров при построении графика

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
...
if get(handles.rbMarkCirc, 'Value')
   % Установлен переключатель маркеры-круги
   set(handles.line, 'Marker', 'o')
end
if get(handles.rbMarkSq, 'Value')
   % Установлен переключатель маркеры-круги
   set(handles.line, 'Marker', 's')
end
```

Сохраните изменения в M-файле и запустите приложение `mygui`. Тип маркеров определяется установленным переключателем при построении графика.

Осталась нерешенной одна проблема. При повторном щелчке по области переключателя он выключается, но всегда должен быть установлен единственный переключатель. Данный недостаток устраняется с привлечением еще одного возможного значения `inactive` свойства `Enable`. Переключатель со значением `inactive` является *неактивным*, он выглядит в работающем приложении как доступный переключатель (со значением `on`), но попытка изменить состояние данного переключателя не приводит к успеху. Усовершенствуйте обработку событий согласно следующему алгоритму.

1. Свойство `Enabled` переключателя, событие которого обрабатывается, должно иметь значение `inactive`, а для остальных двух `on`. Если не задать `on` для других переключателей, то в результате все они станут неактивными.
2. При нажатии на кнопку **Построить** свойству `Enable` всех переключателей присваивается значение `on`, а затем определяется установленный в данный момент переключатель и в `Enable` заносится `inactive`.

Дополните подфункции `btnPlot_Callback`, `rbMarkCirc_Callback`, `rbMarkSq_Callback` и `rbMarkNone_Callback` необходимыми операторами (листинг 12.7).

Листинг 12.7. Предотвращение выключения переключателя повторным щелчком мыши

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
...
% Поиск установленного переключателя и определение его, как неактивного
if get(handles.rbMarkCirc, 'Value')
   set(handles.rbMarkCirc, 'Enable', 'inactive')
end
if get(handles.rbMarkSq, 'Value')
   set(handles.rbMarkSq, 'Enable', 'inactive')
end
if get(handles.rbMarkNone, 'Value')
   set(handles.rbMarkNone, 'Enable', 'inactive')
end

function varargout = rbMarkCirc_Callback(h, eventdata, handles, varargin)
...
% Переключатель, событие которого обрабатывается, должен стать
% неактивным, а остальные — активными
set(h, 'Enable', 'inactive')
set(handles.rbMarkSq, 'Enable', 'on')
set(handles.rbMarkNone, 'Enable', 'on')

function varargout = rbMarkSq_Callback(h, eventdata, handles, varargin)
...
% Переключатель, событие которого обрабатывается, должен стать
% неактивным, а остальные — активными
set(h, 'Enable', 'inactive')
set(handles.rbMarkCirc, 'Enable', 'on')
set(handles.rbMarkNone, 'Enable', 'on')

function varargout = rbMarkNone_Callback(h, eventdata, handles, varargin)
...
% Переключатель, событие которого обрабатывается, должен стать
% неактивным, а остальные — активными
```

```
set(h, 'Enable', 'inactive')
set(handles.rbMarkSq, 'Enable', 'on')
set(handles.rbMarkCirc, 'Enable', 'on')
```

Правильная обработка переключателей (см. листинги 12.4—12.7) требует учета всех ситуаций, которые могут возникнуть при взаимодействии пользователя с приложением. Раскрывающиеся списки реализуют альтернативный способ выбора пользователем только одной из предлагаемых опций.

Списки

Модернизируйте интерфейс приложения `mygui`, предоставьте пользователю возможность выбора цвета линии графика из раскрывающегося списка (синий, красный, зеленый). Перейдите в режим редактирования и добавьте при помощи панели управления раскрывающийся список (рис. 12.5). В редакторе свойств установите свойство `Tag` в значение `'pmColor'`.

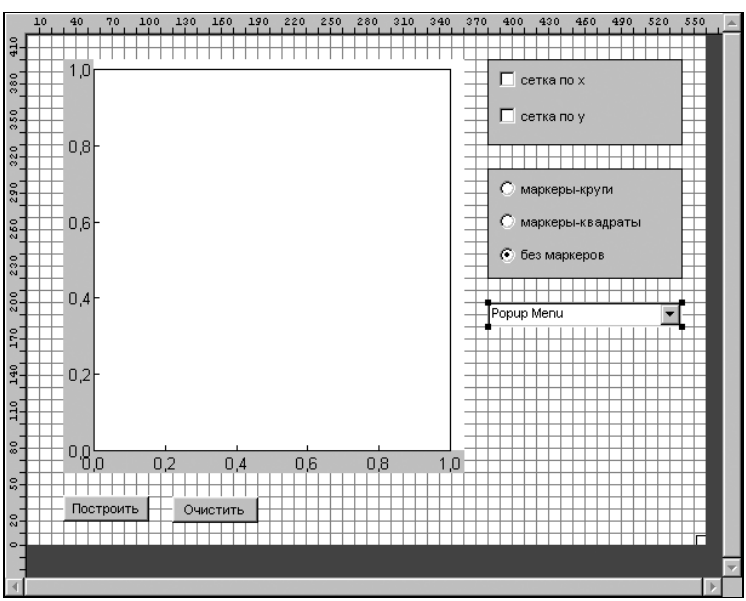


Рис. 12.5. Добавление раскрывающегося списка

Элементами раскрывающегося списка являются строки, которые вводятся в редакторе свойств. Нажмите кнопку в строке со свойством `String` раскрывающегося списка, появляется окно **String**. Наберите в нем строки "синий",

"красный", "зеленый" (без кавычек), разделяя их при помощи клавиши <Enter> (рис. 12.6).



Рис. 12.6. Окно **String**

Запустите `mygui` и убедитесь, что раскрывающийся список содержит требуемые строки. Выбор различных строк пока не приводит к изменению цвета линии — требуется запрограммировать событие `Callback` раскрывающегося списка.

Обработка события `Callback` раскрывающегося списка состоит в определении выбора пользователя и соответствующем изменении цвета линии. Свойство списка `Value` содержит номер выбранной строки (строки списка нумеруются с единицы). Перейдите к подфункции `pmColor_Callback` и запрограммируйте обработку выбора пользователя. Используйте оператор `switch` для установки цвета линии в зависимости от номера выбранной строки списка (листинг 12.8).

Листинг 12.8. Обработка выбора пользователя из раскрывающегося списка

```
function varargout = pmColor_Callback(h, eventdata, handles, varargin)
% Определение номера выбранной строки
Num = get(h, 'Value');
switch Num
case 1
   % Выбрана первая строка, следует сделать линию синей
   set(handles.line, 'Color', 'b');
```

```
case 2
   % Выбрана вторая строка, следует сделать линию красной
   set(handles.line, 'Color', 'r');
case 3
   % Выбрана третья строка, следует сделать линию зеленой
   set(handles.line, 'Color', 'g');
end
```

Запустите приложение, постройте график, нажав на кнопку **Построить**, и убедитесь в том, что раскрывающийся список позволяет изменять цвет линии графика функции. Несложно заметить, что интерфейс `mygui` имеет ряд недостатков.

- Повторное построение графика не учитывает текущий выбор цвета в раскрывающемся списке.
- Выбор цвета при отсутствии линии на графике приводит к ошибке (`handles.line` указывает на несуществующий объект).
- Рядом со списком требуется разместить текст, поясняющий назначение списка.

Устраните первый недостаток, поместите в подфункции `btnPlot_Callback` обработки нажатия кнопки **Построить** блок `switch` для задания цвета построенной линии в зависимости от выбора опции раскрывающегося списка (листинг 12.9).

Листинг 12.9. Учет выбора опции раскрывающегося списка при построении графика

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
...
% Определение номера выбранной строки
Num = get(handles.pmColor, 'Value');
% Установка требуемого цвета линии
switch Num
case 1
   set(handles.line, 'Color', 'b');
case 2
   set(handles.line, 'Color', 'r');
case 3
   set(handles.line, 'Color', 'g');
end
```

Изменение цвета линии при отсутствии графика лишено смысла, поэтому следует запретить доступ пользователя к раскрывающемуся списку, и, напротив, разрешить, при построении графика. В начале работы приложения список должен быть недоступен для пользователя. Установите в редакторе свойств для раскрывающегося списка `Enable` в `off`. Внесите необходимые дополнения в подфункции `btnPlot_Callback` и `btnClear_Callback`, соответствующие нажатию на кнопки, используйте свойство списка `Enable` (листинг 12.10).

Листинг 12.10. Разрешение и запрещение доступа к раскрывающемуся списку

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
...
% Разрешение доступа к раскрывающемуся списку
set(handles.pmColor, 'Enable', 'on')

function varargout = btnClear_Callback(h, eventdata, handles, varargin)
...
% Запрещение доступа к раскрывающемуся списку
set(handles.pmColor, 'Enable', 'off')
```

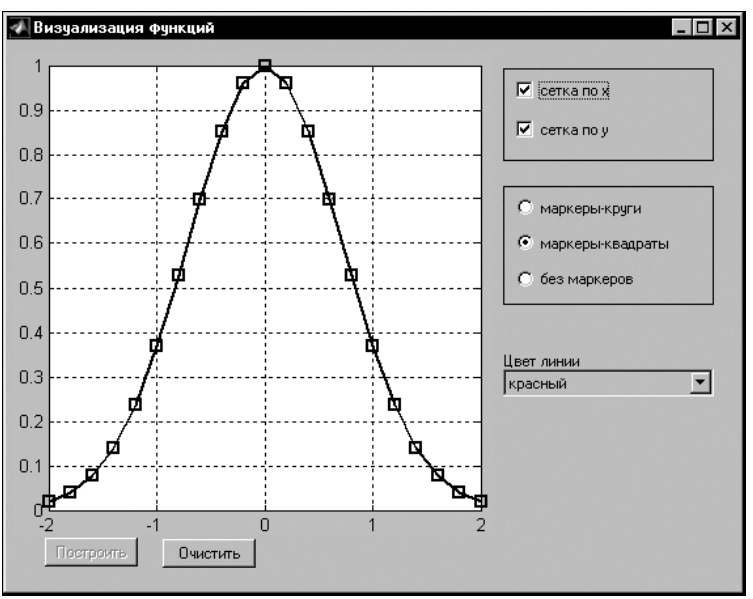


Рис. 12.7. Добавление текста

Многие элементы интерфейса, в частности раскрывающиеся списки, следует сопровождать поясняющим текстом. Перейдите в режим редактирования и при помощи панели управления разместите текстовую область над списком. Установите в редакторе свойств `String` в значение `Цвет линии`, а `HorizontalAlignment` в значение `left` для добавленного объекта, используйте кнопки в строках с названиями свойств. Теперь работающее приложение имеет более наглядный интерфейс (рис. 12.7).

Программирование событий обычных списков производится практически аналогично. Отличие состоит в том, что в обычных списках может быть выделено несколько элементов. Свойство `Value` содержит вектор номеров выбранных элементов. Разрешение выбора нескольких элементов определяется значениями свойств `Max` и `Min`. Если разность `Max` − `Min` больше единицы, то пользователь может выделить несколько строк.

Полосы скроллинга

Усовершенствуйте интерфейс приложения `mygui`, предоставив пользователю возможность устанавливать ширину линии при помощи полосы скроллинга. Добавьте полосу скроллинга в окно приложения и задайте название `scrWidth` в свойстве `Tag` полосы. Снабдите полосу скроллинга текстовым пояснением "Толщина линии" так же, как и раскрывающийся список (рис. 12.8).

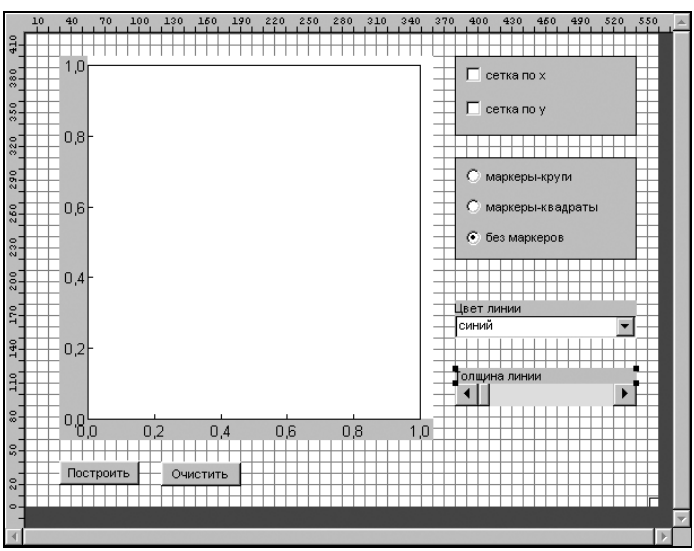


Рис. 12.8. Добавление полосы скроллинга

Замечание

Простой щелчок мышью в окне приложения при добавлении полосы скроллинга приводит к появлению вертикально расположенной полосы. Вертикальное или горизонтальное направление зависит от соотношения ширины и высоты полосы скроллинга. Измените размер для получения горизонтально расположенной полосы. Проще всего сразу нарисовать прямоугольник полосы скроллинга, удерживая нажатой левую кнопку мыши.

Теперь следует определить соответствие между положением бегунка полосы и числовым значением свойства `Value`.

Проделайте следующие установки из редактора свойств.

1. В `Max` занесите десять, а в `Min` — единицу. Свойства `Max` и `Min` полосы скроллинга отвечают за границы значений, записываемых в `Value`, при перемещении бегунка.
2. Определите начальное положение, записав в `Value` единицу. Нажмите кнопку в строке с названием свойства и в появившемся окне **Value** (см. рис. 12.4) измените значение на единицу.
3. Обратитесь к свойству `SliderStep`. Его значением является вектор из двух компонентов, первый из которых определяет относительное изменение `Value` при нажатии на кнопки со стрелками полосы скроллинга, а второй — при перетаскивании бегунка мышью. Следует установить значение [0.1 0.2] свойства `SliderStep` для того, чтобы нажатие на кнопки полосы изменяло `Value` на десять процентов, а щелчок мыши справа или слева от бегунка на двадцать. Раскройте строку `SliderStep` щелчком мыши по знаку плюс слева от названия свойства и в появившихся строках **x** и **y** введите 0.1 и 0.2 (рис. 12.9).

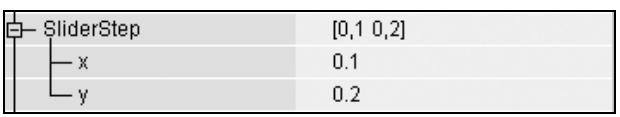


Рис. 12.9. Ввод значений `SliderStep`

Осталось запрограммировать событие `Callback` полосы скроллинга с именем `scrWidth`, которое состоит в задании ширины линии, равной округленному значению `Value`. Перейдите к подфункции `scrWidth_Callback` и добавьте в ней оператор установки ширины линии (листинг 12.11).

Листинг 12.11. Обработка события `Callback` полосы скроллинга

```
function varargout = scrWidth_Callback(h, eventdata, handles, varargin)
% Получение ширины линии в зависимости от положения бегунка
```

```
% на полосе скроллинга
width = get(h, 'Value');
% Установка толщины линии
set(handles.line, 'LineWidth', round(width))
```

Запустите `mygui` и убедитесь, что полоса скроллинга позволяет легко изменять толщину линии построенного графика. Устраните самостоятельно некоторые недочеты интерфейса. Полоса скроллинга должна быть недоступной после очистки осей кнопкой **Очистить**, построение графика при помощи кнопки **Построить** произведите с учетом установленной ширины линии. Данные недостатки исправляются внесением соответствующих изменений в обработку событий `Callback` перечисленных кнопок так же, как и в предыдущих разделах.

Область ввода текста

Обычные текстовые области, использовавшиеся на протяжении предыдущих разделов, позволяют лишь вывести некоторый текст в окно приложения. Обмен текстовой информацией между пользователем и приложением осуществляется при помощи областей ввода текста. Предоставьте пользователю возможность размещать заголовок на графике. Текст заголовка пользователь вводит в соответствующей строке.

Добавьте в окно приложения область ввода текста, установите значение `editTitle` свойству `Tag` области ввода и снабдите ее пояснением в текстовой области, расположенной выше так, как показано на рис. 12.10. В редакторе свойств удалите из `String` строку `Edit Text`, для чего нажмите кнопку в строке с названием свойства и сотрите текст в окне **String**.

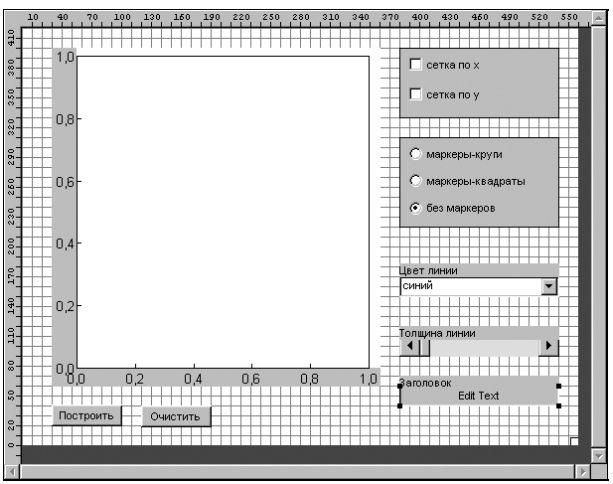


Рис. 12.10. Добавление области ввода текста

Запрограммируйте событие `Callback` области ввода, перейдите к подфункции `editTitle_Callback` и поместите операторы, которые считывают значение свойства `String` области ввода и наносят заголовок на график (листинг 12.12).

Листинг 12.12. Обработка события Callback области ввода текста

```
function varargout = editTitle_Callback(h, eventdata, handles, varargin)
   txt = get(h, 'String');
   title(txt)
```

Запустите приложение `mygui` и нанесите заголовок на график, набрав его в области ввода и нажав клавишу <Enter>. Дополните обработку события, соответствующего нажатию кнопки **Очистить**, очисткой заголовка при помощи команды `title('')`.

Вышеописанный пример демонстрирует использование области ввода, состоящей из одной строки. Разность значений свойств `Max` и `Min` области ввода определяет, позволяет ли данная область ввод многострочного текста. Если разность больше единицы, то заносимый пользователем текст записывается в массив ячеек текстовых строк, который хранится в свойстве `String`.

Глава 13



Диалоговые окна и меню приложения

Данная глава освещает использование диалоговых окон и принципы создания собственных меню и контекстных меню при разработке приложения в MatLab 5.3 и 6.*x*. Описано конструирование меню и программирование действий, выполняемых при выборе пункта меню пользователем. Изложение ведется на примере приложения `mygui`, созданию которого были посвящены *главы 10*, *11* и *12*.

Виды диалоговых окон

Удобный интерфейс приложения во многом определяется диалоговыми окнами, облегчающими работу с файлами, или предназначенными для предостережения пользователя о событиях, которые могут повлечь его действия. MatLab предоставляет разработчику приложения возможность использовать стандартные диалоговые окна Windows.

Окно подтверждения

Некоторые действия приложения требуют повторного подтверждения пользователя. Например, пользователь приложения `mygui` может случайно нажать кнопку **Очистить**, предназначенную для очистки осей. Следует вывести диалоговое окно, в котором пользователь укажет, действительно ли требуется очистить оси.

Диалоговое окно подтверждения создается функцией `questdlg`, которая в самом простом случае имеет два входных параметра — строки с текстом внутри диалогового окна и заголовком окна. Окно, создаваемое таким образом, имеет три кнопки — **Yes**, **No** и **Cancel**. Выбор пользователя возвращается в строковом выходном аргументе функции `questdlg`, его значение совпадает с надписью на кнопке.

Усовершенствуйте обработку нажатия кнопки **Очистить** так, чтобы соответствующие операторы выполнялись только в том случае, если пользователь нажал кнопку **Yes** в появляющемся диалоговом окне с текстом **Очистить оси?** и заголовком **mygui**. Используйте условный оператор `if` и функцию `strcmp` для сравнения выходного аргумента `questdlg` со строкой `Yes` (листинг 13.1).

Листинг 13.1. Программирование диалогового окна запроса

```
button = questdlg('Очистить оси?','mygui');
   if strcmp(button, 'Yes')
      % здесь размещаются все операторы,
      % обрабатывающие нажатие на кнопку Clear
   end
```

Нажатие на кнопку **Очистить** приводит к появлению диалогового окна, изображенного на рис. 13.1. Выбор пользователя определяет дальнейшие действия приложения `mygui`.

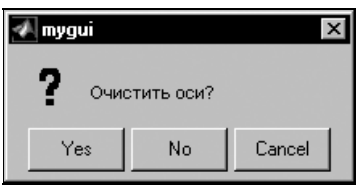


Рис. 13.1. Диалоговое окно подтверждения

Функция `questdlg` позволяет управлять видом диалогового окна. Строка с названием кнопки, переданная в третьем дополнительном аргументе, определяет кнопку окна, которая может быть нажата пользователем при помощи клавиши <Enter>. Например, вызов

```
button = questdlg('Очистить оси?','mygui', 'Yes');
```

предполагает, что в диалоговом окне нажатие клавиши <Enter> эквивалентно выбору кнопки **Yes**.

Число кнопок и надписи на них определяются создателем приложения, например, следующая форма обращения к функции `questdlg`

```
button = questdlg('Очистить оси?','mygui','Да','Нет','Нет')
```

приводит к появлению диалогового окна с текстом **Очистить оси?**, заголовком **mygui** и двумя кнопками **Да** и **Нет**, причем нажатие <Enter> заменяет выбор **Нет**.

Окна открытия файла и записи в файл

Передача данных из файла в приложение сопряжена с заданием имени и пути к файлу. Наиболее простой способ состоит в использовании диалогового окна открытия файла, которое создается функцией `uigetfile`. Данная функция приводит к появлению диалогового окна открытия файла и воз-

вращает в первом выходном аргументе имя, а во втором — путь к файлу, выбранному пользователем. Выходные аргументы равны нулю, если пользователь отменил открытие, или при открытии файла произошла ошибка.

Дополните приложение `mygui` кнопкой **Из файла**, нажатие на которую вызывает диалоговое окно открытия файла. Пользователь выбирает файл, содержащий два столбика чисел одинаковой высоты — таблично заданную функцию. Данные считываются из файла в матрицу с двумя столбцами и визуализируются при помощи `plot`. Используйте `strcat` для сцепления строк с путем к файлу и его именем для образования полного имени файла. Полное имя задается в качестве входного аргумента функции `load`, которая производит считывание данных из файла в массив. Операторы простейшей обработки нажатия кнопки **Из файла** для MatLab 5.3 приведены в листинге 13.2. При работе в MatLab 6.*x* следует записать указатель на линию в структуру `handles.line` и сохранить ее функцией `guidata`.

Обмен данными между подфункциями приложения в версии 6.x описан в разд. "Переключатели" главы 12.

Листинг 13.2. Программирование считывания данных из файла

```
[fname, pname] = uigetfile; % получение имени и пути к файлу
% Проверка, был ли открыт файл
if fname ~= 0
   % Образование полного имени файла
   fullname = strcat(pname, fname);
   % Считывание данных из файла в массив
   Mas = load(fullname);
   % Графическое отображение данных
   Hline = plot(Mas(:,1), Mas(:,2));
end
```

Создайте файл с данными в нужном формате, назовите его, например, my.dat. Запустите приложение `mygui` и нажмите на кнопку **Из файла**. Обратите внимание, что в диалоговом окне открытия файла в раскрывающемся списке **Files of type** установлен фильтр **M-files(*.m)**. Выберите **All files(*.*)** и откройте файл my.dat. На осях строится график данных. Обработку нажатия кнопки **Из файла** следует производить так же, как и **Построить** с учетом взаимодействия с остальными элементами интерфейса. После появления графика должны быть доступны переключатели, флаги, раскрывающийся список и область ввода текста. Произведите требуемую доработку самостоятельно.

Приложение, которое предназначено для работы с файлами определенного типа, например dat, проще в использовании, если в диалоговом окне открытия файла автоматически устанавливается нужный фильтр. Функция `uigetfile` предусматривает задание стандартного расширения во входном аргументе, например, вызов

```
[fname, pname] = uigetfile('*.dat');
```

соответствует диалоговому окну с фильтром файлов типа dat. Указание `'*.*'` приводит к отображению файлов всех типов в диалоговом окне открытия файла.

Функция `uiputfile` предназначена для создания диалогового окна сохранения файла. Входные и выходные аргументы `uiputfile` имеют то же назначение, что у функции `uigetfile`. В качестве входного аргумента `uiputfile` можно указать не только фильтр, но и полное имя файла, предлагаемого для записи. Разумеется, при программировании записи данных в файл следует применять `save`.

См. разд. "Считывание и запись данных" главы 2.

Функции `uiputfile` и `uigetfile` позволяют задать заголовок диалоговых окон во втором дополнительном строковом аргументе.

Окно с сообщением об ошибке

Некоторые действия пользователя, в частности открытие файла с данными в неизвестном формате, могут привести к ошибке в работе приложения. Такие исключительные ситуации следует предусматривать при написании алгоритма приложения и сопровождать их сообщением об ошибке. Лучше всего выводить сообщение в диалоговое окно, которое автоматически размещается поверх всех остальных окон и требует нажатия кнопки **OK** для продолжения работы.

Функция `errordlg` предназначена для создания диалогового окна с сообщением об ошибке. Входными аргументами `errordlg` являются строки с текстом и заголовком окна. Дополните построение графика данных, считанных из файла (см. листинг 13.2), проверкой на размерность и тип содержимого массива `Mas` при помощи функций `size`, `ndims` и `isnumeric` и выведите сообщение в случае несоответствующего формата данных. Заключите считывание и визуализацию данных в блок `try...catch` для предотвращения ошибки при обращении к `load` (листинг 13.3 для версии 5.3). При работе в MatLab 6.*x* следует записать указатель на линию в структуру `handles.line` и сохранить, используя функцию `guidata`.

Листинг 13.3. Обработка исключительных ситуаций с сообщением об ошибке

```
try
   % Считывание данных из файла в массив
   Mas = load(fullname);
   % Определение размеров массива
   SMas = size(Mas);
   % Проверка массива данных
   if (SMas(2) ~= 2) | (ndims(Mas) ~= 2) | ~isnumeric(Mas)
      errordlg('Неизвестный формат файла с данными', 'Ошибка!')
   else
      % Графическое отображение данных
       Hline = plot(Mas(:,1), Mas(:,2));
   end
catch
   % Произошла ошибка при выполнении load
   errordlg('Неизвестный формат файла с данными', 'Ошибка!')
end
```

Меню графического окна

Приложение MatLab может использовать стандартное меню графического окна. Среда GUIDE позволяет программисту дополнять стандартное меню или создать собственные меню. Свойство `MenuBar` окна приложения (объекта `figure`) отвечает за наличие стандартных меню **File**, **Edit**, **Tools**, **Window** и **Help** в работающем приложении. Значение `figure` данного свойства соответствует отображению стандартных меню, а `none` приводит к приложению без строки с меню. Вне зависимости от значения свойства `MenuBar`, разработчик приложения имеет возможность размещать собственные меню, которые в случае значения `figure` добавляются к стандартным меню графического окна. Размещение и программирование меню производится при помощи редактора меню.

Создание меню в редакторе в версии 5.3

Продолжите работу над приложением `mygui`, начатую при чтении *главы 11*. Перейдите в режим редактирования приложения в среде GUIDE. Принцип конструирования меню проще всего понять, создавая новое меню — убедитесь, что свойство `MenuBar` графического окна установлено в `off`. Запустите редактор меню из панели управления (см. рис. 10.5), появляется окно **Guide Menu Editor**, изображенное на рис. 13.2. Окно редактора содержит кнопки

New Menu и **New Context Menu**, предназначенные для создания меню окна приложения и контекстных меню графических объектов. Навигатор меню вместе с кнопками со стрелками позволяет разработчику интерактивно управлять порядком расположения пунктов меню и вложенностью подпунктов. Строки ввода **Label**, **Tag** и **Callback** служат для определения надписи пунктов меню, задания их имен или тегов для идентификации в программе и программирования события `CallBack`, возникающего при выборе пункта меню пользователем. Обратите внимание, что все проделанные изменения в редакторе меню отображаются только в *запущенном* приложении. Интерактивное упорядочение элементов меню доступно также при работающем приложении.

Программирование событий `Callback` лучше всего реализовывать в блоках `case` файл-функции, содержащей обработку событий всех элементов интерфейса (для приложения `mygui` такой файл-функцией является `myguiprog`). Предлагаемый подход позволяет осуществить взаимодействие пунктов меню, имеющихся элементов интерфейса и объектов, создаваемых приложением.

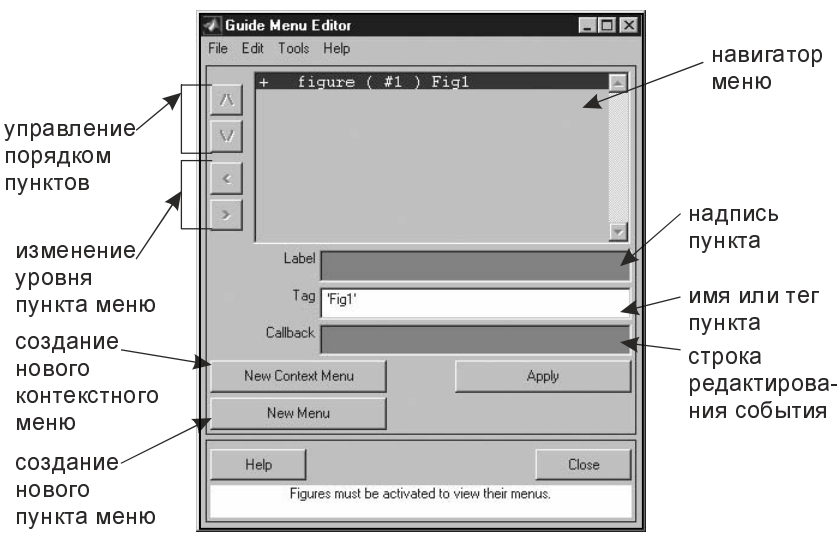


Рис. 13.2. Редактор меню **Guide Menu Editor** в версии 5.3

Добавьте новое меню при помощи кнопки **New Menu**, установите текст "График" в качестве надписи меню в строке **Label**, дайте ему имя `mnGraph` в строке **Tag** (имя и надпись заключите в апострофы), нажмите кнопку **Apply** и запустите приложение из панели управления. Окно приложения `mygui` теперь содержит строку с единственным меню **График** (рис. 13.3).

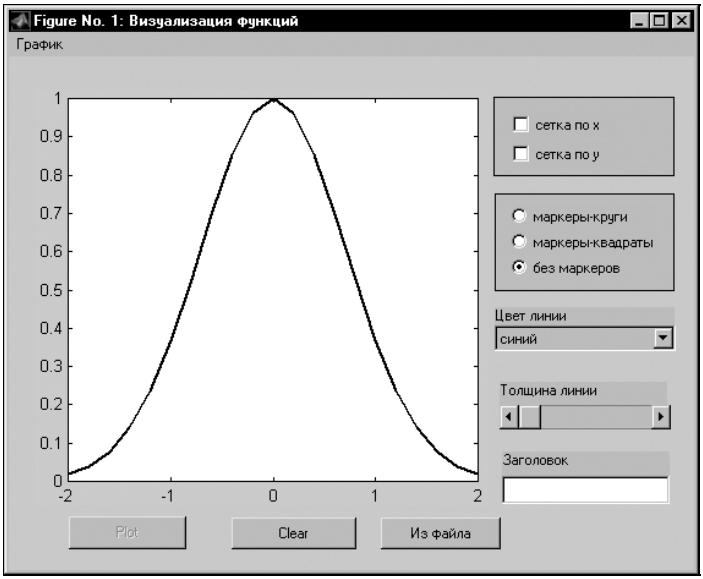


Рис. 13.3. Окно приложения с пунктом меню

Разумеется, выбор меню **График** не приводит к раскрытию меню, требуется теперь создать пункты меню. Перейдите в режим редактирования приложения, сделайте строку с меню **График** текущей в навигаторе меню и нажмите **New Menu**. В меню **График** добавился новый пункт, вложенный в меню. Задайте надпись **Построить** и имя `mnGraphPlot`. Сделайте снова строку с меню **График** текущей и добавьте еще один пункт меню с надписью **Очистить** и именем `mnGraphClear`. Иерархия созданных элементов меню, отображаемая в навигаторе объектов, приведена на рис. 13.4.

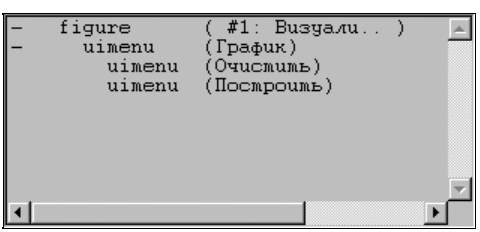


Рис. 13.4. Иерархия элементов меню

Замечание

Если в навигаторе какое-либо меню является текущим, то нажатие кнопки **New Menu** приводит к образованию пункта данного меню. Если текущим является пункт, то добавляется подпункт, т. е. создаваемый элемент меню имеет уро-

вень, на единицу меньше текущего. В том случае, когда требуется разместить меню самого верхнего уровня, следует выделить в навигаторе меню строку `figure`. Удаление пункта меню производится при помощи нажатия комбинации клавиш <Ctrl>+<x>.

Запустите приложение `mygui` и убедитесь в том, что меню **График** содержит пункты **Построить** и **Очистить**. Выбор пунктов меню не приводит к соответствующим действиям приложения. Следует теперь запрограммировать события `Callback` пунктов меню.

Создание меню в редакторе в версии 6.*x*

Продолжите работу над приложением `mygui`, начатую при чтении *главы 12*. Перейдите в режим редактирования приложения в среде GUIDE. Принцип конструирования меню проще всего понять, создавая новое меню — убедитесь, что свойство `MenuBar` графического окна установлено в `off`. Запустите редактор меню из панели управления (см. рис. 10.11), появляется окно **Menu Editor**, изображенное на рис. 13.5.

Окно редактора меню содержит две вкладки: **Menu Bar**, предназначенную для создания строки меню приложения, и **Context Menus** для контекстного меню. Области навигатора и свойств элементов меню пока пустые. Создайте меню, нажав соответствующую кнопку на панели инструментов редактора меню (убедитесь, что выбрана вкладка **Menu Bar**), в навигаторе появилась строка **Untitled 1**, сделайте ее текущей щелчком мыши. Обратите внимание (рис. 13.6), что в области свойств находятся строки ввода.

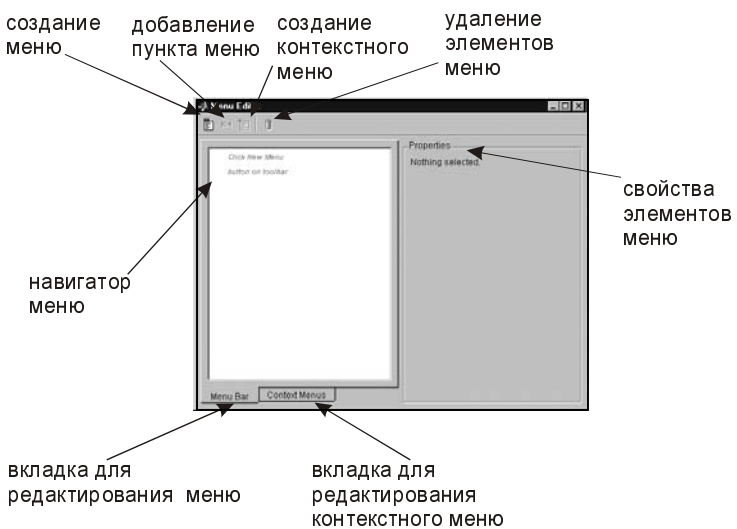


Рис. 13.5. Редактор меню **Guide Menu Editor** в версии 6.*x*

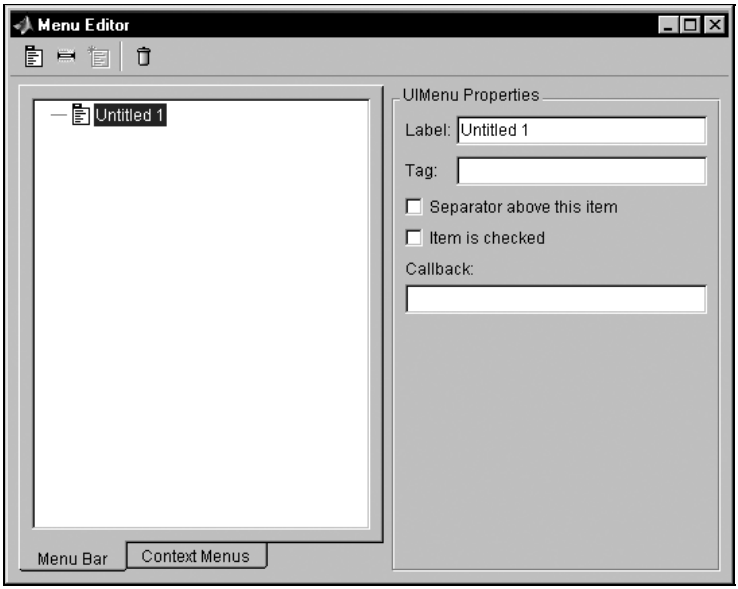


Рис. 13.6. Задание свойств меню в редакторе

Строка **Label** служит для задания надписи меню или пункта меню, а **Tag** — для определения названия созданного объекта, об использовании **Callback** сказано ниже. Введите текст "График" в строку **Label** (без кавычек) и задайте имя `mnGraph` — меню построить. Запустите приложение `mygui` и убедитесь в наличии меню **График**. Выбор меню **График** в работающем приложении не приводит к раскрытию меню, следует создать пункты меню. Перейдите в режим редактирования, сделайте текущей строку **График** в навигаторе редактора меню и добавьте пункт, нажав соответствующую кнопку (см. рис. 13.5). Установите надпись пункта **Построить** и дайте ему имя `mnGraphPlot`. Добавьте еще один пункт меню, сделав предварительно текущей строку **График** в навигаторе. Аналогичным образом задайте надпись **Очистить** и имя `mnGraphClear`. Навигатор меню должен содержать структуру, изображенную на рис. 13.7. Меню **График** имеет первый уровень, а пункты **Построить**, **Очистить** — второй.

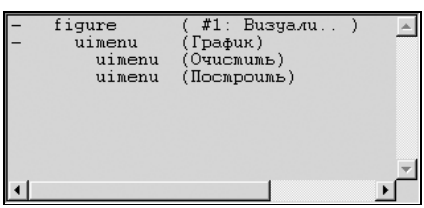


Рис. 13.7. Иерархия элементов меню

Запустите приложение mygui, выбор меню **График** приводит к раскрытию меню. Разумеется, при обращении к пунктам **Построить** и **Очистить** ничего не происходит, следует запрограммировать события CallBback пунктов меню. Событие Callback самого меню **График** не требует обработки, т. к. происходит автоматическое раскрытие меню.

Программирование пунктов меню в версии 5.3

Перейдите в режим редактирования и в редакторе меню занесите в строку **Callback** вызовы функции myguiprog с соответствующими аргументами: для пункта **Построить** установите myguiprog('selmnGraphPlot'), а для **Очистить** — myguiprog('selmnGraphClear'). Наберите вызовы функций в апострофах и закончите набор нажатием кнопки **Apply**, обратите внимание, что аргументы функции теперь заключены в двойные апострофы, заменять их на одинарные не следует. Если символ строки является апострофом, то он задается двойным апострофом с целью отличия его от апострофа, завершающего строку. Снабдите файл-функцию myguiprog соответствующими блоками case обработки событий Callback пунктов меню (листинг 13.4).

Листинг 13.4. Обработка событий Callback пунктов меню Построить и Очистить

```
case 'selmnGraphPlot'
   x = [-2:0.2:2];
   y = exp(-x.^2);
   Hline = plot(x,y);
case 'selmnGraphClear'
   cla
```

Сохраните изменения в файл-функции myguiprog и запустите приложение. Теперь выбор пунктов **Построить** и **Очистить** меню **График** приводит, соответственно, к отображению графика и очистке осей.

Дополните меню **График** пунктом **Из файла**, выбор которого должен приводить к появлению диалогового окна открытия файла с визуализируемыми данными.

Применение диалоговых окон описано в разд. "Виды диалоговых окон" данной главы.

Следующий этап состоит в обеспечении согласованной работы всех элементов управления, включая меню. В начале работы приложения являются доступными кнопки **Построить** и **Из файла** и одноименные пункты меню **График**; кнопка и пункт меню **Очистить** и элементы интерфейса, отвечающие

за оформление графика, должны быть недоступны. После построения графика становятся доступными элементы управления, связанные с установкой свойств линии и осей, а также кнопка и пункт меню **Очистить**. Взаимодействие всех элементов интерфейса, включая меню, производится с использованием их свойства `Enable`. Произведите соответствующую доработку приложения `mygui`.

См. разд. "Установка свойств объектов, функция findobj" главы 11.

Программирование пунктов меню в версии 6.*x*

Выбор элемента меню в навигаторе редактора меню приводит к отображению его свойств на панели **Properties**. Строка ввода **Callback** предназначена для вызова подфункции M-файла приложения, содержащего обработку событий элементов интерфейса. Программирование события `Callback` элементов меню происходит вручную, в отличие от других элементов интерфейса. Обработка `Callback`, например кнопок или флагов, приводила к автоматическому созданию соответствующей подфункции в M-файле, которую затем следовало наполнить содержимым — операторами, производящими требуемые действия. Имя подфункции образовывается из имени или тега объекта, знака подчеркивания и слова `Callback`, входные и выходные аргументы имеют одинаковое назначение и порядок у всех подфункций обработки события.

Редактор меню MatLab 6.*x* не предполагает автоматического создания заготовки подфункции для обработки события `Callback` элементов меню. Программист должен сам дописать соответствующую подфункцию в M-файл приложения и занести вызов данной подфункции в строку ввода **Callback** редактора.

Откройте файл mygui.m в редакторе M-файлов и определите две подфункции в соответствии с листингом 13.5 (три точки используются для переноса длинной строки в редакторе M-файлов, если строка с заголовком или командой помещается в окне редактора, то можно набирать без переноса).

Листинг 13.5. Программирование события `Callback` пунктов меню

```
function varargout = mnGraphPlot_Callback(h, eventdata, handles,...
varargin)
% Обработка выбора пункта Построить меню График
% Построение графика функции
x = [-2:0.2:2];
y = exp(-x.^2);
% Сохранение указателя на линию
```

```
handles.line = plot(x,y);
guidata(gcbo, handles);
% --------------------------------------------------------------------
function varargout = mnGraphClear_Callback(h, eventdata, handles,...
varargin)
% Обработка выбора пункта Очистить меню График
cla % очистка осей
```

Установите в редакторе меню вызовы функций `mnGraphPlot_Callback` и `mnGraphClear_Callback`. Сделайте пункт **Построить** текущим в навигаторе и поместите в строке **Callback**

```
mygui('mnGraphPlot_Callback',gcbo,[],guidata(gcbo))
```

Аналогично, для пункта **Очистить**

```
mygui('mnGraphClear_Callback',gcbo,[],guidata(gcbo)).
```

Замечание

MatLab 6.*x* использует вышеописанный вариант вызова для событий `Callback` всех объектов. Посмотрите в редакторе свойств значение свойства `Callback` раскрывающегося списка **Цвет линии** приложения `mygui`. Свойство `Callback` имеет значение `mygui('pmColor_Callback',gcbo,[],guidata(gcbo))`.

Запустите приложение `mygui`, выбор пунктов **Построить** и **Очистить** меню **График** приводит к отображению графика функции и, соответственно, очистке осей. Внесите дополнения в подфункции (листинг 13.6) для обеспечения согласованной работы пунктов меню и кнопок окна приложения. Действия, происходящие после выбора пользователем пункта **Построить**, должны быть эквивалентны тому, что происходит после нажатия на одноименную кнопку, т. е. следует запретить доступ к кнопке **Построить**, а кнопка **Очистить**, напротив, должна стать доступной. Пункты меню **График** должны взаимодействовать между собой так же, как и перечисленные кнопки. Для этого используйте свойство `Enable` пунктов меню, указатели на них содержатся в структуре `handles`.

См. разд. "Программное изменение свойств" главы 12.

Листинг 13.6 содержит операторы, которые делают активным пункт **Построить** меню **График** после отображения графика и активизируют **Очистить** и наоборот.

Листинг 13.6. Согласованное программирование пунктов меню

```
function varargout = mnGraphPlot_Callback(h, eventdata, handles,...
    varargin)
```

```
% Обработка выбора пункта Построить меню График
...
% Запрет доступа к пункту Построить
set(h,'Enable', 'off')
% Разрешение доступа к пункту Очистить
set(handles.mnGraphClear, 'Enable', 'on')
% --------------------------------------------------------------------
function varargout = mnGraphClear_Callback(h, eventdata, handles,...
    varargin)
% Обработка выбора пункта Очистить меню График
...
% Запрет доступа к пункту Очистить
set(h,'Enable', 'off')
% Разрешение доступа к пункту Построить
set(handles.mnGraphPlot, 'Enable', 'on')
```

После запуска `mygui` доступными являются оба пункта меню **График**, но пункт **Очистить** должен быть недоступен. Свойства элементов меню не удастся задать в редакторе свойств. Возникает вопрос, где в файле mygui.m размещаются операторы, инициализирующие приложение сразу после его запуска командой `mygui`. Команда `mygui` приводит к выполнению *основной функции* с одноименным названием, которая содержится в файле mygui.m.

См. разд. "Подфункции" главы 8.

Основная функция (листинг 13.7) проверяет число входных аргументов, с которыми была вызвана `mygui`. Если входные аргументы не были заданы, то происходит вызов функции `openfig`, отображающей окно приложения на экране и возвращающей указатель на окно приложения. Далее устанавливается стандартная цветовая схема окна приложения. Создание и сохранение структуры `handles` с указателями на объекты приложения происходит при помощи `guihandles` и `guidata`. После выполнения данных операторов можно использовать указатели для установки требуемых значений свойств объектов. Измените значение `Enable` пункта **Очистить** на `off` для того, чтобы после запуска данный пункт был недоступен. Вызов функции `mygui` с выходным аргументом приводит к записи в него указателя на окно приложения.

Вызов функции `mygui` со строкой в первом аргументе является событием от объекта приложения, пример такого вызова приведен выше при рассмотрении обработки события `Callback` пунктов меню. В результате такого вызова происходит переход к соответствующей подфункции с использованием функции `feval` в блоке обработки исключительных ситуаций `try...catch`.

Использование `feval` *описано в разд. "Функции от функций" главы 8, а конструкции* `try...catch` *посвящен разд. "Обработка исключительных ситуаций, try...catch" главы 7.*

Листинг 13.7. Основная функция M-файла приложения

```
function varargout = mygui(varargin)
if nargin == 0  % Запуск приложения, если нет входных аргументов
       fig = openfig(mfilename,'reuse'); % Отображение окна приложения
       % Установка схемы цвета для окна приложения
       set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
       % Создание и сохранение структуры с указателями на объекты
       handles = guihandles(fig);
       guidata(fig, handles);
       % ********************************************************
       %   Здесь программист размещает операторы, выполняемые
       %   при запуске приложения
       set(handles.mnGraphClear, 'Enable', 'off')
       % ********************************************************
       % Если mygui вызвана с выходным аргументом, то в него
       % заносится указатель на окно приложения
       if nargout > 0
              varargout{1} = fig;
       end
elseif ischar(varargin{1})
% Если первый входной аргумент строка, то происходит выполнение
% соответствующей подфункции обработки события от объекта
       try
              [varargout{1:nargout}] = feval(varargin{:})
       catch
              disp(lasterr);
       end
end
```

Сохраните проделанные изменения в редакторе M-файлов и запустите приложение `mygui`. Пункт **Очистить** становится доступным только после выбора пункта **Построить**.

Флаги состояния и разделительные линии

Некоторые пункты меню могут находиться в одном из положений — включено или выключено, о чем свидетельствует флаг рядом с надписью пункта. Например, при создании меню для управления сеткой на графике с двумя пунктами, каждый из которых наносит или убирает сетку по выбранной координате, следует снабдить пункты меню флагами. Кроме того, между пунктами меню можно помещать разделительную линию для удобства использования меню с большим числом пунктов.

Пункты меню с флагами состояния

Добавьте меню **Сетка** с пунктами **Сетка по x**, **Сетка по y** в режиме редактирования приложения. Меню **Сетка** и **График** должны быть одного уровня, поэтому при работе в версии 5.3 следует сначала сделать строку **figure** текущей в навигаторе меню, а затем нажать **New Menu** с целью появления заготовки для меню **Сетка**. В версии 6.*x* достаточно просто нажать кнопку создания нового меню (см. рис. 13.5)

Дальнейшие действия аналогичны созданию меню **График**. В результате навигатор меню в версии 5.3 должен выглядеть так, как показано на рис. 13.8. Вид навигатора объектов в MatLab 6.*x* приведен на рис. 13.9.

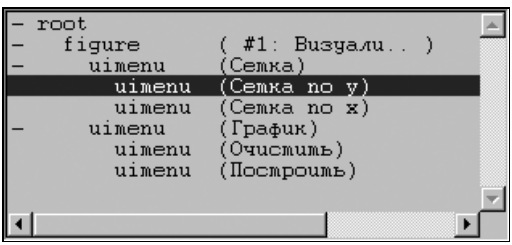


Рис. 13.8. Иерархия элементов меню **Сетка** и **График** в навигаторе меню (версия 5.3)

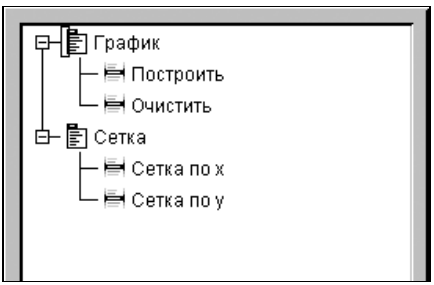


Рис. 13.9. Иерархия элементов меню **График** и **Сетка** (версия 6.*x*)

Дайте новым элементам имена `mnGrid`, `mnGridX` и `mnGridY` соответственно. Теперь следует запрограммировать события `Callback` пунктов меню так, чтобы, например, при выборе пункта **Сетка по x** включался флаг и наносились линии сетки, а при повторном обращении к данному пункту убиралась сетка и выключался флаг. Свойство пункта меню `Checked` отвечает за наличие флага рядом с названием пункта меню, значение `on` приводит к включению флага, а `off` — к выключению. Алгоритм достаточно простой, следует проверить текущее состояние флага, которое определяет предыдущий выбор пользователя, перевести флаг в противоположное положение и нанести или скрыть сетку (листинг 13.8 для версии 5.3)

Листинг 13.8. Программирование пунктов меню с флагами в версии 5.3

```
case 'selmnGridX'
   % Выбран пункт Сетка по x меню Сетка
   if strcmp(get(gcbo, 'Checked'), 'on')
      % Флаг пункта Сетка по x был установлен,
      % теперь следует выключить его и убрать сетку
      set(gcbo, 'Checked', 'off') % выключение флага
      set(gca, 'XGrid', 'off') % скрытие сетки
   else
      % Флаг пункта Сетка по x не был установлен,
      % теперь следует включить его и нанести сетку
      set(gca, 'XGrid', 'on') % включение флага
      set(gcbo, 'Checked', 'on') % нанесение сетки
   end
case 'selmnGridY'
   % Выбран пункт Сетка по y меню Сетка
   if strcmp(get(gcbo, 'Checked'), 'on')
      % Флаг пункта Сетка по y был установлен,
      % теперь следует выключить его и убрать сетку
      set(gcbo, 'Checked', 'off') % выключение флага
      set(gca, 'YGrid', 'off') % скрытие сетки
   else
      % Флаг пункта Сетка по y не был установлен,
      % теперь следует включить его и нанести сетку
      set(gca, 'YGrid', 'on') % включение флага
      set(gcbo, 'Checked', 'on') % нанесение сетки
   end
```

При работе в MatLab 6.*x* следует создать соответствующие подфункции (листинг 13.9) обработки события `Callback` пунктов **Сетка по x**, **Сетка по y** и занести их вызов в строку **Callback** редактора меню.

Листинг 13.9. Программирование пунктов меню с флагами в версии 6.*x*

```
function varargout = mnGridX_Callback(h, eventdata, handles,...
    varargin)
   if strcmp(get(gcbo, 'Checked'), 'on')
      % Флаг пункта Сетка по x был установлен,
      % теперь следует выключить его и убрать сетку
      set(h, 'Checked', 'off') % выключение флага
      set(gca, 'XGrid', 'off') % скрытие сетки
   else
      % Флаг пункта Сетка по x не был установлен,
      % теперь следует включить его и нанести сетку
      set(gca, 'XGrid', 'on') % включение флага
      set(h, 'Checked', 'on') % нанесение сетки
   end
function varargout = mnGridY_Callback(h, eventdata, handles,...
    varargin)
   % Выбран пункт Сетка по y меню Сетка
   if strcmp(get(gcbo, 'Checked'), 'on')
      % Флаг пункта Сетка по y был установлен,
      % теперь следует выключить его и убрать сетку
      set(h, 'Checked', 'off') % выключение флага
      set(gca, 'YGrid', 'off') % скрытие сетки
   else
      % Флаг пункта Сетка по y не был установлен,
      % теперь следует включить его и нанести сетку
      set(gca, 'YGrid', 'on') % включение флага
      set(h, 'Checked', 'on') % нанесение сетки
   end
```

Если некоторые пункты меню должны быть включены сразу после запуска приложения, то следует установить значение `on` свойству `Checked` в редакторе свойств при работе в версии 5.3 или воспользоваться флагом **Item is checked** в редакторе меню MatLab 6.*x*.

Добавьте самостоятельно операторы, которые обеспечивают согласованную работу пунктов меню **Сетка** и одноименных флагов окна приложения.

Разделительные линии

Разделительные полосы между пунктами меню применяются для выделения групп пунктов, объединенных по смыслу (например, меню **File** рабочей среды MatLab). Разделительную полосу можно добавить как при создании пункта меню, так и в готовом меню приложения. В версии 5.3 следует установить значение `on` свойству `Separator` того пункта меню, *над которым* следует разместить разделительную линию. Редактор меню MatLab 6.*x* позволяет определить положение линии при помощи флага **Separator above this item**. Линия размещается над текущим элементом меню.

Упорядочение меню в версии 5.3

Удобная организация меню очень часто выявляется только в ходе работы над приложением. Возникает необходимость в перегруппировке пунктов меню и изменении их уровня. Кнопки, расположенные слева от навигатора меню в редакторе версии 5.3 (см. рис. 13.2), предназначены для упорядочения меню. Верхние две кнопки служат для перестановки пунктов меню одного уровня, а нижние — для повышения или понижения уровня пункта меню. Отступ в навигаторе меню определяется уровнем пункта меню, например меню **График** и **Сетка** имеют первый уровень, а пункты **Построить**, **Очистить**, **Сетка по x** и **Сетка по y** — второй (см. рис. 13.8). Предположим, что требуется включить в меню **График** приложения `mygui` подменю **Сетка** вместе с пунктами **Сетка по x** и **Сетка по y**, а пункт **Очистить** выделить в отдельное меню. Процесс переработки меню можно разбить на три этапа:

1. Всем элементам меню присваивается первый уровень.
2. Элементы упорядочиваются требуемым образом.
3. Устанавливается нужный уровень каждому элементу меню.

Редактор меню позволяет производить изменения в работающем приложении и наблюдать за результатом. Запустите приложение `mygui` из панели управления и приведите все пункты меню второго уровня к первому, для чего последовательно выделяйте их в навигаторе меню и нажимайте кнопку со стрелкой влево. Строка меню *работающего* приложения `mygui` теперь выглядит так, как показано на рис. 13.10, ей соответствует вид навигатора меню, изображенный на рис. 13.11.

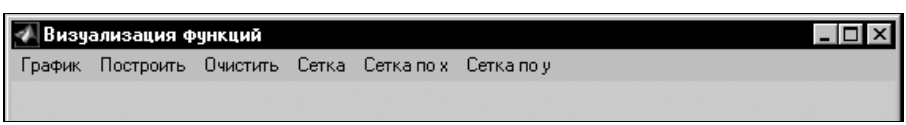


Рис. 13.10. Строка меню (все элементы имеют первый уровень)

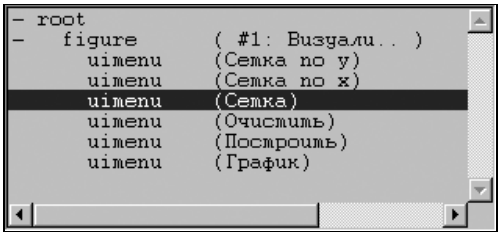


Рис. 13.11. Навигатор меню
(все элементы имеют первый уровень)

Обратите внимание, что самому левому меню **График** в окне приложения соответствует нижнее положение в навигаторе меню. Теперь следует упорядочить все меню одинакового уровня так, чтобы последующее изменение уровней привело бы к требуемой организации меню. Для изменения положения элемента меню на одну строку вверх в навигаторе следует сделать элемент, *расположенный ниже него*, текущим и нажать кнопку со стрелкой вверх (обратите внимание на перемену мест элементов). Следите за тем, чтобы данная операция была корректной, т. е. поднимаемый элемент не должен быть первым в списке. Аналогичным образом происходит перемещение элемента, находящегося под выделенным, вниз по списку при помощи кнопки со стрелкой вниз. Расположите все элементы меню так, как показано на рис. 13.12.

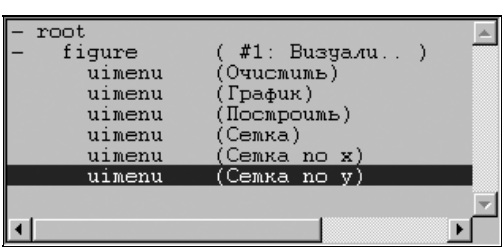


Рис. 13.12. Расположение элементов
перед изменением уровней

Понизьте уровень меню **Сетка по x** и **Сетка по y**, начиная с самого нижнего в списке (рис. 13.13).

Меню **Сетка** развернуто, о чем свидетельствует знак минус в навигаторе меню. Сверните его при помощи двойного щелчка мыши по соответствующей строке в навигаторе. Сейчас видны только меню первого уровня (рис. 13.14), знак плюс означает, что меню содержит пункты или подменю.

Завершающий этап состоит в задании уровней меню **Сетка** и **Построить**, которые должны стать пунктами меню **График**. Установите одинаковый вто-

рой уровень при помощи кнопок со стрелками вправо и влево (рис. 13.15). Пункты **Сетка по x** и **Сетка по y** в результате получили третий уровень, в чем легко убедиться, раскрыв строку с подменю **Сетка**.

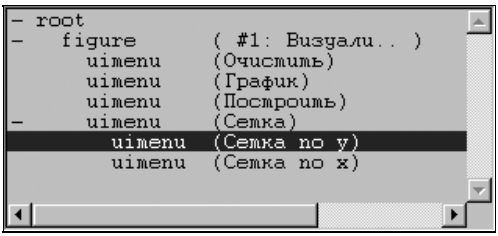


Рис. 13.13. Понижение уровня пунктов **Сетка по x** и **Сетка по y**

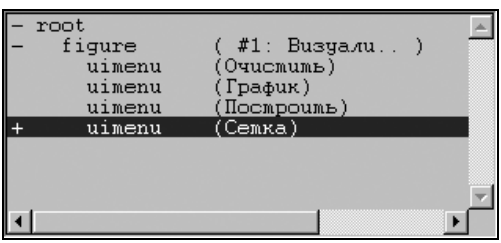


Рис. 13.14. Свернутое меню **Сетка**

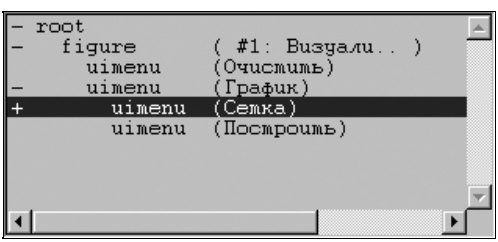


Рис. 13.15. Окончательный вид навигатора меню

Меню приложения `mygui` приобрело требуемую структуру (рис. 13.16).

Модификация структуры меню приложения выполняется в редакторе меню достаточно просто. Следует иметь в виду, что для изменения порядка пунктов меню должно быть не менее трех пунктов одного уровня. Если пунктов только два, например **График** и **Очистить**, как в вышеописанном примере, то можно временно добавить вспомогательный пункт того же уровня, произвести перестановки и затем удалить его.

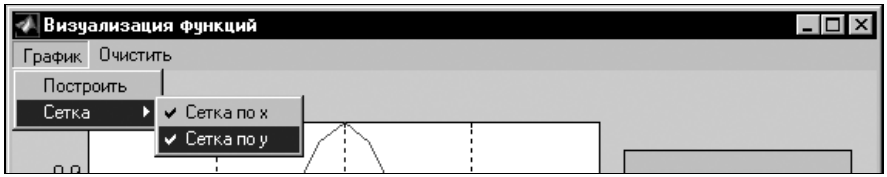


Рис. 13.16. Перестроенное меню приложения `mygui`

Контекстное меню объектов

Объекты, в том числе и созданные в ходе работы приложения, могут иметь собственное контекстное меню, которое активизируется щелчком левой кнопки мыши. Контекстное меню позволяет получить быстрый доступ к часто используемым свойствам объекта. Конструирование контекстного меню состоит в создании его в редакторе меню, определении событий `Callback` пунктов меню и последующем связывании меню с объектом. Данный раздел освещает вопрос создания контекстного меню выбора цвета линии графика на примере приложения `mygui`.

Создание меню в версии 5.3

Выделите строку `figure` в навигаторе меню и нажмите кнопку **New Context Menu**, в навигаторе появляется строка `uicontextmenu`. Задайте имя `cmLine` меню. Теперь следует определить пункты меню. Добавьте последовательно три пункта при помощи кнопки **New Menu**. Перед добавлением нового пункта делайте текущей строку с контекстным меню для того, чтобы все пункты имели одинаковый второй уровень. Если просто нажимать **New Menu**, то каждый новый пункт меню становится подпунктом предыдущего, всегда можно привести пункты меню к одному уровню при помощи кнопок со стрелками вправо и влево. Содержимое навигатора объектов должно в результате соответствовать рис. 13.17.

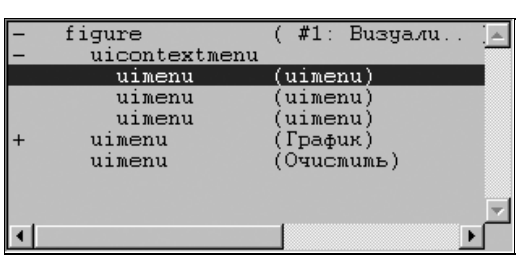


Рис. 13.17. Отображение контекстного меню в навигаторе объектов (версия 5.3)

Установите пунктам меню в строке **Tag** имена `cmLineBlue`, `cmLineRed`, `cmLineGreen` и в строке **Label**, соответственно, надписи `синий`, `красный`, `зеленый`. Обратите внимание, что при запуске приложения щелчок правой кнопкой мыши по линии графика *не приводит* к отображению контекстного меню. Сейчас контекстное меню `cmLine` присутствует в приложении как объект, но другой объект — линия, создаваемая при нажатии, например, на **Построить**, "не знает" о том, что у нее есть собственное контекстное меню. Следующий этап состоит в связывании линии с созданным меню `cmLine`.

Создание меню в версии 6.*x*

Перейдите к вкладке **Context Menus** в редакторе меню и нажмите кнопку создания контекстного меню (см. рис. 13.5), в навигаторе меню появляется строка для меню. Задайте ему имя `cmLine`. Обратите внимание, что на панели свойств нет строки ввода **Label**, т. к. раскрывающееся меню не должно иметь надписи. Создайте три пункта меню при помощи той же кнопки, что применяется для добавления пунктов меню окна приложения. Определите для них надписи `синий`, `красный`, `зеленый` и имена `cmLineBlue`, `cmLineRed`, `cmLineGreen` соответственно. В результате навигатор меню должен содержать структуру, приведенную на рис. 13.18.

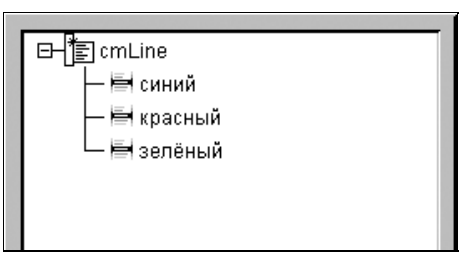


Рис. 13.18. Отображение контекстного меню в навигаторе объектов (версия 6.*x*)

В работающем приложении щелчок правой кнопкой мыши по линии графика *не приводит* к отображению контекстного меню. Сейчас контекстное меню `cmLine` присутствует в приложении как объект, но другой объект — линия, создаваемая при нажатии, например, на **Построить**, "не знает" о том, что у нее есть собственное контекстное меню. Следующий этап состоит в связывании линии с созданным меню `cmLine`.

Связывание контекстного меню с объектом

Любой объект, размещенный в окне приложения, имеет свойство `UIContextMenu`, значением которого может являться указатель на имеющееся

контекстное меню. Для того чтобы созданный объект, т. е. линия графика, обладал контекстным меню, следует установить свойству `UIContextMenu` значение указателя на меню `cmLine`, найденное при помощи функции `findobj` (в версии 5.3) или содержащееся в структуре `hadles` (в версии 6.*x*). Построение линии в приложении `mygui` производится или при нажатии пользователем кнопок **Построить**, **Из файла**, либо при выборе пункта **Построить** меню **График**. Присвойте свойству линии `UIContextMenu` требуемое значение во всех блоках M-файла, связанного с `mygui`, которые отвечают за отображение графика функции (листинг 13.10 для версии 5.3 и листинг 13.11 для версии 6.*x*).

Листинг 13.10. Связывание контекстного меню с линией в версии 5.3

```
switch event
case 'pressPlot'
% Пользователь нажал кнопку Построить
   ...
   Hline = plot(x,y);
   HcmLine = findobj('Tag', 'cmLine');
   set(Hline, 'UIContextMenu', HcmLine);
   ...
case 'pressFromFile'
% Пользователь нажал кнопку Из файла
   ...
         if (SMas(2) ~= 2) | (ndims(Mas) ~= 2) | ~isnumeric(Mas)
             errordlg('Неизвестный формат файла с данными', 'Ошибка!')
         else
             % Графическое отображение данных
             Hline = plot(Mas(:,1), Mas(:,2));
             HcmLine = findobj('Tag', 'cmLine');
             set(Hline, 'UIContextMenu', HcmLine);
         end
   ...
case 'selmnGraphPlot'
% Пользователь выбрал пункт Построить меню График
   ...
   Hline = plot(x,y);
   HcmLine = findobj('Tag', 'cmLine');
   set(Hline, 'UIContextMenu', HcmLine);
```

Листинг 13.11. Связывание контекстного меню с линией в версии 6.*x*

```
function varargout = btnPlot_Callback(h, eventdata, handles, varargin)
% Пользователь нажал кнопку Построить
...
handles.line = plot(x,y);
guidata(gcbo, handles);
% Связывание контекстного меню cmLine с линией графика
set(handles.line, 'UIContextMenu', handles.cmLine)
...
function varargout = btnFromFile_Callback(h, eventdata, handles,...
varargin)
% Пользователь нажал кнопку Из файла
...
   if (SMas(2) ~= 2) | (ndims(Mas) ~= 2) | ~isnumeric(Mas)
          errordlg('Неизвестный формат файла с данными', 'Ошибка!')
   else
          % Графическое отображение данных
          handles.line = plot(Mas(:,1), Mas(:,2));
          guidata(gcbo, handles);
          % Связывание контекстного меню cmLine с линией графика
          set(handles.line, 'UIContextMenu', handles.cmLine);
    end
...
function varargout = mnGraphPlot_Callback(h, eventdata, handles,...
    varargin)
% Пользователь выбрал пункт Построить меню График
...
handles.line = plot(Mas(:,1), Mas(:,2));
guidata(gcbo, handles);
guidata(gcbo, handles);
% Связывание контекстного меню cmLine с линией графика
set(handles.line, 'UIContextMenu', handles.cmLine);
...
```

Запустите приложение `mygui`, постройте линию любым из доступных способов и убедитесь, что щелчок правой кнопкой мыши по линии приводит к появлению контекстного меню с пунктами **синий**, **красный**, **зеленый**. Выбор пунктов не приводит к изменению цвета линии, очевидно, что следует запрограммировать событие `Callback` каждого пункта.

Программирование контекстного меню в версии 5.3

Обработка событий `Callback` пунктов всплывающего меню производится аналогично программированию меню приложения. Перейдите в редакторе меню к пункту `cmLineBlue` и занесите в строку ввода **Callback** вызов `myguiprog` с аргументом `'selcmLineBlue'`. Установите соответствующие вызовы `myguiprog` с аргументами `'selcmLineRed'` и `'selcmLineGreen'` для пунктов, имеющих имена `'cmLineRed'` и `'cmLineGreen'`. Сохраните изменения, нажав кнопку **Apply**. Осталось добавить в файл-функцию `myguiprog` блоки `case` обработки выбора пунктов меню пользователем (листинг 13.12).

Листинг 13.12. Программирование контекстного меню в версии 5.3

```
case 'selcmLineBlue'
   % Пользователь выбрал синий цвет линии в контекстном меню
   set(Hline, 'Color', 'b')
case 'selcmLineRed'
   % Пользователь выбрал красный цвет линии в контекстном меню
   set(Hline, 'Color', 'r')
case 'selcmLineGreen'
   % Пользователь выбрал зеленый цвет линии в контекстном меню
   set(Hline, 'Color', 'g')
```

Запрограммированное и связанное с линией контекстное меню разрешает быстрый доступ пользователя к цвету линии. Осталось обеспечить согласованную работу контекстного меню со списком **Цвет линии** (см. рис. 11.10) с именем `pmColor`. Выбор цвета из меню должен приводить не только к изменению цвета линии, но и к появлению соответствующей строки в раскрывающемся списке. В каждый блок `case` обработки события `Callback` пункта контекстного меню следует добавить операторы, устанавливающие нужное значение (1, 2 или 3) свойства `Value` раскрывающегося списка (листинг 13.13).

Листинг 13.13. Согласованная работа меню и списка выбора цвета в версии 5.3

```
case 'selcmLineBlue'
   % Пользователь выбрал синий цвет линии в контекстном меню
   set(Hline, 'Color', 'b')
   % Поиск указателя на раскрывающийся список и программная
   % установка первой строки "синий" в списке
```

```
   HpmColor = findobj('Tag', 'pmColor');
   set(HpmColor, 'Value', 1)
case 'selcmLineRed'
   % Пользователь выбрал красный цвет линии в контекстном меню
   set(Hline, 'Color', 'r')
   % Поиск указателя на раскрывающийся список и программная
   % установка второй строки "красный"  в списке
   HpmColor = findobj('Tag', 'pmColor');
   set(HpmColor, 'Value', 2)
case 'selcmLineGreen'
   % Пользователь выбрал зеленый цвет линии в контекстном меню
   set(Hline, 'Color', 'g')
   % Поиск указателя на раскрывающийся список и программная
   % установка третьей строки "зеленый" в списке
   HpmColor = findobj('Tag', 'pmColor');
   set(HpmColor, 'Value', 3)
```

Программирование контекстного меню в версии 6.*x*

Обработка событий `Callback` пунктов контекстного меню производится аналогично программированию меню приложения. Установите в редакторе меню для пунктов `cmLineBlue`, `cmLineRed`, `cmLineGreen` соответственно вызовы:

```
mygui('cmLineBlue_Callback',gcbo,[],guidata(gcbo))
mygui('cmLineRed_Callback',gcbo,[],guidata(gcbo))
mygui('cmLineGreen_Callback',gcbo,[],guidata(gcbo))
```

а в файле mygui.m опишите данные подфункции в соответствии с листингом 13.14.

Листинг 13.14. Программирование контекстного меню в версии 6.*x*

```
function varargout = cmLineBlue_Callback(h, eventdata, handles, varargin)
% Пользователь выбрал синий цвет линии  в контекстном меню
set(handles.line,'Color', 'b')
function varargout = cmLineRed_Callback(h, eventdata, handles, varargin)
% Пользователь выбрал красный цвет линии  в контекстном меню
set(handles.line,'Color', 'r')
```

```
function varargout = cmLineGreen_Callback(h, eventdata, handles,...
varargin)
% Пользователь выбрал зеленый цвет линии  в контекстном меню
set(handles.line,'Color', 'g')
```

Запрограммированное и связанное с линией контекстное меню разрешает быстрый доступ пользователя к цвету линии. Осталось обеспечить согласованную работу контекстного меню со списком **Цвет линии** с именем `pmColor`. Выбор цвета из меню должен приводить не только к изменению цвета линии, но и к появлению соответствующей строки в раскрывающемся списке. В каждую подфункцию обработки события `Callback` пункта контекстного меню следует добавить операторы, устанавливающие нужное значение (1, 2 или 3) свойства `Value` раскрывающегося списка (листинг 13.15).

Листинг 13.15. Согласованная работа меню и списка выбора цвета в версии 6.*x*

```
function varargout = cmLineBlue_Callback(h, eventdata, handles, varargin)
% Пользователь выбрал синий цвет линии  в контекстном меню
set(handles.line,'Color', 'b')
% Программная установка первой строки "синий" в списке
set(handles.pmColor, 'Value', 1)
function varargout = cmLineRed_Callback(h, eventdata, handles, varargin)
% Пользователь выбрал красный цвет линии  в контекстном меню
set(handles.line,'Color', 'r')
% Программная установка второй строки "красный" в списке
set(handles.pmColor, 'Value', 2)
function varargout = cmLineGreen_Callback(h, eventdata, handles,...
varargin)
% Пользователь выбрал зеленый цвет линии  в контекстном меню
set(handles.line,'Color', 'g')
% Программная установка третьей строки "зеленый" в списке
set(handles.pmColor, 'Value', 3)
```

Глава 14



Программирование событий

Приложение `mygui` с графическим интерфейсом, разработка которого описана в предыдущих главах, основано только на одном событии `Callback` объектов, размещенных в окне приложения. При нажатии, например на кнопку, возникает ее событие `Callback`, которое затем обрабатывается соответствующим блоком или подфункцией M-файла приложения. В MatLab определен еще ряд событий, полезных для написания приложений. Следующий раздел посвящен примеру приложения, использующего событие, которое происходит при щелчке мышью по области осей.

Событие осей *ButtonDownFcn*

Оси служат для вывода графической информации на экран, однако возможно реализовать и обратную задачу — позволить пользователю задавать данные щелчком мыши по области осей, сразу же отображать их на экране и заносить в массивы.

Размещение элементов интерфейса

Создайте окно приложения с осями и кнопкой. Установите в редакторе свойств надпись на кнопке **Очистить** и дайте ей имя `btnClear`. Задайте свойствам осей `XGrid` и `YGrid` значения `on` для отображения сетки, а `XLimMode` и `YLimMode` — `manual` для предотвращения изменения пределов по каждой из осей при визуализации данных. Область окна приложения должна иметь вид, приведенный на рис. 14.1.

Обратите внимание, что координаты изменяются от нуля до единицы по обеим осям и такие пределы сохранятся при отображении любых данных, благодаря фиксации пределов свойствами `XLimMode` и `YLimMode`, значения которых изменены с `auto` (по умолчанию) на `manual`. Если координаты каких-либо точек графика выходят за пределы осей, то автоматического изменения пределов теперь не произойдет, точек просто не будет на осях. Сохраните приложение с именем `bdf`.

