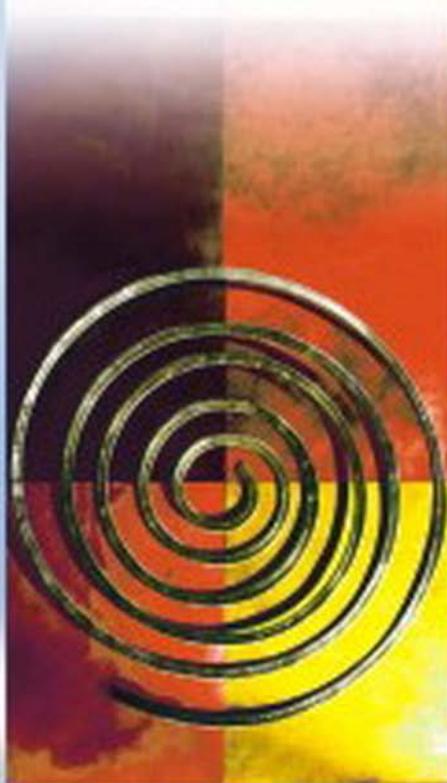


Игорь Ануфриев
Александр Смирнов
Елена Смирнова



MATLAB 7



- Работа с массивами, графика
- Решение классических вычислительных задач
- Программирование
- Решение специальных задач
- Интеграция с MS Office

Наиболее
полное
руководство

+CD



В ПОДЛИННИКЕ®

**Игорь Ануфриев
Александр Смирнов
Елена Смирнова**

MATLAB 7

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.2
А73

Ануфриев И. Е., Смирнов А. Б., Смирнова Е. Н.
А73 MATLAB 7. — СПб.: БХВ-Петербург, 2005. — 1104 с.: ил.
 ISBN 5-94157-494-0

Книга посвящена применению пакета MATLAB и его расширений (Toolbox) для решения различных математических, экономических задач, задач математической физики, обработки данных и ряда других. Подробно рассмотрена работа с массивами, описаны возможности высокуюровневой и низкоуровневой графики. Значительный объем материала отведен вычислительным задачам: решению уравнений, систем линейных и нелинейных уравнений, интегрированию, аппроксимации функций, решению систем обыкновенных дифференциальных уравнений и уравнений в частных производных, задачам оптимизации и работе с разреженными матрицами. Изложены основы программирования на встроенным языке и принципы эффективного написания приложений в MATLAB, вопросы интеграции с MS Word и MS Excel.

Описаны возможности расширений Toolbox. Приведено множество примеров и заданий для самостоятельной работы. Для удобства читателей тексты программ собраны на прилагаемом компакт-диске.

Для научных работников, преподавателей, инженеров и студентов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Алексей Семенов</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталия Першакова</i>
Дизайн обложки	<i>Игоря Цырульникова</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.03.05.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 89,01.

Тираж 5000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 55.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитов
в ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29.

Оглавление

Введение	1
B1. О назначении и возможностях пакета MATLAB и его расширений	1
B2. О содержании книги	10
ЧАСТЬ I. Основы работы в MATLAB	19
Глава 1. Простейшие вычисления	21
Рабочая среда MATLAB	21
Арифметические вычисления	24
Простейшие вычисления	24
Форматы вывода результата вычислений	25
Использование элементарных функций	28
Встроенные элементарные функции	33
Тригонометрические, гиперболические и обратные к ним функции	33
Экспоненциальная функция, логарифмы, степенные функции	34
Функции для работы с комплексными числами	34
Округление и остаток от деления	34
Использование переменных	35
Сохранение и восстановление рабочей среды	38
Просмотр и удаление переменных, выбор имен переменных	42
Эффективная работа из командной строки (<i>Command History</i>)	44
Задания для самостоятельной работы	48
Глава 2. Работа с массивами	50
Ввод, сложение и вычитание векторов	51
Обращение к элементам вектора	57
Применение функций обработки данных к векторам	59
Поэлементные операции с векторами	62
Построение таблицы значений функции	65
Построение графиков функций одной переменной	70
Умножение векторов	75
Скалярное произведение	75
Векторное произведение	76
Внешнее произведение	77
Ввод матриц, простейшие операции	78
Различные способы ввода	78
Обращение к элементам матриц	79
Логическое индексирование	81

Сложение, вычитание, умножение, транспонирование и возвведение в степень	84
Перемножение матрицы и вектора	86
Решение систем линейных уравнений.....	86
Считывание и запись данных	87
Блочные матрицы	89
Конструирование блочных матриц.....	89
Выделение блоков	91
Удаление строк и столбцов	91
Заполнение матриц при помощи индексации	92
Создание матриц специального вида	93
Визуализация матриц.	97
Поэлементные операции и встроенные функции	99
Поэлементные операции с матрицами.....	99
Вычисление математических функций от элементов матриц.....	101
Применение функций обработки данных к матрицам.....	102
Графики функций двух переменных	105
Задания для самостоятельной работы	109
Задания на векторы.....	109
Задания на матрицы	109
Глава 3. Высокоуровневая графика	112
Построение графиков из окна <i>Workspace</i>	112
Диаграммы и гистограммы.....	116
Представление векторных данных	116
Диаграммы векторных данных	116
Гистограммы векторных данных	122
Представление матричных данных	126
Графики функций.....	129
Графики функций одной переменной	129
Графики в линейном масштабе	129
Графики в логарифмических масштабах	132
Изменение свойств линий	133
Оформление графиков.....	135
Графики параметрических и кусочно-заданных функций.....	137
Графики функций двух переменных	140
Трехмерные графики функций	140
Контурные графики	145
Оформление графика	148
Поворот графика, изменение точки обзора	153
Построение параметрически заданных поверхностей и линий	156
Построение освещенной поверхности	159
Анимированные графики.....	161
Работа с несколькими графиками	162
Вывод графиков в отдельные окна	163
Вывод нескольких графиков на одни оси.....	165
Несколько графиков в одном графическом окне	166
Визуализация векторных полей.....	169
Задания для самостоятельной работы	174

Глава 4. Интерактивная среда для построения графиков.....	176
Графические объекты	177
Редактор графиков.....	178
Свойства осей, подписи, заголовок.....	183
Цветовое оформление, разметка и сетка	184
Подписи и заголовок	186
Свойства линий и поверхностей.....	187
Свойства линий	187
Свойства поверхностей	190
Дополнительные элементы оформления	191
Обзор графиков и поверхностей	196
Изменение масштаба, определение значений функции, поворот	196
Камера для обзора графического объекта	197
Панель инструментов камеры	199
Сохранение, экспорт и печать	201
Задания для самостоятельной работы	204
Глава 5. М-файлы	205
Работа в редакторе М-файлов	205
Настройки редактора М-файлов	209
Типы М-файлов	212
Файл-программы	212
Установка путей	215
Команды для установки путей	218
Файл-функции	219
Файл-функции с одним входным аргументом	220
Файл-функции с несколькими входными аргументами.....	225
Файл-функции с несколькими выходными аргументами	225
Разновидности функций	227
Подфункции.....	228
Вложенные функции	232
Приватные функции	234
Разбиение М-файла на ячейки	234
Диагностика М-файлов	237
Задания для самостоятельной работы	239
ЧАСТЬ II. ВЫЧИСЛЕНИЯ И ПРОГРАММИРОВАНИЕ	241
Глава 6. Методы вычислений в MATLAB	243
Исследование функций.....	243
Встраиваемые и анонимные функции.....	243
Решение уравнений	246
Решение произвольных уравнений.....	246
Вычисление всех корней полинома	252
Нахождение экстремумов функций.....	253
Минимизация функции одной переменной.....	253
Минимизация функции нескольких переменных.....	255
Управление ходом вычислений	258

Более подробно о <i>fplot</i>	262
Исследование функций, зависящих от параметров.....	264
Интегрирование функций	265
Вычисление определенных интегралов	265
Вычисление двойных интегралов.....	268
Вычисление некоторых интегралов	269
Интегралы, зависящие от параметра	269
Интегралы с переменным верхним пределом	271
Полиномы и интерполяция.....	271
Операции с полиномами	271
Умножение, деление, сложение и вычитание	271
Вычисление производных	273
Интерполирование и сглаживание.....	274
Приближение по методу наименьших квадратов.....	274
Интерполяция сплайнами	276
Интерполяция двумерных и многомерных данных.....	278
Задачи линейной алгебры	281
Системы уравнений, определители, обращение матриц.....	281
Системы с плохо обусловленными матрицами.....	283
Переопределенные и недоопределенные системы	285
Решение систем при помощи функции <i>linsolve</i>	287
Обращение матриц.....	290
Собственные числа и векторы матрицы, функции матриц	290
Решение дифференциальных уравнений.....	293
Решение задачи Коши.....	293
Решение уравнений Лотки—Вольтерры.....	298
Выбор солвера для решения задачи Коши	300
Управление процессом решения	301
Задание точности вычислений и шага интегрирования.....	303
Управление выводом результатов.....	307
Задание матрицы Якоби для повышения эффективности вычислений	312
Задачи с известными параметрами	314
Системы, не разрешенные относительно производной, дифференциально-алгебраические уравнения	316
Решение дифференциальных уравнений с запаздывающим аргументом	323
Решение граничных задач	330
Схема решения.....	330
Простой пример граничной задачи	333
Возможности солвера <i>bvp4c</i> , управление вычислениями	335
Граничные задачи с неизвестными параметрами.....	337
Решение задачи с особенностью на границе	342
Задания для самостоятельной работы	345
Глава 7. Управляющие конструкции языка программирования	347
Операторы цикла	347
Цикл <i>for</i>	347
Цикл <i>while</i> , суммирование рядов	357
Операторы ветвлений	361
Условный оператор <i>if</i>	361

Проверка входных аргументов.....	362
Организация ветвления	366
Оператор <i>switch</i>	371
Выход из файл-функции, оператор <i>return</i>	374
Прерывание и продолжение циклов	375
Обработка исключительных ситуаций	377
Логические выражения с массивами и числами	379
Операции отношения.....	379
Логические операции с числами и массивами	380
Приоритет логических и арифметических операций.....	383
Задания для самостоятельной работы	384
Глава 8. Обработка данных и приемы программирования в MATLAB	386
Работа со строками.....	386
Простейшие операции со строками	386
Ввод и сцепление строк	386
Сервисные функции для работы со строками.....	388
Массивы строк	390
Текстовые файлы	392
Открытие файла, считывание данных и закрытие файла	393
Запись в текстовый файл.....	396
Запись строк	397
Форматный вывод	399
Простые структуры.....	403
Массивы структур и массивы яческ	408
Массивы структур.....	408
Создание файл-функций для работы массивами структур	413
Запись данных массивов структур в текстовый файл	414
Считывание информации из текстового файла	416
Операции с массивами структур	420
Массивы яческ	421
Приложения с интерфейсом из командной строки	427
Простой пример, программа-калькулятор.....	428
Формирование и исполнение команд, функция <i>eval</i>	432
Организация вывода текстовых результатов.....	437
Файл-функции с переменным числом аргументов	437
Функции от функций	446
Перманентные переменные.....	450
Рекурсивные функции.....	455
Диалоговая отладка программ	467
Точки останова, пошаговое выполнение программы.....	467
Пример диалоговой отладки	471
Задания для самостоятельной работы	477
Глава 9. Дескрипторная графика	480
Основы дескрипторной графики	480
Свойства графических объектов	481
Функции <i>set</i> и <i>get</i> , текущие объекты	481
Свойства осей	482
Свойства линий и поверхностей.....	486

Указатели на объекты.....	489
Изменение свойств линий и осей.....	489
Добавление линий графиков.....	492
Удаление и очистка объектов	493
Влияние команд <i>hold</i> , <i>cla</i> , <i>clf</i> и <i>reset</i> на свойства окна и осей.....	494
Получение информации о свойствах графических объектов	495
Использование указателей, примеры.....	497
Задание свойств в аргументах графических функций.....	499
Размещение окон, осей и текста	500
Расположение графических окон и осей.....	500
Вывод текстовой информации.....	509
Графические объекты	522
Иерархия объектов	523
Объект Root.....	524
Объект Figure (графическое окно).....	525
Базовые объекты (Core Objects).....	527
Объекты Rectangle и Line, блок-схемы и диаграммы.....	527
Объект Patch, цветовое оформление объектов	531
Освещение объектов, объект Light (источник света).....	541
Управление объектами, копирование, поиск, скрытые указатели	545
Объекты-группы hggroup и hgtransform.....	553
Рисованные объекты (Plot Objects).....	557
ЧАСТЬ III. ПРИЛОЖЕНИЯ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ	561
Глава 10. Принципы создания приложений с GUI.....	563
Среда GUIDE	564
Программирование событий	568
Глава 11. Конструирование интерфейса	575
Управление свойствами объектов	575
Работа над приложением	577
Запуск приложения и его редактирование	577
Размеры объектов и их выравнивание.....	579
Всплывающие подсказки и пиктограммы	581
Программирование элементов интерфейса	582
Флаги, рамки	582
Переключатели	586
Списки	593
Полосы скроллинга	598
Область ввода текста	600
Свойства приложения.....	602
Изменение размеров приложения	602
Взаимодействие приложения со средой MATLAB.....	605
Способы программирования событий	606
Порядок обхода элементов управления клавишей <Tab>	607

Глава 12. Диалоговые окна и меню приложения	609
Виды диалоговых окон.....	609
Окно подтверждения	609
Окна открытия и сохранения файла	611
Окно с сообщением об ошибке.....	613
Меню графического окна.....	614
Редактор меню.....	614
Программирование пунктов меню	617
Оформление меню.....	618
Пункты меню с флагами состояния	619
Разделительные линии.....	620
Упорядочение меню	621
Контекстное меню объектов	622
Создание контекстного меню в редакторе.....	623
Связывание контекстного меню с объектом	624
Программирование контекстного меню	624
Глава 13. Программирование событий	627
События графических объектов	627
Приложение для получения ASCII-кода символа	628
Как вызываются подфункции обработки событий	630
Событие <i>ButtonDownFcn</i>	634
Событие <i>ButtonDownFcn</i> осей	634
Событие <i>ButtonDownFcn</i> линии	637
Создание приложений с GUI без среды GUIDE	638
Свойства объектов, полезные при программировании событий	641
Прерывание обработки событий	641
Изменение формы курсора	642
ЧАСТЬ IV. ИСПОЛЬЗОВАНИЕ TOOLBOX И РЕШЕНИЕ ПРИКЛАДНЫХ ЗАДАЧ	643
Глава 14. Решение задач математической физики	645
Простой пример	645
Постановка задачи.....	646
Среда <i>pdetool</i> , конструирование области	646
Определение уравнения и граничных условий	649
Решение и визуализация результата	652
Описание возможностей PDE Toolbox	654
Эллиптическое уравнение	655
Переменные коэффициенты и правая часть уравнения	656
Параболическое и гиперболическое уравнения	657
Пример нестационарной задачи.....	658
Задача на собственные значения	661
Системы дифференциальных уравнений	661
Параметры триангуляции и управление процессом решения	663
Конструирование геометрии области	665
Геометрические примитивы.....	665
Задание структуры области	666

Композитные материалы	668
Использование сетки	669
Использование функций PDE Toolbox	670
Задание геометрии области	670
Триангуляция	678
Границочные условия и коэффициенты уравнения	680
Сольверы	682
Визуализация результата	685
Решение модельной задачи	686
Функции PDE Toolbox	688
Создание геометрических примитивов	689
Геометрия области и триангуляция	689
Глава 15. Разреженные матрицы	692
Работа с разреженными матрицами	692
Схема хранения	692
Создание разреженных матриц	694
Операции с разреженными матрицами	699
Задачи линейной алгебры	703
Факторизация матриц	703
Профайлер	708
Решение систем уравнений и исследование спектра	716
Глава 16. Оптимизация.....	718
Optimization Toolbox.....	718
Линейное и нелинейное программирование	718
Линейное программирование	718
Квадратичное программирование	721
Нелинейное программирование	724
Нелинейные задачи	727
Задача о достижении границы	727
Минимизация функции с полубесконечными ограничениями	728
Минимаксная задача	731
Решение нелинейных уравнений	733
Метод наименьших квадратов	735
Подбор параметров	736
Параметры оптимизации	739
Примеры	741
Решение системы нелинейных уравнений	742
Пример приложения с GUI	746
Глава 17. Символьные вычисления	751
Символьные переменные и функции	751
Определение переменных и функций и работа с ними	751
Матрицы и векторы	754
Вычисления с символьными переменными	756
Графическое представление функций	758
Упрощение, преобразование и вычисление выражений	760

Решение задач.....	763
Задачи линейной алгебры	763
Суммирование и разложение в ряд	767
Пределы, дифференцирование и интегрирование.....	769
Решение уравнений и систем.....	775
Решение дифференциальных уравнений и систем	778
Глава 18. Работа со сплайнами в Spline Toolbox	784
Сплайны и формы их представления	784
Кусочно-полиномиальная форма (<i>pp</i> -форма).....	785
В-форма (разложение по базисным сплайнам).....	785
Интерполяционные сплайны	786
Построение кубического сплайна	786
Стандартные краевые условия.....	786
Операции над сплайнами	788
Построение сплайна для вектор-функции	790
Произвольные краевые условия	792
Использование сплайнов в В-форме	795
Сглаживающие сплайны	802
Интерактивное построение кривых	807
Приложение <i>splinetool</i>	809
Сплайны для поверхностей	814
Глава 19. Приближение данных и подбор параметров в Curve Fitting Toolbox.....	819
Приложение Curve Fitting Tool и его средства	820
Создание множества данных для приближения.....	821
Предварительная обработка данных	824
Исключение данных из таблицы	824
Начальная фильтрация табличной функции	826
Приближение табличных функций	829
Создание приближений	829
Контроль качества приближений	832
Типы аппроксимации для подбора параметров	836
Определение собственной параметрической модели	840
Анализ построенных приближений	845
Глава 20. Решение экономических задач	848
Функции для работы с датами и временем	849
Представление времени и дат в MATLAB	849
Функции определения числа дней между датами	852
Расчеты денежных потоков.....	856
Расчеты по обслуживанию кредитов	864
Расчеты по долговым ценным бумагам	867
Дисконктные активы	867
Купонные облигации	871
Портфельный анализ рисковых активов	881
Построение эффективной границы рисковых активов	883
Оптимальный выбор портфеля	887
Дополнительные ограничения при анализе портфелей	896

ЧАСТЬ V. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ MATLAB	909
Глава 21. Связь MATLAB и MS Office	911
Публикация результатов работы	911
М-книги.....	914
Настройка MATLAB и создание М-книги.....	914
Группировка ячеек	917
Управление М-книгой	920
Совместная работа в MATLAB и MS Excel	921
Конфигурирование MS Excel.....	922
Обмен данными между MATLAB и MS Excel	923
Обращение к основным функциям MS Excel Link.....	925
Функции MS Excel Link	927
Глава 22. Модернизация приложений с GUI версии 5.3.....	929
Пример приложения для MATLAB 5.3.....	929
Модернизация приложения	931
Сохранение приложения в формате FIG	932
Переход к форматам FIG и M	934
Глава 23. Повышение производительности приложений MATLAB	938
Ускорение работы М-файлов, экономия памяти.....	938
Поэлементные операции	938
Экономия памяти	941
Выделение памяти под массивы	944
Связь MATLAB с другими языками программирования	946
Конфигурирование MATLAB Compiler	947
Простой пример, сложение двух чисел	948
Работа с комплексными переменными	952
Обмен массивами данных	954
Ускорение работы при использовании циклов.....	958
ПРИЛОЖЕНИЯ.....	961
Приложение 1. Основные команды и функции MATLAB и Toolbox	963
Управление средой, файлами и переменными	963
Получение справочной информации	963
Управление средой MATLAB.....	964
Управление переменными	967
Манипулирование файлами и каталогами.....	969
Операторы и специальные символы	971
Логические операции и операторы	972
Побитовые операции.....	973
Логические функции	978
Программирование.....	981
Конструкции языка	981
Сервисные функции и переменные.....	982

Интерактивный ввод	984
Объектно-ориентированное программирование и преобразование типов	986
Функции даты и времени	986
Двоичные и текстовые файлы	987
Функции для работы с массивами ячеек	995
Функции для работы со структурами	999
Звуковые и графические файлы	1003
Чтение, запись и преобразование звуковых данных	1003
Графические файлы	1004
Операции со строками	1007
Обработка строк	1007
Преобразования "строка-число"	1011
Преобразование системы счисления	1015
Работа с матрицами и массивами	1016
Создание матриц и массивов	1016
Операции с массивами	1018
Математические функции	1019
Специальные функции	1019
Преобразование координат	1024
Решение различных математических задач	1025
Матричный анализ	1025
Решение спектральных задач	1028
Решение линейных уравнений, разложение и обращение матриц	1029
Вычисление функций от матриц	1032
Поиск корней	1033
Интерполяция и приближение данных	1035
Минимизация и оптимизация	1036
Дифференцирование и конечные разности	1037
Интегрирование	1038
Решение дифференциальных уравнений и систем	1039
Графика и визуализация данных	1039
Интерактивная среда для построения и редактирования графиков	1039
Двумерные графики	1040
Трехмерные и контурные графики	1046
Визуализация векторных полей	1061
Визуализация функции на непрямоугольной области	1065
Оформление графиков	1068
Управление видом графика, камера	1072
Приложение 2. Описание компакт-диска	1081
Список литературы	1082

Введение

В1. О назначении и возможностях пакета MATLAB и его расширений

Название MATLAB является сокращением от Matrix Laboratory, и первоначально пакет MATLAB разрабатывался как средство доступа к библиотекам программ LINPACK и EISPACK, предназначенных для матричных вычислений. Пакет MATLAB создан компанией MathWorks около двадцати 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 и его расширений (Toolbox), охватывает: матричный анализ, обработку сигналов и изображений, задачи математической физики, оптимационные задачи, финансовые задачи, обработку и визуализацию данных, работу с картографическими изображениями, нейронные сети, нечеткую логику и многое другое. Около сорока специализированных Toolbox 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. Доступ из командной строки к кратким сведениям о встроенных функциях обеспечивает возможность быстрого 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 ба-

зу, 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.

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. Интерактивная среда для построения графиков позволяет обойтись без графических функций для визуализации данных. Кроме того, она служит и для оформления результата желаемым образом: размещения поясняющих надписей, задания цвета и стиля линий и поверхностей, словом, для получения изображения, пригодного для включения в отчет или статью. Полный доступ к изменению свойств отображаемых графиков дают низкоуровневые функции, применение которых подразумевает понимание принципов компьютерной графики и владение приемами программирования.

В MATLAB реализованы классические численные алгоритмы решения уравнений, задач линейной алгебры, нахождения значений определенных интегралов, аппроксимации, решения систем или отдельных дифференциальных уравнений. Для применения базовых вычислительных возможностей достаточно 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 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.

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 со своими методами.

MATLAB прекрасно интегрируется со многими приложениями и средами программирования. Связь MATLAB и MS Word обеспечивает возможность написания в редакторе MS Word интерактивных документов, так называемых М-книг, основанных на специальном шаблоне. Пользователь, работающий с М-книгой, может запускать блоки команд MATLAB непосредственно из документа MS Word, причем результат выполнения команд отображается в М-книге. Данное средство прекрасно подходит для создания электронных отчетов и учебных пособий.

Надстройка MS Excel Link, поставляемая вместе с MATLAB, существенно расширяет возможности MS 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 (VBA). MATLAB Builder for MS Excel позволяет реализовывать алгоритмы MATLAB в виде COM-объектов и использовать их в приложениях на VBA.

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.

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.

Программный интерфейс приложения (API) реализует связь среды MATLAB с программами, написанными на C, Fortran или Java. Библиотека программного интерфейса позволяет вызывать имеющиеся модули на C, Fortran или Java из среды или программ MATLAB, обращаться к функциям MATLAB из программ на C или Fortran, осуществлять обмен данными между приложениями MATLAB и другими программами. Средства MATLAB Builder for COM предназначены для преобразования программ MATLAB в COM-объекты, доступные в других приложениях.

Для разработки интернет-приложений MATLAB создан MATLAB Web Server, причем процесс создания приложения достаточно прост — кроме умения программировать в MATLAB требуется только знание основ HTML.

Подводя итог вышесказанному, можно сделать вывод, что начинающий пользователь MATLAB может в процессе работы совершенствовать свои знания как в области моделирования и численных методов, так и программирования, и визуализации данных. 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.

Далее мы перечислим основные возможности Toolbox. Чтение этих сведений вы можете сопровождать запуском демонстрационных файлов MATLAB. Для этого следует запустить MATLAB и в меню **Help** выбрать пункт **MATLAB Help**. Появляется окно интерактивной справочной системы, в левой части которого на вкладке **Demos** содержатся разделы MATLAB и Toolboxes. Щелчок мышью по названию раздела приводит к его раскрытию и отображению содержимого подразделов в правой части окна справочной системы (рис. B1).

Посмотрите, например, обзор базовых графических возможностей пакета, который содержится в подразделах **Graphics** и **3-D Visualization**. К примеру, в подразделе **Graphics** перейдите к пункту **Functions of Complex Variables** и запустите соответствующую демонстрацию при помощи двойного щелчка мышью по названию пункта или щелчка мышью по ссылке **Run this demo** в верхнем правом углу окна справочной системы. В появившемся окне **Functions of Complex Variables** нажмайте кнопку **Next**, при этом строятся графики функций комплексных переменных, а сами выражения для функций выводятся в заголовке графиков. Обратите внимание на текстовую область под графиком, в которой отображаются те команды, которые надо вызвать в MATLAB для получения подобных результатов.



Рис. В1. Окно справочной системы со ссылками на демонстрационные файлы

Некоторые возможности MATLAB представлены видеодемонстрациями, например: использование интерактивной среды для построения графиков (пункт **Interactive Plot Creation with the Plot Tools** подраздела **Graphics**), создание приложений с графическим интерфейсом (пункт **Creating a GUI with GUIDE** подраздела **Creating Graphical User Interfaces**). Если вы работали в прежних версиях MATLAB, то вам окажутся полезными видеообзоры новых возможностей среды, редактора, графики и программирования, которые содержатся в подразделе **New Features in Version 7**.

Двойной щелчок мышью по разделу **Toolboxes** приводит к раскрытию списка подразделов. Темы каждого подраздела охватывают многие практические важные задачи, которые могут быть решены при помощи данного Toolbox. Разумеется, возможности Toolbox не исчерпываются представленными в демонстрациях. Далее приведено краткое описание некоторых Toolbox.

Примечание

Для того чтобы узнать, какие именно Toolbox входят в установку MATLAB на вашем компьютере, достаточно запустить MATLAB и в командной

строке (обозначенной символом **>>**) набрать `ver` и нажать <Enter>. Выводятся названия всех доступных Toolbox с указанием их версий.

Средства обработки сигналов собраны в нескольких Toolbox. Базовые инструменты для решения задач обработки сигналов находятся в Signal Processing Toolbox:

- генерация, импорт и экспорт сигналов;
- разработка, анализ и применение цифровых фильтров с конечной и бесконечной импульсной характеристикой;
- конструирование аналоговых фильтров;
- преобразования, спектральный анализ и статистическая обработка сигналов;
- моделирование параметрических временных рядов.

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. Более широкие возможности для конструирования фильтров предоставляют Filter Design Toolbox и Filter Design HDL Coder. Разработка и анализ высокочастотных цепей могут быть осуществлены в RF Toolbox. Для построения моделей систем на основе экспериментально полученных данных служит System Identification Toolbox.

При исследовании и обработке сигналов и изображений оказывается очень полезным вейвлетный анализ, который можно проделать средствами Wavelet Toolbox.

Алгоритмы обработки изображений собраны в Image Processing Toolbox, функции которого позволяют осуществить:

- 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 Image Processing Toolbox входит несколько демонстрационных приложений, охватывающих решение задач о нахождении границ изображений, фильтрации и разработки фильтров, сжатии изображения. Для работы с картографическими изображениями имеется отдельный Mapping Toolbox. Получение видеинформации от внешних устройств может быть выполнено средствами Image Acquisition Toolbox.

Среда MATLAB позволяет осуществить импортирование цифровых и аналоговых данных с использованием совместимого с РС оборудования, их обработку и экспортацию. Для обмена данными служат Data Acquisition Toolbox и Instrument Control Toolbox, функции которых поддерживают работу с оборудованием известных производителей.

Для статистической обработки информации и анализа данных имеется несколько Toolbox. 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 Statistics Toolbox, включая:

- исследование линейных моделей;
- параметрическое оценивание;
- проверку гипотез;
- планирование эксперимента;
- задачи кластерного анализа и др.

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

Приближение данных различными способами реализовано в Curve Fitting Toolbox и Spline Toolbox, предоставляющих следующие возможности:

- предварительная обработка табличных данных до приближения;
- параметрическое и непараметрическое сглаживание;

- аппроксимация с использованием линейных и нелинейных моделей, причем имеется библиотека широко распространенных моделей и предусмотрена возможность создания собственных;
- устойчивые методы подбора параметров;
- вычисление различных критериев приближений;
- анализ данных: экстраполяция, интегрирование, дифференцирование;
- представление сплайнов в кусочно-полиномиальной форме и *B*-форме, преобразование из одной формы в другую;
- интерполяция и сглаживание при помощи сплайнов;
- вычисление производных, интегралов и отображений от сплайнов;
- выбор оптимального расположения узлов сплайна;
- тензорное произведение сплайнов для конструирования многомерных сплайн-функций;
- рациональные сплайны;
- применение сплайнов для решения нелинейных обыкновенных дифференциальных уравнений.

Проблемы, возникающие в различных областях экономики и финансов, могут быть исследованы при помощи специализированных алгоритмов нескольких Toolbox: Financial, Financial Derivatives, GARCH, Financial Time Series, Datafeed и Fixed-Income. Перечислим основные задачи, решение которых может быть выполнено функциями и приложениями данных Toolbox:

- вычисление и анализ цены, доходности и чувствительности отдельных финансовых активов (основных и производных ценных бумаг) на срочном и спотовом рынках;
- проведение анализа для управления портфелями ценных бумаг;
- подбор и оценивание стратегий хеджирования (страхования) операций на различных рынках;
- управление рисками, их выявление и оценивание;
- вычисление потоков платежей и их анализ, в том числе оценка инвестиционных проектов, анализ и прогнозирование экономических показателей;
- проектирование составных финансовых инструментов, включая валютные операции;

- моделирование и оценивание волатильности временных рядов методами математической статистики, проверка статистических гипотез, выявление корреляции обрабатываемых случайных процессов;
- калькуляция цены, доходности и чувствительности активов с фиксированным доходом по методикам Ассоциации индустрии ценных бумаг США и Канады;
- импортирование статистических данных с финансовых рынков и проведение на их основе технического анализа с привлечением большого количества индикаторов.

Как мы уже упоминали, MATLAB поддерживает символьные вычисления. Symbolic Math Toolbox содержит функции, обеспечивающие доступ к вычислительному ядру Maple. Пользуясь ими, вы можете: производить интегрирование и суммирование, вычислять пределы и находить разложение функций в ряд, упрощать выражения, находить определители, решать задачи на собственные значения, применять различные преобразования, решать алгебраические и дифференциальные уравнения, проводить вычисления с любой точностью, словом, использовать все возможности символьной математики Maple. Интерфейс с Maple и входящими в его состав пакетами может быть наложен средствами Extended Symbolic Math Toolbox.

Optimization Toolbox 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.

Partial Differential Equations Toolbox (PDE Toolbox) создан для решения задач математической физики, описываемых дифференциальными уравнениями и системами в частных производных, методом конечных элементов. 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, например таких, как:

- теория упругости;
- электростатика и магнитостатика;

- теплопроводность;
- теория диффузии.

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

PDE Toolbox является хорошим компактным пакетом для обучения основам метода конечных элементов и введения в конечноэлементные пакеты. Однако серьезные инженерные практические задачи вряд ли могут быть решены в нем, поскольку он поддерживает только один тип конечных элементов — линейные треугольные. До 2003 г. в состав MATLAB входило приложение FEMLAB, которое затем стало отдельным программным продуктом, но FEMLAB может быть установлен и как приложение MATLAB. FEMLAB позволяет моделировать двумерные и трехмерные задачи, описываемые дифференциальными уравнениями в частных производных: перенос, течения, упругость и электромагнетизм.

Следует подчеркнуть, что MATLAB и его расширения 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. В 7-ую версию MATLAB включен Bioinformatics Toolbox, предназначенный для решения некоторых задач, лежащих на стыке биологии и информатики.

В начале введения мы упомянули пакет Simulink, поставляемый вместе с MATLAB, который служит для исследования нелинейных динамических систем. Он хорошо интегрируется в среду MATLAB и расширяет ее возможности. Например, сочетание Simulink и Signal Processing Blockset позволяет в удобной среде разрабатывать алгоритмы обработки сигналов и генерировать код на С. Обзор возможностей, предоставляемый Simulink, занимает много места — мы отсылаем заинтересованных читателей к нескольким книгам [2, 5].

B2. О содержании книги

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* описаны рабочая среда и приемы эффективной работы из командной строки. Объяснено использование переменных и вычисление арифметических выражений, изменение формата вывода чисел и основные встроенные математические функции.

Глава 2 книги подробно разъясняет принципы работы с матрицами и векторами, включая основы визуализации векторных и матричных данных. Особенности представления данных в виде массивов, в частности, матриц и векторов, дают пользователю более широкие возможности по сравнению с большинством языков программирования. Набор специальных функций и средств унифицирует работу с массивами данных, делая ее очень эффективной. Отсутствие навыков оперирования с массивами в MATLAB приводит к многочисленным затруднениям даже при решении самых простых задач.

Глава 3 нацелена на обучение читателя свободному владению средствами высокогоуровневой графики для построения диаграмм и гистограмм, линий, поверхностей и векторных полей. Пакет MATLAB обладает чрезвычайно мощными возможностями визуализации одномерных и многомерных данных различных типов, включая и построение графиков функций. Приведены команды, служащие для организации и оформления графических результатов с целью получения хорошо читаемых графиков.

Интерактивная среда для построения графиков позволяет визуализировать данные, не прибегая к командам MATLAB. Кроме того, инструменты интерактивной среды могут быть использованы для редактирования существующих графиков, изменения свойств всех содержащихся на них объектов и манипулирования графиком, в частности, для осмотра поверхности со всех сторон. Эти вопросы, а также экспорт графических результатов и их печать описаны в *главе 4*.

Работа из командной строки, разумеется, не очень удобна и подходит только для решения простых задач. Выход состоит в использовании М-файлов, т. е. программ и функций, содержащих нужную последовательность команд MATLAB. Написание основных типов М-файлов (файл-программ и файл-функций) во встроенном редакторе разобрано в *главе 5*. М-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 М-файлах, что дает опытному пользователю уникальную возможность разбираться в осо-

бенностях 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 одном из алгоритмических языков, например, Basic, С 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, знаний, как минимум в объеме программы технических вузов. Поиск корней и минимизация функций, интегрирование и интерполирование, решение задач линейной алгебры, обыкновенных дифференциальных уравнений и систем разобраны в главе 6. Вычислительные алгоритмы MATLAB допускают их настройку на получение результата с определенной точностью и задание ряда опций для управления ходом вычислений. Данный круг вопросов также освещается в главе 6.

Глава 7 содержит описание основных конструкций языка программирования MATLAB, включая операторы ветвления и циклов. Описаны логические операции и логическое индексирование в применении к массивам, которые зачастую позволяют сократить объем программы и повысить ее эффективность.

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. Уделено внимание созданию рекурсивных функций. Программирование сложных алгоритмов нередко требует их отладки. Редактор MATLAB содержит набор средств для отладки программ, использование которых также пояснено в главе 8.

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 интерактивной среды для редактирования графиков. MATLAB является

объектно-ориентированной системой, все его 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 главе 9, демонстрируют основные возможности, имеющиеся в распоряжении разработчика графических программ в системе 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.

В главе 10 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.

Читая главу 11, вы продолжите работу над созданным приложением, пополняя его интерфейс флагами, переключателями, областями ввода и полосами скроллинга и программируя их с учетом обеспечения согласованной работы всех элементов управления.

О том, как снабдить собственное приложение диалоговыми окнами, меню, в том числе контекстными, сообщается в главе 12. Удобство работы с приложением во многом определяется хорошо продуманной структурой меню. Изменение структуры меню так же описано в этой главе.

Глава 13, завершающая третью часть книги, содержит некоторые дополнительные сведения о программировании событий графических объектов, например, щелчка мышью или нажатия клавиши, и ряде свойств, связанных с этими событиями.

Четвертая часть книги посвящена применению Toolbox для исследования некоторых специальных задач. Глава 14 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 с графическим интерфейсом: 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 PDE Toolbox. Непосредственное использование данных функций в собственных программах позволяет проводить более сложные исследования по сравнению с возможностями `pdetool`. В связи с этим в главе 14 приведено описание форматов представления данных, связанных с реализацией метода конечных элементов в PDE Toolbox, и разобраны примеры использования функций Toolbox.

Решение многих современных сложных 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 15 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 Optimization Toolbox разобрано в главе 16. Эффективное использование оптимизационных алгоритмов для решения сложных задач требует понимания методов и умения работать с разреженными матрицами. Приведен пример решения большой системы нелинейных уравнений. Отдельный раздел главы 16 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 Symbolic Math Toolbox. Символьные вычисления основаны на мощном ядре Maple, при этом пользователь имеет доступ ко всем ресурсам MATLAB. Глава 17 этой книги нацелена на обучение пользователя работе с символьными выражениями, включая упрощение, преобразование и вычисление 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.

В главе 18 рассматривается приближение интерполяционными и сглаживающими сплайнами в Spline Toolbox. Сплайны могут быть сконструированы в *pp*-форме и *B*-форме. Описываются возможности задания различных граничных условий и их влияние на точность аппроксимации. Обсуждается ключевой момент в создании *B*-сплайнов — кратность узлов. Средства Toolbox позволяют приближать функции с различной степенью гладкости. Приводятся примеры построения кривых и поверхностей с помощью сплайнов.

Возможности аппроксимации табличных функций, реализованные в Curve Fitting Toolbox, излагаются в главе 19. Описываются способы предварительной обработки данных, основанные на регрессионном анализе и других методах для начальной фильтрации данных. Разобраны создание собственных параметрических моделей и использование одной из стандартных на примере рациональной или частичной суммы ряда Фурье. Материал этой главы позволит читателю выполнить непараметрическое приближение либо сглаживающими сплайнами, либо интерполяционными методами. Поясняется, как провести анализ результатов, включающий экстраполяцию табличной функции, интегрирование и дифференцирование полученного приближения.

Глава 20 знакомит читателя с возможностями решения экономических задач в Financial Toolbox. Рассмотрены вопросы анализа потоков платежей в различных сферах бизнеса. Иллюстрируется применение функций Toolbox для расчетов, связанных с обращением купонных и бескупонных облигаций. На примере модельных задач демонстрируются возможности по управлению портфелями рискованных ценных бумаг. Рассматриваются различные виды ограничений на состав портфеля и поясняются правила формирования и использования этих ограничений.

Последняя, пятая, часть книги охватывает несколько вопросов, которые могут быть полезны читателям с различными уровнями подготовки.

Глава 21 рассказывает об автоматической генерации отчетов и о работе с MATLAB в популярных форматах MS Word, MS Power Point, HTML и TeX. Интегрирование MATLAB с MS Word позволяет создавать в MS Word интерактивные документы (*M*-книги) для представления постановки задачи, методов и результатов расчетов в наглядной форме с использованием всех возможностей мощного текстового редактора MS Word и среды MATLAB. Читатель *M*-книги может запускать блоки команд MATLAB и получать текстовый и графический результат прямо в *M*-книге. Раздел "*Совместная работа в MATLAB и MS Excel*" главы 21 содержит информацию о конфигурировании MS Excel и организации совместной работы в MATLAB и MS Excel. Возможен не только обмен данными между средой MATLAB и таблицами MS Excel, но и вызов функций MATLAB, как из ячеек листа, так и из приложений на VBA.

Пользователям, которые имеют приложения с графическим интерфейсом, созданные в MATLAB версии 5.3, несомненно, окажется полезной информация о модернизации приложений в формат, принятый в новых версиях. Глава 22 описывает процесс преобразования приложений из формата m/mat, поддерживаемого в MATLAB 5.3, в формат m/fig, который используется в версиях, начиная с шестой.

Эффективное оперирование с данными большого объема в MATLAB подразумевает применение ряда приемов, которые описаны в главе 23. В ней обсуждаются: распределение памяти, преимущество встроенных поэлементных операций по сравнению с циклической организацией обработки данных и выбор данных подходящего типа. Работа пользователя MATLAB не ограничена только возможностями среды и модулей Toolbox. В пакет MATLAB входит библиотека функций MATLAB API, реализующих программный интерфейс приложений. Глава 23 содержит также основные сведения о MATLAB API и примеры интерфейса для внешних модулей, написанных на других языках программирования. В ряде случаев задействование внешних модулей повышает эффективность приложений MATLAB.

Основные функции MATLAB и ряда Toolbox, сгруппированные по категориям, приведены в *приложении 1*. Краткое описание различных вариантов вызова функций снабжено ссылками на соответствующие разделы книги, в которых обсуждается использование данных функций.

Для удобства работы с книгой все листинги приводимых программ занесены на прилагаемый компакт-диск, структура которого описана в *приложении 2*. Изложение материала в книге сопровождается примерами, а в конце некоторых глав приведены задания для самостоятельной работы.

Данная книга ни в коей мере не претендует на полноту изложения. Достаточно сказать, что документация по MATLAB и Toolbox весьма объемна, в частности, описание PDE Toolbox содержит около трехсот страниц, Optimization Toolbox — около четырехсот, а описание Statistics Toolbox превосходит девятьсот страниц. Следует иметь в виду, что справочная 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. Поэтому часто мы приводим ссылки на разделы справочной системы для самостоятельного изучения материала.

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 кропотливой самостоятельной работы. Список литературы, касающейся программирования и решения задач в MATLAB, приведен в конце книги.

Пользователям предыдущей версии MATLAB будут интересны новшества, появившиеся в седьмой версии. Расширены возможности рабочей среды и ее компонент. Стало удобнее работать с несколькими файлами и графическими окнами — окна редактора M-файлов, графические окна и браузер переменных могут быть встроены в рабочую среду. Сохранив вид рабочей среды, вы легко восстановите его во время следующего сеанса работы. Появилась новая панель инструментов для размещения ярлыков, обеспечивающих быстрый доступ к собственным программам, приложениям и командам MATLAB. Редактор массивов и браузер переменных содержат средства для визуализации данных; кроме того, редактор массивов позволяет просматривать структуры, в том числе и вложенные.

Графические средства претерпели некоторые изменения. Вместо редактора графиков появилась интерактивная среда для визуализации данных, не требующая обращения к графическим функциям. Данная среда содержит ряд компонентов, которые могут быть использованы для редактирования графиков. При оформлении графиков теперь допускается запись математических формул в формате LaTeX и добавление новых объектов: выносных подписей, геометрических фигур и разного рода стрелок. Эти поясняющие объекты могут быть привязаны к точке с заданными координатами и не менять своего положения, например, при выборе нового масштаба.

Инструмент **Data Cursor** графического окна предназначен для отображения координат точек графиков при помощи мыши и прикрепления к ним ярлыков со значениями координат. Построив и оформив графики, вы имеете возможность автоматически сгенерировать код, выполнение которого приведет к созданию графического окна с тем же содержимым. Данный код легко добавить в собственное приложение и модифицировать по мере надобности.

Несколько изменилась структура графических объектов. Теперь в нее включены объекты-группы, облегчающие выполнение однотипных действий с набором объектов.

В процессе программирования и отладки приложений оказывается полезным анализатор кода M-Lint, который подсказывает эффективные способы ускорения алгоритма и оптимизации работы с памятью. Отладчик M-файлов снабжен условной точкой останова.

Предлагается новый способ организации работы в M-файле — блоки команд являются ячейками и могут быть выполнены в нужной последователь-

ности. С этим новшеством связан удобный способ представления результатов работы в виде автоматически генерируемого отчета в одном из распространенных форматов MS Word, MS Power Point, HTML или LaTeX.

Среда визуального программирования предлагает ряд новых объектов — панель для упрощения работы с переключателями и кнопки-переключатели. Ваше приложение с графическим интерфейсом теперь может использовать ActiveX-компоненты.

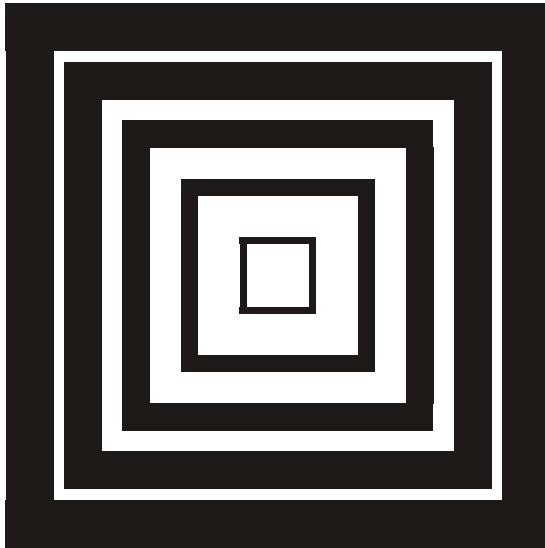
Появились два новых типа функций — анонимные и вложенные. Они, в частности, могут быть использованы для решения математических задач, содержащих параметры. Соответствующие вычислительные функции MATLAB поддерживают и прежний способ обращения к ним, при котором параметры указываются в качестве дополнительных аргументов.

Расширены и базовые вычислительные возможности — в состав MATLAB вошли солверы для решения обыкновенных дифференциальных уравнений и систем, не разрешенных относительно старшей производной.

Изменения коснулись ряда Toolbox и компонент MATLAB; более того, появились несколько новых. Например, кроме упомянутых нами Bioinformatics Toolbox и Filter Design HDL Coder создан Genetic Algorithm and Direct Search Toolbox. Он содержит функции, расширяющие возможности Optimization Toolbox и вычислительных функций MATLAB для решения оптимизационных задач.

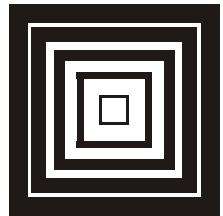
Подробная информация о новшествах содержится на сайте производителя MATLAB — компании MathWorks: <http://www.mathworks.com>. На этом сайте доступна документация в формате PDF, также вы можете найти руководства по использованию различных возможностей MATLAB, не включенные в справочную систему, и готовые решения. Из русскоязычных сетевых ресурсов мы рекомендуем сайт <http://matlab.exponenta.ru>, который содержит разделы, связанные с различными аспектами работы в MATLAB, Toolbox, Simulink и FemLab. Ведущие разделов не только размещают материалы, но и отвечают на вопросы пользователей.

Мы с благодарностью примем ваши замечания и пожелания по поводу нашей книги и постараемся ответить на все ваши вопросы. Желаем вам успехов в изучении MATLAB.



ЧАСТЬ I

**Основы работы
в MATLAB**



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

При запуске MATLAB на экране открывается рабочая среда **MATLAB**, изображенная на рис. 1.1.

Основными элементами рабочей среды являются:

- меню;
- панель инструментов с кнопками и раскрывающимся списком;
- окна с вкладками **Workspace** и **Current Directory** для просмотра переменных и установки текущего каталога;
- окно **Command Window**, служащее для ввода команд и вывода результата;
- окно **Command History**, предназначенное для просмотра и повторного выполнения ранее введенных команд (окно **Command History** может быть не пустым, если до этого пакет MATLAB использовался);
- строка состояния с кнопкой **Start**.

При нажатии на кнопку **Start** открывается меню, приведенное на рис. 1.2. С его помощью обеспечивается доступ ко всем основным средствам MATLAB.

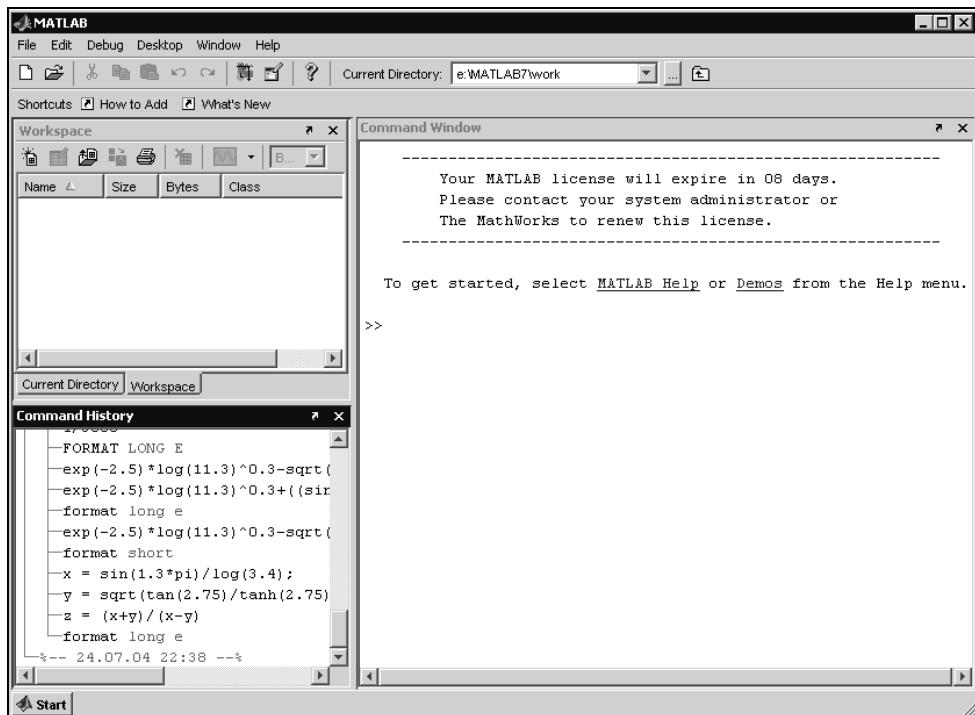


Рис. 1.1. Рабочая среда MATLAB

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

- заголовка с названием окна и двумя кнопками справа;
- рабочей области с командной строкой, в которой находится мигающий вертикальный курсор;
- полос скроллинга.

Примечание

Если рабочая среда MATLAB выглядит не так, как изображено на рис. 1.1, то следует в меню **Desktop** выбрать пункт **Desktop Layout**, а затем в ниспадающем меню пункт **Default** (По умолчанию). Пункты меню **Desktop** позволяют добавлять одноименные окна в рабочую среду или убирать их. Открытые окна в меню отмечены флагом.

В поле заголовка каждого окна рядом с кнопкой закрытия находится кнопка **Undock ...** для извлечения окна из рабочей среды MATLAB, если оно

встроено, или кнопка **Dock ...** для встраивания отдельного окна в рабочую среду. В следующих главах описано использование указанного средства для различных окон.

Все команды, описанные в этой главе, следует набирать в командной строке. Сам символ `>>` приглашения командной строки, приведенный в примерах, набирать не нужно. Для просмотра рабочей области командного окна 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 команда 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.

Примечание

При запуске пакета в рабочей области окна **Command Window** появляются две ссылки **MATLAB Help** и **Demos** для вызова справочной системы или демонстрационных примеров. Об использовании среды **Help** см. далее в *этой главе*.



Рис. 1.2. Раскрывающееся меню по кнопке **Start**

Арифметические вычисления

Встроенные математические функции 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

Выберите вид рабочей среды "по умолчанию", как было описано выше, 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. Переменная `ans` автоматически создается, когда вычисляемое выражение не присваивается некоторой переменной. Информация о переменной `ans` сразу же появилась в окне **Workspace** (рис. 1.3). В первом столбце **Name** записано имя переменной. Следующий столбец **Value** показывает значение переменной, если это возможно. Содержимое столбца **Size**, по существу, демонстрирует основной принцип работы MATLAB. Программа MATLAB *все данные представляет в виде массивов*. Переменная `ans` является двумерным массивом размера один на один и занимает 8 байт памяти, о чем свидетельствует столбец **Bytes**. Наконец, в последнем столбце **Class** указан тип переменной — `double array`, т. е. массив, состоящий из чисел двойной точности. Любой столбец можно скрыть или отобразить, если на заголовке окна щелкнуть правой кнопкой и вызвать контекстное меню.

Примечание

По умолчанию все числа представляются с двойной точностью (`double`). В главе 2 говорится об использовании других типов чисел.

В окне **Command Window** ниже ответа расположена командная строка с мигающим курсором, обозначающая, что среда MATLAB готова к дальней-

шим вычислениям. Можно набирать в командной строке новые выражения и находить их значения.

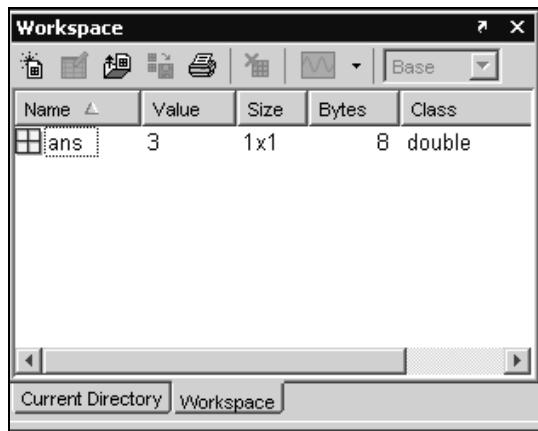


Рис. 1.3. Окно **Workspace** с информацией об использованных переменных среды

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
>> |
```

Вид, в котором выводится результат вычислений, зависит от формата вывода, установленного в MATLAB. Рассмотрим подробнее этот вопрос.

Форматы вывода результата вычислений

Требуемый формат вывода результата определяется пользователем из меню рабочей среды MATLAB. Выберите в меню **File** пункт **Preferences**. На экране появится диалоговое окно **Preferences**, изображенное 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**.

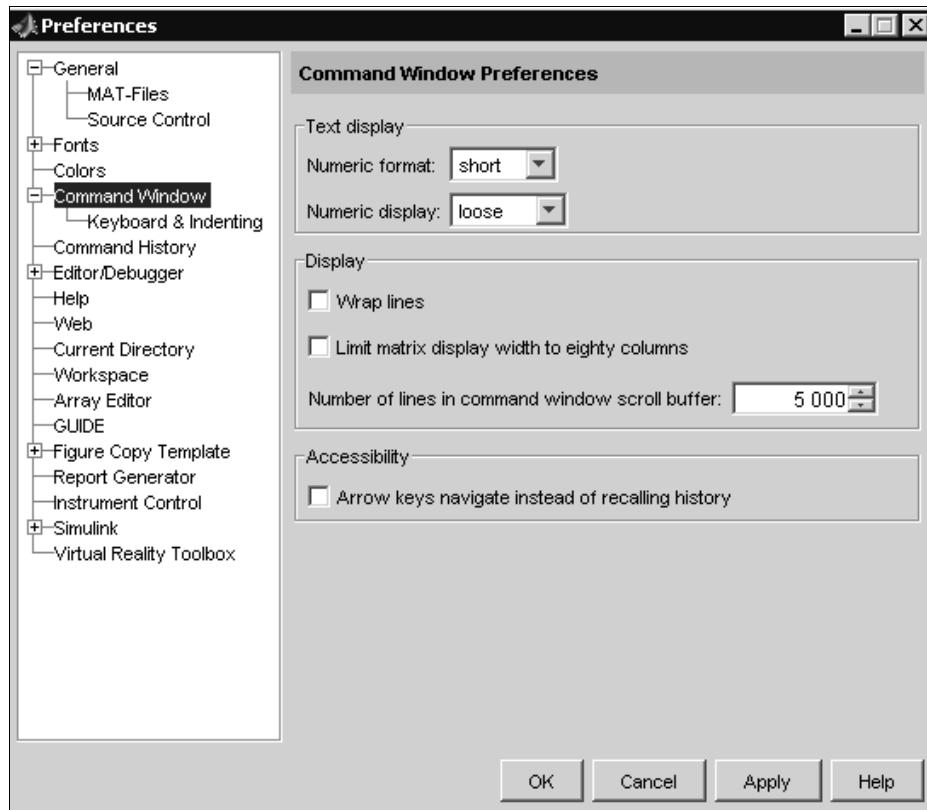


Рис. 1.4. Диалоговое окно Preferences MATLAB

Разберем пока только наиболее часто используемые форматы. Выберите **short** в раскрывающемся списке **Numeric format** на панели **Text display** диалогового окна. 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
```

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
```

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 $1e10$, чем $10\ 000\ 000\ 000$, 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
```

```
|
```

```
Error: Missing MATLAB operator.
```

Если требуется получить результат вычислений более точно, то следует выбрать на панели **Text display** в раскрывающемся списке **Numeric Format** значение **long**. Результат будет отображаться в длинном формате с плавающей точкой `long` с четырнадцатью цифрами после десятичной точки. Форматы `short` и `long` 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
??? Undefined command/function 'FORMAT'.
```

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** выбрать **compact** из раскрывающегося списка **Numeric display**. Добавление пустых строк обеспечивается выбором **loose** из раскрывающегося списка **Numeric display**.

Примечание

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:

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

Ведите 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:

- возвведение 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

$$e^{-2.5} \cdot (\ln 11.3)^{0.3} + \left(\frac{\sin 2.45\pi + \cos 3.78\pi}{\operatorname{tg} 3.3} \right)^2,$$

то не обязательно 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 возвведение 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 \uparrow до тех пор, пока в командной строке не появится требуемое выражение и вычислить его, нажав `<Enter>`.

```
>> format long e
>> exp(-2.5)*log(11.3)^0.3 - sqrt((sin(2.45*pi) + cos(3.78*pi))/tan(3.3))
ans =
-3.210497097863031e+000
```

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

```
>> format short
>> ans
ans =
-3.2105
```

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. Более подробные сведения можно получить из обширной интерактивной справочной системы, выбрав в меню **Help** рабочей среды пункт **MATLAB Help**. Открывается окно справочной системы. В левой части окна на вкладке **Contents** приведено ее оглавление. Для просмотра содержимого какого-либо раздела в правом окне следует выбрать его название в левом окне при помощи щелчка мыши. Знак "+" слева от названия раздела служит для отображения его подразделов. Найдите

раздел **Functions -- Categorical List** (Функции по категориям), затем в подразделе **Mathematics** (Математика) — пункт **Elementary Math**. Перед вами список всех элементарных математических функций, имеющихся в MATLAB (рис. 1.5). Щелчок по гиперссылке с названием функции позволяет получить полную информацию о ней. Для перехода вперед и назад по просмотренным страницам справочной системы используйте кнопки со стрелками.

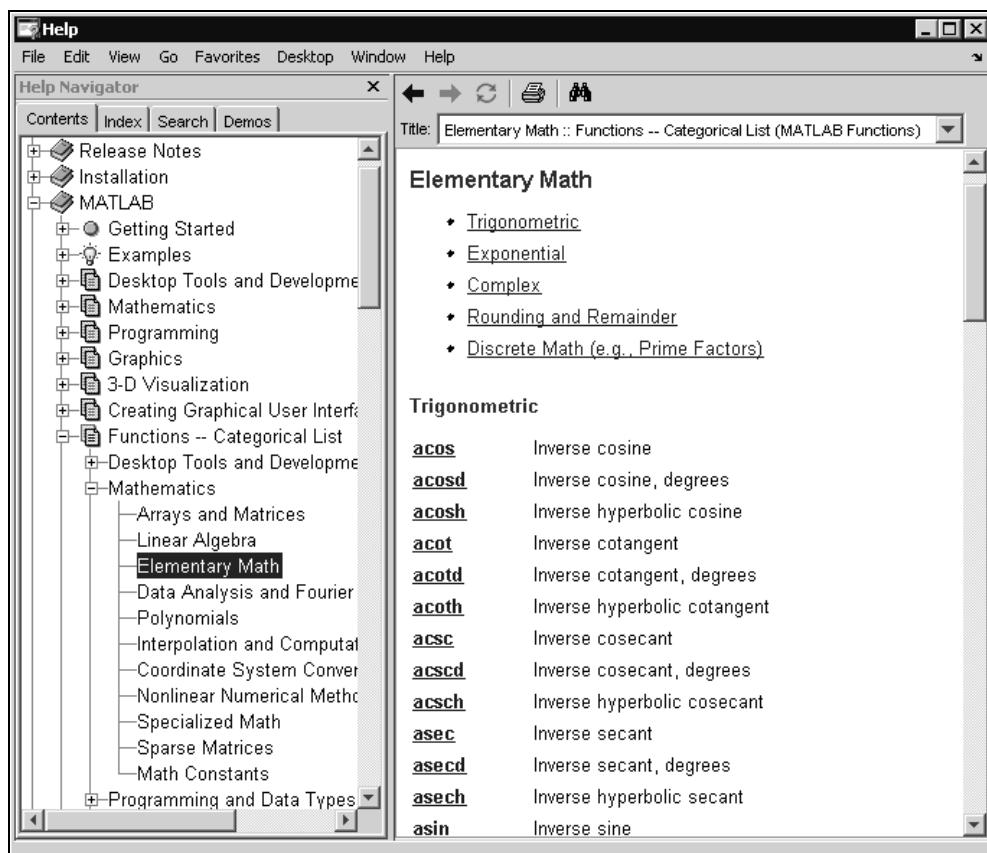


Рис. 1.5. Окно интерактивной справочной системы

Примечание

Можно получить быстрый доступ к информации о функции, используя контекстное меню в окне **Command Window**. Для этого надо выделить имя

функции в ранее набранной команде и, щелкнув правой кнопкой мыши, выбрать в меню пункт **Help on Selection**.

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` — аркsecанс, арккосеканс.

Примечание

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 аркsecанс и арккосеканс.

Экспоненциальная функция, логарифмы, степенные функции

Ниже перечислены названия этих функций в MATLAB:

- `exp` — экспоненциальная функция;
- `log` — натуральный логарифм;
- `log10` — десятичный логарифм;
- `log2` — 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` — мнимая и действительная 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)
```

```
ans =
```

```
1
```

```
>> fix(-1.9)
```

```
ans =
```

```
-1
```

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

```
>> floor(3.2)
```

```
ans =
```

```
3
```

```
>> ceil(3.2)
```

```
ans =
```

```
4
```

- `round` — округление до ближайшего целого:

```
>> round(4.1)
```

```
ans =
```

```
4
```

```
>> round(4.5)
```

```
ans =
```

```
5
```

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

```
>> mod(5,2)
```

```
ans =
```

```
1
```

```
>> mod(5,-2)
```

```
ans =
```

```
-1
```

- `rem` — остаток от целочисленного деления (со знаком первого аргумента):

```
>> mod(5,2)
```

```
ans =
```

```
1
```

```
>> mod(5,-2)
```

```
ans =
```

```
1
```

- `sign` — знак числа.

### Примечание

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. В интерактивной справочной системе аналогичные сведения содержит пункт **Specialized Math**, который находится в подразделе **Mathematics** раздела **Functions -- Categorical List**.

## 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, пе-

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 63 (такие длинные имена редко бывают нужны).

В качестве упражнения на использование переменных найдите значение следующего выражения:

$$\frac{\frac{\sin 1.3\pi}{\ln 3.4} + \sqrt{\frac{\operatorname{tg} 2.75}{\operatorname{th} 2.75}}}{\frac{\sin 1.3\pi}{\ln 3.4} - \sqrt{\frac{\operatorname{tg} 2.75}{\operatorname{th} 2.75}}}.$$

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
```

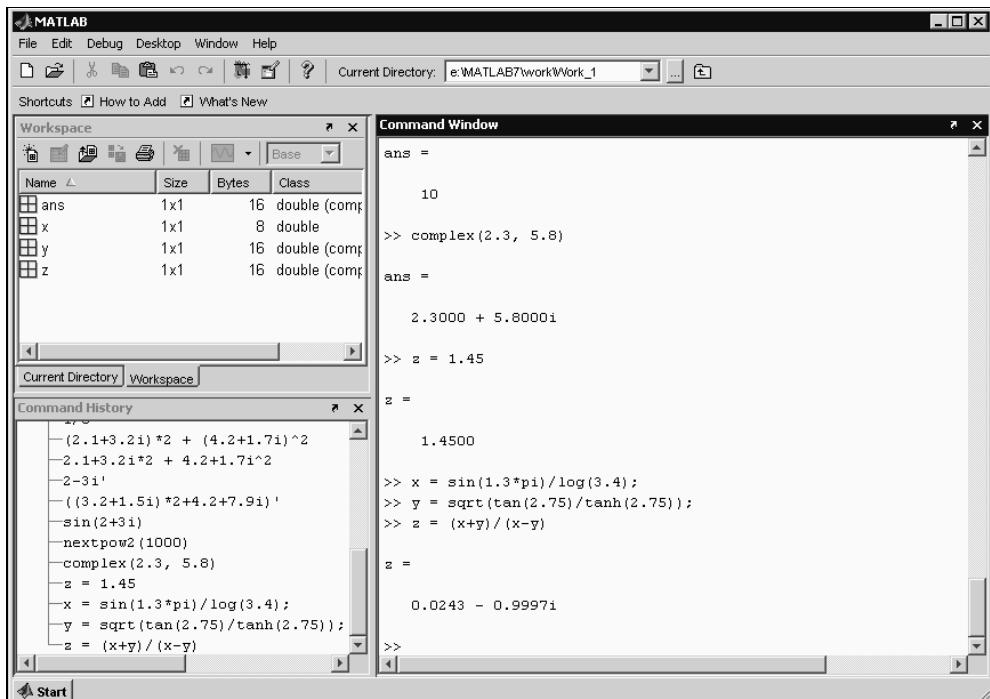
Последний оператор присваивания не завершается точкой с запятой для того, чтобы сразу получить значение исходного выражения.

Обратите внимание на то, что все введенные переменные сразу появились в окне **Workspace**, как показано на рис. 1.6 (в дополнение к стандартной переменной *ans*, которая использовалась ранее).

Конечно, можно было бы ввести 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
```



**Рис. 1.6.** Рабочая среда MATLAB после вычисления переменных

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.

### Примечание

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 про-

ланы 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

$$\left[ \frac{\sin 1.3\pi}{\ln 3.4} - \sqrt{\frac{\operatorname{tg} 2.75}{\operatorname{th} 2.75}} \right]^{3/2},$$

то достаточно ввести следующую команду:

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

Вызов функций в MATLAB обладает достаточной гибкостью. Например, вычислить  $e^{3.5}$  возможно, 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 и восстановление рабочей среды

Самый простой способ сохранить значения всех переменных — использовать в меню **File** пункт **Save Workspace As**. При этом появляется диалоговое окно **Save to MAT-File**, в котором следует указать каталог и имя файла. Предлагается сохранить файл в текущем каталоге (по умолчанию 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, work20-01-04. Выполните **Save** (Сохранить). MATLAB сохранит результаты работы в файле work20-01-04.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 закрытия окна MATLAB в правом верхнем углу его заголовка.

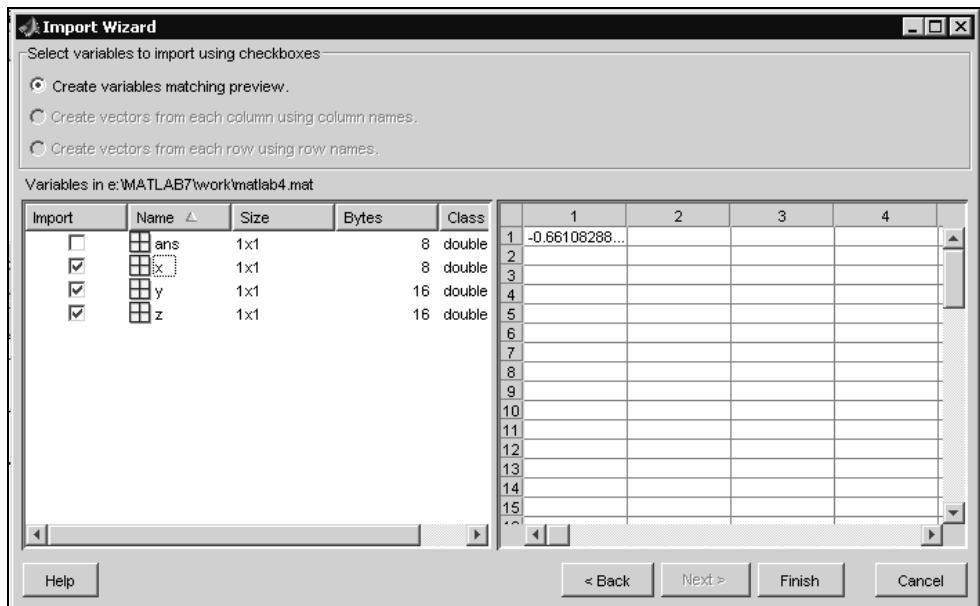
### Примечание

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 следующем сеансе работы для восстановления значений переменных следует открыть файл work20-01-04.mat при помощи подпункта **Open** меню **File**. Теперь все переменные, определенные в прошлом сеансе, стали доступными. Их можно использовать во вновь вводимых командах. Сохранить и восстановить переменные среды можно также с помощью кнопок **Load data file** и **Save** на панели инструментов окна **Workspace**. Для скрытия или отображения панели инструментов окна надо вызвать контекстное меню, щелкнув правой кнопкой на заголовке. При восстановлении переменных после выбора файла для их загрузки возникает диалоговое окно, изображенное на рис. 1.7, позволяющее просмотреть значения переменных и отметить флагами те, которые следует загрузить.

Можно сохранить значения одной или нескольких переменных. Для этого выберите переменную, щелкнув мышью по ее имени в окне **Workspace**. Для выбора нескольких переменных используйте стандартную комбинацию Windows — щелчок левой кнопкой мыши с удержанием клавиш <Ctrl> или <Shift>. Затем на выделении сделайте правый щелчок мышью и выберите пункт **Save As** в контекстном меню. Открывается диалоговое окно **Save to MAT-File**, в котором следует выбрать имя файла.

Обратите внимание, что созданный файл появился в окне **Current Directory**. Двойной щелчок мышью в этом окне по строке с именем файла приводит к восстановлению записанных в него переменных в рабочей среде.



**Рис. 1.7.** Диалоговое окно  
для восстановления переменных среды **Workspace**

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

```
>> save work20-01-04
```

Расширение можно не указывать, MATLAB сохранит переменные рабочей среды в файле `work20-01-04.mat`. В начале следующего сеанса работы для считывания переменных следует ввести команду

```
>> load work20-01-04
```

Более подробные сведения о командах `save` и `load` можно получить, набрав в командной строке `help save` или `help load`. В интерактивной справочной системе вся информация об этих командах находится в пункте **Opening, Loading, Saving Files** подраздела **File I/O** раздела **Functions -- Categorical List**.

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 d20-01-04.txt
>> a1 = 3;
>> a2 = 2.5;
>> a3 = a1 + a2
>> a3 =
>> 5.5000
>> save work20-01-04
>> quit
```

производит следующие действия:

1. Открывает файл d20-01-04.txt.
2. Производит вычисления.
3. Сохраняет переменные в двоичном файле work20-01-04.mat.
4. Сохраняет на диске в подкаталоге work корневого каталога MATLAB журнал работы в файле d20-01-04.txt и закрывает MATLAB.

Посмотрите содержимое файла d20-01-04.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 work20-01-04
quit
```

Запустите снова MATLAB и восстановите значения введенных переменных, которые хранятся в файле work20-01-04.mat, при помощи любого из описанных в начале этого раздела способов: команды `load`, или пункта **Open** меню **File** рабочей среды, или окна **Current Directory**. Изучим возможность просмотра переменных, определенных в рабочей среде.

## Просмотр и удаление переменных, выбор имен переменных

При работе с достаточно большим количеством переменных необходимо знать имена использованных переменных. Самый простой способ — воспользоваться окном **Workspace**, в котором находятся имена всех существующих в рабочей среде переменных. По умолчанию все переменные в списке расположены в алфавитном порядке. Щелчок мышью по заголовку столбца **Name** меняет порядок на обратный. Аналогично можно упорядочивать переменные по размеру занимаемой памяти и типу. Введите кроме вещественных переменных комплексную и проверьте результат упорядочивания по разным критериям.

Двойной щелчок по строке с переменной в окне **Workspace** (или нажатие на кнопку **Open** панели инструментов окна **Workspace**) приводит к отображению ее содержимого в отдельном окне **Array Editor**, что, как следует из его названия (редактор массивов), будет особенно полезно при работе с массивами, которым посвящена глава 2. В окне редактора массивов можно изменять значение элементов массива, добавить новые элементы или удалить существующие. Изменение формата вывода чисел осуществляется через меню **Preferences** в раскрывающемся списке **Numeric Format** так, как было объяснено ранее. Подчеркнем еще раз, что все данные в MATLAB представляются в виде массивов, мы пока работаем с массивами из одной ячейки размера 1 на 1 (работа с массивами большей размерности подробно описана в главе 2).

Вернемся в окно **Workspace**. Кроме сохранения и восстановления рабочей среды, окно **Workspace** позволяет удалять переменные. Для удаления необходимо выделить одну или несколько переменных при помощи мыши и клавиш **<Ctrl>** или **<Shift>**, нажать кнопку **Delete** и подтвердить удаление в появляющемся диалоговом окне **Confirm Delete**.

Аналогичные возможности предоставляют команды и функции MATLAB, предназначенные для оперирования с рабочей средой. Их описание содержится в разделе **Functions -- Categorical List** (раздела **Desktop Tools and Development**, пункт **Workspace, File, and Search Path**) справочной системы. Остановимся на некоторых из них.

Для вывода в командное окно имен используемых переменных служит команда **who**

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

Команда `whos` позволяет получить более подробную информацию о переменных в виде таблицы, аналогичной таблице окна **Workspace**:

```
>> 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
```

Кроме того, команда `whos` дает возможность узнать общий объем памяти, занимаемой всеми переменными рабочей среды. В нашем примере три переменные занимают 24 байта, о чем сказано в строке под таблицей.

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
```

Очень полезной оказывается функция `exist`, которая сообщает, занято ли имя переменной в MATLAB. При вводе новой переменной следует не только убедиться в том, что это имя еще не занято под переменную пользователя, но и быть уверенным, что оно не используется в MATLAB (например, какстроенная функция или зарезервированное слово языка программирования). Вот простой пример. Запишем в переменную `pi` значение синуса от 3:

```
>> pi = sin(3)
pi =
0.1411
```

Теперь посчитайте `cos` от  $\pi$

```
>> z = cos(pi)
z =
0.9901
```

Вычисления выполнены неверно, поскольку изменена системная константа `pi`. До тех пор пока значение `pi` не будет удалено из рабочей среды, воспользоваться константой  $\pi$  не удастся. Поэтому перед определением новой

переменной желательно вызвать функцию `exist`, указав имя переменной в апострофах во входном аргументе, например:

```
>> exist('d7')
ans =
0
```

Если ответ — ноль, то имя этой переменной не конфликтует с зарезервированными словами MATLAB, и ей можно пользоваться. Сравните: `exist('max')`, `exist('for')`, `exist('pi')`, `exist('fzero')`. Значение,озвращаемое функцией `exist`, определяется тем, подо что занято запрашиваемое имя. В частности, если возвращается 1, то такая переменная уже определена в рабочей среде. Смысл всех значений объяснен в справочной системе.

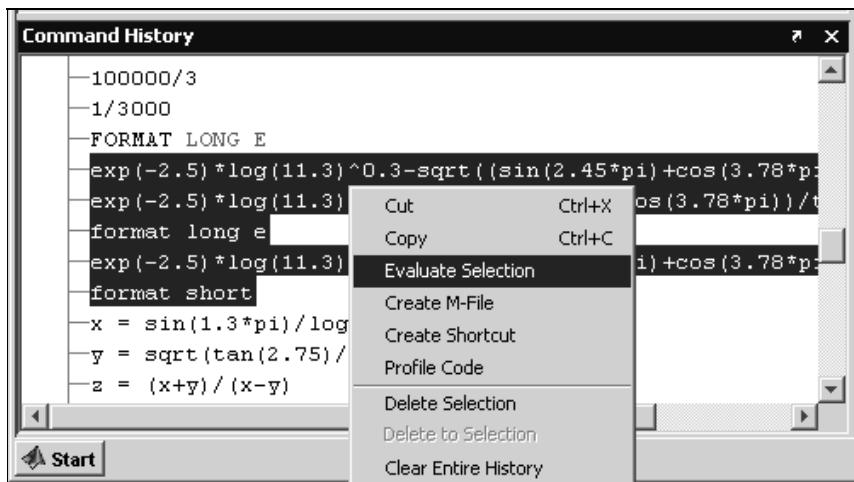
## Эффективная работа из командной строки (*Command History*)

В рабочей среде MATLAB для вызова ранее введенных команд имеется дополнительное удобное средство — окно **Command History** с историей команд (см. рис. 1.1). История команд представляет собой запись команд всех проведенных сеансов работы с MATLAB, которые автоматически сохраняются в текстовом файле `history.m`. В начале протокола работы каждого сеанса отмечены время и дата его начала. Можно отключить запись вводимых команд в историю команд. Для настройки возможностей окна **Command History** выберите пункт **Preferences** в меню **File** рабочей среды. Открывается диалоговое окно **Preferences**. Выбор пункта **Command History** в левой половине окна приводит к отображению свойств окна истории команд (**Command History Preferences**) в правой половине окна **Preferences**. Элементы управления расположены на двух панелях: **Settings** и **Saving**. В случае, если вы предполагаете использовать историю команд предыдущих сеансов работы, есть смысл обратить внимание на два значения переключателя. Одно из них: **Save history file on quit**, т. е. сохранение истории команд перед завершением сеанса работы с пакетом. Следует иметь в виду, что в случае аварийного завершения работы история команд не запишется в файл `history.m` и, следовательно, введенные команды будут недоступны в следующем сеансе работы. Наиболее безопасный способ состоит в установке количества выполненных команд, после которого произойдет обновление файла `history.m` (**Save after ... commands**). Например, значение 1 гарантирует, что ни одна ко-

манды, кроме, быть может, последней не будет потеряна. Смысл остальных флагов и переключателей ясен из их названия.

При загрузке MATLAB окно **Command History** появляется на экране по умолчанию. Чтобы закрыть его, нужно в меню **Desktop** выбрать пункт **Command History** или нажать кнопку закрытия окна в самом окне **Command History**. Повторный выбор пункта **Command History** приводит к появлению одноименного окна в рабочей среде.

История команд делает работу пользователя более эффективной, избавляя от необходимости вновь набирать введенные ранее команды. Если щелкнуть на какой-либо команде в окне левой кнопкой мыши, то данная команда становится текущей. Текущая команда в окне подсвечена синим. Для ее выполнения надо применить двойной щелчок мыши. Аналогичного результата можно добиться в окне **Command History** при помощи клавиш  $\langle\uparrow\rangle$ ,  $\langle\downarrow\rangle$  и  $\langle\text{Enter}\rangle$ . Лишнюю команду можно убрать из окна. Для этого ее надо сделать текущей и удалить при помощи клавиши  $\langle\text{Delete}\rangle$ . Можно выделить несколько идущих подряд команд (рис. 1.8 — выделено пять команд) с использованием комбинации клавиш  $\langle\text{Shift}\rangle+\langle\uparrow\rangle$ ,  $\langle\text{Shift}\rangle+\langle\downarrow\rangle$  и выполнить их при помощи  $\langle\text{Enter}\rangle$ , или удалить клавишей  $\langle\text{Delete}\rangle$ .



**Рис. 1.8.** Окно **Command History**  
с группой выделенных команд и контекстным меню

Выделение последовательно идущих команд можно производить левой кнопкой мыши с одновременным удерживанием клавиши  $\langle\text{Shift}\rangle$ . Если команды не идут одна за другой, то для их выделения следует использовать

левую кнопку мыши с удерживанием клавиши <Ctrl>. Выделенную команду или команды можно перетащить (**Drag and Drop**) в окно **Command Window**, отредактировать (если требуется) и затем выполнить, нажав клавишу <Enter>.

При щелчке правой кнопкой мыши по области окна **Command History** появляется всплывающее меню. Пункт **Cut** служит для удаления выделенной команды или группы команд. Выбор пункта **Copy** приводит к копированию выделенной команды или целой группы в буфер **Windows**. При помощи **Evaluate Selection** можно выполнить отмеченную группу команд. Для удаления текущей команды или группы команд предназначен пункт **Delete Selection**, для удаления всех команд — **Delete to Selection**, для удаления всех команд — **Clear Entire History**. Можно также скопировать или вырезать команды с помощью пунктов **Copy** и **Cut** меню **Edit** рабочей среды. Скопированные команды можно поместить в командное окно, отредактировать их если требуется, а затем запустить, нажав клавишу <Enter>. Пункт **Create M-File** всплывающего меню позволяет создать файл-программу, содержащую выделенные команды (работе с файл-программами посвящена глава 5).

Для запуска группы часто используемых команд удобно создать ярлык (*shortcut*), который будет размещен на отдельной панели рабочей среды. Щелчок мышью по ярлыку приводит к выполнению команд этой группы. Для создания ярлыка с использованием **Command History** следует выделить группу команд и выбрать в контекстном меню пункт **Create Shortcut**. Открывается окно редактора ярлыков, изображенное на рис. 1.9; его поле **Callback** содержит выделенные команды, которые можно отредактировать или добавить новые. Наберите в поле **Label** имя ярлыка, выберите в раскрывающемся списке **Category** значение **Toolbar Shortcut** и нажмите кнопку **Save**. Вы создали ярлык для выполнения связанных с ним команд, который автоматически разместился на панели ярлыков **ShortCuts Toolbar**.

В дальнейшем ярлык и связанные с ним команды можно редактировать, если вызвать контекстное меню для ярлыка, которое показано на рис 1.10. С помощью этого же меню можно создать новый ярлык (**New Shortcut**) без использования окна истории команд. Ярлык можно разместить не только на панели ярлыков, выбрав другие значения поля **Category** (рис. 1.9) при его создании. Доступ ко всем ярлыкам осуществляется также через пункт **Shortcuts** всплывающего меню, которое появляется при нажатии на кнопку **Start** (см. рис. 1.2).

Если требующиеся команды не видны в окне **Command History**, то их можно попытаться найти, используя пункт **Find** в меню **Edit**. Откроется диалоговое окно **Find** (рис. 1.11).

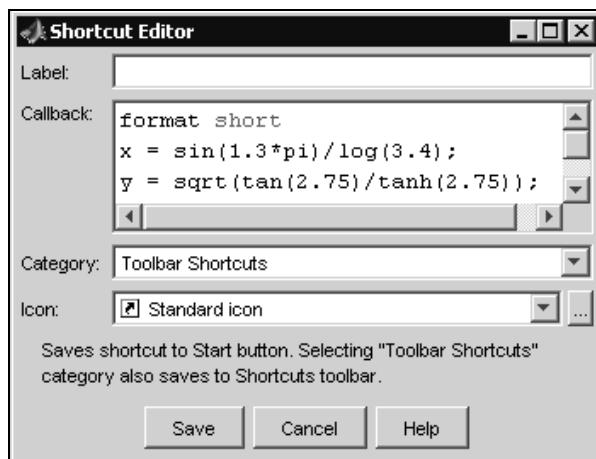


Рис. 1.9. Диалоговое окно Shortcut Editor

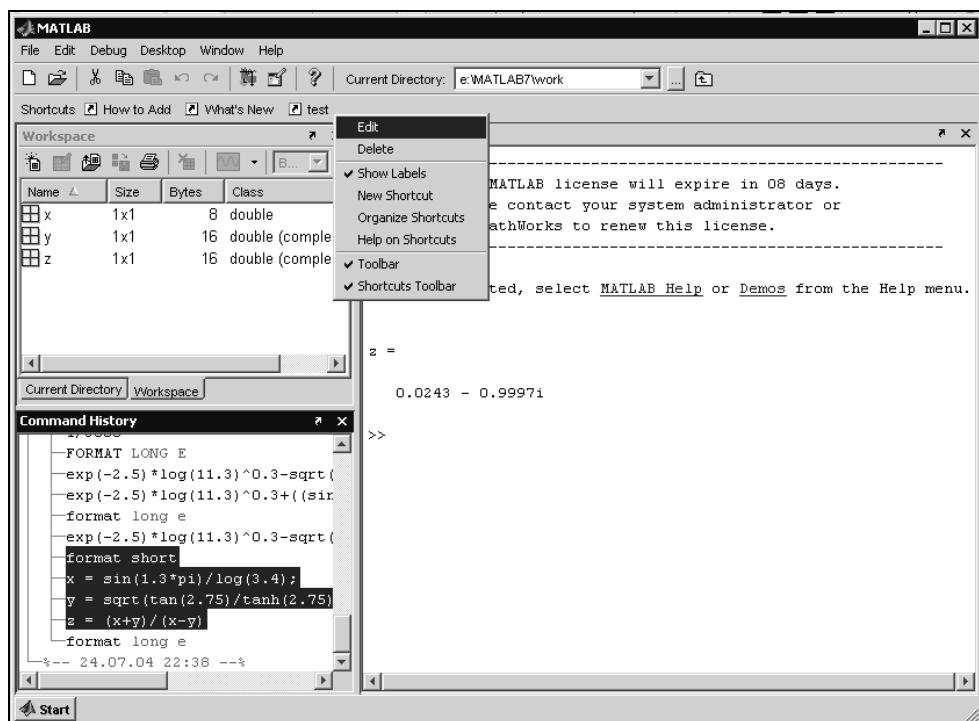


Рис. 1.10. Размещение ярлыка в окне MATLAB и его контекстное меню



**Рис. 1.11.** Диалоговое окно Find окна Command History.

В поле **Find what** диалогового окна **Find** введите требуемую команду или ее часть, задайте направление поиска нажатием кнопок **Previous** (Предшествующая) или **Next** (Следующая) и условия поиска с помощью флагов: **Match case** (Поиск с учетом регистра), **Whole word** (Строгое совпадение), **Wrap around** (При достижении конца файла возобновить поиск с начала файла). Поиск осуществляется от текущей команды.

Умение работать в командной строке MATLAB понадобится вам при чтении всей книги. Поэтому в следующих трех главах все примеры предполагают использование командного окна. В главе 5 объяснено создание собственных программ и функций, которые делают работу в MATLAB более эффективной. Однако практика показывает, что командная строка оказывается полезной даже достаточно опытным пользователям MATLAB, например, для быстрого получения справочной информации, независимой проверки и тестирования отдельных операторов программы. Кроме того, собственные программы и функции вызываются именно из командной строки. При возникновении затруднений обращайтесь снова к этой главе.

## Задания для самостоятельной работы

Задайте переменные  $x$  и  $y$  и вычислите значения арифметических выражений. Выведите результат в различных форматах и сохраните полученные значения в файле.

$$1. \ Z = \operatorname{arctg} \frac{\sqrt[3]{x - \sin y}}{\sqrt{1 - x^2}} - \frac{|x| \sqrt{1 - x^2}}{\sqrt[3]{x - \sin y}}.$$

$$2. \quad T = \frac{(\sin y + \sin 2y + \sin 3y)^4}{1 + \frac{\sin y + \sin 2y + \sin 3y}{1 + e^x}} + \sqrt{1 + \frac{\sin y + \sin 2y + \sin 3y}{1 + e^x}}.$$

$$3. \quad W = \left( 1 + \frac{\ln y}{x + \operatorname{tg} y} \right)^{1 + \frac{x + \operatorname{tg} y}{\ln y}}.$$

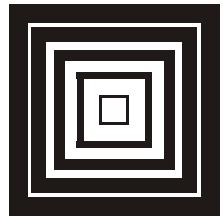
$$4. \quad R = \operatorname{sh} \frac{(x + \ln y)^3}{\sqrt{|x - \ln y|}} \operatorname{ch} \left[ (x + \ln y) \sqrt{|x - \ln y|} \right].$$

$$5. \quad H = \frac{\sqrt{\cos 2y + \sin 4y + \sqrt{e^x + e^{-x}}}}{(e^x + e^{-x})^3 (\cos 2y + \sin 4y - 2)^2}.$$

$$6. \quad A = \sqrt[5]{x(1+x)^2(1+2x)^3} + \sqrt[3]{\frac{x(1+x)^2(1+2x)^3}{\ln|\operatorname{ctg} y|}}.$$

$$7. \quad S = \operatorname{arctg} \sqrt{\left| \frac{x - \sin y}{x + \sin y} + \frac{x + \sin y}{x - \sin y} \right|} + e^{(x - \sin y)(x + \sin y)}.$$

$$8. \quad B = \frac{1 + \arcsin(\cos 2y)}{2^x + 3^{-x}} + \left( \frac{2^x + 3^{-x} - 1}{1 + \arcsin(\cos 2y)} \right)^2.$$



## 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, статистики, обработки данных 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, поскольку 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 и матрица, 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 случаях, когда важен способ хранения вектора в MATLAB. Векторы и матрицы обозначаются курсивом, а соответствующие

им массивы прямым шрифтом "Courier", 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

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

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

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

#### Примечание

В главе 1 было сказано, что точка с запятой в конце выражения используется для подавления вывода результата на экран. Оказывается, что этот символ предназначен и для разделения элементов вектор-столбцов.

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
```

Просмотрите появившуюся в окне **Workspace** информацию. Векторы  $a$ ,  $b$  и  $c$  хранятся в двумерных массивах. В столбце **Size** указано  $3 \times 1$ , т. е. массивы имеют по три строки и одному столбцу, каждый из них занимает 24 байта памяти (столбец **Bytes**).

Аналогичные сведения можно получить при помощи встроенных функций `ndims` и `size`:

```
>> na = ndims(a)
na =
2
>> sa = size(a)
sa =
3 1
```

На первый взгляд может показаться, что `ndims` и `size` лишь дублируют информацию окна **Workspace**. Однако они позволяют не только вывести характеристики массивов в командное окно, но и записать их в переменные для дальнейшего использования, что существенно при создании собственных приложений в MATLAB.

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.

### Примечание

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 в отличие от вектор-столбца 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 окно **Workspace** или функции `ndims`, `size` и 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 размernости один на четыре. Для определения длины вектор-столбцов или вектор-строк служит встроенная функция `length`:

```
>> L = length(s1)
```

```
L =
```

```
4
```

По умолчанию все числа (элементы массивов) хранятся с двойной точностью (`double`) и занимают 8 байтов. Большие массивы требуют для хранения значительных объемов памяти. Для уменьшения объема занимаемой массивами памяти можно применять другие способы хранения элементов массива: `single` для вещественных чисел, требующих для размещения 4 байта, и `int8`, `int16`, `int32` — для целых чисел, занимающих 1, 2 или 4 байта соответственно. Использование таких данных значительно экономит память и не влияет на функциональные возможности пакета MATLAB.

Для изменения точности представления чисел предназначены одноименные с типом (**Class**) данных функций `single`, `int8`, `int16`, `int32`:

```
>> q4 = single(s4);
>> q3 = int32(s3);
>> q2 = int16(s2);
>> q1 = int8(s1);
```

Если теперь посмотреть распределение памяти под массивы, то легко заметить существенную разницу в отведенной для хранения памяти:

```
>> whos
```

| Name | Size | Bytes | Class        |
|------|------|-------|--------------|
| q1   | 1x4  | 4     | int8 array   |
| q2   | 1x4  | 8     | int16 array  |
| q3   | 1x4  | 16    | int32 array  |
| q4   | 1x4  | 16    | single array |
| s1   | 1x4  | 32    | double array |
| s2   | 1x4  | 32    | double array |
| s3   | 1x4  | 32    | double array |
| s4   | 1x4  | 32    | double array |

Для массивов большой размерности экономия памяти может быть существенной. Выполнение арифметических операций с вещественными числами разного типа допустимо и дает результат с наименьшей точностью — `single`:

```
>> qs44 = q4 + s4;
```

```
>> whos qs44
```

| Name | Size | Bytes | Class        |
|------|------|-------|--------------|
| qs44 | 1x4  | 16    | single array |

Попытка выполнить арифметические операции над целыми числами разных типов приводит к ошибке:

```
>> q31 = q3 + q1
??? Error using ==> plus
Integers can only be combined with integers of the same class, or
scalar doubles.
```

Более того, при преобразовании целого числа, когда для точного представления не хватает отводимой памяти, получается максимальное число для данного типа (т. е. неверный результат) без предупреждения об этом:

```
>> w1 = 34567;
>> r1 = int8(w1)
r1 =
 127
>> t1 = int16(w1)
t1 =
 32767
```

Поэтому применение целых чисел, имеющих меньшую точность представления, ограничено. Вернемся к работе с векторами и матрицами, используя принятое по умолчанию представление чисел `double`.

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
```

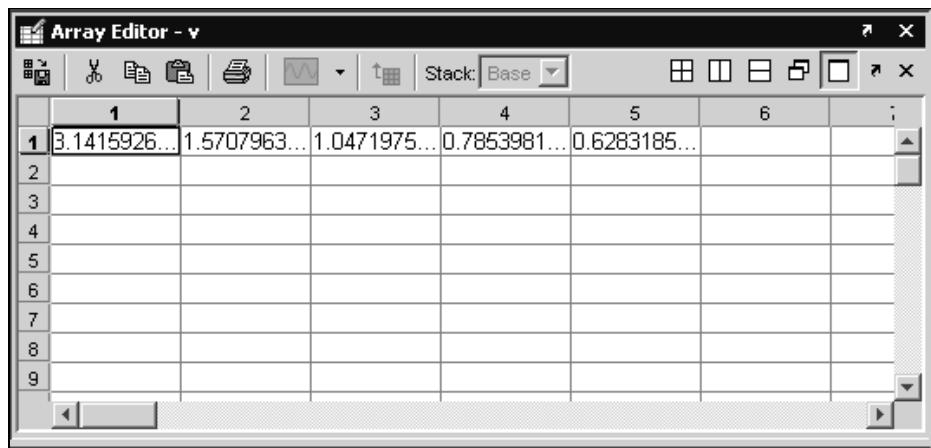
Для сцепления вектор-строк также применяются квадратные скобки, но сцепляемые вектор-строки отделяются пробелами или запятыми:

```
>> v1 = [pi pi/2];
>> v2 = [pi/3 pi/4 pi/5];
```

```
>> v = [v1 v2]
v =
3.1416 1.5708 1.0472 0.7854 0.6283
```

Запоминать правила сцепления вектор-строк и вектор-столбцов не требуется, достаточно посмотреть на вектор-строку *v* как на блочную строку. Элементы этой блочной строки, т. е. вектор-строки *v1* и *v2*, разделены пробелом в соответствии с правилом набора вектор-строк. Аналогичное рассуждение верно и для сцепления вектор-столбцов.

Для просмотра и изменения значений элементов массивов удобно использовать редактор массивов (**Array Editor**). Дважды щелкните мышью по имени массива *v* в окне **Workspace** или нажмите кнопку **Open Selection** на панели инструментов окна **Workspace** при положении курсора на имени массива — откроется окно редактора массивов с содержимым вектор-строки *v* (рис. 2.1). Имя просматриваемого массива указано в строке заголовка окна.



**Рис. 2.1.** Окно редактора массивов **Array Editor**

Двойной щелчок мышью по ячейке или нажатие клавиши <F2> (как и в таблицах MS Excel) позволяет редактировать содержимое соответствующего элемента массива. Убедитесь в этом, изменяя значения элементов и выводя массив в командное окно. При заполнении пустых клеток изменяются размеры массива так, что он остается прямоугольным наименьшей размерности и содержит все определенные элементы. При этом не определенные пользователем элементы приобретают нулевое значение. Для уменьшения размеров следует использовать элемент управления **Delete** (в контекстном

или оконном меню или на панели инструментов), а не **Cut**, который приводит лишь к обнулению элементов массива.

Для работы с данными редактор массивов MATLAB предоставляет возможности, аналогичные MS Excel. Увеличьте длину вектор-строки *v* до 10, для чего выделите первые пять ее элементов, скопируйте их при помощи всплывающего меню (или кнопки **Copy** панели инструментов, или одноименного пункта меню **Edit**), сделайте текущей шестую ячейку и используйте вставку (**Paste**) для дублирования первых пяти элементов вектор-строки. Проверьте при помощи командного окна изменился ли вектор *v*.

Редактор массивов позволяет просматривать значения нескольких переменных. Двойной щелчок по строке с *v1* в окне **Workspace** приводит к появлению нового окна в редакторе массивов. Каждое из окон можно сделать активным при помощи соответствующей вкладки внизу окна редактора или выбрав его название в меню **Window**. Кнопки в правой части панели инструментов соответствуют различным способам расположения таблиц с содержимым массивов в окне редактора **Array Editor**. Также редактор массивов оказывается очень полезным при отладке собственных программ (отладке программ посвящен разд. "Диалоговая отладка программ" главы 8).

## 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 обращения, например, к четвертому элементу используется *индексация*:

```
>> h = v(4);
```

```
h =
```

```
8.2000
```

Указание 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*. Например, запись в вектор-строку *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 заданной вектор-строке *w* 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 =
```

```
0.1000 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 вектор-строку *w2*, содержащую 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
```

Вместо `w(5:7)` можно написать `w(5:end)`. Такая запись означает, что берутся элементы, начиная с пятого и заканчивая последним. Если же ввести просто `w(end)`, то получим последний элемент массива:

```
>> w(end)
ans =
9.8000
```

Еще один способ индексирования — логическое индексирование — описан в разд. *"Логическое индексирование" данной главы.*

Элементы массива могут входить в выражения. Вычисление, например, среднего геометрического из элементов вектора `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 вектор-столбца или вектор-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 геометрическое 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
```

При обращении к функции `min` с двумя векторами в качестве входных аргументов получится вектор, каждый элемент которого есть минимум из двух элементов исходных векторов с одинаковыми номерами.

```
>> p = [3 12 8];
```

```
>> s = [4 10 7];
```

```
>> min(p, s)
```

```
ans =
```

```
3 10 7
```

Часто необходимо знать не только значение минимального или максимального элемента в массиве, но и его индекс (порядковый номер). Вы уже видели, что интерфейс функций MATLAB достаточно универсален — большинство из них допускают обращение к ним с переменным числом входных и выходных аргументов.

Вызовите 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 быстрого получения подсказки следует набрать в командной строке `help` и через пробел имя функции. MATLAB выведет в командное окно всевозможные способы обращения к функции с дополнительными пояснениями. Аналогичные сведения можно найти в интерактивной справочной системе, которой вы уже пользовались при чтении *главы 1*. Если известно имя функции, то проще всего воспользоваться индексным поиском (вкладка **Index** в левой части окна справочной системы MATLAB). В строке ввода **Search index for:** следует набрать имя функции, например, `min`, затем в левом окне выбрать раздел **MATLAB** и ознакомиться с содержимым правого окна. Информация обо всех функциях MATLAB для работы с векторными и матричными данными содержится в пункте **Arrays and Matrices** подраздела **Mathematics** раздела **Functions -- Categorical List** (вкладка **Contents**).

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 входят: 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` или обратившись к пункту **Data Analysis and Fourier Transforms** подраздела **Mathematics** раздела **Functions -- Categorical List** интерактивной справочной системы MATLAB. В последующих разделах описано применение функций обработки данных к матричным данным.

### Примечание

Дополнительные функции содержатся в специализированных Toolbox. Команда `help stats` выводит в командное окно список статистических функций, доступных в MATLAB, если установка MATLAB включает Statistics Toolbox. Более подробную информацию можно почерпнуть из раздела Statistics Toolbox интерактивной справочной системы.

## 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, в частности, для определения собственных функций и построения их графиков.

Ведите две вектор-строки:

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

Операция `.*` (не вставляете 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 `.`

```
>> 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 `.\`

```
>> 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-строку приводит к сообщению об ошибке:

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

### Примечание

При делении числа на вектор-столбец сообщение об ошибке не выдается. Это связано с тем, что в данном случае происходит решение системы линейных уравнений с прямоугольной матрицей, в которой число неизвестных превосходит число уравнений. Решение систем линейных уравнений разобрано в главе 6.

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:

$$u = \begin{bmatrix} 2+3i \\ 1-2i \\ 3+2i \end{bmatrix}, u^* = [2-3i \ 1+2i \ 3-2i], u^T = [2+3i \ 1-2i \ 3+2i].$$

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. Следующие два оператора приводят к одинаковым вектор-строкам:

```
>> 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 начальное значение должно быть больше или равно конечному для получения непустой вектор-строки.

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 набрать знак апострофа вместо точки с апострофом. Круглые скобки использованы не случайно, без них операция транспонирования применилась бы только к числу 0.5, и в результате вектор *x* был бы строкой.

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

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

### Примечание

Выше мы рассмотрели индексацию при помощи двоеточия. Очевидно, что при этом создается вектор индексов с постоянным шагом, который используется для выделения нужных элементов вектора.

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 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).

### Примечание

Количество элементов, выводимых в одну строку, определяется текущими размерами окна и форматом вывода данных.

Одним из способов получения таблицы является формирование матрицы из двух столбцов, первый из которых содержит значения абсцисс, а второй — ординат:

```
>> [x' y']
ans =
0 0
0.0500 0.4560
0.1000 0.7614
.
.
```

Далее в этой главе мы уделим достаточно внимания матрицам, а пока обратимся к простейшим способам графического представления функций, которое часто является более наглядным и удобным по сравнению с таблицей ее значений.

## Построение графиков функции одной переменной

MATLAB обладает хорошо развитыми графическими возможностями для визуализации данных. Графике в MATLAB посвящена глава 3. 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 не нужно, 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 1** с графиком функции, изображенное на рис. 2.2. Окно содержит меню, панель инструментов и область графика. В главе 3, посвященной графике в MATLAB, описаны команды, предназначенные для оформления графика. Сейчас нас будет интересовать сам принцип построения графиков и некоторые простейшие возможности визуализации функций.

Для построения графика функции в рабочей среде MATLAB должны быть определены два вектора одинаковой размерности, например  $x$  и  $y$ . Вектор  $x$  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 командное окно MATLAB и окно с графиком так, чтобы они не перекрывались. Например, используя кнопку **Dock Figure** (справа в строке меню окна), можно встроить графическое окно в рабочую среду так, как показано на рис. 2.3.

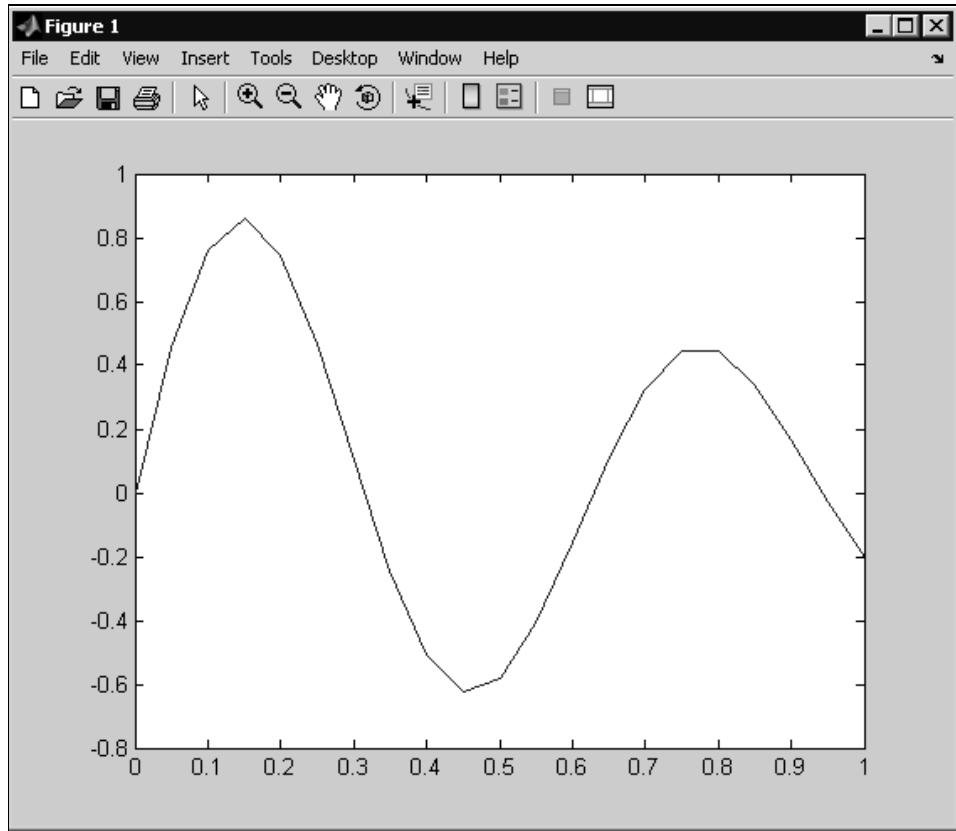


Рис. 2.2. Простейший график функции

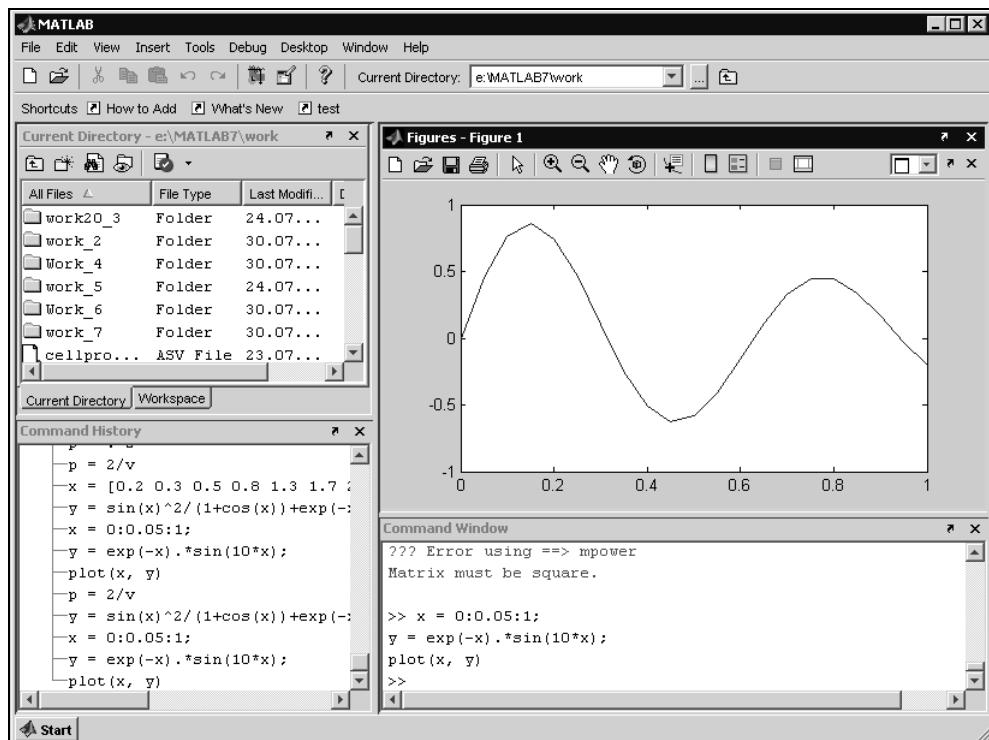
### Примечание

Для получения отдельного графического окна следует нажать на кнопку **Undock Figure**.

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, затем отредактировать их и выполнить, нажав  $<\text{Enter}>$ . 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. 2.4 (далее в книге почти всюду приводится только область графика, без заголовка окна, меню и панели инструментов).



**Рис. 2.3.** Расположение окна с графиком в окне MATLAB

### Примечание

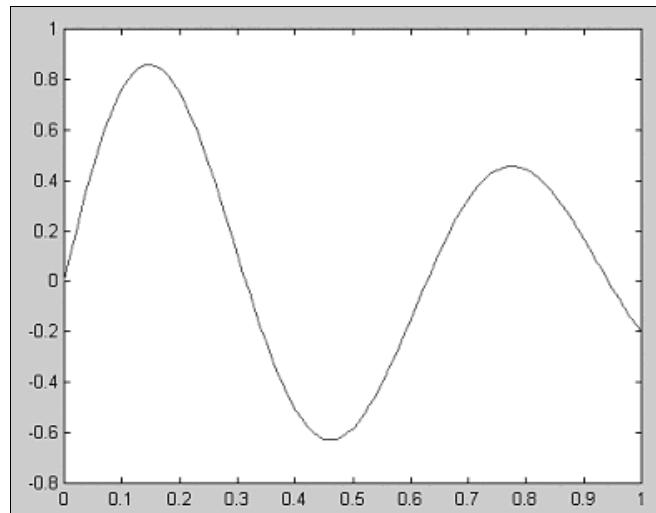
Построение графика можно осуществить с помощью инструментов окна **Workspace**. Эта возможность рассматривается в следующей главе.

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

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

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. 2.4.** Гладкий график функции

Получившиеся графики, приведенные на рис. 2.5, 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. 2.6). 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 закрытия окна в его правом верхнем углу.

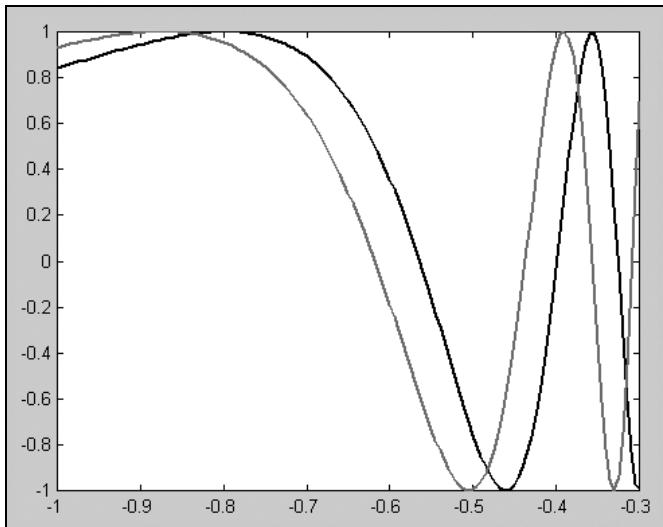


Рис. 2.5. Два графика на одних осях

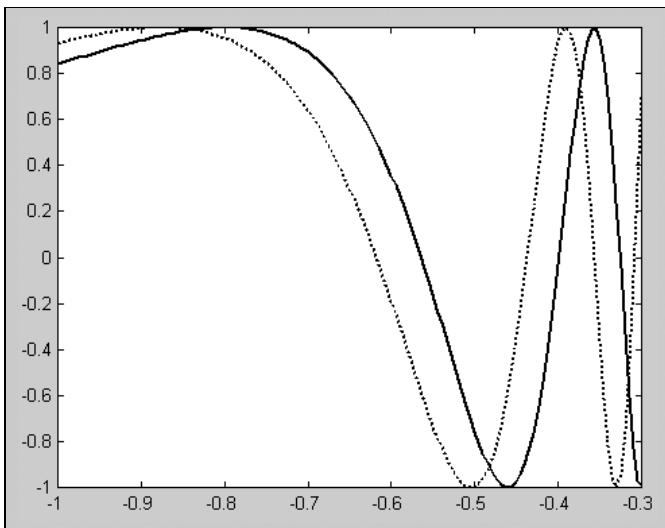


Рис. 2.6. Изменение стиля и цвета линий графиков

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 были разобраны выше. Две следующие главы посвящены более детальному использованию графических средств пакета. Перейдем теперь к умножению векторов.

## 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

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

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
```

Скалярное произведение векторов можно также вычислить, применив функцию MATLAB `dot`

```
>> s = dot(a,b);
```

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

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

**Решение:**

```
>> d = sqrt(dot(a, a))
d =
3.4886
```

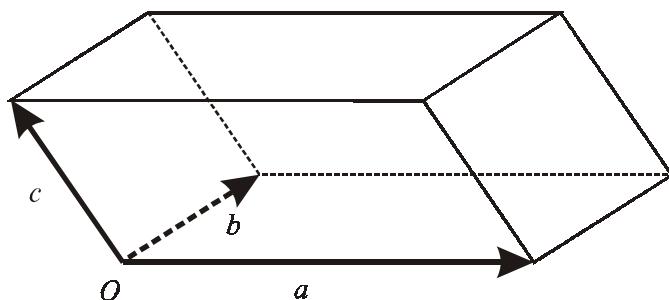
## Векторное произведение

*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. 2.7.



**Рис. 2.7.** Параллелепипед, образованный тремя векторами

Найдите объем параллелепипеда, если

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

Правильные действия таковы:

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

## Внешнее произведение

Внешним произведением векторов  $a = (a_j)_{j=1, \dots, N}$ ,  $b = (b_k)_{k=1, \dots, M}$  называется матрица  $C = (c_{jk})_{j=1, \dots, N, k=1, \dots, M}$  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  $C = ab^T$ , 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 или окно **Workspase**. Числа, векторы и матрицы хранятся в двумерных мас-

сивах 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 приспособливаются 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 нам понадобится еще несколько матриц. 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
 2 7 0
 -5 1 2]
```

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:

```
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 в окне **Workspace** или наберите в командной строке `whos`. 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
```

В качестве индексов могут выступать векторы, содержащие номера нужных строк и столбцов. Например, для выделения элементов первой и второй строк второго и третьего столбцов введенной выше матрицы  $B$  достаточно ввести команды:

```
>> i = [1 2];
>> j = [2 3];
>> B1 = B(i, j)
B1 =
3 -1
7 0
```

Расположение элементов матрицы в памяти компьютера определяет еще один способ обращения к ним. MATLAB хранит элементы матрицы в памяти по столбцам. Элементы  $q_{ij}$  матрицы  $Q$  размера  $m$  на  $n$  содержатся в памяти в последовательности:

$$q_{11}, q_{21}, \dots, q_{m1}, q_{12}, q_{22}, \dots, q_{m2}, \dots, q_{1n}, q_{2n} \dots, q_{mn}.$$

Следовательно, для доступа к элементам матрицы можно использовать *один индекс*, задающий порядковый номер элемента матрицы в векторе. Например, элементы матрицы  $C$ , определенной в предыдущем разделе, записаны в таком порядке

$C(1, 1), C(2, 1), C(1, 2), C(2, 2), C(1, 3), C(2, 3)$

Поэтому обращение к элементам матрицы как к элементам вектора при помощи одного индекса (индексация при помощи порядкового номера) приводит к предсказуемому результату

```
>> C(5)
ans =
7
```

но может послужить и источником ошибок в вычислениях, если один индекс случайно указан вместо двух — никакого предупреждения не выводится. Как правило, часто лучше использовать два индекса, за исключением некоторых специальных случаев, например, прохода элементов матрицы по столбцам.

Еще одним способом обращения сразу ко всем элементам матрицы, удовлетворяющим некоторому условию, является *логическое индексирование*. Логи-

ческое индексирование не является необходимым для выполнения обычных операций матричной алгебры, поэтому при первом чтении следующий раздел можно пропустить. Однако оно существенно расширяет возможности обработки векторных и матричных данных, позволяя наглядно и компактно записывать выражения с достаточно сложной логикой без программирования перебора данных.

## Логическое индексирование

Логическое индексирование (logical subscripting) позволяет выбрать из массива элементы, удовлетворяющие определенным условиям, которые заданы логическим выражением. Подробно логические выражения будут обсуждаться в главе 7, а здесь приведем простой пример. Пусть из введенной выше матрицы  $B$  требуется выбрать все отрицательные элементы и записать их в вектор  $f$ . Сначала выполним, казалось бы, недопустимое действие: запишем в переменную `ind` результат сравнения матрицы и числа ноль.

```
>> ind = B < 0
ind =
0 0 1
0 0 0
1 0 0
```

Образовался логический массив (logical array) `ind` того же размера, что и `B` (см. окно **Workspace**), состоящий из нулей и единиц, причем единицы соответствуют отрицательным элементам массива `B`. Указание логического массива `ind` в качестве единственного индекса исходного массива `B` позволяет решить поставленную задачу:

```
>> f = B(ind)
f =
-5
-1
```

Разумеется, можно было обойтись и без вспомогательного массива `ind`, написав сразу `f = B(B < 0)`.

Если требуется присвоить новое значение элементам массива, удовлетворяющим определенному условию, то выражение `B(B < 0)` должно войти в левую часть оператора присваивания.

Только что мы рассмотрели новый тип данных MATLAB — логические массивы. В нашем примере логический массив `ind` был автоматически создан при выполнении операции сравнения `B > 0`. Было бы ошибкой считать,

что для выделения нужных элементов достаточно просто создать массив из нулей и единиц и указать его в качестве индекса массива. Введите, например, из командной строки массив `ind1` с теми же элементами, что и `ind`:

```
>> ind1 = [0 0 1; 0 0 0; 1 0 0];
```

и попытайтесь использовать его для логического индексирования, получается ошибка:

```
>> B(ind1)
```

```
??? Subscript indices must either be real positive integers or logicals.
```

Выход состоит в преобразовании числового массива `ind1` в логический массив `ind2` при помощи функции `logical`, который затем применяется для индексирования:

```
>> ind2 = logical(ind1);
```

Убедитесь, что `ind2` — логический массив, изучив информацию о нем в окне **Workspace**. Логическое индексирование векторов производится аналогично.

Логическое индексирование позволяет получить значения нужных элементов матрицы или вектора или изменить их, но индексы элементов остаются неизвестными. Для поиска индексов элементов, удовлетворяющих определенному условию, служит функция `find`. Вот простой пример: требуется найти номера всех элементов вектора, равных максимальному значению. Вызов функции `max` с двумя выходными аргументами не решает эту задачу, поскольку находится только один элемент и его номер:

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

```
>> [m, k] = max(x)
```

```
m =
```

```
5
```

```
k =
```

```
3
```

Теперь мы знаем значение `m` максимального элемента, оно равно 5, и могли бы использовать логическое индексирование для записи всех максимальных значений в векторе. Двойной знак == обозначает логическое равенство:

```
>> xm = x(x == 5)
```

```
xm =
```

```
5 5 5
```

но номера максимальных элементов все равно неизвестны. Вместо логического индексирования используем функцию `find`, указав во входном ее аргументе логическое выражение `x == 5`

```
>> km = find(x == 5)
```

```
km =
```

```
3 6 8
```

Функция `find` вернула номера элементов вектора, совпадающих с максимальным значением.

Аналогичный поиск в матрице так же осуществляется при помощи `find`. Пусть, например, надо найти индексы всех неположительных элементов матрицы  $B$ , введенной выше. Вызовем `find` с двумя выходными аргументами — векторами, в которые требуется записать значения строчных и столбцевых индексов искомых элементов матрицы

```
>> [i, j] = find(B <= 0)
```

```
i =
```

```
3
```

```
1
```

```
2
```

```
j =
```

```
1
```

```
3
```

```
3
```

Действительно, все эти элементы  $b_{31}$ ,  $b_{13}$  и  $b_{23}$  меньше или равны нулю. В этом примере к функции `find` можно обратиться и с одним выходным аргументом:

```
>> k = find(B <= 0)
```

```
k =
```

```
3
```

```
7
```

```
8
```

В этом случае вектор `k` содержит номера требуемых элементов матрицы с учетом описанной выше схемы хранения в памяти по столбцам.

Обратимся теперь к основным операциям с матрицами в MATLAB.

## Сложение, вычитание, умножение, транспонирование и возвведение 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
-1 0 2
```

```
3 7 1
-1 0 2
```

Сопряжение и транспонирование матриц, содержащих комплексные числа, приведут к разным матрицам:

```
>> 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 на саму себя.

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

$$(A+C)B^3(A-C)^T.$$

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 не определена операция деления для 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:

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

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

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

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

### Примечание

Алгоритм решения систем линейных уравнений при помощи оператора \ определяется структурой матрицы коэффициентов системы. В частности, MATLAB исследует, является ли матрица треугольной, или может быть приведена перестановками строк и столбцов к треугольному виду, симметрична 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 следует использовать специальные методы (подробнее о решении систем линейных уравнений сказано в главе 6).

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. Для ввода данных можно воспользоваться редактором массивов, который предоставляет удобный способ ввода и дает возможность легко проверить введенные данные на наличие ошибок. Вот один из возможных способов. Создайте в рабочей среде два пустых массива размера ноль на ноль при помощи квадратных скобок:

```
>> A = [];
>> b = [];
```

Откройте в редакторе массив **A** и определите нужный размер в строках ввода **Size**: Затем введите элементы матрицы в ячейки таблицы. Обратите внимание, что при нажатии <Enter> происходит переход к ячейке, расположенной под текущей. Это хорошо, если матрица вводится по столбцам. Если предпочтительнее заносить значения элементов по строкам, то следует выбрать пункт **Preferences...** в меню **File** редактора массивов и в появившемся окне **Array Editor Preferences** задать желаемое направление перехода в раскрывающемся списке **Direction**. Флаг **Move selection after Enter** должен быть включен. Аналогичным способом вводится вектор правой части системы.

Матрица и вектор правой части системы могут храниться 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. 2.8.

| 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. 2.8. Файлы с матрицей и вектором правой части системы

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. Применение load и save для сохранения и считывания переменных рабочей среды описано в главе 1, однако при работе с файлами данных используется функциональный способ вызова этих команд с выходными аргументами:

```
>> A = load('matr.txt');
>> b = load('rside.txt');
>> x = A\b;
>> save 'sol.txt' x -ascii
```

Параметр `-ascii` означает запись в текстовом формате. После выполнения данных команд в каталоге work создается 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. 2.9 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. 2.9. 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 'matrA.txt' A -ascii`. Запись в файл и считывание из файла осуществляются по строкам.

## 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

Ведите матрицы

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

и создайте из них блочную матрицу

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

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 пример для проверки знаний о работе с массивами в MATLAB. Требуется составить блочную матрицу

$$M = \begin{pmatrix} S & a \\ b & 2.5 \end{pmatrix},$$

где

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

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  $M = [[S; b] [a; 2.5]]$ .

MATLAB позволяет конструировать блочно-диагональные матрицы с помощью функции `blkdiag`. Рассмотрим пример:

```
>> R = [1 2; 3 4];
>> Q = [5 6 7; 8 9 10; 11 12 13];
>> T = [-3 5; 6 7];
Z = blkdiag(R, Q, T)

Z =
1 2 0 0 0 0 0
3 4 0 0 0 0 0
0 0 5 6 7 0 0
0 0 8 9 10 0 0
0 0 11 12 13 0 0
0 0 0 0 0 -3 5
0 0 0 0 0 6 7
```

Блоки, используемые функцией `blkdiag`, не обязательно должны быть квадратными и одного размера.

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

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

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 (т. е. 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 end:

```
>> p = P(2, 2:end)
```

```
p =
```

```
10 12 5
```

## Удаление строк и столбцов

Как уже было сказано, 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 посмотреть в окне **Workspace** или проверить при помощи `size`. Аналогичным образом удаляются и столбцы. Для удаления нескольких идущих подряд столбцов (или строк) им нужно присвоить пустой массив. Удалите второй и третий столбец в `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 считывание из файла и использование редактора массивов. Однако если матрица обладает простой структурой, то бывает проще сгенерировать ее, чем вводить. 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(1, :) = 1 >> A(end, 3:end) = -1
A = A = A =
0 0 0 0 0 1 1 1 1 1 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 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
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
>> 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
```

Обращение к функциям `rand` и `randn` с одним входным аргументом приводит к формированию квадратных матриц.

Вопрос об автоматическом заполнении вектор-столбцов или вектор-строк не должен поставить нас в тупик, поскольку мы знаем, что вектор-столбец 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 вектор-столбца 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
```

Все функции MATLAB для создания матриц специального вида описаны в справочной системе, см. пункт **Arrays and Matrices** подраздела **Mathematics** раздела **Functions -- Categorical List**.

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

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

Матрицу  $M$  можно представить в виде блочной матрицы из четырех квадратных блоков размера пять, диагональные блоки формируются при помощи `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  $\tau_k$  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).

Матрицы больших размеров удобно представлять в наглядном виде. В программе 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

```
>> G = 2*eye(7) + diag(ones(1, 6), 1) + diag(ones(1, 6), -1)
```

После выполнения команды `spy` на экране появляется графическое окно **Figure 1**. На рис. 2.10 изображена часть окна без заголовка, меню и панели инструментов.

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 квадрата, цвет которого соответствует величине элемента. Для того чтобы узнать соответствие цвета и величины элемента, следует использовать команду `colorbar`, выводящую 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.

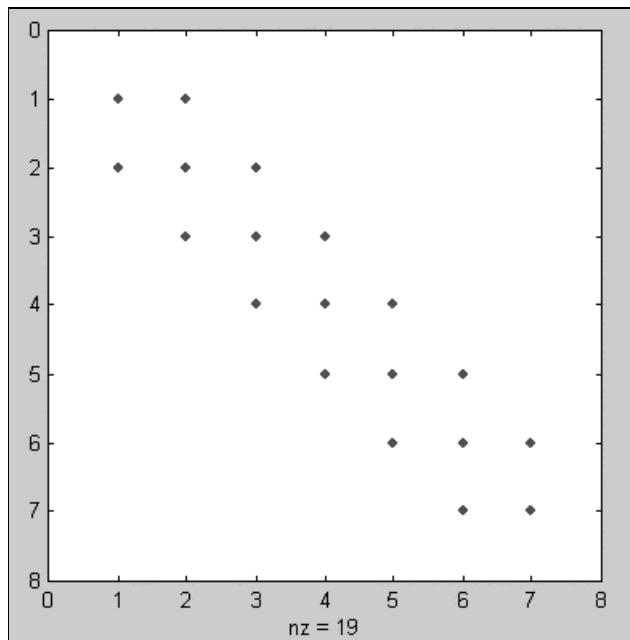


Рис. 2.10. Шаблон разреженной матрицы

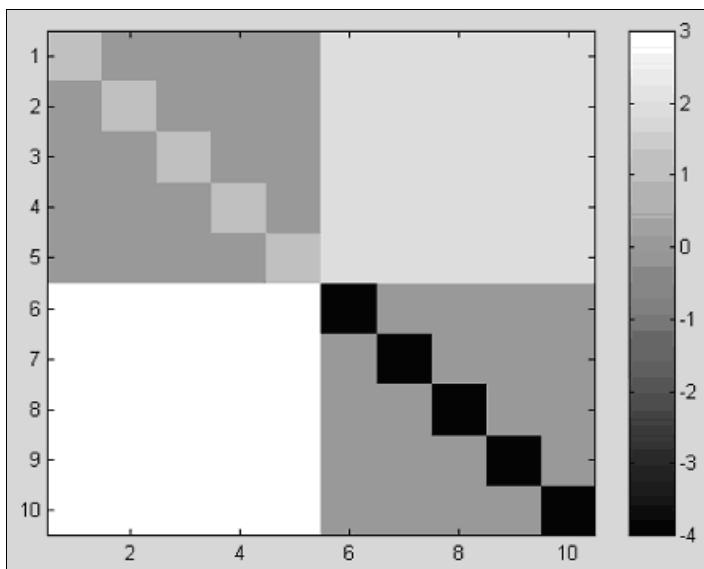


Рис. 2.11. Интерпретация матрицы как изображения

Набирайте команды, указанные ниже, и следите за состоянием графического окна:

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

В результате получается наглядное представление матрицы, приведенное на рис. 2.11.

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

Ведите две матрицы

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

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  $.^{\wedge}$

```
>> 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 *
```

```
0.000500000000000 0.025000000000000 0.001000000000000
```

```
2.187000000000000 0.00001562500000 0.00000001693509
```

Заметьте, что повторного вычисления элементов матрицы  $PB$  не потребовалось. Вне зависимости от установленного формата вывода все вычисления производятся с двойной точностью (в предположении, что данные типа `Single` не участвуют в операциях). 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, векторы 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 `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`:

|                                 |                                     |
|---------------------------------|-------------------------------------|
| <pre>&gt;&gt; p = prod(M)</pre> | <pre>&gt;&gt; p2 = prod(M, 2)</pre> |
| <code>p =</code>                | <code>p2 =</code>                   |
| 6 -24 0                         | 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:

|                                  |                                     |
|----------------------------------|-------------------------------------|
| <pre>&gt;&gt; MC = sort(M)</pre> | <pre>&gt;&gt; MR = sort(M, 2)</pre> |
| <code>MC =</code>                | <code>MR =</code>                   |
| 1 -6 -4                          | -4 -2 1                             |

|        |        |
|--------|--------|
| 2 -2 0 | -6 3 4 |
| 3 -2 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) >> mn = min(M)
mx = mn =
3 -2 4 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) >> [mn, n] = min(M)
mx = mn =
3 -2 4 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 `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
```

Рассмотрим функцию `rot90`, поворачивающую массив на 90 градусов против часовой стрелки. Примените эту функцию к упоминавшемуся массиву `p`:

```
>> q1 = rot90(p)
q1 =
-1 7
```

```
3 0
4 2
```

Примените функцию повторно, теперь уже к массиву q1.

```
>> q2 = rot90(q1)
q2 =
```

```
7 0 2
-1 3 4
```

Функция `fliplr` обеспечивает зеркальное отражение от условной вертикали, проходящей через середину массива:

```
>> fliplr(q2)
ans =
2 0 7
4 3 -1
```

Обратите внимание на то, что вектор-столбец не изменяется под воздействием функции `fliplr`:

```
>> q3 = [1; 2; 3; 4; 5];
>> fliplr(q3)
ans =
1
2
3
4
5
```

Более подробно про обработку матричных данных можно узнать из приложения или вывести список всех встроенных функций обработки данных командой `help datafun`, а затем посмотреть информацию о нужной функции, например, `help max`, или обратиться к интерактивной справочной системе.

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. 2.12, и вычислить значения функций в узлах, обозначенных точками.

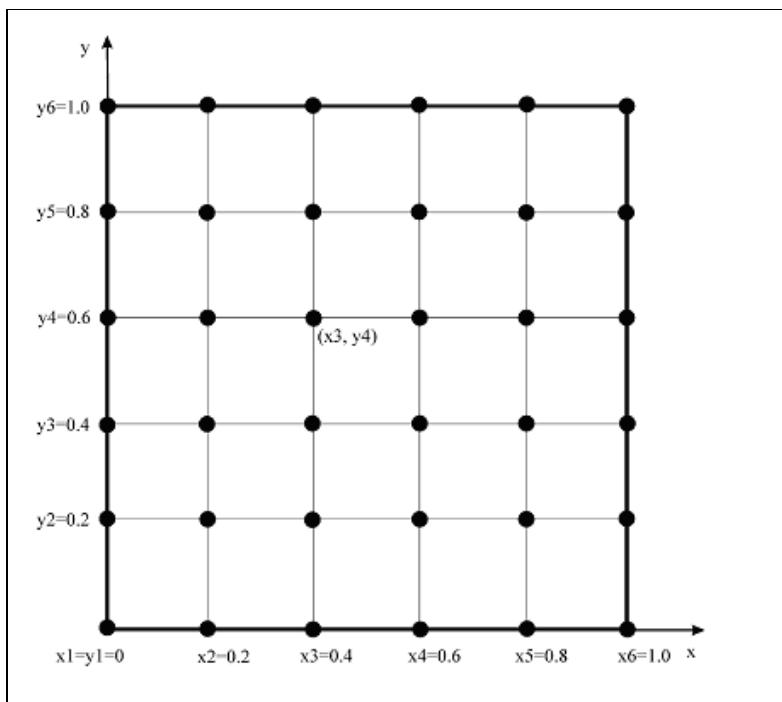


Рис. 2.12. Прямоугольная область построения графика

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  $x_1, x_2, \dots, x_6$ , а  $Y$  содержит одинаковые столбцы с  $y_1, y_2, \dots, y_6$ . Значения функции в узлах сетки запишем в матрицу  $Z$  такой же размерности ( $6 \times 6$ ), причем для вычисления матрицы  $Z$  используем выражение для функции, но с *поэлементными* матричными операциями. Тогда, например,  $Z(3, 4)$  как раз будет равно зна-

чению функции  $z(x, y)$  в точке  $(x_3, y_4)$ . 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, изображенным на рис. 2.13 (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)
```

График функции, изображенный на рис. 2.13, 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. 2.14.

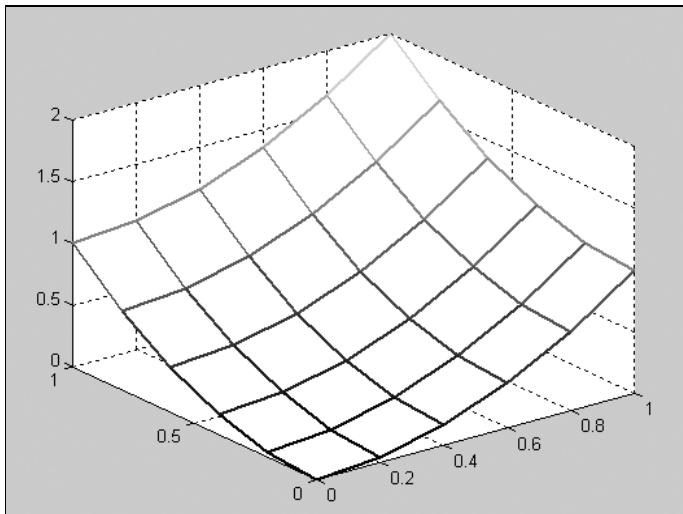


Рис. 2.13. График функции  $z(x, y)$

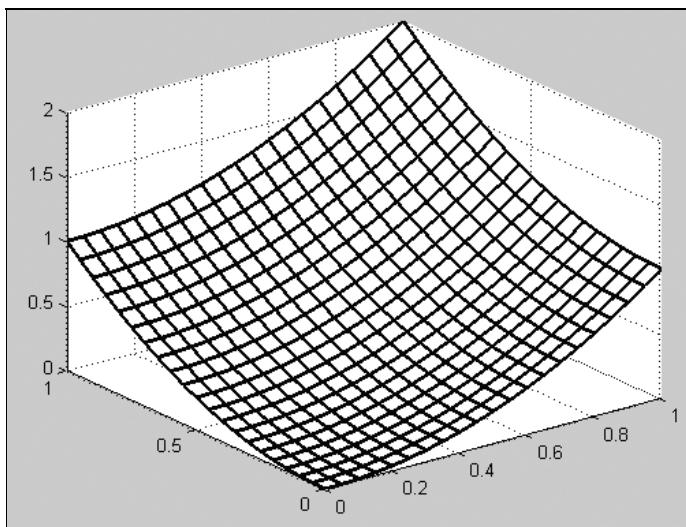


Рис. 2.14. График функции  $z(x, y)$  на более мелкой сетке

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

# Задания для самостоятельной работы

Успешная работа в MATLAB и большинстве Toolbox невозможна без понимания принципов обращения с массивами данных. Самостоятельное решение приведенных ниже заданий поможет вам убедиться, что вы освоили операции с векторами и матрицами, и эти вопросы не будут вызывать затруднений при чтении следующих разделов. При необходимости обращайтесь к материалу этой главы и справочной системе.

## Задания на векторы

В следующих заданиях необходимо придумать универсальный способ решения, пригодный для произвольных векторов, и протестировать его на различных показательных примерах.

1. Переставить элементы вектора в обратном порядке, используя индексацию подходящим вектором, и записать результат в новый вектор.
2. Выделить в новый вектор элементы вектора с четными номерами.
3. Найти сумму только положительных элементов вектора.
4. Заменить элементы вектора, отличающиеся от среднего геометрического его элементов более чем на 10%, на среднее геометрическое.
5. Заменить все минимальные элементы вектора максимальным значением его элементов.
6. Найти число положительных и отрицательных элементов вектора.

## Задания на матрицы

1. Задана квадратная матрица

$$A = \begin{pmatrix} -1.2 & 4.6 & -0.3 \\ 2.8 & 9.9 & -0.7 \\ 0.9 & -2.5 & 7.1 \end{pmatrix}.$$

Элементы матрицы  $A$  обозначим  $a_{ik}$ . Требуется узнать и записать размер матрицы в переменную  $n$  (конечно, он равен 3, но лучше использовать

вать функцию для автоматического определения размера) и вычислить приведенные ниже величины (нормы матрицы):

$$p = \max_{1 \leq i \leq n} \sum_{k=1}^n |a_{ik}|, \quad q = \max_{1 \leq k \leq n} \sum_{i=1}^n |a_{ik}|, \quad s = \sum_{i, k=1}^n |a_{ik}|.$$

2. При помощи встроенных функций для заполнения стандартных матриц, индексации двоеточием и, возможно, поворота, транспонирования или вычеркивания получите следующие матрицы:

$$\left[ \begin{array}{ccccccc} 2 & 1 & 0 & 0 & 0 & 0 & 5 \\ -1 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 6 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 7 & 0 \\ 5 & 0 & 0 & 0 & 0 & -1 & 8 \end{array} \right]; \quad \left[ \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right];$$

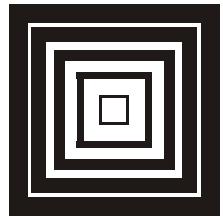
$$\left[ \begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right]; \quad \left[ \begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right].$$

3. Сконструируйте блочные матрицы (используя функции для заполнения стандартных матриц) и определите заданные величины, применяя функции обработки данных и поэлементные операции.

$$A = \begin{bmatrix} 1 & 0 & 0 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 4 & 0 \\ 0 & 0 & 1 & 0 & 0 & 4 \\ 2 & 2 & 2 & 3 & 3 & 3 \\ 2 & 2 & 2 & 3 & 3 & 3 \\ 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix}; \quad r = \left( \sum_{i, k=1}^n a_{ik}^2 \right)^{1/2};$$

$$A = \begin{bmatrix} 1 & 1 & -3 & -3 & -3 & -3 \\ 1 & 1 & -3 & -3 & -3 & -3 \\ -3 & -3 & 2 & 0 & 0 & 0 \\ -3 & -3 & 0 & 2 & 0 & 0 \\ -3 & -3 & 0 & 0 & 2 & 0 \\ -3 & -3 & 0 & 0 & 0 & 2 \end{bmatrix}; \quad s = \sum_{i=1}^5 a_{i,i+1}.$$

- Переставьте столбцы матрицы в порядке возрастания суммы элементов столбца.
- Найдите сумму всех положительных элементов матрицы.
- Считайте матрицу из файла matr.txt, замените в ней все элементы, большие среднего арифметического ее элементов, на среднее арифметическое и запишите ее в файл newmatr.txt.
- Определите максимальный столбцовый и строчный индексы отрицательных элементов матрицы.



## Глава 3

# Высокоуровневая графика

В данной главе описаны основные возможности высокоуровневой (high-level) графики MATLAB для отображения функций двух и трех переменных, визуализации векторных и матричных данных и векторных полей. Высокоуровневая графика позволяет пользователю получать результаты в графическом виде, прикладывая минимум усилий с использованием функций из командной строки. Работу, связанную с масштабированием осей и подбором цветов, MATLAB берет на себя. Удобным средством интерактивного создания графиков и редактирования свойств изображаемых объектов является редактор графических объектов (далее будем называть его редактором графиков), которому посвящена следующая глава. Его изучение разумно рассмотреть после ознакомления с основными функциями, поскольку он является графическим интерфейсом для вызова этих функций. Кроме того, окна **Workspace** также предоставляют возможность визуализации данных, что бывает особенно удобно на начальной стадии создания рисунков. С этой возможностью мы и начнем обсуждение средств MATLAB для высокоуровневой графики.

Для чтения этой главы необходимо уметь работать с массивами — векторами и матрицами, понимать выполнение поэлементных операций с массивами. Все эти необходимые сведения были изложены в главе 2.

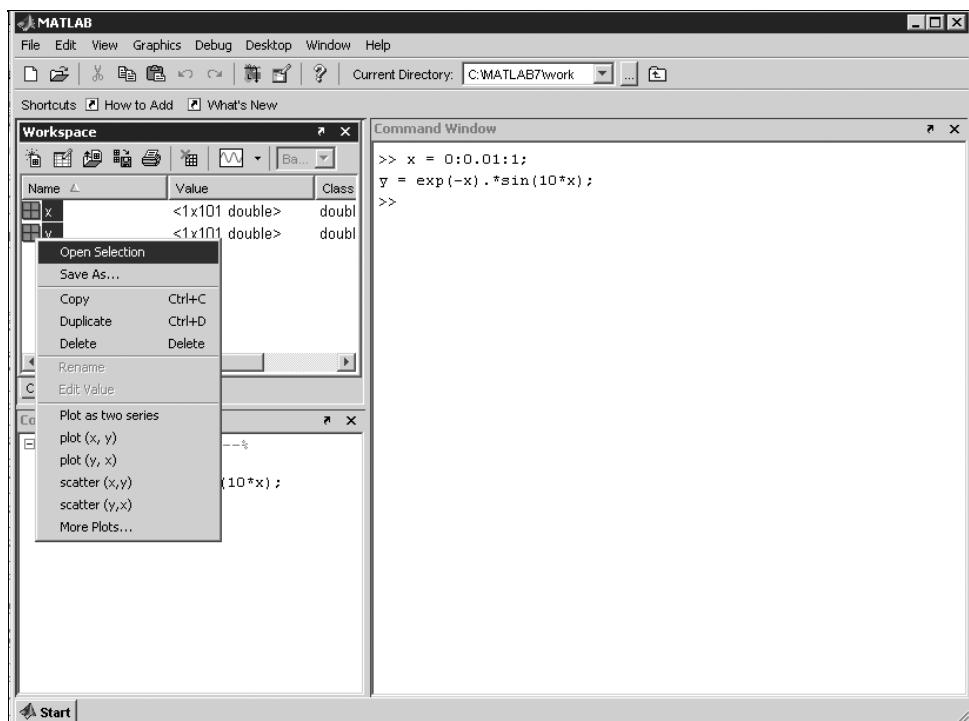
## Построение графиков из окна *Workspace*

Создание различных типов графиков, о которых пойдет речь далее, можно осуществить с помощью средств графического интерфейса окна **Workspace**. В предыдущей главе мы разобрали применение функции `plot` для построе-

ния графика функции  $y(x) = e^{-x} \sin 10x$  на отрезке  $[0, 1]$ . Для получения графика необходимо создать массивы исходных данных

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

Вместо вызова **plot** обратимся теперь к окну **Workspace**. В нем появились два массива **x** и **y**. Для построения графика функции  $y(x)$  следует выделить эти массивы щелчком мыши по имени массива с удержанием **<Ctrl>**, и выбрать тип графика **plot(x,y)** либо из контекстного меню, открываемого щелчком правой кнопкой мыши (рис. 3.1), либо из раскрывающегося списка типов графиков на панели инструментов окна **Workspace** (рис. 3.2).

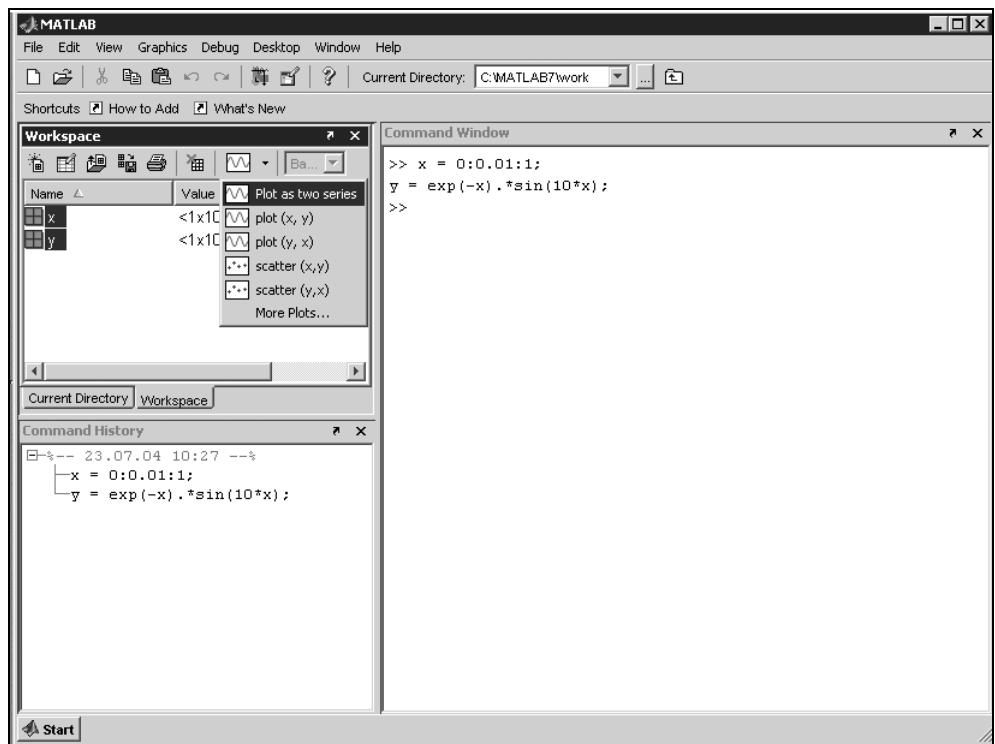


**Рис. 3.1.** Контекстное меню для выбора типа графика

### Примечание

Если панель инструментов окна **Workspace** отсутствует, то ее можно отобразить при помощи контекстного меню окна. Для этого следует щелк-

нуть правой кнопкой мыши по заголовку окна **Workspace** и в появившемся меню выбрать пункт **Workspace Toolbar**.



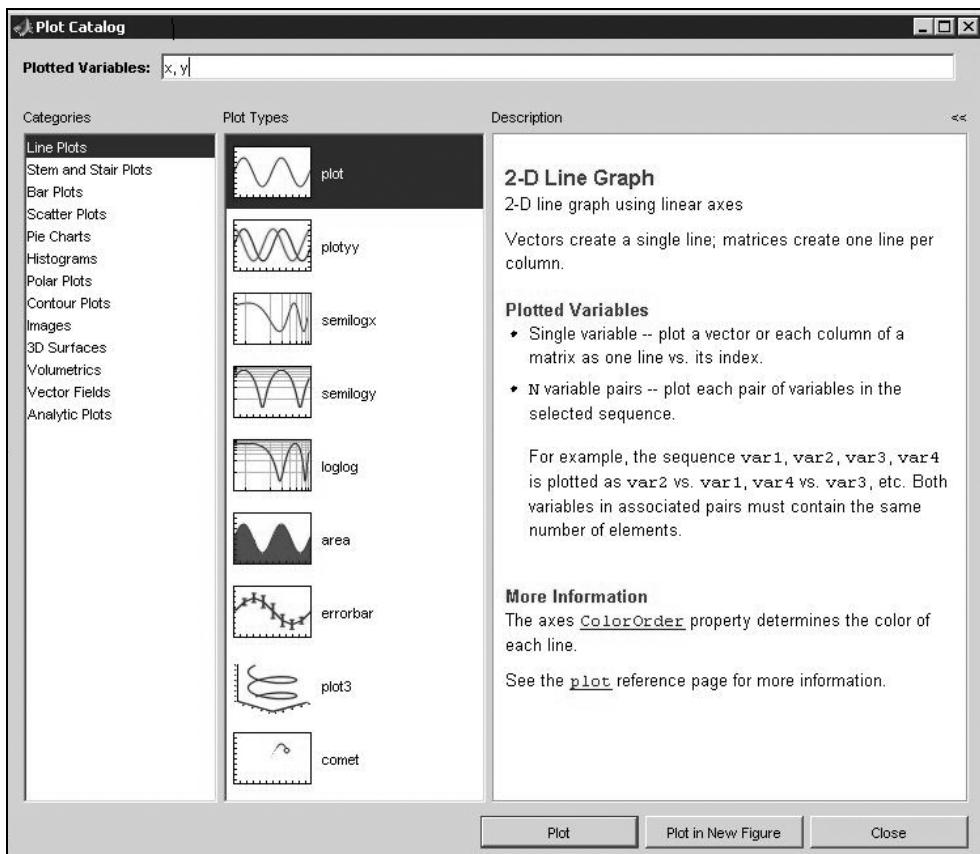
**Рис. 3.2.** Раскрывающийся список для выбора типа графика

Выбор **plot(x,y)** в контекстном меню или раскрывающемся списке приводит к автоматическому вызову **plot**, который отображается в командном окне

```
plot(x, y, 'DisplayName', 'y vs x', 'XDataSource', 'x',
 'YDataSource', 'y')
```

и появлению графика функции. Мы пока не будем обсуждать смысл входных аргументов функции **plot**, поскольку в данном примере она автоматически вызывается с использованием средств низкоуровневой графики, которые достаточно подробно описаны в главе 9. Следует отметить, что для графического отображения пары векторов **x** и **y** допустимо несколько способов, им соответствуют остальные пункты меню или элементы списка. Например, выбор **plot(y, x)** приводит к построению обратной зависимости **x(y)**.

Список типов графиков содержит пункт **More Plots** (Другие графики), выбор которого приводит к открытию дополнительного окна для выбора типа графика (рис. 3.3).



**Рис. 3.3.** Дополнительное окно  
для выбора типа графика

Дополнительное окно разделено на три колонки. В первой (**Categories**) пользователь выбирает категорию, к которой относится график, во второй (**Plot Types**) отмечает конкретный тип графика, а в третьей (**Description**) содержится описание для отмеченного типа графика. Кнопки в данном окне позволяют либо построить график в том же окне (**Plot**), либо в новом (**Plot New Figure**), либо закрыть окно без построения (**Close**).

При использовании средств окна **Workspace** для визуализации данных необходимо следить за тем, чтобы выбранные данные соответствовали друг другу. В предыдущем примере таким условием являлось совпадение длин векторов  $x$  и  $y$ . Попытка использовать векторы разной длины приведет к сообщению об ошибке!

Перед тем, как перейти к описанию функций высокого уровня графики, приведем еще один пример графического представления только одного вектора. Изучите пункты всплывающего меню для вектора  $y$ . Оказывается, можно построить "график вектора", т. е. зависимость элементов вектора от их номера (пункт **plot(y)**). Векторные данные могут быть представлены диаграммами различных типов (пункты **bar(y)**, **pie(y)**).

Знакомство с функциями высокого уровня графики MATLAB мы начнем с диаграмм и гистограмм.

## 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

$$x = [1.2 \ 1.7 \ 2.2 \ 2.4 \ 2.5 \ 1.3 \ 1.1 \ 0.5 \ 0.4 \ 0.1]$$

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. 3.4, содержащее столбчатую диаграмму вектор-строки (далее в книге на рисунках приводится только область построения графика, а заголовок окна, меню и па-

нель инструментов опускаются, за исключением следующей главы, где рассматривается редактор графиков). По горизонтальной оси откладывается номер элемента вектора, а по вертикальной — его значение. Аргументом функции `bar` может быть как вектор-строка, так и вектор-столбец.

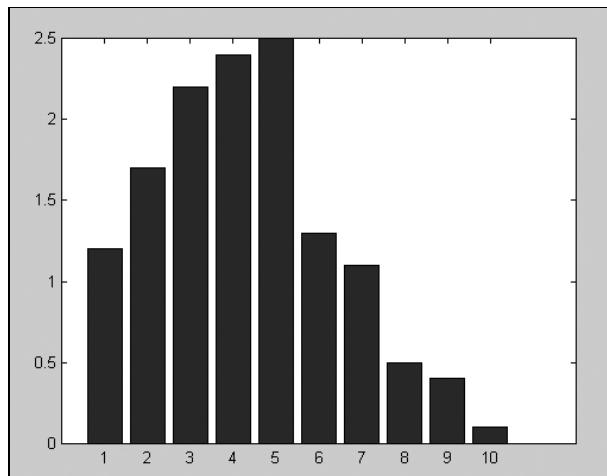


Рис. 3.4. Столбчатая диаграмма вектора

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. 3.5. 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 равной 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)
```

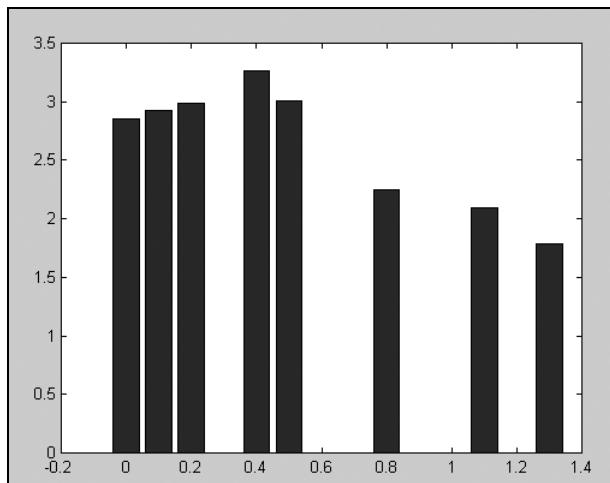


Рис. 3.5. Результаты измерений

Результат (диаграмма без промежутков между столбиками) показан на рис. 3.6.

### Примечание

Функция `barh` строит горизонтальную столбчатую диаграмму, т. е. повернутую на 90 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)
```

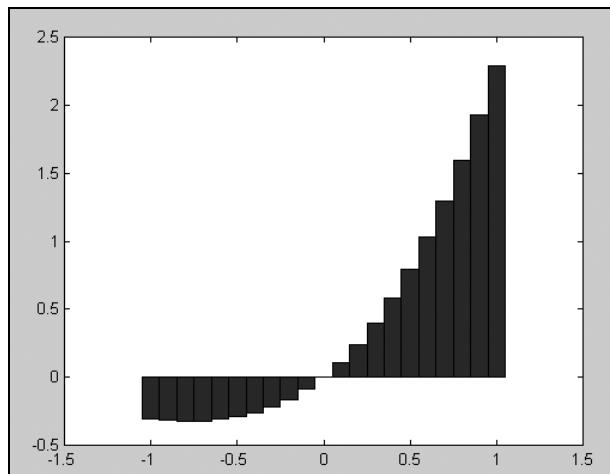


Рис. 3.6. Отображение функции в виде диаграммы

В результате получается диаграмма, изображенная на рис. 3.7, в которой площади секторов отвечают процентному вкладу каждого из элементов вектора в общую сумму, т. е. MATLAB нормирует значения, вычисляя `data/sum(data)`.

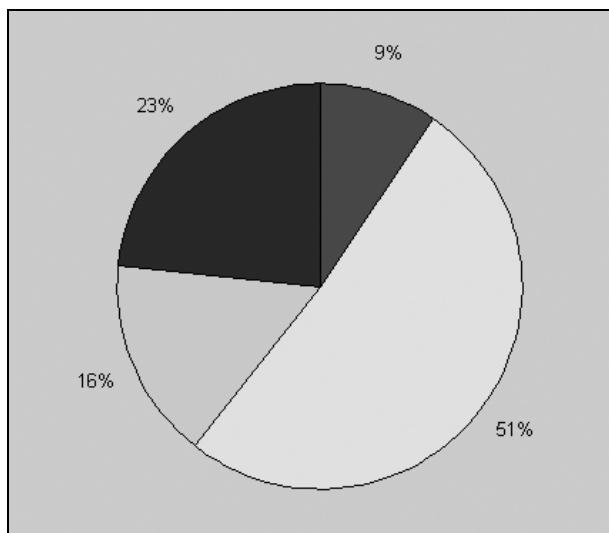
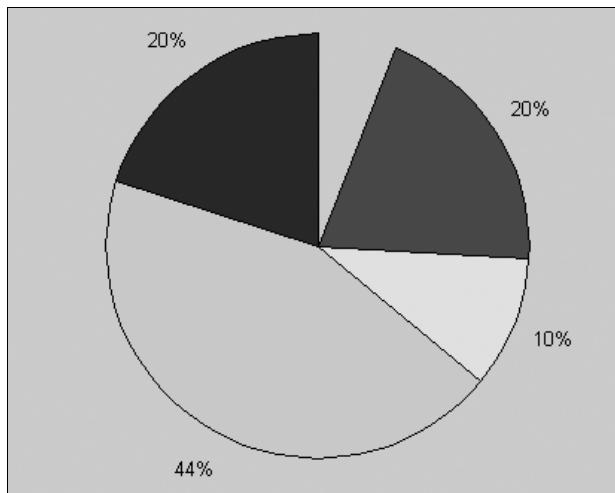


Рис. 3.7. Круговая диаграмма

Текстовые пояснения к секторам диаграммы более информативны, чем только значений долей в процентах. Разместить пояснения можно двумя способами. Один из них — воспользоваться редактором графиков (редактированию графиков посвящена *глава 4*).

Если же в результате работы приложения должна появиться готовая диаграмма с подписями данных, то придется применить специальные команды низкоуровневой графики (автоматическое размещение подписей к данным с использованием команд низкоуровневой графики объяснено в *главе 9*).

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. 3.8.



**Рис. 3.8.** Круговая диаграмма с выброшенным сектором

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 отдельным сектором (рис. 3.9), 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)
```

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

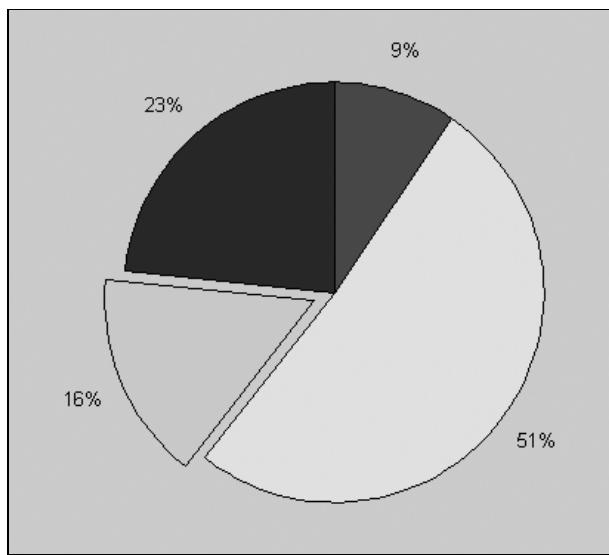


Рис. 3.9. Круговая диаграмма с отделенным сектором

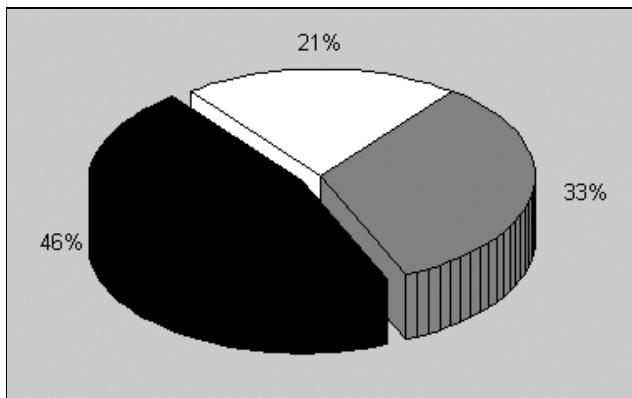
В качестве упражнения напишите команды построения диаграммы с отделенным сектором, соответствующим максимальному значению среди элементов вектора, автоматически создав вспомогательный вектор. Используйте функции `zeros` для создания нулевого вектора той же длины, что `x`, и `max` с двумя выходными аргументами 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)
```

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

```
>> data = [24.1 10.2 17.4 11.9];
>> parts = [1 0 0 0];
>> pie3(data, parts)
```

приводят к появлению трехмерной круговой диаграммы с отделенным сектором, изображенной на рис. 3.10.



**Рис. 3.10.** 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 `data` 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. 3.11.

### Примечание

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)
```

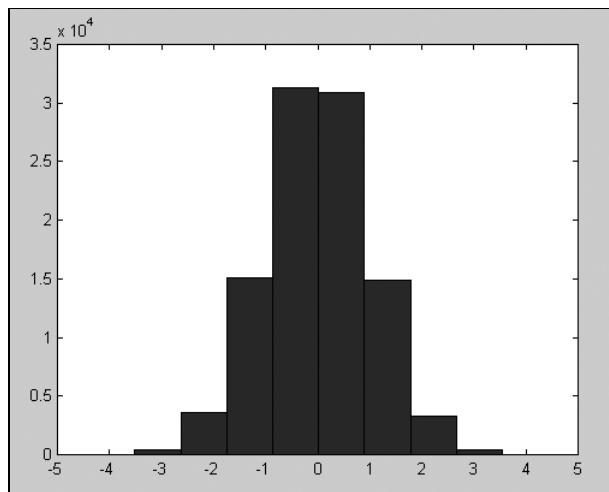


Рис. 3.11. Гистограмма распределения чисел по десяти интервалам

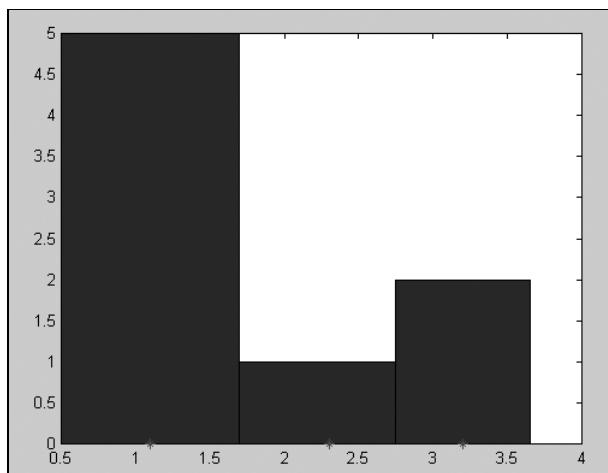


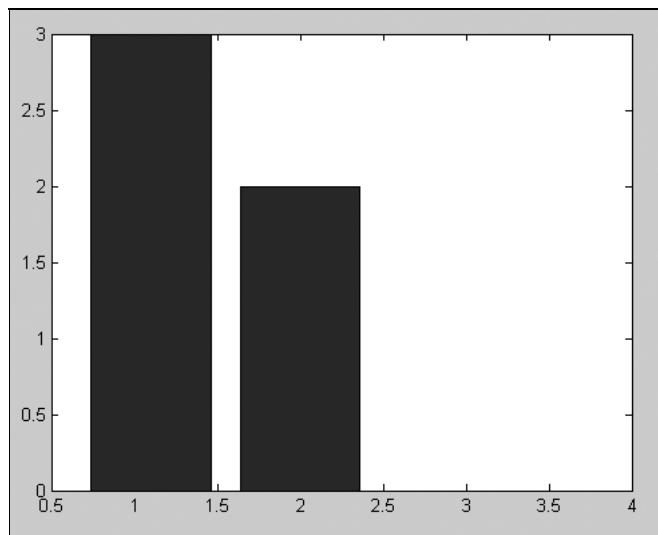
Рис. 3.12. Гистограмма распределения по интервалам, задаваемым центрами

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

Часто необходимо задать не центры, а границы интервалов. Для построения таких гистограмм следует использовать функцию `histc` в сочетании с вышеописанной `bar`. Функция `histc` возвращает вектор, содержащий число величин, попавших в заданные вторым аргументом полуоткрытые интервалы с включенным левым концом. 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(data, intervals)
count =
 3 2 0
>> bar(intervals, count)
```

В результате выполнения вышеописанных команд появляется гистограмма, приведенная на рис. 3.13.



**Рис. 3.13.** Гистограмма распределения по интервалам, задаваемым границами

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 записыва-

ется вектор, содержащий распределение данных по интервалам. Следующий пример демонстрирует создание вектора `count` из пяти элементов, каждая компонента которого является числом элементов из `data`, попавших в один из пяти интервалов

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

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
```

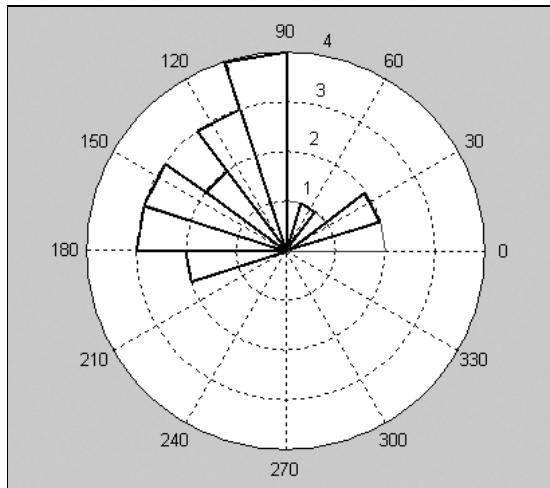
Функция `rose` предназначена для построения угловых гистограмм (в полярных координатах). Аргументом функции `rose` является вектор значений в радианах. Угловые гистограммы дают наглядное представление о данных, связанных с измерениями направлений. Пусть, например, в течение суток каждый час измерялось 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. 3.14. Из нее следует, что преобладающее направление примерно равно  $100^\circ$ .

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.



**Рис. 3.14.** Круговая гистограмма распределения направлений ветра

## 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. 3.15. На диаграмме расположены четыре группы данных, каждая из которых состоит из трех столбиков, соответствующих измеряемым величинам.

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. 3.16, получена при помощи `bar(DATA, 1.7)`.

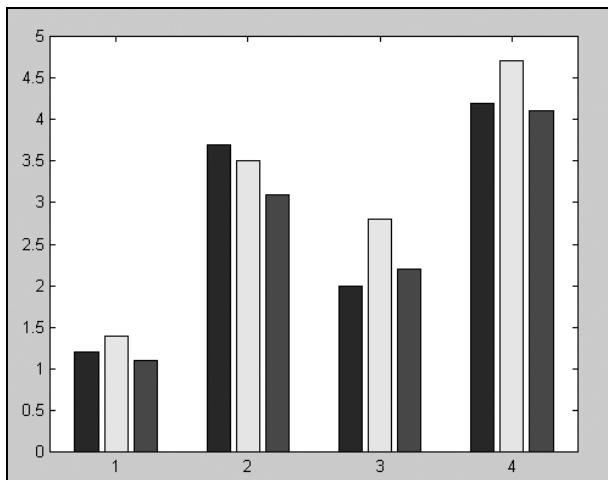


Рис. 3.15. Диаграмма сгруппированных данных

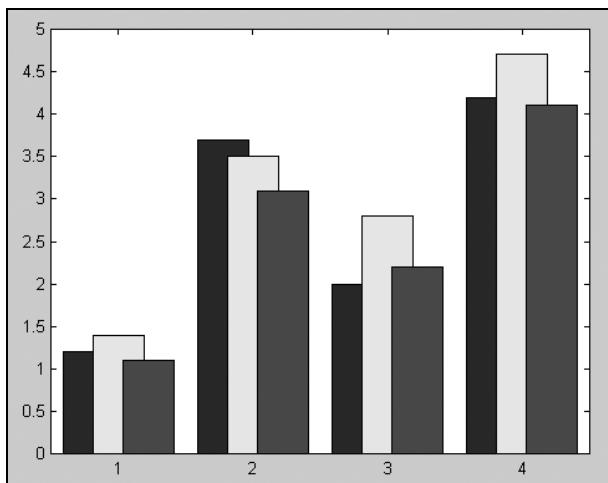


Рис. 3.16. Диаграмма сгруппированных данных с перекрывающимися столбцами

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. 3.17 показана диаграмма с накоплением, построенная при помощи функции `bar` с дополнительным аргументом: `bar(DATA, 'stack')`.

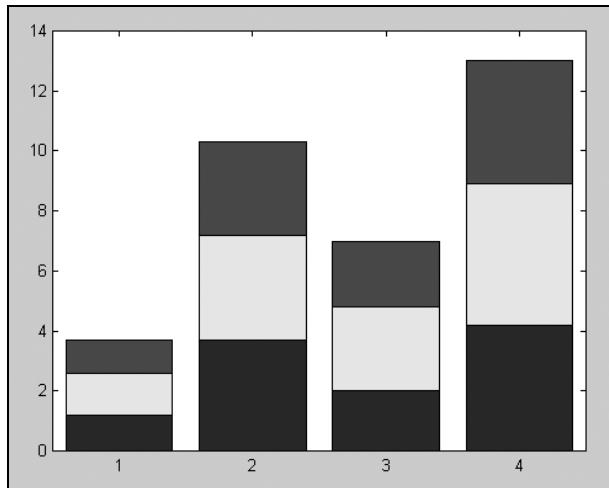


Рис. 3.17. Диаграмма с накоплением

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

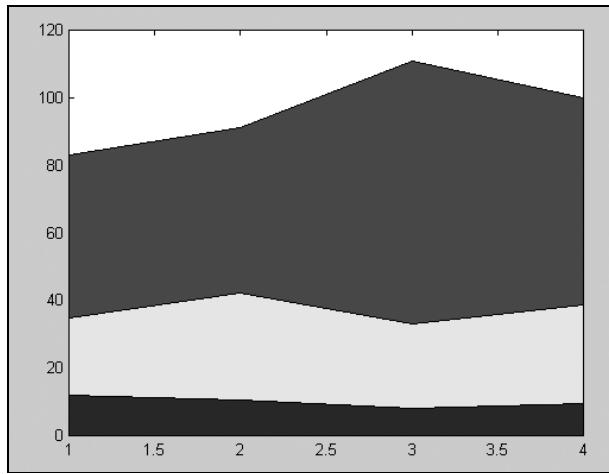


Рис. 3.18. Диаграмма с областями

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

ми. Запишите в матрицу GAIN поквартальную прибыль от продаж трех видов продукции и проследите за изменением прибыли при помощи диаграммы с областями (рис. 3.18).

```
>> 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)
```

Дополнительные возможности для визуализации разреженных матриц и представлении матриц в виде изображений предоставляют функции `spru` и `imagesc` (применение `spru` 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. В главе 2 приведен пример вывода простейшего графика функции одной пере-

менной и объяснено, почему при вычислении вектора значений функции необходимо применять поэлементные операции

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

Обратите внимание, что фактически мы строим график табличной функций, т. е. зависимость одного вектора от другого. Неудачный выбор шага по оси абсцисс может исказить реальную картину поведения функции. MATLAB предоставляет другую возможность (`fplot`) для визуализации аналитически заданных функций с адаптивным подбором шага, учитывающим особенности поведения исследуемой функции (использование `fplot` объяснено в главе 5).

Итак, всюду в этой главе мы будем иметь дело с таблично заданными функциями одной и двух переменных. При создании таблицы значений следует уделять повышенное внимание выбору шага вычисления. В этом убеждает простой пример. Требуется построить график функции  $f(x) = e^{-x} (\sin x + 0.1 \sin(100\pi x))$  на отрезке  $[0, 1]$ . Выберите сначала шаг 0.01 по оси  $x$ , а затем 1/99 (не забудьте пересчитать вектор со значениями функции). Первый график для шага 0.01 просто неверный (почему?).

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-строки  $f$  и  $g$ , содержащие значения функций. Команда `plot` с двумя парами аргументов приводит к графику, изображенному на рис. 3.19.

```
>> x = -2*pi:0.01:2*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 векторы, 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)
```

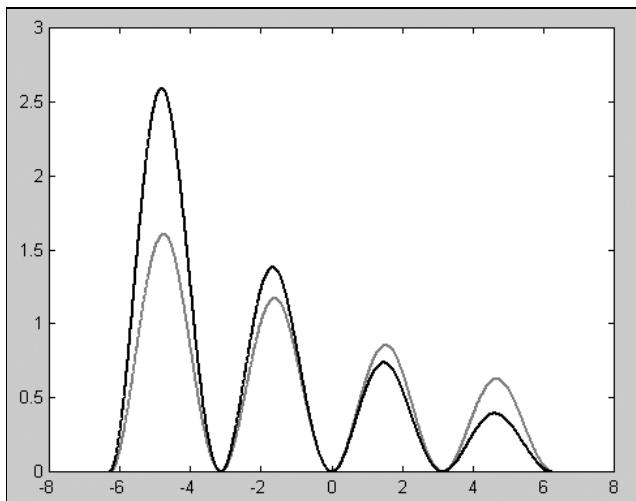


Рис. 3.19. Графики двух функций

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

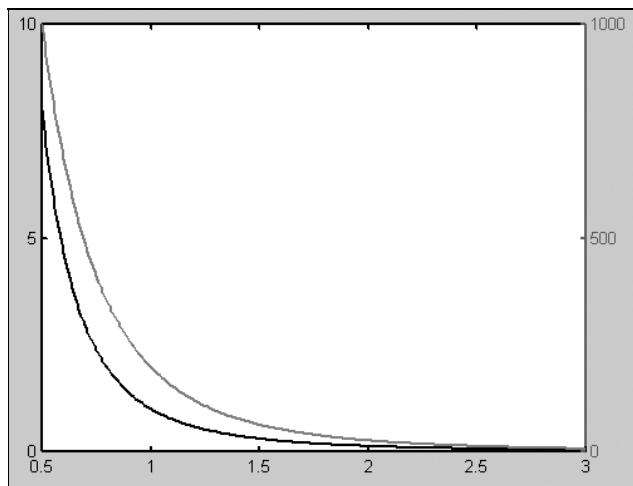
### Примечание

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 сливаются 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  $f(x) = x^{-3}$  и  $F(x) = 1000 \cdot (x + 0.5)^{-4}$

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

Результат приведен на рис. 3.20. При выполнении этого примера обратите внимание, что цвет графика совпадает с цветом соответствующей ему оси ординат.



**Рис. 3.20.** Сравнение функций при помощи `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 функции:

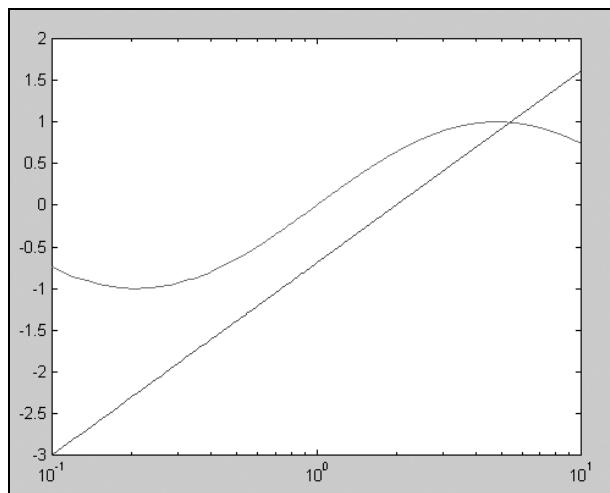
- `loglog` (логарифмический масштаб по обеим осям);
- `semilogx` (логарифмический масштаб только по оси абсцисс);
- `semilogy` (логарифмический масштаб только по оси ординат).

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. 3.21.



**Рис. 3.21.** Графики в полулогарифмической шкале

Функции `loglog` и `semilogy` вызываются аналогичным образом.

## 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. Используются 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                         | .                          |
| m    | розовый                        | о                          |
| c    | голубой                        | x                          |
| r    | красный                        | +                          |
| g    | зеленый                        | *                          |
| b    | синий                          | s                          |
| w    | белый                          | d                          |
| k    | черный                         | v<br>^<br><<br>><br>p<br>h |
|      | точка                          | -                          |
|      | кружок                         | :                          |
|      | крестик                        | -.                         |
|      | знак "плюс"                    | --                         |
|      | звездочка                      |                            |
|      | квадрат                        |                            |
|      | ромб                           |                            |
|      | треугольник<br>вершиной вниз   |                            |
|      | треугольник<br>вершиной вверх  |                            |
|      | треугольник<br>вершиной влево  |                            |
|      | треугольник<br>вершиной вправо |                            |
|      | пятиконечная<br>звезда         |                            |
|      | шестиконечная<br>звезда        |                            |

Например, для построения первого графика (рис. 3.19) красными точечными маркерами без линии, а второго пунктирной черной линией следует использовать команду `plot(x, f, 'r.', x, g, 'k:')`. Результат приведен на рис. 3.22. Обратите внимание, что абсциссы маркеров совпадают со значениями аргумента, содержащимися в `x`. Это не всегда хорошо, ведь для получения гладкой кривой требуется вычислить вектор значений функции в достаточно большом числе точек, что приводит к слишком частому расположению маркеров или даже их перекрытию. Простой прием позволяет по-

местить маркеры в заранее выбранные позиции. Строится два графика функции, один — сплошной линией, а второй — только маркерами для небольшого набора значений аргумента:

```
>> x = -1:0.01:1;
>> y = sin(2*pi*x.^2);
>> xm = -1:0.2:1;
>> ym = sin(2*pi*xm.^2);
>> plot(x, y, 'k', xm, ym, 'ko')
```

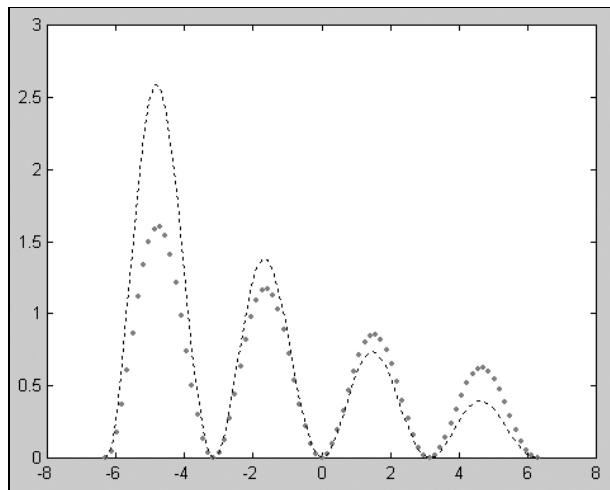


Рис. 3.22. Изменение параметров линий

## Оформление графиков

Удобство использования графиков во многом зависит от дополнительных элементов оформления: координатной сетки, подписей к осям, заголовка и легенды. Такие возможности реализуются либо с помощью дополнительных параметров, задающих свойства объектов, либо с помощью вспомогательных команд и функций. Перечислим основные из них. Сетка наносится командой `grid on`, функции `xlabel`, `ylabel` служат для размещения подписей к осям, а `title` — для заголовка. При необходимости сопроводить график легендой следует использовать функцию `legend`. Все перечисленные команды применимы к графикам как в линейном, так и в логарифмическом и полулогарифмическом масштабах. Следующие команды выводят графики из-

менения суточной температуры, изображенные на рис. 3.23, 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')
```

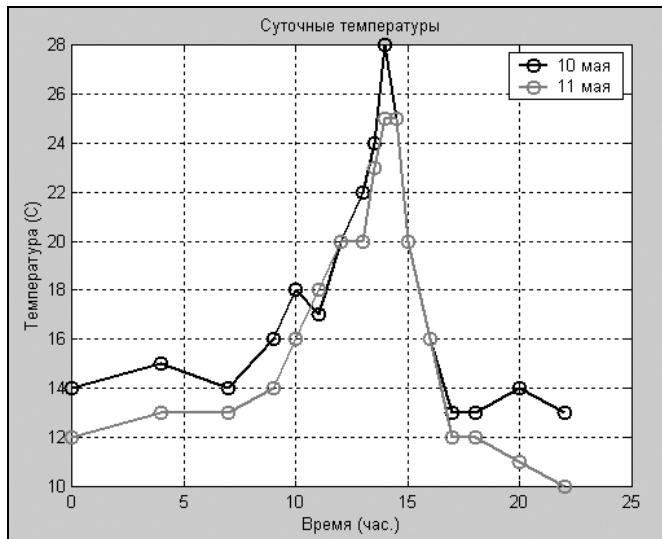


Рис. 3.23. График изменения суточной температуры

### Примечание

Символы кириллицы могут неправильно отображаться в нелокализованной версии MATLAB. Один из способов решения проблемы заключается в изменении текстового файла matlabrc.m, находящегося в подкаталоге toolbox\local основного каталога MATLAB. Используя любой текстовый редактор, добавьте в конец файла строку: `set(0,'DefaultAxesFontName','имя_шрифта_с_русскими_символами')` (см. также примечание в разд. "Сервисные функции для работы со строками" главы 8).

При размещении легенды следует учесть, что порядок и количество аргументов команды `legend` должны соответствовать линиям на графике. Последним 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 нижнем правом углу графика.

Такой способ задания места легенды был принят в прежних версиях MATLAB и поддерживается в версии 7. Кроме того, появилась другая возможность для указания положения легенды за счет привлечения ее свойства `Location`. Восемнадцать допустимых его значений приведены в справочной системе. Для того чтобы посмотреть их, достаточно набрать в командной строке `help legend` или обратиться к аналогичной информации в интерактивной справочной системе, воспользовавшись индексным поиском по слову `legend`. Например, для вывода легенды справа от осей следует выбрать значение '`EastOutside`'.

```
>> legend('10 мая', '11 мая', 'Location', 'EastOutside')
```

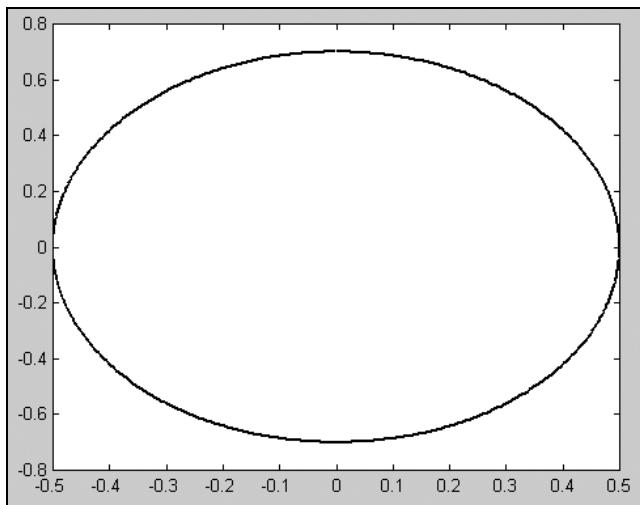
В заголовке графика, легенде и подписях осей допускается добавление формул и изменение стилей шрифта при помощи формата TeX, подробно об этом написано далее.

## 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. 3.24, получается при помощи следующих команд:

```
>> t = 0:0.01:2*pi;
>> x = 0.5*sin(t);
```

```
>> y = 0.7*cos(t);
>> plot(x, y)
```



**Рис. 3.24.** График параметрически заданной функции

Для того чтобы проверить свои знания о работе с массивами, постройте график кусочно-заданной функции (кусочно-функциональной зависимости):

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

Сначала необходимо вычислить каждую из трех ветвей, т. е. фактически получить три пары массивов  $x_1$  и  $y_1$ ,  $x_2$  и  $y_2$ ,  $x_3$  и  $y_3$ , затем объединить значения абсцисс в вектор  $x$ , а значения ординат в  $y$  и вывести график функции, задаваемой парой массивов  $x$  и  $y$ :

```
>> x1 = -2*pi:pi/30:-pi;
>> y1 = pi*sin(x1);
>> x2 = -pi: pi/30:pi;
>> y2 = pi-abs(x2);
>> x3 = pi: pi/30:2*pi;
>> y3 = pi*sin(x1).^3;
>> x = [x1 x2 x3];
```

```
>> y = [y1 y2 y3];
>> plot(x, y)
```

При таком подходе получается график, изображенный на рис. 3.25.

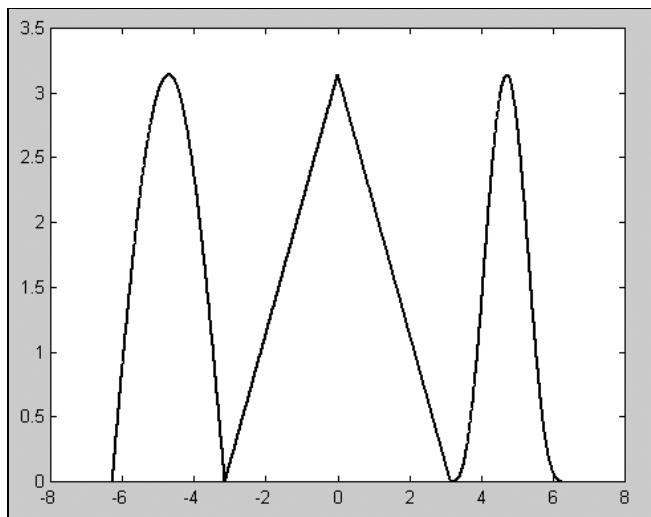


Рис. 3.25. График функции, заданной кусочным образом

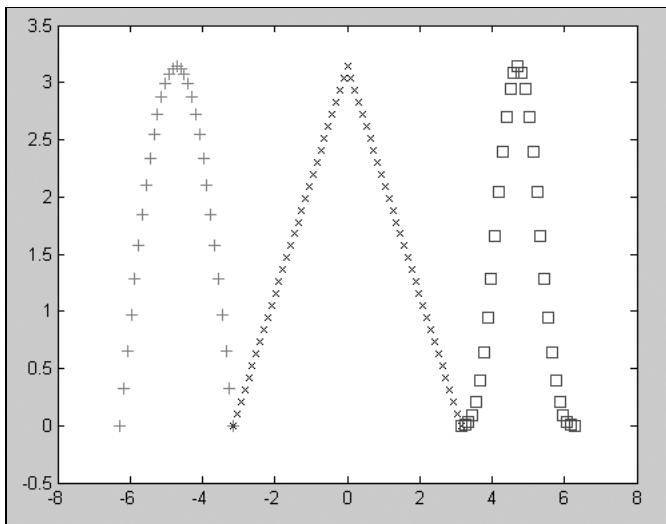


Рис. 3.26. График функции, заданной кусочным образом (разные цвета и маркеры ветвей)

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

```
>> plot(x1, y1, 'r+', x2, y2, 'kx', x3, y3, 'bs')
```

В этом случае график имеет более наглядный вид, т. к. каждая ветвь функции отображается своим цветом (рис. 3.26).

## 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`. Напомним, что 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, например, `mesh`.
4. Нанести на график дополнительную информацию, в частности, соответствие цветов значениям функции.

Сетка генерируется функцией `meshgrid`, вызываемой с двумя входными и двумя выходными аргументами. Входными аргументами являются векторы, элементы которых соответствуют сетке на прямоугольной области построения функции. Если область построения функции — квадрат и шаг сетки по обоим направлениям одинаков, то допустимо указать один аргумент, например: `[X, Y] = meshgrid(-1:0.05:1)`. Выходными аргументами являются матрицы с абсциссами и ординатами узлов сетки. Как было объяснено в главе 2, их структура влечет необходимость использования поэлементных операций при вычислении матрицы со значениями функции в узлах сетки.

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

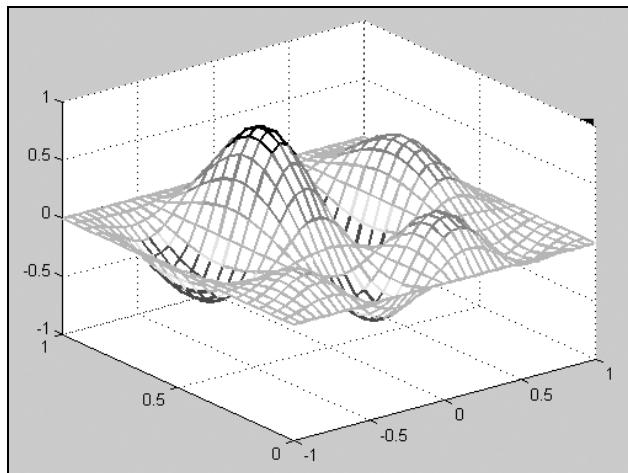
$$z(x, y) = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot (1 - x^2) \cdot y \cdot (1 - y)$$

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. 3.27, используется функция `mesh`, вызываемая с тремя аргументами

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



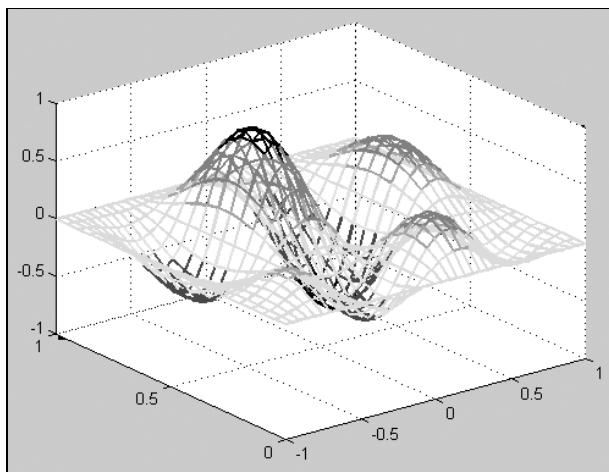
**Рис. 3.27.** Каркасная поверхность (`mesh`)

### Примечание

Интерфейс функции `mesh` и аналогичных ей графических функций, которые описаны ниже, достаточно гибок. Кроме приведенного способа обращения к `mesh` с тремя входными аргументами (матрицами), допускается также ряд других. В частности, если указан только один входной аргумент — матрица, то на осях абсцисс и ординат откладываются значения, соответственно, столбцевых и строчных индексов ее элементов. Вместо номеров строк и столбцов можно указать векторы, состоящие из требуемых чисел. Эти векторы задаются в первых двух входных аргументах, а матрица — в третьем.

Цвет линий поверхности соответствует значениям функции. По умолчанию рисуется только видимая часть поверхности. При помощи команды `hidden off` можно сделать каркасную поверхность "прозрачной", добавив

скрытую от взора часть (рис. 3.28). Команда `hidden on` убирает невидимую часть поверхности, возвращая графику прежний вид.



**Рис. 3.28.** 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 графику, изображенному на рис. 3.29.

В пределах каждой клетки цвет постоянный. Команда `shading flat` позволяет убрать каркасные линии. Для получения поверхности, плавно залитой цветом, зависящим от значений функции (рис. 3.30), предназначена команда `shading interp`.

При помощи `shading faceted` можно вернуться к виду поверхности, приведенной на рис. 3.29.

Трехмерные графики, изображенные на рис. 3.27—3.30, 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 цветовую шкалу, устанавливающую соответствие между цветом и значением функции. Постройте при помощи `surf` график поверхности и дополните его информацией о цвете.

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

```
>> colorbar
```

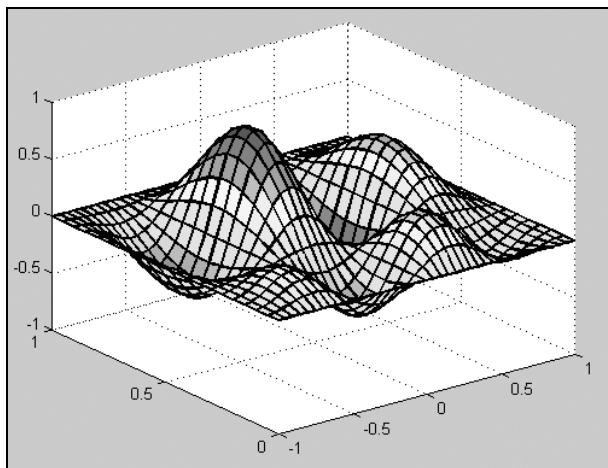


Рис. 3.29. Каркасная поверхность, залитая цветом (`surf`)

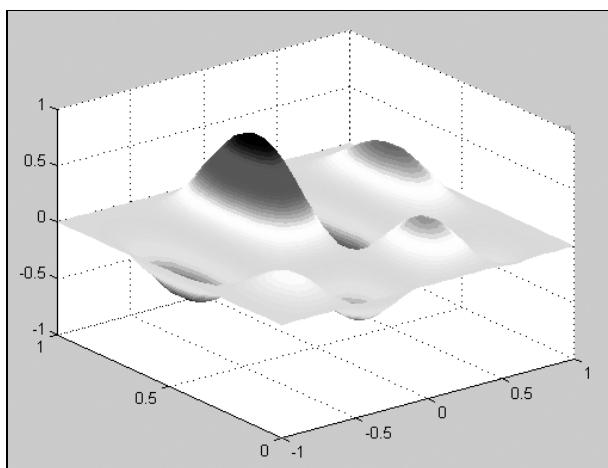


Рис. 3.30. Поверхность, залитая цветом (`surf, shading interp`)

Рисунок 3.31 иллюстрирует получающийся результат. Окно вывода графика несколько уменьшается из-за того, что рядом размещается цветовая шкала. Команду `colorbar` можно применять в сочетании со всеми функциями, строящими трехмерные объекты.

Однако, пользуясь графиком, изображенным на рис. 3.31, трудно сделать вывод о значении функции в той или иной точке плоскости  $xy$ . Более информативным является график, содержащий линии уровня функции, т. е.

линии постоянства значений функции, на плоскости  $xy$  (рис. 3.32). Для получения такого графика следует использовать `meshc` или `surfzc` вместо `mesh` или `surf` соответственно:

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

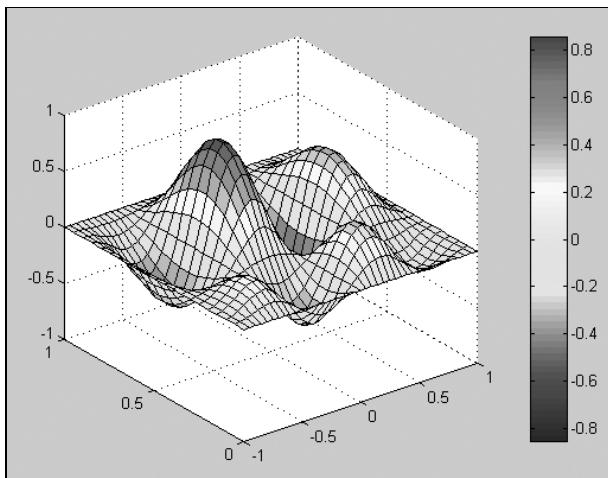


Рис. 3.31. Соответствие цвета и значений функции (colorbar)

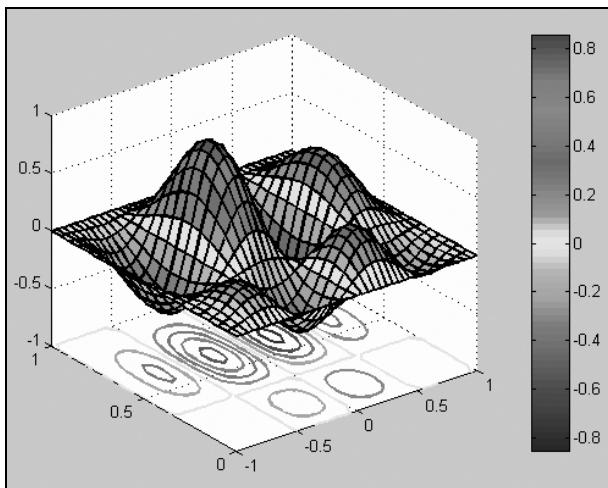
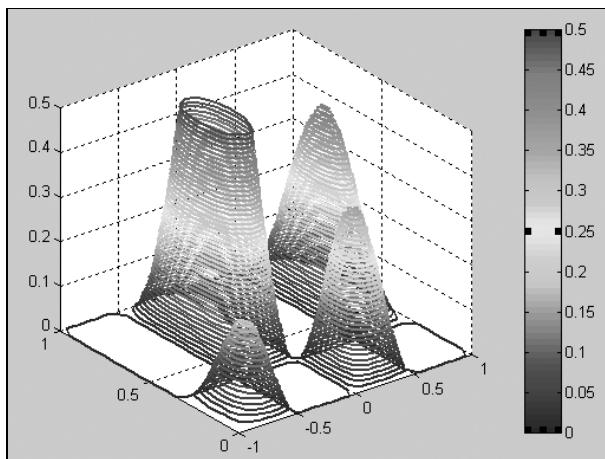


Рис. 3.32. График поверхности с линиями уровня на плоскости  $xy$  (surfc)

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. 3.33.



**Рис. 3.33.** График среза функции, состоящий из линий уровня (`contour3`)

Часто более содержательную информацию о числовых значениях дают плоские контурные графики с линиями уровня исследуемой функции.

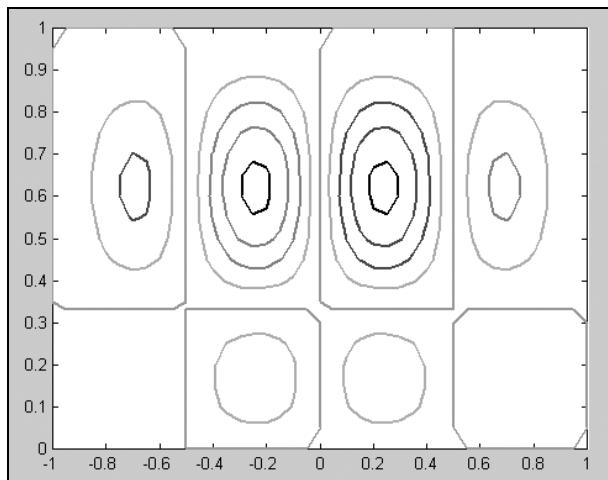
## Контурные графики

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

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

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

приводит к графику, изображенному на рис. 3.34, на котором показаны линии уровня на плоскости  $xy$ .



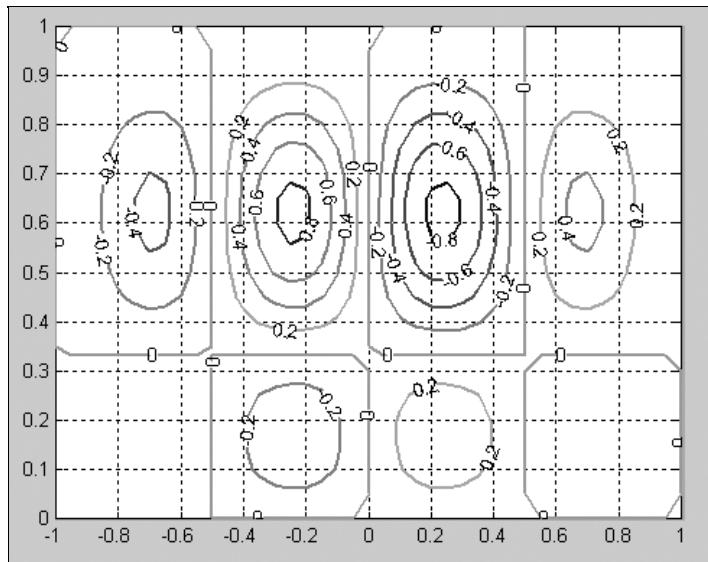
**Рис. 3.34.** Линии уровня функции (`contour`)

Такой график является малоинформационным, 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 ярлыком с соответствующим значением исследуемой функции 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 матрице `CMatr` содержится информация о линиях уровня, а в векторе `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. 3.35.

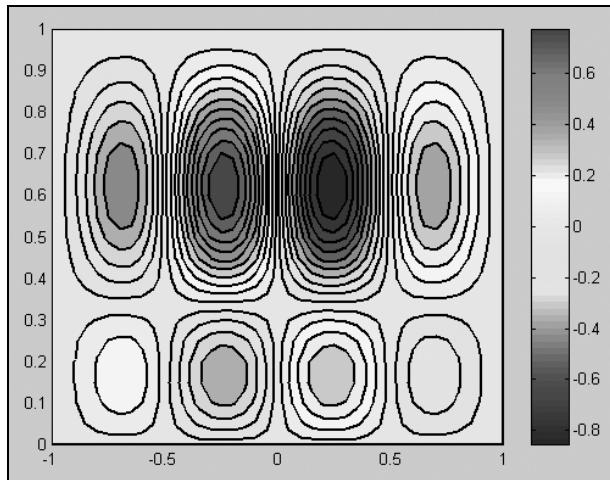


**Рис. 3.35.** Маркированные линии уровня (`contour`, `clabel`)

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 прямоугольной области определения 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. 3.36, который состоит из двадцати линий уровня, а промежутки между ними заполнены цветами, соответствующими значениям исследуемой функции:

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



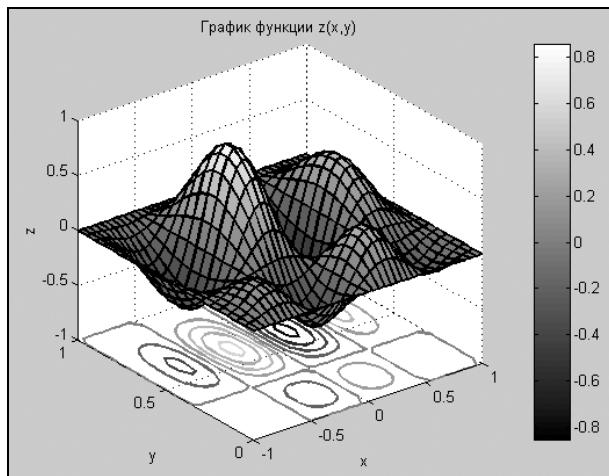
**Рис. 3.36.** Заливка цветом промежутков между линиями уровня (`contourf`)

## 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. 3.37.

```
>> 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.



**Рис. 3.37.** График с нанесенными обозначениями (палитра gray)

**Таблица 3.2.** Палитры цвета

| Палитра   | Изменение цвета                                                          |
|-----------|--------------------------------------------------------------------------|
| autumn    | Плавное изменение: красный-оранжевый-желтый                              |
| 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                                               |

Таблица 3.2 (окончание)

| Палитра | Изменение цвета                       |
|---------|---------------------------------------|
| vga     | Палитра Windows из шестнадцати цветов |
| white   | Один белый цвет                       |
| winter  | Оттенки синего и зеленого             |

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 формулу в заголовок или рядом с вертикальной осью. Использование в аргументах команд некоторых математических обозначений в формате TeX позволяет добавлять формулы на график. В заголовок графика, изображенного на рис. 3.37, 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 (три точки служат для размещения команды 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})')
```

которая приводит к появлению требуемого заголовка. 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 | <code>\itx</code>                        | <i>x</i>                     |
|                                              | <code>1.2\itP</code>                     | <i>1.2P</i>                  |
|                                              | <code>\itГиперболический</code><br>синус | <i>Гиперболический</i> синус |

Таблица 3.3 (окончание)

| Что требуется                                      | Команда TeX                                                                   | Результат                                                |
|----------------------------------------------------|-------------------------------------------------------------------------------|----------------------------------------------------------|
| Выделение жирным шрифтом одного символа или текста | Шаблон матрицы $\{\bf M\}$<br>$\{\bf A\chi\}$ фильтра                         | Шаблон матрицы <b>M</b><br><b>A</b> <b>χ</b> фильтра     |
| Набор символа или текста жирным курсивом           | Векторы $\{\bf \it x\}$ и $\{\bf \it y\}$<br>$\{\bf \it Optimalnaya\}$ кривая | Векторы <i>x</i> и <i>y</i><br><i>Optimalnaya</i> кривая |
| Изменение шрифта и его размера                     | $\{\fontname{arial}\fonts size{14} Z\text{-функция}\}$                        | <b>Z-функция</b>                                         |
| Степень, верхний индекс                            | $x^2$                                                                         | $x^2$                                                    |
|                                                    | $\{\it x\}^{2.5}$                                                             | $x^{2.5}$                                                |
|                                                    | $\{\it e\}^{\{\it x\}}$                                                       | $e^{-x}$                                                 |
| Нижний индекс                                      | $f_{\{5\}}$                                                                   | $f_5$                                                    |
|                                                    | $f_{\{\it xx\}}$                                                              | $f_{xx}$                                                 |

### Примечание

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  $a = \pi$ ". 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   |
|---------------------|----------|----------------------|-----------|---------------------|----------|
| <code>\alpha</code> | $\alpha$ | <code>\lambda</code> | $\lambda$ | <code>\chi</code>   | $\chi$   |
| <code>\beta</code>  | $\beta$  | <code>\mu</code>     | $\mu$     | <code>\psi</code>   | $\psi$   |
| <code>\gamma</code> | $\gamma$ | <code>\nu</code>     | $\nu$     | <code>\omega</code> | $\omega$ |

Таблица 3.4 (окончание)

| Команда  | Символ     | Команда | Символ   | Команда | Символ    |
|----------|------------|---------|----------|---------|-----------|
| \delta   | $\delta$   | \xi     | $\xi$    | \Gamma  | $\Gamma$  |
| \epsilon | $\epsilon$ | \rho    | $\rho$   | \Delta  | $\Delta$  |
| \eta     | $\eta$     | \sigma  | $\sigma$ | \Theta  | $\Theta$  |
| \theta   | $\theta$   | \tau    | $\tau$   | \Lambda | $\Lambda$ |
| \kappa   | $\kappa$   | \phi    | $\phi$   | \Phi    | $\Phi$    |

Таблица 3.5. Специальные символы

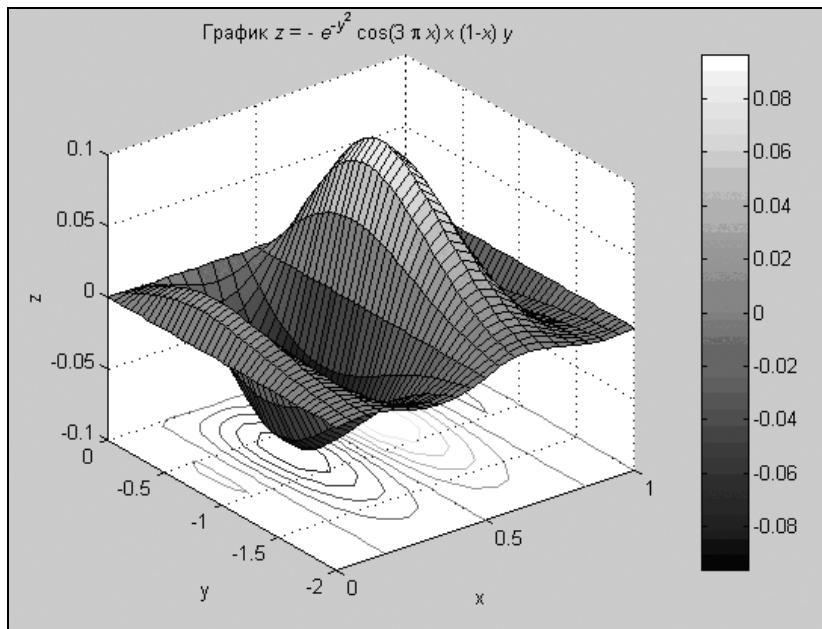
| Команда  | Символ     | Команда         | Символ            |
|----------|------------|-----------------|-------------------|
| \leq     | $\leq$     | \leftrightarrow | $\leftrightarrow$ |
| \geq     | $\geq$     | \leftarrow      | $\leftarrow$      |
| \pm      | $\pm$      | \rightarrow     | $\rightarrow$     |
| \propto  | $\propto$  | \downarrow      | $\downarrow$      |
| \partial | $\partial$ | \uparrow        | $\uparrow$        |

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. 3.38.

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;
>> surf(X, Y, Z)
>> colormap(gray)
>> colorbar
>> title('График $\{z\} = -e^{-y^2} \cos(3\pi x) \cdot x \cdot (1 - x) \cdot y$ ')
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```



**Рис. 3.38.** Задание для самостоятельной работы

По умолчанию в MATLAB поддерживается основной формат TeX, однако имеется возможность перейти к расширенному формату LaTeX для набора более сложных формул. Мы вернемся к этому вопросу в главе 9, поскольку он связан с обращением к свойствам текстовых объектов, каковыми являются заголовок и подписи к осям (см. разд. "Текстовые объекты" главы 9).

## 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. 3.39 положительные направления отсчета обозначены стрелками.

Изменение положения наблюдателя относительно графика в MATLAB осуществляется функция *view*. Аргументами *view* являются азимут и угол воз-

вышения, отсчитываемые в градусах. По умолчанию  $Az = -37.5^\circ$ ,  $El = 30^\circ$ . Для того чтобы узнать текущее положение наблюдателя, следует вызвать `view` с двумя выходными аргументами:

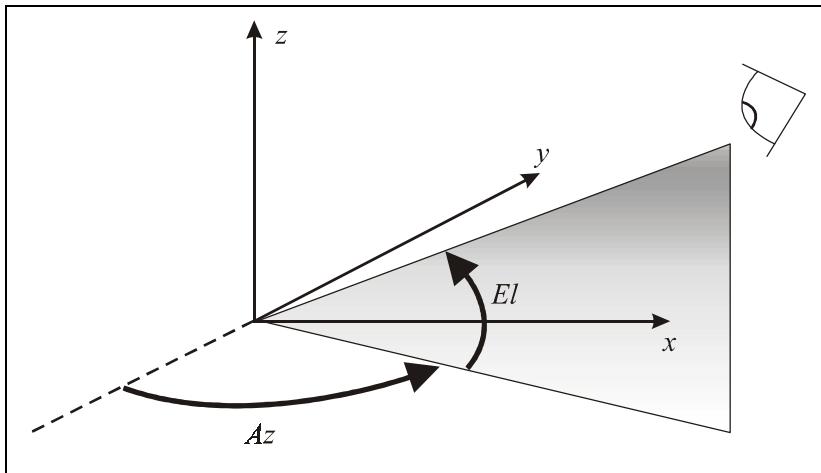
```
>> [Az, El] = view
```

```
Az =
```

```
-37.5000
```

```
El =
```

```
30
```

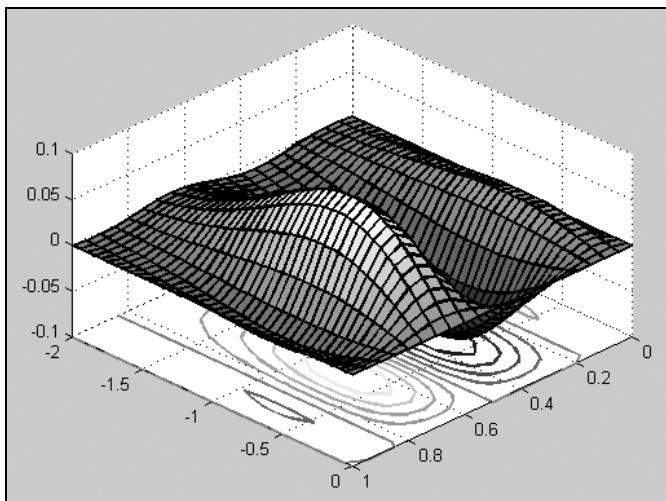


**Рис. 3.39.** Положение наблюдателя

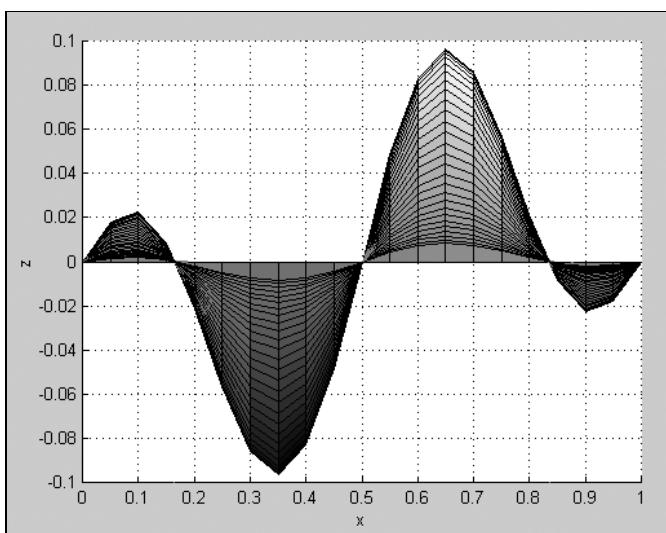
Положение наблюдателя задается входными аргументами `view`. Посмотрите, например, на поверхность, изображенную на рис. 3.38, возвышаясь над биссектрисой первого квадранта плоскости  $xy$  под углом  $45^\circ$ , для того чтобы увидеть скрытую часть поверхности. Используйте команду `view(135, 45)`, при этом получается график, приведенный на рис. 3.40.

Разверните график поверхности так, как показано на рис. 3.41, чтобы посмотреть на него вдоль оси  $u$  со стороны плоскости  $xz$ .

Несложно догадаться, что необходимо использовать `view(0, 0)`. Попробуйтесь самостоятельно, наблюдая за графиком поверхности из различных точек.



**Рис. 3.40.** График с точки зрения наблюдателя  
с  $Az = 135^\circ$ ,  $El = 45^\circ$



**Рис. 3.41.** Наблюдатель смотрит на график  
вдоль оси  $y$  со стороны плоскости  $xz$

## Построение параметрически заданных поверхностей и линий

MATLAB позволяет строить трехмерные линии, определенные формулами:

$$x = x(t), \quad y = y(t), \quad z = z(t), \quad t \in [a, b],$$

и поверхности, задаваемые зависимостями

$$x = x(u, v), \quad y = y(u, v), \quad z = z(u, v), \quad u \in [c, d], \quad v \in [e, f].$$

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

$$x = e^{-|t-50|/50} \sin t, \quad y = e^{-|t-50|/50} \cos t, \quad z = t, \quad t \in [0, 100].$$

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
```

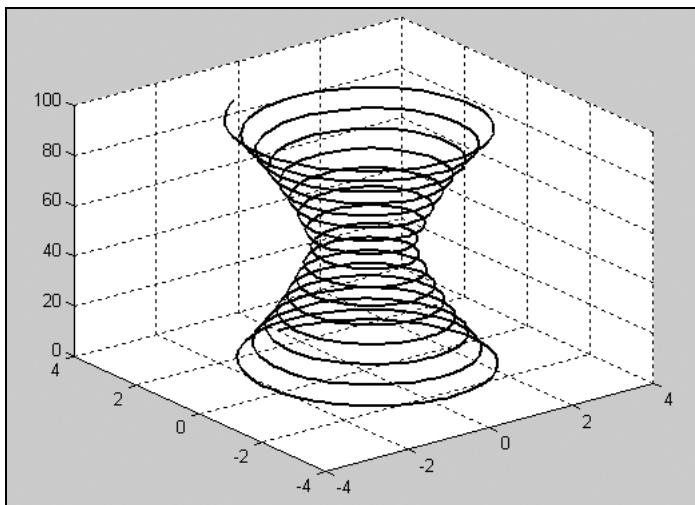
В результате выводится график, изображенный на рис. 3.42.

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  $y_2(a)$  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

$$x(u, v) = 0.3 \cdot u \cdot \cos v, \quad y(u, v) = 0.3 \cdot u \cdot \sin v, \quad z(u, v) = 0.6 \cdot u, \quad u, v \in [-\pi, \pi].$$



**Рис. 3.42.** 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);
```

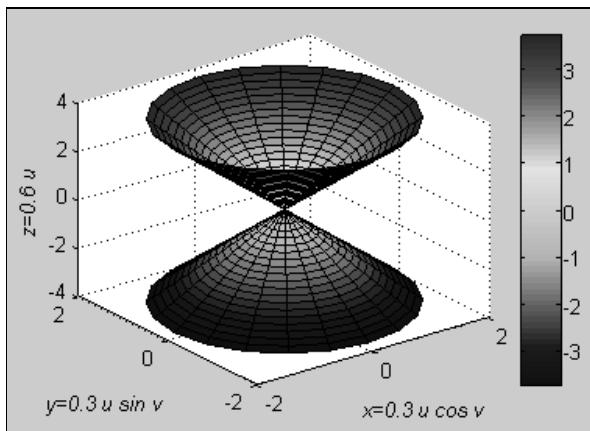
Матрица  $z$  должна быть того же размера, что  $x$  и  $y$  и, кроме того, она должна содержать значения, соответствующие значениям параметров. Если бы в функцию  $z(u, v)$  входило произведение  $u$  и  $v$ , то матрицу  $z$  можно было заполнить аналогично  $x$  и  $y$  при помощи внешнего произведения. 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('y = 0.3 \itu \sin \itv')
>> zlabel('z = 0.6 \itu')
```

приводит к графику, изображенному на рис. 3.43.



**Рис. 3.43.** 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 = 0.7*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 `surf1`.

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 `surf1` 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. 3.44.

```
>> [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);
>> surf1(X, Y, z)
>> colormap('copper')
>> shading interp
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

По умолчанию источник света имеет азимут, больший на  $45^\circ$ , чем наблюдатель, и тот же угол возвышения. Дополнительным четвертым аргументом `surf1` может быть вектор-строка из двух элементов — азимута и угла возвышения источника света. Измените, например, азимут источника на  $-90^\circ$  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
```

В результате получается освещенная поверхность, изображенная на рис. 3.45.

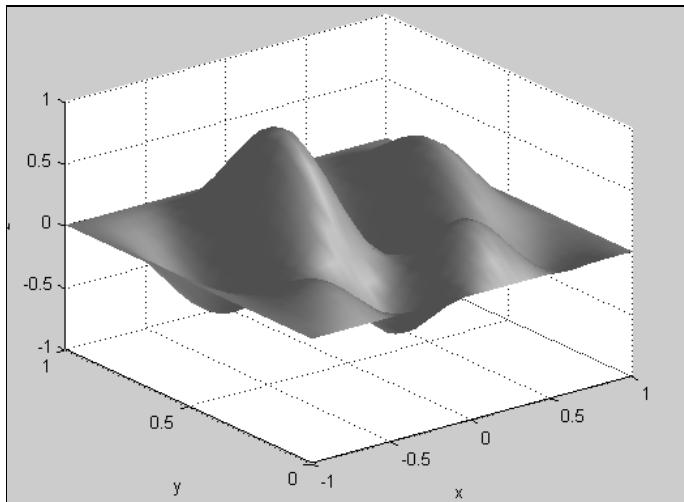


Рис. 3.44. Освещенная поверхность

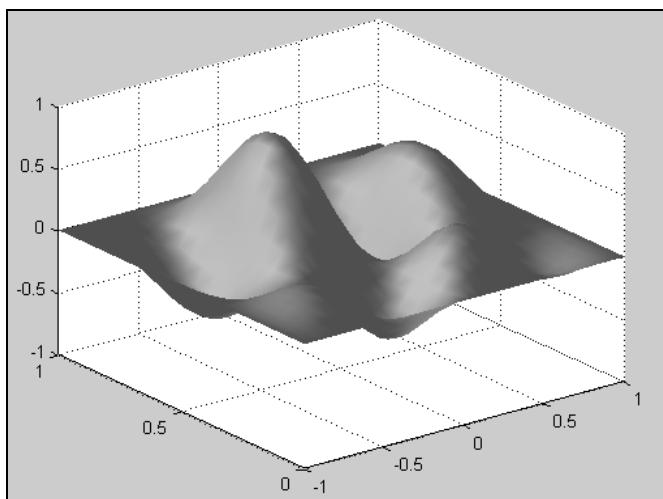


Рис. 3.45. Изменение положения источника света

## Анимированные графики

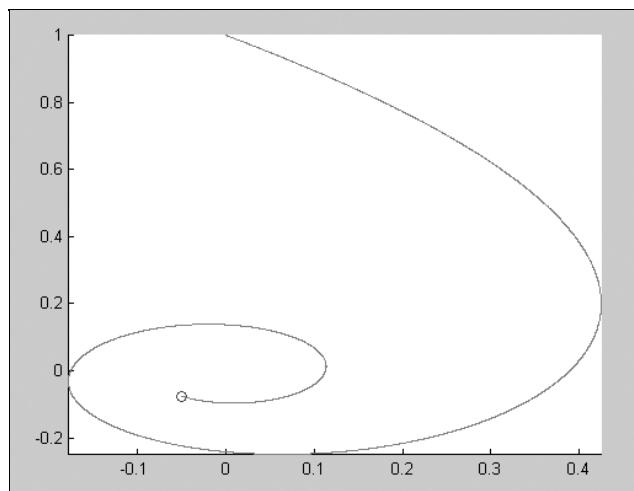
При изучении движения точки на плоскости или в трехмерном пространстве полезно не только построить траекторию точки, но и следить за движением точки по траектории. MATLAB предоставляет возможность получить анимированный график, на котором кружок, обозначающий точку, перемещается на плоскости или в пространстве, 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

$$x(t) = \frac{\sin t}{t+1} \quad y(t) = \frac{\cos t}{t+1}.$$

Действуйте точно так же, как при построении графика параметрически заданной функции, но для визуализации результата используйте `comet`:

```
>> 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. 3.46.



**Рис. 3.46.** Окончательный вид траектории движения (`comet`)

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 изменились по следующему закону

$$x = e^{-|t-50|/50} \sin t, \quad y = e^{-|t-50|/50} \cos t, \quad z = t.$$

Отобразите траекторию движения точки, применяя приведенные ниже 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 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 1**, содержащего каркасную поверхность, и **Figure 2** с освещенной поверхностью. Окно **Figure 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 1** текущим, следует щелкнуть на нем мышкой, вернуться в рабочую среду MATLAB и продолжать ввод команд. Команды повлекут изменения в окне **Figure 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`, указав 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 (рис. 3.47). 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('f=sinx')
>> figure(lnGr)
>> title('g=lnx')
>> grid on
```

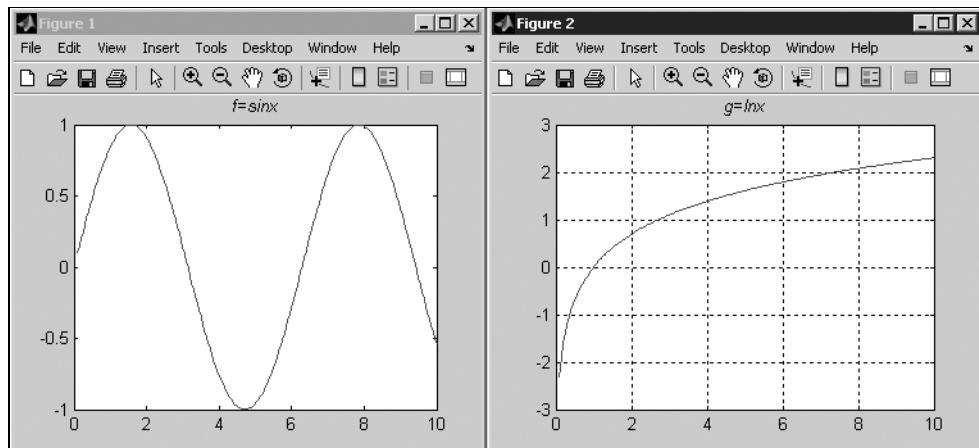


Рис. 3.47. Вывод графиков в разные окна

Для того чтобы очистить графическое окно с указателем `lnGr`, следует использовать `clf(lnGr)`. Удаление графика из первого окна, на которое указывает `sinGr`, производится при помощи `cla(sinGr)`.

## Вывод нескольких графиков на одни оси

Возможность отображения нескольких графиков функций одной переменной на одних осях использовалась при изучении `plot`, `plotyy`, `semilogx`, `semilogy`, `loglog`. Перечисленные команды позволяют выводить графики нескольких функций, задавая соответствующие векторные аргументы параметрически, например, `plot(x, f, x, g)`. Однако при построении трехмерных графиков или различных 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. 3.48.

```
>> 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
```

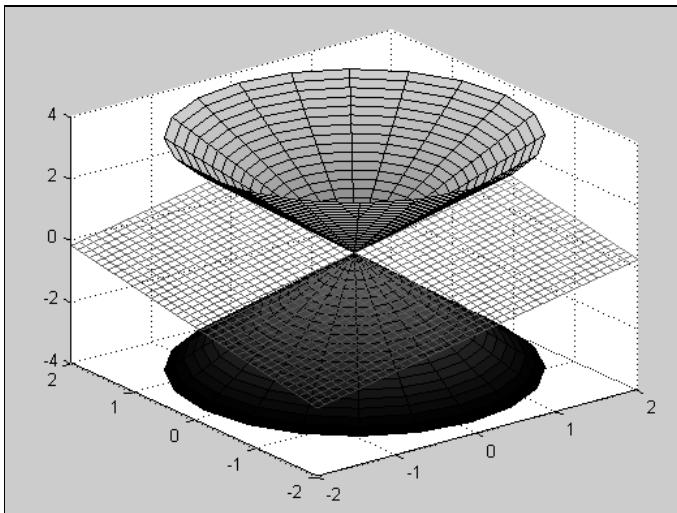
Команда `hidden off` применена для того, чтобы показать часть конуса, находящуюся под плоскостью.

Обратите внимание, что `hold on` распространяется на все последующие выводы графиков. Для размещения графиков на новых осях 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)
```



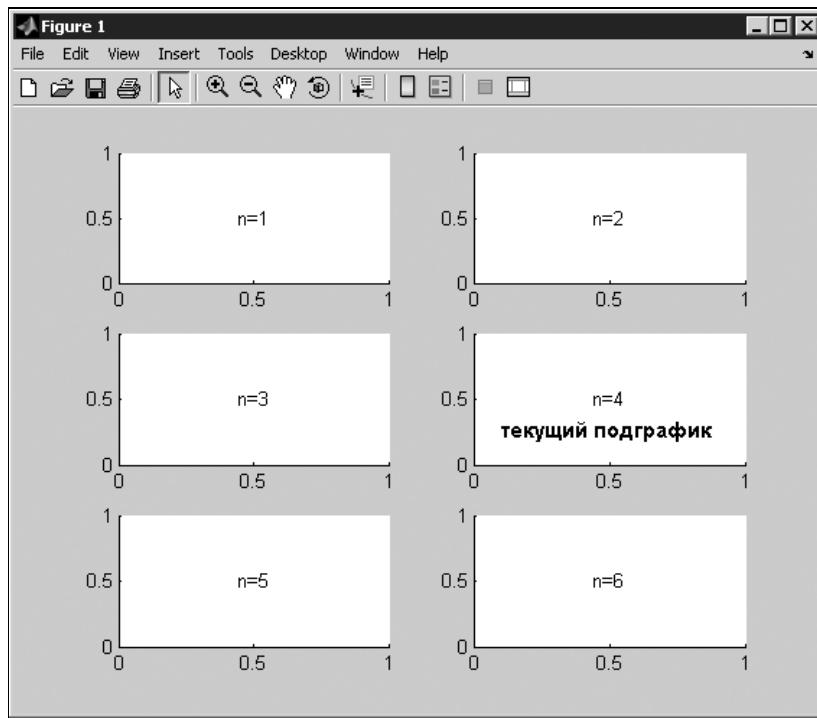
**Рис. 3.48.** Пересечение плоскости и конуса

## Несколько графиков в одном графическом окне

MATLAB позволяет разместить в графическом окне несколько осей и вывести на них различные графики. Самый простой способ заключается в разбиении окна на определенное число частей по вертикали и горизонтали с использованием функции `subplot`, которая располагает оси 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. Последовательность вызовов

```
>> subplot(3, 2, 1)
>> subplot(3, 2, 2)
...
>> subplot(3, 2, 6)
```

приводит к размещению шести осей координат в графическом окне (рис. 3.49). Текущими являются последние созданные оси, т. е. все графические функции будут осуществлять вывод на правые нижние оси. Для вывода графиков на другие оси их надо сделать текущими. Это достигается либо щелчком мыши по ним, либо повторным вызовом функции `subplot`. Например, команда `subplot(3, 2, 4)` предполагает наличие шести подграфиков и делает четвертый текущим, что схематично изображено на рис. 3.49.



**Рис. 3.49.** Схема расположения подграфиков  
после выполнения команды `subplot (3, 2, 4)`

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)
>> surfcc(X, Y, Z)
>> title('surfcc')
>> subplot(3, 2, 5)
>> contour3(X, Y, Z)
>> title('contour3')
>> subplot(3, 2, 6)
>> surfl(X, Y, Z)
>> shading interp
>> title('surfl')
>> colormap(gray)
```

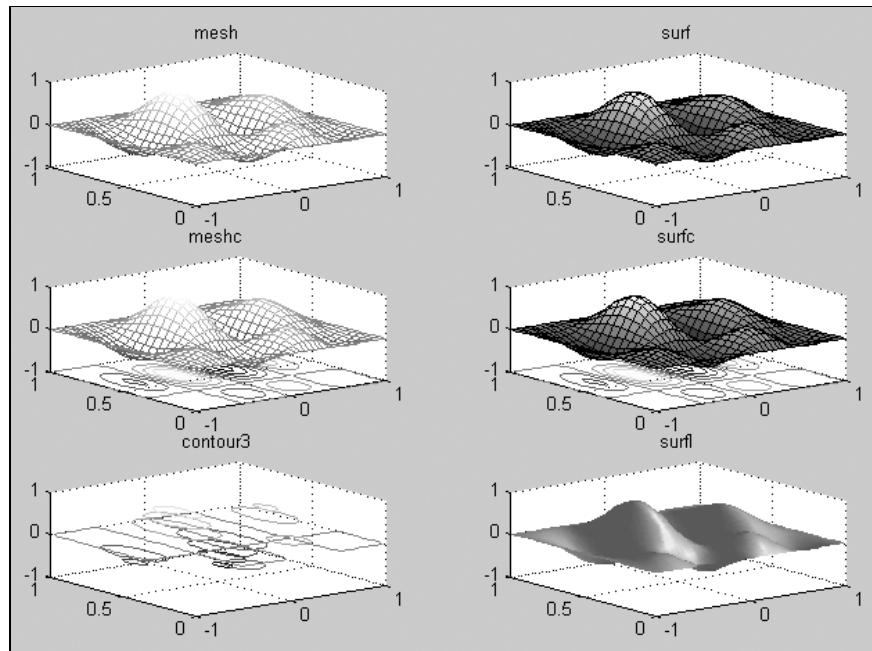


Рис. 3.50. Использование подграфиков (subplot)

В результате получается графическое окно, изображенное на рис. 3.50, 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.

## Визуализация векторных полей

Функции `compass`, `feather` и `quiver` графически представляют совокупность двумерных векторов. В качестве примера построим зависимость вектора скорости  $u = (u_x, u_y)$  тела, брошенного под углом к горизонту, от времени:

$$u_x = u_{x0}, \quad u_y = u_{y0} - \frac{g}{m}t,$$

здесь  $u_{x0}$ ,  $u_{y0}$  — проекции вектора начальной скорости;  $m$  — масса;  $g$  — ускорение свободного падения;  $t$  — время. Векторы скорости должны исходить из точек, принадлежащих траектории движения тела. Если его начальное положение совпадает с началом координат, то траектория движения описывается законом:

$$x = u_{x0} t, \quad y = u_{y0} t - \frac{g}{2m}t^2.$$

Выберем параметры задачи так, чтобы при  $u_{x0} = 0.5$ ,  $u_{y0} = 0.8$  продолжительность полета составила две секунды до падения тела на поверхность, и проследим за изменением его скорости через каждые 0.2 секунды

```
>> t = 0:0.2:2;
```

Запишите координаты тела в эти моменты времени в массивы `x` и `y`, а проекции скоростей на оси в `ux` и `uy`:

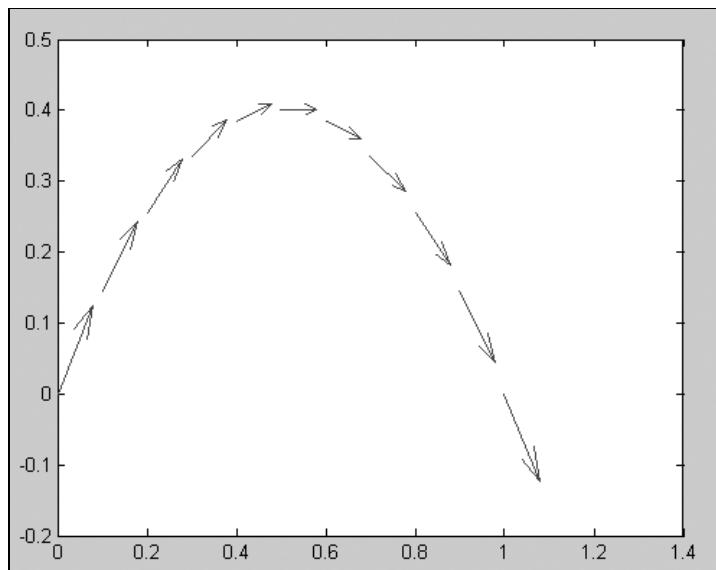
```
>> x = 0.5*t;
>> y = 0.8*t.* (1 - 0.5*t);
>> ux(1:length(x)) = 0.5;
>> uy = 0.8*(1 - t);
```

Для визуализации поля скоростей вызовите функцию `quiver`, указав в качестве первой пары входных аргументов массивы `x` и `y` с координатами начальных точек векторов, в качестве второй — векторы `ux` и `uy` со значениями

ми проекций векторов скоростей и масштабирующий множитель 0.5 последним пятым аргументом:

```
>> quiver(x, y, ux, uy, 0.5)
```

Получается наглядная картина движения тела, изображенная на рис. 3.51. В нашем примере функция `quiver` изображает векторное поле, заданное на кривой линии.



**Рис. 3.51.** Зависимость скорости тела от времени (функция `quiver`)

Обратите внимание, что векторы масштабируются, их длины на графике не совпадают с реальными значениями. Во-первых, автоматического масштабирования можно избежать, указав в качестве пятого аргумента ноль

```
>> quiver(x, y, ux, uy, 0)
```

Во-вторых, можно оставить автоматическое масштабирование, но указать коэффициент увеличения (или уменьшения) длины векторов после него. Сравните:

```
>> quiver(x, y, ux, uy, 1.5)
```

```
>> quiver(x, y, ux, uy, 0.3)
```

Функция `quiver` допускает замену стрелок, которыми рисуются векторы по умолчанию, на начинающиеся с маркера отрезки. Тип линий, цвет и маркер

указываются вместо масштабирующего множителя или после него (т. е. пятым или шестым входным аргументом).