Следующий этап состоит в программировании приложения. Пользователь щелчком мыши по области графика наносит точку, которая соединяется с

предыдущей отрезком прямой линии, образуя ломаную. Координаты точек записываются в массивы по мере их появления, что позволяет просто задавать геометрию плоских объектов при помощи координат вершин.

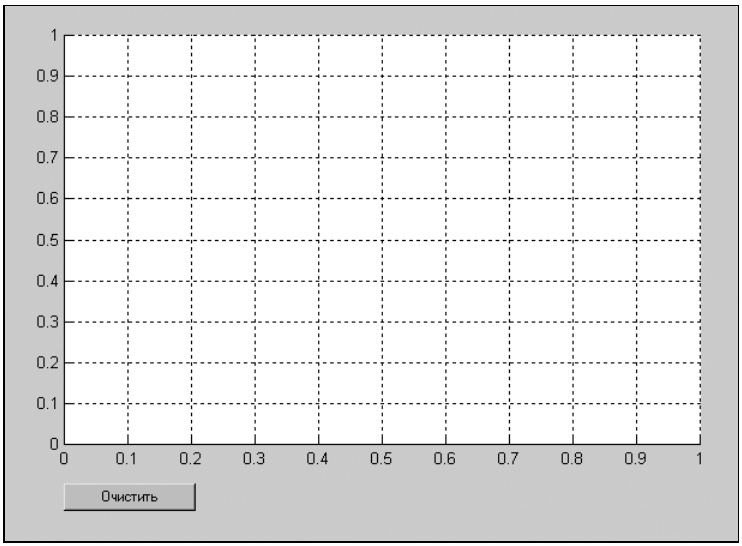


Рис. 14.1. Область окна приложения

Программирование приложения

Начните с простого примера, для того, чтобы понять обработку события `ButtonDownFcn`. Пусть пока щелчок мыши в точке, принадлежащей двумерным осям, приводит к появлению круглого маркера в данной точке. Когда пользователь щелкает мышью по области осей, возникает событие `ButtonDownFcn` осей. Блок операторов, обрабатывающих данное событие, должен определять координаты точки щелчка и вывести в нее круглый маркер. Свойство осей `CurrentPoint` содержит матрицу размера два на три с информацией о положении указателя мыши в момент щелчка в системе координат осей. Предполагается, что оси трехмерны и точка находится на линии, перпендикулярной экрану.

Линия проходит через оси, пересекая параллелепипед осей в двух точках (рис. 14.2), три координаты которых хранятся в матрице

$$\begin{bmatrix} xb & yb & zb \\ xf & yf & zf \end{bmatrix}$$

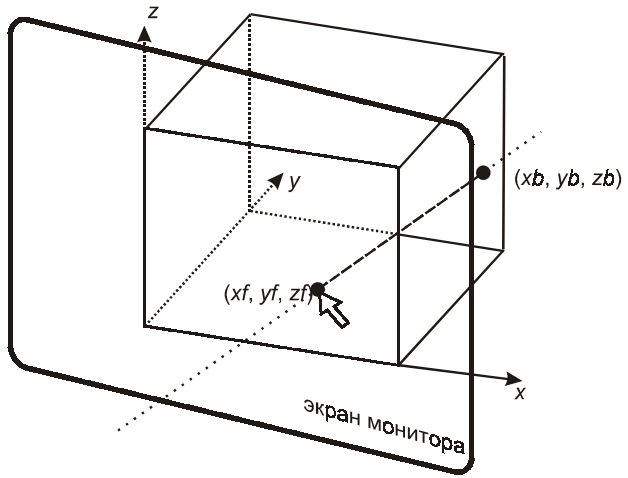


Рис. 14.2. Координаты точек, хранящиеся в `CurrentPoint`

Поскольку приложение предназначено для описания геометрии плоских фигур, то оси двумерны и, следовательно, точки с координатами (*xb*, *yb*, *zb*) и (*xf*, *yf*, *zf*) совпадают. Более того, третья координата не понадобится.

Запрограммируйте событие `ButtonDownFcn` осей, поместив в блок обработки события операторы получения координат текущей точки, приведенные в листинге 14.1. В версии 5.3 разместите операторы в редакторе откликов, предварительно выделив оси (убедитесь, что в раскрывающемся списке выбрано `ButtonDownFcn`). При работе в MatLab 6.*x* выделите оси в редакторе приложений, и во всплывающем меню перейдите к пункту **Edit ButtondownFcn**, в редакторе M-файлов автоматически создается подфункция `axes1_ButtondownFcn`, в которой следует набрать операторы листинга 14.1. В версии 6.*x* можно использовать аргумент `h` для указания на оси, вместо `gcbo`.

Листинг 14.1. Определение координат точки и отображение ее на осях

```
% Получение матрицы с координатами (xb,yb,zb) и (xf,yf,zf)
Mp = get(gcbo,'CurrentPoint');
% Выбор нужных элементов матрицы с абсциссой и ординатой точки
Xcoord = Mp(2,1);
Ycoord = Mp(2,2);
hold on % для последующего вывода на те же оси
% Отображение точки на осях
plot(Xcoord, Ycoord, 'ko')
```

Запустите приложение bdf и убедитесь, что в месте щелчка мыши по области осей появляется круглый маркер. Осталось обработать событие ButtonDownFcn осей так, чтобы добавляемые маркеры соединялись отрезками прямых, а их координаты заносились не в переменные, а добавлялись в массивы Xcoord, Ycoord. Кроме того, нажатие **Очистить** должно приводить к пропаданию ломаной линии и удалению данных из массивов. Алгоритм достаточно простой: если при возникновении события ButtonDownFcn осей массивы Xcoord, Ycoord пусты (первый щелчок), то следует вывести маркер в точке щелчка, а если не пусты (не первый щелчок), то надо в аргументах plot указать два вектора из двух координат каждый. Первый вектор содержит абсциссы текущей и предыдущей точек, а второй — их ординаты.

При работе в версии 5.3 оформите обработку события ButtonDownFcn осей и Callback кнопки **Очистить** в файл-функции bdfprog (листинг 14.2), не забудьте добавить в редакторе событий вызовы bdfprog с соответствующими аргументами. Массивы с координатами Xcoord и Ycoord проще всего объявить как глобальные переменные, иначе при попытке вычисления массива Xcoord длины несуществующего в начале работы приложения возникнет сообщение об ошибке.

Листинг 14.2. Файл-функция bdfprog обработки событий приложения bdf

```
function bdfdemoprog(event)
global Xcoord Ycoord % объявление массивов координат
switch event
case 'clickMouse'
   % Обработка события ButtonDownFcn
   % Пользователь щелкнул мышью по области осей
   % Получение матрицы с информацией о щелчке
   Mp = get(gcbo,'CurrentPoint');
   % Длина массива равна числу проделанных ранее щелчков
   Npoints = length(Xcoord);
   % Сделан новый щелчок, увеличение счетчика щелчков
   Npoints = Npoints +1;
   % Выделение абсциссы и ординаты точки
   Xcoord(Npoints) = Mp(2,1);
   Ycoord(Npoints) = Mp(2,2);
   % Проверка общего числа щелчков
   if Npoints == 1
      % Текущий щелчок первый, маркером отображается место щелчка
      hold on
      plot(Xcoord(Npoints), Ycoord(Npoints), 'ko','LineWidth', 2)
```

```
   else
      % Текущий щелчок не первый, точка щелчка соединяется
      % с предыдущей точкой отрезком линии
      x1 = Xcoord(Npoints-1);
      y1 = Ycoord(Npoints-1);
      x2 = Xcoord(Npoints);
      y2 = Ycoord(Npoints);
      plot([x1 x2], [y1 y2], 'ko-', 'LineWidth', 2)
   end
case 'pressClear'
% Пользователь нажал кнопку Очистить
   cla % очистка осей
   % Удаление информации из массивов
   Xcoord = [];
   Ycoord = [];
   Npoints = 0; % обнуление числа щелчков
end
```

М-файл с основной функцией и подфункциями обработки событий для версии 6.*x* приведены в листинге 14.3. Переменные, предназначенные для хранения информации о координатах, содержатся в полях структуры `handles` и сохраняются после каждого изменения при помощи `guidata`. Обратите внимание, что в основной функции, выполняющейся при запуске приложения, инициализируются поля `Xcoord` и `Ycoord` с массивами координат.

Листинг 14.3. М-файл bdf.m с подфункциями обработки событий приложения `bdf`

```
function varargout = bdf(varargin)
...
   handles = guihandles(fig);
   % Инициализация массивов при запуске приложения в основной функции
   handles.Xcoord = [];
   handles.Ycoord = [];
   ...
% --------------------------------------------------------------------
function varargout = axes1_ButtondownFcn(h, eventdata, handles, varargin)
% Обработка события ButtonDownFcn
% Пользователь щелкнул мышью по области осей
% Получение матрицы с информацией о щелчке
```

```
Mp = get(h,'CurrentPoint');
% Длина массива равна числу проделанных ранее щелчков
Npoints = length(handles.Xcoord);
% Сделан новый щелчок, увеличение счетчика щелчков
Npoints = Npoints +1;
% Выделение абсциссы и ординаты точки
handles.Xcoord(Npoints) = Mp(2,1);
handles.Ycoord(Npoints) = Mp(2,2);
% Проверка общего числа щелчков
if Npoints == 1
    % Текущий щелчок первый, маркером отображается место щелчка
    hold on
    plot(handles.Xcoord(Npoints),...
       handles.Ycoord(Npoints),'ko','LineWidth', 2)
else
   % Текущий щелчок не первый, точка щелчка соединяется
   % с предыдущей точкой отрезком линии
    x1 = handles.Xcoord(Npoints-1);
    y1 = handles.Ycoord(Npoints-1);
    x2 = handles.Xcoord(Npoints);
    y2 = handles.Ycoord(Npoints);
    plot([x1 x2], [y1 y2], 'ko-', 'LineWidth', 2)
end
% Сохранение данных в структуре
guidata(gcbo, handles);
% --------------------------------------------------------------------
function varargout = btnClear_Callback(h, eventdata, handles, varargin)
% Пользователь нажал кнопку Очистить
cla % очистка осей
% Удаление данных из полей с массивами
handles.Xcoord = [];
handles.Ycoord = [];
% Сохранение данных в структуре
guidata(gcbo, handles);
```

Запустите приложение `bdf` и убедитесь, что при помощи мыши можно построить ломаную линию (рис. 14.3), а нажатие на кнопку **Очистить** приводит к ее исчезновению. Несложно дополнить приложение `bdf` кнопкой, вызывающей диалоговое окно для сохранения координат вершин ломаной в файле.

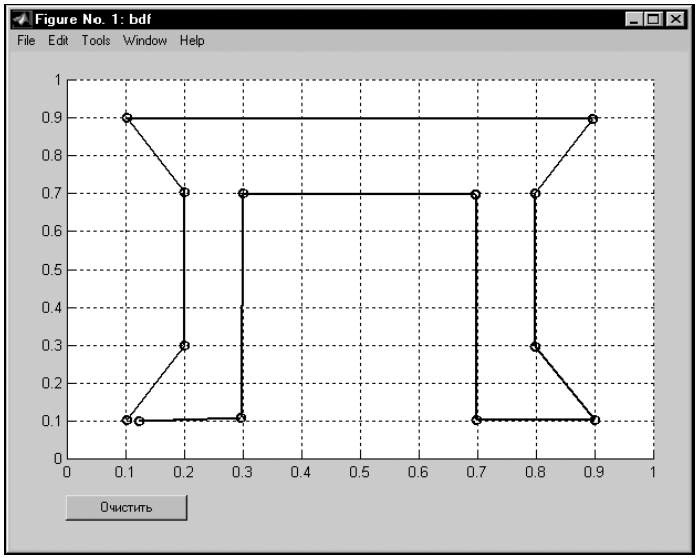


Рис. 14.3. Работающее приложение `bdf`

Приложение `bdf` не позволяет построить замкнутую ломаную, более того, щелчок по линии или вершине ломаной не приводит к нанесению нового отрезка. Дело в том, что событие `ButtonDownFcn` *осей* возникает при щелчке мыши по области осей, не занятой другими объектами. Щелчок мыши по линии вызывает событие `ButtonDownFcn` *линии*, а оно не запрограммировано. Можно просто решить проблему добавлением невидимых осей тех же размеров, что и существующие.

См. разд. "Размещение текста в графическом окне" главы 9.

Новые оси должны быть расположены *поверх* осей с ломаной. Щелчок мыши вызывает событие `ButtonDownFcn` невидимых осей, а полученные данные визуализируются на других осях.

Альтернативой хранения данных в глобальных переменных (для версии 5.3) или в структуре `handles` (для версии 6.*x*) является использование свойства `UserData` осей. Данные, отображаемые на осях, могут являться содержимым свойства `UserData` и таким образом быть связанными с осями. Следующий раздел посвящен описанию событий и свойств объектов MatLab, которые могут быть полезными при программировании собственных приложений.

События и свойства объектов в MatLab

MatLab является объектно-ориентированной системой, что, в частности, позволяет просто и эффективно реализовывать графический интерфейс при-

ложений. Данный раздел посвящен описанию событий и некоторых связанных с ними свойств графических объектов MatLab.

Иерархия объектов

Упрощенная иерархия объектов была приведена на рис. 4.5. В ней не определено место элементов управления, таких как кнопки, области ввода, меню, контекстные меню и т. д. Полная схема иерархии объектов MatLab проиллюстрирована на рис. 14.4.

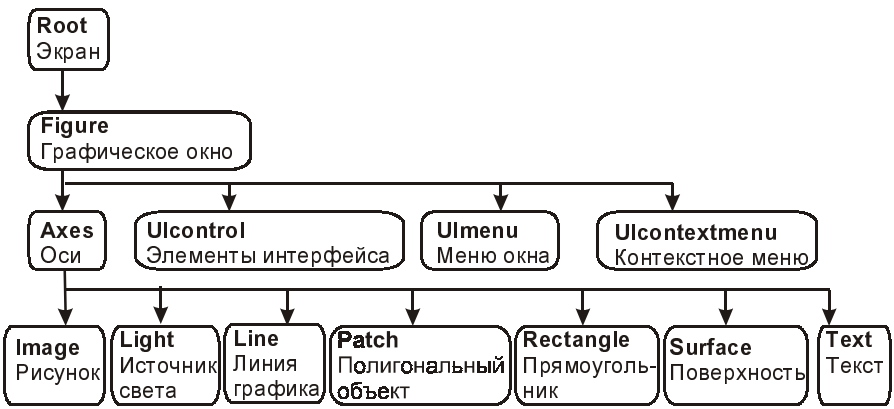


Рис. 14.4. Иерархия объектов MatLab

Для освоения материала необходимы два понятия из области объектно-ориентированного программирования — *предок* и *потомок*. Например, оси являются потомками графического окна и предками для линий, поверхностей и текста.

Свойства осей, линий, поверхностей и текста изложены в главе 9.

Условно можно считать, что объект `Root` соответствует экрану монитора. Объект `Root` не имеет предков, а его потомками являются графические окна — объекты `Figure`. Свойства и события `Root` позволяют произвести некоторые глобальные установки MatLab. Объект `Figure` представляет собой отдельное графическое окно, у `Root` может быть много потомков `Figure`. Графическое окно имеет несколько потомков: оси (объект `Axes`), элементы интерфейса, такие, как кнопки, переключатели, списки и т. д. (объекты `UIcontrol`), меню окна с пунктами (объект `UImenu`), контекстные меню (объекты `UIcontextmenu`). Все объекты, отвечающие за визуализацию данных (`Image`, `Light`, `Line`, `Patch`, `Rectangle`, `Surface` и `Text`), являются потомками `Axes`.

Объект *Root*

Объект `Root` автоматически создается при запуске MatLab и предназначен для установки общих свойств, характеризующих работу MatLab. Указатель на `Root` всегда равен нулю. Свойство `CallbackObject` содержит указатель на тот объект, событие `Callback` которого обрабатывается в данный момент времени. Указатель на текущее графическое окно является значением свойства `CurrentFigure`. Положение указателя мыши на экране монитора хранится в векторе из двух элементов, являющемся значением свойства `PointerLocation`, причем безразлично, находится ли курсор мыши на каком-либо объекте MatLab или нет. Единицы измерения устанавливаются значением свойства `Units` объекта `Root`, по умолчанию используются пикселы. Свойство `PointerWindow` содержит указатель на графическое окно, содержащее в данный момент курсор мыши и ноль, если курсор вне пределов графических окон.

Задание формата вывода результатов вычислений в командное окно может быть осуществлено не только при помощи функции `format` с соответствующим аргументом, но и установкой свойству `Format` например, значений `short`, `shortE` (по умолчанию), `long`, `longE` и свойству `FormatSpacing` значений `compact` для вывода на каждой строке вместо `loose`, установленного по умолчанию.

Свойства `ScreenDepth` и `ScreenSize` позволяют получить информацию о цветовой палитре монитора (разрядности цвета) и размере экрана в единицах, определяемых свойством `Units`. Размер экрана определяется вектором из четырех элементов с координатами левого нижнего и правого верхнего угла монитора. Поскольку по умолчанию единицами измерения являются пикселы, то `ScreenSize` возвращает текущее разрешение монитора.

Вектор указателей на все имеющиеся графические окна содержится в свойстве `Children` объекта `Root`. Упорядочение данного вектора позволяет расположить графические окна на экране в нужной последовательности.

Объект *Figure*

Потомками `Root` являются графические окна — объекты `Figure`, которые создаются функцией `figure` или автоматически при использовании графических функций высокого уровня `plot`, `surf` и т. д. Свойства создаваемого объекта `Figure` определяются при его создании аргументами `figure`, например, `figure('Color', 'w')`. Если аргументы не заданы, то создается графическое окно со свойствами, установленными по умолчанию. Доступ к установленным по умолчанию свойствам объектов `Figure` производится только с уровня предка `Figure`, т. е. объекта `Root`. Используется следующий формат названия свойства:

```
DefautНазваниеОбъектаНазваниеСвойства
```

НазваниеОбъекта в данном случае, графического окна, есть Figure, а НазваниеСвойства — любое свойство объекта Figure. Например, следующий вызов get

```
>> get(0,'DefaultFigureColor')
ans =
    0.8000    0.8000    0.8000
```

приводит к получению вектора цвета области графического окна в палитре RGB, используемого по умолчанию. Изменение цвета производится функцией set:

```
set(0,'DefaultFigureColor','w')
```

Теперь область всех создаваемых графических окон имеет белый цвет. Свойства, установленные по умолчанию, используются при создании всех графических окон, однако свойства, указанные во входных аргументах функции figure, *имеют больший приоритет* по сравнению с определенными по умолчанию.

Знание свойств и событий графического окна существенно облегчает программирование приложений с графическим интерфейсом.

Свойства, отвечающие за расположение графического окна на экране, описаны в разд. "Управление положением графических окон" главы 9.

Графическое окно приложения, создаваемое MatLab 5.3, позволяет пользователю изменять его размеры, что негативно отражается на расположении элементов интерфейса и осей. Значение off свойства Resize позволяет запретить изменение размеров окна пользователем, а on, соответственно, разрешить. Среда GUIDE в версии 6.*x* устанавливает свойство Resize окна приложения в off. Событие ResizeFcn объекта Figure возникает при изменении размеров окна пользователем. Обработка данного события может включать управление положением объектов, расположенных в пределах графического окна.

Окна делятся на два типа: обычные, между которыми пользователь может переключаться, и *модальные*, которые требуют завершения работы с данным окном. Свойство WindowStyle может принимать два значения: normal (по умолчанию) или modal. Модальные окна полезны при создании диалоговых окон, определяющих дальнейший ход работы приложения.

Свойства CurrentAxes и CurrentObject содержат указатели на текущие оси и объекты, размещенные в области окна. Данные свойства позволяют определить последние объекты, к которым обратился пользователь.

Ряд свойств и событий объекта Figure позволяет эффективно обрабатывать действия пользователя, связанные с мышью и клавиатурой. Свойство

CurrentCharacter содержит символ, который задал пользователь при помощи клавиатуры, работая с окном приложения.

Позиция окна, в которой был произведен щелчок мышью, хранится в векторе из двух элементов в свойстве CurrentPoint. Тип щелчка определяется значением свойства SelectionType (табл. 14.1).

Таблица 14.1. *Значения свойства* SelectionType

Значение	Тип щелчка
Normal	Левой кнопкой мыши
Extended	Левой кнопкой с удержанием клавиши <Shift>
Alternate	Правой кнопкой или левой кнопкой с удержанием клавиши <Ctrl>
Open	Двойной щелчок

Ряд событий MatLab, связанных с мышью, позволяет обрабатывать действия пользователя в окне приложения. Событие ButtonDownFcn возникает при щелчке мышью по области окна, *не занятой другими объектами*. Для программирования щелчка мыши в пределах окна приложения применяются следующие события:

- WindowsButtonDownFcn — нажатие на кнопку мыши;
- WindowsButtonMotionFcn — перемещение указателя мыши;
- WindowsButtonUpFcn — отпускание кнопки мыши.

Тип указателя мыши задается свойством Pointer, которое может принимать, например, значения:

- crosshair — перекрестие;
- arrow — обычная стрелка (по умолчанию);
- watch — песочные часы;
- ibeam — вертикальный курсор.

Перечень всех возможных значений приведен в справочной системе MatLab. Значение custom позволяет определить собственную квадратную пиктограмму размером шестнадцать пикселов для курсора мыши в матрице и задать ее в качестве значения свойства PointerShapeCData. Единица означает черный цвет, двойка — белый, а прозрачность пиксела достигается установкой значения NaN в соответствующие элементы матрицы. Последовательность команд, приведенная в листинге 14.4, превращает курсор мыши в квадратную прозрачную рамку.

Листинг 14.4. Генерация указателя мыши в виде квадратной прозрачной рамки

```
P = ones(16);
MP(2:15,2:15) = NaN;
set(gcf, 'Pointer', 'custom')
set(gcf, 'PointerShapeCData', MP)
```

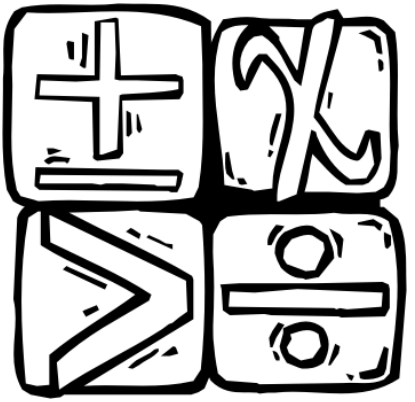
Обработка событий, которая занимает заметное или достаточно большое время, приводит к необходимости позаботиться о порядке выполнения событий. Свойства `BusyAction` и `Interruptible` предназначены для управления последовательностью событий. Подробная информация об использовании данных свойств содержится в справочной системе MatLab.

Пользователь может закрыть окно приложения при помощи кнопки в правом верхнем углу, или выбором пункта **Close** меню **File** окна, или командой `close`. При закрытии графического окна обрабатывается событие `CloseRequestFcn`, которое по умолчанию содержит вызов функции `closerec`, закрывающей окно. Программирование события с выводом диалогового окна подтверждения позволяет предостеречь пользователя от случайного закрытия окна приложения (листинг 14.5).

Листинг 14.5. Обработка `CloseRequestFcn` при закрытии окна приложения

```
s = questdlg('Закрыть окно приложения?',  'Подтверждение закрытия',...
  'Да','Нет','Нет');
switch s
   case 'Да',
     delete(gcf)
   case 'Нет'
     return
end
```

Следует отметить, что среда GUIDE позволяет достаточно просто написать приложение для проведения собственных исследований. Основное преимущество визуальной среды GUIDE по сравнению со многими другими современными языками программирования состоит в том, что разработчик может использовать большой набор готовых функций MatLab, которые реализуют алгоритмы решения широкого спектра задач. Основные функции, предназначенные для простейших вычислений и визуализации данных, были описаны в первой части книги. Как уже упоминалось, функции, направленные на решение специальных задач, собраны в пакеты, называемые ToolBox. Использованию некоторых ToolBox посвящена следующая часть книги.



Часть IV

Использование ToolBox

Глава 15. Решение задач математической физики

Глава 16. Разреженные матрицы

Глава 17. Оптимизация

Глава 18. Символические вычисления

Глава 15



Решение задач математической физики

Большинство задач механики, теплопроводности, течения жидкости, электростатики и электродинамики сводятся к дифференциальным уравнениям в частных производных в областях сложной формы. Данная глава посвящена описанию возможностей ToolBox Partial Differential Equations (PDE), предназначенного для решения граничных задач для дифференциальных уравнений в частных производных в двумерных областях методом конечных элементов. В состав данного ToolBox входит приложение `pdetool` с графическим интерфейсом пользователя, использование которого не требует глубокого понимания метода конечных элементов. Кроме того, ToolBox PDE обладает набором функций, полезных при написании собственных приложений для решения граничных задач методом конечных элементов. Следующий раздел посвящен основам работы с `pdetool`.

Простой пример

Среда `pdetool` позволяет задать геометрию области, тип и коэффициенты дифференциального уравнения, граничные и начальные условия, произвести разбиение области на конечные элементы (триангуляцию), решить получающуюся систему линейных уравнений и визуализировать результат. Пользователь должен сформулировать задачу, т. е. написать уравнение и граничные условия, и последовательно выполнять вышеописанные действия. Изучите основы работы в `pdetool` на примере задачи теплопроводности в простой области.

Постановка задачи

Требуется найти распределение температуры T в области, изображенной на рис. 15.1.

Круговое отверстие расположено в центре прямоугольника. Правая и левая границы прямоугольной области теплоизолированы. Внутри области нет источников тепла и коэффициент теплопроводности k равен 200. Распределение температуры описывается дифференциальным уравнением

$$k\nabla \cdot \nabla T = 0;$$

граничные условия на правой и левой границе задают нулевой поток тепла (n — вектор нормали к границе)

$$n \cdot k\nabla T = 0;$$

на верхней и нижней границе $T = 600$, а на окружности $T = 500$. Размерности единиц не указываются.

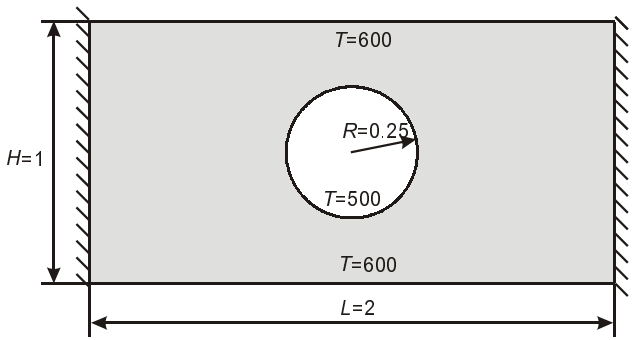


Рис. 15.1. Область и граничные условия

Поставленная задача просто решается при помощи среды `pdetool`. Инструкции, приведенные в следующих разделах, помогают освоить работу в `pdetool`. Более подробному описанию возможностей посвящен отдельный раздел.

Среда *pdetool*, конструирование области

Начните работу, выполнив команду `pdetool` в командном окне, появляется окно **PDE Toolbox** среды `pdetool`, изображенное на рис. 15.2.

Данное окно содержит следующие основные элементы:

- строку меню, каждое меню соответствует определенному этапу решения задачи;
- панель инструментов рисования геометрических примитивов, определяющих область;
- панель инструментов для задания граничных условий, коэффициентов уравнения, триангуляции, решения и визуализации результата;
- область ввода **Set formula** для конструирования области из геометрических примитивов;
- оси для создания области.

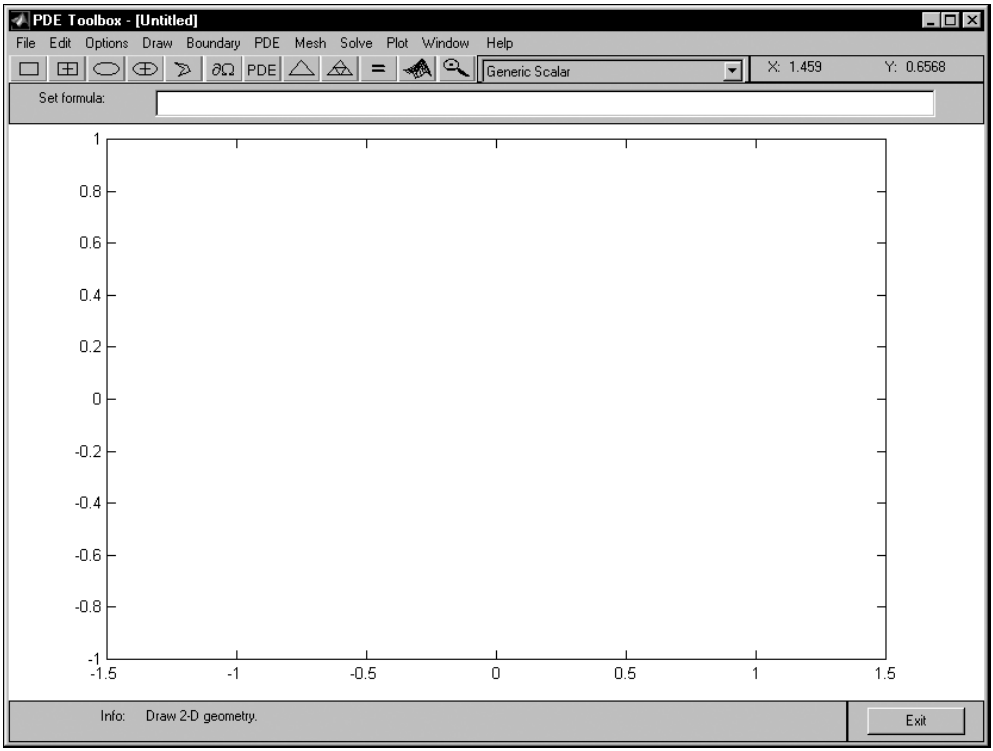


Рис. 15.2. Среда `pdetool`

Конструирование области, приведенной на рис. 15.1, включает в себя создание прямоугольника и круга заданных размеров и вычитание круга из прямоугольника. Удобно расположить область в начале координат, тогда координаты нижнего левого угла прямоугольника будут (−1, −0.5), а его высота и ширина 1 и 2 соответственно. Центр круга совпадает с началом координат.

Начните с создания прямоугольника. Выберите в меню **Draw** пункт **Rectangle/square** или воспользуйтесь соответствующим инструментом (его пиктограмма содержит прямоугольник). Нарисуйте мышью требуемую прямоугольную область на осях от угла, удерживая нажатой левую кнопку. Скорее всего, точно выдержать размеры и положение не удалось. Двойной щелчок мыши по прямоугольнику приводит к появлению диалогового окна **Object Dialog** (рис. 15.3), предназначенного для установки заданных размеров и положения объекта.

Введите в строках **Left** и **Bottom** координаты нижнего левого угла прямоугольника: −1 и −0.5, в **Width** и **Height** задайте ширину и высоту: 2 и 1 соответственно. Поле **Name** предназначено для определения названия объекта, в данном случае прямоугольника. По умолчанию имена создаваемых прямо-

угольников состоят из буквы R (от Rectangle) и порядкового номера. При конструировании области с простой геометрией нет необходимости изменять имена объектов. Нажмите кнопку **OK** и убедитесь, что прямоугольник принял нужное положение и размеры на осях.

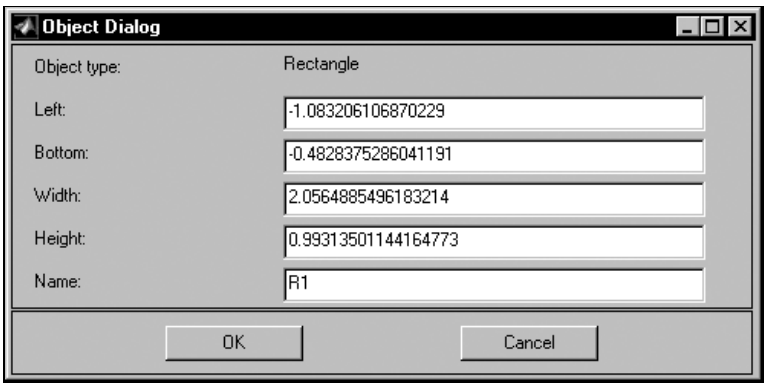


Рис. 15.3. Диалоговое окно **Object Dialog**

Нарисуйте круг с центром в точке (0, 0) и радиусом 0.25. Выберите в меню **Draw** пункт **Ellipse/circle (centered)** или воспользуйтесь соответствующим инструментом, его пиктограмма — эллипс с перекрестием.

Замечание

Перекрестие на пиктограмме инструмента, или слово **centered** в названии пункта меню **Draw** означают, что соответствующий объект рисуется на осях из центра при нажатой левой кнопки мыши.

Поместите курсор мыши в начало координат осей и нарисуйте окружность при помощи мыши с одновременным удержанием в нажатом состоянии клавиши <Ctrl>. Клавиша <Ctrl> позволяет рисовать геометрические примитивы квадрат и круг, если активны, соответственно, инструменты **Rectangle/square, Rectangle/square (centered), Ellipse/circle, Ellipse/circle (centered).** Перейдите к свойствам круга двойным щелчком мыши и уточните в диалоговом окне **Object Dialog** его положение и радиус. В строки **X-center**, **Y-center** следует занести нули, а в **Radius** — величину радиуса 0.25. Круг имеет название C1 (от Circle).

Геометрические примитивы для задания области созданы. Сейчас установлен режим рисования (переход в него происходит при использовании инструментов рисования или выборе пункта **Draw Mode** меню **Draw**), поэтому на осях присутствуют все добавленные объекты. Следующий этап состоит в определении взаимосвязи между примитивами, образующими область, — круг должен

быть удален из прямоугольника. Связь между примитивами определяется в строке **Set Formula** среды `pdetool`. Знак плюс означает объединение объектов, а минус — вычитание. Области, изображенной на рис. 15.1, соответствует формула `R1-C1`, из большего объекта вычитается меньший по размерам.

Область, в которой решается дифференциальное уравнение, сконструирована, теперь задайте коэффициенты уравнения и граничные условия.

Определение уравнения и граничных условий

Меню **Options** содержит подменю **Application**, которое позволяет задать тип решаемой задачи. Пункт **Heat Transfer** соответствует задаче о распределении тепла. Выберите данный пункт, слева от названия появится флаг — среда `pdetool` теперь настроена на решение задачи теплопроводности. Использование раскрывающегося списка, размещенного на панели инструментов, приводит к аналогичному результату. Установите режим дифференциального уравнения, выбрав пункт **PDE Mode** в меню **PDE**. Оси теперь содержат область с отверстием. Именно для этой области и следует определить коэффициенты и правую часть дифференциального уравнения.

Перейдите к пункту **PDE Specification** меню **PDE** или примените двойной щелчок по области, появляется диалоговое окно **PDE Specification**, изображенное на рис. 15.4. Обратите внимание, что вверху диалогового окна на панели **Equation** содержится общий вид уравнения теплопроводности, которое может быть решено в среде `pdetool`:

```
-div(k*grad(T))=Q+h*(Text-T)
```

Значения коэффициентов устанавливаются в строках ввода, расположенных на правой панели диалогового окна. Левая панель служит для выбора типа уравнения, переключатель **Elliptic** соответствует задаче о стационарном распределении тепла, описываемой эллиптическим дифференциальным уравнением, а **Parabolic** — нестационарному случаю. Решение нестационарных задач описано ниже. Убедитесь в том, что включен **Elliptic** и задайте в строках ввода коэффициенты уравнения рассматриваемой задачи.

Выбор подходящих коэффициентов позволяет свести дифференциальное уравнение, записанное в общем виде, к уравнению, которое описывает задачу, поставленную в начале данного раздела. Установите коэффициенту теплопроводности k значение 200. Поскольку в рассматриваемой задаче нет распределенных источников тепла, то поток тепла Q равняется нулю. Коэффициент конвективного теплообмена h и внешняя температура, входящие в общий вид дифференциального уравнения, также должны иметь нулевое значение. Нажмите **OK** для сохранения проделанных изменений.

Перейдите к заданию граничных условий. Выберите пункт **Boundary Mode** меню **Boundary**, среда `pdetool` находится в режиме установки граничных условий, в окне отображаются только границы области. Обратите внимание,

что прямоугольник имеет четыре границы, по числу сторон, и окружность также составлена из четырех дуг.

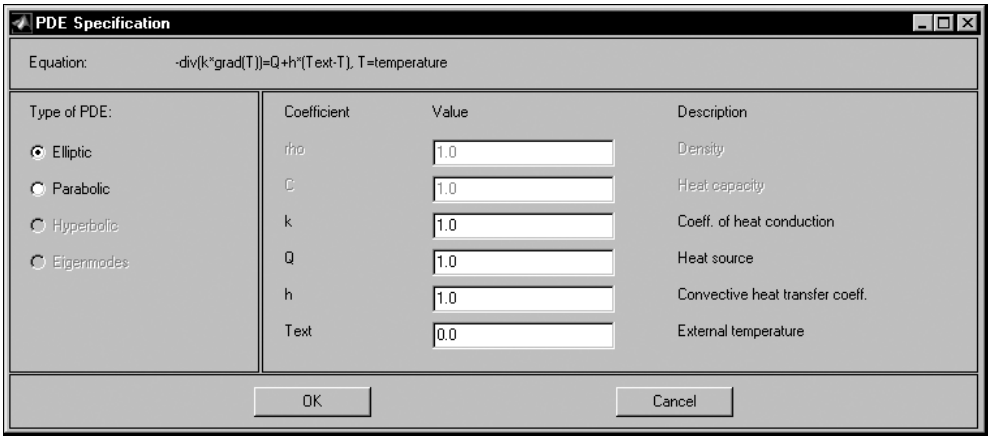


Рис. 15.4. Диалоговое окно **PDE Specification**

Сделайте текущей верхнюю границу прямоугольника щелчком мыши и выберите в меню **Boundary** пункт **Specify Boundary Conditions**, появляется диалоговое окно **Boundary Condition**, предназначенное для выбора типа граничного условия (рис. 15.5).

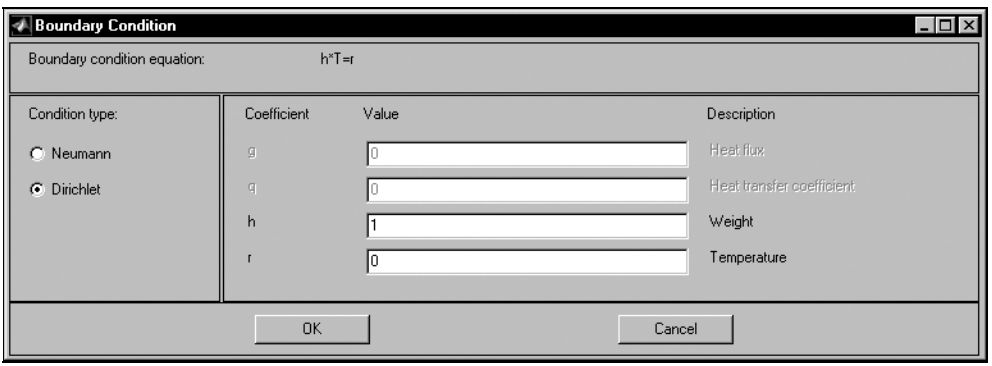


Рис. 15.5. Диалоговое окно **Boundary Condition**

Граничное условие предлагается в общем виде, выбор коэффициентов позволяет получить требуемый частный случай. На верхней границе прямоугольника задано значение температуры, равное 600 (см. рис. 15.1). Это граничное условие Дирихле. Убедитесь, что переключатель **Condition type** окна

Boundary Condition установлен в положение **Dirichlet**, соответствующее условию Дирихле. Вверху окна на панели **Boundary condition equation** присутствует общий вид условия `h*T = r`, где `h` — весовой коэффициент, а `r` — заданная температура. Установите при помощи строк ввода (доступны только две строки для данного положения переключателя) значение `h` в единицу, а `r` приравняйте 600. Сохраните значения, нажав **OK**, и проделайте аналогичную операцию для нижней стороны прямоугольника, предварительно сделав ее текущей щелчком мыши.

Граничные условия можно задавать по отдельности на каждой части границы, или объединить несколько частей и определить одинаковые граничные условия сразу для всей группы. Добавление части границы в группу производится щелчком мыши с одновременным удержанием <Shift>. Двойной щелчок мышью по части границы или по группе частей дает быстрый доступ к диалоговому окну **Boundary Condition**. Установите температуру 500 на границе отверстия, сгруппировав предварительно четыре части окружности.

На правой и левой стороне прямоугольной области поток тепла равен нулю, поскольку данные границы теплоизолированы (см. рис. 15.1). Равенство нулю потока есть частный случай условий Неймана. Объедините две границы в группу и откройте диалоговое окно **Boundary Condition**. Установите переключатель **Condition type** в положение **Neumann**, соответствующее граничным условиям Неймана. Вверху окна отобразился общий вид условия Неймана `n*k*grad(T)+q*T = g`, отвечающего конвективному теплообмену через границу с окружающей средой. Очевидно, что для получения теплоизолированных границ требуется задать коэффициенты q и g равными нулю.

Замечание

В режиме установки граничных условий границы с условиями Дирихле отображаются красным цветом, а синий цвет выделяет границы, на которых задано условие Неймана.

Доступ к диалоговым окнам **Boundary Condition** и **PDE Specification** осуществляется не только двойным щелчком мыши или из меню, но и при помощи кнопок панели инструментов с надписями $\partial\Omega$ и **PDE**.

Уравнение и граничные условия определены, следующим этапом является решение задачи и визуализация результата.

Решение и визуализация результата

Первый шаг состоит в триангуляции — покрытии области сеткой, состоящей из треугольников. Триангуляция и установка ее параметров производится в меню **Mesh**. Перейдите в режим триангуляции, выбрав пункт **Mesh Mode**, область разбивается на достаточно крупные треугольные элементы, причем считается, что граница области составлена из сторон некоторых

элементов (рис. 15.6). Инициализация триангуляции может быть произведена кнопкой с треугольником. Для получения решения с приемлемой точностью начальной триангуляции недостаточно, следует уменьшить шаг разбиения области. Выберите пункт **Refine Mesh** или нажмите кнопку с треугольником, разделенным на четыре части. Каждый выбор данного пункта приводит к равномерному уменьшению размеров треугольников.

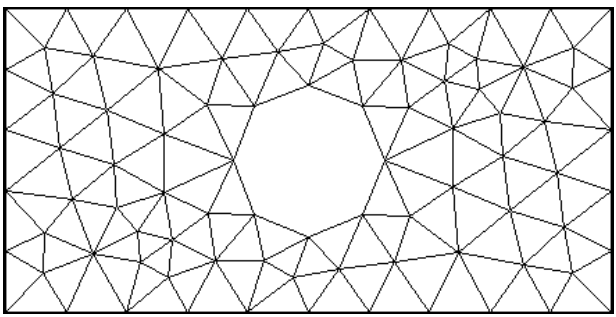


Рис. 15.6. Начальная триангуляция

Учтите, что выбор слишком мелкой сетки может привести к значительным затратам времени на решение системы линейных уравнений методом конечных элементов. Для возврата к начальной триангуляции служит пункт **Initialize Mesh**. Уменьшите треугольники исходной сетки в несколько раз. Обратите внимание, что уменьшение размеров ячеек сетки улучшает вид границ области (рис. 15.7).

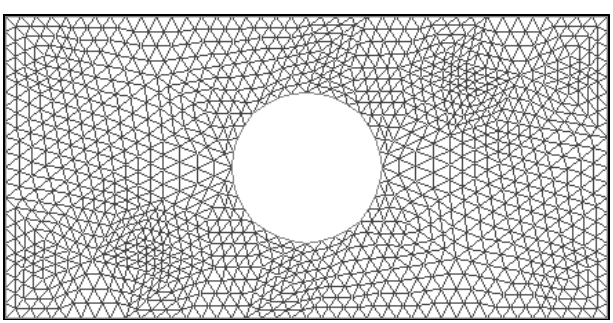


Рис. 15.7. Расчетная триангуляция

Решение задачи на расчетной сетке производится выбором пункта **Solve PDE** меню **Solve** или нажатием на кнопку со знаком "равно". Используется способ решения задачи, установленный по умолчанию.

Выбор различных способов решения описан в разд. "Параметры триангуляции и управление процессом решения" данной главы.

Найденное распределение температуры отображается в окне среды `pdetool` контурным графиком с цветовой заливкой, рядом с которым расположен столбик с информацией о соответствии цвета значению температуры (рис. 15.8).

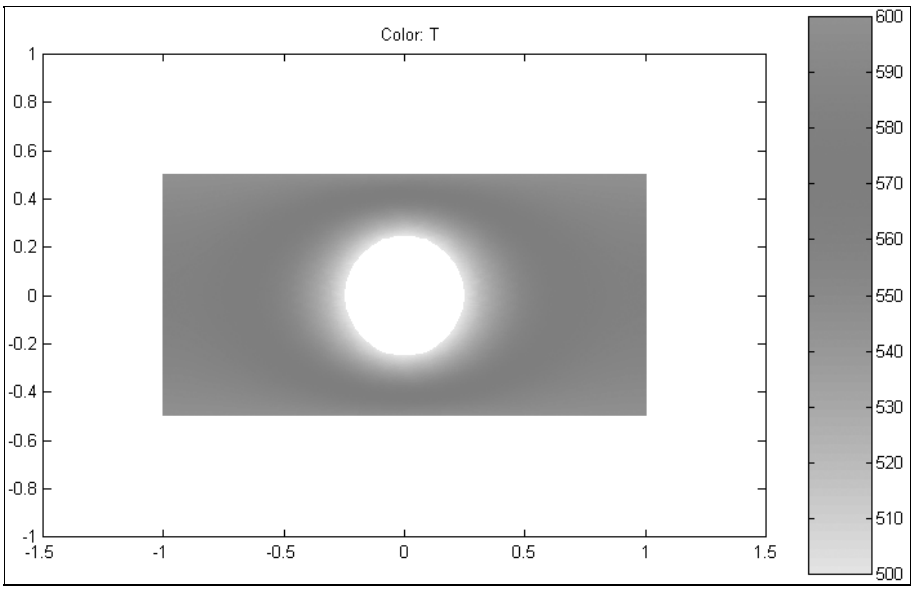


Рис. 15.8. Графическое представление результата

Изменение геометрии области, граничных условий, типа уравнения и его коэффициентов может быть выполнено даже если решение уже найдено. Следует перевести среду `pdetool` в соответствующий режим и произвести требуемые действия. Сохранение работы производится в M-файле из пункта **Save as** (или **Save** для сохранения в существующем M-файле с тем же именем) меню **File**. M-файл содержит функции ToolBox PDE, которые вызываются в среде `pdetool` в соответствии с последовательностью действий пользователя. Данный M-файл содержит не только геометрию области, тип и коэффициенты уравнения и граничных условий, но и текущие установки среды, что позволяет впоследствии продолжить решение задачи.

Среда `pdetool` позволяет распечатать график решения. Выбор пункта **Print** меню **File** приводит к появлению диалогового окна печати. Для установки параметров страницы следует нажать кнопку **Page Setup**, отображается одноименное диалоговое окно, работа с которым описана ранее.

См. разд. "Сохранение, экспорт и печать" (соответствующий версии MatLab) главы 4.

Возможности среды `pdetool` не исчерпываются решением стационарной задачи теплопроводности. Следующие разделы посвящены подробному описанию допустимых классов задач, некоторым приемам, полезным при конструировании геометрии области, управлению процессом решения и видом получающихся графиков.

Описание возможностей ToolBox PDE

Интерфейс среды `pdetool` облегчает доступ пользователя к функциям, входящим в состав ToolBox PDE. Читатель, знакомый с методом конечных элементов и владеющий навыками программирования в MatLab, может использовать функции этого ToolBox для написания собственных приложений или реализовывать разнообразные алгоритмы решения задачи, создавая новые функции. Стандартный набор функций ToolBox PDE позволяет решать достаточно широкий спектр *двумерных задач в ограниченных* областях, включая:

- задачи теории упругости;
- стационарные и нестационарные задачи теплопроводности;
- течения в пористых средах;
- задачи диффузии;
- ламинарное течение жидкости;
- электростатические задачи;
- распространение акустических и электромагнитных волн;
- определение собственных колебаний конструкций.

Обобщая приведенный список, можно сказать, что практически любая задача, описываемая уравнением или системой уравнений в частных производных, может быть решена при помощи ToolBox PDE. В следующих разделах приведены типы граничных задач для уравнений в частных производных, решение которых может быть найдено с использованием ToolBox PDE. Область обозначена через Ω, а ее граница — $\partial\Omega$.

Эллиптическое уравнение

Стационарная задача теплопроводности, постановке и решению которой посвящены предыдущие разделы, описывается эллиптическим дифференциальным уравнением. Эллиптическое дифференциальное уравнение в общем случае имеет вид:

$$-\nabla \cdot (c\nabla u) + au = f$$

в области Ω.

Граница $\partial\Omega$ области может быть поделена на несколько участков, на каждом из них ставятся либо условия Дирихле $hu = r$, либо обобщенное условие Неймана $n \cdot (c\nabla u) + qu = g$, где n — вектор внешней нормали к границе. В примере, приведенном в предыдущем разделе (см. рис. 15.1), коэффициенты уравнения и граничных условий были постоянны, однако ToolBox PDE способен решать задачи с переменными коэффициентами $c(x, y)$, $a(x, y)$ и правой частью $f(x, y)$ уравнения и $h(x, y)$, $r(x, y)$ $q(x, y)$, $g(x, y)$ граничных условий. Более того, нелинейные задачи, в которых коэффициенты зависят от искомой функции $u(x, y)$, также могут быть решены в ToolBox PDE. Допустимы комплексные значения всех коэффициентов.

Настройка среды `pdetool` на решение эллиптического уравнения осуществляется выбором пункта **Generic Scalar** подменю **Application** меню **Options** или при помощи раскрывающегося списка, размещенного на панели инструментов среды `pdetool`. В диалоговом окне **PDE Specification** следует установить переключатель **Type of PDE** в положение **Elliptic**. Диалоговое окно **PDE Specification** настроено на ввод коэффициентов эллиптического уравнения.

Переменные коэффициенты и правая часть уравнения

Разберем решение дифференциального уравнения с переменной правой частью (переменные коэффициенты уравнения задаются аналогично), зависящей от x и y, на примере уравнения

$$\nabla \cdot \nabla u = 16(x^2 + y^2).$$

Область является кругом единичного радиуса с центром в начале координат, на границе которого задано условие Дирихле $u = 1$. Нарисуйте единичный круг в среде `pdetool`, используйте диалоговое окно **Object Dialog** для точного определения радиуса и центра. Выберите эллиптический тип уравнения, задайте коэффициенты $a = 1$ и $c = 0$, а для правой части f введите выражение *в соответствии с правилами поэлементных операций* с массивами, используя переменные `x` и `y`.

Поэлементные операции с массивами описаны в главе 2.

Требуемое выражение должно выглядеть так: `-16*(x.^2+y.^2)`, знак минус соответствует общему виду дифференциального уравнения, приведенного в предыдущем пункте. Инициализируйте триангуляцию и решите задачу на первой предлагаемой сетке.

Полученное приближенное решение можно сравнить с точным $(x^2 + y^2)^2$, построив их разность в области. Среда `pdetool` позволяет выводить график

не только приближенного решения, но и функции, в которую это решение входит. В нашем случае такой функцией является погрешность, т. е. разность приближенного и точного решений. Выберите в меню **Plot** пункт **Parameters**, появляется диалоговое окно **Plot Selection** (рис. 15.9), которое позволяет выбрать тип и вид визуализируемого результата.

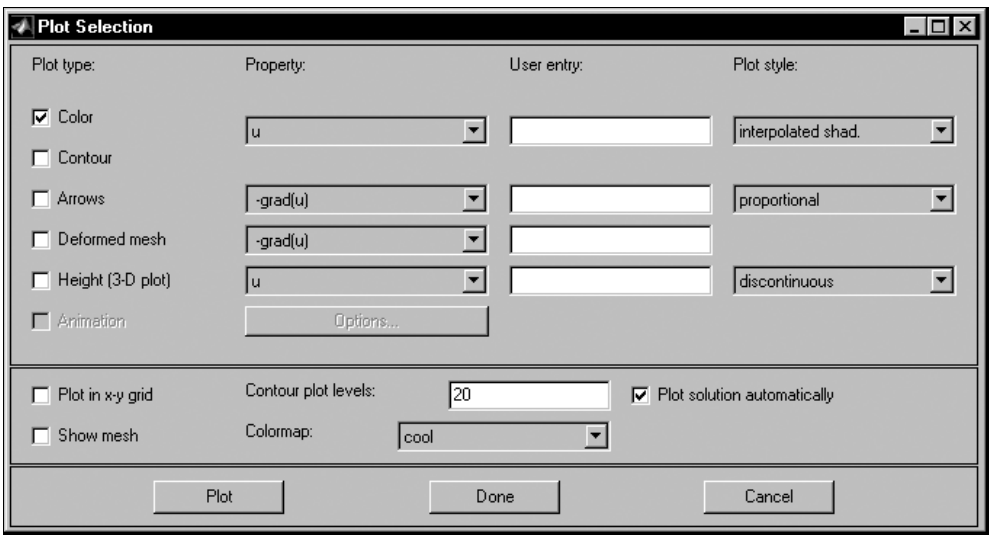


Рис. 15.9. Диалоговое окно **Plot Selection**

Верхняя панель окна **Plot Selection** организована в виде таблицы, левая колонка которой содержит флаги, соответствующие способу визуализации результатов. Столбик **Property** состоит из раскрывающихся списков, предназначенных для выбора отображаемой функции от приближенного решения. Опция **user entry** делает доступной строку ввода в колонке **User entry**. Введите в ней выражение для вычисления погрешности `u-(x.^2+y.^2).^2`. Нажатие кнопки **Plot** приводит к отображению залитого цветом контурного графика погрешности. Раскрывающийся список **Plot style** позволяет получать графики с плавным изменением цвета на каждом элементе (опция **interpolated shad**) или выбрать постоянный цвет на элементе (опция **flat shading**). Решайте задачу, поставленную в начале раздела, уменьшая ячейки сетки, и наблюдайте за изменением погрешности решения.

Переменные коэффициенты уравнения задаются аналогично с использованием переменных `x` и `y` и поэлементных операций. Коэффициенты, входящие в граничные условия, могут зависеть либо от `x` и `y`, либо от значения `s` параметра части границы. В начальной точке границы `s` равно нулю, в конечной — единице. Направление границы определяется стрелкой в режиме

задания граничных условий. Значение параметра пропорционально длине части границы.

Параболическое и гиперболическое уравнения

Параболическое и гиперболическое уравнения, которые могут быть решены в ToolBox PDE, выглядят следующим образом:

$$d\frac{\partial u}{\partial t} - \nabla \cdot (c\nabla u) + au = f; \qquad d\frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c\nabla u) + au = f.$$

Коэффициенты d, c, a и правая часть f могут зависеть как от x и y, так и от времени t. Граничные условия, как и в случае эллиптического уравнения, на одних частях границы являются условиями Дирихле, а на других — Неймана. Допускается зависимость коэффициентов граничных условий от времени и координат или параметра границы. Задача, описываемая параболическим и гиперболическим уравнениями, требует определения начальных условий в нулевой момент времени. Решение при $t = 0$ может зависеть от x и y. ToolBox PDE позволяет найти решение только в ограниченной области Ω.

Среда `pdetool` предоставляет пользователю возможность вывода наглядных графиков полученного приближенного решения. Можно визуализировать решение в любой момент времени, либо проследить за развитием процесса в отдельном графическом окне, в которое выводится последовательность слайдов, содержащих залитые контурные графики решения в различные моменты времени.

Пример нестационарной задачи

В качестве примера рассмотрим нестационарную задачу о распределении тепла в области, изображенной на рис. 15.10. Границы прямоугольника теплоизолированы, а края отверстия подвергаются нагреву, температура изменяется линейно со временем. Внутри области нет распределенных источников тепла, в начальный момент времени температура равна нулю во всей области. Нас будет интересовать изменение температуры на протяжении пятнадцати секунд.

Задайте геометрию области и настройте среду `pdetool` на решение задачи теплопроводности. Перейдите к диалоговому окну **PDE Specification** и выберите параболический тип уравнения, установив переключатель **Parabolic**. Введите единицы в строки, соответствующие плотности `rho`, теплоемкости `C` и коэффициенту теплопроводности `k`, а `Q`, `h` и `Text` обнулите. На границах прямоугольника поставьте условия равенства нулю потока тепла, а для ок-

ружности введите формулу `100*t`, переменная `t` используется для обозначения времени.

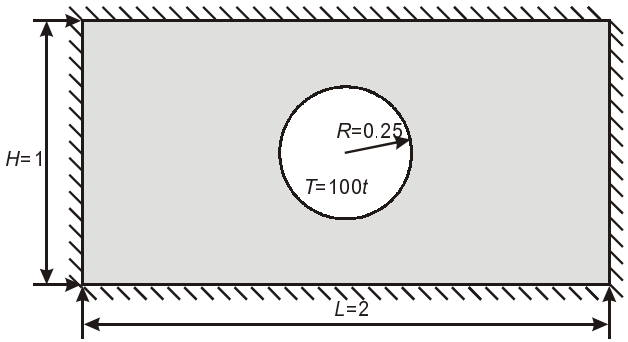


Рис. 15.10. Задача о нестационарном распределении температуры

Задайте распределение температуры в начальный момент времени и значения времени, в которые следует найти приближенное решение. Выберите пункт **Parameters** в меню **Solve**. Появляется диалоговое окно **Solve Parameters**, вид которого соответствует типу решаемой задачи. Для нестационарной задачи теплопроводности, описываемой параболическим уравнением, окно **Solve Parameters** (рис. 15.11) позволяет установить в строке **Time** вектор моментов времени, а в строке **u(t0)** — начальное распределение температуры, которое в общем случае может зависеть от x и y.

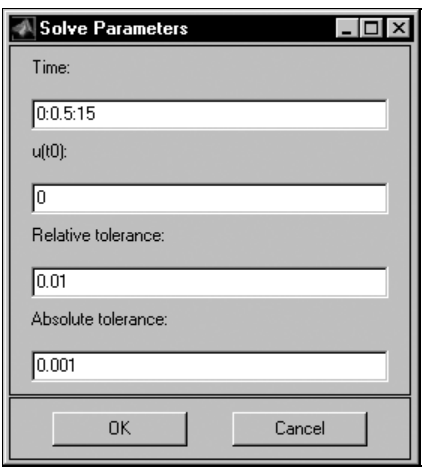


Рис. 15.11. Диалоговое окно **Solve Parameters** для нестационарной задачи

Введите ноль для начального распределения и задайте вектор моментов времени от нуля до пятнадцати с шагом 0.5, примените двоеточие для генерации вектора значений.

Замечание

Часто требуется найти решение не в равноотстоящие моменты времени. Больший интерес может представлять начало процесса распределения тепла. Можно задать в **u(t0)** вектор значений, например, [0 0.1 0.2 0.3 0.5 1 4 10 15], или логарифмическую шкалу времени командой `logspace(-2, log10(15), 20)`, которая генерирует 20 равноотстоящих в логарифмической шкале точек от 10^{-2} до 15.

Инициализируйте триангуляцию, уменьшите шаг сетки в несколько раз и решите задачу. Решение может занять достаточно много времени, в зависимости от производительности компьютера. Процесс решения сопровождается выводом информации в командное окно. В окне `pdetool` появляется распределение температуры в области в конечный момент времени при $t = 15$. Для того чтобы проследить за динамикой процесса, следует установить в диалоговом окне **Plot Selection** флаг **Animation** и воспользоваться кнопкой **Options** для определения параметров анимированных результатов. Нажатие на **Options** приводит к открытию окна **Animation Options**, изображенного на рис. 15.12.



Рис. 15.12. Диалоговое окно **Animation Options**

Строка ввода **Animation rate (fps)** служит для задания числа кадров в секунду (frames per second), которое будет использоваться при отображении распределения температуры в зависимости от времени. Число повторов анимации устанавливается в строке **Number of repeats**. Введите желаемые значения, или оставьте те, что используются по умолчанию, закройте **Animation Options**

и нажмите кнопку **Plot** в окне **Plot Selection**. Динамическое распределение температуры в области отображается в отдельном графическом окне.

Задача на собственные значения

ToolBox PDE предоставляет возможность решения задачи на собственные значения

$$-\nabla \cdot (c\nabla u) + au = \lambda du,$$

причем коэффициенты c, a и d могут зависеть от x и y в области Ω. В задаче на собственные значения допустимы только однородные граничные условия Дирихле $u = 0$ или Неймана $n \cdot (c\nabla u) + qu = 0$. Решением задачи являются собственные значения и собственные функции.

Настройка среды `pdetool` на решение задачи на собственные значения производится установкой переключателя **Eigenmodes** в диалоговом окне **PDE Specification**. Интервал поиска собственных значений определяется в окне **Solve Parameters**, которое в данном случае содержит единственную строку ввода **Eigenvalue search range**. Интервал задается вектор-строкой или вектор-столбцом из двух элементов.

Когда решение найдено, в окне `pdetool` отображается график собственной функции, отвечающей первому найденному значению из введенного интервала. Диалоговое окно **Plot Selection** содержит раскрывающийся список **Eigenvalue**, который позволяет выбрать собственное значение и отобразить в окне среды `pdetool` график соответствующей собственной функции.

Системы дифференциальных уравнений

Функции, входящие в состав ToolBox PDE, позволяют решить систему дифференциальных уравнений произвольной размерности. Среда `pdetool` оперирует только с системой второго порядка:

$$-\nabla \cdot (c_{11}\nabla u_1) - \nabla \cdot (c_{12}\nabla u_2) + a_{11}u_1 + a_{12}u_2 = f_1;$$

$$-\nabla \cdot (c_{12}\nabla u_1) - \nabla \cdot (c_{22}\nabla u_2) + a_{21}u_1 + a_{22}u_2 = f_2.$$

Граничные условия на различных частях $\partial\Omega$ могут быть трех типов:

Дирихле

$$h_{11}u_1 + h_{12}u_2 = r_1;$$

$$h_{21}u_1 + h_{22}u_2 = r_2;$$

Неймана

$$n \cdot (c_{11}\nabla u_1) + n \cdot (c_{12}\nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1;$$

$$n \cdot (c_{21}\nabla u_1) + n \cdot (c_{22}\nabla u_2) + q_{21}u_1 + q_{22}u_2 = g_2$$

или смешанные граничные условия

$$h_{11}u_1 + h_{12}u_2 = r_1;$$

$$n\cdot(c_{11}\nabla u_1) + n\cdot(c_{12}\nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1 + h_{11}\mu;$$

$$n\cdot(c_{21}\nabla u_1) + n\cdot(c_{22}\nabla u_2) + q_{21}u_1 + q_{22}u_2 = g_2 + h_{12}\mu.$$

Нестационарные задачи, описываемые системами дифференциальных уравнений, также могут быть решены в ToolBox PDE. Выбор опции **Generic System** в подменю **Application** меню **Options** (или в раскрывающемся списке на панели инструментов) настраивает среду `pdetool` на решение подобной системы уравнений.

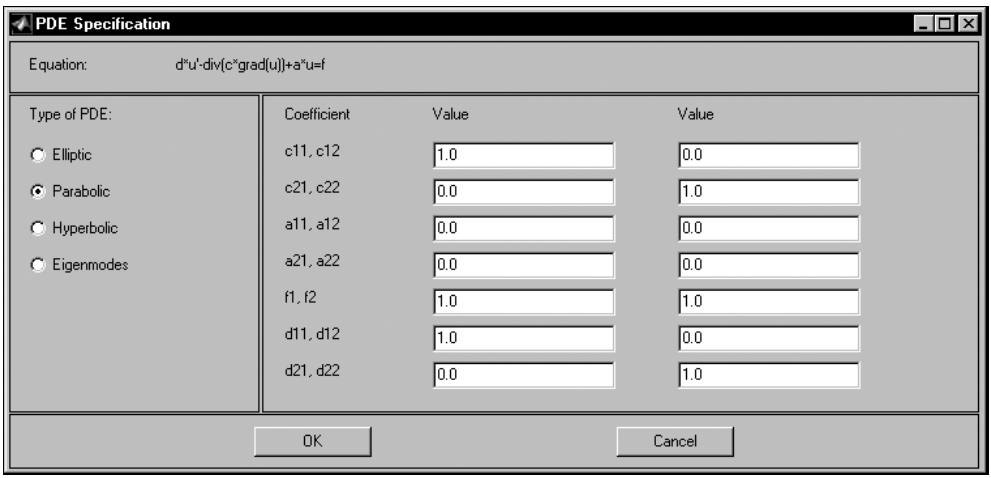


Рис. 15.13. Диалоговое окно **PDE Specification**, настроенное на решение системы параболических уравнений

Диалоговое окно **PDE Specification** позволяет задать коэффициенты стационарных и нестационарных систем (рис. 15.13) или перейти к решению задачи на собственные значения. Окно **Boundary Condition** содержит переключатели **Neumann**, **Dirichlet** и **Mixed** для выбора одного из трех типов условий, перечисленных выше, и строки ввода для задания коэффициентов граничных условий.

Решением системы является вектор-функция из двух компонентов $[u_1(x,y), u_2(x,y)]$. Раскрывающиеся списки диалогового окна **Plot Selection** предоставляют возможность визуализировать первый или второй компоненты решения (они обозначены через **u** и **v**), вывести график заданной пользователем функции от компонентов решения (опция **user entry**), или отобразить решение векторным полем (флаг **Arrows**).

Параметры триангуляции и управление процессом решения

Начальная триангуляция области достаточно грубая, дальнейшее уменьшение шага сетки производится выбором пункта **Refine Mesh** меню **Mesh** или нажатием соответствующей кнопки на панели инструментов среды `pdetool`. Диалоговое окно **Mesh Parameters**, открывающееся при переходе к пункту **Parameters** меню **Mesh**, предназначено для задания параметров триангуляции, в частности максимальная длина стороны элементов вводится в строке **Maximum eadg size**.

Среда `pdetool` позволяет пользователю выбирать некоторые алгоритмы решения в диалоговом окне **Solve Parameters**, доступ к которому производится выбором пункта **Parameters** меню **Solve**. Вид данного окна зависит от типа уравнения. Если решается задача для эллиптического уравнения, то окно имеет вид, приведенный на рис. 15.14 (для наглядности включены флаги вверху каждой панели).

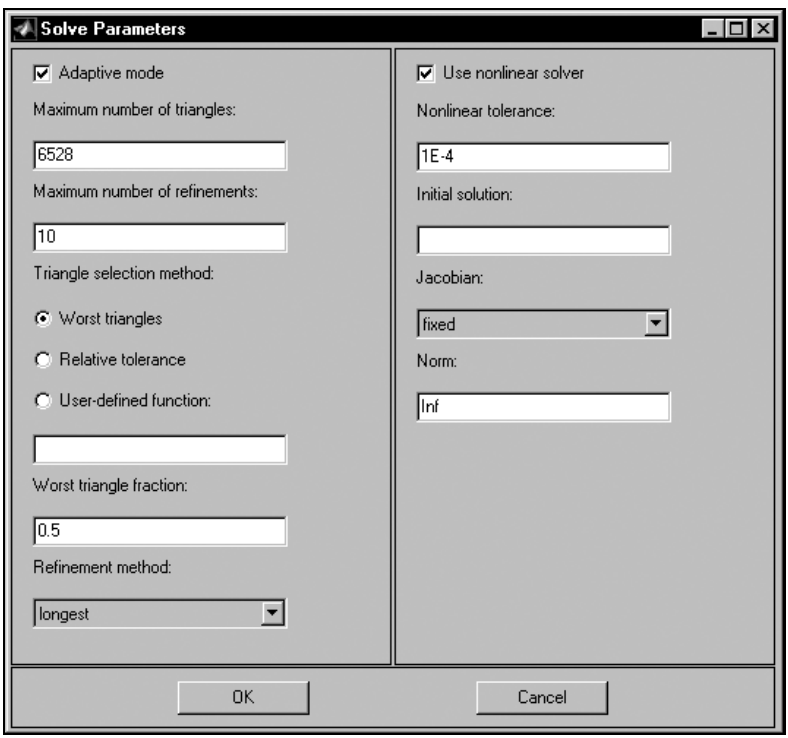


Рис. 15.14. Диалоговое окно **Solve Parameters** для эллиптического уравнения

Левая панель окна **Solve Parameters** предназначена для решения эллиптических задач в адаптивном режиме, смысл которого заключается в автоматическом уменьшении шага сетки в ходе решения с целью повышения эффективности вычислений. Установка флага **Adaptive mode** настраивает среду `pdetool` на решение в адаптивном режиме и разрешает доступ к элементам управления, расположенным на данной панели.

Строка **Maximum number of tiangles** позволяет определить максимальное количество треугольников, а задание `Inf`, наоборот, не ограничивает нижний предел шага сетки. Наибольшее число попыток уточнения вводится в строке **Maximum number of refinements**. Уточнение сетки может происходить на основе одного из трех критериев:

- по наихудшим треугольникам (переключатель **Worst triangles**), в строке **Worst triangle fraction** можно установить условный критерий качества от нуля до единицы;
- по относительной погрешности (переключатель **Relative tolerance**), в одноименной строке следует ввести погрешность;
- по критерию пользователя (переключатель **User-defined function**), дополнительная информация содержится в справочной системе по ToolBox PDE.

Способ уменьшения шага сетки выбирается в раскрывающемся списке **Refinement method**, допустимо либо регулярное уточнение (**regular**), при котором каждая сторона треугольника делится пополам, образуя четыре малых треугольника, либо по наибольшей стороне треугольника (**longest**).

Решение нелинейных граничных задач для эллиптических уравнений требует применения нелинейного солвера, который включается флагом **Use nonlinear solver**, расположенным на правой панели окна **Solve Parameters**. Описание нелинейного солвера выходит за рамки данной книги, подробная информация приведена в справочной системе по ToolBox PDE.

Конструирование геометрии области

Создание плоских областей сложной формы в среде `pdetool` требует понимания принципов конструктивной блочной геометрии или CSG (constructive solid geometry). Среда `pdetool` предлагает пользователю ряд средств, облегчающих задание геометрии области.

Геометрические примитивы

Плоская ограниченная область является объединением, пересечением или разностью геометрических примитивов. В режиме рисования из меню **Draw** или панели инструментов доступны инструменты рисования:

- прямоугольника от угла (пункт **Rectangle/square**);

- прямоугольника из центра (пункт **Rectangle/square (centered)**);
- эллипса от угла прямоугольной рамки, содержащей эллипс (пункт **Ellipse/circle**);
- эллипса из центра (пункт **Ellipse/circle (centered)**);
- многоугольника (пункт **Polygon**).

Использование первых четырех инструментов при удержании в нажатом состоянии клавиши <Ctrl> позволяет нарисовать квадрат или круг соответственно. Многоугольник обязательно должен быть замкнут, выбор другого инструмента при разомкнутом контуре прямоугольника приводит к исчезновению контура. Треугольник является простейшим видом многоугольника. Каждый геометрический примитив при создании получает имя, состоящее из типа примитива и номера, определяемого порядком создания. Символы `E` и `C` означают эллипс и круг, `R` и `SQ` — прямоугольник и квадрат, а `P` — многоугольник.

Характеристики размеров примитивов и их положение устанавливаются в диалоговом окне **Object Dialog**, которое позволяет точно определить соответствующие величины и, кроме того, задать любое имя. Окно **Object Dialog** появляется при двойном щелчке по области объекта. Если требуемый объект накрывается более крупным, то следует применить двойной щелчок в области имени меньшего объекта. Вид окна **Object Dialog** зависит от типа объекта, например, прямоугольник и квадрат задаются координатами левого нижнего угла, шириной и высотой (см. рис. 15.3). Круг определяется координатами центра и радиусом, эллипс — координатами центра и величинами полуосей, кроме того, в свойства эллипса включен угол поворота против часовой стрелки в градусах. Диалоговое окно, соответствующее многоугольнику, приведено на рис. 15.15. Раскрывающийся список **Coordinates** предназначен для выбора вершины, координаты которой помещаются в строки ввода **X-value edit box** и **Y-value edit box** и становятся доступны для изменения.

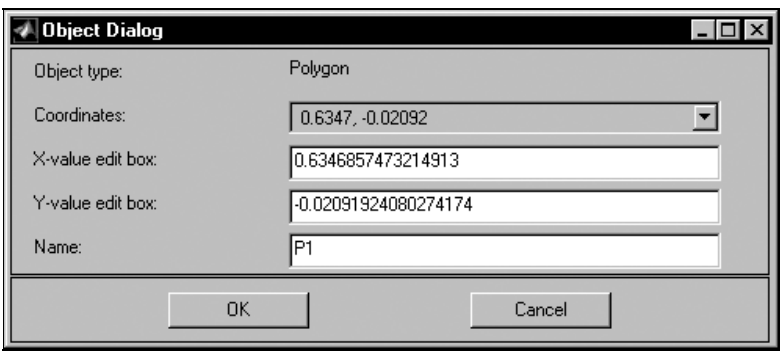


Рис. 15.15. Диалоговое окно **Object Dialog** со свойствами многоугольника

Любой созданный объект можно повернуть на заданный угол, для чего следует выделить объект и перейти в меню **Draw** к пункту **Rotate**. Появляется диалоговое окно **Rotate**, содержащее строку ввода **Rotation (degrees)** для значения в градусах угла поворота против часовой стрелки и флаг **Use center-of-mass**. Установленный флаг приведет к повороту объекта вокруг его центра масс, а выключенный — делает доступным строку ввода **Rotation center**, предназначенную для определения центра вращения.

Задание структуры области

Структура области, составленной из геометрических примитивов, определяется формулой в строке **Set formula** среды `pdetool`, в которую входят имена примитивов, знаки плюс, минус, умножить и скобки. Все создаваемые примитивы по умолчанию объединяются, между их именами ставится знак плюс. При редактировании формулы следует придерживаться перечисленных ниже правил:

- минус предназначен для вычитания примитива, операция имеет наивысший приоритет;
- плюс означает объединение примитивов, а умножить — пересечение, операции имеют одинаковый приоритет;
- для изменения приоритета используются круглые скобки.

Создайте прямоугольник с отверстиями, изображенный на рис. 15.16, нарисуйте прямоугольник `R1` и три круга `C1`, `C2`, `C3` и введите в строку **Set formula** выражение `R1-(C1+C2+C3)`.

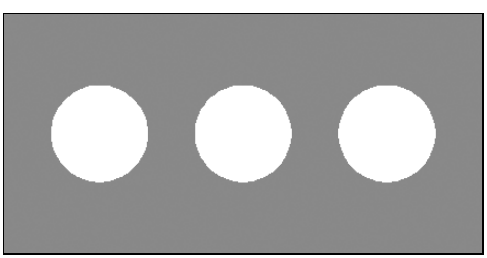


Рис. 15.16. Прямоугольник с отверстиями

В режиме рисования на области присутствуют все созданные примитивы, даже если они вычитаются из других примитивов. Для отображения реальной геометрии области перейдите в режим дифференциального уравнения (пункт **PDE Mode** в меню **PDE**). Обратите внимание, что формула `R1-C1-C2-C3` в данном случае является неверной, т. к. согласно приоритету операций сначала выполняется вычитание `C2-C3`, что лишено смысла.

Замечание

Редактирование формулы доступно только в режиме рисования, в который можно перейти в любой момент, даже после того, как решение найдено, и внести изменения в геометрию области.

Нарисуйте прямоугольник со скругленными углами, такой, как изображен на рис 15.17. Очевидно, что следует вычесть из прямоугольника четыре небольших квадрата, центры которых расположены в вершинах прямоугольника, и объединить полученную область с четырьмя кругами с центрами в вершинах прямоугольника и диаметром, равным стороне удаленных квадратов.

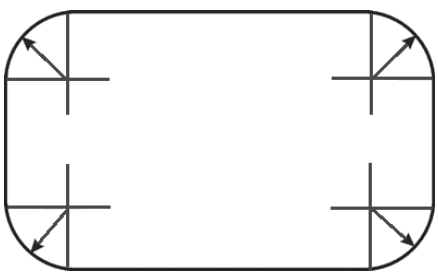


Рис. 15.17. Прямоугольник со скругленными углами

Начинающему пользователю следует иметь в виду, что при конструировании геометрии области многие действия, в частности, удаление примитива, *необратимы*. Пункт **Undo** меню **Edit** позволяет только отменить последнюю нарисованную линию многоугольника. Хорошим решением проблемы является периодическое сохранение геометрии области, в случае неудачных изменений можно воспользоваться резервной копией.

Объединение объектов позволяет просто задать геометрию области, однако следует иметь в виду, что при объединении сохраняются внутренние границы. Если область однородна, то следует удалить ненужные границы при помощи пункта **Remove Subdomain Border** меню **Boundary**, предварительно выделив их в режиме границ (пункт **Boundary Mode** меню **Boundary**). Все внутренние границы удаляются выбором пункта **Remove All Subdomain Borders**. Задачи в областях, составленных из материалов с разными свойствами, не предполагают удаления внутренних границ.

Композитные материалы

Исследуем стационарное распределение тепла в сечении теплоизолирующей оболочки с двумя нагревающимися стержнями, коэффициенты теплопроводности оболочки и стержней различны (рис. 15.18). Внешняя граница

оболочки поддерживается при температуре 0°, в оболочке нет источников тепла, а плотность источников тепла в сечении стержней равна 1000.

Составьте область из трех кругов, формула должна иметь вид `C1+C2+C3`. Выберите тип решаемой задачи (Heat transfer). В режиме задания граничных условий установите нулевую температуру на внешней границе, не удаляйте внутренние границы! Перейдите в режим дифференциального уравнения и для каждой области задайте коэффициент теплопроводности и плотность источников тепла. Двойной щелчок мыши по каждому из примитивов приводит к появлению диалогового окна **PDE Specification**, служащего для задания коэффициентов и правой части именно для данного примитива. Можно объединить несколько примитивов в группу щелчком мыши с одновременным удержанием клавиши <Shift> и произвести установки для группы подобластей.

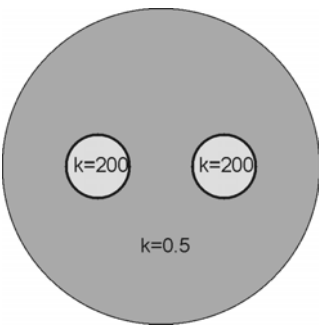


Рис. 15.18. Задача о распределении температуры в теплоизолирующей оболочке

Выберите подходящую триангуляцию и решите задачу. В окне среды `pdetool` отображается искомое распределение температуры в сечении рассматриваемой конструкции (рис. 15.19).



Рис. 15.19. Распределение температуры в теплоизолирующей оболочке

Использование сетки

Среда `pdetool` предоставляет пользователю возможность изменять пределы осей, наносить вспомогательную сетку и использовать ее при рисовании и перемещении объектов. Отображение сетки в окне среды производится выбором пункта **Grid** меню **Draw**. Параметры сетки задаются в диалоговом окне **Grid Spacing**, которое выводится при выборе одноименного пункта в меню **Options**. Строки ввода **X-axis linear spacing** и **Y-axis linear spacing** предназначены для задания координат линий сетки по соответствующим направлениям. Используется либо вектор с постоянным шагом (при помощи двоеточия), либо координаты, разделенные пробелом. Равномерной сетки может быть недостаточно, координаты дополнительных линий вводятся в строках **X-axis extra ticks** и **Y-axis extra ticks**.

Пределы осей устанавливаются в диалоговом окне **Axes Limits**, которое появляется при выборе соответствующего пункта меню **Options**. Область проще конструировать, когда оси имеют равный масштаб; тогда не происходит, например, искажения кругов. Для обеспечения равного масштаба следует выбрать пункт **Axes Equal**.

Рисование сложных областей по заданному чертежу облегчается, если включить привязку к сетке, для чего следует выбрать в меню **Options** пункт **Snap**. Теперь вершины прямоугольников и квадратов, центры окружностей и эллипсов при перетаскивании и рисовании могут находиться только на линиях сетки. Рамки, ограничивающие эллипсы и окружности, также размещаются на линиях сетки. Вершины многоугольников при их рисовании допустимы только в узлах. Выбор шага сетки, соответствующего размерам мелких элементов чертежа, позволяет существенно ускорить создание области.

Использование функций ToolBox PDE

Среда `pdetool` с графическим интерфейсом пользователя предназначена лишь для облегчения доступа к ядру ToolBox PDE — набору более чем из пятидесяти функций, реализующих основные этапы решения задачи от задания области, уравнения и граничных условий до визуализации результата. Механизм вызова данных функций скрыт от пользователя при работе в среде. Многие функции имеют достаточно сложный интерфейс, их использование требует понимания метода конечных элементов. Следующие разделы посвящены описанию функций, связанных с основными этапами решения граничных задач для дифференциальных уравнений.

Задание геометрии области

Имеется два способа определения геометрии области, в которой решается задача. Первый заключается в параметрическом задании частей границ об-

ласти в файл-функции, имеющей определенный формат. Второй, более простой способ, состоит в задании области в среде `pdetool`, последующем экспорте информации в переменные рабочей среды и преобразовании в формат, понятный другим функциям ToolBox.

Нарисуйте в среде `pdetool` прямоугольник шириной два и высотой один, центр которого расположен в начале координат. Выберите в меню **Draw** пункт **Export Geometry Description, Set Formula, Labels**, появляется диалоговое окно **Export**, изображенное на рис. 15.20. Данное окно предлагает сохранить CSG-модель (конструктивную блочную геометрию) в глобальных переменных рабочей среды, а именно:

- матрицу геометрии области (Geometry description matrix) в массиве `gd`;
- формулу связи геометрических примитивов в строке `sf`;
- соответствие между столбцами `gd` и названиями областей в `sf` в массиве `ns`.

Нажмите кнопку **OK** и займитесь изучением содержимого экспортированных массивов.

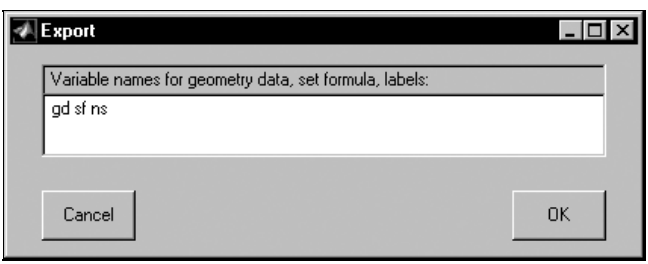


Рис. 15.20. Диалоговое окно **Export**

Число столбцов матрицы геометрии области `gd` совпадает с числом геометрических примитивов, составляющих область. В рассматриваемом примере матрица состоит из одного столбца, соответствующего прямоугольнику. Выведите содержимое `gd` в окно рабочей среды:

```
>> gd
gd =
    3.0000
    4.0000
   -1.0000
    1.0000
    1.0000
   -1.0000
```

```
    0.5000
    0.5000
   -0.5000
   -0.5000
```

Первый элемент столбца, равный трем, означает, что данный столбец соответствует прямоугольнику, элемент второй строки равен числу сторон. Очевидно, что в прямоугольнике четыре стороны, но прямоугольник является частным случаем многоугольника, формат хранения которого описан ниже. В остальные элементы столбца записаны координаты вершин прямоугольника.

Всего допустимы четыре типа геометрических примитивов: круг, многоугольник, прямоугольник и эллипс. Первый элемент столбца матрицы геометрии области, соответствующей кругу, равен единице, второй и третий являются координатами центра, а четвертый — радиусом. В случае эллипса первый элемент равен четырем, далее хранятся координаты центра и величины полуосей, а последний шестой элемент содержит угол поворота. Многоугольник идентифицируется двойкой в первом элементе вектора, число сторон записано во второй элемент, а далее следуют координаты вершин в порядке их создания.

Функции, связанные с триангуляцией и заданием граничных условий, используют другой формат описания геометрии области — *матрицу декомпозиционной геометрии* (Decomposed Geometry matrix). В данном формате хранится информация о частях границы области, которыми могут быть: отрезок, часть дуги окружности или эллипса. Переход от конструктивной блочной геометрии к декомпозиционной производится при помощи функции `decsg`, во входных аргументах которой задаются переменные с информацией о конструктивной блочной геометрии, в рассматриваемом случае `gd`, `sf` и `ns`. Выходным аргументом `decsg` является матрица декомпозиционной геометрии. Преобразуйте экспортированные величины, вызвав `decsg`, например из командной строки:

```
dl = decsg(gd, sf, ns)
dl =
    2.0000    2.0000    2.0000    2.0000
   -1.0000    1.0000    1.0000   -1.0000
    1.0000    1.0000   -1.0000   -1.0000
    0.5000    0.5000   -0.5000   -0.5000
    0.5000   -0.5000   -0.5000    0.5000
         0         0         0         0
    1.0000    1.0000    1.0000    1.0000
```

Каждый столбец матрицы декомпозиционной геометрии отвечает элементу границы. Очевидно, что прямоугольник представляется четырьмя отрезками.

Двойка в первом элементе столбца означает, что столбец соответствует отрезку, второй и третий элемент содержат абсциссы начала и конца отрезка, а четвертый и пятый — ординаты. Преобразование в декомпозиционную геометрию предполагает, что вся область разбивается на некоторые непересекающиеся *минимальные подобласти*. Идентификаторы подобластей, расположенных слева и справа от части границы (в данном случае отрезка) записаны в шестом и седьмом элементе. Положение определяется движением от начала части границы до ее конца, в случае области из одного прямоугольника он же и является минимальной подобластью, поэтому слева от частей границ (отрезков) нет подобластей. Столбцы матрицы декомпозиционной геометрии, соответствующие дугам эллипса или окружности, содержат в остальных элементах информацию о центре и радиусе окружности или величине полуосей эллипса и угле поворота.

Функция `pdegplot` предназначена для вывода области в отдельное графическое окно, входным аргументом `pdegplot` является матрица или файл-функция декомпозиционной геометрии. В рассматриваемом примере `pdegplot(dl)` отображает оси, содержащие прямоугольник.

Структуру файл-функции с описанием декомпозиционной геометрии проще всего понять, автоматически создав ее при помощи функции `wgeom`. Входными аргументами `wgeom` являются матрица декомпозиционной геометрии и имя M-файла, в который следует записать файл-функцию. Выходной аргумент принимает значение минус единица при неудачной попытке записи в файл.

Сгенерируйте файл-функцию `recgeom`, которая описывает прямоугольную область, задаваемую матрицей *dl*. Убедитесь, что в глобальной переменной `dl` содержится матрица декомпозиционной геометрии области, создание которой описано выше. В командном окне выполните `wgeom(dl, 'recgeom')`. В текущем каталоге MatLab появился файл recgeom.m, структура которого приведена в листинге 15.1.

Листинг 15.1. Файл-функция `recgeom` геометрии области

```
function [x,y]=recgeom(bs,s)
%RECGEOM      Gives geometry data for the recgeom PDE model.
%   NE=RECGEOM gives the number of boundary segments
%   D=RECGEOM(BS) gives a matrix with one column for each boundary
%   segment specified in BS.
%   Row 1 contains the start parameter value.
%   Row 2 contains the end parameter value.
%   Row 3 contains the number of the left-hand regions.
%   Row 4 contains the number of the right-hand regions.
```

```
%   [X,Y]=RECGEOM(BS,S) gives coordinates of boundary points.
%   BS specifies the boundary segments and S the corresponding parameter
%   values. BS may be a scalar.

nbs=4;
if nargin==0,
  x=nbs; % number of boundary segments
  return
end
d=[
  0 0 0 0 % start parameter value
  1 1 1 1 % end parameter value
  0 0 0 0 % left hand region
  1 1 1 1 % right hand region
];
bs1=bs(:)';
if find(bs1<1 | bs1>nbs),
  error('Non-existent boundary segment number')
end
if nargin==1,
  x=d(:,bs1);
  return
end
x=zeros(size(s));
y=zeros(size(s));
[m,n]=size(bs);
if m==1 & n==1,
  bs=bs*ones(size(s)); % expand bs
elseif m~=size(s,1) | n~=size(s,2),
  error('bs must be scalar or of same size as s');
end
if ~isempty(s),
% boundary segment 1
ii=find(bs==1);
if length(ii)
x(ii)=(1-(-1))*(s(ii)-d(1,1))/(d(2,1)-d(1,1))+(-1);
y(ii)=(0.5-(0.5))*(s(ii)-d(1,1))/(d(2,1)-d(1,1))+(0.5);
end
```

```
% boundary segment 2
ii=find(bs==2);
if length(ii)
x(ii)=(1-(1))*(s(ii)-d(1,2))/(d(2,2)-d(1,2))+(1);
y(ii)=(-0.5-(0.5))*(s(ii)-d(1,2))/(d(2,2)-d(1,2))+(0.5);
end
% boundary segment 3
ii=find(bs==3);
if length(ii)
x(ii)=(-1-(1))*(s(ii)-d(1,3))/(d(2,3)-d(1,3))+(1);
y(ii)=(-0.5-(-0.5))*(s(ii)-d(1,3))/(d(2,3)-d(1,3))+(-0.5);
end

% boundary segment 4
ii=find(bs==4);
if length(ii)
x(ii)=(-1-(-1))*(s(ii)-d(1,4))/(d(2,4)-d(1,4))+(-1);
y(ii)=(0.5-(-0.5))*(s(ii)-d(1,4))/(d(2,4)-d(1,4))+(-0.5);
end
end
```

Файл-функция `recgeom` предназначена для получения координат `x` и `y` точек части границы с номером `bs`, соответствующих значению параметра `s`. Локальная переменная `nbs` содержит число частей границы, а матрица `d` — информацию о параметризации. Число столбцов *d* равно числу участков границы, в первый и второй элементы каждого столбца занесены начальное и конечное значения параметра ноль и единица, в третьем и четвертом элементах записаны номера примитивов, расположенных по левую и правую сторону от участка границы (по направлению возрастания параметров). В рассматриваемом случае имеется один примитив — сам прямоугольник — с номером один. В блоках, которым предшествуют комментарии `% boundary segment 1` — `% boundary segment 4`, происходит поиск нужной части границы, вычисление значений параметрически заданных функций и запись их в массивы `x` и `y`. Длины массивов `x`, `y` совпадают с длиной входного массива `s`, содержащего значения параметра.

Умение создавать файл-функции с декомпозиционной геометрией области необходимо в том случае, когда область не может быть сконструирована только с использованием стандартных геометрических примитивов, таких как круг, эллипс, квадрат, прямоугольник или многоугольник. Один из возможных подходов, удобный для начинающих пользователей, описан ниже.

В качестве примера рассмотрим написание файл-функции для области, приведенной на рис. 15.21. Область является криволинейным четырехугольником с верхней границей, задаваемой параболой $y = x^2$. Нижние углы прямые.

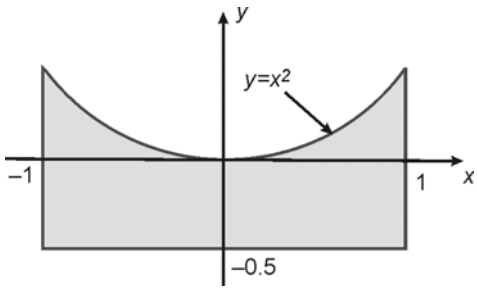


Рис. 15.21. Пример области, не состоящей из примитивов

Последовательность действий, направленных на получение файл-функции с декомпозиционной геометрией, приведена ниже.

1. Нарисуйте прямоугольник с центром в начале координат, шириной два и высотой один в среде `pdetool`.
2. Экспортируйте геометрию области из среды `pdetool` в глобальные массивы `gd`, `sf` и `ns`.
3. Перейдите к матрице декомпозиционной геометрии при помощи функции `decsg`, матрица возвращается в выходном элементе `dl = decsg(gd, sf, ns)`.
4. Сгенерируйте файл-функцию `mydomain` с декомпозиционной геометрией области, используя вызов `wgeom(dl, 'mydomain')`.
5. Откройте файл mydomain.m в редакторе M-файлов. Найдите нужный условный оператор `if`, в котором определяется параметрическое задание верхней стороны прямоугольника, и замените операторы присваивания на те, которые обеспечивают требуемый вид части границы.

Вызов функции `mydomain` из командной строки с первым аргументом, равным номеру части границы, и со вторым — значением параметра из интервала от нуля до единицы (например 0.5) — позволяет установить номер нужной части границы. Последовательно задавайте номера границ и следите за значениями `x` и `y`. Получение координат `x` и `y` точки, принадлежащей искомому участку, означает, что найден номер части границы, подлежащей изменению. В рассматриваемом примере верхняя сторона прямоугольника имеет номер, равный единице:

```
>> [x,y]=mydomain(1,0.5)
x =
     0
```

```
y =
    0.5000
```

Блок операторов, описывающих верхнюю сторону прямоугольника, снабжен в автоматически сгенерированной файл-функции `mydomain` комментариями `% boundary segment 1`. В данном блоке содержится линейная зависимость абсцисс и ординат точек от параметра `s`. Первый столбец матрицы *d*, инициализируемой в начале файл-функции, свидетельствует о том, что начальное значение параметра равно нулю, а конечное — единице. Направление границы, соответствующее увеличению параметра, легко устанавливается из параметрического задания (в данном случае слева направо). Измените операторы присваивания в блоке `% boundary segment 1`, обеспечив требуемую криволинейную границу в виде параболы. Очевидно, что параболический участок границы задается параметрическими зависимостями $x = 2s - 1$, $y = 0.5(2s-1)^2$. Операторы присваивания приведены в листинге 15.2.

Листинг 15.2. Параметрическое задание параболического участка границы

```
x(ii)=2*s(ii)-1;
y(ii)=0.5*(2*s(ii)-1).^2
```

Сохраните файл-функцию и убедитесь, что она соответствует области, изображенной на рис. 15.21. Постройте геометрию области при помощи функции `pdegplot`. Вызов `pdegplot('mydomain')` приводит к появлению графического окна с границей (см. рис. 15.22).

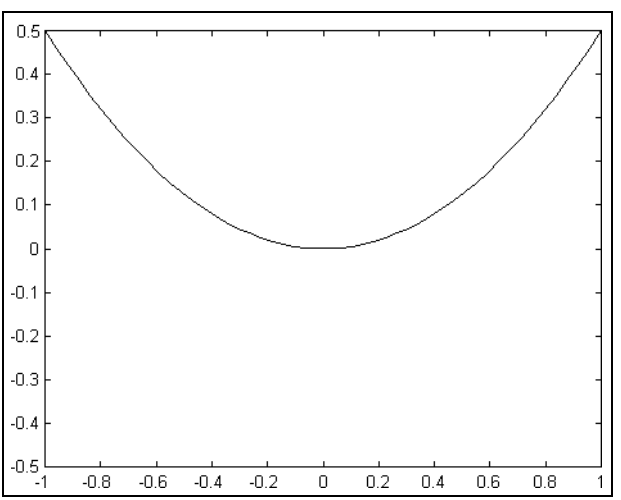


Рис. 15.22. Геометрия области (`pdegplot('mydomain')`)

Вышеописанный подход позволяет просто создать файл-функцию с декомпозиционной геометрией области, изменяя автоматически сгенерированную файл-функцию для нарисованной заготовки в среде `pdetool`.

Наличие матрицы декомпозиционной геометрии или файл-функции открывает доступ ко многим возможностям ToolBox PDE, в частности к автоматической триангуляции и уменьшению шага сетки.

Триангуляция

Функция `initmesh` триангулирует плоскую область, ее входным аргументом является матрица декомпозиционной геометрии. Три выходных аргумента (массива) содержат информацию о триангуляции. Вызов

```
[p, e, t] = initmesh(dl);
```

приводит к заполнению трех матриц *p*, *e* и *t*. В матрицу *p*, состоящую из двух строк, заносятся координаты узлов сетки. Матрица *e* содержит информацию об узлах, расположенных на границах минимальных подобластей, в матрице *t* хранится соответствие между локальной и глобальной нумерациями узлов.

Замечание

Понять структуру массивов `p`, `e` и `t` проще всего следующим образом. В среде `pdetool` следует инициализировать сетку, например для прямоугольника, причем лучше установить достаточно крупный размер стороны элемента в диалоговом окне **Mesh Parameters**. Пронумеровать узлы и элементы позволяет выбор пунктов **Show Node Labels** и **Show Triangle Labels** меню **Mesh**. Далее необходимо экспортировать сетку в глобальные массивы при помощи пункта **Export Mesh**, их содержимое можно посмотреть в рабочей среде.

Функция `initmesh` может содержать дополнительные аргументы, задаваемые парами: `название свойства, значение`. Например, свойство `Hmax` означает максимально допустимое значение длины стороны треугольного элемента.

Полученную сетку можно отобразить в отдельном окне при помощи функцию `pdemesh`, входными аргументами которой являются вышеперечисленные массивы. Последовательность команд, приведенная ниже

```
[p, e, t] = initmesh(dl, 'Hmax', 1.0);
pdemesh(p, e, t)
```

инициализирует и выводит сетку, приведенную на рис. 15.23.

Функция `refinemesh` служит для уменьшения шага сетки. Первым входным аргументом является матрица декомпозиционной геометрии, далее задаются массивы с информацией о триангуляции, в выходных аргументах возвращают-

ся массивы, соответствующие измельченной сетке. По умолчанию сторона каждого элемента делится пополам, каждый треугольный элемент исходной сетки разбивается на четыре части. Команды, приведенные ниже, производят вышеописанное уменьшение шага сетки (сравните рис. 15.23 и 15.24).

```
[p1, e1, t1] = refinemesh(dl, p, e, t);
pdemesh(p1, e1, t1)
```

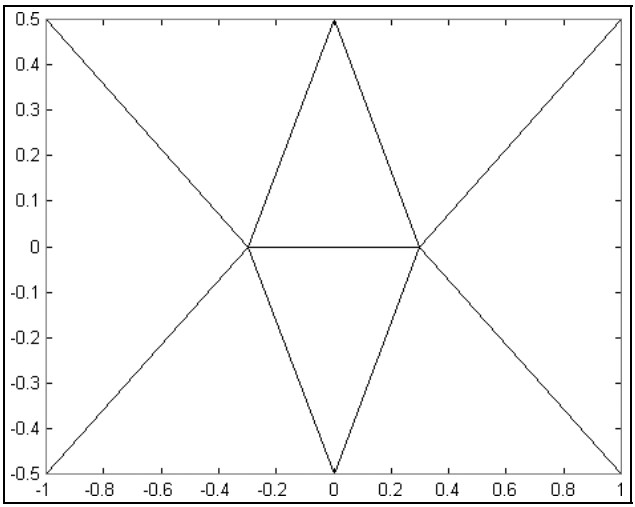


Рис. 15.23. Сетка с максимально допустимой длиной стороны элемента, равной единице

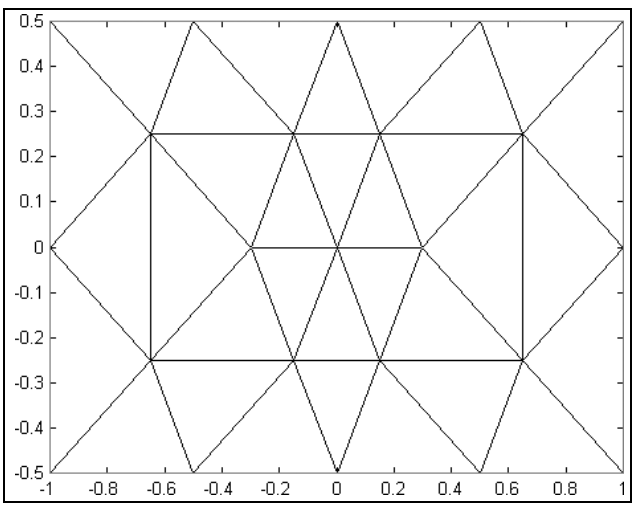


Рис. 15.24. Сетка с уменьшенным шагом (`refinemesh`)

Функция `refinemesh` позволяет задать ряд дополнительных параметров, управляющих процедурой дробления сетки, в частности делить только некоторые треугольные элементы или минимальные подобласти. Кроме того, можно выбрать другой алгоритм уменьшения шага сетки, который делит на две равные части наибольшую сторону конечного элемента.

Первым входным аргументом может быть не только матрица декомпозиционной геометрии, но и имя файл-функции, описывающей декомпозиционную геометрию области.

Граничные условия и коэффициенты уравнения

Граничные условия в ToolBox PDE задаются двумя способами, либо при помощи матрицы граничных условий (Boundary Condition matrix), либо в файл-функции. Число столбцов матрицы граничных условий совпадает с числом столбцов в матрице декомпозиционной геометрии, т. е. каждый столбец отвечает части границы области. На каждой части границы может быть задано условие одного типа: Дирихле $hu = r$, или обобщенное условие Неймана $n \cdot (c\nabla u) + qu = g$.

Разберем структуру матрицы граничных условий. Убедитесь, что среда `pdetool` настроена на решение эллиптического уравнения общего вида. Поставьте на всех границах прямоугольника граничное условие Дирихле, правая часть которого зависит от координат, например $x^2 + y^2$. Не забудьте использовать поэлементные операции, т. е. в диалоговом окне **Boundary Condition** следует ввести единицу в строку **h** и `x.^2+y.^2` в строку **r**. Экспортируйте в рабочую среду матрицу граничных условий, выбрав в меню **Boundary** пункт **Export Decomposed Geometry, Boundary Cond's**. Появляется диалоговое окно **Export**, в котором по умолчанию стоит массив `b` для хранения матрицы граничных условий и, одновременно, предлагается сохранить декомпозиционную геометрию в `g`. Получение декомпозиционной геометрии из матрицы геометрии области при помощи функции `decsg` было описано выше. Сейчас нам понадобится матрица b граничных условий. Нажмите **OK**, в рабочей среде появился двумерный массив `b`

```
b =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     9     9     9     9
    48    48    48    48
    48    48    48    48
    49    49    49    49
```

```
   120   120   120   120
    46    46    46    46
    94    94    94    94
    50    50    50    50
    43    43    43    43
   121   121   121   121
    46    46    46    46
    94    94    94    94
    50    50    50    50
```

Структура матрицы граничных условий достаточно сложная, она приспособлена для хранения граничных условий в задачах, описываемых системой дифференциальных уравнений. Каждый столбец соответствует части границы. В рассматриваемом случае, когда область является прямоугольником с граничными условиями Дирихле с $r = x^2 + y^2$ и $h = 1$ на каждой стороне, матрица состоит из четырех одинаковых столбцов. Столбец условно делится на две части, в нижней части записаны формулы коэффициентов граничных условий, причем каждый символ строки с формулой для коэффициента хранится в последовательно идущих элементах. Верхняя часть столбца матрицы граничных условий содержит информацию о типе граничных условий и длинах формул. Обратите внимание, что в нижней части столбца содержатся *коды символов*, составляющих формулы. Для того чтобы отобразить в командное окно сами символы, следует применить функцию `char`. В случае уравнения (а не системы) формулы хранятся в элементах столбца, начиная с седьмого, в чем несложно убедиться, преобразовав элементы нижней части матрицы из кодов в символы:

```
>> char(b(7:end,:))
ans =
0000
0000
1111
xxxx
....
^^^^
2222
++++
yyyy
....
^^^^
2222
```

Первые два элемента зарезервированы под коэффициенты условия Неймана (равны нулю, поскольку поставлено условие Дирихле). Третий элемент содержит значение `h`, а остальные (см. сверху вниз) — символы выражения `x.^2+y.^2`. Формат матрицы граничных условий описан в документации к ToolBox PDE, содержащейся в файле pde.pdf. Впрочем, достаточно легко разобраться в данном формате, задавая граничные условия Дирихле и Неймана и отображая символы матрицы граничных условий в командном окне.

Коэффициенты уравнения и правая часть задаются несколькими способами, в частности, для эллиптического уравнения коэффициенты `a`, `b` и `c` могут быть:

- константами;
- строками формул, содержащими переменные `x` и `y` и знаки поэлементных операций (для коэффициентов нелинейных уравнений, используется переменная `u`);
- массивами со значениями для каждого конечного элемента.

При решении систем уравнений коэффициенты системы и граничных условий являются матрицами.

Солверы

Приближенным решением граничной задачи является вектор значений в узлах сетки. Следует иметь в виду, что предварительно должна быть задана матрица или файл-функция граничных условий в массиве `b`, триангуляция в массивах `p`, `e`, `t` и коэффициенты уравнения (разумеется, переменные могут называться по-другому). Если решается нестационарная задача, то необходимо указать решение в начальный момент времени и вектор со значениями времени, в которые следует найти решение. Некоторые солверы требуют задания декомпозиционной геометрии.

Солверы реализованы в файл-функциях с достаточно сложным интерфейсом. ToolBox PDE обладает несколькими солверами для нахождения решения различных типов задач:

- `assempde` — для эллиптических уравнений и систем;
- `parabolic` — для параболических уравнений и систем;
- `hiperbolic` — для гиперболических уравнений и систем;
- `pdenonlin` — для нелинейных стационарных уравнений;
- `adaptmesh` — для адаптивной генерации сетки и нахождения решения эллиптических уравнений и систем с заданной точностью;
- `pdeeig` — для решения эллиптических задач на собственные значения.

Интерфейс солвера `assempde` является наиболее универсальным, он позволяет не только решить систему линейных уравнений методом конечных элементов, исключая граничные условия Дирихле, но и получить матрицу

жесткости, матрицу масс и вектор нагрузки, или все составные части матрицы и вектора системы метода конечных элементов. Первым входным аргументом `assempde` задается массив граничных условий, например `b`, или имя файл-функции с граничными условиями, например `boundcond`, далее указываются матрицы с информацией о триангуляции и коэффициенты уравнения. Вызовы файл-функции с одним выходным аргументом

```
u = assempde(b, p, e, t, c, a, f);
u = assempde('bouncond', p, e, t, c, a, f);
```

приводят к записи в массив `u` значений решения в узлах сетки, координаты которых хранятся в массиве `p`. Сборка матрицы и вектора системы *без ее решения* происходит при обращении к `assempde` с двумя выходными аргументами:

```
[K, F] = assempde(b, p, e, t, c, a, f);
[K, F] = assempde('bouncond', p, e, t, c, a, f);
```

Остальные способы вызова `assempde` подробно описаны в документации по ToolBox PDE. Использование всех возможностей `assempde` требует понимания того, как учитываются граничные условия при вычислении матриц и векторов правых частей элементов. Файл pde.pdf содержит раздел "The Finite Element Method" с необходимой информацией.

Солверы `parabolic` и `hyperbolic` сразу выдают матрицу с решением.. Входные аргументы данных солверов такие же, как у `assempde`, кроме того, задаются вектор значений решения *u0* в начальный момент времени (в случае гиперболического уравнения еще и вектор *ut0* со значением производной решения), вектор *tlist* моментов времени, в которые требуется найти решение, и коэффициент `d` уравнения при производной по времени. Каждый столбец матрицы *u* есть вектор со значениями приближенного решения в узлах сетки в соответствующий момент времени, указанный в `tlist`:

```
u = parabolic(u0, tlist, b, p, e, t, c, a, f, d)
u = hiperbolic(u0, ut0, tlist, b, p, e, t, c, a, f, d)
```

При решении задач на собственные значения солвером `pdeeig` требуется указать вектор *r* длины два с границами интервала, на котором разыскиваются собственные значения. Левая граница интервала может быть минус бесконечностью (`-Inf`)

```
[v, l] = pdeeig(b, p, e, t, c, a, d, r)
```

Выходными аргументами `pdeeig` являются матрица *v*, каждый столбец которой содержит значения собственной функции в узлах сетки, определяемых массивом `p`, и *l* — вектор собственных значений.

Солвер `pdenonlin`, предназначенный для решения нелинейных стационарных задач, имеет те же входные параметры, что и `assempde`

```
[u, res] = pdenonlin(b, p, e, t, c, a, f)
```

Выходными аргументами являются вектор решения и норма вектора невязки при решении нелинейной задачи методом Ньютона. Дополнительные аргументы солвера pdenonlin, управляющие вычислительным процессом, задаются парами: название, значение. Таблица возможных значений приведена в документации к ToolBox.

Замечание

Панель диалогового окна **Solve Parameters** среды pdetool, соответствующая нелинейному солверу позволяет получить представление о возможных дополнительных опциях солвера pdenonlin.

Эффективное нахождение решения, имеющего большие градиенты, требует адаптивного изменения сетки. Солвер adaptmesh реализует адаптивный алгоритм решения эллиптического уравнения. Простой вызов солвера с указанием декомпозиционной геометрии g, матрицы граничных условий *b*, коэффициентов уравнения c, a и правой части f

```
[u, p, e, t] = adaptmesh(g,b,c,a,f)
```

приводит к поиску решения с установками солвера, принятыми по умолчанию. Дополнительные аргументы, задаваемые парами: свойство, значение, позволяют определить максимально допустимое число конечных элементов, начальную триангуляцию, способ дробления элементов и использование нелинейного солвера. Солвер adaptmesh возвращает в выходных аргументах вектор решения *u* и массивы p, e и t, содержащие информацию о расчетной сетке.

Визуализация результата

Функции pdegplot и pdemesh предназначены для графического отображения геометрии области и сетки, их использование описано в предыдущих разделах. Основной функцией для визуализации решения является pdeplot, которая позволяет управлять видом получаемых графиков при помощи ряда дополнительных параметров. Функции pdecont и pdesurf, строящие линии уровня решения или трехмерный график, реализуют частные случаи обращения к pdeplot. Входными аргументами pdecont и pdesurf являются матрица *p* с координатами узлов, матрица *t* соответствия глобальной и локальной нумераций и вектор *u* со значениями решения в узлах. Четвертым дополнительным аргументом pdecont может являться вектор значений, которые требуется отобразить линиями уровня, или их число.

Интерфейс функции pdeplot достаточно универсален. Первые три входных аргумента являются массивами p, e и t с информацией о триангуляции. Далее задаются пары: свойство, значение (табл. 15.1).

Таблица 15.1. *Дополнительные параметры* `pdeplot`

Свойство	Значение
`xydata`	Вектор со значениями решения в узлах для построения двумерного графика
`xystyle`	`off`, `flat`, `interp` (по умолчанию) — способы заливки контуров
`contour`	`on`, `off` (по умолчанию) — отображение или скрытие контурных линий
`zdata`	Вектор со значениями решения в узлах для построения трехмерного графика
`colormap`	`cool`, `hot`, `gray`, `bone` ... — цветовые палитры
`mesh`	`on`, `off` (по умолчанию) — отображение конечноэлементной сетки
`colorbar`	`off`, `on` (по умолчанию) — вывод шкалы соответствия цвета и значения
`title`	Строка с заголовком
`levels`	Число линий уровня или вектор со значениями решения, отображаемого линиями уровня

Пример использования `pdeplot` приведен в следующем разделе, посвященном решению некоторой модельной задачи с использованием функций ToolBox PDE.

Решение модельной задачи

Рассмотрим решение модельной задачи для эллиптического уравнения

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=-5\pi^2 \sin \pi x \cdot \sin 2\pi y$$

в прямоугольнике с центром в начале координат, высотой единица и шириной, равной двум. На сторонах прямоугольника поставлены однородные условия Дирихле. Известно точное решение $u(x,y)=\sin \pi x \cdot \sin 2\pi y$. Требуется последовательно уменьшать шаг сетки и решать задачу, пока приближенное решение не будет отличаться от точного менее, чем на 0.01.

Задайте геометрию области и граничные условия в среде `pdetool` и экспортируйте их в глобальные массивы `g` и `b` из меню **Boundary**. Сгенерируйте два M-файла с декомпозиционной геометрией myg.m и граничными условиями myb.m при помощи функций `wbound` и `wgeom`. Алгоритм оформите в виде файл-функции `modelexam`, входными аргументами которой являются: имена файлов с геометрией области и граничными условиями и точность решения. В M-файл включите подфункцию для вычисления точного решения. Учтите, что данная подфункция предназначена для нахождения решения в узлах

сетки и вызывается от массивов с координатами узлов, поэтому следует использовать поэлементное умножение. Листинг 15.3 содержит текст основной файл-функции `modelexam` и подфункции `exsol`, снабженный подробными комментариями.

Листинг 15.3. Решение модельной задачи

```
function modelexam(gfile, bfile, err)
% Файл-функция находит решение модельной граничной
% задачи Дирихле для эллиптического дифференциального
% уравнения в прямоугольной области.
% Использование:
%    modelexam('файл с декомп. геом.', 'файл с гран. услов.', err)

% Инициализация сетки с максимальной стороной элемента 0.2
[p, e, t] = initmesh(gfile, 'Hmax', 0.2);
% Задание коэффициентов уравнения
a = 0;
c = 1;
% Определение строки с формулой правой части уравнения
f = '5*pi^2*sin(pi*x).*sin(2*pi*y)';

% Организация циклического измельчения сетки, пока
% не достигнута требуемая точность
erhelp = 1;
while erhelp > err
   % Измельчение сетки
   [p, e, t] = refinemesh(gfile, p, e, t);
   % Решение уравнения
   u = assempde(bfile, p, e, t, c, a, f);
   % Вычисление точного решения в узлах сетки,
   % абсциссы и ординаты узлов храняться в строках матрицы p
   uex = exsol(p(1,:), p(2,:));
   % Вычисление нормы погрешности приближенного решения (u является
   % вектор-столбцом, а подфункция exsol вызывается от строк, поэтому
   % вектор с точным решением необходимо транспонировать)
   erhelp = norm(u-uex',Inf);
end
```

```
% Решение найдено с требуемой точностью
% Визуализация расчетной триангуляции и вывод контурного графика
% решения, залитого цветом
figure
subplot(2,1,1)
pdemesh(p,e,t)
subplot(2,1,2)
pdeplot(p, e, t, 'xydata', u, 'colormap', 'gray', 'colorbar', 'off')

function z = exsol(x,y)
% Подфункция для вычисления точного решения
z = sin(pi*x).*sin(2*pi*y);
```

Вызов файл-функции `modelexam('myg', 'myb', 1.0e-02)` приводит к появлению графического окна, в которое выводится расчетная сетка и залитый цветом контурный график решения модельной задачи с требуемой точностью (рис. 15.25).

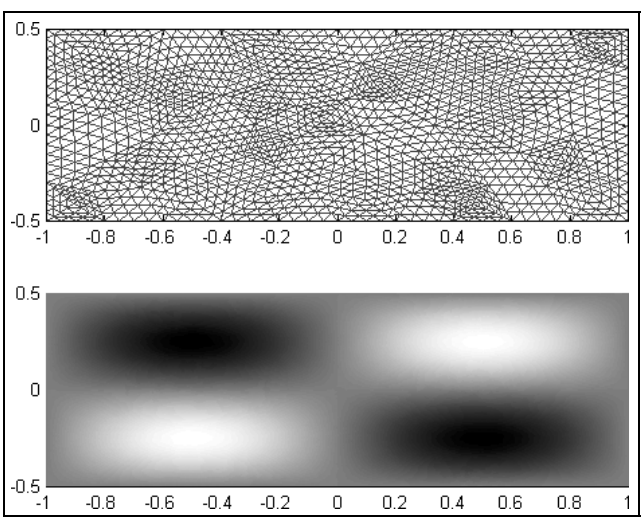


Рис. 15.25. Расчетная сетка и решение модельной задачи

ToolBox PDE содержит несколько демонстрационных файлов: pdedemo1.m—pdedemo8.m с решением различных задач. Данные примеры позволяют самостоятельно разобраться в возможностях ToolBox. M-файл pdedemos.m является файл-программой с интерфейсом командной строки, облегчающей доступ к каждому из примеров.

Функции ToolBox PDE

ToolBox предоставляет в распоряжение пользователя около пятидесяти функций, предназначенных для реализации этапов решения задачи от задания геометрии области до визуализации результата. Данный раздел посвящен описанию функций, служащих для конструирования области и разбиения ее на конечные элементы. Информацию о функции всегда можно получить при помощи команды `help`. Детальное описание всех функций и реализованных в них алгоритмов имеется в справочной системе по ToolBox PDE.

Создание геометрических примитивов

Геометрический примитив может быть создан либо из меню среды `pdetool`, либо при помощи соответствующей функции. Вызов функции приводит к добавлению соответствующего примитива в окно среды (если `pdetool` не запущена, то появляется окно `pdetool` и примитив отображается в нем). Последним дополнительным аргументом каждой функции может являться название примитива, заключенное в апострофы. Если название не задано, то используются стандартные имена `C1`, `E1`, `R1`,...

- `pdecirc(xc,yc,r)`, `pdecirc(xc,yc,r,label)` — круг с центром в `xc`, `yc` и радиусом `r`.
- `pdeellip(xc,yc,a,b,phi)`, `pdeellip(xc,yc,a,b,phi,label)`) — эллипс с центром в `xc`, `yc` и полуосями `a` и `b`, повернутый вокруг центра на угол `phi` (задаваемый в радианах).
- `pdepoly(vx, vy)`, `pdepoly(vx, vy, label)` — многоугольник с вершинами, абсциссы и ординаты которых указываются в `vx` и `vy`.
- `pderect([xmin xmax ymin ymax])`, `pderect([xmin xmax ymin ymax],label)` — прямоугольник, определяемый координатами вершин.

Геометрия области и триангуляция

Некоторые функции преобразования геометрии области и разбиения области на конечные элементы были описаны ранее. Ниже приведен список всех имеющихся функций, реализующих геометрические алгоритмы.

- `gstat=csgchk(gd)` — проверка правильности задания матрицы *gd* геометрии области. Выходной аргумент является вектор-строкой, каждый элемент которой содержит информацию о соответствующем примитиве.

  Структура матрицы геометрии области описана в разд. "Задание геометрии области" данной главы.

- `dl=descsg(gd, sf, ns)` — генерация матрицы *dl* декомпозиционной геометрии области из матрицы геометрии *gd*, строки с формулой связи геометрических примитивов *sf* и матрицы соответствия столбцов *gd* с именами примитивов в формуле. Вызов `desceg` с двумя выходными аргументами

`[dl,bt]=descsg(gd,sf,ns)` приводит к записи в *bt* информации о соответствии минимальных подобластей геометрическим примитивам.

- `[newdl,newbtl]=csgdel(dl,bt)` — удаление из матрицы *dl* декомпозиционной геометрии области всех границ между минимальными подобластями. Выходные аргументы содержат обновленную матрицу *newdl* декомпозиционной геометрии и таблицу связей примитивов и минимальных подобластей.
- `[p,e,t]=initmesh(gl)`, `[p,e,t]=initmesh('mygeom')` — инициализация сетки области, декомпозиционная геометрия которой задается матрицей или M-файлом. Возможно управление процессом инициализации сетки при помощи пар дополнительных параметров: `свойство, значение`. Параметр `Hmax` устанавливает максимальную длину стороны треугольных элементов. Начальная триангуляция области создается с учетом геометрии области, вблизи более мелких объектов сетка гуще, чем в крупных. Значение параметра `Hgrad` определяет скорость роста размеров элементов при отдалении от мелких объектов. Значения `Hgrad`, близкие к единице (100%), приводят к медленному росту размеров, напротив, значения, близкие к двум (200%), соответствуют достаточно быстрому увеличению размеров (рис. 15.26).
- Ряд параметров `initmesh` предназначен для визуализации этапов алгоритма построения триангуляции и улучшения ее качества, подробная информация об их использовании приведена в документации к ToolBox. Кроме того, функция `jigglemesh` реализует алгоритм улучшения триангуляции.

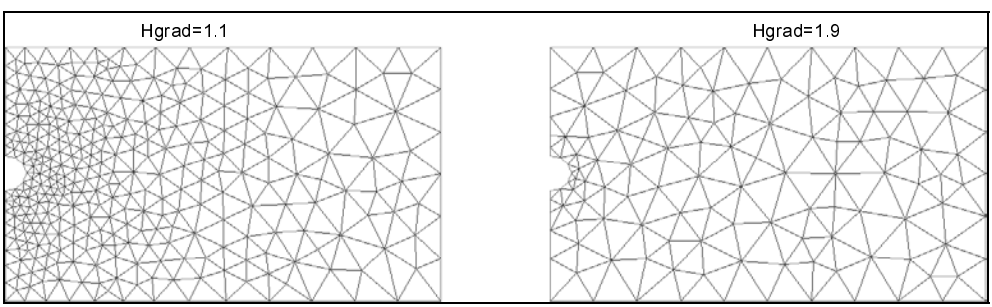


Рис. 15.26. Влияние параметра Hgrad на размеры элементов

- Функции `poimesh(gl,nx,ny)`, `poimesh('mygeom',nx,ny)` служат для инициализации *регулярной сетки* на *прямоугольной* области, декомпозиционная геометрия которой задается матрицей или M-файлом. Число узлов по каждой из координат задается в массивах `nx` и `ny`. Последовательность команд для прямоугольника, декомпозиционная геометрия которого описана файл-функцией `myg`:

```
[p, e, t] = poimesh('myg', 15, 10);
pdemesh(p,e,t)
```

приводит к регулярной сетке, изображенной на рис. 15.27. Функция `poimesh` предназначена только для прямоугольных областей.

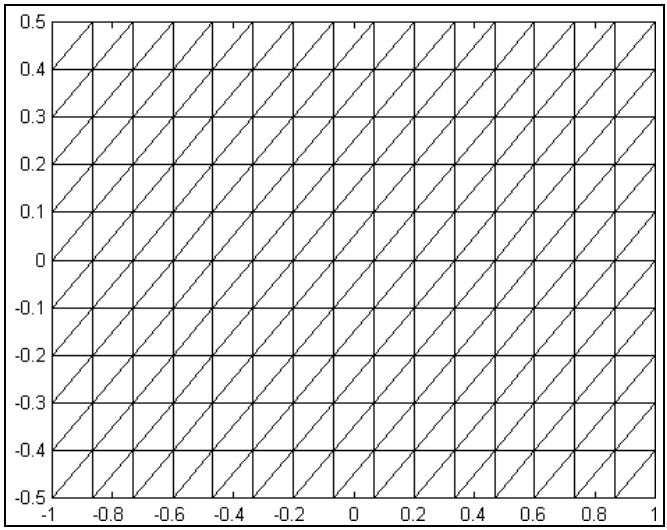


Рис. 15.27. Регулярная сетка в прямоугольнике

- `[p1,e1,t1]=refinemesh(p,e,t)` — уменьшение шага сетки, задаваемой массивами `p`, `e`, `t` делением каждого треугольного элемента на четыре части. Массивы, соответствующие новой сетке, возвращаются в `p`, `e`, `t`. Возможно указание четвертого дополнительного аргумента `'longest'` для уменьшения шага сетки делением наибольшей части треугольника на две части, по умолчанию используется `'regular'`. Исходная и преобразованные каждым из способов сетки приведены на рис. 15.28.

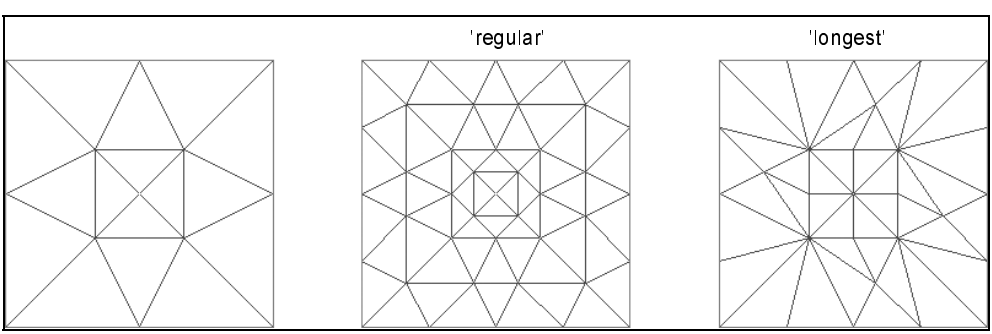


Рис. 15.28. Способы уменьшения шага сетки

Глава 16



Разреженные матрицы

Матрицы, содержащие достаточно большое число нулевых элементов, называются *разреженными*. Разреженные матрицы очень часто встречаются в самых различных областях, например, при решении дифференциальных уравнений методом конечных элементов или конечных разностей, обработке изображений, в криптографии и оптимизационных задачах. Использование специальных способов хранения разреженных матриц и алгоритмов существенно снижает затраты компьютерных ресурсов и уменьшает время вычислений. Строго говоря, матрица является разреженной, если специальные алгоритмы для ее хранения и обработки дают выигрыш по сравнению с обычными, которые воспринимают матрицу как полностью заполненную.

Работа с разреженными матрицами

Разреженные матрицы содержат небольшое число ненулевых элементов по сравнению с их размерами. Хранить все элементы, включая нулевые, нет смысла. Достаточно записать в память только ненулевые значения и информацию об их положении в матрице. В качестве примера рассмотрим одну из возможных схем хранения, реализованную в MatLab.

Схема хранения

Разреженные матрицы в MatLab помещаются в специальные переменные типа `sparse array`. Ненулевые элементы матриц хранятся по столбцам сверху вниз, для каждого элемента запоминается номер строки и столбца, соответствующие его положению в исходной полной матрице. Задайте прямоугольную матрицу размера пять на шесть, приведенную ниже, например, из командной строки:

```
>> A = [5  0 -3  0  0  0
        0 -2  0  0  2  0
        0  0  1  0  0  0
        0  0  0  0  0  0
        9  7  0  0  0  0];
```

Функция `sparse` позволяет получить из полной матрицы *A* ее компактное представление *AN*:

```
>> AN = sparse(A)
AN =
   (1,1)         5
   (5,1)         9
   (2,2)        -2
   (5,2)         7
   (1,3)        -3
   (3,3)         1
   (2,5)         2
```

Первый столбик *AN* содержит пары `(i,j)` с номерами строк и столбцов, а второй — значения ненулевых элементов исходной матрицы *A(i, j)*. Функция `sparse` последовательно просматривает элементы каждого столбца, начиная с первого, и записывает в *AN* найденные ненулевые элементы вместе с соответствующими парами индексов. Схему хранения на примере одного элемента матрицы поясняет рис. 16.1.

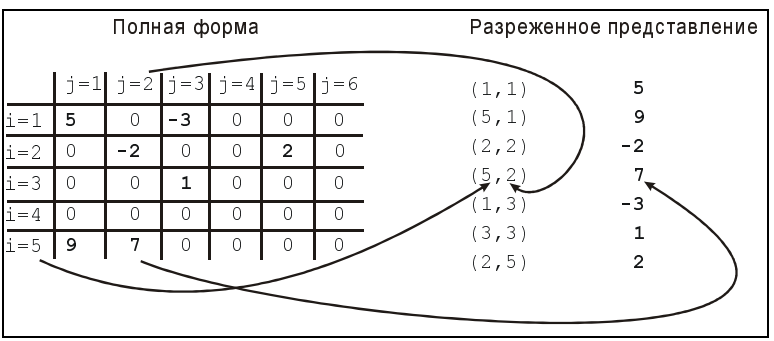


Рис. 16.1. Схема хранения

Обратите внимание, что вышеописанная схема существенно экономит память, выделяемую под хранение матрицы. Действительно, под массив `A` отводится $5 \cdot 6 = 30$ ячеек с вещественными числами, каждая размера 8 байтов, итого для записи *A* требуется $30 \cdot 8 = 240$ байт. Массив `AN` содержит $7 \cdot 2 = 14$ ячеек с целыми числами (индексами) и 7 — с вещественными. Поскольку целое число занимает 4 байта, то для хранения `AN` выделяется $14 \cdot 4 + 7 \cdot 8 = 112$ байт, т. е. более чем в два раза меньше, чем для *A*. Можно было и не производить вычисление затрат памяти, а воспользоваться командой `whos`:

```
>> whos A AN
  Name      Size            Bytes    Class
```

```
  A         5x6          240      double array
  AN        5x6          112      sparse array
```

Для реальных вычислительных задач часто оказывается, что число ненулевых элементов имеет порядок квадратного корня из общего числа всех элементов матрицы, поэтому разреженная форма хранения дает существенные преимущества по сравнению с обычным представлением матрицы.

Функция `full` позволяет получить полное представление разреженной матрицы. Очевидно, что результат совпадает с исходной матрицей *A*:

```
>> Af = full(AN)
Af =
     5     0    -3     0     0     0
     0    -2     0     0     2     0
     0     0     1     0     0     0
     0     0     0     0     0     0
     9     7     0     0     0     0
```

Разумеется, для инициализации компактного представления не требуется сначала создавать полную матрицу, указывая нулевые и ненулевые элементы, а затем переходить к разреженной структуре при помощи `sparse`. Информация о расположении ненулевых элементов и их значения достаточны для создания массива типа `sparse array`.

Создание разреженных матриц

Функция `sparse` предоставляет возможность непосредственного создания массива типа `sparse array`, соответствующего разреженной матрице. Обращение к `sparse` выглядит следующим образом:

```
AN = sparse(irow, jcol, nzer, m, n)
```

Здесь `irow` — вектор номеров строк, а `jcol` — вектор номеров столбцов ненулевых элементов матрицы, которые помещены в вектор `nzer`. В последних аргументах `m` и `n` задаются размеры исходной полной матрицы. Сконструируйте компактное представление прямоугольной матрицы *A* размера пять на шесть, определенной в предыдущем разделе. Обратите внимание, что порядок элементов в массивах `irow`, `jcol` и `nzer` должен быть одинаковым. Очевидно, что следует выполнить последовательность команд, приведенную ниже:

```
>> irow = [1 5 2 5 1 3 2];
>> jcol = [1 1 2 2 3 3 5];
>> nzer = [5 9 -2 7 -3 1 2];
>> AN=sparse(irow, jcol, nzer, 5, 6)
AN =
   (1,1)        5
```

```
   (5,1)         9
   (2,2)        -2
   (5,2)         7
   (1,3)        -3
   (3,3)         1
   (2,5)         2
```

Произведите проверку, выведите полное представление для *AN*, используя функцию `full(AN)`. Обход по столбцам в полном представлении при поиске ненулевых элементов является необязательным. Можно просматривать каждую строку и формировать требуемые векторы `irow`, `jcol` и `nzer`, результат будет тот же самый.

```
>> irow = [1 1 2 2 3 5 5];
>> jcol = [1 3 2 5 3 1 2];
>> AN=sparse(irow, jcol, nzer, 5, 6);
```

Более того, можно просматривать элементы разреженной матрицы в произвольном порядке, важно только правильно указать номер строки и столбца для ненулевого элемента в массивах `irow`, `jcol`. Следующий способ инициализации компактного представления *AN* матрицы *A* эквивалентен приведенным выше вариантам.

```
>> irow = [1 2 3 2 1 5 5];
>> jcol = [3 2 3 5 1 2 1];
>> nzer = [-3 -2 1 2 5 7 9];
>> AN=sparse(irow, jcol, nzer, 5, 6);
```

Задание аргументов функции `sparse` имеет ряд особенностей. Если какие-либо ненулевые элементы имеют одинаковые позиции, т. е. совпадают соответствующие им пары с номерами строк и столбцов, то происходит сложение данных элементов и запись их суммы в указанную парой индексов позицию. Например, при конструировании компактной формы хранения в пятую строку первого столбца кроме девяти добавьте элемент, равный десяти, занеся в массивы `irow`, `jcol` и `nzer` соответствующие числа, в результате элемент матрицы с индексами пять и один будет равен девятнадцати:

```
>> irow = [1 2 3 2 1 5 5 5];
>> jcol = [3 2 3 5 1 2 1 1];
>> nzer = [-3 -2 1 2 5 7 9 10];
>> AN=sparse(irow, jcol, nzer, 5, 6);
>> full(AN)
ans =
     5     0    -3     0     0     0
     0    -2     0     0     2     0
```

```
     0     0     1     0     0     0
     0     0     0     0     0     0
    19     7     0     0     0     0
```

Нулевые элементы, записанные в `nzer`, пропускаются при создании функцией `sparse` компактной формы хранения разреженной матрицы:

```
>> irow = [1 5 2 4 3 4 5 1 3 2];
>> jcol = [1 1 2 5 2 6 2 3 3 5];
>> nzer = [5 9 -2 0 0 0 7 -3 1 2];
```

Функция `sparse` допускает обращение к ней с различным числом аргументов. Размеры матрицы *m* и *n* можно не указывать. Максимальные значения векторов `irow` и `jcol` будут использоваться в качестве *m* и *n*. Однако при таком вызове `sparse` следует соблюдать осторожность. Попытка создать, например квадратную матрицу размера два, в которой ненулевые элементы расположены только в первой строке, при помощи следующих команд приведет к неверному результату:

```
>> irow = [1 1];
>> jcol = [1 2];
>> nzer = [2 3];
>> SN=sparse(irow, jcol, nzer)
>> full(SN)
ans =
     2     3
```

Вместо квадратной матрицы получилась прямоугольная один на два, поскольку максимальный элемент вектора `irow` равен единице. Можно воспользоваться тем обстоятельством, что нулевые элементы в `nzer` игнорируются, и добавить один фиктивный элемент, равный нулю, который расположен в правом нижнем углу матрицы:

```
>> irow = [1 1 2];
>> jcol = [1 2 2];
>> nzer = [2 3 0];
>> SN=sparse(irow, jcol, nzer);
>> full(SN)
ans =
     2     3
     0     0
```

Дополнительный шестой аргумент `sparse` указывает на количество элементов, выбираемых из `irow`, `jcol` и `nzer`, для конструирования компактной формы.

Матрица, записанная в компактной форме в текстовом файле, может быть считана командой `load` и затем преобразована функцией `spconvert` в массив типа `sparse array`. Файл должен состоять из трех столбцов, в первых двух записаны индексы строк и столбцов ненулевых элементов, а в третьем — сами ненулевые элементы. Содержимое текстового файла для матрицы *A* из данного раздела приведено в листинге 16.1.

Листинг 16.1. Тестовый файл с компактной формой хранения разреженной матрицы

```
1  1  5
5  1  9
2  2 -2
5  2  7
1  3 -3
3  3  1
2  5  2
```

Теперь следует считать данные из файла в некоторый массив и преобразовать их в компактную форму хранения:

```
>> s = load('spmatr.dat');
>> A = spconvert(s);
```

Замечание

Элементы матрицы могут быть комплексными. Использование `sparse` аналогично случаю вещественных матриц, вектор `nzer` в функции `sparse` должен содержать комплексные элементы. Импортирование матрицы из файла имеет одну особенность, файл должен иметь четыре столбца, причем первые два содержат индексы строк и столбцов матрицы, а третий и четвертый — вещественную и мнимую части элементов матрицы.

Матрицы, ненулевые элементы которых расположены на главных или побочных диагоналях, часто хранят по диагоналям, записывая их в столбцы вспомогательной матрицы меньшего размера. Например, матрицу

```
A = [5  0 -3  0  0
     1  3  0 -1  0
     0  1  7  0 -2
     0  0  1 22  0
     0  0  0  1  8];
```

можно упаковать следующим образом. Занесите в столбцы матрицы *B* диагонали *A*, начиная с нижней:

```
B = [1     5     0
     1     3     0
```

```
     1     7    -3
     1    22    -1
     0     8    -2];
```

Теперь создайте вектор *d* с информацией о соответствии столбца матрицы *B* номеру диагонали в *A*, учтите, что главная диагональ имеет номер ноль, нижние диагонали нумеруются отрицательными числами, а верхние положительными. В рассматриваемом примере вектор *d* состоит из трех элементов:

```
d =[-1 0 2];
```

Функция `spdiags` позволяет перейти от формы хранения по диагоналям к компактной форме, реализованной в `sparse`. Аргументами `spdiags` являются: матрица *B* с диагоналями, вектор *d* с номерами диагоналей и размеры формируемой матрицы:

```
A = spdiags(B, d, 5, 5);
```

Убедитесь в правильности результата, выведите полную и разреженную формы *A*. Указание в качестве входного аргумента в `spdiags` исходной матрицы *A* приводит к решению обратной задачи, состоящей в записи диагоналей *A* в матрицу *B*, а информации об их расположении в вектор *d*:

```
>> [B,d] = spdiags(A)
B =
     1     5     0
     1     3     0
     1     7    -3
     1    22    -1
     0     8    -2
d =
    -1
     0
     2
```

Отображение шаблона матрицы, т. е. расположения ненулевых элементов, существенно облегчает исследование больших разреженных матриц. Функция `spy` выводит шаблон матрицы в отдельное графическое окно.

См. разд. "Визуализация матриц" главы 2.

Для просмотра отдельных блоков следует воспользоваться инструментом увеличения масштаба, расположенным на панели инструментов графического окна.

Операции с разреженными матрицами

Полное представление матрицы, хранящейся в массиве A, имеет очевидную связь с соответствующими векторами irow, jcol и nzer: A(irow(k), jcol(k))=nzer(k). Однако, если компактная форма *AN* хранения создается при помощи функции sparse, то использовать перечисленные векторы нет смысла. Дело в том, что для нахождения произвольного элемента исходной матрицы *A* с индексами i и j, в ее компактном представлении требуется произвести поиск индексов в массивах irow и jcol, если для некоторого k выполняются равенства i=irow(k), j=jcol(k), то искомый элемент матрицы хранится в nzer(k), а если подходящего номера k нет, то элемент матрицы нулевой. MatLab производит поиск автоматически, позволяя получать доступ к элементам матриц, хранящихся в массивах типа sparse array, так же, как к обычным матрицам — при помощи двух индексов, заключаемых в круглые скобки. Например

```
>> AN(5,1)
ans =
     9
>> AN(4,3)
ans =
     0
```

Реализация матричного сложения, вычитания и умножения для компактной формы хранения также скрыта от пользователя. Данные операции производятся над разреженными матрицами так же, как над обычными, и в результате получается разреженная матрица, хранящаяся в массиве типа sparse array. Если одна из матриц хранится в обычном массиве double array, то результат будет также переменной типа double array. Поэлементное умножение матрицы на матрицу приводит к формированию разреженной матрицы, если хотя бы один из множителей записан в компактной форме. Значения математических функций от разреженных матриц также являются разреженными, например sin(AN). Математические функции применяются поэлементно, так же, как и к обычным матрицам. В том случае, когда необходимо вычислить значения *только от ненулевых элементов*, следует использовать spfun. Входными аргументами spfun являются имя файл-функции, или встроенной математической функции и массив типа sparse array с компактной формой хранения матрицы.

Замечание

Работа с массивами sparse array с точки зрения пользователя не отличается от работы с обычными массивами double array, хотя на самом деле используются другие алгоритмы. Поскольку MatLab является объектно-ориентированной системой, то типы данных выстроены в определенную иерархию, в

которой класс `sparse array` является потомком `double array`, переопределяя его методы. Пользователь имеет возможность создавать свои классы и реализовывать собственные методы.

Функция `size` возвращает число строк и столбцов матрицы, компактное представление которой указано во входном аргументе. Такие функции, как `max`, `min` и `sum`, возвращают разреженный вектор, т. е. матрицу из одной строки. Данные функции производят те же вычисления, что и над обычными матрицами, а тип результата соответствует типу входного аргумента. Аналогичным образом конструируются блочные разреженные матрицы, и производится индексация при помощи вектора значений индексов.

Структура разреженной матрицы определяет наиболее удобный способ ее формирования. Справочная система MatLab содержит пример генерации симметричной разреженной матрицы, соответствующей разностной аппроксимации второй производной. На диагонали матрицы стоят числа -2, а на побочных диагоналях единицы. Сначала создаются векторы с диагональными элементами, затем на их основе — две разреженные матрицы, диагональная и с одной побочной диагональю. Искомая матрица является суммой диагональной матрицы, матрицы с побочной диагональю и транспонированной к ней. Создайте, в качестве упражнения такую трехдиагональную матрицу.

Информацию о разреженной матрице можно получить при помощи нескольких функций MatLab. Функция `find` производит обратную операцию, по отношению к `sparse`, она заполняет векторы с номерами строк и столбцов ненулевых элементов и вектор ненулевых элементов. Вызов `find` с двумя выходными аргументами обеспечивает заполнение массивов индексов строк и столбцов: `[irow, jcol]=find(AN)`, указание третьего выходного аргумента приводит к записи в него ненулевых элементов: `[irow, jcol, nzer]=find(AN)`. Если требуется получить только ненулевые элементы, то проще воспользоваться функцией `nonzeros`, которая возвращает вектор ненулевых элементов разреженной матрицы, упорядоченных по столбцам. Количество ненулевых элементов устанавливается при помощи функции `nnz`.

При программировании собственных алгоритмов полезно убедиться, что матрица является разреженной, т. е. массивом типа `sparse array`. Данную проверку производит функция `issparse`, возвращая логическую единицу для разреженной матрицы и ноль — в противном случае.

Задачи линейной алгебры

Ряд функций MatLab предназначен для решения задач линейной алгебры таких, как: факторизация матриц, решение систем линейных уравнений прямыми и итерационными методами, нахождение собственных чисел и

векторов. Алгоритмы, реализованные в файл-функциях, учитывают разреженность матриц, благодаря чему являются очень эффективными. Далее описаны основные алгоритмы, приведен пример использования профайлера MatLab для получения временных затрат алгоритмов.

Факторизация матриц

MatLab предлагает несколько классических способов факторизации разреженных матриц: LU-разложение, полное и неполное разложение Холецкого, QR-разложение. Факторизация разреженных матриц имеет свои особенности, следует стремиться к тому, чтобы получить максимально разреженные матрицы, входящие в разложение. Разберем один простой пример, демонстрирующий технику разложения разреженных матриц. Сохраните в массиве `BN` типа `sparse array` разреженную матрицу *B*, приведенную ниже. Можно сначала создать полную матрицу, например из командной строки

```
>> B = [4  1  0  0  1  0
        1  4  1  0  0  0
        0  1  4  0  0  0
        0  0  0  4  0  1
        1  0  0  0  4  1
        0  0  0  1  1  4];
```

а затем применить `BN=sparse(B)` для получения компактной формы хранения. Итак, в массиве `BN` содержится разреженная матрица *B*. Обратите внимание, что матрица симметричная и положительно определенная. Найдите разложение Холецкого $R^TR=B$ данной матрицы, используя функцию `chol`, которая возвращает верхнюю треугольную матрицу *R*. В качестве входного аргумента `chol` укажите массив `BN` с компактной формой хранения для того, чтобы результат был массивом типа `sparse array`.

```
R =  chol(BN);
```

Отобразите шаблон матрицы *R*, используя команду `spy`, получающееся расположение ненулевых элементов приведено на рис. 16.2. Матрица *R*, входящая в разложение, имеет довольно большой разброс ненулевых элементов относительно главной диагонали. На практике стремятся свести ширину ленты множителя разложения к минимуму, подбирая подходящие перестановки строк и столбцов в исходной матрице. Перед разысканием разложения Холецкого разреженной матрицы следует применить алгоритм упорядочения, обеспечивающий уменьшение ширины ленты. Данный алгоритм реализует функция `symrcm`, ее входным аргументом является разреженная матрица, подлежащая упорядочению, а выходным — вектор с перестановками индексов строк и столбцов.

В рассматриваемом примере получается следующий вектор:

```
>> rind = symrcm(BN)
rind =
     3     2     1     5     6     4
```

Сама матрица *BN*, естественно, не изменяется в ходе алгоритма, осуществляется только соответствующая перенумерация. Предлагаемые перестановки исходной матрицы действительно уменьшают ширину ленты, в чем несложно убедиться, отобразив полную форму матрицы *BN* с переставленными строками и столбцами. Используйте индексацию при помощи вектора `rind`:

```
>> full(BN(rind,rind))
ans =
     4     1     0     0     0     0
     1     4     1     0     0     0
     0     1     4     1     0     0
     0     0     1     4     1     0
     0     0     0     1     4     1
     0     0     0     0     1     4
```

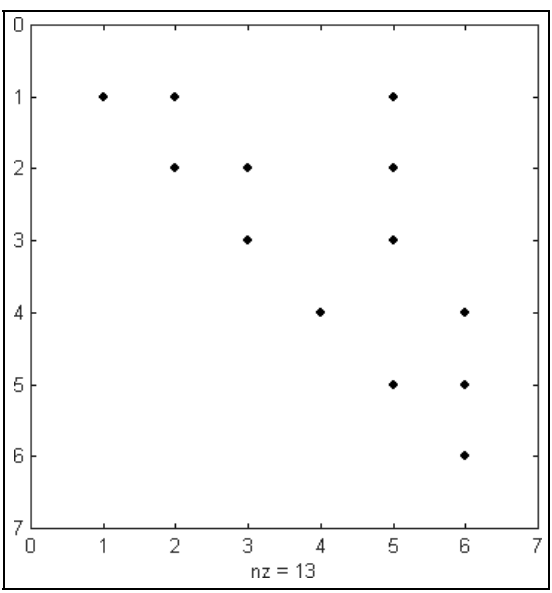


Рис. 16.2. Разложение Холецкого без переупорядочения

Произведите разложение Холецкого для матрицы с переставленными строками и столбцами и отобразите шаблон матрицы *R*.

Используйте индексацию при помощи вектора.

```
>> R = chol(BN(rind,rind));
>> spy(R)
```

Расположение и количество ненулевых элементов, обозначенное `nz` в графическом окне с шаблоном, в матрице *R* (рис. 16.3) свидетельствует об эффективности предварительных перестановок в исходной матрице с целью уменьшения ширины ленты.

Функция `lu` предназначена для нахождения LU-разложения матрицы. Обращение `[L,U]=lu(AN)` приводит к записи в выходные аргументы соответствующих матриц разложения. Процесс разложения требует перестановок строк и столбцов для обеспечения вычислительной устойчивости алгоритма. Указание третьего дополнительного выходного аргумента позволяет получить матрицу *P* проделанных перестановок: `[L,U,P]=lu(AN)`, причем выполняется равенство `P*S = L*U`. Функция `qr` находит QR-разложение матрицы. Функции `luinc` и `cholinc` вычисляют неполное LU-разложение и разложение Холецкого. Особенности применения `qr`, `luinc` и `luinc` подробно описаны в справочной системе MatLab.

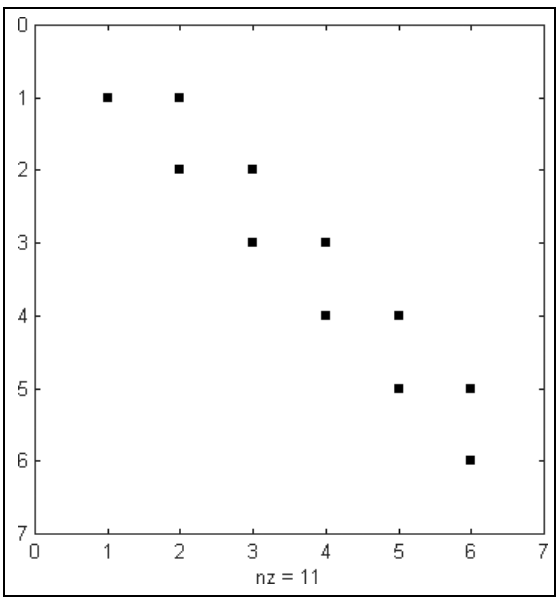


Рис. 16.3. Разложение Холецкого с переупорядочением

Функции `lu`, `chol` и `qr` применимы и к матрицам, хранящимся в полной форме в массивах типа `double array`, однако для эффективного решения

задач следует использовать компактную форму записи в массивах `sparse array`. Все алгоритмы MatLab для разреженных матриц, хранящихся в массивах типа `sparse array`, работают с учетом структуры матрицы, обеспечивая значительное ускорение вычислений по сравнению с полными матрицами. Профайлер MatLab позволяет убедиться в вышесказанном.

Профайлер

Профайлер MatLab служит для получения информации о временных затратах на выполнение команд или M-файлов и является удобным средством при создании эффективных алгоритмов. Профайлер помогает выявить блок операторов, выполнение которых занимает неоправданно большое время. Статистика временных затрат представляется в виде подробного отчета или наглядной столбцевой диаграммой.

Запуск профайлера производится командой `profile on`, в конце которой могут быть указаны дополнительные параметры, в частности опция `-detail builtin` приводит к сбору информации о времени работы всех встроенных функций MatLab. После команды `profile` следует выполнять операторы и функции, время работы которых подлежит определению. Остановка профайлера и вывод отчета осуществляются командой `profile report`. Открывается браузер с отчетом в формате HTML, содержащим подробную статистику вычислительных затрат.

Используйте профайлер для того, чтобы установить преимущество использования разреженной структуры матрицы на примере разложения Холецкого. Задана трехдиагональная симметричная матрица S достаточно большого размера $n = 1000$

$$S = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix}.$$

Создайте данную разреженную матрицу и сохраните ее в массиве `SN` типа `sparse array`. Приведите массив `SN` с разреженной матрицей к массиву `S` типа `double array` и замерьте время, необходимое для разложения Холецкого той же самой матрицы, но представленной в полной форме вместе с нулевыми элементами. Затем установите время, затрачиваемое на разложение Холецкого разреженной матрицы, и сравните затраты компьютерного времени.

Конструирование массива SN типа sparse array с трехдиагональной симметричной матрицей не представляет большого труда, если заметить, что матрица *S* является суммой диагональной матрицы, матрицы с одной побочной диагональю и транспонированной к ней. Последовательность операторов, приведенная в листинге 16.2, обеспечивает требуемое заполнение массива SN.

Листинг 16.2. Создание симметричной трехдиагональной матрицы

```
n=1000; % определение размера матрицы
% Запись в D разреженной диагональной матрицы,
% на диагонали которой стоят двойки
D = sparse(1:n,1:n,2*ones(1,n),n,n);
% Занесение в D1 разреженной матрицы из одной побочной нижней диагонали,
% на которой расположены минус единицы
D1 = sparse(2:n,1:n-1,-ones(1,n-1),n,n);
% Нахождение требуемой трехдиагональной матрицы
SN = D1+D+D1';
```

Приведите матрицу к полной форме, запустите профайлер, найдите разложение Холецкого для SN и получите отчет при помощи следующих команд:

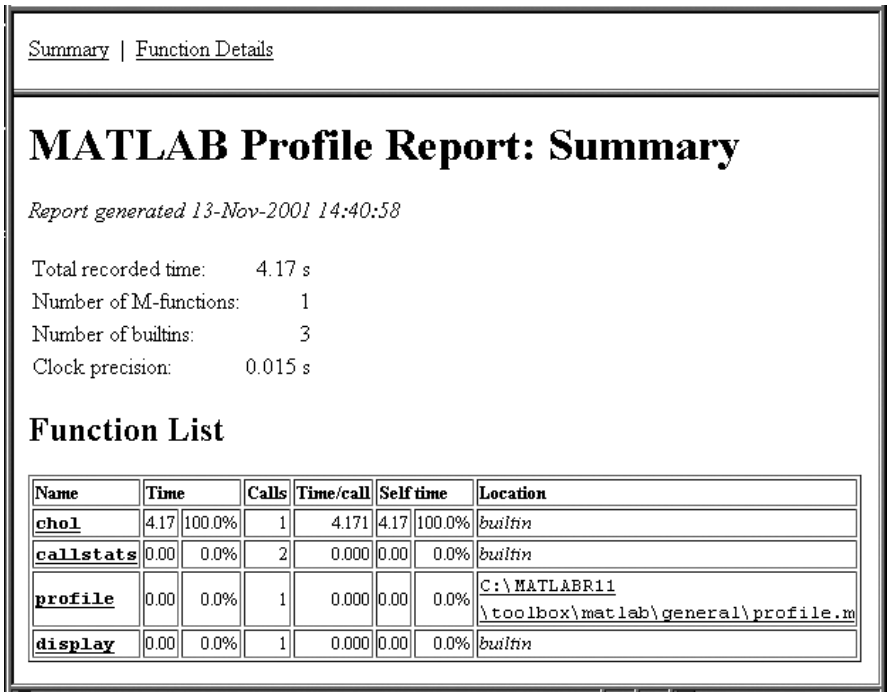
```
>>  S = full(SN);
>>  profile on -detail builtin
>>  R = chol(S);
>>  profile report
```

В открывающемся браузере появляется страница с отчетом, изображенная на рис. 16.4 (в MatLab версии 6.*x* одновременно появляется графическое окно с горизонтальной столбцевой диаграммой). Таблица **Function List** содержит информацию о времени работы каждой из функций, выполняемой после инициализации профайлера и до создания отчета. В столбце **Time** записано общее время, а в **Self time** — время работы операторов функции без учета временных затрат на вызов из нее других встроенных функций. Выше таблицы помещена, в частности, погрешность замера времени, которая составляет 0.015 с. Детальную информацию о каждой функции можно получить, перейдя по ссылке с именем функции, или по ссылке **Function Details** в верхнем фрейме.