```
>> quiver(x, y, ux, uy, 'or--')
>> quiver(x, y, ux, uy, 2.1, '*k:')
```

Подумайте, как отобразить не только векторы скорости, но и траекторию движения в виде плавной кривой. Очевидно, что нужно заново вычислить координаты тела с достаточно малым шагом по времени и использовать команду `hold on` для наложения линии движения тела на построенный график поля скоростей.

Функции `compass` и `feather` реализуют другие способы графического отображения векторных полей. Они отличаются от `quiver` тем, что не требуют задания начальных точек векторов. Функция `compass` представляет все векторы исходящими из начала координат, а `feather` — исходящими из равнодistantных точек на одной прямой. Эти функции удобны, если надо отследить изменение скорости безотносительно к траектории. Вызовите `compass` и `feather`, указав в качестве входных аргументов массивы со значениями проекций вектора скорости:

```
>> compass(ux, uy)
>> feather(ux, uy)
```

Результаты представлены, соответственно, на рис. 3.52 и 3.53.

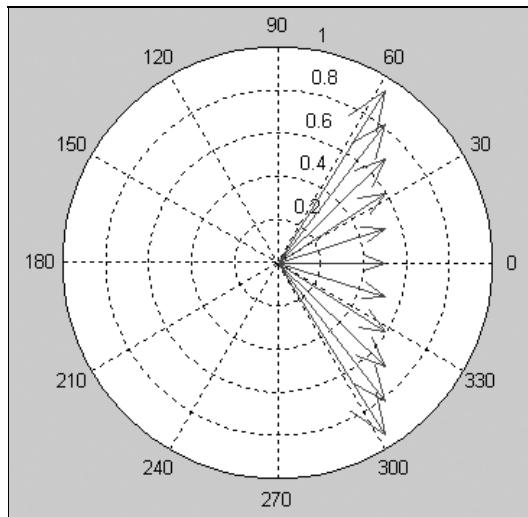
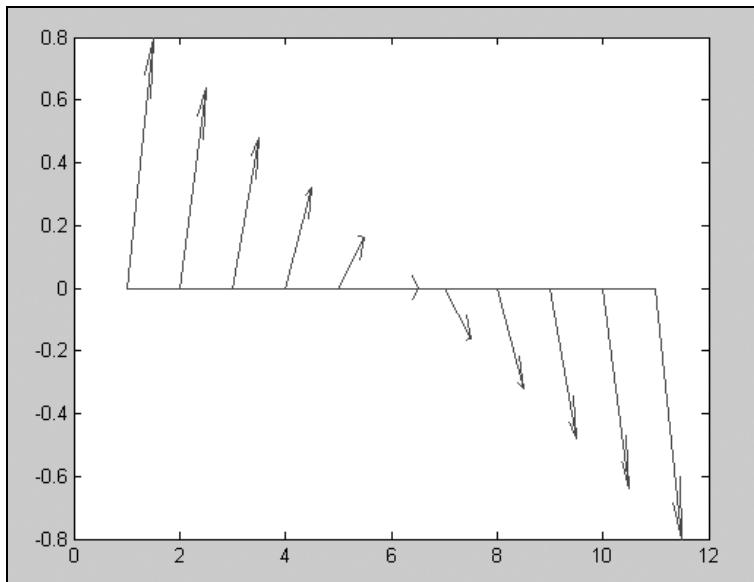


Рис. 3.52. Зависимость скорости тела от времени (функция `compass`)

Функции `compass` и `feather` могут иметь единственный входной аргумент — комплексный вектор, что эквивалентно вызову от двух векторов, первый из которых — вектор вещественных частей, а второй — мнимых. Управление свойствами линий, используемых вместо стрелок, производится так же, как и в `quiver`.



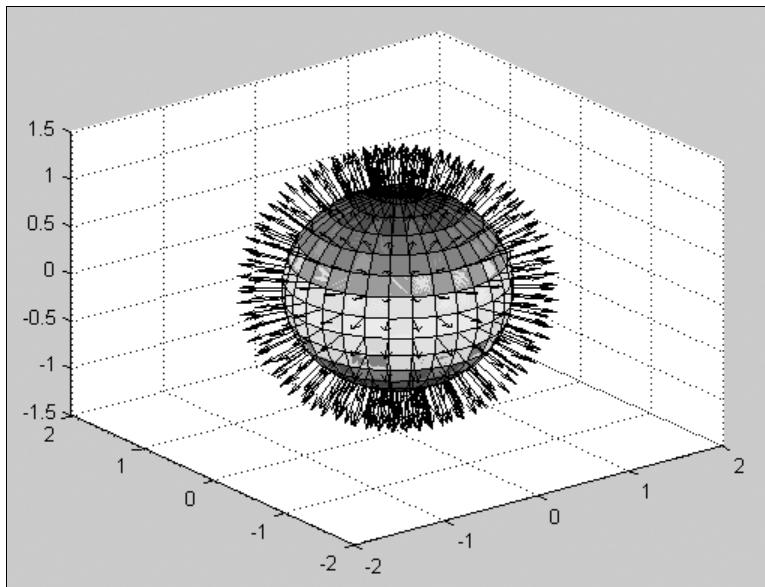
**Рис. 3.53.** Зависимость скорости тела от времени (функция `feather`)

Полная информация о функциях `quiver`, `compass` и `feather` содержится в интерактивной справочной системе MATLAB и *приложении 1*. Отметим, что функция `quiver` может быть использована для визуализации двумерных векторных полей, например, градиента функции (см. пример в справочной системе).

Для представления трехмерных векторных полей, заданных на поверхности, служит `quiver3`. Один из возможных примеров — векторы нормали к поверхности — приведен в справочной системе. Для нахождения нормалей в этом примере привлечена функция `surfnorm`. Она вычисляет компоненты векторов нормали в точках поверхности, которые соответствуют точкам сетки на плоскости ( $x, y$ ), генерируемым функцией `meshgrid`. Изучите

пример и самостоятельно постройте внешние нормали к шару радиуса 1 (рис. 3.54). Проверьте себя:

```
>> u = (-pi:pi/15:pi)';
>> v = -pi:pi/15:pi;
>> X = sin(u)*cos(v);
>> Y = sin(u)*sin(v);
>> Z = cos(u)*ones(size(v));
>> surf(X, Y, Z)
>> [U, V, W] = surfnorm(X, Y, Z);
>> hold on
>> quiver3(X, Y, Z, U, V,W, 4, 'k');
```



**Рис. 3.54.** Внешние нормали  
к шару (функция quiver3)

Для изображения трехмерных векторных полей, таких как распределение скоростей в движущейся жидкости или газе, MATLAB предоставляет функцию coneplot. Краткое описание функции дано в *приложении 1*.

## Задания для самостоятельной работы

1. Постройте графики функций одной переменной на отрезке  $[0.01, 2\pi]$ .

$$f(x) = \frac{\sin x}{x}, \quad g(x) = e^{-x} \cos x.$$

Выведите графики различными способами:

- в отдельные графические окна;
- в одно окно на одни оси;
- в одно окно на отдельные оси.

Дайте заголовки, разместите подписи к осям, легенду, используйте различные цвета, стили линий и типы маркеров, нанесите сетку.

Нарисуйте часть графика для отрицательных значений функции синим цветом, а для положительных — красным. Примите во внимание, что на самом деле отображается зависимость одного вектора от другого. Следовательно, можно применить функцию `find` для поиска индексов требуемых элементов вектора со значениями функции и индексацию вектором для выделения нужных компонент.

2. Визуализируйте функцию двух переменных на прямоугольной области определения

$$z(x, y) = (\sin x^2 + \cos y^2)^{xy}, \quad x \in [-1, 1], \quad y \in [-1, 1].$$

Выведите графики различными способами:

- каркасной поверхностью;
- залитой цветом каркасной поверхностью;
- промаркованными линиями уровня (самостоятельно выбрать значения функции, отображаемые линиями уровня);
- освещенной поверхностью.

Расположите графики в отдельных графических окнах и в одном окне с соответствующим числом осей. Представьте вид каркасной или освещенной поверхности с нескольких точек обзора.

Отметьте на трехмерном графике точки экстремумов. Используйте то, что значения функции в узлах сетки хранятся в матрице. Определите

максимальное значение функции при помощи `max`. Функция `find` позволит узнать столбцевые и строчные индексы этих элементов матрицы. Завершающий этап состоит в вызове `plot3` для расположения маркеров в точках трехмерного пространства на графике.

### 3. Постройте векторное поле градиента функции.

Визуализируйте трехмерное векторное поле на поверхности (возьмите разные поверхности):

- поверхность является гиперболоидом;
- поверхность является параболоидом;
- поверхность задана параметрически:

$$x(u, v) = \cos u \cdot \cos v, \quad y(u, v) = \sin u \cdot \sin v, \quad z(u, v) = u \cdot v, \quad u, v \in [0, 3].$$

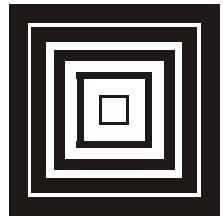
Постройте векторные поля:

- поле направлений отраженного света, падающего параллельным пучком на поверхность;
- поле направлений преломленного света, падающего параллельным пучком на поверхность;
- поле направлений отраженного света, направленного из точечного источника.

Рассмотрите изменение полей направлений в зависимости от взаимного расположения поверхности и источника.

При построении поля направлений подберите масштабный множитель и количество точек приложения векторов на поверхности для получения наилучшего представления о характере поля. Поле нормалей к поверхности, необходимое для определения направления лучей отраженного или преломленного вектора, вычислите с помощью функции `surfnorm`.

## Глава 4

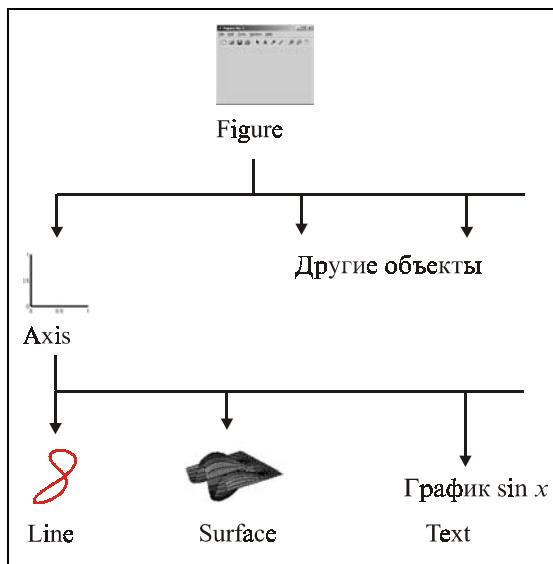


# Интерактивная среда для построения графиков

MATLAB предоставляет удобное интерактивное средство для построения графиков и корректировки их вида, нанесения дополнительной информации, подготовки их к печати, сохранения и экспорта в различные графические форматы — набор инструментов высокого уровня графики (*Plot Tools*), который далее будем называть редактором графиков. Использование редактора оправданно, когда требуется подготовить небольшое число графиков для печати или для вставки в какой-либо документ. Редактор графиков позволяет выполнять простейшие операции: задавать тип и цвет линии или свойства поверхности непосредственно при построении графика, размещать заголовок и легенду, подписывать оси. Начинающим пользователям этого вполне достаточно. Однако при написании собственных приложений в MATLAB с графическим выводом результатов часто возникает необходимость получить уже готовый график без дальнейшего редактирования. Возможна и более сложная ситуация, при которой отдельные элементы графиков должны изменяться прямо в ходе работы приложения, например, цвет или толщина уже существующей линии. В случае, когда приложение использует несколько графических окон со своими осями, необходимо выводить результаты на нужные оси в определенном графическом окне. Для решения этих задач не обойтись без средств дескрипторной графики (*Handle Graphics*), которой посвящена глава 9. Работая в редакторе графиков, вы познакомитесь со структурой графических объектов MATLAB и многообразием их свойств, что впоследствии облегчит понимание дескрипторной графики.

## Графические объекты

MATLAB является *объектно-ориентированной системой*, причем все *графические объекты* расположены в определенной иерархической последовательности. Для освоения средств, описываемых в настоящей главе, достаточно использовать упрощенное представление о графических объектах. Основным объектом является графическое окно (*figure*), в каждом окне могут быть расположены одна или несколько систем координат, определяемых своими осями (*axes*). На каждой системе осей размещаются другие графические объекты, например линии (*line*) или поверхности (*surface*). Кроме того, в области графического окна можно разместить различные поясняющие объекты: стрелки (*arrow*, *doublearrow*), стрелки с текстовой надписью (*textarrow*), надписи (*textbox*), геометрические фигуры (*ellipse*, *rectangle*). Поясняющие объекты принадлежат специальному слову *примечаний* (*annotation-layer*) графического окна.



**Рис. 4.1.** Структура объектов MATLAB  
(упрощенный вариант)

Важно понимать, что при вызове функций высокого уровня графики, например, `plot` (не важно из командной строки или средствами графического интерфейса), сначала создается графическое окно и принадлежащие ему оси, на которые выводятся линии графика. Другими словами, команда

`plot` создает все необходимые в иерархии графические объекты, а не "просто строит график".

В случае других графических функций (например, `surf`, `mesh` или `contour` и др.) ситуация аналогична, только мы уже имеем дело не с линиями, а с поверхностями. Редактор графиков позволяет легко изменить свойства каждого из вышеперечисленных объектов.

## Редактор графиков

Для начала работы с редактором графиков (далее в этой главе просто редактор) следует подготовить данные для визуализации. Проиллюстрируем работу на следующем примере: на одной паре осей построим графики функций  $\sin x$  и  $\cos x$  на отрезке  $[0, 5]$ , а на другом — поверхность функции  $e^{-x^2-y^2} \cdot (x-1)^2 \cdot \sin 2\pi y$  для  $x, y \in [-1, 1]$ . Ниже приведена последовательность команд для подготовки данных:

```
>> x = 0:0.1:5;
>> f = sin(pi*x);
>> g = cos(pi*x);
>> [X, Y] = meshgrid(-1:0.05:1);
>> z = exp(-X.^2 - Y.^2).* (X - 1).^2.*sin(2*pi*Y);
```

Для вызова редактора графиков воспользуйтесь командой

```
>> plottools
```

Появляется среда редактора графиков, изображенная на рис. 4.2.

Редактор графиков содержит графическое окно **Figure 1** с меню и панелью инструментов и три вспомогательных окна: **Figure Palette** (Шаблоны графики), **Plot Browser** (Браузер объектов) и **Property Editor - Figure** (Редактор свойств графических объектов). Обратите внимание, что на панели инструментов графического окна выбран инструмент **Edit Plot**, предназначенный для перехода в режим редактирования.

Наличие вспомогательных окон определяется состоянием флагов в одноименных пунктах меню **View** графического окна. Кроме того, для одновременного отображения или скрытия всех вспомогательных окон служат две последние кнопки на панели инструментов графического окна: **Hide Plot Tools** (Закрыть окна) и **Show Plot Tools** (Показать окна), причем всегда доступна только одна из них.

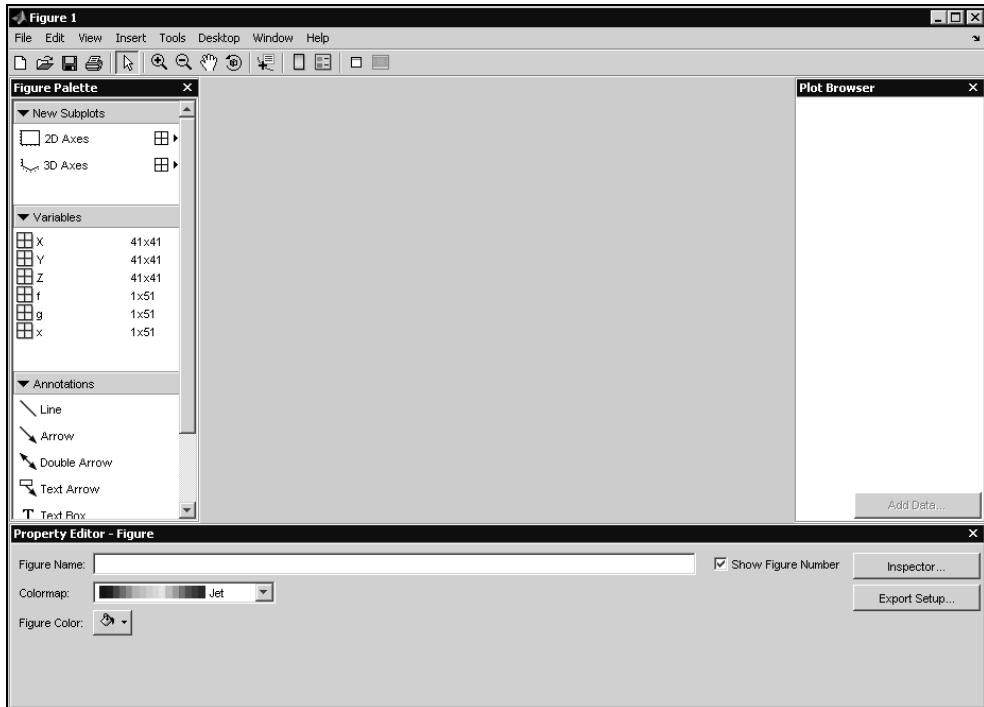


Рис. 4.2. Среда редактора графиков

### Примечание

Для перехода в среду редактора графиков мы вызвали команду `plottools`. Если в MATLAB не открыто ни одного графического окна, то в результате выполнения этой команды создается графическое окно со всеми вспомогательными элементами среды редактора графиков. Если же одно или несколько окон открыто, то режим редактирования включается для текущего графического окна. Быстрый переход в среду редактора графиков осуществляется нажатием кнопки **Show Plot Tools** на панели инструментов графического окна.

Окно шаблонов графики (**Figure Palette**) используется для размещения осей в пределах графического окна, задания данных для визуализации и размещения пояснений. Соответствующие части окна имеют подзаголовки: **New Subplots**, **Variables**, **Annotations**. Любая часть может быть свернута или открыта с помощью щелчка мышью на заголовке.

Для создания осей следует щелкнуть мышью по типу системы осей (**2D Axes** или **3D Axes**) в разделе **New Subplots** либо использовать кнопку "добавить"

области" ( для указания количества подобластей для построения графиков и способа их расположения в графическом окне. В нашем примере требуются две системы координат: двумерные оси для графиков тригонометрических функций и трехмерные — для поверхности. Последовательные щелчки мышью по **2D Axes** и **3D Axes** приводят к размещению нужных нам осей, причем при создании новых осей уже имеющиеся автоматически изменяют свои размеры так, чтобы избежать перекрытия. Сейчас оси находятся друг под другом. Для горизонтального расположения осей следовало бы вместо щелчка по **3D Axes** использовать кнопку "добавить области" и, применяя протаскивание мышью, указать желаемый способ добавления осей.

В режиме редактирования размеры и расположение осей изменяются при помощи мыши. Выделенные щелчком мыши оси заключаются в рамку с квадратными маркерами, при наведении курсора мыши на них он меняет форму на двустороннюю стрелку. Изменение размеров производится перемещением мыши с удержанием левой кнопки, причем маркеры в вершинах рамки служат для пропорционального изменения размеров, а маркеры в серединах сторон — для независимого изменения ширины и высоты. Для перемещения выделенных осей в пределах окна следует подвести курсор мыши к одной из координатных осей (курсор должен состоять из четырех стрелок) и применить перетаскивание мышью.

Заметьте, что при создании осей в окне браузера объектов (**Plot Browser**) появляется информация о созданных графических объектах. В окне редактора свойств (**Property Editor**) отображаются доступные для редактирования свойства текущего объекта. Далее мы обсудим изменение свойств графических объектов.

Раздел **Variables** содержит переменные рабочей среды, в том числе и созданные только что массивы  $x$ ,  $f$ ,  $g$ ,  $X$ ,  $Y$ ,  $z$ . ПРИступим к их визуализации. Постройте сначала зависимость  $f$  от  $x$  на двумерных осях. Для этого сделайте оси текущими, выделите массивы  $f$  и  $x$  в разделе **Variables** (при помощи мыши с удержанием  $<\text{Ctrl}>$ ) и выберите **plot(x,f)** в контекстном меню. На осах появился график синуса. Теперь требуется добавить график косинуса, которому соответствуют массивы  $g$  и  $x$ . Однако повтор описанных выше действий для пары  $g$  и  $x$  приведет к исчезновению графика синуса. Поэтому следует воспользоваться кнопкой **Add Data** в окне браузера объектов, либо одноименным пунктом контекстного меню осей. В обоих случаях появляется диалоговое окно **Add Data to Axes**, в раскрывающихся списках которого **X Data Source** и **Y Data Source** необходимо выбрать подходящие массивы рабочей среды, т. е.  $x$  и  $g$  соответственно. Раскрывающийся список **Plot Type** позволяет задать тип графика — в нашем случае следует оставить предлагаемый по умолчанию. После нажатия на **OK** график косинуса добавился на оси.

### Примечание

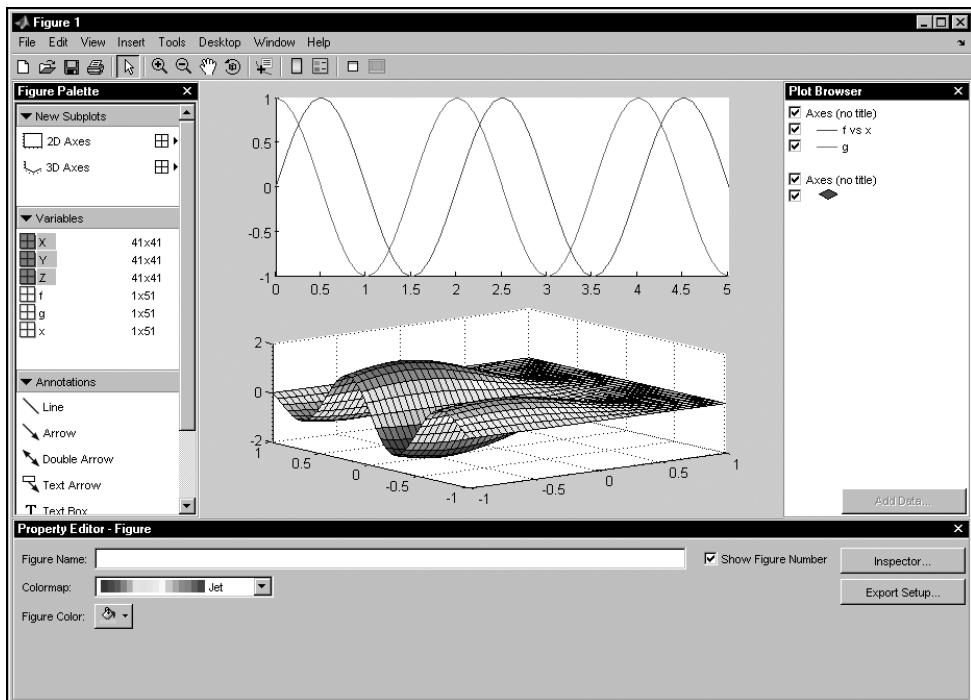
При создании одного графика можно предварительно не создавать систему осей. Например, при перетаскивании массива данных из области **Variables** в графическое окно редактора автоматически будет создан объект системы осей нужной размерности и изображен график. Аргументом функции будут индекс (индексы) элемента массива.

Перейдем теперь к построению графика поверхности, определяемой массивами  $x$ ,  $y$  и  $z$ . Сделайте текущими трехмерные оси и нанесите на них график поверхности одним из двух разобранных выше способов — при помощи раздела **Variables** либо кнопки **Add Data**. Заметьте, что в браузере объектов отображаются построенные линии (потомки двумерных осей) и поверхность (потомок трехмерных осей). Слева от названия каждого объекта находится флаг, сброс которого позволяет сделать объект невидимым. Например, если требуется получить поверхность, изображенную не на осях, а в самом окне, то следует сделать соответствующие оси невидимыми. Аналогичным образом скрываются линии и поверхности. При этом они существуют как графические объекты, и установка флага приводит к их отображению.

Для добавления объекта в группу выделенных объектов можно пользоваться как графическим окном (щелчок мышью с удержанием **<Shift>**), так и браузером (щелчок мышью с удержанием **<Ctrl>**). Разумеется, невидимые объекты можно выделить только в браузере. Для удаления выделенного объекта или группы служит клавиша **<Delete>**, либо пункт **Delete** контекстного меню. Кроме того, для удаления всех потомков осей проще всего очистить оси, выбрав в их контекстном меню пункт **Clear Axes**.

Потренируйтесь выделять объекты, делать их невидимыми и удалять, но затем приведите графическое окно в исходное состояние, изображенное на рис. 4.3, и сделайте его текущим объектом, щелкнув мышью по свободному месту окна. Сейчас редактор свойств отображает его настройки. Страна ввода **Figure Name** предназначена для определения заголовка графического окна. Например, введите текст "Редактор графиков". Сброс флага **Show Figure Number** обеспечивает удаление из заголовка слова **Figure** и номера окна. Выпадающий список **Colormap** служит для задания палитры графического окна, которая определяет способ закраски трехмерных поверхностей. Цвет фона графического окна изменяется при помощи кнопки **Figure Color**. Мы перечислили наиболее часто используемые настройки графического окна. Для доступа ко всем его свойствам придется прибегнуть к инспектору свойств, окно которого **Property Inspector** открывается при нажатии на кнопку **Inspector** в окне редактора свойств.

Обзору свойств графических объектов посвящена глава 9.



**Рис. 4.3.** Окно редактора графиков с графическими объектами

### Примечание

Окно редактора свойств всегда содержит средства для редактирования только часто изменяемых характеристик текущего графического объекта, а кнопка **Inspector** позволяет открыть окно инспектора свойств с полным перечнем свойств объекта. Часть элементов окна редактора свойств доступна из контекстного меню при щелчке правой кнопкой мыши на текущем объекте.

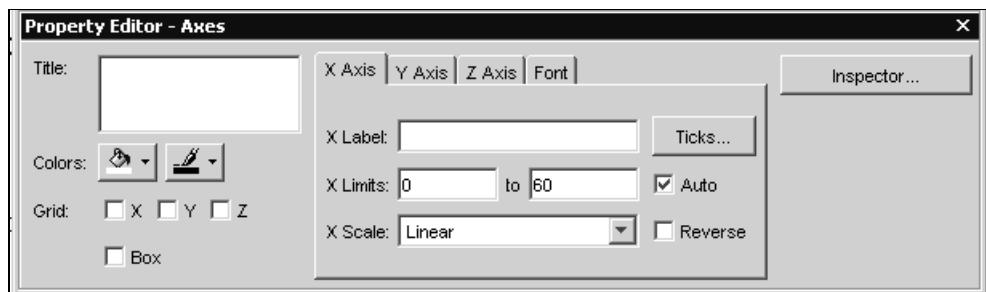
Редактор свойств отображает настройки текущего объекта — графического окна, осей, линий и поверхностей. Следовательно, перед редактированием любого объекта необходимо сделать его текущим.

Мы уже обсудили основные свойства графического окна и обратимся теперь к свойствам его потомков. В заключение этого раздела заметим, что для изменения свойств графических объектов существует несколько спосо-

бов. Один из них — использование редактора свойств, другой — контекстного меню объекта. Скажем, для задания цвета фона графического окна достаточно выбрать пункт **Color** его контекстного меню. Кроме того, ряд опций доступен из меню графического окна, например, меню **Edit** позволяет определить палитру (пункт **Colormap**), или удалить всех его потомков (пункт **Clear Figure**). Как правило, мы будем указывать одно из возможных решений, оставляя читателю возможность найти остальные и остановить свой выбор на том, которое кажется наиболее удобным.

## Свойства осей, подписи, заголовок

Как мы уже замечали, для перехода к свойствам объекта необходимо сделать его текущим. Сделайте, например, текущими двумерные оси (см. рис. 4.3), щелкнув мышью по свободному месту в пределах осей или выбрав их в списке браузера объектов. При использовании браузера объектов легко выбрать нужные оси, т. к. они содержат две линии, указанные в списке браузера. Выбранные оси выделяются рамкой с квадратными маркерами в графическом окне. Она не только свидетельствует о том, что оси стали текущим объектом, но и позволяет изменять их размеры, как было объяснено в предыдущем разделе.



**Рис. 4.4.** Диалоговое окно **Property Editor**  
для формирования осей

После выбора осей диалоговое окно редактора свойств соответствует свойствам осей, что отражено в его заголовке — **Property Editor - Axes** (рис. 4.4). Заметьте, что ряд свойств задается независимо для каждой из осей при помощи соответствующей вкладки **X Axis**, **Y Axis** или **Z Axis**.

Элементы управления окна редактора свойств обеспечивают доступ к наиболее часто используемым свойствам осей, которые мы сейчас опишем. Для обращения ко всем свойствам осей придется воспользоваться инспектором свойств, окно которого появляется при нажатии на кнопку **Inspector**. Далее мы приведем несколько примеров, когда обращение к инспектору свойств оказывается необходимым.

## Цветовое оформление, разметка и сетка

Для задания цвета фона и цвета линий осей следует воспользоваться соответствующими кнопками окна редактора свойств в группе **Colors** (см. рис. 4.4). Но при таком способе цвета всех линий осей совпадают. Для независимого выбора цвета каждой линии следует обратиться к инспектору свойств, который появляется при нажатии на кнопку **Inspector**. В левой части появившегося окна **Property Inspector** найдите свойства **XColor**, **YColor** и **ZColor**, которые отвечают за цвет каждой из осей, и выберите желаемый цвет, нажав на кнопку справа от названия свойства.

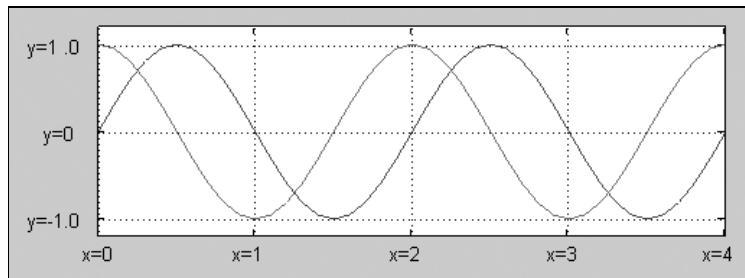
Обсудим допустимые настройки разметки осей, их масштаба и сетки. Масштаб каждой из осей (линейный или логарифмический) устанавливается независимо для каждой оси на соответствующей ей вкладке в раскрывающихся списках **X Scale**, **Y Scale** или **Z Scale**. На этих же вкладках задаются пределы каждой из осей в полях ввода **X Limits**, **Y Limits** и **Z Limits**. Для каждой из осей возможно либо обычное направление (флаг **Reverse** сброшен), либо обратное (флаг **Reverse** установлен). Для размещения координатной сетки, очевидно, служат флаги **X**, **Y** и **Z** группы **Grid**. Обратите внимание, что координаты линий сетки совпадают с координатами разметки, которая выбирается автоматически. Разберем этот вопрос более подробно. Имеется возможность задавать координаты разметки и подписи к ним для каждой оси. Для этого следует нажать кнопку **Ticks** и в появившемся диалоговом окне **Edit Axis Ticks** перейти к вкладке, соответствующей нужной оси. Размещенные на вкладке элементы управления позволяют задать равномерный шаг разметки (переключатель **Step by** и соответствующая строка ввода) или определить произвольные координаты. Для задания произвольных координат разметки следует установить флаг **Manual** в группе **X Tick Location** и добавить или удалить их в столбце **Locations** при помощи кнопок **Insert** или **Delete**. При этом существуют два способа подписей к разметке. Они либо являются значениями координат (при установленном переключателе **Auto** группы **X Tick Labels**), либо вводятся в столбце **Labels** (при вводе автоматически устанавливается переключатель **Manual**).

### Примечание

При маркировке осей не допускается использование формата TeX.

Кроме основной разметки, можно отобразить также более мелкую, установив флаг **Show minor ticks**. Соответственно, кроме редкой сетки, отвечающей основной разметке, может быть построена мелкая сетка. Здесь не обойтись без инспектора свойств, в котором следует установить свойствам **XMinorGrid**, **YMinorGrid** или **ZMinorGrid** значение **on** при помощи кнопки справа от названия свойства.

Потренируйтесь применять описанные выше средства редактора на примере двумерных осей рис. 4.3. Приведите их к виду, представленному на рис. 4.5.



**Рис. 4.5.** Масштабирование, пределы и разметка осей

Для установки требуемых свойств осей необходимо произвести следующие действия.

- Сделайте доступными поля **Limits** оси  $x$ , отключив флаг **Auto**, введите правый предел 4 для оси абсцисс и нажмите **<Enter>** для того, чтобы изменения вступили в силу. Аналогичным образом задайте пределы  $-1.2$ ,  $1.2$  по оси  $y$ .
- Добавьте сетку, включив флаги **Grid** для осей  $x$  и  $y$ .
- Откройте диалоговое окно **Edit Axis Ticks**, нажав кнопку **Ticks**. Установите переключатель **Step by** и внесите значение шага 1 в расположенную рядом с ним строку ввода. Используйте кнопку **Apply**, а не **OK**, поскольку в окне **Edit Axis Ticks** еще надо будет сделать ряд настроек.
- Задайте теперь подписи к разметке по оси  $x$ . Для этого установите переключатель **Manual** в группе **X Ticks Labels** и введите в каждой ячейке столбца **Labels** обозначения разметки  $x = 0$ ,  $x = 1$ ,  $x = 2$ ,  $x = 3$ ,  $Y(t_0)$ .

5. Разметьте ось  $y$  в соответствии с рис. 4.5, перейдя к вкладке **Y Axis** в окне **Edit Axis Ticks**.

Вкладка **Font** окна редактора свойств позволяет выбирать шрифт осей в раскрывающемся списке **Font Name**, изменить его размер при помощи списка **Size** и стиль. Левый список в группе **Style** предназначен для задания толщины шрифта, а правый — наклона. Цвет текста совпадает с цветом соответствующей координатной оси.

## Подписи и заголовок

Заголовок графика набирается в области ввода **Title** окна редактора свойств, а подписи к координатным осям задаются в строках ввода **Label** на вкладках со свойствами каждой из осей. После ввода подписи к координатной оси достаточно нажать **<Enter>** для ее появления в графическом окне. Заголовок же может состоять из нескольких строк, разделяемых **<Enter>**, поэтому для его отображения следует щелкнуть мышью по осям. После ввода заголовка в окне браузера надпись **Axes (no title)** изменится на текст заголовка, что облегчает выбор нужных осей для редактирования. Подписи к текущим осям и заголовок можно ввести при помощи меню **Insert** графического окна (пункты **X Label**, **Y Label**, **Z Label**, **Title**). Выбор соответствующего пункта меню приводит к появлению области ввода для заголовка или подписи к координатной оси прямо в графическом окне. Набор текста следует завершить щелчком мыши вне области ввода.

Заголовок может содержать формулы, задаваемые в форматах **TeX** и **LaTeX**. Дайте такой заголовок верхнему графику графического окна, как показано на рис. 4.6. Для получения требуемого заголовка следует набрать в области ввода **Title** строку в формате **TeX**: График функций  $\sin \pi/\sqrt{x}$  и  $\cos \pi/\sqrt{x}$  и щелкнуть мышью по осям. Появился заголовок — принадлежащий осям текстовый объект.

Правила записи математических формул в формате **TeX**, используемые в **MATLAB**, приведены в главе 3.

Подпишите координатные оси, например так, как на рис. 4.6.

Обсудим теперь основные возможности по редактированию заголовка и подписей к осям. Двойной щелчок мышью по этим объектам позволяет изменять текст прямо в графическом окне без обращения к редактору свойств.

Для перехода в редакторе свойств к основным свойствам текстовых объектов подписей и заголовка следует сделать их текущими при помощи щелчка мыши. В левой части содержатся раскрывающиеся списки для задания ха-

рактеристик текстового поля: стиля линии рамки вокруг текста (**Line Style**), толщины линии (**Line Width**), цвета фона (**Background**) и рамки (**Edge Color**).

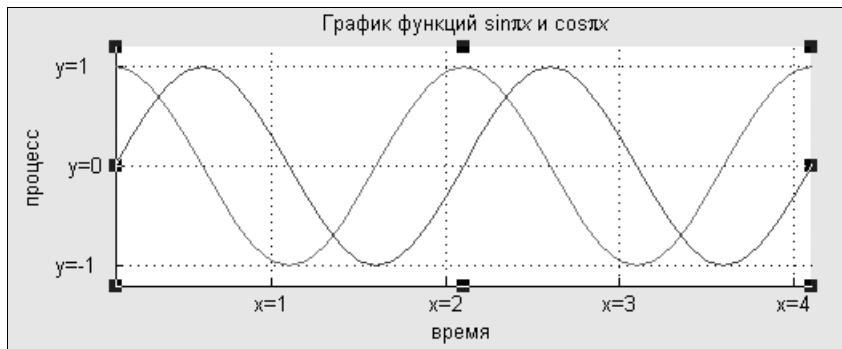


Рис. 4.6. График с заголовком и подписями осей

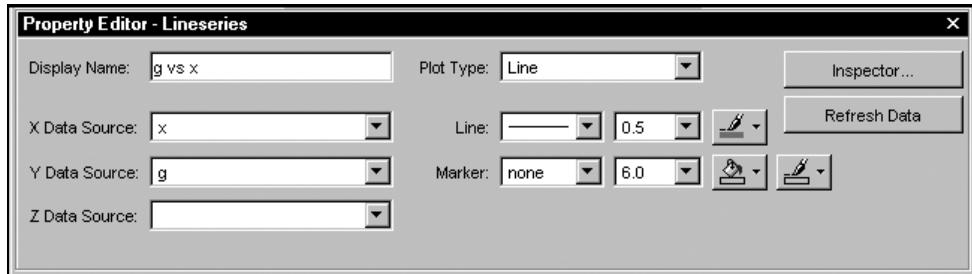
Остальные элементы управления позволяют изменять свойства шрифта. Список **Interpreter** служит для указания уровня поддержки стандарта TeX (об этом шла речь в предыдущей главе). Переключатель **Alignment** указывает способ выравнивания текста в поле. Под именем **Font** объединены кнопка задания цвета символов и четыре раскрывающихся списка для указания: имени шрифта, его размера, толщины и стиля. Большинство перечисленных установок можно сделать и из всплывающего меню текстового объекта. Дополнительные возможности форматирования текста становятся доступными в окне инспектора свойств. Например, угол поворота текста в градусах указывается при помощи его свойства **Rotation**.

## Свойства линий и поверхностей

Редактор свойств позволяет легко установить стиль, цвет и толщину линий и маркеров по своему усмотрению, изменять свойства образующих поверхности элементов, а также выбирать различные способы визуализации одномерных и двумерных массивов.

### Свойства линий

Перейдите к свойствам какой-нибудь из линий верхнего графика, приведенного на рис. 4.6, сделав ее текущим объектом щелчком мыши. Окно редактора свойств приобретет вид, изображенный на рис. 4.7.



**Рис. 4.7.** Диалоговое окно **Property Editor**  
для изменения свойств линий

В левой части окна редактора свойств расположены:

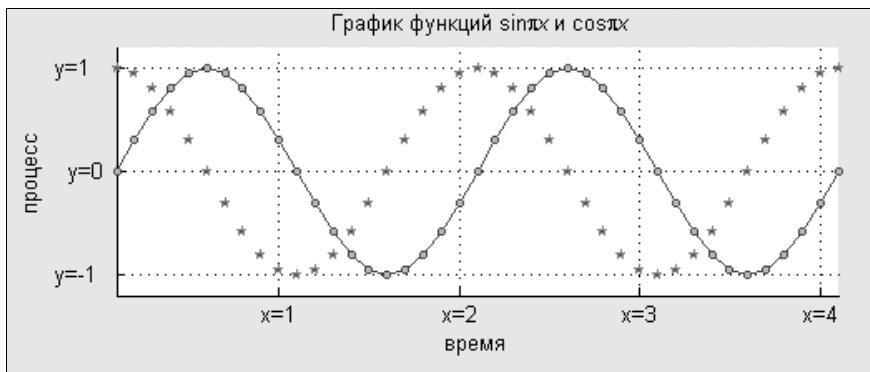
- одно поле ввода **Display Name**, предназначенное для изменения имени линии, которое отображается в легенде при ее добавлении;
- три однотипных раскрывающихся списка (**X Data Source**, **Y Data Source**, **Z Data Source**), позволяющие выбрать массивы данных из рабочей среды для построения графика. Опция **auto** для независимой переменной означает, что в качестве аргумента используется индекс массива.

После выбора данных следует нажать кнопку **Refresh Data** для того, чтобы обновление вступило в силу.

- Инструменты правой части окна редактора связаны со способом визуализации данных и видом графика.
- Раскрывающийся список **Plot Type** служит для переопределения типа графика. Например, для одномерного массива можно изменить обычный график (**Line**) на столбцовую диаграмму (**Bar**).
- Три раскрывающихся списка в поле **Line** определяют вид линии (сплошная, штриховая, пунктирная, штрихпунктирная), ее толщину в пунктах (1пт = 1/72 дюйма) и цвет.
- Четыре раскрывающихся списка в поле **Marker** используются для задания типа маркера, его размера в пунктах, цвета границ и внутренности (только для полых маркеров).

При помощи этих элементов управления легко изменить вид линии графика по своему усмотрению. Пункты **No line (none)** и **No marker (none)** раскрывающихся списков для выбора стиля линии и маркеров означают их отсутствие на графике. Ряд настроек может быть также сделан из контекстного меню линий.

Измените свойства линий и маркеров графиков, изображенных на рис. 4.6, так, чтобы получившиеся графики имели вид, приведенный на рис. 4.8.



**Рис. 4.8.** Изменение стиля линий и маркеров

Вам предстоит выполнить следующие действия.

- Сделайте текущим график косинуса, щелкнув по линии графика левой кнопкой мыши для доступа к ее свойствам в редакторе.
- В раскрывающемся списке поля **Line** для типа линии установите значение **no line**.
- В раскрывающемся списке поля **Marker** выберите тип маркера в форме пятиконечной звезды и задайте в соседних списках красный цвет для границы и синий для внутренности маркера.
- Задайте размер маркера 5 пт (в списке такого значения нет, поэтому введите его с клавиатуры).
- Сделайте текущим график синуса.
- Измените цвет линии на синий.
- Установите тип маркера в форме кружка, размер 4 пт, задайте желтый цвет для внутренности маркера и синий для контура.

Перейдем теперь к изучению свойств поверхности. Поскольку сетка поверхности состоит из линий, то при чтении следующего раздела вы встретитесь с аналогичной настройкой вида этих линий.

## Свойства поверхностей

В графическом окне, приведенном на рис. 4.3, сделайте поверхность текущей и обратитесь к окну редактора свойств. Ребра сетки поверхностей имеют те же свойства, что и линии двумерных графиков, рассмотренные в предыдущем разделе. Инструменты для настройки их вида содержатся в разделах **Edges** и **Markers**. Они предназначены для одновременного изменения или удаления линий, параллельных как оси абсцисс, так и ординат. Если же требуется оставить линии только одного из направлений, то понадобится инспектор свойств. В нем следует найти свойство **MeshStyle** и установить его значение в **row** для отображения линий сетки, параллельных оси *x*, или **column** — для оси *y*.

Цвет заливки ячеек поверхности задается в поле **Faces**. Раскрывающийся диалог позволяет отказаться от закраски граней (опция **No Face Color**), либо выбрать:

- один из цветов, общий для всех ячеек;
- свой цвет каждой ячейки в зависимости от значения функции, или от освещения поверхности, если для ее построения была выбрана функция **surf** (опция **Faceted**);
- плавное изменение цвета (опция **Blended**).

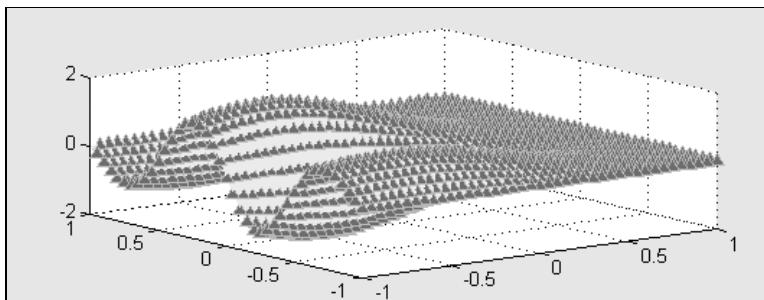
Результат выбора последних двух опции зависит также и от цветовой палитры графического окна. Возможно и явное указание цвета при помощи матрицы в раскрывающемся списке **C Data Source**, но этот способ мы рассмотрим при детальном изучении свойств графических объектов (см. главу 9 "Дескрипторная графика").



Рис. 4.9. Диалоговое окно **Property Editor**  
для изменения свойств поверхностей

С существующей поверхностью могут быть связаны любые подходящие по размерам массивы рабочей среды, которые указываются в списках **X Data Source**, **Y Data Source** и **Z Data Source**. После их выбора требуется нажать кнопку **Refresh Data** для обновления графика.

В качестве упражнения измените приведенную на рис. 4.3 поверхность в соответствии с рис. 4.10.



**Рис. 4.10.** Изменение свойств поверхности

Для получения требуемого результата следует выполнить такие действия.

1. Определите тип маркера треугольник ( $\Delta$ ).
2. Установите размер маркера, равный 6 в списке, задающем его размер.
3. Задайте зеленый цвет заливки маркера.
4. Задайте розовый цвет контура маркера.
5. Отмените изображение ребер поверхности (значение **no line** для типа линии в разделе **Edges**).
6. Выберите серый цвет в списке **Faces**.

## Дополнительные элементы оформления

Размещение дополнительной информации на графике осуществляется при помощи стрелок, линий, надписей, легенды и шкалы палитры (colorbar), которая устанавливает соответствие между цветом участков поверхности и значением функции. Выбор нужного объекта производится в меню **Insert** графического окна из раздела **Annotation** окна шаблонов графики (**Figure Palette**) или при помощи соответствующих инструментов панелей графических окон **Figure Toolbar** и **Plot Edit Toolbar**, которые добавляются или уби-

раются в меню **View**. Фактически любой дополнительный элемент может быть размещён одним из указанных способов, поэтому в дальнейшем будем пояснить использование одного из инструментов для отдельных объектов. Для того чтобы добавить стрелку (линию), следует в меню **Insert** выбрать пункт **Arrow (Line)** и нарисовать ее мышью при нажатой левой кнопке. Находясь в режиме редактирования, можно перемещать в пределах графического окна все объекты, включая стрелки и линии, и изменять их размеры (используя маркеры выделения границ) при помощи движения мыши, удерживая нажатой левую кнопку. В режиме редактирования выделенный графический объект удаляется из контекстного меню выбором пункта **Cut**, или нажатием на клавишу <Delete>.

В любое место в пределах графического окна можно поместить надпись, для чего следует, например, в разделе **Annotation** окна шаблонов графики (**Figure Palette**) выбрать элемент **Text Box** и щелкнуть левой кнопкой мыши в рабочем поле графического окна. Появится поле для ввода текста. При наборе текста допустимо использование форматов **TeX** и **LaTeX**, 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 редактировать его содержимое.

При необходимости нанести на рисунок текстовый комментарий со стрелкой, лучше использовать элемент **Text Arrow**, объединяющий стрелку и поясняющий текст.

Добавьте стрелки и подписи на график так, как показано на рис. 4.11.

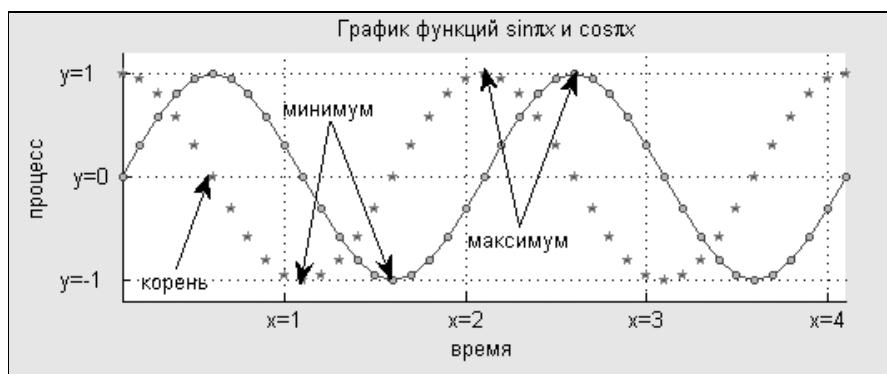
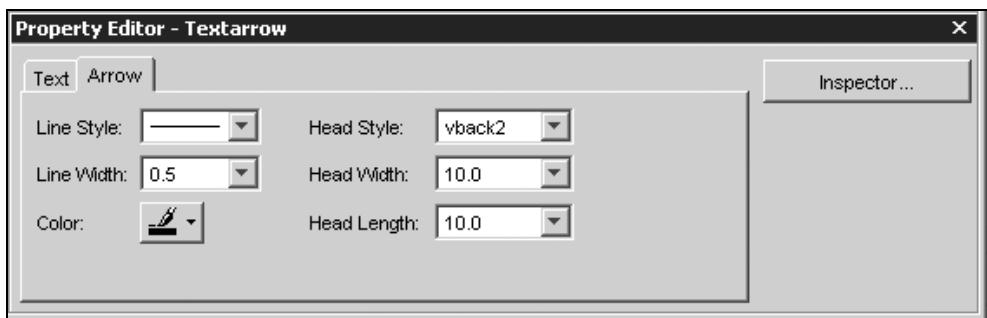


Рис. 4.11. Добавление стрелок и надписей

При обозначении точек минимума и максимума в приведенном варианте можно использовать либо два объекта (**Text Arrow** и **Arrow**), либо три объекта (**Text** и два элемента **Arrow**).

Выделение линии, стрелки или текстового объекта приводит к автоматическому отображению их свойств в диалоговом окне **Property Editor**. Например, для объекта **Text Arrow** окно редактора свойств содержит две вкладки (рис. 4.12). Одна из них, **Text**, относится к тексту, ее состав рассмотрен выше в разд. "Подписи и заголовок" этой главы. Вторая, **Arrow**, содержит поля для определения свойств линии и указателя (наконечника) стрелки: список **Head Style** определяет форму указателя, а два других — **Head Width** и **Head Length** — линейные размеры (ширину и длину).



**Рис. 4.12.** Диалоговое окно **Property Editor**  
для объекта **Text Arrow**

Если оси содержат графики нескольких функций одной переменной, то их полезно сопроводить легендой, содержащей образцы линий и подписи к ним. В качестве подписи выбирается значение одного из свойств линии — ее имени, которое задается в поле **Display Name** редактора свойств. Дайте имена **sin** и **cos** линиям наших графиков синуса и косинуса (см. рис. 4.3) и добавьте легенду, например, из контекстного меню осей, выбрав в нем пункт **Show Legend**, или при помощи меню **Insert** графического окна или панели инструментов. Теперь график приобретает вид, изображенный на рис. 4.13.

Когда текущим объектом является легенда, окно редактора свойств позволяет изменить ее свойства (рис. 4.14). Элемент **Location** служит для выбора места легенды в пределах осей или вне их. Установка переключателя **best** приводит к выбору наилучшего положения легенды с наименьшим перекрытием других элементов. Впрочем, легенда может быть перемещена в любое место графического окна при помощи мыши. Раскрывающийся список **Orientation** предназначен для определения способа отображения элементов

легенды: в строку (опция **Horizontal**), или в столбец (опция **Vertical**). Инструменты поля **Color** позволяют назначить цвета рамки и фона, а **Font** — характеристики шрифта легенды.

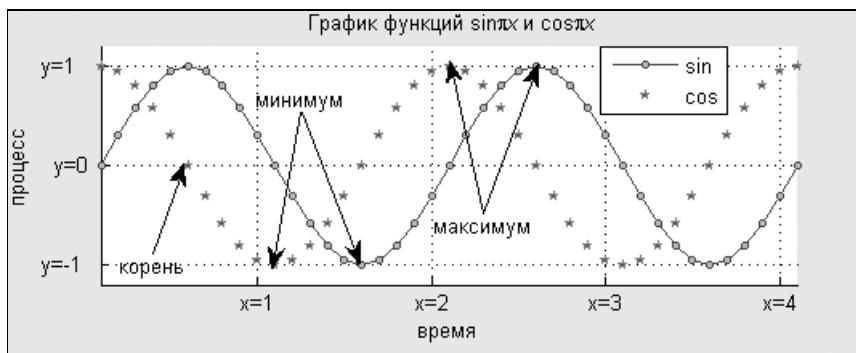


Рис. 4.13. Добавление легенды

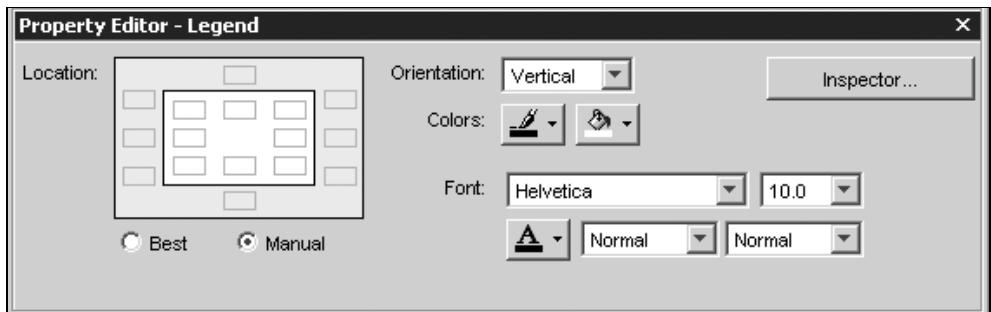


Рис. 4.14. Диалоговое окно **Property Editor**  
для объекта **Legend**

Трехмерные графики, построенные при помощи функций `mesh` и `surf`, полезно снабдить шкалой палитры, т. е. информацией о соответствии цвета поверхности значению функции. Наиболее просто ее отобразить при помощи кнопки **Insert Colorbar** панели инструментов **Figure Toolbar** графического окна, или пункта **Colorbar** меню **Insert**. В любом случае это приводит к размещению вертикальной шкалы с цветовой гаммой справа от выделенной пары осей с графиком поверхности (эквивалент команды `colorbar`). В окне редактора свойств доступен элемент управления **Location**, такой же, как и для легенды, который позволяет выбрать положение и ориентацию шкалы.

Контекстное меню легенды дает ряд дополнительных возможностей по сравнению с редактором свойств. Пункт **Standart Colormaps** позволяет быстро изменить палитру графического окна на одну из предопределенных. Эти палитры легко могут быть изменены по своему желанию одним из двух способов. Во-первых, выбор пункта **Interactive Colormap Shift** приводит к изменению вида курсора мыши, движение которого вдоль шкалы модифицирует набор цветов или оттенков для закраски поверхности. Во-вторых, для более тонкого редактирования палитры имеется специальный редактор **Colormap editor** (рис. 4.15), который запускается выбором пункта **Launch Colormap Editor** контекстного меню.

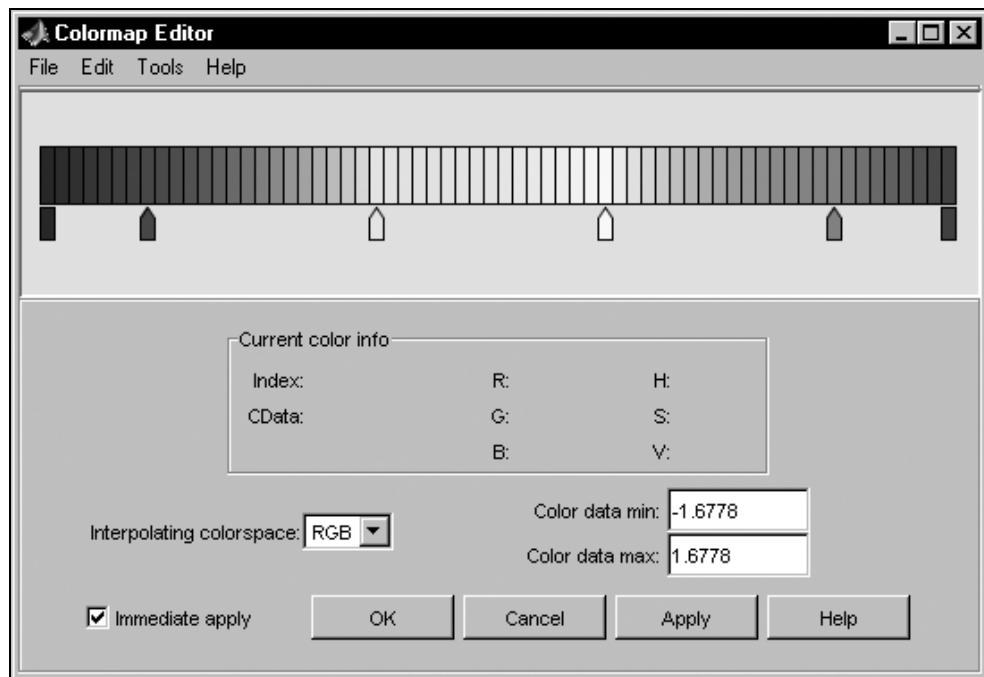


Рис. 4.15. Редактор палитры

Работа в редакторе палитры достаточно проста. Указатели, расположенные под шкалой, служат для задания цвета текущего участка, который выбирается двойным щелчком мыши по указателю. Для добавления указателя следует щелкнуть под нужным участком шкалы, а для удаления — выделить его и выбрать в меню **Edit** пункт **Delete**. Выбор одной из стандартных палитр

производится в меню **Tools**, например, **white** (только белый цвет) хорошо подходит для создания собственной палитры.

Для поверхностей можно дополнительно задавать источники освещения, используя пункт **Light** меню **Insert** графического окна. В редакторе свойств этого объекта указываются: цвет источника освещения **Color**, способ освещения — **Style** и три координаты точки расположения источника света в системе осей — **Position**. Способ освещения **Infinite** означает, что пучок света состоит из параллельных лучей (в этом случае координаты источника игнорируются), а **Local** — лучи радиально исходят из точечного источника света, находящегося в заданной позиции.

## Обзор графиков и поверхностей

Инструменты графического окна позволяют увеличивать и уменьшать графики линий и поверхностей, перемещаться по ним, интерактивно получать значение функции в выбранной точке графика, осматривать поверхности с различных сторон.

### Изменение масштаба, определение значений функции, поворот

На панели **Figure Toolbar** расположены три инструмента, позволяющие более детально изучить график. Это инструменты уменьшения (**Zoom out**) или увеличения масштаба (**Zoom in**) и перемещения по графику (**Pan**). Например, если вы выбрали режим увеличения масштаба **Zoom in**, то в области графика курсор мыши приобретает форму, изображенную на пиктограмме соответствующего инструмента. Вы можете либо протаскиванием мыши при нажатой левой кнопке определить область для просмотра, либо щелкнуть в ту точку графика, которая после изменения масштаба должна быть в центре. Контекстное меню, вызываемое правой кнопкой, позволяет установить параметры масштабирования (**Zoom Options**), вернуться к первоначальному масштабу (**Reset to Original View**) или к предыдущему (**Zoom Out**). Если установлен режим перемещения по графику, то при нажатой левой кнопке мыши вы можете передвигать график без изменения масштаба.

Дополнительные элементы (стрелки, надписи, легенда и т. п.) не перемещаются вместе с графиком, поскольку они не "привязаны" к осям. Однако средствами редактора можно зафиксировать координаты некоторой точки объекта в системе координат осей графика. Предположим, что график снабжен стрелкой (объект **arrow**), которая при изменении масштаба или пе-

ремещении графика должна вести себя так же, как и сам график. Для этого следует выбрать в меню **Tools** пункт **Pin to Axes** (при этом курсор мыши изменит форму — указателем является тонкий конец) и щелкнуть левой кнопкой мыши по той точке стрелки, которую следует привязать к осям. Если таких точек две, то снова выберите пункт **Pin to Axes** и отметьте вторую точку. Следите, чтобы при этом выбранный объект остался текущим, ибо его смена будет означать, что привязка объекта не выполнена. Проверьте, что стрелка теперь перемещается, увеличивается и уменьшается вместе с графиком. Для привязки всей стрелки, как и других поясняющих объектов, служит пункт **Pin to Axes** контекстного меню объекта.

MATLAB предоставляет удобное средство для интерактивного определения координат точек на графиках линий и поверхностей и для размещения соответствующих ярлыков со значениями независимых переменных и функции. Выберите инструмент **Data Cursor** и щелкните левой кнопкой мыши по точке на графике — появляется ярлык с координатами точки. Перемещение мыши с удержанием левой кнопки вдоль графика приводит к перемещению ярлыка и обновлению информации в нем. Контекстное меню ярлыка позволяет управлять процессом отображения, например, можно добавить еще одно окно, выбрав пункт **Create New Datatip** (тогда предыдущее остается связанным с последней точкой), или удалить все образованные окна — **Delete All Datatips**.

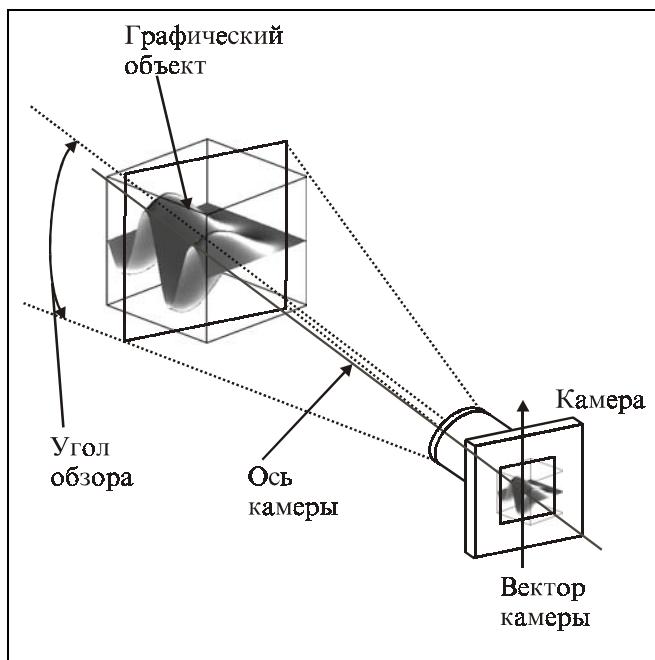
Еще один инструмент **Rotate 3D** панели **Figure Toolbar** графического окна служит для поворота поверхности движением мыши при нажатой левой кнопке. Во время вращения в левом нижнем углу графического окна отображаются текущие значения азимута (*Az*) и угла склонения (*El*), смысл которых мы обсуждали при использовании функции `view` для задания точки обзора (см. разд. "Поворот графика, изменение точки обзора" главы 3).

MATLAB предоставляет более развитое средство для просмотра трехмерных объектов — камеру, возможностям которой посвящены следующие два раздела.

## Камера для обзора графического объекта

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, осуществляющую при помощи некоторой камеры (Camera в

MATLAB). Рисунок 4.16 поясняет взаимное расположение камеры и графического объекта.



**Рис. 4.16.** Взаимное расположение камеры и графического объекта

Работа с камерой определяется следующими параметрами.

□ Режим отображения удаленного объекта. В пакете предусматривается использование двух режимов:

- `orthographic`, размеры элементов объекта зависят только от расстояния от объекта до камеры, причем не происходит искажения (параллельные прямые остаются параллельными), т. е. оси координат остаются ортогональными;
- `perspective`, изображение элементов объекта зависит как от расстояния от объекта до камеры, так и от размера самого элемента, удаленные части мельче близлежащих (параллельные прямые могут изображаться непараллельными), т. е. оси координат в перспективе.

- Угол обзора (**View angle**) в градусах, больший нуля и меньший либо равный 180. Изменение угла обзора влияет на размер графического объекта на экране, а перспектива осей не претерпевает изменений.
- Положение камеры относительно объекта (**Position**). Положение камеры в системе координат осей задается вектором с координатами  $[x, y, z]$ . Приближение камеры к объекту при фиксированном значении угла обзора **View angle** позволяет увеличить масштаб просмотра, отдаление камеры приводит к уменьшению масштаба.
- Положение графического объекта (**Target**). Точка с координатами  $[x, y, z]$ , на которую направлена камера. Вместе с **Position** это свойство определяет ось камеры, приведенную на рис. 4.16.
- Поворот камеры вокруг оси просмотра (**Up vector**), задается координатами  $[x, y, z]$  вектора камеры (рис. 4.16).

## Панель инструментов камеры

Быстрый доступ к свойствам камеры производится из панели инструментов **Camera Toolbar**, которая появляется при установке флага **Camera Toolbar** в меню **View** графического окна. Панель **Camera Toolbar** состоит из нескольких групп. Группа инструментов, 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

В эту группу 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 своей оси вращением мыши по (или против) часовой стрелке в пределах графического окна.

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 выбрать тип движения камеры или источника света, и затем, перемещением указателя мыши в пределах графического окна при нажатой левой кнопке мыши, осуществить желаемое действие.

Часть инструментов управления движением камеры: **Orbit Camera**, **Pan/Tilt 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 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** и представленных на рис. 4.19, можно:

- добавить или удалить один источник света (**Toggle Scene Light**);
- установить ортогональную проекцию осей на экран (**Orthographic Projection**);

- отобразить оси в перспективе (**Perspective Projection**);
- вернуть графику первоначальный вид (**Reset Camera and Scene Light**);
- остановить движение графика (**Stop Camera/Light Motion**), что может быть полезно, если вы задали слишком много перемещений движением мыши и MATLAB долго обрабатывает изменение графического окна.



**Рис. 4.19.** Дополнительные инструменты работы с камерой

## Сохранение, экспорт и печать

Для сохранения графического окна используются пункты **Save** или **Save as** меню **File** графического окна. MATLAB сохраняет графическое окно в файле с расширением `fig`. Открыть графическое окно в текущем и следующих сеансах работы с MATLAB можно при помощи пункта **Open** меню **File** любого графического окна или рабочей среды MATLAB (при открытии из рабочей среды требуется выбрать `fig` в фильтре расширений диалогового окна **Open**).

Экспорт графики из MATLAB производится в различные графические форматы, в частности: EPS, AI, BMP, GIF, TIFF, JPEG и др. При экспорте в графический файл записывается только область графического окна без меню и панели инструментов. Для экспортирования предназначен пункт **Save as** меню **File** при выборе типа файла, отличного от `fig`.

Кроме сохранения содержимого графического окна как файла с расширением `fig` или графического файла, имеется и альтернативная возможность — автоматическое создание соответствующего кода на языке программирования MATLAB, выполнение которого приведет к появлению графика. Для получения такого кода в меню **File** графического окна следует выбрать пункт **Generate M-File**, после чего начнется генерация кода. Текст кода откроется в редакторе M-файлов, и его можно сохранить и запустить на выполнение. Мы уделим достаточно внимания программированию в MATLAB, в частности, следующие две главы посвящены конструкциям встроенного

языка программирования и написанию собственных программ. Поэтому мы отложим обсуждение возможности генерации М-файла до следующей главы. Еще один пример создания М-файла приведен в в разд. "Объекты *Rectangle* и *Line*, блок-схемы и диаграммы" главы 9.

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. 4.20.

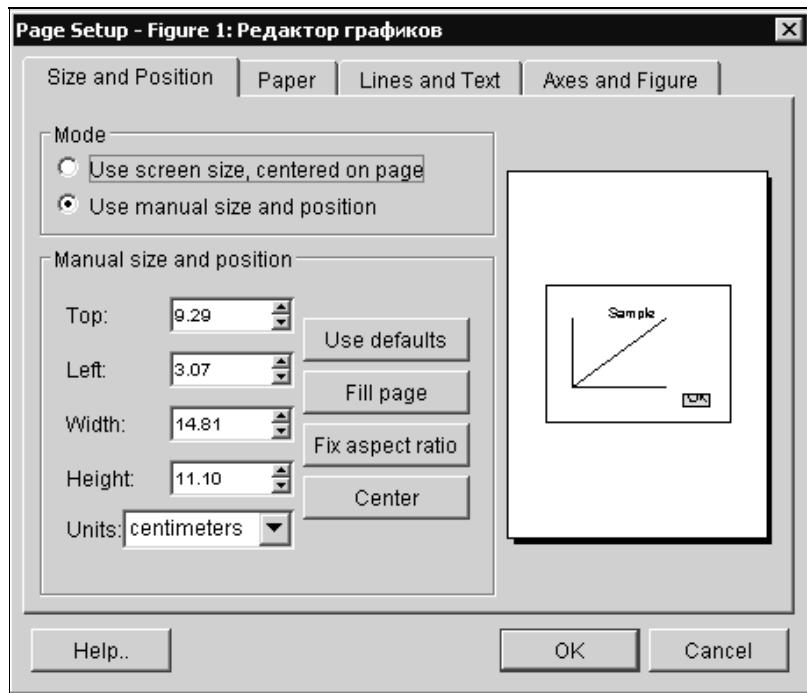


Рис. 4.20. Диалоговое окно **Page Setup**

Задание размеров графика и его положения на странице осуществляется при помощи элементов управления, расположенных на вкладке **Size and Position**, причем эта вкладка содержит образец страницы. По умолчанию установлен 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 управле-

ния, 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 **Fill Page**. 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, так же 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 выберете **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 (установлен переключатель **Force white background**). Если же требуется напечатать график на фоне графического окна, то следует установить переключатель в положение **Keep screen background color**.

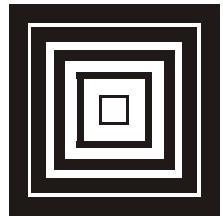
Диалоговое окно для установки параметров принтера доступно из пункта **Print Setup** меню **File** графического окна, а диалоговое окно печати — из пункта **Print**. После установки параметров страницы перед печатью полезно воспользоваться предварительным просмотром при помощи пункта **Print Preview** меню **File** графического окна.

Более полную информацию можно получить, обратившись к разделам справочной системы из меню **Help** графического окна.

## Задания для самостоятельной работы

Постройте поверхности из задания к главе 3. Используя средства редактора свойств **Property Editor**, оформите их графическое представление:

- задайте характеристики осей;
- подберите цветовую палитру для лучшего отражения свойств поверхности;
- поварырийте свойства граней и маркеров точек сетки;
- воспользуйтесь камерой для всестороннего осмотра и изучения поверхности.



## Глава 5

# М-файлы

В предыдущих главах мы рассмотрели достаточно простые примеры, для выполнения которых требуется набрать несколько команд в командной строке. Для более сложных задач число команд возрастает, и работа в командной строке становится непродуктивной. Использование истории команд, сохранение переменных рабочей среды или ведение дневника при помощи `diary` незначительно повышают производительность работы. Эффективное решение состоит в оформлении собственных алгоритмов в виде программ (М-файлов), которые можно запустить из рабочей среды или из редактора. Встроенный в MATLAB редактор М-файлов позволяет не только набирать текст программы и запускать ее целиком или частями, но и отлаживать алгоритм. Подробная классификация М-файлов приведена ниже.

Отладка программ описана в *главе 8*.

## Работа в редакторе М-файлов

Раскройте меню **File** рабочей среды MATLAB и в пункте **New** выберите подпункт **M-file** или нажмите кнопку **New M-file** на панели инструментов рабочей среды. Новый файл открывается в окне редактора М-файлов, которое приведено на рис. 5.1.

Вид строки меню и панели инструментов зависит от ширины окна. Если оно достаточно узкое, то часть инструментов перемещается в раскрывающийся список.

Наберите в редакторе команды для построения двух графиков на разных осях в одном графическом окне. Не обязательно набирать много команд — наша цель сейчас состоит в том, чтобы научиться выполнять команды из

редактора M-файлов. Ограничтесь командами, приведенными в листинге 5.1.

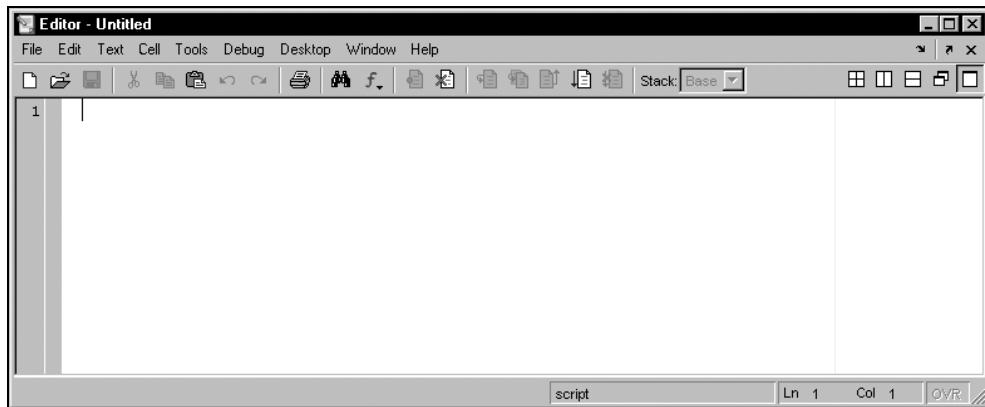


Рис. 5.1. Окно редактора M-файлов MATLAB

#### Листинг 5.1. Команды для построения графиков

```
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 в меню **File** редактора пункт **Save as**. Для запуска на выполнение *всех команд*, содержащихся в файле, следует выбрать пункт **Run** в меню **Debug**, или просто нажать <F5>. На экране появится графическое окно **Figure 1**, содержащее графики функций. Результат эквивалентен последовательному выполнению команд листинга 5.1 в командном окне. Однако если вы решили построить график косинуса вместо синуса, то достаточно просто изменить оператор присваивания  $g = \sin(x)$  в M-файле на  $g = \cos(x)$  и запустить из редактора все команды. Аналогичные действия из командной строки потребовали бы больше времени.

### Примечание

Если вы создали новый файл в редакторе и набрали в нем команды, то необязательно сначала сохранять его из меню **File**, а только потом выполнять. Можно сразу раскрыть меню **Debug**. Пункт **Run** в этом случае заменяется на **Save and Run**, он позволяет запустить программу, предварительно сохранив ее.

Очень удобной возможностью редактора М-файлов является *выполнение части команд*. Закройте графическое окно **Figure 1**. Выделите при помощи мыши, удерживая левую кнопку, первые четыре команды листинга 5.1. Затем откройте контекстное меню правой кнопкой мыши и выберете пункт **Evaluate Selection**. То же самое можете сделать, используя клавиатуру: клавишами со стрелками при нажатой **<Shift>** выделите первые четыре команды листинга 5.1 и выполните их из пункта **Evaluate Selection** меню **Text** или нажмите клавишу **<F9>**. Обратите внимание, что в графическое окно вывелся только один график, соответствующий выполненным командам. 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.

Другая возможность для выполнения фрагмента М-файла рассматривается далее в разд. "Разбиение М-файла на ячейки" этой главы.

Если в М-файле при наборе сделана ошибка, то она выявляется в процессе исполнения. MATLAB выполняет команды до неправильно введенной, после чего в командное окно выводится сообщение об ошибке. Создайте в редакторе новый файл *mydemo2.m*, например, при помощи кнопки **New M-file** панели инструментов редактора, со следующими командами:

```
y = [1 2 3]
z = y*y
x = y
```

Очевидно, что во второй строке допущена ошибка. Попытка выполнения такого файла приведет к выводу в командное окно следующего сообщения с гиперссылкой на место ошибки:

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
Error in ==> mydemo2 at 2
z = y*y
```

Щелчок мыши по гиперссылке с именем М-файла делает окно редактора активным и помещает курсор в строку с ошибкой.

## Примечание

Выполняемые команды осуществляют вывод результата в командное окно. Например, в предыдущем примере вывелоось значение  $y$ . Для подавления вывода следует завершать команды точкой с запятой.

Отдельные блоки М-файла (особенно большого размера) целесообразно 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 (по умолчанию), например:

```
%построение графика sin(x) в отдельном окне
```

Для исключения части исполняемого кода без его удаления или если количество строк комментариев достаточно велико, можно использовать блок комментариев, начинающийся со строки из двух символов `%{` (знака процента и открывающейся фигурной скобки) и заканчивающийся строкой из двух символов `%}` (закрывающейся фигурной скобки и знака процента).

Открытие существующего М-файла производится при помощи пункта **Open...** меню **File** рабочей среды либо редактора М-файлов. Открыть файл в редакторе можно и командой `edit` из командной строки, указав в качестве аргумента имя файла, например:

```
>> edit mydemo
```

Команда `edit` без аргумента приводит к открытию редактора и созданию нового файла без имени (**Untitled**). Если вы ввели команду с именем несуществующего файла `mydemo4`:

```
>> edit mydemo4
```

то MATLAB воспримет это как желание создать новый М-файл с указанным именем. На экран будет выведено диалоговое окно с запросом: "File mydemo4.m does not exist. Do you want to create it?" (Файл не существует, хотите ли вы создать такой файл?). При выборе **Yes** файл будет создан и откроется в редакторе М-файлов.

## Примечание

В этом же окне имеется флаг **Do not show this prompt again**, установка которого отключит появление запроса на создание нового файла. В этом случае указание имени несуществующего файла в качестве параметра команды `edit` сразу приведет к его созданию. Вы можете восстановить появление этого запроса, обратившись к настройкам редактора. Для этого

выберите в меню **File** рабочей среды или редактора M-файлов пункт **Preferences**. Появляется одноименное окно, в левом поле которого следует перейти к пункту **Editor/Debugger** и установить флаг **Show dialog prompt when editing files that do not exist** в правом поле.

В редакторе M-файлов может быть одновременно открыто несколько файлов. MATLAB позволяет менять способ отображения файлов в редакторе. По умолчанию окно **Editor** редактора только одно, и при открытии каждого нового файла оно снабжается закладкой внизу рабочей области с именем файла для быстрого перехода к окну с требуемым файлом или для его закрытия кнопкой на закладке. Последние пять кнопок на панели инструментов (см. рис. 5.1) дают возможность выбрать способ расположения окон с файлами в рабочей области редактора. Например, ее можно разделить по горизонтали или по вертикали для отображения двух файлов. Используя технику **dock/undock**, можно открыть каждый файл в своем окне редактора или, наоборот, встроить его в другое окно редактора. Для выполнения указанных операций в правой части строки меню присутствуют соответствующие инструменты **Dock** и **Undock**. Если окно редактора единственное, то использование **Dock** приводит к встраиванию окна редактора M-файлов в окно рабочей среды MATLAB, при этом кнопка **Undock** дополнительно появляется в заголовке окна редактора для получения обратного эффекта.

## Настройки редактора M-файлов

Для изменения настроек редактора M-файлов следует выбрать в меню **File** редактора или рабочей среды пункт **Preferences**. Появляется одноименное диалоговое окно для настройки ряда компонент рабочей среды MATLAB. В левой части окна отображены названия компонент, часть которых представлена раскрывающимся списком (слева находится знак +), позволяющим перейти к требуемой группе свойств. При изменении опций той или иной компоненты следует выбрать ее в списке и перейти к элементам управления в правой части окна **Preferences**. Среди компонент есть и редактор M-файлов — раскрывающийся список **Editor/Debugger**. Рассмотрим далее наиболее важные настройки редактора.

При выборе заголовка раскрывающегося списка **Editor/Debugger** в правой части окна отображаются общие настройки, связанные с редактированием файлов в MATLAB. Панель **Editor** позволяет использовать вместо стандартного редактора MATLAB любой другой текстовый редактор, скажем, Notepad (Блокнот). Причем создание нового M-файла будет осуществляться по-прежнему в редакторе M-файлов, а открытие файлов — в выбранном

редакторе. К общим настройкам относится также длина списка последних открытых файлов (**Number of entries**), который располагается в меню **File** редактора или рабочей среды. При запуске MATLAB возможно автоматическое открытие тех файлов, с которыми велась работа во время предыдущей сессии, если при завершении работы редактор не был отдельно закрыт. Для этого следует установить флаг **On restart reopen files from previous MATLAB session**.

Сделайте активным пункт **Display** в левой части окна. В правой части окна появятся средства для изменения режимов. Например, в разделе **General Display Options** расположены два флага. Флаг **Show line numbers** установлен и указывает на то, что в рабочей области выделена колонка для нумерации строк текста в файле, а флаг **Enable data tips in edit mode** сброшен. Его установка позволяет вывести значение переменной рабочей среды на всплывающую подсказку при наведении на переменную курсора мыши в редакторе. Разумеется, соответствующие переменные должны существовать в рабочей среде, поэтому их просмотр имеет смысл после выполнения M-файла. Воспользуйтесь этой возможностью для получения значений переменных созданного вами файла mydemo.m.

При записи выражений, содержащих много скобок, очень полезным оказывается автоматический контроль за их парностью, который настраивается в пункте **Keyboard&Indenting**. Контроль может производиться в процессе набора, для чего следует установить флаг **Match parentheses while typing**. В раскрывающемся списке **Show match with** вы можете выбрать, как при наборе выражения редактор будет показывать парную скобку: **Underline** — подчеркиванием, **Highlight** — выделением фона символа или **Balance** — выделением фона двух парных скобок. Редактор распознает незакрытые скобки и информирует вас одним из способов, представленных в раскрывающемся списке **Show mismatch with:** **Beep** — звуковым сигналом, **Strikethrough** — перечеркнутым символом или **None** — никак. Возможен также быстрый поиск парной скобки в уже набранном выражении при наведении на нее курсора или оповещение об отсутствии таковой. Для настройки этой опции установите флаг **Match parentheses on arrow key or mouse movement** и обратитесь к раскрывающимся спискам, расположенным под ним.

Еще одна удобная возможность — режим автосохранения файлов, параметры которого можно изменить в пункте **Autosave**. Для включения автосохранения следует установить флаг **Autosave on**. После этого появляется доступ ко всем элементам управления, в частности, можно выбрать интервал времени, через который будет происходить сохранение файла. По умолчанию файл сохраняется с тем же именем и расширением asv, которое можно изме-

нить и на любое другое, установив переключатель **Replace extension with**, и задав новое расширение в строке ввода справа от переключателя. Мы не рекомендуем использовать для этих целей расширение `m`, поскольку при каждом автоматическом сохранении файла будет выводиться диалоговое окно с предупреждением об изменении файла вне редактора. По умолчанию копии автоматически сохраняемого файла размещаются в том же каталоге, что и оригинал, но можно выбрать и отдельный каталог, установив переключатель **Single directory** и воспользовавшись кнопкой справа от него.

Настройка выделения цветом различных фрагментов текста (ключевых слов, текстовых строк, комментариев, ошибок и т. п.) производится в пункте **Color**, а выбор шрифта и его характеристик — в пункте **Font (Custom)**. Для изменения шрифта следует в списке **Desktop tools** выбрать **Editor** и при установленном переключателе **Custom** на панели **Font to Use** выбрать тип шрифта, его размер и стиль.

Мы обсудили только общие настройки редактора, остальные мы будем упоминать по мере надобности. Например, при наборе программ возможен автоматический отступ, что особенно удобно при использовании вложенных циклов или условных операторов. О наиболее важных опциях, связанных с автоотступом, сказано в главе 7, которая посвящена конструкциям языка программирования MATLAB.

Приведенные в книге примеры лучше всего набирать и сохранять в М-файлах, снабжая их необходимыми комментариями. Применение численных методов и программирование в MATLAB, описанное в следующей части книги, как правило, требуют создания М-файлов.

### Примечание

Можно использовать редактор М-файлов и без запуска MATLAB. Для этого дважды щелкните по значку с М-файлом в окне с содержимым папки, в которой он хранится. Файл откроется в редакторе М-файлов. Однако при этом редактор является самостоятельным приложением. Файл можно только редактировать, но не выполнять. Разумеется, расширение `m` в Windows должно быть ассоциировано с приложением `meditor.exe` (редактором М-файлов), что выбирается либо при установке MATLAB, либо в свойствах папки в Windows.

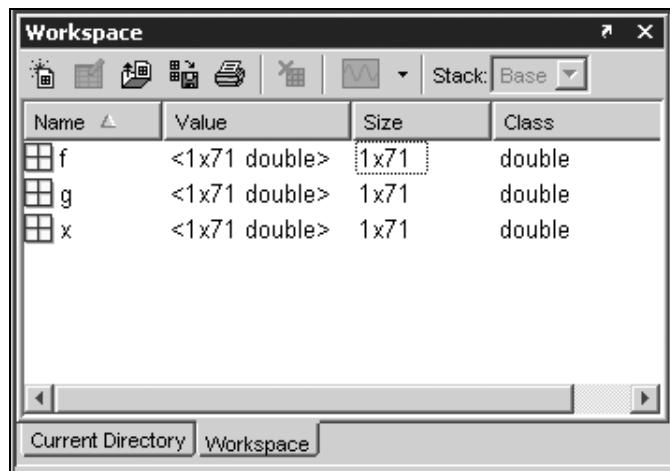
Итак, у нас есть более удобное средство выполнения команд, чем командная строка. Разберем теперь, какие типы М-файлов существуют в MATLAB.

## 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-программы представляют собой простейший тип М-файлов. Они не имеют входных и выходных аргументов и оперируют переменными, существующими в рабочей среде, или могут создавать новые переменные. Файл-программу `mydemo` вы написали при прочтении предыдущего раздела. Все переменные, объявленные в файл-программе, становятся доступными в рабочей среде после ее выполнения. Запустите файл-программу `mydemo`, приведенную в листинге 5.1. Перейдите в окно **Workspace** (рис. 5.2) и убедитесь, что все введенные в М-файле переменные появились в рабочей среде.



**Рис. 5.2.** Диалоговое окно **Workspace**  
после выполнения файл-программы `mydemo`

Все созданные при исполнении М-файла переменные остаются в рабочей среде после его завершения, и их можно использовать в других файл-программах и в командах, выполняемых из командной строки.

Запуск файл-программы 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 (без расширения).

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)* (установка путей поиска и текущего каталога описана в *следующем разделе*). Найденный М-файл выполняется в MATLAB.

Если ни одно из вышеперечисленных действий не привело к успеху, то в командное окно выводится сообщение, например, если сделать ошибку:

```
>> mydem
??? Undefined function or variable 'mydem'.
```

Последовательность поиска MATLAB говорит о том, что очень важно правильно задавать имя собственной файл-программы при сохранении ее в М-файле. Во-первых, ее имя не должно совпадать с именем существующих функций в MATLAB. Узнать, занято имя или нет можно при помощи функции `exist`, которую вы уже использовали при работе с переменными (см. разд. "Просмотр и удаление переменных, выбор имен переменных" главы 1).

Во-вторых, имя файла не должно начинаться с цифры, знаков "+" или "-", словом с тех символов, которые могут быть интерпретированы MATLAB

как ошибка при вводе выражения. Например, если вы назовете М-файл с файлом-программой `5prog.m`, то при ее запуске из меню редактора или по `<F5>` получите сообщение

```
??? 5prog
|
Error: Missing MATLAB operator.
```

Это не удивительно, т. к. MATLAB ждет от вас `5 + prog` (или `5,prog`) для вычисления арифметического выражения с переменной `prog` (или добавления 5 в качестве первого элемента к вектор-строке `prog`). Следовательно, правильным было бы имя `prog5.m` (или хотя бы `p5rog.m`), но только начинаяющееся с буквы.

Обратите внимание, что если вы запускаете на выполнение выделенные команды (могут быть выделены все команды) М-файла с неверным именем при помощи `<F9>`, то ошибки не будет. Фактически происходит последовательное выполнение команд, не отличающееся от их вызова из командной строки, а не работа файл-программы.

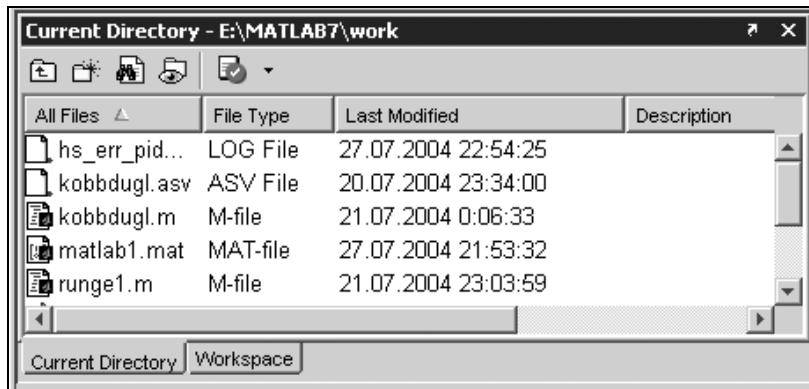
Очень распространена еще одна ошибка при задании имени файл-программы, которая на первый взгляд имеет необъяснимые последствия: программа запускается только один раз. Повторный запуск не приводит к выполнению программы. Разберем эту ситуацию на примере файл-программы из листинга 5.1, которую вы сохранили в файле `mydemo.m`. Переименуйте файл в `x.m`, затем удалите все переменные рабочей среды из окна браузера переменных **Workspace** или из командной строки

```
>> clear all
```

Выполните файл-программу, например, из редактора, нажав `<F5>`. Появляется графическое окно с двумя графиками и ничего не предвещает подвоха. Закройте теперь графическое окно и запустите программу снова. Графическое окно больше не создается, зато в командное окно вывелись значения массива `x` в соответствии с первым пунктом приведенного выше алгоритма поиска MATLAB. Эти обстоятельства следует учитывать при выборе имени файл-программы. Не менее важный вопрос связан с третьим пунктом алгоритма поиска MATLAB — текущим каталогом и путями поиска. Как правило, собственные М-файлы хранятся в каталогах пользователя. Для того чтобы система MATLAB могла найти их, следует установить пути, указывающие расположение М-файлов.

## Установка путей

Установка текущего каталога и путей поиска производится при помощи интерфейса рабочей среды MATLAB либо с использованием команд. Содержимое текущего каталога отображается в окне **Current Directory** с одноименной вкладкой (рис. 5.3). Наличие этого окна в рабочей среде зависит от того, установлен ли флаг слева от названия пункта **Current Directory** меню **View** рабочей среды. Если флага нет, то следует выбрать указанный пункт меню.



**Рис. 5.3.** Диалоговое окно **Current Directory**  
в режиме отображения файлов

Текущий каталог устанавливается выбором из раскрывающегося списка **Current Directory** на панели инструментов рабочей среды MATLAB. Если в списке нет нужного каталога, то его можно добавить в диалоговом окне **Browse for Folder**, которое появляется после нажатия на кнопку, расположенную справа от списка. Текущий каталог можно задать или создать при помощи инструментов окна **Current Directory**. Инструмент **Show Visual Directory** позволяет сделать содержимое окна более информативным (рис. 5.4).

Навигация по дереву каталогов осуществляется в области **Subfolders**, где перечислены имена вложенных папок с указанием в скобках числа файлов в них и ссылка на папку верхнего уровня **<UP>**. Ниже расположены флаги для получения сведений о файлах текущего каталога и кнопка **Refresh** для обновления содержимого окна в случае установки или сброса какого-либо флага. Перед списком файлов расположены две ссылки для создания нового файла (**New file**) или генерации файла contents.m с информацией о содержимом каталога (**run contentsrp**). Это средство удобно, если в текущем катало-

где много файлов, и вы хотите иметь список файлов с примечанием об их назначении. После создания файла `contents.m`, состоящего из строк комментариев с именами файлов, появляется еще одна ссылка `edit Contents.m`. Дополнительно в последнем столбце таблицы с файлами выводится первая встречающаяся строчка комментария в М-файле или информация `No help` при отсутствии таковой.

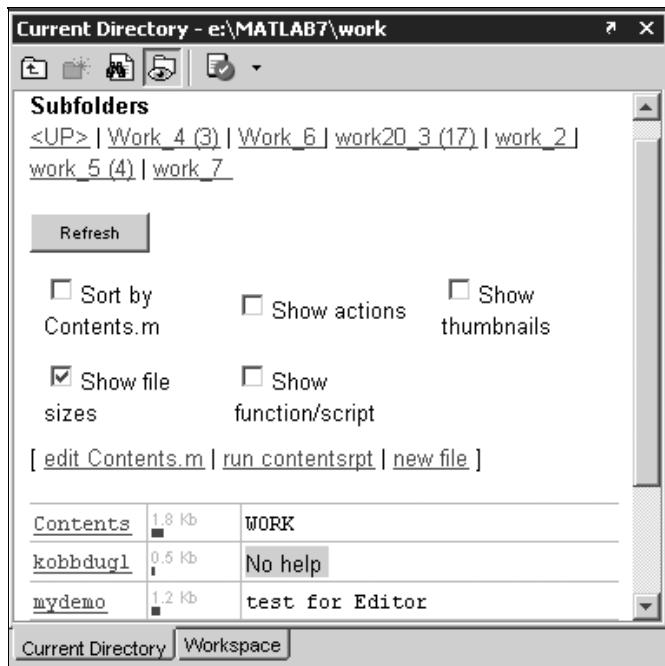
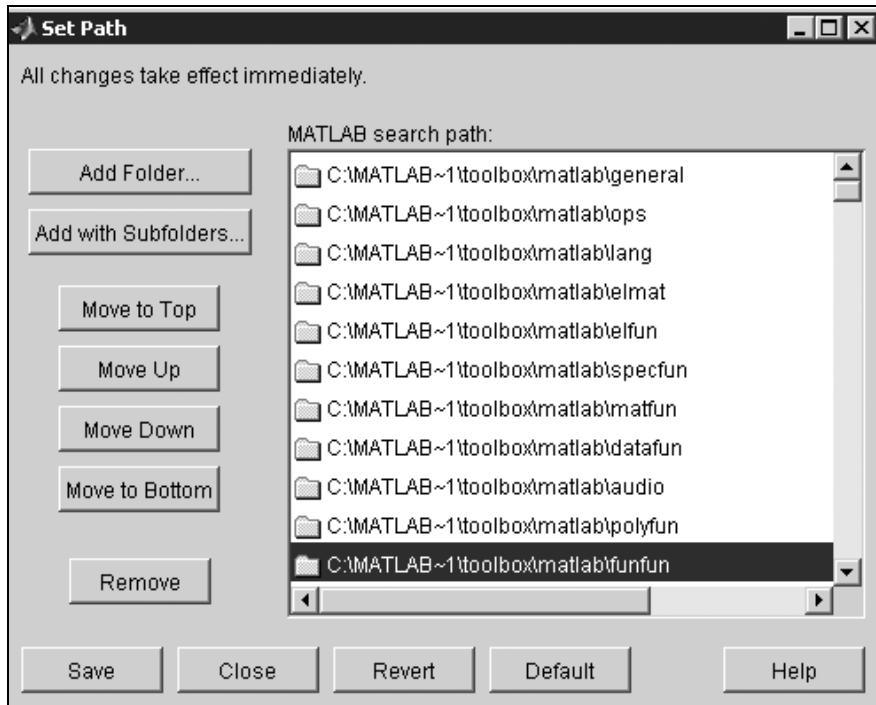


Рис. 5.4. Диалоговое окно **Current Directory** в режиме навигации

### Примечание

Комментарий в таблице окна **Current Directory** будет корректно воспроизведется в окне, если он не содержит символов национального алфавита. В противном случае только в файле `contents.m` будет верно отражен текст комментария (см. примечание в разд. "Оформление графиков" главы 3 по использованию символов кириллицы).

Определение путей поиска файлов 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. 5.5.



**Рис. 5.5.** Диалоговое окно Set Path

Для добавления каталога нажмите кнопку **Add Folder** и в появившемся диалоговом окне **Browse for Folder** выберите требуемый каталог. Добавление каталога со всеми его подкаталогами осуществляется при нажатии на кнопку **Add with Subfolders**. Путь к добавленному каталогу появляется в списке **MATLAB Search Path**. Порядок поиска соответствует расположению путей в нем — первым просматривается каталог, путь к которому размещен вверху списка. Порядок поиска можно изменить, или вообще удалить путь к какому-либо каталогу. Для этого требуемый каталог выделяется в списке и его положение определяется при помощи следующих кнопок:

- Move to Top** — поместить вверх списка;
- Move Up** — переместить вверх на одну позицию;
- Move Down** — переместить вниз на одну позицию;
- Move to Bottom** — поместить вниз списка;
- Remove** — удалить из списка.

Внесенные изменения действуют только до конца текущего сеанса MATLAB. Если желательно сохранить установки путей и для следующих сеансов, то после внесения изменений следует сохранить информацию о путях поиска, нажав кнопку **Save**. При помощи кнопки **Default** можно восстановить стандартные установки, а кнопка **Revert** предназначена для возврата к предыдущим сохраненным установкам.

### Примечание

Рекомендуется хранить собственные М-файлы вне подкаталога toolbox основного каталога MATLAB 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.

## Команды для установки путей

Действия по установке путей дублируются командами. Текущий каталог устанавливается командой `cd`, например:

```
>> cd C:\users\igor
```

Ряд команд MATLAB допускает их вызов в функциональной форме. Например, эквивалентное обращение к `cd` с входным аргументом — строкой следующее: `cd('C:\users\igor')`. Функциональная форма является более универсальной, поскольку позволяет задавать в качестве входных аргументов не только строки, но и строковые переменные, содержащие имя каталога. Это оказывается полезным при написании собственных приложений.

Работа со строковыми переменными описана в главе 8.

Функция `cd`, вызванная без аргумента, возвращает путь к текущему каталогу. Для добавления каталогов в пути поиска служит команда `addpath`, которая по умолчанию помещает каталог в начало списка поиска, например:

```
>> addpath C:\elena
```

Это можно проверить при помощи `path`, которая возвращает список каталогов, входящих в пути поиска. Заметьте, что вызов `path` как команды приводит к отображению списка путей поиска в командном окне, а обращение в функциональной форме с выходным аргументом `p = path` позволяет занести пути поиска в строковую переменную (в нашем случае `p`).

Для добавления каталога в конец списка следует использовать параметр `-end`

```
>> addpath C:\alex -end
```

Так же, как и `cd`, команда `addpath` может быть вызвана в функциональной форме: `addpath(C:\elena)`. При добавлении сразу нескольких каталогов их имена указываются во входных аргументах `addpath` через запятую. Аналогичные возможности предоставляет `path`: `path(path, 'c:\users\igor')`, которая помещает каталог в конец списка, а `path('c:\users\igor', path)` — в начало.

Для удаления определенного каталога из списка путей поиска предназначена функция `rmpath`. Например, `rmpath('c:\users\igor')` удаляет путь к каталогу `c:\users\igor` из списка путей. Функция `rmpath` может быть вызвана и в командной форме: `rmpath('c:\users\igor')`.

### 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.

Мы привели в этом разделе только наиболее используемые функции (команды) MATLAB для манипулирования каталогами. Полная информация содержится в справочной системе в разделах: **MATLAB Functions: Functions - Categorical List; Desktop Tools and Development; Workspace, Search Path, and File; Search Path** или **File Operations**.

## Файл-функции

Рассмотренные выше файл-программы являются последовательностью команд MATLAB, они не имеют входных и выходных аргументов. Для решения вычислительных задач и написания собственных приложений в MATLAB часто требуется программировать файл-функции, которые производят необходимые действия с входными аргументами и возвращают результат в выходных аргументах. Число входных и выходных аргументов зависит от решаемой задачи — может быть только один входной и один выходной аргумент, несколько и тех и других, или только входные аргументы. Возможна ситуация, когда входные и выходные аргументы отсутствуют. В этом разделе разобрано несколько простых примеров, позволяющих понять работу с файл-функциями.

Более сложные примеры файл-функций приведены в главе 8.

Файл-функции, так же как и файл-программы, создаются в редакторе M-файлов.

## Файл-функции с одним входным аргументом

Предположим, что в вычислениях часто необходимо использовать значение функции

$$e^{-x} \cdot \sqrt{\frac{x^2 + 1}{x^4 + 0.1}}.$$

Имеет смысл один раз написать файл-функцию, а потом вызывать ее всюду, где необходимо вычисление этой функции для заданного аргумента. Откройте в редакторе M-файлов новый файл и наберите текст листинга 5.2.

### Листинг 5.2. Файл-функция с одним входным и одним выходным 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 размещаются *имя функции* и списки входных и выходных аргументов. Входные аргументы записываются в круглых скобках после имени функции. В нашем примере есть только один входной аргумент — `x`. Выходной аргумент `f` указывается слева от знака равенства в заголовке функции. При выборе имени файл-функции следует позаботиться об отсутствии конфликтов с занятыми именами в MATLAB. Аналогичный вопрос мы обсуждали выше: как сохранить файл-программу в файле с уникальным именем. Тот же самый подход, основанный на обращении к функции `exist`, вы можете применить для задания имени файл-функции.

После заголовка размещается тело файл-функции — один или несколько операторов (их может быть достаточно много), которые реализуют алгоритм получения значения выходных переменных из входных. В нашем примере алгоритм простой — по заданному `x` вычисляется арифметическое выражение и результат записывается в `f`.

Теперь сохраните файл в рабочем каталоге. Обратите внимание, что выбор пунктов **Save** или **Save as...** меню **File** приводит к появлению диалогового окна сохранения файла, в поле **File name** которого уже содержится название `myfun`. 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
```

При создании файл-функции `myfun` мы подавили вывод значения `f` в командное окно, завершив оператор присваивания точкой с запятой. Если этого не сделать, то оно выведется при обращении `y = myfun(1.3)`. Как правило, лучше избегать вывода в командное окно результатов промежуточных вычислений внутри файл-функции.

Имя файл-функции не обязательно совпадать с именем файла, однако обращение к ней происходит по имени файла. Например, если в файле `f22.m` содержится функция с заголовком `g = init(z)`, то ее следует вызывать так:

```
>> f = f22(-0.9)
```

а вовсе не

```
>> f = init(-0.9)
```

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 использовании встроенных функций.

```
>> x = [1.3 7.2];
>> y = myfun(x)
??? Error using ==> mpower
Matrix must be square.
Error in ==> myfun at 2
f = exp(-x)*sqrt((x^2 + 1)/(x^4 + 0.1));
```

Если вы изучили работу с массивами, то устранение этого недостатка не вызовет затруднений. Необходимо просто при вычислении значения функции применить 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!

### Примечание

Внесение изменений в текст файл-функции без сохранения файла — часто распространенная ошибка начинающих пользователей. Если файл не сохранен, то, как не сложно убедиться, будет вычисляться старая функция. О том, что файл изменен и не сохранен, свидетельствует звездочка в заголовке окна редактора рядом с именем редактируемого файла.

#### 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  $[0, 4]$  при помощи файл-программы или из командной строки:

```
>> x = 0:0.5:4;
>> y = myfun(x);
>> plot(x, y)
```

Решение вычислительных задач средствами MATLAB потребует от вас умения программировать файл-функции, соответствующие поставленной задаче (например, правая часть системы дифференциальных уравнений или по-длинтегральная функция).

Решению вычислительных задач посвящена глава 6.

Мы рассмотрим сейчас только один простой пример того, как использование файл-функций упрощает визуализацию математических функций. Только что вы построили график при помощи `plot`. Заметьте, что для вычисления вектора `u` не обязательно было вызывать `myfun` — можно сразу записать выражение для него и потом указать пару `x` и `u` в `plot`. Имеющаяся в нашем распоряжении файл-функция `myfun` позволяет обратиться к специальной функции `fplot`, которой требуется указать имя нашей файл-функции (в апострофах) или указатель на нее (с оператором `@` перед именем функции) и границы отрезка для построения графика (в векторе из двух элементов)

```
>> fplot('myfun', [0 4])
```

или

```
>> fplot(@myfun, [0 4])
```

Постройте графики `plot` и `fplot` на одних осях (используйте `hold on`) так, как показано на рис. 5.6. График, построенный `fplot`, более точно отражает поведение функции, т. к. алгоритм `fplot` автоматически подбирает шаг аргумента, уменьшая его на участках быстрого изменения исследуемой функции.

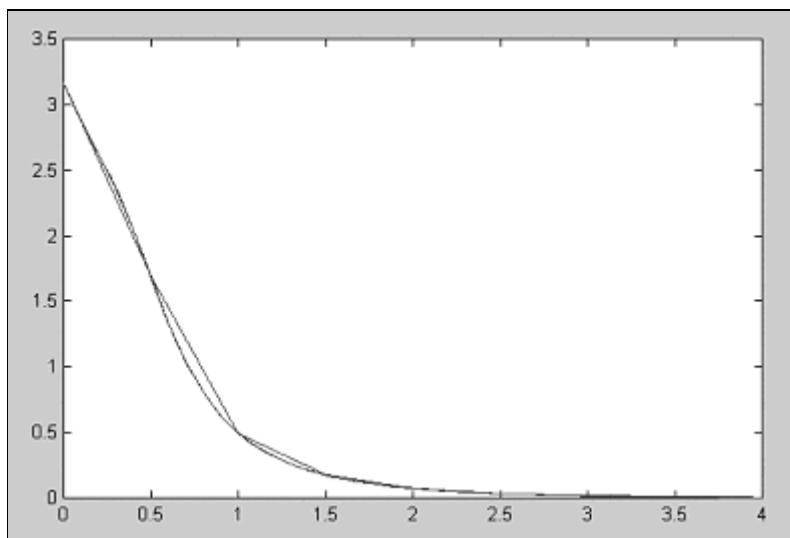


Рис. 5.6. Сравнение `plot` и `fplot`

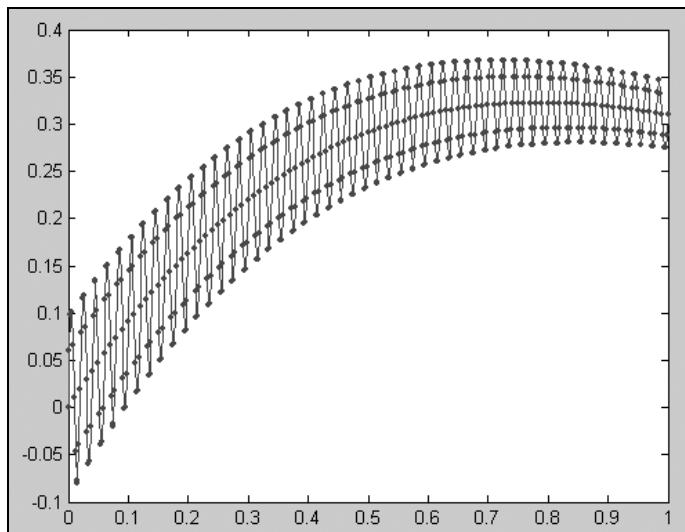
Как было отмечено в главе 3, существенную роль играет выбор шага, при чем неудачный выбор шага может привести к неверному результату. Вернитесь к примеру из главы 3, в котором при помощи `plot` требовалось построить график функции  $f(x) = e^{-x} (\sin x + 0.1 \sin(100\pi x))$  на отрезке  $[0, 1]$  и предлагалось выбрать шаг 0.01. Запрограммируйте соответствующую файл-функцию `myfun1` (листинг 5.4) и сохраните ее в файле `myfun1.m`.

#### Листинг 5.4. Файл-функция для вычисления исследуемой функции

```
function z = myfun1(t)
z = exp(-t).*(sin(t) + 0.1*sin(100*pi*t));
```

#### Примечание

В этом примере мы намеренно использовали `z` и `t`. Имена аргументов файл-функции могут быть любые, они никак не связаны с теми переменными, от которых будет вызываться файл-функция или в которые будет записываться результат. Например, при вызове `a = myfun1(b)` входной аргумент `t` станет равным `b`, затем вычислится `z`, и переменная `a` примет ее значение.



**Рис. 5.7.** Выбор шага функцией `fplot`

Для построения графика осталось вызвать `fplot`. Вы можете узнать о выборе значений аргумента функцией `fplot`, указав третьим ее входным аргументом свойства линии и маркера так же, как и в `plot`. Возьмите точку в качестве маркера и сплошную линию

```
>> fplot(@myfun1, [0 1], '.-')
```

Получается график, приведенный на рис. 5.7. Абсциссы маркеров соответствуют адаптивно подобранным значениям независимой переменной, в которых `fplot` вычислила исследуемую функцию для корректной визуализации.

Функция `fplot` позволяет задавать ряд параметров, управляющих выбором значений абсцисс (см. разд. "Более подробно о `fplot`" главы 6).

## Файл-функции с несколькими входными аргументами

Написание файл-функций с несколькими входными аргументами практически не отличается от случая одного аргумента. Все входные аргументы размещаются в списке через запятую. Например, листинг 5.5 содержит файл-функцию, вычисляющую длину радиус-вектора точки трехмерного пространства  $\sqrt{x^2 + y^2 + z^2}$ .

### Листинг 5.5. Файл-функция с тремя входными 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. имеющих 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 запя-

тую в список выходных аргументов, а сам список заключается в *квадратные скобки*. Листинг 5.6 содержит пример такой файл-функции `hms` для перевода времени, заданного в секундах, в часы, минуты и секунды.

#### Листинг 5.6. 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 возможность создавать файл-функции, которые сами приспосабливаются к числу входных и выходных аргументов. Большинство функций MATLAB работают именно таким образом.

Подробнее о создании файл-функций и файл-программ написано в части II, посвященной программированию собственных приложений.

Функции MATLAB обладают еще одним полезным качеством — возможностью получения информации о них при помощи команды `help`, например, `help fplot`. Собственные файл-функции так же можно наделить этим свойством, используя строки комментариев. Все строки комментариев после заголовка и до тела функции или пустой строки выводятся в командное окно командой `help`. Изучите содержимое файла `fplot.m` с файл-функцией `fplot`, который расположен в подкаталоге `\toolbox\matlab\specgraph\` основного каталога MATLAB. Разместите в файл-функции `hms` комментарии о ее назначении (листинг 5.7) и проверьте: `help hms`.

**Листинг 5.7. Помещение комментариев в файл-функцию**

```
function [hour, minute, second] = hms(sec)
%hms - перевод секунд в часы, минуты и секунды
% Функция hms предназначена для перевода секунд
% в часы минуты и секунды.
% [hour, minute, second] = hms(sec)
%
% sec - число секунд
%
% hour - число полных часов
%
% minute - число полных минут
%
% second - остаток секунд
hour = floor(sec/3600);
minute = floor((sec - hour*3600)/60);
second = sec - hour*3600 - minute*60;
```

Мы намеренно записали в первой строке комментариев краткую информацию о файл-функции. Первая строка комментариев после заголовка функции называется H1-line и используется при поиске командой `lookfor`. Эта команда ищет указанное слово в строках H1-line всех файл-функций в каталогах, указанных в путях поиска, в том числе и в текущем каталоге. Продверьте: `lookfor hms`. Нашлась как ваша функция, так и ряд встроенных функций для преобразования формата времени. Все встроенные функции MATLAB оформлены именно в таком стиле, поэтому при создании набора функций, предназначенных для широкого круга пользователей, есть смысл придерживаться данных соглашений.

**Примечание**

Блок комментариев нельзя использовать для получения подсказки по команде `help`.

## Разновидности функций

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 различные способы организации связей между функциями — *приватные функции, подфункции, вложенные функции*.

## Подфункции

Использование подфункций и вложенных функций основано на выделении части алгоритма в самостоятельную функцию, текст которой содержится в том же файле, что и основная функция. Рассмотрим модельный пример. 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 5.8).

### Листинг 5.8. 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 5.9. 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. Заголовок новой подфункции одновременно является признаком конца предыдущей. Основная функция обменивается информацией с подфункциями только при помощи входных и выходных параметров. Переменные, определенные в подфункциях и в основной функции, являются локальными, они доступны в пределах своей функции. Листинг 5.10 содержит пример файл-функции с подфункцией, приводящий к ошибке!

#### Листинг 5.10. Недопустимое использование локальных параметров

```
function simple1;
ALP = 5.3;
BET = 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 + ALP*BET;
```

Попытка выполнить функцию `simple1` приведет к выводу сообщений о том, что переменная `ALP` (она первой встретилась при интерпретации оператора) не определена:

```
>> simple1
??? Undefined function or variable 'ALP'.
Error in ==> simple1 > f at 6
z = x^3 - 2*y^3 + 3*(x^2 + y^2) - x*y + 9 + ALP*BET;
Error in ==> simple1 at 4
f1 = f(1.1, 2.1)
```

Один из возможных вариантов использования переменных, которые являются общими для всех функций М-файла, состоит в объявлении данных переменных в начале основной функции и подфункции как *глобальных*, при помощи `global` со списком имен переменных, разделяемых пробелом (листинг 5.11).

#### Листинг 5.11. Объявление глобальных переменных

```
function simple2;
global ALP BET
ALP = 5.3;
```

```
BET = 9.1;
f1 = f(1.1, 2.1)
function z = f(x, y)
global ALP BET
z = x^3 - 2*y^3 + 3*(x^2 + y^2) - x*y + 9 + ALP*BET;
```

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 обращается к одноименным глобальным переменным.

Отметим особенности работы с подфункциями, которые часто приводят к ошибкам. Источником ошибок часто является неверное указание подфункции в качестве входного аргумента таких функций MATLAB, как `fplot`, `fzero`, `fminsearch`, `quad` и других, вызываемых внутри основной функции. Обязательно использование ссылки на подфункцию, а не ее имени, иначе MATLAB игнорирует подфункцию и начинает искать файл-функцию в текущем каталоге и в путях поиска. Листинг 5.12 содержит простой пример основной функции `mainfun` с подфункцией `subfun`, демонстрирующий недопустимость задания имени подфункции в первом входном аргументе `fplot`.

#### Листинг 5.12. Правильный и неправильный вызов `fplot` в основной функции

```
function mainfun
% основная функция
figure % создание графического окна
subplot(2, 1, 1) % создание верхней пары осей
fplot(@subfun, [1 5]) % правильный вызов fplot с указателем на подфункцию
subplot(2, 1, 2) % создание нижней пары осей
fplot('subfun', [1 5]) % неправильный вызов fplot с именем подфункции

function y = subfun(x)
% подфункция
y = sin(pi*x);
```

В результате вызова

```
>> mainfun
```

создается графическое окно с двумя парами осей, причем только верхний график, полученный при вызове `fplot` с указателем на подфункцию, являет-

ся верным. Если в текущей папке и путях поиска нет файл-функции `subfun`, то нижний график содержит прямую линию. Так всегда работает `fplot`, когда MATLAB не может найти М-файл с файл-функцией для построения графика. Заметьте, что при наличии в текущей папке файл-функции `subfun` в М-файле `fplot` будет пытаться построить ее график.

Подфункция доступна только внутри основной функции. Если вы решите вызвать подфункцию `f` не из файл-функции `simple` (см. листинг 5.11), а из другой файл-функции, файл-программы или просто из командной строки, то получите сообщение об ошибке:

```
>> f(1, 1)
??? Undefined command/function 'f'.
```

Разумеется, если в текущем каталоге (или путях поиска MATLAB) находится файл-функция `f`, хранящаяся в файле `f.m`, или в рабочей среде объявлена переменная `f`, то ошибки может и не быть, или она будет другого характера. Если доступны различные одноименные функции (приватные, подфункции, встроенные, пользовательские из текущего каталога, функции из Toolbox), то приоритет связывания имени и объекта следующий.

1. Ищется имя в текущей рабочей среде: могут быть определены либо переменная с таким именем, либо вложенная функция (см. далее в этой главе), либо анонимная функция или `inline`-функция, которые рассматриваются в следующей главе.
2. Ищется подфункция в текущем М-файле.
3. Ищется приватная функция (см. следующий раздел).
4. Ищется встроенная функция (`built-in function`).
5. Далее производится поиск требуемой файл-функции в текущем каталоге и путях поиска MATLAB.

Таким образом, пользовательские функции имеют приоритет за исключением случая файл-функций из текущего каталога по отношению к встроенной. Как узнать, является ли функция MATLAB встроенной или нет? Для этой цели служит специальная функция `exist`, о которой шла речь в главе 1. Возвращаемые значения в случае, если аргумент функция, таковы: 1 — имя занято под переменную рабочей среды, 2 — функция расположена в путях поиска файлов, 5 — встроенная функция (`built-in function`). Приватные функции и подфункции не идентифицируются. Сравните, например `max` и `bar`:

```
>> exist max
ans =
```

```
>> exist bar
ans =
2
```

Итак, функция `max` является встроенной, а `bar` — нет. Поэтому, если в текущем каталоге есть две функции `max` и `bar`, которые вызываются из командной строки или некоторой функции, то обращение будет происходить к встроенной функции `max` и пользовательской `bar`.

## Вложенные функции

Другой разновидностью функций, доступных в одном М-файле, являются вложенные функции. Если подфункция является внешней по отношению к основной функции, то вложенная функция является внутренней. В силу этого обстоятельства переменные из рабочей среды основной функции доступны и во вложенной функции. Простейшая структура функции `main`, содержащей вложенную `fnested`, представлена в листинге 5.13.

### Листинг 5.13. Простейшая вложенная функция

```
function main;
ALP = 5.3;
BET = 9.1;
f1 = fnested(1.1, 2.1)
 function z = fnested(x, y)
 z = x^3 - 2*y^3 + 3*(x^2 + y^2) - x*y + 9 + ALP*BET;
 end
end
```

Сравните эту функцию с функцией `simple1` из листинга 5.10, где функция `f` реализовалась как подфункция, и переменные `ALP` и `BET` были неизвестны внутри подфункции. Теперь переменные `ALP` и `BET` доступны в теле функции. При написании вложенных функций следует использовать оператор `end` для закрытия тела функции. Поэтому вложенная функция может размещаться в любом месте тела функции, ее содержащей. Основная функция также завершается оператором `end`. В одном М-файле допускается использование подфункций и вложенных функций одновременно, но тогда последним оператором подфункции должен быть `end`.

Уровень вложенности функций не ограничен. Поэтому при многоуровневом вложении возникает вопрос, какие вызовы допустимы, а какие нет. Функция может обратиться к своей вложенной функции, но не может использовать вложенную функцию нижнего уровня. Вложенная функция может обратиться к функции того же уровня. Функция нижнего уровня может вызвать функцию верхнего уровня, в которую она вложена, и все функции, доступные из нее. На практике такой сложной структуры вложенности функций, как правило, не требуется.

Другая проблема многоуровневого вложения — это доступность переменных в среде вложенных и внешних функций. Переменные, определенные во внешней функции, доступны и во вложенной, и наоборот. Исключение составляет случай коллизии переменных для функций одного уровня. В этом случае во вложенных функциях это разные локальные переменные с одним именем. Естественно, что во внешней функции доступ к двум переменным с одним именем не возможен, поэтому ни одна из них не доступна.