В рассматриваемом примере интерес представляет время, потраченное на разложение Холецкого, т. е. время работы функции chol, которое равно 4.17 с. (значения для различных компьютеров могут отличаться от приведенного в книге). Сравните его со временем разложения Холецкого разре-

женной матрицы, т. е. матрицы, хранящейся в массиве `S` типа `sparse array`, выполнив команды

```
profile on -detail builtin
R = chol(SN);
profile report
```



Summary | Function Details

MATLAB Profile Report: Summary

Report generated 13-Nov-2001 14:40:58

Total recorded time:	4.17 s
Number of M-functions:	1
Number of builtins:	3
Clock precision:	0.015 s

Function List

Name	Time		Calls	Time/call	Self time		Location
chol	4.17	100.0%	1	4.171	4.17	100.0%	*builtin*
callstats	0.00	0.0%	2	0.000	0.00	0.0%	*builtin*
profile	0.00	0.0%	1	0.000	0.00	0.0%	C:\MATLABR11 \toolbox\matlab\general\profile.m
display	0.00	0.0%	1	0.000	0.00	0.0%	*builtin*

Рис. 16.4. Отчет о временных затратах

Обратите внимание на существенное преимущество, проявляющееся при использовании компактной формы хранения матриц.

Профайлер имеет ряд полезных опций, удобных для оценивания эффективности алгоритмов. Команда `profile plot` отображает информацию, представленную в столбце **Time** таблицы **Function List**, в виде столбцевой диаграммы. Отчет можно получить в заданном файле формата HTML, например myreport.html, при помощи `profile report myreport`. Файл myreport.html создается в текущем каталоге вместе с несколькими сопутствующими файлами. Команда `profile off` приостанавливает сбор информации, а `profile resume` — возобновляет. Указание опции `operator` вместо `builtin` расширяет множество действий, для которых осуществляется расчет временных затрат, включая арифметические операции.

Решение систем уравнений и исследование спектра

Методы решения систем линейных уравнений делятся на два класса: прямые и итерационные. Прямые методы основаны на предварительной факторизации матрицы системы и последующем решении двух систем с треугольными матрицами. Данный подход позволяет, в частности, эффективно решить несколько систем с одной матрицей и различными векторами правых частей.

В MatLab операция \ реализует прямой метод решения системы линейных уравнений $Ax = b$, причем в случае нескольких систем $Ax = b^{(1)},...,$ $Ax = b^{(k)}$ векторы правых частей записываются в столбцы некоторой вспомогательной матрицы B. В общем случае находится решение системы

$$AX = B,$$

где $B = \left[b^{(1)},\ldots,b^{(k)}\right]$.

Каждый столбец матрицы X содержит решение соответствующей системы линейных алгебраических уравнений.

Применение обратной косой черты для решения систем общего вида было кратко описано в *главе 6*. Операция \ реализует следующий алгоритм. Сначала MatLab проверяет, является ли матрица системы треугольной, или может быть приведена к треугольному виду при помощи перестановок строк и столбцов. Системы с треугольными матрицами быстро решаются последовательным нахождением неизвестных и алгоритм заканчивает работу. Если матрица системы симметрична и имеет положительные диагональные элементы, то MatLab пытается найти разложение Холецкого, предварительно проделав перестановки, обеспечивающие уменьшение ширины ленты исходной матрицы. В случае успешной факторизации решаются системы с треугольными матрицами, и возвращается решение системы (или систем) линейных уравнений. Если же разложение Холецкого найти не удалось, то MatLab решает систему линейных уравнений, предполагая, что матрица системы имеет общий вид. Сначала производится перестановка столбцов, обеспечивающая разреженность матричных сомножителей в LU-разложении, затем выполняется LU-разложение и находится решение систем с треугольными матрицами.

В приложениях очень часто встречаются системы линейных уравнений, для решения которых итерационные методы оказываются наиболее эффективными. MatLab обладает достаточно большим набором функций, которые реализуют основные итерационные алгоритмы: предобусловленный метод сопряженных градиентов (`pcg`), метод бисопряженных градиентов (`bicg`), метод обобщенных минимальных невязок (`gmres`) и другие методы. Обсуждение алгоритмов перечисленных итерационных методов выходит за рамки данной книги.

Справочная система MatLab содержит некоторые классические примеры, в частности решение системы линейных уравнений, соответствующей пятиточечной разностной аппроксимации оператора Лапласа, при помощи метода предобусловленных сопряженных градиентов. Предобусловливатель строится на основе неполного разложения Холецкого исходной матрицы. Пользователь может выбрать любую подходящую матрицу в качестве предобусловливателя, запрограммировать быстрое решение системы линейных уравнений с данной матрицей в файл-функции и указать имя файл-функции в соответствующем аргументе `pcg`, вместо предобусловливателя. Гибкость интерфейса `pcg` предоставляет вычислителю широкие возможности для проведения исследований с целью написания эффективных солверов для определенных классов задач.

Неполная проблема на собственные значения $Ax = \lambda x$ и $Ax = \lambda Bx$ решается при помощи функции `eigs`, которая позволяет разыскать заданное число собственных значений (в том числе и комплексных). Интерфейс функции `eigs` допускает нахождение собственных значений, начиная с наибольшего или наименьшего по модулю или вблизи некоторого числа. Например, вызов

```
lambda=eigs(A,k)
```

приводит к занесению в вектор `lambda` первых `k` наибольших по модулю собственных значений. Для нахождения первых `k` наименьших по модулю собственных значений применяется обращение:

```
lambda=eigs(A,k,0)
```

В общем случае, когда требуется определить несколько собственных значений вблизи заданного числа `s`, следует указать его третьим аргументом в `eigs`:

```
lambda=eigs(A,k,s)
```

Одновременное нахождение собственных векторов происходит при вызове `eigs` с двумя выходными аргументами, что аналогично использованию функции `eig`, которая предназначена для решения полной проблемы на собственные значения.

См. разд. "Собственные числа и векторы матрицы, функции матриц" главы 6.

При решении обобщенной проблемы на собственные значения с симметричной положительно определенной матрицей B следует указывать ее вторым входным аргументом после матрицы A. Функция `eigs` допускает задание ряда дополнительных параметров, управляющих вычислительным процессом.

Глава 17



Оптимизация

В состав MatLab входит ToolBox Optimization, предназначенный для решения линейных и нелинейных оптимизационных задач. Функции этого ToolBox реализуют основные алгоритмы оптимизации, причем понимание алгоритма позволяет пользователю настроить нужную функцию на эффективное решение поставленной задачи. ToolBox Optimization не имеет приложений с графическим интерфейсом. Последний раздел данной главы содержит пример приложения, облегчающего доступ к нужным функциям ToolBox и управление вычислительным процессом.

ToolBox Optimization

Минимизация функций одной переменной и нескольких переменных без ограничений описана в *главе 6*. Данный раздел посвящен более сложным задачам с ограничениями, которые могут быть решены с использованием функций ToolBox Optimization.

Линейное и нелинейное программирование

Линейное программирование

Задача линейного программирования состоит в нахождении вектора x, который минимизирует целевую линейную функцию $f^T x$, где f — вектор коэффициентов, и удовлетворяет заданным линейным ограничениям: неравенствам $Ax \le b$ и равенствам $A_{eq} x = b_{eq}$. Кроме того, могут быть поставлены двусторонние покомпонентные ограничения в векторной форме $lb \le x \le ub$. Функция `linprog` предназначена для решения задач линейного программирования.

Замечание

В задачах оптимизации могут быть заданы не все типы ограничений, например, имеются только ограничения в виде неравенств. Интерфейс функций ToolBox Optimization достаточно гибкий. Все функции могут быть вызваны с переменным числом входных и выходных параметров, в зависимости от задачи и требуемого вида результата.

Первым аргументом `linprog` всегда является вектор *f*, далее задается матрица *A* и вектор *b*. При наличии ограничений в виде равенств дополнительными аргументами могут быть `Aeq` и `beq`, наконец, двусторонние ограничения являются шестым и седьмым аргументами `linprog`.

Решите классическую задачу линейного программирования о составлении рациона питания. Имеются три продукта П1, П2, П3 разной цены, каждый из которых содержит определенное количество питательных ингредиентов И1, И2, И3, И4 (табл. 17.1). Известно, что в день требуется: И1 — не менее 250, И2 — не менее 60, И3 — не менее 100 и И4 не менее 220. Требуется минимизировать затраты на приобретение продуктов. Очевидно, что количество приобретаемых продуктов не может быть отрицательным.

Таблица 17.1. *Питательность и цена продуктов*

	П1	**П2**	**П3**
И1	4	6	15
И2	2	2	0
И3	5	3	4
И4	7	3	12
цена	44	35	100

Запишите целевую функцию, матрицу A, вектора b и lb ограничений в соответствии с требованиями ToolBox, обозначив искомые количества продуктов через x_1, x_2 и x_3 соответственно. Поскольку линейные ограничения содержат "меньше или равно", а количество ингредиентов в рационе не должно быть менее заданных величин, то следует изменить знаки обеих частей системы.

$$f^T x = 44 \cdot x_1 + 35 \cdot x_2 + 100 \cdot x_3;$$

$$A = \begin{bmatrix} -4 & -6 & -15 \\ -2 & -2 & 0 \\ -5 & -3 & -4 \\ -7 & -3 & -12 \end{bmatrix}; \qquad b = \begin{bmatrix} -250 \\ -60 \\ -100 \\ -220 \end{bmatrix}; \qquad bl = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Для решения задачи составьте файл-программу `ration`. При вызове `linprog` вместо неиспользуемых аргументов (нет ограничений в виде равенств и верхней границы для неизвестных) задайте пустые массивы, обозначаемые в MatLab квадратными скобками. Листинг 17.1 содержит операторы файл-программы `ration`.

Листинг 17.1. Файл-программа `ration`

```
% Задание матрицы и вектора правой части системы неравенств
A = [4  6  15
     2  2  0
     5  3  4
     7  3  12];
A = -A;
b = [250; 60; 100; 220];
b = -b;
% Определение коэффициентов целевой функции
f = [44;      35;    100];
% Задание ограничений снизу на переменные
lb =[0; 0; 0];
% Решение и вывод результата в командное окно
x = linprog(f, A, b, [], [], lb, [])
```

Выполнение файл-программы `ration` приводит к следующему результату:

```
>> Optimization terminated successfully.
x =
   13.2143
   16.7857
    6.4286
```

Обращение к `linprog` с двумя выходными аргументами позволяет не только получить вектор решения, но и значение целевой функции, т. е. минимальную стоимость рациона в рассматриваемой задаче:

```
>> [x, p] = linprog(f, A, b, [], [], lb, []);
Optimization terminated successfully.
>> p
p =
  1.8118e+003
```

Квадратичное программирование

В задачах квадратичного программирования целевая функция имеет вид

$$\frac{1}{2}x^T Hx + f^T x,$$

а ограничения ставятся такие же, как в задаче линейного программирования. Функция `quadprod` предназначена для решения задач квадратичного

программирования. Интерфейс `quadprog` практически не отличается от `linprog`, за исключением того, что первыми двумя входными параметрами являются массив `H` и вектор-столбец *f*, соответствующие матрице H и вектору f целевой функции. Вместо матриц и векторов неиспользуемых ограничений задаются пустые массивы.

Нелинейное программирование

Toolbox Optimization позволяет решать ряд оптимизационных задач, в которых к линейным ограничениям добавляются нелинейные. Общая постановка задачи нелинейного программирования такова: требуется разыскать $\min f(x)$ среди всех векторов x, удовлетворяющих системе неравенств и равенств

$$c(x) \le 0; \quad c_{eq}(x) = 0; \quad Ax \le b; \quad A_{eq}x = b_{eq}; \quad lb \le x \le ub.$$

Решение поставленной задачи производится при помощи функции `fmincon`. Основное отличие интерфейса `fmincon` от `linprog` и `quadprog` состоит в том, что нелинейные ограничения $c(x) \le 0$ и $c_{eq}(x) = 0$ задаются в файл-функции. Обращение к `fmincon` в достаточно общем случае выглядит следующим образом:

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)
```

Указание второго дополнительного выходного аргумента позволяет получить значение функции в точке минимума, а третьего — информацию о результате. Если третий аргумент больше нуля, то результат найден с требуемой точностью, ноль — достигнуто максимальное число итераций в процессе решения, меньше нуля — решение не найдено. Первый входной аргумент `fun` является именем файл-функции, вычисляющей исследуемую функцию $f(x)$, которая может зависеть от нескольких параметров. Значения параметров, в случае их наличия, передаются в последних аргументах `P1`, `P2`,...

Работа с математическими функциями, которые зависят не только от аргумента, но и от некоторых параметров, описана в разд." Интегралы, зависящие от параметра" главы 6.

Входным аргументом файл-функции `fun` должен быть вектор, длина которого совпадает с числом переменных, т. е. компонент вектора x. Неиспользуемые векторы и матрицы ограничений заменяются в списке входных аргументов `fmincon` квадратными скобками (пустым массивом). Начальное приближение к решению указывается в `x0`. Список входных аргументов `fmincon` содержит параметр `options`, предназначенный для установок, управляющих процессом минимизации. Изучению данного вопроса посвящено несколько разделов.

См. разд. "Параметры оптимизации" и "Решение большой системы нелинейных уравнений" данной главы.

Нелинейные ограничения (неравенства и равенства) программируются в файл-функции, ее имя указывается в аргументе `nonlcon`. Входным аргументом `nonlcon` является вектор `x`, соответствующий искомому вектору x, а двумя выходными аргументами — вектора `c` и `ceq` левых частей нелинейных ограничений c и c_{eq}. Последние *идущие подряд* аргументы могут быть опущены, если они не используются, например, в случае отсутствия нелинейных ограничений и параметров применяется следующий вызов `fmincon`:

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
```

Найдите решение следующей простой задачи (очевидно, что решение — нулевой вектор).

$$\min 3(x_1)^2 + 2(x_2)^2, \quad (x_1)^2 + (x_1)^2 \le 1.$$

Обратите внимание, что имеется только одно нелинейное ограничение в виде неравенства, которое следует преобразовать к виду $c(x) \le 0$, перенеся единицу в левую часть. Написание файл-функции, вычисляющей $3(x_1)^2 + 2(x_2)^2$, не представляет труда (листинг 17.2). При программировании нелинейного ограничения учтите, что соответствующая файл-функция возвращает два вектора левых частей нелинейных ограничений (листинг 17.3). В рассматриваемом примере первый вектор состоит только из одного компонента, а второй должен быть пустым, поскольку нет ограничений в виде равенств.

Листинг 17.2. Минимизируемая функция

```
function f = myfun(x)
f = 3*x(1)^2+2*x(2)^2;
```

Листинг 17.3. Файл-функция с ограничениями

```
function [c, ceq] = mycon(x)
% Задание ограничений
c(1) = x(1)^2+x(2)^2-1; % ограничения в виде неравенства
% Правая часть ограничений-равенств является пустым массивом,
% поскольку данных ограничений нет
ceq = [];
```

Выполнение fmincon, например из командной строки

```
>> x = fmincon('myfun', [0.7 0.7], [],[],[],[],[],[], 'mycon')
```

приводит к выводу некоторой информации о ходе вычислений и результата

```
x =
  1.0e-004 *
    0.1895   -0.0235
```

Обращение к fmincon с тремя выходными аргументами позволяет получить значение функции в точке минимума:

```
>> [x, f, flag] = fmincon('myfun', [0.7 0.7], [],[],[],[],[],[], 'mycon')
x =
  1.0e-004 *
    0.1895   -0.0235
f =
  1.0882e-009
flag =
     1
```

Значение flag большее нуля свидетельствует о том, что решение успешно найдено.

Нелинейные задачи

Нелинейное программирование не исчерпывает класс нелинейных задач, которые могут быть решены в ToolBox Optimization. Функции ToolBox fgoalattain, fminmax, fseminf позволяют исследовать задачи о достижении границы, находить решение в задаче о минимаксе. Ниже приведены формулировки задач и интерфейс функций, предназначенных для их решения. Ограничения на переменные могут быть как линейные, так и нелинейные, в общем случае они совпадают с ограничениями в задаче нелинейного программирования, описанной выше.

Задача о достижении границы: $\min\gamma$, $F(x)-w\gamma\le g$ решается при помощи функции fgoalattain:

```
x = fgoalattain(fun,x0,g,w,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)
```

Входной аргумент fun является именем файл-функции, вычисляющей вектор-функцию $F(x)$. Аргументы g и w — векторы, длина которых совпадает с числом значений вектор-функции. Использование остальных аргументов аналогично fmincon.

Минимаксная задача имеет вид:

$$\min_{x} \max_{\{F_i\}} \{F_i(x)\}.$$

Ограничения на x в минимаксной задаче такие же, как и в задаче нелинейного программирования. Функция `fminimax` предназначена для решения данной задачи, ее применение в самом общем случае выглядит следующим образом:

```
x = fminimax(fun,x0,g,w,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)
```

Первый входной аргумент `fminimax` является файл-функцией, возвращающей вектор значений $\{F_i(x)\}$.

Решение нелинейных уравнений

ToolBox Optimization позволяет решить систему нелинейных уравнений, в которой число неизвестных совпадает с числом уравнений. Общий вид системы задается при помощи нелинейной вектор-функции: $F(x) = 0$, а покомпонентная запись имеет вид

$$\begin{cases} F_1(x_1, x_2, \ldots x_n) = 0; \\ F_2(x_1, x_2, \ldots x_n) = 0; \\ \ldots \\ F_n(x_1, x_2, \ldots x_n) = 0. \end{cases}$$

Первым аргументом `fsolve` является имя файл-функции, вычисляющей $F(x)$. Данная файл-функция принимает на входе вектор аргументов и возвращает вектор значений $F(x)$ той же длины. Начальное приближение указывается во втором аргументе `fsolve`. В общем случае вызов `fsolve` выглядит следующим образом:

```
x = fsolve(fun,x0,options,P1,P2,...)
```

где необязательные аргументы `P1`, `P2`,... есть значения параметров, от которых может зависеть левая часть уравнения.

Замечание

Алгоритм, реализованный в функции `fsolve`, основан на методе наименьших квадратов, поэтому решением считается та точка, в которой значения $F(x)$ достаточно малы. На самом деле вблизи найденной точки корня может и не быть.

Решите систему из двух нелинейных уравнений

$$\begin{cases} x_1(2 - x_2) = \cos x_1 \cdot e^{x_2}; \\ 2 + x_1 - x_2 = \cos x_1 + e^{x_2}. \end{cases}$$

Файл-функция, соответствующая правой части системы, программируется просто (листинг 17.4).

Листинг 17.4. Файл-функция, вычисляющая левую часть системы уравнений

```
function F = mysys(x)
F(1) = x(1)*(2-x(2))-cos(x(1))*exp(x(2));
F(2) = 2+x(1)-x(2)-cos(x(1))-exp(x(2));
```

Указание начальной точки (0, 0) в функции `fsolve` приводит к следующему результату:

```
>> [x,f] = fsolve('mysys', [0 0])
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
    0.7391    0.4429
f =
  1.0e-012 *
    0.5649    0.3442
```

Алгоритм решения определенных классов задач, включая и нахождение корней системы нелинейных уравнений, основан на методе наименьших квадратов.

Метод наименьших квадратов

Метод наименьших квадратов применяется, например, для решения систем линейных уравнений, в которых число неизвестных не совпадает с числом уравнений.

См. разд. "Переопределенные и недоопределенные системы" главы 6.

Функция `lsqnonneg` предназначена для поиска только *неотрицательных* решений систем $Cx = d$, т. е. векторов x, все компоненты которых больше либо равны нуля. По существу, при помощи метода наименьших квадратов решается задача на нахождение минимума $\min_x \|Cx - d\|^2$ среди всех $x_i \geq 0$. Через $\|\cdot\|$ обозначена вторая векторная норма, являющаяся квадратным корнем из суммы квадратов компонентов вектора. Достаточно общий вариант вызова `lsqnonneg` выглядит следующим образом:

```
[x,resnorm,residual,flag]=lsqnonneg(C,d,x0)
```

где первые два входных аргумента содержат матрицу и вектор системы, `x0` — начальное приближение, в `x` возвращается решение, а в дополнительных

выходных аргументах возвращается норма невязки, вектор невязки и информация о завершении процесса вычислений.

Описание значений `flag` *см. в разд. "Нелинейное программирование" данной главы.*

Выходные аргументы и `x0` являются необязательными параметрами, если значение `x0` не указано, то по умолчанию используется нулевое начальное приближение.

Функция `lsqlin` позволяет найти решение более общей задачи с линейными ограничениями на решение

$$\min_x \|Cx - d\|^2; \qquad Ax \le b; \qquad A_{eq}x = b_{eq}; \qquad lb \le x \le ub.$$

Подбор параметров

Предположим, что в некоторый физический закон $y = F(a_1, a_2, a_3, a_4, x)$ входят неизвестные параметры a_1, a_2, a_3 и a_4. Проделан ряд экспериментов и получено n опытных данных $(xdat_i, ydat_i)$, с целью установления значений параметров. Возникает вопрос, как выбрать параметры физического закона так, чтобы результаты эксперимента соответствовали ему некоторым наилучшим образом.

Решение задачи о подборе параметров в ToolBox Optimization основано на методе наименьших квадратов, т. е. находится минимум выражения

$$\frac{1}{2}\sum_{i=1}^{n}\left[F(a_1, a_2, a_3, a_4, xdat_i) - ydat_i\right]^2$$

по всевозможным значениям a_1, a_2, a_3 и a_4. Функция `lsqcurvefit` предназначена для решения задачи о подборе параметров (число параметров может быть произвольным). Обращение к ней практически не отличается от вызова других функций ToolBox:

```
x = lsqcurvefit(fun,a0,xdat,ydat)
```

Первый входной параметр файл-функции `fun` должен быть массивом, он служит для передачи значений параметров в файл-функцию. Второй аргумент `fun` является аргументом x функции $y = F(a_1, a_2, a_3, a_4, x)$.

Дополнительно могут быть поставлены ограничения на параметры, которые в векторной записи имеют вид $lb \le a \le ub$. В этом случае векторы lb и ub задаются в пятом и шестом аргументах `lsqcurvefit`:

```
x = lsqcurvefit(fun,a0,xdat,ydat,lb,ub)
```

Определите параметры в следующем примере. Пусть $F = a_1 e^{a_2 x} + a_3 \sin a_4 x$, в результате эксперимента получены следующие значения:

$xdat = [0.0\ 0.1\ 0.2\ 0.3\ 0.4\ 0.5\ 0.6\ 0.7\ 0.8\ 0.9\ 1.0]$

$ydat = [1.1\ 2.1\ 3.5\ 3.9\ 4.3\ 5.1\ 4.2\ 4.0\ 3.3\ 2.2\ 2.1]$

Известно, что значение каждого из параметров может находиться в промежутке $[-10, 10]$. Первым шагом является создание файл-функции (листинг 17.5). Параметры в `fitfun` передаются через вектор `par`.

Листинг 17.5. Файл-функция, зависящая от вектора параметров и аргумента

```
function y = fitfun(a, x)
y = a(1)*exp(x*a(2))+a(3)*sin(a(4)*x);
```

Теперь следует задать векторы `xdat` и `ydat`, отобразить данные на графике, выбрать начальное приближение `a0`, построить график при начальном приближении, задать границы и вызвать `lsqcurvefit`. После отыскания параметров необходимо вывести график функции и убедиться, что функция с найденными параметрами достаточно точно описывает данные. Последовательность команд, приведенная в листинге 17.6, реализует вышеописанные действия, в результате получается график, изображенный на рис. 17.1.

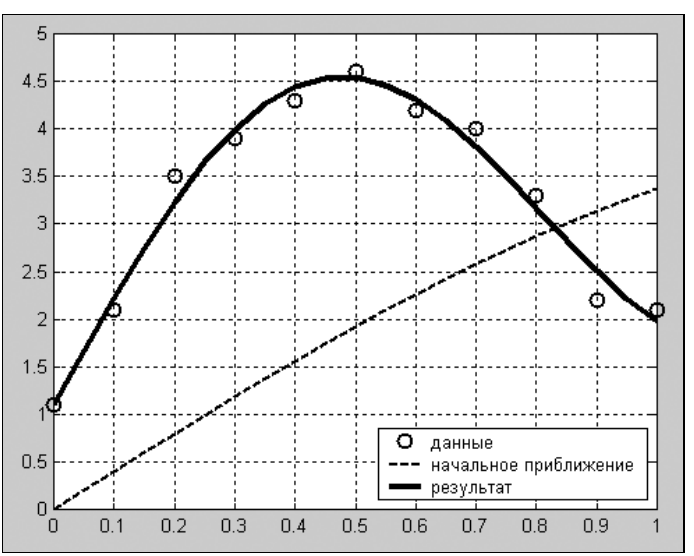


Рис. 17.1. Результат подбора параметров

В командное окно вывелся вектор найденных значений параметров:

```
a =
1.1678    2.7003    3.7505
```

Листинг 17.6. Подбор параметров

```
% Ввод данных
xdat = (0:0.1:1);
ydat = [1.1 2.1 3.5 3.9 4.3 4.6 4.2 4.0 3.3 2.2 2.1];
% Отображение данных на графике
plot(xdat, ydat, 'o');
grid on
% Выбор начального приближения
a0 = [0.0 0.0 4.0 1.0];
% Построение графика функции от начального приближения
x = [0:0.05:1];
ya0 = fitfun(a0, x);
hold on;
plot(x, ya0, '--b')
% Задание границ области параметров
LB   = [-10 -10 -10 -10];
UB   = [10 10 10 10];
% Подбор параметров, точка с запятой в конце команды не ставится для
% вывода результата в командное окно
a = lsqcurvefit('fitfun',a0, xdat, ydat, LB, UB)
% Визуализация функции с найденными значениями параметров
ya = fitfun(a,x);
Hfit = plot(x, ya);
set(Hfit, 'LineWidth', 2)
legend('данные', 'начальное приближение', 'результат', 4)
```

Обратите внимание, что начальное приближение оказывает существенное влияние на получаемый результат. Если, к примеру, в качестве начального приближения к искомым параметрам взять вектор `a0=[4.0 -1.0 0.0 0.0]`, то подбор параметров не приведет к хорошему результату (рис. 17.2).

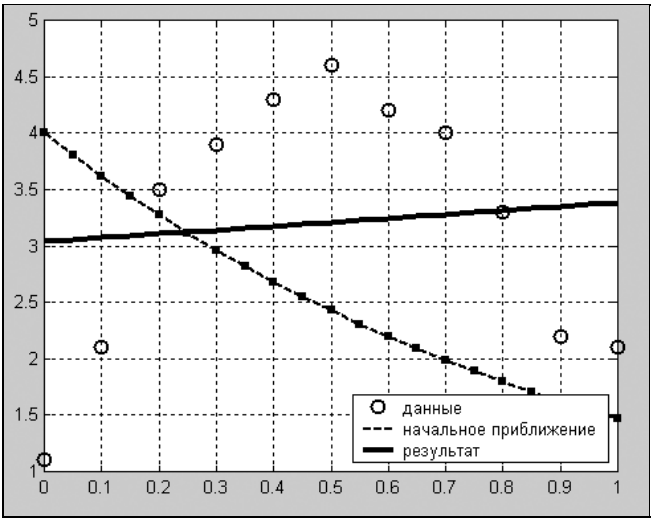


Рис. 17.2. Результат подбора параметров при плохом начальном приближении

Параметры оптимизации

Задачи оптимизации условно разделены в ToolBox Optimization на два класса: Medium-Scale (средние) и Large-scale (большие), в зависимости от размерности задачи, т. е. числа переменных. Для решения каждого из классов задач реализованы соответствующие численные методы, объяснение которых выходит за рамки данной книги. Впрочем, краткая информация об алгоритмах содержится в справочной системе по ToolBox. Пользователь имеет возможность отслеживать ход вычислительного процесса и задавать параметры, управляющие вычислениями.

Для установки параметров следует сформировать структуру `options` при помощи функции `optimset` и затем указать данную структуру в качестве параметра функции ToolBox Optimization, выбранной для решения поставленной задачи.

Формирование структуры `options` *для функций* `fzero`, `fminbnd` *и* `fminsearch` *описано в разд. "Задание дополнительных параметров" главы 6.*

Функции, предназначенные для решения задач оптимизации, обладают достаточно большим набором параметров. Команда `optimset`, вызванная с именем функции в качестве входного аргумента, выводит в командное окно таблицу всех допустимых установок для алгоритмов, реализованных в функции. Получите установки функции `fsolve`, использовав обращение

optimset('fsolve'). В командном окне отображаются параметры и их значения, ниже приведены параметры, используемые в данном разделе.

```
ans =
                    ...
          Diagnostics: 'off'
              Display: 'final'
             Jacobian: 'off'
           LargeScale: 'on'
          MaxFunEvals: '100*numberOfVariables'
              MaxIter: 400
           MaxPCGIter: 'max(1,floor(numberOfVariables/2))'
      PrecondBandWidth: 0
               TolFun: 1.0000e-006
               TolPCG: 0.1000
                 TolX: 1.0000e-006
```

Назначение параметров Display, MaxFunEvals, MaxIter, TolFun, TolX такое же, как и у функций fzero, fminbnd и fminsearch.

См. табл. 6.1 в разд. "Задание дополнительных параметров" главы 6.

Следует иметь в виду, что все параметры делятся на три группы в зависимости от размерности задачи.

- Параметры установки Large-scale-алгоритмов.
- Параметры установки Medium-scale-алгоритмов.
- Общие параметры для Large- и Medium-scale-алгоритмов.

Применяемый алгоритм зависит от значения LargeScale, 'on' разрешает использование Large-scale-алгоритма, если он допустим для решаемой задачи. Имеются некоторые ограничения на область применения Large-scale-алгоритмов, в частности, получающаяся в ходе решения система не должна быть недоопределенной, т. е. число уравнений не может превосходить число неизвестных.

Задание якобиана вектор-функции исследуемой задачи значительно ускоряет вычисления. Файл-функция, вычисляющая левую часть исследуемой системы нелинейных уравнений (например, в случае fsolve), может иметь два выходных аргумента, во втором возвращается якобиан системы. Соответствующий пример разобран ниже.

Такие параметры, как JacobPattern, LevenbergMarquardt, MaxPCGIter, TolPCG, PrecondBandWidth соответствуют Large-scale-алгоритмам. Получающиеся в процессе вычислений системы линейных уравнений решаются методом предобусловленных сопряженных градиентов (PCG), значение PrecondBandWidth за-

дает ширину верхней полуленты в исходной матрице, на основе которой строится предобусловливатель. По умолчанию выбирается диагональный предобусловливатель, т. е. ширина верхней полуленты равна нулю. Максимальное число итераций в методе сопряженных градиентов определяется значением `MaxPCGIter`, а критерий останова — `TolPCG`. Вместо вычисления якобиана левой части системы нелинейных уравнений можно задать его шаблон, т. е. расположение ненулевых элементов, в `JacobPattern`, тогда только ненулевые элементы якобиана будут аппроксимироваться в ходе вычислений по методу конечных разностей, что значительно ускорит вычисления.

Примеры

Следующие разделы посвящены применению некоторых установок, управляющих вычислительным процессом, на примере решения большой системы нелинейных уравнений. Разобрано создание приложения с графическим интерфейсом пользователя, облегчающее использование возможностей ToolBox Optimization.

Решение большой системы нелинейных уравнений

Рассмотрим пример на применение Large-scale-алгоритма, в котором требуется запрограммировать файл-функцию не только для вычисления левой части системы, но и якобиана и, кроме того, произвести некоторые установки, определяющие процесс вычислений. Необходимо решить систему нелинейных уравнений $F(x) = 0$ для $n = 1000$, уравнения системы приведены ниже.

$$\begin{cases} 2x_1^2 - x_2 = 1; \\ -x_1 + 2x_2^2 - x_3 = 1; \\ \cdots \\ -x_{n-1} + 2x_n^2 = 1. \end{cases}$$

Файл-функция `largesys`, которая вычисляет левую часть системы, программируется достаточно просто. Следует в цикле определить значения вектор-функции (листинг 17.7) и использовать `fsolve`.

Листинг 17.7. Файл-функция, вычисляющая левую часть системы

```
function F = largesys(x)
n = length(x);
F = rand(n,1);
```

```
F(1) = 2*x(1)^2-x(2)-1;
for i = 2:n-1
   F(i) = -x(i-1)+2*x(i)^2-x(i+1)-1;
end
F(n) = -x(n-1)+2*x(n)^2-1;
```

Размерность задачи достаточно большая и решение может занять значительное время, если не принимать во внимание разреженную структуру исходной системы. В данном примере вычисление якобиана дает существенный выигрыш во времени. Якобиан решаемой системы имеет достаточно простую структуру. Несложно вычислить якобиан по формуле

$$J = \left(\frac{\partial F_i}{\partial x_j} \right)_{i,j=1,2,\ldots n}.$$

Заметьте, что в результате получается сильно разреженная (трехдиагональная) матрица:

$$J = \begin{bmatrix} 4x_1 & -1 & & 0 & 0 \\ -1 & 4x_2 & & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & & 4x_{n-1} & -1 \\ 0 & 0 & & -1 & 4x_n \end{bmatrix}.$$

Якобиан следует создать при помощи функции `sparse`, предназначенной для инициализации разреженных матриц.

Работа с разреженными матрицами описана в главе 16.

Очевидно, что якобиан представляется суммой $J = D + D1 + D1^T$ разреженных матриц, где

$$D = \begin{bmatrix} 4x_1 & 0 & & 0 & 0 \\ 0 & 4x_2 & & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & & 4x_{n-1} & 0 \\ 0 & 0 & & 0 & 4x_n \end{bmatrix}; \quad D1 = \begin{bmatrix} 0 & -1 & & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & & 0 & -1 \\ 0 & 0 & & & 0 \end{bmatrix}.$$

Теперь следует дополнить файл-функцию `largesys` нахождением разреженного представления для якобиана и вернуть результат во втором выходном аргументе. Вычислять якобиан необходимо, если `largesys` вызывается с двумя выходными параметрами. Используйте условный оператор `if` и переменную `nargout` для проверки числа выходных аргументов.

Проверка числа параметров, с которыми вызвана файл-функция, описана в разд. "Условный оператор if" главы 7.

Текст модернизированной файл-функции `sparse` приведен в листинге 17.8.

Листинг 17.8. Файл-функция `largesys`, возвращающая якобиан

```
function [F, J] = largesys(x)
n = length(x);
F = rand(n,1);
F(1) = 2*x(1)^2-x(2)-1;
for i = 2:n-1
   F(i) = -x(i-1)+2*x(i)^2-x(i+1)-1;
end
F(n) = -x(n-1)+2*x(n)^2-1;
% Если число выходных аргументов более единицы, то требуется найти
% разреженное представление якобиана
if nargout>1
   % якобиан является суммой трех матриц J = D+D1+D1'
   % Формирование диагонали матрицы D
   d = 4*x;
   % Инициализация разреженного представления для матрицы D
   Diag = sparse(1:n, 1:n, d, n, n);
   % Формирование вектора побочной диагонали
   d2 = -ones(1,n-1);
   % Инициализация разреженного представления для матрицы D1
   D1 = sparse(2:n, 1:n-1, d2, n, n);
   % Вычисление разреженного якобиана
   J = Diag+D1+D1';
end
```

Обращение к `fsolve` оформите в файл-программе (листинг 17.9). Задайте число переменных и вектор начального приближения, затем сформируйте структуру `options`. Учтите, что параметр `Jacobian` должен иметь значение `on` для использования якобиана в процессе вычислений.

Листинг 17.9. Файл-программа для решения большой системы нелинейных уравнений

```
n = 1000; % число переменных
x0 = ones(1,n); % начальное приближение
```

```
% Формирование структуры options
options = optimset('Display', 'iter', 'Jacobian', 'on',...
   'Diagnostics', 'on');
% Решение системы нелинейных уравнений и вывод результата
% в командное окно
x = fsolve('largesys', x0, options)
```

В результате выполнения операторов листинга 17.9 в командное окно выводится информация об исследуемой системе, поскольку параметр `Diagnostics` имеет значение `on`:

```
Diagnostic Information
Number of variables: 1000
Functions
 Objective and gradient:                largesys
Algorithm selected
   large-scale
```

Работа `fsolve` сопровождается отображением сведений о каждом шаге вычислительного процесса (`Display` установлен в `iter`). Сведения представлены в виде таблицы:

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
1	2	998	1	3	0
2	3	27.808	10	1.3	2
3	4	0.0770947	1.60717	0.569	2
4	5	0.000288307	0.0612466	0.0347	3
5	6	1.65907e-006	0.00337864	0.00204	3
6	7	1.23544e-008	0.000297358	0.000245	3
7	8	9.85685e-011	2.34199e-005	1.62e-005	3

Столбец `Iteration` содержит номер итерации; `Func-count` — число вызовов функции; `f(x)` — сумма квадратов значений левых частей уравнений системы; `Norm of step` — норма шага на текущей итерации; `First-order optimality` — бесконечная норма градиента, вычисленного в текущем приближении; `CG-iterations` — число итераций предобусловленного метода сопряженных градиентов на каждой итерации вычислительного процесса.

После таблицы в командное окно выводится вектор-строка решения:

```
x =
  Columns 1 through 7
    1.0746    1.3096    1.3553    1.3640    1.3656    1.3660    1.3660
...
```

```
Columns 995 through 1000
    1.3660    1.3656    1.3640    1.3553    1.3096    1.0746
```

Данный раздел описывает только решение большой системы нелинейных уравнений при помощи `fsolve`. Перед решением других задач оптимизации полезно предварительно узнать допустимые параметры, указав в качестве аргумента `optimset` имя используемой функции ToolBox Optimization.

Пример приложения с GUI

ToolBox Optimization не имеет приложения с графическим интерфейсом пользователя для доступа к функциям оптимизации. Решение оптимизационных задач значительно упростилось бы при наличии соответствующего приложения. Текущий раздел посвящен написанию приложения с графическим интерфейсом для решения задачи о подборе параметров.

Задача состоит в таком подборе параметров некоторой функции, чтобы она наилучшим образом удовлетворяла набору данных. Решение задачи о подборе параметров с использованием `lsqcurvefit` описано выше на примере функции $F = a_1 e^{a_2 x} + a_3 \sin a_4 x$.

См. разд. "Подбор параметров" данной главы.

Регулярное использование файл-программы, приведенной в листинге 17.6, для различных данных и функций не очень удобно — требуется изменять имя файла и файл-функции. При выборе начального приближения и задании границ также необходимо вносить изменения в файл-программу. Создание приложения с графическим интерфейсом позволит существенно экономить время при подобной многократной обработке данных. Приложение должно предоставлять пользователю контроль над начальными установками, производить процедуру подбора параметров и отображать графическое представление результата. Приложение Подбор параметров, окно которого изображено на рис. 17.3, обладает вышеописанными возможностями.

Кнопка **Считать данные** приводит к появлению диалогового окна открытия файла, в котором пользователь выбирает нужный файл с данными, имя файла заносится в область **Файл с данными**. Считанные данные отображаются маркерами на осях в окне приложения. Имя файл-функции, вычисляющей функцию с параметрами, пользователь задает в строке ввода **Файл-функция**. Соответствующие строки позволяют выбрать начальное приближение, верхнюю и нижнюю границы параметров. График начального приближения строится при нажатии на кнопку **График для нач. прибл.** Кнопка **ПОДБОР** служит для запуска процедуры подбора параметров. Найденные значения выводятся в область **Оптимальные параметры**. Удаление графиков производится при помощи кнопки **Очистить оси**.

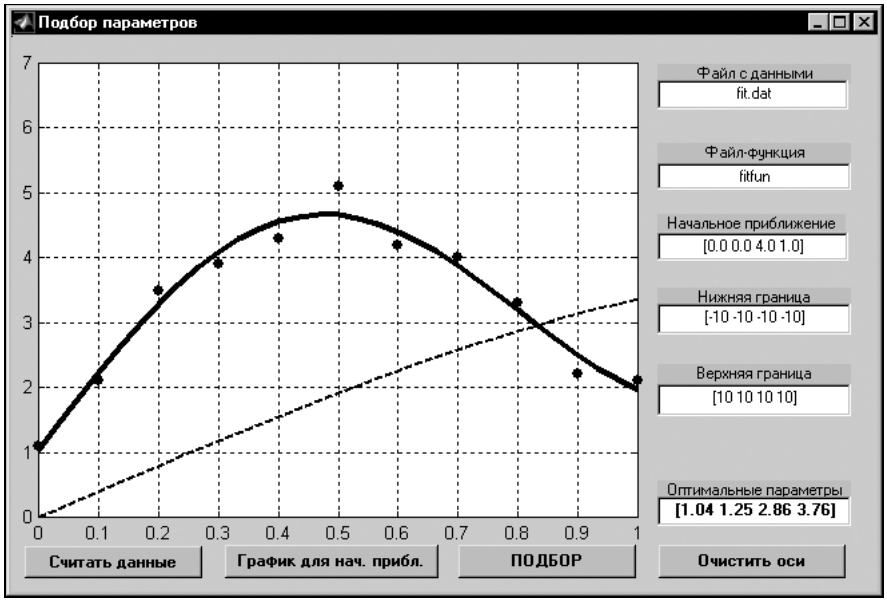


Рис. 17.3. Окно приложения Подбор параметров

Создание и программирование вышеописанного приложения Подбор параметров не представляет большого труда. Основные вопросы, связанные с программированием приложений, разобраны в третьей части книги. Файл-функция, реализующая обработку событий элементов управления приложения для версии 5.3, приведена в листинге 17.10. Подфункции обработки событий для версии 6.*x* содержатся в листинге 17.11. Имена основных объектов, расположенных в окне приложения, помещены в табл. 17.2.

Таблица 17.2. *Имена объектов приложения Подбор параметров*

Объект	Имя
Кнопка **Считать данные**	`LoadBtn`
Кнопка **График для нач. прибл.**	`GuessBtn`
Кнопка **ПОДБОР**	`FitBtn`
Кнопка **Очистить оси**	`ClearBtn`
Область **Файл с данными**	`DataFileEdt`
Строка ввода **Файл-функция**	`FunEdt`
Строка ввода **Начальное приближение**	`GuessEdt`
Область **Оптимальные параметры**	`ParEdt`

Листинг 17.10. Файл-функция `fitprog` с обработкой событий для версии 5.3

```
function fitprog(action)
global xdat ydat % вектора с данными
% Обработка событий от элементов интерфейса
switch(action)
case 'ClearBtnClick'
   % Очистка осей
   cla;
case 'LoadBtnClick',
   % Нажата кнопка "Считать данные"
   % Вывод диалогового окна открытия файла
   DataFileStr = uigetfile('*.dat', 'Открыть файл с данными');
   if DataFileStr ~= 0
     % Запись в область ввода "Файл с данными" имени файла
     HDataFileEdt = findobj('Tag', 'DataFileEdt');
     set(HDataFileEdt, 'String', DataFileStr);
     % Извлечение данных из файла и запись их
     % в глобальные векторы xdat, ydat
     data = load(DataFileStr);
     xdat = data(:,1)';
     ydat = data(:,2)';
     % Построение графика данных маркерами
     Hdata =  plot(xdat, ydat, '.r');
     set(Hdata, 'MarkerSize', 15);
     % Очистка строки ввода значений параметров "Оптимальные параметры"
     HParEdt = findobj('Tag', 'ParEdt');
     set(HParEdt, 'String', '')
   end
case 'GuessBtnClick'
   % Считывание вектора начального приближения для параметров из
   % строки ввода "Начальное приближение"
   HGuessEdt = findobj('Tag', 'GuessEdt');
   ParStr = ['par0=', get(HGuessEdt,'String')];
   % Выполнение строки, запись начального приближения в вектор par0
   eval(ParStr);
   % Построение исследуемой функции от начального приближения
   % Задание вектора значений абсцисс
```

```
   x = (min(xdat):(max(xdat)-min(xdat))/30:max(xdat));
   % Вычисление файл-функции, имя которой пользователь ввел в строке
   % ввода "Файл-функция"
   HFunEdt = findobj('Tag', 'FunEdt')
   FunName = get(HFunEdt, 'String')
   eval(['y =', FunName, '(par0, x);']);
   % Отображение графика для начального приближения
   plot(x, y, '--b' )
case 'FitBtnClick'
   % Задание установок для подбора параметров
   options=optimset('Display','iter');
   % Формирование строки с командой для засылки в вектор par0
   % начального приближения
   HGuessEdt = findobj(gcbf, 'Tag', 'GuessEdt');
   ParStr = ['par0=', get(HGuessEdt,'String')];
   % Выполнение строки, запись начального приближения в вектор par0
   eval(ParStr);
   % Считывание нижней и верхней границы из строк ввода "Нижняя
   % граница" и "Верхняя граница" и занесение их в векторы LB и UB
   HLBEdt = findobj('Tag', 'LBEdt');
   HUBEdt = findobj('Tag', 'UBEdt');
   eval(['LB=', get(HLBEdt, 'String')]);
   eval(['UB=', get(HUBEdt, 'String')]);
   % Занесение в FunName имени функции из строки ввода "Файл-функция"
   HFunEdt = findobj('Tag', 'FunEdt');
   FunName = get(HFunEdt, 'String')
   % Вызов функции lsqcurvefit для подбора параметров
   par = lsqcurvefit(FunName,par0, xdat, ydat, LB, UB, options);
   % Запись результатов в область "Оптимальные параметры"
   HParEdt = findobj('Tag', 'ParEdt');
   ParStr = mat2str(par, 3);
   set(HParEdt, 'String', ParStr);
   % Построение исследуемой функции от найденных параметров
   x = (min(xdat):(max(xdat)-min(xdat))/30:max(xdat));
   eval(['y =', FunName, '(par, x);']);
   Hplot = plot(x, y, '-g')
   set(Hplot(1), 'LineWidth', 2)
end
```

Листинг 17.11. Подфункции обработки событий для версии 6.*x*

```
function varargout = LoadBtn_Callback(h, eventdata, handles, varargin)
   % Нажата кнопка "Считать данные"
   % Вывод диалогового окна открытия файла
   DataFileStr = uigetfile('*.dat', 'Открыть файл с данными');
   if DataFileStr ~= 0
     % Запись в область ввода "Файл с данными" имени файла
     set(handles.DataFileEdt, 'String', DataFileStr);
     % Извлечение данных из файла и запись их в поля структуры handles
     data = load(DataFileStr);
     handles.xdat = data(:,1)';
     handles.ydat = data(:,2)';
     % Сохранение структуры handles
     guidata(gcbo, handles)
     % Построение графика данных маркерами
     Hdata =  plot(handles.xdat, handles.ydat, '.r');
     set(Hdata, 'MarkerSize', 15);
     % Очистка области ввода значений параметров "Оптимальные параметры"
     set(handles.ParEdt, 'String', '')
   end
function varargout = PlotGuessBtn_Callback(h, eventdata, handles,...
     varargin)
   % Считывание вектора начального приближения для параметров из
   % строки ввода "Начальное приближение"
   ParStr = ['par0=', get(handles.GuessEdt,'String')];
   % Выполнение строки, запись начального приближения в вектор par0
   eval(ParStr);
   % Построение исследуемой функции от начального приближения
   % Задание вектора значений абсцисс
   x = (min(handles.xdat):(max(handles.xdat)-...
        min(handles.xdat))/30:max(handles.xdat));
   % Вычисление файл-функции, имя которой пользователь ввел в строке
   % ввода "Файл-функция"
   FunName = get(handles.FunEdt, 'String')
   eval(['y =', FunName, '(par0, x);']);
   % Отображение графика для начального приближения
   plot(x, y, '--b' )
```

```
function varargout = FitBtn_Callback(h, eventdata, handles, varargin)
   % Задание установок для подбора параметров
   options=optimset('Display','iter');
   % Формирование строки с командой для засылки в вектор par0
   % начального приближения
   ParStr = ['par0=', get(handles.GuessEdt,'String')];
   % Выполнение строки, запись начального приближения в вектор par0
   eval(ParStr);
   % Считывание нижней и верхней границы из строк ввода "Нижняя
   % граница" и "Верхняя граница" и занесение их в векторы LB и UB
   eval(['LB=', get(handles.LBEdt, 'String')]);
   eval(['UB=', get(handles.UBEdt, 'String')]);
   % Занесение в FunName имени функции из строки ввода "Файл-функция"
   FunName = get(handles.FunEdt, 'String')
   % Вызов функции lsqcurvefit для подбора параметров
   par = lsqcurvefit(FunName,par0, handles.xdat, handles.ydat, LB, UB,...
      options);
   % Запись результатов в область "Оптимальные параметры"
   ParStr = mat2str(par, 3);
   set(handles.ParEdt, 'String', ParStr);
   % Построение исследуемой функции от найденных параметров
   x = (min(handles.xdat):(max(handles.xdat)-...
        min(handles.xdat))/30:max(handles.xdat));
   eval(['y =', FunName, '(par, x);']);
   Hplot = plot(x, y, '-g')
   set(Hplot(1), 'LineWidth', 2 )
function varargout = ClearBtn_Callback(h, eventdata, handles, varargin)
   % Очистка осей
   cla;
```

Глава 18



Символические вычисления

В состав MatLab входит ToolBox Symbolic Math, предназначенный для вычислений в символическом виде. Преобразование выражений, разыскание аналитического решения задач линейной алгебры, дифференциального и интегрального исчисления, получение численного результата с любой точностью — вот далеко не полный перечень возможностей, предоставляемых данным ToolBox. Функции ToolBox Symbolic Math реализуют интерфейс между средой MatLab и библиотекой функций, являющихся вычислительным ядром Maple, причем работа в MatLab не требует установки Maple. Расширение ToolBox позволяет пользователям, имеющим опыт работы в Maple, использовать ресурсы ядра Maple практически в полном объеме, включая и программирование в Maple.

Символические переменные и функции

Объектно-ориентированный подход, реализованный в MatLab, позволил сделать работу с символическими выражениями простой и удобной. Если вы освоили работу с арифметическими выражениями, то символические преобразования не должны вызвать затруднений.

Определение переменных и функций и работа с ними

Символические переменные и функции являются объектами класса `sym object`, в отличие от числовых переменных, которые содержатся в массивах `double array`. Символический объект создается при помощи функции `syms`. Команда

```
>> syms x a b
```

создает три символические переменные `x`, `a` и `b`. Размер памяти, отводимый под символические переменные, достаточно большой, посмотрите информацию об определенных только что переменных:

```
>> whos x a b
  Name      Size          Bytes    Class
```

```
  a          1x1          126      sym object
  b          1x1          126      sym object
  x          1x1          126      sym object
Grand total is 6 elements using 378 bytes
```

Конструирование символических функций от переменных класса `sym object` производится с использованием обычных арифметических операций и обозначений для встроенных математических функций, например:

```
>> f=(sin(x)+a)^2*(cos(x)+b)^2/sqrt(abs(a+b))
f =
(sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)
```

Запись формулы для выражения в одну строку не всегда удобна, более естественный вид выражения выводит в командное окно функция `pretty`:

```
>> pretty(f)
                                  2              2
                        (sin(x) + a)  (cos(x) + b)
                        ---------------------------
                                      1/2
                              | a + b |
```

Определенная функция `f` также является символической переменной типа `sym object`, в чем несложно убедиться при помощи команды `whos`:

```
>> whos f
  Name      Size         Bytes    Class
  f         1x1          204      sym object
Grand total is 41 elements using 204 bytes
```

Имеющиеся символические переменные и функции позволяют образовывать новые символические выражения:

```
>> syms y
>> g = (exp(-y)+1)/exp(y)
g =
(exp(-y)+1)/exp(y)
>> h=f*g
h =
(sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)*(exp(-y)+1)/exp(y)
>> pretty(h)
                              2              2
                   (sin(x) + a)  (cos(x) + b)  (exp(-y) + 1)
                   ----------------------------------------
                                      1/2
                            | a + b |    exp(y)
```

Символическую функцию можно создать без предварительного объявления переменных при помощи sym, входным аргументом которой является строка с выражением, заключенная в апострофы:

```
>> z = sym('c^2/(d+1)')
z =
c^2/(d+1)
>> pretty(z)
                                    2
                                   c
                                  -----
                                  d + 1
```

Замечание

Функция sym может быть использована для объявления символических переменных. Команда syms a,b,c эквивалентна последовательности a=sym('a'); b=sym('b'); c=sym('c').

При работе в области комплексных чисел следует указать, что определяемые переменные являются, в общем случае, комплексными. Комплексные символические переменные задаются командой syms с опцией unreal. Опция real означает, что переменные трактуются как вещественные. Убедитесь, что результат символических вычислений зависит от того, какие символические переменные используются — вещественные или комплексные. Объявите две вещественные переменные a и b, образуйте комплексное число, считая, что a является действительной частью, а b — мнимой, и найдите сопряженное к нему при помощи conj.

```
>> syms a b real
>> p=conj(a+i*b)
p =
a-i*b
```

Произведите аналогичные действия, предварительно объявив a и b как комплексные переменные:

```
>> syms a b unreal
>> q=conj(a+i*b)
q =
conj(a+i*b)
```

Обратите внимание на значения символических переменных p и q.

Матрицы и векторы

Символические переменные могут являться элементами матриц и векторов. Элементы строк матриц при вводе отделяются пробелами или запятыми, а столбцов — точкой с запятой, так же как и для обычных матриц. В результа-

те образуются символические матрицы и векторы, к которым применимы матричные и поэлементные операции и встроенные функции.

```
>> syms a b c d e f g h
>> A=[a b; c d]
A =
[ a, b]
[ c, d]
>> B=[e, f; g, h]
B =
[ e, f]
[ g, h]
>> C=A*B
C =
[ a*e+b*g, a*f+b*h]
[ c*e+d*g, c*f+d*h]
```

Функция `sym` позволяет преобразовать значения числовых переменных в символические. Введите массив типа `double array`

```
>> A = [1.3 -2.1 4.9
       6.9  3.7 8.5];
```

и образуйте соответствующий ему символический массив:

```
>> B = sym(A)
B =
[  13/10, -21/10,  49/10]
[  69/10,  37/10,   17/2]
```

При переходе от числовых к символическим выражениям используется запись чисел в виде рациональной дроби. Создайте вектор-столбец

```
>> c=[3.2; 0.4; -2.1];
```

и занесите в вектор *d* его символическое представление

```
>> d=sym(c);
```

Перемножьте матрицу *B* на вектор *d*, результат является символической переменной, причем все вычисления проделаны над рациональными дробями.

```
>> e=B*d
e =
[ -697/100]
[  571/100]
```

Использование рациональных дробей при выполнении символических вычислений означает, что всегда получается точный результат, не содержащий погрешность округления. Убедиться в вышесказанном можно на очень про-

стом примере. Установите формат `long` для отображения максимально возможного числа значащих цифр для значений числовых переменных и найдите сумму чисел 10^{10} и 10^{-10}.

```
>> format long
>> 1.0e+10+1.0e-10
ans =
    1.000000000000000e+010
```

Запишите каждое из чисел в символические переменные и снова вычислите сумму:

```
>> large = sym(1.0e10);
>> small = sym(1.0e-10);
>> s = large+small
s =
10000000000000000001/10000000000
```

Рациональная дробь является точным значением суммы. Разумеется, символические вычисления требуют значительных временных затрат по сравнению с обычными.

Преобразование в числовые значения

Вычисления с рациональными дробями позволяют получить значение символического выражения с любой степенью точности, т. е. найти сколь угодно много значащих цифр результата. Функция `vpa` предназначена для вычисления символических выражений:

```
>> c=sym('sqrt(2)');
>> cn=vpa(c)
cn =
1.4142135623730950488016887242097
```

По умолчанию удерживается тридцать две значащие цифры. Второй дополнительный входной параметр `vpa` позволяет производить вычисления с заданной точностью:

```
>> cn=vpa(c,70)
cn =
1.4142135623730950488016887242096980785696718753769480731766797379907324
```

Замечание

Второй аргумент `vpa` задает удерживаемое число значащих цифр только для данного вызова `vpa`. Для глобальной установки служит функция `digits`, во входном аргументе которой указывается требуемое количество цифр.

Важно понимать, что выходной аргумент функции `vpa` является символической переменной:

```
>> whos cn
  Name      Size          Bytes  Class
  cn        1x1             266  sym object
```

Для перевода символических переменных в числовые, т. е. переменные типа `double array`, используется функция `double`:

```
>> cnum=double(cn)
cnum =
   1.41421356237310
```

Графическое представление функций

Визуализация символической функции одной переменной осуществляется при помощи `ezplot`. Самый простой вариант использования `ezplot` состоит в указании символической функции в качестве единственного входного аргумента, при этом в графическое окно выводится график функции на отрезке $[-2\pi, 2\pi]$

```
>> f=sym('x^2*sin(x)');
>> ezplot(f)
```

Обратите внимание (рис. 18.1), что автоматически создается соответствующий заголовок. Вторым аргументом может быть задан вектор с границами отрезка, на котором требуется построить график функции

```
>> ezplot(f,[-3 2])
```

Функция `ezplot` имеет некоторые отличия от своего аналога — функции `fplot`, применяемой к числовым функциям. В частности, возможно указание символической функции, зависящей от двух аргументов:

```
>> z=sym('x^2+y^3');
>> ezplot(z,[-2 1 -3 4])
```

В данном случае выведется линия, на которой исследуемое выражение равно нулю.

Замечание

Пределы изменения определяются названиями аргументов. Первые два числа соответствуют первому по алфавиту аргументу, а последние — второму, например в `ezplot(z,[-2 1 -3 4])`, где `z=sym('x^2+a^3')`, считается, что `a` изменяется от –2 до 1, а `x` — от –3 до 4.

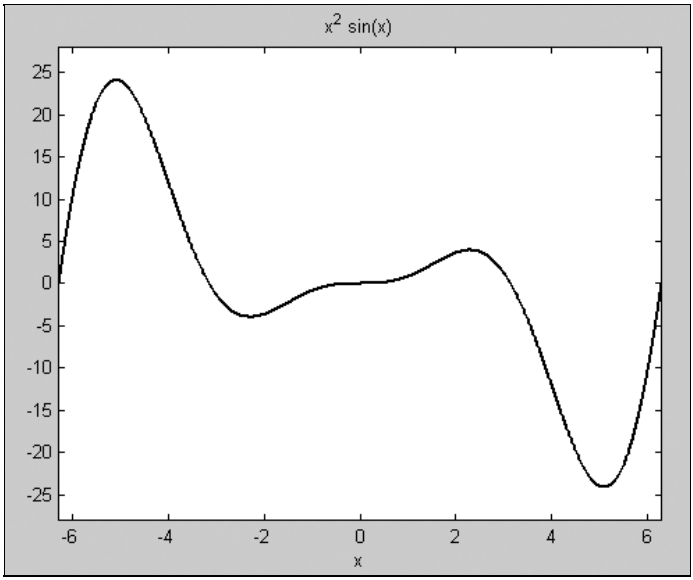


Рис. 18.1. График символической функции

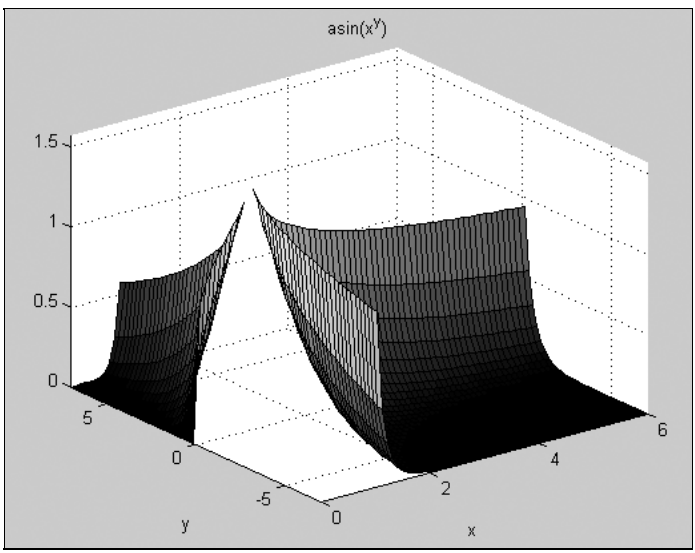


Рис. 18.2. График символической функции двух переменных

При помощи `ezplot` возможно также отображение параметрически заданных функций.

ToolBox Symbolic Math предоставляет пользователю целый набор средств для визуализации символических функций: `ezmesh`, `ezmeshc`, `ezplot`, `ezplot3`, `ezpolar`, `ezsurf`, `ezsurfc`. Функция `ezsurf` отображает график символической функции только для допустимых значений аргументов, остальные значения отбрасываются, что позволяет исследовать область определения функции двух переменных. Например, обращение

```
>> ezsurf('asin(x^y)', [0 6 -7 7])
```

приводит к построению графика, изображенного на рис. 18.2.

Упрощение и преобразование выражений

Сложные алгебраические и тригонометрические выражения нередко могут быть приведены к эквивалентным путем упрощения. ToolBox Symbolic Math имеет ряд сервисных функций, предназначенных для различных преобразований символических выражений. Пользователь может производить как стандартные операции над полиномами, так и использовать более общий алгоритм, предназначенный для упрощения выражений, которые содержат встроенные символические функции.

Операции с полиномами реализуют четыре функции: `collect`, `expand`, `horner` и `factor`. Вычисление коэффициентов при степенях независимой переменной производится с использованием функции `collect`. Введите полином и отобразите его в командном окне при помощи `pretty`.

```
>> p=sym('(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3');
>> pretty(p)
                        4           3           2
                (x + a)  + (x - 1)  - (x - a)  - a x + x - 3
```

Преобразуйте `p` к виду, содержащему степени `x` с соответствующими коэффициентами:

```
>> pc=collect(p);
>> pretty(pc)
      4               3              2   2                3       4         2
     x  + (1 + 4 a) x  + (-4 + 6 a ) x  + (4 + a + 4 a ) x + a  - 4 - a
```

По умолчанию в качестве переменной выбирается `x`, однако можно было считать, что `a` — независимая переменная, а `x` входит в коэффициенты полинома, зависящего от `a`. Второй аргумент функции `collect` предназначен для указания переменной, при степенях которой следует найти коэффициенты

```
>> pca=collect(p, 'a');
>> pretty(pca)
   4       3            2  2       3           4          3    2
  a  + 4 x a  + (-1 + 6 x ) a  + (4 x  + x) a + x  + (x - 1)  - x  - 3 + x
```

Функция `expand` представляет полином суммой степеней без приведения подобных слагаемых:

```
>> pe=expand(p);
>> pretty(pe)
     4       3     2  2       3    4    3     2                  2
    x  + 4 a x  + 6 x  a  + 4 x a  + a  + x  - 4 x  + 4 x - 4 + a x - a
```

Аргументом `expand` может быть не только полином, но и символическое выражение, содержащее тригонометрические, экспоненциальную и логарифмическую функции, например:

```
>> f=sym('sin(arccos(3*x))+exp(2*log(x))');
>> fe=expand(f);
>> pretty(fe)

                                        2 1/2    2
                                 (1 - 9 x )    + x
```

Символические полиномы разлагаются на множители функцией `factor`, если получающиеся множители имеют рациональные коэффициенты:

```
>> p=sym('x^5+13*x^4+215/4*x^3+275/4*x^2-27/2*x-18');
>> pf=factor(p);
>> pretty(pf)
                 1/4 (2 x + 1) (2 x - 1) (x + 6) (x + 4) (x + 3)
```

Представление числа в виде произведения простых чисел также выполняется при помощи `factor`:

```
>> syms a
>> a=sym('230010');
>> s=factor(a)
s =
(2)*(3)*(5)*(11)*(17)*(41)
```

Обратите внимание, что обращение

```
>> s1=factor(230010)
s1 =
     2     3     5    11    17    41
```

выводит в командное окно аналогичный результат, однако переменная `s` является символьной, а `s1` — вещественной, в чем несложно убедиться при помощи `whos`. MatLab является объектно-ориентированной средой, числовые переменные типа `double array` образуют *класс* со своим *методом* `factor`, запрограммированным в файл-функции \toolbox\matlab\specfun\factor.m. Метод `factor` класса, которому принадлежат символические переменные, реализован в другой файл-функции \toolbox\symbolic\@sym\factor.m. MatLab определяет по типу аргумента соответствующий метод класса и выполняет его.

Замечание

Пользователь MatLab может создавать собственные классы и определять их методы, в том числе и переопределять методы предка класса. Выполнение подобных действий требует понимания основ объектно-ориентированного программирования. Создание классов в MatLab описано в справочной системе в разделе "MATLAB Classes and Objects".

Упрощение выражений общего вида производится при помощи функций `simple` и `simplify`, которые основаны на разных подходах. Функция `simplify` реализует мощный алгоритм упрощения выражений, содержащих как тригонометрические, экспоненциальную и логарифмическую функции, так и специальные: гипергеометрическую, Бесселя и гамма-функцию. Кроме того, `simplify` способна преобразовывать выражения, содержащие символическое возведение в степень, суммирование и интегрирование. Алгоритм, заложенный в `simple`, пытается получить выражение, которое представляется меньшим числом символов, чем исходное, последовательно применяя все функции упрощения ToolBox.

Функция `subs` позволяет произвести подстановку одного выражения в другое. В общем виде `subs` вызывается с тремя входными аргументами: именем символической функции, переменной, подлежащей замене, и выражением, которое следует подставить вместо переменной. Функция `subs`, в частности, облегчает ввод громоздких символических выражений, имеющих определенную структуру:

```
>> f=sym('(a^2+b^2)/(a^2-b^2)+a^4/b^4');
>> f=subs(f,'a','(exp(x)+exp(-x))');
>> f=subs(f,'b','(sin(x)+cos(x))');
>> pretty(f)
                          2                    2                    4
        (exp(x) + exp(-x))  + (sin(x) + cos(x))    (exp(x) + exp(-x))
        ------------------------------------------ + -------------------
                          2                    2                    4
        (exp(x) + exp(-x))  - (sin(x) + cos(x))    (sin(x) + cos(x))
```

Подстановка вместо переменной ее числового значения приводит к вычислению символической функции от значения аргумента, например:

```
>> f=sym('exp(x^3+2*x^2+x+5)');
>> q=subs(f,'x',1.1)
q =
  1.8977e+004
```

Число можно заменить его символическим представлением и затем найти значение функции с произвольной точностью при помощи `vpa`:

```
>> q=subs(f,'x','1.1')
q =
exp(1.1^3+2*1.1^2+1.1+5)
```

```
>> vpa(q,50)
ans =
18977.322639183802289522844472139578548863931041740
```

Решение задач

Решение систем линейных и нелинейных уравнений, разыскание пределов и производных, нахождение определенных и неопределенных интегралов, поиск аналитических решений дифференциальных уравнений и систем, словом, все основные математические задачи могут быть исследованы при помощи ToolBox Symbolic Math. Разумеется, следует учитывать, что далеко не все задачи имеют аналитическое решение.

Задачи линейной алгебры

Функции ToolBox Symbolic Math выполняют в символическом виде операции с матрицами и векторами. Все задачи, описанные в *главе 6*, могут быть решены в символической форме, причем соответствующие функции имеют те же названия, что и функции для численного решения.

Вычисление определителя в символической форме производится с использованием `det`, например:

```
>> A=sym('[a b c; d e f; g h j]');
>> D=det(A)
D =
j*a*e-a*f*h-j*d*b+d*c*h+g*b*f-g*c*e
```

Функция `inv` предназначена для символического обращения матриц. Найдите обратную к матрице A, определенной выше.

```
>> AI=inv(A);
>> pretty(AI)
                [ -i e + f h      i b - c h       b f - c e]
                [ ----------      ---------     - ---------]
                [     %1            %1              %1    ]
                [                                           ]
                [  -i d + f g      i a - c g     a f - c d ]
                [- ----------    - ---------     --------- ]
                [      %1            %1            %1     ]
                [                                           ]
                [ -d h + e g      a h - b g       a e - b d]
                [ ----------      ---------     - ---------]
                [     %1            %1              %1    ]
              %1 := -i a e + a f h + i d b - d c h - g b f + g c e
```

Обратите внимание на форму отображения результата в командном окне. Знаменатель каждого элемента обратной матрицы *AI* одинаков для всех элементов, поэтому используется подстановка `%1`, соответствующее выражение для `%1` приведено ниже матрицы.

Характеристический полином матрицы, зависящий от переменной `x`, находит функция `poly`.

```
>> pA=poly(A);
>> pretty(pA);
   3      2      2                        2
  x  - i x  - e x  + i x e - x f h - a x  + i a x + a e x - i a e + a f h
        - d b x + i d b - d c h - g b f - g c x + g c e
```

Второй дополнительный аргумент `poly` позволяет указать, от какой переменной должен зависеть характеристический полином матрицы.

Элементы символических матриц могут быть не только символическими переменными, но и выражениями и рациональными числами. В качестве примера найдите собственные числа матрицы Гильберта пятнадцатого порядка, используя сначала символическое представление матрицы, а затем сравните их с результатом численного алгоритма. Определите временные затраты этих двух способов, сгенерировав отчет при помощи профайлера.

См. разд. "Профайлер" главы 16.

Файл-программа для нахождения собственных чисел матрицы Гильберта двумя способами и замера времени приведена в листинге 18.1.

Листинг 18.1. Нахождение собственных чисел матрицы Гильберта

```
profile on -detail builtin
AS=sym(hilb(15));
vs=eig(AS)
A=hilb(15);
v=eig(A)
profile report
```

Решение систем линейных алгебраических уравнений в символическом виде производится при помощи знака обратной косой черты. Следующий пример демонстрирует нахождение линейных базисных функций треугольного конечного элемента в виде, пригодном для помещения в собственную программу. Задача состоит в получении формул для вычисления трех линейных функций `N1(x,y)`, `N2(x,y)`, `N3(x,y)`, каждая из которых равна единице в

одной из вершин треугольника, а в двух остальных обращается в ноль (рис. 18.3). Вершины треугольника имеют координаты `(x1,y1)`, `(x2,y2)`, `(x3,y3)`. Достаточно записать общее выражение для линейной функции двух переменных с коэффициентами `a`, `b` и `c`, а затем разыскать их для каждой функции, решая соответствующую систему линейных уравнений. Листинг 18.2 содержит файл-программу `basis` для нахождения формул, по которым вычисляются требуемые функции. Обратите внимание, что ToolBox Symbolic Math позволяет сгенерировать выражения в виде, пригодном для занесения в собственную программу на Fortran или C. В файл-программе `basis` использована функция `ccode`, разработчикам программ на Fortran следует использовать `fortran` вместо `ccode`.

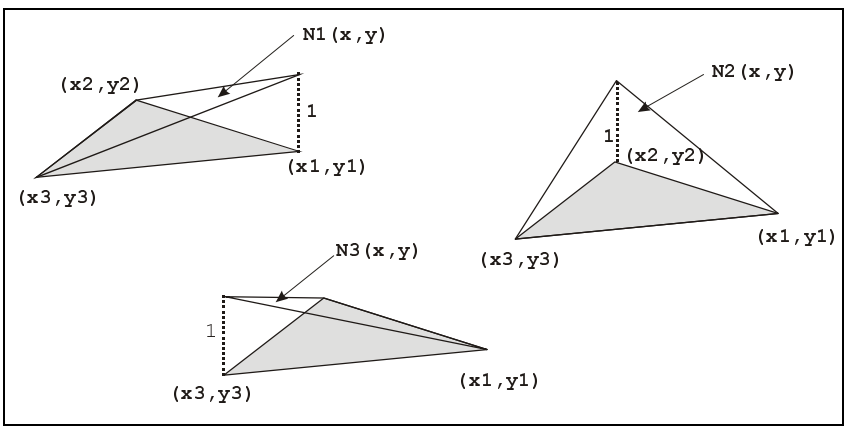


Рис. 18.3. Линейные базисные функции

Листинг 18.2. Файл-программа `basis` для получения базисных линейных функций

```
syms a b c x y
% Задание общего вида линейной функции
N = a + b*x + c*y;
% Определение символической матрицы системы уравнений
M = sym('[1 x1 y1; 1 x2 y2; 1 x3 y3]');
% Нахождение N1
r = sym('[1; 0; 0]'); % правая часть системы уравнений для N1
v = M\r;              % решение системы
% Подстановка выражений для коэффициентов a, b, c
N1 = subs(N,a,v(1));
```

```
N1 = subs(N1,b,v(2));
N1 = subs(N1,c,v(3));
N1 = simplify(N1);    % упрощение N1
pretty(N1)            % вывод N1 в командное окно
ccode(N1)             % генерация Си-кода для N1
% Нахождение N2
r = sym('[0; 1; 0]'); % правая часть системы уравнений для N2
v = M\r;              % решение системы
% Подстановка выражений для коэффициентов a, b, c
N2 = subs(N,a,v(1));
N2 = subs(N2,b,v(2));
N2 = subs(N2,c,v(3));
N2 = simplify(N2);    % упрощение N2
pretty(N2)            % вывод N2 в командное окно
ccode(N2)             % генерация Си-кода для N2
% Нахождение N3
r = sym('[0; 0; 1]'); % правая часть системы уравнений для N3
v = M\r;
% Подстановка выражений для коэффициентов a, b, c
N3 = subs(N,a,v(1));
N3 = subs(N3,b,v(2));
N3 = subs(N3,c,v(3));
N3 = simplify(N3);    % упрощение N3
pretty(N3)            % вывод N3 в командное окно
ccode(N3)             % генерация Си-кода для N3
```

В результате работы файл-программы `basis` в командное окно выводятся выражения для трех базисных функций `N1`, `N2` и `N3` и их представление в формате C. Ниже приведен результат только для первой базисной функции:

```
                  -x3 y2 + x2 y3 - x y3 + x y2 + y x3 - y x2
                 ------------------------------------------
                 x3 y1 + x2 y3 - x2 y1 - x3 y2 - x1 y3 + x1 y2
t0 = (-x3*y2+x2*y3-x*y3+x*y2+y*x3-y*x2)/(x3*y1+x2*y3-x2*y1-x3*y2-x1*y3+x1*y2);
...
```

Соответствующие строки можно легко поместить из командного окна в собственную программу при помощи буфера обмена Windows.