Вложенные функции могут быть доступны не только в M-файле, если использовать указатель на вложенную функцию. В листинге 5.14 приведен пример такого использования вложенной функции для создания двух однотипных функций с разными параметрами.

#### Листинг 5.14. Указатель на вложенную функцию

```
function pointer = fixed_parm(a, b, c)
% a, b, c - параметры процесса.

% Создание указателя на вложенную функцию
pointer = @process;

function y = process (t)
 y = a*sin(b.*t + c);
end

end
```

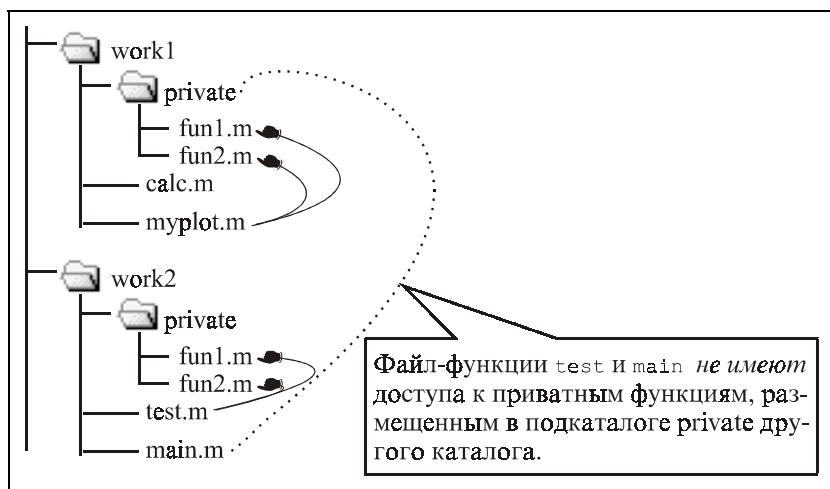
Запишите функцию `fixed_parm` в текущий каталог и выполните команды:

```
>> f = fixed_parm (1.5, 2, 0);
>> g = fixed_parm(1, 10, 25);
>> fplot(f, [-2, 2])
>> hold on;
>> fplot(g, [-2, 2])
```

В результате будут построены графики двух разных функций. Таким образом, созданы две функции `f` и `g`, зависящие от одной переменной, для которых значения параметров зафиксированы на момент обращения к функции `fixed_parm`.

## Приватные функции

Простейший прием сделать некоторые функции видимыми или нет, состоит в размещении М-файлов в рабочих каталогах. Определенная структура каталога с пользовательскими файл-функциями позволяет задать некоторые вспомогательные функции, которые используются только файл-функциями, содержащимися в М-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. Рисунок 5.8 поясняет доступ к приватным функциям.



**Рис. 5.8.** Схема доступа к приватным функциям

## Разбиение М-файла на ячейки

Вернемся теперь к файл-программам и обсудим удобное средство выполнения отдельных их частей. Один из способов — выделение фрагмента кода и

его выполнение при помощи <F9> — был рассмотрен выше. Кроме этого, редактор М-файлов позволяет разбить всю программу на ячейки и выполнять их независимо. Текст разбивается на ячейки при помощи строк комментариев, начинающихся с двух идущих подряд знаков процента (%%).

Для использования средств редактора при работе с ячейками кода следует установить режим **Cell Mode**, выбрав в меню **Cell** пункт **Enable Cell Mode**. При этом становятся доступными другие пункты этого меню, а выбранный изменится на **Disable Cell Mode** для отмены режима. Кроме того, появляется панель инструментов для работы с ячейками. Для оформления файла можно использовать следующие пункты меню **Cell**:

- **Insert Cell Divider** — вставка разделителя ячеек из двух символов процента (%%). Если курсор находится не в первой позиции строки, то разделитель вставляется за текущей строкой, иначе перед ней;
- **Insert Cell Divider around Selection** — вставка разделителей ячеек (%%) до и после выделенного фрагмента (строки предварительно должны быть выделены);
- **Insert Text Markup** — раскрывающийся пункт меню для вставки строк образцов комментариев в место, где расположен курсор.

Способ исполнения ячеек файла также определяется в меню **Cell**:

- **Evaluate Current Cell** — выполняются строки текущей ячейки, и она остается текущей;
- **Evaluate Current Cell and Advance** — выполняются строки текущей ячейки, и текущей становится следующая ячейка;
- **Evaluate Entire File** — выполняется весь файл.

При выполнении файла требуется, чтобы переменные, используемые в командах и функциях, присутствовали в среде **Workspace**. Поэтому не рекомендуется выполнять ячейки файл-функций, т. к. имена аргументов при обращении к функции и имена параметров в тексте функции, как правило, не совпадают.

### ◀ Предупреждение ▶

Выполнение ячеек М-файла в режиме **Cell Mode** происходит без предварительного сохранения его на диске, в отличие от запуска всего файла при помощи <F5>. Поэтому, если вы отлаживаете файл, не забудьте его сохранить, иначе внесенные изменения могут быть потеряны.

Перемещение по ячейкам вверх и вниз осуществляется выбором в меню **Cell** пунктов **Previous Cell** и **Next Cell** соответственно. Цвет фона ячейки определяется в настройках редактора в диалоговом окне **Preferences**. В нем следует выбрать пункт **Display**, установить флаг **Show cell highlighting** и выбрать цвет в раскрывающемся списке справа от флага.

Для освоения простейших принципов работы с М-файлами, разбитыми на ячейки, создайте новый файл и скопируйте туда текст из файла mydemo.m (см. листинг 5.1). Переместите курсор в конец строки `plot(x, f)` и в меню **Cell** выберите пункт **Insert Cell Divider**. После этой строки вставится строка из двух знаков процента `%%`, делящая файл на две ячейки. Выполните весь файл, используя пункт **Evaluate Entire File** или соответствующую кнопку на панели инструментов. Измените функцию `exp(-x)` на `log(1 + x)` и выполните содержащийся в первой ячейке код, выбрав пункт **Evaluate Current Cell**. Вид первого графика изменится. Аналогичным образом можно вывести график другой функции на другие оси, выполнив вторую ячейку. Не закрывайте графическое окно, потому что мы сейчас рассмотрим еще один М-файл, разбитый на ячейки, — автоматически созданную файл-функцию, которая соответствует нашему графическому окну.

Перейдите в графическое окно и сгенерируйте код для его построения, выбрав в меню **File** пункт **Generate M-File**. В редакторе М-файлов создается новое окно с текстом файл-функции `createfigure`, приведенным в листинге 5.15.

#### Листинг 5.15. Автоматически созданная функция для графического окна

```
function createfigure(x1, y1, y2)
%CREATEFIGURE(X1, Y1, Y2)
%
% X1: vector of x data
% Y1: vector of y data
% Y2: vector of y data
% Auto-generated by MATLAB on 24-Jul-2004 08:16:35

%
% Create figure
figure1 = figure('PaperPosition', [0.6345 6.345 20.3 15.23], 'PaperSize',
[20.98 29.68]);

%
% Create axes
axes1 = axes('OuterPosition', [0 0 0.4823 1], 'Parent',figure1);
hold(axes1,'all');
```

```
%% Create plot
plot1 = plot(x1, y1, 'Parent', axes1);

%% Create axes
axes2 = axes('OuterPosition', [0.4823 0 0.5177 1], 'Parent', figure1);
hold(axes2, 'all');

%% Create plot
plot2 = plot(x1, y2, 'Parent', axes2);
```

MATLAB создает файл-функцию с выделением ячеек для каждого шага построения: создания графического окна, осей и графиков, вы можете изменить и дополнить текст этой файл-функции по своему усмотрению. Файл-функция зависит от трех аргументов  $x_1$ ,  $y_1$  и  $y_2$ , являющихся формальными параметрами, которые не определены в рабочей среде **Workspace**. Можно, например, удалить заголовок, превратив файл-функцию в файл-программу, и заменить формальные переменные  $x_1$ ,  $y_1$  и  $y_2$  на переменные рабочей среды  $x$ ,  $f$  и  $g$  соответственно. Последовательное выполнение ячеек полученной файл-программы продемонстрирует все шаги для создания графического окна с графиками нужной функции. В качестве упражнения добавьте в нужные ячейки команды для вывода заголовков к графикам.

Разбиение файл-программы на ячейки и их выполнение дает возможность контроля за ходом работы программы. Для файл-функций применение этой технологии сопряжено с определенными трудностями, поскольку ее переменные являются локальными. Это обстоятельство подчеркнуто в справочной системе MATLAB. Редактор M-файлов включает в себя более мощный инструмент — отладчик, описанию и демонстрации возможностей которого мы уделим далее внимание при программировании приложений (средства диалоговой отладки рассматриваются в главе 8).

Обратимся теперь к еще одной привлекательной возможности MATLAB, которая помогает эффективно реализовывать собственные алгоритмы — к средству M-Lint для диагностики M-файлов.

## Диагностика М-файлов

В состав среды MATLAB включено средство M-Lint для проверки качества и корректности написанного кода. Для этого после сохранения M-файла следует в меню редактора Tools выбрать пункт **Check Code with M-Lint** (если

файл не был сохранен, то этот пункт меню называется **Save and Check Code with M-Lint**). Воспользуйтесь этим средством для проверки файла-функции из листинга 5.9. После выполнения проверки появится окно **M-Lint Code Check Report** с отчетом, представленное на рис. 5.9.

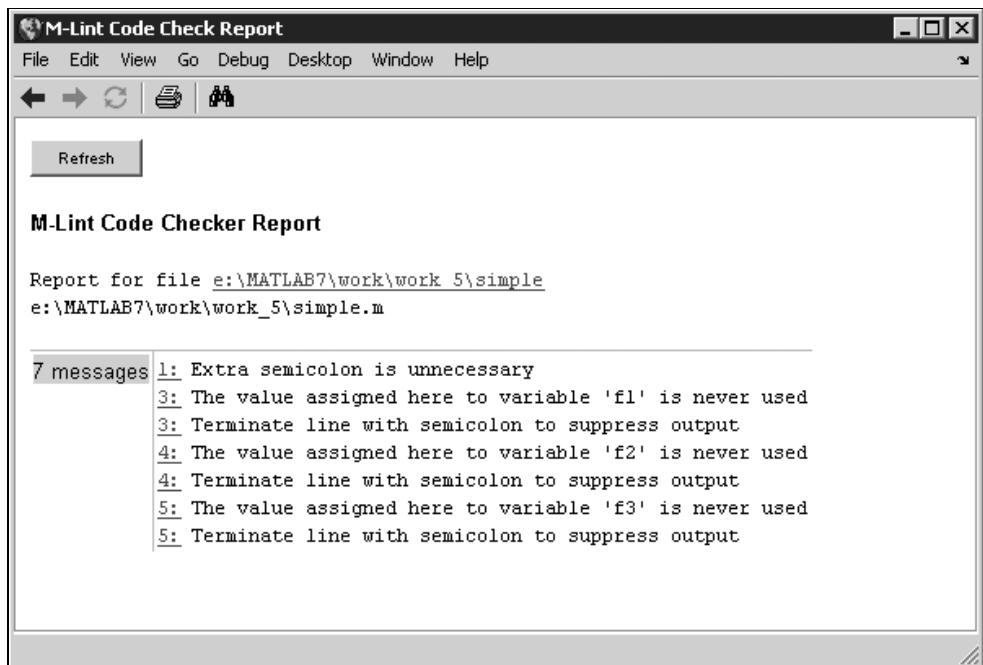


Рис. 5.9. Окно M-Lint: Code Check Report с диагностическими сообщениями

В каждом предупреждении указаны номер строки и выявленный недочет. Номер строки является гиперссылкой, щелчок мыши по которой приводит к переходу к соответствующей строке в редакторе М-файлов.

В нашем примере выявлены три типа недостатков. Во-первых, выведено сообщение о том, что в первой строке не требуется символ точки с запятой. Во-вторых, переменные f1, f2, f3 нигде не используются (строки 3, 4 и 5). В-третьих, пользователю рекомендуется ставить точку с запятой для подавления вывода результатов выполнения на экран (строки 3, 4 и 5). В нашем случае имеет смысл учесть информацию об использовании точек с запятой. Остальная диагностика не важна, т. к. пример с файл-функцией simple иллюстрировал другие возможности пакета MATLAB.

## Задания для самостоятельной работы

1. Напишите и выполните файл-программу построения графиков следующих функций с заголовком, подписями к осям, сеткой. Используйте различные типы линий и маркеров. Для вычисления значений функций создайте файл-функцию. Для `fplot` и `plot` используйте две подобласти на одном рисунке:

- $u(x) = \sin(\ln(x+1)) + \cos(\ln(x+1))$ ,  $x \in [0, 2\pi]$ ;

- $f(x) = \frac{1}{1 + \frac{1}{1+x}}$ ,  $x \in [-0.9, 0.9]$ ;

- $g(x) = x^{x^x}$ ,  $x \in [0.5, 1.5]$ ;

- $h(x) = \sin\left(6\pi\left|x - \frac{2}{3}\right| x^2\right)$ ,  $x \in [0, 1]$ .

2. Напишите и выполните файл-программу для построения поверхностей. Для вычисления значений функций создайте файл-функцию:

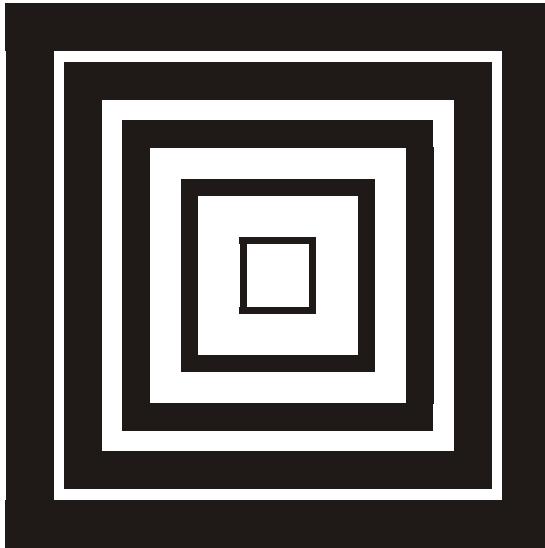
- $z(x, y) = e^{3x \sin 0.5\pi y} + e^{3y \sin 0.5\pi x}$ ,  $x, y \in [-1, 1]$ ;

- $w(x, y) = \sin(e^{2x} - e^{-2y}) + \cos(e^{2y} - e^{-2x})$ ,  $x, y \in [0, 1]$ .

3. Напишите файл-функцию для решения следующих задач:

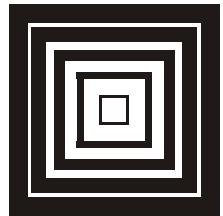
- по заданному вектору определить номер его элемента с наибольшим отклонением от среднего арифметического всех элементов вектора;
- вычислить сумму всех элементов вектора с нечетными индексами;
- найти максимальное значение среди диагональных элементов заданной матрицы;
- переставить первый столбец квадратной матрицы с ее диагональю;
- просуммировать все внедиагональные элементы заданной матрицы;

- заменить максимальный элемент вектора средним значением всех его элементов;
- заменить элемент матрицы с индексами 1,1 произведением всех элементов матрицы;
- построить многоугольник (замкнутый) с координатами вершин  $(x_i, y_i)$ , заданными векторами  $x$  и  $y$ ;
- отобразить элементы заданного вектора синими маркерами, а максимальный элемент — красным и вернуть значение и номер максимального элемента.



# ЧАСТЬ II

# Вычисления и программирование



## Глава 6

# Методы вычислений в MATLAB

MATLAB обладает большим набором специальных функций, реализующих различные численные методы. Нахождение корней уравнений и систем, приближение табличных функций, интегрирование, решение задач линейной алгебры, оптимизация, решение систем обыкновенных дифференциальных уравнений, работа с разреженными матрицами — вот далеко не полный перечень возможностей, предоставляемых MATLAB. В этой главе разобрано решение типовых задач на применение вычислительных методов. Теоретические проблемы численного анализа рассматриваемых задач выходят за рамки настоящей книги.

## Исследование функций

Решение уравнений, нахождение максимума или минимума функции одной или нескольких переменных осуществляются вызовом специальных функций MATLAB. Число аргументов этих функций может быть различным, в зависимости от требуемого вида результата. Для работы с ними, как правило, необходимо запрограммировать исследуемую функцию, например, в виде файл-функции. При этом можно обращаться к файл-функции либо по имени файла, либо по ссылке на нее (*см. разд. "Файл-функции" главы 5*).

Если исследуемая функция задается достаточно простой и короткой формулой, то не обязательно составлять файл-функцию. Вместо этого удобно ввести встраиваемую функцию (*inline-функцию*), воспользовавшись функцией `inline`, или определить анонимную функцию.

## Встраиваемые и анонимные функции

Встраиваемая функция определяется при помощи функции `inline`, обращение к которой выглядит следующим образом:

```
Имя_функции = inline('формула', список_аргументов)
```

Список аргументов не обязателен, а 'формула' является текстовой строкой и задает выражение для вычисления значения функции.

Следующий пример демонстрирует создание в рабочей среде встраиваемой функции fun:

```
>> fun = inline('sin(x) - x.^2.*cos(x)')
fun =
 Inline function:
 fun(x) = sin(x) - x.^2.*cos(x)
```

Inline-функция fun может быть использована как любая другая функция MATLAB, например:

```
>> y = fun(0.5)
y =
 0.2600
```

Если функция зависит от нескольких переменных, то все они являются аргументами введенной inline-функции и располагаются в алфавитном порядке:

```
>> fun1 = inline('sin(a*x) - x.^2.*cos(b*x)')
fun1 =
 Inline function:
 fun1(a,b,x) = sin(a*x) - x.^2.*cos(b*x)
```

Для изменения порядка аргументов их следует перечислить через запятые в списке после выражения, определяющего вид функции:

```
>> fun2 = inline('sin(a*x) - x.^2.*cos(b*x)', 'x', 'a', 'b')
fun2 =
 Inline function:
 fun2(x, a, b) = sin(a*x) - x.^2.*cos(b*x)
```

Если в списке случайно пропущен хотя бы один из аргументов, то inline-функцией воспользоваться не удастся:

```
>> fun3 = inline('sin(a*x) - x.^2.*cos(b*x)', 'x', 'b')
fun3 =
 Inline function:
 fun3(x,b) = sin(a*x) - x.^2.*cos(b*x)
```

Даже при наличии переменной a в рабочей среде вызов функции fun3 приведет к сообщению о том, что аргумент a не задан:

```
>> a = 1;
>> fun3(5,0)
```

```
??? Error using ==> inlineeval
Error in inline expression ==> sin(a*x) - x.^2.*cos(b*x)
??? Error using ==> eval
Undefined function or variable 'a'.
Error in ==> inline.subsref at 25
 INLINE_OUT_ = inlineeval(INLINE_INPUTS_, INLINE_OBJ_.inputExpr,
INLINE_OBJ_.expr);
```

Этот пример демонстрирует, что при вычислении значения встраиваемой функции переменные рабочей среды недоступны. Все аргументы функции `inline` должны быть символьными строками, заключенными в апострофы, или строковыми переменными. В противном случае получается недопустимая конструкция. Работа со строковыми переменными описана далее в книге, а при чтении этой главы достаточно придерживаться простого правила — ставить апострофы в аргументах функции `inline`.

Альтернативный способ задания исследуемой функции состоит в объявлении анонимной функции с помощью оператора указателя (@):

Имя\_функции = @(список\_аргументов) формула

В отличие от `inline`-функции, и аргументы, и формула записываются в обычном виде, а не как текстовые строки в апострофах. Кроме того, анонимной функции доступны переменные рабочей среды, которые входят в формулу. Однако они являются константами, в качестве которых берутся значения этих переменных в момент создания анонимной функции, и последующее изменение их значений не будет учитываться при вычислении функции:

```
>> a= 1;
>> gun3 = @(x, b) (sin(a*x) - x.^2.*cos(b*x))
gun3 =
 @(x, b) sin(a*x) - x.^2.*cos(b*x)
>> gun3(5, 0)
ans =
 -25.9589
>> a = 1000;
>> gun3(5, 0)
ans =
 -25.9589
```

По способу использования анонимная функция напоминает `inline`-функцию, но отличается тем, что создается указатель на функцию, который связан с

исполняемым кодом. Это хорошо видно либо в окне **Workspace**, либо при выводе информации о функциях с помощью `whos`:

```
> whos gun3
 Name Size Bytes Class
 gun3 1x1 16 function_handle array
Grand total is 1 element using 16 bytes
>> whos fun3
 Name Size Bytes Class
 fun3 1x1 908 inline object
Grand total is 79 elements using 908 bytes
```

Информация о выделенной под функции памяти показывает, что для анонимной функции исполняемый код и указатель на нее отделены, а для inline-функции это единый объект.

Вышеописанные способы задания исследуемых математических функций (по имени М-файла, по ссылке, анонимная функция или inline-функция) допустимы во всех вычислительных алгоритмах MATLAB, о которых пойдет речь в следующих разделах. Мы будем использовать все эти возможности и ряд других. Подчеркнем, что для достаточно простых исследуемых функций удобнее всего определить соответствующую inline или анонимную функцию. В то же время, если функция задана громоздким выражением, лучше всего написать соответствующий М-файл. А в случае, когда функция вычисляется по сложному алгоритму, это оказывается просто необходимым.

## Решение уравнений

Для нахождения корней произвольных уравнений служит функция `fzero`, а для определения всех корней полиномов применяется `roots`.

## Решение произвольных уравнений

Функция `fzero` позволяет приближенно вычислить корень уравнения на некотором интервале или ближайший к заданному начальному приближению. В простейшем варианте `fzero` вызывается с двумя входными и одним выходным аргументом `x = fzero('func_name', x0)`, где `func_name` — имя файл-функции, вычисляющей левую часть уравнения, `x0` — начальное приближение к корню, `x` — найденное приближенное значение корня. Решите, например, на отрезке  $[-5, 5]$  уравнение:

$$\sin x - x^2 \cos x = 0. \quad (6.1)$$

Перед нахождением корней полезно построить график функции, входящей в левую часть уравнения. Для получения графика можно прибегнуть к `plot`, но все равно понадобится запрограммировать 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 функции `myf`, используя `fplot`, и нанесите сетку.

```
>> fplot('myf', [-5 5])
>> grid on
```

Из графика функции, изображенного на рис. 6.1 (пояснения на графике нанесены средствами MATLAB), видно, что на этом отрезке имеются четыре корня. Один корень равен нулю, в чем несложно убедиться, подставив  $x = 0$  в уравнение (6.1).

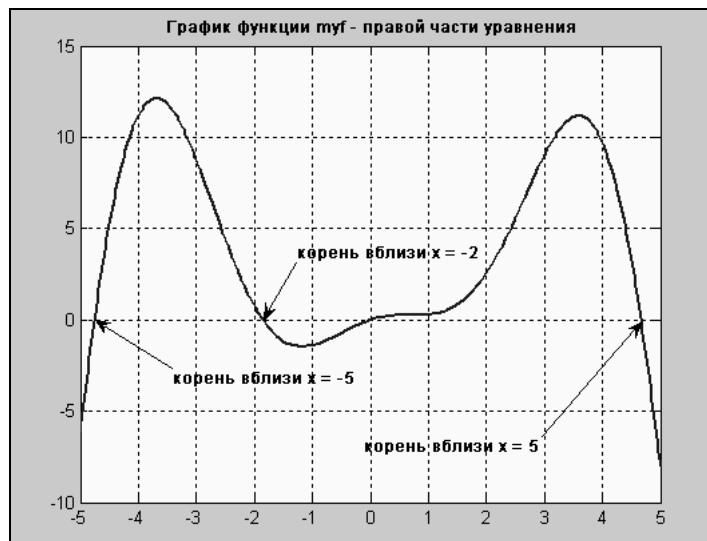


Рис. 6.1. График левой части уравнения (6.1)

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  $-4.7566$ . При указании начального приближения к корню алгоритм `fzero` автоматически отделяет корень, т. е. вблизи заданного начального приближения находится отрезок, содержащий корень. В этом случае `fzero` может использовать больший интервал определения функции, чем исходный отрезок  $[-5, 5]$ . 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 достаточную точность найденного корня. Гарантированная точность приближенного значения определяется расстоянием до его истинного значения или (что фактически то же самое) количеством верных значащих цифр.

Заданию точности вычислений посвящен *разд. "Управление ходом вычислений"* данной главы. Для того чтобы увидеть больше значащих цифр корня `x1`, следует установить формат `long` и вывести `x1` еще раз (точность проводимых пакетом MATLAB расчетов 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. На компьютере вычисления производятся с числами, имеющими 52 двоичных разряда в мантиссе (без учета порядка числа). Это соответствует относительной погрешности представления чисел, которую возвращает функция `eps`, вызываемая без входных аргументов:

```
>> eps
ans =
2.220446049250313e-016
```

Алгоритм функции `fzero` по умолчанию находит корень уравнения с точностью `eps`, т. е.  $\pm 2$  в шестнадцатом знаке после десятичной точки — практически с максимально возможной точностью. Чтобы убедиться в этом, измените формат представления данных на `long e` и выведите `x1`:

```
>> format long e
>> x1
x1 =
-4.756559405702904e+000
```

Число значащих цифр увеличилось на одну (по сравнению с форматом `long`). Это самое точное представление числа в десятичной форме для пакета MATLAB. Измените последнюю цифру 4 на 5 и вычислите значение `myf` в точке `x1`:

```
>> x1 = -4.756559405702905e+000;
>> myf(x1)
ans =
-1.765254609153999e-014
```

Поскольку исследуемая функция сменила знак, то между этими значениями лежит искомый корень уравнения. Следовательно, ошибка в найденном корне в последней значащей цифре.

### Примечание

Численное решение любой задачи требует дополнительных расчетов, подтверждающих (как правило, только косвенно) правильность полученных результатов. Для задачи нахождения корня уравнения вы только что проделали такой численный эксперимент.

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`.

Найдите самостоятельно корни `x2` и `x3`, расположенные около точек  $-2$  и  $-5$ .

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!

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
```

Допустимы другие способы вызова `fzero`, о которых шла речь ранее в этой главе. Во-первых, функцию с исследуемой математической функцией можно задать при помощи указателя на нее:

```
>>x2 = fzero(@myf, [-3 -1]);
x2 =
-1.8539
```

Во-вторых, воспользовавшись функцией `inline`:

```
>> fun = inline('sin(x) - x.^2.*cos(x)')
fun =
Inline function:
fun(x) = sin(x) - x.^2.*cos(x)
>> x1 = fzero(fun, -5)
x1 =
-4.7566
>> fun(x1)
ans =
2.6645e-015
```

В-третьих, создав анонимную функцию:

```
>> fun = @(x) sin(x) - x.^2.*cos(x)
fun =
@(x) sin(x) - x.^2.*cos(x)
>> x1 = fzero(fun, -5)
```

```
x1 =
-4.7566
```

Обращение к `fzero` с двумя выходными аргументами позволяет не только приближенно найти корень, но и получить значение функции в найденной точке.

```
>> [x2, f] = fzero(@myf, -2)
x2 =
-1.8539
f =
-2.2204e-016
```

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:

```
>> fun=inline('x.^2');
>> x = fzero(fun, -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 значения функции `myf` имеют различные знаки, что гарантировало бы существование корня на этом промежутке. Такой промежуток, естественно, определить не удалось, и `fzero` вывела сообщение об ошибке в командное окно.

Программирование собственных приложений с использованием вычислительных функций требует получения сведений о завершении вычислительного процесса для перехода к соответствующему блоку алгоритма. Обращение к `fzero` с тремя выходными аргументами

```
>> [x1, f1, flag] = fzero(fun, 0.1);
```

позволяет выбрать дальнейшие действия в зависимости от содержимого переменной `flag`. Положительное значение `flag` свидетельствует об успешном завершении вычислительного процесса. Отрицательное значение говорит о том, что либо не удалось отделить интервал со сменой знака функции на границах, либо в процессе вычислений получилось комплексное значение, бесконечность, или выполнена операция с неопределенным результатом, например, деление нуля на ноль.

Функция `fzero`, так же как и большинство вычислительных функций MATLAB, допускает управление многими параметрами заложенных в ней алгоритмов. Кроме того, для контроля выполнения алгоритма и изменения его параметров следует получить информацию о ходе вычислений. Управление алгоритмом и контроль за ним требуют специальных способов вызова функций MATLAB (см. разд. "Управление ходом вычислений" данной главы и приложение I).

## 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 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
```

Известно, что число всех корней полинома совпадает с его степенью. 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
```

В верхней строке результата содержится общий множитель  $1.0\text{e}-012$ , на который следует умножить каждую компоненту получившегося вектора.

## Нахождение экстремумов функций

Вычислительные функции MATLAB позволяют найти точки минимума функции одной или нескольких переменных. Результатом является локальный минимум, т. е. точка, в окрестности которой значения исследуемой функции превышают ее значение в точке локального минимума. Для определения точек локального максимума нет специальной функции, поскольку достаточно искать минимум функции с обратным знаком.

### 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

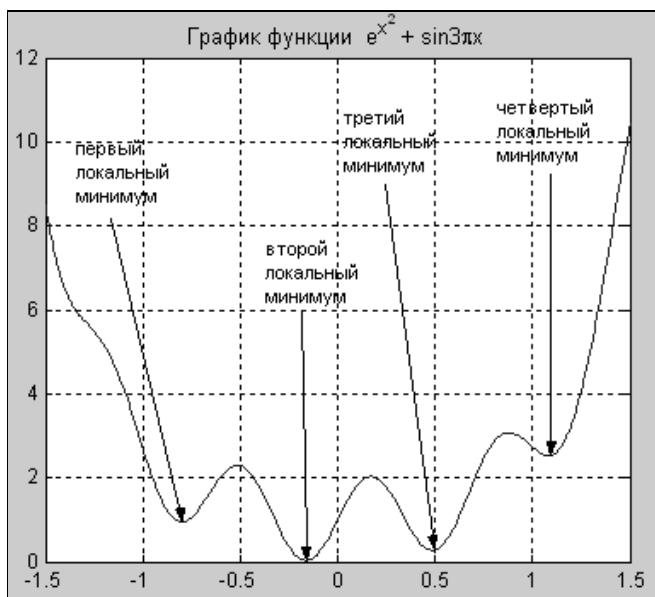
$$e^{x^2} + \sin 3\pi x \quad (6.2)$$

на отрезке  $[-1.5, 1.5]$ . Требуется предварительно создать соответствующую файл-функцию, назвав ее, к примеру `ftest`, или ввести `inline` или анонимную функцию (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 четыре локальных минимума. Вычислите значение  $x$ , при котором достигается второй локальный минимум, задав первым аргументом `fminbnd` имя файл-функции или указатель на нее, а вторым и третьим — границы отрезка, на котором ищется локальный минимум (установка точности и дополнительных параметров минимизации описана в разд. "Управление ходом вычислений" данной главы):

```
>> x2 = fminbnd(@ftest, -0.5, 0)
x2 =
-0.1629
```



**Рис. 6.2.** Расположение локальных минимумов функции (6.2)

Для лучшего понимания работы `fminbnd` задайте интервал поиска, содержащий все точки локальных минимумов.

```
>> xx = fminbnd('ftest', -1.5, 1.5)
xx =
0.4861
```

Для одновременного вычисления значения функции в точке минимума следует вызвать `fminbnd` с двумя аргументами:

```
>> [x2, f] = fminbnd(@ftest, -0.5, 0)
x2 =
-0.1629
```

```
f =
0.0275
```

Так же, как и `fzero`, функция `fminbnd` может быть вызвана с тремя выходными аргументами:

```
>> [x2, f, flag] = fminbnd(@ftest, -0.5, 0);
```

Аргумент `flag` может принимать три значения: положительное — при нахождении локального минимума, нулевое — при достижении максимального количества вызовов исследуемой функции и отрицательное — в случае расходности вычислительного процесса.

Найдите самостоятельно остальные локальные минимумы и максимумы функции (6.2).

Рекурсивный алгоритм для поиска всех локальных минимумов на отрезке приведен в главе 8.

Искать минимум функции можно также с применением `fminsearch`, рассматриваемой далее, которая предназначена для минимизации функции одной и нескольких переменных.

## Минимизация функции нескольких переменных

Минимизация функции нескольких переменных является более сложной задачей по сравнению с минимизацией функции одной переменной, однако комбинированный адаптивный алгоритм функции `fminsearch` позволяет во многих случаях успешно решить эту задачу. Функция `fminsearch` требует указания начального приближения для искомой точки минимума, которое в случае функции одной переменной должно быть числом, а для функции нескольких переменных — вектором.

Задайте начальное приближение равное  $-0.5$  в примере предыдущего раздела и найдите точку локального минимума:

```
>> x2 = fminsearch(@ftest, -0.5)
x2 =
-0.1629
```

Выбирая последовательно подходящие начальные приближения, найдите все точки минимумов и максимумов функции.

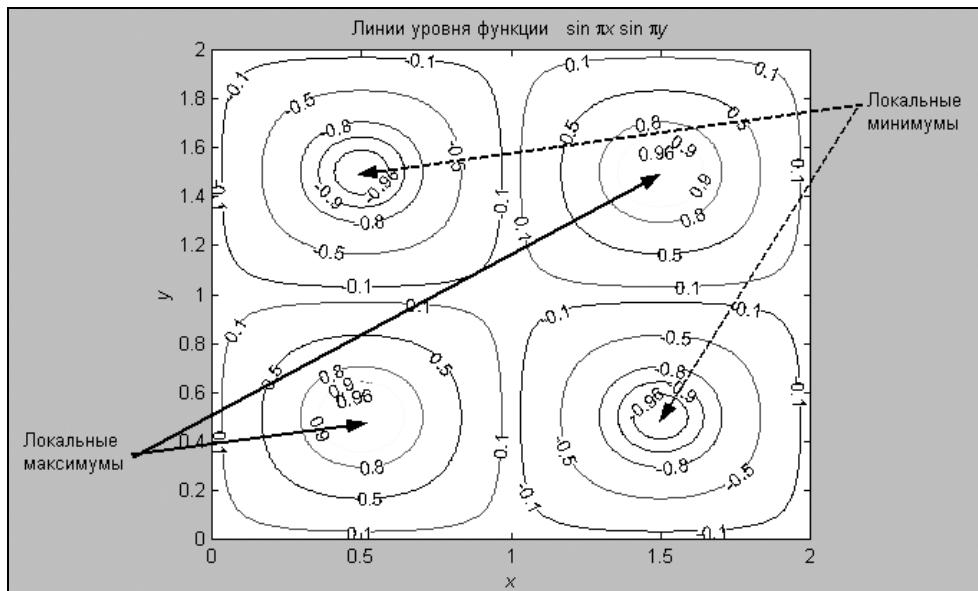
Рассмотрим теперь минимизацию функций нескольких переменных на примере функции двух переменных

$$f(x, y) = \sin \pi x \cdot \sin \pi y. \quad (6.3)$$

Сначала получите представление о поведении функции, построив ее линии уровня при помощи следующих команд (отображение линий уровня функции двух переменных описано в разд. "Контурные графики" главы 3):

```
>> [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)
```

На получившемся графике, приведенном на рис. 6.3, видно расположение локальных минимумов и максимумов.



**Рис. 6.3.** Расположение локальных максимумов и минимумов функции (6.3)

Перед применением `fminsearch` необходимо создать файл-функцию, вычисляющую значения искомой функции, причем аргументом файла-функции должен быть вектор, первый элемент которого соответствует переменной  $x$ , а второй —  $y$ . Текст требуемой файла-функции `ftest2` приведен в листинге 6.2.

**Листинг 6.2. Файл-функция ftest2**

```
function f = ftest2(v)
x = v(1);
y = v(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])
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])
M =
 1.5000 0.5000
f =
 -1.0000
```

Обращение к `fminsearch` с третьим дополнительным выходным аргументом `flag`

```
>> [M, f, flag] = fminsearch(@ftest2, [1.4, 0.6]);
```

позволяет записать в него информацию о причине останова вычислительно-го алгоритма. Смысл его значений тот же, что и для функции `fminbnd`, рассмотренной в предыдущем разделе.

Исследуемая функция может зависеть от произвольного числа переменных  $n$ . В этом случае входной аргумент `argvect` соответствующей файл-функции и начальное приближение должны быть векторами длины  $n$ .

В нашей задаче не обязательно было программировать файл-функцию в М-файле. Поскольку входным аргументом анонимной функции может быть вектор, то следующие команды приведут к поиску минимума:

```
>> fun = @(v) sin(pi*v(1)).*sin(pi*v(2));
>> [M, f] = fminsearch(fun, [1.4, 0.6])
```

```
M =
 1.5000 0.5000
f =
 -1.0000
```

### Примечание

Optimization Toolbox, которому посвящена глава 16, содержит дополнительные средства минимизации функций. Например, `fminunc` можно использовать для поиска минимума функции нескольких переменных.

## Управление ходом вычислений

Функции `fzero`, `fminbnd` и `fminsearch` допускают определение дополнительных параметров для управления вычислительным процессом и контроля за ним. Параметры задаются в управляющей структуре, которую мы будем называть `options`, как в справочной системе MATLAB, хотя имя может быть произвольным. Перед вызовом вычислительных функций следует предварительно сформировать переменную `options` в соответствии с характером требуемого контроля, воспользовавшись функцией `optimset`. Переменная `options` на самом деле является *структурой*. Это новый тип данных — до сих пор вы имели дело только с числовыми массивами. При чтении разделов этой главы умение работать со структурами не требуется, можно просто следовать приведенным правилам заполнения, просмотра и использования управляющей структуры `options` (*работа со структурами данных описана в главе 8*).

Приступим к формированию структуры `options` на примере минимизации функции одной переменной (6.2) при помощи `fminbnd`. Получим сначала информацию о работе алгоритма минимизации на его последнем шаге. Для этого вызовем функцию `optimset` с одним выходным аргументом `options` и двумя входными '`Display`' и '`final`', а затем укажем `options` в дополнительном четвертом входном аргументе `fminbnd`. Всюду дальше мы будем использовать термины *свойство*, *вид контроля*, *опция* или *параметр* (в данном случае `Display`) и *значение* свойства или параметра (в данном случае '`final`'). Установите формат вывода `long`, поскольку нам понадобятся все значащие цифры для анализа результата.

```
>> format long
>> options = optimset('Display', 'final');
>> x2 = fminbnd(@ftest, -0.5, 0, options)
Optimization terminated successfully:
```

```
the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-004
```

```
x2 =
-0.1629
```

Кроме координаты точки локального минимума выводится информация об успешном завершении вычислительного процесса и точности, с которой (по умолчанию) найдено решение. Для изменения точности следует заново сформировать структуру options, указав еще одну пару входных аргументов. В следующем примере мы задаем не только опцию Display, но и точность  $10^{-9}$  по аргументу при помощи параметра TolX.

```
>> options = optimset('Display', 'final', 'TolX', 1.0e-09);
>> x2 = fminbnd(@ftest, -0.5, 0, options)
```

```
Optimization terminated successfully:
```

```
the current x satisfies the termination criteria using OPTIONS.TolX of
1.000000e-009
```

```
x2 =
-0.16289942841268
```

Сравните полученный результат со значением, вычисленным fminbnd с используемой по умолчанию точностью  $10^{-4}$ . Различие получено в шестом знаке после десятичной точки, следовательно, при первом вычислении точность была  $10^{-5}$ , а не  $10^{-4}$  как выведено в сообщении. Информация, возвращаемая функцией fminbnd, содержит гарантированную точность, в то время как реальная точность может быть больше.

Аналогичным образом точность задается при нахождении корней и минимизации функции нескольких переменных.

В общем случае входные аргументы optimset задаются попарно

```
options = optimset(...,'вид_контроля', значение,...)
```

В двух предыдущих примерах Display и TolX — виды контроля, а 'final' и 1.0e-09, соответственно — их значения. Возможные сочетания параметров вид\_контроля и значение приведены в табл. 6.1.

*Таблица 6.1. Параметры optimset*

| Вид контроля | Значение | Результат                                                   | Примечание |
|--------------|----------|-------------------------------------------------------------|------------|
| Display      | 'off'    | Информация о вычислительном процессе не выводится           |            |
|              | 'iter'   | Выводится информация о каждом шаге вычислительного процесса |            |

Таблица 6.1 (окончание)

| Вид контроля | Значение                         | Результат                                                                      | Примечание                    |
|--------------|----------------------------------|--------------------------------------------------------------------------------|-------------------------------|
| Display      | 'final'                          | Выводится информация только о завершении вычислительного процесса              |                               |
|              | 'notify'                         | Выводится предупреждение, если процесс не сходится (используется по умолчанию) |                               |
| MaxFunEvals  | Целое число                      | Максимальное количество вызовов исследуемой функции                            | Только для fminbnd fminsearch |
| MaxIter      | Целое число                      | Максимальное количество итераций вычислительного процесса                      | Только для fminbnd fminsearch |
| TolFun       | Положительное вещественное число | Точность по функции для останова вычислений                                    | Только для fminbnd fminsearch |
| 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. В некоторых случаях приходится уменьшать точность, например, если вычисление исследуемой функции занимает много времени, а требуется получить только несколько первых значащих цифр ответа.

Пользователям, имеющим представление о численных методах, полезны сведения о ходе вычислений, выводимые на экран при значении параметра Display, равном 'iter'. Последовательность команд

```
>> options = optimset('Display', 'iter', 'TolX', 1.0e-09);
>> x2 = fminbnd(@ftest, -0.5, 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 и метод, применяемый на данной итерации.

| Func-count | x         | f(x)     | Procedure |
|------------|-----------|----------|-----------|
| 1          | -0.309017 | 0.873024 | initial   |

```

2 -0.190983 0.063294 golden
3 -0.118034 0.117247 golden
. . .
9 -0.162899 0.0275217 parabolic
10 -0.162899 0.0275217 parabolic

```

Функция `optimset` служит также для модификации существующей структуры. Например, если сначала была задана только точность по аргументу

```
>> options = optimset('TolX', 1.0e-09);
```

а в дальнейшем потребовалось дополнительно указать точность по функции и получить информацию о каждом шаге вычислительного процесса, то достаточно обратиться к функции `optimset` следующим образом:

```
>> options = optimset(options, 'TolFun', 1.0e-07, 'Display', 'iter');
```

Примите во внимание, что можно создать несколько структур с различными именами, к примеру: `options1` и `options2`, для различных вариантов управления вычислительным алгоритмом.

Для просмотра всех текущих опций можно вывести структуру `options` в командное окно или открыть ее в редакторе массивов двойным щелчком по строке с `options` в окне браузера переменных рабочей среды. Получение значения отдельного параметра производится при помощи функции `optimget`, входными аргументами которой являются имя структуры и название требуемого параметра:

```
>> err = optimget(options, 'TolX')
err =
1.0000e-009
```

Если данному параметру не было присвоено значение при генерации структуры, то возвращается пустой массив.

Вызов функции `optimset` без входных аргументов

```
>> optimset
```

приводит к отображению в командном окне названия всего множества параметров вместе с их допустимыми значениями. В фигурных скобках указаны значения, используемые в вычислительном алгоритме по умолчанию.

В разделе справочной системы MATLAB **Mathematics: Function Functions: Minimizing Functions and Finding Zeros** размещена подробная информация о минимизации функций и решении уравнений и, кроме того, приведены ссылки на литературу с описанием вычислительных методов, реализованных в MATLAB. Некоторые дополнительные возможности функций `fplot`, `fzero`, `fminbnd` и `fminsearch` обсуждаются в следующем разделе. В заключение этого раздела отметим, что большинство вычислительных функций, в

том числе `fzero`, `fminbnd` и `fminsearch`, написаны на языке программирования MATLAB. Они имеют открытый код и расположены в подкаталоге `\toolbox\matlab\funfun` основного каталога MATLAB.

## Более подробно о *fplot*

Мы уже отмечали, что `fplot` учитывает поведение функции за счет адаптивного выбора шага, что приводит к графикам, достаточно точно отображающим поведение исследуемой функции. Уделим теперь более пристальное внимание тем способам вызова `fplot`, которые позволяют настраивать адаптивный выбор шага, получать вместо графика таблицу значений функции и работать с математическими функциями, зависящими от одного или нескольких параметров.

До сих пор при использовании `fplot` мы задавали во входных аргументах имя файл-функции (указатель на нее, `inline` или анонимную функцию), пределы независимой переменной и свойства линии. Самый общий способ вызова `fplot` таков:

```
[X, Y] = fplot(@pfun, limits, tol, n, LineSpec, P1, P2, ...)
```

Следует учесть, что при указании необязательных выходных аргументов в `X` заносятся значения независимой переменной, в `Y` — соответствующие значения функции, а сам график не выводится. Таким образом, `fplot` позволяет получить таблицу значений функции, для визуализации которой достаточно выполнить `plot(X, Y)`.

Длина списка входных аргументов должна быть больше или равна двум, т. е. функция, вычисляющая значения математической функций, и вектор с пределами для построения графика задаются всегда. Отметим, что вектор `limits` может содержать как два элемента — пределы по оси абсцисс, так и четыре — пределы по оси абсцисс и ординат, например, `fplot(@sin, [-2*pi 2*pi -0.5 0.5])`. Алгоритм `fplot` адаптивно подбирает шаг, изменяя его вблизи участков быстрого роста или убывания функции так, чтобы обеспечить относительную точность  $2 \cdot 10^{-3}$ . При необходимости, точность можно задать в третьем входном аргументе `tol` (она должна быть меньше единицы). Сравните графики функции  $\sin(1/x)$  на отрезке  $[0.01, 1]$ , построенные с точностью  $2 \cdot 10^{-3}$  по умолчанию и с точностью  $10^{-4}$ .

```
>> fun = inline('sin(1/x)')
>> subplot(2, 1, 1)
>> fplot(fun, [0.01 1])
>> subplot(2, 1, 2)
>> fplot(fun, [0.01 1], 1.0e-4)
```

Другой способ управления алгоритмом `fplot` состоит в задании минимального числа разбиений исходного отрезка  $n$  (1 по умолчанию), причем  $n$  должно быть больше либо равно единице.

Пятый параметр `LineSpec` предназначен для определения типа линии, цвета и маркера. Поскольку точность `tol` может быть только меньше единицы, минимальное число разбиений  $n$  больше единицы, а `LineSpec` не является числом, то функция `fplot` допускает один из следующих вызовов с тремя входными аргументами

```
>> fplot(@pfun, limits, tol)
>> fplot(@pfun, limits, n)
>> fplot(@pfun, limits, LineSpec)
```

и сама выбирает нужную последовательность действий.

Разберем теперь назначение входных аргументов, которые указываются, начиная с шестой позиции списка после `LineSpec`. Исследуемая математическая функция может зависеть от нескольких параметров, например:

$y(x) = e^{p_1 x} - p_2 \sin x$ . Требуется построить графики функций для следующих пар значений:  $p_1 = 0.1$ ,  $p_2 = 2$ ;  $p_1 = 0.2$ ,  $p_2 = 2.5$ ;  $p_1 = 0.3$ ,  $p_2 = 3.7$ . Разумеется, можно написать три функции для вычисления  $y(x)$ , или каждый раз вносить изменения в одну и ту же функцию. Гораздо эффективнее запрограммировать функцию с тремя входными аргументами, первый из которых — независимая переменная, а остальные — параметры функции (листинг 6.3).

#### Листинг 6.3. Файл-функция для функции, зависящей от параметров

```
function f = pfun(x, p1, p2)
f = exp(p1*x) - p2*sin(x);
```

Теперь при вызове `fplot` следует указать текущие значения параметров в списке входных аргументов, начиная с шестой позиции. Для этого необходимо задать пять предыдущих аргументов, причем в качестве `tol`,  $n$  или `LineSpec` можно взять пустой массив — будут использоваться установленные по умолчанию значения. Удовлетворимся стандартной точностью и минимальным числом разбиений, определим только тип линии для каждой пары параметров:

```
>> fplot(@pfun, [-3 3], [], [], '- ', 0.1, 2);
>> hold on
>> fplot(@pfun, [-3 3], [], [], '-- ', 0.2, 2.5);
>> fplot(@pfun, [-3 3], [], [], ': ', 0.3 ,3.7);
```

Возможность работы с функциями, зависящими от параметров, реализована не только в `fplot`. Все обсуждаемые в этой главе вычислительные задачи допускают наличие параметров. В следующем разделе мы продемонстрируем минимизацию и нахождение корней таких функций.

## Исследование функций, зависящих от параметров

Функции `fzero`, `fminbnd` и `fminsearch` поддерживают работу с математическими функциями, зависящими от параметров. Заметим, что версия MATLAB 7.0 унаследовала от предыдущей тот же подход, что остался в `fplot`, хотя его описание исключено из справочной системы. Значения параметров последовательно задаются в списке входных аргументов после управляющей структуры `options`. Заполнение самой структуры не обязательно, если при решении уравнения или минимизации функции достаточно определенных по умолчанию значений '`Display`', '`MaxFunEvals`', '`MaxIter`', '`TolFun`', '`TolX`'. Как и в случае с `fplot` допускается указание пустого массива вместо неинтересующей нас управляющей структуры, например

```
>> [x1, f1] = fzero(@pfun, 0, [], 0.1, 2)
x1 =
 0.5570
f1 =
 0
```

Аналогичным образом решается задача о поиске локального минимума функции, зависящей от параметров.

В версии 7.0 рекомендуется применять подходы, основанные на использовании анонимных или вложенных функций. В первом случае достаточно задать значения параметров в переменных рабочей среды и обратиться к функции, определив в качестве аргумента анонимную функцию:

```
>> p1 = 0.1;
>> p2 = 2;
>> [x1, f1] = fzero(@(x) exp(p1*x) - p2*sin(x), 0)
x1 =
 0.5570
f1 =
 0
```

Поскольку при создании анонимной функции переменные среды фиксируются, то нет смысла определять отдельно анонимную функцию и передавать ее имя в качестве аргумента, ибо для новых параметров все команды надо повторить. В листинге 6.4 приведен пример использования вложенных функций для решения той же задачи.

**Листинг 6.4. Использование функции, зависящей от параметра, в качестве аргумента `fzero`**

```
function [x, y] = froot(p1, p2, x0)
[x, y] = fzero(@pfun, x0);
function f = pfun(x)
f = exp(p1*x) - p2*sin(x);
end
end
```

Теперь вместо вызова функции `fzero` следует обращаться к собственной функции `froot`:

```
>> [x, y] = froot(0.1, 2, 0)
x =
0.5570
y =
0
```

При необходимости можно добавить управляющую структуру `options` в список входных аргументов для передачи ее `fzero`. Пользователь пакета может выбрать наиболее приемлемый для него вариант.

## Интегрирование функций

Этот раздел посвящен приближенному вычислению 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 функции, вычисляющей подынтегральное выражение, ее текст приведен в листинге 6.5.

#### Листинг 6.5. Файл-функция, вычисляющая подынтегральное выражение

```
function f = fint(x)
f = exp(-x).*sin(x);
```

Теперь для вычисления интеграла вызовите `quad`, задав первым аргументом ссылку на функцию `fint`, а вторым и третьим — нижний и верхний пределы интегрирования. В качестве выходного аргумента можно указать имя переменной, которую следует записать найденное значение:

```
>> format long
>> I = quad(@fint, -1, 1)
I = -0.66349146785310
```

#### Примечание

Подынтегральная функция может быть задана именем, либо определена как inline-функция или анонимная так же, как и в случае `fzero`, `fminsearch`, `fminbnd`.

По умолчанию функция `quad` вычисляет приближенное значение интеграла с точностью  $10^{-6}$ . Для изменения точности вычислений следует задать дополнительный четвертый аргумент:

```
>> I = quad(@fint, -1, 1, 1.0e-07)
I =
-0.66349366574399
```

Реализованный в `quad` рекурсивный алгоритм основан на квадратурной формуле Симпсона с автоматическим подбором шага интегрирования для достижения требуемой относительной погрешности. Сравнивая полученные результаты, можно сделать вывод, что относительная погрешность  $10^{-6}$  не дает ожидаемого количества верных знаков после десятичной точки. Проблема заключается в способе оценки погрешности вычислений в применяемом алгоритме, но этот вопрос из области вычислительной математики выходит за рамки излагаемого материала. При использовании средств пакета необходимо просто учитывать его особенности. Если пользователь знаком с теоретическими аспектами применяемого метода, то ему окажется полезной трассировка вычислений, которая выводится в командное окно при ненулевом значении пятого входного аргумента `quad`. Информация о ходе вычислений представлена в виде таблицы. В каждой ее строке приведены значе-

ния количества вычислений подынтегральной функции, начальная точка текущего промежутка интегрирования, его длина и значение интеграла:

```
>> I = quad(@fint, -1, 1, 1.0e-07, 3)
 9 -1.00000000000 5.43160000e-001 -0.7696256630
 11 -1.00000000000 2.71580000e-001 -0.4927577867
 13 -1.00000000000 1.35790000e-001 -0.2772168074
 15 -0.8642100000 1.35790000e-001 -0.2155409801
 . . .
 37 0.4568400000 5.43160000e-001 0.1696814429
 39 0.4568400000 2.71580000e-001 0.0830983395
 41 0.7284200000 2.71580000e-001 0.0865830829
```

Проанализируйте количество точных знаков в зависимости от задаваемой точности вычислений в `quadl` для примера, разобранного выше.

Функции `quad` и `quadl` могут выводить следующие диагностические сообщения:

- ❑ 'minimum step size reached' — шаг интегрирования достиг минимального значения, но требуемая точность не получена. Возможно, функция имеет неинтегрируемую особенность;
- ❑ 'maximum function count exceeded' — достигнут предел максимального количества вычислений значений подынтегральной функции (более 10 000). Возможно, функция ведет себя как функция с неинтегрируемой особенностью;
- ❑ 'Infinite or Not-a-Number function value encountered' — произошло переполнение при вычислении интегranda во внутренней точке интервала интегрирования.

Например, при интегрировании функции  $y = x^{-0.9}$ , имеющей интегрируемую особенность, по отрезку  $[0, 1]$

```
>> fun = inline('x.^-0.9');
>> I = quad(fun, 0, 1, 1e-7)
Warning: Minimum step size reached; singularity possible.
> In quad at 88
I =
 9.8955
```

в командное окно выводится сообщение о достижении минимально возможного шага интегрирования, причем делается вывод о возможной осо-

бенности подынтегральной функции. Диагностическое сообщение сопровождается номером строки файл-функции `quad`, выполнение которой привело к предупреждению.

Очевидно, что полученный ответ 9.8955 не соответствует заданной точности — обращение к `quad1` также не улучшит ситуацию. Причина заключается в методах, реализованных в MATLAB для вычисления определенных интегралов. Численное интегрирование функций с особенностями может быть основано на квадратурных формулах, узлы которых не являются границами интервала, или на квадратурных формулах с весовой функцией для выделения особенности. Такой алгоритм вы сможете запрограммировать самостоятельно после чтения главы 7, посвященной конструкциям языка программирования MATLAB. В главе 8 рассмотрено написание собственных файл-функций для вычисления определенных интегралов на основе других квадратурных формул.

После вывода предупреждения о том, что функция может иметь особенность, вычисления продолжаются. Попробуйте вычислить интеграл от функции  $y=1/x$  по отрезку  $[0, 1]$ , на котором она имеет неинтегрируемую особенность.

### Примечание

При интегрировании негладких функций (типа  $|x|$ , или имеющих интегрируемый разрыв) интервал интегрирования следует разбивать на части так, чтобы на подинтервалах функция была гладкой. Вычисление интегралов от быстроменяющихся функций так же возможно при разбиении исходного промежутка на участки монотонности подынтегральной функции.

## Вычисление двойных интегралов

В MATLAB определена 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

$$\int_0^1 \int_{-\pi}^{\pi} \left( e^x \sin y + e^{-x} \cos y \right) dx dy .$$

Функция `int2` должна содержать два входных аргумента `x` и `y`, ее текст приведен в листинге 6.6.

**Листинг 6.6. Файл-функция, вычисляющая подынтегральную функцию**

```
function f = fint2(x, y)
f = exp(x).*sin(y).^2 + exp(-x).*cos(y).^2;
```

**Примечание**

Вместо файл-функции можно использовать анонимные или inline-функции.

Функция dblquad имеет пять входных 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. Поскольку в dblquad реализованы двумерные квадратурные формулы, основанные на одномерных, то при вычислении двойного интеграла dblquad использует quad. Если седьмым аргументом указать 'quadl', то будет применяться соответствующий алгоритм

```
>> format long
>> dblquad(@fint2, -pi, pi, 0, 1, 1.0e-012, 'quadl')
ans =
23.09747871451549
```

В случае гладких подынтегральных функций задание 'quadl' позволяет сократить время счета.

## Вычисление некоторых интегралов

В данном разделе разобрано несколько примеров: интегрирование функции, зависящей от параметров, и вычисление интеграла с переменным верхним пределом.

### Интегралы, зависящие от параметра

Функции quad и quadl позволяют находить значения интегралов, зависящих от параметров. Приемы передачи параметров те же, что и для функции fzero. Опишем ту возможность, которая осталась недокументированной в последней версии пакета. Аргументами функции, вычисляющей подынтег-

гральное выражение, должна быть не только переменная интегрирования, но и все параметры. Значения параметров указываются через запятую, начиная с шестого аргумента `quad` или `quadl`. Вычислите интеграл

$$\int_{-1}^1 (p_1 x^2 + p_2 \sin x) dx$$

при значениях параметров  $p_1 = 22.5$ ,  $p_2 = -5.9$  по квадратурным формулам Ньютона—Котеса с автоматическим выбором шага.

Текст функции `fparam`, зависящей от трех аргументов, которая вычисляет значение подынтегральной функции, приведен в листинге 6.7.

#### Листинг 6.7. Параметрически заданная функция

```
function z = fparam(x, Par1, Par2)
z = Par1.*x.^2 + Par2.*sin(x);
```

Найдите значение интеграла при помощи `quad`, использование `quadl` производится аналогично.

```
>> q = quad(@fparam, -1, 1, 1.0e-05, 1, 22.5, -5.9)
q =
15.00000005742834
```

Пятый параметр, равный единице, поставлен для вывода в командное окно таблицы с информацией о ходе рекурсивного алгоритма интегрирования. Если нужно получить только значение интеграла, то пятый параметр должен быть нулем. Заметьте, что если пропустить этот параметр, то появится сообщение об ошибке, т. к. функция `fparam` предполагает наличие трех входных аргументов. Итак, при вычислении интеграла, зависящего от параметров, их следует указывать, начиная с шестого аргумента `quad` или `quadl`.

Также для вычисления интеграла, зависящего от параметров, может быть использована анонимная функция:

```
>> Par1 = 22.5;
>> Par2 = -5.9;
>> q = quad(@(x) Par1.*x.^2 + Par2.*sin(x), -1, 1, 1.0e-05)
q =
15
```

В качестве упражнения примените вложенную функцию для решения этой задачи.

## 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 функции: fint — для подынтегральной функции, и Fy — которая находит значения интеграла для каждого значения  $y$ . Тексты требуемых функций приведены в листинге 6.8, причем Fy является основной функцией, а fint — подфункцией.

### Листинг 6.8. Файл-функция, вычисляющая значение интеграла

```
function f = Fy(y)
f = quadl(@fint, 0, y, 1.0e-09);
function f = fint(x)
f = exp(x).* (sin(x) - cos(x));
```

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

Текущий раздел посвящен операциям с полиномами и решению задачи интерполяции при построении полинома, наилучшим образом приближающего двух- и трехмерные табличные данные.

## Операции с полиномами

### Умножение, деление, сложение и вычитание

Напомним, что 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).

Умножение двух полиномов осуществляется при помощи `conv`. Например, для вычисления произведения  $s(x)$  полиномов

$$p(x) = x^5 + x^3 + 1 \quad \text{и} \quad q(x) = x^2 + 2x + 3$$

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  $s$

$$s(x) = x^7 + 2x^6 + 4x^5 + 2x^4 + 3x^3 + x^2 + 2x + 3.$$

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 "+" 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 6.9 приведен текст файл-функции `polysum`, находящей значение суммы полиномов.

**Листинг 6.9. Файл-функция polysum, вычисляющая сумму полиномов**

```

function s = polysum(p, q)
% вычисление суммы полиномов, заданных векторами коэффициентов

% нахождение наибольшей из длин входных векторов
maxlen = max(length(p), length(q));
% создание вспомогательных векторов p1 и q1 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**

Функция polyder 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.

## Интерполярование и сглаживание

Табличные данные очень часто удобно интерпретировать как некоторую функцию, в частности, полиномиальную или *сплайн* — гладкую функцию, которая на отрезках области определения равна полиномам определенной степени. Будем различать два критерия приближения табличных функций: *интерполярование*, при котором аппроксимирующая функция совпадает с табличной в узлах, и *сглаживание*, основанное на минимизации некоторого критерия, например, суммы квадратов отклонений в узлах. Итак, возникает задача о построении полиномиальной или кусочно-полиномиальной функции для приближения некоторых дискретных данных. Набор вычислительных функций MATLAB содержит функции для решения таких задач, как в случае одномерных, так и многомерных данных. Одним из простейших способов приближения табличных функций, предлагаемым MATLAB, является полиномиальное сглаживание методом наименьших квадратов.

### Приближение по методу наименьших квадратов

Пусть исследуемая функция дана в виде табл. 6.2. Построение полинома фиксированной степени для приближения табличной функции одной переменной производится при помощи `polyfit`. Первые два ее входные аргу-

мента являются векторами со значениями абсцисс и ординат табличной функции, а третий — требуемой степенью полинома.

**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 6.10.

**Листинг 6.10. Приближение полиномами методом наименьших квадратов**

```
% заполнение массивов для 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')
% вычисление коэффициентов полиномов разных степеней,
% приближающих табличную функцию по методу наименьших
% квадратов
p4 = polyfit(x, y, 4); % нахождение коэффициентов полинома 4-ой степени
p5 = polyfit(x, y, 5); % нахождение коэффициентов полинома 5-ой степени
p6 = polyfit(x, y, 6); % нахождение коэффициентов полинома 6-ой степени
% 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. 6.4. Приближение методом наименьших квадратов дает хороший результат не всегда, более того, при увеличении степени возможно

ухудшение приближения, как происходит, например, при  $n = 6$ . В нашем случае полином шестой степени является интерполяционным полиномом, значения которого совпадают со значениями табличной функции. Тем не менее такое приближение 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.

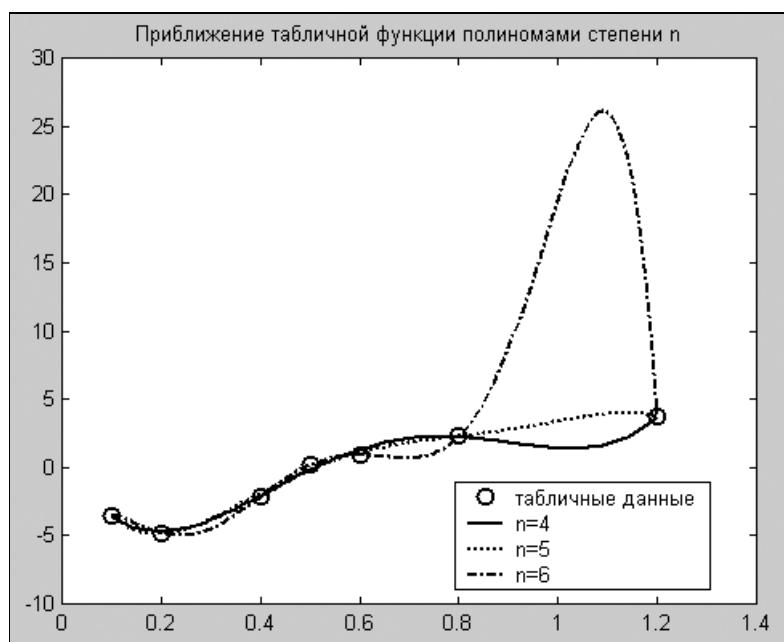


Рис. 6.4. Приближение полиномами по методу наименьших квадратов (`polyfit`)

Приближение табличных функций по методу наименьших квадратов является простейшей задачей подбора параметров. Более детально этот класс задач разобран в главе 19, посвященной описанию Curve Fitting Toolbox.

## Интерполяция сплайнами

Самым простым способом интерполяции является аппроксимация данных сплайном нулевого порядка (на каждом участке степень полинома равна нулю), при которой значение в каждой промежуточной точке принимается равным ближайшему значению, заданному в таблице. В результате данные

приближаются ступенчатой функцией, а само приближение называется *интерполяцией по соседним точкам*. *Линейная интерполяция* основана на соединении соседних точек отрезками прямых — табличные данные приближаются ломаной линией (сплайн первого порядка дефекта единицы). Для получения более гладкой функции следует применять *интерполяцию кубическими сплайнами*. Все эти способы реализованы в функции `interp1`, для использования которой следует задать координаты абсцисс промежуточных точек, в которых вычисляются значения интерполянта, и способ интерполяции:

- 'nearest' — приближение по соседним элементам;
- 'linear' — линейная интерполяция (применяется по умолчанию, если способ интерполяции не задан);
- 'spline' — интерполяция кубическими сплайнами;
- 'pchip' — интерполяция кубическими эрмитовыми сплайнами.

При использовании `pchip` вторая производная сплайна может быть разрывна (в этом случае говорят, что сплайн имеет дефект два) в отличие от `spline`. Однако кубический эрмитовый сплайн сохраняет участки монотонности исходных данных.

Выходным аргументом `interp1` является вектор значений интерполянта в промежуточных точках. Текст файл-программы `interp1dem` для сравнения различных способов интерполяции приведен в листинге 6.11.

#### Листинг 6.11. Файл-программа `interp1dem`

```
% заполнение массивов для 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 интерполяции
xi = [x(1):0.01:x(length(x))];
% вычисление ступенчатой функции в промежуточных точках
ynear = interp1(x, y, xi, 'nearest');
% вычисление кусочно-линейной функции в промежуточных точках
yline = interp1(x, y, xi, 'linear'); % можно не указывать 'linear'
% вычисление кубического сплайна в промежуточных точках
yspline = interp1(x, y, xi, 'spline');
hold on
```

```
% построение графиков интерполянтов
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. 6.5, который наглядно демонстрирует способы интерполяции. Добавьте самостоятельно график кубического Эрмитова сплайна, указав '`rchip`' в качестве четвертого аргумента функции `interp1`.

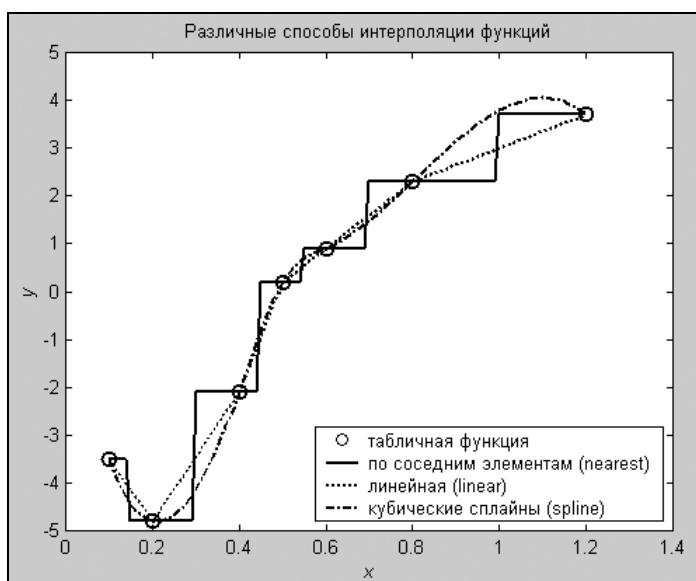


Рис. 6.5. Различные способы аппроксимации функций (`interp1`)

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 ин-

терполирования 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' или 'linear' — билинейная интерполяция (применяется по умолчанию, если способ интерполирования не задан);
- 'bicubic' или 'cubic' — интерполяция бикубическими сплайнами;
- 'spline' — интерполяция кубическими сплайнами.

### Примечание

Для ускорения вычислений в случае *равноотстоящих точек с монотонно изменяющимися координатами*, соответствующих табличным данным, используются, соответственно, аргументы '`*nearest`', '`*linear`', '`*cubic`' или '`*spline`'.

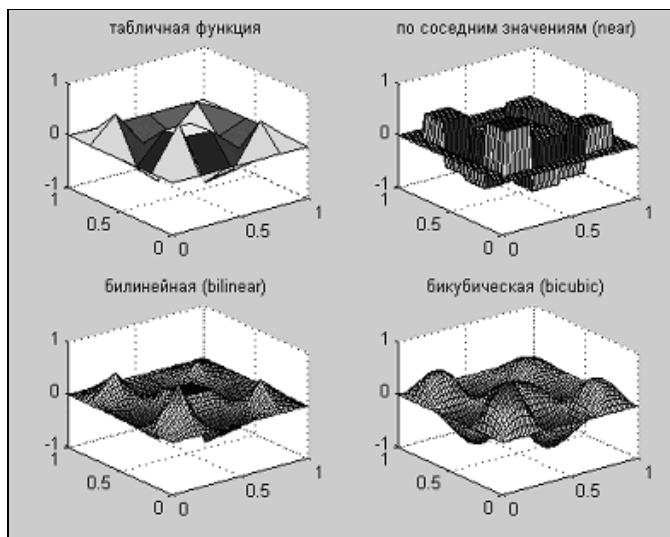
Сравнение первых трех вышеперечисленных способов интерполяции может быть осуществлено при помощи файл-программы `interp2dem`, текст которой приведен в листинге 6.12. Для избежания утомительного ввода таблицы двумерных данных они генерируются при помощи некоторой функции двух переменных.

#### Листинг 6.12. 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 (можно не указывать 'bilinear')
ZiBiLin = interp2(X, Y, Z, Xi, Yi, 'bilinear');
% бикубическая интерполяция
ZiBiCub = interp2(X, Y, Z, Xi, Yi, 'bicubic');
```

```
% вывод результатов
subplot(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. 6.6 приведены графики, полученные при помощи `interp2dem`. Графики отражают поведение интерполирующих функций, полученных различными способами.



**Рис. 6.6.** Различные способы  
приближения двумерных данных (`interp2`)

Для аппроксимации трехмерных данных служит функция `interp3`, для многомерных — `interpN`. Создание многомерных сеток осуществляется функцией `ndgrid`. Многомерное приближение производится аналогично двумерному, подробнее о нем можно узнать из справочной системы по MATLAB, набрав в разделе **Index** имя функции `interpN`.

В этом разделе разобраны самые простые способы приближения табличных функций сплайнами различных порядков. Применению сплайнов посвящен Spline Toolbox (см. главу 18).

## 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, специальные 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. Причем, как матрица, так и вектор правой части могут быть комплексными. 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. В справочной системе приведен алгоритм решения систем линейных урав-

нений, который учитывает вид матрицы системы уравнений. Кратко этот алгоритм описывается следующим образом.

1. Если матрица квадратная и ленточная с ограниченной шириной ленты, то применяется специальный алгоритм решения для ленточных матриц.
2. Если матрица треугольная, то используется метод подстановки.
3. Если матрица перестановками строк и столбцов сводится к треугольной, то применяется модификация метода подстановки.
4. Если матрица симметричная (эрмитова в комплексном числовом пространстве) с положительными элементами на диагонали, то вызывается функция `chol`, которая пытается выполнить разложение Холецкого. При этом в процессе разложения выясняется положительная определенность матрицы. В случае положительно определенной матрицы находится множитель разложения Холецкого, с которым решаются две системы методом подстановки для получения решения исходной системы уравнений. Если матрица системы разрежена, то перед разложением Холецкого производится симметричная перестановка строк и столбцов по алгоритму минимальной степени с целью уменьшения заполнения множителей Холецкого. Для выполнения этой операции вызывается функция `symmd`.
5. Если матрица хессенбергова, но не разреженная, то применяется метод сведения к системе с треугольной матрицей.
6. Если матрица квадратная и не удовлетворяет условиям пп. 1—5, то используется метод Гаусса в форме LU-факторизации. Для разреженных матриц предварительно переставляются строки и столбцы с целью уменьшения заполнения матричных множителей LU-разложения.
7. Если матрица прямоугольная, то происходит обращение к функции `qr` для QR-факторизации матрицы, которая позволяет найти наилучшее решение системы в смысле наименьших квадратов. Алгоритм `qr` учитывает, разреженная матрица, или нет.

Функции MATLAB, используемые в вышеописанном алгоритме, основаны на процедурах популярных библиотек LAPACK и UMFPACK для решения задач линейной алгебры. Соответствующая информация содержится в справочной системе MATLAB. Для ее получения достаточно обратиться к разделу **MATLAB: Functions -- Categorical List: Mathematics: Linear Algebra** и перейти в нем по гиперссылке `\ and /`. При этом открывается страница справочной системы, пункт **Algorithm** которой посвящен обсуждаемому здесь алгоритму. В нем упомянуты те функции библиотеки LAPACK, к которым обращается MATLAB при решении систем линейных уравнений. В ряде случаев, например, для разреженных матриц, MATLAB вызывает про-

цедуры библиотеки UMFPACK. Внизу страницы расположены две гиперссылки для доступа к документации этих библиотек в Интернете.

В главе 15 рассматриваются особенности решения систем с разреженными матрицами.

## Системы с плохо обусловленными матрицами

Рассмотрим простой пример: требуется найти решение следующей системы:

$$\begin{cases} 2x_1 + 3x_2 + 3x_3 = 8; \\ 4x_1 + 2x_2 + 3x_3 = 7; \\ 6x_1 + 5x_2 + 6x_3 = 7. \end{cases}$$

Ведите матрицу и вектор из командной строки и примените обратную ко-  
сую 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 вырожде-  
на или плохо обусловлена, и затем полученное 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 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 решения системы линейных алгебраических уравнений с приемлемой точностью определяется *числом обусловленности* матрицы, характеризующим чувствительность решения к ошибкам вычисления и точности представления данных в компьютере. Если оно велико, то ответ может получиться не очень точным или даже неверным. Перед решением системы имеет смысл вычислить число обусловленности матрицы системы при помощи 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.

### Примечание

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 найденное решение системы может быть неправильным.

Знак обратной косой черты может быть применен для одновременного решения нескольких систем с одной и той же матрицей и разными правыми частями. Для этого из вектор-столбцов правых частей следует сформировать матрицу и поместить ее после знака обратной косой черты. Результатом является матрица, каждый столбец которой есть решение соответствующей системы линейных уравнений. В отличие от последовательного решения систем, такой подход существенно экономит время, поскольку разложение матрицы выполняется только один раз.

## Переопределенные и недоопределенные системы

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 системы встречаются в задачах подбора нескольких параметров с целью удовлетворения соотношениям, число которых больше, чем число параметров.

Рассмотрим задачу о подборе параметров  $a$  и  $b$  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 ни 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

$$A = \begin{bmatrix} e^{-t_1} & t_1 \\ e^{-t_2} & t_2 \\ \vdots & \vdots \\ e^{-t_6} & t_6 \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_6 \end{bmatrix}.$$

Для нахождения неизвестных следует решить эту систему в MATLAB при помощи знака обратной косой черты. Напишите самостоятельно файл-функцию без параметров, решающую данную систему и строящую маркерами график исходных данных и линией график физического закона с полу-

чившимися параметрами. Физический закон лучше оформить в виде функции от трех аргументов — переменной  $t$  и параметров  $a$ ,  $b$ . Текст файл-функции `fit` с вложенной функцией `law` содержится в листинге 6.13.

### Листинг 6.13. Файл-функция `fit` подбора параметров

```
function [a, b]=fit

function y = law(t, a, b)
y = a.*exp(-t) + b.*t;
end

% задание таблицы с экспериментальными данными
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];
% создание матрицы системы линейных уравнений
A = [exp(-t) t];
% решение системы и получение искомых коэффициентов физического закона
x = A\y;
a = x(1);
b = x(2);
% построение графика экспериментальных данных
plot(t, y, 's')
hold on
% построение графика физического закона
fplot(@law, [0 0.5], [], [], '- ', a, b)
end
```

Файл-функция `fit` строит изображенные на рис. 6.7 графики данных и физического закона при найденных значениях параметров, которые она возвращает в выходных аргументах

```
>> [a, b] = fit
>> a =
 4.2478
b =
 0.9070
```

Внесите самостоятельно в файл-функцию `fit` некоторые изменения. Пусть табличные данныечитываются из текстового файла функцией `load`, а на график наносится вся необходимая информация.

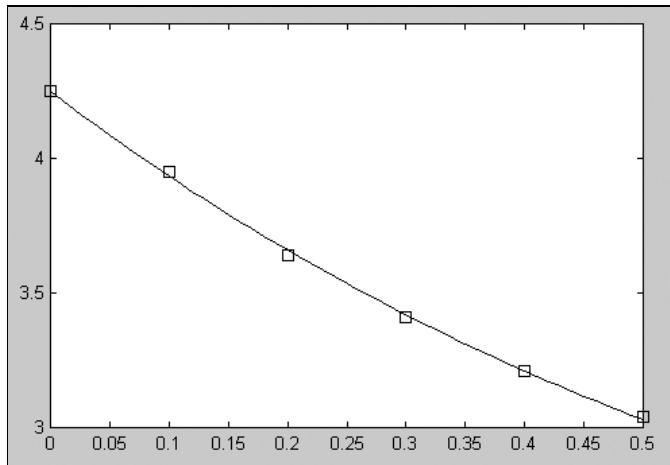


Рис. 6.7. Подбор параметров физического закона

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.

Для решения нескольких переопределенных или недоопределенных систем с одной и той же матрицей применим тот же самый подход, что и при решении систем с квадратной матрицей, описанный в конце предыдущего раздела.

## Решение систем при помощи функции *linsolve*

При использовании знака обратной косой черты для решения системы линейных уравнений выбор метода решения остается за MATLAB. Более широкие возможности предоставляет функция *linsolve*, которая, так же как и знак обратной косой черты, позволяет решать систему или несколько систем с одной и той же необязательно квадратной матрицей  $AX = B$ , и при этом допускает выбор метода ее решения в соответствии со свойствами

матрицы, указанными пользователем. В простейшем случае обращение к функции `linsolve` имеет вид:

```
X = linsolve(A, B)
```

При этом для решения системы в случае квадратной матрицы  $A$  применяется LU-разложение с выбором ведущего элемента, а для прямоугольной — QR-разложение. Рассмотрим, как повлиять на выбор более эффективного метода решения, учитывая свойства матрицы. Для этого следует сформировать управляющую структуру, содержащую поля со значениями `true` или `false` в зависимости от свойств матрицы, и указать ее в качестве третьего входного аргумента функции `linsolve`. Будем, как и ранее, использовать для управляющей структуры имя `options`, хотя оно может быть произвольным, более того, если решается много систем с разными свойствами, целесообразно создать несколько управляющих структур, используя нужную в конкретном случае. Поле `TRANSA` определяет, как задана матрица системы: значение `true` означает, что солверу передается транспонированная матрица системы.

Например, если введена матрица  $A$  и надо решать систему  $A^T X = B$ , то следует обратиться к солверу следующим образом:

```
>> options.TRANSB = true;
>> y = linsolve(A, B, options)
```

Параметры, определяющие свойства матрицы системы, сведены в табл. 6.4.

**Таблица 6.4. Флаги, определяющие свойства матрицы системы**

| Флаг   | Свойство матрицы                          |
|--------|-------------------------------------------|
| LT     | Нижняя треугольная                        |
| UT     | Верхняя треугольная                       |
| UHESS  | Верхняя почти треугольная (хессенбергова) |
| SYM    | Симметричная                              |
| POSDEF | Положительно определенная                 |
| RECT   | Прямоугольная                             |

В табл. 6.5 представлены возможные комбинации значений флагов в управляющей структуре. Некоторые комбинации очевидны, например, хессенбергова матрица не может быть треугольной, а некоторые, которые могли бы совмещаться, не допустимы. Например, треугольная матрица может быть

положительно определенной, но при выборе способа решения определяющим фактором является ее профиль.

**Таблица 6.5. Возможные комбинации флагов**

| LT    | UT    | UHESS | SYM   | POSDEF | RECT       |
|-------|-------|-------|-------|--------|------------|
| true  | false | false | false | false  | true/false |
| false | true  | false | false | false  | true/false |
| false | false | true  | false | false  | false      |
| false | false | true  | true  | true   | true       |
| false | false | false | false | false  | true/false |

### ◀ Предупреждение ▶

Функция `linsolve` не проверяет правильность свойств матрицы, которые указаны в управляющем параметре, и не выводит никаких предупреждений. Поэтому результат может быть неверный.

Если матрица системы плохо обусловлена или ее ранг меньше размерности, то выводится предупреждение. Например, решение системы с линейно зависимыми строками дает результат:

```
> A = [2 3 3
 -2 -3 -3];
>> b = [8; 7];
>> X = linsolve(A, b)
Warning: Rank deficient, rank = 1, tol = 2.8262e-015.
X =
 0
 0.1667
 0
```

Сообщение можно подавить, если задать второй выходной аргумент, в который записывается ранг матрицы (в случае прямоугольной матрицы) или величина, обратная к числу обусловленности (в случае квадратной матрицы).

```
>> [X, r] = linsolve(A, b)
X =
 0
```

```
0.1667
0
r =
1
```

Если предупреждения подавляются, то следует использовать значение выходного аргумента для оценки правильности получаемого решения.

## 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 размера три самостоятельно, убедитесь в ее невырожденности при помощи `det` и найдите обратную, 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 существенно больше времени и памяти, к тому же он может дать большую погрешность решения. Поэтому для решения систем следует применять знак обратной косой черты или функцию `linsolve`.

Функция `pinv` позволяет найти *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*  $\lambda_i$  и *собственные векторы*  $u_i \neq 0$  квадратной матрицы  $A$  удовлетворяют равенствам  $A \cdot u_i = \lambda_i u_i$ . Функция `eig`, вызванная с входным аргументом матрицей, находит все собственные числа матрицы и записывает их в выходной аргумент — вектор:

```
>> 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` представляет собой матрицу, столбцы которой являются собственными векторами. 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
```

Проверьте, правильно ли найдены, например, второе собственное число и соответствующий ему собственный вектор. Воспользуйтесь определением:

```
>> A*U(:, 2) - Lam(2, 2)*U(:, 2)
```

```
ans =
1.0e-015 *
 0.4441
-0.8882
```

Возведение квадратной матрицы в любую целую степень осуществляется при помощи знака `^`, например:

```
>> B = A^2
```

```
B =
13 21
21 34
```

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 операции `.^`. Для отрицательных степеней вычисляется степень обратной матрицы. Возможно использование дробных степеней. Если требуется извлечь корень из квадратной матрицы, то лучше применить функцию `sqrtm`. Матричные экспоненты и логарифм вычисляются при помощи функций `expm` и `logm`.

Пользователь может найти значение любой функции от матрицы. Для этого следует создать собственную файл-функцию, анонимную функцию или inline-функцию, а затем использовать специальную функцию `funm`, задав первым аргументом матрицу, а вторым имя файл-функции в апострофах или указатель на функцию. Вычислите, например  $e^A \cdot \sin A$ . Текст соответствующей файл-функции `matrf` приведен в листинге 6.14. При создании файл-функции обязательно использование поэлементных операций!

#### Листинг 6.14. 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 другие) можно вычислять от матриц аналогичным образом, например

```
>> B = funm(A, 'sin')
```

Предпочтительнее производить вычисления функций от матриц с использованием специальных матричных функций `expm`, `logm`, `sqrtm`, т. е., например, `expm(A)` вместо `funm(A, 'exp')` и т. д. В этих 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

$$\frac{dX}{dt} = A \cdot X, \quad X(0) = X_0$$

может быть найдено по формуле  $X(t) = X_0 \cdot e^{tA}$ . Решению дифференциальных уравнений и систем посвящен следующий раздел.

# Решение дифференциальных уравнений

Данный раздел посвящен описанию возможностей, предоставляемых MATLAB, для численного решения задач Коши и краевых задач для обыкновенных дифференциальных уравнений произвольного порядка и систем, включая краевые задачи с неизвестными параметрами и вырождающимися коэффициентами. Рассматривается также решение дифференциальных уравнений с запаздывающим аргументом. Для решения этих задач предназначены специальные функции MATLAB, в вычислительной математике их называют *сольверы*. MATLAB имеет достаточно большой набор сольверов, основанных на различных численных методах.

## Решение задачи Коши

Для решения задачи Коши в MATLAB существует семь сольверов: `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t` и `ode23tb`. Методика их использования одинакова, включая способы задания входных и выходных аргументов. В достаточно общем случае вызов сольвера для решения задачи Коши производится следующим образом (здесь под `solver` понимается один из перечисленных выше сольверов):

```
[T, Y] = solver(odefun,interval,Y0,options)
```

где `odefun` — функция для вычисления вектор-функции правой части системы уравнений, `interval` — массив из двух чисел, задающий промежуток для решения уравнения, `Y0` — заданный вектор начальных значений искомой вектор-функции, `options` — структура для управления параметрами и ходом вычислительного процесса. Сольвер возвращает массив `T` с координатами узлов сетки, в которых найдено решение, и матрицу решений `Y`, каждый столбец которой является значением компоненты вектор-функции решения в узлах сетки.

Далее мы обсудим применение сольверов и управление процессом решения на ряде показательных примеров из перечисленных выше классов задач.

Задача Коши для дифференциального уравнения состоит в нахождении функции, удовлетворяющей дифференциальному уравнению произвольного порядка

$$y^{(n)} = f(t, y, y', \dots, y^{(n-1)})$$

и начальным условиям при  $t = t_0$

$$y(t_0) = u_0, \quad y'(t_0) = u_1, \dots, \quad y^{(n-1)}(t_0) = u_{n-1}.$$

Схема решения таких задач в MATLAB состоит из следующих этапов.

1. Приведение дифференциального уравнения к системе дифференциальных уравнений первого порядка (если изначально задана система, то в этом нет необходимости).
2. Написание специальной функции для системы уравнений.
3. Вызов подходящего солвера.
4. Визуализация результата.

Разберем решение дифференциальных уравнений на примере задачи о колебаниях материальной точки под воздействием внешней силы в среде, оказывающей сопротивление колебаниям. Перемещение точки в среде описывается уравнением второго порядка

$$y'' + 2y' + 10y = \sin t.$$

Предположим, что координата точки в начальный момент времени равнялась единице, а скорость — нулю. Тогда соответствующие начальные условия имеют вид

$$y(0) = 1, \quad y'(0) = 0.$$

Пример носит демонстрационный характер, поэтому размерности физических величин указываться не будут. Теперь исходную задачу надо привести к системе дифференциальных уравнений. Для этого вводят столько вспомогательных функций, сколько порядок уравнения. В данном случае необходимы две вспомогательные функции  $y_1$  и  $y_2$ , определяемые формулами

$$y_1 = y, \quad y_2 = y'.$$

Несложно догадаться, что система дифференциальных уравнений с начальными условиями, требуемая для дальнейшей работы, такова

$$\begin{cases} y_1' = y_2; \\ y_2' = -2y_2 - 10y_1 + \sin t; \end{cases} \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \quad (6.4)$$

Второй этап состоит в написании функции для системы дифференциальных уравнений. 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 функции является вектор правой части системы. Текст функции `oscil` для разбираемого примера приведен в листинге 6.15.

Решите задачу, используя, например, солвер `ode113`. Входными аргументами солверов в самом простом случае являются: указатель на функцию (или ее имя в апострофах), вектор с начальным и конечным значением времени наблюдения за процессом и вектор начальных условий. Выходных аргументов два: вектор, содержащий значения времени, и матрица значений искомых 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 второй столбец — значения ее производной. Как правило, размеры матрицы и вектора достаточно велики, поэтому лучше сразу отобразить результат на графике. Применение солвера для нахождения решения при  $t \leq 15$  и визуализация результата продемонстрированы на примере файл-функции `solvdem`, приведенной в листинге 6.15, где функция `oscil` реализована как подфункция.

#### Листинг 6.15. Файл-функция `solvdem` для решения дифференциального уравнения (6.4)

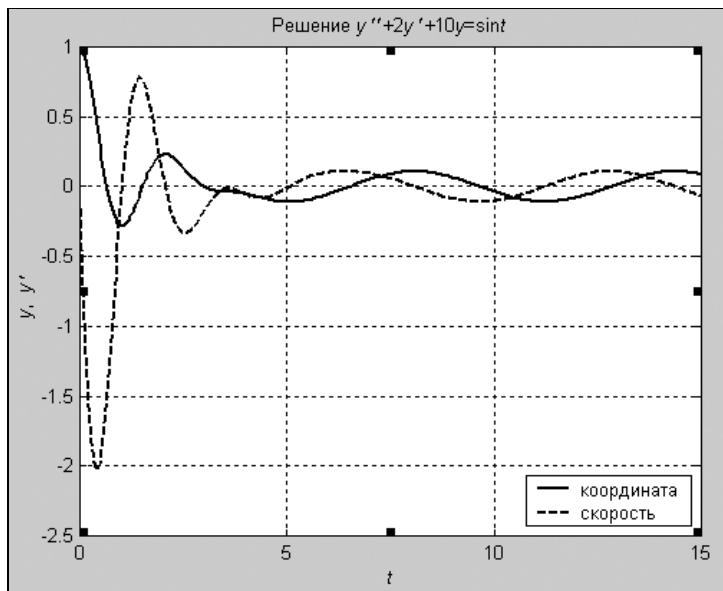
```
function solvdem
% формирование вектора начальных условий
Y0 = [1; 0];
% вызов солвера от функции oscil, начального и конечного
% момента времени и вектора начальных условий
[T, Y] = ode113(@oscil, [0 15], Y0);
% вывод графика решения исходного дифференциального уравнения
% (маркеры - точки, линия - сплошная)
plot(T, Y(:, 1), 'r.-')
% вывод графика производной от решения исходного
% дифференциального уравнения (маркеры - точки, линия - пунктир)
hold on
plot(T, Y(:, 2), 'k.:')
% вывод пояснений на график
title('Решение \dot{y}_1 \prime\prime+2 \dot{y}_2 \prime+10 y_1 = sin (t) ')
xlabel('t')
ylabel('y_1, y_2')
legend('координата', 'скорость', 4)
grid on
hold off
```

```
% подфункция вычисления правых частей уравнений
function F = oscil(t, y)
F = [y(2); -2*y(2) - 10*y(1) + sin(t)];
```

В результате выполнения файла-функции `solvdem` на экран выводятся графики, изображенные на рис. 6.8, 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  $t = 5$ .

### Примечание

Солверы могут найти приближенное решение для заданных значений независимой переменной. Для этого следует в качестве второго входного аргумента указать вектор с этими значениями, упорядоченным по возрастанию или убыванию.



**Рис. 6.8.** Решение дифференциального уравнения (6.4)

Заметим, что интерфейс солверов допускает обращение к ним с одним выходным аргументом:

```
>> sol = ode13(@oscil, [0 15], Y0);
```

Такой вызов солвера приводит к образованию структуры `sol` с информацией о полученном приближенном решении. Поле `x` структуры `sol` содержит вектор-строку значений независимой переменной, а поле `y` — матрицу из соответствующих значений компонент решения, записанных в строки (работа со структурами данных подробно описана в главе 8).

Проще говоря, `sol.x` эквивалентно `T'`, а `sol.y` — `Y'` для рассмотренного выше обращения к солверу с двумя выходными аргументами `T` и `Y` в файл-функции `solvdem`, текст которой приведен в листинге 6.15 (напомним, что апостроф означает транспонирование). Если бы в `solvdem` солвер был вызван с единственным выходным аргументом, то для построения графиков компонент приближенного решения следовало бы использовать команды (проверьте!):

```
plot(sol.x, sol.y(1, :), 'r.-')
plot(sol.x, sol.y(2, :), 'k.:')
```

Мы привели альтернативный вариант интерфейса солверов с одним выходным аргументом, поскольку он позволяет ответить на вопрос: как получить значение решения в любой точке отрезка. Действительно, солверы вычисляют приближенное решение задачи Коши в конечном числе точек на заданном отрезке, причем координаты точек выбираются солвером или задаются пользователем. Для определения значений компонент решения в промежуточных точках придется прибегнуть к интерполяции. Эффективную интерполяцию осуществляет функция `deval`, использование которой требует структуры `sol` с информацией о решении. Самый простой способ вызова функции `deval`

```
>> sxint = deval(sol, xint)
```

предполагает указание вектора `xint` с координатами точек, в которых следует вычислить решение. Найденные значения компонент построчно записываются в выходном аргументе, т. е. в `sxint(:, i)` хранятся значения всех компонент искомой вектор-функции в точке с координатой `xint(i)`. Например, для вычисления компонент решения на отрезке  $[0, 5]$  в равноотстоящих точках с шагом 0.02 и построения графиков достаточно выполнить команды:

```
>> xint = 0:0.02:5;
>> sxint = deval(sol, xint);
>> plot(xint, sxint(1, :), xint, sxint(2, :))
```

Интерполяция возможна только для некоторых компонент решения, в этом случае необходимо указать их номера в векторе `idx` и включить его в список входных аргументов

```
>> sxint = deval(sol, xint, idx)
```

В нашем примере решение задачи Коши для системы дифференциальных уравнений, соответствующей исходной задаче для дифференциального уравнения второго порядка, было получено при помощи солвера `ode113`, который основан на методе Адамса—Бэшфорта—Милтона. Кроме солвера `ode113`, MATLAB имеет еще ряд солверов для задачи Коши: `ode45`, `ode23`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`, интерфейс которых не отличается от `ode113`.

### Примечание

Вызов солвера с одним и двумя выходными аргументами приводит к одноковому выбору узлов на заданном отрезке. Единственный солвер `ode45` добавляет промежуточные точки и сам осуществляет интерполяцию в случае обращения к нему с двумя выходными аргументами.

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 системы уравнений Лотки—Вольтерры демонстрируется важность соответствия солвера решаемой задаче.

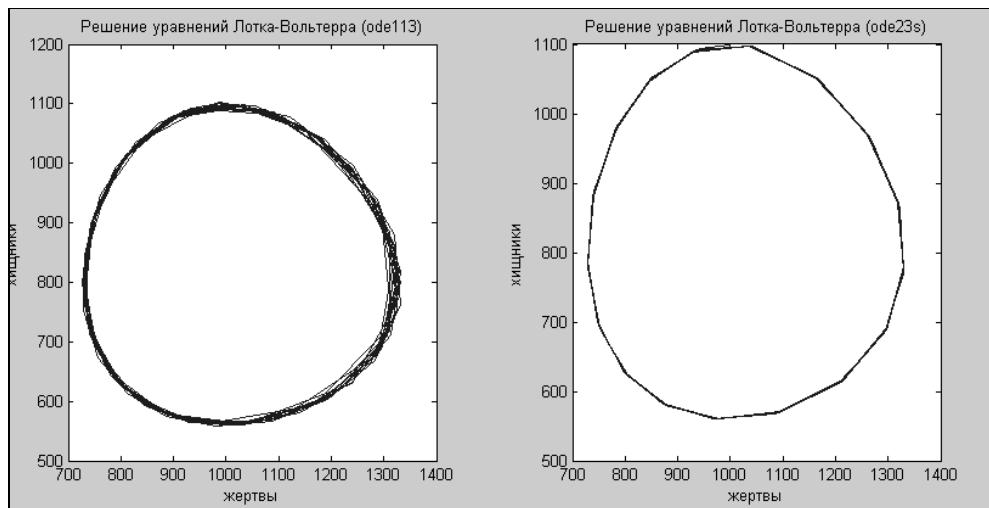
## Решение уравнений Лотки—Вольтерры

В качестве объекта исследования возьмем модель Лотки—Вольтерры 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

$$\begin{cases} y'_1 = P \cdot y_1 - p \cdot y_1 y_2; \\ y'_2 = -R \cdot y_2 + r \cdot y_1 y_2, \end{cases} \quad (6.5)$$

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. Возьмите для примера  $P = 0.8$ ,  $R = 1$ ,  $p = r = 0.001$ , считайте, что в начальный момент времени было 1000 жертв и 1100 хищников. При помощи солвера `ode113` решите эту систему дифференциальных уравнений для  $t \leq 100$ , а затем используйте `ode23s` (задание его аргументов производится аналогично `ode113`). Выведите полученные при помощи `ode113` и `ode23s` решения на разные графики, и сравните их. Листинг 6.16 содержит текст необходимых функций.

Как известно, точным решением задачи Лотки—Вольтерры на плоскости  $y_1y_2$  является замкнутая кривая. Графики приближенных решений, приведенные на рис. 6.9, сильно отличаются друг от друга, хотя вычисления по умолчанию в `ode113` и `ode23s` происходят с одинаковой точностью. Приближенное решение, полученное солвером `ode23s`, намного точнее, чем в случае `ode113`. Дело в том, что уравнения Лотки—Вольтерры являются примером так называемых *жестких систем*, для решения которых в пакете MATLAB имеются специальные солверы. Один из них — `ode23s`. В справочной системе MATLAB приведен другой пример жесткой задачи — уравнение Ван-дер-Поля с большим значением параметра, для исследования которой следует применять специальный солвер `ode15s` (см. разд. **Mathematics: Examples: Differential Equations - Initial Value Problems**, подразделы **The van der Pol Equation,  $\mu = 1000$  (Stiff)** и **Stiff Problem (van der Pol Equation)** справочной системы MATLAB).



**Рис. 6.9.** Сравнение солверов `ode113` и `ode23s`

Итак, при решении в MATLAB дифференциальных уравнений и систем с начальными условиями следует правильно выбирать солвер в зависимости от свойств задачи. В следующем разделе кратко обсуждаются области применения солверов MATLAB.

**Листинг 6.16. Файл-функция для сравнения ode45 и ode23s**

```

function comparesolvers
% формирование вектора начальных условий
Y0 = [1000; 1100];
% вызов солвера ode113
[T, Y] = ode113(@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 Лотки–Вольтерры (ode113)')
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 Лотки–Вольтерры (ode23s)')
xlabel('жертвы')
ylabel('хищники')
% подфункция вычисления правых частей уравнений
function F = LotVol(t, y)
F = [0.8*y(1) - 0.001*y(1)*y(2); -1.0*y(2) + 0.001*y(1)*y(2)];

```

**Выбор солвера для решения задачи Коши**

В данном разделе описана стратегия применения солверов MATLAB для решения 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 справочной системе MATLAB.

Очень часто солвер `ode45` дает вполне хорошие результаты, им стоит воспользоваться в первую очередь. Он основан на формулах Рунге—Кутты четвертого и пятого порядка точности. Солвер `ode23` также основан на формулах Рунге—Кутты, 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 лучший 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  $10^{-3}$  и абсолютной —  $10^{-6}$ . Хорошим тестом качества приближенного решения является увеличение точности вычислений (задание точности вычислений и ряда других параметров описано в следующем разделе).

Если все попытки применения `ode45`, `ode23s`, `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 `ode113` (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*.

## 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 допускают указание параметров для контроля и управления вычислительным процессом. Способ задания параметров солверов `ode45`,

`ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t` и `ode23tb` аналогичен тому, который вы применяли при нахождении корней функции или локальных минимумов. Значения параметров записываются в управляющую структуру, которая создается функцией `odeset`. В общем случае обращение к `odeset` имеет вид:

```
options = odeset(..., 'вид_контроля', значение, ...)
```

Параметры солверов, сгруппированные в категории по своему назначению, приведены в табл. 6.6. Следует иметь в виду, что неоправданное изменение многих параметров может повлечь уменьшение эффективности солвера или получение неверных результатов.

**Таблица 6.6. Параметры `odeset`**

| Группа                             | Имя параметра (вид контроля)                                  |
|------------------------------------|---------------------------------------------------------------|
| Контроль точности вычислений       | RelTol, AbsTol, NormControl                                   |
| Шаг интегрирования                 | InitialStep, MaxStep                                          |
| Выходные данные                    | OutputFcn, OutputSel, Refine, Stats                           |
| Якобиан                            | Jacobian, JPattern, Vectorized                                |
| Матрица масс и матрица системы ОДУ | Mass, MStateDependence, MvPattern, MassSingular, InitialSlope |
| События                            | Events                                                        |
| Только для <code>ode15s</code>     | MaxOrder, BDF                                                 |

Структура с опциями солверов модифицируется так же, как и в случае с `optimset` при решении уравнений и минимизации функций:

```
options = odeset(options, вид_контроля, значение,...)
```

Вызов `odeset` без входных аргументов позволяет посмотреть в командном окне имена всех свойств и их возможные значения, причем в фигурных скобках указаны значения, используемые солверами по умолчанию.

Функция `odeget` предназначена для извлечения значения свойства из структуры

```
значение = odeget(options, вид_контроля)
```

При генерации структуры функцией `odeset` значение могло быть не определено, в этом случае возвращается пустой массив.

Предназначение всех параметров солверов раскрыто в интерактивной справочной системе. Для получения этой информации обратитесь к описанию

функции `odeset` (воспользовавшись, например, индексным поиском на вкладке **Index**) и к разд. **Mathematics: Differential Equations: Initial Value Problems for ODEs and DAEs: Changing ODE Integration Properties**. Ряд опций солверов, к изменению которых приходится прибегать наиболее часто, мы рассмотрим в следующих нескольких разделах.

## Задание точности вычислений и шага интегрирования

Точность вычислений оказывает существенное влияние на качество приближенного решения. Управляющие параметры солверов и их значения для данного вида контроля перечислены в табл. 6.7. Допускается два способа контроля точности в зависимости от значения параметра `NormControl`:

1. По локальной погрешности  $e_i$   $i$ -ой ( $y_i$ ) компоненты вектора решений, если параметр `NormControl` имеет значение '`off`' (по умолчанию).
2. По евклидовой норме погрешности, для `NormControl`, установленного в '`on`'.

В первом случае точность считается достигнутой при выполнении условий  $|e_i(t_k)| \leq \max\{\text{RelTol} \cdot |y_i(t_k)|, \text{AbsTol}(i)\}$  для всех компонент вектора решений на каждом шаге по времени  $t_k$ . Во втором случае принимается во внимание суммарная характеристика для всех компонент решения — евклидова норма вектора  $\| \cdot \|$  (вычисляемая встроенной функцией `norm`) — и точность считается достигнутой, если на каждом шаге по времени  $t_k$  выполняется неравенство  $\| e \| \leq \max\{\text{RelTol} \cdot \| y \|, \text{AbsTol}\}$ . В случае покомпонентной оценки параметр `AbsTol` (абсолютная точность) может быть числом, если требуется задать фиксированную абсолютную точность для всех компонент решения. Для контроля абсолютной точности каждой компоненты следует указать вектор значений. Можно считать, что относительная точность `RelTol` определяет число верных значащих цифр во всех компонентах решения. Однако если среди компонент решения есть близкие к нулю, то для более точного их вычисления следует уменьшать соответствующий элемент вектора значений `AbsTol`.

Шаг интегрирования солвера определяется двумя свойствами:

- `MaxStep` — задает максимальный шаг (по умолчанию десятая часть промежутка интегрирования);
- `InitialStep` — задает начальный шаг, и если он не определен, то выбирается солвером с учетом начальных значений для вектора решений. При нулевых значениях шаг может оказаться слишком большим.

**Таблица 6.7.** Параметры контроля точности вычислений

| Вид контроля | Значение                                      | Примечание                                                      |
|--------------|-----------------------------------------------|-----------------------------------------------------------------|
| NormControl  | 'off'<br>(по умолчанию)                       | Контроль точности по максимальной локальной погрешности         |
|              | 'on'                                          | Контроль точности по второй норме вектора локальной погрешности |
| RelTol       | число<br>( $10^{-3}$ по умолчанию)            | Относительная точность вычислений                               |
| AbsTol       | число или вектор<br>( $10^{-6}$ по умолчанию) | Точность для контроля компонент вектора решения                 |

Убедитесь 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

$$\begin{cases} y_1' = y_2; \\ y_2' = -1/t^2 \end{cases}$$

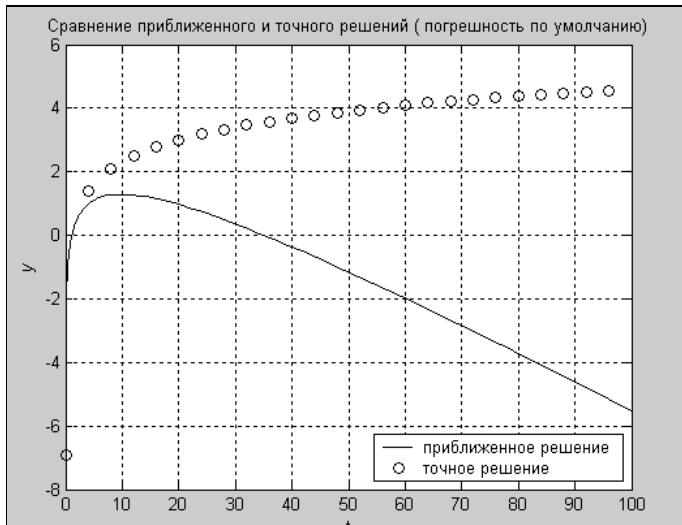
на отрезке  $[a, 100]$  при начальных условиях  $y_1(a) = \ln a$ ,  $y_2(a) = 1/a$ , взяв  $a = 0.001$ .

Легко проверить, что точным решением этой системы является  $y_1 = \ln t$ ,  $y_2 = 1/t$ .

Напишите самостоятельно файл-функцию для решения данной системы солвером `ode45`, содержащую подфункцию `rsbad` для вычисления правой части системы. Расположите на одном графике точное и приближенное решение. Результат, приведенный на рис. 6.10, является довольно неожиданным для погрешности  $10^{-3}$  (используемой по умолчанию). Применение других солверов и задание более мелкого максимального или начального шагов не улучшает ситуацию.

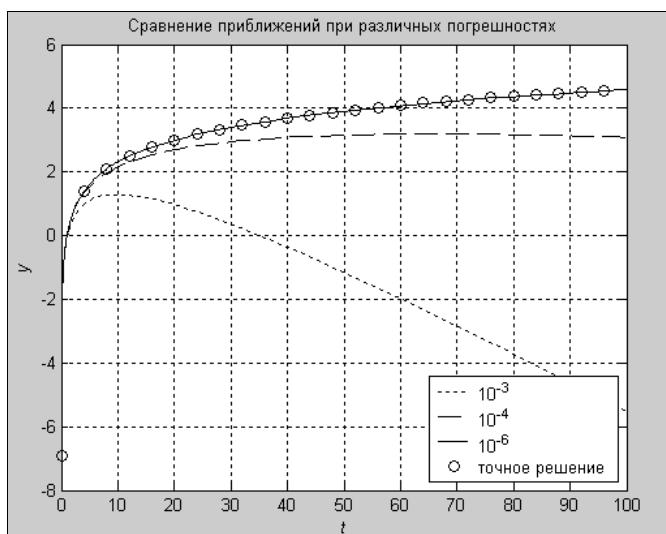
Выход состоит в уменьшении относительной погрешности вычислений при помощи формирования `options` с использованием `odeset` и включении `options` дополнительным четвертым аргументом в солвер. Для задания относительной погрешности служит параметр `RelTol`, например

```
options = odeset('RelTol', 1.0e-04)
```



**Рис. 6.10.** Сравнение приближенного решения с точным (погрешность по умолчанию)

Дополните созданную файл-функцию вызовами `ode45`, предваряя каждое обращение к солверу установкой точности. Не забывайте включать `options` в список аргументов солвера!



**Рис. 6.11.** Сравнение приближенного решения с точным при различных погрешностях

При возникновении затруднений обратитесь к листингу 6.17. Выведите графики приближенных решений для погрешностей  $10^{-3}$ ,  $10^{-4}$ ,  $10^{-6}$  так, как показано на рис. 6.11.

Только точность  $10^{-6}$  обеспечивает получение приближенного решения, график которого почти совпадает с графиком точного решения.

### Листинг 6.17. Файл-функция для исследования влияния погрешности

```
function difficultproblem
a = 0.001;
Y0 = [log(a); 1/a]; % задание начальных условий
% вызов солвера ode45 для решения с различной точностью
% и вывод графиков приближенных решений

% относительная точность 0.001
options = odeset('RelTol', 1.0e-3);
[T, Y] = ode45(@rsbad, [a 100], Y0, options);
plot(T, Y(:,1), 'k:')

% относительная точность 0.0001
options = odeset('RelTol', 1.0e-4);
[T, Y] = ode45(@rsbad, [a 100], Y0, options);
hold on
plot(T, Y(:, 1), 'k--')

% относительная точность 0.000001
options = odeset('RelTol', 1.0e-6);
[T, Y] = ode45(@rsbad, [a 100], Y0, options);
plot(T, Y(:, 1), 'k-')

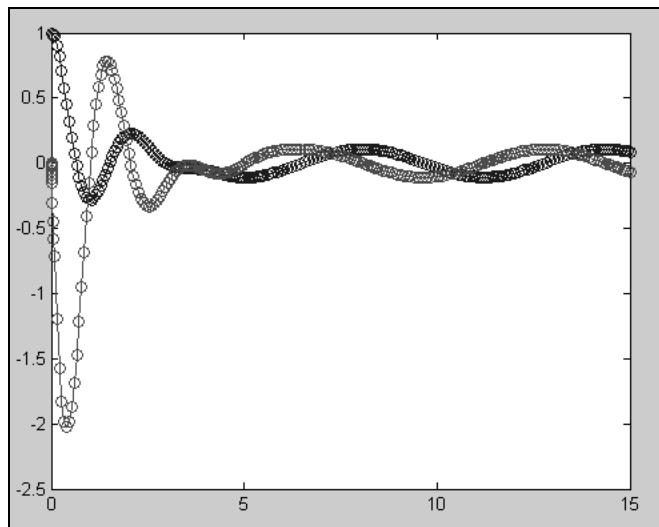
% вывод графика точного решения
t = [a:4:100];
y = log(t);
plot(t, y, 'ko')

% 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
% подфункция вычисления правых частей уравнений
function F = rsbad(t, y)
F = [y(2); -1.0/t^2];
```

## Управление выводом результатов

Примеры предыдущих разделов предполагали вызов солверов с двумя выходными аргументами — массивами или одним — структурой. В них возвращался вектор значений независимой переменной и матрица со значениями компонент решения в соответствующих точках. Полученные массивы мы использовали для визуализации и анализа результата. Вернитесь к уравнению (6.4), для решения которого была создана файл-функция, приведенная в листинге 6.15, уберите выходные аргументы у солвера и вывод решения функцией `plot`. Вызов солвера без выходных аргументов приводит к появлению графического окна, изображенного на рис. 6.12, в котором отображается процесс численного интегрирования дифференциального уравнения и выводятся все компоненты вектора решения.



**Рис. 6.12.** Графическое отображение процесса численного интегрирования

Возможности вывода результата, предоставляемые солверами MATLAB, не исчерпываются только таким способом визуализации решения. Пользователь может выбрать альтернативное графическое представление результата, более того, допускается контролировать процесс численного интегрирования и отображать его на графике по своему усмотрению. Для этого следует воспользоваться одним из видов контроля управляющей структуры — `OutputFcn`. Его значение должно быть указателем на функцию (или ее именем), производящую требуемые операции. Имеется несколько стандартных функций:

- `odeplot` — построение графиков компонент решения;
- `odephas2` — построение графиков компонент решения в фазовых координатах для двумерного процесса;
- `odephas3` — построение графиков компонент решения в фазовых координатах для трехмерного процесса;
- `odeprint` — печать решения.

Заметьте, что вызов солвера без выходных аргументов приводит к появлению тех же графиков, что и применение `odeplot`, поскольку вид контроля `OutputFcn` принимает значение `@odeplot`:

```
>> options = odeset('OutputFcn', @odeplot)
>> [T,Y] = ode45(@oscil, [0 15], Y0, options);
```

Представление решения на фазовой плоскости такое, как на рис. 6.13, производится при помощи `odephas2`:

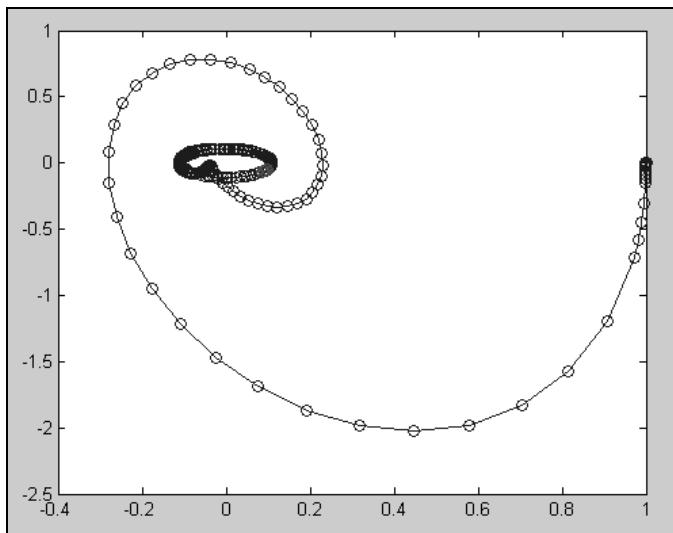
```
>> options = odeset('OutputFcn', @odephas2)
>> [T,Y] = ode45(@oscil, [0 15], Y0, options);
```

Пользователь может создавать свои файл-функции для визуализации решения или обработки результатов каждого шага численного интегрирования.

Рассмотрим один простой пример. Требуется производить численное интегрирование дифференциального уравнения

$$y'' + y = e^x (\sin x + 2 \cos y) \quad (6.6)$$

при начальных условиях  $y(0) = 0$ ,  $F(t, Y, Y')$  до тех пор, пока модуль значения функции  $y(x)$  не превзойдет 100. Приведите уравнение к системе обыкновенных дифференциальных уравнений и запрограммируйте ее правую часть в файл-функции `syst` (листинг 6.18).



**Рис. 6.13.** Визуализация решения дифференциального уравнения (6.4) на фазовой плоскости (`odephas2`)

#### Листинг 6.18. Правая часть системы дифференциальных уравнений (6.6)

```
function F = syst(t, y)
F = [y(2); exp(t).* (sin(t) + 2*cos(t)) - y(1)];
```

При вызове солвера, к примеру, `ode113`, требуется указать отрезок интегрирования, но его правая граница заранее неизвестна — мы не знаем априори, в какой момент времени модуль значения функции станет больше 100. Например, для отрезка  $[0, 10]$  получаются существенно большие значения:

```
>> [T,Y] = ode113(@syst, [0, 10], [0 1])
>> plot(T, Y(:, 1))
```

Выход состоит в написании файл-функции для обработки значения, вычисленного на текущем шаге интегрирования. Солвер будет вызывать эту функцию после каждого шага и осуществлять дальнейшие действия в зависимости от возвращаемого ей значения. Назовем эту функцию `solproc`. Ее заголовок должен иметь вид:

```
function status = solproc(t, y, flag)
```

Входной аргумент `flag` является строковой переменной и может принимать одно из трех значений, которое в свою очередь определяет содержимое `t` и `y`:

- '`init`' — при первом вызове функции `solproc` солвером до начала процесса интегрирования. При этом `t` является вектором из двух элементов — границ отрезка интегрирования по времени, а `y` — вектором начальных значений;
- '' (пустая строка) — после каждого шага интегрирования. Во входных аргументах `t` и `y` находятся текущие значения аргумента и приближенного решения. Аргумент `t` может содержать несколько текущих значений, при этом в `i`-ом столбце массива `y` записаны значения компонент решения для `t(i)`;
- '`done`' — после завершения численного решения системы дифференциальных уравнений. В качестве `t` и `y` передаются пустые массивы.

В случае, когда `flag` является пустой строкой, длина входного аргумента `t` определяется значением свойства `Refine`. По умолчанию оно равно 1 и все солверы (кроме `ode45`) на каждом шаге будут вызывать функцию `solproc` только от одного значения независимой переменной. Соответственно, `y` будет вектором со значениями компонент решения, чем мы и будем пользоваться в нашем примере для солвера `ode113`.

### Примечание

Управляющий параметр `Refine` предназначен для увеличения числа точек, в которых вычисляется решение на каждом шаге интегрирования. Если `Refine` установлен в 1 (по умолчанию), то приближенное решение находится только для конца отрезка, по которому производится интегрирование на текущем шаге. При увеличении `Refine` вводятся промежуточные точки на отрезке интегрирования, которые разбивают его на столько интервалов, сколько указано в `Refine`. Решение в этих точках вычисляется при помощи специальных продолжений, экономящих время счета. Солвер `ode45`, в отличие от остальных, всегда по умолчанию использует 4 интервала.

В результате работы функции `solproc` формируется выходной аргумент `status`, который может быть 1 либо 0. Если солвер обнаруживает, что функция `solproc` вернула 1, то процесс интегрирования прекращается, а если 0 — то продолжается.