Суммирование и разложение в ряд

Разложение математических функций в ряд Тейлора позволяет проделать функция `taylor`, например:

```
>> f=sym('1/(1+x)');
>> tf=taylor(f);
>> pretty(tf)
                                  2    3    4    5
                        1 - x + x  - x  + x  - x
```

По умолчанию выводится шесть членов ряда разложения в окрестности точки ноль. Число членов разложения можно задать во втором дополнительном параметре `taylor`. Третий параметр указывает, по какой из переменных следует производить разложение, в том случае, когда символическая функция определена от нескольких переменных:

```
>> syms y
>> g=sym('1/(x+y)');
>> tg=taylor(g,7,y);
>> pretty(tg)
                                 2      3      4      5      6
                       y       y      y      y      y      y
                1/x - ---- + ---- - ---- + ---- - ---- + ----
                        2      3      4      5      6      7
                       x      x      x      x      x      x
```

Точка, в окрестности которой проводится разложение, указывается в четвертом входном аргументе `taylor`. В качестве функций, подлежащих разложению в ряд, могут выбираться в том числе и встроенные символические функции, например:

```
>> pretty(taylor('exp(x)',4,x,1/2))
                                                          2
  exp(1/2) + exp(1/2) (x - 1/2) + 1/2 exp(1/2) (x - 1/2)
                                  3
        + 1/6 exp(1/2) (x - 1/2)
```

В состав ToolBox Symbolic Math входит приложение `taylortool` с графическим интерфейсом, предназначенное для наглядной демонстрации разложения в ряд различных функций, в том числе и определенных пользователем. Команда `taylortool` приводит к появлению окна приложения, изображенного на рис. 18.4.

Пользователь может вводить формулы различных функций в строке **f(x)=** в соответствии с правилами MatLab и исследовать приближение функции на произвольном интервале отрезком ряда Тейлора, содержащим различное число членов разложения. Интерфейс приложения `taylortool` достаточно простой и не требует дополнительных пояснений.

Нахождение символических выражений для сумм, в том числе и бесконечных, позволяет осуществить функция `symsym`. Обращение к `symsum` в общем виде предполагает задание четырех аргументов: слагаемого в символической форме, зависящего от индекса, самого индекса и верхнего и нижнего предела суммы. Сумма

$$s = \sum_{k=1}^{\infty} \frac{(-1)^k}{k^2}$$

вычисляется при помощи следующих команд:

```
>> syms k
>> s = symsum('(-1)^k/k^2',k,1,Inf)
s =
-1/12*pi^2
```

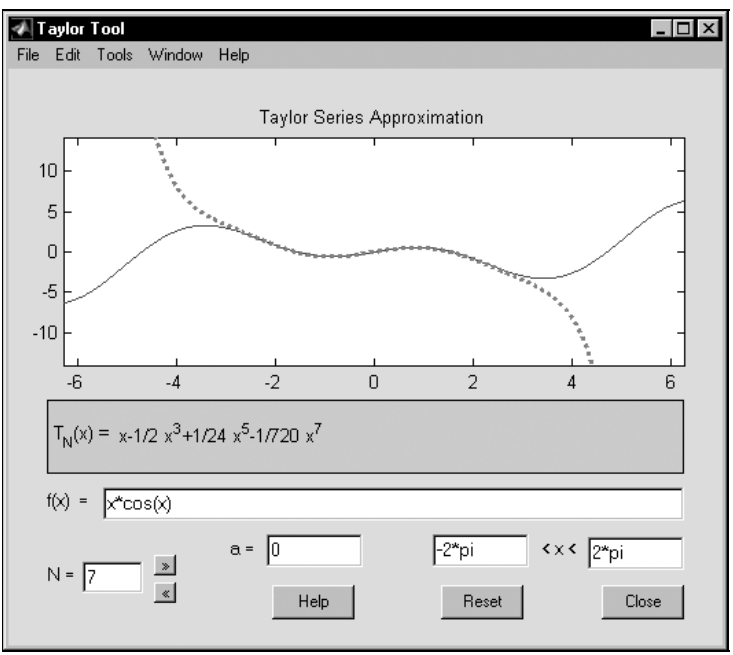


Рис. 18.4. Приложение `taylortool`

Возможно суммирование слагаемых, зависящих не только от индекса, но и от некоторой символической переменной. Если в слагаемые входит факториал, то следует применить к выражению для факториала функцию `sym`. Найдите значение бесконечной суммы, являющейся разложением функции $\sin x$ в ряд

$$s = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Используйте команды:

```
>> syms k x
>> s = symsum((-1)^(k)*x^(2*k+1)/sym('(2*k+1)!'),k,0,Inf)
s =
sin(x)
```

Пределы, дифференцирование и интегрирование

Ряд функций ToolBox Symbolic Math предназначен для решения задач дифференциального и интегрального исчисления. Функция `limit` находит предел функции в некоторой точке, включая и плюс или минус бесконечность. Первым входным аргументом `limit` является символическое выражение, вторым — переменная, а третьим — точка, в которой разыскивается предел. Пусть, например, требуется вычислить

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^{ax}.$$

Для получения ответа определите `a` и `x`, как символические переменные, и используйте `Inf` в качестве точки предела:

```
>> syms a x
>> limit((1+1/x)^(x*a),x,Inf)
ans =
exp(a)
```

Функция `limit` позволяет находить односторонние пределы, для нахождения предела справа следует указать четвертый дополнительный аргумент `'right'`, а слева — `'left'`. Найдите решение следующих двух задач

$$\lim_{x\to 0+}(10+x)^{1/x}; \qquad \lim_{x\to 0-}(10+x)^{1/x}.$$

Очевидно, что следует выполнить следующие команды:

```
>> syms b x
>> limit((10+x)^(1/x),x,0,'left')
ans =
0
>> limit((10+x)^(1/x),x,0,'right')
ans =
inf
```

Обратите внимание, что обычный предел в точке ноль в предыдущем примере не существует:

```
>> limit((10+x)^(1/x),x,0)
ans =
NaN
```

Определение производной через предел позволяет применять `limit` для дифференцирования функций. Найдите первую производную функции $\operatorname{arctg} x$, используя равенство

$$\frac{d}{dx}\operatorname{arctg} x = \lim_{h\to 0}\frac{\operatorname{arctg}(x+h)-\operatorname{arctg} x}{h}.$$

```
>> syms h x
>> L=limit((atan(x+h)-atan(x))/h,h,0);
>> pretty(L)
                                    1
                                  ------
                                       2
                                  1 + x
```

Вычисление производных любого порядка проще производить при помощи функции `diff`. Символическая запись функции указывается в первом входном аргументе, переменная, по которой производится дифференцирование, — во втором, а порядок производной — в третьем. Применение `diff` для вычисления производной в предыдущем примере, разумеется, приводит к эквивалентному результату:

```
>> P=diff('atan(x)',x,1);
>> pretty(P)
                                    1
                                  ------
                                       2
                                  1 + x
```

Напишите файл-функцию `tangent` для исследования скорости роста функций. Входными аргументами `tangent` являются строка с символическим представлением функции одной переменной `x` и числовое значение абсциссы точки, в которой следует провести касательную. Файл-функция `tangent` выводит в одно графическое окно графики функции и касательной к ней в заданной точке. К примеру, вызов

```
>> tangent('sin(x)*x^2',2)
```

должен приводить к графикам, изображенным на рис. 18.5.

Алгоритм файл-функции включает:

1. Определение символической функции по строке при помощи `sym`.
2. Нахождение производной.

3. Формирование символического выражения для касательной, и подстановки в него значения производной, абсциссы и ординаты точки, в которой проводится касательная.

Для построения касательной линии используйте `ezplot`. Отображение графика исследуемой функции жирной линией выполните при помощи `plot`, для чего предварительно сгенерируйте вектор со значениями аргумента и получите вектор соответствующих численных значений символической функции. В качестве границ отрезка, на котором выводятся графики функции и касательной к ней, выберите точки, отстоящие на единицу вправо и влево от заданной. Файл-функция `tangent` не требует выходных аргументов. Обратитесь к листингу 18.3 за дополнительной информацией в случае возникновения затруднений при программировании.

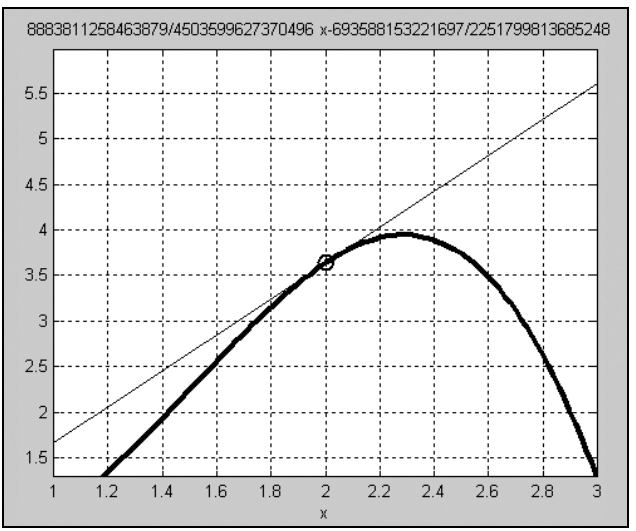


Рис. 18.5. Графики функции и касательной

Листинг 18.3. Файл-функция `tangent`

```
function tangent(funstr, X0)
% Файл-функция для построения касательной
% к графику функции funstr в точке X0
% funstr – строка с символическим выражением функции
% Использование:
%     tangent('exp(x)',0)

% Задание символической функции
f=sym(funstr);
```

```
% Вычисление функции в точке X0
Y0=subs(f,'x',X0)
% Определение интервала для построения графиков
% функции и касательной
A=X0-1;
B=X0+1;
% Вывод графика функции жирной линией
% Генерация вектора значений аргумента
X=[A:(B-A)/100:B];
% Подстановка вектора в символическое представление функции
% и образование вектора значений функции
F=subs(f,'x',X);
% Вывод графика и установка толщины линии
Hline = plot(X,F);
set(Hline, 'LineWidth', 2)
% Нахождение символического выражения для первой производной
syms x
k=diff(f,x,1);
% Вычисление коэффициента касательной
K=subs(k,'x',X0)
% Символическое задание уравнения касательной
yt=sym('y0+k*(x-x0)');
% Подстановка коэффициента, абсциссы и ординаты
% в уравнение касательной
yt=subs(yt,'k',K);
yt=subs(yt,'x0',X0);
yt=subs(yt,'y0',Y0);
% Вывод графика касательной на те же оси, где находится
% график функции
hold on
ezplot(yt,[A B])
% Точка касания отмечается маркером-кружком
plot(X0,Y0,'o')
grid on
hold off
```

Символическое интегрирование является значительно более сложной задачей, чем дифференцирование. ToolBox Symbolic Math позволяет работать как с неопределенными интегралами, так и с определенными. Неопределенные интегралы от символических функций вычисляются при помощи

`int`, в качестве входных аргументов указываются символическая функция и переменная, по которой происходит интегрирование, например:

```
>> syms x
>> f=sym('x^3*exp(x)');
>> I=int(f,x)
I =
x^3*exp(x)-3*x^2*exp(x)+6*x*exp(x)-6*exp(x)
>> pretty(I)
                 3                 2
                x  exp(x) - 3 x  exp(x) + 6 x exp(x) - 6 exp(x)
```

Произведите проверку, продифференцировав полученную первообразную:

```
>> diff(I,x,1)
ans =
x^3*exp(x)
```

Разумеется, функция `int` не позволяет получить неопределенный интеграл от произвольной функции. В некоторых случаях `int` возвращает выражение для первообразной через специальные функции, например, получите значение интеграла

$$\int e^{\sin^2 x} \cos x \, dx .$$

Определите подынтегральную функцию и вызовите `int`:

```
>> syms x
>> f=sym('exp(sin(x)^2)*cos(x)');
>> I=int(f,x);
>> pretty(I)
                                    1/2
                        - 1/2 i pi     erf(i sin(x))
```

Ответ содержит так называемую функцию ошибки, которая определяется интегралом с переменным верхним пределом:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt .$$

Кроме того, в полученное выражение входит комплексная единица, хотя подынтегральная функция вещественна. Требуются дополнительные преобразования для достижения окончательного результата, функции `simple` и `simplify` не смогут упростить данное выражение.

Для нахождения определенного интеграла в символическом виде следует задать нижний и верхний пределы интегрирования, соответственно, в третьем и четвертом аргументах `int`:

```
>> syms x a b
>> f=sym('(x^3+1)/(x-1)');
>> I=int(f,x,a,b);
>> pretty(I)
       3       2                            3       2
   1/3 b  + 1/2 b  + b + 2 log(b - 1) - 1/3 a  - 1/2 a  - a - 2 log(a - 1)
```

Двойные интегралы вычисляются повторным применением функции `int`. Найдите, например, интеграл

$$\int_c^d \int_a^b y \sin x \, dx dy .$$

Определите символические переменные `a`, `b`, `c`, `d`, `x`, `y`, подынтегральную функцию `f` от `x` и `y` и проинтегрируйте сначала по `x`, а затем по `y`:

```
>> syms a b c d x y
>> f=sym('y*sin(x)');
>> Ix=int(f,x,a,b)
Ix =
-y*cos(b)+y*cos(a)
>> Iy=int(Ix,y,c,d)
Iy =
1/2*(-cos(b)+cos(a))*(d^2-c^2)
>> pretty(Iy)
                                                 2     2
                        1/2 (-cos(b) + cos(a)) (d  - c )
```

Аналогичным образом решаются задачи, связанные с вычислением любых кратных интегралов в символическом виде.

Решение уравнений и систем

Алгебраические уравнения до четвертого порядка включительно решаются точно, функция `solve` выводит ответ в степенях рациональных чисел. Результат решения уравнения третьего порядка, к примеру, отображается в командном окне с использованием подстановок:

```
>> syms x
>> f=sym('x^3-x^2-5*x+1');
>> r=solve(f,x);
>> pretty(r)
```

```
[                1/3 %2 + 16/3 %1 + 1/3                  ]
[                                                        ]
[                                  1/2                   ]
[- 1/6 %2 - 8/3 %1 + 1/3 + 1/2 i 3    (1/3 %2 - 16/3 %1)]
[                                                        ]
[                                  1/2                   ]
[- 1/6 %2 - 8/3 %1 + 1/3 - 1/2 i 3    (1/3 %2 - 16/3 %1)]

                                 1
                %1 := --------------------
                                    1/2 1/3
                      (10 + 6 i 111   )

                                    1/2 1/3
                %2 := (10 + 6 i 111   )
```

Корни уравнения записываются в символический массив `r`. Проверьте, что корни найдены верно, вычислите значение `f` от первого корня. Используйте функцию `siplify` для преобразования результата:

```
>> simplify(subs(f,'x',r(1)))
ans =
0
```

Допустимо использование символических переменных в выражении для левой части уравнения. Функция `solve` позволяет, например, вывести в окно рабочей среды формулы для нахождения корней алгебраического уравнения четвертой степени общего вида.

```
>> syms x
>> f=sym('a*x^4+b*x^3+c*x^2+d*x+c');
>> pretty(solve(f,x))
```

Результат занимает достаточно много места, несмотря на использование подстановок, и в книге не приводится.

Алгебраические уравнения высших порядков и трансцендентные уравнения, как правило, не могут быть разрешены точно. В этом случае выводятся приближенные значения корней.

Решение системы нелинейных уравнений также находится при помощи `solve`. Входными аргументами `solve` являются в данном случае левые части уравнений и переменные, по которым требуется разрешить систему. Например, для системы из двух уравнений с правыми частями, которые определены в символических функциях `f1` и `f2`, зависящих от `x1` и `x2`, вызов `solve` выглядит так: `s=solve(f1,f2,x1,x2)`. Результатом является структура `s` с полями `x1` и `x2`, каждое из которых содержит символическое представление решения.

Работе со структурами данных посвящен разд. "Массивы структур и массивы ячеек" главы 10.

Найдите решение системы уравнений, приведенной ниже, и сравните ответ с полученным в *разд. "Решение нелинейных уравнений" главы 17.*

$$\begin{cases} x_1(2-x_2) = \cos x_1 \cdot e^{x_2}; \\ 2 + x_1 - x_2 = \cos x_1 + e^{x_2}. \end{cases}$$

Определите две символические функции и переменные и вызовите `solve` с одним выходным аргументом:

```
>> syms x1 x2
>> f1=sym('x1*(2-x2)-cos(x1)*exp(x2)');
>> f2=sym('2+x1-x2-cos(x1)-exp(x2)');
>> s=solve(f1,f2,x1,x2);
```

Выведите в окно рабочей среды значения полей структуры `s`, обратите внимание, что поля структуры содержат символические векторы:

```
>> s.x1
ans =
[               .7390851332]
[ .6717920395-1.339070269*i]
>> s.x2
ans =
[             -lambertw(exp(2))+2]
[ log(.6717920395-1.339070269*i)]
```

Функция `solve` нашла две пары решений, вещественные `s.x1(1)`, `s.x2(1)` и комплексные `s.x1(2)`, `s.x2(2)`. Первый компонент вектора `s.x2` выражен через функцию Ламберта $w = L(x)$, которая определена как зависимость решения x трансцендентного уравнения $we^w = x$ от параметра w, входящего в уравнение. Для получения решения системы нелинейных уравнений с заданной точностью следует применить функцию `vpa` к полям структуры `s`:

```
>> vpa(s.x1,30)
ans =
[                                  .739085133215160641655312087674]
[ .671792039467180096393251253120-1.339070269494918131470473583 23*i]
>> vpa(s.x2,30)
ans =
[                                   .442854401002388583141328000000]
[ .404222171370585369483393079830-1.105801285688515552037575835 84*i]
```

Вещественное решение совпадает с решением, полученным в *главе 17* при помощи `fsolve`. Кроме того, функция `solve` нашла комплексные корни, решая систему нелинейных уравнений в символическом виде.

Интерфейс функции `solve` допускает использование символических переменных в качестве выходных аргументов, вместо структуры. Эквивалентное обращение к `solve` в предыдущем примере имеет вид:

```
>> [x1, x2]=solve(f1,f2,x1,x2);
```

Задание левых частей уравнений символическими функциями не является обязательным. Входными аргументами `solve` могут быть строки с уравнениями, заключенные в апострофы, причем необязательно переносить все слагаемые в левую часть. Независимые переменные можно так же не указывать, например обращение

```
>> s=solve('x1*(2-x2)=cos(x1)*exp(x2)','2+x1-x2=cos(x1)+exp(x2)')
```

аналогично приведенным выше.

Решение дифференциальных уравнений и систем

ToolBox Symbolic Math позволяет разыскать решение дифференциальных уравнений и систем в аналитическом виде. Возможно нахождение как общего решения, зависящего от констант, так и решения, удовлетворяющего поставленным граничным условиям. Решение дифференциальных уравнений и систем производится при помощи функции `dsolve`, входными аргументами которой являются строки с уравнением, граничными условиями, при их наличии, и независимой переменной. Если независимая переменная не указана, то по умолчанию используется `t`. Производные в строках задаются так: `Dy`, `D2y`,... Граничные условия могут содержать производные от неизвестной функции. В качестве примера найдите аналитическое решение уравнения Риккати (для некоторого частного случая значений коэффициентов), удовлетворяющее граничному условию

$$y' + y^2 = x^{-2}, \qquad y(0.5) = -1.$$

Отобразите полученное решение на отрезке [0.5, 7] и сравните его с приближенным решением, полученным при помощи солвера `ode45`.

Использование солверов описано в разд. "Решение дифференциальных уравнений" главы 6.

Вызов функции `dsolve` в рассматриваемом примере выглядит следующим образом:

```
>> y=dsolve('Dy+y^2=x^(-2)', 'y(0.5)=-1','x');
```

Выведите в командное окно полученное решение, используя `pretty`:

```
>> pretty(y)
                                                          1/2
                 1/2                1/2                  5    - 1
          1 - 5    tanh(- 1/2 5    log(x) + 1/2 log(--------))
```

$$1/2\ \frac{5^{1/2}+1}{x}$$

Теперь решите уравнение Риккати численно, создайте файл-функцию `rikkaty` (листинг 18.4), вычисляющую правую часть уравнения $y' = -y^2 + x^{-2}$, и задайте ее имя в качестве входного аргумента солвера `ode45` вместе с отрезком и граничным условием:

```
>> [X, Y] = ode45('rikkaty', [0.5 7], -1);
```

Листинг 18.4. Файл-функция, вычисляющая правую часть уравнения Риккати

```
function F = rikkaty(x, y)
F = [1/x^2-y(1)^2];
```

Отобразите полученные приближенное и точное решения на одних осях:

```
>> ezplot(y)
>> hold on
>> plot(X,Y,'o')
>> grid on
```

Проверка (рис. 18.6) показывает, что аналитическое решение найдено верно.

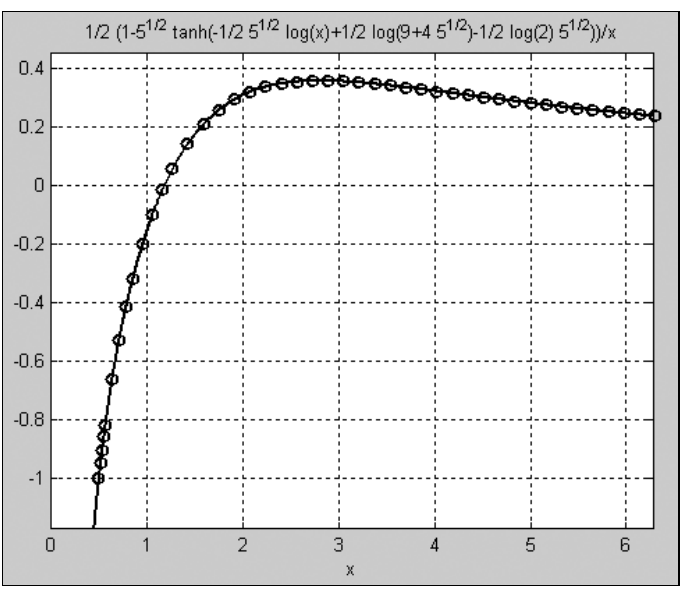


Рис. 18.6. Точное и приближенное решения уравнения Риккати

Поиск аналитического решения системы дифференциальных уравнений требует указания во входных аргументах `dsolve` строк, содержащих дифференциальные уравнения с тем же обозначением `D` для производных, и строк с граничными условиями при их наличии. Найдите общее решение системы из двух дифференциальных уравнений первого порядка

$$\begin{cases} f' = e^{-g}; \\ g' = e^{-f}. \end{cases}$$

В данном случае `dsolve` можно вызвать с двумя входными аргументами, по умолчанию считается, что независимая переменная `t` (если бы в уравнения явно входила переменная `x`, то ее следовало бы указать во входных аргументах так же, как и в предыдущем примере). Используйте в качестве выходного аргумента вектор значений, каждый элемент которого станет символической переменной:

```
>> [f,g]=dsolve('Df=exp(-g)', 'Dg=exp(-f)');
```

Выведите полученное решение в командное окно:

```
>> pretty(f)
                              exp(t C1 + C2 C1) - 1
                          log(----------------------)
                                        C1
>> pretty(g)
                               exp(t C1 + C2 C1) C1
                          -log(----------------------)
                               exp(t C1 + C2 C1) - 1
```

Произведите проверку, подставив найденные символические функции в дифференциальные уравнения системы:

```
>> syms t
>> u1=diff(f,t)-exp(-g)
u1 =
0
>> u2=diff(g,t)-exp(-f)
u2 =
-(C1^2*exp(t*C1+C2*C1)/(exp(t*C1+C2*C1)-1)-
exp(t*C1+C2*C1)^2/(exp(t*C1+C2*C1)-
1)^2*C1^2)/exp(t*C1+C2*C1)*(exp(t*C1+C2*C1)-1)/C1-1/(exp(t*C1+C2*C1)-
1)*C1
>> simplify(u2)
ans =
0
```

Аналитическое решение найдено верно. Найдите решение системы, удовлетворяющее граничным условиям $f'(5)=1$, $g'(5)=3$. Очевидно, что следует обратиться к `dsolve` со следующими аргументами:

```
>> [f,g]=dsolve('Df=exp(-g)', 'Dg=exp(-f)', 'Df(5)=1', 'Dg(5)=3');
```

Допустима запись уравнений в одну строку через запятую и граничных условий также в одну строку:

```
>> [f,g]=dsolve('Df=exp(-g), Dg=exp(-f)', 'Df(5)=1, Dg(5)=3');
```

Дифференциальные уравнения высших порядков решаются аналогично. Рассмотрим один пример, демонстрирующий использование функций Maple. Требуется найти аналитическое решение дифференциального уравнения

$$(1-x^2)\frac{d^2y}{dx^2}-2x\frac{dy}{dx}+n(n+1)y=0.$$

Обращение к `dsolve` с двумя входными аргументами — уравнением и независимой переменной — приводит к решению, выраженному через полином Лежандра и функцию Лежандра второго рода:

```
>> y=dsolve('(1-x^2)*D2y-2*x*Dy+n*(n-1)*y=0','x')
y =
C1*LegendreP(n-1,x)+C2*LegendreQ(n-1,x)
```

Попытка получить сведения MatLab о функциях `LegendreP` и `LegandreQ` при помощи встроенной справки приводит к сообщению об отсутствии одноименных M-файлов. Функции ToolBox Symbolic Math на самом деле реализуют интерфейс между средой MatLab и библиотекой основных функций Maple. Функции `LegendreP` и `LegandreQ` не определены в MatLab. Для доступа к информации о функциях Maple предназначена команда `mhelp`, использование которой схоже с командой `help`, например, указание в качестве параметра имени функции

```
>> mhelp LegendreP
```

выводит определение функции `LegendreP` (и связанной с ней `LegendreQ`), варианты вызова и подробное описание с примерами использования:

```
LegendreP, LegendreQ - The Legendre functions and associated Legendre
functions
of the first and second kinds
Calling Sequence:
     LegendreP(v, x)
     LegendreQ(v, x)
     LegendreP(v, u, x)
     LegendreQ(v, u, x)
...
```

Для получения списка специальных математических функций Maple, доступных из MatLab, служит команда `mfunlist`. Функция MatLab `mfun` позволяет вычислить значение функций Maple, к примеру

```
 >> mfun('LegendreP', 5, 0.7)
ans =
   -0.3652
```

Вышеописанное обращение является одной из предоставляемых ToolBox Symbolic Math возможностей пользователю, который желает использовать ресурсы вычислительного ядра Maple при работе в MatLab с символическими выражениями. Если вы имеете опыт работы в Maple, то существенную пользу принесет ToolBox Extended Symbolic Math. Данное расширение основного ToolBox Symbolic Math позволяет оперировать как со всеми функциями Maple (исключая графические), так и создавать и выполнять приложения, написанные на языке программирования, встроенном в Maple.



Часть V

Дополнительные возможности MatLab

Глава 19. Связь MatLab и MS Office

Глава 20. Редактирование приложений с GUI версии 5.3 в версии 6.x

Глава 21. Повышение производительности приложений MatLab

Глава 19



Связь MatLab и MS Office

Среда MatLab допускает достаточно простое интегрирование с MS Word и MS Excel. Редактор Word используется для написания интерактивных документов, так называемых M-книг, которые позволяют наглядно оформить расчеты на MatLab в документе Word. Обработка данных существенно облегчается при сочетании работы в MatLab и Excel. Надстройка Excel Link, входящая в поставку MatLab, снабжает пользователя Excel доступом ко всем функциям MatLab, которые значительно расширяют возможности электронных таблиц.

M-книги

M-книги могут содержать как текст, таблицы, рисунки и другие элементы оформления документа MS Word, так и *команды MatLab и результаты их выполнения*. Причем, набираемые команды активизируются прямо из документа (M-книги) и результат помещается также в документ. Пользователь имеет возможность работать со средой MatLab, сопровождая свои действия текстовыми комментариями, набором формул в редакторе Microsoft Equation, словом, используя все средства Word. Получающиеся интерактивные документы могут, например, использоваться в качестве учебных пособий для изучения различных разделов математики, физики и других дисциплин, или при составлении отчетов о решении задач в MatLab.

Настройка MatLab и создание M-книги

Перед началом работы над M-книгой необходимо произвести некоторые настройки MatLab на конфигурацию и версию MS-Word, установленного на компьютере. Действия, описанные ниже, производятся *только один раз* при создании первой M-книги. Продолжение работы над существующими M-книгами и разработка новых не требуют повторных настроек. Разумеется, при переустановке Word или MatLab придется произвести процесс настройки сначала.

Запустите MatLab и наберите в командном окне `notebook -setup`. Запрашивается номер версии Word, установленной на вашем компьютере. Выберите нужную цифру и следуйте появляющимся инструкциям. Сначала выводится сообщение о том, что после нажатия на любую клавишу появится диалого-

вое окно, в котором следует указать путь к файлу winword.exe (он обычно находится в подкаталоге Office каталога с MS Office). В следующем диалоговом окне задайте путь к шаблону normal.dot (шаблоны Word расположены в подкаталоге Шаблоны или Templates, в зависимости от локализации версии). Настройка MatLab завершена. В каталоге Шаблоны (или Templates) появился файл m-book.dot, являющийся шаблоном для создания M-книг.

Имеется несколько способов, позволяющих начать работу над новой M-книгой. Команда `notebook` приводит к появлению в Word нового файла, основанного на шаблоне m-book.dot. Если Word не был открыт, то он запускается после выполнения данной команды. Аналогичный результат получается при создании нового файла при помощи пункта **Создать** меню **Файл** MS Word. В диалоговом окне **Создание документа** на вкладке **Общие** следует выбрать шаблон m-book.dot, установить переключатель **Создать документ** и нажать кнопку **OK**.

Обратите внимание, что в Word создалось меню **Notebook**, предназначенное для управления и редактирования интерактивной M-книги. В меню **Файл** добавился пункт **New M-book**, кроме того, всплывающее меню приобрело дополнительные пункты. Список стилей также пополнился стилями, определенными в m-book.dot: AutoInit, Calc, Error, Input, NoGraph, Output. По умолчанию используется стиль Обычный.

Замечание

Вне зависимости от локализации версии Word, все элементы, добавляемые при подключении шаблона m-book.dot, имеют англоязычные названия.

Наберите в документе какую-нибудь команду MatLab, к примеру

```
f=sin(3/4*pi)*exp(-1)
```

Поместите курсор в набранную строку и выберите в меню **Notebook** пункт **Define Input Cell**. Обратите внимание, что стиль набранного текста изменился на Input, сам текст заключился в квадратные скобки, а цвет шрифта изменился на зеленый:

```
f=sin(3/4*pi)*exp(-1)
```

Образовалась так называемая *ячейка ввода* (Input Cell). Для выполнения команды MatLab, содержащейся в ячейке ввода, следует убедиться, что данная ячейка является текущей, т. е. в ней находится курсор, и выбрать в меню **Notebook** пункт **Evaluate Cell**. Ниже ячейки ввода в документе появляется *ячейка вывода* с результатом в привычном для пользователя MatLab виде:

```
f =

    0.2601
```

Абзацы ячейки вывода имеют стиль Output, начало и конец ячейки ограничены квадратными скобками, а цвет шрифта синий. Пользователь может

переопределить стили шаблона m-book.dot так же, как и любого другого стиля, выбрав в меню **Формат** пункт **Стиль** и произведя нужные установки в появившемся диалоговом окне.

Группировка ячеек

Решение задачи линейной оптимизации о составлении рациона, описанное в *главе 17*, может быть наглядно продемонстрировано в М-книге. Сначала следует привести условие задачи и вспомогательную таблицу, а затем набрать операторы файл-программы `ration` (см. листинг 17.1) в тексте документа. Каждый оператор следует заключить в ячейку ввода, выбирая в меню **Notebook** пункт **Define Input Cell**, либо используя комбинацию клавиш <Alt>+<D>. Содержимое М-книги должно соответствовать листингу 19.1.

Листинг 19.1. Задание ячеек ввода

```
A = [4  6  15
     2  2  0
     5  3  4
     7  3  12];
A = -A;
b = [250; 60; 100; 220];
b = -b;
f = [44;  35;  100];
lb =[0; 0; 0];
x = linprog(f, A, b, [], [], lb, [])
```

Несколько команд, выполняемых последовательно, лучше заключить в группу ячеек ввода (Cell Group). Выделите ячейки, подлежащие объединению в группу (в данном случае это все ячейки листинга 19.1) и в меню **Notebook** выберите пункт **Group Cells**. Команды образовавшейся группы выполняются из пункта **Evaluate Cell** меню **Notebook** или **Evaluate Cells** всплывающего меню. Сочетание клавиш <Alt>+<Enter> также приводит к активизации команд группы. В результате содержимое М-книги дополняется ячейкой вывода с решением задачи линейного программирования (листинг 19.2).

Листинг 19.2. Группа ввода ячеек и результат ее выполнения

```
A = [4  6  15
     2  2  0
     5  3  4
     7  3  12];
A = -A;
```

```
b = [250; 60; 100; 220];
b = -b;
f = [44;  35;  100];
lb =[0; 0; 0];
x = linprog(f, A, b, [], [], lb, [])

Optimization terminated successfully.
x =
   13.2143
   16.7857
    6.4286
```

Создание группы ячеек имеет ряд особенностей. Группа не должна содержать текст или ячейки вывода. Текст, разделяющий ячейки перед их объединением, помещается после группы. Ячейки вывода пропадают, но зато соответствующие им ячейки ввода добавляются в группу.

Продолжите работу над M-книгой, создайте группу ячеек ввода с командами, обеспечивающей отображение круговой диаграммы полученного решения и выполните ее (листинг 19.3). Соответствующая ячейка вывода содержит область графического окна с диаграммой (рис. 19.1). Важно понимать, что рисунок в ячейке вывода, заключенной в большие квадратные скобки, является объектом Word. Данное обстоятельство позволяет обращаться с ним, как с обычным рисунком, внедряемым в документ.

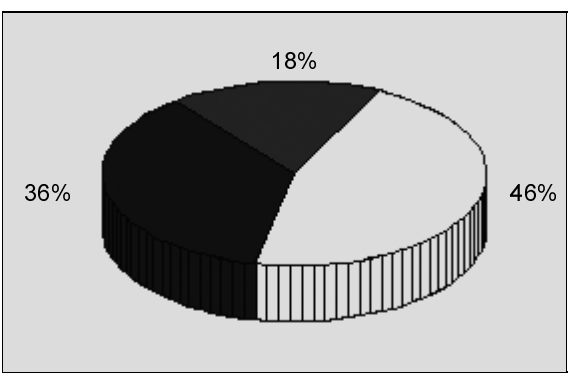


Рис. 19.1. Рисунок в ячейке вывода листинга 19.3

Листинг 19.3. Использование графических команд MatLab

```
pie3(x)
HT=title('Пропорции продуктов в смеси');
```

```
set(HT, 'FontSize', 16)
[здесь размещается рис. 19.1]
```

М-книга может содержать сколь угодно много групп ячеек ввода и отвечающих им ячеек вывода. Ячейки вывода можно перемещать в произвольное место документа, выделив их и перетащив при помощи мыши как обычные абзацы документа Word. Повторное выполнение команд, находящихся в соответствующих ячейках ввода, не нарушит расположение ячеек.

Работа с М-книгой большого объема становится проще, если предусмотреть разбиение ее на разделы (Calc Zone). Выделенный фрагмент книги выносится в отдельный раздел выбором пункта **Define Calc Zone** меню **Notebook**. Выполнение команд всех ячеек или групп ячеек ввода раздела производится при помощи пункта **Evaluate Calc Zone**, а сразу всей М-книги — **Evaluate M-Book**. Ошибка в командах ячеек ввода, возникающая в процессе выполнения всей М-книги, не приводит к останову вычислений. Для остановки по ошибке следует установить флаг **Stop evaluating on error** в диалоговом окне **Notebook Options**, появляющемся при выборе пункта **Notebook Options** в меню **Notebook**.

Переменные различных разделов являются общими, к примеру, если ячейке одного раздела переменной `x` было присвоено некоторое значение, то `x` можно использовать и в остальных разделах. Переменные М-книги являются глобальными. Более того, если в редакторе Word открыто несколько М-книг, то их переменные определены в одной рабочей среде.

Открытие М-книги в Word не приводит к автоматическому выполнению содержимого ячеек ввода. Часто требуется инициализировать некоторые переменные без вмешательства пользователя. Команды ячеек, имеющих стиль AutoInit, запускаются сразу после открытия М-книги. Полезно включить в первую такую ячейку `clear` для очистки рабочей среды. Пункт **Define Autoinit Cell** меню **Notebook** предназначен для определения стиля AutoInit ячейки ввода.

Содержимое ячейки или группы можно выполнить циклически, для чего следует выделить нужные ячейки или сделать текущей группу и выбрать в меню **Notebook** пункт **Evaluate Loop**, или нажать <Alt>+<L>. Появившееся диалоговое окно **Evaluate Loop** позволяет установить число повторов в поле **Stop After** и выбрать скорость кнопками **Slower** и **Faster**.

Управление М-книгой

Разработчик М-книги имеет возможность изменять вид ячеек вывода как с текстовой, так и с графической информацией. Меню **Notebook** содержит пункт **Notebook Options**, выбор которого приводит к появлению одноименного диалогового окна, изображенного на рис. 19.2. Панель **Numeric Format**

содержит раскрывающийся список для выбора формата и переключатели **Loose** и **Compact** для добавления промежуточных пустых строк при отображении числовых значений в ячейках вывода.

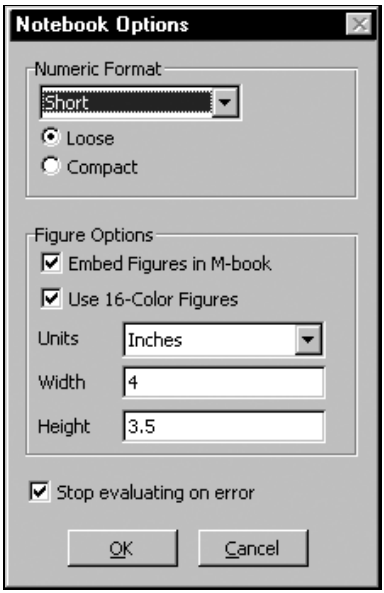


Рис. 19.2. Диалоговое окно **Notebook Options**

Панель **Figure Options** предназначена для управления видом графических результатов, помещаемых в ячейки вывода. Установленный флаг **Embed Figures in M-book** обеспечивает размещение графиков в ячейках вывода, а сброшенный — приводит к визуализации результатов в отдельных графических окнах. Размер и единицы измерения графиков, помещаемых в ячейки ввода, определяются в строках **Width** и **Height** и раскрывающемся списке **Units**.

Ячейки вывода с окончательными результатами преобразовываются в текст выбором пункта **Undefine Cells** меню **Notebook**. Пользователь может по своему усмотрению изменить стили шаблона m-book.dot средствами Word точно так же, как и любого другого шаблона. Переопределение стилей производится в диалоговом окне **Стиль**, которое появляется при переходе к пункту **Стиль** меню **Формат**.

Квадратные скобки, ограничивающие ячейки и группы ячеек, пропадают при выборе пункта **Hide Cell Markers** меню **Notebook**. Пункт **Show Cell Markers** служит для отображения скобок в документе. При печати М-книги скобки не выводятся.

Excel Link

Интегрирование MatLab и Excel позволяет пользователю Excel обращаться к многочисленным функциям MatLab для обработки данных, различных вычислений и визуализации результата. Надстройка excllink.xla реализует данное расширение возможностей Excel. Для связи MatLab и Excel определены специальные функции.

Конфигурирование Excel

Перед тем как настраивать Excel на совместную работу с MatLab, следует убедиться, что Excel Link входит в установленную версию MatLab. В подкаталоге exlink основного каталога MatLab или подкаталога toolbox должен находиться файл с надстройкой excllink.xla. Запустите Excel и в меню **Сервис** выберите пункт **Надстройки**. Открывается диалоговое окно (рис. 19.3), содержащее информацию о доступных в данный момент надстройках.

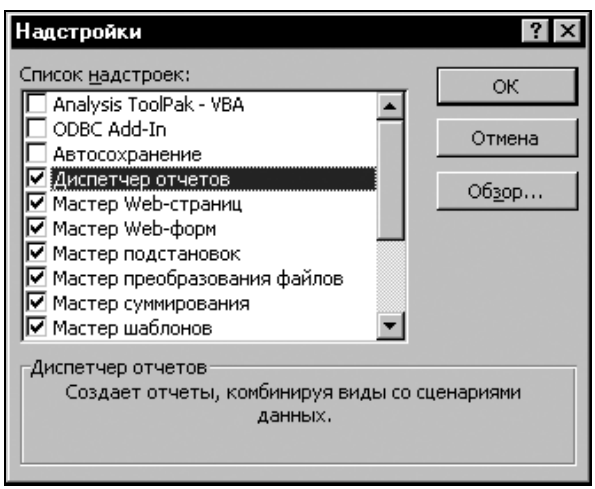


Рис. 19.3. Диалоговое окно **Надстройки**

Используя кнопку **Обзор**, укажите путь к файлу excllink.xla. В списке надстроек диалогового окна появилась строка **Excel Link 1.1.2 for use with MatLab** с установленным флагом. Нажмите **OK**, требуемая надстройка добавлена в Excel. Обратите внимание, что в Excel присутствует панель инструментов **Excel Link**, содержащая три кнопки: **putmatrix**, **getmatrix**, **evalstring**. Данные кнопки реализуют основные действия, требуемые для осуществления взаимосвязи между Excel и MatLab — обмен матричными данными, и выполнение команд MatLab из среды Excel.

При повторных запусках Excel надстройка excllink.xla подключается автоматически. Избежать подключения надстройки можно сбросом соответствующего флага в диалоговом окне **Надстройки**.

Согласованная работа Excel и MatLab требует еще нескольких установок, которые приняты в Excel по умолчанию (но могут быть изменены). В меню **Сервис** перейдите к пункту **Параметры**, открывается диалоговое окно **Параметры**. Выберите вкладку **Общие** и убедитесь, что флаг **Стиль ссылок R1C1** выключен, т. е. ячейки нумеруются `A1`, `A2` и т. д. На вкладке **Правка** должен быть установлен флаг **Переход к другой ячейке после ввода**.

Обмен данными между MatLab и Excel

Запустите Excel, проверьте, что проделаны все необходимые настройки так, как описано в предыдущем разделе (MatLab должен быть закрыт). Введите в ячейки с `A1` по `C3` матрицу (рис. 19.4), для отделения десятичных знаков используйте запятую в соответствии с требованиями Excel.

	A	B	C
1	5,5	1,6	-0,8
2	2,3	6,1	0,2
3	0,1	0,4	3,9

Рис. 19.4. Ввод матрицы в ячейки

Выделите на листе данные ячейки и нажмите кнопку **putmatrix**, появляется окно Excel с предупреждением о том, что MatLab не запущен. Нажмите **OK**, дождитесь открытия MatLab.

Замечание

Открывается только командное окно MatLab вне зависимости от того, какая установлена версия (5.3 или 6.*x*).

Появляется диалоговое окно Excel со строкой ввода, предназначенной для определения имени переменной рабочей среды MatLab, в которую следует экспортировать данные из выделенных ячеек Excel. Введите, к примеру, `M` и закройте окно при помощи кнопки **OK**. Перейдите к командному окну MatLab и убедитесь, что в рабочей среде создалась переменная `M`, содержащая массив три на три:

```
>> M
M =
    5.5000    1.6000   -0.8000
    2.3000    6.1000    0.2000
    0.1000    0.4000    3.9000
```

Проделайте некоторые операции в MatLab с матрицей *M*, например, обратите ее:

```
>> IM = inv(M);
```

Замечание

Вызов `inv` для обращения матрицы, как и любой другой команды MatLab можно осуществить прямо из Excel. Нажатие на кнопку **evalstring**, расположенную на панели **Excel Link**, приводит к появлению диалогового окна, в строке ввода которого следует набрать команду MatLab `IM=inv(M)`. Результат аналогичен полученному при выполнении команды в среде MatLab.

Вернитесь в Excel, сделайте текущей ячейку `A5` и нажмите кнопку **getmatrix**. Появляется диалоговое окно со строкой ввода, в которой требуется ввести имя переменной, импортируемой в Excel. В данном случае такой переменной является `IM`. Нажмите **OK**, в ячейки с `A5` по `C7` введены элементы обратной матрицы.

MatLab производит вычисления с двойной точностью. Для отображения всех значащих цифр в ячейках листа Excel выделите нужные ячейки и перейдите в меню **Формат** к пункту **Ячейки**. В открывшемся диалоговом окне **Формат ячеек** выберите **Числовой** в списке **Формат ячеек** и установите 15 в строке ввода **Число десятичных знаков**.

Итак, для экспорта матрицы в MatLab следует выделить подходящие ячейки листа Excel, а для импорта достаточно указать одну ячейку, которая будет являться верхним левым элементом импортируемого массива. Остальные элементы запишутся в ячейки листа согласно размерам массива, переписывая содержащиеся в них данные, поэтому следует соблюдать осторожность при импорте массивов.

Вышеописанный подход является самым простым способом обмена информацией между приложениями — исходные данные содержатся в Excel, затем экспортируются в MatLab, обрабатываются там некоторым образом и результат импортируется в Excel. Пользователь переносит данные при помощи кнопок панели инструментов Excel Link. Информация может быть представлена в виде матрицы, т. е. прямоугольной или квадратной области рабочего листа. Ячейки, расположенные в строку или столбец, экспортируются, соответственно, в вектор-строки и вектор-столбцы MatLab. Аналогично происходит и импорт вектор-строк и вектор-столбцов в Excel.

Наряду с числовыми массивами, объектами, подлежащими обмену между MatLab и Excel, могут быть текстовые данные. Введите в ячейки некоторый текст (рис. 19.5).

	A	B	C
1	January	February	March
2	April	May	June
3	July	August	September
4	October	November	December

Рис. 19.5. Ввод в ячейки текстовой информации

Выделите ячейки с `A1` по `C4` и экспортируйте данные в переменную `mounth` рабочей среды MatLab при помощи кнопки **putmatrix**. Выясните тип переменной `mounth`, используя команду `whos`:

```
>> whos mounth
  Name          Size          Bytes    Class
  mounth        4x3           1252     cell array
Grand total is 86 elements using 1252 bytes
```

Оказывается, текстовая информация из ячеек Excel записывается в массив ячеек (`cell array`) MatLab. Экспорт в MatLab текста только одной ячейки рабочего листа Excel приводит к помещению ее содержимого в символьный массив типа `char array`.

Работа с символьными массивами и массивами ячеек подробно описана в главе 8.

Импорт массива ячеек в Excel приводит к заполнению прямоугольной области на рабочем листе, размеры которой совпадают с размерами импортируемого массива. Символьный массив импортируется в одну ячейку листа Excel.

Обмен данными между приложениями может быть осуществлен не только при помощи кнопок панели инструментов **Excel Link**, но и с использованием функций, определенных в надстройке Excel Link.

Обращение к основным функциям Excel Link

Всего в Excel Link определено одиннадцать функций, распадающихся на две категории: функции для обмена данными между MatLab и Excel и функции, предназначенные для установления связи между приложениями. Данные функции могут вызываться как из ячеек рабочего листа книги Excel, так и из приложений на Visual Basic. Для начала работы необходимы три основные функции, которые фактически дублируются кнопками панели инструментов **Excel Link**.

Функция `MLPutMatrix` служит для помещения данных из ячеек листа Excel в массив рабочей среды MatLab. Первым входным аргументом `MLPutMatrix` является имя переменной, заключенное в кавычки, а вторым — пределы

области ячеек. Обратную операцию производит функция `MLGetMatrix`, в первом аргументе которой указывается имя переменной рабочей среды MatLab с данными, а во втором — пределы области ячеек рабочего листа. Оба аргумента заключаются в кавычки. Следует иметь в виду, что входные аргументы функций в Excel разделяются точкой с запятой при вызове функций из ячеек листа.

Обращение из Excel к командам MatLab производится при помощи функции `MLEvalString`. Команды, подлежащие выполнению, задаются в единственном входном аргументе `MLEvalString`, который заключается в кавычки. Возможно указание строки с несколькими командами, разделенными точкой с запятой, но все равно в кавычки берется вся строка, а не отдельные команды. Входной аргумент у `MLEvalString` только один.

Наберите в ячейках квадратную матрицу (см. рис. 19.4), затем поместите в ячейку `E2` вызов функции `=MLPutMatrix("M";A1:C3)`. Обращение к функции из ячейки рабочего листа начинается со знака "равно". Нажатие на клавишу <Enter> для завершения ввода в ячейку приводит к выполнению ее содержимого. В данном случае происходит считывание содержимого области ячеек с `A1` по `C3` в числовой массив `M`. Занесите в ячейку `E4` вызов `=MLEvalString("IM=inv(M)")`. После выхода из `E4` MatLab обращает матрицу *M* и записывает результат в *IM*. Вызовите из ячейки `E6` функцию `=MLGetMatrix("IM"; "A5:C7")`, импортирующую обратную матрицу в ячейки Excel с `A5` по `C7`. Вид рабочего листа приведен на рис. 19.6, в ячейках с формулами отображаются нули, содержимое каждой из трех ячеек указано на рисунке отдельно.

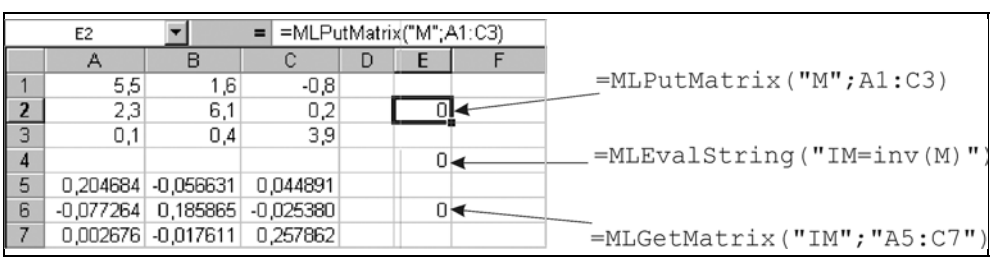


Рис. 19.6. Содержимое рабочего листа

Пользователи, имеющие опыт программирования на Visual Basic, могут использовать функции Excel Link в своих программах. Листинг 19.4 содержит текст модуля с процедурой `MyInv`, выполняющей те же действия, что и последовательный вызов из ячеек рабочего листа трех функций: `MLPutMatrix`, `MLEvalString` и `MLGetMatrix`. При написании данной процедуры в среде

Microsoft Visual Basic следует установить ссылку на файл excllink.xla при помощи пункта **Ссылки** меню **Сервис**.

Листинг 19.4. Процедура `MyInv`

```
Function MyInv(Mrange, IMrange)
MLPutMatrix "M", Mrange
MLEvalString "IM=inv(M)"
MLGetMatrix "IM", IMrange
End Function
```

Вызов процедуры `MyInv` производится из свободной ячейки рабочего листа и выглядит следующим образом: `=MyInv(A1:C3; "A5:C7")`. Функции Excel Link могут оперировать с различным заданием области ячеек, в том числе и по имени. Подробные сведения содержатся в справочной системе Excel в разделе "Справочник по Visual Basic". Несколько примеров, касающихся интегрирования MatLab и Excel, включены в книгу ExliSamp.xls, которая находится в том же подкаталоге MatLab, что и надстройка excllink.xla.

Функции Excel Link

В Excel Link определено семь функций, обеспечивающих экспорт и импорт данных при совместной работе в MatLab и Excel. Три из них: `MLPutMatrix`, `MLEvalString` и `MLGetMatrix` описаны в предыдущем разделе.

Функция `MLAppendMatrix` так же, как и `MLPutMatrix`, предназначена для экспорта данных в MatLab. Основное отличие состоит в том, что в случае экспорта данных из ячеек в массив, существующий в рабочей среде, функция `MLAppendMatrix`т пытается добавить данные к содержимому массива. Способ занесения данных требует совпадения числа строк или столбцов в массиве и области ячеек. В случае неоднозначности, т. е. когда содержимое ячеек может быть добавлено как в виде строк, так и в виде столбцов, создаются новые строки. Если же размеры области ячеек не соответствуют массиву, то функция `MLAppendMatrix` возвращает ошибку. Во внимание принимается также тип данных.

Удаление массива рабочей среды MatLab производится при помощи функции `MLDeleteMatrix`, входным аргументом которой является имя массива, заключенное в кавычки. В случае отсутствия массива в рабочей среде выдается сообщение.

При программировании на Visual Basic оказывается полезной функция `MLPutVar`, которая предназначена для экспорта значения переменной в ра-

бочую среду MatLab. Первым аргументом `MLPutVar` является имя переменной MatLab, а вторым — имя переменной, которая используется в процедуре на Basic. Импорт данных из рабочей среды в переменную процедуры осуществляет функция `MLGetVar`. Порядок аргументов `MLGetVar` такой же, как и у `MLPutVar`. Очевидно, что две эти функции используются только в процедурах при программировании на Basic.

Четыре функции: `matlabinit`, `MLAutoStart`, `MLClose`, `MLOpen`, обеспечивающие согласованную работу MatLab и Excel, образуют вторую группу функций Excel Link. Подробная информация о данных функциях содержится в справочной системе по Excel Link.

Глава 20



Редактирование приложений с GUI версии 5.3 в версии 6.*x*

Среда MatLab GUIDE предназначена для разработки приложений с графическим интерфейсом пользователя (GUI), однако в версиях 5.3 и 6.*x* реализованы различные подходы к программированию приложений и их хранению. Переход на новую версию MatLab требует некоторой переработки старых приложений с графическим интерфейсом пользователя для того, чтобы их можно было использовать в версии 6.*x*. Приложение с графическим интерфейсом в версии 5.3 содержится в двух файлах с расширениями m и mat. Обработка событий может быть запрограммирована в файл-функции в отдельном M-файле, создаваемом программистом. Версия 6.*x* предполагает другой формат хранения — с приложением связаны файлы с расширениями fig и m, причем M-файл создается автоматически и содержит файл-функцию с подфункциями. Заголовки нужных подфункций генерируются при описании событий элементов управления. Данная глава посвящена описанию способа модернизации приложений с GUI, написанных в MatLab 5.3, позволяющего работать с ними в более новой версии MatLab.

Пример приложения для MatLab 5.3

Данный раздел содержит пример простого приложения `myplot` с GUI, созданного в среде GUIDE версии 5.3. Приложение `myplot` хранится в файлах myplot.mat и myplot.m. Окно запущенного приложения приведено на рис. 20.1.

Пользователь определяет в строке ввода функцию переменной `x` в соответствии с правилами MatLab, нажимает кнопку **Построить график** и получает график функции на отрезке [0, 1]. Кнопка **Очистить оси** позволяет убрать линию графика. Приложение `myplot` достаточно простое, оно предназначено для демонстрации способа модернизации приложений с графическим интерфейсом, написанных в MatLab 5.3, до версии 6.*x*. С приложением связана файл-функция `myplotprog`, в которой запрограммирована обработка событий `Callback` элементов управления: нажатие на кнопки и завершение ввода в строку при помощи <Enter>. Текст файл-функции `myplotprog` со-

держится в листинге 20.1. Имена объектов окна приложения и вызовы `myplotprog` на события `Callback` данных объектов приведены в табл. 20.1.

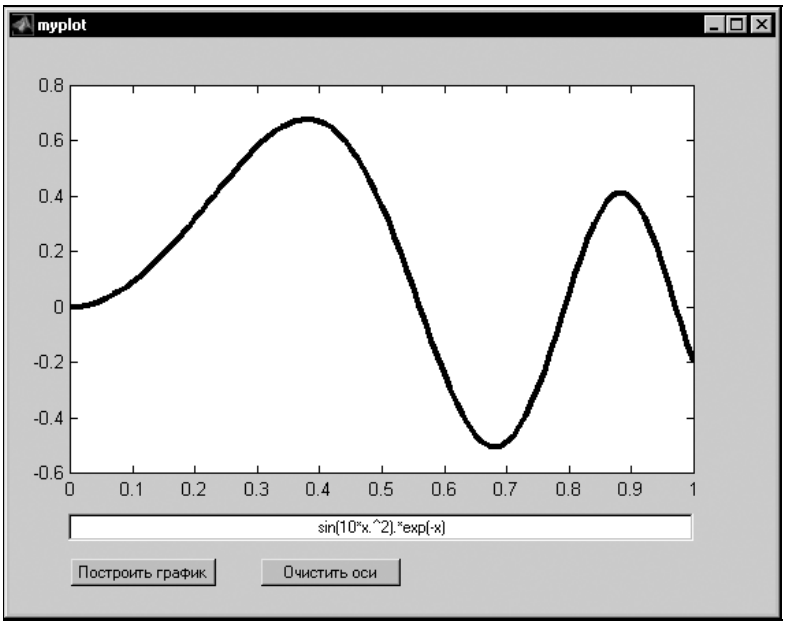


Рис. 20.1. Окно приложения `myplot`

Таблица 20.1. *Имена объектов приложения* `myplot`

Объект	**Имя**	**Событие** `Callback`
Строка ввода	`FunEdt`	`myplotprog('SetFun')`
Кнопка **Построить график**	`PlotBtn`	`myplotprog('PressPlot')`
Кнопка **Очистить оси**	`ClearBtn`	`myplotprog('PressClear')`

Листинг 20.1. Файл-функция `myplotprog`

```
function myplotprog(event)
global FunStr
switch event
case 'SetFun'
   % Поиск указателя на строку ввода и занесение его в HFunEdt
   HFunEdt = findobj('Tag', 'FunEdt');
```

```
   % занесение в глобальную переменную FunStr содержимого строки ввода
   FunStr = get(HFunEdt, 'String');
case 'PressPlot'
   % Задание вектора аргументов для построения графика
   x = [0:0.01:1];
   % Конструирование и выполнение строки для вычисления вектора значений
   % аргумента
   eval(['y=' FunStr ';'])
   plot(x,y,'LineWidth', 3, 'Color', 'k') % Построение графика
case 'PressClear'
   cla  % очистка осей
end
```

Сведения, изложенные в следующем разделе, предполагают понимание принципов конструирования приложений с графическим интерфейсом пользователя как в версии 5.3, так и в 6.*x*. Обратитесь при необходимости к соответствующим разделам книги, посвященным дескрипторной графике и созданию приложений в MatLab.

Использование дескрипторной графики описано в главе 9, третья часть книги освещает вопросы написания приложений с GUI с учетом особенностей версий MatLab.

Модернизация приложения для версии 6.*x*

Приложение версии 5.3 с графическим интерфейсом пользователя, как правило, содержится в файлах с расширениями mat и m. Пример такого приложения `myplot` приведен в предыдущем разделе. Данный раздел описывает основные этапы перевода файлов приложения в форматы fig и m, свойственные версии 6.*x*, и изменения связанной с приложением файл-функции обработки событий. Приложение приводится к такому виду, который обеспечивает дальнейшую работу над ним в среде GUIDE версии 6.*x* без каких-либо особенностей. Процесс модернизации не требует установленной версии 5.3, необходимо иметь только файлы приложения.

Сохранение приложения в формате FIG

Сохраните файлы myplot.m, myplot.mat и myplotprog.m в текущем каталоге MatLab 6.*x* и запустите приложение, набрав `myplot` в командной строке MatLab 6.*x*. Появляется окно приложения, причем оно функционирует так

же, как и версии 5.3. Пользователь может задавать формулы в строке ввода и отображать графики функций.

Следующий этап состоит в получении указателя на окно приложения `myplot`. Убедитесь, что кроме окна приложения больше нет открытых графических окон. Свойство `HandleVisibility` окна `myplot` может быть установлено в `off`, что воспрепятствует нахождению требуемого указателя. Поэтому сначала следует задать MatLab, что все указатели на объекты должны быть доступны. Данная установка производится выбором значения `on` свойства `ShowHiddenHandles` объекта `Root`. Указатель на объект `Root` равен нулю. Используйте функцию `set` для требуемого изменения значения:

```
>> set(0,'ShowHiddenHandles','on')
```

Теперь все указатели доступны. В данный момент открыто только одно графическое окно приложения, которое является потомком `Root`.

Иерархия объектов описана в разд. "События и свойства объектов в MatLab" главы 14.

Свойство `Children` объекта `Root` содержит указатели на открытые графические окна. Примените функцию `get`, возвращающую значение свойства объекта:

```
>> h = get(0,'Children');
```

Переменная `h` содержит значение указателя на окно приложения. Укажите `h` во входном аргументе `guide` для перехода к визуальной среде GUIDE. Команда

```
>> guide(h)
```

приводит к появлению среды GUIDE, в которой уже открыто приложение `myplot`. Обратите внимание, что приложение `myplot` осталось запущенным, поскольку открыто его графическое окно. Данное окно следует закрыть и продолжить работу в среде GUIDE.

Выберите в меню **File** среды GUIDE пункт **Save as**, появляется диалоговое окно **Save Figure As**, в котором следует указать имя файла myplot.fig. Можно сохранить приложение и под другим именем, но тогда для его запуска придется набирать в командной строке уже не `myplot`. Прежние имена приложений облегчат работу в новой версии MatLab, особенно если модернизации подлежат несколько часто используемых ранее приложений. Закройте среду GUIDE и перейдите к изучению содержимого текущего каталога.

В текущем каталоге MatLab 6.*x* появился файл myplot.fig. Наберите в командной строке `myplot` и убедитесь, что приложение работает так же, как и в старой версии. Запуск приложения обеспечивается наличием файла myplot.m, в котором записана файл-функция `myplot`, обеспечивающая инициализацию приложения при запуске. Часть текста файл-функции `myplot`

приведена в листинге 20.2. Обратите внимание, что первым действием является загрузка файла myplot.mat при помощи `load`. Отсутствие расширения файла в `load` приводит к загрузке именно двоичного mat-файла. После команды `load` следуют вызовы функций, создающих объекты с заданными свойствами: графическое окно (функция `figure`), оси (функция `axes`) и т. д.

Листинг 20.2. Содержимое файла myplot.m (файл-функция `myplot`)

```
function fig = myplot()
load myplot % загрузка файла myplot.mat
% Создание объектов
h0 = figure('Color',[0.8 0.8 0.8], ...
       'Colormap',mat0, ...
       'FileName','C:\MATLABR11\work\GUI53to60\myplot.m', ...
       'MenuBar','none', ...
       'Name','myplot', ...
       'NumberTitle','off', ...
       'PaperPosition',[18 180 576 432], ...
       'PaperUnits','points', ...
       'Position',[446 284 560 420], ...
       'Tag','Fig1', ...
       'ToolBar','none');
...
```

Важно понимать, что запуск приложения командой `myplot` не требует наличия файла myplot.fig. На самом деле работает старая комбинация файлов с расширениями mat и m. Запустить файл myplot.fig можно с использованием `open`:

```
>> open('myplot.fig')
```

Обработка событий происходит в файл-функции, хранящейся в myplotprog.m, т. к. myplot.fig содержит соответствующие обращения к файл-функции `myplotprog`. Такой способ работы с приложениями не очень удобен. Следующий раздел описывает модернизацию приложения для версии 5.3, которая направлена на полный отказ от файлов старой версии и переход к форматам FIG и M.

Переход к форматам FIG и M

Приложение с графическим интерфейсом пользователя представляется в MatLab 6.*x* либо комбинацией файлов с расширениями fig и m, либо только fig. M-файл содержит файл-функцию, инициализирующую приложение и

подфункции обработки событий элементов интерфейса. Программирование событий в версии 6.*x* удобнее производить, если с приложением ассоциирован M-файл.

Сделайте доступными указатели на объекты так, как описано в предыдущем разделе, использовав `set(0,'ShowHiddenHandles','on')`. Запустите приложение `myplot` из командной строки (запускается файл myplot.m) и получите указатель на графическое окно приложения `h=get(0,'Children')`, предварительно закрыв все лишние окна. Откройте в среде GUIDE MatLab 6.*x* приложение при помощи `guide(h)`.

Выберите в меню **Tools** среды GUIDE пункт **Application Options**, появляется диалоговое окно **GUIDE Application Options**, позволяющее настроить приложение. Поставьте переключатель в положение **Generate .fig and .m file**, становятся доступными флаги в нижней части окна. Флаг **Generate callback function prototypes** должен быть включен для автоматической генерации подфункций обработки событий. В раскрывающемся списке **Command-line accessibility** следует выбрать опцию **On (recommended for plotting windows)**.

Сохраните приложение с именем myplot.fig в среде GUIDE, выбрав в меню **File** пункт **Save as**. Если вы ранее осуществили действия, описанные в предыдущем разделе, то появится окно с предупреждением о том, что файл myplot.fig уже существует — перепишите его, нажав **Yes**. Далее снова появится окно с сообщением о существовании файла myplot.m. Предлагается либо заменить старый файл на новый (кнопка **Replace**), либо добавить к нему новый файл myplot.m (кнопка **Append**). Прежний файл содержит файл-функцию `myplot` (см. листинг 20.2), инициализирующую приложение в стиле версии 5.3. Поскольку требуется полностью отказаться от формата версии 5.3, то следует выбрать кнопку **Replace**.

Нажатие на **Replace** приводит к запуску редактора M-файлов, в котором открыт новый файл myplot.m, содержащий автоматически сгенерированную файл-функцию `myplot`. В среде GUIDE выделите щелчком мыши строку ввода и в контекстном меню, активизируемом правой кнопкой мыши, выберите пункт **Edit Callback**. Обратите внимание, что в файле myplot.m появилась подфункция `FunEdt_Callback` обработки события `CallBack` строки ввода. Аналогичным образом последовательно создайте подфункции `PlotBtn_Callback` и `ClearBtn_Callback`, соответствующие кнопкам **Построить график** и **Очистить оси**. В результате файл myplot.m должен иметь структуру, приведенную в листинге 20.3. Запустите из среды GUIDE приложение и убедитесь, что оно работает правильно.

Листинг 20.3. Содержимое сгенерированного файла myplot с подфункциями

```
function varargout = myplot(varargin)
...
```

```
if nargin == 0  % LAUNCH GUI
        fig = openfig(mfilename,'reuse');
        handles = guihandles(fig);
        guidata(fig, handles);
        if nargout > 0
                varargout{1} = fig;
        end
elseif ischar(varargin{1})
        try
                [varargout{1:nargout}] = feval(varargin{:}); % FEVAL
switchyard
        catch
                disp(lasterr);
        end
end
% --------------------------------------------------------------------
function varargout = FunEdt_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.FunEdt.
myplotprog('SetFun')
% --------------------------------------------------------------------
function varargout = PlotBtn_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.PlotBtn.
myplotprog('PressPlot')
% --------------------------------------------------------------------
function varargout = ClearBtn_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.ClearBtn.
myplotprog('PressClear')
```

Изучите три полученные подфункции обработки событий. Каждая подфункция содержит единственный оператор вызова файл-функции `myplotprog` с соответствующим входным аргументом. При возникновении события `Callback` от любого из элементов управления происходит обращение к подфункции `myplotprog`, в которой выполняются команды одного из блоков `case` оператора `switch`. Последний этап модернизации приложения заключается в программировании подфункций обработки событий. Данные подфункции должны выполнять те же действия, что и блоки `case` в `myplotprog`, разумеется, с учетом принципов написания приложений в среде GUIDE MatLab 6.*x*.

Указатели на графические объекты содержатся в полях структуры `handles`, имена которых совпадают со значениями свойства `Tag` данных объектов. Об-

мен данными между подфункциями производится при помощи создания дополнительных полей в структуре `handles`. После занесения в поле `handles` нужного значения следует сохранить структуру, используя функцию `guidata`.

Пример использования структуры `handles` *приведен в разд. "Переключатели" главы 12.*

Подфункции обработки событий элементов интерфейса обновленного приложения `myplot` приведены в листинге 20.4.

Листинг 20.4. Подфункции обработки событий

```
% --------------------------------------------------------------------
function varargout = FunEdt_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.FunEdt.
% Занесение в HFunEdt указателя на строку ввода
HFunEdt = handles.FunEdt;
% Образование нового поля FunStr структуры handles и занесение в него
% содержимого строки ввода
handles.FunStr = get(HFunEdt, 'String');
% Сохранение обновленной структуры handles
guidata(gcbo, handles)

% --------------------------------------------------------------------
function varargout = PlotBtn_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.PlotBtn.
% Задание вектора аргументов для построения графика
x = [0:0.01:1];
% Конструирование и выполнение строки для вычисления вектора значений
% аргумента
eval(['y=' handles.FunStr ';'])
plot(x,y,'LineWidth', 3, 'Color', 'k') % Построение графика

% --------------------------------------------------------------------
function varargout = ClearBtn_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.ClearBtn.
cla % очистка осей
```

Запустите приложение `myplot` и проверьте его работу. Файлы myplot.mat и myplotprog.m, ассоциированные со старой версией приложения, больше не нужны, их можно удалить. Модернизированное приложение с графическим интерфейсом пользователя содержится в двух файлах myplot.fig и myplot.m.

Глава 21



Повышение производительности приложений MatLab

Данная глава описывает некоторые принципы эффективной разработки приложений в MatLab. Описаны способы увеличения скорости матричных вычислений в M-файлах, компилирование файл-функций для получения исполняемых MEX-файлов и исходного кода на языке C. На достаточно простом уровне объяснено использование программного интерфейса MatLab API для написания совместных с Fortran приложений.

Ускорение работы M-файлов

MatLab интерпретирует команды, записанные в M-файлах, в машинный код и последовательно выполняет их. Процесс интерпретации занимает много времени в том случае, когда алгоритм обработки большого объема данных содержит циклы, поскольку каждая строка цикла интерпретируется столько раз, сколько выполняется цикл. Следовательно, при разработке приложений MatLab необходимо свести использование циклов к минимуму. Эффективность приложений также определяется распределением памяти под создаваемые большие массивы. Принципиально другой способ повышения быстродействия программ состоит в компилировании программ при помощи MatLab Compiler в машинный код и обращении к нему из среды и приложений MatLab.

Поэлементные операции

Достаточно часто требуется произвести преобразование элементов массива, к примеру, разделить все элементы одной матрицы на соответствующие элементы другой. Данную операцию можно осуществить двумя способами: перебрать все элементы матрицы во вложенных циклах `for` и поделить на соответствующие значения, либо использовать операцию поэлементного деления. Первый вариант реализован в файл-программе `test1` (листинг 21.1), а второй — в файл-программе `test2` (листинг 21.2). Для примера выбраны квадратные матрицы A и B одинаковых размеров, заполняемые при помощи функции `ones`, требуется записать результат в матрицу A, т. е. `A(i,j)=A(i,j)/B(i,j)`.

Листинг 21.1. Файл-программа `test1` (перебор в цикле)

```
A=ones(400);
B=5*ones(400);
for i=1:400
    for j=1:400
        A(i,j)=A(i,j)/B(i,j);
    end
end
```

Листинг 21.2. Файл-программа `test2` (использование поэлементных операций)

```
A=ones(400);
B=5*ones(400);
A=A./B;
```

Сравните время, затрачиваемое на работу `test1` и `test2`. Можно воспользоваться, например, профайлером для получения детальной информации о работе файл-программ.

См. разд. "Профайлер" главы 16.

В рассматриваемых примерах требуется определить только время работы каждого из M-файлов test1.m и test2.m, поэтому проще применить последовательность команд `tic` и `toc`, которая предназначена для установления временных затрат на выполнение M-файла или нескольких коротких команд. Определите время вычисления по первому и второму способам:

```
>> tic, test1, toc
elapsed_time =
    3.2500
>> tic, test2, toc
elapsed_time =
    0.0940
```

Обратите внимание, что поэлементные операции уменьшают временные затраты в десятки и сотни раз!

Замечание

Значения времени, полученные на различных компьютерах, могут не совпадать с приведенными в книге, но соотношение остается тем же.

Следующая задача имеет менее очевидное решение. Предположим, что требуется преобразовать элементы матрицы по формуле: `A(i,j)=A(i,j)/(i*j)`. Деление всех элементов матрицы `A` во вложенных циклах не представляет труда (листинг 21.3).

Листинг 21.3. Файл-программа `test3` (деление элементов во вложенных циклах)

```
A=ones(400);
for i=1:400
    for j=1:400
        A(i,j)=A(i,j)/(i*j);
    end
end
```

Для применения поэлементных операций требуется сформировать квадратную матрицу того же размера, что и A, элементы которой являются произведением номера строки на номер столбца. Вспомогательная матрица M является внешним произведением подходящего столбца и строки (листинг 21.4).

Листинг 21.4. Файл-программа `test4` (поэлементное деление на вспомогательную матрицу)

```
A=ones(400);
row=[1:400];
M=row'*row;
A=A./M;
```

Сравнение временных затрат убеждает в эффективности способа, реализованного в файл-программе `test4`:

```
>> tic, test3, toc
elapsed_time =
    3.1410
>> tic, test4, toc
elapsed_time =
    0.0780
```

Еще одним приемом, позволяющим сократить время расчетов, является предварительное выделение памяти под заполняемые массивы.

Выделение памяти под массивы

Массивы среды MatLab не требуют предварительного объявления, в отличие от многих других языков программирования. Тип массива определяется во время присвоения значения. Аналогичным образом, размеры массива изменяются по мере присвоения новым его элементам некоторых значений. К примеру, оператор присваивания, заполняющий матрицу *A*

```
>> A=[-3 4; 9 7];
```

и последовательность команд, приведенная ниже, приводят к созданию одинаковых матриц *A* и *B* (предполагается, что переменные `A` и `B` до этого не были инициализированы в рабочей среде).

```
>> B(1,1)=-3;
>> B(1,2)=4;
>> B(2,1)=9;
>> B(2,2)=7;
```

Работа с небольшими массивами не приводит к заметному увеличению временных затрат, однако оперирование большими объемами данных делает существенной разницу между двумя перечисленными способами заполнения массивов. Составьте две файл-процедуры для заполнения квадратной матрицы размером четыреста. Элементы матрицы вычисляются во вложенных циклах по `i` и `j` по формуле `1/(i^2+j^2)`, причем в начале одной файл-процедуры создайте нулевую квадратную матрицу при помощи функции `zeros` (листинги 21.5 и 21.6).

Листинг 21.5. Файл-программа `test5` без предварительного выделения памяти

```
for i=1:400
    for j=1:400
        B(i,j)=1/(i^2+j^2);
    end
end
```

Листинг 21.6. Файл-программа `test6` с предварительным выделением памяти

```
A=zeros(400);
for i=1:400
    for j=1:400
        A(i,j)=1/(i^2+j^2);
    end
end
```

Теперь найдите затраты машинного времени на выполнение каждого из M-файлов: test5.m и test6.m. Учтите, что перед использованием `tic` и `toc` следует очистить рабочую среду MatLab командой `clear all`, поскольку массив `B` мог быть создан ранее, а в этом случае под него уже была выделена память.

```
>> clear all
>> tic, test5, toc
elapsed_time =
    7.8900
>> tic, test6, toc
elapsed_time =
    3.4060
```

За счет предварительного выделения памяти под заполняемый массив достигнута существенная экономия времени, более чем в два раза. Важно понимать, что первый запуск `test5` приводит к последовательному отведению места в памяти под массив, а по завершении работы файл-программы весь массив находится в памяти. Следовательно, повторная работа `test5` (без очистки рабочей среды) займет столько же времени, сколько и `test6`:

```
>> tic, test5, toc
elapsed_time =
    3.4060
```

Перечисленные в данном разделе основные подходы повышают быстродействие работы M-файлов, однако сам принцип их выполнения остается тот же — интерпретация каждой строки в машинный код и последующее выполнение. Пакет MatLab имеет в своем составе компилятор MatLab Compiler, который позволяет кардинально решить проблему увеличения эффективности приложений, написанных на встроенном языке MatLab.