Итак, нам требуется запрограммировать функцию, реализующую простой алгоритм: если `flag` — пустая строка и модуль значения `y(1)` превосходит 100, то вернуть 1, иначе вернуть 0. Текст требуемой файл-функции приведен в листинге 6.19.

**Листинг 6.19. Файл-функция solproc для проверки критерия останова солвера**

```
function status = solproc(t, y, flag)
% функция определяет, превысил ли модуль первой компоненты
% решения число 100 после каждого шага интегрирования
% если да – status = 1, если нет – status = 0
status = (length(flag) == 0) && (abs(y(1)) > 100)
```

В теле функции для определения значения выходного аргумента мы записали логическое выражение с оператором `&&` (логическое "и"), который при невыполнении первого условия второе уже не проверяет. Действительно, второе условие имеет смысл проверять только в том случае, когда солвер на текущем шаге интегрирования вызвал нашу функцию с `flag`, равным пустой строке. При последнем вызове `solproc` солвером во входном аргументе `u` передается пустой массив, и обращение к его первому элементу привело бы к ошибке (запись логических выражений в MATLAB подробно рассмотрена в главе 7).

При формировании структуры `options` задайте указатель на функцию `solproc` в качестве значения `OutputFcn`, вызовите солвер `ode113` и постройте график первой компоненты решения.

```
>> options = odeset('OutputFcn', @solproc)
>> [T, Y] = ode113(@syst, [0, 10], [0 1], options)
>> plot(T, Y(:,1))
```

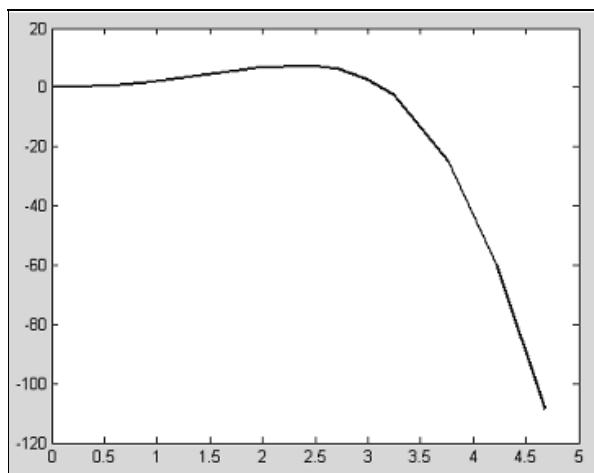
Получившийся график, приведенный на рис. 6.14, свидетельствует о свое-временном останове вычислительного процесса.

Мы привели, пожалуй, самый простой пример использования информации, получаемой в процессе численного интегрирования. Функция обработки решения `solproc` могла бы содержать и более сложный алгоритм, записанный на языке программирования MATLAB, который реализует собственный способ визуализации решения и ряда его характеристик. После освоения конструкций языка программирования и дескрипторной графики вам будут понятны тексты файл-функций `odeplot`, `odephas2`, `odephas3`, расположенных в подкаталоге `\toolbox\matlab\funfun\` основного каталога MATLAB. Изучение алгоритмов этих файл-функций позволит получить представление о том, как организовать обработку выходных данных солверов (программированию и дескрипторной графике посвящены главы 7—9).

При помощи параметра `OutputSel` вы можете определять номера компонент решения, которые будут передаваться в файл-функцию обработки. Для этого необходимо указать вектор номеров требуемых компонент или один но-

мер, как это сделано в следующем примере для графического отображения процесса нахождения только первой компоненты решения.

```
>> options = odeset('OutputFcn', @odeplot, 'OutputSel', 1)
>> [T, Y] = ode113(@syst, [0, 10], [0 1], options)
```



**Рис. 6.14.** Решение дифференциального уравнения (6.6)  
с критерием останова солвера по значению функции

При контроле за наступлением некоторых событий в процессе численного интегрирования часто важно достаточно точно знать время наступления события. Для этого следует запрограммировать файл-функцию специального вида, задать указатель на нее в качестве значения параметра `Events` и вызвать солвер с пятью выходными аргументами или структурой.

## Задание матрицы Якоби для повышения эффективности вычислений

После завершения счета солвер может вывести в командное окно суммарную информацию о вычислительной работе, включающую количество вычислений правой части матрицы Якоби, успешных и неуспешных шагов солвера и другие сведения. Для этого следует установить свойство `'Stats'` в `'on'`. Подробное описание всех свойств управляющей структуры и примеры содержатся в справочной системе MATLAB в разд. **Mathematics: Differential Equations: Initial Value Problems for ODEs and DAEs: Changing ODE Integration Properties**.

При решении жестких систем дифференциальных уравнений солверы `ode15s`, `ode23s`, `ode23t` и `ode23tb` аппроксимируют матрицу Якоби правой части системы методом конечных разностей. Эффективность вычислений может быть повышена путем задания матрицы Якоби в явном виде. Для этого следует написать файл-функцию, которая возвращает матрицу Якоби для текущих значений независимой переменной и компонент решения. Заголовок функции должен иметь вид

```
function J = jmatr(t, y)
```

Для того чтобы солвер мог воспользоваться явными формулами, необходимо присвоить свойству `Jacobian` управляющей структуры указатель на функцию `jmatr`. Если матрица Якоби постоянна, то не требуется программировать файл-функцию — достаточно указать ее в качестве значения свойства `Jacobian`.

Для рассмотренной выше системы уравнений Лотки—Вольтерры (6.5), при  $P = 0.8$ ,  $R = 1$ ,  $p = r = 0.001$ , функция, вычисляющая матрицу Якоби, приведена в листинге 6.20.

#### Листинг 6.20. Программирование матрицы Якоби

```
function J = JLotVol(t, y)
J = [0.8 - 0.001*y(2) - 0.001*y(1)
 0.001*y(2) -1 + 0.001*y(1)];
```

Внесите `JLotVol` в качестве подфункции в файл-функцию `comparesolvers` (листинг 6.16) и перед вызовом солвера `ode23s` в основной функции сформируйте управляющую структуру `options` и укажите ее в качестве входного аргумента `ode23s`. Соответствующие команды приведены в листинге 6.21.

#### Листинг 6.21. Указание матрицы Якоби солверу

```
options = odeset('Jacobian', @JLotVol);
[T, Y] = ode23s(@LotVol, [0 100], Y0, options);
```

Теперь при обращении к файл-функции `comparesolvers` солвер `ode23s` будет использовать явные формулы для элементов матрицы Якоби.

Системы дифференциальных уравнений большой размерности могут иметь матрицу Якоби с небольшим числом ненулевых элементов, т. е. разреженную. В этом случае имеет смысл подсказать солверу, где расположены ее ненулевые элементы, подлежащие аппроксимации в процессе численного интегрирования. Для этих целей служит свойство `JPattern`, значением ко-

торого должна быть разреженная матрица с элементами 0 и 1, определяемая шаблоном матрицы Якоби (разреженным матрицам посвящена глава 15).

## Задачи с известными параметрами

Солверы MATLAB допускают решение систем обыкновенных дифференциальных уравнений, правая часть которых зависит от некоторых параметров  $p_1, p_2, \dots$  с известными значениями. В предыдущих версиях MATLAB передача параметров была организована посредством интерфейса солверов. Эта возможность сохранилась и в версии 7, однако ее описание исключено из справочной системы, поскольку реализованы другие методы передачи параметров, основанные на использовании вложенных или анонимных функций.

Способ обращения к солверам, унаследованный из предыдущих версий, состоит в указании значений параметров в списке входных аргументов, начиная с пятой позиции после управляющей структуры `options`

```
>> [T, Y] = ode45(@syst, [t0, t1], Y0, options, p1, p2, ...)
```

или

```
>> sol = ode45(@syst, [t0, t1], Y0, options, p1, p2, ...)
```

Вызов других солверов для задачи Коши производится аналогично; кроме того, вместо структуры `options` допускается задание пустого массива для нахождения приближенного решения с точностью и другими опциями, установленными по умолчанию.

Такой подход накладывает определенные требования на интерфейс функции, вычисляющей правую часть системы дифференциальных уравнений — ее заголовок должен иметь вид:

```
function F = syst(t, y, p1, p2, ...)
```

Если в процессе решения используются явные формулы для вычисления матрицы Якоби, то параметры включаются в список входных аргументов соответствующей функции

```
function F = jac(t, y, p1, p2, ...)
```

Приведенная выше система дифференциальных уравнений Лотки—Вольтерры зависит от четырех параметров:  $P, p, R$  и  $r$ , и при многократном ее решении для различных значений параметров целесообразно написать соответствующую файл-функцию, вместо того, чтобы каждый раз изменять значения этих параметров. Текст такой файл-функции приведен в листинге 6.22.

**Листинг 6.22. Передача параметров через входные аргументы солвера**

```

function LVpar(P, p, R, r)
Y0 = [1000; 1100]; % задание начальных условий
% формирование управляющей структуры с матрицей Якоби
options = odeset('Jacobian', @JLotVolPar);
% вызов солвера и передача значений параметров
[T, Y] = ode23s(@LotVolPar, [0 100], Y0, options, P, p, R, r);
% построение графика решения
figure
plot(Y(:, 1), Y(:, 2))
% подфункция для вычисления правой части системы, зависящей от параметров
function F = LotVolPar(t, y, P, p, R, r)
F = [P*y(1) - p*y(1)*y(2)
 -R*y(2) + r*y(1)*y(2)];
% подфункция для вычисления матрицы Якоби, зависящей от параметров
function J = JLotVolPar(t, y, P, p, R, r)
J = [P - p*y(2) - p*y(1)
 r*y(2) - R + r*y(1)];

```

Теперь обращение к файл-функции `LVpar` позволяет решить задачу Лотки—Вольтерры для различного набора значений параметров, например

```
>> LVpar(0.75, 0.002, 0.95, 0.001)
```

или

```
>> LVpar(0.8, 0.001, 0.9, 0.003)
```

Альтернативный способ решения задач с известными параметрами состоит в использовании анонимных или вложенных функций. Мы обсуждали этот подход при поиске корней и локальных минимумов функций, поэтому приведем здесь только текст соответствующей файл-функции с двумя вложенными функциями без дополнительных комментариев (листинг 6.23). Обращение к `LVpar1` производится так же, как и к `LVpar`.

**Листинг 6.23. Передача параметров через входные аргументы солвера**

```

function LVpar1(P, p, R, r)
% вложенная функция для вычисления правой части системы,
% зависящей от параметров
function F = LotVolPar(t, y)

```

```

F = [P*y(1) - p*y(1)*y(2)
 -R*y(2) + r*y(1)*y(2)];
end

% вложенная функция для вычисления матрицы Якоби, зависящей от параметров
function J = JLotVolPar(t, y)
J = [P - p*y(2) - p*y(1)
 r*y(2) - R + r*y(1)];
end;

Y0 = [1000; 1100]; % задание начальных условий
% формирование управляющей структуры с матрицей Якоби
options = odeset('Jacobian', @JLotVolPar);
% вызов солвера
[T, Y] = ode23s(@LotVolPar, [0 100], Y0, options);
% построение графика решения
figure
plot(Y(:, 1), Y(:, 2))
end

```

## Системы, не разрешенные относительно производной, дифференциально-алгебраические уравнения

До сих пор мы рассматривали задачу Коши для систем дифференциальных уравнений, разрешенных относительно производных

$$Y' = f(t, Y).$$

MATLAB позволяет решать системы дифференциальных уравнений, заданных в неявной форме

$$F(t, Y, Y') = 0.$$

Важным частным случаем таких систем являются системы с матрицей масс

$$M(t, Y) Y' = F(t, Y),$$

причем  $M(t, Y)$  может быть как невырожденной, так и вырожденной.

Для решения систем вида  $F(t, Y, Y') = 0$  служит солвер `ode15i`, а системы с матрицей масс могут быть решены любым из солверов. Заметим, что область применения солвера `ode23s` ограничена только постоянными матри-

цами, а в случае вырожденной матрицы необходимо прибегать к солверам `ode15s` и `ode23t`.

Вырожденная матрица масс соответствует системе дифференциально-алгебраических уравнений, которая наряду с дифференциальными  $y' = f(t, y, u)$  содержит алгебраические связи  $g(t, y, u) = 0$ . Здесь  $f$  и  $g$  — заданные вектор-функции, а  $y$  и  $u$  — искомые вектор-функции. Такая система может быть решена в MATLAB при условии, что ее индекс равен единице, т. е. матрица  $(\partial g_i / \partial u_j)$  невырождена. При решении дифференциально-алгебраических уравнений применяется их сведение к системе  $M Y' = F(t, Y)$  с вырожденной матрицей масс

$$M = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix},$$

где  $I$  — единичная матрица, размер которой совпадает с длиной вектора неизвестных  $y$ , а остальные блоки нулевые и

$$F = \begin{pmatrix} f \\ g \end{pmatrix}, \quad Y = \begin{pmatrix} y \\ u \end{pmatrix}.$$

Если дифференциальные уравнения в системе дифференциально-алгебраических уравнений не разрешены относительно производной, то для решения такой системы следует применять солвер `ode15i`.

Рассмотрим сначала решение систем с вырожденной матрицей масс на частоте приводимом примере, опубликованном Робертсоном, для иллюстрации решения жестких уравнений — нелинейную задачу химической кинетики. Одно из дифференциальных уравнений системы ( $y'_3(t) = 3 \cdot 10^7 y_2(t)^2$ ) заменено уравнением баланса.

$$\begin{cases} y'_1(t) = -0.04 y_1(t) + 10^4 y_2(t) y_3(t); \\ y'_2(t) = 0.04 y_1(t) - 10^4 y_2(t) y_3(t) - 3 \cdot 10^7 y_2(t)^2; \\ y_1(t) + y_2(t) + y_3(t) - 1 = 0; \\ t \in [0, 100], \quad y_1(0) = 1, \quad y_2(0) = 0, \quad y_3(0) = 0. \end{cases} \quad (6.7)$$

### Примечание

С некоторыми отличиями эта же задача приведена в демонстрационном примере из пакета MATLAB (функция `ihb1dae`).

В нашем случае матрица масс постоянна и имеет вид:

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

Для иллюстрации применения солвера `ode15s` при решении задач указанного класса написана файл-функция (листинг 6.24) без аргументов, с использованием вложенной функции для вычисления вектора  $F(t, Y)$ . График каждой функции выводится на свои оси, при этом автоматически масштабируется ось ординат для компоненты  $y_2$ .

#### Листинг 6.24. Решение системы дифференциальных уравнений с матрицей масс

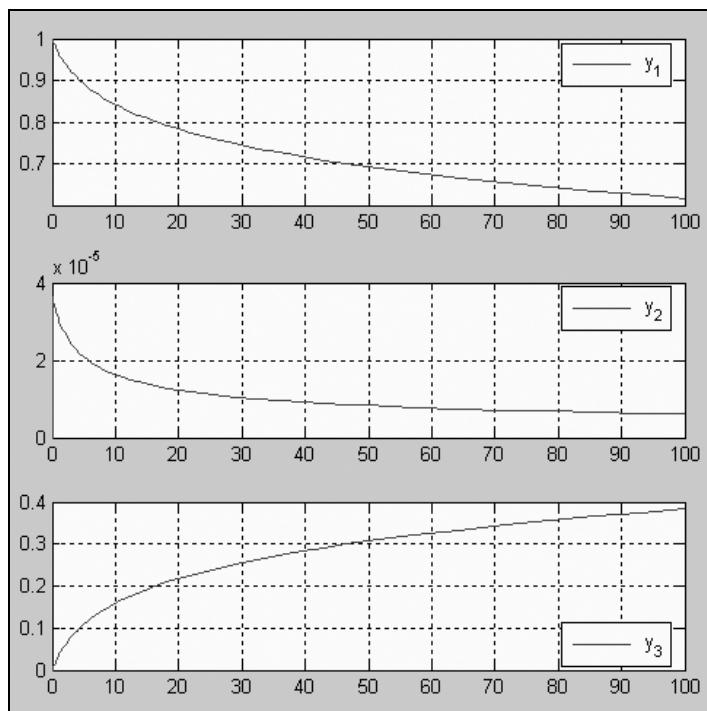
```
function example_dae
% Вложенная функция вычисления вектора f(x, y)
function resid = rob(t, y)
resid = zeros(3, 1);
resid (1) = -0.04*y(1) + 1e4*y(2)*y(3);
resid (2) = +0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)^2;
resid (3) = y(1) + y(2) + y(3) - 1;
end
% начальные условия.
y0 = [1; 0; 0];
% матрица масс
m = diag([1, 1, 0]);
% интервал интегрирования.
xtime = [0 100];
% задание управляемой структуры (точность)
options = odeset('RelTol', 1e-4, 'mass', m);
% вызов солвера
[t,y] = ode15s(@rob, xtime, y0, options);
% вывод графиков решений
subplot(3, 1, 1); plot(t, y(:, 1));
grid on
legend ('y_{1}');
subplot(3, 1, 2); plot(t, y(:, 2));
```

```

grid on
legend ('y_{2}');
subplot(3, 1, 3); plot(t, y(:, 3));
grid on
legend ('y_{3}', 4);
end

```

После выполнения файла-функции из листинга 6.24 получаются результаты, представленные на рис 6.15.



**Рис. 6.15.** Решение задачи химической кинетики (6.7)

Подробные сведения о настройках солверов для решения задач с матрицей масс приведены в справочной системе MATLAB. Управляющая структура позволяет задать кроме матрицы масс еще ряд опций, в частности: степень зависимости матрицы масс от неизвестных, расположение ее ненулевых элементов для задач большой размерности и информацию о вырожденности матрицы. В разд. **Mathematics: Differential Equations: Initial Value Problems for**

**ODEs and DAEs: Examples: Applying the ODE Initial Value Problem Solvers**  
 приведен пример уравнения в частных производных, которое методом прямых приводится к системе обыкновенных дифференциальных уравнений с матрицей масс.

Обратимся теперь к решению задачи Коши для дифференциальных уравнений вида  $F(t, Y, Y') = 0$ , не разрешенных относительно старшей производной, при помощи солвера `ode15i`. Обращение к нему несколько отличается от случая рассмотренных выше солверов, поскольку во входных его аргументах должно быть задано значение  $Y_{t_0}$  производной решения в начальный момент времени  $Y'(t_0)$ :

```
[T, Y] = ode15i(odefun, interval, Y0, Yp0, options)
```

Смысл остальных его аргументов тот же самый: `odefun` — функция для вычисления вектор-функции  $F$ ; `interval` — отрезок, по которому производится интегрирование, или упорядоченный вектор значений независимой переменной, для которых следует найти решение; `Y0` — вектор начальных значений  $Y(t_0)$ ; `options` — управляющая структура. Моменты времени и найденное решение в них записываются в вектор-столбец `T` и матрицу `Y`. Возможен вывод результата и в структуру с полями `x` и `y`, которая в дальнейшем используется для вычисления решения в произвольной точке из отрезка интегрирования (см. разд. "Решение задачи Коши" этой главы).

Возникает вопрос, откуда взять начальное значение для производной, которое не может быть произвольным, поскольку в начальный момент времени  $t_0$  требуется выполнение условия совместности  $F(t_0, Y(t_0), Y'(t_0)) = 0$ . Для приближенного удовлетворения этому условию служит функция `decic`. В достаточно общем случае обращение к ней выглядит следующим образом

```
[Y0, Yp0] = decic(odefun, t0, Y0Init, Yp0Init, Yp0Flag)
```

и подразумевает, что заданы начальные приближения к  $Y(t_0)$  и  $Y'(t_0)$  соответственно в векторах `Y0Init` и `Yp0Init`. Каждому из них сопутствует свой флаг — векторы `Y0Flag` и `Yp0Flag` той же длины, они указывают компоненты векторов `Y0Init` и `Yp0Init`, которые могут изменяться при удовлетворении условия согласования. Для запрета изменения некоторой компоненты решения или производной следует установить элемент вектора флага с номером компоненты в 1, а для разрешения в 0. Действительно, не всегда допустимо произвольное изменение начальных значений. В задаче Коши вектор  $Y(t_0)$  задан и требуется найти подходящее  $Y'(t_0)$ . Для этого следует задать флаг `Y0Init`, целиком состоящий из единиц. Если никакие компоненты векторов `Y0Init` или `Yp0Init` не фиксируются, то соответствующий флаг можно не задавать. Возвращаемые значения `Y0` и `Yp0` приближенно удовлетворяют условиям совместности и могут служить входными аргументами солвера `ode15i`.

### Примечание

По умолчанию условия совместности удовлетворяются с относительной точностью  $10^{-3}$ . Для изменения точности следует обратиться к функции `decic` с седьмым входным аргументом — управляющей структурой `options`, предварительно сформировав ее при помощи `odeset` (поле `RelTol` отвечает за относительную погрешность). Для контроля выполнения условий совместности полезно вызвать функцию `decic` с третьим входным аргументом, в который заносится невязка, т. е. значение  $F(t_0, Y(t_0), Y'(t_0))$  для приближенно найденных  $Y(t_0)$ ,  $Y'(t_0)$ .

Продемонстрируем использование солвера `ode15i` на примере уравнения Клеро:

$$y = y' \cdot t + h(y').$$

Известно, что  $y = Ct + h(C)$  есть общее решение этого уравнения, с ним мы и будем сравнивать получаемое приближенное решение. Рассмотрим задачу Коши для уравнения Клеро при  $h(s) = e^s$ :

$$y = y' t + e^{y'}, \quad y(0) = e, \quad t \in [0, 5]$$

с точным решением  $y = x + e$ .

Поскольку это дифференциальное уравнение первого порядка, то неизвестной является всего одна функция, и при обращении к `decic` в качестве флагов используются числа `Y0Flag = 1` и `Yp0Flag = 0` (листинг 6.25). После выполнения функции `decic` полученные начальные значения для функции и ее производной выводятся в командное окно. Обратите внимание, что начальное значение функции не изменилось из-за того, что соответствующий флаг равен единице, а начальное значение производной найдено достаточно хорошо (точное значение равно единице) так, что невязка при удовлетворении условиям совместности имеет порядок  $10^{-9}$ . Вычисленные величины `Y0` и `Yp0` мы передаем в качестве входных аргументов солвера `ode15i` и сравниваем приближенное решение с точным (рис. 6.16).

#### Листинг 6.25. Использование солвера `ode15i` для решения уравнения Клеро

```
function klero
t0 = 0; % начальная точка отрезка интегрирования [0, 5]
Y0Init = exp(1); % заданное начальное условие
Y0Flag = 1; % начальное значение искомой функции не должно изменяться
% при удовлетворении условиям совместности
```

```

Yp0Init = 0; % приближение для начального значения производной
Yp0Flag = 0; % начальное значение производной искомой функции может
 % измениться при удовлетворении условиям совместности
format long e
% Поиск начального значения производной, удовлетворяющего
% условиям совместности
[Y0, Yp0] = decic(@klerofun, t0, Y0Init, Y0Flag, Yp0Init, Yp0Flag)
% Решение уравнения Клеро на отрезке [0, 5]
[T, Y] = ode15i(@klerofun, [t0 5], Y0, Yp0);
% Построение графика приближенного решения
plot(T, Y, 'o');
hold on
% Построение графика точного решения
exsol = inline('x + exp(1)');
fplot(exsol, [t0 5])
legend('приближенное решение', 'точное решение')

```

```

function F = klerofun(t, Y, Yp)
F = Y - Yp*t - exp(Yp);

```

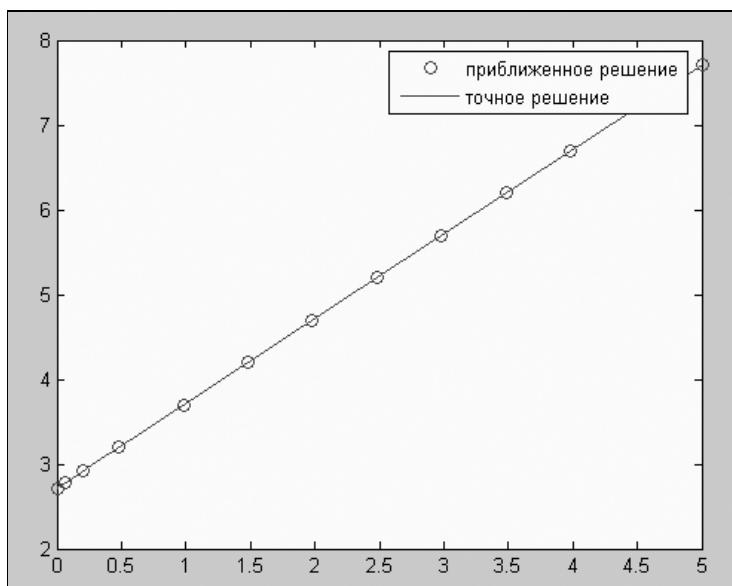


Рис. 6.16. Решение уравнения Клеро

В случае системы дифференциальных уравнений схема решения принципиально не отличается от разобранного примера, только  $\mathbf{Y}_0$ ,  $\mathbf{Y}_{\mathbf{p}0}$ ,  $\mathbf{Y}_{\mathbf{0Init}}$ ,  $\mathbf{Y}_{\mathbf{p}0Init}$ ,  $\mathbf{Y}_{\mathbf{0Flag}}$  и  $\mathbf{Y}_{\mathbf{p}0Flag}$  должны быть векторами, как и выходной аргумент функции для вычисления  $F(t, Y, Y')$ . Дополнительные возможности солвера `ode15i`, в том числе решение систем с заданными параметрами, и опции управляющей структуры описаны в справочной системе MATLAB (см. разд. MATLAB: Mathematics: Differential Equations: Initial Value Problems for ODEs and DAEs: Solver for Fully Implicit ODEs и следующий за ним, а также страницы, посвященные функциям `ode15i` и `decic`).

### Предупреждение

При формировании структуры `options` некоторые параметры используются не так, как для остальных солверов. Например, по другому передается матрица Якоби с постоянными коэффициентами, а именно в массиве ячеек, содержащем две матрицы  $\partial F/\partial y$  и  $\partial F/\partial y'$ . Работа с массивами ячеек описана в главе 8.

Мы рассмотрели решение задачи Коши для дифференциальных уравнений и систем, включая дифференциально-алгебраические уравнения и общий случай уравнений, не разрешенных относительно старшей производной. Обратимся теперь к другому типу дифференциальных уравнений — дифференциальным уравнениям с запаздывающим аргументом.

## Решение дифференциальных уравнений с запаздывающим аргументом

В пакет MATLAB входит солвер `dde23` для решения задач, описываемых системами дифференциальных уравнений вида:

$$y'(t) = f(t, y(t), y(t - \tau_1), y(t - \tau_2), \dots, y(t - \tau_k)), \quad t \in [a, b], \quad (6.8)$$

где  $y(t)$  — искомая вектор-функция;  $f$  — известная вектор-функция правой части системы;  $\tau_1, \tau_2, \dots, \tau_k$  — моменты запаздывания (положительные числа). Для решения системы уравнений (6.8) требуется знать поведение решения для  $t < a$ , т. е. вектор-функцию предыстории  $S(t)$ .

Схема решения дифференциальных уравнений с запаздывающим аргументом в MATLAB состоит из следующих этапов.

1. Написание функции для вычисления правой части системы уравнений, входными аргументами которой являются не только  $t$  и  $y$ , но и значения компонент решения в моменты времени  $\tau_1, \tau_2, \dots, \tau_k$ .

2. Написание функции для предыстории с единственным входным аргументом  $t$  (если вектор-функция предыстории является постоянной, то эта функция не является обязательной).
3. Вызов солвера `dde23` с указанием исходных данных: функции правой части системы, моментов запаздывания  $\tau_1, \tau_2, \dots, \tau_k$ , функции предыстории (или постоянного вектора) и границ отрезка  $[a, b]$ .
4. Визуализация результата, который солвер `dde23` возвращает в структуре.

Разберем решение дифференциальных уравнений на следующем модельном примере: требуется найти решение системы на отрезке  $[0, 4]$  с предыстроей  $S(t)$ :

$$\begin{cases} y'_1 = \frac{\pi}{3}(y_2(t) + y_1(t - 1.5) + y_2(t - 2)); \\ y'_2 = -\frac{\pi}{3}(y_1(t) + y_1(t - 2) + y_2(t - 0.5)); \end{cases} \quad S(t) = \begin{pmatrix} \sin \pi t \\ \cos \pi t \end{pmatrix}. \quad (6.9)$$

Начнем с программирования функции `ddefun`. Ее входными аргументами должны являться: время  $t$ , вектор  $y$  компонент решения и матрица  $Z$ , столбцы которой содержат значения компонент решения в моменты запаздываний. В нашем случае  $\tau_1 = 0.5$ ,  $\tau_2 = 1.5$ ,  $\tau_3 = 2$  и матрица  $Z$  имеет следующий вид:

$$Z = \begin{pmatrix} y_1(t - \tau_1) & y_1(t - \tau_2) & y_1(t - \tau_3) \\ y_2(t - \tau_1) & y_2(t - \tau_2) & y_2(t - \tau_3) \end{pmatrix}$$

В функции `ddefun` мы ввели вспомогательные переменные для вектор-функции  $y(x)$ , вычисленной в моменты запаздываний (листинг 6.26).

**Листинг 6.26. Файл-функция `ddefun` для правой части системы уравнений (6.9)**

```
function F = ddefun(t, y, Z)
% Вычисление правой части системы запаздывающих дифференц. уравнений
% t - время
% y - вектор со значениями компонент y(1), y(2), ..., y(k)
% Z - матрица со значениями компонент в момент запаздываний:
% Z(:, j) = y(t - tau(j)), j = 1, 2, ..., k
```

```

Y1 = Z(:,1); % вектор значений компонент для t = tau(1)
Y2 = Z(:,2); % вектор значений компонент для t = tau(2)
Y3 = Z(:,3); % вектор значений компонент для t = tau(3)
F = [pi/3*(y(2) + Y2(1) + Y3(2))
 -pi/3*(y(1) + Y3(1) + Y1(2))];
```

Перейдем теперь к написанию функции `ddehistory` с предысторией решения. В нашем случае она простая, ее текст приведен в листинге 6.27.

#### Листинг 6.27. Файл-функция `ddehistory` предыстории решения

```

function S = ddehistory(t)
% вычисление вектора компонент предыстории решения
S = [sin(pi*t)
 cos(pi*t)];
```

Осталось вызвать солвер `dde23`, графически отобразить решение и проанализировать его. В отличие от солверов для решения систем обыкновенных дифференциальных уравнений, солвер `dde23` возвращает решение только в одном виде — в структуре:

```
>> sol = dde23(@ddefun, [0.5 1.5 2], @ddehistory, [0 4]);
```

Содержимое структуры `sol` следующее:

- `sol.x` — вектор-строка со значениями независимой переменной  $t$ , для которых найдено приближенное решение;
- `sol.y` — массив со значениями  $y(t)$ , каждая строка которого содержит соответствующую компоненту решения;
- `sol.yr` — массив со значениями  $y'(t)$ , каждая строка которого содержит первую производную от соответствующей компоненты решения.

При анализе решения примите во внимание, что точным решением модельной задачи являются функции  $y_1(t) = \sin \pi x$  и  $y_2(t) = \cos \pi x$ . Постройте графики каждой компоненты точного и приближенного решения на своей паре осей в одном графическом окне и сравните их. Имеет смысл написать файл-функцию, включив в нее и вызов солвера, и все вспомогательные функции (листинг 6.28).

#### Листинг 6.28. Файл-функция `ddetest`

```

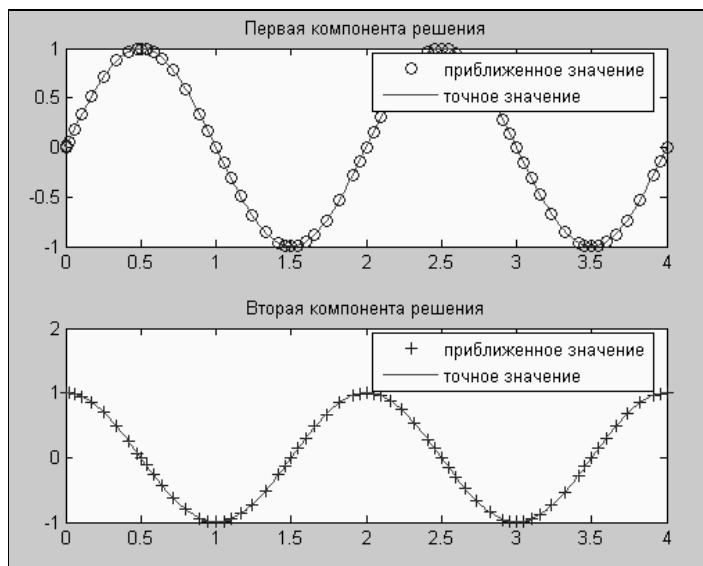
function ddetest
% вызов солвера dde23
```

```

sol = dde23(@ddefun, [0.5 1.5 2],@ddehistory, [0 4]);
figure
subplot(2, 1, 1)
% график приближенного значения первой компоненты
plot(sol.x, sol.y(1, :), 'o')
% задание первой компоненты точного решения
y1 = inline('sin(pi*t)');
hold on
% построение графика первой компоненты точного значения
fplot(y1, [0,4], 'r')
title('Первая компонента решения')
legend('приближенное значение', 'точное значение')
subplot(2, 1, 2)
% график приближенного значения второй компоненты
plot(sol.x, sol.y(1, :), '+') % задание первой компоненты точного решения
y2 = inline('cos(pi*t)');
hold on
% построение графика второй компоненты точного значения
fplot(y2, [0, 4], 'r')
title('Вторая компонента решения')
legend('приближенное значение', 'точное значение')
%
function S = ddehistory(t)
% вычисление вектора компонент предыстории решения
S = [sin(pi*t)
 cos(pi*t)];
%
function F = ddefun(t, y, Z)
% Вычисление правой части системы запаздывающих дифференц. уравнений
% t - время
% y - вектор со значениями компонент y(1), y(2), ..., y(k)
% Z - матрица со значениями компонент в момент запаздываний:
% Z(:, j) = y(t - tau(j)), j = 1, 2, ..., k
Y1 = Z(:, 1); % вектор значений компонент для t = tau(1)
Y2 = Z(:, 2); % вектор значений компонент для t = tau(2)
Y3 = Z(:, 3); % вектор значений компонент для t = tau(3)
F = [pi/3*(y(2) + Y2(1) + Y3(2))
 -pi/3*(y(1) + Y3(1) + Y1(2))];

```

После вызова `ddestest` получаются графики, приведенные на рис. 6.17. Они свидетельствуют о правильно найденном решении.



**Рис. 6.17.** Решение дифференциального уравнения (6.9) с запаздывающим аргументом

### Примечание

Для вычисления значений приближенного решения в произвольных точках отрезка следует применить функцию `deval` так же, как и в случае рассмотренной выше задачи Коши.

Обсудим теперь задание параметров вычислительного процесса. Управляющая структура формируется функцией `ddeset`, использование которой аналогично `odeset` при решении задачи Коши для систем обыкновенных дифференциальных уравнений. Солвер `dde23` позволяет задавать относительную и абсолютную точность (по умолчанию  $10^{-3}$  и  $10^{-6}$  соответственно), способ контроля точности — по норме или покомпонентно, начальный и максимальный шаг, например:

```
>> options = ddeset('RelTol', 1e-5, 'AbsTol', 1e-8)
>> sol = dde23(@ddefun, [0.5 1.5 2], @ddehistory, [0 4], options)
```

Отслеживание событий и обработка получаемых в процессе решения значений, в том числе и визуализация приближенного решения, требуют привлечения свойств `Events`, `OutputFcn` и `OutputSel`. Для вывода суммарной информации о вычислительном процессе следует воспользоваться свойством `Stats`.

При решении дифференциальных уравнений с запаздывающим аргументом существенным обстоятельством является потеря непрерывности производных решения низких порядков. Обычно разрывы возникают уже в начальной точке отрезка и повторяются затем через моменты времени, совпадающие с запаздываниями, но для производных более высоких порядков. Поэтому набор свойств солвера `ode23` пополнен опцией `Jumps`, значением которой должен быть вектор с координатами разрывов как предыстории, так и коэффициентов дифференциальных уравнений.

Начальным значением искомой вектор-функции  $y(a)$  по умолчанию является  $S(a)$ . Вообще говоря, начальное значение может быть отличным от  $S(a)$ , в этом случае поле `InitialY` управляющей структуры должно содержать заданный вектор  $y(a)$ . Примеры решения дифференциальных уравнений с запаздывающим аргументом и их обсуждение приведены в справочной системе MATLAB в пунктах разд. **Mathematics: Differential Equations: Initial Value Problems for DDEs**.

В завершение этого раздела обратимся к несколько измененной модели Лотки—Вольтерры, в которой увеличение количества хищников происходит по истечении времени  $\tau$  после встречи хищника и жертвы (решение уравнений Лотки—Вольтерры описано выше в этой главе).

Этот процесс описывается системой дифференциальных уравнений с запаздывающим аргументом

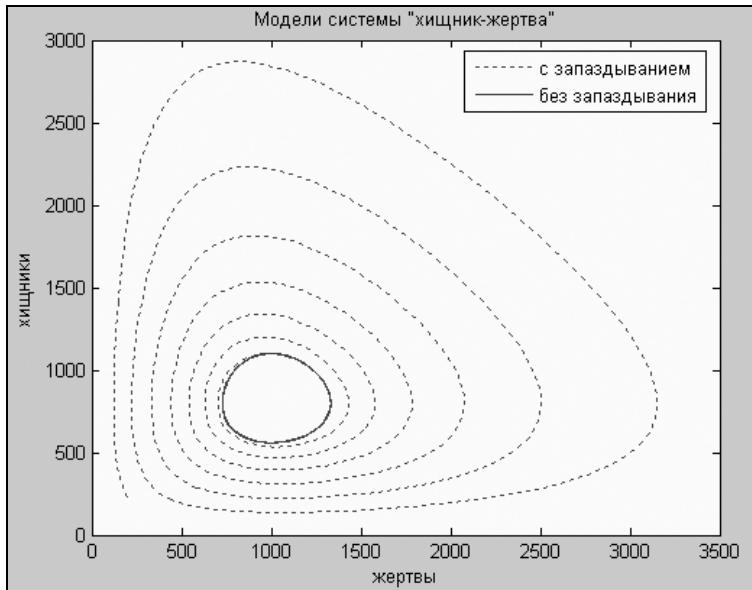
$$\begin{cases} y'_1(t) = P \cdot y_1(t) - p \cdot y_1(t) y_2(t); \\ y'_2(t) = -R \cdot y_2(t) + r \cdot y_1(t - \tau) y_2(t - \tau). \end{cases} \quad (6.10)$$

Возьмите  $\tau = 0.1$ , а значения коэффициентов те же, что и для рассмотренных выше уравнений Лотки—Вольтерры (6.5):  $P = 0.8$ ,  $R = 1$ ,  $p = r = 0.001$ . Будем считать предысторию постоянной — 1000 жертв и 1100 хищников. Запрограммируйте функции: `LotVolDel` — для вычисления правой части системы и `LotVolHis` — для предыстории решения, причем предусмотрите возможность указания значений параметров при вызове солвера после входного аргумента `options`. Решите систему дифференциальных уравнений с запаздыванием на временном отрезке  $[0, 50]$  и сравните полученный ре-

зультат с решением аналогичной системы без запаздывания. При решении систем установите одинаковую относительную точность, к примеру  $10^{-5}$ . Тексты функций приведены в листинге 6.29, а получающийся после выполнения файл-функции график — на рис. 6.18. При вызове `dde23` постоянную предысторию можно было бы задать вектором вместо указателя на функцию `LotVolHis`. Мы использовали функцию для того, чтобы подчеркнуть необходимость дополнительных входных аргументов в случае, когда правая часть системы дифференциальных уравнений зависит от параметров. Разумеется, этот же пример можно выполнить и с помощью вложенных или анонимных функций.

#### Листинг 6.29. Файл-функция для сравнения двух моделей

```
function LotVol2(P, p, R, r)
% решение системы дифференциальных уравнений с запаздыванием
options = ddeset('RelTol', 1.0e-05); % установка относительной точности
% вызов солвера с указанием коэффициентов системы
sol = dde23(@LotVolDel, 0.1, @LotVolHis, [0 50], options, P, p, R, r);
plot(sol.y(1, :), sol.y(2, :), ':') % вывод графика приближенного решения
% решение системы дифференциальных уравнений без запаздывания
options = odeset('RelTol', 1.0e-05) % установка относительной точности
% вызов солвера с указанием коэффициентов системы
[T, Y] = ode23s(@LotVolPar, [0 50], [1000 1100], options, P, p, R, r);
hold on
plot(Y(:, 1), Y(:, 2)) % вывод графика приближенного решения
% вывод пояснений на график
title('Модели системы "хищник-жертва"')
xlabel('жертвы')
ylabel('хищники')
legend('с запаздыванием', 'без запаздывания')
% подфункция LotVolDel для вычисления правой части системы (6.10)
function F = LotVolDel(t, y, Z, P, p, R, r)
F = [P*y(1) - p*y(1)*y(2); -R*y(2) + r*Z(1)*Z(2)];
% подфункция LotVolHis для вычисления предыстории решения
function H = LotVolHis(t, P, p, R, r)
H = [1000; 1100];
```



**Рис. 6.18.** Решение уравнения Лотки—Вольтерры (6.10) с запаздывающим аргументом

## Решение граничных задач

В данном разделе на простых примерах иллюстрируются возможности решения граничных задач для обыкновенных дифференциальных уравнений и систем дифференциальных уравнений первого порядка в MATLAB при помощи солвера `bvp4c`. Описаны возможности солвера для решения задач с особенностями и нахождения неизвестных параметров дифференциальных уравнений.

### Схема решения

Рассмотрим решение граничных задач на примере обыкновенного дифференциального уравнения второго порядка. Требуется найти функцию  $y(x)$ , удовлетворяющую на отрезке  $[a, b]$  дифференциальному уравнению

$$y'' = f(x, y, y')$$

и граничным условиям

$$\alpha \cdot y(a) + \beta \cdot y'(a) = A, \quad \gamma \cdot y(b) + \delta \cdot y'(b) = B.$$

Здесь  $\alpha, \beta, \gamma, \delta, A, B$  — заданные числа.

Решение этой задачи состоит из следующих этапов.

1. Преобразование дифференциального уравнения второго порядка к системе двух уравнений первого порядка.
2. Написание функции для вычисления правой части системы.
3. Написание 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 (см. разд. "Решение задачи Коши" данной главы).

Введение вспомогательных функций  $y_1(x)$  и  $y_2(x)$  приводит к системе уравнений первого порядка относительно них:

$$\begin{cases} y'_1 = y_2; \\ y'_2 = f(x, y_1, y_2). \end{cases}$$

Функция правой части системы зависит от  $x$  и вектора  $y$ , состоящего из двух компонент,  $y(1)$  соответствует  $y_1$ , а  $y(2) — y_2$ , и программируется так же, как при решении задачи Коши.

При постановке граничных условий для вспомогательных функций требуется преобразовать их так, чтобы в *правых частях* стояли нули:

$$\alpha \cdot y_1(a) + \beta \cdot y_2(a) - A = 0, \quad \gamma \cdot y_1(b) + \delta \cdot y_2(b) - B = 0.$$

Функция, описывающая граничные условия, зависит от двух аргументов — векторов  $ya$  и  $yb$  и имеет следующую структуру (листинг 6.30).

#### Листинг 6.30. Файл-функция `bound` для задания граничных условий

```
function g = bound(ya, yb)
g = [alpha*ya(1) + beta*ya(2) - A; gamma*yb(1) + delta*yb(2)];
```

Вместо `alpha`, `beta`, `gamma`, `delta`, `A` и `B` следует подставить заданные числа.

Солвер `bvp4c` основан на методе конечных разностей, т. е. получающееся решение есть *векторы значений* неизвестных функций в точках отрезка (в

узлах сетки). Выбор начальной сетки и приближения может оказать влияние на решение, выдаваемое солвером `bvp4c`. Для задания начальной сетки и приближения предназначена функция `bvpinit`, обращение к которой в самом простом случае выглядит следующим образом:

```
initsol = bvpinit(начальная сетка, начальное приближение к решению)
```

Начальная сетка определяется вектором координат узлов, упорядоченных по возрастанию и принадлежащих отрезку  $[a, b]$ . Если имеется априорная информация о решении, то разумно среди точек начальной сетки указать те, в которых решение сильно изменяется. Формирование равномерной сетки целесообразно производить функцией `linspace`:

```
X = linspace(a, b, n)
```

возвращающей вектор  $x$  из  $n$  равноотстоящих узлов между  $a$  и  $b$ , включая границы. Заданная сетка модифицируется солвером в процессе решения для обеспечения требуемой точности. Постоянное начальное приближение задается вектором из двух элементов для функций  $y_1(x)$ ,  $y_2(x)$ . Начальное приближение может зависеть от  $x$ , в этом случае требуется запрограммировать функцию, которая по заданному  $x$  возвращает вектор из двух компонент со значениями  $y_1(x)$ ,  $y_2(x)$ , и поместить указатель на нее во втором входном аргументе `bvpinit`. В результате работы `bvpinit` генерируется структура `initsol` с информацией о начальном приближении.

После определения начального приближения вызывается солвер `bvp4c`, входными аргументами которого являются указатели на функции правой части системы и граничных условий, начальное приближение и, при необходимости, управляющая структура с параметрами вычислительного процесса. Управляющая структура формируется при помощи функции `bvpset`. Солвер `bvp4c` возвращает единственный выходной аргумент — структуру с информацией о расчетной сетке, значения искомой функции и ее производной. Солвер `bvp4c` так же, как и солвер `ode23` для дифференциальных уравнений с запаздыванием, позволяет получить результат только в виде структуры, в отличие от солверов для задачи Коши (работа со структурами подробно описана в главе 8).

Для вычисления значений приближенного решения в произвольных точках отрезка следует применить функцию `deval` так же, как и в случае рассмотренной выше задачи Коши. В следующих разделах процесс решения граничных задач разобран на простом модельном примере; кроме того, приведены возможности солвера `bvp4c` для исследования более широких классов граничных задач.

## Простой пример граничной задачи

Требуется решить граничную задачу для обыкновенного дифференциального уравнения второго порядка

$$y'' = -\sin x, \quad y(0) = 0, \quad y'(11\pi/2) + y(11\pi/2) = -1.$$

при помощи солвера `bvp4c` и сравнить полученное решение с точным  $y = \sin x$ .

Система дифференциальных уравнений первого порядка, соответствующая исходному уравнению, и граничные условия для нее находятся просто:

$$\begin{cases} y'_1 = y_2; \\ y'_2 = -\sin x; \end{cases} \quad y_1(0) = 0, \quad y_2(11\pi/2) + y_1(11\pi/2) + 1 = 0. \quad (6.11)$$

Написать функцию `rside` для системы уравнений также не представляет большого труда, ее текст приведен в листинге 6.31.

Как было указано в предыдущем разделе, функция для граничных условий имеет два входных аргумента. Каждый аргумент является вектором значений неизвестных функций  $y_1$  и  $y_2$  в начальной и конечной точке промежутка. Поэтому функция граничных условий должна быть такой, как `bound` в листинге 6.31.

Решение граничной задачи оформите в виде файл-функции, в которой необходимо:

- При помощи `bvpinit` задать начальную сетку на отрезке  $[0, 11\pi/2]$ , например, из 10 узлов, и постоянное начальное приближение:  $y_1(x) = 1$ ,  $y_2(x) = 0$ .
- Вызвать солвер `bvp4c` и получить приближенное решение.
- Отобразить графически приближенное решение, извлекая нужные компоненты из структуры, возвращаемой солвером.
- Добавить график точного решения  $y = \sin(x)$  на отрезке  $[0, 11\pi/2]$ .

Текст файл-функции содержится в листинге 6.31.

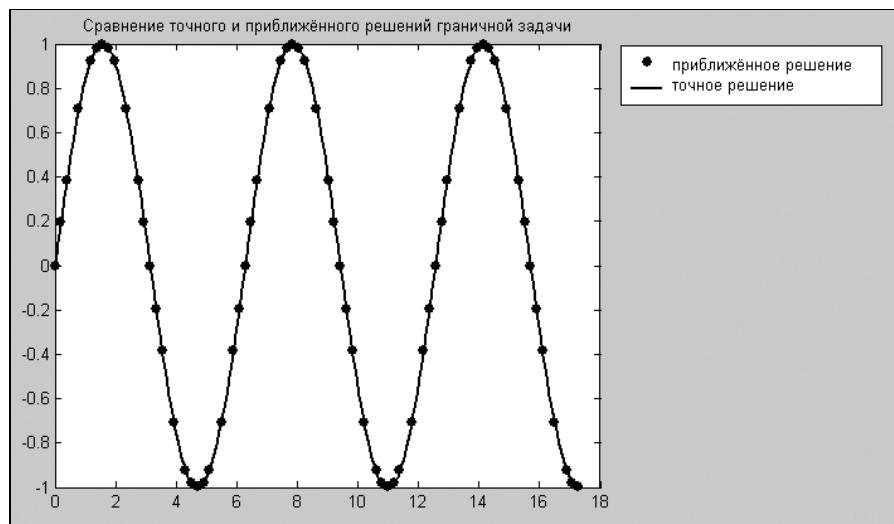
### Листинг 6.31. Файл-функция для решения граничной задачи (6.11)

```
function boundproblem
% формирование начальной сетки и постоянного приближения y1 = 1, y2 = 0
meshinit = linspace(0, 11*pi/2, 20);
yinit = [1 0];
```

```

initSol = bvpinit(meshInit, yInit);
% 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
hold on
fplot(@(x), [0 11*pi/2])
title('Решение граничной задачи солвером bvp4c')
legend('приближенное решение', 'точное решение')
% функция rside для правых частей системы уравнений (6.11)
function f = rSide(x, y)
f = [y(2); -sin(x)];
% функция bound граничных условий
function f = bound(ya, yb)
f = [ya(1); yb(2) + yb(1) + 1];

```



**Рис. 6.19.** Сравнение приближенного и точного решений граничной задачи (6.11)

Получающиеся графики приближенного и точного решений исходной задачи приведены на рис. 6.19, они свидетельствуют о том, что для простых задач солвер `bvp4c` дает хорошие результаты. По умолчанию решение найдено с относительной точностью  $10^{-3}$  и абсолютной  $10^{-6}$  по невязке. Задание точности вычислений, ряда других опций солвера `bvp4c` и решение более сложных задач обсуждается в следующих разделах.

## Возможности солвера `bvp4c`, управление вычислениями

Солвер `bvp4c` позволяет решать граничные задачи для систем обыкновенных дифференциальных уравнений произвольного порядка  $n$

$$y' = f(x, y), \quad x \in [a, b], \quad g(y(a), y(b)) = 0.$$

Здесь  $y$  — неизвестная вектор-функция;  $g$  — вектор-функция граничных условий. Правило программирования функции для вычисления правой части системы и вектор-функции граничных условий остается тем же, что и в случае системы второго порядка, рассмотренной в предыдущем разделе. Точность решения контролируется при помощи двух опций управляющей структуры `RelTol` — для относительной точности и `AbsTol` — для абсолютной. Сама управляющая структура генерируется функцией `bvpset`

```
Options = bvpset('RelTol', 1.0e-05, 'AbsTol', 1.0e-07)
```

и задается в качестве четвертого входного аргумента солвера `bvp4c`. Абсолютная точность участвует в оценке каждой компоненты вектор-функции невязки  $r(x) = \tilde{y}'(x) - f(x, \tilde{y}(x))$ , где  $\tilde{y}(x)$  — приближенное решение, и может быть числом, либо вектором из  $n$  элементов. Параметр `RelTol` используется для оценки интегральной нормы  $\| \cdot \|$  компонент невязки  $r_i$  на каждом из участков расчетной сетки. Вычисления останавливаются при выполнении условия:

$$\| r_i / \max\{|f_i|, \text{AbsTol}(i) / \text{RelTol}\} \| < \text{RelTol}.$$

Возможна ситуация, когда решение граничной задачи найдено, но по каким-то причинам требуется вычислить его с большей точностью. В этом случае в качестве начального приближения для солвера `bvp4c` допускается указание полученной структуры с приближенным решением. Например, в примере предыдущего раздела было получено решение с установленной по умолчанию точностью:

```
sol = bvp4c(@rsode, @bound, initSol);
```

Для нахождения решения, скажем, с относительной точностью  $10^{-7}$  достаточно сформировать управляющую структуру

```
options = bvpset('RelTol', 1e-7);
```

и обратиться к солверу

```
sol = bvp4c(@rside, @bound, sol, options);
```

Солвер может прервать вычисления, не достигнув заданной точности, поскольку по умолчанию число узлов расчетной сетки не должно превышать целой части от  $1000/n$ . При появлении такого сообщения следует увеличить максимально допустимое число узлов, установив подходящее значение для параметра `Nmax` управляющей структуры.

В алгоритме солвера `bvp4c` частные производные функций  $g$  и  $f$  аппроксимируются конечными разностями. Использование аналитических выражений для них может повысить эффективность вычислений. В случае системы линейных уравнений матрица Якоби правой части системы постоянна и содержащий ее массив задается в качестве значения свойства `FJacobian`. Для нелинейных систем потребуется запрограммировать функцию, так же, как и в случае решения задачи Коши (решение задачи Коши для систем обыкновенных дифференциальных уравнений описано выше в *этой главе*).

Функция, вычисляющая производные вектор-функции  $g(y_a, y_b)$  граничных условий  $\partial g / \partial y_a$ ,  $\partial g / \partial y_b$ , должна иметь заголовок

```
function [dgdy_a, dgdy_b] = boundjac(ya, yb)
```

и по заданным  $y(a)$  и  $y(b)$  возвращать векторы  $\partial g / \partial y_a$ ,  $\partial g / \partial y_b$ . Солвер `bvp4c` будет вызывать `boundjac` для нахождения частных производных, если свойству `BCJacobian` управляющей структуры установить значение `@boundjac`.

Ускорение работы солвера возможно за счет специального способа программирования функции для правой части системы. В примере предыдущего раздела (листинг 6.31) функция `rside` по заданной паре значений  $x$  и  $y(x)$  вычисляла  $f(x, y(x))$ . Альтернативный вариант интерфейса предполагает возможность вычисления матрицы  $F = f(X, y(X))$  для заданного вектора значений независимой  $X$  переменной и матрицы  $y(X)$ . Модифицированная функция приведена в листинге 6.32.

### Листинг 6.32. Векторизованная файл-функция для правой части системы (6.11)

```
function F = rsidevect(X, Y)
% Вычисление правой части системы дифференциальных уравнений
```

```
% для вектора значений X. Возвращается матрица F, каждый столбец
% которой F(:, j) есть f(x(j), y(x(j))).
L = length(X); % определение количества значений независимого аргумента
F = zeros(2, L); % создание нулевой матрицы F подходящего размера
% вычисление компонент вектор-функции правой части для всех значений
% независимой переменной
F(1, :) = Y(2, :); % вычисление первой компоненты
F(2, :) = -sin(X); % вычисление второй компоненты
```

Для переключения солвера `bvp4c` на использование векторизованной функции служит свойство `Vectorized` управляющей структуры, которое по умолчанию равно `'off'`. В листинге 6.31 следует заменить подфункцию `rside` на `rsidevect` и изменить обращение к солверу:

```
options = bvpset('Vectorized', 'on');
sol = bvp4c(@rsidevect, @bound, initSol, options);
```

Суммарная информация о вычислениях — количество узлов расчетной сетки, максимум невязки и число обращений к функциям правой части системы и граничных условий — выводится в командное окно, если значением свойства `Stats` является `'on'`.

Класс задач, решаемых солвером `bvp4c`, включает в себя граничные задачи для систем обыкновенных дифференциальных уравнений с неизвестными параметрами и систем с вырождающимися коэффициентами. В следующих двух разделах демонстрируется решение этих задач на простых примерах.

## Граничные задачи с неизвестными параметрами

Солвер `bvp4c` так же, как и солверы для задач Коши и уравнений с запаздыванием, позволяет решать граничные задачи для систем обыкновенных дифференциальных уравнений с параметрами. Однако в случае граничных задач параметры могут быть как *известные*, так и *подлежащие определению* в процессе вычислений. Рассмотрим общую постановку граничной задачи с подлежащими определению параметрами. Требуется найти вектор-функцию  $y(x)$  и вектор параметров  $p$ , удовлетворяющие на отрезке  $[a, b]$  системе дифференциальных уравнений  $y' = f(x, y, p)$  и граничным условиям:  $g(y(a), y(b), p) = 0$ . Решение таких задач предполагает указание начального приближения как для искомой вектор-функции, так и для всех параметров. Для этого следует инициализировать структуру с начальным приближением, задав три входных аргумента: `meshinit` — массив узлов начальной сетки, `yinit` — начальное приближение для искомой вектор-

функции (постоянный вектор или указатель на функцию) и `parinit` — вектор стартовых значений параметров.

```
solinit = bvpinit(meshinit, yinit, parinit)
```

Функция `bvpinit` создает структуру `solinit`, поле `parameters` которой содержит начальные приближения для неизвестных параметров. При программировании функций для вычисления правой части системы и граничных условий следует предусмотреть возможность передачи значений параметров в третьем входном аргументе — векторе. Заголовки функций должны иметь вид:

```
function f = rsidepar(x, y, par) и function f = boundpar(ya, yb, par)
```

В случае успешного завершения вычислений солвер `bvp4c` возвращает структуру `sol`

```
>> sol = bvp4c(@rsidepar, @boundpar, solinit)
```

Узлы сетки записаны в `sol.x`, приближение к вектор-функции в `sol.y`, а найденное значение параметров в `sol.parameters`.

Разберем поиск неизвестных параметров граничных задач на модельном примере с одним параметром. Требуется найти функцию  $u(x)$  и число  $\lambda$ , удовлетворяющие на отрезке  $[0, \pi]$  дифференциальному уравнению

$$u'' + 2\lambda u' + 2\lambda^2 u = 0$$

и граничным условиям  $u(0) = 0$ ,  $u(\pi) = 0$ .

Такого класса задачи часто возникают при определении собственных чисел и собственных функций дифференциальных операторов. Приведенная выше задача, очевидно, имеет тривиальное решение  $u(x) = 0$ , не представляющее практического интереса. Как правило, имеется целый спектр параметров, при которых существует ненулевое решение  $u(x)$ , определенное с точностью до мультипликативной константы. Наша задача не полностью определена, поскольку если  $u(x)$  — решение, то  $c \cdot u(x)$  также является решением для любой константы  $c$ . Следовательно, необходимо подчинить искомую вектор-функцию некоторому нормировочному условию, например:  $u'(0) = 1$ .

Введем обозначения  $y_1(x) = u(x)$ ,  $y_2(x) = u'(x)$  и перейдем от дифференциального уравнения второго порядка к системе обыкновенных дифференциальных уравнений первого порядка с соответствующими граничными условиями

$$y_1(0) = 0, \quad y_1(\pi) = 0, \quad y_2(0) - 1 = 0. \quad (6.12)$$

### Примечание

В такой постановке эту задачу можно трактовать, как задачу Коши с дополнительным условием на другом конце для определения параметра. При этом нельзя гарантировать существование решения.

Создадим теперь функции `rsidepar` для вычисления правой части системы и `boundpar` для граничных условий в соответствии с приведенным выше правилом (листинг 6.32). Важно, что хотя граничные условия не зависят от параметра, все равно третий аргумент `par` должен формально присутствовать в списке входных аргументов функции `boundpar`.

Осталось запрограммировать функцию для определения начального приближения, которое существенным образом влияет на результат вычислений для задач с неизвестными параметрами. В нашем примере хорошим начальным приближением может оказаться  $y_1(x) = \sin x$  (и, соответственно  $y_2(x) = \cos x$ ), поскольку оно точно удовлетворяет всем трем граничным условиям. Текст функции `yinit` приведен в листинге 6.33.

Теперь сгенерируйте структуру `solinit` с информацией о начальном приближении, выбрав сетку с 10 равноотстоящими узлами на промежутке  $[0, \pi]$  и взяв  $\lambda = 0.8$  в качестве стартового значения параметра. Вызовите солвер `bvp4c`, отобразите графически первую компоненту вектор-функции решения (т. е. искомую функцию  $u(x)$ ) и найдите параметр  $\lambda$ . Текст файл-функции `boundparproblem`, осуществляющей перечисленные действия, приведен в листинге 6.33. В качестве выходного аргумента она возвращает значение искомого параметра.

#### Листинг 6.33. Файл-функция для решения граничной задачи с параметром (6.12)

```
function par = boundparproblem
Meshinit = linspace(0, pi, 10); % генерация начальной сетки
Pinit = 0.8; % стартовое приближение для параметра
% генерация структуры с информацией о начальном приближении
solinit = bvpinit(meshinit, @yinit, pinit);
% вызов солвера
sol = bvp4c(@rsidepar, @boundpar, solinit);
% занесение в par найденного значения параметра
par = sol.parameters
```

```
% построение графика первой компоненты найденной вектор-функции
plot(sol.x, sol.y(1,:), 'o')

%
% функция правой части системы дифференциальных уравнений (6.12)
function f = rsidepar(x, y, par)
f = [y(2); -2*par*y(2) - 2*par^2*y(1)];
end

% функция граничных условий
function f = boundpar(ya, yb, par)
f = [ya(1); yb(1); ya(2) - 1];
end

% функция начального приближения
function f = yinit(x);
f = [sin(x); cos(x)];
end
```

В результате работы файла-функции получается  $\lambda = 0.99999361754175$  (т. е.  $\lambda=1$  с достаточной степенью точности) и функция  $u(x)$ , представленная на рис. 6.20.

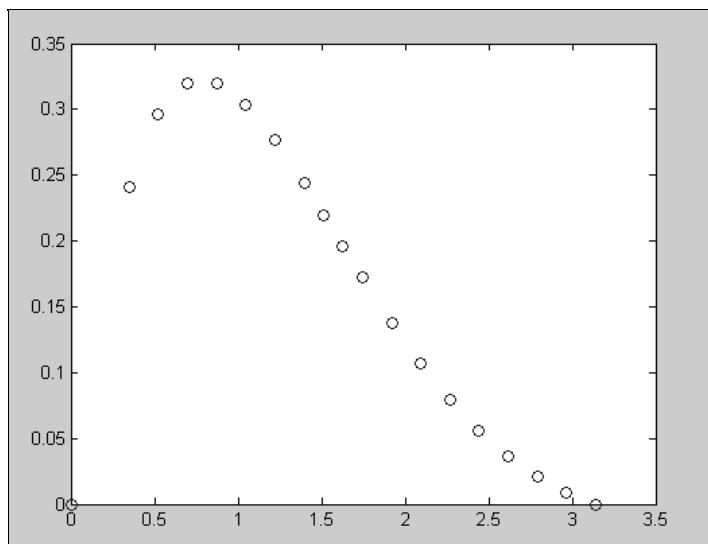


Рис. 6.20. Решение граничной задачи с параметром (6.12)

В нашей модельной задаче известно точное решение. Значениями  $\lambda$  могут быть целые числа, а соответствующими функциями  $u(x)$  являются

$$u(x) = \frac{1}{\lambda} e^{-\lambda x} \sin \lambda x.$$

Добавьте график функции  $u(x) = e^{-x} \sin x$ , отвечающей  $\lambda = 1$ , к графику приближенного решения и убедитесь, что оно вычислено верно. Найдите еще несколько значений  $\lambda$  и соответствующих функций  $u(x)$ , задавая в качестве стартового приближения к параметру числа, близкие к целым: 2.3, 3.2, 4.2, 5.1. Солвер возвращает хорошие приближения к точным (целым) значениям искомых параметров. Заметьте, что с увеличением значения параметра возрастает число смен знака соответствующей ему функции  $u(x)$  на отрезке  $[0, \pi]$ .

Обсудим теперь влияние начального приближения к вектор-функции. Например, исходный выбор  $y_1(x) = \sin x$ ,  $y_2(x) = \cos x$  не позволяет определить значение параметра даже для хорошего начального приближения 6.001 к точному значению 6 параметра  $\lambda$ . Следовательно, необходимо каким-то образом выбрать другое начальное приближение к вектор-функции. Конечно, точное решение свидетельствует о том, что хорошим начальным приближением может служить  $y_1(x) = \sin 6x$ ,  $y_2(x) = 6 \cos 6x$ . Но если бы точное решение было неизвестно, то можно было бы воспользоваться нашим наблюдением об увеличении количества участков смены знака решения  $u(x)$  при возрастании  $\lambda$ . Модифицируйте функцию `yinit` и найдите решение для стартового значения 6.2 параметра  $\lambda$ .

Как было отмечено в начале этого раздела, в общем случае граничная задача может содержать как неизвестные, так и известные параметры. Наличие неизвестных параметров приводит к более сложной задаче с вычислительной точки зрения, в то время как присутствие известных параметров требует только использования расширенного интерфейса солвера `bvp4c` и функций, к которым он обращается.

В общем случае с неизвестными и известными параметрами заголовки функций граничных условий и правой части системы дифференциальных уравнений должны иметь вид:

```
function f = rsidepar(x, y, par, p1, p2, ...)
res = boundpar(ya, yb, par, p1, p2, ...)
end
```

где `par` — вектор неизвестных параметров, а аргументы `p1, p2, ...` являются заданными параметрами. Если неизвестных параметров нет, то заголовки функций граничных условий и правой части системы дифференциальных уравнений упрощаются:

```
function f = rsidepar(x, y, p1, p2,...)
res = boundpar(ya, yb, p1, p2,...)
end
```

При вызове солвера значения известных параметров `p1, p2, ...` следует указать в списке входных аргументов, начиная с пятой позиции после управляющей структуры.

```
>> sol = bvp4c(odefun, bcfun, solinit, options, p1, p2...)
```

Разумеется, в случае отсутствия подлежащих вычислению параметров не требуется подбирать приближение для них при инициализации структуры с начальным приближением функцией `bvpinit`.

Эффективность вычислений может быть повышена за счет задействования аналитических выражений элементов матрицы Якоби правой части системы дифференциальных уравнений и частных производных вектор-функции граничных условий. Для получения информации об интерфейсе соответствующих функций обратитесь к описанию функции `bvpset`, воспользовавшись, например, индексным поиском в справочной системе MATLAB.

В следующем разделе приведен еще один класс задач — граничные задачи с вырождающимися коэффициентами — решение которых может быть получено при помощи солвера `bvp4c`.

## Решение задачи с особенностью на границе

Солвер `bvp4c` позволяет решать граничные задачи для дифференциальных уравнений с вырождающимися коэффициентами при условии, что они эквивалентны системе дифференциальных уравнений первого порядка

$$y' = \frac{1}{x} S y + f(x, y), \quad x \in [0, b], \quad b > 0$$

с граничными условиями  $g(y(0), y(b)) = 0$ . При этом матрица  $S$  должна быть постоянной и удовлетворять необходимому условию существования непрерывного решения:  $S y(0) = 0$ . Возможна и более общая постановка граничной задачи, включающая неизвестные параметры, которые могут входить как в вектор-функцию  $f$ , так и в вектор-функцию граничных условий  $g$ .

Последовательность действий при исследовании вырождающихся задач в целом схожа с решением рассмотренных выше классов граничных задач. Отличие состоит в том, что функция правой части системы должна вычислять только  $f(x, y)$ , а матрица  $S$  передается солверу в качестве значения свойства `SingularTerm` управляющей структуры.

Разберем решение вырождающихся граничных задач на модельном примере. Требуется найти решение уравнения

$$u'' - \frac{2}{x} u' + 2u = 2(1 - x^3)$$

на отрезке  $[0, 2]$ , удовлетворяющее граничным условиям:  $u'(0) = 0$ ,  $u(2) = -7$ . Легко убедиться, что точным решением является функция  $1 - x^3$ . Преобразуйте исходную задачу к граничной задаче для системы уравнений первого порядка, введя новые функции  $y_1(x) = u(x)$ ,  $y_1(\pi) = 0$ :

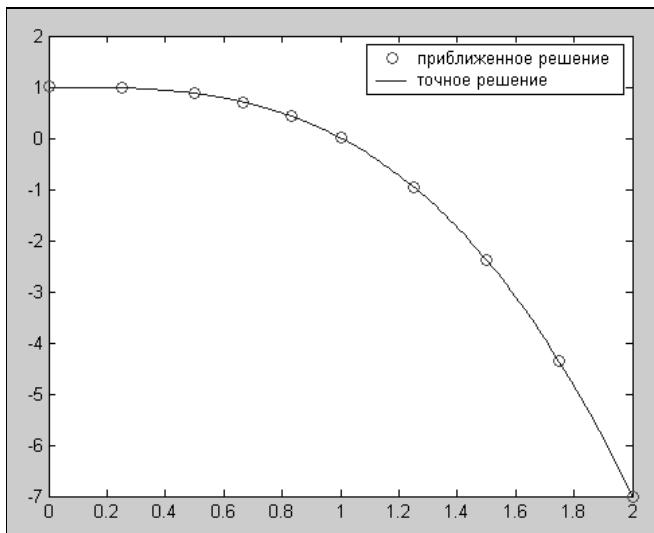
$$\begin{cases} y'_1 = y_2; \\ y'_2 = \frac{2}{x} y_2 - 2y_1 + 2(1 - x^3); \end{cases} \quad \begin{cases} y_1(2) + 7 = 0; \\ y_2(0) = 0. \end{cases} \quad (6.13)$$

Запись системы уравнений в векторном виде позволяет получить матрицу  $S$  и вектор-функцию  $f(x, y)$ :

$$y' = \frac{1}{x} Sy + f(x, y), \quad S = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix}, \quad f(x, y) = \begin{pmatrix} y_2 \\ -2y_1 + 2(1 - x^3) \end{pmatrix}.$$

Полученная задача входит в класс вырождающихся граничных задач, которые могут быть решены при помощи солвера `bvp4c`. Действительно, матрица  $S$  обеспечивает выполнение условия  $Sy(0) = 0$ , т. к.  $y_2(0) = 0$ .

Запрограммируйте функцию `rsbside` для вычисления  $f(x, y)$  и функцию `sbound` граничных условий. Определите подходящее начальное приближение, например, постоянное  $y_0 = [-7, 0]$ , для которого  $Sy_0 = 0$ . Задайте исходную сетку на отрезке  $[0, 2]$  из пяти равноотстоящих узлов и решите задачу с относительной точностью  $10^{-6}$ . Сравните графически полученное решение с точным, так как это сделано на рис. 6.21. В случае возникновения затруднений воспользуйтесь листингом 6.34.



**Рис. 6.21.** Сравнение приближенного решения вырождающейся граничной задачи с точным

**Листинг 6.34. Файл-функция для решения вырождающейся граничной задачи (6.13)**

```

function SingularProblem
S = [0 0; 0 2]; % Задание матрицы S
y0 = [-7 0]; % Определение начального приближения
meshinit = linspace(0, 2, 5); % Задание начальной сетки
% Формирование структуры с начальным приближением
solinit = bvpinit(meshinit, y0);
% Формирование управляющей структуры
options = bvpset('SingularTerm', S, 'RelTol', 1e-6)
% Вызов солвера
sol = bvp4c(@rsbside, @sbound, solinit, options)
% Построение графика первой компоненты решения
plot(sol.x, sol.y(1,:), 'or');
% Добавление графика точного решения и легенды
fun = inline('1 - x.^3');
hold on
fplot(fun, [0 2])

```

```

legend('приближенное решение', 'точное решение')
%
% функция для вычисления f(x, y)
function f = rsbside(x, y)
f = [y(2)
 -2*y(1) + 2*(1 - x.^3)];
%
% функция граничных условий
function g = sbound(ya, yb)
g = [ya(2)
 yb(1) + 7];

```

## Задания для самостоятельной работы

1. Вычислите интеграл от функции

$$\cos\left(x - \sqrt{2}\right)e^{2 \sin x} - 1$$

по промежутку между ее двумя соседними корнями, принадлежащими отрезку  $[0, 4]$ .

2. Вычислите интеграл от функции

$$\sin x - x^2 \cos x = 0$$

по промежутку между ее локальным максимумом и локальным минимумом, абсциссы которых принадлежат отрезку  $[-5, 0]$ .

3. Найдите корень уравнения  $f(z) = 0.1$ , если

$$f(z) = \int_0^z x \operatorname{tg}^2 x dx.$$

4. Решите систему дифференциальных уравнений для  $x \in [0, 5]$ :

$$\begin{cases} y'_1(x) = y_1(x-1); \\ y'_2(x) = -y_2(x-0.5) + y_1(x-1); \end{cases} \quad \begin{cases} S_1(x) = \sin x + 0.1; \\ S_2(x) = x - 2 \end{cases}$$

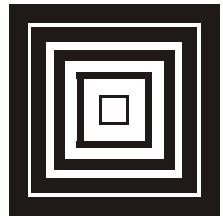
и для полученного решения вычислите:

- интеграл по всему промежутку от  $y_1^3(x)$ ;
- корни уравнения  $y_1(x) = -1$ .

5. Решите дифференциальное уравнение для  $x \in [0, 7]$ :

6.  $y'' + 3y - 0.1y^2 = 2\sin x, y(0) = 0.02, y'(0) = 1$

и для полученного решения найдите все локальные максимумы и минимумы  $y(x)$ .



## Глава 7

# Управляющие конструкции языка программирования

Файл-функции и файл-программы, которые вы создавали при чтении двух предыдущих глав, являются самыми простыми примерами программ. Все команды MATLAB, содержащиеся в них, выполняются *последовательно*. Для решения многих более серьезных задач требуются программы, в которых действия повторяются циклически, а в зависимости от некоторых условий выполняются различные части программы. В данной главе описаны управляющие конструкции языка программирования MATLAB, которые могут быть использованы при написании как файл-программ, так и файл-функций.

## Операторы цикла

Схожие и повторяющиеся действия выполняются при помощи операторов цикла `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`

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 каж-

дом 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 графики семейства кривых для  $x \in [0, 2\pi]$ , которое задано функцией  $y(x, a) = e^{-ax} \sin x$ , зависящей от параметра  $a$ , для значений параметра  $a$  от  $-0.1$  до  $0.1$  с шагом  $0.02$ . 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`. В редакторе М-файлов наберите текст файл-программы, приведенный в листинге 7.1, сохраните его в файле `FORdem1.m` и запустите файл-программу на выполнение (или из редактора М-файлов, или из командной строки, набрав в ней `FORdem1` и нажав `<Enter>`) (работа в редакторе М-файлов и запуск файл-программ описаны в главе 5).

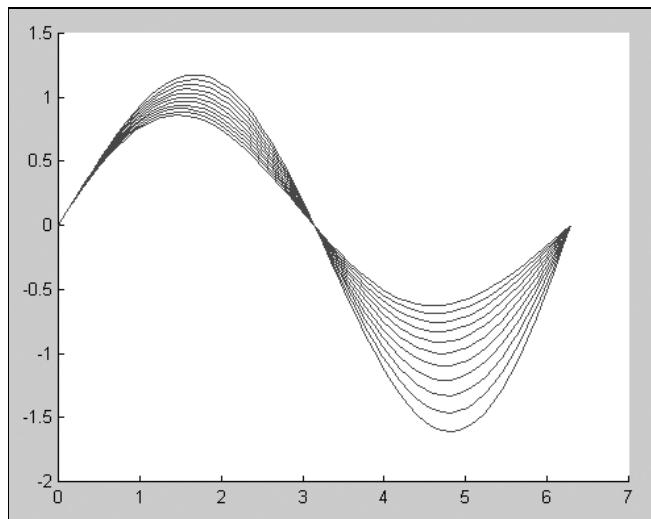
### Листинг 7.1. Файл-программа `FORdem1` для вывода семейства кривых

```
figure % создание графического окна
x = 0:pi/30:2*pi; % вычисление вектора значений аргумента
% перебор значений параметра в цикле
for a = -0.1:0.02:0.1
 % вычисление вектора значений функции для текущего значения параметра
 y = exp(-a*x).*sin(x); % добавление графика функции
 hold on
 plot(x, y)
end
```

### Примечание

Редактор М-файлов автоматически предлагает расположить операторы внутри цикла с отступом от левого края. Используйте эту возможность для структурирования текста программы. Редактор автоматически устанавливает отступы подходящей длины, если выбран переключатель **Smart indent** на вкладке **Language** для компоненты **Editor/Debugger** в диалоговом окне **Preferences**.

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  $1/k!$  (т. е.  $1/1!$ ), 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.

### Примечание

Если шаг цикла равен 1, то его можно не указывать.

#### 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 с шагом 1
for k = 1:10
 S = S + 1/factorial(k);
end
% вывод результата в командное окно
S
```

Наберите файл-программу в редакторе М-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 отображения значения переменной `S` в командное окно

```
S =
1.7183
```

Обратите внимание, что остальные строки файл-программы, которые могли бы повлечь вывод 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 назначении файл-программы `FORdem2`

```
>> help FORdem2
ФАЙЛ-ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ СУММЫ
1/1! + 1/2! + ... + 1/10!
```

При написании файл-программ и файл-функций не пренебрегайте комментариями, особенно если вы планируете продолжить работу с ними!

Важно понять, что все переменные, использующиеся в файл-программе, становятся доступными в рабочей среде (см. окно **Workspace**). 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, т. е. переменные всех файл-программ и рабочей среды являются общими. Это может послужить источником ошибок, поскольку файл-программа изменяет содержимое одноименных переменных рабочей среды, которые могли быть инициализированы из командной строки или в другой файл-программе.

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 `FORdem2`, 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 верхний предел суммы, а выходным — значение суммы  $S(x)$ . Использование конструкций языка программирования, в частности, циклов, в файл-функциях производится так же, как и в файл-программах.

Создавая файл-функцию, внесите некоторое усовершенствование в алгоритм. При вычислении  $k$ -го слагаемого  $a_k = x^k/k!$  требуется возвести  $x$  в степень  $k$  и вычислить факториал  $k$ . Для небольшого числа слагаемых эти действия выполняются практически мгновенно, но при увеличении числа слагаемых и большом наборе значений  $x$  время счета может оказаться весьма большим. В нашем примере несложно заметить связь между двумя соседними слагаемыми. Предположим, что уже вычислено  $a_5 = x^5/5!$ , тогда очевидно, что  $a_6 = a_4 \cdot x/6$ , а для произвольного  $k$  справедливо рекуррентное соотношение:  $a_k = a_{k-1} \cdot x/k$ , связывающее текущее и предыдущее слагаемые, в котором формально  $a_0 = 1$ . Достаточно часто такой подход позволяет эффективно вычислять суммы, возникающие в практических

задачах. Коэффициент, на который следует умножать предыдущее слагаемое, равен отношению  $a_k/a_{k-1}$ .

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;
%начальное значение, соответствующее m = 0
u = 1 ;
% накопление суммы в цикле
for m = 1:N
 u = u.*x/m;
 s = s + u;
end
```

О назначении и способе вызова файл-функции `sumN` пользователь может узнать, набрав в командной строке `help sumN`. В командное окно выводятся 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* или *локальные* переменные `m` и `u` файл-функции `sumN` не являются глобальными. Попытка просмотра значения переменной `m` или `u` из командной строки после выполнения файл-функции `sumN` приводит к сообщению о том, что функция, или переменная, не определена (разумеется, если она не была инициализирована до вызова `sumN`). 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 (см. разд. "Подфункции" главы 5).

Циклы `for` могут быть *вложены* друг в друга, при этом переменные вложенных циклов должны быть *разными*. Вложенные циклы удобны для заполнения матриц. Например, в разд. **MATLAB: Programming: Basic Program Components: Program Control Statements: Loop Control -- for, while, continue, break** справочной системы MATLAB содержится пример заполнения матрицы Гильберта при помощи вложенных циклов. Элементы матрицы Гильберта порядка  $n$  определяются формулами  $a_{i,j} = 1/(i + j - 1)$ , для  $i, j = 1, 2, \dots, n$ .

Скопируйте пример в новый М-файл при помощи буфера обмена Windows, сохраните с именем `HILdem.m` и продолжите работу с ним, как с файл-программой.

### Примечание

Не пренебрегайте возможностью запускать примеры, приведенные в справочной системе MATLAB, и изменять их по своему усмотрению, добиваясь желаемого результата.

Перед вычислением элементов матрицы целесообразно инициализировать ее при помощи функции `zeros`, которая создает массив, заполненный нулями. Иначе в цикле размеры массива будут динамически увеличиваться, что потребует дополнительного времени. 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);
% 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.

### Примечание

Инициализация любых используемых переменных считается хорошим стилем написания программ, она необходима во многих языках программирования. В MATLAB инициализация не обязательна, формально команду `a = zeros(n)` можно было пропустить, поскольку MATLAB автоматически увеличивает размеры массива по мере присвоения значений его элементам и выхода за верхнюю границу быть не может. Эта особенность может послужить источником ошибок. Пренебрежение инициализацией таит в себе еще одну опасность. Пусть в рабочей среде уже есть массив `a` размера 10 на 10, а в файл-программе `HILdem` инициализация отсутствует. Тогда после выполнения `HILdem` массив не изменит размеры, а элементы, строчные и столбцевые индексы которых превосходят четыре, останутся без изменений.

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, изменения  $n$  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  $i$  некоторых вспомогательных векторов.

В конце следует визуализировать результат (операции с матрицами, в частности, нахождению спектра, посвящен разд. "Задачи линейной алгебры" главы 6).

Напишите файл-функцию `HilSpect`, которая выводит на разные оси графического окна зависимости максимального и минимального собственных чисел матрицы Гильберта от порядка матрицы (для минимального собственного числа используйте логарифмическую шкалу по оси ординат). Входным аргументом файла-функции `HilSpect` должен быть максимальный порядок исследуемых матриц, выходные аргументы в данном случае не нужны. При возникновении затруднений воспользуйтесь листингом 7.5.

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.

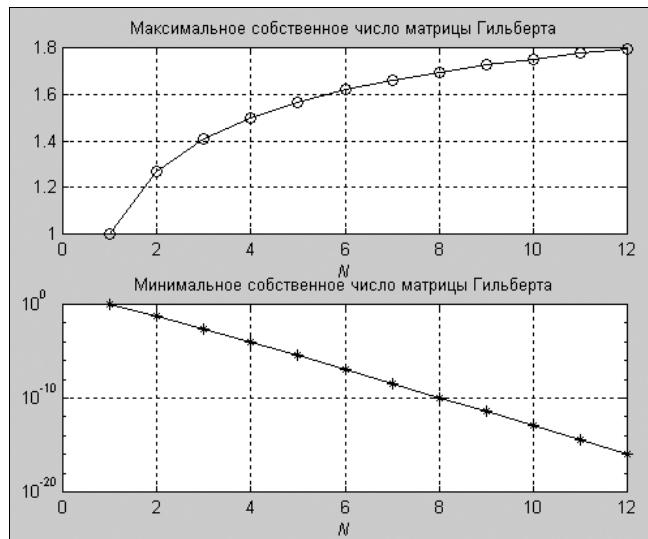


Рис. 7.2. Границы спектра матрицы Гильберта

### Примечание

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 программы пользователю посвящена глава 23.

**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
```

В алгоритм заполнения последовательности матриц можно внести усовершенствование, поскольку при увеличении  $n$  на единицу новая матрица образуется из старой добавлением строки и столбца. Воспользуйтесь симметричностью матрицы Гильберта и тем, что MATLAB динамически увеличивает размеры массива по мере необходимости. При возникновении затруднений обратитесь к листингу 7.6, в котором приведен подходящий блок операторов.

#### Листинг 7.6. Вычисления матриц Гильберта добавлением строки и столбца

```
A = 1;
for n = 2:N
 for k = 1:n - 1
 A(n, k) = 1/(n + k - 1);
 A(k, n) = A(n, k);
 end
 A(n, n) = 1/(2*n - 1);
end
```

В заключение этого раздела отметим еще одну особенность цикла `for`, которая наряду с возможностью задания вещественного счетчика цикла с постоянным шагом делает цикл `for` достаточно универсальным. В качестве значений переменной цикла допускается использование массива значений:

```
for count = A
 команды MATLAB
end
```

Если  $A$  — вектор-строка, то `count` последовательно принимает значение ее элементов при каждом заходе в цикл. В случае двумерного массива  $A$  на  $i$ -ом шаге цикла `count` содержит столбец  $A(:, i)$ . Разумеется, если  $A$  является вектор-столбцом, то цикл выполнится всего один раз со значением `count`, равным  $A$ .

Цикл `for` оказывается полезным при выполнении определенного конечного числа действий. Существуют алгоритмы с заранее неизвестным количеством повторений, реализовать которые позволяет более гибкий цикл `while`.

## Цикл `while`, суммирование рядов

Цикл `while` служит для организации повторений однотипных действий в случае, когда число повторений заранее неизвестно и определяется выполнением некоторого условия. Рассмотрим пример на вычисление суммы, по-

хожий 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 с заданной точностью. Известно, что для знакопеременного ряда теоретически достаточно удерживать слагаемые, превышающие по модулю заданную точность, например,  $10^{-10}$ . Однако суммирование таких рядов с ограниченной точностью вычислений может привести к потере значащих цифр и, в конечном итоге, к неверному результату. Значение  $k$ , обеспечивающее малость текущего слагаемого, заранее неизвестно, поэтому циклом `for` воспользоваться не удастся. Выход состоит в применении цикла `while`, который работает, пока выполняется *условие цикла*:

```
while условие повторения цикла
 команды MATLAB
end
```

В данном примере условием повторения цикла является то, что модуль текущего слагаемого  $x^{2k+1}/(2k+1)!$  больше  $10^{-10}$ . Для записи условия в форме, понятной MATLAB, следует использовать знак " $>$ " (больше). Текст файл-функции `mysin`, вычисляющей сумму ряда на основе рекуррентного соотношения

$$a_k = \frac{x^2}{2k(2k+1)} a_{k-1},$$

приведен в листинге 7.7.

### Примечание

Конечно, в общем случае малость слагаемого — понятие относительное, слагаемое может быть, скажем, порядка  $10^{-10}$ , но и сама сумма того же порядка. В этом случае условие окончания суммирования должно быть другим, а именно малость модуля отношения текущего слагаемого к уже накопленной части суммы. Пока не будем обращать на это внимания — нашей задачей является изучение работы с циклами.

#### Листинг 7.7. Файл-функция `mysin`, вычисляющая синус разложением в ряд

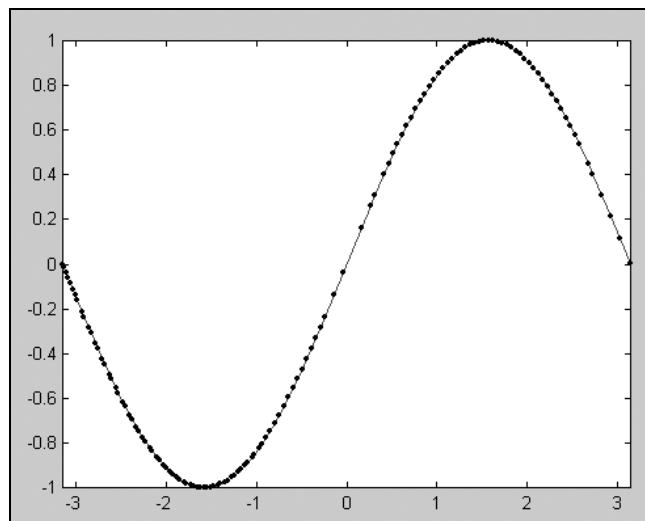
```
function s = mysin(x)
% Вычисление синуса разложением в ряд
% Использование: y = mysin(x), -pi < x < pi
```

```
% вычисление первого слагаемого суммы для k = 0
k = 0;
u = x;
s = u;
% вычисление вспомогательной переменной
x2 = x*x;
while abs(u) > 1.0e-10
 k = k + 1;
 u = -u* x2 / (2*k) / (2*k + 1);
 s = s + u;
end
```

Обратите внимание, что у цикла `while`, в отличие от `for`, нет переменной цикла, поэтому пришлось до начала цикла `k` присвоить единицу, 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.')
```



**Рис. 7.3.** Сравнение `mysin` и `sin`

Получающиеся графики изображены на рис. 7.3, они свидетельствуют о правильной работе файл-функции `mysin`.

Условие цикла `while` может содержать логическое выражение, составленное из операций отношения и логических операций или операторов. Для задания условия повторения цикла допустимы *операции отношения*, приведенные в табл. 7.1.

**Таблица 7.1. Операции отношения**

| Обозначение        | Операция отношения |
|--------------------|--------------------|
| <code>==</code>    | Равенство          |
| <code>&lt;</code>  | Меньше             |
| <code>&lt;=</code> | Меньше или равно   |
| <code>&gt;=</code> | Больше или равно   |
| <code>~=</code>    | Не равно           |

Задание более сложных условий производится с применением логических *операторов* или *операций*. Например, условие  $-1 \leq x < 2$  состоит в одновременном выполнении неравенства  $x \geq -1$  и  $x < 2$  и записывается при помощи логического оператора `and`

`and (x >= -1, x < 2)`

или эквивалентным образом с применением логической операции "и" — `&`

`(x >= -1) & (x < 2)`

Основные логические операции и операторы и примеры их записи приведены в табл. 7.2 (логические выражения подробно описаны в разд. "Логические операции с числами и массивами" этой главы).

**Таблица 7.2. Логические выражения**

| Тип выражения    | Выражение            | Логический оператор                 | Логическая операция                    |
|------------------|----------------------|-------------------------------------|----------------------------------------|
| Логическое "и"   | $x < 3$ и<br>$k = 4$ | <code>and (x &lt; 3, k == 4)</code> | <code>(x &lt; 3) &amp; (k == 4)</code> |
| Логическое "или" | $x = 1$ или $2$      | <code>or (x == 1, x == 2)</code>    | <code>(x == 1)   (x == 2)</code>       |
| Отрицание "не"   | $a \neq 1.9$         | <code>not (a == 1.9)</code>         | <code>~(a == 1.9)</code>               |

### Примечание

Операторы `not`, `and` и `or` являются функциями, возвращающими значения "истина" (логическая единица) или "ложь" (логический ноль). Такие же значения принимает любое логическое выражение.

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 количество слагаемых:

```
(abs(u) > 1.0e-10) & (k <= 100000)
```

или в эквивалентной форме:

```
and(abs(u) > 1.0e-10, k <= 100000)
```

### Примечание

Для задания порядка выполнения логических операций следует использовать круглые скобки (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).

При программировании алгоритмов кроме организации повторяющихся действий в виде циклов часто требуется выполнить тот или иной блок команд в зависимости от некоторых условий, т. е. использовать *ветвление* алгоритма.

## Операторы ветвления

Условный оператор `if` и оператор переключения `switch` позволяют создать гибкий разветвляющийся алгоритм, в котором при выполнении определенных условий выполняется соответствующий блок операторов или команд MATLAB. Практически во всех языках программирования имеются аналогичные операторы.

### 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 — создайте файл-функцию для вычисления

$$\sqrt{x^2 - 1}.$$

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 сообщение о том, что результат является комплексным числом. Перед вычислением функции следует произвести проверку значения аргумента  $x$  и вывести в командное окно предупреждение, если модуль  $x$  меньше 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`. Условие является логическим выражением и записывается по правилам, описанным в предыдущем разделе.

Файл-функция, проверяющая значение аргумента, приведена в листинге 7.8. В нем мы использовали функцию `warning`, которая служит для вывода предупреждения в командное окно.

### Листинг 7.8. 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 + 0.97979589711327i
```

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.

### Примечание

В отличие от warning, выполнение error приводит к останову программы.

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 7.9.

#### Листинг 7.9. 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, будет рассмотрено далее (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, соответственно, число входных и выходных аргументов, с которыми была вызвана файл-функция. Для проверки, являются ли входные аргументы числами или массивами, 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 7.10 содержит правильно запрограммированную файл-функцию, при использовании которой не должно возникнуть сложностей.

#### Листинг 7.10. 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. При наличии определенного опыта программирования и желания писать собственные вычислительные программы большую пользу может принести самостоятельное изучение алгоритмов файл-функций, рас-

положенных 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 поясняет пример файл-функции ifdem, приведенной в листинге 7.11.

### Листинг 7.11. Файл-функция ifdem, демонстрирующая работу if-elseif-else

```
function ifdem(a)
% пример использования структуры if-elseif-else
if (a == 0)
 disp('а равно нулю')
elseif a == 1
 disp('а равно единице')
elseif a == 2
 disp('а равно двум')
elseif a >= 3
 disp('а больше или равно трем')
else
 disp('а 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)
а равно единице
>> ifdem(1.2)
а меньше трех, но не ноль, не единица и не двойка
>> ifdem(2)
```

```

а равно двум
>> ifdem(3)
а больше или равно трем
>> ifdem(-1)
а 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('а равно нулю')
else
 disp('а не равно нулю')
end

```

В случае двух ветвей используется завершающее else, а elseif пропускается. Оператор if должен заканчиваться end.

Файл-функция ifdem наглядно демонстрирует работу оператора if, но на практике оказывается бесполезной. Действительно полезный пример — вычисление кусочно-заданной функции

$$y(x) = \begin{cases} \sin x - 1, & x < -\pi; \\ x/\pi, & -\pi \leq x < \pi; \\ -\cos x, & x \geq \pi, \end{cases}$$

которое реализуется файл-функцией pwfun. Ее текст приведен в листинге 7.12. Обратите внимание на следующие моменты.

1. Число ветвей if-elseif-else равно трем.
2. Во второй ветви достаточно только проверить, что  $x < \pi$ , условие  $x \geq -\pi$  уже выполнено (иначе бы отработала первая ветвь в if-elseif-else и оператор if закончил работу).
3. В последней ветви нет смысла проверять никакие условия, она работает, если все предыдущие условия неверны, что как раз соответствует  $x \geq \pi$ .

#### Листинг 7.12. Файл-функция pwfun

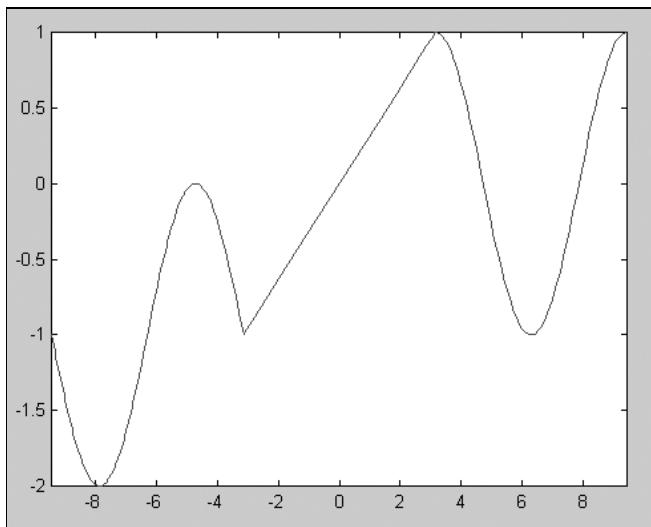
```

function y = pwfun(x)
% вычисляет кусочно-линейную функцию
% sin(x) - 1, если x < -pi
% y(x)= x/pi, если -pi <= x < pi
% -cos(x), если x >= pi

```

```
% 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 графику, изображенному на рис. 7.4.



**Рис. 7.4.** График кусочно-заданной функции

Мы не случайно выбрали `fplot` для построения графика кусочно-заданной функции. Функцией `plot` воспользоваться не удастся, т. к. требуется предварительно вычислить вектор значений функции от вектора аргументов, а `pwfun` не умеет работать с *входным аргументом-вектором*. Убедиться в этом можно, построив график `pwfun` командами:

```
>> x = -3*pi:pi/100:3*pi;
>> y = pwfun(x);
>> plot(x, y)
```

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 ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $\sim$ ) могут специальным образом применяться и к векторам, о чём подробно написано ниже, а при обычном применении в данном примере не дают ожидаемого результата. Выход состоит в программировании 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 файла-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 7.13.

#### Листинг 7.13. 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/pi, если -pi <= x < pi
% -cos(x), если x >= pi
% 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 выводим 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
% у 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 pwfun1 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 pwfun1 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 7.14.

#### Листинг 7.14. Файл-функция pwfun2 (входной аргумент — двумерный массив)

```
function y = pwfun2(x)
% вычисляет кусочно-линейную функцию
% sin(x) - 1, если x < -pi
% y(x)= x/pi, если -pi <= x < pi
% -cos(x), если x >= pi
% использование y = pwfun(x), x — число, вектор или матрица;
```

```

% 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 у 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

```

### Примечание

В файл-функции pwfun2 можно было бы обойтись одним циклом, если учесть схему расположения двумерных массивов в памяти и применить индексацию порядковым номером (см. главу 2). В этом случае переменная  $i$  единственного цикла должна меняться от 1 до  $Nx \cdot Mx$ , а вместо  $x(i, j)$  и  $y(i, j)$  следует указать  $x(i)$  и  $y(i)$ .

## Оператор **switch**

Предположим, что при работе с функцией двух аргументов

$$e^{-|x-y|} \sin \pi x \cdot \cos \pi x^2$$

приходится визуализировать ее четырьмя различными способами: каркасной поверхностью, сплошной поверхностью, выводить линии уровня или 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, 2, 3 и 4')
end
```

Каждая ветвь определяется оператором `case`, переход в нее выполняется тогда, когда переменная оператора `switch` (в данном примере `a`) принимает значение, указанное после `case`, или одно из значений списка `case`. После выполнения какой-либо из ветвей происходит выход из `switch`, при этом значения, заданные в других ветвях `case`, уже *не проверяются*. Если подходящих значений для `a` не нашлось, то выполняется ветвь оператора `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 7.15.

**Листинг 7.15. 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 – заливая поверхность
% vis = 3 – линии уровня
% vis = 4 – освещенная поверхность

% проверяем числа 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 % заливая 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`.

## Выход из файл-функции, оператор `return`

Файл-функция всегда прекращает свою работу после выполнения последнего оператора. Если алгоритм файл-функции предполагает ветвление, то это часто приводит к необходимости завершить функцию в каждой ветви. Применение условного оператора приводит к достаточно сложной структуре `if-elseif-else`, которая может быть упрощена за счет задействования оператора `return`. При достижении этого оператора выполнение файл-функции

прекращается и происходит возврат в точку вызова — командное окно, другую файл-функцию или файл-программу.

В качестве примера мы рассмотрим вычисление кусочно-заданной функции, которое было организовано с использованием конструкции `if-elseif-else` в файл-функции `pwfun` (см. листинг 7.12). Оператор `return` позволяет за-программировать файл-функцию без ветвления так, как показано в лис-тинге 7.16.

**Листинг 7.16. Файл-функция `pwfun3` с досрочным завершением работы по `return`**

```
function y = pwfun3(x)
% вычисляет кусочно-линейную функцию
% sin(x) - 1, если x < -pi
% y(x) = x/pi, если -pi <= x < pi
% -cos(x), если x >= pi
% использование y = pwfun3(x), x - число;

if x < -pi
 y = sin(x) - 1;
 return
end
% здесь x >= -pi
if x < pi
 y = x/pi;
 return
end
% здесь x >= pi
y = -cos(x);
```

## Прерывание и продолжение циклов

Циклическое выполнение блока операторов в ряде случаев может потребо-ваться либо прерывания цикла, т. е. досрочного выхода из него, либо перехо-да к следующему шагу цикла без выполнения части оставшихся операторов.

При организации циклических вычислений операторами `while` или `for` следует позаботиться о том, чтобы внутри цикла не возникало ошибок или нежелательных действий. Например, пусть дан вектор `x`, состоящий из целых чисел, и требуется сформировать новый вектор `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 вектора равен нулю, то при делении получится `Inf` и последующие вычисления могут оказаться бесполезны. Предотвратить подобную ситуацию можно досрочным выходом из цикла, если текущее значение  $x(i)$  равно нулю. Следующий фрагмент программы демонстрирует использование оператора `break` для прерывания цикла (листинг 7.17).

### Листинг 7.17. Прерывание цикла оператором `break`

```

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.

#### Примечание

Грамотным условием проверки вещественного числа на ноль является условие малости его абсолютной величины, поскольку "величина" машинного нуля (т. е. интервал вещественных чисел эквивалентных машинному нулю) существенно больше минимального расстояния между компьютерными числами. Функция `realmin` дает минимальное вещественное положительное число, которое представляется в компьютере с предусмотренным количеством значащих цифр.

Все числа, меньшие по абсолютной величине этого значения, представляются машинным нулем. Функция `realmax` возвращает максимальное по модулю вещественное число, превышение которого приводит к значению `Inf`.

Внутри цикла может возникнуть ситуация, при которой выполнение оставшейся части операторов не требуется, а сразу следует перейти к новому шагу цикла. В этом случае оказывается удобным оператор `continue`, который хоть и не является незаменимым, но позволяет обойтись без разветвленной структуры `if-elseif-else`, что делает текст программы более понятным.

## Обработка исключительных ситуаций

Хорошо написанная программа предотвращает ошибочные действия, которые приводят к досрочному ее завершению. К немедленному прерыванию работы файл-функции приводит, например, неверный ее вызов пользователем. В приведенных выше примерах (см. листинги 7.10, 7.13—7.15) мы контролировали возможные ошибки в интерфейсе и прекращали работу файл-функции, выводя информацию о допустимом ее использовании.

Следующий источник ошибок — сам алгоритм. Часть некорректных математических операций в MATLAB, в отличие от многих языков программирования, не приводит к останову выполнения программы. При делении на ноль получается бесконечность (`Inf`), деление нуля на ноль приводит к `Nan`, сумма бесконечности и числа имеет результатом бесконечность. Однако попытка считывания информации из несуществующего файла при помощи `load`, обращение к несуществующей переменной или функции, несовпадение размеров массивов в операторе присваивания обязательно прервут работу программы.

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 7.18.

### Листинг 7.18. Фрагмент программы обработки данных из файла

```
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 7.18 работает успешно (схема поиска файлов, которую применяет MATLAB, описана в разд. "Установка путей" главы 5; работа с текстовыми файлами подробнее разбирается в главе 8).

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 7.18, лучше оформить с использованием `try...catch` так, как это сделано в листинге 7.19.

### Листинг 7.19. 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
```

В качестве аргумента функции `disp` разумно использовать функцию `lasterr`, возвращающую системное сообщение об ошибке. Замените в листинге 7.19 строку `disp('не могу найти файл my.dat')` на строку `disp(lasterr)` и выполните файл-программу — в командное окно выводится диагностическое сообщение о возникшей ошибке:

```
Error using ==> load
Unable to read file my.dat: file does not exist.
```

## Логические выражения с массивами и числами

MATLAB является средой, ориентированной на вычисления с матричными данными. 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. Универсальным средством обработки матричных данных служат логические операции. Они позволяют просто и наглядно записывать алгоритмы, реализация которых в других языках достаточно громоздка. 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, как любого логического выражения, 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 = 1;
>> b = 5;
>> c = 3;
>> a = a + (b > c)
a =
 2
```

Условие  $b > c$  выполняется, т. е. является "истиной". Результат операции отношения "больше" равен единице, которая прибавляется к  $a$ . Использование операции "меньше" привело бы к логическому нулю, сложение которого с переменной  $a$  не изменило бы ее значения.

Результат выполнения логического выражения может быть присвоен переменной:

```
>> f = a <= b
f =
 1
```

Посмотрите тип `f` в окне **Workspace** браузера рабочей среды — переменная `f` является логической, точнее логическим массивом размера 1 на 1.

Операции отношения применимы к массивам одинакового размера. Происходит *поэлементное* сравнение и результатом является логический 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
```

При сравнении двумерных массивов больших размеров удобно использовать команду `spru` для визуализации результата, которая приводит к появлению графического окна с шаблоном матрицы. Элементы результирующей матрицы отображаются точками, что наглядно показывает, для каких элементов выполняется проверяемое условие (построение шаблона матрицы командой `spru` описано в разд. "Визуализация матриц" главы 2).

## Логические операции с числами и массивами

Конструирование условий операторов `if` и `while` осуществляется с помощью логических операторов `and`, `or` и `not` или в эквивалентном виде с использованием логических операций `&`, `|` и `~`. Вычисление логических выражений, содержащих `and` (`&`), `or` (`|`) и `not` (`~`), с массивами 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 массивов 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.

### Примечание

Операндами логических выражений могут быть как логические массивы, так и числовые. Одним из operandов логических выражений может быть число или логический ноль или единица. В этом случае происходит поэлементное выполнение логической операции для каждого элемента массива и числа. Выполнение операторов **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
 1
```

Кроме операторов **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.

Логические операции **&** и **|** учитывают оба операнда для вычисления результата. В то же время значение логического выражения в ряде случаев определяется значением только первого операнда. Если первый operand логического "или" является "истиной", то результат всегда будет "истина". Если первый operand логического "и" — "ложь", то результат — "ложь". Операции **&&** и **||** отличаются от **&** и **|** тем, что в перечисленных двух ситуациях

они не проверяют значение второго операнда (операция `&&` была использована нами в главе 6 при управлении процессом решения дифференциального уравнения для простой реализации алгоритма файл-функции `solproc` (см. листинг 6.19)).

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.

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.

При обработке матричных данных так же оказывается полезной достаточно универсальная функция `find` и логическое индексирование (применение `find` и логического индексирования рассмотрено в разд. "Логическое индексирование" главы 2).

В приведенных выше примерах программ условия операторов `if-elseif-else` и `while` содержали логические выражения со скалярными значениями — либо "истина", либо "ложь". Однако в эти конструкции языка программирования MATLAB могут входить выражения с массивами, причем как с числовыми, так и логическими. Формальное правило таково: условие считается выполненным, если результатом выражения является массив, состоящий только из логических единиц или ненулевых элементов. Пустой массив и "ложь" эквивалентны.

## Приоритет логических и арифметических 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 возвведение в степень, затем сложение и, наконец, сравнение значения суммы `A + B.^2` и `C`.

Приоритет операций отражен в следующем списке (в порядке выполнения).

1. Логические операторы: `and`, `or`, `not`, `xor` (поскольку они являются функциями).
2. Отрицание `~`.
3. Транспонирование, возвведение в степень (в том числе поэлементное), знак плюс или минус перед числом.
4. Умножение и деление (в том числе поэлементное).
5. Сложение и вычитание.
6. Операции отношения: `>`, `>=`, `<`, `<=`, `==`.
7. Логическое "и" `&`.
8. Логическое "или" `|`.
9. Логическое "и" `&&`.
10. Логическое "или" `||`.

### Примечание

Сначала выполняются операции над аргументами функций `and`, `or`, `not` и `xor`, например, два выражения: `and(A, B) + C` и `A&B+C` не эквивалентны! В более ранних версиях MATLAB операции `&` и `|` имели одинаковый приоритет, что следует учитывать при переносе приложений, разработанных в старых версиях MATLAB, в новую версию пакета.

Для изменения порядка операций следует применять круглые скобки. Они также полезны для придания достаточно длинному выражению легкочитаемого вида.

## Задания для самостоятельной работы

Напишите файл-функцию для решения поставленной задачи. Там, где это возможно, предложите два решения: с использованием конструкций языка программирования и без них, применяя функцию `find` и др.

1. Вычислить произведение элементов вектора, не превосходящих среднее арифметическое значение его элементов.
2. Подсчитать число нулей и единиц в заданной матрице.
3. Определить количество положительных элементов вектора, расположенных между его максимальным и минимальным элементами.
4. Просуммировать отрицательные элементы матрицы, лежащие ниже главной диагонали.
5. Заменить положительные элементы вектора суммой всех его отрицательных элементов.
6. Заполнить квадратную матрицу  $A$ , каждый элемент которой  $a_{ij}$  определяется следующим образом:

$$a_{ij} = \begin{cases} i - j, & i > j; \\ i + j, & i = j; \\ i^2 + j^2, & i < j. \end{cases}$$

7. Вычислить сумму:

$$s(x) = \sum_{i=1}^n \sum_{j=1}^m \frac{x^{i+j}}{(i+j)^2}.$$

8. Для матрицы  $A = (a_{ij})$  размера  $n$  на  $m$  найти значение выражения:

$$w = \sum_{i=1}^n \prod_{j=1}^m a_{ij}.$$

9. По заданному  $x$  найти максимальное значение  $n$ , для которого следующая сумма не превосходит 100:

$$s(x) = \sum_{k=1}^n kx^k.$$

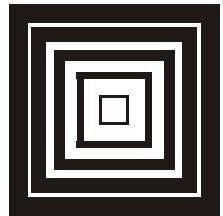
10. Вычислить сумму:

$$s(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

с заданной точностью  $\epsilon$ . Суммировать следует, пока модуль отношения текущего слагаемого к уже накопленной части суммы превосходит  $\epsilon$ .

Сравнить результат с точным значением, построив графики  $e^x$  и  $s(x)$  для  $x \in [0, 5]$ .

11. Заданы окружности, координаты их центров содержатся в массивах  $x$  и  $y$ , а радиусы в массиве  $r$ . Известны координаты некоторой точки. Требуется вывести график, на котором маркером отмечено положение точки, синим цветом изображены те окружности, внутри которых лежит точка, а остальные окружности нарисованы красным цветом.



## Глава 8

# Обработка данных и приемы программирования в MATLAB

Простейшие способы организации последовательности выполнения команд и операторов MATLAB изложены в главе 7. Данная глава посвящена описанию возможностей MATLAB, предназначенных для программирования более сложных алгоритмов. Разобрано использование массивов-структур и массивов-ячеек для хранения и обработки данных, операции со строками, работа с текстовыми файлами, создание файл-функций с переменным числом входных и выходных аргументов, рекурсивных файл-функций, поиск ошибок средствами встроенного отладчика и некоторые другие вопросы, тесно связанные с эффективным программированием собственных алгоритмов.

## Работа со строками

Операции со строками являются важным элементом программирования в MATLAB. Вы уже использовали строки при чтении предыдущих глав. Например, входной аргумент 'ro-' функции `plot` или входной аргумент '`TolX`' функции `optimset` являются примерами строк. 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, World!`' (набирать следует в апострофах).

Не завершайте оператор присваивания точкой с запятой для того, чтобы сразу увидеть значение str. При наборе строки обратите внимание, что не завершенная апострофом строка отображается синим цветом.

```
>> str = 'Hello, World!'
```

```
str =
```

```
Hello, World!
```

Посмотрите информацию о переменной str в окне **Workspace** (или в командном окне при помощи whos)

| Name | Size | Bytes | Class      |
|------|------|-------|------------|
| str  | 1x13 | 26    | char array |

Переменная str является одномерным массивом символов (char array) из тринадцати элементов, занимающим в памяти 26 байт, 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 осуществляется `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 производится поиск, — в аргументе большего размера, поэтому необязательно заботиться о порядке входных аргументов.

### Примечание

Символы кирилицы могут некорректно отображаться в некоторых приложениях Windows, в частности, в нелокализованной версии MATLAB. Разрешение проблемы состоит в следовании рекомендациям, которые дают эффект для других программ, например, 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". См. также замечание в разд. "Оформление графиков" главы 3.

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, "6.5" на "7.0" в предложении "MATLAB 6.5 является последней версией пакета". Используйте `strrep` так, как указано ниже:

```
>> str = 'MATLAB 6.5 является последней версией пакета';
>> newstr = strrep(str, '6.5', '7.0')
newstr =
MATLAB 7.0 является последней версией пакета
```

Преобразование всех прописных букв в строчные производит функция `upper`. Функция `lower` осуществляет обратное преобразование. Стока, 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 созданию простых программ с интерфейсом пользователя из командной строки рабочей среды MATLAB (полный список сервисных функций обработки строк приведен в *приложении 1*).

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.

#### 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 nargin > 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;
% создаем пустой массив для хранения позиций цифр в строке
pos = [];
% 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, увеличивая его длину
 pos(digits) = k;
 end
end

```

Далее в этой главе в разд. "Диагностическая отладка программ" функция strnumpos реализована в более компактной форме.

## 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
```

Использование строк разной длины недопустимо:

```
>> surnames = ['Иванов'; 'Васильев'; 'Петров']
```

```
??? Error using ==> vertcat
```

```
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 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
```

К первой и последней строке добавились справа два пробела, 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 помещать строки в начало или конец существующих массивов строк, например:

```
>> surnames = char(surnames, 'Сидоров')
```

```
surnames =
```

```
Иванов Сидоров
```

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('Ma', 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. Подготовка и корректировка текстового файла могут быть осуществлены средствами любого текстового редактора, в том числе редактора M-файлов MATLAB.

## Текстовые файлы

Обработка информации в файле включает в себя: чтение данных, их изменение или использование и, наконец, сохранение полученных результатов. Программирование обработки информации состоит из следующих этапов: открытие файла, считывание данных, запись информации, закрытие файла. Файлы можно использовать либо только для чтения (входные файлы с исходными данными) или только для записи (выходные файлы с результатами обработки), либо для изменения (входной и выходной файл одновременно). Следующие разделы посвящены описанию команд MATLAB, реализующих вышеперечисленные действия, и демонстрации их использования на некоторых простых примерах.

### Примечание

Средства работы с файлами в MATLAB имеют много общего со средствами среды программирования языка C. Существенным отличием является ввод данных из файла, ориентированный на работу с массивами.

## 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:

- '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;
- 'at' — открываемый текстовый файл предназначен только для добавления данных в конец файла (если файла не существует, то он создается);
- 'at+' — открываемый текстовый файл предназначен для добавления данных в конец файла и чтения данных (если файл не существует, то он создается).

### Примечание

Символ `t` указывает на то, что файл текстовый. Вышеперечисленные способы доступа возможны и для двоичных файлов. Например: '`w+`' означает создание пустого двоичного файла для чтения и записи.

Выходным аргументом `fopen` является *идентификатор* (*ссылочный или логический номер*), присвоенный файлу. Если файл открыть не удалось, то идентификатор становится равным минус единице. Ошибки часто возникают из-за того, что MATLAB не может найти требуемый для чтения файл. Всегда лучше указывать полное имя файла, при задании только имени и расширения MATLAB производит поиск в текущем каталоге и путях поиска. Например, если в вашем текущем каталоге нет файла `beep.m`, то `fopen('beep.m', 'rt')` найдет его в подкаталоге `\toolbox\matlab\general\` основного каталога MATLAB. Функция `fopen` может быть вызвана и со вторым дополнительным выходным аргументом — строковой переменной с сообщением о результате открытия.

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 по одной от начала до конца файла. Контроль за достижением конца файла осущест-

вляется функцией `feof` с входным аргументом — идентификатором файла, `f.eof` возвращает логическую единицу, если в файле нет больше строк, и логический ноль в противном случае. По окончании работы необходимо закрыть файл командой `fclose`, указав в качестве входного аргумента идентификатор файла.

Файл-функция `myview` (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 М-файл с файл-функцией `myview` хранится в текущем каталоге MATLAB, то `myview('myview.m')` выводит листинг самой файл-функции (текст, набранный кириллицей, может выводиться некорректно).

### 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 nargin ~= 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 должен быть 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 открытся, 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 строку в текстовом файле и возвращает вектор с номерами строк файла, содержащих ее. Файл-функция должна иметь два входных аргумента — имя файла и искомую строку 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
 rows = []; % создание пустого массива для хранения номеров строк
 % 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 `fprintf`, 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, '\nЭтот 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);
```

**Примечание**

Идентификатор файла может быть опущен, в этом случае производится вывод в командное окно.

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= sin(x) |
```

Поставленная задача решается при помощи четырех вызовов `fprintf` (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 sintable(filename)

% Добавьте проверку входных и выходных параметров

% Открытие нового файла для записи
[F, mes] = fopen(filename, '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= sin(x) |\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, 'список форматов', список переменных)
```

Здесь первый аргумент, как и в случае вывода строк, является идентификатором файла, второй — строка с кодами форматов, которые определяют вид записи значений переменных из списка, заданного третьим аргументом. В списке может быть одна или несколько переменных, в том числе и массивов.