Компилирование M-файлов

Процесс компилирования M-файлов возможен только при наличии компонента MatLab Compiler. Команда `ver`, выполняемая из командной строки рабочей среды, выводит список всех пакетов, входящих в установленную на компьютере версию MatLab. Перед изучением текущего раздела следует убедиться, что имеется MatLab Compiler, иначе придется воспользоваться инсталляционной программой для его добавления. MatLab Compiler позволяет преобразовать файл-функцию, написанную на встроенном языке MatLab, в C-код и получить из него исполняемый модуль, так называемый MEX-файл, который вызывается из среды или приложений MatLab и динамически линкуется во время выполнения. В системе Windows MEX-файлы имеют рас-

ширение dll. С точки зрения пользователя MatLab, вызов MEX-файла не отличается от M-файла, однако скомпилированный MEX-файл имеет ряд преимуществ по сравнению с M-файлом. Во-первых, существенно повышается скорость работы приложения, особенно содержащего циклически повторяющиеся действия. Во-вторых, MatLab Compiler позволяет получить независимые от MatLab приложения, и в-третьих, скрывается реализация алгоритма, поскольку M-файл с открытым кодом не нужен для работы соответствующего MEX-файла.

Конфигурирование MatLab Compiler

Перед компиляцией программ необходимо произвести некоторые установки, связанные с выбором компилятора. Наберите в командной строке

```
>> mex -setup
```

Запускается программа конфигурирования MatLab Compiler с интерфейсом из командной строки. В версии 5.3 открывается отдельное окно, а в 6.*x* сообщения выводятся прямо в командное окно MatLab. Предлагается выбрать компилятор C, причем имеется возможность автоматического поиска всех компиляторов, имеющихся на компьютере.

```
Please choose your compiler for building external interface (MEX) files:
 Would you like mex to locate installed compilers [y]/n?
```

Введите `y` на запрос об автоматическом поиске. Появляется список доступных компиляторов C и Fortran (который может отличаться от приведенного ниже):

```
Select a compiler:
[1] Compaq Visual Fortran version 6.1 in C:\cvf61
[2] Digital Visual Fortran version 6.0 in C:\Microsoft Visual Studio
[3] Lcc C version 2.4 in C:\MATLAB\sys\lcc
[4] Microsoft Visual C/C++ version 6.0 in C:\Microsoft Visual Studio
[0] None
Compiler:
```

Выберите номер одного из компиляторов C. Если C не установлен на вашем компьютере, то придется воспользоваться Lcc, который входит в поставку MatLab. Далее предлагается подтвердить сделанный выбор:

```
Please verify your choices:
 Compiler: Microsoft Visual C/C++ 6.0
Location: C:\Program Files\Microsoft Visual Studio
 Are these correct?([y]/n):
```

Введите y и нажмите клавишу <Enter>. Конфигурирование MatLab Compiler завершено (в версии 5.3 следует закрыть окно, выполнив в его командной строке exit). Перейдите к изучению некоторых примеров, разобранных в следующем разделе.

Компилирование файл-функций

MatLab Compiler предназначен для компилирования только файл-функций, попытка скомпилировать файл-программу приведет к сообщению об ошибке. Создайте в текущем каталоге простейшую файл-функцию exam, заполняющую квадратную матрицу некоторыми числами. Текст файл-функции exam приведен в листинге 21.7.

Листинг 21.7. Файл-функция exam

```
function A=exam(n)
A=zeros(n);
for i=1:n
    for j=1:n
        A(i,j)=1/(i^2+j^2);
    end
end
```

Установите, сколько времени занимает заполнение матрицы размера, к примеру, четыреста. Используйте команды tic и toc, завершите вызов exam точкой с запятой для подавления вывода элементов матрицы в командное окно:

```
>> tic, A=exam(400); toc
elapsed_time =
    3.3910
```

Скомпилируйте файл-функцию exam, для чего используйте команду mcc, которая запускает MatLab Compiler. Укажите опцию -x для генерации MEX-файла, являющегося для Windows библиотекой динамической линковки:

```
>> mcc -x exam
```

Процесс компилирования занимает некоторое время, в случае успешного завершения не выдается никаких сообщений, просто появляется командная строка для продолжения работы. Если все же возникли ошибки, то следует проверить, правильно ли запрограммирована файл-функция, является ли каталог с ней текущим и был ли сконфигурирован MatLab Complier так, как

описано в предыдущем разделе. После компилирования в текущем каталоге появляется четыре новых файла:

- файлы exam.c и exam.h содержат код на языке C, реализующий алгоритм файл-функции `exam`;
- файл exam_mex.c является интерфейсом к MEX-функции;
- MEX-файл exam.dll.

Читатели, знакомые с языком программирования C, могут разобраться в структуре файлов exam.c, exam.h и exam_mex.c при помощи справочной системы MatLab. Сейчас нас будут интересовать преимущества использования MEX-файла exam.dll вместо файл-функции `exam`, содержащейся в M-файле.

Как уже было отмечено, обращение из среды MatLab к MEX-файлу, или файл-функции в виде M-файла происходит аналогичным образом. Если при вызове функции MatLab обнаруживает MEX-файл и M-файл с одинаковым именем, то выполняется MEX-файл. Следующие команды приводят к выполнению именно MEX-файла:

```
>> tic, A=exam(400); toc
elapsed_time =
    0.5780
```

Обратите внимание на существенную разницу во времени вычислений (см. выше предыдущий замер времени). Выполнение скомпилированного файла, содержащего циклы, практически всегда происходит быстрее, чем интерпретация команд.

Важно понимать, что файлы с расширениями c и h не нужны для того, чтобы MatLab выполнил MEX-файл, т. е. функцию из библиотеки динамической линковки exam.dll. Файлы с кодом на C создаются командой `mcc` потому, что процесс компиляции M-файлов в MEX-файлы состоит из двух этапов. Сначала автоматически создаются нужные файлы с исходным кодом на C, а затем применяется команда `mex` для их компилирования.

Замечание

Пользователь MatLab имеет возможность использовать собственные программы на C или Fortran, предварительно написав соответствующий интерфейс. Подробнее об этом сказано ниже.

Все образовавшиеся файлы (кроме exam.dll) можно удалить, но учтите, что внесение изменений в алгоритм потребует модернизации M-файла и его повторного компилирования. Обязательно оставляйте M-файл с исходным кодом на языке MatLab, если в дальнейшем потребуется продолжение работы над приложением!

Компилирование нескольких файл-функций

Приложение MatLab может состоять из нескольких файл-функций, в одной из которых происходит вызов остальных функций. Пример такого приложения приведен в листингах 21.8 и 21.9.

Листинг 21.8. Основная функция `mainfun`, содержащаяся в mainfun.m

```
function d=mainfun(n)
d=zeros(1,n);
for i=1:n
    % Вызов подфункции в цикле
    A=subfun(i);
    d(i)=det(A);
end
```

Листинг 21.9. Вспомогательная функция `subfun`, содержащаяся в файле subfun.m

```
function M=subfun(k)
M=zeros(k);
for i=1:k
    for j=1:k
        M(i,j)=1/(i^2+j^2);
    end
end
```

Одним из возможных вариантов создания MEX-файла является компилирование только `mainfun`:

```
>> mcc -x mainfun
```

Замерьте время, затрачиваемое на вычисления определителя матриц до сотого порядка включительно:

```
>> tic, v=mainfun(100); toc
elapsed_time =
    7.6410
```

Теперь скомпилируйте обе файл-функции в один MEX-файл. Имя основной файл-функции (вызывающей другую) указывается первым в команде `mcc`:

```
>> mcc -x mainfun subfun
```

Эффективность MEX-файла, созданного из вызывающей и вызываемой функций, выше, чем в предыдущем случае:

```
>> tic, v=mainfun(100); toc
elapsed_time =
    1.1250
```

Работа с файл-программами

Файл-программу, в отличие от файл-функции, не удастся скомпилировать при помощи `mcc`, выведется сообщение об ошибке и процесс компилирования остановится. Выход состоит в преобразовании файл-программы в файл-функцию. Предположим, что требуется получить MEX-файл из M-файла, содержимое которого приведено в листинге 21.10.

Листинг 21.10. Файл-программа `exam1`

```
n=4;
A=rand(n);
B=randn(n);
C=A*B;
```

Читателю, знакомому с принципами программирования в MatLab, не составит труда образовать файл-функцию на основе файл-программы `exam1`.

См. разд. "Типы M-файлов" главы 5, главы 7 и 8.

Требуемая файл-функция с тем же именем `exam1` приведена в листинге 21.11.

Листинг 21.11. Файл-функция `exam1`

```
function C=exam1(n)
A=rand(n);
B=randn(n);
C=A*B;
```

Теперь можно использовать MatLab Compiler для получения MEX-файла exam1.dll, и вызывать `exam1` из среды или других приложений MatLab:

```
>> mcc -x exam1
>> C=exam1(3)
C =
   -1.1775    1.6221    0.6792
```

```
-1.4012    1.8370    0.5019
-1.0146    1.3565    0.4850
```

Пользователи MatLab, имеющие опыт программирования на Fortran и C, несомненно, заинтересуются возможностями, которые предоставляет MatLab Compiler. Он применяется не только для получения MEX-файлов, но и для создания самостоятельных приложений, независящих от MatLab. Некоторые функции приложения могут являться M-файлами, а остальные запрограммированы на Fortran или C. Данное обстоятельство позволяет использовать имеющиеся вычислительные модули в приложениях MatLab.

Производительность генерируемого из M-файлов C-кода может быть существенно повышена за счет использования ряда дополнительных опций при вызове `mcc`, в частности:

- оптимизация работы с массивами;
- оптимизация циклов с целым счетчиком цикла;
- оптимизация условных операторов.

Дополнительную информацию можно получить в соответствующем справочном разделе по MatLab Compiler.

Генерация MEX-файлов

Довольно часто встречается ситуация, когда имеются модули, написанные на C или Fortran, и требуется использовать их в приложении MatLab. Пакет MatLab предоставляет возможность вызова внешних программ благодаря Application Program Interface (API), т. е. программному интерфейсу приложения. Функции MatLab API делятся на две категории: функции, обеспечивающие создание и доступ к массивам MatLab, и функции, позволяющие оперировать в рабочей среде MatLab. Создание MEX-файлов из процедур Fortran заключается в написании интерфейсной процедуры с использованием функций MatLab API и последующей генерации MEX-файла при помощи команды `mex`. Генерация MEX-файлов из процедур Fortran требует установки одного из компиляторов: Digital Visual Fortran 5.0 или Compaq Visual Fortran 6.1 (только для MatLab 6.*x*). Встроенный компилятор Fortran не входит в пакет MatLab, в отличие от компилятора Lcc для приложений на C.

Простой пример, сложение двух чисел

Начните изучение создания MEX-файлов с самого простого примера. Предположим, что из среды MatLab требуется вызвать процедуру `sum`, написанную на Fortran. Процедура `sum` складывает два вещественных числа `a` и `b` и записывает результат в `c` (листинг 21.12). Будем считать, что процедура `sum` содержится в файле mysum.f.

Листинг 21.12. Исходная процедура sum на Fortran (файл mysum.f)

```
      subroutine sum(a,b,c)
C   Процедура sum складывает два вещественных числа
      real*8 a,b,c
      c=a+b
      end
```

Непосредственно вызвать процедуру sum из среды MatLab, разумеется, не удастся. Необходимо написать интерфейсную процедуру на Fortran с использованием функций MatLab API, которая будет точкой входа MEX-файла mysum.dll при вызове функции mysum из среды MatLab. Интерфейсная процедура должна называться mexFunction и иметь четыре аргумента типа integer*4:

- nrhs — число *входных* аргументов, с которыми будет происходить обращение к MEX-функции mysum из среды MatLab;
- prhs — массив указателей на *входные* аргументы mysum;
- nlhs — число *выходных* аргументов, с которыми будет происходить обращение к MEX-функции mysum из среды MatLab;
- plhs — массив указателей на *выходные* аргументы mysum.

Обмен данными между процедурой sum и средой MatLab посредством интерфейсной процедуры mexFunction осуществляется только при помощи вышеперечисленных аргументов. Для выделения данных из prhs и занесения результата в plhs предназначены специальные функции MatLab API. Интерфейсная для sum процедура mexFunction приведена в листинге 21.13, назовите файл с ней mysumg.f. Итак, интерфейсная процедура имеет четыре параметра типа integer*4, два массива plhs, prls и два скаляра nlhs, nrhs.

В mexFunction будут использоваться две функции MatLab API: mxCreateFull и mxGetPr, возвращающие значения типа integer*4. Функция mxCreateFull выделяет память под массив и возвращает указатель на массив. В рассматриваемом примере этот массив служит для возврата результата (суммы двух чисел) в рабочую среду MatLab и поэтому имеет размеры один на один. Функция mxGetPr служит для получения указателя на первый вещественный элемент в массиве. Целочисленные переменные a_pr, b_pr и c_pr понадобятся для хранения указателей на массивы, содержащие входные аргументы и выходной аргумент. Для записи значений аргументов определены вещественные переменные a, b и c.

Предупреждение

При наборе процедур вне редактора среды Fortran следует учитывать, что первые шесть позиций в начале строки отводятся под метки, а операторы начина-

ются с седьмой позиции. Если пренебречь этим правилом, то при компиляции возникнут ошибки. Сообщения об ошибках выводятся в командное окно MatLab.

Процедура `mexFunction` начинается с получения указателей на первые (а в данном случае и единственные) вещественные элементы массивов, на которые *указывают* первый и второй элементы `prhs`. Важно понимать, что массив `prhs` содержит указатели на массивы специального типа `mxArray`. Данный тип определен в MatLab API и является единственно возможным способом обмена данными между средой MatLab и процедурой на Fortran. Все данные MatLab передаются при помощи `mxArray`, в зависимости от типа передаваемых данных следует применять соответствующие функции для извлечения указателей. Для извлечения указателей на *вещественные* массивы, содержащие значения входных аргументов MEX-функции, например `a_pr` и `b_pr`, используется функция `mxGetPr` MatLab API.

Следующий шаг состоит в записи значений входных аргументов в вещественные переменные `a` и `b`. Процедура MatLab API, которая называется `mxCopyPtrToReal8`, заносит вещественные данные в массив, объявленный в процедуре Fortran. Первым входным аргументом `mxCopyPtrToReal8` является указатель, вторым — имя массива, а третьим — количество записываемых элементов (в рассматриваемом примере требуется записать только один элемент). Процедура вызывается два раза, для занесения значений входных аргументов MEX-функции в переменные `a` и `b` интерфейсной процедуры `mexFunction`.

На данный момент в интерфейсной процедуре известны значения аргументов, от которых была вызвана MEX-функция в среде MatLab. Данное обстоятельство позволяет обратиться к процедуре `sum`, ее использование и было основной задачей. Результат, т. е. сумма `a` и `b` (см. листинг 21.12), занесен в переменную `c`.

Осталось обеспечить возвращение MEX-функцией в выходном аргументе значения переменной `c`. Предварительно необходимо создать массив при помощи функции `mxCreateFull` MatLab API. Первыми двумя входными аргументами `mxCreateFull` являются число строк и столбцов создаваемого массива, а третьим — ноль или единица. Ноль, указанный в третьем аргументе, соответствует вещественному массиву, а единица — комплексному. Функция `mxCreateFull` возвращает указатель на созданный массив, который следует записать в `plhs(1)`. Выше было сказано, что массив `plhs` должен содержать указатели на массивы с результатом, поскольку именно `plhs` используется для передачи значений в рабочую среду MatLab. Последним действием является занесение в массив значения вещественной переменной `c`, для чего применяется процедура `mxCopyReal8ToPtr` MatLab API. Первый входной аргумент `mxCopyReal8ToPtr` является переменной, второй — указателем на массив, подлежащий заполнению, а третий — количеством записываемых элементов массива.

Листинг 21.13. Интерфейсная процедура `mexFunction` (файл mysumg.f)

```
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C   Интерфейсная процедура для sum
C   Описание типов аргументов
      integer nlhs, nrhs, plhs(*), prhs(*)
C   Описание типов используемых функций из MatLab API
      integer mxCreateFull, mxGetPr
C   Описание типов указателей на используемые переменные
      integer a_pr, b_pr, c_pr
C   Описание типов используемых переменных
      real*8  a, b, c

C   Получение указателей на первый и второй входные аргументы, с которыми
C   вызывается MEX-функция, указатели хранятся в массиве prhs,
C   используется функция MatLab API
      a_pr = mxGetPr(prhs(1))
      b_pr = mxGetPr(prhs(2))
C   Запись значений первого и второго входных аргументов с указателями
C   a_pr и b_pr в переменные a и b при помощи процедуры MatLab API
      call mxCopyPtrToReal8(a_pr, a, 1)
      call mxCopyPtrToReal8(b_pr, b, 1)
C   Вызов процедуры sum, которая заносит в переменную c сумму a и b
      call sum(a, b, c)
C   Создание выходного аргумента — вещественного массива размера один на
C   один при помощи функции MatLab API, выходной аргумент есть указатель
C   на массив
      plhs(1) = mxCreateFull(1, 1, 0)
      c_pr = mxGetPr(plhs(1))
C   Копирование значения c в массив с указателем c_pr
      call mxCopyReal8ToPtr(c, c_pr, 1)
      end
```

Использование процедуры `sum` в MatLab станет возможным только после создания MEX-функции из `sum`, хранящейся в файле mysum.f и интерфейсной процедуры `mexFunction`, которая записана в файле mysumg.f. Для создания MEX-функции служит команда `mex`. Перед применением `mex` следует определить, каким компилятором Fortran будет пользоваться MatLab для

генерации MEX-функции. Используйте команду `mex` в сочетании с опцией, позволяющей сконфигурировать MatLab Compiler:

```
>> mex -setup
```

См. разд. "Конфигурирование MatLab Compiler" данной главы.

В ответ на запрос о компиляторе выберите один из поддерживаемых установленной версией MatLab: Digital Visual Fortran 5.0 или Compaq Visual Fortran 6.1 (только для MatLab версии 6.*x*). После настройки MatLab на использование компилятора Fortran можно приступать к окончательному этапу — созданию MEX-функции. Убедитесь, что файлы mysum.f и mysumg.f находятся в текущем каталоге MatLab и выполните команду `mex`, указав в качестве ее параметров имена файлов:

```
>> mex mysum.f mysumg.f
```

Обратите внимание, что имя файла с интерфейсной процедурой является вторым параметром `mex`. В текущем каталоге появился файл mysum.dll, который и содержит требуемую MEX-функцию. Проверьте ее работу, обратившись к ней из командной строки:

```
>> s=mysum(1,2)
s =
     3
```

Пример создания MEX-функции из простейшей процедуры Fortran, приведенный выше, демонстрирует основные принципы использования интерфейса приложений MatLab API. Пользователю MatLab, не имеющему достаточного опыта в области разработки интерфейса приложений, но желающему вызывать процедуры Fortran из среды MatLab, достаточно знать небольшой набор функций MatLab API. Требуются только функции, которые позволяют обмениваться основными типами данных. Интерфейсные процедуры, как правило, принципиально не отличаются для различных процедур на Fortran и состоят из трех частей:

- получение значений входных аргументов;
- вызов процедуры на Fortran;
- возвращение значений в среду MatLab.

В следующих разделах показаны особенности реализации интерфейсных процедур и соответствующие функции MatLab API для обмена данных различных типов.

Работа с комплексными переменными

Использование процедуры на Fortran, оперирующей с комплексными переменными, требует соответствующей организации обмена данных в интер-

фейсной процедуре. Кроме функции mxGetPr, придется использовать mxGetPi, которая возвращает указатель на комплексную часть первого элемента массива. Получение и запись значения переменной по указателю производится при помощи процедур mxCopyPtrToComplex16 и mxCopyComplex16ToPtr, причем входным аргументом, кроме указателя на вещественную, является еще и указатель на комплексную часть. При создании массива функцией mxCreateFull следует задать единицу в качестве третьего аргумента, поскольку предполагается хранение комплексных чисел. Текст исходной процедуры на Fortran приведен в листинге 21.14, а отвечающая ей интерфейсная процедура — в листинге 21.15.

Листинг 21.14. Процедура сложения комплексных чисел (файл mycsum.f)

```
      subroutine sum(a, b, c)
C   Процедура sum складывает два комплексных числа
      complex*16 a, b, c
      c = a + b
      end
```

Листинг 21.15. Интерфейсная процедура mexFunction (файл mycsumg.f)

```
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C   Интерфейсная процедура для sum
C   Описание типов аргументов
      integer nlhs, nrhs, plhs(*), prhs(*)
C   Описание типов используемых функций из MatLab API
      integer mxCreateFull, mxGetPr, mxGetPi
C   Описание типов указателей на используемые переменные
      integer a_pr, b_pr, c_pr, a_pi, b_pi, c_pi
C   Описание типов используемых переменных
      complex*16  a, b, c
C   Получение указателей на вещественные и мнимые части первого и второго
C   входных аргументов, с которыми вызывается MEX-функция, указатели
C   хранятся в массиве prhs, используется функции MatLab API
      a_pr = mxGetPr(prhs(1))
      a_pi = mxGetPi(prhs(1))
      b_pr = mxGetPr(prhs(2))
      b_pi = mxGetPi(prhs(2))
```

```
C   Запись значений первого и второго входных аргументов с указателями
C   a_pr, a_pi и b_pr, b_pi в переменные a и b при помощи
C   процедуры MatLab API
      call mxCopyPtrToComplex16(a_pr, a_pi, a, 1)
      call mxCopyPtrToComplex16(b_pr, b_pi, b, 1)
C   Вызов процедуры sum, которая заносит в переменную c сумму a и b
      call sum(a,b,c)
C   Создание выходного аргумента — комплексного массива размера один на
C   один при помощи функции MatLab API, выходной аргумент есть указатель
C   на массив
      plhs(1) = mxCreateFull(1, 1, 1)
      c_pr = mxGetPr(plhs(1))
      c_pi = mxGetPi(plhs(1))
C   Копирование значения c в массив с указателями
C   c_pr (на вещественную часть) и c_pi (на мнимую часть)
      call mxCopyComplex16ToPtr(c, c_pr, c_pi, 1)
      end
```

Создайте в текущем каталоге MatLab файлы mycsum.f и mycsumg.f и скомпилируйте MEX-функцию при помощи команды `mex` (подробности описаны в предыдущем разделе):

```
>> mex mycsum.f mycsumg.f
```

Убедитесь в том, что полученная MEX-функция `mycsum` работает верно.

```
>> s=mycsum(1+2i, 3-5i)
s =
   4.0000 - 3.0000i
```

Разумеется, `mycsum` правильно вычисляет сумму не только комплексных, но и вещественных аргументов:

```
>> s=mycsum(1, 3)
s =
     4
```

Обмен массивами данных

Передачу массивов данных в процедуру на Fortran и возвращение их в рабочую среду MatLab лучше всего демонстрирует пример перемножения двух матриц. Создание процедуры для нахождения произведения двух матриц не представляет труда, ее текст приведен в листинге 21.16.

Листинг 21.16. Процедура перемножения матриц (файл mymult.f)

```
      subroutine multmtr(n1, n2, n3, X, Y, Z)
C Вычисление Z=X*Y, где X — n1xn2,  Y — n2xn3,  Z — n1Xn3
      integer n1, n2, n3, i, j, k
      real*8 X(n1,n2), Y(n2,n3), Z(n1,n3)
      do 30 i=1,n1
         do 20 j=1,n3
            Z(i,j)=0.0d0
            do 10 k=1,n2
               Z(i,j)=Z(i,j)+X(i,k)*Y(k,j)
 10         continue
 20      continue
 30   continue
      end
```

Требуется сгенерировать MEX-функцию `mymult` для вычисления произведения двух матриц, причем лучшим вариантом обращения к `mymult` из среды MatLab было бы указание во входных аргументах только исходных матриц (без их размеров) и получение в выходном аргументе результата. Следовательно, работу по определению размеров перемножаемых матриц должна взять на себя интерфейсная процедура `mexFunction` (листинг 21.17).

В интерфейсной процедуре необходимо объявить указатели на массивы, соответствующие исходным матрицам и их произведению, и сами массивы. Поскольку размеры массивов заранее неизвестны, то можно ограничить их, к примеру, пятьюдесятью (в следующем разделе приведен пример использования динамических массивов). Получение количества строк и столбцов во входных аргументах MEX-функции производится при помощи функций `mxGetM` и `mxGetN` MatLab API. Данные функции вызываются от указателя на входной аргумент MEX-функции, т. е. от элемента массива `prhs`, и возвращают количество строк или столбцов.

Перед нахождением произведения матриц процедурой `mymult` целесообразно проверить соответствие размеров перемножаемых матриц, а именно совпадение `n` и `q`. В случае, когда `n` не равно `q`, необходимо прекратить работу программы и вывести соответствующее сообщение в командное окно MatLab. Процедура `mexErrMsgTxt` MatLab API служит для останова по ошибке. Строка с сообщением об ошибке указывается в качестве входного аргумента `mexErrMsgTxt`.

Замечание

Все функции и процедуры MatLab API делятся на две группы. Префикс `mx` в имени означает, что данная функция или процедура предназначена для оперирования с данными рабочей среды MatLab. Функции и процедуры, имя которых начинается с префикса `mex`, позволяют выполнять команды MatLab, выводить сообщения в командное окно, словом взаимодействовать с самой средой MatLab. Описание всех функций и процедур содержится в разделе Application Program Interface справочной системы MatLab.

После проверки входных аргументов необходимо извлечь указатели на соответствующие массивы и вызвать процедуру `mxCopyPtrToReal8` для заполнения массивов `A` и `B`, определенных в интерфейсной процедуре. Третьим аргументом `mxCopyPtrToReal8` является количество записываемых данных в каждый массив, т. е. произведение числа строк на число столбцов. Вызов процедуры `mymult` приводит к занесению результата в массив `C`. Для возвращения результата в рабочую среду MatLab осталось сформировать вещественный массив размера `m` на `q` при помощи функции `mxCreateFull`, записать указатель на него в первый элемент массива `plhs` и воспользоваться `mxGetPr` и `mxCopyReal8ToPtr`.

Листинг 21.17. Интерфейсная процедура (файл mymultg.f)

```
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C     Интерфейсная процедура для mymult
C     Описание типов аргументов
      integer nlhs, nrhs
      integer plhs(*), prhs(*)
C     Описание типов используемых функций из MatLab API
      integer mxGetM, mxGetN, mxGetPr, mxCreateFull
C     Описание типов указателей на используемые переменные
      integer A_pr, B_pr, C_pr
C     Переменные m, n, p, q используются для записи размеров матриц
      integer m, n, p, q
C     Предполагается, что матрицы не могут иметь размер, больший 50
      real*8  A(50,50), B(50,50), C(50,50)
C     Получение размеров матриц — входных аргументов MEX-функции
      m = mxGetM(prhs(1))
      n = mxGetN(prhs(1))
      p = mxGetM(prhs(2))
      q = mxGetN(prhs(2))
```

```
C   Проверка соответствующих размеров матриц
      if (n.ne.p) then
         call mexErrMsgTxt('Matrix dimensions must argee!')
      endif
C   Проверка размеров матриц на выход за допустимые пределы (<=50)
      if ((m.gt.50).or.(n.gt.50).or.(p.gt.50).or.(q.gt.50)) then
         call mexErrMsgTxt('Size of martix > 50')
      endif
C   Запись указателей на матрицы в соответствующие переменные A_pr и B_pr
      A_pr = mxGetPr(prhs(1))
      B_pr = mxGetPr(prhs(2))
C   Занесение данных из матриц в массивы A и B
      call mxCopyPtrToReal8(A_pr, A, m*n)
      call mxCopyPtrToReal8(B_pr, B, p*q)
C   Вызов процедуры multmtr, которая заносит в массив C произведение A*B
      call multmtr(m, n, q, A, B, C)
C   Создание выходного аргумента — вещественного массива размера m на q
C   при помощи функции MatLab API, выходной аргумент является указателем
C   на массив
      plhs(1) = mxCreateFull(m, q, 0)
      C_pr = mxGetPr(plhs(1))
C   Копирование значений C в массив с указателем C_pr
      call mxCopyReal8ToPtr(C, C_pr, m*q)
      end
```

Сгенерируйте MEX-функцию `mymult`, поместив файлы mymult.f и mymultg.f в текущий каталог MatLab, и проверьте ее работу:

```
>> mex mymult.f mymultg.f
>> A=[1 2 3; 4 5 6; 7 8 9];
>> B=[1 1 0 2; 3 -1 -2 0; 5 -2 -1 2];
>> C=mymult(A,B)
C =
    22    -7    -7     8
    49   -13   -16    20
    76   -19   -25    32
>> A*B
ans =
    22    -7    -7     8
    49   -13   -16    20
    76   -19   -25    32
```

Обратите внимание, что MEX-функция `mymult` вычисляет произведение не только матриц, но и чисел, поскольку числа в MatLab представляются в виде массивов размера один на один:

```
>> c=mymult(3,-7)
c =
   -21
```

Ограничение на размеры матриц является существенным недостатком интерфейсной процедуры `mexFunction`, текст которой содержится в листинге 21.17, поскольку процедура `multmtr` (см. листинг 21.16) перемножает матрицы без ограничения на их размеры. Объявление динамических массивов в интерфейсной процедуре, приведенной в листинге 21.18, позволяет решить эту проблему. Создайте MEX-файл, используя улучшенную процедуру, и проверьте его работу.

Листинг 21.18. Интерфейсная процедура с динамическими массивами

```
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
      integer plhs(*), prhs(*)
      integer mxGetM, mxGetN, mxGetPr, mxCreateFull
      integer A_pr, B_pr, C_pr
      integer m, n, p, q
C     Объявление вещественных динамических массивов A, B и C
      real*8, allocatable :: A(:,:),B(:,:), C(:,:)

      m = mxGetM(prhs(1))
      n = mxGetN(prhs(1))
      p = mxGetM(prhs(2))
      q = mxGetN(prhs(2))
      if (n.ne.p) then
         call mexErrMsgTxt('Matrix dimensions must argee!')
      endif
      if ((m.gt.50).or.(n.gt.50).or.(p.gt.50).or.(q.gt.50)) then
         call mexErrMsgTxt('Size of martix > 50')
      endif
C     Выделение памяти для записи значений первого входного
C     аргумента MEX-функции в массив A
      allocate(A(m,n))
      A_pr = mxGetPr(prhs(1))
      call mxCopyPtrToReal8(A_pr, A, m*n)
```

```
C     Выделение памяти для записи значений второго входного
C     аргумента MEX-функции в массив B
      allocate(B(p,q))
      B_pr = mxGetPr(prhs(2))
      call mxCopyPtrToReal8(B_pr, B, p*q)
C     Выделение памяти под результат процедуры multmtr
      allocate(C(m,q))
      call multmtr(m, n, q, A, B, C)
C     Высвобождение памяти, занятой под массивы A и B
      deallocate(A,B)
      plhs(1) = mxCreateFull(m, q, 0)
      C_pr = mxGetPr(plhs(1))
      call mxCopyReal8ToPtr(C, C_pr, m*q)
C     Высвобождение памяти, занятой под массив C
      deallocate(C)
      end
```



Часть VI

Приложения

Приложение 1. Основные команды и функции MatLab и ToolBox

Приложение 2. Описание дискеты

Приложение 1



Основные команды и функции MatLab и ToolBox

Приложение содержит основные функции и команды MatLab, сгруппированные по разделам. Приведено описание различных вариантов вызова функций, имеются примеры использования некоторых функций и команд. Даны ссылки на соответствующие главы и разделы книги, в которых подробно разобрано применение функций и команд MatLab.

Управление средой, файлами и переменными

Получение справочной информации

- **doc** — запуск справочной системы по всем разделам MatLab. В версии 5.3 HelpDesk HTML открывается в браузере Web-страниц (например, Internet Explorer). В MatLab 6.x отображается справочная система в отдельном окне **Help**. В качестве параметра может быть указано имя функции или команды, для которой требуется получить справочную информацию:

```
doc plot
```

- **help** — вывод разделов встроенной справки в командное окно. Информацию о разделе можно получить, добавив его имя в качестве параметра

```
help elfun
```

  Указание имени функции или команды после `help` приводит к отображению информации о них в командном окне

```
help max
```

  В тексте названия функций и команд приведено заглавными буквами, но при работе надо набирать команды и функции строчными буквами.

- **helpwin** — запуск краткой справочной системы по разделам MatLab в окне **MatLab Help Window** в версии 5.3, в MatLab 6.*x* аналогичная информация появляется в окне **Help**.
- **ver** — вывод в командное окно версий MatLab, ToolBox и дополнительных пакетов, входящих в установленную версию.
- **version** — вывод в командное окно версии MatLab.

- **whatsnew** — вывод информации о новшествах, отличающих данную версию от предыдущей:
  - `whatsnew matlab` — обновления, касающиеся MatLab;
  - `whatsnew pde` — новые возможности ToolBox PDE.

Управление средой MatLab

- **addpath** — добавление каталога в пути поиска MatLab. Примеры:
  - `addpath c:\user\igor\mywork`
  - `addpath c:\user\igor\mywork1 c:\user\igor\mywork2`
  - `addpath(c:\user\igor\mywork1, c:\user\igor\mywork2)`
  - `addpath c:\user\igor\mywork1 c:\user\igor\mywork2 -end` — добавление в конец списка;
  - `addpath c:\user\igor\mywork1 c:\user\igor\mywork2 -begin` — добавление в начало списка.

Подробнее про пути поиска написано в разд. "Типы М-файлов" главы 5.

- **clc** — очистка содержимого командного окна.
- **echo** — управление отображением строк выполняемого М-файла в командном окне.
  - `echo on` — вывод строк.
  - `echo off` — подавление вывода строк.
- **format** — определение формата вывода чисел в командное окно.
  - `format short` — формат с плавающей точкой с 5 цифрами после десятичной точки.
  - `format long` — формат с плавающей точкой с 15 цифрами после десятичной точки.
  - `format short e` — экспоненциальный формат с 5 значащими цифрами.
  - `format long e` — экспоненциальный формат с 15 значащими цифрами.
  - `format short g` — наиболее подходящий из экспоненциального формата и формата с плавающей точкой с 5 цифрами.
  - `format long g` — наиболее подходящий из экспоненциального формата и формата с плавающей точкой с 15 цифрами.
  - `format hex` — шестнадцатеричный формат.
  - `format +` — положительные, отрицательные или нулевые числа отображаются знаками плюс, минус или пробел, мнимая часть игнорируется.

- `format bank` — формат с двумя десятичными знаками.
- `format rat` — числа приближенно представляются отношением двух целых чисел.
- `format compact` — подавление вывода пустых строк между строками результата.
- `format loose` — вывод пустых строк между строками результата.

См. разд. "Форматы вывода результата вычислений" главы 1.

❒ **`path`** — управление путями поиска. Вызов `path` без аргументов приводит к выводу всех путей в командное окно.

- `p = path` — занесение в строковую переменную `p` путей поиска.
- `path(pathdef)` — установка путей, принятых по умолчанию.

Возможно, также, добавление путей в начало и конец списка путей поиска:

- `path('c:\user\igor\mywork', path)` — добавление в начало списка;
- `path(path, 'c:\user\igor\mywork')` — добавление в конец списка.

❒ **`rmpath`** — удаление одного или нескольких каталогов из путей поиска:

- `rmpath c:\user\igor\mywork`
- `rmpath c:\user\igor\mywork1 c:\user\igor\mywork2`
- `rmpath(c:\user\igor\mywork1, c:\user\igor\mywork2)`

❒ **`pathtool`** — запуск диалогового окна навигатора путей.

Работа с навигатором путей описана в разд. "Установка путей в версии 5.3" и "Установка путей в версии 6.x" главы 5.

❒ **`lasterr`**, **`lastwarn`** — возвращают строку с последним сообщением об ошибке или предупреждением. Вызовы `lasterr('')` и `lastwarn('')` приводят к очистке последнего сообщения об ошибке или предупреждении.

❒ **`profile`** — управление профайлером.

Команда `profile on` приводит к запуску профайлера. Общий вид команды для старта профайлера:

```
profile on -detail level -history
```

Значения `level`: `mmex` (сбор статистики временных затрат на выполнение M- и MEX-файлов), `builtin` (то же, что `mmex`, дополнительно находятся временные затраты на выполнение встроенных функций), `operator` (аналогично `builtin`, но с учетом времени на арифметические и другие встроенные операции).

Опция `-history` служит для запоминания последовательности вызова функций.

- `profile report` — остановка профайлера, генерация отчета в формате HTML и отображение его в браузере.

- `profile report myreport` — остановка профайлера, генерация отчета в myreport.html и нескольких связанных с ним файлах и отображение отчета в браузере.
- `profile plot` — остановка профайлера и построение столбцевой диаграммы временных затрат.
- `profile off` — прекращение сбора информации.
- `profile resume` — возобновление процесса сбора информации.
- `profile clear` — уничтожение накопленной профайлером информации.
- `profile status` — отображение структуры с установками профайлера.

Основы работы с профайлером описаны в разд. "Профайлер" главы 16.

❒ **`profreport`** — остановка профайлера, генерация отчета в формате HTML и отображение его в браузере.

`profreport(myreport)` — остановка профайлера, генерация отчета в myreport.html и нескольких связанных с ним файлах и отображение отчета в браузере.

❒ **`type`** — вывод содержимого текстового файла в командное окно

```
type filename
```

❒ **`quit`** — окончание сеанса MatLab.

❒ **`what`** — вывод содержимого текущего каталога (файлов с расширениями m, mat, mex) в командное окно.

- `what dirname` — вывод содержимого каталога dirname (файлов с расширениями m, mat, mex) в командное окно.
- `w = what('dirname')` — запись информации о каталоге dirname и его содержимом в структуру `w`.

❒ **`which`** — вывод пути к функции, например:

- `>> which fzero`

  `C:\MATLABR11\toolbox\matlab\funfun\fzero.m`
- `which funname -all` — отображение путей ко всем функциям с именем `funname`;
- `w=which('funname')` — занесение в строковую переменную `w` пути к функции с именем `funname`;
- `w=which('funname', '-all')` — занесение в массив ячеек `w` путей к функциям с именем `funname`.

Управление переменными

- **clear** — удаление переменных рабочей среды.
  - `clear A m R` — удаление переменных `A`, `m` и `R` из рабочей среды.
  - `clear global` — удаление всех глобальных переменных.
  - `clear functions` — удаление всех откомпилированных M-функций.
  - `clear mex` — удаление ссылок на MEX-файлы и M-функции.
  - `clear all` — удаление переменных и ссылок на MEX-файлы и M-функции.
- **disp** — вывод текста или значения переменной в командное окно.

  См. разд. "Проверка входных аргументов" главы 7.
- **length** — нахождение длины вектор-строки или вектор-столбца.

  См. разд. "Ввод, сложение и вычитание векторов" главы 2.
- **load** — считывание значений переменных с диска.

  См. разд. "Сохранение рабочей среды" главы 1, "Считывание и запись данных" главы 2.
- **mlock** — предотвращение удаления переменных работающего в данный момент M-файла командой `clear`.
- **munlock** — разрешение удаления переменных работающего в данный момент M-файла командой `clear`.
- **openvar** — запуск редактора M-файлов со специальным окном, позволяющим интерактивное редактирование значений переменных.

  `openvar A` — редактирование переменной `A`.
- **pack** — создание виртуальной памяти на диске в файле pack.tmp, в которую заносятся значения всех имеющихся переменных и очистка рабочей среды. Значения используемых переменных считываются из pack.tmp. Используется при обработке большого объема данных.
- **save** — запись значений переменных на диск.

  См. разд. "Сохранение рабочей среды" главы 1, "Считывание и запись данных" главы 2.
- **saveas** — сохранение графического окна с указателем `h` в файле определенного формата.
  - `saveas(h,'filename.ai')` — формат Adobe Illustrator.
  - `saveas(h,'filename.bmp')`
  - `saveas(h,'filename.emf')`

- `saveas(h,'filename.eps')`
- `saveas(h,'filename.fig')` — графическое окно MatLab.
- `saveas(h,'filename.jpg')`
- `saveas(h,'filename.m')` — M-файл MatLab.
- `saveas(h,'filename.pcx')`
- `saveas(h,'filename.tif')`

❒ **`size`** — определение размеров массива.

См., например, разд. "Ввод, сложение и вычитание векторов" и "Блочные матрицы" главы 2.

❒ **`who`** — вывод имен всех определенных в рабочей среде переменных.

❒ **`whos`** — получение информации о переменных рабочей среды.

См. разд. "Просмотр переменных" главы 1 и "Ввод, сложение и вычитание векторов" главы 2.

❒ **`workspace`** — вызов браузера рабочей среды MatLab. Браузер рабочей среды является приложением с графическим интерфейсом, которое позволяет просматривать и изменять значения переменных рабочей среды.

Манипулирование файлами и каталогами

❒ **`cd`** — установка текущего каталога, например

- `cd c:\user\igor\mywork1`
- `cd` — вывод текущего каталога в командное окно.
- `cd ..` — переход на один уровень вверх.

❒ **`copyfile`** — копирование файла.

- `copyfile('c:\mywork1\test.m','c:\matlab\work\myplot.m')` — копирование файла c:\mywork1\test.m в c:\matlab\work\myplot.m.
- `copyfile('c:\mywork1\test.m','c:\matlab\work\myplot.m', 'writable')` — запись файла c:\mywork1\test.m в c:\matlab\work\myplot.m с проверкой возможности копирования.
- `s=copyfile('c:\mywork1\test.m','c:\matlab\work\myplot.m')` — возвращает единицу, если копирование прошло успешно, и ноль — в противном случае.
- `[s,mes]=copyfile('c:\mywork1\test.m','c:\matlab\work\myplot.m')` — дополнительно заносит в `mes` строку с сообщением об ошибке, если таковая возникла в процессе копирования.

- **delete** — удаление файлов и графических объектов.
  - `delete c:\mywork1\test.m` — удаление файла c:\mywork1\test.m.
  - `delete('c:\mywork1\test.m')` — удаление файла c:\mywork1\test.m, такую форму вызова `delete` удобно использовать, когда имя файла хранится в строковой переменной.
  - `delete(h)` — удаление графического объекта с указателем `h`.
- **diary** — ведение дневника сеанса работы в MatLab.

  См. разд. "Сохранение рабочей среды" главы 1.
- **dir** — вывод содержимого текущего каталога.
  - `dir dirname` — вывод содержимого каталога dirname.
  - `DIRMAS=dir('dirmame')` — запись содержимого каталога dirname в массив структур `DIRMAS`. Каждая структура имеет четыре поля: `name` — имя файла или подкаталога, `date` — дата изменения, `bytes` — размер, флаг `isdir` — принимает значение единица для подкаталога и ноль для файла.
- **edit** — запуск редактора M-файлов, в котором создается новый файл.
  - `edit filename` — открытие в редакторе M-файлов файла filename.m.
  - `fileparts` — возвращает путь к файлу, имя и расширение, пример:

    ```
    [path,name,ext] = fileparts('имя файла')
    ```
- **fullfile** — конструирование полного имени файла из иерархии подкаталогов и имени файла, результат заносится в строковую переменную, пример:

  ```
  fn=fullfile('c:','user','igor','work','myfun.m')
  ```
- **inmem** — отображает функции MatLab, загруженные в память.
  - `MMAS=inmem` — занесение в массив ячеек `MMAS` имен загруженных M-функций.
  - `[MMAS, MXMAS]=inmem` — занесение в массивы ячеек `MMAS` и `MXMAS` имен загруженных M- и MEX-функций.
- **matlabroot** — имя каталога, в который установлена программа MatLab.
- **mkdir** — создание подкаталога, примеры:
  - `mkdir('work2')` — создание подкаталога work2 в текущем каталоге;
  - `mkdir('work','work2')` — создание подкаталога work2 в существующем на диске каталоге work;

- `[stat,msg]=mkdir('work2')` — создание подкаталога, `stat` равен единице в случае успешного выполнения функции, двойке, если подкаталог существует и нулю при невозможности создания подкаталога;
- `[stat,msg]=mkdir('work','work2')` — строка `msg` содержит сообщение об ошибке в случае неудачной попытки создания каталога.

❐ **open** — открытие файла или содержимого переменной, способ открытия зависит от расширения.

- `open('A')` — запуск редактора M-файлов и отображение в нем значений массива `A`. Возможно редактирование значений элементов.
- `open('my.fig')` — открытие графического окна my.fig.
- `open('myfun.m')` — запуск редактора M-файлов и открытие в нем myfun.m.
- `open('mymod.mdl')` — открытие модели mymod.mdl в среде Simulink.

Пользователь имеет возможность определить способ открытия файлов с другими расширениями, создав подходящую файл-функцию. Например, файлы с расширениями my1 требуют наличия функции `openmy1`.

❐ **pwd** — отображение имени текущего каталога, пример:

`s=pwd` — запись в строковую переменную `s` имени текущего каталога.

❐ **tempdir** — отображение имени каталога, предназначенного для хранения временных файлов, пример:

`s=tempdir` — запись в строковую переменную имени каталога, предназначенного для хранения временных файлов.

❐ **tempname** — отображение имени текущего временного файла, пример:

`s=tempname` — запись в строковую переменную имени текущего временного файла.

❐ **!** — выполнение команды операционной системы, например:

`!help` — вывод разделов справочной системы Windows.

Операторы и специальные символы

Использованию арифметических и логических операций, в том числе и применению их к массивам, посвящены отдельные разделы книги.

См. главы 1, 2 и 7.

Арифметические и матричные операции приведены в табл. П1.

Таблица П1. *Арифметические и матричные операции*

Операция	Назначение
`+, -`	Унарные плюс и минус, сложение и вычитание для скаляров и массивов. Массивы должны быть одинаковых размеров. Один из операндов может быть скаляром
`*`	Перемножение скаляров или матриц подходящих размеров. Один из операндов может быть скаляром
`kron`	Кронекеровское или тензорное произведение матриц `C=kron(A,B)`
`/`	Деление скаляров. Поэлементное деление матрицы на скаляр. Если оба операнда — матрицы, то `A/B=A*inv(B)`
`^`	Возведение скаляра в степень. Вычисление степени квадратной матрицы
`\`	Левое матричное деление. Если `A` является квадратной матрицей, то `A\B=inv(A)*B`. Если `A` — квадратная матрица размера `n`, `B` — вектор-столбец из `n` элементов, то `X=A\B` содержит решение системы линейных уравнений `AX=B`. Допускаются переопределенные и недоопределенные системы. *См. разд. "Задачи линейной алгебры" главы 6 и "Решение систем уравнений и исследование спектра" главы 16*
`.*`	Поэлементное умножение массивов одинаковых размеров, например `C=A.*B` приводит к `C(i,j)=A(i,j)*B(i,j)`
`./`	Поэлементное деление массивов одинаковых размеров, например, `C=A./B` приводит к `C(i,j)=A(i,j)/B(i,j)`
`.\`	Поэлементное левое деление массивов одинаковых размеров, например, `C=A.\B` приводит к `C(i,j)=B(i,j)/A(i,j)`
`.^`	Поэлементное возведение матрицы в степени, являющиеся элементами другой матрицы тех же размеров, например, `C=A.^B` приводит к `C(i,j)=A(i,j)^B(i,j)`
`'`	Нахождение сопряженной матрицы
`.'`	Транспонирование матрицы. Для вещественных матриц `'` и `.'` приводят к одинаковым результатам

Логические операции и операторы

Логические операции применимы к массивам одинаковых размеров или к массиву и скаляру, в последнем случае скаляр расширяется до размеров массива.

См. разд. "Логические выражения с массивами и числами" главы 7.

Операторы отношения представлены в табл. П2.

Таблица П2. *Операторы отношения*

Операция	Назначение
>	Больше: `a>b`
<	Меньше: `a<b`
==	Равно: `a==b`
~=	Не равно: `a~=b`

Логические операторы имеют некоторые особенности по сравнению со многими языками программирования. Логические операции (табл. П3) могут применяться к массивам.

См. разд. "Логические выражения с массивами и числами" главы 7.

Таблица П3. *Логические операции*

Оператор	Назначение
`&`	Логическое И: `(a>b)&(a<c)`
`|`	Логическое ИЛИ: `(a>b)|(a<c)`
`xor`	Логическое исключающее ИЛИ: `xor(a>b,a==c)`
`~`	Логическое отрицание: `~(a==0)`

Побитовые операции

- **`bitand`** — поразрядное И.

  `c=bitand(a,b)` — возвращает результат побитового И для двух целых неотрицательных чисел, меньших `bitmax` (см. `bitmax` ниже), например:

```
>> c=bitand(798,336)
c =
  272
```

  Функция `dec2bit` позволяет убедиться в правильности полученного результата.

  См. разд. "Преобразование системы счисления" приложения 1 ниже.

  Пример использования поразрядного И:

```
>> dec2bin(798)
ans =
1100011110
```

```
>> dec2bin(336)
ans =
101010000
>> dec2bin(272)
ans =
100010000
```

❒ **bitcmp** — поразрядное дополнение.

`c=bitcmp(a,n)` — возвращает поразрядное дополнение целого неотрицательного числа `a`, состоящее из `n` разрядов, например:

```
>> bitcmp(598,10)
ans =
  425
>> dec2bin(598)
ans =
1001010110
>> bitcmp(598,10)
ans =
  425
>> dec2bin(425)
ans =
110101001
```

❒ **bitor** — поразрядное ИЛИ.

`c=bitor(a,b)` — возвращает результат побитового ИЛИ для двух целых неотрицательных чисел, меньших `bitmax` (см. `bitmax` ниже), например:

```
>> dec2bin(24)
ans =
11000
>> dec2bin(89)
ans =
1011001
>> c=bitor(24,89)
c =
  89
>> dec2bin(c)
ans =
1011001
```

- **bitmax** — возвращает максимально допустимое целое без знака.
- **bitset** — установка разряда.
  - `c=bitset(a,bit)` или `c=bitset(a,bit,1)` — установка в единицу двоичного разряда с номером `bit` (не более 52) целого неотрицательного числа `a`, например:

```
>> dec2bin(257)
ans =
100000001
>> c=bitset(257,3)
c =
  261
>> dec2bin(261)
ans =
100000101
```

  - `c=bitset(a,bit,0)` — установка в ноль двоичного разряда с номером `bit` целого неотрицательного числа `a`.
- **bitshift** — сдвиг разрядов.
  - `c=bitshift(a,k,n)` — результат является поразрядным сдвигом целого неотрицательного числа `a` (не превосходящего `bitmax`) на `k` битов. Если выходной аргумент представляется числом битов, большим `n`, то происходит отбрасывание лишних разрядов.
  - `c=bitshift(a,k)` — эквивалентно `c=bitshift(a,k,53)`.

  Положительные значения `k` приводят к сдвигу влево, а отрицательные — вправо. Пример вызова `bitshift`:

```
>> c=bitshift(272,5)
c =
    8704
>> dec2bin(272)
ans =
100010000
>> dec2bin(8704)
ans =
10001000000000
```

- **bitget** — получение значения разряда.

  `val=bitget(a,bit)` — в выходном аргументе `val` возвращается значение (ноль или единица) разряда с номером `bit` (не более 52) целого неотрицательного числа `a`.

❒ **bitxor** — поразрядное исключающее ИЛИ.

`c=bitxor(a,b)` — возвращает результат исключающего побитового ИЛИ для двух целых неотрицательных чисел, меньших `bitmax` (см. `bitmax` выше), например:

```
>> dec2bin(139)
ans =
10001011
>> dec2bin(116)
ans =
1110100
>> c=bitxor(139,116)
c =
  255
>> dec2bin(255)
ans =
11111111
```

В табл. П4 приведены специальные символы, использующиеся в выражениях MatLab.

Таблица П4. *Специальные символы*

Символы	Назначение
=	Оператор присваивания

[]	Квадратные скобки используются для формирования вектор-строк, вектор-столбцов и массивов, например:
	`a=[1 2 3]; b = [1+2i, 3-9i]; c=[0.2; -3; -4; 8];`
	`A=[1 2 3; 4 5 6; 7 8 9];`
	Конструирование блочных матриц так же производится при помощи квадратных скобок:
	`M=[A B; C D];`
	Пустые квадратные скобки используются для определения пустого массива и удаления строк или столбцов: `A(2,:)=[];`
	Квадратные скобки позволяют вызвать функцию с несколькими выходными аргументами: `[m,k]=max(x)`

{}	Фигурные скобки предназначены для заполнения массивов ячеек. *См. разд. "Массивы ячеек" главы 8.*
()	Круглые скобки определяют порядок выполнения арифметических и логических операций. Кроме того, индексы массивов и входные аргументы функций заключаются в круглые скобки

Таблица П4 *(окончание)*

Символы	Назначение
:	Двоеточие позволяет обратиться к сечению массива: `B=A(2:5, 4:7)` и создать вектор, компоненты которого изменяются с постоянным шагом: `a=[-1:0.05:2]`
.	Десятичная точка, отделение поля структуры от имени
..	Переход на один каталог выше в команде `cd`
...	Продолжение команды на следующей строке. Используется как при наборе в командной строке, так и в редакторе М-файлов
,	Запятой отделяются индексы массива и аргументы функций. Несколько команд, набранных в одной строке, так же отделяются запятой, например, `a=1,c=2`
;	Точка с запятой отделяет строки матрицы при наборе элементов внутри квадратных скобок. Завершение выражения точкой с запятой приводит к подавлению вывода результата в командное окно
%	Начало комментария в М-файле

Логические функции

- **`all`** — проверка на наличие нулевого элемента в массиве.

  `f=all(A)` — возвращает логическую единицу, если в массиве `A` все элементы ненулевые, и ноль, если хотя бы один элемент массива равен нулю.

- **`any`** — проверка на наличие ненулевого элемента в массиве.

  `f=any(A)` — возвращает логическую единицу, если в массиве `A` есть хотя бы один ненулевой элемент, и ноль, если все элементы массива равны нулю.

- **`exist`** — проверка существования переменной или файла.

  `a=exist('name')` — возвращает тип проверяемого объекта:

  0, если `name` не существует;

  1, если `name` является переменной рабочей среды;

  2, если `name` — имя М-файла из каталога, находящегося в путях поиска, или тип файла неизвестен;

  3, если в каталоге, находящемся в путях поиска, есть файл name.mex;

  4, если в каталоге, находящемся в путях поиска, есть файл name.mdl;

  5, если `name` является именем встроенной функции MatLab;

6, если существует P-файл с именем `name` в каталоге, имеющемся в путях поиска;

7, если `name` является именем каталога.

❒ **`find`** — нахождение индексов и значений ненулевых элементов массива.

- `k=find(x)` — в вектор *k* заносятся номера ненулевых элементов массива `x`. Если `x` является матрицей, то она трактуется как вектор, составленный из ее столбцов.
- `[i,j]=find(x)` — в вектора *i* и *j* записываются индексы ненулевых элементов матрицы *x*, что удобно, например, при работе с разреженными матрицами.

См. разд. "Операции с разреженными матрицами" главы 16.

- `[i,j,v]=find(x)` — в дополнительном выходном аргументе `v` возвращаются значения ненулевых элементов матрицы *x*.

❒ **`is...`** — выявление типа и значений переменной.

- `k=iscell(C)` — возвращает логическую единицу, если `C` — массив ячеек, и ноль — в противном случае.
- `k=iscellstr(S)` — возвращает логическую единицу, если `S` — массив ячеек строк, и ноль — в противном случае.
- `k=ischar(S)` — возвращает логическую единицу, если `S` — массив символов, и ноль — в противном случае.
- `k=isempty(A)` — возвращает логическую единицу, если `A` — пустой массив, и ноль — в противном случае. Пустым считается массив, у которого хотя бы один размер равен нулю.
- `k=isequal(A,B,...)` — возвращает логическую единицу, если входные массивы одинаковы (то есть одних размеров и соответствующие элементы совпадают), и ноль — в противном случае.
- `k=isfield(S,'field')` — возвращает логическую единицу, если `field` является одним из полей структуры `S`, и ноль — в противном случае.
- `TF=isfinite(A)` — возвращает массив `TF`, в котором логические единицы соответствуют числам массива `A`, а нули — `Inf`, `-Inf` или `NaN` в `A`.
- `k = isglobal(name)` — возвращает логическую единицу, если `name` объявлена как глобальная переменная, и ноль — в противном случае.
- `TF = ishandle(H)` — возвращает массив `TF`, в котором логические единицы соответствуют элементам массива `H`, которые являются указателями на существующие графические объекты. Остальные элементы `TF` нулевые.

- `k = ishold` — возвращает логическую единицу, если `hold` установлено в `on`, т. е. при выводе графиков происходит их добавление в текущее окно, ноль соответствует `hold off`.
- `TF = isinf(A)` — возвращает массив `TF`, в котором логические единицы соответствуют элементам `Inf`, `-Inf` массива `A`, а нули — остальным значениям.
- `TF = isletter('str')` — возвращает массив `TF`, в котором логические единицы соответствуют символам алфавита в строке `str`, а нули — остальным значениям.
- `k = islogical(A)` — возвращает логическую единицу, если `A` — логический массив, и ноль — в противном случае.
- `TF = isnan(A)` — возвращает массив `TF`, в котором логические единицы соответствуют элементам `NaN` массива `A`, а нули — остальным значениям.
- `k = isnumeric(A)` — возвращает логическую единицу, если `A` — числовой массив (то есть `double array` или `sparse array`), и ноль — в противном случае.
- `k = isobject(A)` — возвращает логическую единицу, если `A` является объектом, и ноль — в противном случае.
- `TF = isprime(A)` — возвращает массив `TF`, в котором логические единицы соответствуют простым числам (не имеющим делителя, кроме единицы и самого числа) массива `A`, а нули — остальным значениям.
- `k = isreal(A)` — возвращает логическую единицу, если все элементы `A` являются вещественными числами, и ноль — в противном случае. Поскольку строковые переменные входят в подкласс `double array`, то для строк `isreal` возвращает логическую единицу.
- `TF = isspace('str')` — возвращает массив `TF`, в котором логические единицы соответствуют пробелам, символам табуляции и пустой строки в `str`, а нули — остальным значениям.
- `k = issparse(S)` — возвращает логическую единицу, если `S` является разреженной матрицей, т. е. массивом типа `sparse array`, и ноль — в противном случае.
- `k = isstruct(S)` — возвращает логическую единицу, если `S` является структурой, и ноль — в противном случае.

❒ **`isa`** — определение принадлежности объекта классу.

- `isa(obj,'class_name')` — возвращает логическую единицу, если `obj` есть объект класса `class_name`, и ноль — в противном случае.

Возможны следующие варианты вызова:

- `isa(obj,'double')`, `isa(obj,'sparse')`, `isa(obj,'struct')`, `isa(obj,'cell')`, `isa(obj,'char')`, `isa(obj,'uint8')`, `isa(obj,'класс пользователя')`.

❒ **`logical`** — преобразование числового массива в логический, который может быть использован для индексации.

См. разд. "Логическое индексирование" главы 7.

❒ **`mislocked`** — проверка на возможность удаления из рабочей среды переменных M-файла.

- `k=mislocked` — возвращает логическую единицу, если можно удалить переменные выполняемого в данный момент M-файла, и ноль — в противном случае.
- `k=mislocked('filename')` — производит аналогичную проверку для M-файла с именем filename.

Программирование

Конструкции языка

Программированию алгоритмов на встроенном языке MatLab посвящены две главы книги.

См. главы 7 и 8.

Ниже приведены все конструкции языка программирования MatLab, предназначенные для определения последовательности выполняемых команд.

❒ **`break`** — выход из циклов `while` и `for`.

❒ **`case`** — начало блока в операторе переключения `switch`.

❒ **`catch`** — начало блока конструкции `try...catch`, соответствующего исключительной ситуации.

❒ **`else`** — ветвь оператора `if`, работающая при невыполнении всех условий.

❒ **`elseif`** — ветвь оператора `if`, работающая при выполнении некоторого условия.

❒ **`end`** — завершение конструкций `for`, `while`, `switch`, `try` и `if`.

❒ **`error`** — отображение в командное окно сообщения об ошибке и прекращение работы файл-функции или файл-программы, пример:
`error('ошибка ввода')`

❒ **`for`** — оператор для организации цикла с известным числом повторов.

- **function** — объявление файл-функции или подфункции.

  См. разд. "Файл-функции" главы 5 и разд. "Подфункции" главы 8.

- **global** — раздел объявления глобальных переменных в файл-функции.
- **if** — условный оператор.
- **otherwise** — начало блока оператора переключения `switch`, выполняющегося в случае, когда ни один из блоков `case` не был выполнен.
- **persistent** — раздел объявления констант в файл-функции.
- **return** — возврат в точку вызова функции, или прекращение режима ввода с клавиатуры.
- **switch** — оператор переключения.
- **try** — начало конструкции обработки исключительных ситуаций.
- **warning** — вывод предупреждения в командное окно, например:

  ```
  warning('деление на ноль')
  ```

- **while** — организация цикла с неизвестным числом повторений, выполняющегося при истинности условия цикла.

Сервисные функции и переменные

- **ans** — автоматически создаваемая переменная для хранения значения выражения или результата функции, если не применяется оператор присваивания.
- **builtin** — вызов исходной функции в файл-функции, определяющей перегруженный метод. Работает так же, как и `feval`, но обращается к исходному, а не перегруженному методу. Используется при создании методов класса. Общий вид обращения:

  ```
  [y1,...,yn]=builtin('fun',x1,...,xk)
  ```

- **computer** — вывод информации о компьютере.
  - `comp=computer` — в строковую переменную заносится `PCWIN`, если компьютер работает под управлением операционной системы Windows.
  - `[comp, maxsize]=computer` — дополнительный выходной аргумент `maxsize` содержит максимально допустимое число элементов массива в данной версии MatLab.
- **eps** — точность вычислений по умолчанию при использовании арифметических операций и большинства функций. Некоторые функции по умолчанию находят результат с меньшей точностью.

  См., например, разд. "Минимизация функции одной переменной" главы 6.

- **eval** — выполнение содержимого строки или строковой переменной, как команды MatLab, например:

  ```
  eval('y=sin(x).*cos(x)')
  y=eval('sqrt(log(1.2))')
  ```

  `eval(s1,s2)` — выполнение содержимого строки `s1`, если в процессе выполнения возникает ошибка, то управление передается командам строки `s2` (аналог конструкции `try...catch`).

  См. разд. "Сервисные функции для работы со строками" главы 8.

- **evalc** — выполнение содержимого строки или строковой переменной, как команды MatLab, результат возвращается в выходном аргументе, который является символьным массивом, примеры:

  ```
  T=evalc(s), T=evalc(s1, s2)
  ```

- **evalin** — выполнение содержимого строки или строковой переменной, как команды MatLab, с использованием переменных рабочей среды или локальных переменных вызывающей функции.

  - `evalin('base', s)` — выполнение содержимого строковой переменной `s`, как команды MatLab, с использованием переменных рабочей среды.
  - `evalin('caller', s)` — выполнение содержимого строковой переменной `s`, как команды MatLab, с использованием локальных переменных вызывающей функции.

  Аналогично с функцией `eval` могут быть заданы две строки:

  ```
  evalin('base', s1, s2), evalin('caller', s1, s2)
  ```

- **feval** — вычисление функции, имя которой задано в строке или в строковой переменной, например `[m, k]=feval('max', x)`.

  См. разд. "Функции от функций" главы 8.

- **flops** — подсчет числа флопов (операций с числами с плавающей точкой).

  - `f=flops` — в переменную `f` заносится число выполненных к настоящему времени флопов.
  - `flops(0)` — обнуление счетчика флопов.

- **nargchk** — проверка количества входных аргументов, с которыми вызвана файл-функция. Используется внутри файл-функции.

  - `msg=nargchk(low, high, num)` — занесение в строковую переменную `msg` сообщения об ошибке, если `num` больше `high` или меньше `low`.

- **nargin** — определение количества входных аргументов, с которыми вызвана файл-функция.
  - `num=nargin` — в переменную `num` заносится число входных аргументов, с которыми была вызвана файл-функция. Используется внутри файл-функции.
  - `num=nargin('fun')` — в переменную `num` заносится число входных аргументов, определенное для файл-функции с именем `fun`. Если `fun` является файл-функцией с переменным числом входных аргументов, то возвращается отрицательное число.

  См. разд. "Условный оператор if" главы 7.

- **nargout** — определение количества выходных аргументов, с которыми вызвана файл-функция.
  - `num=nargout` — в переменную `num` заносится число выходных аргументов, с которыми была вызвана файл-функция. Используется внутри файл-функции.
  - `num=nargout('fun')` — в переменную `num` заносится число выходных аргументов, определенное для файл-функции с именем `fun`. Если `fun` является файл-функцией с переменным числом выходных аргументов, то возвращается отрицательное число.

  См. разд. "Условный оператор if" главы 7.

- **realmax** — возвращает максимально допустимое в MatLab вещественное положительное число.
- **realmin** — возвращает минимально допустимое в MatLab вещественное положительное число.
- **varargin** — определение переменного числа входных аргументов в заголовке файл-функции `varargin` является массивом ячеек, из которого следует извлекать аргументы для их использования в файл-функции.

  См. разд. "Файл-функции с переменным числом аргументов" и "Массивы ячеек" главы 8.

- **varargout** — определение переменного числа выходных аргументов в заголовке файл-функции `varargout` является массивом ячеек, в который следует занести аргументы для возвращения файл-функцией их значений.

  См. разд. "Файл-функции с переменным числом аргументов" и "Массивы ячеек" главы 8.

Интерактивный ввод

- **input** — запрос на ввод с клавиатуры. Используется при создании приложений с интерфейсом из командной строки.

  См. разд. "Приложения с интерфейсом из командной строки" главы 8.

  - `p=input('Введите значение p')` — ожидание ввода пользователем значения `p` с клавиатуры и занесение введенного значения в `p`.
  - `name=input('Введите Ваше имя','s')` — ожидание ввода пользователем строки с клавиатуры и занесение введенных символов в строковую переменную `name`.

  Вывод многострочного текста осуществляется при помощи `\n`.

  Команда `p=input('Выберите цвет:\n 1-синий \n 2-красный \n 3-зеленый \n')`

  приводит к появлению следующего текста в командном окне:

```
Выберите цвет:
 1-синий
 2-красный
 3-зеленый
```

- **keyboard** — передает управление клавиатуре, используется в М-файлах. Для продолжения работы М-файла следует набрать `return` в командной строке.
- **menu** — вывод диалогового окна с возможностью выбора. Входными аргументами являются строки или строковые переменные. Первый входной аргумент определяет заголовок окна, а остальные — названия кнопок. Команда

```
p=menu('Выбор цвета', 'синий', 'красный', 'зеленый')
```

приводит к появлению диалогового окна (рис. П1). Нажатие на кнопку **синий** заносит в `p` единицу и закрывает окно. Кнопки **красный** или **зелёный** соответствуют значениям два и три переменной `p`.

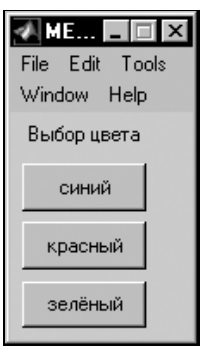


Рис. П1. Диалоговое окно выбора

- **pause** — приостанавливает выполнение M-файла.
  - `pause` — ожидание нажатия пользователем любой клавиши.
  - `pause(t)` — задержка выполнения команд на `t` секунд.

Объектно-ориентированное программирование и преобразование типов

- **class** — создание объекта или определение, какому классу принадлежит объект.
- **double** — преобразование к двойной точности.
- **inferiorto**, **superiorto** — задание иерархии классов и методов.
- **inline** — определение встраиваемой функции.
- **int8**, **int16**, **int32** — преобразование к целому числу со знаком.
- **isa** — проверка на принадлежность объекта классу. Использование аналогично функции `isobject`.
- **loadobj** — расширение функции `load` на объекты, определяемые пользователем.
- **saveobj** — расширение функции `save` на объекты, определяемые пользователем.
- **single** — преобразование к одинарной точности.
- **uint8**, **uint16**, **uint32** — преобразование к целому без знака.

Функции даты и времени

- **calendar** — календарь любого месяца года:
  - `calendar` — отображение календаря текущего месяца в командное окно в виде таблицы.
  - `calendar(y,m)` — отображение календаря месяца с номером `m` года `y` в командное окно в виде таблицы.
  - `C=calendar` — занесение дат текущего месяца в матрицу *C* размера шесть на семь.
  - `C=calendar(y,m)` — занесение дат месяца с номером `m` года `y` в матрицу *C* размера шесть на семь.
- **clock** — получение даты и времени:

  `c=clock` — занесение в вектор *c* даты и времени в формате `[год месяц день час минута секунда]`

- **cputime** — вычисляет процессорное время с момента предыдущего обращения к данной функции, пример использования:

  ```
  T=cputime;
  B=inv(hadamard(20));
  T=cputime-T
  ```

- **date** — получение текущей даты, пример:

  `str=date` — в строковую переменную `str` заносится текущая дата в формате `'дд-ммм-гггг'`.

- **datenum** — преобразование строки с датой в серийную дату (см. `now`).

- **datestr** — преобразование серийной даты (см. `now`) в строку.

- **datevec** — преобразование даты в формат `[год месяц день час минута секунда]`, использование:

  `c=datevec(d)`, `[y,m,d,h,mi,s]=datevec(d)` — входной аргумент может быть строкой с датой (результат `date`, `datestr`) или серийной датой (результат `num`, `datenum`). Задание массива дат во входном аргументе приводит к получению соответствующего преобразованного массива.

- **eomday** — завершающий день любого месяца года: `d=eomday(y,m)`.

- **etime** — вычисление количества секунд во временном интервале.

- **t=etime(t1,t0)** — возвращает время в секундах между `t1` и `t2`, которые представлены в формате `[год месяц день час минута секунда]`.

- **now** — получение текущей даты и времени.

  `d=now` — информация о текущей дате и времени кодируется вещественным числом (серийная дата).

- **tic**, **toc** — запуск секундомера и вывод времени, набираются в одной строке, пример:

  `>> tic, B=inv(hadamard(20)); toc`

- **weekday** — возвращает день недели.

  `[dnum,dname]=weekday(d)` — в `dnum` заносится порядковый номер дня недели (начиная с воскресенья), а в строковую переменную `dname` — аббревиатура дня. Входной аргумент задается так же, как и в `datevec`.

Двоичные и текстовые файлы

- **fclose** — закрытие файлов.

  - `status=fclose(fid)` — закрытие файла с идентификатором `fid`. Если файл закрыт успешно, то `staus=0`, если нет, то `staus=-1`.

- `status=fclose('all')` — закрытие всех файлов, кроме файлов с идентификаторами `0`, `1` и `2`.

❐ **`fopen`** — открытие файла и получение информации о файлах.

- `fid=fopen(filename,permission)` — открытие файла с именем `filename`, в переменную `fid` заносится идентификатор файла, который используется для указания на файл в других низкоуровневых функциях. Если файл не может быть открыт, то `fid=-1`. Аргумент `permission` означает способ доступа к файлу:
  - ◊ `'r'` — открытие двоичного файла для чтения;
  - ◊ `'r+'` — открытие двоичного файла для чтения и записи;
  - ◊ `'w'` — открытие нового двоичного файла для записи (если файл с таким же именем существует, то его содержимое будет удалено);
  - ◊ `'w+'` — открытие нового двоичного файла для чтения и записи (если файл с таким же именем существует, то его содержимое будет удалено);
  - ◊ `'a'` — создание нового двоичного файла или открытие существующего для записи, происходит добавление в конец файла;
  - ◊ `'a+'` — создание нового двоичного файла или открытие существующего для чтения и записи, происходит добавление в конец файла;
  - ◊ `'rt'` — открытие текстового файла для чтения;
  - ◊ `'rt+'` — открытие текстового файла для чтения и записи;
  - ◊ `'wt'` — открытие нового текстового файла для записи (если файл с таким же именем существует, то его содержимое будет удалено);
  - ◊ `'wt+'` — открытие нового текстового файла для чтения и записи (если файл с таким же именем существует, то его содержимое будет удалено);
  - ◊ `'at'` — создание нового текстового файла или открытие существующего для записи, происходит добавление в конец файла;
  - ◊ `'at+'` — создание нового текстового файла или открытие существующего для чтения и записи, происходит добавление в конец файла;
- `[fid,message]=fopen(filename,permission)` — если файл открыть не удалось `(fid=-1)`, то строковая переменная `message` содержит дополнительную информацию.

❐ **`fread`** — чтение двоичных файлов.

`[A,count]=fread(fid,size,precision)` — чтение двоичных данных из файла с идентификатором `fid` и запись их в матрицу *A*. Необязательный входной аргумент `size` задает размер матрицы, возможны значения:

`n` — чтение `n` элементов в вектор-столбец (`Inf` — до конца файла);

`[m n]` — чтение в матрицу *A* по столбцам, `size(A)=[m n]`, `n` может быть `Inf`.

Выходной аргумент `count` возвращает число считанных элементов. Тип считываемых данных определяется значением входного аргумента `precision`, который может принимать значения:

`'uchar'` или `'unsigned char'` — символ без знака, 8 битов;

`'schar'` или `'signed char'` — символ со знаком, 8 битов;

`'int8'` или `'integer*1'` — целое, 8 битов;

`'int16'` или `'integer*2'` — целое, 16 битов;

`'int32'` или `'integer*4'` — целое, 32 бита;

`'int64'` или `'integer*8'` — целое, 64 бита;

`'uint8'` или `'integer*1'` — целое без знака, 8 битов;

`'uint16'` или `'integer*2'` — целое без знака, 16 битов;

`'uint32'` или `'integer*4'` — целое без знака, 32 бита;

`'uint64'` или `'integer*8'` — целое без знака, 64 бита;

`'single'` или `'real*4'` или `'float32'` — вещественное с плавающей точкой, 32 бита;

`'double'` или `'float64'` или `'real*8'` — вещественное с плавающей точкой, 64 бита.

- **`fwrite`** — запись двоичных данных в файл.

  `count=fwrite(fid,A,precision)` — запись элементов матрицы *A* по столбцам в файл с идентификатором `fid`. Использование `precision` такое же, как в `fread`. Выходной аргумент возвращает количество записанных элементов.

- **`fgetl`** — получение следующей строки текстового файла без символа перевода строки.

  `line=fgetl(fid)` — возвращает следующую строку файла с идентификатором `fid` в строковой переменной `line`. Если достигнут конец файла, то выходной аргумент равен `-1`.

- **`fgets`** — получение следующей строки файла с символом перевода строки.

  `line=fgets(fid)` — возвращает следующую строку файла с идентификатором `fid` в строковой переменной `line`, которая завершается символом перевода строки. Если достигнут конец файла, то выходной аргумент равен `-1`.

- **`fprintf`** — форматный вывод в текстовый файл.

- **`fscanf`** — чтение данных из текстового файла, записанных в определенном формате.

  Подробная информация о форматной записи в файл с примерами использования содержится в разд. "Текстовые файлы" главы 8.

- **feof** — проверка достижения конца файла.

  `feof(fid)` — возвращает единицу, если обнаружен конец файла, и ноль — в противном случае.

- **ferror** — получение сведений об ошибках при работе с файлами.

  - `message=ferror(fid)` — возвращает последнюю возникшую ошибку ввода/вывода при работе с файлом, идентификатор которого `fid`.
  - `[message,errnum]=ferror(fid)` — дополнительный выходной аргумент `errnum` содержит номер ошибки.
  - `ferror(fid,'clear')` — очистка списка ошибок для файла с идентификатором `fid`.

- **frewind** — переход на начало файла.