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 восемь 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);
myview('twonum.txt');
```

Обратите внимание на второй аргумент команды `fprintf`. Код `%7.4f` задает формат вывода переменной  $x$ , которая расположена на первом месте в спи-

ске вывода. Знак процента указывает на начало формата, цифра 7 обозначает, что всего под значение переменной *x* отводится семь позиций (включая десятичную точку и место под знак, которое резервируется и для положительных чисел), 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 у.

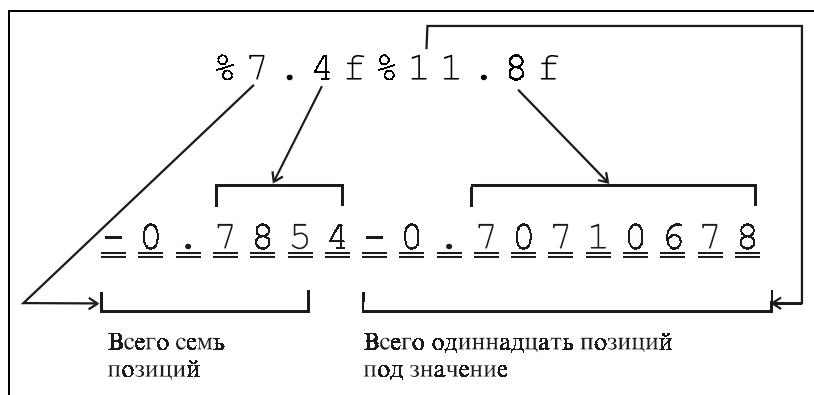
## Примечание

При использовании форматного вывода следует учесть, что если количество элементов в списке вывода больше числа кодов формата, то форматные коды применяются повторно, начиная с первого.

После выполнения файл-программы из листинга 8.6 содержимое файла `twonum.txt` составляют два числа — значения переменных `x` и `y` с требуемым числом цифр, выводимых на экран ранее написанной функцией `myview`:

-0.7854-0.70710678

Соответствие форматов и получаемого результата приведено на схеме, изображенной на рис. 8.1, каждая позиция подчеркнута.



**Рис. 8.1.** Схема соответствия форматов вывода и результата

Значения  $x$  и  $y$  вывелись одно за другим без разделителя, что затрудняет чтение результата. Можно было бы указать в списке форматов больше позиций под вывод второго числа, например, '`%7.4f%15.8f`'. Поскольку у меньше единицы и выравнивание числа в выделенном для него поле по умолчанию производится по правому краю, то числа в данном случае будут разделены пробелами. Но если в листинге 8.6 заменить `y = sin(x)` на

`y = 1000000*sin(x)`, то числа снова не будут разделены. Кроме того, пятнадцати позиций не хватило под вывод `y` с восьмью знаками после десятичной точки, и MATLAB автоматически увеличил поле вывода. Поэтому при задании форматов следует предусмотреть некоторые разделители между числами.

Форматы данных можно разделять любым текстом, включая пробелы, который запишется в текстовый файл между соответствующими значениями. 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
```

Помещение символа `\n` в список форматов 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:

```
a = +0.560 b = 1.1 c = 0001.22
```

Полезной особенностью `fprintf` является то, что список ввода может быть матрицей. Матрица выводится *по столбцам* — с последовательным применением форматов из списка.

## 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  $R$  размера три на 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. `fprintf` работает с матрицей по столбцам. Матрица, отображенная в командном окне, совпадает с матрицей, записанной в файл `randmatr.txt`.

Информации о форматах, приведенной выше, вполне достаточно для завершения работы над файл-функцией `sintable`, предназначеннной для вывода таблицы значений 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 функцией `fclose`). Операторы, осуществляющие запись в файл названия и шапки таблицы, 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);
```

Вставьте вышеприведенный блок операторов в вашу файл-функцию `sintable` и вызовите ее, указав в качестве входного аргумента строку с именем файла `table.dat`. В результате создается файл `table.dat`, содержимое которого приведено ниже:

ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ  $\sin(x)$

|       |  |       |        |           |       |
|-------|--|-------|--------|-----------|-------|
| ----- |  | x     |        | y= sin(x) | ----- |
| ----- |  | 0.000 | 0.0000 |           | ----- |

```
1.571	1.0000
3.142	0.0000
4.712	-1.0000
6.283	-0.0000
```

## Простые структуры

Структура представляет собой элемент данных, содержащий разнотипные поля, например, числа, массивы и строки. Вам уже приходилось иметь дело со структурами при чтении главы 6, в которой была описана генерация структуры `options` специальными функциями `optimset` и `odeset` с целью управления процессом вычислений. Решение уравнений при помощи `fzero` потребовало задания значений полей `TolX`, `TolFun`, `Display`, причем сами значения были как числами, так и строками. Научимся теперь создавать собственные структуры и оперировать данными, хранящимися в их полях.

Создание простой структуры осуществляется командой `struct`, имеющей формат:

```
имя_структурь = struct(имя_поля1, значение1, имя_поля2, значение2, ...)
```

где именами полей являются строки или строковые переменные, а значениями — данные любых типов, включая числовые массивы, строки, массивы строк и структуры. Следующая команда создает структуру `transaction`, содержащую параметры сделки по ценным бумагам (можно набирать в одну строку без символа переноса строки ..., используемого в MATLAB):

```
>> transaction = struct('time', [10,20,46], 'stock', 'EESR', 'volume',...
 10000, 'price', 0.23, 'currency', 'USD', 'bid', 0.21, 'ask', 0.27)
```

Результат сразу же вывелся в командное окно в форме таблицы, каждая строка которой содержит название поля и его значение:

```
transaction =
 time: [10 20 46]
 stock: 'EESR'
 volume: 10000
 price: 0.2300
 currency: 'USD'
 bid: 0.2100
 ask: 0.2700
```

Поля структуры `transaction` имеют следующий смысл: `time` — время совершения сделки, `stock` — код ценной бумаги, `volume` — объем сделки, `price` — цена исполнения сделки, `currency` — код валюты расчетов по сделке, `bid` — цена спроса на момент совершения сделки, `ask` — цена предложения на мо-

мент совершения сделки. В рассматриваемой структуре значениями полей являются числа, строки и массив чисел. Значения полей можно посмотреть и в окне **Workspace**.

### Примечание

До главы 20, где будут детально рассмотрены способы представления даты и времени и функции для работы с ними, мы будем записывать эти данные в собственном формате. В данном случае время задается массивом из трех чисел: часы (от 0 до 24), минуты и секунды (от 0 до 60).

Обращение к значению поля обеспечивается идентификатором, состоящим из имени структуры и имени поля, разделенных точкой. Такое составное имя всегда интерпретируется как обращение к полю структуры.

Сумма сделки в рублях может быть вычислена с использованием содержащегося поля структуры и курса ЦБ РФ на дату совершения сделки `exchange`:

```
>> exchange = 28.5214;
>> format bank
>> settlement = exchange * transaction.price * transaction.volume
settlement =
65599.22
```

Изменение формата вывода результата желательно для получения ответа в форме, удобной при проведении финансовых вычислений.

Полями структуры могут быть другие структуры. В структуре `transaction` имеются три однотипных поля — это цена исполнения, цена предложения и цена спроса. Все они связаны с валютой. Изменим структуру `transaction`, выделив эти поля в одну новую структуру `strike`:

```
>> transaction = struct('time', [10,20,46], 'stock', 'EESR', 'volume', ...
10000, 'strike', struct('currency', 'USD', 'price', 0.23, 'bid', 0.21, ...
'ask', 0.27))
transaction =
 time: [10 20 46]
 stock: 'EESR'
 volume: 10000
 strike: [1x1 struct]
```

Для доступа к полю второй структуры следует использовать составное имя, включающее имя основной структуры, имя вложенной структуры и имя поля, разделенные точкой. Цена сделки теперь вычисляется по формуле:

```
>> settlement = exchange*transaction.strike.price*transaction.volume
```

Выражения для задания вложенной структуры можно (и нужно для упрощения) разбить на части, результат получится тот же самый

```
>> A = struct('currency', 'USD', 'price', 0.23, 'bid', 0.21, ...
'ask', 0.27);
>> transaction = struct('time', [10, 20, 46], 'stock', 'EESR', ...
'velume', 10000, 'strike', A)
transaction =
 time: [10 20 46]
 stock: 'EESR'
 volume: 10000
 strike: [1x1 struct]
```

Обратите внимание, что при выводе структуры `transaction` в командное окно поля вложенной структуры `strike` не отображаются. Для того чтобы посмотреть значения полей вложенной структуры, следует обратиться именно к ней:

```
>> transaction.strike
ans =
 currency: 'USD'
 price: 0.2300
 bid: 0.2100
 ask: 0.2700
```

Существующая структура может быть изменена, если добавить или удалить поле. Добавление поля происходит при присвоении некоторого значения ранее не существовавшему полю.

### Примечание

Присвоение полю значения позволяет создавать структуры. Например, если в рабочей среде нет переменной `S`, или она есть, но не является структурой, то оператор присваивания `S.h = 17` приводит к созданию структуры `S` с полем `h`, в котором записано число 17.

Добавим поле `exchange` в структуру `A`.

```
>> A.exchange = 28.5214
A =
 currency: 'USD'
 price: 0.23
 bid: 0.21
 ask: 0.27
 exchange: 28.5214
```

При этом не происходит автоматического изменения структуры `transaction`:

```
>> transaction.strike.exchange
??? Reference to non-existent field 'exchange'.
```

Для внесения изменений в структуру `transaction` следует изменить поле `strike`:

```
>> transaction.strike = A;
>> transaction.strike.exchange
ans =
 28.5214
```

Удаление поля осуществляется с помощью функции `rmfield`. Удалите, к примеру, поле `time`

```
>> transaction = rmfield(transaction,'time')
```

Если в качестве выходного аргумента `rmfield` указать другое имя, то старая структура останется без изменений, а новая будет соответствовать старой с удаленным полем.

При обращении к полю структуры его имя можно задавать текстовой строкой в круглых скобках после точки:

```
>> transaction.('volume')
ans =
 10000
```

Это дает возможность использовать строковую переменную для идентификации поля:

```
>> field ='volume';
>> transaction.(field)
ans =
 10000
```

Возникает вопрос, как по имени структуры узнать названия ее полей и сохранить их в строковых переменных. Для этой цели служит встроенная функция `fieldnames`:

```
>> names = fieldnames(transaction)
names =
 'stock'
 'volume'
 'strike'
```

Посмотрите в окне **Workspace** браузера переменных рабочей среды информацию о переменной `names`. Она представляет собой не массив строк, а от-

носится к типу `cell array` (массив ячеек). Работу с таким типом данных мы рассмотрим далее в этой главе. Заметим сейчас, что для выделения содержимого ячейки из массива следует указать ее номер в фигурных скобках после имени массива ячеек. Например, обращение

```
>> f2 = names{2}
```

```
f2 =
```

```
volume
```

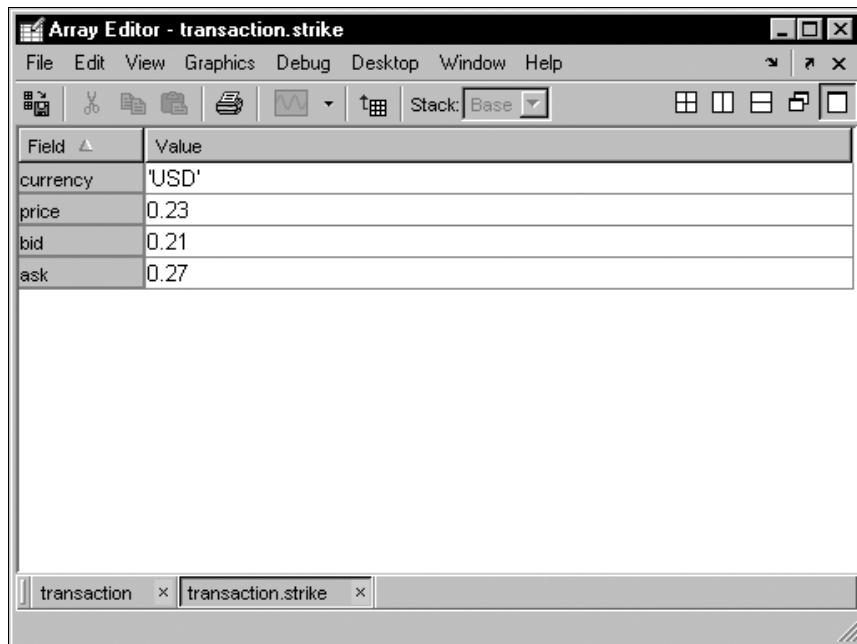
приводит к записи содержимого второго элемента массива ячеек `names` в строковую переменную `f2`, которую можно теперь использовать для доступа к нужному полю структуры `transaction`

```
>> v = transaction.(f2)
```

```
v =
```

```
10000
```

Просмотр и редактирование содержимого полей структуры `transaction` удобно производить в редакторе массивов (**Array Editor**), который открывается двойным щелчком кнопкой мыши по имени структуры в окне **Workspace**. При чтении главы 2 вы уже прибегали к редактору массивов для доступа к элементам числовых массивов. Рассмотрим теперь некоторые его возможности для работы со структурами.



**Рис. 8.2.** Просмотр вложенной структуры с помощью **Array Editor**

Редактор массивов позволяет обратиться к значениям полей структуры, в том числе и к полю со вложенной структурой `strike`, при помощи двойного щелчка мыши по названию поля в столбике **Field**. Содержимое поля открывается в новом окне редактора массивов, причем окно снабжено вкладкой с именем поля (рис. 8.2). Итак, двойной щелчок мышью приводит к отображению содержимого поля. Обычный щелчок по имени поля в столбике **Field** делает это поле текущим, а повторный щелчок позволяет изменить название поля. Ввод нового имени поля завершается нажатием <Enter> или щелчком мыши по другому полю. Кнопка **Up** на панели инструментов редактора массивов служит для перехода вверх по уровням вложенности, т. е. при активном окне с содержимым некоторого поля структуры `transaction`, например, поля `strike`, нажатие на **Up** приведет к отображению структуры `transaction`.

## Массивы структур и массивы ячеек

Обычные массивы удобны при работе с однородными данными — числами или строками. Информация может быть представлена в виде таблицы с полями, содержащими однотипные элементы, в этом случае наилучшим выбором является *массив структур*. Эффективное оперирование группой данных различных типов позволяет осуществить *массивы ячеек*, работу с которыми мы рассмотрим чуть позже.

## Массивы структур

Проиллюстрируем использование массивов структур на простом примере. Предположим, что успеваемость группы студентов по шести предметам представлена в табл. 8.1.

*Таблица 8.1. Успеваемость студентов группы 201*

| № | Фамилия  | Имя     | Год рож-дения | 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 естественно задействовать массив структур, каждый элемент которого является *структурой* с одинаковым набором *полей*, содержащих 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. 8.3.

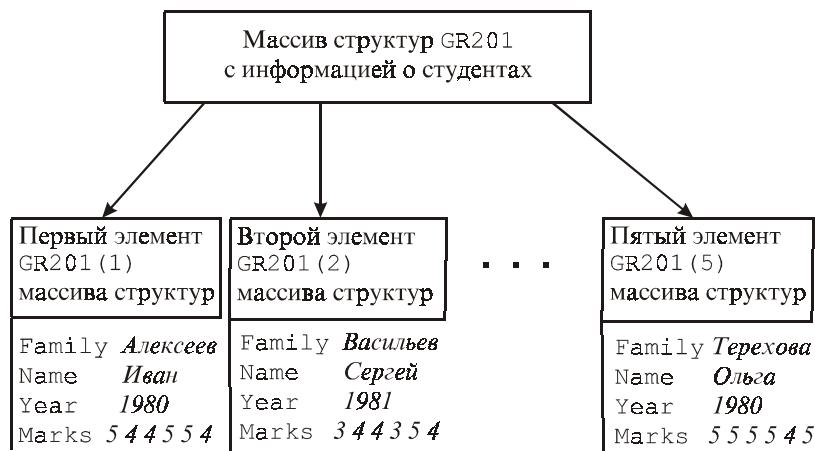


Рис. 8.3. Организация таблицы в виде массива структур

При работе с массивами структур необходимо придерживаться двух правил.

1. Доступ к структурам, входящим в массив, осуществляется при помощи индексации.
2. Поле отделяется от имени элемента массива структур при помощи точки.

Поскольку каждый элемент массива структур является структурой, то допускаются два способа создания и заполнения массива структур — операторами присваивания для всех полей каждой структуры массива или функцией `struct`, позволяющей занести значения сразу во все поля структуры. Операторы присваивания, последовательно заполняющие массив структур данными из табл. 8.1, приведены в листинге 8.8. Альтернативный вариант занесения информации в поля структур массива состоит в последовательном

применении функции `struct` для заполнения каждой структуры, входящей в массив, например, для первой структуры:

```
GR201(1) = struct('Family', 'Алексеев', 'Name', 'Иван', ...
 'Year', 1980, 'Marks', [5 4 4 5 5 4]);
```

#### 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 `GR201Fill` в соответствии с листингом 8.8. Посмотрите значение переменной `GR201` из командной строки:

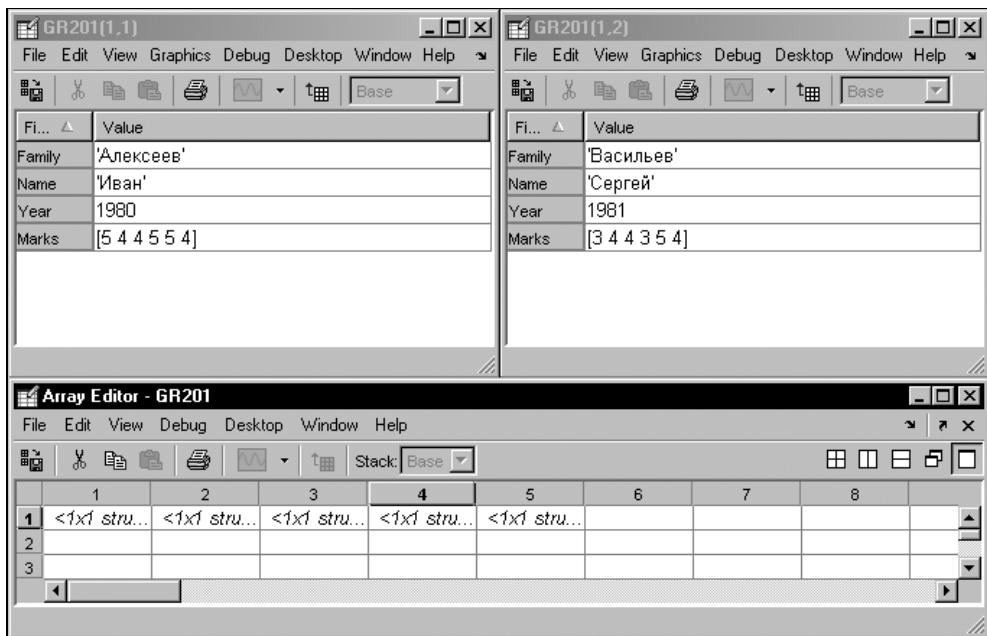
```
>> GR201
GR201 =
```

1x5 struct array with fields:

```
Family
Name
Year
Marks
```

В отличие от переменных, содержащих числа, массивы, строки или простые структуры, 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 массива требует задания ее индекса:

```
>> GR201(3)
ans =
Family: 'Кашин'
Name: 'Павел'
Year: 1979
Marks: [4 3 4 4 5 4]
```



**Рис. 8.4.** Просмотр элементов массива структур с помощью **Array Editor**

Для контроля и изменения значений полей массива структур удобно пользоваться редактором массивов. Двойной щелчок мышью по одной из клеток таблицы со структурой приводит к появлению ее содержимого в окне, снабженном вкладкой с именем массива и номером структуры. Если требуется одновременно просмотреть несколько структур массива (рис. 8.4), удобно вывести их в разные окна редактора при помощи кнопки **Undock** в строке меню.

Указание поля структуры позволяет получить доступ к его значению:

```
>> GR201(2).Year
```

```
ans =
```

```
1981
```

Функция `length`, примененная к массиву структур `GR201`, возвращает его длину, т. е. количество входящих в него структур:

```
>> length(GR201)
```

```
ans =
```

```
5
```

Дополните файл-программу, приведенную в листинге 8.8, операторами, выводящими значения полей структур. Используйте цикл `for` для перебора всех элементов массива `GR201` (листинг 8.9).

#### Листинг 8.9. Вывод значений полей структуры в командное окно

```
Len = length(GR201);
for k = 1:Len
 disp(GR201(k))
end
```

Массивы структур могут быть двумерными, тогда обращение к какой-либо структуре осуществляется при помощи двух индексов, а для выяснения размеров массива следует обратиться к функции `size`. Проход по всем элементам двумерного массива структур можно организовать при помощи двух вложенных циклов `for`.

#### Примечание

Двумерные массивы структур хранятся в памяти по столбцам, как и двумерные числовые массивы, поэтому допустимо обращение к их элементам с одним индексом.

## 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 имен полей структуры так, как описано выше. Операции, применяемые к значениям полей, должны соответствовать содержащимся в них данным. Создайте файл-функцию `groupprog`, отображающую среднюю успеваемость группы студентов по каждому курсу (см. табл. 8.1). Результат представьте в виде столбцовой диаграммы, число столбцов которой равно числу курсов. Алгоритм решения достаточно прост. Следует определить число студентов, используя функцию `length` 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 столбцовую диаграмму, используя функцию `bar`. Текст файл-функции `groupprog` приведен в листинге 8.10.

### Листинг 8.10. Файл-функция `groupprog` для определения успеваемости группы

```
function meanmarks = groupprog(GROUP);
% функция вычисляет средний балл студентов по каждому предмету
% и выводит результат в виде столбцовой 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 = length(GROUP);
% Определение количества курсов по информации для
% первого студента
Courses = length(GROUP(1).Marks);
% Инициализация массива meanmarks и заполнение его нулями
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);
```

Проверьте работу файла-функции `groupprog` на примере структуры GR201.

## 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 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. Для вывода значения поля структуры, содержащего строку, примените спецификатор формата `s`. Например, формат '`%-10s`' означает выравнивание строки по левому краю в поле вывода из 10 позиций.

**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 = length(GROUP);
% Открытие файла с именем filename для записи
F = fopen(filename, 'w');
% Запись шапки таблицы с выравниванием по левому краю каждой строки
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\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 из текстового файла, разделенные одним или несколькими пробелами, 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, числом считываемых в данном формате объектов и одним выходным аргументом, в который записывается результат.

`a = fscanf(идентификатор, 'список_форматов', число считываемых элементов)`  
Для считывания строки предусмотрен спецификатор формата `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 фамилию Александров в строковую переменную `Family`, целое число (год) 1990 — в переменную `Year`, звание 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` (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 перед закрытием файла следовало бы 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. Содержимое текстового файла для ввода матрицы

```
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 элементычитываются последовательно и помещаются в нужные позиции. Для того чтобы убедиться в этом, считайте из файла (листинг 8.15) матрицу командами

```
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 (доступ к полям структур массива описан в разд. "Массивы структур" данной главы). 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`.

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 числа студентов
count = 0;
% Создание пустого массива структур
GROUP = [];

% Последовательное считывание строк (начиная со второй)
% и распределение информации, содержащейся в ней,
% по полям структур массива GROUP. Каждая строка файла
% содержит информацию об одном студенте
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

Каждый элемент массива структур является простой структурой, и к нему применимы операции и функции для простой структуры, описанные выше. Однако все структуры массива должны иметь одинаковые имена полей, поэтому изменение поля (но не его значения) в одной структуре неизбежно влечет аналогичные изменения во всех структурах массива.

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 (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.

Для удаления поля простой структуры мы использовали функцию `rmfield`. В случае массива структур она удаляет выбранное поле во всех структурах массива. Первым ее входным аргументом является имя массива, а вторым — строка или строковая переменная с названием поля. 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. Определение названий полей массива структур, запись их в строковые переменные и дальнейшее использование ничем не отличаются от случая простой структуры.

```
>> GRFields = fieldnames(GR201)
GRFields =
 'Family'
 'Name'
 'Year'
 'Marks'
 'NBook'

>> f4 = GRFields{4}
f4 =
Marks
>> M5 = GR201(5).(f4)
M5 =
 5 5 5 5 4 5
```

Полученные строковые переменные или строки с названием имени поля используются в качестве входных аргументов функций `getfield` и `setfield`, которые, соответственно, предназначены для получения и установки значения поля структуры. Примеры вызова `getfield` и `setfield` приведены ниже.

```
>> Name1 = getfield(GR201(1), 'Name');
>> GR201(2) = setfield(GR201(2), '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" (логическая единица), либо "ложь" (логический 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. Функция `isfield` возвращает так же логический ноль или единицу.

## Массивы ячеек

Кроме числовых массивов, массивов строк и структур в MATLAB определен еще один тип переменных — массив ячеек, который хорошо приспособ-

лен для хранения *разнородных* данных. В данном разделе приведены 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. Организация данных массива ячеек более удобна 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. 8.5). 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. 8.5) в контейнеры массива ячеек `EXPER`.

|                                            |                                           |                                                 |
|--------------------------------------------|-------------------------------------------|-------------------------------------------------|
| <b>ячейка 1,1</b>                          | <b>ячейка 1,2</b>                         | <b>ячейка 1,3</b>                               |
| доц. Петров                                | доц. Гришин                               | асс. Зинин                                      |
| <b>ячейка 2,1</b>                          | <b>ячейка 2,2</b>                         | <b>ячейка 2,3</b>                               |
| 2.2 0.3 1.7                                | вариант N2                                | 2.0 0.2 1.4                                     |
| <b>ячейка 3,1</b>                          | <b>ячейка 3,2</b>                         | <b>ячейка 3,3</b>                               |
| Family Иванов<br>Name Алексей<br>Group 201 | Family Сергеев<br>Name Антон<br>Group 202 | Family Пашин<br>Name Антон<br>Info Вечерн. ф.-т |
| <b>ячейка 4,1</b>                          | <b>ячейка 4,2</b>                         | <b>ячейка 4,3</b>                               |
| -1.33 0.35 1.74<br>0.99 0.98 0.78          | -1.43 0.24 1.88<br>0.90 0.91 0.59         | рез. не получен                                 |

Рис. 8.5. Содержимое ячеек

## Примечание

Если имя создаваемого массива ячеек занято в рабочей среде под обычный массив, то перед поэлементным заполнением массива ячеек следует удалить его из рабочей среды командой `clear` или из окна **Workspace** браузера переменных, 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, обращение к которой

```
>> cellplot(EXPER)
```

приводит к появлению графического окна, изображенного на рис. 8.6. Массивы чисел или текстовые строки обозначаются макетами таблиц с соответствующим числом строк и столбцов, а записи — квадратами. Для наглядности на рис. 8.6 текстовая строка "Вариант N2" вынесена над обозначением массива, ее содержащего. В других ячейках текстовые строки не выводятся, поскольку они содержат много символов.

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.'}
```

Функция size находит размер массива ячеек:

```
>> size(EXPER)
ans =
```

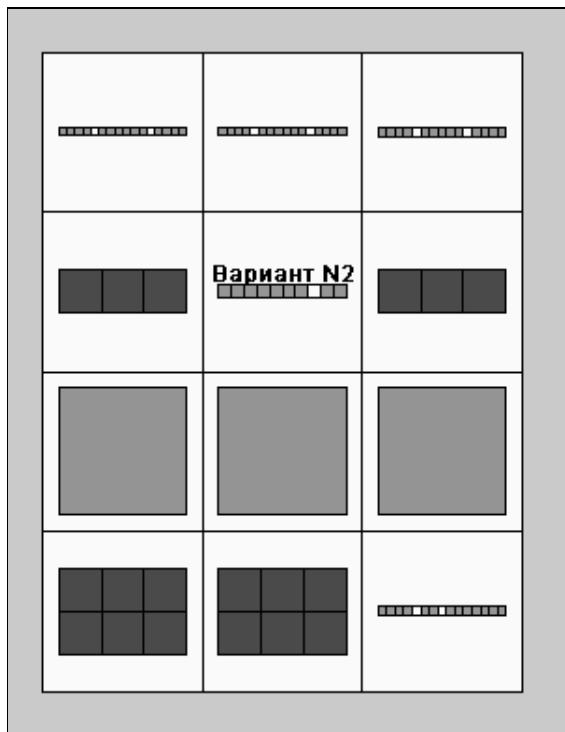


Рис. 8.6 Схема массива ячеек (celldisp)

### Примечание

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` возвращает число ячеек.

Доступ к содержимому контейнеров массива ячеек осуществляется при помощи индексов. Обратите внимание, что при обращении к элементам массива ячеек индексы могут заключаться как в круглые, так и фигурные скобки.

```
>> S = EXPER{3, 2}
```

```
S =
```

```
Family: 'Сергеев'
```

```
Name: 'Антон'
```

```
Group: 202
```

```
>> C = EXPER(3, 2)
```

```
C =
```

```
[1x1 struct]
```

В чем же разница? Изучите тип переменных *s* и *c*, например, в окне **Workspace** браузера рабочей среды. Переменная *s* является структурой (массивом структур размера 1 на 1), а *c* — ячейкой (массивом ячеек размера 1 на 1). Поэтому для обращения к содержимому элемента массива ячеек следует использовать *фигурные скобки*, если с ним требуется выполнить дальнейшие операции, определяемые типом содержимого ячейки. Если же необходимо выделить именно ячейку, то следует указать индексы в круглых скобках.

```
>> S.Name
```

```
ans =
```

```
Антон
```

Выбирайте нужные скобки, иначе получите сообщение об ошибке

```
>> C.Name
```

```
??? Attempt to reference field of non-structure array 'C'.
```

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" (логическую единицу), если ее входной аргумент — массив ячеек, и "ложь" (логический ноль) в противном случае. Имеет смысл также проверять, содержит ли текущая ячейка структуру или нет. Для этого следует воспользоваться функцией `isstruct`. Кроме того, перед обращением к полю структуры полезно убедиться в его наличии, для чего служит функция `isfield`.

## 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 выполнени-

ем 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. Как мы уже упоминали, вывод текста в командное окно осуществляется функцией `disp`.

Применение `disp` обеспечивает только одностороннюю связь программы с пользователем. Для интерактивного взаимодействия следует привлечь еще функцию `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`. Диалог файл-функции `calc` приведен ниже:

```
>> calc
Ведите первое число 1.2
Ведите арифметическую операцию (+, -, *, /) +
Ведите 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 чисел 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.

### Примечание

Часто кроме конвертирования числа в строку требуется обратное преобразование строки в число, которое осуществляется функцией `str2num`. Во входном аргументе задается строка из символов, например: '123.77', или '1.98e-03', а в выходном аргументе возвращается соответствующее число. Входным аргументом может быть и массив строк, который преобразуется в числовой массив. Если преобразование невозможно, например, `a = str2num('12A99')`, то результатом работы будет пустой массив (подробно о функциях преобразования сказано в *приложении 1*).

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
Ведите первое число 1.2
Ведите арифметическую операцию (+, -, *, /) +

```

Ведите второе число 3.1

1.2 + 3.1 = 4.3

### Листинг 8.20. Улучшение вида результата работы файл-функции calc

```
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 опе-

рации, 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 дополнитель но использовать возведение в степень, ввод числа  $\pi$ , и придает программе более наглядный вид. Действительно незаменимой и очень полезной функцией является eval, которая служит для выполнения команды MATLAB, записанной в строке или строковой переменной. Формирование строки с командой и ее выполнение функцией eval описаны в следующем разделе.

## Формирование и выполнение команд, функция eval

Встроенная функция eval позволяет выполнить строку, которая содержит команду MATLAB. Стока 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 переменной 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. 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 `у` 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

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, ':');
```

```
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`.

## Организация вывода текстовых результатов

Вывод результатов в удобно читаемой форме — одно из необходимых условий хорошо написанной программы. Для отображения в командном окне результатов работы файл-программы или файл-функции мы применяли функцию `disp` совместно с преобразованием чисел в строки или допускали вывод значения выражения в командное окно, не завершая оператор присваивания точкой с запятой. Этих приемов зачастую оказывается недостаточно для написания программы, которая общается с пользователем через командное окно.

Один из способов организации вывода текстовых данных в командное окно состоит в специальном вызове функции `fprintf` (которую вы использовали для записи текстовых файлов). Если в качестве идентификатора файла указать единицу или вообще пропустить его, то вывод происходит не в файл, а в командное окно. Создайте, например, файл-программу с текстом листинга 8.7, только исключите идентификатор файла `F` из списка входных аргументов функции `fprintf`. Запустите ее и убедитесь, что выводимая в командное окно информация соответствует той, которая ранее записывалась в текстовый файл.

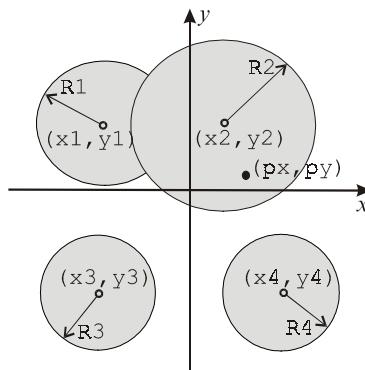
Возможности отображения массивов данных в MATLAB не ограничиваются их выводом в текстовый файл или командное окно. Функция `openvar` позволяет открыть окно редактора массивов **Array Editor** с содержимым числового массива, массива строк, структур или ячеек, определенного в рабочей среде. Стока или строковая переменная с именем массива указываются во входном аргументе `openvar`. Например, если в рабочей среде есть массив `MAS`, то следует написать `openvar('MAS')`. Примите во внимание, что `openvar` работает только с переменными рабочей среды. Переменные, созданные внутри файл-функции, являются локальными и не видны в рабочей среде, поэтому эта возможность доступна только в файл-программах (классификация переменных описана в разд. "Подфункции" и "Вложенные функции" главы 5).

## 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 более выходными аргументами и с тремя или более входными. При этом алгоритм `fminbnd` адаптируется к выбору пользователя и осуществляет требуемую последовательность действий. Обратитесь к встроенной справке по `fminbnd` при помощи `help` с именем функции для того, чтобы узнать о всех возможных вариантах вызова. В данном разделе описано создание собственных файл-функций, способных работать с переменным числом аргументов. Вам понадобится использовать массивы ячеек, основные сведения о которых вы получили при чтении этой главы (*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  $(x_1, y_1, R_1)$ ,  $(x_2, y_2, R_2)$  и т. д.) и точка с координатами  $(px, py)$ . Требуется определить, лежит ли точка внутри какого-либо круга или нет (рис. 8.7).



**Рис. 8.7.** 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. 8.7, вызов функции `point` будет выглядеть следующим образом:

```
f = point(px, py, [x1 y1 R1], [x2 y2 R2], [x3 y3 R3], [x4 y4 R4])
```

Первые три входных аргумента являются обязательными, но число вектор-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. 8.8.

| ячейка 1 | ячейка 2 | ячейка 3   | ячейка 4   | ячейка 5   | ячейка 6   |
|----------|----------|------------|------------|------------|------------|
| px       | py       | [x1 y1 R1] | [x2 y2 R2] | [x3 y3 R3] | [x4 y4 R4] |

Рис. 8.8. 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, `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 point**

```
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.

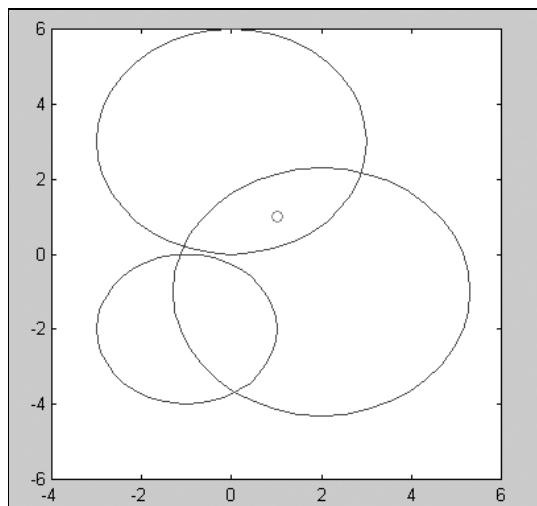
**Листинг 8.25. Вывод данных в графическое окно**

```
% 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. 8.9.



**Рис. 8.9.** 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.

1.  $f = \text{point}(\text{px}, \text{py}, [\text{x}_1 \text{ } \text{y}_1 \text{ } \text{R}_1], [\text{x}_2 \text{ } \text{y}_2 \text{ } \text{R}_2], \dots)$ . В f записывается ноль или единица, в зависимости от принадлежности точки какому-либо кругу.
2.  $[f, Nc] = \text{point}(\text{px}, \text{py}, [\text{x}_1 \text{ } \text{y}_1 \text{ } \text{R}_1], [\text{x}_2 \text{ } \text{y}_2 \text{ } \text{R}_2], \dots)$ . В f записывается ноль или единица, а в Nc — число кругов, которым принадлежит точка.
3.  $[f, Nc, Num] = \text{point}(\text{px}, \text{py}, [\text{x}_1 \text{ } \text{y}_1 \text{ } \text{R}_1], [\text{x}_2 \text{ } \text{y}_2 \text{ } \text{R}_2], \dots)$ . 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 vararginout, для чего следует предусмотреть операторы, записывающие выходные аргументы в соответствующие ячейки данного массива. Заголовок файл-функции с переменным числом входных и выходных аргументов в M-файле выглядит следующим образом:

```
function varargout = myfun(varargin)
```

Аргументы, указание которых обязательно, могут быть помещены в списки входных и выходных аргументов до varargin и vararginout соответственно. Например, первый 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 аргументов при помощи vararginout.

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 и 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 программированию 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 М-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 переменной, а ее вычисление производится при помощи команды `feval`. Например, синус можно вычислить обычным способом, вызывав `sin(x)`, или используя `feval` с входными аргументами — названием '`sin`' и аргументом `x`:

```
>> x = pi/2;
>> feval('sin', x)
ans =
 1
```

Первый входной аргумент `feval` может быть и указателем на inline-функцию или анонимной функцией, результат будет аналогичный:

```
>> feval(@sin, x)
ans =
 1
```

В общем случае все входные аргументы исследуемой функции задаются в списке аргументов `feval` через запятую после имени функции, например, следующие три вызова некоторой функции `myfun` эквивалентны:

```
[a, b] = myfun(x, y, z)
[a, b] = feval('myfun', x, y, z)
[a, b] = feval(@myfun, x, y, z)
```

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 М-файле. Считается, что задан отрезок, на кото-

ром ищется корень, и точность вычислений. Алгоритм метода половинного деления очень простой.

1. Проверить, что на границах отрезка  $f(x)$  принимает значения разных знаков.
2. Если длина отрезка меньше заданной точности, взять в качестве приближенного значения корня любую из его границ.
3. Найти среднюю точку отрезка и вычислить в ней значение функции. Если в центре отрезка функция равна нулю, то корень найден и вычисления останавливаются.
4. В качестве нового отрезка выбрать ту половину, на границах которой знаки  $f(x)$  различны, и продолжить вычисления с п. 2.

Напишите файл-функцию `half`, реализующую метод половинного деления (листинг 8.27). Обращение к `half` должно выглядеть следующим образом:

```
>> r = half('myf', a, b, e)
```

или

```
>> 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('Однаковые 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 = right;

```

Теперь в качестве исследуемой функции может выступать как встроенная математическая функция MATLAB, так и функция с одним входным аргументом, описанная пользователем в М-файле, например:

```
>> r = half('sin', 3, 3.5, 1.0e-4)
```

```
r =
```

```
3.1416
```

или

```
>> r = half(@sin, 3, 3.5, 1.0e-4)
```

```
r =
```

```
3.1416
```

Путь к М-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).

Улучшите интерфейс файл-функции `half` (см. 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 параметр 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 vararginout.

### Листинг 8.28. Файл-функция half с переменным числом аргументов

```
function [root, vararginout] = 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('Однаковые 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 = right;
if nargout == 2
 varargout{1} = feval(fname, root);
end
```

Теперь файл-функция `half` стала более универсальной, например, возможно такое обращение к ней:

```
>> [r, f] = half(@sin, 3, 3.5)
r =
 3.1416
f =
 2.1609e-005
```

### Примечание

При передаче функции в качестве аргумента кроме ее имени или указателя можно использовать также имя `inline`-функции или анонимную функцию (см. разд. "Встраиваемые и анонимные функции" главы 6).

## Перманентные переменные

Кроме глобальных и локальных переменных имеется еще один тип переменных — *перманентные* (*persistent*). Так же как и локальные, они доступны только в пределах своей функции, но в отличие от них сохраняются в памяти после выхода из функции, и их значения можно использовать при

следующем ее вызове. При первом вызове функции все перманентные переменные являются пустыми массивами.

### Примечание

Перманентные переменные — это фактически локальные статические переменные. Однако мы будем придерживаться терминологии пакета MATLAB.

Перманентные переменные оказываются полезными при программировании рекурсивных функций, т. е. функций, повторно обращающихся к себе (в разд. "Рекурсивные функции" этой главы приведены примеры таких функций).

Поясним использование перманентных переменных на примере вычисления определенного интеграла по квадратурным формулам. В случае подынтегральных функций с интегрируемой особенностью на границе промежутка стандартные функции quad и quadl приводят к появлению предупреждения о достижении минимально возможного шага дробления исходного промежутка и нахождении интеграла с невысокой точностью (см. разд. "Вычисление определенных интегралов" главы 6).

Сейчас мы приступим к написанию собственной файл-функции для вычисления определенных интегралов при помощи чебышевской квадратурной формулы:

$$\int_c^d f(x)dx \approx \frac{d-c}{n} \sum_{k=1}^n f\left(c + \frac{d-c}{2}(1+t_k)\right).$$

с  $n$  узлами  $t_k$ , которые принадлежат базовому промежутку  $[-1, 1]$  и не содержат точек 1 и -1. Например, для  $n = 5$ , значения узлов квадратурной формулы следующие:

$$t_1 = -\frac{1}{2}\sqrt{\frac{5-\sqrt{11}}{3}}, \quad t_2 = -\frac{1}{2}\sqrt{\frac{5+\sqrt{11}}{3}}, \quad t_3 = 0, \quad t_4 = \frac{1}{2}\sqrt{\frac{5-\sqrt{11}}{3}}, \quad t_5 = \frac{1}{2}\sqrt{\frac{5+\sqrt{11}}{3}},$$

для  $n = 4$ :

$$t_1 = -\frac{1}{3} - \frac{2}{3\sqrt{5}}, \quad t_2 = -\frac{1}{3} + \frac{2}{3\sqrt{5}}, \quad t_3 = +\frac{1}{3} - \frac{2}{3\sqrt{5}}, \quad t_4 = +\frac{1}{3} + \frac{2}{3\sqrt{5}}.$$

Алгоритм вычисления интеграла предполагает использование обобщенной квадратурной формулы, т. е. деление исходного интервала на заданное количество равных частей и вычисление интеграла на каждом из подинтервалов по квадратурной формуле. Изучая программирование рекурсивных функций при чтении следующего раздела этой главы, вы сможете улучшить этот алгоритм за счет адаптивного выбора шага дробления и получить функцию, пригодную для практических расчетов.

Реализуйте алгоритм вычисления определенного интеграла по обобщенным квадратурным формулам с пятью узлами в файл-функции `chn5` с подфункцией `ch5`. Основная функция `chn5` должна иметь четыре входных аргумента.

1. Имя, или указатель на файл-функцию с подынтегральной функцией.
2. Левая граница отрезка интегрирования.
3. Правая граница отрезка интегрирования.
4. Число дроблений исходного отрезка.

Результат вычислений возвращается в выходном аргументе файл-функции `chn5`. Подфункция `ch5` служит для нахождения приближенного значения интеграла по произвольному отрезку  $[c, d]$  по квадратурной формуле чебышевского типа с пятью узлами и будет вызываться в основной функции для каждого из подинтервалов. Список входных аргументов `ch5` должен содержать:

1. Имя, или указатель на файл-функцию с подынтегральной функцией.
2. Левую границу отрезка интегрирования.
3. Правую границу отрезка интегрирования.

В подфункции `ch5` следует вычислить значения узлов для базового промежутка  $[-1, 1]$ , записать их в вектор длины 5, найти значение интеграла по квадратурной формуле и вернуть его в выходном аргументе. Очевидно, что массив со значениями узлов достаточно заполнить только один раз при первом обращении к `ch5`. Следовательно, хорошим выбором является объявление этого массива как перманентной переменной. Не забудьте сделать проверку содержимого перманентной переменной на пустой массив, который означает, что она еще не была инициализирована. Для проверки воспользуйтесь функцией `isempty`, которая возвращает логическую единицу, если ее входной аргумент оказался пустым массивом, и ноль, если массив непустой.

В листинге 8.29 представлена возможная реализация основной функции `chn5` и подфункции `ch5`. В теле подфункции `ch5` определен перманентный массив узлов `x5` для базового промежутка. Он заполняется только один раз при первом обращении к подфункции.

#### Листинг 8.29. Вычисление интеграла по чебышевской квадратурной формуле

```
function int_val = chn5(f, a, b, n)
% Вычисление определенного интеграла
% по чебышевским квадратурным формулам
% Использование int_val = chn5(f, a, b, n)
```

```
% f - подынтегральная функция
% имя файл-функции, указатель или inline-функция
% a, b - нижний и верхний пределы интегрирования
% n - число дроблений отрезка интегрирования
```

```
J = 0; % начальное значение суммы
h = (b - a)/n; % шаг дробления отрезка [a, b]
% накопление суммы интегралов по каждому подинтервалу
for i = 1:n
 b = a + h;
 % вычисление интеграла по подинтервалу
 % по чебышевской квадратурной формуле
 J = J + ch5(f, a, b);
 a = b;
end;
int_val = J;
```

```
function chval = ch5(f, c, d)
% объявление перманентного массива для хранения
% значений узлов квадратурной формулы
persistent x5
% проверка, был ли массив инициализирован
if isempty(x5)
 % вычисление значений узлов квадратурной формулы
 d1 = (5 - sqrt(11))/3;
 d2 = (5 + sqrt(11))/3;
 x5(4) = sqrt(d1)/2;
 x5(5) = sqrt(d2)/2;
 x5(1) = -x5(5);
 x5(2) = -x5(4);
 x5(3) = 0;
end;
% вычисление интеграла по [c, d]
xc = c + (d - c)*0.5*(1 + x5);
fc = feval(f, xc);
chval = (d - c)/5*sum(fc);
```

Для проверки вычислите интеграл по промежутку  $[0, 1]$  от функции  $y = x^{-0.9}$ , увеличивая количество дроблений исходного промежутка от 10 с возрастанием на порядок до 10 000. Формирование вектора таких значений, равнотостоящих в логарифмической шкале, удобно производить при помощи специальной функции `logspace`, например, `z = logspace(2, 6, 5)` генерирует вектор из пяти чисел  $10^2, 10^3, \dots, 10^6$ . В листинге 8.30 приведена файл-программа `Chn5_test` для тестирования функции `chn5`.

### Листинг 8.30. Файл-программа Chn5\_test для тестирования функции chn5

```
% Определение inline-функции
func=inline('x.^-0.9');
% вывод шапки таблицы
disp([' Интеграл ', ' Число дроблений ']);
% увеличение числа дроблений на порядок в цикле
% и вывод таблицы значений
for n = logspace(1, 4, 4)
 int_ch5 = chn5(func, 0, 1, n);
 d1 = num2str(int_ch5);
 d2 = num2str(n);
 disp([blanks(3), d1, blanks(14), d2]);
end;
```

Выполнение файл-программы дает следующий результат:

```
>> Chn5_test
Интеграл Число дроблений
 4.6802 10
 5.7743 100
 6.6434 1000
 7.3338 10000
```

Пока файл-функция `chn5`, основанная на обобщенных квадратурных формулах, работает намного хуже, чем `quadl`. Даже при дроблении интервала на 10 000 частей не удалось получить хорошую точность из-за особенности на левой границе отрезка ( $a = 0$ ). Вычислите интеграл от функции  $y = x^{-0.5}$ . Результат получается лучше, но тоже не очень точный. Известно, что в случае интегрируемой особенности для достижения приемлемой точности необходимо выбирать переменный шаг дробления исходного промежутка. Вы

сможете усовершенствовать файл-функцию `chn5`, добавив адаптивный выбор шага, при чтении следующего разд. "Рекурсивные функции" этой главы.

## Рекурсивные функции

Содержание этого раздела не претендует на описание технологии проектирования рекурсивных алгоритмов. Мы ограничимся только иллюстративными примерами с пояснениями их реализации в MATLAB. Поддержка выполнения рекурсивных алгоритмов состоит в том, что при рекурсивном вызове функции все ее переменные сохраняются в памяти, а при возврате в точку вызова они восстанавливаются. Использование перманентных переменных может быть полезно для организации дополнительной передачи данных при вызове функции, т. к. они не восстанавливаются после завершения текущего вызова, а имеют те значения, которые были перед выходом из тела функции. Далее эти приемы иллюстрируются на нескольких примерах.

В качестве первого примера рекурсивной функции рассмотрим адаптивное изменение шага дробления отрезка при интегрировании по обобщенным квадратурным формулам. Итак, вернемся к проблеме интегрирования функций с интегрируемой особенностью на границе промежутка. Для решения поставленной задачи вы начали работу над файл-функцией `chn5` при чтении разд. "Перманентные переменные" этой главы (см. листинг 8.29). Файл-функция `chn5` делит отрезок интегрирования на заданное число подынтервалов равной длины и вычисляет интеграл по каждому из подынтервалов по квадратурной формуле чебышевского типа с пятью узлами. Известно, что для эффективного численного интегрирования функции с особенностью желательно сгущать точки дробления в области, прилегающей к особой точке, добиваясь примерно одинаковой погрешности интегрирования  $\epsilon$  на каждом из подынтервалов. Возникает вопрос: как оценить погрешность величины интеграла, полученного при помощи квадратурной формулы, если точного значения интеграла мы не знаем. Для оценки точности на подынтервале интегрирования можно воспользоваться приемом, основанным на идее Рунге, — вычислить два приближенных значения интеграла по квадратурным формулам с различным числом узлов и сравнить их (значения узлов квадратурной формулы приведены в разд. "Перманентные переменные" этой главы).

Алгоритм численного интегрирования состоит из следующих шагов.

1. Начальное разбиение отрезка — единственный подынтервал совпадает с отрезком.
2. Для каждого подынтервала находится два приближения  $J_4$  и  $J_5$  по квадратурным формулам чебышевского типа с четырьмя и пятью узлами.

3. Если  $|J_4 - J_5| < \varepsilon$ , то точность на подынтервале считается достигнутой, если это неравенство не выполняется, то он делится пополам и для каждого из образовавшихся подынтервалов повторяются действия пп. 2 и 3.

Таким образом, вычисления происходят в адаптивном режиме. Подынтервалы автоматически делятся на два типа: часть из них остается неизменными (на которых достигнута заданная точность) и интегрирование по ним не производится, а остальные подвергаются дроблению для уменьшения погрешности. Оставим в стороне вопрос о выборе параметра  $\varepsilon$  (приведенной точности), ибо он выходит за рамки излагаемого материала. Отметим только, что  $\varepsilon$  не определяет число верных знаков результата, а служит для равномерного распределения погрешности по подынтервалам.

Привлечение рекурсивного вызова позволяет написать простую файл-функцию, поскольку число подынтервалов и их длины заранее неизвестны и изменяются в процессе вычислений, а для всех появляющихся подынтервалов требуется выполнить одинаковые действия. Имеет смысл предусмотреть ограничение на минимальную длину подынтервалов, иначе уровень вложенности рекурсивных вызовов может быть слишком большим, что повлечет увеличение времени счета.

### Примечание

MATLAB имеет ограничение по количеству вложенных вызовов функции — 500, что можно узнать при помощи команды `get(0, 'RecursionLimit')`. Изменение этого параметра, например на 700 производится при помощи `set(0, 'RecursionLimit', 700)`. Следует иметь в виду, что увеличение допустимого числа рекурсивных вызовов может привести к нехватке памяти и зависанию пакета.

Реализуйте адаптивный алгоритм интегрирования в файл-функции `cheb45` с одним выходным аргументом — приближенным значением интеграла и пятью входными:

- Подынтегральная функция, которая задается ссылкой на соответствующую файл-функцию, или ее именем, или `inline`-функцией, или анонимной функцией.
- Нижний предел интегрирования  $a$ .
- Верхний предел интегрирования  $b$ .
- Параметр  $\varepsilon$  для контроля точности на подынтервалах интегрирования.
- Параметр  $\varepsilon_m$  для ограничения на минимальную длину подынтервалов.

Используйте подфункцию `ch5` для вычисления интеграла по подынтервалам при помощи квадратурной формулы с пятью узлами (см. листинг 8.29) и

подфункцию ch4 для случая четырех узлов, которая программируется аналогичным образом. В случае возникновения затруднений воспользуйтесь листингом 8.31, содержащим текст основной функции и двух подфункций.

### Листинг 8.31. Рекурсивная функция для вычисления определенного интеграла

```
function J = cheb45(fun, a, b, e, em)
% Вычисление определенного интеграла
% Использование J = cheb45(fun, a, b, e, em)
% fun - имя файл-функции, указатель или inline-функция
% a - нижний предел интегрирования
% b - верхний предел интегрирования
% e - параметр для контроля за равномерным распределением погрешности
% em - минимальная длина подинтервалов
% J - значение интеграла

% вычисление интеграла по пятиузловой квадратурной формуле
J5 = ch5(fun, a, b);
% проверка на достижение минимальной длины подинтервала
if abs(b - a) > em
 % вычисление интеграла по четырехузловой квадратурной формуле
 J4 = ch4(fun, a, b);
 % оценка погрешности на текущем подинтервале
 if abs(J4 - J5) <= e
 % погрешность меньше или равна допустимой
 J = J5;
 else
 % погрешность больше допустимой
 % находим половину длины подинтервала
 h = (b - a)/2;
 % вычисляем интеграл по левой половине подинтервала
 JL = cheb45(fun, a, a + h, e, em);
 % вычисляем интеграл по правой половине подинтервала
 JR = cheb45(fun, a + h, b, e, em);
 % вычисляем интеграл по исходному подинтервалу
 J = JL + JR;
 end
end
```

```

function I = ch5(fun, c, d)
% Вычисление интеграла по пятиузловым формулам
persistent x5
if isempty(x5)
 d1 = (5 - sqrt(11))/3;
 d2 = (5 + sqrt(11))/3;
 x5(4) = sqrt(d1)/2;
 x5(5) = sqrt(d2)/2;
 x5(1) = -x5(5);
 x5(2) = -x5(4);
 x5(3) = 0;
end;
h = (d - c)/2;
xc = c + h*(1 + x5);
fc = feval(fun, xc) ;
I = sum(fc)*(d - c)/5;

function I = ch4(fun, c, d)
% Вычисление интеграла по четырехузловым формулам
persistent x4
if isempty(x4)
 d1 = (1 - 2/sqrt(5))/3;
 d2 = (1 + 2/sqrt(5))/3;
 x4(3) = sqrt(d2);
 x4(4) = sqrt(d1);
 x4(1) = -x4(4);
 x4(2) = -x4(3);
end;
h = (d - c)/2;
xc = c + h*(1 + x4);
fc = feval(fun, xc);
I = sum(fc)*(d - c)/4;

```

Проверьте работу файл-функции cheb45, вычисляя интеграл

$$\int_0^1 x^{-0.9} dx$$

для различных значений параметра  $\epsilon = 10^{-1}, 10^{-2}, \dots, 10^{-9}$ . Для тестирования удобно создать файл-программу, в которой значение  $\epsilon$  изменяется в цикле, а результаты сводятся в таблицу. Возможный вариант файл-программы Ch\_test45 приведен в листинге 8.32.

#### Листинг 8.32. Файл-программа Ch\_test45 для тестирования функции cheb45

```
% определение inline-функции
f = inline('x.^-0.9');
% вывод шапки таблицы
disp([' e ', ' Значение интеграла ']);
% изменение точности в цикле
for e = logspace(-1, -9, 9)
 % вычисление интеграла, минимальная длина подынтервалов - ноль
 J = cheb45(f, 0, 1, e, 0);
 % вывод строки таблицы
 fprintf('%9.0e %15.10f\n', e, J)
end
```

Наша файл-функция достаточно успешно работает в случае подынтегральных функций с интегрируемой особенностью.

```
>> Ch_test45
 e Значение интеграла
1e-001 4.9244236668
1e-002 9.4845672033
1e-003 9.9475656427
1e-004 9.9945745917
1e-005 9.9993836026
1e-006 9.9999301713
1e-007 9.9999914100
1e-008 9.9999989635
1e-009 9.9999998738
```

Сравнение с файл-функцией chn5 (листинг 8.29), допускающей подынтервалы только равной длины, свидетельствует об эффективности адаптивного рекурсивного алгоритма. При интегрировании функции с более сильной особенностью, например,  $x^{-0.95}$ , потребуется увеличение максимально допустимого числа вложенных вызовов.

## Примечание

Конечно, результат для тестовых примеров будет еще более убедительным, если применить специальные квадратурные формулы с весами, учитывающими характер интегрируемой особенности. Но построение обобщенных формул такого класса в общем случае не представляется возможным. Поэтому для постоянного применения они неудобны и здесь не рассматриваются.

При использовании рекурсии часто необходимо получить информацию о числе рекурсивных вызовов и значениях данных на каждом уровне рекурсии. В нашем примере это может быть номер вызова функции и границы текущего подинтервала интегрирования. Одна из возможных модификаций файл-функции `cheb45` приведена в листинге 8.33, который следует дополнить подфункциями `ch4` и `ch5`. Новая файл-функция `cheb45c` не только вычисляет интеграл, но и выводит промежуточные сведения об аддитивном подборе шага интегрирования. Переменное число входных и выходных параметров позволяет осуществить сбор статистики после первого рекурсивного обращения к файл-функции.

### Листинг 8.33. Функция `cheb45c` с выводом информации о рекурсивных вызовах

```

function [J, varargout] = cheb45c(fun, a, b, e, em, varargin)
if isempty(varargin)
 % первый вызов файл-функции
 % в счетчик числа вызовов заносится 1
 count = 1;
 varargout{1} = count;
 % вывод шапки таблицы
 disp(['номер вызова', ' левая граница', ' правая граница'])
else
 % рекурсивный вызов
 % увеличение счетчика числа вызовов на 1
 varargout{1} = varargin{1} + 1;
 count = varargout{1};
end
% вывод строки таблицы о номере рекурсивного вызова
% и границах подинтервала
fprintf('%7d %18.4e %14.4e\n', count, a, b)
J5 = ch5(fun, a, b);
if abs(b - a) > em

```

```
J4 = ch4(fun, a, b);
if abs(J4 - J5) <= e
 J = J5;
else
 h = (b - a)/2;
 [JL, count] = cheb45c(fun, a, a + h, e, em, count);
 [JR, count] = cheb45c(fun, a + h, b, e, em, count);
 J = JL + JR;
end
end
% возвращение в выходном аргументе текущего номера вызова функции
varargout{1} = count;
```

Проверьте работу файл-функции cheb45c:

```
>> J = cheb45c(f, 0, 1, 1e-5, 0)
```

### Примечание

Если оценка количества вызовов рекурсивной функции не требуется по существу алгоритма, а нужна только на этапе отладки программы, то удобнее пользоваться глобальными переменными, т. к. после завершения отладки глобальные счетчики вызовов могут быть удалены или закомментированы.

Алгоритмы стандартных функций MATLAB quad, quadl являются рекурсивными с адаптивным выбором шага интегрирования. Вы можете самостоятельно изучить их, поскольку соответствующие файл-функции имеют открытый код. Они находятся в подкаталоге \toolbox\matlab\funfun\ основного каталога MATLAB.

Большинство вычислительных алгоритмов можно реализовать, не используя механизм рекурсии, однако встречаются задачи рекурсивные по своей природе. Рассмотрим задачу о выводе содержимого полей структуры в командное окно, поля которой могут быть также структурами, причем уровень вложенности структур произвольный.

Создайте структуру transaction, подобную рассмотренной в разд. "Простые структуры" этой главы.

```
>> A = struct('currency', 'USD', 'price', 0.23, 'bid', 0.21, ...
'ask', 0.27);
>> transaction = struct('time', [10,20,46], 'stock', 'EESR', ...
'volume', ... 10000, 'strike', A, 'exchange', 28.5214);
```

При просмотре содержимого структуры `transaction` в командном окне имена полей вложенной структуры `a` и их значения не отображаются. Запрограммируйте рекурсивную функцию `structfield`, выводящую значения всех полей структуры, включая поля вложенных структур. В начале алгоритма задайте перманентную переменную, которая имеет смысл счетчика вложенности. Далее запишите в массив ячеек имена всех полей структуры, воспользовавшись функцией `fieldnames`. Организуйте цикл по всем полям структуры. В цикле проверяйте содержимое текущего поля при помощи функции `isstruct`, которая возвращает логическую единицу, если переданный ей параметр является структурой, или логический ноль в противном случае. Если значение текущего поля — структура, то вывод названия ее полей и их содержимого осуществляется рекурсивным вызовом функции. В остальных случаях требуется просто вывести название поля и его значение. Для более наглядного отображения вложенных структур в командном окне предусмотрите отступ от левого края, пропорциональный уровню структуры, т. е. значению счетчика вложенности.

Листинг 8.34 содержит один из вариантов рекурсивной файл-функции `structfield`.

#### Листинг 8.34. Рекурсивная файл-функция для просмотра вложенных структур

```
function structfield(somestruct);
% Файл-функция для вывода содержимого вложенных структур

% вводим перманентную переменную - счетчик вложенности
persistent level
if isempty(level)
 % первый вызов функции, инициализация счетчика вложенности
 level = 0;
end
% запись имен полей в массив ячеек
fnames = fieldnames(somestruct);
% определение количества полей
L = length(fnames);
% проход по всем полям структуры в цикле
for k = 1:L
 % получение имени текущего поля
 field = fnames{k};
 % создаем строку из пробелов для отступа от левого края
 str = blanks(10*level);
```

```
% проверка, является ли текущее поле структурой
if isstruct(somestruct.(field))
 % текущее поле - структура
 % увеличиваем счетчик вложенности
 level = level + 1;
 % выводим имя поля, являющегося структурой
 disp([str, 'значения полей вложенной структуры ', field])
 % реурсивный вызов функции
 structfield(somestruct.(field));
 % уменьшаем счетчик вложенности
 level = level - 1;
else
 % текущее поле не является структурой
 % выводим имя поля и его значение
 disp([str 'значение поля ', field])
 disp(somestruct.(field))
end
end
```

Проверьте работу файла-функции structfield на примере структуры transaction:

```
>> structfield(transaction)
значение поля time
 10 20 46
значение поля stock
EESR
значение поля volume
 10000
значения полей вложенной структуры strike
 значение поля currency
USD
 значение поля price
0.2300
 значение поля bid
0.2100
 значение поля ask
0.2700
```

значение поля `exchange`

28.5214

Дополните самостоятельно функцию `structfield` контролем вводимого параметра, который должен быть структурой.

### Примечание

В функции `structfield` счетчик уровня вложенности `level` не является обязательным. Он служит только для вычисления длины отступа от левого края при выводе вложенных структур.

В заключение рассмотрим пример организации алгоритма при помощи рекурсивного вызова *подфункции*, что оказывается удобным при накоплении данных в ходе рекурсии. Задача состоит в поиске всех локальных минимумов функции  $f(x)$  внутри заданного отрезка  $[a, b]$ . Для ее решения подходит следующая последовательность действий.

1. Выбираем середину отрезка в качестве начального приближения к точке минимума и вычисляем абсциссу  $z$  локального минимума  $f(x)$  на  $[a, b]$ , используя стандартную функцию `fminsearch`.
2. Если  $z$  лежит внутри промежутка, то в качестве нового отрезка  $[a, b]$  последовательно выбираем  $[a, z]$  и  $[z, b]$  и выполняем для них п. 1. Поиск для текущего отрезка завершается, если  $z = a$ , либо  $z = b$ .

Этот алгоритм применим для многих функций, имеющих конечное число локальных минимумов.

Перед обсуждением программирования алгоритма разберем подробнее п. 2, в котором проверяется условие нахождения точки внутри промежутка. Пусть функция `fminsearch` вычислила абсциссу локального минимума  $z$ , удовлетворяющую неравенству  $a < z < b$ . В соответствии с алгоритмом, поиск локальных минимумов осуществляется теперь на отрезках  $[a, z]$  и  $[z, b]$ . Пусть в качестве нового отрезка  $[a, b]$  берется отрезок  $[a, z]$ . Если внутри  $[a, b]$  нет локального минимума, то `fminsearch` вернет значение  $z$ , близкое к  $b$ , поскольку локальный минимум находится приближенно. Выполнение неравенства  $a < z < b$  повлечет ошибочное предположение о наличии локального минимума и ненужное разбиение отрезка  $[a, b]$  на две части. Для предотвращения такой ситуации необходимо учесть погрешность, с которой `fminsearch` вычисляет локальный минимум, и вместо усло-

вия  $a < z < b$  проверять  $a + \epsilon < z < b - \epsilon$ , где величина  $\epsilon$  превосходит погрешность счета локального минимума.

Перейдем теперь к описанию возможной реализации алгоритма в файл-функции с основной функцией `allmin` и рекурсивной подфункцией `loc_min`. Заголовок основной функции

```
function [xmin, ymin] = allmin(fun, a, b)
```

предполагает ее вызов от указателя на файл-функцию, вычисляющую  $f(x)$ , и границ исходного отрезка  $a$  и  $b$ . Вместо указателя может быть имя файл-функции или `inline`-функция. Основная файл-функция должна вернуть пару векторов со значениями абсцисс и ординат точек локальных минимумов.

Интерфейс подфункции удобно организовать следующим образом.

```
function xmin = loc_min(fun, a, b, e, new_set)
```

где в качестве  $a$  и  $b$  выступают границы текущего промежутка, на котором следует найти локальный минимум;  $e$  — величина  $\epsilon$  в условии принадлежности локального минимума отрезку; `new_set` — флаг инициализации накапливаемых переменных. При первом вызове функции флаг `new_set` следует задать отличным от 0, например, 1. При повторных вызовах функции его значение должно быть равно 0. В обсуждаемом примере иллюстрируется еще одно применение перманентных переменных, о котором шла речь ранее — передача данных при повторных вызовах функции. Выходной аргумент `xmin` — массив с вычисленными абсциссами локальных минимумов.

Мы намеренно вынесли рекурсивную часть алгоритма в подфункцию, поскольку список ее входных аргументов содержит параметры  $e$  и `new_set`, которые не входят в исходные данные задачи, задаваемые пользователем. Один из возможных вариантов основной функции с подфункцией приведен в листинге 8.35. Искомые точки упорядочиваются по возрастанию абсцисс и отмечаются маркерами на графике исследуемой функции

#### Листинг 8.35. Функция поиска локальных минимумов функции на промежутке

```
function [xmin, ymin] = allmin(fun, a, b)
% Поиск локальных минимумов fun внутри заданного интервала (a, b)
% Использование [xmin, ymin] = allmin(fun, a, b)

% задание погрешности для проверки условия попадания точки внутрь (a, b)
e = 1e-4*(b - a);
% вызов рекурсивной подфункции
xmin = loc_min(fun, a, b, e, 1);
```

```
% сортировка точек минимума в порядке возрастания
xmin = sort(xmin);

% вычисление значений функции в точках локального минимума
ymin = feval(fun, xmin);

% вывод графика функции с отмеченными точками локального минимума
figure
fplot(fun, [a b])
hold on
plot(xmin, ymin, 'ro')

function xmin = loc_min(fun, a, b, e, new_set)
% Рекурсивная подфункция для поиска всех локальных минимумов

% Начальная инициализация для накапливаемых результатов
persistent n pnt
if new_set ~= 0
 pnt = [];
 n = 0;
end;
% задание начального приближения
c = 0.5*(a + b);
% поиск локального минимума на [a, b]
z = fminsearch(fun, c);
if and(z > a + e, z < b - e)
 % найден новый локальный минимум увеличиваем счетчик
 % и добавляем локальный минимум в массив
 n = n + 1;
 pnt(n) = z;
 % поиск локальных минимумов на подинтервалах
 loc_min(fun, a, z, e, 0);
 loc_min(fun, z, b, e, 0);
end
% возвращаем массив найденных точек
xmin = pnt;
```

Проверьте работу файл-функции, вычислив локальные минимумы функции  $f(x) = \sin(x) + \sin(10x)$  на отрезке  $[-2, 2]$ .

```
>> f = inline('sin(x) + sin(10*x)')
>> [xmin, ymin] = allmin(f, -2, 2)
```

```
xmin =
-0.1669 -1.4153 -0.7924 1.0950 0.4623 1.7295
ymin =
-1.1613 -1.9878 -1.7096 -0.1100 -0.5500 -0.0124
```

### Примечание

Функцию `loc_min` можно выделить в отдельный файл и использовать самостоятельно. В этом случае флаг `new_set` (если его значение ноль) позволяет к найденным ранее точкам добавить локальные минимумы той же функции из других промежутков.

Выбор средней точки отрезка в качестве начального приближения в `fminsearch` не всегда приводит к нахождению локального минимума внутри отрезка. Можно усовершенствовать алгоритм, выбирая различные начальные приближения, например, равноотстоящие с некоторым шагом точки.

## Диалоговая отладка программ

Среда программирования в MATLAB имеет стандартные средства диалоговой отладки программ (файл-программ и файл-функций). Эти средства встроены в редактор М-файлов.

### Точки останова, пошаговое выполнение программы

При отладке программы используется понятие точки останова (Breakpoint). Точка останова это строка исходного текста программы, при достижении которой выполнение программы приостанавливается, и у вас появляется возможность обратиться к тексту программы в редакторе М-файлов для просмотра текущих значений переменных, изменения точек останова или прекращения процесса отладки в случае локализации ошибки. Точки останова можно установить в любом М-файле, открытому в редакторе.

Точки останова бывают безусловные, условные и временные. Безусловная точка останова связывается с исполняемой строкой программы. Исполняемые строки программы (в отличие от комментариев) отмечены в редакторе знаком "-", расположенным слева от строки. Щелчок мыши по нему приводит к появлению безусловной точки останова в данной строке, при этом знак "-" заменяется на красный кружок. Для удаления безусловной точки останова применяется щелчок мышью по кружку. Аналогичного результата для текущей строки программы можно добиться выбором пункта **Set/Clear**

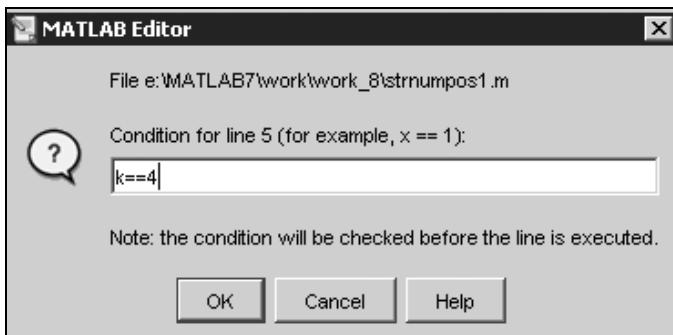
**Breakpoint** в меню **Debug** редактора М-файлов, или нажатием <F12>, или кнопкой **Set/Clear Breakpoint** на панели инструментов.

### Примечание

Если в редакторе М-файлов отсутствует панель инструментов, то следует выбрать в меню **Desktop** пункт **Editor Toolbar**.

Вместо красного кружка может появится серый, свидетельствующий о том, что в программе есть синтаксическая ошибка или в текст программы были внесены изменения и файл не сохранен. В первом случае выводится диалоговое окно **MATLAB Editor** с предупреждением о том, какого характера и в какой строке М-файла ошибка. Во втором случае достаточно сохранить файл. Если каталог с М-файлом не является текущим или путь к нему не указан в путях поиска MATLAB, то появляется диалоговое окно **MATLAB Editor**, переключатели которого позволяют сделать каталог текущим или добавить путь к нему в пути поиска.

При достижении безусловной точки останова выполнение программы всегда приостанавливается. Это в ряде случаев может оказаться неудобным, например, если при отладке цикла `for` требуется приостановить выполнение программы, только если счетчик цикла принимает значение 100. Гораздо лучше использовать условную точку останова, которая сработает только при выполнении некоторого условия. Для размещения условной точки останова в некоторой строке с исполняемыми операторами следует установить в нее курсор и в меню **Debug** выбрать пункт **Set/Modify Conditional Breakpoints**. В появившемся диалоговом окне (рис. 8.10) надо записать логическое выражение, которое будет проверяться до исполнения этой строки и, если оно удовлетворяется, произойдет временное прекращение (приостанов) исполнения кода. Условная точка останова, связанная со строкой текста программы, отмечается желтым кружком на месте знака "-" слева от текста. Можно сделать эту точку останова неактивной, не удаляя ее (она будет игнорироваться), с целью последующей ее активизации. Переключение из активного состояния в неактивное производится выбором в меню **Debug** пункта **Enable/Disable Breakpoint**. При деактивации точки останова кружок перечеркивается. Такой же эффект получается и для безусловных точек останова. Деактивировать условную точку останова можно щелчком левой кнопки мыши по желтому кружку. Учитите, что при щелчке мыши по неактивной точке останова она удаляется, а не активируется. Аналогичные возможности предоставляет контекстное меню, появляющееся при щелчке левой кнопкой мыши в поле редактора с номерами строк файла. В частности, легко превратить безусловную точку останова в условную при помощи пункта меню **Set/Modify Condition**.



**Рис. 8.10.** Диалоговое окно для задания условия останова

При достижении условной или безусловной точки останова вы имеете возможность продолжить отладку одним из способов, которые выбираются в меню **Debug**, или при помощи соответствующих кнопок на панели инструментов.

- **Step** (или <F10>) — пошаговое выполнение одной строки программы и остановка на следующей. Точка останова возникает в следующей строке и уничтожается автоматически после дальнейшего выполнения программы.
- **Step In** (или <F11>) — пошаговое выполнение, включая операторы вызываемых файл-программ и файл-функций.
- **Step Out** (или <Shift>+<F11>) — выполнение функции до конца и возврат в вызываемый М-файл (при условии, что при выполнении не встретится безусловная или условная точка останова).
- **Run (Continue)** (или <F5>) — начало (продолжение) выполнения М-файла до следующей безусловной или условной точки останова.
- **Go Until Cursor** — продолжение исполнения до строки М-файла, в которой находится курсор.
- **Exit Debug Mode** — прекращение процесса отладки.

При этом на строку, в которой выполнение программы временно приостановилось, указывает зеленая стрелка.

Для отмены сразу всех условных и безусловных точек останова служит пункт **Clear Breakpoints in All Files** меню **Debug** либо одноименная кнопка на панели инструментов редактора М-файлов.

Отметим еще один вид условных точек останова, которые не связаны ни со строкой М-файла, ни с каким-либо М-файлом, а определяются рядом исключительных ситуаций. Пусть, например, в процессе вычислений выводится предупреждение, скажем, о делении на ноль, и мы хотим выяснить, в ка-

ком M-файле и по какой причине это происходит, и посмотреть значения соответствующих переменных. Самый удобный способ — автоматически открыть файл в редакторе и поместить временную точку останова в "подозрительную" строку, выполнение которой повлекло данное предупреждение. Для этого, во-первых, убедитесь, что в меню редактора **Debug** рядом с названием пункта **Open m-files when Debugging** установлен флаг (если его нет, то просто выберите этот пункт). Во-вторых, в том же меню перейдите к пункту **Stop if Errors/Warnings**. Появляется многостраничное диалоговое окно **Stop if Errors/Warnings for All Files**, изображенное на рис. 8.11. Каждая его вкладка позволяет задать действия MATLAB при возникновении одной из исключительных ситуаций:

- Errors** — возникает событие, приводящее при обычной работе программы к сообщению об ошибке (например, обращение к неопределенной переменной или функции, или выход за границы массива). Ошибки, возникающие внутри блока обработки исключительных ситуаций `try...catch`, игнорируются;
- Try/Catch Errors** — возникает любая ошибочная ситуация внутри блока `try...catch`;
- Warnings** — программа выполнила действия, сопровождающиеся выводом предупреждения (например, решение системы с плохо обусловленной матрицей);
- Nan or Inf** — управление приостановкой исполнения, если при вычислении арифметических выражений получается `NaN` или `Inf` (например, при делении на ноль).



Рис. 8.11. Диалоговое окно для остановок при исключительных ситуациях

На каждой вкладке находится несколько переключателей. Выбор первого отменяет появление условной точки останова. Второй переключатель как раз и предназначен для автоматического приостановления работы М-файла и открытия его в редакторе (при условии, что в меню **Debug** рядом с названием пункта **Open m-files when Debugging** должен быть флаг). Итак, в зависимости от желаемого вида контроля следует установить второй переключатель на одной или нескольких вкладках и нажать кнопку **OK**. Более того, окно редактора можно закрыть, поскольку при возникновении исключительной ситуации оно откроется и укажет вам на оператор М-файла, послуживший источником ошибки или предупреждения.

### Примечание

Третий переключатель на всех вкладках кроме **Nan or Inf** служит для отслеживания ошибок или сообщений только с определенными идентификаторами. Подробнее об этом написано в справке по окну **Stop if Errors/Warnings for All Files**, которая выводится при нажатии на кнопку **Help**.

Заметим, что выбор в меню **Debug** пункта **Clear Breakpoints in All Files** или нажатие на одноименную кнопку панели инструментов редактора приводит к отмене описанного выше режима.

## Пример диалоговой отладки

Разберем процесс отладки на конкретном примере. При чтении разд. "Сервисные функции для работы со строками" этой главы вы создали функцию `strnumpos` (см. листинг 8.1), предназначенную для нахождения позиций всех цифр, входящих в строку. Результатом выполнения функции является вектор с номерами позиций цифр. Попытаемся теперь упростить ее так, как это сделано в листинге 8.36. Вместо перебора всех цифр мы использовали функцию `isnumeric` для проверки, является ли текущая позиция цифрой, или нет. Новая функция названа `strnumpos1`, из нее исключена проверка входных и выходных аргументов и оставлен только алгоритм поиска для упрощения визуального контроля при отладке.

### Листинг 8.36. Функция `strnumpos1` для диалоговой отладки

```
function pos = strnumpos1(str)
slen = length(str);
digits = 0;
pos = [];
for k = 1:slen
 if isnumeric(str(k))
```

```

digits = digits + 1;
pos(digits) = k;
end
end

```

Создайте файл-программу `check` (листинг 8.37), в которой вызывается функция `strnumpos1`, и запустите ее.

**Листинг 8.37. Файл-программа `check` для отладки функции `strnumpos1`**

```

line = 'В 2003 году Санкт-Петербург - 300 лет'
position = strnumpos1(line)

```

При исполнении М-файла получается заведомо неправильный результат — пустой массив:

```

>> check
line =
В 2003 году Санкт-Петербург - 300 лет
position =
[]

```

Приступая к отладке, целесообразно вначале очистить окно **Command Window** (командой `clc`) и **Workspace**, т. е. удалить все переменные рабочей среды (командой `clear all` или средствами самого браузера переменных). Оставьте видимыми необходимые окна так, чтобы ими было удобно пользоваться. Откройте файлы `check` и `strnumpos1` в редакторе М-файлов, сделав текущей файл-программу `check`. Наиболее удобно встроить окно редактора М-файлов в рабочую среду и расположить его поверх окна **Command History**, как показано на рис. 8.12. Для этого надо воспользоваться кнопкой **Dock Editor** на панели инструментов редактора М-файлов и затем перетащить его за заголовок при нажатой левой кнопке мыши на окно **Command History**. Меню окна редактора М-файлов объединится с меню рабочей среды.

**Примечание**

Можно избрать и другой подход для компоновки окон — извлечь из среды MATLAB браузер переменных и командное окно и разместить необходимые окна, включая редактор М-файлов, на экране без перекрытия так, чтобы ими было удобно пользоваться. Для отделения окон от рабочей среды служит кнопка **UnDock...** на заголовке окна или одноименный пункт меню **Desktop**. Однако в этом случае при открытии других окон, на-

пример, окна редактора массивов или окна редактора графиков придется располагать все окна заново.

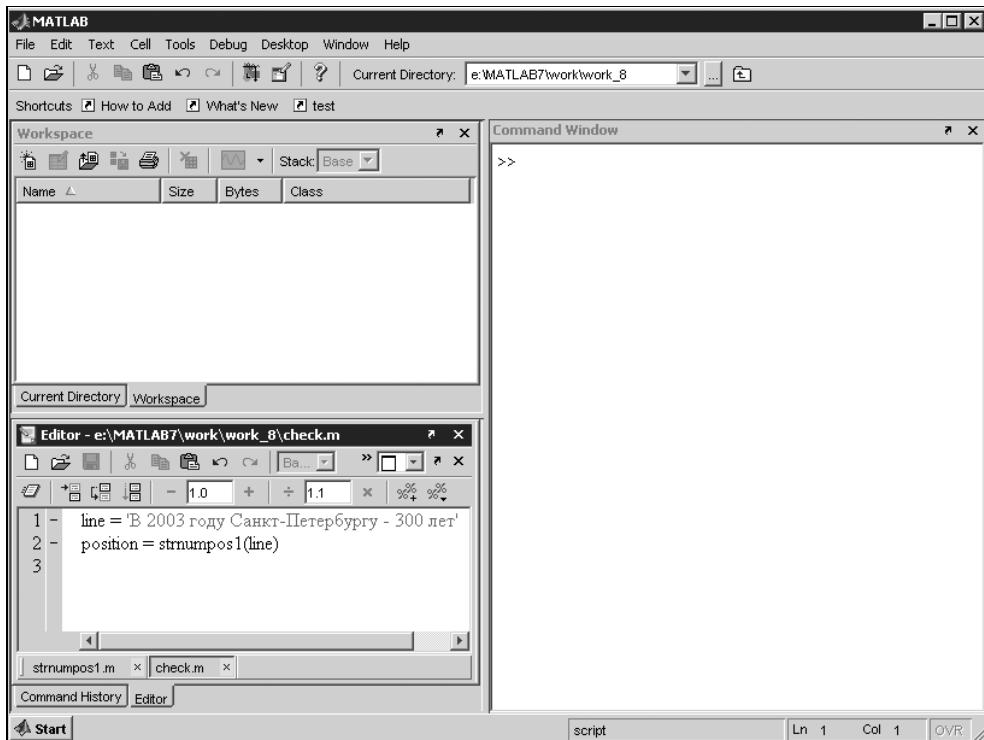


Рис. 8.12. Размещение окон при отладке на экране

Перейдите в окно редактора М-файлов и установите безусловную точку останова на первой команде файл-программы `check`. Для этого щелкните мышкой по знаку "-" справа от номера строки или поместите курсор в первую строку и нажмите `<F12>`. Знак минус изменился на красный кружок. Использование функциональных клавиш существенно повышает эффективность работы, поэтому далее мы будем упоминать нужную клавишу (или комбинацию), а в скобках приводить название соответствующего пункта меню **Debug** или кнопки на панели инструментов редактора М-файлов.

Нажмите `<F5>` (**Run**) для выполнения программы. Происходит переход в режим отладки и выполнение программы до первой точки останова, т. е. до первой строки. На первый взгляд отладчик ничего не сделал, но это не так — в первой строке появилась временная точка останова, о чем свидетельствует знак стрелки, указывающей на первую команду. Этот знак дает

возможность определить место в программе, где выполнение операторов приостановлено. Точно так же происходит и при достижении условной точки останова. Кроме того, изменилось приглашение командной строки в окне **Command Window**, свидетельствующее о проведении диалоговой отладки.

Далее будем выполнять программу в пошаговом режиме. Нажатие на **<F10> (Step)** приводит к выполнению первой команды в файл-программе `check`. Обратите внимание, что в окне браузера переменных рабочей среды **Workspace** появилась переменная `line`, доступная для просмотра. В окне **Command Window** отобразился результат выполненной команды. В конце первой строки файла-программы `check` мы намеренно не поставили точку с запятой (листинг 8.37) для вывода полученного результата в командное окно. Временная точка останова переместилась во вторую строку файла-программы, слева от нее расположена стрелка. Поскольку наша цель — нахождение ошибки в файл-функции `strnumpos1`, то теперь следует пошагово выполнять ее операторы. Нажатие **<F11> (Step In)** влечет автоматическое открытие файла `strnumpos1.m` с нашей файл-функцией в окне редактора и помещение временной точки останова в строке с первым оператором файл-функции.

### Примечание

Разница между **<F10> (Step)** и **<F11> (Step In)** проявляется только в тех строках программы, которые содержат вызов файл-функций или файл-программ. Если в процессе отладки требуется зайти в файл-функцию или файл-программу и пошагово выполнять ее операторы, то следует выбрать **<F11> (Step In)**. С другой стороны, если в строке программы происходит обращение к стандартной файл-функции, например, `fzero`, и ее отладка не входит в ваши планы, то лучше выполнить строку за один шаг при помощи **<F10> (Step)**. Всегда можно отказаться от пошагового выполнения файл-функции, выполнить ее оставшиеся строки и вернуться к оператору, следующему за ее вызовом. Для этого необходимо воспользоваться комбинацией клавиш **<Shift>+<F11> (Step Out)**.

Выполните пошагово часть операторов файла-функции `strnumpos1` до условного оператора `if`, нажимая **<F10>**. Обратите внимание, что все локальные переменные файл-функции и входной аргумент `str` отображаются в окне браузера **Workspace**. В раскрывающемся списке **Stack** на панели инструментов редактора М-файлов или окна **Workspace** установлено имя файл-функции. Этот список содержит сейчас всего три строки: **strnumpos1**, **check** и **Base** и позволяет в любой момент отладки отобразить в окне браузера соответственно: локальные переменные файл-функции `strnumpos1` и ее аргументы (входные и выходные), если они уже созданы; переменные файл-программы `check` или переменные рабочей среды. Напомним, что перемен-

ные файл-программ и рабочий среды являются общими, поэтому выбор **check** приводит к тому же результату, что и **Base**.

### Примечание

В случае рекурсивной файл-функции каждое обращение к ней фиксируется в списке **Stack**, что дает возможность отслеживать значения переменных для вызовов различной вложенности.

За изменением значений переменных в процессе отладки удобно следить в окне редактора массивов **Array Editor**. Откройте окно **Array Editor** со значением **k** при помощи двойного щелчка мыши по имени переменной **k** в окне браузера (или из всплывающего меню, или с помощью кнопки **Open** на панели инструментов окна **Workspace**). Аналогичным образом отобразите содержимое переменной **str** в окне редактора массивов так, как показано на рис. 8.13.

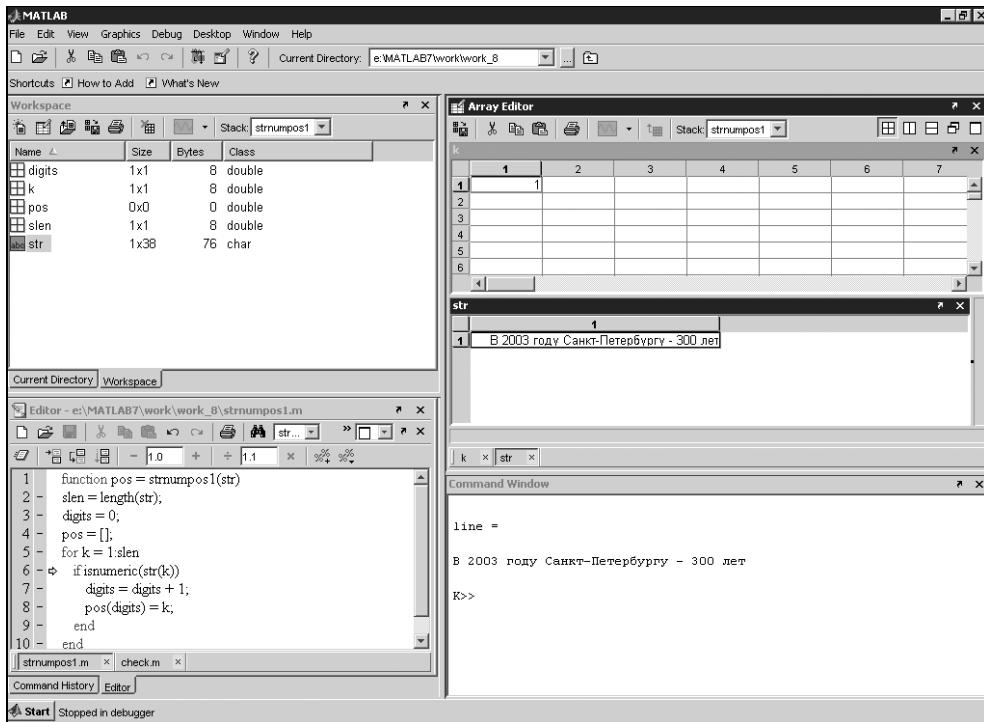


Рис. 8.13. Контроль значений переменных в процессе отладки

Пошагово выполняя файл-функцию `strnumpos1`, убедитесь в том, что для значений 1 и 2 счетчика цикла `k` условие оператора `if` не выполняется, т. е. не происходит перехода к операторам, расположенным внутри тела `if`. Так и должно быть, поскольку первый и второй символы строки `str` не являются цифрами. Когда значение `k` достигает 3, должно получиться верное условие в операторе `if`, т. к. третий символ в строке является цифрой. Однако перехода к блоку внутри тела оператора `if` не происходит. Ошибка локализована и процесс отладки можно пока прервать, выбрав в меню **Debug** пункт **Exit Debug Mode** или нажав одноименную кнопку на панели инструментов редактора М-файлов.

Итак, необходимо выяснить, что же на самом деле проверяет оператор `if`. Внесите небольшие изменения в файл-функцию `strnumpos1`. Перед условным оператором присвойте вспомогательной переменной `temp1` значение `str(k)`, а `temp2` — результат, возвращаемый `isnumeric(temp1)`. Не забудьте сохранить файл `strnumpos1.m`. Теперь мы изберем более быстрый путь отладки программы, установив только одну условную точку останова в строке файла-функции с оператором `if` при достижении счетчиком значения 3. Для этого установите курсор в строку с оператором `if`, выберите в меню **Debug** пункт **Set/Modify Conditional Breakpoints**. В появившемся окне введите: `k==3`. В файл-программе `check` точка останова уже не понадобится и ее можно удалить. Начните отладку снова, запустив файл-программу `check` при помощи <F5> (**Run**). Теперь приостанов произойдет только в функции `strnumpos1` при значении счетчика цикла `k`, равном 3. Проанализируйте значения переменных `temp1` и `temp2`, открыв их для просмотра в **Array Editor**. Значение `temp1` показывает, что третий символ строки `str` на самом деле воспринимается не как число (`double`), а как символ (`char`), вернее массив символов размера 1 на 1. Поэтому функция `isnumeric` возвращает логический ноль, что подтверждается значением переменной `temp2`. Допущена ошибка в используемом типе данных. Прервите отладку и подумайте, как проверить, является ли символ цифрой или нет.

Преобразование строки в число функцией `str2num`, о котором было сказано в этой главе, позволяет корректно произвести нужную проверку. При преобразовании текущего символа строки `str` возможны всего два варианта: число, если текущий символ — цифра, и пустой массив во всех остальных случаях. Следовательно, достаточно проверить равенство нулю длины возвращаемого `str2num` массива (листинг 8.38).

#### Листинг 8.38. Новая функция `strnumpos1` для диалоговой отладки

```
function pos = strnumpos1(str)
slen = length(str);
digits = 0;
pos = [];
```

```
for k = 1:slen
 a = str2num(str(k));
 if length(a) ~= 0
 digits = digits + 1;
 pos(digits) = k;
 end
end
```

Повторите отладочные действия и убедитесь, что происходит правильное заполнение элементов выходного массива. Внесите окончательные изменения в файл-функцию `strnumpos1`, а именно верните комментарии и проверку входных аргументов так, как было сделано в функции `strnumpos` (см. листинг 8.1).

### Примечание

Проверка условия соответствия символа цифре использует особенности пакета MATLAB. Однако в таком случае алгоритм становится непереносимым в другие среды. Поэтому условие проверки лучше записать в форме, приемлемой для любой среды программирования. Например, `str(k) >= '0' & str(k) = < '9'`.

В этом разделе мы на простом примере привели основные приемы отладки программ средствами MATLAB. При самостоятельной работе над заданиями следующего раздела воспользуйтесь отладчиком MATLAB для пошагового выполнения алгоритма.

## Задания для самостоятельной работы

1. Напишите файл-функцию для решения поставленной задачи.
  - Подсчитайте число вхождений подстроки в строку.
  - Найдите количество пробелов в строке.
  - Определите количество цифр в строке.
  - Удалите идущие подряд одинаковые символы в строке.
  - Замените идущие подряд одинаковые символы в строке на один.
  - Стока является предложением, в котором слова разделены пробелами. Переставьте первое и последнее слово.

- Образуйте строку, состоящую из первых букв строк, входящих в массив строк.
  - Выведите номера одинаковых строк в массиве строк.
  - Определите количество символов в каждой строке массива строк без учета пробелов.
  - По заданному массиву строк образуйте новый, исключив повторяющиеся строки.
  - Замените в строке цифры числительными (вместо 1, 2, ... — один, два, ...).
  - Задана строка, содержащая текст и числа. Выделите числа в числовой массив.
2. Напишите файл-функцию для считывания данных из файла в структуру или массив структур с подходящими полями.

|              |            |                     |             |
|--------------|------------|---------------------|-------------|
| Алексеев     | Сергей     | 1980                | 5 4 4 5 3 5 |
| Иванов       | Константин | 1981                | 3 4 3 4 3 5 |
| Петров       | Олег       | 1980                | 5 5 5 4 4 5 |
| <br>21 марта | 2002       | 0.56 0.58 0.49 0.44 |             |
| 23 марта     | 2002       | 0.36 0.32 0.28 0.25 |             |
| 25 марта     | 2002       | 1.62 1.68 1.71 1.91 |             |
| <br>195251   | СПб        | Политехническая     | 29          |
| 195256       | СПб        | Науки               | 49          |
| 195256       | СПб        | Науки               | 24          |

Результаты наблюдений

Time= 0.0 0.1 0.2 0.3 0.4 0.5 0.6

Mass=

2.1 2.3 2.3 1.9 1.8 2.4 2.9  
0.8 0.7 0.5 1.1 3.2 0.3 0.4

|          |         |           |
|----------|---------|-----------|
| Алексеев | Иван    | 121-22-04 |
| Сидоров  | Николай | 101-21-99 |
| Тимофеев | Сергей  | 570-00-03 |

(номера телефонов должны быть записаны в поля структур как целые числа).

3. Считайте матрицы и векторы из файла в подходящие по размеру массивы. Обратите внимание, что в файлах содержится рядом две или три матрицы или вектора, их следует занести в разные массивы.

0.1 0.2 0.3                  9.91

1.9 0.4 0.1                  8.01

4.7 5.1 3.9                  7.16

1 2 3 4                  99 80

5 6 7 8                  33 21

15 90

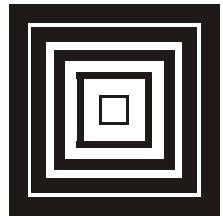
1 2 3 4                  100

6 7 8 9                  0.1 0.2 0.3 0.4                  200

0.5 0.6 0.7 0.8                  300

4. Проведите диалоговую отладку файл-функции `allmin` с рекурсивной подфункцией из листинга 8.37, если условие попадания точки минимума в интервал сформулировать без использования погрешности  $\epsilon$ . Для исключения "зацикливания" алгоритма введите условие, ограничивающее количество найденных точек числом 100.

5. Напишите файл-функцию для поиска всех корней функции  $f(x)$  на заданном отрезке с использованием стандартной функции `fzero`.



## Глава 9

# Дескрипторная графика

Визуализация результатов связана с управлением видом получающихся графиков. Хорошо написанное приложение само формирует окончательный вид графических результатов и не требует от пользователя внесения каких-либо изменений, например, в редакторе графиков, который описан в главе 4. Более того, в процессе работы приложения может возникнуть необходимость изменить, например, толщину линии, вывести текст в графическое окно или удалить поверхность. В этом случае редактор графиков оказывается бесполезным.

MATLAB предоставляет в распоряжение программиста так называемую *дескрипторную* или *управляемую* *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 и изменять их по своему усмотрению. Средства управления графическими объектами, описанные в данной главе, потребуются при разработке приложений MATLAB с графическим интерфейсом.

Созданию приложений с графическим интерфейсом посвящены главы 10—13.

## Основы дескрипторной графики

Поскольку MATLAB является объектно-ориентированной системой, то все элементы (графическое окно, оси, линии, поверхности, текстовые области и т. д.) являются *объектами*. Объекты расположены в определенной иерархической структуре, которую мы подробно обсудим ниже. Проще говоря, графическому окну может принадлежать одна или несколько пар осей со своими объектами: линиями, поверхностями или текстом. То есть линия, поверхность или текст не могут просто быть выведены в графическое ок-

но — в нем обязательно должны находиться оси, которым принадлежат эти объекты.

Каждый объект имеет обширный набор свойств, изменение значений которых позволяет добиться требуемого вида объекта. В главе 4 рассмотрен процесс задания свойств объектов в редакторе графиков, который состоит из двух этапов.

1. Выбор нужного объекта из списка в окне **Plot Browser** или указание его щелчком мыши (при этом объект становится текущим).
2. Установка нужных свойств при помощи элементов управления окна редактора графиков, соответствующего текущему объекту.

При написании собственных приложений, осуществляющих отображение результатов в графическом виде, необходимо знать доступные свойства каждого объекта и уметь изменять их значения в программе. В этом разделе на ряде примеров мы рассмотрим принципы управления основными свойствами графических объектов, а детальное описание их иерархической структуры, свойств и точная терминология приведены в разд. "Графические объекты" этой главы.

## Свойства графических объектов

Функции `set` и `get` позволяют получить и установить свойства любого из объектов в определенные значения. Применение данных функций состоит, по существу, из тех же этапов, что и задание значений в редакторе графиков, 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. В данном случае в результате работы `plot` создались три графических объекта: графическое окно **Figure 1**, оси и линия 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. Имеются три стандартных функции, возвращаемые значения которых 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` открывает доступ к свойствам *текущего* окна, осей или объекта. В данном случае есть только одно графическое окно, оно и является текущим. Единственные оси в графическом окне также являются текущими. Про использование `gco` для определения текущего объекта сказано ниже.

## Свойства осей

Выведите в командное окно свойства осей, содержащих график функции, который был построен в предыдущем разделе, для чего выполните команду:

```
>> get(gca)
```

В командном окне отображается таблица свойств и их значений. Свойств достаточно много, среди них есть очевидные, а назначение некоторых на первый взгляд кажется непонятным. Свойства осей условно можно разделить на две группы — общие свойства и свойства каждой из осей  $x$ ,  $y$  или  $z$ . Название свойств второй группы начинается с соответствующей буквы  $x$ ,  $y$  или  $z$ . Таблицы 9.1 и 9.2 содержат основные свойства, которые часто применяются при создании приложений.

Функция `get` допускает обращение к ней с двумя входными 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
```

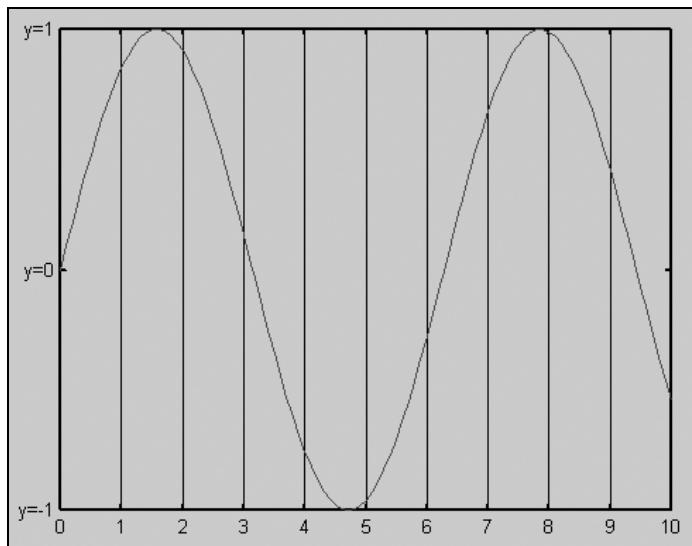
*Таблица 9.1. Свойства, отвечающие за общий вид осей*

| Название свойства  | Описание                                                     | Значения                                                                                                                                                           |
|--------------------|--------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Box                | Заключение осей в прямоугольную рамку                        | 'on' (по умолчанию) 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). По умолчанию цвет белый |
| FontAngle          | Наклон шрифта разметки осей                                  | 'normal' (по умолчанию) или 'italic'                                                                                                                               |
| FontName           | Название шрифта                                              | Строка с названием шрифта, например 'Courier'                                                                                                                      |
| FontSize           | Размер шрифта (по умолчанию в пунктах, 1 пункт = 1/72 дюйма) | Целое число (по умолчанию 12)                                                                                                                                      |
| FontWeight         | Толщина шрифта                                               | 'normal' (по умолчанию), 'bold', 'light', или 'demi'                                                                                                               |
| GridLineStyle      | Стиль линий сетки                                            | '-', '--', ':' (по умолчанию) '-.' или 'none'                                                                                                                      |
| LineWidth          | Толшина линий осей                                           | Значение в пунктах (по умолчанию 0.5)                                                                                                                              |
| Visible            | Отображение осей                                             | 'on' (по умолчанию оси видны), 'off'                                                                                                                               |
| DataAspectRatio    | Масштаб осей                                                 | Вектор из трех элементов, задающий относительный масштаб по каждой из осей                                                                                         |
| PlotBoxAspectRatio | Размеры осей                                                 | Вектор из трех элементов, задающий относительные размеры каждой из осей                                                                                            |

**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 I) |
| XDir              | Направление оси                          | 'normal' или 'reverse' (обратное)                                                                                                             |
| XGrid             | Сетка, перпендикулярная оси              | 'on' или 'off'                                                                                                                                |
| XAxisLocation     | Расположение оси (для оси z отсутствует) | 'top' или 'bottom' ('right' или 'left' для оси y)                                                                                             |
| XLim              | Пределы изменения переменной             | Вектор из двух компонент, 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<br>{'zero'; 'one'; 'three'; 'five'}    |

Команда `set` позволяет установить каждому свойству 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

Для этого расположите командное окно MATLAB и графическое окно на экране так, чтобы они не перекрывали друг друга, и при выполнении приведенных ниже команд следите за состоянием осей. Следующие команды задают толщину линий осей, координаты и обозначения разметки, наносят линии сетки и устанавливают цвет фона осей, совпадающий с цветом графического окна:

```
>> 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' соответствует автоматическому подбору 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.

Итак, 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 возможные значения не отличаются от приведенных в инспекторе свойств, который вы использовали при чтении главы 4 (см. разд. "Цветовое оформление, разметка и сетка").

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 и текстовых объектах подробно написано ниже (см. разд. "Указатели на объекты" и "Текстовые объекты" данной главы).

Свойства осей `XLim`, `YLim`, `ZLim`, `DataAspectRatio`, `PlotBoxAspectRatio`, отвечающие за пределы, масштаб и относительные размеры осей, взаимосвязаны. Например, относительные размеры осей зависят от пределов и масштаба, более того, при заданных `DataAspectRatio`, `XLim`, `YLim` и, возможно, `ZLim` значение свойства `PlotBoxAspectRatio` игнорируется. Таблица связей между этими свойствами и сопутствующими к ним `XLimMode`, `YLimMode`, `ZLimMode`, `DataAspectRatioMode` и `PlotBoxAspectRatioMode` приведена в справочной системе MATLAB (см. информацию о свойстве `DataAspectRatio` осей, например в разд. **MATLAB: Functions – Alphabetical List** (гиперссылка **Axes Properties**)).

В этом разделе мы рассмотрели только наиболее часто используемые свойства осей, заметим, что всего их более 100. Ряд свойств отвечают за расположение осей в пределах графического окна. Этот вопрос мы обсудим ниже (см. разд. "Управление положением осей" данной главы).

Графические объекты MATLAB обладают достаточно большим набором свойств, описание которых слишком объемно. Цель этой главы — дать представление об организации графических объектов, их типах и основных приемах работы с ними. Поэтому мы иногда будем только упоминать о тех или иных свойствах, одновременно давая ссылки на соответствующие разделы интерактивной справочной системы. Одним из основных разделов является **MATLAB: Graphics: Handle Graphics Objects**. Кроме того, быстрый доступ к назначению всех свойств графических объектов производится из браузера свойств, размещенного в разд. **MATLAB: Handle Graphics Property Browser**.

Использование браузера свойств графических объектов описано в разд. "Получение информации о свойствах графических объектов" этой главы.

## Свойства линий и поверхностей

Стандартная функция `gca`, задаваемая в качестве аргумента `set` и `get`, позволяет получить доступ к свойствам текущих осей. Однако для обращения

к текущей линии графика или поверхности 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)
```

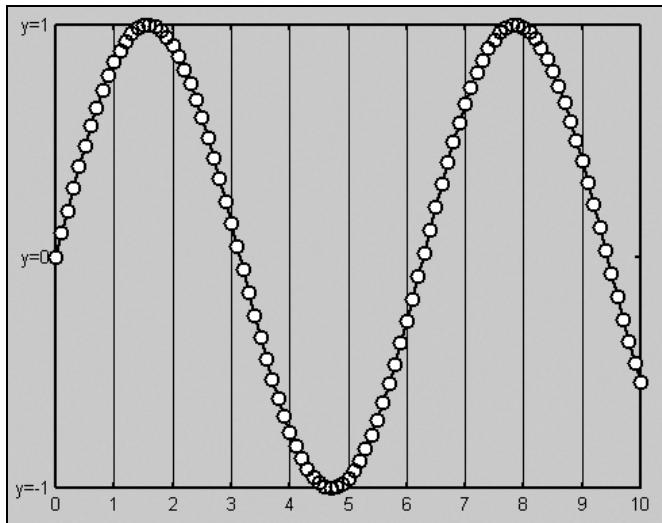
Многие из этих свойств вы изменяли в редакторе при прочтении главы 4. Таблица 9.3 содержит наиболее часто употребляемые свойства линий.

**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                            | '-' (по умолчанию), '--', ':', '-.' или 'none'                                                                                                |
| LineWidth         | Толщина линии в пунктах (1 пункт = 1/72 дюйма) | Положительное число                                                                                                                           |
| Marker            | Тип маркера                                    | Одно из стандартных обозначений, например 'o', 's' (см. табл. 3.1)                                                                            |
| MarkerEdgeColor   | Цвет границы маркера                           | Такие же, как у <code>Color</code>                                                                                                            |
| MarkerFaceColor   | Цвет маркера                                   | Такие же, как у <code>Color</code>                                                                                                            |
| MarkerSize        | Размер маркера в пунктах                       | Положительное число                                                                                                                           |

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)
```



**Рис. 9.2.** Изменение свойств линии

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 и разд. MATLAB: Handle Graphics Property Browser справочной системы MATLAB).

### Примечание

Функции высокоуровневой графики для построения поверхностей (`mesh`, `surf`, `surfl` и др.) основаны на низкоуровневой функции `surface`, которая позволяет создать поверхность с любыми заданными свойствами.

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, щелкнув 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 и т. д.), 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 объекта не являющимся текущими. Эта задача легко решается при помощи указателей.

## 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 любой текущий объект соответственно. 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`, `surf` 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 указателей на линии графика. Первый его элемент есть указатель на линию, отвечающую первой паре входных аргументов, второй элемент — указатель на вторую линию и т. д.

## Изменение свойств линий и осей

Оформите последовательность команд для построения графиков двух функций, приведенную в листинге 9.1, в виде файл-программы в М-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

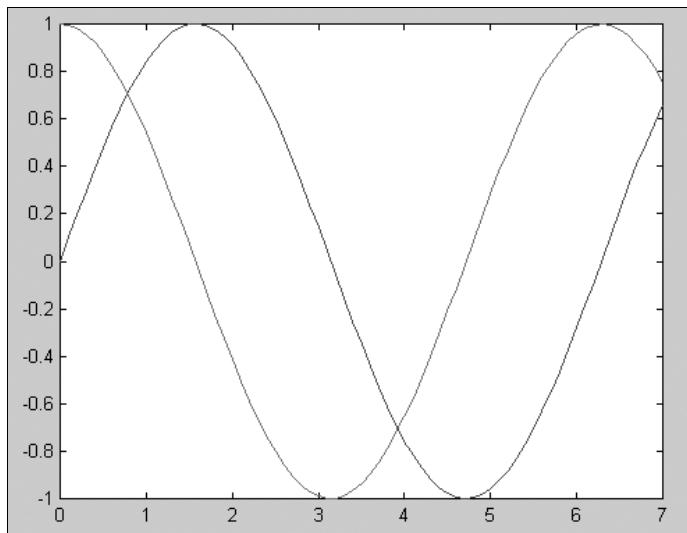
hLines(1) содержит указатель на первую линию ( $\sin(t)$ )

hLines(2) содержит указатель на вторую линию ( $\cos(t)$ )

```
%}
```

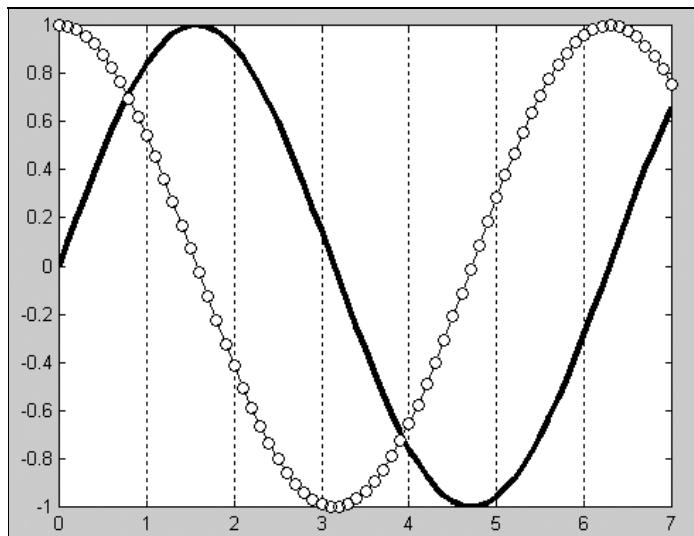
```
hLines = plot(t,x,t,y);
```

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.



**Рис. 9.3.** Исходный вид графиков

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.



**Рис. 9.4.** Измененные графики (`set` и указатели на объекты)

Для этого следует использовать функцию `set` с указателем на объект и парой 'Свойство', значение (листинг 9.2).

#### Листинг 9.2. Изменение свойств линий и осей при помощи указателей

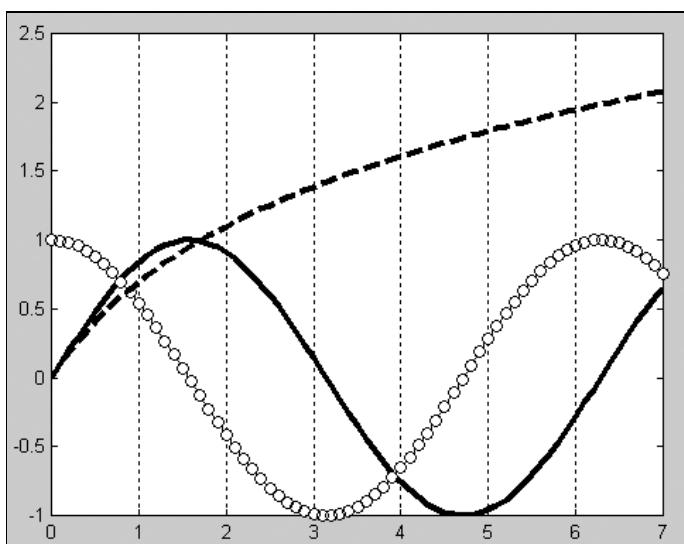
```
% Установка свойства XGrid осей с указателем hAx в 'on'
set(hAx, 'XGrid', 'on')
% Задание цвета первой линии (графика синуса) с указателем hLines(1)
set(hLines(1), 'Color', 'k')
% Задание толщины первой линии (графика синуса) с указателем hLines(1)
set(hLines(1), 'LineWidth', 3)
% Задание цвета второй линии (графика косинуса) с указателем hLines(2)
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')
```

Первым входным аргументом функции `set` может быть вектор указателей на некоторые объекты, в этом случае изменяются свойства всех заданных объектов. 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.

В любой момент можно вернуться к значениям свойств объекта, установленным по умолчанию. Для этого служит функция `reset`, в качестве входного аргумента которой задается указатель на объект.

## Добавление линий графиков

Команда `plot` осуществляет вывод графика на текущие оси, поэтому для добавления линий следует сначала сделать требуемые оси текущими, задав входным аргументом функции `axes` указатель на них, выполнить `hold on`, 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.



**Рис. 9.5.** Добавление линии графика на заданные оси

Очевидно, что файл-программу, осуществляющую построение графиков синуса и косинуса и изменение их свойств (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);
```

Заметьте, что можно было не создавать отдельную переменную для хранения указателя на новую линию, а дополнить вектор указателей `hLines` при помощи оператора `hLines(3) = plot(t, z)`. Хранение указателей на однотипные объекты в массивах оказывается удобным при одновременном изменении их свойств. При таком подходе команда `set(hLines, 'Color', 'r')` изменяет цвет всех трех линий. Еще один пример работы с массивом указателей приведен ниже.

## 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. Сначала надо сде-

лать 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`.

## Влияние команд `hold`, `cla`, `clf` и `reset` на свойства окна и осей

Остановимся более подробно на влиянии команды `hold` на свойства осей и графического окна. Графическое окно и оси имеют свойство `NextPlot`, определяющее способ вывода графических объектов при помощи высокочувствительных функций (`plot`, `surf` и т. д.), которое может принимать одно из следующих значений: '`'add'`', '`'replace'`' или '`'replacechildren'`'.

В зависимости от значения свойства `NextPlot` текущего графического окна при новом графическом выводе возможна одна из трех ситуаций:

- '`'add'`' (по умолчанию) — для вывода используется текущее графическое окно;
- '`'replace'`' — удаляются все объекты, принадлежащие графическому окну, а все его свойства (кроме `Position`, которое отвечает за положение окна на экране монитора) принимают значения, установленные по умолчанию;
- '`'replacechildren'`' — удаляются все объекты, принадлежащие графическому окну, но его свойства не изменяются.

Перед выводом графических объектов высокочувствительные функции проверяют значение свойства `NextPlot` текущих осей, которое определяет способ вывода:

- '`'add'`' — графический объект добавляется на оси;
- '`'replace'`' (по умолчанию) — перед размещением нового объекта удаляются все старые, принадлежащие оси, а свойства осей (кроме `Position`, которое отвечает за положение осей в пределах графического окна) принимают значения, установленные по умолчанию;
- '`'replacechildren'`' — то же, что и `replace`, но без изменения свойств осей.

Очистка осей командой `cla` приводит к установке значения '`'replacechildren'`' свойству `NextPlot` осей, а последовательное выполнение `cla` и `reset` — значения '`'replace'`'. То же самое верно и для `clf` в применении к графическому окну. Команда `hold on` устанавливает значение

свойств `NextPlot` текущего графического окна и осей в '`'add'`', а `hold off` изменяет только значение `NextPlot` текущих осей на '`'replace'`'.

## Получение информации о свойствах графических объектов

Подробное описание всех свойств графических объектов приведено в нескольких разделах справочной системы MATLAB. Во-первых, структура графических объектов и приемы работы с ними разобраны в разд. **MATLAB: Graphics: Handle Graphics Objects**. Во-вторых, список функций дескрипторной графики с гиперссылками на соответствующие страницы приведен в разд. **MATLAB: Functions -- Categorical List: Graphics: Handle Graphics**. Кроме того, возможен быстрый доступ к информации о свойствах графических объектов из браузера свойств, размещенного в разд. **MATLAB: Handle Graphics Property Browser**. Обсудим, как пользоваться этим разделом.

При выборе разд. **MATLAB: Handle Graphics Property Browser** в правом окне справочной системы MATLAB появляются вкладки с названиями объектов, которые являются гиперссылками.

Вкладки **Figure** и **Axes** не требуют пояснений — они служат для доступа к свойствам графического окна и осей. Щелчок мышью по этим гиперссылкам приводит к появлению списка всех свойств объекта и их назначения в нижней части окна, причем она разделяется на два столбика. В левом столбике с полосой скроллинга находятся гиперссылки с названиями свойств объекта, при переходе по гиперссылке в правом столбике отображается описание выбранного свойства и его возможные значения. Установленные по умолчанию значения заключены в фигурные скобки. Кроме того, вверху окна показывается место выбранного объекта в иерархии графических объектов MATLAB.

Вкладка **UI Objects** предназначена для элементов графического интерфейса пользователя, которому посвящены следующие несколько глав.

Выбор вкладки **Core Objects** приводит к появлению части иерархической структуры объектов, в которую включены базовые объекты: линия (**Line**), текстовый объект (**Text**), полигональный (**Patch**) и т. д. (работа с базовыми объектами рассмотрена в разд. *"Графические объекты"*, а с текстовым объектом — в разд. *"Текстовые объекты"* данной главы).

Большинство высокоуровневых графических функций создают рисованные объекты (**Plot Objects**), используя базовые объекты. Щелчок мышью по однотипной вкладке в окне браузера свойств позволяет отобразить названия рисованных объектов (гиперссылки) и их место в иерархии. Заметьте, что нарисованная функцией `plot` линия в действительности является объектом `Lineseries` с более широким набором свойств, чем у базового объекта `Line`.

Аналогично, функции `loglog`, `semilogx`, `semilogy` и `plot3` так же приводят к созданию объекта `Lineseries`. Функции для построения столбчатых диаграмм `bar` и `barh` рисуют объект `Barseries`, а функция `area` — объект `Areaseries`. Контурные графики, полученные при помощи `contour`, это объекты `Contourgroup`. Результатом работы функций `mesh`, `meshc`, `surf` и `surfc` является объект `Surfaceplot`.

Таблица соответствий высокоуровневых графических функций и создаваемых ими объектов приведена в разделе справочной системы **MATLAB: Graphics: Handle Graphics Objects: Plot Objects**.

Для того чтобы выяснить, какой именно объект создает высокоуровневая графическая функция, достаточно перейти к ее описанию в справочной системе, например, в разд. **MATLAB: Functions -- Categorical List: Graphics** или в разд. **MATLAB: Functions -- Alphabetical List**. Итак, если требуется получить свойства объектов, порожденных большинством высокоуровневых графических функций, то следует обратиться к содержимому вкладки **Plot Objects**.

Графические объекты MATLAB могут быть сгруппированы для выполнения однотипных операций или доступа к ним как к одному объекту. Вкладка **Group Objects** предназначена для просмотра свойств двух возможных типов сгруппированных объектов: `hggroup` и `hgtransform` (работа со сгруппированными объектами описана в разд. "Объекты-группы `hggroup` и `hgtransform`" данной главы)..

Последняя вкладка **Annotation Objects** служит для просмотра свойств поясняющих объектов: стрелок, линий, надписей и т. п., которые вы создавали при помощи интерактивной среды для построения графиков при чтении главы 4 (создание поясняющих объектов объяснено в разд. "Размещение текста, линий и стрелок в графическом окне" данной главы).

Еще одну возможность для получения информации о свойствах и их допустимых значениях предоставляют функции `get` и `set`. Функция `get`, как было описано выше, позволяет вывести установленные значения всех свойств или только одного свойства. Список всех 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 (`hL` — указатель на некоторую линию):

```
>> set(hL, 'Marker')
[+ | o | * | . | x | square | diamond | v | ^ | > | < | pentagram |
hexagram | {none}]
```

## Использование указателей, примеры

Наличие большого числа графических объектов требует умения оперировать их свойствами. Эффективной оказывается запись указателей на однотипные объекты в числовой массив. Разберем обработку графических объектов на примере файл-функции с интерфейсом из командной строки, предназначеннной для исследования различных математических функций. 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 среди всех введенных функций. По завершении работы (пользователь ввел `end` на запрос программы) графическое окно закрывается. Предусмотрите также 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`. Запомните номер функции с максимальным значением и само значение в некоторых переменных. 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 для исследования функций на отрезке [a, b]
% Использование: maxfun(a, b)

% Создание графического окна (оно становится текущим)
hF = figure;
% Создание осей в текущем графическом окне
hAx = axes;
% Установка NextPlot для добавления графиков на оси
set(hAx, 'NextPlot', 'add')
% Задание вектора значений аргумента
x = a:(b - a)/30:b;
str = ''; % инициализация строки запроса ввода пользователя
```

```
funcount = 0; % инициализация счетчика введенных 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
 % Формирование команды для вычисления массива значений
 eval(strcat('y =',str,';'));
 funcount = funcount + 1; % увеличение счетчика функций
 % Оси hAx должны быть текущими для вывода графика
 axes(hAx)
 % 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', 'MarkerEdgeColor', 'k', ...
 'MarkerFaceColor', 'w', 'Color', 'k')
 if funcount == 1 % пользователь ввел одну функцию
 maxval = max(y); % находим ее максимальное значение
 Nmaxval = 1; % пока первая функция имеет максимальное значение
 % Рисуем ее график жирной линией
 set(hFuns(funcount), 'LineWidth', 3)
 else % пользователь ввел две или более функций
 % Удаление маркеров с графика предыдущей функции
 set(hFuns(funcount - 1), 'Marker', 'none')
 M = max(y); % находим максимальное значение новой функции
 if M > maxval
 % Новая функция принимает самое большое значение среди
 % всех функций, введенных пользователем, поэтому
 % ее график должен рисоваться жирной линией
 end
 end
 end
 end
end
```

```
set(hFuns(funcount), 'LineWidth', 3)
% Линия графика функции, которая ранее имела максимальное
% значение, теперь не должна быть жирной
set(hFuns(Nmaxval), 'LineWidth', 1)
% Запоминаем новое максимальное значение и номер функции
maxval = M;
Nmaxval = funcount;
else % значения новой функции не превосходят значений предыдущих
 set(hFuns(funcount), 'LineWidth', 1)
end
end
end
end
% Выход из цикла while производится, когда пользователь ввел end
delete(hF) % удаление графического окна с экрана
```

Сделайте работу файл-функции maxfun более надежной, применив конструкцию `try...catch` для обработки исключительной ситуации, которая может возникнуть при неверном вводе формулы для исследуемой функции.