  - `frewind(fid)` — установка текущей позиции файла с идентификатором `fid` на начало файла.

- **fseek** — установка текущей позиции в файле.

  `status=fseek(fid,offset,origin)` — перемещение текущей позиции в файле с идентификатором `fid` на `offset` байтов относительно `origin`.

  Допустимые значения `offset` и `origin`:

  - ◊ `offset>0` — передвижение к концу файла;
  - ◊ `offset=0` — текущая позиция не изменяется;
  - ◊ `offset<0` — передвижение к концу файла.
  - ◊ `origin='bof'` или `-1` — смещение на `offset` байтов от начала файла;
  - ◊ `origin='cof'` или `0` — смещение на `offset` байтов от текущей позиции;
  - ◊ `origin='eof'` или `1` — смещение на `offset` байтов от конца файла.

- **ftell** — получение текущей позиции в файле, пример:

  `position=ftell(fid).`

- **sprintf, sscanf** — форматная запись данных в строку и форматное чтение данных из строки.

  Использование `sprintf` и `sscanf` аналогично `fprintf` и `fscanf`, за исключением того, что результат помещается в строковую переменную, а не записывается в файл.

- **dlmread** — чтение числовых данных с разделителями из текстового файла в матрицу.

  - `M=dlmread(filename)` — чтение чисел из текстового файла и занесение их в матрицу *M*. Элементы строк матрицы в текстовом файле должны быть отделены друг от друга запятой, а сами строки — символом перевода строки.

- `M=dlmread(filename,dlm)` — чтение чисел из текстового файла и занесение их в матрицу *M*. Элементы строк матрицы в текстовом файле должны быть отделены друг от друга разделителем, указанным в `dlm`, например: `M=dlmread(filename,':')`, а сами строки — символом перевода строки. Если элементы строки матрицы в файле разделены табуляцией, то следует применить вызов `M=dlmread(filename,'\t')`.
- `M=dlmread(filename,dlm,nrow,ncol)` — чтение чисел из файла, начиная со строки с номером `nrow` и столбца `ncol`. Нумерация строк и столбцов в файле начинается с нуля.
- `M=dlmread(filename,dlm,rng)` — чтение прямоугольной области из файла в матрицу. Вектор `rng` задает область `rng=[rowstart colstart rowend colend]`. Возможно указание границ области в стиле Excel: `rng='A5..D4'`.

Замечание

Если в текстовом файле между разделителями пропущено число, то соответствующие элементы матрицы будут равны нулю.

- **`dlmwrite`** — запись содержимого матрицы в текстовый файл с разделителями.
  - `dlmwrite(filename,M)` — запись элементов матрицы *M* через запятую в текстовый файл с именем `filename`. Строки матрицы в файле отделяются символом перевода строки.
  - `dlmwrite(filename,M,dlm)` — запись элементов матрицы *M* через разделитель `dim` в текстовый файл, например `dlmwrite(filename,M,'#')`. Если требуется разделить табуляцией элементы строки матрицы в файле, то следует применить вызов `dlmwrite(filename,M,'\t')`.
  - `dlmwrite(filename,M,dlm,nrow,ncol)` — запись матрицы *M* в файл, начиная со строки `nrow` и столбца `ncol`. Нумерация строк и столбцов в файле начинается с нуля.

Замечание

Нулевые значения элементов матрицы пропускаются при записи в текстовый файл, соответствующие разделители добавляются для сохранения табличной структуры данных.

- **`textread`** — чтение данных из текстового файла, имеющего табличную структуру.
  - `[a,b,c,...]=textread(filename,format)` — чтение всех данных из файла с именем `filename` в массивы `a`, `b`, `c` и т. д. Число и тип пере-

менных определяется спецификаторами форматов, указанных в строковой переменной или строке `format`:

- `%d` — чтение целого со знаком, соответствующий выходной аргумент является вещественным массивом;
- `%nd` — чтение `n` цифр целого со знаком;
- `%u` — чтение целого числа, соответствующий выходной аргумент является вещественным массивом;
- `%nu` — чтение `n` цифр целого числа;
- `%f` — чтение вещественного числа, соответствующий выходной аргумент является вещественным массивом;
- `%nf` — чтение `n` цифр вещественного числа;
- `%n.pf` — чтение `n` цифр вещественного числа, из них `p` после десятичной точки;
- `%s` — чтение строки, отделенной пробелами, символами табуляции или перевода строки (размечающими символами), соответствующий выходной аргумент является массивом ячеек;
- `%ns` — чтение `n` символов строки;
- `%q` — то же, что и `%s`, но если строка заключена в кавычки, то в ячейки массива кавычки не заносятся;
- `%nq` — чтение `n` символов строки;
- `%c` — чтение символов, в том числе и пробелов, соответствующий выходной аргумент является массивом символов;
- `%nc` — чтение `n` символов строки;
- `%[...]` — чтение строки, которая содержит символы, заключенные в квадратных скобках, соответствующий выходной аргумент является массивом ячеек;
- `%[^...]` — чтение строки, которая не содержит символы, заключенные в квадратных скобках, соответствующий выходной аргумент является массивом ячеек;

Все вышеперечисленные спецификаторы могут использоваться и для пропуска соответствующей позиции при считывании данных, что достигается заменой символа `%` на `%*`.

Следующий пример демонстрирует использование спецификаторов формата при чтении из текстового файла staff.dat с табличной структурой, содержимое которого приведено ниже.

```
Sam 23 US 83.278 p231
Nike 24 UK 102.22 k18N
```

Требуется считать содержимое файла staff.dat: первое поле — в массив ячеек `names`, второе — в числовой массив `age`, третье поле необходимо пропустить, а четвертое и пятое занести в числовой массив `dat` и массив ячеек `code` соответственно. Следующее обращение к `textread` приводит к образованию массивов и заполнению их нужной информацией.

```
>> [names,age,dat,code] = textread('staff.dat','%s %d %*s %f %s')
names =
  'Sam'
  'Nike'
age =
  23
  24
dat =
  83.2780
 102.2200
code =
  'p231'
  'k18N'
```

Спецификатор формата может предваряться строкой, общей для полей файла. В этом случае указанная строка не считывается, а происходит занесение следующих за ней данных в элементы подходящего по типу массива. Например, пусть в файле table.dat хранится информация:

```
book part4 pages400
article part2 pages20
```

Общие части полей `part` и `pages`, не подлежащие считыванию, указываются перед соответствующими спецификаторами:

```
>> [kind, s, p]=textread('printed.dat', '%s part%d pages%d')
kind =
  'book'
  'article'
s =
   4
   2
p =
  400
  20
```

Таким образом функция `textread` позволяет указывать параметры, управляющие чтением из файла. Названия параметров и их значения задаются парами в конце списка входных аргументов:

```
[...]=textread(...,param1,value1,param2,value2,...)
```

Возможно указание следующих параметров:

- ◊ 'whitespace' — вектор из символов, которые считаются размечающими. Допустимы значения \b (backspace), \f (form feed), \n (новая строка), \r (символ возврата каретки), \t (табуляция), \\ (обратная косая черта), \\'' или \'' (апостроф), %% (знак процента). По умолчанию установлен вектор '[\b \r \n \t]';
- ◊ 'delimiter' — символы, использующиеся в качестве разделителя (по умолчанию не установлены) ;
- ◊ 'expchars' — символы, применяемые для экспоненциальной записи чисел. По умолчанию 'eEdD';
- ◊ 'bufsize' — максимальная длина строки в байтах (по умолчанию 4095);
- ◊ 'headerlines' — число пропускаемых строк от начала файла при чтении;
- ◊ 'commentstyle' — задание символов, определяющих пропуски при чтении. Возможны значения: 'matlab' (игнорируются символы после %), 'shell' (игнорируются символы после #), 'c' (игнорируются символы между /* и */), 'c++' (игнорируются символы после //).

Функции для работы с массивами ячеек

- ❐ **cell** — создание пустого массива ячеек заданного размера.

  C=cell(n) — ячейки массива C размера n на n являются пустыми массивами. Возможно также создание прямоугольных массивов ячеек и массивов произвольной размерности: cell(m,n), cell(m,n,p,...), например:

```
>> A=ones(2,6);
>> C=cell(size(A))
C =
    []    []    []    []    []    []
    []    []    []    []    []    []
```

- ❐ **cellfun** — применение функции к содержимому массива ячеек.
  - A=cellfun(fun, C) — вызов одной из нижеперечисленных функций от каждой ячейки массива C и запись результата в числовой массив A, причем size(A)=size(C). Входной аргумент fun может быть:
    - ◊ 'isreal' — если содержимое ячейки состоит из вещественных чисел, то в соответствующий элемент A заносится единица, и ноль — в противном случае;
    - ◊ 'isempty' — если ячейка является пустым массивом, то в соответствующий элемент A заносится единица, и ноль — в противном случае;

- ◊ `'islogical'` — если ячейка является логическим массивом, то в соответствующий элемент `A` заносится единица, и ноль — в противном случае;
- ◊ `'length'` — элементы массива `A`, соответствующие каждой ячейке из `C`, принимают значения длин ячеек;
- ◊ `'ndims'` — число размерностей ячеек, в `C` помещаются соответствующие элементы `A`;
- ◊ `'prodofsize'` — значения элементов массива `A` равны количеству элементов в ячейках массива `C`.

Пример использования `cellfun`:

```
>> C={ones(3,5,2) 'fsfhsfh'; rand(10) strvcat('a1sd', 'hhh')}
C =
  [ 3x5x2 double]  'fsfhsfh'
  [10x10 double]  [2x4 char]
>> A=cellfun('prodofsize',C)
A =
  30   7
  100   8
```

- `A=cellfun('size',C,k)` — в массиве `A` возвращаются размеры содержимого ячеек вдоль `k`-ой размерности, например, для определенного выше массива ячеек `C`:

```
>> A=cellfun('size',C,2)
A =
   5   7
  10   4
```

- `A=cellfun('isclass',C,classname)` — логические единицы в массиве `A` соответствуют тем ячейкам, содержимое которых является объектом класса `classname`. Например, проверка ячеек инициализированного выше массива `C` на принадлежность классу `char` осуществляется так:

```
>> A=cellfun('isclass',C,'char')
A =
   0   1
   0   1
```

Входной аргумент `classname` может, также принимать значения: `'double'`, `'sparse'`, `'struct'`, `'cell'`, или являться именем класса, определенного пользователем.

❒ **`cellstr`** — преобразование массива символов в массив ячеек.

```
Create cell array of strings from character array
```

`C=cellstr(strmas)` — ячейки массива `C` образуются из строк массива символов `strmas`, пример:

```
>> strmas=['aaaaaa';'bbbbbb';'cccccc'];
>> C=cellstr(strmas)
C =
  'aaaaaa'
  'bbbbbb'
  'cccccc'
```

- **`cell2struct`** — преобразование массива ячеек в массив структур.

  `struct=cell2struct(c,fields,dim)` — содержимое ячеек массива `C` (вдоль размерности `dim`) записывается в поля структур массива `struct`. Названия полей определяются элементами массива `fields`, который может быть либо массивом символов, либо массивом ячеек из строк.

  Пример использования `cell2struct`:

```
>> C={'March' 12 '10-30'; 'April' 26 '17-45'; 'May' 10 '12-00'};
>> fields={'Month' 'Day' 'Time'};
>> struct=cell2struct(C, fields, 2)
struct =
3x1 struct array with fields:
  Month
  Day
  Time
>> struct(1)
ans =
  Month: 'March'
   Day: 12
   Time: '10-30'
```

  Неправильное указание размерности, вдоль которой происходит преобразование, приводит к несоответствующему преобразованию массива ячеек в массив структур (в случае совпадения числа элементов в `fields` с длиной `C` вдоль данной размерности):

```
>> struct=cell2struct(C, fields, 1);
>> struct(1)
ans =
  Month: 'March'
   Day: 'April'
   Time: 'May'
```

В случае несовпадения числа полей в `fields` и размера `C` вдоль `dim` выводится сообщение об ошибке:

```
>> fields={'Month' 'Day'};
>> struct=cell2struct(C, fields, 2);
??? Error using ==> cell2struct
Number of field names must match number of fields in new structure.
```

❒ **`celldisp`** — вывод содержимого массива ячеек в командное окно.

- `celldisp(C)` — последовательный вывод содержимого каждой ячейки массива `C`.

```
>> C={'May' 10 ones(2)};
>> celldisp(C)
C{1} =
May
C{2} =
  10
C{3} =
   1   1
   1   1
```

- `celldisp(C,name)` — при выводе имя массива заменяется на строку `name`.

❒ **`cellplot`** — отображение содержимого массива ячеек в графическом окне.

См. разд. "Массивы ячеек" главы 8.

Информация о соответствии цвета типу содержимого ячейки выводится при указании второго дополнительного входного аргумента: `cellplot(C, 'legend')`.

❒ **`num2cell`** — преобразование числового массива в массив ячеек.

`C=num2cell(A)` — элементы числового массива `A` помещаются в отдельные ячейки массива `C`, например:

```
>> A=[1 2 3 4; 5 6 7 8];
>> C=num2cell(A)
C =
  [1]  [2]  [3]  [4]
  [5]  [6]  [7]  [8]
```

Размеры образующегося массива ячеек `C` совпадают с размерами `A`.

`C=num2cell(A,dim)` — числа массива `A` вдоль размерности `dim` помещаются в отдельные ячейки:

```
>> C=num2cell(A,2);
>> celldisp(C)
```

```
C{1} =
   1   2   3   4
C{2} =
   5   6   7   8
```

Функции для работы со структурами

- **deal** — копирование одних переменных в другие, или запись содержимого одной переменной в несколько. Данная функция оказывается полезной для организации доступа к полям массивов структур. Ниже приведены варианты вызова `deal` с примерами.
  - `[S.field]=deal(x)` — присвоение значения `x` полям `field` структур массива `S`, например: `[s.name]=deal('Bill')`. Если массив структур `S` не существует, то следует использовать обращение: `[S(1:n).name]=deal('Bill')`, где `n` — число структур в создаваемом массиве `S`.
  - `[C{:}]=deal(S.field)` — копирование значений полей `field` структур массива `S` в массив ячеек `C`. Если массив ячеек `C` не существует, то следует использовать обращение: `[C(1:n).name]=deal(S.field)`, где `n` — число ячеек в создаваемом массиве `C`.
  - `[a,b,c,...]=deal(S.field)` — копирование значений полей `field` структур массива `S` в отдельные переменные `a`, `b`, `c`,...
- **fieldnames** — получение названий полей структуры.

  `C=fieldnames(S)` — массив ячеек из строк `C` содержит имена полей структур массива `S`, например:

```
>> S(1).name='Bill';
>> S(1).age=30;
>> S(2).name='Smith';
>> S(2).age=32;
>> C=fieldnames(S)
C =
  'name'
  'age'
```

- **getfield** — получение содержимого определенных полей структуры.

  `f=getfield(S,'field')` — возвращает содержимое полей с именем `field` структуры `S`. Входной аргумент `S` должен быть структурой, а не массивом, т. е. `size(S)=[1 1]`. Например, для структуры `S`, определенной выше, следует вызывать `getfield` в цикле `for`:

```
S(1).name='Bill';
S(1).age=30;
```

```
S(2).name='Smith';
S(2).age=32;
for i=1:2
  fage(i)=getfield(S(i),'age');
end
fage
```

В командное окно выводится

```
fage =
  30  32
```

Запись значений одноименных полей следует согласовывать с типом массива, в который заносится результат. Например, попытка выполнения цикла

```
for i=1:2
  f(i)=getfield(S(i),'name');
end
```

приведет к ошибке, поскольку строки в полях `name` имеют разную длину и образовать из них символьный массив не удается. Выход состоит в занесении значений полей в массив ячеек

```
for i=1:2
  fname{i}=getfield(S(i),'name');
end
```

- **`rmfield`** — удаление полей структуры.

  `S=rmfield(S,'field')` — удаление поля с именем `field` из структур массива `S`, например, для заполненного выше массива `S` поле `name` удаляется при помощи обращения:

```
>> S=rmfield(S,'name')
S =
1x2 struct array with fields:
  age
   structure array S.
```

  Возможно удаление сразу нескольких полей: `S=rmfield(S,F)`, здесь `F` должен быть массивом символов или ячеек из строк и содержать имена удаляемых полей.

- **`setfield`** — присвоение значения полю структуры.

  `S=setfield(S,'field',v)` — в поле с именем `field` структуры `S` заносится значение переменной `v`. Входным аргументом `S` должна быть именно структура, а не массив, т. е. `size(S)=[1 1]`. Заполнение массива структур производится при помощи цикла (см. `getfield`).

- **struct** — создание структур и массива структур.

  Общий вид обращения к `struct` выглядит следующим образом:

  ```
  S=struct('field1',val1,'field2',val2,...).
  ```

  Размеры выходного аргумента `S` определяются способом указания значений `val1`, `val2`,... полей `field1`, `field2`,...

  ```
  >> myS=struct('month','May','dat',10,'time','12-00')
  myS =
    month: 'May'
     dat: 10
     time: '12-00'
  ```

  Если `val1`, `val2`,... являются массивами ячеек одинаковой длины, то в результате `S` будет массивом структур:

  ```
  >> myS=struct('month',{'May' 'June'},'dat',{10 12},'time',{'12-00' '23-30'});
  >> myS(1)
  ans =
    month: 'May'
     dat: 10
     time: '12-00'
  >> myS(2)
  ans =
    month: 'June'
     dat: 12
     time: '23-30'
  ```

  Указание в качестве значения `val1`, `val2` и т. д. массива из одной ячейки приводит к заполнению данным значением полей всех структур выходного массива, например:

  ```
  >> myS1=struct('month',{'May'},'dat',{10 12},'time',{'12-00' '23-
  30'});
  >> myS1(1).month
  ans =
  May
  >> myS1(2).month
  ans =
  May
  ```

- **struct2cell** — преобразование массива структур к массиву ячеек.

  `C=struct2cell(S)` — содержимое полей структур массива `S` заносится в ячейки массива `C`. Если входной аргумент имеет размеры

`size(S)=[m n]` и каждая структура массива `S` состоит из `p` полей, то `size(S)=[p m n]`.

```
>> S(1,1)=struct('name','Sam','age',16);
>> S(1,2)=struct('name','Nik','age',18);
>> S(1,3)=struct('name','Dan','age',17);
>> S
S =
1x3 struct array with fields:
  name
  age
>> C=struct2cell(S)
C(:,:,1) =
  'Sam'
  [ 16]
C(:,:,2) =
  'Nik'
  [ 18]
C(:,:,3) =
  'Dan'
  [ 17]
>> size(C)
ans =
   2   1   3
```

Звуковые и графические файлы

Чтение, запись и преобразование звуковых данных

- **`lin2mu`** — мю-кодирование.

  `mu=lin2mu(y)` — логарифмическое кодирование сигнала с амплитудой от −1 до 1, записанного в массив `y`. Элементами вектора *mu* являются целые числа из диапазона [0, 255].

- **`mu2lin`** — обратное по отношению к `lin2mu` преобразование.

  `y=mu2lin(mu)` — входной аргумент (вектор с целыми числами от 0 до 255) преобразуется в вектор *y*, элементы которого принадлежат [−32124/32768, 32124/32768].

- **`sound`** — воспроизведение звука.

  - `sound(y,fs)` — воспроизведение дискретизованного звукового сигнала `y` с частотой дискретизации `fs`. Амплитуда `y` должна принадлежать

[−1, 1]. Значения с большей амплитудой усекаются до −1 или 1. Размер `size(y)=[n 2]` отвечает стереофоническому звуку.

- `sound(y)` — воспроизведение сигнала с частотой дискретизации, равной по умолчанию 8192 Гц.
- `sound(y,fs,bits)` — воспроизведение сигнала с частотой дискретизации `fs` и разрядностью `bits`.

❒ **`soundsc`** — воспроизведение звука с предварительным масштабированием.

- `sound(y,fs)`, `sound(y)`, `sound(y,bits)` — работают аналогично `sound`, но амплитуда сигнала предварительно масштабируется и приводится к отрезку [−1, 1] так, чтобы звук был максимальной громкости без усечения амплитуды (то есть без потери качества).
- `soundsc(y,...,slim)` — проигрывание звукового сигнала с предварительным масштабированием к [−1, 1] тех значений `y`, которые принадлежат отрезку, заданному вектором `slim=[slow shigh]`.

❒ **`wavread`** — считывание звука из wav-файла.

- `y=wavread(filename)` — в массив `y` заносятся амплитуды отсчетов (масштабированные к [−1, 1]) дискретизованного звука из wav-файла с именем `filename`. Вектор *y* соответствует монофоническому звуку, а матрица с двумя столбцами — стереофоническому, первому каналу отвечает `y(:,1)`, второму — `y(:,2)`.
- `[y,fs,nbits]=wavread(filename)` — в дополнительных выходных параметрах содержится информация о дискретизованном звуке (см. `sound`)
- `[...]=wavread(filename,n)` — считывание первых `n` сэмплов каждого канала из wav-файла `filename`.
- `[...]=wavread(filename,[n1 n2])` — считывание сэмплов от `n1` до `n2` каждого канала из wav-файла `filename`.
- `siz=wavread(filename,'size')` — возвращает размер дискретизованного звука, записанного в файле `filename`. Выходной аргумент является вектором, первый элемент которого равен числу сэмплов, а второй — числу каналов.
- `[y,fs,nbits,opts]=wavread(...)` — возвращает сэмплы в массиве `y`, частоту дискретизации в `fs`, разрядность в `bits` и структуру `opts`, поля которой содержат информацию о формате файла.

❒ **`wavwrite`** — запись звуковых данных в формате WAV.

- `wavwrite(y,fs,nbits,wavefile)` — в wav-файл с именем `filename` записываются звуковые данные из массива `y`. Частота дискретизации указывается в `fs` (в Гц), а разрядность в `bits` (допустимы только зна-

чения 8 и 16). Стереофонический звук представляется массивом размера `size(y)=[n 2]`. Значения амплитуды должны принадлежать отрезку [−1, 1], остальные значения усекаются до −1 или 1.

- `wavwrite(y,fs,wavefile)` — запись по умолчанию происходит с разрядностью 16 битов.
- `wavwrite(y,wavefile)` — запись по умолчанию происходит с разрядностью 16 битов и частотой дискретизации 8192 Гц.

Графические файлы

❒ **`imfinfo`** — получение информации о графическом файле.

`info=imfinfo(filename,fmt)` — возвращает структуру, содержащую информацию о графическом файле с именем `filename` и типом `fmt`. Поддерживаются широко используемые форматы файлов, которым соответствуют следующие значения входного аргумента `fmt`: `'bmp'`, `'jpg'` (или `'jpeg'`), `'pcx'`, `'png'`, `'tif'` (или `'tiff'`). Следует иметь в виду, что если файл в формате TIFF содержит несколько изображений, то информация возвращается в массиве структур. Доступ, к свойствам, например, второго изображения производится при помощи указания номера структуры массива: `info(2)`.

Первые девять полей структуры не зависят от формата графического файла и содержат следующую информацию:

- `Filename` — строка с именем файла, если файл находится не в текущем каталоге, то в строку заносится полное имя;
- `FileModDate` — строка со временем последнего изменения файла;
- `FileSize` — размер файла в байтах;
- `Format` — строка из трех символов, содержащая формат файла;
- `FormatVersion` — строка или числовая переменная с версией формата;
- `Width` — ширина изображения в пикселах;
- `Height` — высота изображения в пикселах;
- `BitDepth` — глубина цвета (бит/пиксел);
- `ColorType` — тип изображения: `'truecolor'`, `'grayscale'` или `'indexed'`.

Остальные поля структуры `info` зависят от формата представления изображения.

Формат файла можно не указывать: `info=imfinfo(filename)`. В этом случае, функция `imfinfo` пытается определить формат хранения графических данных исходя из структуры файла.

- **imread** — чтение графической информации из файла в массив или массивы MatLab.
  - `A=imread(filename,fmt)` — запись в массив `A` графической информации из файла с именем `filename` типа `fmt` (см. `imfinfo`). Размерность массива `A` зависит от типа изображения. Если файл содержит изображение в оттенках серого (функция `infinfo` возвращает `'grayscale'` в поле `ColorType`), то `A` является двумерным массивом. Размеры массива определяются шириной и высотой изображения. Если `info=imfinfo(filename,fmt)`, то `size(A)=[info.Height info.Width]`.

    Цветное изображение (функция `infinfo` возвращает `'truecolor'` в поле `ColorType`) заносится в трехмерный массив `A`, `size(A)=[info.Height info.Width 3]`. Третье измерение представляет информацию о цвете: `A(i,j,:)=[R G B]`.
  - `[A,MAP]=imread(filename,fmt)` — запись в массив `A` графической информации из файла с индексированным цветом. Массивы `A` и `MAP` являются двумерными, причем значения `MAP` масштабированы от нуля до единицы.

  Формат графических данных можно не указывать. Функция `imread` пытается определить формат хранения графических данных и считать их, исходя из структуры файла.
- **imwrite** — запись графических данных из матрицы в файл.
  - `imwrite(A,filename,fmt)` — запись графических данных, содержащихся в матрице *A*, в файл с именем `filename` типа `fmt` (см. `imfinfo`). Если тип массива `A` есть `double array`, а его элементы имеют значения от нуля до единицы, то происходит предварительное преобразование к 8-битовым целым числам. Указание массива `A` класса `unit8` приводит к получению изображения либо в оттенках серого, либо цветного, в зависимости от размерности массива (см. `imread`).
  - `imwrite(A,MAP,filename,fmt)` — запись индексированных графических данных, содержащихся в массивах `A` и `MAP`, в файл с именем `filename` типа `fmt` (см. `imfinfo`). Если `A` есть `double array`, то он предварительно преобразовывается: `A=unit8(A-1)`. Если `A` класса `unit8` или `unit16`, то преобразования не происходит. Массив `MAP` должен являться цветовой палитрой, поддерживаемой MatLab.

  Формат графических данных можно не указывать. Функция `imwrite` выбирает формат, исходя из расширения файла, указанного в `filename`.

  Запись в графические файлы форматов TIFF, JPEG и PNG может потребовать установки дополнительных параметров. В данном случае используется вызов `imwrite` вида `imwrite(...,param1,val1,param2,val2,...)`.

Формат JPEG позволяет указать один параметр 'Quality', определяющий качество изображения. Значением 'Quality' может быть число от единицы до ста, причем бóльшие значения соответствуют лучшему качеству при сжатии изображения (соответственно увеличивается размер файла). Запись в формате TIFF управляется тремя параметрами:

- 'Compression' — значения: 'none', 'packbits', 'ccitt';
- 'Description' — строка с описанием файла (см. поле ImageDescription выходного аргумента imfinfo);
- 'Resolution' — вектор [XResolution YResolution].

Работа со строками

Обработка строк

❒ **deblank** — удаление пробелов в конце строки.

- snew=deblank(s) — удаление пробелов в конце строки или строковой переменной s.
- masnew=deblank(mas) — удаление пробелов в конце каждой строки массива ячеек из строк mas.

❒ **findstr** — поиск подстроки в строке.

k=findstr(s1,s2) — выходной аргумент — вектор k содержит позиции, с которых подстрока начинается в строке. Входными аргументами s1 и s2 являются строки или строковые переменные. Подстрокой считается входной аргумент меньшей длины.

❒ **lower** — преобразование в строчные буквы.

snew=lower(s) — преобразование символов строки s в строчные буквы. Допускается применение функции lower к массиву ячеек, состоящих из строк (см. deblank).

❒ **strcat** — сцепление строк.

S=strcat(s1,s2,s3,...) — горизонтальное сцепление строк s1, s2, s3,... и запись результата в строку S. Завершающие пробелы в каждой сцепляемой строке игнорируются. Если входные аргументы являются массивами символов, то выходной аргумент также массив символов. Указание в качестве входных аргументов массивов ячеек из строк приводит к образованию нового массива ячеек из строк. Каждая ячейка нового массива содержит результат сцепления строк, входящих в соответствующие ячейки каждого из массивов. Массивы должны быть одинаковых размеров, например:

```
>> S=strcat({'May', 'June'},{'12', '23'})
S =
  'May12'  'June23'
```

Допустимо указание массива, состоящего из одной ячейки:

```
>> S=strcat({'May', 'June'},{'12'})
S =
  'May12'  'June12'
```

- **strcmp** — сравнение строк.

  `flag=strcmp(s1,s2)` — возвращает единицу в случае совпадения строк `s1` и `s2`, и ноль — в противном случае.

  Входными аргументами могут быть массивы (одинаковых размеров) ячеек из строк. В данном случае выходной аргумент является массивом того же размера, что и исходные, состоящий из единиц и нулей, например:

```
>> flag=strcmp({'May10','May14','June02'},{'May11','May14','June02'})
flag =
0   1   1
```

  Если один из входных массивов имеет размер, равный единице, (или является строкой или строковой переменной), то происходит поэлементное сравнение:

```
>> flag=strcmp({'May10','May14','June02'},{'May14'})
flag =
   0   1   0
```

- **strcmpi** — сравнение строк, прописные и строчные буквы не различаются.

  Использование `strcmpi` аналогично `strcmp`.

- **strjust** — выравнивание элементов в строке.

  - `news=strjust(s)` или `news=strjust(s,'right')` — выравнивание по правому краю:

```
>> news=strjust('          text  ')
news =
           text
```

  - `news=strjust(s,'left')` — выравнивание по левому краю:

```
>> news=strjust('         text  ', 'left')
news =
text
```

  - `news=strjust(s,'center')` — выравнивание по центру:

```
>> news=strjust('         text  ', 'center')
news =
     text
```

- **strmatch** — поиск в массиве символов или ячеек из строк совпадений с заданной строкой.

- `k=strmatch(str,MAS)` — поиск в массиве символов или массиве ячеек из строк `MAS` строки, начинающейся с `str`. Выходной аргумент `k` является массивом с номерами подходящих строк в `MAS`.
- `k=strmatch(str,MAS,'exact')` — возвращает номера строк из `MAS`, в которые `str` входит как целая строка:

```
>> k=strmatch('Ma',{'March','April','May'})
k =
   1
   3
>> k=strmatch('Ma',{'March','April','May'},'exact')
k =
   []
```

❒ **strncmp** — сравнение первых `n` символов двух строк.

`flag=strncmp(s1,s2,n)` — возвращает единицу, если первые `n` символов в строках `s1` и `s2` совпадают, и ноль — в противном случае. Входными аргументами могут быть массивы (одинаковых размеров) ячеек строк. В данном случае возвращается массив из нулей и единиц, единицы соответствуют строкам, первые `n` символов которых совпадают, например:

```
>> flag=strncmp({'March','April','May'},{'May','May','May'},2)
flag =
   1   0   1
```

❒ **strrep** — замена в строке одной подстроки на другую.

`new=strrep(str,subold,subnew)` — замена в строке `str` подстрок `subold` на подстроки `subnew`. Входные аргументы могут быть массивами (одинакового размера) ячеек из строк, например:

```
>> new=strrep({'March','April','May'},{'ar','pr','ay'},
{'AR','PR','AY'})
strnew =
  'MARch'  'APRil'  'MAY'
```

Возможно указание массива из одной ячейки в качестве входного аргумента:

```
>> strnew=strrep({'March','April','May'},{'ar','pr','ay'},{'###'})
strnew =
  'M###ch'  'A###il'  'M###'
>> strnew=strrep({'March', 'April', 'May'}, {'Ma'}, {'##'})
strnew =
  '##rch'  'April'  '##y'
```

```
>> strnew=strrep({'March', 'April', 'May'}, {'Ma'}, {'##', '**',
'&&'})
strnew =
  '##rch'  'April'  '&&y'
```

❐ **strtok** — поиск первой подстроки, отделенной пробелами в строке.

- `tok=strtok(str)` — возвращает в строковой переменной `tok` первую подстроку из строковой переменной или строки `str`, отделенную пробелами или табуляцией. Пробелы (табуляция) справа и слева игнорируются, например:

```
>> tok=strtok('  ABC DEFG H')
tok =
ABC
```

- `tok=strtok(str,delim)` — возвращает в строковой переменной `tok` первую подстроку из строковой переменной или строки `str`, отделенную одним из символов, входящим в `delim`.

```
>> tok=strtok('ABC-DEFG H','-')
tok =
ABC
```

- `[tok,rem]=strtok(...)` — второй дополнительный аргумент содержит остаток строки после `tok`:

```
>> [tok,rem]=strtok('ABC-DEFG H','-')
tok =
ABC
rem =
-DEFG H
```

❐ **strvcat** — вертикальное сцепление строк.

`mas=strvcat(str1,str2,str3,..)` — формирование двумерного массива символов, каждая строка которого содержит `str1`, `str2`, `str3`,... Строки `mas` автоматически дополняются пробелами до нужной длины.

```
>> mas=strvcat('March','April','May')
mas =
March
April
May
>> whos spring
 Name     Size     Bytes  Class
 spring    3x5     30    char array
```

❒ **upper** — преобразование в прописные буквы.

`snew=upper(s)` — преобразование символов строки `s` в прописные буквы. Допускается применение функции `upper` к массиву ячеек, состоящих из строк (см. `deblank`).

Преобразования строка-число

❒ **char** — создание массива символов или строки.

- `ch=char(code)` — преобразование массива `code`, содержащего целые числа, в массив символов. Целые числа от 32 до 127 соответствуют печатаемым символам:

```
>> ch=char(32:127);
>> ch(1:70)
ans =
 !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcde
>> ch(71:end)
ans =
fghijklmnopqrstuvwxyz{|}~□
```

Символы, соответствующие целым числам, большим 127, зависят от шрифта, установленного в командном окне. Например, для шрифта Courier

```
>> ch=char(224:256)
ch =
абвгдежзийклмнопрстуфхцчшщъыьэюя
```

- `chmas=char(s1,s2,s3,..)` — формирование массива символов `chmas` из строк или строковых переменных `s1`, `s2`, `s3`,... Каждая строка дополняется пробелами справа для приведения к одинаковым размерам. Пустые строки, указанные во входных аргументах, учитываются при конструировании массива символов, например:

```
>> chmas=char('AAAAAAAAAAAA','','BBBBBBBBB')
chmas =
AAAAAAAAAAAA

BBBBBBBBB
```

Входные аргументы могут быть массивами символов:

```
>> chmas1=char('AAAA', 'BB');
>> chmas2=char('CCC', 'DDDDDD');
>> chmas=char(chmas1,chmas2)
```

```
chmas =
AAAA
BB
CCC
DDDDDD
```

- **int2str** — преобразование чисел в массив символов.

  `chmas=int2str(A)` — округление элементов матрицы *A* и запись результата в массив символов.

- **mat2str** — преобразование матрицы в строку.

  - `str=mat2str(A)` — строковая переменная `str` содержит представление матрицы *A*, в том виде, в котором матрица задается из командной строки или в M-файле, например:

    ```
    >> A=pi*eye(2);
    >> str=mat2str(A)
    str =
    [3.14159265358979 0;0 3.14159265358979]
    ```

    При преобразовании матрицы в строку округления элементов матрицы не происходит.

  - `str=mat2str(A,n)` — округление до `n` цифр после десятичной точки.

- **num2str** — преобразование матрицы в массив символов.

  - `chmas=num2str(A)` — элементы строк матрицы *A* образуют строки массива символов `chmas`. Удерживается четыре цифры после десятичной точки и при необходимости используется экспоненциальная форма записи числа (аналогично формату `short e`).

  - `chmas=num2str(A,n)` — округление происходит до `n` цифр после десятичной точки.

  - `chmas=num2str(A,format)` — форматное преобразование, строка `format` формируется из спецификаторов аналогично `sprintf`.

- **sprintf** — форматная запись в строку.

  - `str=sprintf(format,A)` — конструирование строки `str` из вещественных данных, содержащихся в матрице *A*, на основе формата, который указан в строке `format`. Спецификаторы формата аналогичны тем, которые используются в `fprintf`.

    Пример использования `sprintf`:

    ```
    >> A=[1.1 3.2; 0.7 -4.2];
    ```

```
>> str=sprintf('A(1,1)=%8.1d A(1,2)=%8.1d \t A(2,1)=%8.1d
A(2,2)=%8.1d',A)
str =
A(1,1)=1.1e+000 A(1,2)=7.0e-001   A(2,1)=3.2e+000 A(2,2)=-4.2e+000
```

- `[str,errmsg]=sprintf(format,A)` — если при форматной записи произошла ошибка, то выходной аргумент `errmsg` содержит соответствующее сообщение.

Подробная информация о форматной записи в файл с примерами использования содержится в разд. "Текстовые файлы" главы 8.

❒ **sscanf** — чтение данных из строки или строковой переменной в заданном формате.

Использование `sscanf` во многом схоже с `fscanf`, за исключением того, что считывание производится из строки, а не из файла.

См. разд. "Текстовые файлы" главы 8.

❒ **str2double** — преобразование чисел, записанных в строках, в числовой массив.

- `a=str2double(str)` — из строки `str` извлекается число и заносится в переменную `a`. Строка `str` может содержать цифры, точку, знаки плюс или минус, символы `e` или `i` и запятую для разделения знаков тысяч, например:

```
>> a=str2double('1,485,000.00')
a =
   1485000
>> a=str2double('-1.2e-2')
a =
  -0.0120
>> a=str2double('-2+3*i')
a =
 -2.0000 + 3.0000i
```

Если строка не может быть преобразована в число, то возвращается `NaN`.

- `A=str2double(masstr)` — содержимое массива ячеек из строк `masstr` преобразуется в элементы числового массива `A` того же размера, что и `masstr`, например:

```
>> A=str2double({'-7' '3*i' 'FFF' '3.19'})
A =
 -7.0000         0 + 3.0000i   NaN       3.1900
```

- **str2num** — преобразование массива символов в массив чисел.

  A=str2num(chmas) — строки массива символов chmas должны состоять из тех же символов, что и в str2double, например:

```
>> chmas=['1.3 0.4 3+2*i';'1-3*i 29 0.05'];
>> A=str2num(chmas)
A =
  1.3000        0.4000        3.0000 + 2.0000i
  1.0000 - 3.0000i 29.0000        0.0500
```

  Если строки в chmas не могут быть преобразованы в числа, то возвращается пустая матрица *A*. Пробелы в строках chmas существенны и определяют количество элементов в *A*, например:

```
>> A=str2num('-1-2i')
A =
 -1.0000 - 2.0000i
>> A=str2num('-1 -2i')
A =
 -1.0000          0 - 2.0000i
```

Преобразование системы счисления

- **bin2dec** — преобразование строки с двоичным числом в десятичное число, например:

```
>> a=bin2dec('1110001101010')
a =
    7274
```

- **dec2bin** — преобразование десятичного числа в строку с двоичным представлением, например:

```
>> str=dec2bin(7274)
str =
1110001101010
```

  Входной аргумент может быть только целым неотрицательным числом, не превосходящим 2^{52}.

- **dec2hex** — преобразование десятичного числа в строку с шестнадцатеричным представлением, например:

```
>> str=dec2hex(7274)
str =
1C6A
```

  Входной аргумент может быть только целым неотрицательным числом, не превосходящим 2^{52}.

- **hex2dec** — преобразование строки с шестнадцатеричным представлением в десятичное число, например:

```
>> a=hex2dec('1C6A')
a =
    7274
```

- **hex2num** — преобразование шестнадцатеричного представления вещественного числа двойной точности (в стандарте IEEE) в число.

```
>> format long
>> a=hex2num('411a243774442a28')
a =
  4.283018635412776e+005
```

Строка, длина которой меньше шестнадцати, дополняется нулями справа. Если входной аргумент является массивом строк, то обрабатывается каждая строка и результат записывается в вещественный массив.

Работа с матрицами и массивами

Работе с матрицами посвящено достаточно много глав и разделов книги.

См., например, главу 2, разд. "Задачи линейной алгебры" главы 6, главу 16.

Создание матриц и массивов

- **blkdiag** — конструирование блочно-диагональных матриц.

`M=blkdiag(A,B,C)` — занесение в массив `M` блочно-диагональной матрицы:

$$M = \begin{bmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{bmatrix}$$

- **compan** — создание сопровождающей матрицы.

`A=compan(v)` — возвращает сопровождающую матрицу для полинома, заданного вектором коэффициентов `v`.

- **eye** — создание единичной матрицы.
  - `I=eye(n)` — `I` содержит квадратную единичную матрицу размера `n`.
  - `I=eye(m,n)` — `I` содержит прямоугольную матрицу размера `m` на `n` с единицами на главной диагонали.
- **gallery** — функция, позволяющая получать более пятидесяти различных стандартных матриц.

Использование:

```
[A1,A2,...] = gallery(name, p1, p2, ...)
```

Как правило, `p1` и `p2` задают размеры матрицы, и функция вызывается с одним выходным аргументом, возвращающим матрицу. Аргумент `name` является именем матрицы, например `'cauchy'`, `'orthog'`.

- **hadamard** — создание матрицы Адамара, например `H=hadamard(n)`.
- **hankel** — создание матрицы Ганкеля, например `H=hankel(n)`.
- **hilb** — создание матрицы Гильберта, например `H=hilb(n)`.
- **invhilb** — вычисление матрицы, обратной к матрице Гильберта, например `H=invhilb(n)`.
- **linspace** — генерация вектора, значения элементов которого изменяются с постоянным шагом.
  - `v=linspace(a,b)` — в вектор *v* заносится 100 значений от `a` до `b`.
  - `v=linspace(a,b,n)` — в вектор *v* заносится `n` значений от `a` до `b`.
- **logspace** — генерация вектора, значения элементов которого изменяются с постоянным шагом в логарифмической метрике.
  - `v=logspace(a,b)` — в вектор *v* заносится 50 значений от 10^a до 10^b.
  - `v=logspace(a,b,n)` — в вектор *v* заносится `n` значений от 10^a до 10^b.
- **magic** — создание "магического квадрата"

  `M=magic(n)` — квадратная матрица *M* размера `n`, состоящая из чисел от 1 до n^2, обладает следующим свойством: сумма элементов любой строки совпадает с суммой элементов любого столбца и диагонали.
- **ones** — создание массива, элементы которого являются единицами.
  - `A=ones(n)` — `A` содержит квадратную матрицу из единиц размера `n`.
  - `A=ones(m,n)` — `A` содержит прямоугольную матрицу размера `m` на `n`, состоящую из единиц.
  - `A=ones(m,n,k)` — `A` содержит массив трех измерений размера `m` на `n` на `k`, состоящий из единиц. Допускается создание массивов большего числа измерений.
- **pascal** — генерация матрицы Паскаля, например `P=pascal(n)`. Структура матрицы соответствует треугольнику Паскаля.
- **rand** — создание массивов равномерно распределенных случайных чисел. Использование аналогично `ones`.
- **randn** — создание массивов, состоящих из чисел, распределенных по нормальному закону. Использование аналогично `ones`.
- **toeplitz** — создание теплицевой матрицы, элементы которой равны вдоль каждой из диагоналей.
  - `T=toeplitz(c,r)` — создание несимметричной теплицевой матрицы при помощи вектор-столбца `c` и вектор-строки `r`.

- `T=toeplitz(r)` — создание симметричной теплицевой матрицы при помощи вектор-строки `r`.

- **`zeros`** — создание массивов, состоящих из нулей. Использование аналогично `ones`.
- **`wilkinson`** — создание матрицы Уилкинсона, которая имеет близкие пары собственных значений.

Операции с массивами

- **`cat`** — сцепление массивов, соответствующие размеры должны совпадать. Если `A` и `B` матрицы, то возможны следующие варианты вызова `cat`:
  - `M=cat(1,A,B)` — сцепление `A` и `B` вдоль первого измерения (массивы `A` и `B` расположены в столбик);
  - `M=cat(2,A,B)` — сцепление `A` и `B` вдоль второго измерения (массивы `A` и `B` расположены в строку);
  - `M=cat(3,A,B)` — образование трехмерного массива.
- **`diag`** — выделение диагонали и конструирование диагональной матрицы.
  - `A = diag(a)` — создание диагональной матрицы *A*, на диагонали которой стоят элементы вектора *a*.
  - `A = diag(a,k)` — создание диагональной матрицы *A*, на побочной `k`-ой диагонали которой стоят элементы вектора *a*.
  - `a = diag(A)` — выделение диагонали матрицы *A* в вектор *a*.

  См. разд. "Создание матриц специального вида" главы 2.
- **`fliplr`** — перестановка столбцов матрицы слева направо, возвращает измененную матрицу.
- **`flipud`** — перестановка строк матрицы сверху вниз, возвращает измененную матрицу.
- **`repmat`** — создание блочной матрицы или многомерного блочного массива из одинаковых блоков.
  - `M=repmat(A,m,n)` — матрица *M* состоит из `m` блоков по вертикали и `n` по горизонтали, каждый блок является матрицей *A*.
  - `M=repmat(A,[m n])` — аналогично `M=repmat(A,m,n)`.
  - `M=repmat(a,[m n p ...])` — конструирование многомерного блочного массива.
- **`reshape`** — изменение формы матрицы или массива.
  - `A=reshape(x,m,n)` — формирование матрицы `m` на `n` из элементов массива `x` длины `m*n`. Элементы `x` выбираются последовательно, образуя столбцы `A`.

- `A=reshape(x,m,n,p)` — формирование трехмерного массива `m` на `n` на `p` из элементов массива `x` длины `m*n*p`. Аналогичным образом создаются многомерные массивы.

❐ **`rot90`** — поворот матрицы.

- `B=rot90(A)` — `B` образуется из `A` поворотом против часовой стрелки на 90°.
- `B=rot90(A,k)` — `B` образуется из `A` поворотом против часовой стрелки на 90° `k` раз.

❐ **`tril`** — выделение нижнего треугольника из матрицы.

- `L=tril(A)` — в матрицу L тех же размеров, что и A, заносятся элементы нижнего треугольника A с диагональю.
- `L=tril(A,k)` — в матрицу L тех же размеров, что и A, заносятся элементы, находящиеся ниже `k`-ой поддиагонали и на ней (нумерация поддиагоналей такая же, как и в `diag`).

См. разд. "Создание матриц специального вида" главы 2.

❐ **`triu`** — выделение верхнего треугольника из матрицы (аналогично `tril`).

Математические функции

Элементарные математические функции подробно описаны в начале книги.

См. разд. "Встроенные элементарные функции" главы 1.

Специальные функции

❐ **`airy`** — функции Эйри первого и второго порядков, являющиеся решениями дифференциального уравнения $w'' - zw = 0$.

- `w=airy(z)` — функция Эйри первого порядка.
- `w=airy(1,z)` — производная функции Эйри первого порядка.
- `w=airy(2,z)` — функция Эйри второго порядка.
- `w=airy(3,z)` — производная функции Эйри второго порядка.

Если `z` является массивом, то результат `w` будет массивом той же размерности со значениями функции Эйри от соответствующих элементов `z`.

- `[w,ierr]=airy(k,z)` — во второй, дополнительный, аргумент заносится информация о нахождении значения функции Эйри:

  `ierr=1` — неверно заданы входные аргументы;

  `ierr=2` — переполнение, ответ будет `Inf`;

`ierr=3` — частичная потеря точности при вычислениях;

`ierr=4` — полная потеря точности при вычислениях, `z` слишком большое;

`ierr=5` — вычислительный процесс не сходится, ответ будет `NaN`.

- **`besselh`** — функции Ганкеля первого и второго рода, выражающиеся через функции Бесселя (см. соответствующее дифференциальное уравнение ниже).
  - `f=besselh(nu,k,z)` — функция Ганкеля первого (для `k=1`) и второго (для `k=2`) рода.
  - `[w,ierr]=besselh(nu,k,z)` — аналогично обращению к `airy`.
  - `f=besselh(nu,1,z,1)` — то же самое, что `besselh(nu,1,z)*exp(-i*z)`.
  - `f=besselh(nu,2,z,1)` — то же самое, что `besselh(nu,2,z)*exp(i*z)`.
- **`besselj`**, **`bessely`** — функции Бесселя, являющиеся решениями дифференциального уравнения $z^2y''+zy'+(z^2-\nu^2)y=0$ для вещественных ν.
  - `f=besselj(nu,z)` — функция Бесселя первого рода.
  - `f=besselj(nu,z,1)` — то же самое, что `besselj(nu,z)*exp(-abs(imag(z)))`.
  - `f=bessely(nu,z)` — функция Бесселя второго рода.
  - `f=bessely(nu,z,1)` — то же самое, что `bessely(nu,z)*exp(-abs(imag(z)))`.

  Возможны вызовы со вторым дополнительным выходным аргументом, аналогично функции `airy`. Допустимы комплексные значения для `z`. Если `z` и `nu` — массивы одинаковых размеров, то результат `f` будет массивом того же размера с соответствующими значениями функции Бесселя. В случае, когда один из входных аргументов `z` или `nu` — число, а второй — массив, скалярный аргумент расширяется до массива и результатом является массив `f`. Таблица значений для различных `nu` и `z` получается, если один из аргументов `z` или `nu` — вектор-строка, а второй — вектор-столбец.
- **`besseli`**, **`besselk`** — модифицированные функции Бесселя, являющиеся решениями дифференциального уравнения $z^2y''+zy'-(z^2+\nu^2)y=0$ для вещественных ν.
  - `f=(nu,z)` — модифицированная функция Бесселя первого рода.
  - `f=besseli(nu,z,1)` — то же самое, что `besseli(nu,z)*exp(-abs(real(z)))`.
  - `f=besselk(nu,z)` — модифицированная функция Бесселя второго рода.

- `f=besselk(nu,z,1)` — то же самое, что `besselk(nu,z)*exp(-abs(real(z)))`.

Интерфейс функций `besseli`, `besselk` такой же, как у `besselj`, `bessely`.

❐ **`beta`**, **`betainc`**, **`betaln`** — бета-функция, неполная бета-функция и логарифм бета-функции. Интегральные представления бета-функции $B(z,w)$ и неполной бета-функции $B_x(z,w)$ выглядят следующим образом:

$$B(z,w)=\int_0^1 t^{z-1}(1-z)^{w-1}\,dt; \qquad B_x(z,w)=\frac{1}{B(z,w)}\int_0^x t^{z-1}(1-t)^{w-1}\,dt.$$

- `f=beta(z,w)` — вычисление бета-функции.
- `f=beta(x,z,w)` — вычисление неполной бета-функции, `x` должен принадлежать отрезку [0, 1].
- `f=betaln(z,w)` — вычисление натурального логарифма от бета-функции с использованием более эффективного алгоритма, чем `log(beta(z,w))`.

Аргументы z и w могут быть вещественными и комплексными числами или массивами одинаковой размерности. Один из аргументов может быть скаляром, в данном случае он расширяется до размеров массива.

❐ **`ellipj`** — эллиптические функции Якоби $sn(u)$, $cn(u)$, $dn(u)$, порождаемые обращением эллиптического интеграла

$$u=\int_0^{\varphi}\frac{d\varphi}{\sqrt{1-m\sin^2\varphi}}.$$

- `[sn,cn,dn]=ellipj(u,m)` — одновременное вычисление всех эллиптических функций Якоби для `m` из отрезка [0, 1].

  Размеры входных аргументов влияют на результат так же, как в `beta`.
- `[sn,cn,dn]=ellipj(u,m,tol)` — одновременное вычисление всех эллиптических функций Якоби для `m` из отрезка [0, 1] с заданной точностью (по умолчанию `eps`). Часто имеет смысл уменьшить точность для сокращения времени вычислений.

❐ **`ellipke`** — полные эллиптические интегралы первого $K(m)$ и второго $E(m)$ рода, которые определяются следующим образом:

$$K(m)=\int_0^{\pi/2}\frac{d\varphi}{\sqrt{1-m\sin^2\varphi}}; \qquad E(m)=\int_0^{\pi/2}\sqrt{1-m\sin^2\varphi}\,d\varphi.$$

- `k=ellipke(m)` — вычисление эллиптического интеграла первого порядка для `m` из отрезка [0, 1].
- `[k,e]=ellipke(m)` — одновременное вычисление эллиптических интегралов первого и второго порядков для `m` из отрезка [0, 1].
- `[k,e]=ellipke(m,tol)` — одновременное вычисление эллиптических интегралов первого и второго порядков для `m` из отрезка [0, 1] с заданной точностью (по умолчанию `eps`). Часто имеет смысл уменьшить точность для сокращения времени вычислений.

❒ **`erf`**, **`erfc`**, **`erfcx`**, **`erfinv`** — вычисление функции ошибок, дополнительного интеграла вероятностей и обратной к функции ошибок:

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt; \qquad \operatorname{erfc} x = 1 - \operatorname{erf} x; \qquad \operatorname{erfcx} x = e^{x^2} \operatorname{erfc} x.$$

- `y=erf(x)` — вычисление функции ошибок.
- `y=erfc(x)` — вычисление дополнительного интеграла вероятностей.
- `y=erfcx(x)` — вычисление масштабированного дополнительного интеграла вероятностей.
- `x=erf(y)` — вычисление функции ошибок, `y` должен принадлежать отрезку $[-1, 1]$.

❒ **`expint`** — интегральная показательная функция:

$$Ei(x) = \int_x^\infty \frac{e^{-t}}{t} dt.$$

`Ei=expint(x)`.

❒ **`factorial`** — факториал.

`p=factorial(n)` — нахождение факториала целого числа `n`, точный ответ получается только для `n`, меньших, чем 22, для остальных — приближенный.

❒ **`gamma`**, **`gammainc`**, **`gammaln`** — гамма-функция $\Gamma(\alpha)$, неполная гамма-функция $\Gamma_x(\alpha)$ и логарифм гамма-функции.

$$\Gamma(\alpha) = \int_0^\infty e^{-t} t^{\alpha-1} dt; \qquad \Gamma(\alpha) = \frac{1}{\Gamma(\alpha)} \int_0^x e^{-t} t^{\alpha-1} dt.$$

`y=gamma(a)`, `y=gammainc(x,a)`, `y=gammaln(a)`.

- **legendre** — присоединенные функции Лежандра первого рода $P_n^m(x)$ и полунормированные присоединенные функции Лежандра $S_n^m(x)$, определяемые формулами

$$P_n^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_n(x); \qquad S_n^m(x) = (-1)^m \sqrt{\frac{2(n-m)!}{(n+m)!}} P_n^m(x),$$

где $P_n(x)$ — полиномы Лежандра.

  - `p=legendre(n,x)` — вычисление присоединенных функций Лежандра первого рода для всех `m=0,1,...,n`. Ограничения на входные аргументы: `m` — целое число, меньшее 256, `x` принадлежит отрезку [−1, 1]. Если `x` — скаляр, то `p` является вектором, длина которого на единицу больше `n`. В случае, когда `x` — вектор, в выходном аргументе `p` возвращается матрица, столбцы которой содержат присоединенные функции Лежандра первого рода для всех `m=0,1,...,n`, вычисленные для каждого элемента вектора *x*.
  - `p=legendre(n,x,'sch')` — вычисление полунормированных присоединенных функций Лежандра первого рода для всех `m=0,1,...,n`. Интерфейс аналогичен `legendre(n,x)`.
- **pow2** — нахождение степеней двойки.
  - `x=pow2(y)` — в векторе *x* возвращается результат возведения числа 2 в степень, записанную в соответствующем элементе `y`.
  - `x=pow2(f,e)` — элементы вектора *x* вычисляются по формуле `x(i)=f(i)*2^e(i)`.
- **rat, rats** — приближение вещественных чисел отношением двух целых чисел (рациональной дробью).
  - `[n,d]=rat(x,tol)` — возвращает два целых числа `n` и `d` таких, что `n/d` приблизительно равно `x` (с точностью `tol`) в следующем смысле: `abs(n/d-x)<=tol*abs(x)`. Если `x` — массив, то `n` и `d` являются массивами того же размера, содержащими соответствующие значения числителя и знаменателя для каждого элемента `x`.
  - `[n,d]=rat(x)` — использует по умолчанию точность `tol=1.e-6*norm(X(:),1)`.
  - `s=rat(x)`, `s=rat(x,tol)` — возвращают ответ в строковой переменной `s`.
  - `s=rats(x,len)` — использует `rat` для приближенного представления числа `x` рациональной дробью со строкой длиной `len`.

Преобразование координат

- **cart2pol** — переход от декартовых координат к полярным или цилиндрическим.
  - `[th,r]=cart2pol(x,y)` — переход от декартовых координат к полярным по формулам `th=atan2(y,x)`, `r=sqrt(x.^2+y.^2)`.
  - `[th,r,z]=cart2pol(x,y,z)` — переход от декартовых координат к цилиндрическим по формулам `th=atan2(y,x)`, `r=sqrt(x.^2+y.^2)`, `z=z`.

  Угол `th` возвращается в радианах. Входные аргументы могут быть массивами одинаковых размеров, в этом случае выходные аргументы являются массивами тех же самых размеров и содержат полярные или цилиндрические координаты для соответствующих пар элементов из `x` и `y` или троек `x`, `y` и `z`.
- **cart2sph** — переход от декартовых координат к сферическим.

  `[th,phi,r]=cart2sph(x,y,z)` — переход от декартовых координат к сферическим по формулам: `th=atan2(y,x)`, `phi=atan2(z, sqrt(x.^2+y.^2))`, `r=sqrt(x.^2+y.^2+z.^2)`.

  Углы `th` и `rho` возвращаются в радианах. Интерфейс аналогичен `cart2pol`.
- **pol2cart** — переход от полярных или цилиндрических координат к декартовым.
  - `[x,y]=pol2cart(th,r)` — переход от полярных координат к декартовым.
  - `[x,y,z]=pol2cart(th,r,z)` — переход от цилиндрических координат к декартовым.

  Угол `th` задается в радианах. Входные аргументы могут быть массивами одинаковых размеров (см. `cart2pol`).
- **sph2cart** — переход от сферических координат к декартовым.

  `[x,y,z]=sph2cart(th,phi,r)`

  Уголы `th` и `phi` задаются в радианах. Интерфейс аналогичен `pol2cart`.

Функции для решения задач линейной алгебры

Решению задач линейной алгебры и матричного анализа посвящены отдельные разделы и главы книги.

См., в частности, разд. "Решение систем линейных уравнений" главы 2, разд. "Задачи линейной алгебры" главы 6, главу 16.

Матричный анализ

- **cond** — нахождение числа обусловленности по отношению к различным матричным нормам (см. `norm`).
  - `c=cond(A)` или `c=cond(A,2)` — число обусловленности по отношению к спектральной матричной норме, т. е. `norm(A)*norm(inv(A))`.
  - `c=cond(A,1)` — число обусловленности по отношению к первой матричной норме, т. е. `norm(A,1)*norm(inv(A),1)`.
  - `c=cond(A,'fro')` — число обусловленности по отношению к евклидовой матричной норме (норме Фробениуса), т. е. `norm(A,'fro')*norm(inv(A),'fro')`.
  - `c=cond(A,Inf)` — число обусловленности по отношению к бесконечной матричной норме, т. е. `norm(A,Inf)*norm(inv(A),Inf)`.
- **condeig** — вычисление косинусов углов между правыми и соответствующими левыми собственными векторами.
  - `c=condeid(A)` — вектор *c* содержит косинусы углов между соответствующими собственными векторами.
  - `[V,D,c]=condeid(A)` — дополнительно возвращается матрица *V*, состоящая из нормированных собственных векторов *A*, и диагональная матрица *D*, на диагонали которой записаны собственные значения *A*.
- **det** — вычисление определителя матрицы: `d=det(A)`.
- **norm** — векторные и матричные нормы.
  - Матричные нормы:
    - ◊ `n=norm(A)`, `n=norm(A,2)` — спектральная норма, т. е. `max(svd(A))` для прямоугольных матриц и `max(sqrt(eig(A*A')))` для квадратных;
    - ◊ `n=norm(A,1)` — максимальная столбцевая норма, равная `max(sum(abs(A)))`;
    - ◊ `n=norm(A,inf)` — максимальная строчная норма, равная `max(sum(abs(A')))`;
    - ◊ `n=norm(A,'fro')` — евклидова норма (или норма Фробениуса), равная `sqrt(sum(sum(abs(A).^2)))`.
  - Векторные нормы:
    - ◊ `n=norm(x)` — евклидова векторная норма, т. е. `sqrt(sum(abs(x).^2))`;
    - ◊ `n=norm(x,p)` — норма Гёльдера с показателем `p` от единицы до бесконечности, равная `sum(abs(x).^p)^(1/p)`;
    - ◊ `n=norm(x,Inf)` — бесконечная векторная норма, равная `max(abs(x))`;

- **null** — нахождение ортонормированного базиса ядра матрицы.

  `K=null(A)` — столбцы матрицы *K* образуют ортонормированный базис ядра `A`, т. е. таких векторов *x*, что `A*x=zeros(m,1)`, где `size(A)=[m n]`, `size(x)=[n,1]`.

- **orth** — нахождение ортонормированного базиса области значений матрицы.

  `R=orth(A)` — столбцы матрицы *R* образуют ортонормированный базис области значений `A`, т. е. таких векторов *y*, что `y=A*x`, для любых `x`. Верно `R'*R=eye(rank(A))`.

- **rank** — вычисление ранга матрицы, т. е. наибольшего числа линейно независимых столбцов. Алгоритм основан на нахождении сингулярных чисел матрицы.

  - `r=rank(A)` — возвращает количество сингулярных чисел матрицы *A*, которые больше, чем `max(size(A))*norm(A)*eps`.
  - `r=rank(A,tol)` — возвращает количество сингулярных чисел матрицы *A*, которые больше, чем `tol`.

- **rcond** — оценка обусловленности матрицы.

  `c=rcond(A)` — вычисляет оценку для обратного к числу обусловленности по отношению к максимальной столбцевой норме.

- **rref**, **rrefmovie** — нахождение приведенно-ступенчатой формы матрицы исключением по Гауссу—Жордану с выбором главного элемента.

  - `R=rref(A)` — в массиве `R` содержится приведенно-ступенчатая форма матрицы *A*, при вычислениях элементы, меньшие `max(size(A))*eps*norm(A,inf)`, полагаются равными нулю.
  - `[R,jb]=rref(A)` — вектор *jb* содержит номера связанных компоненов решения системы линейных уравнений с матрицей *A*; `length(jb)` является рангом *A*, найденным при помощи исключения; столбцы матрицы `A(:,jb)` составляют базис области значений *A*; `R(1:length(jb),jb)=eye(length(jb))`.
  - `[R,jb]=rref(A,tol)` — при вычислениях элементы, меньшие `tol`, полагаются равными нулю.
  - `rrefmovie(A)` — отображение каждого шага процесса исключения в командном окне MatLab.

- **subspace** — вычисление угла между двумя подпространствами.

  `phi=subspace(A,B)` — возвращает угол между двумя подпространствами, базисными векторами которых являются столбцы матриц *A* и *B*.

- **trace** — след матрицы.

  `t=trace(A)` — нахождение следа *A*, т. е. `sum(diag(A))`.

Решение спектральных задач

- **balance** — масштабирование элементов матрицы при помощи балансировки. Масштабирование применяется для предварительной обработки матрицы в случае сильного разброса абсолютных значений элементов. Балансировка матрицы A заключается в нахождении диагональной матрицы D такой, чтобы в одноименных строках и столбцах `B=inv(D)*A*D` суммы модулей элементов были примерно одинаковы. Балансировка не изменяет спектр матрицы.

  Использование:

  `B=balance(A)`, `[D,B]=balance(A)` — элементы D являются целыми степенями двойки. Если A есть симметричная матрица, то балансировки не происходит и `B=A`, `D=eye(size(A))`.

  Предварительная балансировка матрицы не всегда оправдана. Например, если относительно малые элементы исходной матрицы есть ошибки округления, то после балансировки они будут сравнимы с остальными элементами матрицы, что заведомо приведет к неверному результату при дальнейших вычислениях.

- **eig** — решение обычной и обобщенной проблемы на собственные значения.
  - `d=eig(A)` — в векторе d возвращаются собственные значения матрицы A, т. е. `A*u=d(i)*u`. Исходная матрица предварительно балансируется.
  - `[V,D]=eig(A)` — столбцы матрицы V являются собственными векторами A, D есть диагональная матрица, состоящая из собственных значений, `A*U=D*U`. Исходная матрица предварительно балансируется.
  - `[V,D]=eig(A,'nobalance')` — то же, что и `[V,D]=eig(A)`, но без предварительной балансировки.
  - `d=eig(A,B)` — в векторе d возвращаются обобщенные собственные значения, являющиеся решением `A*u=d(i)*B*u`, используется QZ-алгоритм.
  - `[V,D]=eig(A,B)` — дополнительно возвращаются обобщенные собственные векторы.
- **gsvd** — обобщенное сингулярное разложение (см. также `svd`).
  - `[U,V,X,C,S]=gsvd(A,B)` — нахождение унитарных матриц U и V, квадратной X и диагональных матриц C и S таких, что: `A=U*C*X'`, `B=V*S*X'`, а `C'*C+S'*S` является единичной. Если `size(A)=[m p]`, `size(B)=[n p]`, то `size(U)=[m m]`, `size(V)=[n n]`, `size(X)=[m min(m+n,p)]`.
  - `[U,V,X,C,S]=gsvd(A,B,0)` — если `m>=p` или `n>=p`, то U и V имеют не более `p` столбцов, а C и S — не более `p` строк. Обобщенные сингулярные значения есть `diag(C)./diag(S)`.

- `s=gsvd(A,B)` — возвращает вектор обобщеных сингулярных значений, определяемый как `diag(C'*C)./diag(S'*S))`.

❒ **schur** — разложение Шура.

- `[U,T]=schur(X)` — для квадратной матрицы X находятся матрица T и унитарная матрица U такие, что `X=U*T*U'` и `U'*U=eye(size(U))`.
- `T=schur(X)` — возвращает только матрицу T разложения Шура.

❒ **svd** — сингулярное разложение и нахождение сингулярных чисел.

- `[U,D,V]=svd(A)` — нахождение матрицы D с неотрицательными диагональными элементами, расположенными в порядке убывания, и унитарных матриц U и V таких, что `A=U*D*V'`. Если `size(A)=[m n]`, то `size(D)=[m n]`, `size(U)=[m m]`, `size(V)=[n n]`.
- `d=svd(A)` — в векторе d возвращаются сингулярные числа матрицы A.
- `[U,D,V]=svd(A,0)` — если `size(A)=[m n]` и `m>n`, то вычисляются только `n` первых столбцов U и `size(D)=[n n]`.

Решение линейных уравнений, разложения и обращение матриц

❒ **chol** — разложение Холецкого положительно определенных эрмитовых (симметричных) матриц.

- `R=chol(A)` — возвращает верхнюю треугольную матрицу R такую, что `R'*R=A`. Если входной аргумент не является положительно определенной матрицей, то выводится сообщение об ошибке.
- `[R,p]=chol(A)` — второй дополнительный выходной аргумент позволяет избежать сообщения об ошибке в случае неположительно определенной A, возвращая в `p` целое положительное число, а в R — верхнюю треугольную матрицу такую, что `R'*R=A(1:p-1,1:p-1)`. В случае положительно определенной матрицы `p=0` и результат аналогичен `R=chol(A)`.

Функция `chol` применима к разреженным матрицам.

См. разд. "Факторизация матриц" главы 16.

❒ **inv** — обращение матрицы.

`B=inv(A)` — возвращает матрицу, обратную к квадратной матрице A. В случае плохой обусловленности A выдается предупреждение.

Задание в качестве входного аргумента символической матрицы приводит к поиску обратной к ней также в символической форме.

См. разд. "Задачи линейной алгебры" главы 16.

- **lscov** — решение системы линейных уравнений с возмущенной по нормальному закону правой частью.
  - `X=lscov(A,B,V)` — возвращает вектор *X*, являющийся решением системы линейных алгебраических уравнений `A*X=B+E`, где элементы матрицы *E* распределены по нормальному закону с нулевым средним и ковариационной матрицей *V*. Должно выполняться условие `m>n`, где `size(A)=[m n]`.
  - `[X,DX]=lscov(A,B,V)` — возвращает стандартную ошибку в `DX`.
- **lu** — LU-разложение квадратной матрицы.
  - `[L,U]=lu(A)` — возвращает верхнюю треугольную матрицу *U* и матрицу *L*, которая может быть сведена к нижней треугольной перестановками.
  - `[L,U,P]=lu(A)` — дополнительно возвращает матрицу перестановок *P* такую, что `P*A=L*U`.
  - `[L,U]=lu(A,tresh)` — входной аргумент `tresh` (из отрезка [0, 1]) позволяет управлять процессом выбора главного элемента при нахождении LU-разложения разреженных матриц. Перестановка производится, если модуль диагонального элемента в `tresh` раз меньше модуля любого поддиагонального элемента в столбце.
- **lsqnonneg** — нахождение положительного решения системы линейных алгебраических уравнений (необязательно с квадратной матрицей) методом наименьших квадратов.
  - `X=lsqnonneg(C,d)` — возвращает вектор *X* с неотрицательными компонентами, которые минимизируют `norm(C*X-d)`.
  - `X=lsqnonneg(C,d,X0)` — вектор *X0>0* используется в качестве начального приближения.
  - `X=lsqnonneg(C,d,X0,options)` — процесс вычислений управляется при помощи задания параметров в структуре `options` функцией `optimset`.

    См. разд. "Параметры оптимизации" главы 17.
  - `[X,resnorm]=lsqnonneg(...)` — в выходном аргументе `resnorm` возвращается квадрат нормы невязки `norm(C*X-d)^2`.
  - `[X,resnorm,residual]=lsqnonneg(...)` — в выходном аргументе `residual` возвращается невязка `C*X-d`.
  - `[X,resnorm,residual,exitflag] = lsqnonneg(...)` — значение единица выходного аргумента `exitflag` свидетельствует об успешном нахождении решения, а ноль означает, что превышено максимально допустимое число итераций.

- `[X,resnorm,residual,exitflag,output]=lsqnonneg(...)` — структура `output` содержит информацию о процессе вычислений.
- `[X,resnorm,residual,exitflag,output,lambda]=lsqnonneg(...)` — возвращает двойственный вектор `lambda`, `lambda(i)<=0`, если `X(i)` приближенно равно нулю и `lambda(i)=0`, если `X(i)>0`.

❒ **`pcg`** — предобусловленный и обычный метод сопряженных градиентов.

- `x=pcg(A,b)` — решение системы линейных алгебраических уравнений `A*x=b` методом сопряженных градиентов. Здесь A — симметричная положительно определенная матрица. Начальным приближением является нулевой вектор. Сходимость считается достигнутой, если в процессе итераций `norm(b-A*b)/NORM(b)<1e-6`. Число итераций по умолчанию `min(n,20)`. По окончании работы выводится сообщение о нахождении решения или о причине останова вычислений.
- `pcg(A,b,tol)` — в дополнительном входном аргументе `tol` задается точность вычислений вместо `1e-6`, установленной по умолчанию.
- `pcg(A,b,tol,maxit)` — задание максимально допустимого числа итераций.
- `pcg(A,b,tol,maxit,M)` — решение системы `A*x=b` предобусловленным методом сопряженных градиентов. В качестве предобусловливателя используется матрица M. Вместо матрицы M можно задать имя файл-функции, эффективно решающей систему линейных уравнений с матрицей M.
- `pcg(A,b,tol,maxit,M1,M2)` — предобуславливатель задается в факторизованной форме `M1*M2`.
- `pcg(A,b,tol,maxit,m1,m2,x0)` — указание `x0` в качестве начального приближения.

❒ **`pinv`** — нахождение псевдообратной матрицы.

- `P=pinv(A)` — возвращает матрицу P такую, что `size(P)=size(A')`, `A*P*A=A`, `X*P*X=P`, а `A*P` и `P*A` являются эрмитовыми.
- `P=pinv(A,tol)` — производит вычисления с заданной точностью.

❒ **`qr`** — QR-разложение матрицы.

- `[Q,R]=qr(A)` — нахождение верхней треугольной матрицы R (`size(R)=size(A)`) и унитарной Q (`Q'*Q=eye(size(Q))`) таких, что `A=Q*R`.
- `[Q,R,E] = qr(A)` — дополнительно возвращает матрицу перестановок `E`, `A*E=Q*R`.
- `[Q,R]=QR(A,0)` — если `size(A)=[m n]` и `m > n`, то возвращаются только первые `n` столбцов Q.
- `[Q,R,E]=QR(A,0)` — то же, что и `[Q,R]=QR(A,0)`, но возвращается матрица перестановок E такая, что `Q*R=A(:,E)`.

Вычисление функций от матриц

- **expm** — матричная экспонента, использование: `F=expm(A)`.
- **funm** — вычисление произвольной функции от матрицы.
  - `F=funm(A,'funname')` — вычисление функции от матрицы *A*, объявленной в файл-функции `funname`.

    Если исходная матрица имеет близкие собственные значения, то выдается предупреждение о том, что результат может быть найден неточно. Для эрмитовых (симметричных) положительно определенных матриц результат, как правило, получается достаточно точный.
  - `[F,errest]=funm(A,'funname')` — выходной аргумент `errest` содержит грубую оценку результата.
- **logm** — матричный логарифм.
  - `L=logm(A)` — вычисление логарифма матрицы *A*. Если *A* имеет отрицательные собственные значения, то результат будет комплексным. Если результат не может быть найден достаточно точно, то выдается предупреждение (см. `funm`).
  - `[L,esterr]=logm(A)` — выходной аргумент `errest` содержит грубую оценку результата.
- **sqrtm** — квадратный корень из матрицы.
  - `X=sqrtm(A)` — возвращает матрицу *X* такую, что `X*X=A`. Если *A* является вырожденной матрицей, то выдается предупреждение.
  - `[X,resnorm]=sqrtm(A)` — возвращает в `reznorm` относительную погрешность `norm(A-X^2,'fro')/norm(A,'fro')`.

Решение различных математических задач

Поиск корней

- **fsolve** — решение нелинейных уравнений и систем вида `f(x)=0`.

  Левая часть уравнения или системы `f(x)=0` должна быть запрограммирована в файл-функции `funname`.

  См. разд. "Решение нелинейных уравнений" главы 17.
  - `x=fsolve(funname,x0)` — возвращает решение, используя `x0` в качестве начального приближения.
  - `x=fsolve(funname,x0,options)` — процесс решения управляется параметрами, задаваемыми в структуре `options`.

- `x=fsolve(funname,x0,options,p1,p2,...)` — решение нелинейных уравнений и систем при фиксированных значениях параметров `p1`, `p2`, ..., от которых зависит левая часть системы `f(x,p1,p2,...)`.
- `[x,fval]=fsolve(funname,x0,...)` — возвращает значение `f`, вычисленное от приближенного решения.
- `[x,fval,exitflag]=fsolve(funname,x0,...)` — значение выходного аргумента `exitflag` содержит информацию о завершении вычислений. Если `exitflag>0`, то процесс сошелся и решение найдено, если `exitflag<0`, то вычислительный процесс оказался расходящимся, а `exitflag=0` свидетельствует о прекращении вычислений из-за превышения максимально допустимого количесва вычислений функции `f`.
- `[x,fval,exitflag,output]=fsolve(funname,x0,...)` — структура `output` содержит подробную информацию о ходе вычислений.
- `[x,fval,exitflag,output,jacob]=fsolve(funname,x0,...)` — в выходной аргумент `jacob` заносится якобиан левой части системы, вычисленный от приближенного решения `x`.

❒ **`fzero`** — нахождение корня функции одной переменной `f(x)`.

Левая часть уравнения `f(x)=0` должна быть запрограммирована в файл-функции `funname`.

- `x=fzero(fun,x0)` — возвращает решение, используя `x0` в качестве начального приближения.
- `x=fzero(fun,[a b])` — возвращает решение на промежутке `[a,b]`, используя `x0` в качестве начального приближения. Предполагается, что `f(a)*f(b)<0`.

См. разд. "Решение произвольных уравнений" главы 6.

Следующие варианты вызова аналогичны `fsolve`:

- `x=fzero(fun,x0,options);`
- `x=fzero(fun,x0,options,p1,p2,...);`
- `[x,fval]= fzero(fun,...);`
- `[x,fval,exitflag,output] = fzero(...).`

Использование `exitflag` несколько отличается от `fzero`.

`[x,fval,exitflag]=fzero(...)` — если `exitflag>0`, то решение найдено, если `exitflag<0`, то либо не определен интервал, на границах которого функция имеет разные знаки, либо были получены `Inf`, `NaN` при вычислении функции.

- **roots** — вычисление всех корней полинома.

  `r=roots(p)` — вектор *r* содержит корни полинома, задаваемого вектором *p*.

  См. разд. "Вычисление всех корней полинома" главы 6.

- **solve** — символическое решение уравнений и систем.

  - `r=solve(f)` — нахождение символического решения уравнения, заданного строкой `f`. Входным аргументом может быть символическая функция. По умолчанию в качестве независимой переменной выбирается результат `findsym(f)`.
  - `r=solve(f,t)` — второй аргумент указывает на независимую переменную.
  - `r=solve(f1,f2,...,fn)` — решение системы уравнений, задаваемых строками `f1`, `f2`,..., `fn`. Входными переменными могут быть символические функции. По умолчанию неизвестными переменными являются те, которые возвращает `findsym`. Поля структуры `r` содержат компоненты решения.
  - `[r1, r2,...,rn]=solve(f1,f2,...,fn)` — решение записывается в символические переменные `r1`, `r2`,..., `rn`.
  - `r=solve(f1,f2,...,fn,t1,t2,...,tn)` — дополнительные входные аргументы `t1`, `t2`, ..., `tn` указывают на переменные, подлежащие определению.
  - `[r1, r2,...,rn]=solve(f1,f2,...,fn,t1,t2,...,tn)` — решение записывается в символические переменные `r1`, `r2`,..., `rn`.

  См. разд. "Решение уравнений и систем" главы 18.

Интерполяция

Примеры, связанные с интерполяцией и приближением данных, приведены в разд. "Полиномы и интерполяция" главы 6.

- **polyfit** — приближение табличной функции одной переменной полиномом заданного порядка по методу наименьших квадратов.
- **griddata** — приближение неравномерно распределенных трехмерных данных поверхностью, построенной на регулярной сетке.

  - `ZI=griddata(x,y,z,XI,YI)` — построение поверхности, наилучшим образом проходящей через точки с координатами `(x(i),y(i),z(i))`. Векторы *x*, *y* и *z* содержат неравномерно распределенные данные. Матрицы *XI* и *YI* задают равномерную сетку (они могут быть сгенерированы при помощи `meshgrid`). Результатом является матрица со значениями интерполирующей функции в узлах равномерной сетки.

- `ZI=griddata(x,y,z,XI,YI,'linear')` — приближение линейными функциями.
- `ZI=griddata(x,y,z,XI,YI,'cubic')` — приближение кубическими функциями.
- `ZI=griddata(x,y,z,XI,YI,'nearest')` — приближение по ближайшему соседу.

❐ **interp1, interp2, interp3, interpn** — интерполяция одномерных, двумерных, трехмерных и многомерных данных различными способами.

❐ **interpft** — одномерная интерполяция с использованием быстрого преобразования Фурье.

`y=interpft(x,n)` — предполагается, что вектор *x* содержит равноотстоящие друг от друга на шаг `dx` элементы, `length(x)=m`. Находится преобразование Фурье от *x*, затем оно дополняется точками с шагом `dx*m/n`. Вычисляется обратное преобразование Фурье и возвращается в векторе *y*.

❐ **spline** — интерполяция кубическими сплайнами.

`yi=spline(x,y,xi)` — табличные данные, заданные векторами *x* и *y*, интерполируются кубическими сплайнами. Возвращаемый вектор *yi* содержит значения сплайна в абсциссах, определяемых вектором *xi*.

Минимизация и оптимизация

Функции MatLab, предназначенные для решения задач минимизации и оптимизации, входят в состав ToolBox Optimization.

См. главу 17, где приведены формулировки основных задач и примеры использования функций.

❐ **fgoalattain** — решение задачи о достжении границы.

❐ **fminbnd** — нахождение локального минимума функции одной переменной на заданном интервале.

См. также разд. "Минимизация функции одной переменной" главы 6.

❐ **fmincon** — решение задач нелинейного программирования.

❐ **fminimax** — решение минимаксной задачи.

❐ **fminsearch** — поиск локального минимума функции нескольких переменных.

См. также разд. "Минимизация функции нескольких переменных" главы 6.

❐ **fminunc** — поиск минимума нелинейной функции без ограничения на переменные.

- **linprog** — решение задач линейного программирования.
- **lsqcurvefit** — подбор параметров.
- **lsqlin** — метод наименьших квадратов для решения систем линейных уравнений с линейными ограничениями на решение.
- **lsqnonlin** — нелинейный подбор параметров.
- **optimget** — получение параметров, определяющих работу функций минимизации и оптимизации.
- **optimset** — задание параметров, управляющих минимизацией и оптимизацией.
- **quadprog** — решение задач квадратичного программирования.

Дифференцирование и конечные разности

- **del2** — вычисление разностного аналога оператора Лапласа.

  Предполагается, что матрица *U* содержит значения некоторой функции в точках сетки.

  - `L=del2(U)` — возвращается матрица со значениями для внутрениих узлов `L(i,j)=0.25*(U(i+1,j)+U(i-1,j)+U(i,j+1)+U(i,j-1))-U(i,j)`

    При вычислении граничных значений используется кубическая экстраполяция. По умолчанию сетка квадратная с шагом, равным единице. Для задания сеток с другими шагами следует применять вызовы: `L=del2(U,h)`, `L=del2(U,hx,hy)`.

    Генерация прямоугольной сетки производится при помощи функции `meshgrid`.

    См. разд. "Графики функций двух переменных" главы 2.

  - `L=del2(U,hx,hy,hz,...)` — аппроксимация оператора Лапласа и в многомерном случае. Для построения многомерных сеток предназначена `ndgrid`.

- **diff** — нахождение конечных разностей и символическое вычисление производных.

  - `D=diff(X)` — по вектору *X* строит вектор `D=[X(2)-X(1),...,X(n)-X(n-1)]`, где `n=length(X)`, причем `length(D)=n-1`. Если входной аргумент является матрицей, то происходит вычисление для каждого столбца `X`.
  - `Dk=diff(X,k)` — вычисление конечных разностей `k`-го порядка. Например, для `k=3` функции `diff(X,3)` и `diff(diff(diff(X)))` приводят к одинаковым результатам.

- `f1=diff(f)` — нахождение первой производной в аналитическом виде от символической функции `f`. По умолчанию в качестве независимой переменной выбирается результат `findsym(f)`. Выходной аргумент является символической функцией.
- `f1=diff(f,t)` — нахождение первой производной в аналитическом виде от символической функции `f` по переменной `t`. Переменная `t` должна быть объявлена как символическая при помощи `syms` или `sym`.

Аналитическое выражение для производной `n`-го порядка возвращается при обращениях: `diff(f,n)`, `diff(f,t,n)`.

См. разд. "Пределы, дифференцирование и интегрирование" главы 18.

❒ **`gradient`** — вычисление градиента сеточной функции.

- `[FX,FY]=gradient(F)` — возвращает компоненты градиента для сеточной функции, значения которой в узлах сетки представлены в матрице *F*. По умолчанию сетка считается квадратной с единичным шагом. Для вычисления градиента на произвольной прямоугольной сетке следует применять обращение

  `[FX,FY]=gradient(F,hx,hy).`
- `[FX,FY,FZ,...]=del2(F,hx,hy,hz,...)` — нахождение компонентов градиента функции нескольких переменных. Для построения многомерных сеток предназначена `ndgrid`.

Интегрирование

❒ **`dblquad`** — вычисление двойных интегралов.

- `result = dblquad('fun',inmin,inmax,outmin,outmax)`
- `result = dblquad('fun',inmin,inmax,outmin,outmax,tol)`
- `result = dblquad('fun',inmin,inmax,outmin,outmax,tol,method)`

См. разд. "Вычисление двойных интегралов" главы 6.

❒ **`int`** — нахождение определенных и неопределенных интегралов в символическом виде.

`int(f)`, `int(f,t)`, `int(f,a,b)`, `int(f,t,a,b)`.

См. разд. "Пределы, дифференцирование и интегрирование" главы 18.

❒ **`quad`**, **`quad8`** — вычисление определенных интегралов по квадратурным формулам Симпсона и Ньютона—Котеса с автоматическим подбором шага интегрирования.

- `q = quad('f',a,b)`
- `q = quad('f',a,b,tol)`

- `q = quad('f',a,b,tol,trace)`
- `q = quad('f',a,b,tol,trace,p1,p2,...)`

См. разд. "Вычисление определенных интегралов" главы 6.

Решение дифференциальных уравнений и систем

Аппроксимация решения стационарных и нестационарных задач, описываемых дифференциальными уравнениями в частных производных, по методу конечных элементов, реализована в функциях ToolBox PDE.

См. главу 15.

- **`bvp4c`** — численное решение граничных задач для обыкновенных дифференциальных уравнений произвольного порядка и систем (только начиная с версии 6.0).

  См. разд. "Решение дифференциальных уравнений" главы 6.

- **`dsolve`** — аналитическое решение дифференциальных уравнений и систем.

  См. разд. "Решение дифференциальных уравнений и систем" главы 18.

- **`ode45`**, **`ode23`**, **`ode113`**, **`ode15s`**, **`ode23s`**, **`ode23t`**, **`ode23tb`** — численное решение задачи Коши для дифференциальных уравнений и систем произвольных порядков. Задание параметров, управляющих вычислительным процессом, производится при помощи `odeset`.

  См. разд. "Решение граничных задач" главы 6.

Графика и визуализация данных

Двумерные графики

- **`bar`** — вертикальная столбцевая диаграмма матричных или векторных данных, примеры:

  `bar(rand(1,10))`, `bar(rand(1,10), 1.2)`, `bar(rand(3,4))`.

- **`barh`** — горизонтальная столбцевая диаграмма матричных или векторных данных, примеры:

  `barh(rand(1,10))`, `barh(rand(1,10), 1.2)`, `barh(rand(3,4))`.

- **`comet3`** — анимированный график плоской линии.

  - `comet(x,y)` — отображение анимированного графика в виде движения кометы по кривой, проходящей через точки с координатами `(x(i),y(i))`.

- `comet(x,y,p)` — дополнительный третий аргумент задает длину хвоста кометы `p*length(Y)`, по умолчанию используется `p=0.1`.

❒ **`fill`** — построение двумерного закрашенного многоугольника.

- `fill(x,y,c)` — векторы *x* и *y* одинаковой длины содержат координаты вершин многоугольника. В случае незамкнутого многоугольника последняя вершина соединяется с первой. Цвет определяется значением третьего входного аргумента `c`. Значение `c` может быть константой из ряда `'r','g','b', 'c','m','y','w','k'`, или вектором из трех элементов в формате `[r g b]`, например:

```
>> fill([-3 0 1 2 0 -5], [2 3 2 1 9 -1], 'c')
>> fill([-3 0 1 2 0 -5], [2 3 2 1 9 -1], [0.4 0.2 0.1])
```

  Плавное изменение цвета заливки в пределах текущей палитры цвета требует указания вектора значений, соответствующих цвету вершин, т. е. `size(c)=size(x)`. Указанные значения сначала масштабируются (см. `caxis`), а затем происходит билинейная интерполяция цвета внутри многоугольной области.

- `fill(X,Y,C)` — построение сразу нескольких многоугольников, число многоугольников равно столбцам матриц *X* и *Y* (предполагается, что матрицы одинаковых размеров). Четвертый аргумент *C*, задающий цвет заливки, может быть вектором, длина которого совпадает с числом столбцов в матрицах *X* и *Y*. Указание матрицы *C*, такой что `size(C)=size(X)`, приводит к плавной заливке каждого многоугольника.

Одним из входных аргументов (`X` или `Y`) может быть матрица, а вторым — вектор, число элементов которого равно числу строк матрицы. Такое обращение к `fill` эквивалентно обычному, в котором вместо вектора указана матрица, столбцы матрицы одинаковы и совпадают с вектором.

Функция `fill` допускает построение многоугольных объектов при помощи указания соответствующих троек аргументов с координатами и цветом, например:

```
fill3(x1,y1,'y',x2,y2,'g')
```

Выходной аргумент, возвращаемый `fill`, является вектором указателей на все построенные многоугольные объекты типа `patch`.

```
h=fill(...)
```

Свойства каждого из графических объектов могут быть изменены в дальнейшем при помощи `set`.

См. главу 9.

❒ **`hist`** — гистограмма матричных или векторных данных.

- `hist(y)` — отображение гистограммы данных, записанных в векторе *y*, для построения гистограммы используется десять интервалов равной длины.

- `n=hist(y)` — выходной аргумент `n` является вектором и содержит число элементов из *y*, попавших в каждый из десяти интервалов. Гистограмма не отображается.
- `hist(y,m)` — отображение гистограммы данных, записанных в векторе *y*, для построения гистограммы используется `m` интервалов равной длины.
- `hist(y,x)` — отображение гистограммы данных, записанных в векторе *y*, для построения гистограммы используются интервалы, центры которых определяются значениями элементов вектора *x*.

❒ **`loglog`**, **`semilogx`**, **`semilogy`** — построение графиков в логарифмическом и полулогарифмических масштабах. Используются так же, как `plot`.

❒ **`pie`** — отображение данных в виде круговой диаграммы.

- `pie(x)` — площадь сектора круговой диаграммы, отвечающего `x(i)`, пропорциональна `x(i)/sum(x)`. Если `sum(x)<1`, то получается неполная круговая диаграмма.
- `pie(x,parts)` — ненулевые компоненты массива `parts` (входные аргументы должны быть одинаковой длины `length(parts)=length(x)`) соответствуют секторам, немного выдвинутым из круга диаграммы.
- `pie(...,labels)` — при построении круговой диаграммы добавляются надписи рядом с каждым сектором. Ячейки массива `labels` должны содержать строки с текстом надписей.
- `h=pie(...)` — возвращает массив `h` с указателями на графические объекты `patch` и `text`, образующие гистограмму.

❒ **`plot`** — визуализация функций одной переменной, векторных и матричных данных.

- `plot(y)` — график зависимости значения элементов вектора с вещественными числами от их номеров, точки с координатами `(i,y(i))` соединяются отрезками прямых. Если среди элементов *y* есть комплексные, то данная команда аналогична вызову `plot` с двумя входными аргументами (см. ниже): `plot(real(y),imag(y))`.
- `plot(x,y)` — график зависимости элементов вектора *y* от элементов вектора `x`, точки с координатами `(x(i),y(i))` соединяются отрезками прямых, пример:

```
>> x=[-pi:pi/30:pi];
>> y=sin(x);
>> plot(x,y)
```

Дополнительный строковый аргумент задает цвет и стиль линии, и тип маркеров, например: `plot(x,y,'r:o')`.

Возможно построение нескольких графиков на одних осях, указывая пары вектора значений аргумента и вектора значений функции:

`plot(x,y1,x,y2,x,y3,...)` или `plot(x1,y1,x2,y2,x3,y3,...)`. С каждой парой может быть указан строковый аргумент.

См. разд. "Изменение свойств линий" главы 3.

- `plot(x,y,'PropName','PropVale','PropName','PropVale',...)` — указание свойств линии каждого графика парами, содержащими название свойства и его значение, пример:

```
>> plot(x,y,'Marker','o','MarkerSize',5,'MarkerEdgeColor','g',
'MarkerFaceColor','y')
```

Свойства линий описаны в разд. "Свойства линий и поверхностей" главы 9.

- `h=plot(...)` — возвращает вектор указателей на графические объекты — линии.

❒ **`polar`** — построение графика в полярных координатах.

- `polar(theta, rho)` — отображение зависимости элементов вектора `rho` от соответствующих значений элементов вектора `theta`, заданных в радианах. На график наносится сетка.
- `plolar(theta,rho,'r:o')` — свойства линии и маркеров определяются дополнительным строковым аргументом (см. `plot`).

Трехмерные графики

❒ **`bar3`** — вертикальная столбцевая трехмерная диаграмма матричных и векторных данных, пример: `bar3(rand(3,4))`.

`bar3h` — горизонтальная столбцевая трехмерная диаграмма матричных и векторных данных, пример: `bar3h(rand(3,4))`.

❒ **`comet3`** — анимированный график трехмерной линии.

- `comet3(x,y,z)` — отображение анимированного графика в виде движения кометы по кривой, проходящей через точки с координатами `(x(i),y(i),z(i))`.
- `comet3(x,y,z,p)` — дополнительный четвертый аргумент задает длину хвоста кометы `p*length(Z)`, по умолчанию используется `p=0.1`.

❒ **`contour`** — построение линий уровня функции двух переменных.

- `contour(X,Y,Z)` — отображение функции, значения которой на сетке, определяемой матрицами *X* и *Y*, записаны в матрицу *Z*. Линии уровня отображаются при автоматически подбираемых значениях функции.
- `contour(Z)` — в качестве области построения выбирается прямоугольник: `x=1:n`, `y=1:m`, где `[n m]=size(Z)`.
- `contour(Z,N)` и `contour(X,Y,Z,N)` — отображаются линии уровня, соответствующие `N` постоянным значениям исследуемой функции, число линий уровня может быть больше `N`.

- `contour(Z,vec)` и `contour(X,Y,Z,vec)` — линии уровня строятся при значениях, являющихся элементами вектора `vec`. Число линий уровня равно `length(vec)`. Для отображения только одной линии уровня, на которой функция принимает заданное значение `v`, следует использовать вызовы: `contour(Z,[v v])` или `contour(X,Y,Z,[v v])`.

- `[C,h] = contour(...)` — выходными аргументами являются матрица с информацией о линиях уровня (см. `contourc`) и вектор с указателями на построенные линии (многоугольные объекты типа `patch`). Возвращаемые указатели позволяют получить доступ к свойствам линий уровня, в частности, цвет границы многоугольника определяется значением свойства `EdgeColor`, а не `Color`, как у линии (объекта `line`). Пример использования указателей:

```
>> [C,h]=contour(X,Y,Z,2);
>> set(h(1),'EdgeColor','g')
>> set(h(2),'Marker','o', 'MarkerSize', 4)
>> set(h(3),'LineWidth',2)
>> set(h(4),'LineStyle',':')
```

  Свойство `UserData` каждой из линий уровня содержит соответствующее значение функции:

```
>> get(h,'UserData')
ans =
  [-2.2049]
  [-2.2049]
  [ 2.2049]
  [ 2.2049]
```

  Информация, содержащаяся в матрице *C*, позволяет расположить рядом с каждой линией уровня соответствующее значение функции при помощи `clabel`.

  См. разд. "Контурные графики" главы 3.

- `contour(X,Y,Z,'k:')`, `[C,h]=contour(X,Y,Z,'k:')` — цвет и стиль всех линий уровня задается при помощи дополнительного строкового входного аргумента (см. `plot`).

❐ **`contourc`** — получение информации о линиях уровня функции двух переменных.

- `C=contourc(X,Y,Z)` — матрица *C* является блочной `C=(C1 C2 ...)`, число блоков совпадает с числом линий уровня, каждый блок имеет следующий формат (на примере `C1`):

```
C1 = [level1 x1 x2 x3 ...; pairs1 y1 y2 y3]
```

Значение функции на данной линии уровня содержится в `levels`, а число пар точек, описывающих линию уровня, как многоугольник (объект `patch`) — в `pairs1`. Вершинами многоугольника являются точки `(x1,y1)`, `(x2,y2)` и т. д.

Способы задания входных аргументов `contourc` совпадают с `contour`.

- **`contourf`** — залитый цветами контурный график, использование аналогично функции `contour`.
- **`cylinder`** — отображение цилиндра и генерация точек, лежащих на поверхности цилиндра.
  - `cylinder` — построение части цилиндрической поверхности единичного радиуса и высоты.
  - `cylinder(r,n)` — построение части цилиндрической поверхности единичной высоты. Входной аргумент `r` является вектором значений радиусов поверхности в зависимости от высоты, а `n` — число точек для построения окружности отрезками прямых, пример: `cylinder([0.1 0.3 0.5 1.3 1.8 1.6 0.1],100)`.
  - `cylinder(r)` — по умолчанию используется двадцать точек вдоль окружности.
  - `[X,Y,Z]=cylinder(...)` — выходными аргументами являются матрицы, определяющие поверхность. Сама поверхность не отображается, ее можно получить при помощи, например: `mesh(X,Y,Z)`, `surf(X,Y,Z)`, `surfl(X,Y,Z)`.
- **`fill3`** — рисование закрашенного цветом многоугольника в трехмерном пространстве.
  - `fill3(x,y,z,c)` — рисует закрашенный цветом многоугольник. Вершины многоугольника `(x(i),y(i),z(i))` содержатся в трех первых входных аргументах, причем `length(x)=length(y)=length(z)`. Многоугольник должен иметь замкнутую границу, поэтому при необходимости последняя точка соединяется с первой. Четвертый входной аргумент определяет цвет и способ заливки.

    Заливка многоугольника одним цветом происходит при указании одного из сокращений для цвета: `'r','g','b', 'c','m','y','w','k'`, или вектора из трех элементов в формате `[r g b]`, например:

    ```
    >> fill3([1 2 -3 1],[0 1 1 0],[-1 7 0 -1],'r')
    >> fill3([1 2 -3 1],[0 1 1 0],[-1 7 0 -1],[0.8 0.9 0.3])
    ```

    Плавное изменение цвета заливки в пределах текущей палитры цвета требует указания вектора значений, соответствующих цвету вершин, т. е. `size(c)=size(a)`. Указанные значения сначала масштабируются

(см. `caxis`), а затем происходит билинейная интерполяция цвета внутри многоугольной области, например:

```
>> fill3([1 2 -3 1],[0 1 1 0],[-1 7 0 -1],[0.8 0.9 0.3 0.7])
```

- `fill3(X,Y,Z,C)` — построение сразу нескольких многоугольников, число многоугольников равно столбцам матриц *X*, *Y* и *Z* (предполагается, что `size(X)=size(Y)=size(Z)`). Четвертый аргумент `C`, задающий цвет заливки, может быть вектором, длина которого совпадает с числом столбцов в матрицах *X*, *Y* и Z. Указание матрицы *C*, такой что `size(C)=size(X)`, приводит к плавной заливке каждого многоугольника.
- `fill3(X,Y,Z,C,'PropName','PropVale','PropName','PropVale',...)` — пары `'PropName','PropVale'` позволяют задать всевозможные свойства многоугольника как полигонального объекта `patch`., например:

```
>> fill3([1 2 -3 1],[0 1 1 0],[-1 7 0 -
1],'y','EdgeColor','g','LineWidth',4)
```

Функция `fill3` допускает построение многоугольных объектов при помощи указания соответствующих четверок аргументов с координатами и цветом, например:

```
fill3(x1,y1,z1,'y',x2,y2,z2,'g')
```

Выходной аргумент, возвращаемый `fill3`, является вектором указателей на все построенные многоугольные объекты типа `patch`.

```
h=fill3(...)
```

Свойства каждого из графических объектов могут быть изменены в дальнейшем при помощи `set`.

См. главу 9.

❐ **`hidden`** — удаление или отображение частей каркасных поверхностей, скрытых от наблюдателя.

- `hidden on` — удаление невидимых частей.
- `hidden off` — отображение невидимых частей.

❐ **`plot3`** — построение линий в трехмерном пространстве.

- `plot3(x,y,z)` — отображение линии, проходящей через точки с координатами `(x(i),y(i),z(i))`, где *x*, *y* и *z* являются векторами одинаковой длины. Дополнительный строковый аргумент (см. `plot`) задает цвет и стиль линии, а также тип маркеров, например: `plot(x,y,z,'r:o')`.
- `plot3(X,Y,Z)` — отображение линий, проходящих через точки с координатами `(X(i,:),Y(i,:),Z(i,:))`, где число линий совпадает с числом столбцов матриц. Матрицы *X*, *Y* и *Z* должны быть одинаковых размеров. Возможен вызов `plot` с четвертым дополнительным аргументом,

определяющим цвет и стиль сразу всех линий и тип маркеров. Для установки свойств линий по отдельности следует использовать обращение: `plot3(x1,y1,z1,s1,x2,y2,z2,s2,...)`, где входные аргументы `x1`, `y1`, `z1`, `x2`, `y2`, `z2` и т. д. могут быть либо векторами, либо матрицами одинаковых размеров, а строковые аргументы `s1`, `s2` и т. д. задают стиль линий.

- `plot(x,y,z,'PropName','PropVale','PropName','PropVale',...)` — указание свойств линии каждого графика парами, содержащими название свойства и его значение (см. `plot`).
- `h=plot3(...)` — выходной аргумент — вектор h содержит указатели на все созданные линии.

❒ **`quiver3`** — визуализация вектор-функции от трех переменных, определенной на некоторой поверхности.

- `quiver3(X,Y,Z,U,V,W)` — построение вектор-функции $[u,v,w]$, где $u=u(x,y,z)$, $v=v(x,y,z)$, $w=w(x,y,z)$. Матрицы X, Y и Z описывают поверхность, а U, V и W содержат компоненты вектор-функции в соответствующих точках пространства. Требуется, чтобы `size(X)=size(Y)=size(Z)=size(U)=size(V)=size(W)`. Происходит автоматическое масштабирование длины стрелок, представляющих вектор-функцию в каждой точке для обеспечения наилучшего вида графика. Пример использования:

```
>> [X,Y,Z]=sphere;
>> [U,V,W] = surfnorm(X,Y,Z);
>> mesh(X,Y,Z)
>> hold on
>> quiver3(X,Y,Z,U,V,W)
```

- `quiver3(X,Y,Z,U,V,W,s)` — после автоматического масштабирования длин стрелок их длина увеличивается в `s` раз. Значение `s=0` предотвращает предварительное масштабирование, и длина стрелок определяется соответствующими элементами матриц X, Y и Z.
- `quiver3(Z,U,V,W)`, `quiver3(Z,U,V,W,s)` — построение вектор-функции на поверхности, определяемой матрицей Z.
- `quiver3(...,'g*-.')` — дополнительный последний аргумент позволяет указать стиль и цвет линии, а также тип маркера (см. `plot`).
- `h=quiver3(...)` — выходной вектор h содержит указатели, `h(1)` является указателем на линии, `h(2)` — на маркеры. Свойства линий и маркеров можно затем изменить при помощи `set`, например:

```
>> h=quiver3(X,Y,Z,U,V,W)
>> set(h(1),'Color','r')
>> set(h(2),'Color','k')
```

- **slice** — визуализация функции трех переменных при помощи отображения значений на различных плоскостях или поверхностях.
  - `slice(X,Y,Z,V,sx,sy,sz)` — значения функции должны быть вычислены в точках трехмерной сетки, определяемой матрицами *X*, *Y* и *Z* одинаковых размеров, и записаны в трехмерный массив `V`. Векторы *sx*, *sy* и *sz* задают плоскости, которые образуют срезы трехмерного пространства. Если, например `sx(1)=-2.5`, то на плоскости `x=-2.5` будет отображаться залитый цветом контурный график исследуемой функции. Цвет в каждой точке определяется при помощи линейной интерполяции. Исследование поведения функции $(x^2 + y^2)z$ в объеме $x \in [-1, 1]$, $y \in [-2, 2]$, $z \in [-1, 1]$ на срезах, образуемых плоскостями $x = \pm 0.4$, $y = 0$, $z = 0, \pm 0.8$, производится при помощи следующих команд:

```
>> [X,Y,Z] = meshgrid(-1:0.1:1, -2:0.2:2, -1:0.1:1);
>> V=(x.^2+y.^2).*z;
>> slice(X,Y,Z,V,[-0.4 0.4], [0], [-0.8 0 0.8])
```

    Объем можно срезать не только плоскостями, но и различными поверхностями. Матрицы, задающие поверхность, указываются вместо векторов во входных аргументах `slice`. Отображение функции, определенной выше, на сфере производится при помощи следующих команд:

```
>> [XI,YI,ZI]=sphere;
>> slice(X,Y,Z,V,XI,YI,ZI)
```

    Первые три входные аргумента, задающие координаты точек со значениями `V`, могут быть опущены: `slice(V,sx,sy,sz)`, `slice(V,xi,yi,zi)`. Предполагается, по умолчанию, что `X=1:n`, `Y=1:m`, `Z=1:p`, где `[n m p]=size(V)`.
  - `slice(...,method)` — способ интерполяции определяется в последнем дополнительном входном аргументе и может быть: `'nearest'`, `'linear'` (по умолчанию) или `'cubic'`.
  - `h=slice(...)` — в выходном аргументе возвращается указатель на созданную поверхность.
- **sphere** — отображение сферы и генерация точек, лежащих на поверхности сферы.
  - `sphere` (без параметров) — построение в графическом окне единичной сферы.
  - `[X,Y,Z]=sphere` — генерация матриц *X*, *Y* и *Z*, соответствующих единичной сфере, причем `size(X)=size(Y)=size(Z)=[21 21]`. Сама поверхность не отображается, ее можно получить при помощи, например: `mesh(X,Y,Z)`, `surf(X,Y,Z)`, `surfl(X,Y,Z)`.

- `[X,Y,Z]=sphere(n)` — выходные матрицы имеют размеры `n+1` на `n+1`, поверхность не отображается.
- `sphere(n)` — построение в графическом окне единичной сферы с использованием `n+1` точек по каждому из направлений осей координат.

❒ **`stem3`** — отображение трехмерных данных в виде черенковой диаграммы.

- `stem3(Z)` — построение зависимости `Z(i,j)` от `i` и `j`. Каждое значение `Z(i,j)` представляется в виде отрезка, начинающегося на плоскости `z=0`, и оканчивающегося круглым маркером.
- `stem3(x,y,z)` — отрезки высоты `z(i)` начинаются в точках плоскости `z=0` с координатами `(x(i),y(i))`, например:

```
>> t=[0:pi/10:2*pi];
>> x=sin(t);
>> y=cos(t);
>> z=exp(-x-y);
>> stem3(x,y,z)
```

- `stem3(...,'filled')`— отрезки оканчиваются сплошными круглыми маркерами. Указание четвертого дополнительного входного аргумента позволяет определить цвет и стиль линии и тип маркера (см. `plot`), например: `stem3(x,y,z,'r*')`.
- `h=stem3(...)` — выходной вектор *h* содержит указатели, `h(1)` является указателем на линии, `h(2)` — на маркеры. Свойства линий и маркеров можно затем изменить при помощи `set`, например:

```
>> h=stem3(x,y,z,'r*')
>> set(h(1),'Color','g')
>> set(h(2),'Color','b')
```

❒ **`waterfall`** — построение каркасной поверхности линиями по одному из направлений.

- `waterfall(X,Y,Z)` — отображается каркасная поверхность (аналогично `mesh`), но без линий, соответствующих столбцам матриц *X*, *Y* сетки. Линии, образующие поверхность, направлены вдоль оси `x`.
- `waterfall(Z)` — аналогично `mesh` с одним входным аргументом.

Для построения каркасной поверхности, линии которой соответствуют столбцам матриц, следует использовать транспонирование: `waterfall(X,Y,Z)`, `waterfall(Z)`.

❒ **`meshc`** — построение каркасной поверхности вместе с линиями уровня на плоскости `xy`. Использование аналогично `mesh`.

❒ **`meshz`** — построение каркасной поверхности вместе с линиями уровня на поверхности. Использование аналогично `mesh`.

❒ **mesh** — построение каркасной поверхности.

- `mesh(X,Y,Z)` — построение каркасной поверхности, высота в каждом узле каркасной сетки плоскости `xy` (сетка задается матрицами *X* и *Y*), содержится в соответствующем элементе матрицы *Z*. Матрицы *X*, *Y* и *Z* должны быть одинаковых размеров. Цвет линий изменяется в пределах текущей палитры в зависимости от высоты точек поверхности. Точка обзора устанавливается при помощи функции `view`.

  См. разд. "Поворот графика, изменение точки обзора" главы 3.

  Разметка осей по умолчанию выбирается так, чтобы обеспечить наилучший вид графика. Изменение разметки осей производится с использованием `axis`.

  Поверхность может соответствовать не только однозначной, но и многозначной функции. Соответствующие примеры приведены в *главе 3*.

  См. разд. "Построение параметрически заданных поверхностей и линий" главы 3.

- `mesh(x,y,Z)` — первые два входных аргумента могут быть векторами. В данном случае узлами каркасной поверхности являются точки `(x(j),y(i),Z(i,j))`, причем длины векторов должны соответствовать размерам матрицы: `size(Z)=[length(y) length(x)]`.
- `mesh(Z)` — в качестве векторов *x* и *y* (см. предыдущий вариант вызова) выбираются `x=[1:n]`, `y=[1:m]`, где `[m,n]=size(Z)`.
- `mesh(...,C)` — для определения цвета поверхности используется матрица *C* (см. функцию `surf`).
- `mesh(...,'PropName',PropValue,'PropName',PropValue)` — отображение поверхности, свойства которой принимают заданные значения, например:

```
>> [X,Y]=meshgrid(-3*pi:pi/5:3*pi,-5:0.5:4);
>> Z=sin(X).*(Y+5).*(4-Y);
>> mesh(X,Y,Z,'LineStyle','none','Marker','.')
```

- `h=mesh(...)` — возвращает указатель на построенную поверхность (объект `surface`). Свойства поверхности в дальнейшем изменяются при помощи `set`.

❒ **peaks** — функция, предназначенная для демонстрации и изучения графических возможностей MatLab. Генерируемые значения соответствуют масштабированному двумерному распределению Гаусса.

- `Z=peaks` — генерация матрицы значений `Z`, `size(Z)=[49 49]`.
- `Z=peaks(n)` — генерация матрицы значений `Z`, `size(Z)=[n n]`.
- `Z=peaks(v)` — генерация матрицы значений `Z`, `size(Z)=length(v)`.

- `Z=peaks(X,Y)` — генерация матрицы значений в узлах сетки, описываемой матрицами *X* и *Y*. Для получения матриц сетки удобно использовать `meshgrid`.

Вызов `peaks` без выходных аргументов приводит к отображению каркасной залитой цветом поверхности функции `peaks`: `peaks`, `peaks(n)`, `peaks(v)`, `peaks(X,Y)`.

Указание в качестве выходных аргументов матриц *X*, *Y* и *Z* приводит к записи координат узлов сетки в матрицы *X* и *Y* и значений peaks в *Z*.

❒ **`surf`** — построение залитой цветом каркасной модели. Входные аргументы с координатами узлов каркасной сетки и значениями высоты и матрица, определяющая цвет поверхности, задаются в таком же порядке, как и в `mesh` (см. выше), например: `surf(X,Y,Z,C)`.

Управление способом заливки цветом ячеек каркасной поверхности производится при помощи команды `shading`, задаваемой после вызова `surf`. Параметры `facetd` (по умолчанию) или `flat` задают постоянный цвет ячейки, а `shading` обеспечивает билинейную интерполяцию цвета между значениями в четырех узлах ячейки. Цвета узлов ячеек определяются значениями элементов матрицы *C*, такого же размера, как и остальные матрицы *X*, *Y* и *Z*. Процесс заливки цветом ячейки проще всего понять на примере поверхности, состоящей из одной ячейки:

```
>> [X,Y]=meshgrid([-1 1])
X =
  -1   1
  -1   1
Y =
  -1  -1
   1   1
>> Z=X+Y;
>> C=[0 0; 0 1];
>> colormap(jet)
>> surf(X,Y,Z,C)
>> shading interp
```

Нулевые элементы матрицы *C* соответствуют синему (в палитре `jet`) цвету узлов сетки с координатами (−1, −1), (−1, 1), (1, −1), а элемент матрицы *C*, равный единице, отвечает красному цвету в вершине (1, 1). Внутри ячейки цвет выбирается при помощи билинейной интерполяции между четырьмя вершинами. Если элементы матрицы *C* не принадлежат [0, 1], то происходит их линейное преобразование к отрезку [0, 1]. Функция `caxis` позволяет устанавливать соответствие между цветом и значением функции для данных осей, а `colormap` — выбирать произвольные

цветовые палитры. Обращение к `surf` без четвертого аргумента, являющегося матрицей с цветами узлов, приводит к выбору в качестве *C* матрицы *Z*, т. е. `surf(X,Y,Z)` эквивалентно `surf(X,Y,Z,Z)`.

- **`surface`** — низкоуровневая функция для создания поверхностей.

  `surface('PropName','PropValue','PropName','PropValue',...)` — общий вид вызова `surface`, позволяющий построить поверхность с заданными свойствами. Ниже перечислены наиболее часто используемые свойства и их возможные значения с необходимыми комментариями. Следует иметь в виду, что поверхность, отображаемая функцией `surface`, выводится на текущие оси графического окна. Если текущих осей (или окна) нет, то создаются оси, содержащие график поверхности. Однако по умолчанию оси являются двумерными, т. е. наблюдатель смотрит на них с точки, имеющей азимут 90° и угол склонения 0°. Высокоуровненевые графические функции, напротив, выводят график, видимый с точки обзора, азимут которой равен −37.5°, а угол склонения 30°. Для получения привычного вида графика поверхности при помощи `surface`, следовательно, требуется установить точку обзора, используя функцию `view`: `view(-37.5,30)`.

  См. разд. "Поворот графика, изменение точки обзора" главы 3.

  Задание визуализируемых данных производится установкой свойств `XData`, `YData` и `ZData`. Значениями данных свойств являются матрицы *X*, *Y* и *Z* с координатами узлов каркасной сетки и значениями высоты в соответствующей точке поверхности (см. `mesh`). Цвет каждого узла сетки задается матрицей в свойстве `CData`. (см. функцию `surf` выше). Последовательность команд, приведенная ниже, отображает поверхность в одном графическом окне при помощи `surf`, а в другом — с использованием эквивалентного (за исключением вывода линий сетки) обращения к `surface`.

```
>> [X,Y]=meshgrid(-3:0.5:3);
>> Z=sin(X).*cos(Y).*exp(abs(X.*Y/10));
>> surf(X,Y,Z)
>> figure
>> surface('XData',X,'YData',Y,'ZData',Z,'CData',Z);
>> view(-37.5,30)
```

  Значение свойства `CData` может быть матрицей *C*, такого же размера, как *X*, *Y* и *Z*. В данном примере *C*=*Z*. Линейное преобразование значений матрицы к отрезку [0, 1] устанавливает соответствие между цветами текущей палитры и значениями матрицы C (индексированный цвет). Способ определения соответствия элементов *C* цвету зависит от значения свойства поверхности `CDataMapping`. По умолчанию значение равно

'scaled', что как раз и обеспечивает линейное преобразование. Установка свойству CDataMapping значения 'direct' приводит к интерпретации значений *C* как номеров цветов текущей палитры, поэтому элементы C(i,j) должны принадлежать отрезку [1,length(colormap)].

Цвет каждого узла каркасной сетки поверхности можно задавать напрямую, без относительно цветовой палитры. Значением свойства CData должна быть не матрица, а трехмерный массив размера size(C)=[m n 3], где [m n]=size(X), а C(:,:,1), C(:,:,2), C(:,:,3) являются числами от нуля до единицы, определяющими пропорции красного, зеленого и синего цвета для каждого узла каркасной сетки.

Способ изменения цвета линий каркасной поверхности определяется значением свойства EdgeColor. Значение 'flat' приводит к постоянному цвету границы в пределах каждой ячейки, 'interp' — обеспечивает плавное изменение от вершины к вершине (используется линейная интерполяция). Фиксированный цвет задается одним из сокращений: 'c', 'm', 'y', 'k' (по умолчанию), 'r', 'g', 'b', 'w', или вектором из трех элементов в RGB. Значение 'none' приводит к тому, что линии каркасной поверхности не отображаются.

Способ заливки ячеек устанавливается при помощи свойства FaceColor. Постоянное значение FaceColor, равное одному из сокращений для цвета или вектору из трех элементов, приводит к заливке всех ячеек одним цветом. Заливка ячейки не производится, если свойство FaceColor имеет значение 'none'. Изменение цвета в пределах ячейки зависит от 'flat' или 'interp' (см. функцию surf). Значение texturemap позволяет использовать матрицу *C* размера, отличного от *X*, *Y* и *Z*. Следующий пример демонстрирует отображение текстурированной поверхности, сама текстура содержится в графическом файле texture.bmp, размера 300 на 300 пикселов, цвет каждого пиксела представляется 24 битами.

```
>> C=imread('texture.bmp');
>> whos C
 Name   Size     Bytes Class
 C    300x300x3   270000 uint8 array
Grand total is 270000 elements using 270000 bytes
```

Массив C является трехмерным, третье измерение соответствует цвету в RGB. Перед использованием массива цвета в качестве текстуры, его следует преобразовать к типу double и масштабировать элементы:

```
>> C=double(C);
>> C(:,:,1)=C(:,:,1)/255;
>> C(:,:,2)=C(:,:,2)/255;
>> C(:,:,3)=C(:,:,3)/255;
```

Теперь трехмерный массив вещественных чисел можно использовать для задания цвета поверхности.

```
>> surface('XData',X,'YData',Y,'ZData',Z,'CData',C,'FaceColor',...
     'texturemap','CDataMapping','direct');
>> view(-37.5,30)
```

Исходная текстура и текстурированная поверхность приведены на рис. П2.

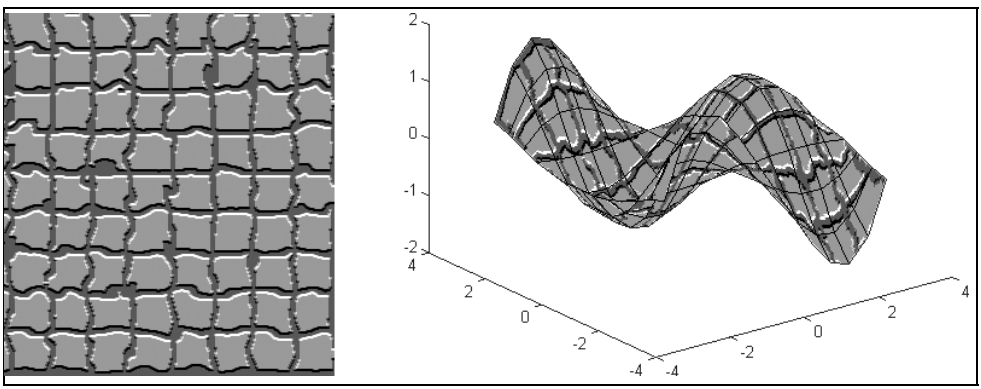


Рис. П2. Текстура и текстурированная поверхность

Стиль и толщина линий каркасной поверхности устанавливаются при помощи свойств `LineStyle` и `LineWidth`. Значениями `LineStyle` могут быть: `'-'`, `':'`, `'--'` или `'-'`, а `LineWidth` — вещественное число, равное толщине линий в пунктах (1 пункт = 1/72 дюйма).

Тип, размер и способ закраски границ и внутренности маркеров, помещаемых в узлах каркасной сетки, определяют свойства `Marker`, `MarkerSize`, `MarkerEdgeColor`, `MarkerFaceColor`.

Можно также освещать поверхность и управлять свойствами, определяющими взаимодействие поверхности со светом. Ниже перечислены свойства поверхности, отвечающие за освещение.

- `AmbientStrength` — интенсивность ненаправленного окружающего света, который освещает всю поверхность. Значением может быть вещественное число от нуля до единицы, по умолчанию используется 0.3. Цвет определяется значением свойства `AmbientLigthColor` осей (объекта `axes`).
- `BackFaceLighting` — определение способа частей поверхности в зависимости от направления нормали к поверхности и расположения наблюдателя. Значение `reverselit` (установленное по умолчанию) соот-

ветствует освещению внутренних поверхностей, нормаль к которым направлена от наблюдателя, так же как и внешних. Внутренние поверхности не освещаются, если свойство `BackFaceLighting` имеет значение `unlit`. Освещение границы замкнутых объектов убирается при помощи `lit`.

- `DiffuseStrength` — интенсивность рассеиваемого поверхностью света, излучаемого источником. Значение свойства `DiffuseStrength` может принимать вещественные значения от нуля до единицы, по умолчанию используется 0.6.
- `EdgeLighting` и `FaceLighting` — способ освещения границ и ячеек каркасной поверхности светом, идущим от источника. Свет не оказывает влияния на границы или ячейки каркасной поверхности, если соответствующее свойство установлено в `'none'`. Самым простым способом является равномерное освещение границ ячеек и самих ячеек. Равномерное освещение задается значением `'flat'`. Более сложным, но дающим лучший эффект, является способ Гуро, который используется при выборе `'gouraud'`. Интенсивность света вычисляется в узлах каркасной сетки, затем интерполируется вдоль границ каждой ячейки. Интенсивность света в точках ячейки определяется при помощи интерполяции вдоль отрезка прямой, соединяющего рёбра. Самое естественное освещение поверхности обеспечивается выбором значения `'phong'`, соответствующего способу Фонга. Способ Фонга состоит в интерполяции нормали сначала вдоль границ ячейки, а затем внутри ячейки. Зная нормаль в каждой точке поверхности (в каждом пикселе), можно определить, как она освещена внешним источником света. Метод Фонга требует достаточно большого объема вычислений, по сравнению с другими методами.

Листинг П1 содержит пример использования функции `surface` для построения параметрически заданной поверхности, освещенной одним источником света, кроме ненаправленного света.

Листинг П1. Пример использования функции `surface`

```
u = [-2*pi:0.02*pi:2*pi]';
v = [-2*pi:0.02*pi:2*pi];
X = 0.3*u*cos(v);
Y = 0.3*u*sin(v);
Z = 0.6*u.^2*ones(size(v));
figure;
axes;
view(-37.5,30)
```

```
surface('XData',X,'YData',Y,'ZData',Z,'CData',Z,...
  'BackFaceLighting','reverselit','LineStyle','none',...
  'FaceLighting', 'phong')
camlight(30,-50)
view(-37.5,30)
colormap(copper)
```

- **surfc** — построение залитой цветом каркасной поверхности и линий уровня на плоскости xy. Использование surfc аналогично surf и meshc.
- **surfl** — построение освещенной поверхности.
  - surfl(Z), surfl(X,Y,Z) — отображение равномерно освещенной поверхности окружающим светом. Входные аргументы имеют тот же смысл, что и в mesh или surf surfl(Z,s), surfl(X,Y,Z,s) — дополнительный аргумент s указывает на направление источника света. Допускается указание либо координат в векторе из трех элементов: s=[sx,sy,sz], либо азимута и угла склонения: s=[az,el]. Источник света (по умолчанию) расположен под углом 45° в направлении против часовой стрелки от текущей точки обзора.
  - surfl(...,'light') — помещает источник света (объект light).
  - h=surfl(...) — возвращает вектор указателей на поверхность и источник света.

Визуализация функции на непрямоугольной области

- **delaunay** — триангуляция Делане.
  - TRI=delaunay(x,y) — построение треугольников, координаты вершин которых задаются в векторах *x* и *y*. Выходным аргументом является матрица *TRI* с тремя столбцами, каждая строка матрицы соответствует некоторому треугольнику. Строка содержит индексы i, j и k такие, что вершинами треугольника являются точки с координатами (x(i),y(i)), (x(j),y(j)), (x(k),y(k)). Множество треугольников выбирается таким образом, что внутри окружности, описанной вокруг любого треугольника, не лежит ни одной из заданных точек, кроме вершин. Пример:

```
>> x=[0 1 0 -1 0];
>> y=[-1 0 1 0 0];
>> TRI=delaunay(x,y)
TRI =
   4   5   1
```

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

Входные данные могут соответствовать вырожденному случаю, т. е. точкам, лежащим на одной прямой. Функция `delaynay` предварительно добавляет к координатам точек малые случайные числа со среднеквадратическим отклонением, равным `4*sqrt(eps)`.

- `TRI=delaunay(x,y,dev)` — используется указанное значение `dev` для среднеквадратического отклонения добавляемого случайного возмущения координат исходных точек.
- `TRI=delaunay(x,y,'sorted')` — данное обращение предполагает, что координаты точек предварительно упорядочены по *y*, а затем по *x*, отсутствуют совпадающие точки, и нет вырожденности.

Функция `delaunay` используется, например, для задания треугольной сетки при визуализации функций, область определения которых не прямоугольная (см. `trimesh` и `trisurf` ниже).

❒ **`trimesh`** — построение каркасной поверхности функции на произвольной (необязательно прямоугольной) области. Область задается сеткой из треугольников. Сетка генерируется при помощи функции `delaunay`.

Листинг П2 содержит пример построения функции на шестиугольной области. Векторы *x* и *y* с координатами узлов сетки генерируются при помощи параметрически заданных окружностей с достаточно большим шагом изменения параметра. Результат приведен на рис. П3.

Листинг П2. Построение функции на непрямоугольной области

```
% Задание вектора со значениями параметра
t=[0:pi/3:2*pi];
% Генерация внешних узлов области (вершин шестиугольника)
x = sin(t);
y = cos(t);
% Генерация двух слоев внутренних узлов
x=[x 2/3*sin(t)];
y=[y 2/3*cos(t)];
x=[x 1/3*sin(t)];
y=[y 1/3*cos(t)];
% Задание дополнительного узла в центре начала координат
x(length(x)+1)=0;
y(length(y)+1)=0;
```

```
% Построение сетки
tri = delaunay(x,y);
% Вычисление значений функции в узлах сетки и запись значений в вектор z
z = -x.^2-y.^2;
subplot(1,2,1)
% Отображение графика функции на треугольной сетке и подпись осей
Hsurf=trimesh(tri,x,y,z)
set(Hsurf,'EdgeColor','k','LineWidth',2)
xlabel('x');
ylabel('y');
% Вывод сетки в то же графическое окно на другие оси и подпись осей
subplot(1,2,2)
% Указание в качестве четвертого аргумента нулевых значений приводит к
% построению сетки
Hmesh=trimesh(tri,x,y,zeros(size(x)));
set(Hmesh,'EdgeColor','k','LineWidth',2)
xlabel('x');
ylabel('y');
% Определение двумерных осей
view(2)
% Установка толщины и цвета линий сетки
set(h,'EdgeColor','k','LineWidth',2)
```

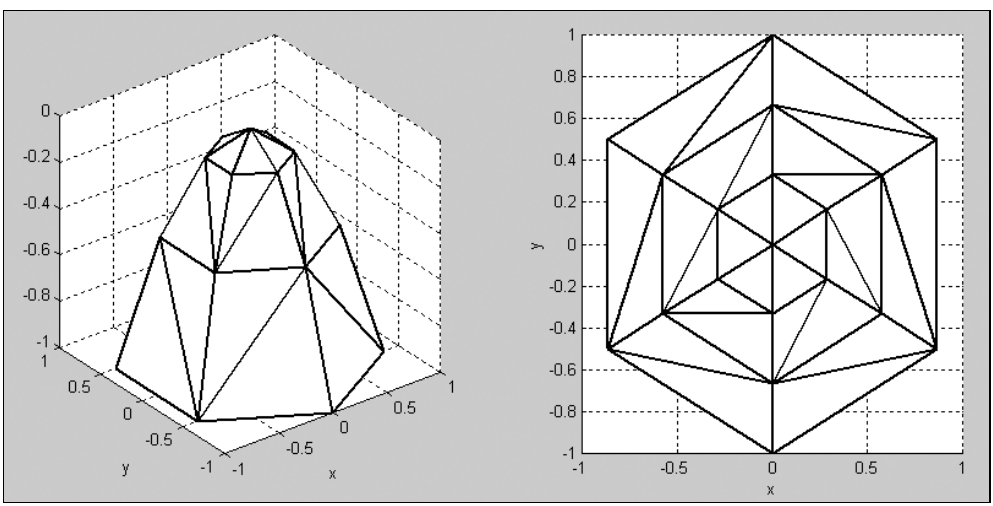


Рис. П3. График функции на треугольной сетке и сетка

Поверхность графика функции является многоугольным объектом (`patch`). Задание свойств поверхности производится при помощи функции `set` или непосредственно во входных аргументах:

`trimesh(...,'PropName','PropValue','PropName','PropValue',...)`.

- **`trisurf`** — построение каркасной закрашенной поверхности функции на произвольной (необязательно прямоугольной) области. Использование аналогично `trimesh` (см. выше).

Оформление графиков

- **`clabel`** — помещение подписей к линиям уровня на контурных графиках.
  - `clabel(C,h)` — входными аргументами являются: матрица *C* с информацией о линиях уровня и вектор указателей *h* на сами линии, являющиеся многоугольниками (графическими объектами типа `patch`). Данные аргументы инициализируются при соответствующем обращении к `contour`, `contourf` или `contour3`.

    Пример см. в разд. "Контурные графики" главы 3.
  - `clabel(C,h,v)` — маркируются только линии уровня, указанные в векторе *v*.
  - `clabel(C,h,'manual')` — переход в режим ручной разметки линий уровня. Щелчок мышью по линии уровня приводит к появлению на ней подписи со значением функции. Нажатие на клавишу <Enter> останавливает режим разметки.
  - `clabel(C)`, `clabel(C,v)` или `clabel(C,'manual')` — практически тот же результат, что и при указании *h*, но линия уровня отмечается маркером (знаком плюс), радом с которым помещается значение функции.
  - `hmark=clabel(...)` — выходной аргумент является вектором с указателями на созданные объекты типа `text` (и `lines`, если в качестве входного аргумента не задавался указатель `h` на линии уровня). Свойство `UserData` каждого текстового объекта содержит значение функции на линии уровня.
  - `clabel(...,'PropName','PropValue',...)` — пары входных аргументов позволяют задать любое свойство текстовых объектов, т. е. подписей к линиям уровня. Дополнительное свойство `'LabelSpacing'` предназначено для определения расстояния в пунктах между подписями, по умолчанию используется 144 (1 пункт=1/72 дюйма).
- **`datetick`** — разметка оси, по которой откладывается время.
  - `datetick(tickaxis,dateform)` — первый входной аргумент `tickaxis` предназначен для указания оси и может принимать значения `'x'`, `'y'`

или 'z'. Второй аргумент определяет формат разметки, например: 'dd-mmm-yyyy HH:MM:SS', 'HH:MM:SS' (все возможные значения dateform приведены в справочной системе MatLab). Для корректной разметки оси времени данные, откладываемые по ней, должны соответствовать серийному времени (см. функции datenum и now).

- **grid** — отображение или скрытие линий координатной сетки.
  - grid on — отображение сетки на текущих осях.
  - grid off — скрытие сетки на текущих осях.
  - grid — выводит текущее состояние сетки.

  Полное управление сеткой осуществляется при помощи свойств XGrid, YGrid и ZGrid координатных осей.

  См. разд. "Свойства осей" главы 9.

- **gtext** — режим интерактивного размещения текстовых подписей в пределах графического окна.
  - gtext('string') — после выполнения данной команды щелчок мыши по области текущего графического окна приводит к помещению текста в выбранную позицию. В случае отсутствия графических окон создается новое окно.
  - gtext(C) — указание в качестве входного аргумента массива ячеек из строк позволяет вводить многострочный текст.
  - gtext(...,'PropName',PropValue,...) — пары входных аргументов определяют свойства добавляемого текста (как объекта text).
- **hold** — управление выводом нескольких графиков в одно окно.
  - hold — возвращает текущее состояние.
  - hold on — каждый новый график добавляется в текущее графическое окно.
  - hold off — каждый новый график отображается в текущем графическом окне, переписывая содержимое окна.
- **legend** — помещение легенды на график.
  - legend(string1,string2,string3, ...) — входные аргументы являются строками или строковыми переменными, содержащими пояснения для линий (объект line), закрашенных многоугольников (объект patch) или поверхностей (объект surface).
  - legend(h,string1,string2,string3, ...) — первый входной аргумент является вектором указателей на графические объекты, информация о которых должна содержаться в легенде. Объекту с указателем h(1) соответствует string1, h(2) — string2 и т. д.

Текст легенды может задаваться не только в строках или строковых переменных, но и в символьном массиве или массиве ячеек из строк: `legend(M)` или `legend(h,M)`.

- `legend(Hax,...)` — размещение легенды на оси с указателем `Hax`.
- `legend off` — удаление легенды с текущих осей.
- `legend(Hax,'off')` — удаление легенды с осей, указатель на которые `Hax`.
- `Hleg=legend` — возвращает указатель на легенду текущих осей, если легенды нет, то `Hleg=[]`.
- `legend(...,pos)` — последний входной аргумент определяет положение легенды. Возможны значения:
  - ◊ `pos=0` — автоматическое определение наилучшего расположения легенды в пределах осей (обеспечивает наименьшее перекрытие графиков);
  - ◊ `pos=1` — верхний правый угол осей (используется по умолчанию);
  - ◊ `pos=2` — верхний левый угол осей;
  - ◊ `pos=3` — нижний левый угол осей;
  - ◊ `pos=4` — нижний правый угол осей;
  - ◊ `pos=-1` — в графическом окне справа от осей.

Произвольное положение легенды определяется в режиме редактирования графиков.

См. главу 4.

Программное изменение положения и свойств объектов легенды производится при помощи обращения `[Hleg,Hobj]=legend(...)`. Выходные аргументы содержат указатели на оси легенды и составляющие ее объекты: текстовые, линии и многоугольники.

❒ **`subplot`** — разбиение графического окна на несколько подграфиков и определение текущего подграфика.

См. разд. "Несколько графиков в одном графическом окне" главы 3.

❒ **`title`** — добавление заголовка на график.

- `title(str)` — текст, содержащийся во входном аргументе (строке или строковой переменной), помещается в графическом окне вверху осей. Текст может быть представлен в формате TeX.

  См. разд. "Оформление графика" главы 3.

  Список всех символов, которые могут быть заданы при помощи команд TeX, содержится в справочной системе MatLab в разделе со свойствами объекта типа `text` (см. свойство `String`).

- `title(str,PropName,PropValue,...)` — помещаемый заголовок (объект `text`) имеет свойства, определяемые парами входных аргументов.
- `h=title(...)` — возвращает указатель на создаваемый заголовок.

❒ **`xlabel`**, **`ylabel`** и **`zlabel`** — подписи к осям.

- `xlabel(str)` — текст, содержащийся во входном аргументе (строке или строковой переменной), используется в качестве подписи к оси `x` (для остальных осей аналогично).
- `xlabel(str,PropName,PropValue,...)` — помещаемая подпись к оси (объект `text`) имеет свойства, определяемые парами входных аргументов.
- `h=xlabel(...)` — возвращает указатель на создаваемую подпись к оси.

Управление видом графика

Несколько разделов книги посвящены описанию способов, предлагаемых MatLab для изменения вида графиков.

См. разд. "Поворот графика, изменение точки обзора" главы 3, "Управление камерой" и "Изменение точки обзора" главы 4

Ниже приведены все основные функции MatLab, предназначенные для установки требуемого вида осей графического окна. Объектом в данном разделе будет называться содержимое осей (то, на что направлена камера), которое может состоять из нескольких графических объектов MatLab, к примеру, поверхности (`surface`) и многоугольника (`patch`).

❒ **`camdolly`** — изменение положения камеры и объекта.

- `camdolly(dx,dy,dz)` — перемещение камеры и объекта на `dx`, `dy` и `dz` в системе координат камеры. Перемещение вправо или влево определяется значением `dx`, вверх или вниз — `dy`, вдоль оси камеры — `dz`. Единицы измерения должны соответствовать видимой области, например, `camdolly(1,-1,0)` приводит к перемещению объекта в левый верхний угол.
- `camdolly(dx,dy,dz,targetmode)` — дополнительный четвертый входной аргумент `targetmode` позволяет задать раздельное перемещение камеры и объекта. Значение `'movetarget'` (используемое по умолчанию) соответствует перемещению и камеры, и объекта, а `'fixtarget'` — только камеры.
- `camdolly(dx,dy,dz,targetmode,coordsys)` — пятый дополнительный аргумент `coordsys` предназначен для указания системы координат и единиц измерения перемещений, задаваемых `dx`, `dy` и `dz`. По умолчанию используется значение `'camera'`, которое обеспечивает передвижение в системе координат камеры (см. `camdolly(dx,dy,dz)` выше).

Для изменения положения камеры на плоскости экрана следует использовать `'pixels'`, причем первые два входных аргумента `dx` и `dy` задают смещение в пикселах, а `dz` игнорируется. Часто удобно определять величины перемещений в системе координат осей, для чего следует в качестве пятого входного аргумента указать `'data'`.

- `camdolly(Hax,...)` — перемещение камеры и объекта осуществляется на осях с указателем `Hax`.

❐ **`camlookat`** — направление камеры на нужный графический объект или объекты. Применяется в том случае, когда на осях расположено несколько графических объектов и требуется укрупнить вид одного или нескольких из них.

`camlookat(h)` — направление камеры на графический объект, указателем на который является *h* (в случае нескольких объектов используется вектор указателей). Последовательность команд, приведенная ниже, обеспечивает построение двух каркасных сферических поверхностей на одних осях и последовательное направление камеры сначала на первую сферу, а затем на вторую:

```
>> [X,Y,Z]=sphere;
>> X1=X-1;
>> Y1=Y-1;
>> Z1=Z-1;
>> X2=X+1;
>> Y2=Y+1;
>> Z2=Z+1;
>> H1=mesh(X1,Y1,Z1);
>> hold on
>> H2=mesh(X2,Y2,Z2);
>> camlookat(H1)
>> camlookat(H2)
```

❐ **`camorbit`** — поворот камеры вокруг объекта.

- `camorbit(dtheta,dphi)` — поворот камеры вокруг объекта текущих осей на угол `dtheta` по горизонтали и `dphi` по вертикали (значения указываются в градусах). Листинг П3 содержит пример использования `camorbit` во вложенных циклах с целью осмотра поверхности со всех сторон.

Листинг П3. Вращение камеры вокруг объекта

```
surf(peaks(40));
for i=1:4
```

```
  pause(1)
  for j=1:36
   camorbit(10,0);
   pause(0.01)
  end
  camorbit(0,90);
end
```

Возможно указание различных способов поворота камеры вокруг объекта.

- `camorbit(dtheta,dphi,coordsys,direction)` — дополнительные входные аргументы `coordsys` и `direction` предназначены для указания системы координат и направления, вокруг которого происходит поворот. Возможны два значения для `coordsys`: `'data'` (используется по умолчанию) и `'camera'`. Если указано `'data'`, то поворот камеры происходит вокруг линии, идущей вдоль выбранного направления от точки, на которую нацелена камера. Направление задается во входном аргументе `direction` и может быть вектором из трех координат `[x y z]` или символами `'x'`, `'y'` или `'z'` для указания поворота вокруг определенной координатной оси. Выбор в качестве `coordsys` значения `'camera'` приводит к повороту на угол `dtheta` по горизонтали и `dphi` по вертикали относительно точки объекта, на которую нацелена камера.
- `camorbit(Hax,...)` — поворот камеры применяется к объектам, расположенным в пределах осей с указателем `Hax`.

❐ **`campan`** — поворот объекта вокруг камеры. Использование аналогично `camorbit`.

❐ **`campos`** — установка или определение положения камеры. Положение камеры определяется вектором из трех элементов в декартовой системе координат.

- `c=campos` — выходной аргумент вектор *c* содержит координаты камеры в декартовой системе координат текущих осей.
- `campos([x y z])` — задание положения камеры в декартовой системе координат осей.
- `cpmode=campos('mode')` — выходной аргумент является строковой переменной и может быть `'auto'` или `'manual'` в зависимости от значения свойства осей `CameraPositionMode`.

См. разд. "Управление камерой" главы 4.

- `campos(mode)` — установка свойства осей `CameraPositionMode`. Входной аргумент может принимать значения `'auto'` или `'manual'`.

- `campos(Hax,...)` — управление положением камеры на осях с указателем `Hax`.

❒ **`camproj`** — установка или определение типа проекции осей трехмерных графиков на экран.

- `proj=camproj` — выходной аргумент `proj` содержит тип проекции трехмерных осей на экран. Возможны два варианта: `'orthographic'` или `'perspective'`, в зависимости от значения свойства `Projection` осей координат.
- `camproj(projection)` — задание типа проекции трехмерных осей на экран (`'orthographic'` или `'perspective'`).
- `camproj(Hax,...)` — установка или определение типа проекции осей с указателем `Hax` на экран.

❒ **`camroll`** — поворот камеры вокруг ее оси.

Взаимное расположение камеры и объекта описано в разд. "Управление камерой" главы 4.

`camroll(dtheta)` — поворот камеры на угол `dtheta` (в градусах) по часовой стрелке. Пример содержится в листинге П4.

Листинг П4. Поворот камеры вокруг ее оси

```
sphere
hold on
cylinder
for i=1:60
  pause(0.01)
  camroll(6)
end
```

Применение поворота камеры к осям с указателем `Hax` производится при помощи обращения `camroll(Hax,dtheta)`.

❒ **`camtarget`** — позиционирование камеры.

- `c=camtarget` — возвращает вектор *c* координат точки, на которую направлена камера, в декартовой системе координат текущих осей.
- `camtarget([x y z])` — устанавливает положение точки, на которую направлена камера, в декартовой системе координат текущих осей, т. е. свойство `CameraTarget` принимает значение `[x y z]`.

См. разд. "Управление камерой" главы 4.

- `ctmode=camtarget('mode')` — выходной аргумент является строковой переменной и содержит значение (`'auto'` или `'manual'`) свойства `CameraTargetMode`.
- `camtarget(mode)` — устанавливает режим позиционирования камеры, т. е. свойство `CameraTargetMode` принимает значение `mode` (`'auto'` или `'manual'`).
- `camtarget(Hax,...)` — позиционирование камеры на осях с указателем `Hax`.

❐ **`camup`** — установка или получение направления вектора камеры.

- `v=camup` — выходной аргумент `v` является вектором камеры в декартовой системе координат текущих осей.

  См. разд. "Управление камерой" главы 4.

- `camup([x y z])` — входной аргумент задает вектор камеры в декартовой системе координат текущих осей. Длина вектора не имеет значения, важно определяемое им направление. Свойство `CameraUpVector` принимает значение `[x y z]`.
- `upmode=camup('mode')` — выходной аргумент является строковой переменной и содержит значение (`'auto'` или `'manual'`) свойства `CameraUpVectorMode`.
- `camup(mode)` — устанавливает режим выбора направления вектора камеры, т. е. свойство `CameraUpVectorMode` принимает значение `mode` (`'auto'` или `'manual'`).
- `camup(Hax,...)` — установка или получение направления вектора камеры осей с указателем `Hax`.

❐ **`camva`** — установка и получение угла обзора объекта камерой.

- `c=camva` — выходной аргумент `c` является углом обзора (в градусах) объекта камерой на текущих осях.

  См. разд. "Управление камерой" главы 4.

- `camva(a)` — входной аргумент `a` задает угол обзора (в градусах) объекта камерой на текущих осях. Свойство `CameraViewAngle` принимает значение `c`.
- `cvamode=camva('mode')` — выходной аргумент является строковой переменной и содержит значение (`'auto'` или `'manual'`) свойства `CameraViewAngleMode`.
- `camva(mode)` — устанавливает режим выбора угла обзора, т. е. свойство `CameraViewAngleMode` принимает значение `mode` (`'auto'` или `'manual'`).

- `camva(ax,...)` — установка и получение угла обзора объекта камерой на осях с указателем `Hax`.

❒ **`camzoom`** — изменение угла обзора объекта камерой.

`camzoom(p)` — значение входного аргумента `p`, большее единицы, приводит к увеличению угла обзора, если `p` меньше единицы, но больше нуля, то угол обзора уменьшается. Свойство `CameraViewAngleMode` принимает значение `'manual'`, а значение `CameraViewAngle` изменяется соответствующим образом. Пример применения `camzoom` приведен в листинге П5.

Листинг П5. Изменение угла обзора объекта

```
sphere
for i=10:-1:3
  pause(0.05)
  camzoom(1/10)
end
for i=3:10
  pause(0.05)
  camzoom(10)
end
```

Изменение угла обзора камерой объекта, расположенного на осях с указателем `Hax` производится при помощи обращения `camzoom(Hax,p)`.

❒ **`daspect`** — изменение или получение масштаба осей.

- `d=daspect` — возвращает вектор *d*, определяющий масштаб текущих осей.
- `daspect([x y z])` — установка соотношения масштабов текущих осей, важна пропорция элементов вектора, например: `daspect([1 2 1])` и `daspect([10 20 10])` приводят к одинаковым результатам. Свойство `DataAspectRatio` принимает значение `[x y z]`, а `DataAspectRatioMode` — `'manual'`. При отображении реальных геометрических объектов для сохранения соотношения геометрических размеров следует устанавливать `[1,1,1]`, например:

```
>> sphere
>> daspect([1 1 1])
```

- `darmode=daspect('mode')` — выходной аргумент является строковой переменной и содержит значение (`'auto'` или `'manual'`) свойства `DataAspectRatioMode`.
- `daspect(mode)` — устанавливает режим выбора масштаба осей, т. е. свойство `DataAspectRatioMode` принимает значение `mode` (`'auto'` или `'manual'`).
- `daspect(Hax,...)` — изменение или получение масштаба осей с указателем `Hax`.

❒ **`pbaspect`** — установка или определение соотношения длин осей.

- `v=pbaspect` — в вектор *v* записывается соотношение длин текущих осей.
- `pbaspect([x y z])` — установка соотношения длин текущих осей, важна пропорция элементов вектора, например: `pbaspect([1 1 1])` и `pbaspect([10 10 10])` приводят к одинаковым результатам. Свойство `PlotBoxAspectRatio` принимает значение `[x y z]`, а `PlotBoxAspectRatioMode` — `'manual'`.
- `pbarmode=pbaspect('mode')` — выходной аргумент `pbarmode` является строковой переменной и содержит значение (`'auto'` или `'manual'`) свойства `PlotBoxAspectRatioMode`.
- `pbaspect(mode)` — устанавливает режим выбора соотношения длин осей, т. е. свойство `PlotBoxAspectRatioMode` принимает значение `mode` (`'auto'` или `'manual'`).
- `pbaspect(ax,...)` — изменение или получение соотношения длин осей с указателем `Hax`.

❒ **`view`** — установка или определение точки обзора.

- `view(az,el)` или `view([az,el])` — задание положения точки обзора при помощи азимута и угла склонения, выраженных в градусах.

  См. разд. "Поворот графика, изменение точки обзора" главы 3.

- `view(2)` — задание двумерных осей с `AZ=0`, `EL=90` (наблюдатель смотрит на оси сверху, вдоль оси `z`).
- `view(3)` — изменение вида осей с азимутом и углом склонения, выбираемыми по умолчанию: `AZ=-37.5`, `EL=30`.
- `[az,el]=view` — получение текущих значений азимута и угла склонения.
- `view(T)` — установка точки обзора при помощи матрицы преобразования `T`, `size(T)=[4 4]` (см. `viewmtx`).
- `T=view` — получение текущей матрицы преобразования значений азимута и угла склонения.

❒ **viewmtx** — вычисление матрицы преобразования.

- `T=viewmtx(az,el)` — возвращает матрицу ортогонального проектирования для отображения трехмерных объектов на плоскости (экране монитора) в соответствии с точкой обзора, определяемой азимутом и углом склонения (см. `view`). Сама точка обзора на текущих осях не изменяется. Для получения матрицы проектирования, соответствующей текущему положению точки обзора, следует использовать обращение `T=view`.
- `T=viewmtx(az,el,phi)` — возвращает матрицу проектирования, обеспечивающую перспективное изображение. Третий входной аргумент `phi` определяет величину перспективы, значение `phi=0` соответствует ортогональной проекции.

Матрица `T` преобразует векторы длины четыре `[x y z 1]'` к векторам, первых два компонента которых, поделенные на четвертую, являются искомыми проекциями на плоскость экрана. Листинг П6 содержит пример изображения куба с различной перспективой.

Листинг П6. Изменение перспективы изображения

```
% Задание координат вершин куба
x = [0 1 1 0 0 0 1 1 0 0 1 1 1 1 0 0];
y = [0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1];
z = [0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0];
% Циклическое изменение значения перспективы
for phi=0:10:90
 % Получение матрицы проектирования
 T=viewmtx(-37.5,30,phi);
 % Нахождение проекций
 v=T*[x; y; z; ones(size(x))];
 x1=v(1,:)./v(4,:);
 y1=v(2,:)./v(4,:);
 % Вывод результата на двумерные оси
 plot(x1,y1)
 pause(1)
end
```

❒ **xlim**, **ylim**, **zlim** — установка или определение пределов осей координат (на примере `xlim`).

- `xl=xlim` — вектор из двух элементов `xl` содержит пределы оси `x`.

- `xlim([xmin xmax])` — входной аргумент задает пределы оси `x`, свойство `XLim` принимает значение `[xmin xmax]`.

  См. разд. "Свойства осей" главы 4.

- `xlmode=xlim('mode')` — выходной аргумент является строковой переменной и содержит значение (`'auto'` или `'manual'`) свойства `XLimMode`.
- `xlim(mode)` — устанавливает режим выбора пределов осей, т. е. свойство `XLimMode` принимает значение `mode` (`'auto'` или `'manual'`).
- `xlim(Hax,...)` — изменение или получение пределов осей с указателем `Hax`.

Приложение 2



Описание дискеты

К книге приложена дискета, которая содержит листинги программ, приведенных в тексте. Дискета организована следующим образом. В корневом каталоге находятся папки с номерами глав и файл readme.txt, в котором кратко описано содержание и работа с дискетой. Каждая папка содержит M-файлы, и файл readmeNN.txt (где NN — номер главы), в котором описано соответствие между именами M-файлов и номерами листингов программ.

Для использования прилагаемых M-файлов следует скопировать их в текущий каталог MatLab и запустить из командной строки или редактора M-файлов, или вызвать как файл-функцию (в зависимости от содержимого M-файла).

Установка текущего каталога и работа в редакторе M-файлов описана в главе 5.

Литература

1. Андриевский Б. Р. Избранные главы теории автоматического управления с примерами на языке MatLab. СПб.: Наука, 1999. — 467 с.
2. Гультяев А. Визуальное моделирование в среде MatLab. СПб.: Питер, 2000. — 430 с.
3. Дьяконов В. П. MatLab. СПб.: Питер, 2001. — 553 с.
4. Дьяконов В. П. MatLab 5.0/5.3. Система символьной математики. М.: Нолидж, 1999. — 633 с.
5. Лазарев Ю. Ф. MatLab 5.x. Киев: BHV: Ирина, 2000. — 383 с.
6. Мартынов Н. Н. MatLab 5.x. Вычисления, визуализация, программирование. М.: КУДИЦ-ОБРАЗ, 2000. — 332 с.
7. Мэтьюз Дж. Г. Численные методы. Использование MatLab: Пер. с англ. Под ред. Ю. В. Козаченко. 3-е изд. М.: Вильямс, 2001. — 713 с.
8. Потемкин В. Г. Система MatLab: Справ. пособие. М.: ДИАЛОГ-МИФИ, 1997. — 350 с.
9. Потемкин В. Г. MatLab 5 для студентов: Новая редакция. 2-е изд., испр. и доп. М.: ДИАЛОГ-МИФИ, 1999. — 447 с.
10. Потемкин В. Г. Система инженерных и научных расчетов MatLab 5.x: В 2-х т. М.: ДИАЛОГ-МИФИ, 1999.
11. Kwon Y. W. The Finite Element Method using MATLAB. — Boca Raton a. o. CRC Press, 1997. — 519 p.