## Задание свойств в аргументах графических функций

Применение указателей на графические объекты полезно при изменении свойств объекта в ходе работы программы. Создание объекта с определенными свойствами может быть осуществлено и при помощи высокогорловневых функций. В качестве дополнительных входных аргументов задаются пары 'свойство', значение. Например, последовательность команд листинга 9.5 приводит к графику, изображенному на рис. 9.4, который ранее мы получали 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);
figure
axes('XGrid', 'on', 'NextPlot', 'add')
```

```
plot(t, x, 'Color', 'k', 'LineWidth', 3)
plot(t, y, 'Color', 'k', 'Marker', 'o', 'MarkerFaceColor', 'w',...
 'MarkerFaceColor', 'w', 'MarkerEdgeColor', 'k');
```

Обратите внимание, что в случае указания нескольких пар векторов со значениями аргумента и функции происходит одновременное изменение свойств всех линий, например:

```
>> plot(t, x, t, y, 'Color', 'r', 'LineWidth', 3, 'Marker', 'o')
```

Если дальнейшее изменение свойств объектов не предполагается, то можно обойтись без выходных аргументов, в которые заносятся указатели на создаваемые объекты. Разумеется, если сохранить указатели, то их можно воздействовать впоследствии для изменения свойств этих объектов. Аналогичным образом задаются свойства графического окна (в списке входных аргументов функции `figure`), осей (в списке входных аргументов функции `axes`) и поверхностей (в списке входных аргументов `surf`, `mesh` и других функций). Следующий раздел посвящен созданию графических окон и осей с заданными размерами и положением на экране, а также выводу на них текстовой информации.

## Размещение окон, осей и текста

Вывод результатов работы требует предварительного создания графических окон и осей с заданными размерами и положением. Кроме того, хорошо написанное приложение не должно быть привязано к какому-либо разрешению, установленному на мониторе. Следующие разделы посвящены описанию тех свойств графических окон и осей, которые оказываются полезными при организации вывода графиков и текста в различные окна и оси.

## Расположение графических окон и осей

Рассмотрим свойства графического окна, определяющие его вид, положение и размеры на экране монитора, а также свойства осей, которые позволяют не только расположить оси в любом месте графического окна, но и сделать согласованными изменения размеров осей и графического окна.

## 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 `height` определяют, соответственно, ширину и высоту области окна.

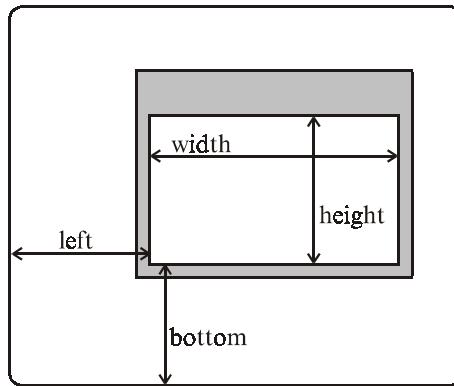


Рис. 9.6. Положение графического окна на экране

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 четыре пикселя.

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

```
>> hF = figure('Color', 'w', 'MenuBar', 'none')
```

и получите его размеры, указав входными аргументами `get` указатель на текущее окно и `Position`:

```
>> p1 = get(hF, 'Position')
p1 =
 232 258 560 420
```

Единицами измерения по умолчанию являются пиксели. 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 разре-

шением  $1024 \times 768$ , если ваш монитор настроен на другой режим, то результаты могут отличаться):

```
>> p2 = get(hF, 'Position')
p2 =
 1 1 1024 749
```

Итак, сейчас ширина рабочей области совпадает с шириной экрана, а  $768 - 749 = 19$  пикселов отводится под заголовок графического окна **Figure 1**, помещенный вверху окна. 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 разд. "Объект `Root`" этой главы).

```
>> s = get(0, 'ScreenSize')
s =
 1 1 1024 768
```

Единицы измерения размера экрана зависят от значения свойства `Units` объекта `Root`, по умолчанию установлены пиксели. 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
hF = figure('Position',[left bottom width height], 'MenuBar', '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 пикселям
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],...
 'MenuBar', 'none', 'Color', 'w');
hFigs(2) = figure('Position', [left bottom2 width height],...
 'MenuBar', '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, причем каждое графическое окно может содержать несколько осей. Функция `subplot` предлагает самый простой способ расположения 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('MenuBar', 'none', 'Color', 'w');
>> hAx = axes('Box', 'on');
>> a = get(hAx, 'Position')
a =
 0.1300 0.1100 0.7750 0.8150
```

Итак, положение рамки осей задается вектором из четырех элементов

`[left bottom width height]`

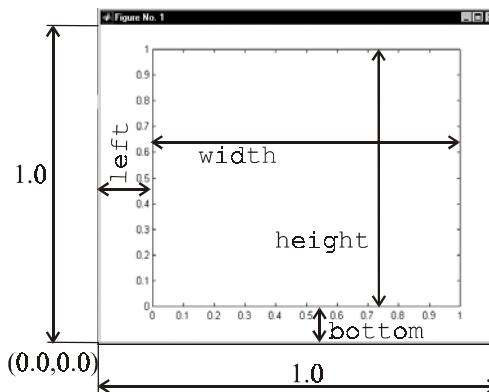
где `left` равно расстоянию от левой границы рабочей области графического окна до рамки, `bottom` — от нижней границы рабочей области графического окна до рамки, а `width` и `height`, соответственно, определяют ширину и высоту рамки осей (рис. 9.7 и 9.8 для случая двумерных и трехмерных осей). По умолчанию используются нормализованные единицы для измерения расстояний, т. е. ширина и высота рабочей области графического окна принимаются 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.

Если при помощи свойств `DataAspectRatioMode` или `PlotBoxAspectRatioMode` задана пропорция между длинами осей (т. е. размеры осей не являются независимыми), то оси будут занимать максимально возможную площадь прямоугольной области ширины `width` и высоты `height`.

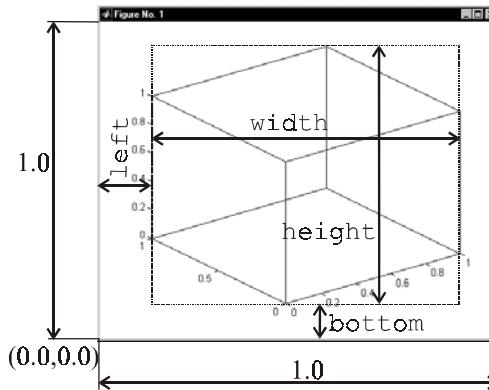
При использовании свойства `Position` не всегда удается расположить оси так, чтобы хватило места для координат разметки, заголовка и подписей к осям. Более гибкое управление положением осей позволяют осуществить три свойства: `ActivePositionProperty`, `OuterPosition` и `TightInset`.

Значением свойства `OuterPosition`, так же как и `Position`, является вектор из четырех элементов. В данном случае он задает левый нижний угол, ширину и высоту воображаемого прямоугольника, который заключает в себя оси и поля вокруг них с заголовком и подписями к осям. Расположение осей

при помощи `OuterPosition` гарантирует, что эти элементы всегда будут видны.



**Рис. 9.7.** Расположение двумерных осей в графическом окне



**Рис. 9.8.** Расположение трехмерных осей в графическом окне

Свойство `TightInset` служит для точного определения размеров полей вокруг рамки, которые нужны для координат разметки осей, заголовка и подписей к осям. Это свойство доступно только для чтения, его значением является вектор из четырех элементов: `[left bottom right top]`, где `left` — ширина добавленного поля слева от рамки осей, а `bottom`, `right` и `top` — снизу, справа и сверху соответственно. Значения свойств `OuterPosition` и `TightInset`, так же как и `Position`, измеряются в единицах, указанных в `Units`.

В качестве примера создайте оси

```
>> hA=axes('Position',[0.2 0.2 0.6 0.6])
```

и выведите значение свойства TightInset

```
>> t=get(hA,'TightInset')
```

```
t =
```

```
0.0393 0.0405 0.0089 0.0190
```

Затем добавьте подпись к оси ординат и снова обратитесь к TightInset

```
>> ylabel('ось y')
```

```
>> t=get(hA,'TightInset')
```

```
t =
```

```
0.0857 0.0405 0.0089 0.0190
```

Для подписи по оси  $y$  дополнительно понадобилось поле шириной  $0.0857 - 0.0393 = 0.0464$  в нормализованных единицах измерения.

Напишите файл-функцию axes3, которая создает три пары осей на заданном графическом окне так, как изображено на рис. 9.9, и возвращает указатели на созданные оси. Входным аргументом axes3 должен быть указатель на графическое окно.

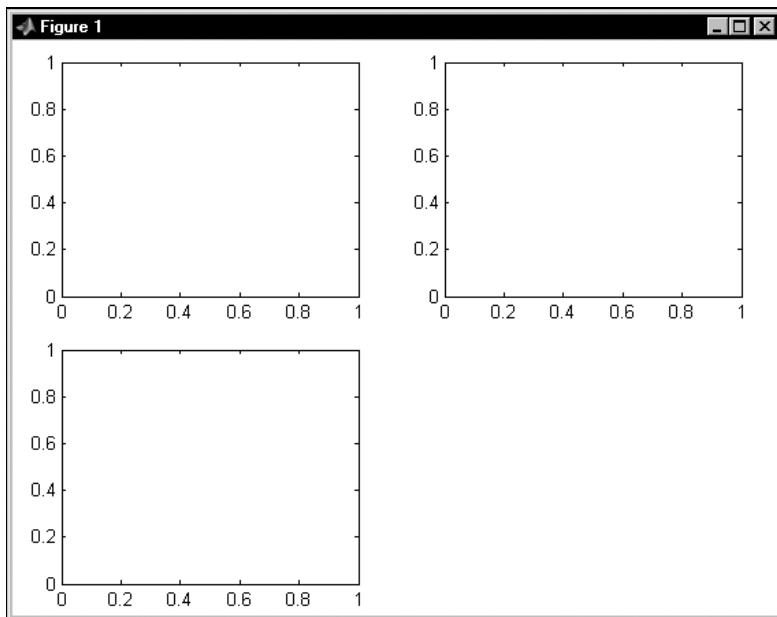


Рис. 9.9. Пример расположения осей

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)

figure(hF) % Окно с указателем hF становится текущим
% Создание осей в текущем окне
hAx(1) = axes('OuterPosition', [0 0.5 0.5 0.5], 'Box', 'on');
hAx(2) = axes('OuterPosition', [0.5 0.5 0.5 0.5], 'Box', 'on');
hAx(3) = axes('OuterPosition', [0 0 0.5 0.5], '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));
```

При изменении размеров графического окна размеры осей также изменяются. При этом MATLAB пересчитывает их, ориентируясь на свойство Position либо OuterPosition. Выбор определяется значением свойства ActivePositionProperty: 'position' либо 'outerposition' (по умолчанию) соответственно. Если ActivePositionProperty установлено в 'outerposition', то заголовок, подписи к осям и разметка не выйдут за пределы графического окна при уменьшении его размеров. Однако если вы создали оси и задали их свойству Position некоторое значение, то ActivePositionProperty автоматически приобретет значение 'position'.

## 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 минимума и корня. Программу оформите в виде файл-функции zeroandmin. Имя исследуемой функции (или указатель на нее, или inline-функция) 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. "Файл-функции с одним входным аргументом" главы 5 и "Исследование функций" главы 6).

### Листинг 9.9. Файл-функция `zeroandmin`, предназначенная для исследования функций

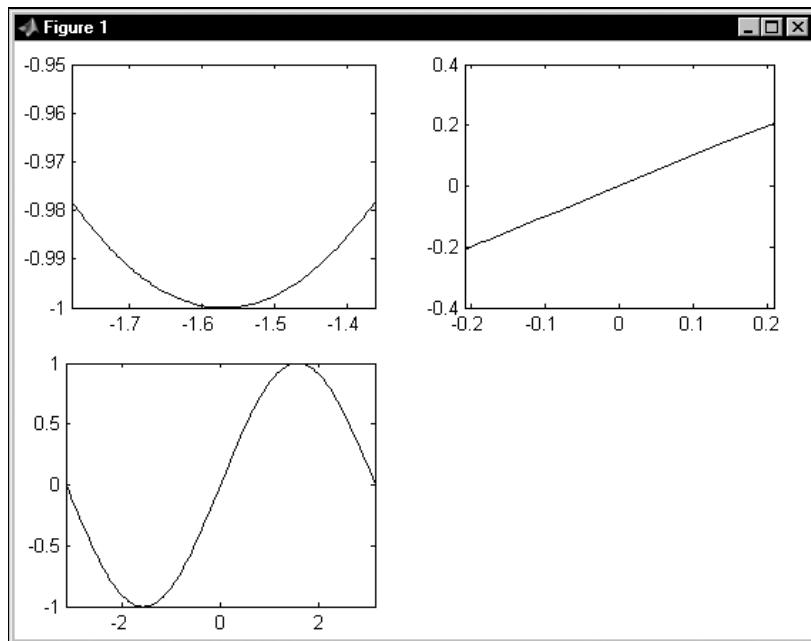
```
function zeroandmin(fun, a, b)
% Исследование поведения функций вблизи корня и
% локального минимума на отрезке [a, b]
% Использование zeroandmin(fun, a, b)

% 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(fun, [a b], options);
% Поиск локального минимума функции на [a, b]
xmin = fminbnd(fun, a, b, options);
% Создание графического окна
hF = figure('MenuBar', '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(fun, [xmin - delta xmin + delta])
axes(hAx(2)) % оси с указателем hAx(2) стали текущими
% Вывод графика функции вблизи ее нуля на верхние правые оси
fplot(fun, [zero - delta zero + delta])
axes(hAx(3)) % оси с указателем hAx(3) стали текущими
% Вывод графика функции на [a, b] на нижние оси
fplot(fun, [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 графиками, которые построены в окрестностях точек, интересующих пользователя.



**Рис. 9.10.** Исследование функции  $\sin(x)$   
при помощи `zeroandmin`

## Вывод 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 и формулы в произвольном месте графического окна.

## Текстовые объекты

Мы уже упоминали, что функции высокогоуровневой графики `title`, `xlabel`, `ylabel` и `zlabel` служат не только для добавления заголовка и подписей к осям, но и возвращают указатели на соответствующие текстовые объекты. Следовательно, можно управлять свойствами полученного объекта для достижения желаемого результата. Команда

```
hTxt = title('График функции sin(\itx)')
```

не только добавляет заголовок над областью текущего графика, 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')
```

Текстовый объект может быть заключен в прямоугольную рамку, причем цвета рамки и фона выбираются независимо. Поля между текстом и рамкой также не являются фиксированными. Свойства текстовых объектов, отвечающие за характеристики шрифта, цветовое оформление и поля вокруг текста и их возможные значения, приведены в табл. 9.4.

**Таблица 9.4.** Общие свойства шрифта

| Название свойства       | Описание        | Значения                                                                                                                                                            |
|-------------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>String</code>     | Текст           | Строка в апострофах или строковая переменная или массив ячеек, состоящий из строк, для получения многострочного текста. Может содержать текст в формате TeX и LaTeX |
| <code>FontAngle</code>  | Наклон шрифта   | 'normal' — прямой (по умолчанию), 'italic' — курсив                                                                                                                 |
| <code>FontName</code>   | Название шрифта | Строка с названием шрифта, установленного на компьютере, например, 'Courier'                                                                                        |
| <code>FontSize</code>   | Размер шрифта   | Целое число                                                                                                                                                         |
| <code>FontWeight</code> | Толщина шрифта  | 'normal' (по умолчанию), 'bold', 'light' или 'demi'                                                                                                                 |

Таблица 9.4 (окончание)

| Название свойства | Описание                                                            | Значения                                                                                                                                   |
|-------------------|---------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Interpreter       | Использование формата TeX для отображения греческих букв и символов | 'tex' — использовать формат TeX (по умолчанию), 'latex' — использовать формат LaTeX, 'none' — не использовать                              |
| Color             | Цвет шрифта                                                         | Вектор из трех элементов, дающий цвет в формате RGB, например [1 1 1] или один из определенных цветов: 'r', 'g' и т. д. (см. приложение 1) |
| BackgroundColor   | Цвет фона                                                           | Вектор из трех элементов, дающий цвет в формате RGB, например [1 1 1] или один из определенных цветов: 'r', 'g' и т. д. (см. приложение 1) |
| LineStyle         | Стиль линий прямоугольной рамки                                     | '-' (по умолчанию), '--', ':' , '-' или 'none'                                                                                             |
| LineWidth         | Толщина линий прямоугольной рамки в пунктах (1 пункт = 1/72 дюйма)  | Положительное число                                                                                                                        |
| EdgeColor         | Цвет рамки                                                          | Вектор из трех элементов, дающий цвет в формате RGB, например [1 1 1] или один из определенных цветов: 'r', 'g' и т. д. (см. приложение 1) |
| Margin            | Ширина полей в пунктах (1 пункт = 1/72 дюйма)                       | Положительное число                                                                                                                        |

Команда `title` использует функцию `text`, которая создает текстовый объект, принадлежащий текущим осям. Создание и размещение текстовых объектов на текущих осях может быть выполнено при помощи функции `text`. Входными аргументами `text` в простейшем случае являются координаты, определяющие положение текста и строки, а в качестве необязательного вы-

ходного 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` (ширина и высота области осей принимаются равными единице, а начало координат помещается в левый нижний угол области окна). Кроме того, удобно 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.

### Примечание

Текстовый объект принадлежит осям согласно иерархии графических объектов, однако он не обязательно находится внутри осей. Выведите, например, значение свойства `Position` для заголовка или подписей к осям и сравните координаты текстового объекта с пределами осей.

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

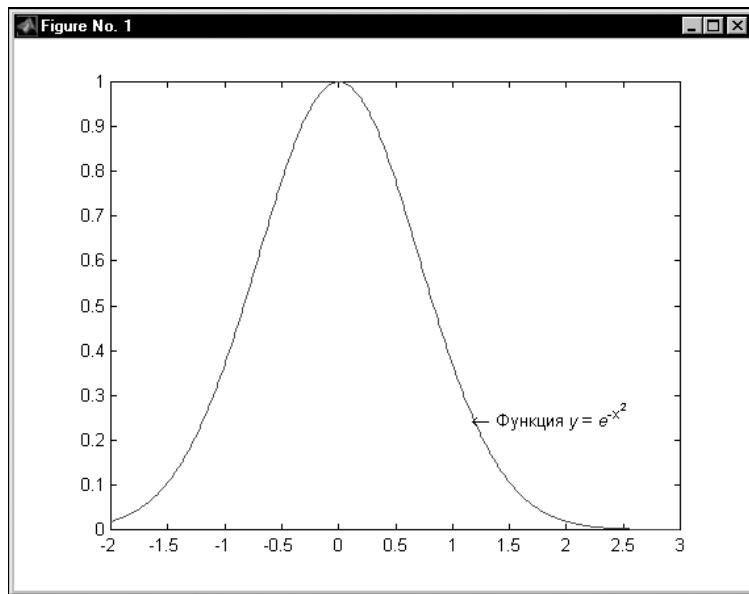
```
% Создание графического окна и осей
hF = figure('MenuBar', 'none', 'Color', 'w');
hAx = axes;
% Генерация векторов значений аргумента и функции
x = -2:0.01:3;
y = exp(-x.^2);
plot(x,y)
% Задание координат положения текстового объекта
Xtext = 1.17;
```

```

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                                                                                                         |
| 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 точки минимума, но и ординату. Используйте обращение к функции `fminbnd`, позволяющее одновременно получить требуемое значение функции в точке локального минимума.

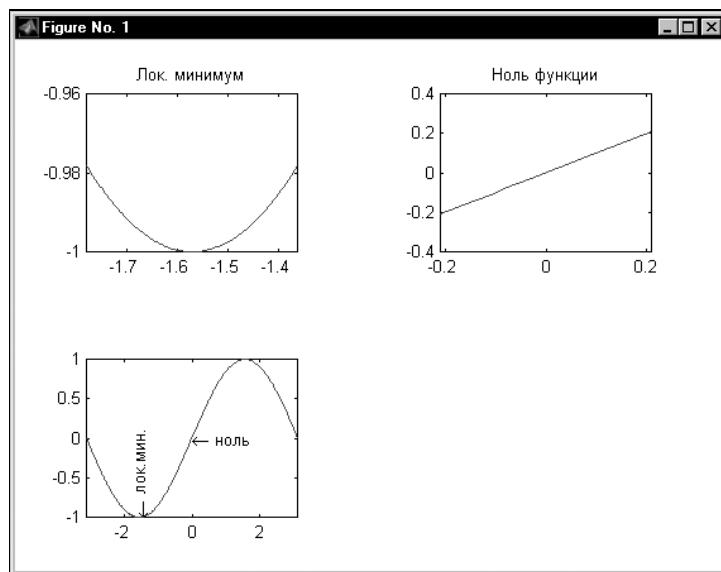
**Листинг 9.11. Изменение файл-функции `zeroandmin`**

...

% Поиск локального минимума функции на [a, b]

```
% Получение значения функции и аргумента локального минимума
[xmin, fmin] = fminbnd(fun, 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' , [xmin 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 результа-

тов. Согласно иерархии объектов, текст может принадлежать только осям, поэтому возникает вопрос, как поместить текст в произвольное место графического окна.

## Линии, стрелки и текстовые пояснения

При чтении главы 3 вы освоили специальные инструменты **Plotting Tools** интерактивной среды для рисования стрелок, линий и добавления текстовой информации в графическое окно. Данные элементы размещаются на специальных невидимых осях — слое пояснений (**Annotation layer**), совпадающих по размерам с графическим окном. Рассмотрим теперь, как добиться аналогичного результата в файл-программе или файл-функции.

Все поясняющие объекты (**Annotation objects**) создаются при помощи одной функции `annotation`. Первым ее входным аргументом является строка с названием объекта: '`line`' — линия (**Annotation Line**), '`arrow`' — стрелка (**Arrow**), '`doublearrow`' — двунаправленная стрелка (**Doublearrow**), '`textarrow`' — стрелка с текстовой надписью (**Textarrow**), '`textbox`' — текстовая надпись (**TextBox**), '`ellipse`' — эллипс (**Ellipse**) или '`rectangle`' — прямоугольник (**Rectangle**). В скобках приведены названия этих объектов в справочной системе. Остальные входные аргументы определяются типом выбранного объекта. Причем для одномерных объектов, т. е. линий и разного рода стрелок, указываются два вектора `x` и `y` с координатами начальной `x(1)`, `y(1)` и конечной `x(2)`, `y(2)` точек. Двумерные объекты (текстовая надпись, эллипс или прямоугольник) требуют задания вектора из четырех элементов в качестве второго входного аргумента функции `annotation`: координат левого нижнего угла, ширины и высоты объекта. В случае эллипса определяется прямоугольная рамка, в которую он вписан. Примите во внимание, что значения координат должны быть в нормализованных единицах графического окна, т. е. ширина и высота окна равны единице, а начало координат находится в левом нижнем углу окна (см. разд. "Управление положением осей" данной главы).

Выходным аргументом функции `annotation` является указатель на созданный объект, который необходим для изменения значений его свойств при помощи `set`.

Рассмотрим свойства поясняющих объектов на примере стрелки с текстовой надписью (объекта **Textarrow**), указывающей на начало координат графического окна. Сначала создадим графическое окно и сам объект, сохранив указатель на него в переменной `hTA`

```
>> figure
>> hTA = annotation('textarrow', [0.5 0], [0.5 0]);
```

Так же как и в случае текстовых объектов, свойство `String` позволяет определить надпись к стрелке.

Значением свойства `String` может быть строка или массив ячеек строк для многострочного текста:

```
>> txt{1} = 'Начало координат';
>> txt{2} = 'графического окна'
>> set(hTA, 'String', txt)
```

Форма остряя стрелки выбирается с привлечением свойства `HeadStyle`, которое может принимать различные значения. По умолчанию свойство `HeadStyle` установлено в значение '`vback2`'. Для получения, например, обычного треугольного закрашенного остряя следует выбрать значение '`plain`':

```
>> set(hTA, 'HeadStyle', 'plain')
```

Сокращения для всех типов стрелок приведены в справочной системе — обратитесь к соответствующей таблице при помощи браузера свойств графических объектов.

Ширина и длина остряя стрелки определяются значениями свойств `HeadWidth` и `HeadLength` соответственно. Толщина линии зависит от значения свойства `LineWidth`. Значения этих свойств задаются в пунктах (1 пункт = 1/72 дюйма).

Разберем более подробно цветовое оформление объекта `Textarrow`, поскольку оно несколько отличается по сравнению с другими поясняющими объектами. Для выбора одинакового цвета стрелки и текста достаточно одного свойства `Color`:

```
>> set(hTA, 'Color', 'm')
```

В данный момент прямоугольная рамка, заключающая в себя текст, не видна, поскольку свойство `TextEdgeColor` по умолчанию установлено в '`none`'. Цвет рамки устанавливается независимо:

```
>> set(hTA, 'TextEdgeColor', 'b')
```

Аналогичным образом можно выбрать и цвет шрифта:

```
>> set(hTA, 'TextColor', 'g')
```

Однако последующее изменение значения свойства `Color` приведет к одновременному изменению значений свойств `TextColor` и `TextEdgeColor` (если только оно не '`none`'). Поэтому при выборе цветов для элементов объекта `Textarrow` сначала устанавливается свойство `Color`, а затем выбирается цвет текста и рамки.

Отметим еще ряд свойств, полезных при создании объекта `Textarrow`: `TextBackgroundColor` — цвет заливки внутренности прямоугольной рамки, `TextLineWidth` — толщина линий рамки в пунктах, `TextRotation` — угол поворота в градусах, `TextMargin` — поля вокруг текста в пикселях. Смысл

свойств `FontName`, `FontSize`, `FontWeight`, `HorizontalAlignment`, `VerticalAlignment` тот же самый, что и у текстовых объектов (см. разд. "Текстовые объекты" данной главы).

Положение объекта `Textarrow` может быть изменено установкой свойств `x` и `y`. Их значениями являются векторы с координатами начальной и конечной точки.

Мы достаточно подробно рассмотрели свойства объекта `Textarrow`, поскольку он состоит из стрелки и текстовой надписи. Среди его свойств много общих с остальными поясняющими объектами. Теперь при помощи браузера свойств графических объектов легко самостоятельно научиться управлять видом других объектов слоя **Annotation layer**.

Функция `annotation`, так же как и многие функции высокогоуровневой графики, позволяет указывать значения свойств поясняющих объектов при их создании (см. разд. "Задание свойств в аргументах графических функций" данной главы).

Например, следующее обращение к `annotation` приводит к созданию прямоугольника шириной 0.5 и высотой 0.6, левая нижняя точка которого имеет координаты (0.3, 0.2) (в нормализованных единицах графического окна). Получающийся прямоугольник с красными границами заливается зеленым цветом.

```
>> hR = annotation('rectangle', [0.3 0.2 0.5 0.6], 'EdgeColor', 'r',
...
'FaceColor', 'g')
```

### Примечание

Свойства поясняющего объекта, созданного функцией `annotation`, могут быть изменены из контекстного меню, которое активизируется щелчком левой кнопки мыши по объекту.

В качестве упражнения завершите работу над файл-функцией `zeroandmin`, снабдив ее операторами, которые выводят в правый нижний угол графического окна значения переменных `fmin`, `xmin` и `zero` в три строки (листинги 9.9 и 9.11). Воспользуйтесь поясняющим объектом `Textbox`. Трехбумный блок приведен в листинге 9.12.

#### Листинг 9.12. Вывод текста в графическое окно

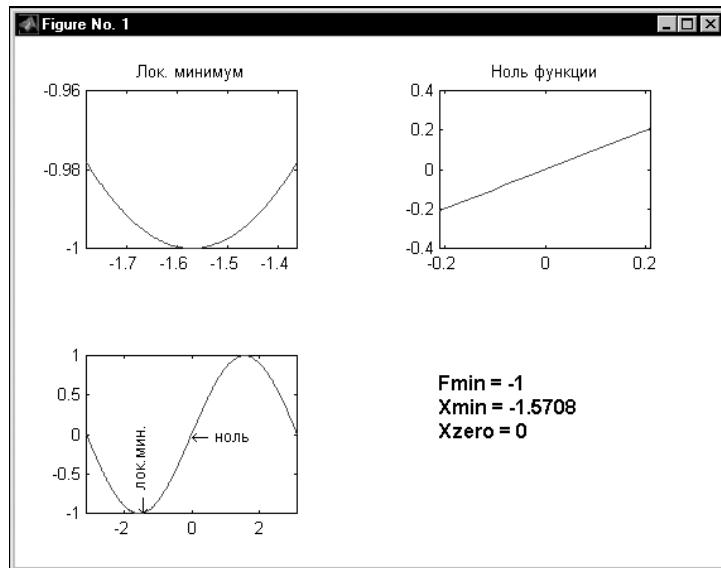
```
% Формирование массива ячеек строк
strmas{1} = ['Fmin = ', num2str(fmin)];
strmas{2} = ['Xmin = ', num2str(xmin)];
strmas{3} = ['Xzero = ', num2str(zero)];
```

```
% Создание объекта Textbox без рамки в правом нижнем углу окна
hTB=annotation('textbox',[0.6, 0.1, 0.4, 0.3], 'LineStyle', 'none');

% Вывод текстовой информации
set(hTB, 'String', strmas)

% Установка свойств шрифта
set(hTB, 'FontSize', 12, 'FontWeight', 'bold')
```

Окончательный вариант файл-функции zeroandmin позволяет получить не только графическую информацию о поведении функции вблизи интересующих точек, но и соответствующие числовые значения (рис. 9.14).



**Рис. 9.14.** Вывод текста в графическое окно

Для вывода текста в любое место графического окна можно использовать также следующий прием. Добавляются невидимые оси, равные по размеру графическому окну, со свойством `Visible`, установленным в `'off'`. Данные оси существуют как объект, но не отображаются в графическом окне. При помощи функции `text` создается текстовый объект, принадлежащий невидимым осям. Он 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 (листинг 9.13).

### Листинг 9.13. Вывод текста в графическое окно

```
% Создание вспомогательных невидимых осей, равных по размерам
% графическому окну
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')
```

При наличии некоторого опыта работы в LaTeX вы можете выводить достаточно сложные формулы в графическое окно. Один пример разобран в следующем разделе.

## Вывод математических формул в формате LaTeX

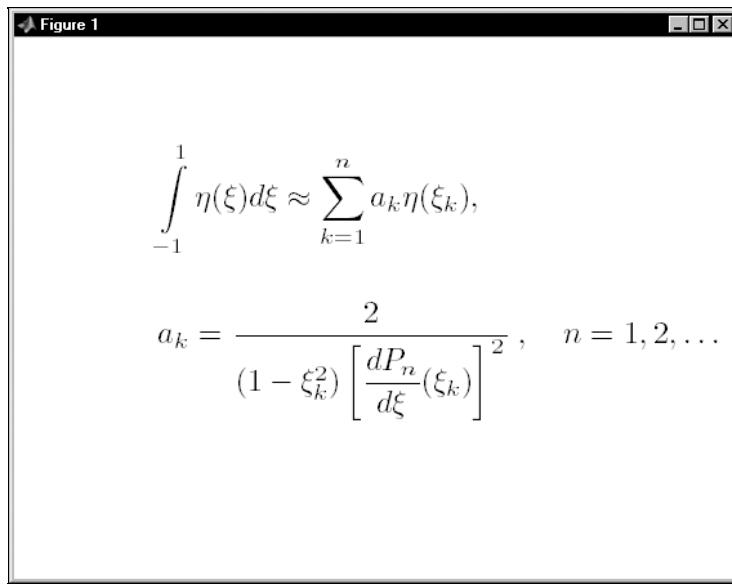
Интерпретаторы TeX и LaTeX, входящие в состав MATLAB, позволяют использовать эти стандарты для получения хорошо выглядящих математических формул. При чтении главы 3, посвященной высокоуровневой графике, вы уже набирали формулы в формате TeX, прибегая только к самым простым его командам для изменения шрифта или записи индексов (см. табл. 3.3—3.5).

Текст с командами TeX записывается в строку, или строковую переменную и указывается во входном аргументе высокоровневых функций `title`, `xlabel` и др., или в качестве значения свойства `String` текстового, или поясняющих объектов `TextBox` и `TextArrow`. При этом происходит автоматическая интерпретация команд TeX, поскольку свойство `Interpreter` данных объектов по умолчанию имеет значение '`tex`'.

Стандарт LaTeX предоставляет более широкие возможности для набора формул. Мы не будем описывать здесь основные возможности LaTeX, а продемонстрируем на примере получение графического окна с формулами, например, изображенного на рис. 9.15.

Последовательность операторов, приводящая к размещению формул в графическом окне, приведена в листинге 9.14. Отличия по сравнению с TeX состоят в том, что мы заключаем каждую формулу в два знака доллара, устанавливаем свойство `Interpreter` в значение '`latex`' и используем определенные в LaTeX команды. Поскольку строки с формулами достаточно длинные, то для наглядности мы ввели переменные, содержащие части строки, которые склеили затем при задании значения свойству `String`.

Кроме того, мы выводим текстовые объекты на невидимые оси, как предлагалось делать в конце предыдущего раздела.



**Рис. 9.15.** Вывод формул в графическое окно

#### Листинг 9.14. Вывод формул в формате LaTeX

```
% Создание графического окна белого цвета без меню
figure('Color', 'w', 'MenuBar', 'none')

% Создание невидимых осей, совпадающих по размеру с графическим окном
axes('Visible', 'off', 'Position', [0 0 1 1])

% Создание первого текстового объекта и указание его координат
hForm1 = text('Position', [0.2 0.7]);

% Ввод первой формулы в формате LaTeX в две строковые переменные
s1 = '$$\\int\\limits_{-1}^1\\eta(\\xi)d\\xi\\approx$';
s11 = '$\\sum\\limits_{k=1}^na_k\\eta(\\xi_k)$';

% Включение интерпретатора LaTeX, задание значения свойству String
% и размера шрифта для первого текстового объекта
set(hForm1, 'Interpreter', 'latex', 'String', [s1 s11], 'FontSize', 18)

% Создание второго текстового объекта и указание его координат
hForm2 = text('Position', [0.2 0.4]);
```

```
% Ввод второй формулы в формате LaTeX в три строковые переменные
s2 = '$$a_k = \frac{2}{(1-\xi_k^2)}\left[\begin{array}{c}
s21 = '\displaystyle\frac{d P_n}{d\xi}' \\
s22 = '(\xi_k)\right]^2\right]; \quad n = 1, 2, \dots';
% Включение интерпретатора LaTeX, задание значения свойству String
% и размера шрифта для второго текстового объекта
set(hForm2, 'Interpreter', 'latex', 'String', [s2 s21 s22], 'FontSize', 18)
```

В качестве упражнения добейтесь аналогичного результата с использованием поясняющего объекта **Textbox**.

## Графические объекты

При чтении предыдущих разделов этой главы вы познакомились с принципами работы с графическими объектами на примере окон, осей, линий, поверхностей, текстовых и поясняющих объектов. Кроме того, нам понадобился сам корневой объект **Root** для получения информации о текущем разрешении монитора. Многообразие графических объектов MATLAB не исчерпывается перечисленными типами объектов и включает в себя ряд других. Среди всех графических объектов выделяются **базовые (Core)**: оси (**Axes**), рисунки (**Image**), источники света (**Light**), линии (**Line**), поверхности (**Surface**), полигональные объекты (**Patch**) и прямоугольники (**Rectangle**). Базовые объекты являются основой для создания более сложных **композитных объектов (Composite objects)**.

Практически все графические объекты, с которыми вы работали при чтении предыдущих разделов, — композитные. Многие высокоуровневые графические функции группируют базовые объекты для получения требуемого результата в виде рисованных объектов (**Plot objects**). Этот вопрос мы уже затрагивали при обсуждении работы с браузером графических объектов (см. разд. *"Получение информации о свойствах графических объектов" данной главы*).

Рассмотренные в предыдущем разделе поясняющие объекты (**Annotation objects**) так же являются композитными. Свойства рисованных и поясняющих объектов позволяют изменять вид составляющих их базовых объектов. Например, двунаправленная стрелка **Doublearrow**, созданная функцией **annotation**

```
hA = annotation('doublearrow', [0 1], [0 1])
```

в действительности образована пятью базовыми объектами — тремя линиями (**Line**) и двумя полигональными объектами (**Patch**). Свойства **Head1Style**, **Head1Length**, **Head1Width**, **Head2Style**, **Head2Length** и **Head2Width**

объекта `Doublearrow` служат для изменения вида и размеров каждого острия стрелки, т. е. базового полигонального объекта (`Patch`). Свойства `LineStyle` и `LineWidth` объекта `Doublearrow` предназначены для управления типом и толщиной одного из базовых объектов `Line`, образующих стрелку. Остальные два базовых объекта `Line` являются невидимыми — они расположены в начальной и конечной точках стрелки.

Другой пример — отображение линий уровня функции двух переменных при помощи `contour` — приведен в справочной системе. Соответствующий рисованный объект `Contourseries` состоит из полигональных объектов (см. раздел справочной системы **MATLAB:Graphics: Handle Graphics Objects: Plot Objects**).

Объекты могут быть сгруппированы при помощи специальных функций. Такие объекты-группы (*Group objects*) позволяют обращаться к ним как к одному объекту, что в ряде случаев оказывается очень удобным (см. разд. “*Объекты-группы hggroup и hgtransform*” этой главы).

Как правило, всюду дальше мы будем называть базовые и различные композитные объекты просто объектами, если это не приведет к неоднозначности.

Сейчас мы более подробно обсудим иерархию и свойства объектов MATLAB за исключением элементов управления (кнопок, переключателей и т. п.), которые используются при написании приложений с графическим интерфейсом.

Написанию приложений с графическим интерфейсом посвящены главы 10—13.

## Иерархия объектов

Достаточно полная структура объектов, за исключением элементов управления приложением с графическим интерфейсом, приведена на рис. 9.16.

Для освоения материала необходимы два понятия из области объектно-ориентированного программирования — *предок* и *потомок*. Объект самого верхнего уровня `Root` не имеет предков, а его потомками являются графические окна — объекты `Figure`. Свойства `Root` позволяют произвести некоторые глобальные установки MATLAB. Объект `Figure` представляет собой отдельное графическое окно. У `Root` может быть много потомков `Figure`, т. е. в MATLAB может быть открыто одновременно произвольное число графических окон. Оси являются потомком графического окна и предком для базовых, рисованных и группированных объектов. Невидимые оси слоя примечаний так же есть потомок графического окна и предок для поясняющих объектов. Свойства `Parent` и `Children` каждого объекта содержат указатели на его предков и потомков соответственно. Поскольку у объекта `Root` нет предка, то значением `Parent` является пустой массив, а у объектов самого низкого уровня отсутствуют потомки, следовательно, значением их

свойства `Children` так же является пустой массив. При наличии у объекта нескольких потомков свойство `Children` содержит вектор указателей на них. Указатели на объекты могут быть скрытыми, подробнее об этом сказано ниже (см. разд. "Управление объектами, копирование, поиск, скрытые указатели" данной главы).

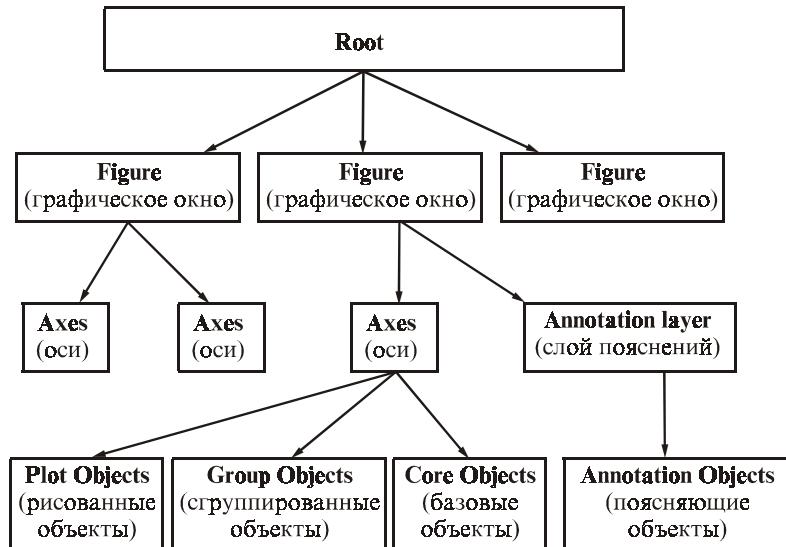


Рис. 9.16. Иерархия графических объектов MATLAB

## Объект `Root`

Корневой объект `Root` автоматически создается при запуске MATLAB, указатель на `Root` всегда равен нулю. Он предназначен для установки общих свойств, характеризующих работу пакета. Например, задание формата вывода результатов вычислений в командное окно может быть осуществлено не только при помощи команды `format` с соответствующим аргументом, но и установкой свойству `Format` объекта `Root` значений: `'short'`, `'shortE'` (по умолчанию), `'long'`, `'longE'` и др., а свойству `FormatSpacing` значения `'compact'` для вывода результатов на каждой строке вместо `'loose'`, принятого по умолчанию.

Свойства `ScreenDepth` и `ScreenSize` позволяют получить информацию о цветовой палитре монитора (разрядности цвета) и размере экрана в единицах, определяемых значением свойства `Units`, которое по умолчанию равно `'pixels'`. Размер экрана содержится в векторе из четырех элементов с коор-

динатами левого нижнего угла монитора, его шириной и высотой. Поскольку по умолчанию единицами измерения являются пиксели, то `ScreenSize` возвращает текущее разрешение монитора. Мы использовали это свойство объекта `Root` при размещении двух графических окон, делящих экран монитора на равные части (см. листинг 9.6). Отметим, что свойство `ScreenSize` доступно только для чтения, т. е. вы не можете изменять разрешение монитора из собственных приложений.

При чтении разд. "Рекурсивные функции" главы 8 вы выполняли команду `get(0, 'RecursionLimit')` для получения максимально возможной глубины рекурсивных вызовов (500 по умолчанию) и `set(0, 'RecursionLimit', 700)` для ее увеличения, т. е. прибегали к свойству `RecursionLimit` объекта `Root`.

Указатель на текущее графическое окно является значением свойства `CurrentFigure`. 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` (графическое окно)

Потомками `Root` являются графические окна — объекты `Figure`, которые создаются функцией `figure` или автоматически при использовании высокоравневых графических функций `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:

DefaultНазваниеОбъектаНазваниеСвойства

НазваниеОбъекта в данном случае графического окна есть **Figure**, а Название свойства — некоторое свойство объекта `Figure`. Например, значение `[0.8 0.8 0.8]` свойства `DefaultFigureColor` (в формате RGB) свидетельствует о том, что по умолчанию графическое окно имеет серый цвет:

```
>> get(0, 'DefaultFigureColor')
ans =
 0.8000 0.8000 0.8000
```

Изменение цвета производится функцией `set`:

```
>> set(0, 'DefaultFigureColor', 'w')
```

Проверьте, что теперь создаваемые графические окна по умолчанию будут белого цвета

```
>> figure
```

Однако свойства, указанные во входных аргументах функции `figure`, имеют больший приоритет по сравнению с определенными по умолчанию:

```
>> figure('Color', 'k')
```

### Примечание

Аналогичным образом можно установить собственные значения свойств всех графических объектов (кроме Plot Objects), которые будут использоваться по умолчанию при их создании. Определение свойств, например, объекта `Line` может быть произведено как с уровня корневого объекта, так и с уровня текущего графического окна или осей. Предпочтение отдается установкам, сделанным на самом низком уровне в иерархии графических объектов.

Определению свойств графических объектов, применяющихся по умолчанию, посвящен раздел справочной системы **MATLAB: Graphics: Handle Graphics Objects: Setting Default Property Values**.

Приведем некоторые свойства графических окон, которые оказываются полезными при организации вывода графических результатов (свойства `Position`, `Color`, `MenuBar` рассмотрены в разд. "Управление положением графических окон" этой главы).

При создании окна его заголовок состоит из слова `Figure` и порядкового номера. Привлечение свойств `NumberTitle` и `Name` позволяет создать графическое окно с желаемым заголовком:

```
>> hF = figure('MenuBar', 'none', 'NumberTitle', 'off',...
 'Name', 'Результаты работы');
```

Графические окна допускают изменение размеров при помощи мыши. Это не всегда удобно, поскольку иногда окно должно быть фиксированного размера. За возможность изменения размера окна отвечает свойство `Resize`, которое по умолчанию имеет значение '`on`'. Для запрета на изменение размера следует установить его в '`off`'.

В иерархии объектов потомками графических окон являются оси и слой пояснений (невидимые оси). Свойствам осей, а также поясняющих объектов было уделено достаточно внимания в этой главе. Также мы привели основные свойства рисованных объектов — линий и поверхностей, которые создаются высокографическими функциями типа `line`, `surf`, `mesh` и др., и свойства базового объекта `Text`. Перейдем теперь к более детальному описанию

свойств потомков осей и приемам работы с ними, демонстрируя их на простых примерах.

## Базовые объекты (Core Objects)

Рассмотрим работу с базовыми объектами на примере линий, прямоугольников, полигональных объектов и источников света. Текстовые объекты мы уже использовали, например, для размещения комментариев на осях графика функции (см. разд. "Текстовые объекты" данной главы).

### Объекты Rectangle и Line, блок-схемы и диаграммы

Хотя название базового объекта `Rectangle` переводится как прямоугольник, он допускает скругленные углы и его форма может изменяться вплоть до эллипса или окружности. Для его создания служит одноименная функция `rectangle`. В самом общем случае она вызывается с входными и выходными аргументами:

```
hR = rectangle(..., 'название свойства', значение,...)
```

возвращая указатель на созданный графический объект — прямоугольник. Размеры и положение прямоугольника задаются вектором из четырех элементов `[x, y, w, h]` в качестве значения свойства `Position`, где `x` и `y` — координаты левого нижнего угла, а `w` и `h` — ширина и высота прямоугольника в системе координат осей. Свойство `Curvature` отвечает за скругление углов, его значением должен быть вектор длины два `[cw ch]`, в котором `cw` — скругляемая часть от ширины, а `ch` — от высоты прямоугольника. Компоненты `cw` и `ch` могут изменяться от 0 до 1, причем 0 означает отсутствие скругления. Например, последовательность команд

```
>> hF = figure;
>> hA = axes('DataAspectRatio', [1 1 1], 'XLim', [0 1], 'YLim', [0 1]);
>> hR = rectangle('Position', [0.25 0.25 0.5 0.5], 'Curvature', [1 1]);
```

создает квадратное графическое окно, на котором расположена пара осей с фиксированными пределами по `x` и `y`, содержащих окружность. Мы использовали равный масштаб по каждой из осей для графического вывода без искажения размеров. Задание пределов осей позволило избежать автоматического их изменения, поскольку свойства осей `XLimMode` и `YLimMode` по умолчанию имеют значение `'auto'`.

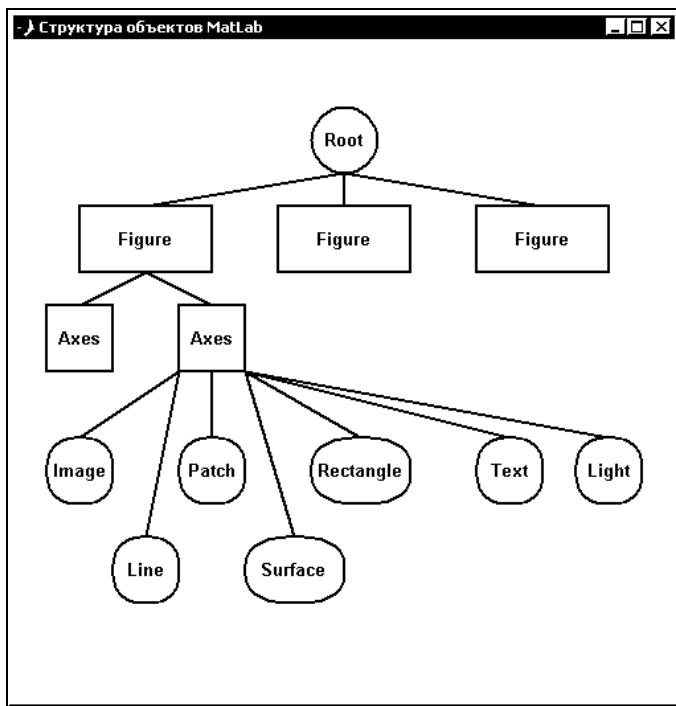
Объект `Rectangle` имеет еще ряд свойств, назначение которых очевидно: `EdgeColor`, `FaceColor` — цвет границы и внутренности прямоугольника, `LineWidth` и `LineStyle` — толщина и стиль линии границы.

При помощи объекта `Rectangle` легко конструировать диаграммы и блок-схемы, элементы которых соединяются линиями — объектами `Line`. Линия может быть ломаной, для ее создания применяется функция `line`, одно из возможных обращений к ней выглядит следующим образом:

```
hL = line(x, y, 'название свойства', значение,...)
```

Векторы `x` и `y` содержат координаты точек плоскости, через которые следует провести линию, а свойства объекта `Line` в целом имеют тот же смысл, что и для объекта `Lineseries` (см. разд. "Свойства линий и поверхностей" этой главы).

Различие между объектами `Line` и `Lineseries` мы обсудим ниже. В выходном аргументе `hL` возвращается указатель на объект `Line`. Для получения линии в трехмерном пространстве вместо пары векторов следует указать три вектора `x`, `y` и `z` с координатами вершин ломаной. Массивы `x`, `y` и `z` могут быть и двумерными, в этом случае каждому столбцу отвечает своя линия, а указатели на все линии записываются в вектор `hL`. Альтернативный способ вызова функции `line` предполагает указание массивов с координатами вершин в качестве значений свойств `XData`, `YData` и `ZData`.



**Рис. 9.17.** Схема, построенная при помощи базовых объектов `Rectangle`, `Line` и `Text`

Разберем теперь рисование простых схем и диаграмм при помощи объектов `Rectangle`, `Line` и `Text` на примере схемы, приведенной на рис. 9.17. Разумеется, рисование таких схем может быть выполнено в интерактивной среде для построения графиков, которой вы пользовались при изучении главы 3. Мы сейчас обсуждаем случай, когда блок-схема является результатом работы собственного приложения. При создании нашей схемы использован тот же самый прием, что и для вывода текста в произвольное место графического окна — невидимые оси, совпадающие по размеру с графическим окном (см. разд. "Размещение текста, линий и стрелок в графическом окне" этой главы).

Перед размещением объектов на схеме требуется определить их координаты и размеры в системе координат осей, для чего можно сначала нарисовать схему на бумаге в клеточку. Гораздо эффективнее задействовать сетку осей на этапе конструирования схемы. Причем удобно сначала расположить оси так, чтобы были видны координаты разметки. Когда схема будет готова, останется выключить сетку, увеличить размеры осей и сделать их невидимыми.

Для получения схемы начните работу над файл-программой `muchart`, воспользовавшись листингом 9.15, а для контроля перед размещением каждого нового объекта выполняйте уже набранный блок команд. Несмотря на то, что на оси все время выводятся новые графические объекты, команда `hold on` или установка свойства `NextPlot` в значение '`'add'`' не требуются. Низкоуровневые графические функции, в том числе `rectangle` и `line`, в отличие от высокоуровневых, всегда добавляют объект на оси.

### Листинг 9.15. Начало файла-программы `muchart`

```
% Создание графического окна
hF = figure('Position', [200 200 500 500], 'MenuBar', 'none',...
'NumberTitle', 'off', 'Name', 'Структура объектов MATLAB', 'Color', 'w');
% Создание вспомогательных осей с сеткой
hA = axes('XLim', [0 100], 'YLim', [0 100], 'DataAspectRatio', [1 1...
1], ...
'XTick', 0:5:100, 'YTick', 0:5:100, 'XGrid', 'on', 'YGrid', 'on');
% Создание круга с надписью Root
hR1 = rectangle('Position', [45 80 10 10], 'Curvature',[1 1],...
'LineWidth', 2);
hT1 = text(50, 85, 'Root', 'HorizontalAlignment', 'center', 'FontWeight',...
'bold')
% Создание левого прямоугольника с надписью Figure
hR2 = rectangle('Position', [10 65 20 10], 'LineWidth', 2);
hT2 = text(20, 70, 'Figure', 'HorizontalAlignment', 'center', ...
```

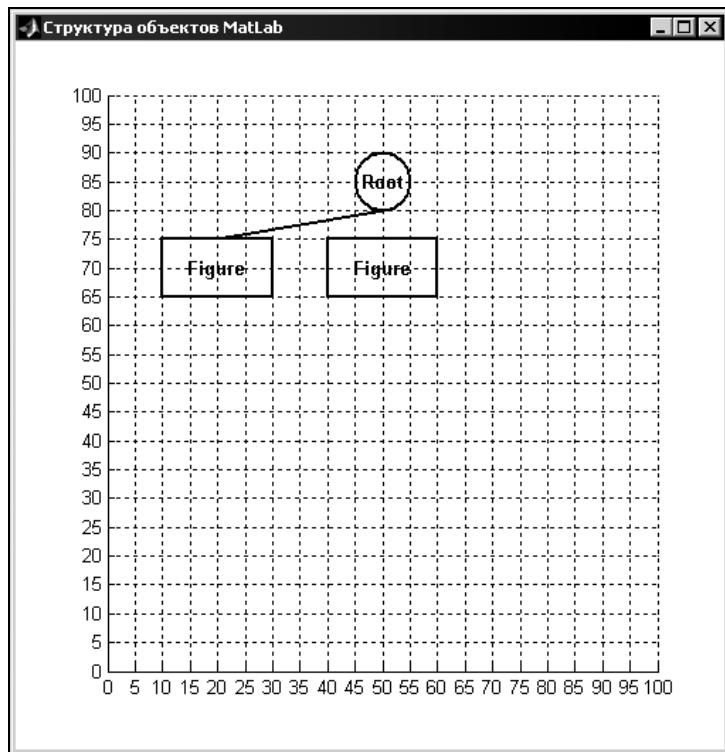
```
'FontWeight', 'bold')

% Создание центрального прямоугольника с надписью Figure
hR3 = rectangle('Position', [40 65 20 10], 'LineWidth', 2);
hT3 = text(50, 70, 'Figure', 'HorizontalAlignment', 'center', ...
'FontWeight', 'bold')

% Создание правого прямоугольника с надписью Figure
hR3 = rectangle('Position', [40 65 20 10], 'LineWidth', 2);
hT3 = text(50, 70, 'Figure', 'HorizontalAlignment', 'center', ...
'FontWeight', 'bold')

% Создание линии, соединяющей круг и левый прямоугольник
hL1 = line([20 50], [75 80], 'LineWidth', 2, 'Color', 'k')
```

На данном этапе схема должна выглядеть так, как показано на рис. 9.18.



**Рис. 9.18.** Использование сетки осей  
для размещения элементов схемы

Разместите остальные элементы самостоятельно, а затем измените вызов функции axes в начале файл-программы mychart на следующий:

```
hA = axes('Position', [0 0 1 1], 'XLim', [0 100], 'YLim', [0 100], ...
'DataAspectRatio', [1 1 1], 'Visible', 'off');
```

Результат должен соответствовать рис. 9.17.

Большие возможности для рисования блок-схем дает слой пояснений и принадлежащие ему объекты: стрелки, текстовые надписи, прямоугольники и эллипсы (см. разд. "Размещение текста, линий и стрелок в графическом окне" этой главы).

Как вы уже знаете, слой пояснений это невидимые оси. На этапе проектирования удобно сделать их видимыми, снабдить сеткой и расположить в графическом окне так, чтобы видеть координаты разметки. Единственная сложность — получение указателя на невидимые оси, который является скрытым (доступ к скрытым указателям обсуждается в разд. "Управление объектами, копирование, поиск, скрытые указатели" данной главы).

Отметим еще один способ быстрого написания кода для рисования блок-схем. Интерактивная среда для построения графиков позволяет вручную располагать поясняющие объекты и генерировать файл-функцию, запуск которой приводит к воспроизведению нарисованной схемы в графическом окне. Блоки операторов этой файл-функции можно копировать в собственную программу для вывода нужных элементов блок-схем.

## Объект Patch, цветовое оформление объектов

Объект Patch (полигональный объект) представляет собой один или несколько многоугольников на плоскости, или фигуру с одной или несколькими гранями в трехмерном пространстве. Свойства объекта Patch допускают гибкое управление его цветом — от постоянного до изменяющегося по заданному закону. Универсальность полигональных объектов позволяет моделировать сложные двумерные и трехмерные физические объекты.

Для создания объекта Patch служит встроенная функция patch, обращение к которой в самом простом случае имеет вид:

```
hP = patch(x, y, C)
```

где векторы x и y содержат координаты вершин многоугольника, а C — способ указания цвета: одно из стандартных сокращений: 'r', 'g' и т. д., или вектор из трех чисел для указания цвета в формате RGB. Совпадение координат первой и последней вершин не требуется, поскольку функция patch автоматически соединяет их отрезком, образуя замкнутый многоугольник. Необязательный выходной аргумент содержит указатель на объект Patch.

В качестве примера напишем приложение, которое позволяет рисовать многоугольные объекты при помощи мыши. В окне приложения пользователь расставляет вершины многоугольника щелчком по левой кнопке мыши и для создания многоугольника нажимает одну из клавиш для задания его цвета (`<r>`, `<g>`, `<b>`, `<c>`, `<m>`, `<y>`, `<k>`, `<w>`), клавиша `<x>` служит для выхода из режима рисования. Нам понадобится функция `ginput`. Она предназначена для получения координат точки осей, в которой был произведен щелчок мышью. Во входном аргументе `ginput` указывается число выбираемых точек `n`, а возвращает она массивы `x` и `y` с координатами точек, и массив `flag` той же длины с информацией о действиях пользователя.

```
[x, y, flag] = ginput(n)
```

Если при выборе точки была нажата левая кнопка мыши, то соответствующий элемент массива `flag` равен 1, средняя — 2 и правая — 3. В случае нажатия клавиши ячейка массива `flag` содержит ASCII код символа. Если пользователь нажал `<Enter>`, то функция `ginput` досрочно прекращает работу.

### Примечание

Для получения ASCII кода используется функция `double`, например, `double('x')`, а для преобразования ASCII кода в символ — функция `char`, например, `char(100)`.

Итак, требуется в цикле считывать координаты выбираемой точки и распознавать нажатую клавишу или кнопку мыши.

Возможный вариант файл-программы `mypatch` приведен в листинге 9.16. Обратите внимание, что оси являются невидимыми и их размеры совпадают с размерами графического окна. Сбор информации и обработка действий пользователя производятся в бесконечном цикле `while`, для выхода из которого служит оператор `break`. Для наглядности на месте выбранных точек появляются маркеры, а после создания многоугольника они удаляются. Рисование маркера организовано при помощи вырождающейся в точку линии — объекта `Line`, положение которого задается парой чисел. Функция `patch`, так же как и `line`, не проверяет значение свойства `NextPlot` осей, а просто добавляет новый объект на оси. Поэтому для размещения нескольких объектов `Patch` команда `hold on` или установка свойства `NextPlot` в значение '`'add'` не требуются.

#### Листинг 9.16. Файл-программа `mypatch` для рисования многоугольника

```
% Создание графического окна
figure('Position',[200 200 400 400], 'MenuBar', 'none', 'Color', 'w', ...
'Name', 'mypatch', 'NumberTitle', 'off')
```

```
% Создание невидимых осей
axes('Position', [0 0 1 1], 'Visible', 'off', 'XLim', [0 1], 'YLim',
[0 1])
% Создание пустых массивов для накопления
% координат вершин многоугольников
X = []; Y = [];
% Создание пустого массива для накопления указателей на маркеры вершин
hM = []
% Сбор информации в бесконечном цикле
while 1
 % Считывание координат точки и выбора пользователя
 [x, y, flag] = ginput(1)
 if flag == double('x')
 % Была нажата клавиша x, выход из цикла
 break
 end
 if flag == 1
 % Был щелчок левой кнопкой мыши
 % Добавление координат точки в вектор-столбцы
 X = [X; x]; Y = [Y; y];
 % Помещение в эту точку маркера и сохранение указателя на него
 h = line(x, y, 'Marker', '.', 'Color', 'k');
 hM = [hM h];
 else
 % Была нажата клавиша или кнопка мыши, отличная от левой
 c = char(flag);
 if (c == 'r') | (c == 'g') | (c == 'b') | (c == 'c') | (c == 'm') |
 (c == 'y') | (c == 'k') | (c == 'w')
 % Создание многоугольника выбранного цвета
 patch(X, Y, c)
 % Удаление маркеров
 delete(hM)
 % Подготовка массивов к созданию нового многоугольника
 hM = []; X = []; Y = [];
 else
 % Была нажата клавиша, для которой нет соответствия цвету
 disp('Неверный цвет')
 end
 end
end
```

Файл-программа `mypatch` позволяет рисовать произвольные объекты, например, объект, представленный на рис. 9.19. На основе листинга 9.16 легко создать файл-функцию, возвращающую указатели на все созданные полигональные объекты для дальнейшего изменения их свойств.

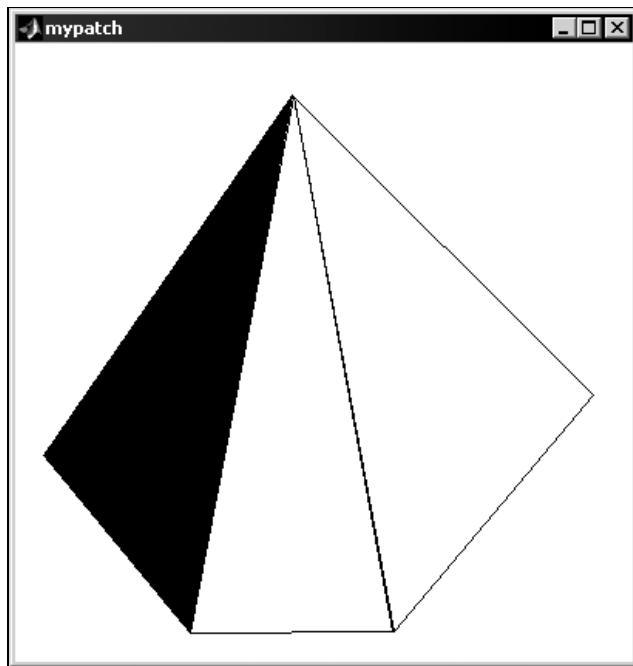


Рис. 9.19. Пример работы с программой `mypatch`

Пирамида, изображенная на рис. 9.19, в действительности состоит из трех плоских объектов `Patch`, каждый из которых является треугольником. Как было отмечено в начале этого раздела, несколько многоугольников могут образовывать *один* объект `Patch`. Для получения такого объекта следует указать двумерные массивы в качестве первых двух входных аргументов функции `patch`, причем каждый столбец массива `x` должен содержать абсциссы вершин соответствующего многоугольника, а `y` — их ординаты. Например, команда

```
>> hP = patch(rand(4, 3), rand(4, 3), 'r')
```

приводит к появлению полигонального объекта `Patch`, являющегося объединением трех красных четырехугольников со случайным образом выбранными координатами вершин.

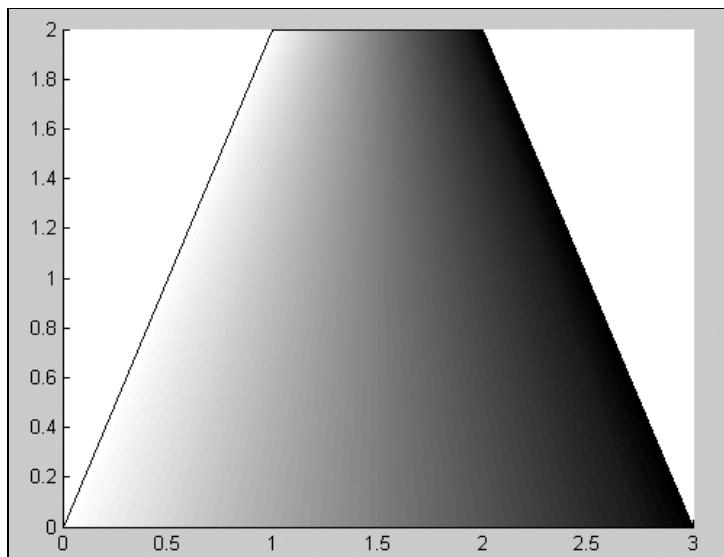
**Примечание**

Если многоугольники объекта `Patch` имеют разное число вершин, то неиспользуемым элементам массивов координат присваивается значение `Nan` (не число).

В приведенных примерах полигональные объекты были фиксированного цвета. Возможности заливки цветом достаточно обширны — в частности, допускается указание цвета каждой вершины с плавным его изменением внутри многоугольных областей, составляющих объект `Patch`. Цвет вершины задается в формате `RGB` или индексируется, т. е. задается числом, которое соответствует цвету палитры графического окна.

Рассмотрим, как обеспечить градиентную заливку трапеции, изображенной на рис. 9.20. Создайте графическое окно с цветовой палитрой `gray` (оттенки серого цвета):

```
>> figure
>> colormap(gray)
```



**Рис. 9.20.** Объект `Patch` с плавным изменением цвета от вершины к вершине

Затем задайте векторы с координатами и индексированным цветом каждой вершины для палитры `gray`, учитывая, что максимальное значение будет соответствовать белому цвету, а минимальное — черному:

```
>> x = [0; 1; 2; 3];
```

```
>> y = [0; 2; 2; 0];
>> C = [1; 1; 0; 0];
```

Для получения требуемого результата осталось вызвать функцию `patch`:

```
>> patch(x, y, C)
```

Задание цвета вершин в формате RGB требует создания массива `C` с подходящим числом измерений. Если объект `Patch` состоит из одного многоугольника, причем `x` и `y` являются векторами, как в только что рассмотренном примере, то массив `C` трехмерный размера 4 на 1 на 3, причем каждое его сечение длины 3 содержит три числа: доли красного, зеленого и синего цветов. Например, последовательность команд:

```
>> figure
>> C(1, 1, 1:3) = [1 0 0];
>> C(2, 1, 1:3) = [0 1 0];
>> C(3, 1, 1:3) = [0 0 1];
>> C(4, 1, 1:3) = [0 0 0];
>> patch(x, y, C)
```

создает трапецию с красной, зеленой, синей и черной вершиной и плавно изменяющимся цветом внутри области.

В общем случае, если `x` и `y` — матрицы размера `m` на `n`, то трехмерный массив `C` должен иметь размеры `m` на `n` на 3. В справочной системе MATLAB приведено общее правило формирования многомерного массива `C` при указании индексированного цвета вершин и цвета в формате RGB для двумерных и трехмерных полигональных объектов (см. описание функции `Patch`, например, в разд. [MATLAB: Functions -- Alphabetical List](#)).

Обратимся теперь к созданию трехмерного объекта `Patch`, для построения которого следует обратиться к функции `patch` с четырьмя входными аргументами:

```
hP = patch(x, y, z, C)
```

где `x`, `y` и `z` — матрицы с координатами вершин каждого многоугольника, образующего грань, а `C` — постоянный цвет или массив поддающей размерности.

В разделе справочной системы [MATLAB: 3-D Visualization: Creating 3-D Models with Patches: Multi-Faceted Patches](#) рассмотрен пример построения куба и указано содержимое массивов.

До сих пор мы использовали высокоуровневый способ вызова функции `patch` и не задействовали свойства объекта. Геометрия полигонального объекта описывается свойствами, приведенными в табл. 9.6.

**Таблица 9.6.** Свойства объекта *Patch*, отвечающие за его геометрию

| Название свойства   | Описание                                            | Значения                                                                                  |
|---------------------|-----------------------------------------------------|-------------------------------------------------------------------------------------------|
| XData, YData, ZData | Координаты вершин многоугольников, образующих грани | Двумерные массивы (или векторы для плоского объекта, состоящего из одного многоугольника) |
| Faces               | Массив связей между вершинами                       | Двумерный массив, каждая строка которого содержит номера вершин для соответствующей грани |
| Vertices            | Координаты вершин многоугольников, образующих грани | Двумерный массив                                                                          |

Назначение свойств XData, YData, ZData очевидно — их значениями являются те же самые массивы, которые указываются при высокуровневом вызове функции `patch`. Следующие два обращения к `patch` приводят к появлению одинаковых по форме двумерных полигональных объектов: `patch(x, y, C)` и `patch('XData', x, 'YData', y)`. Отличие состоит в том, что в первом случае задан цвет, а при низкоуровневом использовании `patch` цвет выбирается по умолчанию (черный), поскольку не были установлены свойства, определяющие цвет полигонального объекта. Эти свойства описаны ниже, мы отметим сейчас только одно: `FaceColor` — цвет граней. Его значением, например, может быть один из стандартных цветов '`r`', '`g`', '`b`', '`w`'. Таким образом, `patch(x, y, 'w')` и `patch('Xdata', x, 'YData', y, 'FaceColor', 'w')` эквивалентны. Обсудим назначение еще двух свойств `Faces` и `Vertices`, которые позволяют задать полигональный объект альтернативным образом.

Объект `Patch` является набором многоугольников со своими вершинами, однако его можно интерпретировать как набор вершин и связей между ними, образующих грани полигонального объекта. Например, для пирамиды, изображенной на рис. 9.21, первая грань является многоугольником с вершинами 1, 2, 3 и 4, вторая — 1, 4 и 5 и т. д. Такой способ описания полигонального объекта более эффективен, чем хранение координат вершин каждой грани, поскольку часто требует меньше памяти.

Массив с координатами вершин задается в качестве значения свойства `Vertices`, причем его  $n$ -я строка содержит координаты вершины с номером  $n$ . Свойство `Faces` служит для определения номеров вершин граней. Его значением является массив с номерами вершин, в котором  $k$ -я строка отвечает  $k$ -й грани, причем число столбцов массива равно максимальному числу вершин среди всех граней. Поскольку разные грани могут иметь неоди-

наковое число вершин (в нашем примере 3 и 4), то будут использоваться не все элементы в строках этого массива — ненужным элементам присваивается NaN.

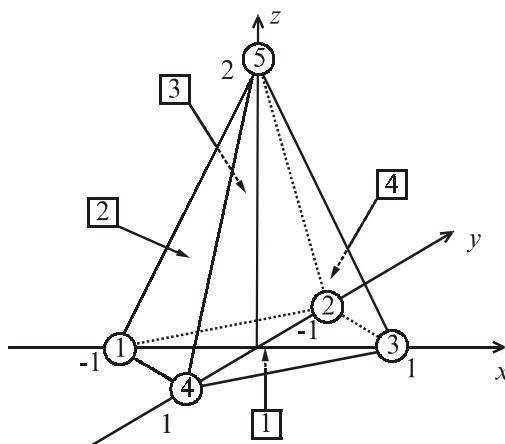


Рис. 9.21. Нумерация граней и вершин пирамиды

Итак, для построения пирамиды необходимо сформировать два массива и указать их в качестве значений свойств `Faces` и `Vertices`. Функция `patch` не изменяет размерность объекта на графике, двумерном по умолчанию, поэтому воспользуйтесь свойством осей `CameraPosition` для выбора точки обзора. Следующие команды создают графическое окно, оси и требуемый полигональный объект с гранями белого цвета.

```

>> hF = figure;
>> hA = axes('CameraPosition', [10 12 8]);
>> v = [-1 0 0
 0 1 0
 1 0 0
 0 -1 0
 0 0 2];
>> f = [1 2 3 4
 5 4 1 NaN
 2 1 5 NaN
 3 2 5 NaN
 4 3 5 NaN];
>> hp = patch('Vertices', v, 'Faces', f, 'FaceColor', 'w')

```

Варьируя точку обзора, убедитесь, что полученный объект действительно трехмерный. Примените функцию `set` для изменения цвета граней, например, на красный.

Объект `Patch` имеет набор свойств для цветового оформления, управления освещенностью и прозрачностью. Обсудим более подробно заливку цветом элементов объекта `Patch`. Существует два способа закраски полигонального объекта: задание цвета граней или вершин. Мы использовали самый простой вариант первого способа при создании призмы — свойство `FaceColor`, т. е. цвет граней, принимало значение '`w`' и все грани были белые. Для получения разноцветных граней следует установить свойство `FaceColor` в значение '`flat`' и определить цвет каждой грани в зависимости от способа описания полигонального объекта. Если объект `Patch` создан при помощи массивов с координатами вершин многоугольников (свойства `XData`, `YData`, `ZData`), то следует привлечь свойство `cData`. Его значением должен быть массив с информацией о цвете вершин или граней, в индексированном виде или в формате `RGB`. Это тот же самый массив `C`, который мы указывали во входных аргументах `patch` при высокогуровневом вызове: `patch(x, y, C)` или `patch(x, y, z, C)`.

Заливка цветом полигонального объекта, который сконструирован на основе связей между вершинами (свойств `Vertices` и `Faces`), потребует задействования свойства `FaceVertexCData`. Его значением может быть скаляр, вектор или матрица. Скаляр служит для задания индексированного цвета, одинакового для всех граней. Вектор-строка длины 3 воспринимается как цвет всех граней в формате `RGB`. Вектор-столбец, длина которого равна числу граней, определяет индексированный цвет каждой грани. Матрица с числом строк, равным числу граней, и тремя столбцами позволяет задать цвет в формате `RGB` для каждой грани, например:

```
>> FC = [0 0 0
 0 1 0
 0 0 1
 1 0 0
 1 1 0]

>> hP = patch('Vertices', V, 'Faces', F, 'FaceColor', 'flat', ...
'FaceVertexCData', FC)
```

Указание цвета вершин предоставляет более широкие возможности по оформлению полигонального объекта по сравнению с фиксированным цветом каждой грани. Как и в случае постоянного цвета внутри грани, способ конструирования объекта `Patch` определяет выбор между свойствами `cData` и `FaceVertexCData`. Только теперь длина соответствующих векторов или число строк в матрице должны равняться числу вершин. Для плавного из-

менения цвета от вершины к вершине следует установить свойство FaceColor в значение 'interp'.

В случае, когда задан цвет вершин и свойство FaceColor принимает значение 'flat', цвет внутри грани постоянный и определяется цветом первой вершины многоугольника, образующего грань.

Примеры закраски куба приведены в справочной системе MATLAB в разд. **MATLAB: 3-D Visualization: Creating 3-D Models with Patches: Multi-Faceted Patches**.

Передние грани полигонального объекта скрывают от взора те, которые расположены за ними. Границы допускают произвольную степень прозрачности, которая устанавливается при помощи свойства FaceAlpha. Его значением является неотрицательное число, например:

```
>> set(hP, 'FaceAlpha', 0.5)
```

причем 0 соответствует полностью прозрачным граням, а 1 — непрозрачным.

Каждая грань может иметь свой коэффициент прозрачности, который определяется в вектор-столбце с числом элементов, равным числу граней. Этот вектор берется в качестве значения свойства FaceVertexAlphaData, а FaceAlpha устанавливается в 'flat'. Более гибкое управление прозрачностью требует определения коэффициентов для каждой вершины в векторе и задания свойству FaceAlpha значения 'interp'. В этом случае прозрачность в пределах грани плавно изменяется от вершины к вершине. Если FaceAlpha установлено в 'flat', то прозрачность всей грани определяется коэффициентом для первой ее вершины.

Перейдем теперь к свойствам ребер полигонального объекта. Свойства линий ребер LineStyle, LineWidth, Marker, MarkerSize комментировать не требуется, они используются так же, как и в случае линии — объекта Line. Отметим только, что маркеры размещаются в вершинах многоугольников. Цвет ребер зависит от значения свойства EdgeColor и может быть одним из сокращений для цвета, либо цветом в формате RGB. Кроме того, значение 'flat' приводит к совпадению цветов ребра и вершины, из которой оно исходит, а 'interp' — к плавному изменению цвета между его вершинами. Ребра не прорисовываются, если EdgeColor установлено в 'none'. Степень прозрачности ребер зависит от значения свойства EdgeAlpha. Если его значение — число от 0 до 1, то все ребра имеют одинаковую прозрачность. Определение прозрачности каждого из ребер потребует привлечения свойства FaceVertexAlphaData и установки значения EdgeAlpha в 'flat' или 'interp'.

В качестве упражнения измените прозрачность граней и ребер призмы, увеличивающуюся от основания.

Одна из возможных последовательностей команд приведена ниже:

```
>> t = [1; 1; 1; 1; 0]
>> set(hP, 'FaceVertexAlphaData', t, 'FaceAlpha', 'interp')
>> set(hP, 'LineWidth', 3, 'EdgeColor', 'c', 'EdgeAlpha', 'interp')
```

### Примечание

Смысл коэффициентов прозрачности ребер или вершин определяется значением свойства `AlphaDataMapping`, которое по умолчанию равно `'scaled'`, т. е. коэффициенты прозрачности масштабируются. Если в нашем примере положить `t = [3; 3; 3; 3; 0]`, то ничего не изменится. Прозрачность определяется также свойством графического окна `AlphaMap` и осей `ALim`, которые в наших примерах принимаются установленными по умолчанию.

Примеры, посвященные изменению цвета ребер и маркеров, приведены в подразделах разд. **MATLAB: 3-D Visualization: Creating 3-D Models with Patches: Patch Edge Coloring** справочной системы MATLAB.

Поверхность, т. е. объект `Surface`, так же как и полигональный объект, допускает различную степень прозрачности, изменение стиля сетки поверхности и цвета ячеек. При этом название соответствующих свойств поверхности то же самое, что и в случае объекта `Patch`: `FaceAlpha`, `FaceColor`, `LineStyle` и т. д. Исключение составляет свойство `FaceVertexAlphaData` — для поверхности вместо него используется `AlphaData`. В следующем разделе мы рассмотрим освещение трехмерных объектов на примере поверхности, поскольку полигональные объекты освещаются аналогичным образом. Объекты `Surface` и `Patch` схожи друг с другом — полигональный объект является более общим случаем поверхности, которая определена на сетке с прямоугольными ячейками. Более того, поверхность может быть преобразована в полигональный объект при помощи специальной функции `surf2patch`.

## Освещение объектов, объект `Light` (источник света)

Освещение поверхностей и полигональных объектов позволяет получить естественный вид моделируемых трехмерных объектов. При чтении главы 3 мы использовали функцию высокогоуровневой графики `surfl` для получения освещенной поверхности, а при работе в редакторе свойств добавляли источники света с заданными характеристиками. В этом разделе мы обсудим возможности низкоуровневой графики для освещения трехмерных объектов.

Источник света создается при помощи функции `light`, которая возвращает указатель на объект `Light`. Источники света бывают двух типов: с парал-

лельными лучами и лучами, направленными радиально, что определяется его свойством `Style`. Его значение '`infinite`' (по умолчанию) соответствует бесконечно удаленному источнику с параллельными лучами, а '`local`' — источнику, испускающему лучи света во все стороны. Положение источника в системе координат осей задается его свойством `Position`, которое по умолчанию принимает значение  $[1 \ 0 \ 1]$ , причем для бесконечно удаленного источника его значение говорит лишь о направлении освещения. Третьей характеристикой объекта `Light` является его цвет (белый по умолчанию), для выбора которого служит свойство `Color`.

Рассмотрим освещение трехмерных объектов на примере параметрически заданного конуса

$$x(u, v) = 0.3 \cdot u \cdot \cos v, \quad y(u, v) = 0.3 \cdot u \cdot \sin v, \quad z(u, v) = 0.6 \cdot u, \quad u, v \in [-\pi, \pi].$$

Построение поверхностей второго порядка, заданных параметрически, в том числе и конуса, описано в разд. "Построение параметрически заданных поверхностей и линий" главы 3.

Создайте подходящие массивы `X`, `Y` и `Z`, графическое окно и оси:

```
>> u = (-pi:0.05*pi:pi)';
>> v = -pi:0.05*pi:pi;
>> X = 0.3*u*cos(v);
>> Y = 0.3*u*sin(v);
>> Z = 0.6*u*ones(size(v));
>> hF = figure;
>> hA = axes;
```

Для получения поверхности примените низкоуровневую функцию `surface`, воспользовавшись свойствами `XData`, `YData`, `ZData`, `FaceColor` и `LineStyle` объекта `Surface`:

```
>> hS = surface('XData', X, 'YData', Y, 'ZData', Z, 'FaceColor', 'g', ...
'LineStyle', 'none')
```

Функция `surface` в отличие от высокоуровневых `surf`, `mesh` и др. не устанавливает трехмерный вид осей, поэтому следует позаботиться о выборе точки обзора:

```
>> set(hA, 'CameraPosition', [-15 -25 20])
```

Создайте теперь источник белого света с параллельными лучами, направление которых определяется вектором  $[0 \ -1 \ 0]$ :

```
>> hLt=light('Position', [0 -1 0])
```

Конус освещается светом, исходящим от бесконечно удаленного источника. Остановимся на некоторых особенностях полученного изображения, кото-

рые связаны со способом освещения трехмерных объектов. Изменение точки обзора позволяет убедиться в отсутствии неосвещенных участков. Например, посмотрите на скрытую от света часть конуса

```
>> set(hA, 'CameraPosition', [0 1 0])
```

Цвет ее темнее освещенной части, но не черный, который должен соответствовать неосвещенной поверхности. Дело в том, что при создании объекта Light на оси автоматически добавляется рассеянный свет белого цвета и свойство *осей* AmbientLightColor принимает значение 'w'. Задайте черный цвет, т. е. отсутствие равномерного освещения

```
>> set(hA, 'AmbientLightColor', 'k')
```

Теперь неосвещенная источником света часть конуса стала черного цвета. Вернитесь к первоначальной точке обзора поверхности и включите равномерное освещение

```
>> set(hA, 'CameraPosition', [-15 -25 20])
```

```
>> set(hA, 'AmbientLightColor', 'w')
```

Заметно, что освещенная поверхность не является гладкой — оттенок зеленого цвета в пределах каждой ее ячейки не изменяется. Плавность освещения определяется свойством FaceLighting, которое по умолчанию равно 'flat' и задает постоянный цвет. Для линейной интерполяции оттенка цвета внутри ячейки следует использовать значение 'goraud':

```
>> set(hS, 'FaceLighting', 'goraud')
```

Значениями свойства FaceLighting могут также быть 'phong' (более сложный алгоритм подбора оттенков, часто приводящий к лучшему результату) и 'none' (ячейки поверхности не реагируют на освещение). Стиль подсветки линий сетки поверхности выбирается аналогичным образом с использованием свойства EdgeLighting:

```
>> set(hS, 'LineStyle', '-')
```

```
>> set(hS, 'EdgeColor', 'g')
```

```
>> set(hS, 'EdgeLighting', 'phong')
```

Видимая часть внутренности конуса освещена не только равномерным светом, но и параллельными лучами источника, хотя она и является скрытой от лучей света частью поверхности. Таков стандартный способ освещения трехмерного объекта — если нормаль к поверхности направлена от камеры, то освещается внутренняя ее часть. Нормаль к конической поверхности действительно направлена наружу, в чем несложно убедиться, построив векторы нормали к поверхности (см. разд. "Визуализация векторных полей" главы 3).

Ответственным за освещение внутренности трехмерных объектов является свойство BackFaceLighting. По умолчанию оно принимает значение

'reverselit', но если изменить его на 'unlit', то лучи света источника перестают падать на внутреннюю поверхность конуса.

```
>> set(hS, 'BackFaceLighting', 'unlit')
```

При выборе 'reverselit' возможно некорректное отображение границ освещенной замкнутой поверхности, поскольку может оказаться, что нормаль в некоторой точке направлена от камеры, хотя сама точка видна. В этом случае прибегают к значению 'lit' (см. пример для сферы в разделе справочной системы **MATLAB: 3-D Visualization: Lighting as a Visualization Tool: Back Face Lighting**).

Мы рассмотрели свойства трехмерных объектов и осей, отвечающих за способы освещения. Обратимся теперь к отражению света от поверхности. Можно считать, что поверхность изготовлена из некоторого материала с заданными характеристиками отражения света. Самый простой способ состоит в выборе одного из стандартных материалов при помощи команды высокогорневой графики `material`. Изменяйте материал и наблюдайте за видом освещенной поверхности.

```
>> material metal
>> material shiny
>> material dull
>> material default
```

Проделайте аналогичные эксперименты для точечного источника света, установив свойство `Style` объекта `Light` в 'local'

```
>> set(hLt, 'Style', 'local')
```

Отражение материалом света, исходящего из источника, определяется четырьмя свойствами трехмерного объекта:

- ❑ `SpecularStrength` — доля отраженного света. Значение — число от 0 до 1, по умолчанию 0.9;
- ❑ `DiffuseStrength` — интенсивность рассеивания света по поверхности. Значение — число от 0 до 1, по умолчанию 0.6;
- ❑ `SpecularExponent` — резкость отраженного света, т. е. величина светового пятна на поверхности. Значение — число, большее 1, по умолчанию 10;
- ❑ `SpecularColorReflectance` — цвет отраженного света, который может изменяться от комбинации цветов объекта и источника до цвета падающего света. Значение — число от 0 до 1, по умолчанию 1 (цвет света источника).

Трехмерный объект отражает также равномерный свет, за цвет которого, как было упомянуто выше, отвечает свойство `AmbientLightColor` осей. Доля отраженного рассеянного света зависит от материала и задается при помо-

щи свойства `AmbientStrength` объекта. Его значением может быть число от 0 до 1, по умолчанию 0.3.

Изменяя значения перечисленных свойств, наблюдайте за видом конической поверхности.

Примеры освещенной поверхности для различных значений этих свойств приведены в справочной системе в подразделах разд. **MATLAB: 3-D Visualization: Lighting as a Visualization Tool: Reflectance Characteristics of Graphics Objects**.

Освещение полигональных объектов, рассмотренных в предыдущем разделе, производится аналогичным образом.

## Управление объектами, копирование, поиск, скрытые указатели

Объектно-ориентированная графика MATLAB предоставляет возможность копирования объектов в соответствии с их иерархией. Например, оси графического окна могут стать потомком только графического окна, того же самого или другого. При этом скопированные оси имеют тех же самых потомков, что и исходные, разумеется, с новыми указателями. Линии, поверхности и другие объекты самого низкого уровня могут быть скопированы только на оси. Другими словами, объект не может быть скопирован сам по себе — для него обязательно должен существовать предок. Копирование объектов производится при помощи функции `copyobj`, обращение к которой имеет вид `h1 = copyobj(h, p)`, где `h` — указатель на копируемый объект, `p` — на предок для нового объекта, а `h1` — на новый объект. Функция `copyobj` копирует объекты вместе со всеми их потомками.

Кроме копирования, существует еще ряд возможностей для управления графическими объектами. Для смены предка объекта служит его свойство `Parent`. Создайте, например, графическое окно с указателем `hF`, на котором находятся оси с диаграммой и новое окно с указателем `hF1`

```
>> hF = figure;
>> hA = axes;
>> hB = bar([1 2 3 1]);
>> hF1 = figure;
```

Оси имеют единственного потомка — объект `Barseries`, представляющий столбцовую диаграмму одномерных данных. Для перемещения осей вместе с их потомком в окно с указателем `hF1` следует установить значение свойства `Parent` осей в `hF1`:

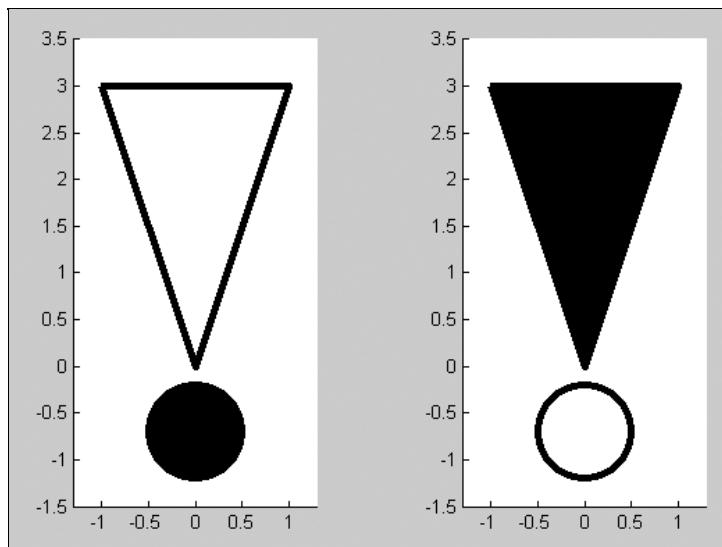
```
>> set(hA, 'Parent', hF1)
```

Согласно иерархии графических объектов, значением свойства `Parent` осей может быть только указатель на графическое окно, а для линий, поверхностей и других объектов самого низкого уровня — указатель на оси. Объект `Root` является предком всех графических окон, а следовательно значение их свойства `Parent` всегда равно нулю.

Ряд сервисных функций позволяет осуществить поиск всех потомков объекта (функция `allchild`), поиск всех потомков с определенными свойствами (функция `findall`), поиск объектов с определенными свойствами (функция `findobj`), удаление (функция `delete`) и ряд других.

Применение функции `delete` для удаления объектов описано в разд. "Удаление и очистка объектов" данной главы. Список всех функций для управления графическими объектами приведен в разделе справочной системы **MATLAB: Functions -- Categorical List: Graphics: Handle Graphics**.

Использование специальных функций для управления графическими объектами делает работу с ними более продуктивной. В качестве примера рассмотрим графическое окно, содержимое которого приведено на рис. 9.22. Разумеется, можно создать две пары осей, нанести на каждую из них полигональные объекты и объекты `Rectangle` и установить свойство `FaceColor` каждого объекта в нужное значение.



**Рис. 9.22.** Графическое окно, получаемое при помощи управления графическими объектами

Поступим по-другому:

1. Нарисуем левые оси и объекты на них.
2. Скопируем оси в то же самое графическое окно, удалим их потомков и поместим новые оси в левую часть окна.
3. На новые оси скопируем объекты Patch и Rectangle и приведем их к требуемому виду.

Более простой способ без удаления потомков осей, основанный на поиске объектов нужного типа, рассмотрен ниже.

Файл-программа exclam2, приведенная в листинге 9.17, выполняет перечисленные действия.

#### Листинг 9.17. Файл-программа exclam2, демонстрирующая копирование объектов

```
% создание графического окна и левой пары осей
hF = figure('MenuBar', 'none');
hA = axes('OuterPosition', [0 0 0.5 1], 'XLim', [-1.3 1.3], ...
'YLim', [-1.5 3.5], 'DataAspectRatio', [1 1 1])
% создание треугольного объекта Patch и круглого объекта Rectangle
hP = patch('XData', [-1 0 1], 'YData', [3 0 3], 'FaceColor', 'w',
'LineWidth', 4)
hR = rectangle('Position', [-0.5 -1.2 1 1], ...
'Curvature', [1 1], 'LineWidth', 4, 'FaceColor', 'k')
% копирование осей hA в графическое окно hF
hA1 = copyobj(hA, hF)
% поиск всех потомков новых осей hA1 и их удаление
h = get(hA1, 'Children')
delete(h)
% размещение новых осей hA1 в левой части графического окна
set(hA1, 'OuterPosition', [0.5 0 0.5 1])
% копирование объектов Patch и Rectangle на новые оси hA1
hP1 = copyobj(hP, hA1)
hR1 = copyobj(hR, hA1)
% изменение цвета заливки объектов Patch и Rectangle
set(hR1, 'FaceColor', 'w')
set(hP1, 'FaceColor', 'k')
```

Остановимся более подробно на поиске потомков графических объектов. В файл-программе exclam2 мы обратились к свойству `Children` для получения указателей на потомков осей — объектов `Patch` и `Rectangle`. На самом деле потомков больше, но они имеют *скрытые указатели*, доступ к которым при помощи свойства `Children` закрыт. Убедиться в этом позволяет функция `allchild`, которая возвращает вектор из указателей на потомков графического объекта. Создайте, например, пустые оси, выведите значение их свойства `Children` и сравните с тем, что возвращает `allchild`:

```
>> hA = axes;
>> hC1 = get(hA, 'Children')
hC1 =
Empty matrix: 0 - by - 1
>> hC = allchild(hA)
hC =
104.0005
103.0005
102.0005
101.0011
```

Итак, даже при отсутствии графических объектов (линий, поверхностей и т. д.), оси всегда имеют потомков. Более того, они не являются невидимыми, о чём говорит значение свойства `Visible`, например, первого потомка:

```
>> get(hC(1), 'Visible')
ans =
on
```

Мы знаем, что согласно иерархии графических объектов осям могут принадлежать потомки различных типов. Тип любого графического объекта записан в его свойстве `Type`, которое может принимать одно из следующих значений: `'root'`, `'figure'`, `'axes'`, `'image'`, `'light'`, `'line'`, `'patch'`, `'rectangle'`, `'surface'`, `'text'`, `'hggroup'`, `'hgtransform'` и `'uicontrol'`, определяющих тип объекта. Объекты последнего из перечисленных типов являются элементами приложений с графическим интерфейсом, которым посвящены следующие несколько глав. Работу со сгруппированными объектами, тип которых `'hggroup'` или `'hgtransform'`, мы рассмотрим чуть позже в этой главе. Отметим только сейчас, что большинство рисованных объектов, создаваемых функциями высокогоуровневой графики, имеют тип `'hggroup'` (например, `Contourgroup` или `Barseries`).

Выясним *тип* потомков осей со скрытыми указателями, вызвав функцию `get` от вектора указателей на них:

```
>> get(hC, 'Type')
ans =
'text'
```

```
'text'
'text'
'text'
```

Содержимое возвращаемого массива ячеек свидетельствует о том, что все четыре потомка осей есть текстовые объекты. Они не видны, потому что значениями их свойств `String` являются пустые строки (проверьте). Эти текстовые объекты служат для размещения заголовка и подписей к осям, причем указатели на них являются значениями свойств осей `Title`, `XLabel`, `YLabel` и `ZLabel`, например, команды

```
>> set(hC(1), 'String', 'заголовок')
```

и

```
>> hT = get(hA, 'Title');
>> set(hT, 'String', 'заголовок')
```

приводят к одинаковому результату — появлению заголовка осей.

Для скрытия указателя на объект достаточно установить его свойство `HandleVisibility` в значение `'off'`, как это и делается для четырех текстовых объектов (потомков осей) при создании осей:

```
>> get(hC, 'HandleVisibility')
ans =
'off'
'off'
'off'
'off'
```

Но все скрытые указатели можно одновременно сделать доступными с уровня корневого объекта, для чего следует его свойство `ShowHiddenHandles` установить в значение `'on'`:

```
>> set(0, 'ShowHiddenHandles', 'on')
```

Проверьте, что теперь видны указатели на текстовые объекты — потомки осей:

```
>> hC1 = get(hA, 'Children')
hC1 =
155.0057
154.0057
153.0057
152.0057
```

Значения их свойства 'HandleVisibility' не изменились, поэтому возврат к прежней глобальной установке:

```
>> set(0, 'ShowHiddenHandles', 'off')
```

снова приведет к скрытию указателей на них.

Объекты со скрытыми указателями применяются для того, чтобы предотвратить их случайное изменение операторами, набираемыми в командной строке или выполняемыми в другом приложении MATLAB. В качестве примера создайте графическое окно со скрытым указателем, установив его свойство HandleVisibility в значение 'off'

```
>> hF = figure('HandleVisibility', 'off')
```

```
hF =
```

```
1
```

При дальнейшем графическом выводе команды MATLAB не обнаружат этого графического окна и создадут новое окно с парой осей или задействуют графическое окно, указатель на которое не является скрытым (если свойство ShowHiddenHandles корневого объекта установлено в 'off'):

```
>> fplot(@sin, [0 3*pi])
```

Заметим, что графический объект со скрытым указателем, равным 1, существует и команды `close(1)` или `delete(1)` приведут к закрытию окна. Более того, возможна модификация свойств окна функцией `set`, например,

```
>> set(1, 'Color', 'w')
```

Для предотвращения случайного изменения свойств графических окон со скрытыми указателями имеет смысл использовать вещественные значения указателей вместо целых, присваиваемых по умолчанию. Для этого следует установить свойство `IntegerHandle` в значение 'off'.

Обсудим теперь влияние скрытых указателей на результаты поиска объектов при помощи других функций: `findall` и `findobj`. Функция `findall` служит для получения указателей на потомков заданного объекта, расположенных ниже по иерархии, причем возможен поиск потомков с определенными свойствами. Например, после выполнения файл-программы `exclam2` создается графическое окно с указателем `hF`, для поиска всех его потомков достаточно обратиться к `findall` следующим образом:

```
>> hC = findall(hF)
```

В командное окно выводится вектор, включающий указатели на графическое окно и все его потомки — оси и принадлежащие им объекты, в том числе и объекты со скрытыми указателями. Функция `findall` позволяет найти объекты с заданными значениями их свойств, для этого следует указать пары 'свойство', значение через запятую в списке входных аргументов. Следующие команды выделяют полигональный объект черного цвета

среди всех потомков графического окна, созданного программой exclam2 (см. листинг 9.17), и изменяют его цвет на серый:

```
>> hC = findall(hF, 'Type', 'patch', 'FaceColor', 'k')
>> set(hC, 'FaceColor', [0.8 0.8 0.8])
```

Функция `findobj` производит схожие с `findall` действия, но *игнорирует* объекты со скрытыми указателями. Кроме того, `findobj` допускает еще ряд способов вызова. В качестве первого входного аргумента может быть задан вектор указателей, тогда поиск объектов с нужными свойствами будет производиться для всех перечисленных объектов и их потомков. Если список входных аргументов `findobj` содержит только пары '*свойство*', значение, то просматриваются все существующие графические объекты. Вызов `findobj` без входных аргументов позволяет получить указатели на все графические объекты. Функция `findobj` позволяет добавлять логические операторы в поиск по значениям и свойствам объектов, делая его более гибким, а также ограничиться поиском до определенного уровня иерархии (см. информацию о `findobj` в справочной системе MATLAB, например в разд. **MATLAB: Functions -- Categorical List: Graphics: Handle Graphics**).

В файл-программе `exclam2`, приведенной в листинге 9.17, можно было не удалять потомки вторых осей, а найти указатели на них при помощи `findobj` и затем установить требуемые значения свойств. Этот подход реализован в файл-программе `exclam2f` (листинг 9.18).

#### Листинг 9.18. Файл-программа `exclam2f`, демонстрирующая копирование объектов

```
% создание графического окна и левой пары осей
hF = figure;
hA = axes('OuterPosition', [0 0 0.5 1], 'XLim', [-1.3 1.3], ...
'YLim', [-1.5 3.5], 'DataAspectRatio', [1 1 1])
% создание треугольного объекта Patch и круглого объекта Rectangle
hP = patch('XData', [-1 0 1], 'YData', [3 0 3], 'FaceColor', 'w',
'LineWidth', 4)
hR = rectangle('Position', [-0.5 -1.2 1 1], 'Curvature', [1 1], ...
'LineWidth', 4, 'FaceColor', 'k')
% копирование осей hA в графическое окно hF
hA1 = copyobj(hA, hF)
% размещение новых осей hA1 в левой части графического окна
set(hA1, 'OuterPosition', [0.5 0 0.5 1])
% поиск потомков (объектов Patch и Rectangle) новых осей
hR1 = findobj(hA1, 'Type', 'rectangle');
hP1 = findobj(hA1, 'Type', 'patch');
```

```
% задание цвета объектов Patch и Rectangle
set(hR1, 'FaceColor', 'w')
set(hP1, 'FaceColor', 'k')
```

Функции поиска объектов позволяют расширить возможности многих высокоуровневых графических функций. Например, обращение к файл-функции `pie3tr` из листинга 9.19 приводит к появлению круговой трехмерной диаграммы с прозрачными отделенными друг от друга секторами. При создании этой файл-функции был использован вектор указателей на графические объекты (`Surface` и `Patch`), которые создает стандартная функция `pie3` при конструировании трехмерной диаграммы.

**Листинг 9.19. Файл-функция `pie3tr` для вывода диаграммы с прозрачными секторами**

```
function h = pie3tr(X)
% Файл-функция для вывода прозрачной диаграммы с отделенными секторами

% построение круговой трехмерной диаграммы с отделенными секторами
h = pie3(X, ones(size(X)))
% задание текущему графическому окну белого цвета
set(gcf, 'Color', 'w')
% поиск полигональных объектов среди элементов диаграммы
% и изменение их свойств
hP = findobj(h, 'Type', 'patch')
set(hP, 'FaceAlpha', 0.1, 'LineStyle', 'none')
% поиск поверхностей среди элементов диаграммы
% и изменение их свойств
hS = findobj(h, 'Type', 'surface')
set(hS, 'FaceAlpha', 0.4, 'LineStyle', 'none')
```

Справочная система MATLAB содержит ряд примеров, в которых применяются функции высокоуровневой графики, а дополнительные элементы оформления наносятся при помощи низкоуровневой графики. Например, возможно изменить значения в процентах у секторов круговой диаграммы на текстовые надписи, управлять видом контурных графиков (см. в справочной системе MATLAB разд. **Graphics: Creating Specialized Plots**).

Функция `pie3` возвращает вектор указателей на базовые графические объекты `Surface`, `Patch` и `Text`. В приведенном выше примере мы воспользовались первыми двумя из них для придания круговой диаграмме желаемого

вида. Результатом работы большинства высокоуровневых графических функций является сгруппированный объект `hggroup`, который позволяет обращаться с ним, как с единым объектом. В начале этого раздела приведен пример о переносе столбцовой диаграммы с одних осей на другие. Проверьте, что созданный функцией `bar` объект `Barseries` имеет тип `hggroup`.

### Примечание

В предыдущих версиях MATLAB выходным аргументом всех высокоуровневых графических функций были указатели на объекты, названные базовыми в версии 7. Для совместимости эта возможность предоставлена и в версии 7. Достаточно указать '`v6`' в качестве первого входного аргумента графической функции. Сравните, например, типы графических объектов, указатели на которые возвращаются при следующих двух обращениях к `bar`: `h = bar([1 2 3])` и `hp = bar('v6', [1 2 3])`.

Рассмотрим теперь более подробно работу со сгруппированными объектами.

## Объекты-группы `hggroup` и `hgtransform`

Согласно иерархии, группы являются потомками осей и предками для всех объектов, которые могут быть потомками осей, т. е. базовых, рисованных и поясняющих объектов. Следовательно, группы могут быть потомками некоторой группы. Различается два типа групп: `hggroup` и `hgtransform`. Создание объекта `hggroup` позволяет обращаться с формирующими его объектами как с единым целым. Например, всю группу объектов можно сделать видимой или невидимой с привлечением ее свойства `Visible`. Группа может быть текущим объектом, если был сделан щелчок мышью по какому-нибудь из ее элементов. Кроме того, вся группа может быть выделена щелчком мыши только по одному из ее объектов. Для реализации двух последних возможностей следует установить свойство `HitTest` в значение '`off`' для каждого объекта, входящего в группу `hggroup`.

Группа `hgtransform` позволяет не только делать объекты видимыми или невидимыми, но и поворачивать, перемещать и масштабировать все сгруппированные объекты одновременно.

Для создания групп служат одноименные функции `hggroup` и `hgtransform`, которые имеют одинаковый интерфейс. Рассмотрим работу с группами на примере группы `hgtransform`, включающей в себя поверхность и полигональный объект.

При помощи следующих команд создайте графическое окно, оси с заданной точкой обзора, плоскую поверхность и стоящую на ней пирамиду:

```
>> hF = figure;
>> hA = axes('CameraPosition', [10 12 5]);
```

```

>> [X, Y] = meshgrid(-2:0.5:2);
>> z = zeros(size(X)).*zeros(size(Y));
>> hS = surface('XData', X, 'YData', Y, 'ZData', Z, 'FaceColor', 'y', ...
 'EdgeColor', 'r', 'FaceAlpha', 0.8, 'EdgeAlpha', 0.8);
>> V = [-1 0 0
 0 1 0
 1 0 0
 0 -1 0
 0 0 1];
>> F = [1 2 3 4
 5 4 1 NaN
 2 1 5 NaN
 3 2 5 NaN
 4 3 5 NaN];
>> hP = patch('Vertices', V, 'Faces', F, 'FaceColor', 'g', ...
 'EdgeColor', 'r', 'FaceAlpha', 0.8, 'EdgeAlpha', 0.8);

```

Объедините теперь поверхность и пирамиду в группу. Для этого следует создать группу при помощи функции `hgtransform`. Входными аргументами группы могут быть пары 'свойство', значение, а выходным является указатель на созданный объект — группу. Нам требуется создать группу, являющуюся потомком осей, следовательно ее свойство `Parent` надо установить в значение `hA` (указателя на оси):

```
>> hGT = hgtransform('Parent', hA);
```

Потомками группы должны стать поверхность и призма, поэтому укажите в качестве значений их свойства `Parent` указатель на группу:

```

>> set(hS, 'Parent', hGT)
>> set(hP, 'Parent', hGT)
```

Перед преобразованиями группы скопируем ее и зададим большую прозрачность входящим в нее объектам, т. е. ее потомкам:

```

>> hGT0 = copyobj(hGT, hA);
>> hC0 = get(hGT0, 'Children')
>> set(hC0, 'FaceAlpha', 0.1, 'EdgeAlpha', 0.1)
```

Все преобразования мы будем осуществлять над группой с указателем `hGT`, а скопированная группа послужит для наглядного представления о преобразованиях.

Обсудим теперь поворот вокруг осей, перемещение и масштабирование группы. Каждое из этих преобразований задается матрицей размера 4 на 4.

Например, для поворота вокруг оси  $x$  следует сформировать матрицу

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x & 0 \\ 0 & \sin \theta_x & \cos \theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

где значение  $\theta_x$  (в радианах) определяет величину поворота. Положительное значение  $\theta_x$  приводит к повороту против часовой стрелки, если смотреть по направлению к началу координат, а отрицательное, соответственно, к повороту по часовой стрелке. Матрицы поворота вокруг остальных двух осей имеют похожий вид. Перемещение и масштабирование так же задаются матрицами размера 4 на 4. Мы не приводим их по двум причинам. Во-первых, их структура указана в справочной системе (см. разд. **MATLAB: Graphics: Handle Graphics Objects: Group Objects: Transforming Objects**).

Во-вторых, существует специальная функция `makehgtform` для формирования этих матриц. Первым ее входным аргументом указывается желаемый тип преобразования, а вторым — его числовые характеристики. Для формирования матрицы поворота  $R_x$  вокруг оси  $x$  следует указать '`xrotate`' и угол поворота:

```
Rx = makehgtform('xrotate', teta)
```

Для получения матриц поворота вокруг осей  $y$  и  $z$  вместо '`xrotate`' используются, соответственно, '`yrotate`' и '`zrotate`'.

Перемещение может быть определено вектором из трех элементов `[tx ty tz]` для каждой из осей. Матрица перемещения  $T$  генерируется при следующем обращении к функции `makehgtform`:

```
T = makehgtform('translate', [tx ty tz])
```

Задание коэффициентов масштабирования вдоль каждой из осей в векторе `[sx, sy, sz]` и вызов

```
M = makehgtform('scale', [sx, sy, sz])
```

приводит к получению матрицы масштабирования  $M$ . При равномерном изменении размеров коэффициенты равны. В этом случае вместо вектора достаточно указать скалярное значение:

```
M = makehgtform('scale', s)
```

Для применения вышеописанных преобразований следует привлечь свойство `Matrix` группы `hgtransform`. Его значением может быть одна из матриц поворота, перемещения или масштабирования, или их произведение для

ряда последовательных преобразований. Продемонстрируем сказанное на примерах.

Сместим группу вверх по оси  $z$  на две единицы, для чего сформируем подходящую матрицу:

```
>> Tz = makehgtform('translate', [0 0 2])
```

и осуществим преобразование

```
>> set(hGT, 'Matrix', Tz)
```

Как и следовало ожидать, плоскость и пирамида сместились вверх на заданное расстояние. Однако последующее смещение вдоль оси  $y$  не сохранит предыдущего преобразования.

```
>> Ty = makehgtform('translate', [0 1 0]);
```

```
>> set(hGT, 'Matrix', Ty)
```

Одновременное применение двух или более преобразований выполняется при помощи произведения соответствующих матриц:

```
>> T = Ty*Tz
```

```
>> set(hGT, 'Matrix', T)
```

Например, для поворота на  $90^\circ$  вниз вокруг линии  $x = -2$  следует сначала произвести смещение на две единицы в положительном направлении оси  $x$ , затем осуществить поворот вокруг оси  $g$  и произвести обратное смещение.

```
>> Tx1 = makehgtform('translate', [2 0 0])
```

```
>> Ry = makehgtform('yrotate', pi/2)
```

```
>> Tx2 = makehgtform('translate', [-2 0 0])
```

```
>> T = Tx2*Ry*Tx1
```

```
>> set(hGT, 'Matrix', T)
```

Обратите внимание, что при преобразовании важен порядок множителей матрицы  $T$  — они учитываются справа налево. Изменение этого порядка приведет к другим преобразованиям, например

```
>> T2 = Tx1*Ry*Tx2
```

```
>> set(hGT, 'Matrix', T)
```

или

```
>> T3 = Tx1*Tx2*Ry
```

```
>> set(hGT, 'Matrix', T3)
```

Для возвращения группы в исходное состояние используется единичная матрица размера 4 на 4:

```
>> T = eye(4)
```

```
>> set(hGT, 'Matrix', T)
```

В справочной системе MATLAB включен пример преобразований для группы hgtransform с достаточно сложной и разветвленной структурой, в котором мы рекомендуем разобраться самостоятельно (см. разд. **MATLAB: Graphics: Handle Graphics Objects: Group Objects: Example -- Transforming a Hierarchy of Objects**).

В следующем разделе мы рассмотрим работу с рисованными объектами, большинство из которых являются сгруппированными объектами hggroup.

## Рисованные объекты (Plot Objects)

Высокоуровневые графические функции создают рисованные объекты, ряд свойств которых позволяет получить простой доступ к составляющим их базовым объектам. Часть свойств рисованных объектов расширяет их возможности по сравнению с соответствующими базовыми объектами. Приведем простой пример. При чтении разд. "Свойства линий и поверхностей" этой главы вы познакомились с основными свойствами объекта Lineseries, получаемого при помощи функции plot. С другой стороны, базовый объект Line обладает теми же самыми свойствами: толщина и цвет линии, тип, размер и цвет маркеров, XData, YData и ZData (для задания координат точек вершин ломаной) и т. д. Сравнение наборов свойств объектов Lineseries и Line позволяет выявить свойства, специфичные для рисованного объекта Lineseries.

Способы получения информации о свойствах графических объектов и их назначении описаны в разд. "Получение информации о свойствах графических объектов" данной главы.

Рисованная линия допускает связывание с ней строки, которая будет выводиться в области легенды рядом с образцом линии. Для этого следует обратиться к свойству DisplayName объекта Lineseries, например:

```
>> x = 0:0.1:10;
>> f = sin(x);
>> g = cos(x);
>> hF = figure
>> hA = axes
>> hL = plot(x, f, x, g)
>> set(hL(1), 'DisplayName', 'sin (\itx)')
>> set(hL(2), 'DisplayName', 'cos (\itx)')
```

Заданные строки появятся при отображении легенды на осях

```
>> hLG = legend(hA, 'show')
```

## Примечание

Функция `legend` возвращает указатель на легенду — пару осей с потомками: линиями и текстовыми объектами. В этом несложно убедиться, посмотрев тип объекта с указателем `hLG` и тип его потомков.

В любой момент с линией может быть связана другая строка:

```
>> set(hL(2), 'DisplayName', 'график косинуса')
```

Теперь для обновления легенды достаточно обратиться к функции `legend` следующим образом:

```
>> legend(hLG)
```

Кроме текстовой информации для легенды, с объектом `Lineseries` можно связать имена переменных, подлежащих графическому отображению. Имена переменных указываются в качестве значения свойств `XDataSource`, `YDataSource` и `ZDataSource`. При изменении значений этих переменных и вызове функции `refreshdata` произойдет обновление графика. Если функция `refreshdata` вызвана без входных аргументов, то изменения коснутся только предков текущего графического окна. Обращение к `refreshdata` с одним входным аргументом — указателем на графическое окно или сам рисованный объект — позволяет более точно идентифицировать обновляемые объекты:

```
>> time = 0:0.05:8;
>> fun = sin(time).^2;
>> hF = figure;
>> hA = axes;
>> hL = plot(time,fun);
>> set(hL, 'XDataSource', 'time', 'YDataSource', 'fun');
>> fun = sin(time).^3;
>> refreshdata(hL)
```

Соответствующие переменные (в нашем примере `time` и `fun`) должны существовать в рабочей среде, иначе выводится предупреждение и график не модифицируется. При вызове `refreshdata` из файл-функции возможно использование и локальных переменных, для чего следует указать '`caller`' в качестве второго входного аргумента `refreshdata`.

Все рисованные объекты: `Areaseries`, `Barseries`, `Contourgroup` и т. д. — обладают перечисленными выше свойствами, только у плоских объектов отсутствует свойство `ZDataSource`, а у объекта `Surfaceplot` — `DisplayName`.

Большая часть рисованных объектов является сгруппированными объектами `hggroup`. Исключение составляют только `Lineseries` и `Surfaceplot`. Например, контурный график функции двух переменных — объект `Contourgroup` — состоит из базовых объектов `Patch` и `Text`. Линии уровня отображаются при помощи полигональных объектов, а текстовые служат для подписи значений функции на линиях уровня.

В главе 3 мы обсудили применение высокоуровневых функций `contour` и подобных ей для построения различных типов контурных графиков. Остановимся теперь на ряде свойств объекта `Contourgroup`, полезных при оформлении контурных графиков.

Напомним, что для отображения графика функции двух переменных требуется создать сетку в прямоугольной области на плоскости  $xy$ , вычислить значения функции в узлах сетки и вызвать `contour`, например:

```
>> [X, Y] = meshgrid(-1:0.2:1, -2:0.2:0);
>> Z = sin(pi*X.*Y).* (X + Y);
>> [C, hC] = contour(X, Y, Z)
```

Выходные аргументы функции `contour` — указатель `hC` на сгруппированный объект `Contourgroup` и матрица `C` из двух строк с информацией о линиях уровня. Структура матрицы достаточно проста — каждой линии уровня соответствует группа идущих подряд столбцов. Первый столбец каждой группы содержит значение функции на данной линии уровня и число вершин полигонального объекта, а в следующих столбцах записаны абсцисса и ордината вершин. Матрица `C` является значением свойства `ContourMatrix` объекта `Contourgroup`.

Массивы данных (сетка на плоскости  $xy$  и значения функции в ее узлах) для контурного графика могут быть изменены с привлечением свойств `XData`, `YData` и `ZData`:

```
>> [X, Y] = meshgrid(-1:0.01:1, -2:0.01:0);
>> Z = sin(pi*X.*Y).* (X + Y);
>> set(hC, 'XData', X, 'YData', Y, 'ZData', Z)
```

Основной характеристикой контурных графиков является набор значений исследуемой функции, для которых следует нарисовать линии уровня. По умолчанию эти значения выбираются автоматически с постоянным шагом. Шаг может быть получен или изменен при помощи свойства `LevelStep`, а для обращения к вектору значений функции понадобится свойство `LevelList`. Пусть, например, требуется увеличить шаг в два раза и добавить две линии уровня, на которых исследуемая функция равна  $-0.6$  и  $-1.7$ . Упомянутые два свойства позволяют легко решить эту задачу:

```
>> LS = get(hC, 'LevelStep')
>> set(hC, 'LevelStep', LS*2)
>> LL = get(hC, 'LevelList')
>> LL = [LL -0.6 -1.7]
>> set(hC, 'LevelList', LL)
```

Каждое из свойств `LevelStep` и `LevelList` имеет сопутствующее свойство — `LevelStepMode` и `LevelListMode` соответственно. По умолчанию эти свойст-

ва установлены в 'auto', что приводит к автоматической генерации значений функции, отображаемых линиями уровня. Значение 'manual' выключает автоматическую генерацию — при изменении значения ZData контурный график перестраивается с прежним шагом (LevelStep), или списком значений (LevelList) исследуемой функции. Свойства LevelStepMode и LevelListMode меняют значение 'auto' на 'manual' при установке свойствам LevelStep и LevelList новых значений.

Ряд свойств объекта Contourgroup отвечает за вывод значений функции на линиях уровня. Для отображения значений следует установить свойство ShowText в значение 'on', а для скрытия — в 'off'. По умолчанию маркируются все линии уровня, однако можно ограничить выбор несколькими линиями при помощи свойств TextList или TextStep. Как и в случае LevelList и LevelStep, задается либо вектор из соответствующих значений функции, либо постоянный шаг. Свойства TextListMode и TextStepMode принимают одно из двух значений: 'auto' (по умолчанию) или 'manual'. Значение 'auto' приводит к автоматической разметке всех линий уровня в соответствии со значениями LevelList и LevelStep, а 'manual' отключает автоматическую разметку. Изменение значений TextStep и TextList приводит к установке соответствующих свойств TextStepMode и TextListMode в 'manual'.

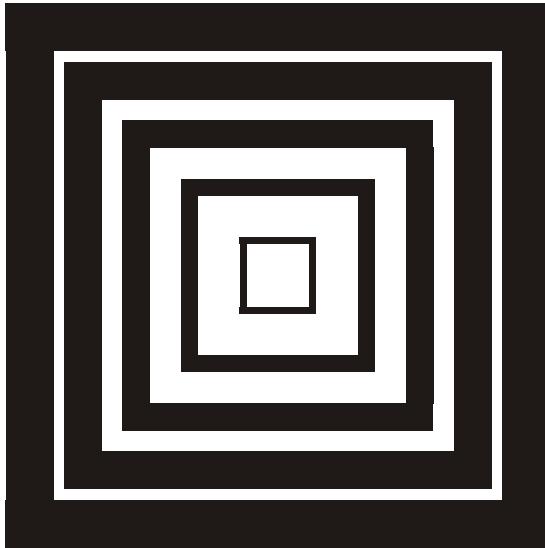
Выбор шрифта и цветовое оформление ярлыков для линий уровня потребуют обращения к потомкам рисованного объекта Contourgroup. Среди них следует выбрать текстовые объекты, т. е. базовые объекты Text, и изменить желаемые свойства. Для поиска потомков с заданными свойствами применяется функция findobj (см. разд. "Управление объектами, копирование, поиск, скрытые указатели" данной главы).

Например, для получения ярлыков зеленого цвета с красной рамкой достаточно выполнить следующие команды:

```
>> hT = findobj(hC, 'Type', 'Text')
>> set(hT, 'BackgroundColor', 'g', 'EdgeColor', 'r')
```

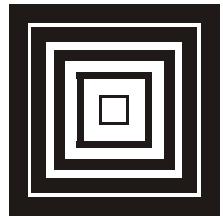
Вид линий уровня, которые являются объектами Patch, определяется значениями свойств LineWidth, LineStyle и LineColor объекта Contourgroup. Для заливки цветом промежутков между линиями уровня в зависимости от значений исследуемой функции служит свойство Fill. Установка его в 'on' приводит к тому же результату, что и обращение к функции contourf.

Мы рассмотрели работу с рисованными объектами на примере Lineseries и Contourgroup. Примеры оформления других типов рисованных объектов приведены в справочной системе MATLAB в разделах с описанием соответствующих высокоуровневых функций. Полная информация о свойствах рисованных объектов может быть получена из браузера свойств графических объектов (см. разд. "Получение информации о свойствах графических объектов" данной главы).



# ЧАСТЬ III

## ПРИЛОЖЕНИЯ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ



## Глава 10

# Принципы создания приложений с GUI

Приложения MATLAB с графическим интерфейсом 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 программирование событий, которые возникают при обращении пользователя к данным объектам, например, при нажатии кнопки. Процесс работы над приложением допускает постепенное добавление элементов в графическое окно, запуск, тестирование приложения и возврат в режим редактирования. Конечным результатом является программа с графическим интерфейсом пользователя (GUI), содержащаяся в одном или нескольких файлах, запуск которой производится указанием ее имени в командной строке или в другом приложении MATLAB.

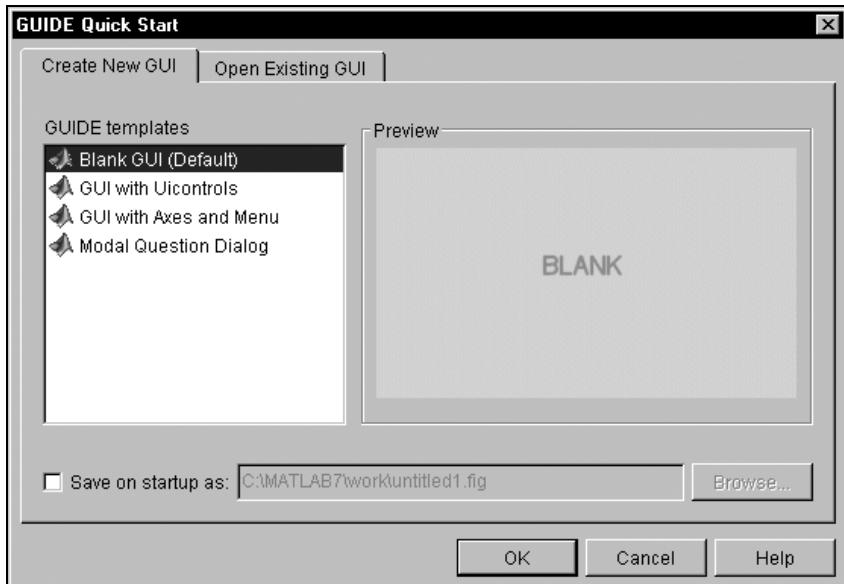
### Примечание

Отметим, что в пакет MATLAB входит Runtime Server, предназначенный для создания приложений, запуск которых не требует установки MATLAB.

Элементы управления являются графическими объектами — потомками графического окна в иерархии объектов. Они могут быть созданы при помощи специальных функций низкоуровневой графики. Однако гораздо эффективнее воспользоваться визуальной средой GUIDE, которая позволяет создать заготовку окна приложения, разместить на ней элементы управления, запрограммировать события и их взаимосвязь.

## Среда GUIDE

Перейдите в визуальную среду GUIDE, выполнив команду `guide` в командной строке. Появляется окно **GUIDE Quick Start**, изображенное на рис. 10.1, которое помогает настроить визуальную среду на создание приложения нужного типа или открыть существующее для продолжения работы.



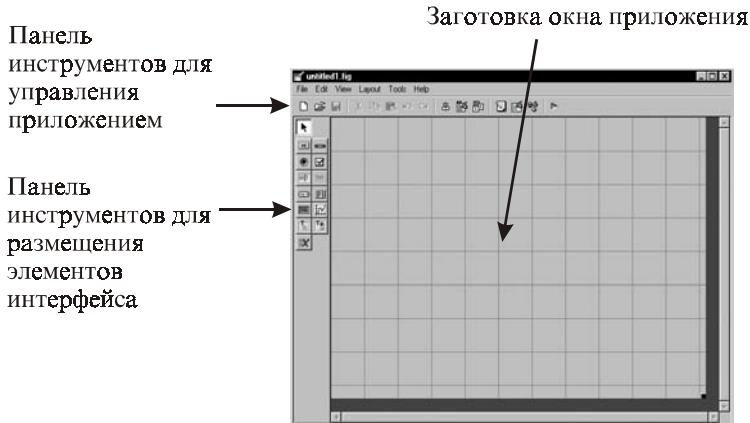
**Рис. 10.1.** Окно быстрого старта для среды GUIDE.

Шаблон для нового приложения выбирается в списке **GUIDE templates** на вкладке **Create New GUI**. Возможен один из следующих вариантов:

- Blank GUI (Default)** — пустое окно приложения без элементов управления, осей и меню;
- GUI with Uicontrols** — окно приложения со строками ввода, переключателями и кнопками;
- GUI with Axes and Menu** — окно приложения, содержащее оси, раскрывающийся список, кнопки и меню;
- Modal Question Dialog** — модальное диалоговое окно с кнопками **Yes** и **No**.

Установка флага **Save on startup as** позволяет задать имя файла, в котором будет содержаться окно приложения.

Мы начнем с пустого окна приложения и будем сами размещать необходимые элементы управления. Имя файла приложения не обязательно выбирать заранее, его всегда можно сохранить из среды GUIDE. Итак, выберите опцию **Blank GUI** и нажмите кнопку **OK**. Появляется редактор среды GUIDE, заголовок которого **untitled.fig** означает, что в нем открыт новый файл, увеличьте немного размеры окна так, как изображено на рис. 10.2.



**Рис. 10.2.** Редактор среды GUIDE

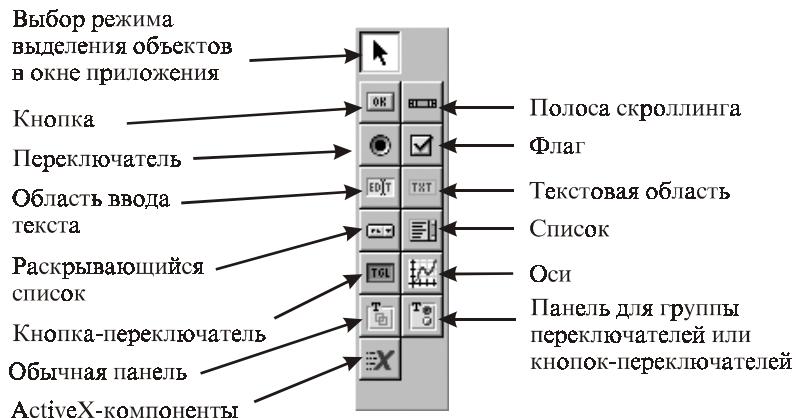
Редактор среды GUIDE содержит:

- строку меню;
- панель инструментов управления приложением;
- заготовку окна приложения с нанесенной сеткой;
- панель инструментов для добавления элементов интерфейса на окно приложения.

Редактор позволяет разместить различные элементы интерфейса (рис. 10.3). Для этого требуется нажать соответствующую кнопку на панели инструментов и поместить выбранный объект щелчком мыши на заготовку 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 кнопкой мыши.

Названия элементов управления появляются на всплывающих подсказках при наведении курсора мыши на кнопки панели инструментов. Впрочем, можно снабдить кнопки соответствующими надписями, для чего следует в

меню **File** среды GUIDE выбрать пункт **Preferences** и в появившемся одноименном диалоговом окне установить флаг **Show names in component palette**.



**Рис. 10.3.** Панель инструментов для добавления элементов интерфейса

Потренируйтесь располагать элементы интерфейса и изменять их размеры, например, получите окно приложения, которое похоже на окно, изображенное на рис. 10.4.

Приложение в данный момент находится в режиме редактирования. Любой объект можно удалить с окна, выделив его и нажав <Delete>. Запуск приложения производится при помощи кнопки **Run** (рис. 10.5). При нажатии на нее появляется диалоговое окно **GUIDE**, которое сообщает о том, что запуск приложения приведет к его сохранению. Установка флага **Do not show this dialog again** позволяет избежать появления этого сообщения в дальнейшем. Нажмите **Yes**, появляется диалоговое окно сохранения файла. Сохраните приложение в файле **tugui0** (расширение **fig** предлагается автоматически).

### Примечание

Для автоматического сохранения приложения перед запуском можнобросить флаг **Show save confirmation on activate** в диалоговом окне **Preferences** с настройками среды GUIDE. Состояние этого флага и флага **Do not show this dialog again** согласованы, если вы установили флаг **Do not show this dialog again**, то флаг **Show save confirmation on activate** при этом сбрасывается. И, наоборот, для появления диалогового окна **GUIDE** с запросом на сохранение приложения перед запуском следует установить флаг **Show save confirmation on activate** в окне **Preferences**.

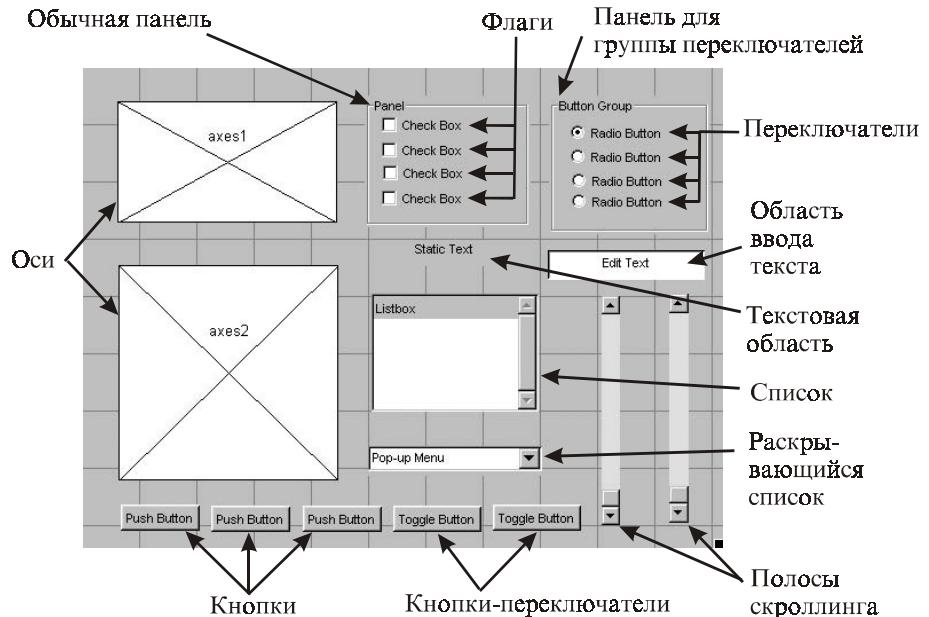


Рис. 10.4. Пример окна приложения с элементами управления

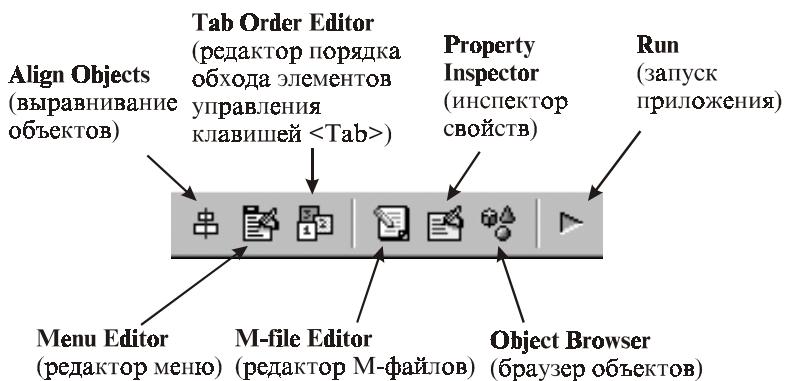


Рис. 10.5. Панель инструментов управления приложением

Приложение запускается в отдельном окне с заголовком **mygui0**. Пользователь может нажимать на кнопки, устанавливать флаги, переключатели, обращаться к спискам, разумеется, при этом ничего полезного не происходит. 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. Обратите внимание, что перед запуском приложения в редакторе M-файлов открылся файл `tugui0.m`, содержащий файл-функцию и ряд подфункций. При чтении следующих разделов вы будете использовать автоматически созданный M-файл для программирования событий элементов интерфейса.

Программирование интерфейса, изображенного на рис. 10.4, является слишком сложным для начинающего программиста. Закройте окно приложения, редактор M-файлов и редактор приложений. Следующий раздел посвящен изучению основ программирования элементов управления на простом примере. Программирование интерфейса более подробно описано в следующей главе.

## Программирование событий

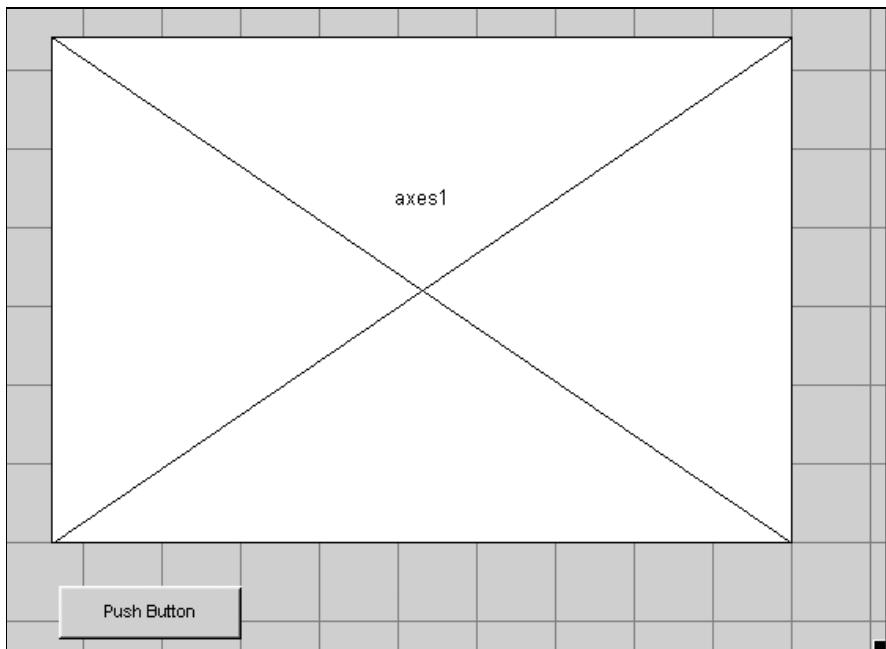
Цель данного раздела состоит в объяснении принципа программирования событий в среде GUIDE. Приложение в MATLAB хранится (по умолчанию) в двух файлах с расширением `fig` и `m`, первый из них содержит информацию о размещенных в окне приложения объектах, а второй является M-файлом с основной функцией и 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 5).

Начните с создания простого приложения, окно которого содержит оси и две кнопки, предназначенные для построения графика и очистки осей.

Перейдите в среду GUIDE командой `guide` с пустым окном приложения так, как было описано в предыдущем разделе. Выберите в меню **Tools** редактора приложений пункт **GUI Options...**, появляется диалоговое окно **GUI Options**. Данное окно позволяет устанавливать некоторые общие свойства создаваемого приложения. Убедитесь, что в раскрывающемся списке **Command-line accessibility** текущей строкой является **Callback (GUI becomes Current Figure within Callbacks)**, включен переключатель **Generate FIG-file and M-file** и установлены все флаги, входящие в группу с ним. В следующей главе мы обсудим смысл всех опций. Сейчас отметим, что выбранные нами значения приводят к автоматической генерации заготовок для подфункций обра-

ботки событий и позволяют использовать окно приложения для графического вывода.

Расположите на форме оси и кнопку так, как показано на рис. 10.6. На кнопке автоматически размещается надпись **Push Button**, а на осях — **axes1**. Следующий этап очень важный. Кнопка и оси являются элементами интерфейса, им следует дать имена, которые уникальным образом идентифицировали бы их среди всех объектов окна приложения (мы добавим их позже) и свидетельствовали об их назначении.



**Рис. 10.6.** Расположение кнопки и осей в окне приложения

Щелчком мыши выделите кнопку **Push Button** и вызовите инспектор свойств **Property Inspector** при помощи панели инструментов управления приложением (рис. 10.5). Появляется окно инспектора свойств, приведенное на рис. 10.7, в котором содержится таблица названий свойств кнопки и их значений. Напомним, что элементы управления являются объектами UI Objects, которые принадлежат графическому окну согласно иерархии графических объектов MATLAB. Если вы внимательно изучили материал главы 9, посвященной дескрипторной графике, то смысл некоторых свойств для вас очевиден (например, `BackgroundColor` или `FontSize`). Важнейшим свойством в данный момент является `Tag`, которое содержит тэг или имя объекта. При добавлении элементов управления им автоматически при-

сваиваются имена, причем по умолчанию они состоят из типа объекта и номера (в нашем случае pushbutton1). Дадим кнопке информативное имя, которые говорит как о типе элемента, так и о его назначении. Эта кнопка будет служить для построения графика, что следует отразить в ее имени.

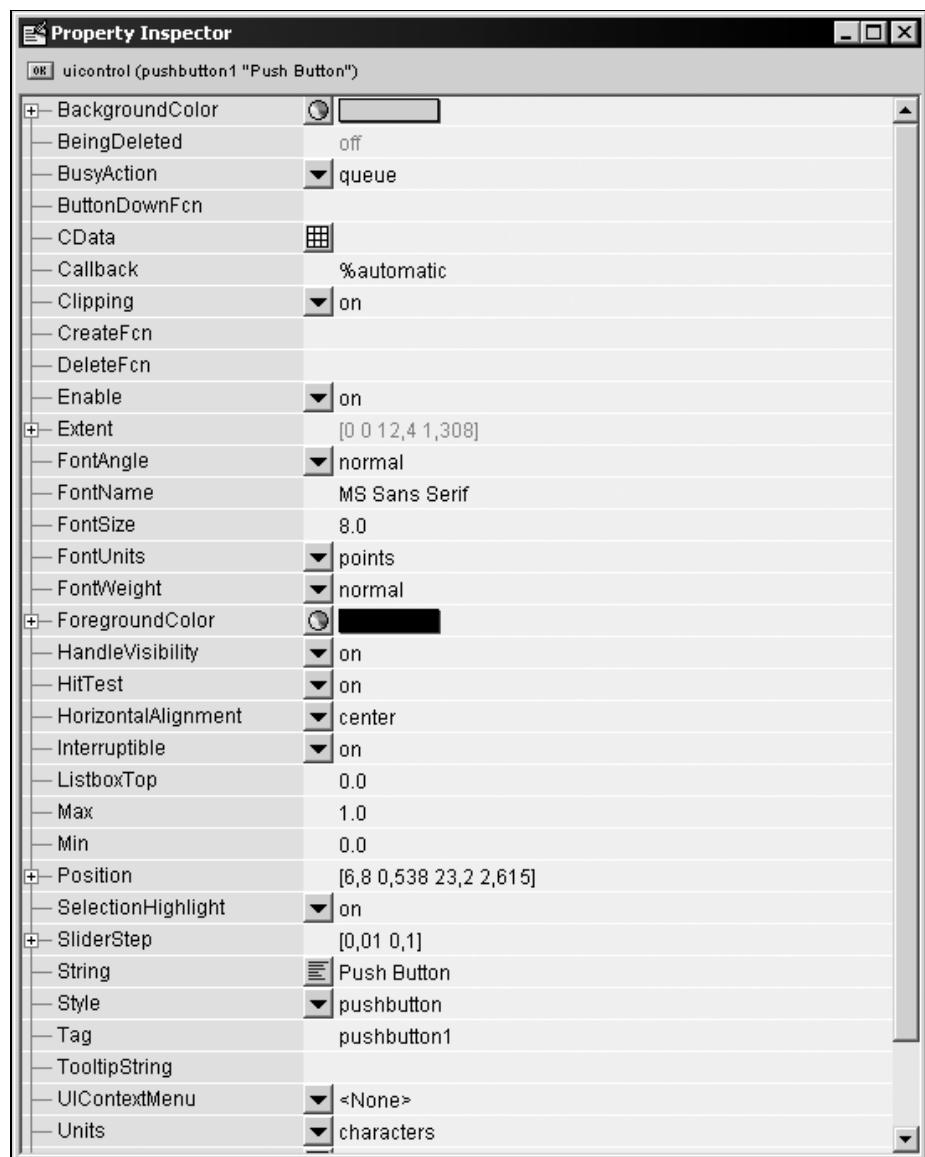


Рис. 10.7. Редактор свойств **Property Inspector**

Установите свойство Tag в значение `btnPlot`, для чего щелкните мышью по строке справа от названия свойства, наберите требуемое значение и нажмите `<Enter>`. Мы выбрали аббревиатуру `btnPlot` исходя из сокращения `btn` для `button` (кнопка) и `Plot` (график).

В дальнейшем будет говориться, что некоторому объекту или элементу управления следует дать имя, или установить его тег в некоторое значение. Аналогичным образом установите тег осей в значение `axMain`.

Следует различать имя объекта и надпись на нем. Сейчас кнопка с именем `btnPlot` содержит текст **Push Button**. Измените его, установив в инспекторе свойств **Property Inspector** свойство `String` в значение Построить. Теперь надпись на кнопке соответствует ее назначению.

При создании приложений с графическим интерфейсом вам придется часто прибегать к инспектору свойств **Property Inspector** для установки свойств объектов в подходящие значения. Не обязательно каждый раз делать объект текущим, а затем открывать окно инспектора свойств **Property Inspector** при помощи соответствующей кнопки на панели инструментов среды GUIDE. Во-первых, если окно **Property Inspector** открыто, то щелчок мышью по объекту приводит к отображению его свойств в этом окне. Во-вторых, щелчок правой кнопкой мыши на любом объекте вызывает появление контекстного меню, содержащего пункт **Property Inspector**. Пожалуй, самое быстрое обращение к **Property Inspector** осуществляется двойным щелчком мыши по объекту. Все эти способы пригодны и для изменения свойств окна приложения.

Сохраните приложение, для чего выберите в меню **File** среды GUIDE пункт **Save as**, создайте папку `MyFirstGui` (например, в текущем каталоге) и дайте имя `mygui.fig` окну приложения. Обратите внимание, что открылся редактор М-файлов, содержащий сопутствующий файл `mygui.m`. Данный файл сформировался автоматически, его структура схематично представлена в листинге 10.1.

#### Листинг 10.1. Структура М-файла приложения с графическим интерфейсом

```
function varargout = mygui(varargin)
...
% --- Executes on button press in btnPlot.
function btnPlot_Callback(hObject, eventdata, handles)
% hObject handle to btnPlot (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

Основная функция `mugui` предназначена для инициализации приложения, поэтому редактировать ее не следует. Сейчас нас будет интересовать подфункция `btnPlot_Callback`. Приложение `mugui` содержит одну кнопку **Построить** с именем `btnPlot`. Когда пользователь нажимает на нее в работающем приложении, то генерируется событие `Callback` данного элемента управления. При этом вызывается подфункция `btnPlot_Callback`. Заметьте, что имя подфункции образовано именем (тегом) кнопки и названием события. Данная подфункция не содержит операторов, и при нажатии на кнопку ничего не происходит.

Завершающий этап состоит в задании действий, которые выполняются при нажатии пользователем на кнопку **Построить**. Запрограммируйте подфункцию `btnPlot_Callback` обработки события `Callback` кнопки **Построить** с именем `btnPlot` в соответствии с листингом 10.2 (автоматически созданные комментарии убирать не обязательно). В дальнейшем мы будем говорить, что требуется запрограммировать событие `Callback` некоторого объекта без указания имени подфункции и подробного описания самого объекта.

#### Листинг 10.2. Обработка события Callback кнопки с именем `btnPlot`

```
function btnPlot_Callback(hObject, eventdata, handles)
x = -2:0.2:2;
y = exp(-x.^2);
plot(x, y)
```

Перейдем теперь к запуску приложения `mugui`. Нажмите кнопку **Run** на панели инструментов среды GUIDE. Поскольку приложение сохранено в отдельном каталоге `MyFirstGui`, то появляющееся диалоговое окно предлагает сделать этот каталог текущим. Убедитесь, что выбран переключатель **Change MATLAB current directory**, и нажмите **OK**. Теперь каталог `MyFirstGui` стал текущим, и при последующих запусках приложения это окно выводиться не будет, при условии, если вы не измените текущий каталог из рабочей среды. Перед запуском может появиться еще одно диалоговое окно, в котором следует подтвердить сохранение файлов с расширениями `fig` и `m`, содержащих приложение (автоматическое сохранение приложения и соответствующая опция среды GUIDE описаны в конце предыдущего раздела этой главы).

После выполнения описанных действий на экране отображается окно приложения `mugui`. Нажатие на кнопку **Построить** приводит к отображению графика функции на осиях. Следующим нашим шагом будет добавление кнопки, предназначенной для очистки осей.

Завершите приложение при помощи кнопки закрытия окна в правом верхнем углу и продолжите работу над `mygui` в среде GIUDE. Добавьте кнопку так, как показано на рис. 10.8, задайте в редакторе свойств имя `btnClear` и надпись **Очистить**. Осталось запрограммировать событие `Callback` этой кнопки. Для быстрого перехода к соответствующей подфункции `btnClear_Callback` воспользуйтесь всплывающим меню, для чего щелкните правой кнопкой мыши по кнопке **Очистить** и выберите в пункте **View Callbacks** подпункт **Callback**. Становится активным окно редактора М-файлов, причем заголовок нужной подфункции выделен. Разместите единственный оператор очистки осей `cla` в подфункции (листинг 10.3).

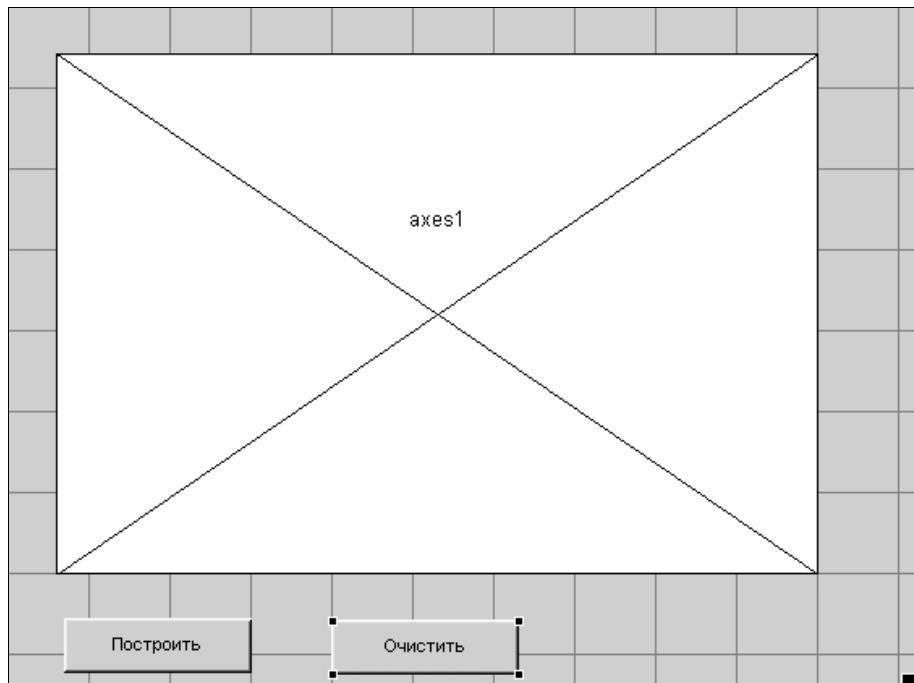


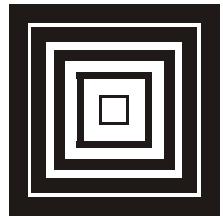
Рис. 10.8. Окно приложения с двумя кнопками

**Листинг 10.3. Обработка события кнопки с именем `btnClear`**

```
function btnClear_Callback(hObject, eventdata, handles)
 cla
```

Запустите приложение и убедитесь, что названия кнопок соответствуют выполняемым действиям.

В этой главе мы рассмотрели основные действия при написании приложений с графическим интерфейсом в среде GUIDE: размещение элементов управления в окне приложения, задание их свойств, программирование событий и запуск приложения. В следующей главе мы обсудим некоторые приемы, применяемые для того, чтобы сделать интерфейс приложения интуитивно понятным. Продолжая работу над приложением *mygui*, вы научитесь использовать другие элементы управления, программировать события и осуществлять их взаимосвязанное поведение.



# Глава 11

## Конструирование интерфейса

Данная глава посвящена продолжению работы над приложением с графическим интерфейсом `mygui`, которая была начата в предыдущей главе. Описан процесс создания приложения, предназначенного для визуализации функций и интерактивного управления видом получающихся графиков. Объяснено, как размещать различные элементы интерфейса в окне приложения и программировать их события в М-файле, связанном с приложением.

### Управление свойствами объектов

Разработка приложения связана с изменением свойств объектов, которые они получают по умолчанию при размещении их на заготовке окна. Элементы управления являются графическими объектами `Uicontrol` и полная информация о назначении их свойств доступна в браузере свойств графических объектов справочной системы MATLAB (использование браузера свойств графических объектов обсуждается в разд. *"Получение информации о свойствах графических объектов"* главы 9).

Некоторые из свойств, например, размер кнопки, надпись на ней и свойства шрифта, устанавливаются при создании объекта в режиме редактирования и изменять их, как правило, не требуется. Ряд свойств объектов необходимо устанавливать программно прямо в ходе работы приложения для обеспечения согласованного поведения элементов управления.

Приступим к усовершенствованию приложения `mygui`. Пусть мы желаем, чтобы при запуске доступной являлась только кнопка **Построить**, а при нажатии на **Построить** выводился график и она становилась недоступной, зато появлялась бы возможность нажать кнопку **Очистить** для удаления графика, и начать все сначала. Итак, всегда доступна только одна из кнопок в зависимости от состояния осей.

Решение поставленной задачи требует привлечения свойства объекта `Enable`, которое отвечает за возможность доступа к нему пользователем, значение '`on`' разрешает доступ, а '`off`', соответственно, запрещает.

Поскольку элементы интерфейса являются графическими объектами, то для изменения их свойств следует прибегнуть к функции `set` (применение функции `set` описано в главе 9).

В данном случае изменяется значение только одного свойства, поэтому функция `set` должна быть вызвана с тремя входными аргументами — указателем на объект, строкой с названием свойства и его значением. Заметьте, что свойства одного объекта должны изменяться в подфункции обработки события `Callback` другого объекта. Следовательно, надо иметь возможность доступа к указателю на любой существующий объект. Требуемые указатели содержат входные аргументы `hObject` и `handles` подфункций, которые обрабатывают события элементов управления. В `hObject` хранится указатель на тот объект, событие которого обрабатывается в данный момент, а `handles` является структурой с указателями на все объекты. Имена полей структуры совпадают со значениями свойств Tag существующих элементов интерфейса. Например, `handles.btnPlot` является указателем на кнопку **Построить** с именем `btnPlot`.

Доступ к кнопке **Очистить** должен быть запрещен в начале работы приложения, пока пользователь не нажмет **Построить**. Установите в инспекторе 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:

- установку свойства `Enable` кнопки **Очистить** в '`on`' (после вывода графика следует разрешить доступ к ней);
- установку свойства `Enable` кнопки **Построить** в '`off`' (после вывода графика она должна стать недоступной).

Аналогичные изменения произведите в обработке события `Callback` кнопки **Очистить**, а именно:

- установку свойства `Enable` кнопки **Построить** в '`on`' (после очистки осей доступ к ней должен быть разрешен);
- установку свойства `Enable` кнопки **Очистить** в '`off`' (после очистки осей следует запретить доступ к кнопке).

Подфункции `btnPlot_Callback` и `btnClear_Callback` должны быть запрограммированы так, как показано в листинге 11.1.

**Листинг 11.1. Обработка событий Callback кнопок btnPlot и btnClear**

```
function btnPlot_Callback(hObject, eventdata, handles)
x = -2:0.2:2;
y = exp(-x.^2);
plot(x, y)

% Кнопка Построить должна стать недоступной после вывода графика
set(hObject, 'Enable', 'off')

% Кнопка Очистить должна стать доступной
set(handles.btnClear, 'Enable', 'on')

function btnClear_Callback(hObject, eventdata, handles)
cla % очистка осей

% Кнопка Очистить должна стать недоступной после очистки осей
set(hObject, 'Enable', 'off')

% Кнопка Построить должна стать доступной
set(handles.btnPlot, 'Enable', 'on')
```

Сохраните изменения в редакторе M-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 и его редактирование

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.

## Примечание

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 продолжить редактирование двумя способами:

1. Выполнить `guide` в командной строке, что приводит к отображению диалогового окна быстрого старта **GUIDE Quick Start**, в котором следует перейти к вкладке **Open Existing GUI** и воспользоваться списком **Recently opened files** или кнопкой **Browse** для поиска нужного файла с расширением `fig`.
2. Указать имя приложения через пробел после команды `guide`, например, `guide mygui`, при этом появляется среда GUIDE, в которой открыто `mygui`. В этом случае каталог с приложением должен быть текущим.

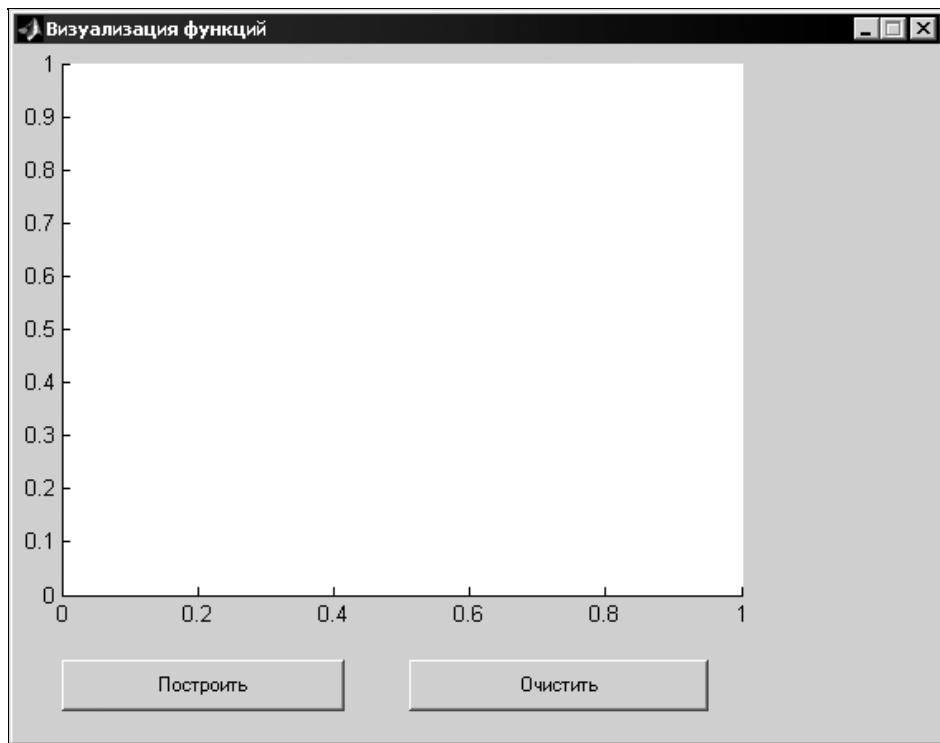


Рис. 11.1. Окно приложения с заголовком

Перейдите в режим редактирования приложения *туда* любым из перечисленных способов и продолжите работу над ним. Измените название окна приложения на "Визуализация функций" так, чтобы работающее приложение *туда* имело вид, изображенный на рис. 11.1. Заголовок окна задается свойством *Name* графического окна. Установите в инспекторе свойств **Property Inspector** свойство *Name* окна приложения в Визуализация функций. Запустите *туда* и убедитесь, что приложение имеет нужный заголовок.

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. Среда GUIDE предоставляет разработчику приложений несколько способов выравнивания добавляемых объектов — сетку, линейку и специальные инструменты.

## Размеры объектов и их выравнивание

Редактор приложений содержит заготовку окна приложения, размер которого изменяется при помощи мыши, удерживая нажатой левую кнопку на квадратике, расположенный в правом нижнем его углу. Для задания точных размеров и положения окна запущенного приложения выберите в инспекторе свойств **Property Inspector** подходящие единицы измерения (пиксели, сантиметры и т. д.), обратившись к свойству *Unit*, и установите свойство *Position* в нужное значение (см. разд. "Управление положением графических окон" главы 9).

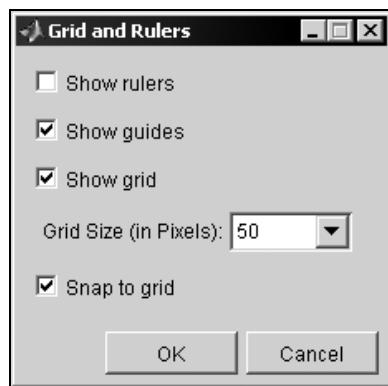
На окно нанесена сетка с достаточно крупным шагом (по умолчанию). Аккуратное расположение большого числа элементов управления требует точного задания их положения. Поскольку элементы управления *Uicontrol* являются графическими объектами (потомками графических окон), то естественно ожидать, что среди их свойств есть предназначенные для задания их размеров и положения в графическом окне. Эти свойства называются *Position* и *Unit*, причем значением *Position* должен быть вектор из четырех чисел (расстояние от левого и нижнего края окна, ширина и высота) в единицах измерения, установленных в *Unit*.

Элементы управления и оси могут быть сгруппированы и размещены на некоторой панели — графическом объекте *Uipanel*. В этом случае они являются потомками объекта *Uipanel* и их положение задается относительно этой панели. Сама панель есть потомок графического окна, поэтому значение свойства *Position* объекта *Uipanel* определяет ее положение в пределах окна. Предком для элементов управления может быть еще одна панель *Uibuttongroup*, которая предназначена для обеспечения согласованного поведения группы переключателей или кнопок-переключателей — когда может быть включен только один элемент управления из всей группы. Для до-

бавления панелей **Uipanel** и **Uibuttongroup** в окно приложения служат, соответственно, кнопки **Panel** и **Button Group** редактора среды GUIDE. Мы используем эти панели для размещения переключателей и флагов в окне нашего приложения *mygui*.

Значения свойств **Position** и **Units** могут быть установлены в инспекторе свойств **Property Inspector** при размещении элементов управления и панелей в окне приложения или в подфункциях обработки событий, если работающее приложение должно изменить вид окна.

Для аккуратного расположения элементов управления и панелей на этапе проектирования интерфейса можно избрать и более простой способ, чем установка свойству **Position** подходящего значения. Он состоит в использовании вспомогательных средств среды GUIDE: сетки, линеек и инструментов выравнивания. Для отображения линеек и доступа к свойствам сетки следует в меню **Tools** выбрать пункт **Grid and Rules**. Появляется диалоговое окно **Grid and Rulers**, изображенное на рис. 11.2.



**Рис. 11.2.** Диалоговое окно **Grid and Rulers**

Установка флага **Show rules** снабдит окно визуальной среды GUIDE горизонтальной и вертикальной линейками. Линейки позволяют размещать элементы интерфейса в позиции с любыми координатами в пикселях, отсчитываемыми от левого нижнего угла заготовки окна приложения. Следует выделить объект (например, кнопку) щелчком мыши и двигать его в окне, следя за указателями его положения на линейке. Однако если установлен флаг **Snap to grid**, то непрерывного движения объекта не происходит — одна или две его границы стремятся совпасть с линиями сетки. Такое поведение перемещаемых объектов называется привязкой к сетке. В сочетании с достаточно мелким шагом сетки, который устанавливается в списке **Grid Size (in Pixels)**, привязка позволит вам быстро оформлять приложение. Плавно из-

менять положение выделенного объекта можно при помощи клавиш со стрелками. Одновременное удержание <Ctrl> приводит к перемещению с учетом привязки к сетке.

Перед размещением элементов управления полезно разметить окно приложения горизонтальными и вертикальными линиями выравнивания. Для этого следует навести курсор мыши на соответствующую линейку (курсор меняет форму на двустороннюю стрелку) и, удерживая левую кнопку мыши, вытащить синюю линию на заготовку для окна приложения. Линии выравнивания отображаются, если установлен флаг **Show guides** в диалоговом окне **Grid and Rules**. 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.

### Примечание

Привязка объектов к линиям выравнивания действует и при сброшенном флаге **Snap to grid**.

Выравнивание группы объектов производится с привлечением специальных средств, доступных в диалоговом окне **Align Objects**, для появления которого следует выбрать в меню **Tools** пункт **Align Objects**. Пиктограммы инструментов этого окна информативны и не требуют комментариев. Отметим только, что перед применением какого-либо из них следует выделить выравниваемые объекты щелчком мыши с одновременным удержанием <Ctrl>.

Отредактируйте вид приложения *mygui*, используя вышеописанные возможности.

## Всплывающие подсказки и пиктограммы

Всплывающие подсказки, которые появляются при наведении курсора мыши на элементы управления, существенно облегчают работу с приложением. Для снабжения элементов управления всплывающими подсказками следует установить их свойству **ToolTipString** требуемое значение. Используйте инспектор свойств **Property Inspector** для добавления всплывающих подсказок к кнопкам **Построить** и **Очистить**, например, "Построение графика функции" и "Удаление графика функции" соответственно (кавычки набирать не следует).

На кнопках и кнопках-переключателях могут быть размещены пиктограммы, поясняющие их назначение. Предположим, что цветная пиктограмма хранится в файле *pict.bmp* с глубиной цвета 24 bit. Сначала необходимо

считать рисунок при помощи функции `imread` в некоторый массив рабочей среды, к примеру `R`:

```
>> R = imread('pict20.bmp', 'bmp');
```

Затем в инспекторе свойств **Property Inspector** следует обратиться к свойству `CData` кнопки или кнопки-переключателя и установить его в значение `R`. Для этого можно либо набрать имя массива в строке ввода справа от названия свойства и нажать `<Enter>`, либо воспользоваться кнопкой, размещенной рядом с названием свойства для открытия диалогового окна **CData**, в котором имя массива вводится в строке **Enter Expression**. При этом в инспекторе свойств **Property Inspector** вместо имени массива отобразится информация о его размерах (трехмерный массив) и типе данных (`uint8`). Пиктограммы отображаются на кнопках после запуска приложения.

Следующие разделы посвящены размещению и программированию основных элементов интерфейса.

## Программирование элементов интерфейса

### Флаги, рамки

Флаги позволяют произвести одну или несколько установок, определяющих ход работы приложения. Продолжите работу над `тугой`, предоставьте пользователю возможность наносить линии сетки на график. Окно приложения должно содержать два флага с названиями `сетка по x` и `сетка по у`. Если пользователь нажимает кнопку **Построить**, то не только строится график функции, но и на оси наносится сетка по выбранным координатам. Нажатие на **Очистить** должно приводить к исчезновению графика функции и скрытию сетки.

Обычно несколько элементов управления со схожим назначением группируются и помещаются на некоторой панели. Мы уже упоминали, что применяются панели двух типов: `Uipanel` и `Uibuttongroup`. В данном случае подойдет обычная панель `Uipanel`, поскольку флаги являются независимыми и могут быть включены или сброшены по отдельности.

Измените размеры осей, освободив справа место для рамки. Нанесите панель на окно приложения при помощи соответствующей кнопки (рис. 10.3). По умолчанию имя панели `uipanel1`, а ее заголовок **Panel**. В инспекторе свойств измените заголовок на **Сетка**, воспользовавшись свойством `Title` объекта `Uipanel`. За положение имени панели отвечает свойство `TitlePosition`, которое по умолчанию имеет значение `'lefttop'`, т. е. в левом верхнем углу панели. Изучите остальные возможные варианты самостоятельно, обратившись к инспектору свойств **Property Inspector** или браузеру свойств графических объектов в справочной системе MATLAB.

На панель добавьте два флага так, как показано на рис. 11.3.

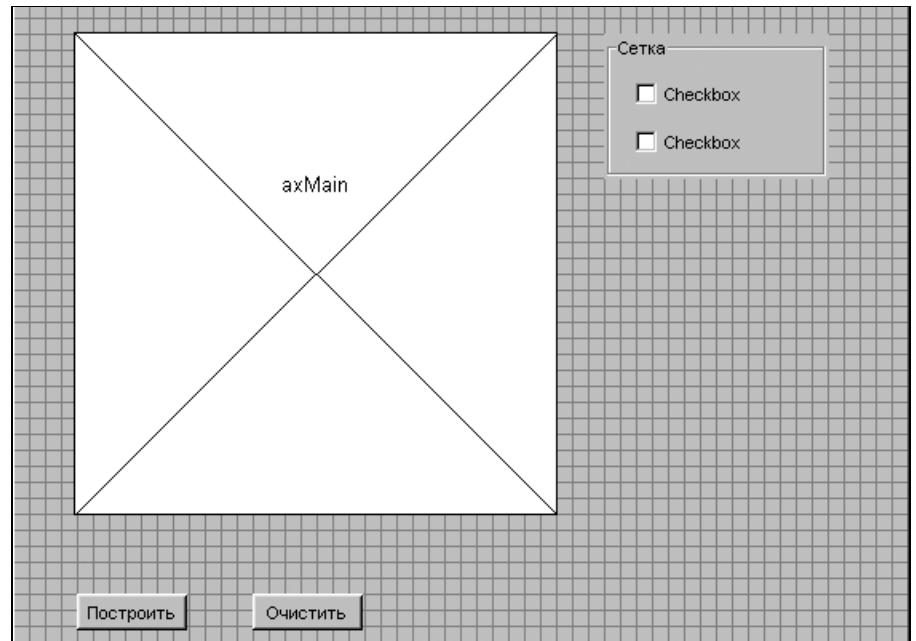
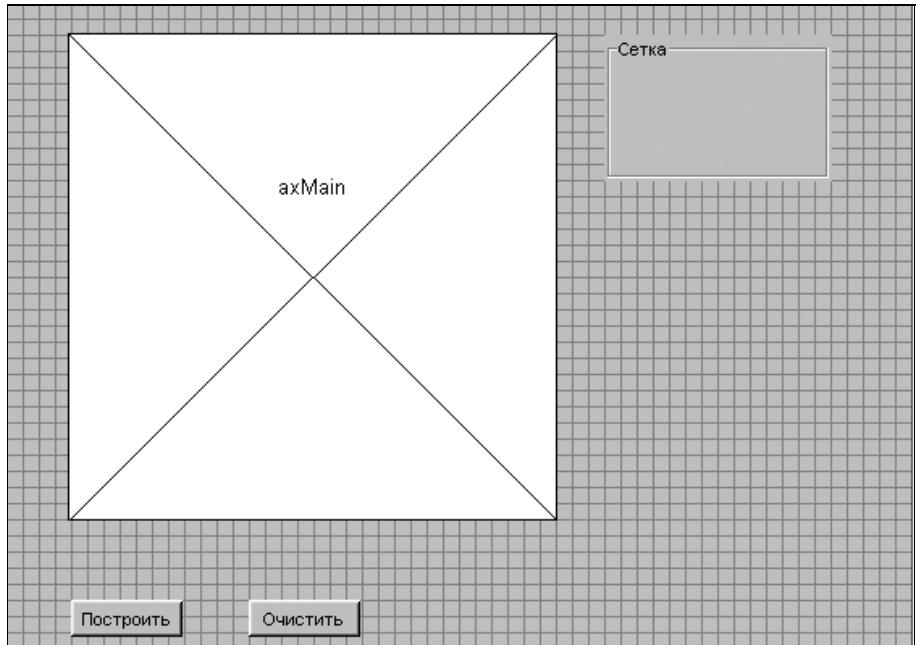


Рис. 11.3. Добавление панели и флагов

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 1 при установке флага пользователем и, соответственно, равно нулю, если флаг сброшен.

### Примечание

У флагов, как и у большинства элементов управления, имеется два свойства Min и Max, принимающие по умолчанию значения 0 и 1 соответственно. Если флаг установлен, то совпадают значения Value и Max, а если сброшен, то Value и Min.

Значение свойства графического объекта позволяет получить функция get, первым входным аргументом которой должен быть указатель на объект, а вторым — название свойства (применение функции get описано в главе 9).

Как и в случае согласования кнопок **Построить** и **Очистить**, в подфункции обработки события Callback одного объекта (кнопки) мы должны получить указатель на другой объект (флаг). Для этого следует обратиться к полям chbxGridX и chbxGridY структуры handles и, в зависимости от их значений, установить свойства осей XGrid и YGrid в 'on' или 'off'.

Произведите необходимые изменения в подфункции обработки события Callback кнопки **Построить** (листинг 11.2).

#### Листинг 11.2. Обработка события Callback кнопки Построить с учетом состояния флагов

```
function btnPlot_Callback(hObject, eventdata, handles)
% 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(hObject, 'Enable', 'off')

% Кнопка Очистить должна стать доступной
set(handles.btnClose, 'Enable', 'on')
```

Запустите приложение `mugui` 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 перейдите к подфункции `chbxGridX_Callback` при помощи всплывающего меню данного объекта. Запрограммируйте событие `Callback` флага. Используйте аргумент  `hObject` соответствующих подфункций, содержащий указатель на объект, событие которого обрабатывается в текущий момент времени. Аналогичным образом обработайте событие `Callback` второго флага **сетка по y** (листинг 11.3).

### Примечание

Для быстрого перехода к подфункциям в редакторе М-файлов предназначена кнопка **Show functions**, нажатие на которую приводит к появлению списка подфункций. Выбор подфункции в этом списке влечет переход к ней в окне редактора.

**Листинг 11.3. Обработка событий Callback флагов сетки**

```
function chbxGridX_Callback(hObject, eventdata, handles)
if get(hObject, 'Value')
 % Флаг включен, следует добавить линии сетки
 set(gca, 'XGrid', 'on')
else
 % Флаг выключен, следует убрать линии сетки
 set(gca, 'XGrid', 'off')
end

function chbxGridY_Callback(hObject, eventdata, handles)
if get(hObject, 'Value')
 % Флаг включен, следует добавить линии сетки
 set(gca, 'YGrid', 'on')
else
 % Флаг выключен, следует убрать линии сетки
 set(gca, 'YGrid', 'off')
end
```

Модифицированные подфункции позволяют пользователю наносить и убирать сетку по каждой координате при помощи флагов без перестройки графика функции.

Самостоятельно внесите дополнения в подфункцию обработки события Callback кнопки **Очистить**, обеспечивающие удаление не только графика функций, но и линий сетки.

Флаги предоставляют пользователю возможность выбора одной или сразу нескольких опций. Одновременный выбор *только одной* опции осуществляется при помощи переключателей.

## Переключатели

Модернизируйте интерфейс приложения mygui, предоставьте пользователю возможность выбирать тип маркера (кружок, квадрат или отсутствие маркера) при помощи трех переключателей. Переключатели обычно группируются по их предназначению, и пользователь может выбрать только одну опцию, т. е. всегда установлен единственный переключатель из группы. Как

мы уже отмечали, для обеспечения такой согласованной работы переключателей служит панель `Uibuttongroup`.

Добавьте в окно приложения панель `Uibuttongroup`, воспользовавшись кнопкой **Button Group** среди GUIDE. Задайте имя **Тип маркера** для этой панели аналогично тому, как вы определяли имя обычной панели при размещении флагов. Нанесите на нее три переключателя. Установите их свойствам `Tag` значения `rbMarkCirc`, `rbMarkSq`, `rbMarkNone`, а `String` — круги, квадраты, без маркеров соответственно (рис. 11.4).

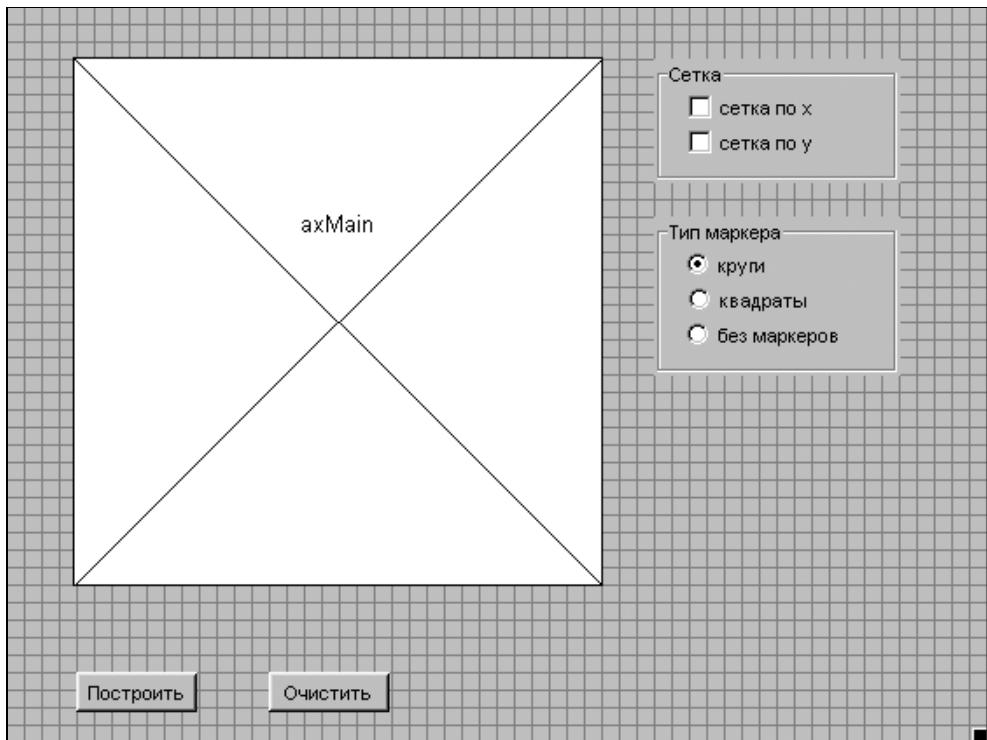


Рис. 11.4. Добавление группы переключателей

Состояние переключателя, так же как и флага, определяется его свойствами `Value`, `Max` и `Min`, причем по умолчанию `Max` установлено в 1, а `Min` — в 0. Для включенного переключателя совпадают значения `Value` и `Max`, а для выключеного — `Value` и `Min`.

Задайте в инспекторе свойств **Property Inspector** значение 1 свойству `Value` переключателя с надписью **без маркеров**, он будет включен при запуске программы. Значение свойства `Value` устанавливается следующим образом.

Выделите переключатель и отобразите его свойства в **Property Inspector**, нажмите кнопку в строке с `Value`. Появляется окно **Value**, изображенное на рис. 11.5.

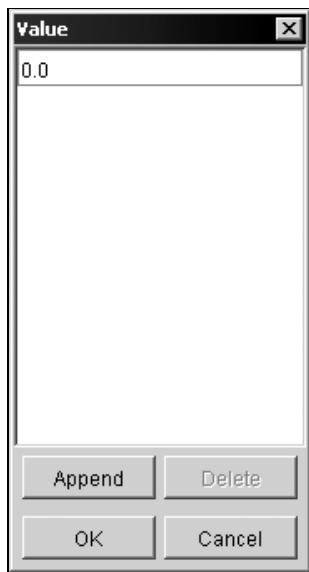


Рис. 11.5. Окно **Value** для установки значения

Выделите при помощи мыши строку со значением 0.0 и перейдите в режим редактирования значения двойным щелчком мыши. Измените 0.0 на 1 и нажмите **OK**. Обратите внимание, что в инспекторе свойств **Property Inspector** значение `Value` изменилось на единицу, и включился переключатель **без маркеров** на заготовке окна приложения. Остальные переключатели панели **Тип маркера** выключены. Предположим, что пользователь установил один из переключателей. Происходит обращение к соответствующей подфункции обработки события `Callback` переключателя, которая должна изменить тип маркеров линии. Обсудим программирование этого события, поскольку оно связано с решением общей проблемы — обменом данными между подфункциями.

Изменение типа маркеров линии не представляет труда — если известен указатель на линию, то достаточно обратиться к свойству линии `Marker` при помощи функции `set` и установить его в нужное значение. Указатель на линию возвращает функция `plot` в выходном аргументе, его следует записать в некоторую переменную, например, `Line`. Использовать указатель на линию придется в других подфункциях, обрабатывающих событие `Callback` пере-

ключателей. Очень важно понять, что переменная Line инициализируется при вызове подфункции btnPlot\_Callback и после ее завершения становится недоступной, поскольку 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 функцией guidata. Тогда входной аргумент — структура handles всех подфункций будет содержать 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)
```

В нашем случае данные, подлежащие сохранению, являются указателем на линию, созданную в результате работы подфункции btnPlot\_Callback. Внесите необходимые изменения в подфункцию btnPlot\_Callback и запрограммируйте обработку событий Callback переключателей в подфункциях rbMarkCirc\_Callback, rbMarkSq\_Callback, rbMarkNone\_Callback (листинг 11.4):

- сохраните указатель на линию в поле Line структуры handles при построении графика командой plot и сохраните обновленную структуру при помощи функции guidata;
- добавьте операторы обработки событий Callback переключателей, каждый из которых наносит на линию соответствующие маркеры или убирает их.

#### Листинг 11.4. Обработка событий переключателей в myuiprog

```
function varargout = btnPlot_Callback(hObject, eventdata, handles)
% 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(hObject, eventdata, handles)
% Выбран переключатель маркеры-круги
set(handles.Line, 'Marker', 'o') % размещение маркеров-кругов на линии

function varargout = rbMarkSq_Callback(hObject, eventdata, handles)
% Выбран переключатель маркеры-квадраты
set(handles.Line, 'Marker', 's') % размещение маркеров-квадратов на линии

function varargout = rbMarkNone_Callback(hObject, eventdata, handles)
% Выбран переключатель без маркеров
set(handles.Line, 'Marker', 'none') % удаление маркеров с линии

```

Запустите приложение *mugui*, отобразите 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 или их удалению. Однако пока еще интерфейс *mugui* имеет ряд недостатков.

- Если пользователь использует **Очистить** для удаления графика, а затем устанавливает любой переключатель, то производится обращение к несуществующему объекту линии (сообщение об ошибке выводится в командное окно).
- Нажатие на **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 *muguir.hgobj*, 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 *<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 11.5, содержащему требуемые операторы.

#### Листинг 11.5. Разрешение и запрещение доступа к группе переключателей

```
function btnPlot_Callback(hObject, eventdata, handles)
...
% Все переключатели должны стать доступными после появления графика
set(handles.rbMarkCirc, 'Enable', 'on')
set(handles.rbMarkSq, 'Enable', 'on')
set(handles.rbMarkNone, 'Enable', 'on')

function btnClear_Callback(hObject, eventdata, handles)
...
% 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 11.6).

#### Листинг 11.6. Изменение маркеров при построении графика

```
function btnPlot_Callback(hObject, eventdata, handles)
...
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
```

Теперь 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.

- Свойство `Enabled` переключателя, событие `Callback` которого обрабатывается, должно иметь значение `inactive`, а для остальных двух `on`. Если не задать `on` для других переключателей, то все они станут неактивными.
- При нажатии на **Построить** свойству `Enable` всех переключателей присваивается `on`, а затем определяется установленный в данный момент переключатель и в `Enable` заносится значение `inactive`.

Дополните подфункции `btnPlot_Callback`, `rbMarkCirc_Callback`, `rbMarkSq_Callback` и `rbMarkNone_Callback` необходимыми операторами (листинг 11.7).

#### Листинг 11.7. Предотвращение выключения переключателя повторным щелчком мыши

```
function btnPlot_Callback(hObject, eventdata, handles)
...
% 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 rbMarkCirc_Callback(hObject, eventdata, handles)
...
% Переключатель, событие которого обрабатывается, должен стать
% неактивным, а остальные — активными
set(hObject, 'Enable', 'inactive')
set(handles.rbMarkSq, 'Enable', 'on')
set(handles.rbMarkNone, 'Enable', 'on')

function rbMarkSq_Callback(hObject, eventdata, handles)
...
% Переключатель, событие которого обрабатывается, должен стать
% неактивным, а остальные — активными
set(hObject, 'Enable', 'inactive')
set(handles.rbMarkCirc, 'Enable', 'on')
set(handles.rbMarkNone, 'Enable', 'on')

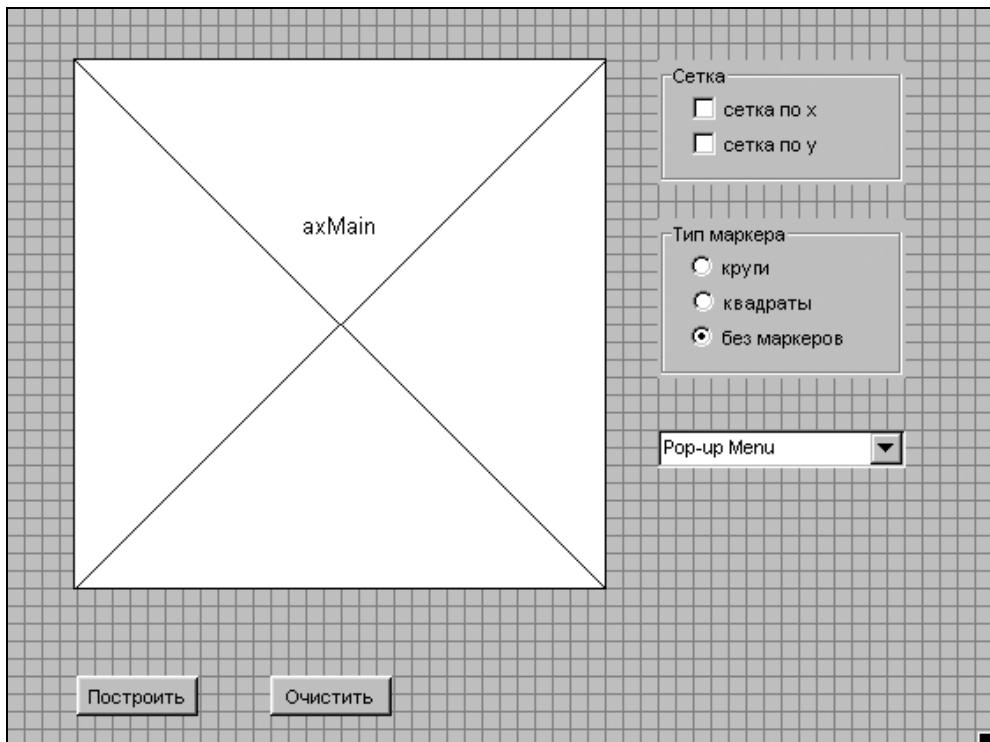
function rbMarkNone_Callback(hObject, eventdata, handles)
...
% Переключатель, событие которого обрабатывается, должен стать
% неактивным, а остальные — активными
set(hObject, 'Enable', 'inactive')
set(handles.rbMarkSq, 'Enable', 'on')
set(handles.rbMarkCirc, 'Enable', 'on')
```

Правильная обработка переключателей (см. листинги 11.4—11.7) требует учета всех ситуаций, которые могут возникнуть при взаимодействии пользователя с приложением.

Использование кнопок-переключателей ничем не отличается от переключателей. Альтернативный способ выбора пользователем только одной из предлагаемых опций реализуют раскрывающиеся списки.

## 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. 11.6). Установите в инспекторе свойств **Property Inspector** Tag в значение pmColor.



**Рис. 11.6.** Добавление раскрывающегося списка

Элементами раскрывающегося списка являются строки, которые вводятся в инспекторе свойств **Property Inspector**. Нажмите кнопку в строке со свойством **String** раскрывающегося списка, появляется окно **String**. В области ввода текста замените текст "Pop-up Menu" на строки "синий", "красный", "зеленый" (без кавычек), разделяя их при помощи <Enter> (рис. 11.7) и нажмите **OK**.

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, которые 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 11.8).

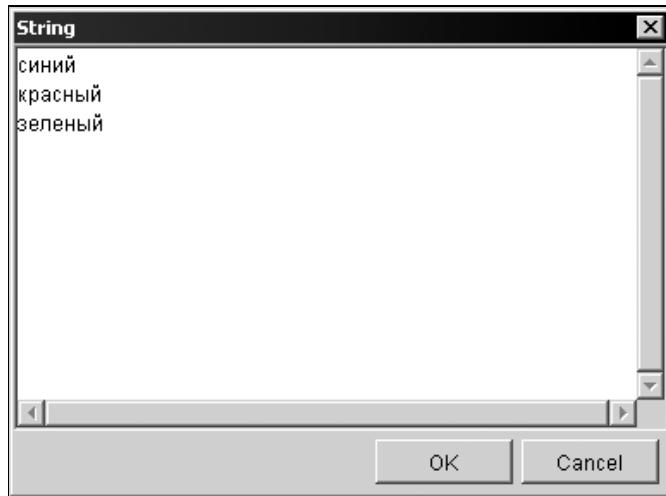


Рис. 11.7. Окно String

**Листинг 11.8. Обработка выбора пользователя из раскрывающегося списка**

```
function pmColor_Callback(hObject, eventdata, handles)
% Определение номера выбранной строки
Num = get(hObject, '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 зеленоой
 set(handles.Line, 'Color', 'g');
end
```

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 еще 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.

Устраним первый недостаток, поместите в подфункции `btnPlot_Callback` обработки нажатия кнопки **Построить** блок `switch` для задания цвета построенной линии в зависимости от состояния раскрывающегося списка (листинг 11.9).

#### Листинг 11.9. Учет состояния раскрывающегося списка при построении графика

```
function varargout = btnPlot_Callback(hObject, eventdata, handles)
...
% 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, на-против, разрешить, при построении графика. В начале работы приложения список должен быть недоступен для пользователя. Установите в инспекторе свойств **Property Inspector** для раскрывающегося списка `Enable` в значение `off`. Внесите необходимые дополнения в подфункции `btnPlot_Callback` и `btnClear_Callback`, соответствующие нажатию на кнопки, используйте свойство списка `Enable` (листинг 11.10).

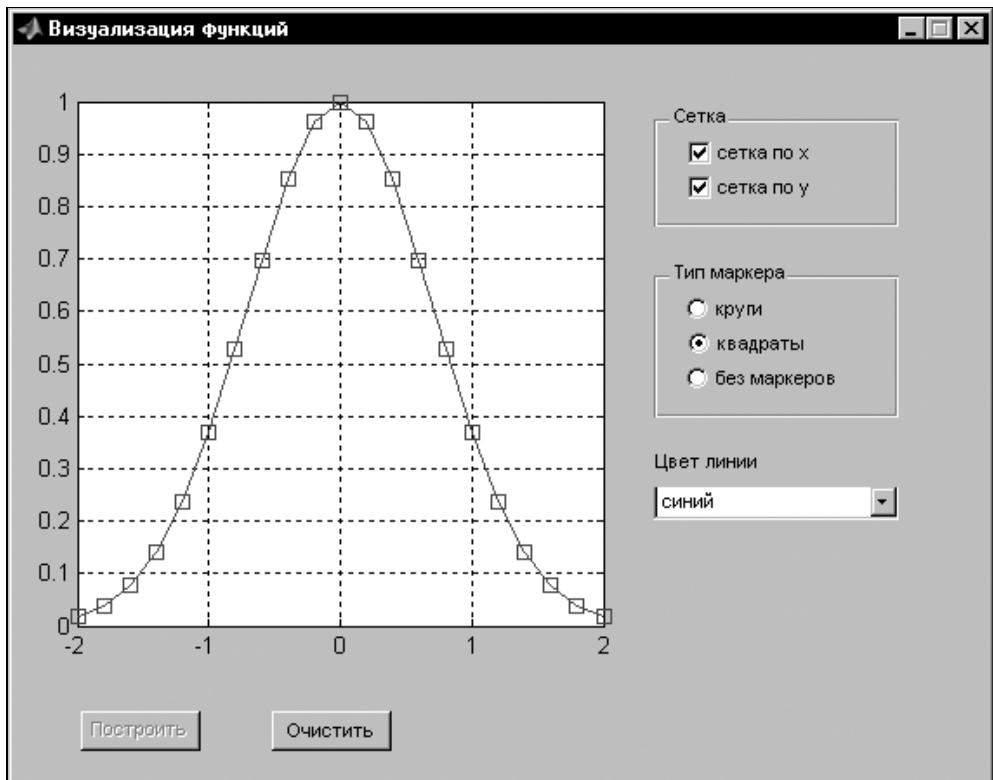
**Листинг 11.10. Разрешение и запрещение доступа к раскрывающемуся списку**

```

function varargout = btnPlot_Callback(hObject, eventdata, handles)
...
% Разрешение доступа к раскрывающемуся списку
set(handles.pmColor, 'Enable', 'on')

function varargout = btnClear_Callback(hObject, eventdata, handles)
...
% Запрещение доступа к раскрывающемуся списку
set(handles.pmColor, 'Enable', 'off')

```

**Рис. 11.8.** Добавление текста

Многие элементы интерфейса, в частности, раскрывающиеся списки, следуют сопровождать поясняющим текстом. Перейдите в режим редактирования

и при помощи панели управления разместите текстовую область (Static Text) над списком. Установите в инспекторе свойств **Property Inspector** String в Цвет линии, а HorizontalAlignment в left для добавленного объекта, используйте кнопки в строках с названиями свойств. Теперь работающее приложение имеет более наглядный интерфейс (рис. 11.8).

Программирование события callBack обычных списков (Listbox) производится практически аналогично. Отличие состоит в том, что в обычных списках может быть выделено несколько элементов, массив номеров которых будет являться значением свойства Value. Разрешение выбора нескольких элементов определяется значениями свойств Max и Min. Если разность Max – Min больше единицы, то пользователь может выделить несколько строк.

Содержимое списков может быть установлено или изменено из программы, для чего формируется массив строк и устанавливается в качестве значения свойства String. Значением свойства String может также быть массив ячеек, содержащих строки, или одна строка, в которой элементы списка разделены символом вертикальной черты |.

## Полосы скроллинга

Усовершенствуйте интерфейс приложения mygui, предоставив пользователю возможность устанавливать ширину линии при помощи полосы скроллинга. Добавьте полосу скроллинга (Slider) в окно приложения и задайте ей имя scrWidth в свойстве Tag. Снабдите полосу скроллинга текстовым пояснением "Толщина линии" так же, как и раскрывающийся список (рис. 11.9).

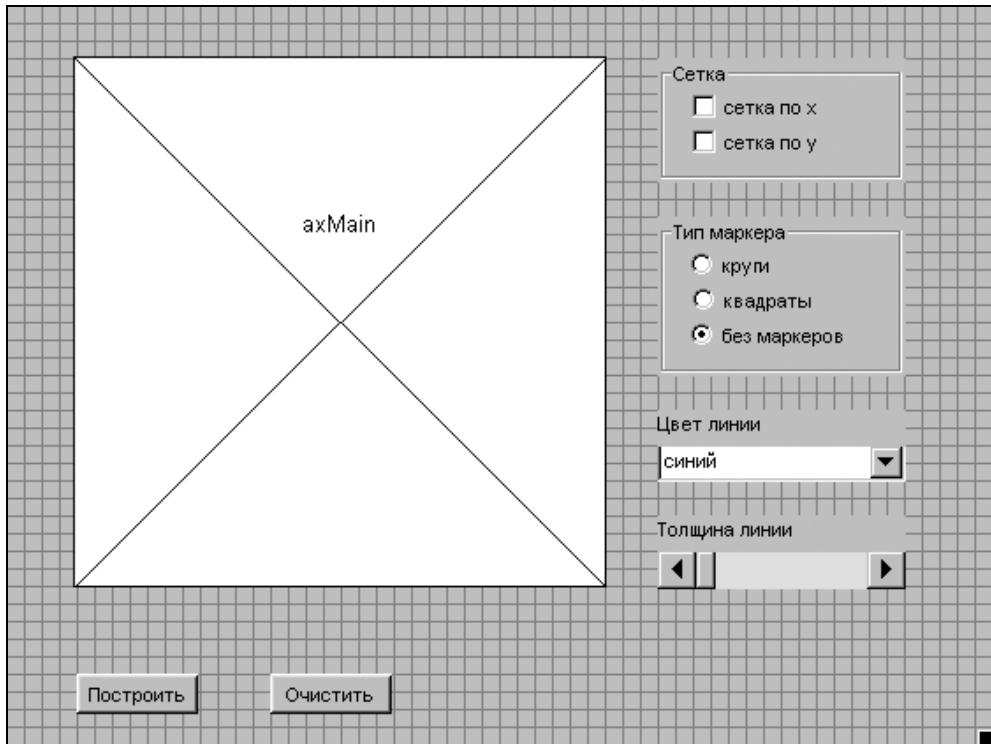
### Примечание

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 ее 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.



**Рис. 11.9.** Добавление полосы скроллинга

2. Определите начальное положение, записав в **Value** единицу. Нажмите кнопку в строке с названием свойства, и в появившемся окне **Value** (см. рис. 11.5) измените значение на единицу.
3. Обратитесь к свойству **SliderStep**. Его значением является вектор из двух компонент, первая из которых определяет относительное изменение **Value** при нажатии на кнопки со стрелками полосы скроллинга, а вторая — при перетаскивании бегунка мышью. Для того чтобы нажатие на кнопки полосы изменяло **Value** на десять процентов, а щелчок мыши справа или слева от бегунка на двадцать, следует установить значение **[0.1 0.2]** свойства **SliderStep**. 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. 11.10).

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 11.11).

|            |           |
|------------|-----------|
| SliderStep | [0,1 0,2] |
| x          | 0.1       |
| y          | 0.2       |

Рис. 11.10. Ввод значений SliderStep

**Листинг 11.11. Обработка события Callback полосы скроллинга**

```
function scrWidth_Callback(hObject, eventdata, handles)
% Получение ширины линии в зависимости от положения бегунка
% на полосе скроллинга
width = get(hObject, 'Value');
% Установка толщины линии
set(handles.Line, 'LineWidth', round(width))
```

Запустите mygui и убедитесь, что полоса скроллинга позволяет легко изменять толщину линии построенного графика. Устранитe самостоятельно некоторые недочеты интерфейса. Полоса скроллинга должна быть недоступной после очистки осей кнопкой **Очистить**, построение графика при помощи **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.

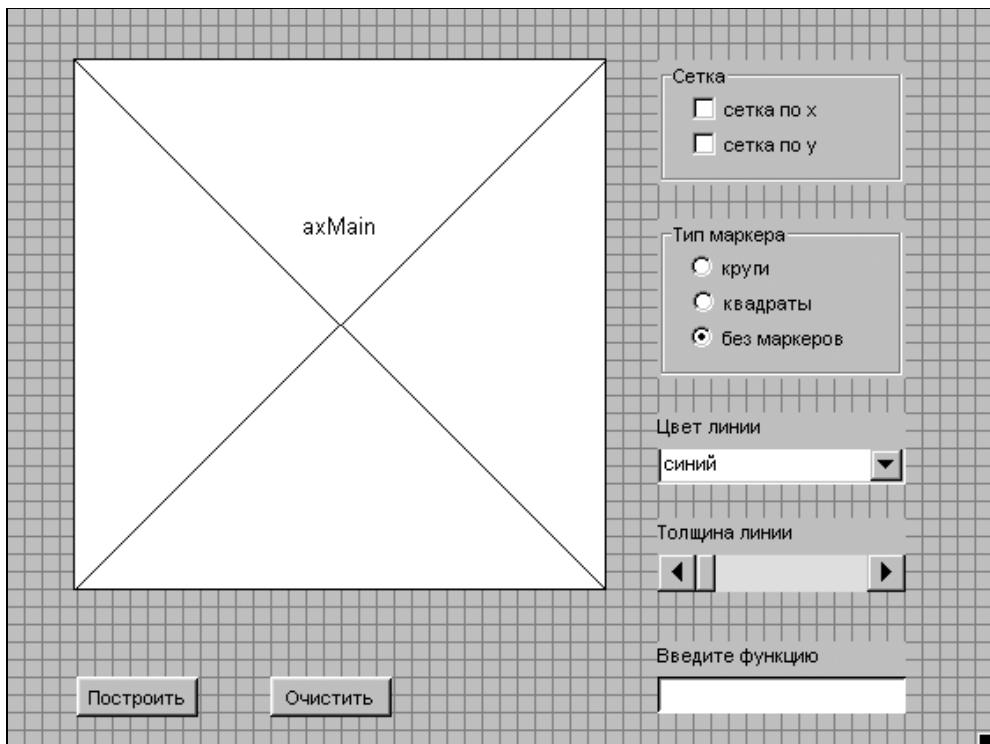
## Область ввода текста

Обычные текстовые области (Static Text), 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 ввести выражение для функции, график которой требуется построить.

Добавьте в окно приложения область ввода текста, установите ее свойство Tag в значение editFun и снабдите ее пояснением в текстовой области, расположенной выше так, как показано на рис. 11.11. При запуске приложения mygui строка для ввода функции должна быть пустой, поэтому в редакторе свойств удалите из String строку Edit Text.

Набранная строка в области ввода должна содержать выражение для функции с учетом правил записи поэлементных операций. Содержимое этой строки является значением свойства String области ввода, его надо будет

преобразовать в inline-функцию при программировании события кнопки **Построить** и вычислить вектор значений функции (создание inline-функций описано в разд. "Встраиваемые и анонимные функции" главы 6).



**Рис. 11.11.** Добавление области ввода текста

С другой стороны, если курсор находится внутри области ввода, то нажатие <Enter> вызовет возникновение ее события `CallBack`. Разумно предусмотреть, чтобы в этом случае построился график функции и произошли все события, запрограммированные в подфункции обработки события `CallBack` кнопки **Построить**. Для этого вовсе не требуется копировать операторы подфункции `btnPlot_Callback` в подфункцию `editFun_Callback` — достаточно просто вызвать `btnPlot_Callback`. Первым ее входным аргументом должен быть указатель на кнопку **Построить**, поскольку `hObject` является указателем на тот объект, событие которого обрабатывается, а в данном случае мы программно моделируем нажатие кнопки пользователем. Листинг 11.12 содержит операторы для построения графика функции, введенной пользователем.

**Листинг 11.12. Обработка событий Callback кнопки Построить и области ввода текста**

```
function btnPlot_Callback(hObject, eventdata, handles)
...
x = -2:0.2:2;
funstr = get(handles.editFun, 'String');
fun = inline(funstr);
y = fun(x);
handles.Line = plot(x, y);
...

function editFun_Callback(hObject, eventdata, handles)
btnPlot_Callback(handles.btnPlot, eventdata, handles)
```

Доработайте интерфейс приложения, обеспечив согласованное поведение кнопок и области ввода текста.

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 *String*.

## Свойства приложения

Приложения с графическим интерфейсом имеют ряд свойств, отвечающих за взаимодействие приложения с пользователем и рабочей средой MATLAB.

Для доступа к основным опциям следует выбрать в меню **Tools** визуальной среды пункта **GUI Options**. Появляется одноименное диалоговое окно, изображенное на рис. 11.12.

Обсудим возможности по настройке графического приложения средствами данного окна.

## Изменение размеров приложения

Раскрывающийся список **Resize behavior** окна **GUI Options**, приведенного на рис. 11.12, предназначен для выбора способа изменения размеров работающего приложения. По умолчанию установлено **Non-resizable**, что не позво-

ляет пользователю менять размеры окна приложения и часто оказывается удобным. Выбор **Proportional** приводит к пропорциональному изменению размеров и положения всех элементов управления, в том числе и тех, которые хотелось бы сохранить постоянными (размеры кнопок, ширина и длина областей ввода и полос скроллинга, размеры панелей и расположение флагов и переключателей в них). Более гибкого управления размерами позволяет добиться опция **Other (Use ResizeFcn)**, ее и следует выбрать.

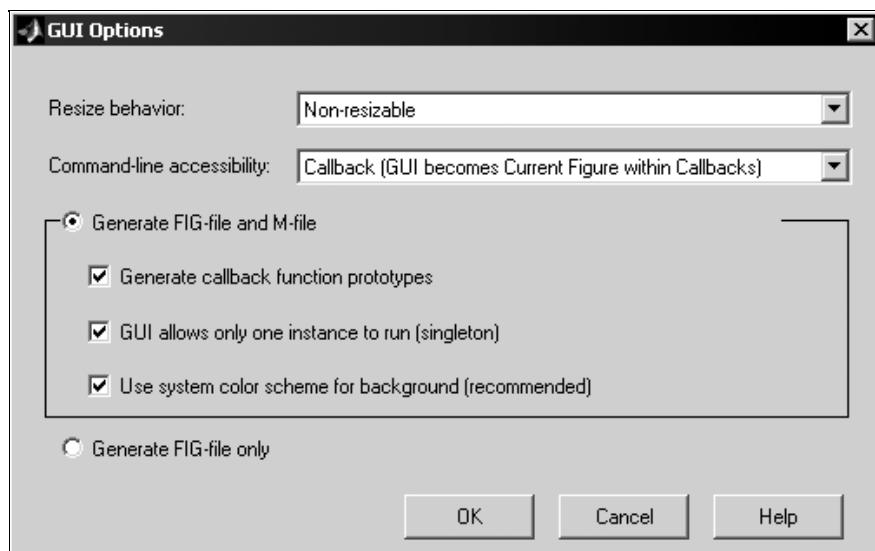


Рис. 11.12. Диалоговое окно **GUI Options** с основными настройками приложения

Мы рассмотрим обработку только нескольких элементов управления нашего приложения `mygui`, поскольку поведение остальных программируется аналогичным образом. В окне приложения есть два флага: **сетка по x** и **сетка по y** с именами `chbxGridX` и `chbxGridY`, размещенные на панели **Сетка** с именем `uipanel1`. Сделаем так, чтобы расстояние между левым краем рамки и правым краем окна всегда было 180 пикселов, между нижним ее краем и верхним краем окна — 90 пикселов, а ширина и высота рамки составляли бы 160 и 70 пикселов соответственно. Флаги будут сохранять свое положение и размеры относительно неизменяющейся рамки, поскольку являются ее потомками.

При изменении размеров окна возникает событие окна `ResizeFcn`, которое программируется по своему усмотрению. Для перехода к подфункции `figure1_ResizeFcn` следует раскрыть всплывающее меню заготовки окна приложения и в пункте **View Callbacks** выбрать подпункт **ResizeFcn**.

Сам процесс программирования, приведенный в листинге 11.13, состоит из пяти этапов.

1. Запоминание единиц измерения размеров и положения всех необходимых объектов.
2. Приведение единиц измерения объектов к единой системе (в нашем случае удобно взять пиксели).
3. Получение текущего положения и размера окна приложения.
4. Задание положения и размеров интересующих нас элементов управления в окне приложения.
5. Возврат к старым единицам измерения.

#### Листинг 11.13. Обработка размеров элементов управления

```
function figure1_ResizeFcn(hObject, eventdata, handles)
% Запоминание единиц измерения графического окна
% и панели Сетка с флагами
u_fig = get(gcf, 'Units');
u_uipanel1 = get(handles.uipanel1, 'Units');
% Установка единиц измерения - пикселя
% для графического окна и панели Сетка
set(hObject, 'Units', 'pixels')
set(handles.uipanel1, 'Units', 'pixels')
% Определение положения и размеров графического окна
p_fig = get(hObject, 'Position');
% Установка положения и размеров рамки
% в окне приложения в пикселях
x = p_fig(3) - 180;
y = p_fig(4) - 90;
width = 160;
height = 70;
set(handles.uipanel1, 'Position', [x, y, width, height])
% Возврат к прежним единицам измерения
% графического окна и панели Сетка
set(hObject, 'Units', u_fig)
set(handles.uipanel1, 'Units', u_uipanel1)
```

Убедитесь, что изменение размеров запущенного приложения не влияет ни на размер рамки и флагов, ни на их положение. Самостоятельно запрограммируйте схожее поведение кнопок, переключателей и других объектов. Учтите, что в ряде случаев имеет смысл допускать автоматическое увеличение некоторых объектов, например, осей или области ввода текста.

## Взаимодействие приложения со средой MATLAB

Приложение является графическим окном со своими объектами и при его создании важно решить, сможет ли пользователь менять вид приложения собственными командами, например, из командной строки. Наше приложение `mogui` содержит оси, однако попытка построить график из командной строки приведет к появлению нового графического окна со своими осями. Дело в том, что по умолчанию в раскрывающемся списке **Command-line accessibility** диалогового окна **GUI Options** (см. рис. 11.12) выбрана опция **Callback (GUI becomes Current Figure within Callbacks)**, соответствующая значению '`'callback'`' свойства `HandleVisibility` графического окна приложения. Графическое окно становится доступным только при выполнении какого-либо события `Callback`, в частности, на его оси может быть выведен график. Эта опция должна быть установлена, если предполагается графический вывод на оси окна приложения.

Выбор **Off (GUI never becomes Current Figure)** приводит к тому, что указатель на него становится скрытым, т. е. его свойство `HandleVisibility` принимает значение '`'off'`' и все выполняемые графические команды приведут к созданию новых окон. Проверьте это, установив данное свойство нашему приложению `mogui`. Нажатие на кнопку **Построить** приведет к появлению отдельного окна с графиком функции.

Третья опция **On (GUI may become Current Figure from Command Line)** предназначена для тех приложений, которые допускают изменение свойств внешними командами: из командной строки или другого приложения.

Выбор последней опции **Other (Use settings from Property Inspector)** дает возможность произвести нужные установки в инспекторе свойств **Property Inspector** или изменять их в ходе выполнения приложения.

Каждая из перечисленных опций соответствует открытому либо скрытому указателю на графическое окно, смысл которых обсуждался в разд. "Управление объектами, копирование, поиск, скрытые указатели" главы 9.

Таблица этих соответствий приведена в справочной системе MATLAB. (см. разд. **Creating Graphical User Interfaces: Laying Out GUIs and Setting Properties: Selecting GUI Options: Command-Line Accessibility**).

Первый запуск приложения с графическим интерфейсом вызывает появление на экране его окна. При повторном наборе имени приложения в командной строке возможны две ситуации: либо окно приложения становится активным (по умолчанию), либо запускается копия приложения. Для разрешения запуска нового окна приложения следуетбросить флаг **GUI Allows Only One Instance to Run (Singleton)**.

Флаг **Using the System Background Colors** отвечает за цветовое оформление приложения. Если он включен, то по умолчанию цвета элементов управления будут совпадать с определенными в операционной системе. При работе в Windows вы можете изменить цветовую схему оформления в диалоговом окне **Display Properties** на вкладке **Appearance** и проследить за изменением цветами элементов управления приложения при различных состояниях флага **Using the System Background Colors**. Впрочем, используя свойство графических объектов **BackgroundColor**, всегда можно задать любой цвет элементу управления.

## Способы программирования событий

Работая над рассмотренным выше приложением *тугці* вы использовали самый простой, но эффективный способ, при котором для нового элемента управления автоматически генерируется заголовок подфункции обработки события **Callback** или **ResizeFcn** и остается лишь запрограммировать подфункцию. Таким образом, само приложение хранится в двух файлах — с расширениями *fig* и *m*. М-файл содержит основную функцию, которая запускается при наборе имени приложения в командной строке и подфункции обработки событий элементов управления. Информация о расположении и свойствах графических объектов, составляющих приложение, записана в файле с расширением *fig*. Такая структура приложения обеспечивается включенным переключателем **Generate Fig-file and M-file** в диалоговом окне **GUI Options** (см. рис. 11.12). Включенный переключатель обеспечивает доступ к трем флагам, включение первого из них **Generate Callback Function Prototypes** приводит к автоматической генерации заголовков подфункций обработки событий.

Альтернативный способ конструирования приложения предполагает использование только одного файла с расширением *fig*, что достигается установкой переключателя **Generate FIG-file only** в диалоговом окне **GUI Options**. При этом свойства каждого из элементов управления, отвечающие за соответствующее событие (например, свойство **Callback** для одноименного события), должны содержать строку с командой, файл-функцией или файл-программой, в которой запрограммированы соответствующие действия.

Обратитесь к нашему приложению *тугці*, которое было создано с привлечением как *fig*, так и М-файла, и изучите в редакторе свойств значение свой-

ства Callback, например, кнопки **Построить**. Этим значением является следующее обращение к файл-функции приложения:

```
mygui('btnPlot_Callback', gcbo, [], guidata(gcbo))
```

Интерфейс файл-функции mygui предполагает обращение к ней с произвольным числом входных аргументов, например, как в данном случае (см. разд. "Файл-функции с переменным числом аргументов" главы 8).

В основной функции mygui работает блок операторов, приведенный в листинге 11.14, который при помощи встроенной функции str2func формирует обращение к соответствующей подфункции btnPlot\_Callback с подходящими входными аргументами.

#### Листинг 11.14. Блок основной функции mygui, вызывающий требуемую подфункцию

```
if nargin & isstr(varargin{1})
 gui_State.gui_Callback = str2func(varargin{1});
end
```

Кроме Callback и ResizeFcn имеется еще ряд событий, назначение которых мы обсудим в главе 13.

## Порядок обхода элементов управления клавишей <Tab>

В работающем приложении пользователь осуществляет переход к элементам управления при помощи мыши или клавиши <Tab>. При использовании <Tab> элементы управления обходятся в порядке их создания, что не всегда желательно. Для изменения порядка обхода служит редактор **Tab Order Editor**, который запускается выбором в меню **Tools** пункта **Tab Order Editor**. Появляется окно редактора, изображенное на рис. 11.13.

Работа в нем достаточно проста — нужный элемент управления выделяется в окне редактора и при помощи кнопок **Move Up**, **Move Down** перемещается на нужное место в списке очередности.

Для определения порядка обхода флагов, переключателей и других элементов управления, находящихся на одной панели, следует сделать ее текущей в среде GUIDE и обратиться к редактору **Tab Order Editor**. Его окно будет содержать только элементы данной панели, очередьность их обхода так же устанавливается кнопками **Move Up** и **Move Down**.

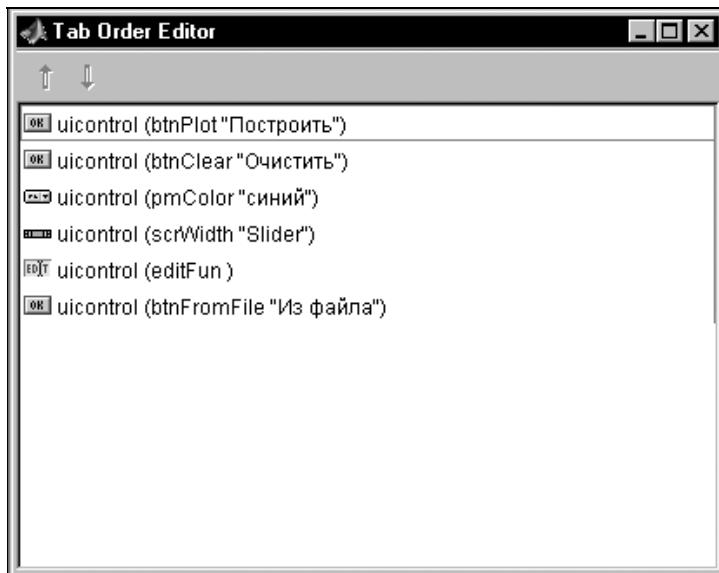
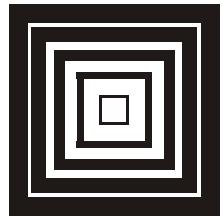


Рис. 11.13. Редактор Tab Order Editor

Следующая глава посвящена созданию диалоговых окон, конструированию меню приложения и контекстного меню.



## Глава 12

# Диалоговые окна и меню приложения

Данная глава освещает использование диалоговых окон и принципы создания собственных и контекстных меню при разработке приложения в MATLAB. Описано конструирование меню и программирование действий, выполняемых при выборе пункта меню пользователем. Изложение ведется на примере приложения `mygui`, созданию которого посвящены главы 10 и 11.

## Виды диалоговых окон

Удобный интерфейс приложения во многом определяется диалоговыми окнами, облегчающими работу с файлами, или предназначенными для предупреждения пользователя о событиях, которые могут повлечь его действия. MATLAB предоставляет разработчику приложения возможность использовать стандартные диалоговые окна.

## Окно подтверждения

Некоторые действия приложения требуют повторного подтверждения пользователя. Например, пользователь приложения `mygui` может случайно нажать кнопку **Очистить**, предназначеннную 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 — строку с текстом внутри диалогового окна (или массив строк или ячеек для многострочного текста) и строку с 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 события Callback 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 12.1).

#### Листинг 12.1. 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 **Очистить** приводит к появлению диалогового окна, изображенного на рис. 12.1. Выбор пользователя определяет дальнейшие действия приложения `mygui`.

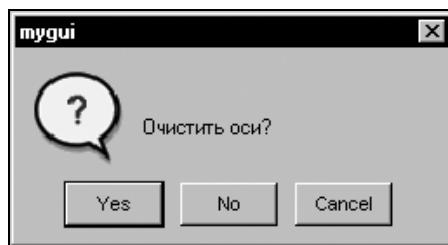


Рис. 12.1. Диалоговое окно подтверждения

По умолчанию нажатие `<Enter>` приведет к выбору **Yes**. Функция `questdlg` позволяет управлять видом диалогового окна. Стока с названием кнопки, переданная в третьем дополнительном аргументе, определяет кнопку окна, которая может быть нажата пользователем при помощи `<Enter>`. Например, вызов

```
button = questdlg('Очистить оси?', 'mygui', 'No');
```

предполагает, что в диалоговом окне нажатие `<Enter>` эквивалентно выбору **No**.

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**.

Диалоговое окно **mygui** является модальным — оно не дает пользователю перейти к окну приложения, другим графическим окнам или в командное окно MATLAB, пока оно открыто, т. е. не сделан выбор между **Yes**, **No** и **Cancel**. Любое графическое окно можно сделать модальным, для чего следует установить его свойство `WindowStyle` в значение '`'modal'`'. Значение '`'normal'` служит для перехода к обычному поведению окна.

Кроме использования функции `questdlg`, возможен второй способ создания модального диалогового окна и использования его в приложении. В окне **GUIDE Quick Start** (которое появляется при запуске среды GUIDE командой `guide` или при переходе в меню **File** среды GUIDE к пункту **New**) следует выбрать создание модального диалогового окна (см. разд. "*Среда GUIDE*" главы 10).

После формирования модального окна в среде GUIDE оно сохраняется в файле, к примеру, в `modaldlg.fig`. При этом соответствующая ему файл-функция с подфункциями запишется в файл `modaldlg.m`. Когда требуется отобразить на экране модальное окно, в основном приложении вызывается эта файл-функция. Ее выходной аргумент содержит строку с надписью на нажатой кнопке, и это значение применяется для выполнения соответствующего блока операторов. Заметьте, что надписи **Yes** и **No** на кнопках модального окна можно изменить на другие (например, **Да** и **Нет**), обращаясь к их свойству `String`.

Этот способ подробно описан в справочной системе MATLAB на примере диалогового окна с подтверждением закрытия приложения (см. разд. **Creating Graphical User Interfaces: Programming GUIs: Example: Using the Modal Dialog to Confirm an Operation**).

При наличии определенного опыта программирования событий в заготовке для модального окна можно расположить дополнительные элементы управления, например, флаг **Не показывать это окно в дальнейшем**, и получить его состояние в выходных аргументах файл-функции модального окна.

## Окна открытия и сохранения файла

Передача данных из файла в приложение сопряжена с заданием имени файла и пути к нему. Наиболее простой способ состоит в использовании диалогового окна открытия файла, которое создается функцией `uigetfile`. Дан-

ная функция приводит к появлению диалогового окна открытия файла и возвращает 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. Данныечитываются 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. Не забудьте сохранить указатель на линию в поле Line структуры handles. Операторы простейшей обработки нажатия кнопки **Из файла** приведены в листинге 12.2.

### Листинг 12.2. 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
 handles.Line = plot(Mas(:, 1), Mas(:, 2));
 % Сохранение обновленной структуры
 guidata(gcbo, handles);
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 my.dat. На осах 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 с расширением dat, проще в использовании, если в диалоговом окне открытия файла автоматически устанавливается определенный фильтр. Функция `uigetfile` предусматривает задание стандартного расширения во входном аргументе, например, вызов

```
[fname, pname] = uigetfile('.dat');
```

соответствует диалоговому окну с фильтром для файлов с расширением 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.

Команды `save` и `load` для работы с файлами, содержащими числа, описаны в разд. "Считывание и запись данных" главы 2. Более подробно считывание и запись текста и чисел разобраны в разд. "Текстовые файлы" главы 8.

Функции `uiputfile` и `uigetfile` позволяют задать заголовок диалоговых окон во втором дополнительном входном аргументе — строке.

## 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, которое выводится в отдельном окне.

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 12.2), 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 12.3).

### Листинг 12.3. 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
 handles.Line = plot(Mas(:,1), Mas(:,2));
 % Сохранение обновленной структуры
 guidata(gcbo, handles);
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, но в случае значения `figure` они добавляются к стандартным меню графического окна. Размещение и программирование меню производится при помощи редактора меню. Меню является графическим объектом `Uimenu` — потомком графического окна.

## Редактор меню

Продолжите работу над приложением `mygui`, начатую при чтении глав 10 и 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 `none`. Запустите редактор меню из панели управления (см. рис. 10.5) или выбрав в меню **Tools** пункт **Menu Editor**. Появляется окно редактора **Menu Editor**, изображенное на рис. 12.2.



Рис. 12.2. Редактор меню **Menu Editor**

Окно редактора меню содержит две вкладки: **Menu Bar**, предназначенную для создания строки меню приложения, и **Context Menus** для контекстного меню. Сейчас нам понадобится вкладка **Menu Bar**, ее область навигатора и панель **Properties** свойств элементов меню (справа от вкладки) пока пустые. При выбранной вкладке **Menu Bar** создайте новое меню, нажав кнопку **New Menu** на панели инструментов редактора меню, в навигаторе появилась строка **Untitled 1**, сделайте ее текущей щелчком мыши. Обратите внимание (рис. 12.3), что в области свойств появились элементы управления для настройки меню.

Строка **Label** служит для задания надписи меню или пункта меню, а **Tag** для определения имени (тэга) созданного объекта `Uimenu`, об использовании остальных элементов панели **Properties** сказано ниже. Введите слово "График" в строку **Label** (без кавычек) и задайте имя `mnGraph` в строке **Tag**. Запустите приложение `mygui` и убедитесь в наличии меню **График**. Выбор меню **График** в работающем приложении не приводит к раскрытию меню, поскольку необходимо еще создать пункты меню. Закройте работающее приложение. В режиме редактирования сделайте текущей строку **График** в навигаторе редактора меню и добавьте пункт, нажав кнопку **New Menu Item** (см. рис. 12.2).

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. 12.4. Меню **График** имеет первый уровень, а пункты **Построить**, **Очистить** — второй.

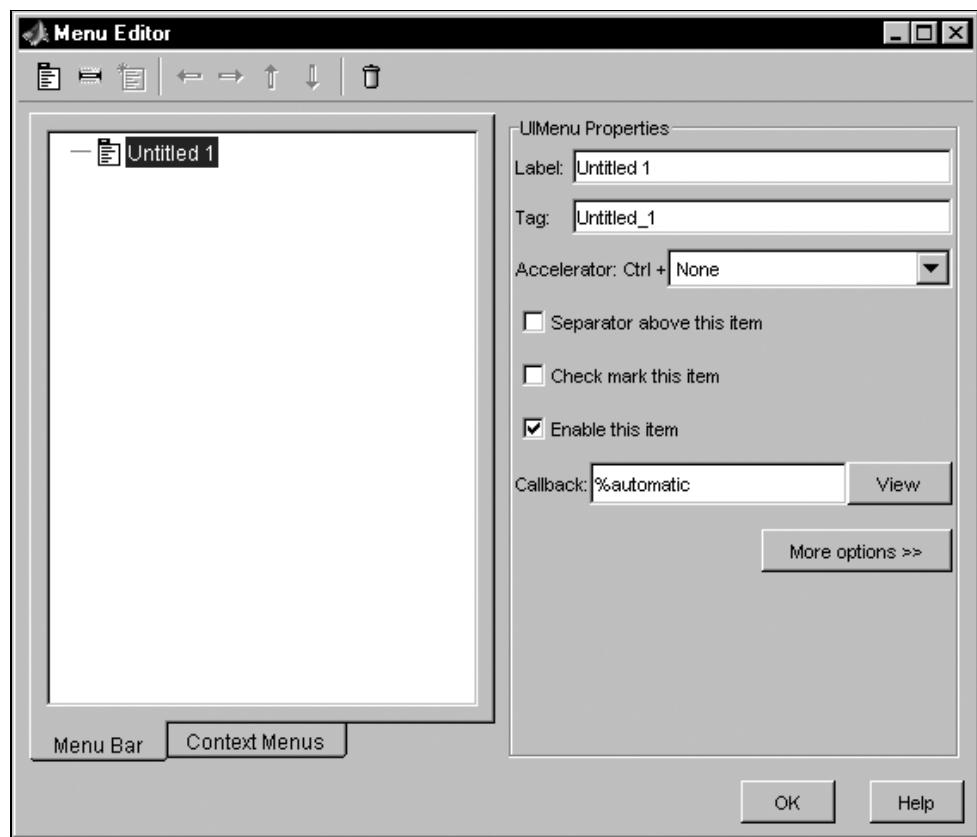


Рис. 12.3. Задание свойств меню в редакторе

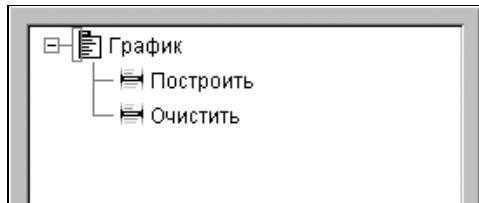


Рис. 12.4. Иерархия элементов меню

Запустите приложение mygui, выбор меню **График** приводит к раскрытию меню. Разумеется, при обращении к пунктам **Построить** и **Очистить** ничего не происходит, следует запрограммировать события **Callback** пунктов меню. Событие **Callback** самого меню **График** не требует обработки, т. к. происходит автоматическое раскрытие меню. Разумеется, если выбор меню должен повлечь некоторые действия приложения, то их необходимо запрограммировать в обработке события **Callback** самого меню.

## Программирование пунктов меню

Выбор элемента меню в навигаторе редактора меню приводит к отображению его свойств на панели **UI Menu Properties**. Стока ввода **Callback** предназначена для вызова подфункции М-файла приложения, содержащего обработку событий элементов интерфейса. Для перехода к соответствующей подфункции в редакторе М-файлов следует нажать кнопку **View**. Сделайте активной строку с пунктом **Построить** меню **График** и обратитесь к подфункции **mnGraphPlot\_Callback** обработки его события **Callback**. Заметьте, что выбор пункта **Построить** должен приводить к тому же самому результату, что и нажатие одноименной кнопки. Поэтому в подфункции **mnGraphPlot\_Callback** достаточно вызвать подфункцию **btnPlot\_Callback** обработки события **Callback** кнопки **Построить**. Аналогичным образом обстоит дело и при программировании пункта **Очистить**. При вызове функций **btnPlot\_Callback** и **btnClear\_Callback** следует учесть, что их первый входной аргумент является указателем на тот объект, событие **Callback** которого будет выполняться (листинг 12.4).

### Листинг 12.4. Программирование события **Callback** пунктов меню

```
function mnGraphPlot_Callback(hObject, eventdata, handles)
btnPlot_Callback(handles.btnPlot, eventdata, handles)

function mnGraphClear_Callback(hObject, eventdata, handles)
btnClear_Callback(handles.btnClear, eventdata, handles)
```

Запустите приложение mygui, выбор пунктов **Построить** и **Очистить** меню **График** приводит к отображению графика функции и, соответственно, очистке осей. Внесите дополнения в подфункции **btnPlot\_Callback** и **btnClear\_Callback** для обеспечения согласованной работы пунктов меню и кнопок. Именно после выбора пользователем пункта **Построить** он должен стать недоступным вместе с одноименной кнопкой, а пункт и кнопка **Очистить** — доступными, и наоборот. Используйте свойство **Enable** пунктов меню, указатели на них содержатся в структуре **handles** (листинг 12.5).

Кроме того, в начале работы приложения пункт **Очистить** должен быть недоступным. Этого можно добиться сбросом флага **Enable this item** в окне редактора меню, что эквивалентно установке свойства **Enable** пункта меню в значение **off**.

### Примечание

Три флага **Separator above this item**, **Check mark this item** и **Enable this item** позволяют установить значения соответствующим свойствам **Separator**, **Checked** и **Enable** пункта меню, т. е. объекта **Uimenu** (про назначение первых двух свойств сказано ниже). Остальные свойства объекта **Uimenu** доступны в инспекторе свойств, для вызова которого следует воспользоваться кнопкой **More Options**.

#### Листинг 12.5. Согласованное программирование пунктов меню и кнопок

```
function btnPlot_Callback(hObject, eventdata, handles)
 .
 .
 set(handles.mnGraphPlot, 'Enable', 'off')
 set(handles.mnGraphClear, 'Enable', 'on')

 function btnClear_Callback(hObject, eventdata, handles)
 button = questdlg('Очистить оси?', 'mygui');
 if strcmp(button, 'Yes')
 .
 .
 set(handles.mnGraphPlot, 'Enable', 'on')
 set(handles.mnGraphClear, 'Enable', 'off')
 end
```

Запустите приложение **mygui**. Теперь поведение кнопок **Построить** и **Очистить** согласовано с поведением одноименных пунктов меню **График**.

В качестве упражнения вы можете добавить пункт **Из файла** в меню **График** приложения **mygui**, предназначенный для графического отображения таблицы данных, и связать его с одноименной кнопкой.

## Оформление меню

Быстрый доступ к меню или его пунктам производится при помощи заданного сочетания клавиш. Для меню или пункта, выделенного в редакторе, желаемое сочетание определяется в раскрывающемся списке **Accelerator: Ctrl +**.

Как следует из названия списка, вы можете назначать комбинации <Ctrl> с любой буквенной клавишей.

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

При помощи кнопок **New menu** и **New Menu Item** редактора меню добавьте в приложение `mygui` меню **Сетка** с тэгом `mnGrid`, содержащее пункты **Сетка по x** и **Сетка по y** с тэгами `mnGridX` и `mnGridY` соответственно. Навигатор должен иметь вид, представленный на рис. 12.5.

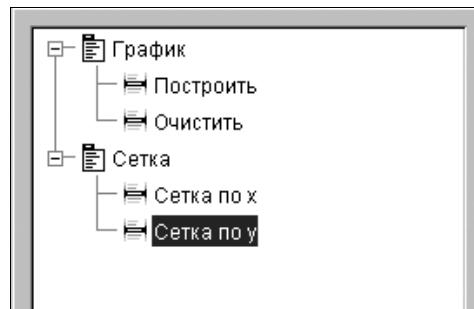


Рис. 12.5. Иерархия элементов меню **График** и **Сетка**

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. Перейдите к заготовкам подфункций, обрабатывающих события `Callback` пунктов **Сетка по x** и **Сетка по y** меню **Сетка**, и занесите в них необходимые операторы (листинг 12.6).

### Листинг 12.6. Программирование пунктов меню с флагами

```

function mnGridX_Callback(hObject, eventdata, handles)
if strcmp(get(hObject, 'Checked'), 'on')
 % Флаг пункта Сетка по у был установлен,
 % теперь следует выключить его и убрать сетку
 set(hObject, 'Checked', 'off') % выключение флага
 set(handles.axMain, 'XGrid', 'off') % скрытие сетки
else
 % Флаг пункта Сетка по у не был установлен,
 % теперь следует включить его и нанести сетку
 set(hObject, 'Checked', 'on') % включение флага
 set(handles.axMain, 'XGrid', 'on') % нанесение сетки
end
% -----
function mnGridY_Callback(hObject, eventdata, handles)
if strcmp(get(hObject, 'Checked'), 'on')
 % Флаг пункта Сетка по х был установлен,
 % теперь следует выключить его и убрать сетку
 set(hObject, 'Checked', 'off') % выключение флага
 set(handles.axMain, 'YGrid', 'off') % скрытие сетки
else
 % Флаг пункта Сетка по х не был установлен,
 % теперь следует включить его и нанести сетку
 set(hObject, 'Checked', 'on') % включение флага
 set(handles.axMain, 'YGrid', 'on') % нанесение сетки
end

```

Если некоторые пункты меню должны быть включены сразу после запуска приложения, то следует установить для них флаг **Item is checked** в редакторе меню MATLAB.

Добавьте самостоятельно операторы, которые обеспечивают согласованную работу пунктов меню **Сетка** и одноименных флагов окна приложения.

## Разделительные линии

Разделительные линии 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 между существующими пунктами меню. Редактор меню позволяет определить положение линии при помощи флага **Separator above this item**. Линия размещается над текущим пунктом меню. Разделительная линия также добавляется при установке свойству **Separator** пункта меню (т. е. объекта **Uimenu**) значения 'on' в работающем приложении.

## Упорядочение меню

Удобная организация меню очень часто выявляется только в ходе работы над приложением. Возникает необходимость в перегруппировке пунктов меню и изменении их уровня. Предположим, что требуется включить в меню **График** приложения **mygui** пункт **Сетка** вместе с подпунктами **Сетка по x** и **Сетка по y**, а пункт **Очистить** выделить в отдельное меню. Процесс переработки меню можно разбить на три этапа.

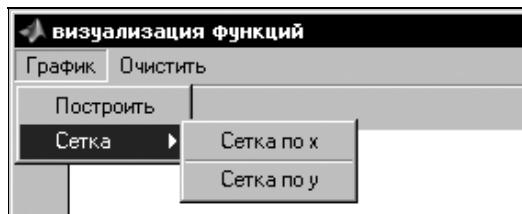
1. В навигаторе меню выделите пункт **Очистить** и при помощи кнопки **Move Selected item Down** сдвиньте его в самый низ.
2. При выделенном пункте **Очистить** нажмите кнопку **Move Selected item Backward**, это приводит к переводу пункта на один уровень вверх в иерархии и пункт становится отдельным меню.
3. Поскольку меню **Сетка** расположено в навигаторе сразу за меню **График**, то для того, чтобы **Сетка** стало пунктом меню **График**, достаточно перевести меню **Сетка** на один уровень вниз в иерархии. Для этого выделите меню **Сетка** и воспользуйтесь кнопкой **Move Selected item Forward**.

В результате меню и пункты расположены в нужной иерархии, что отражено в навигаторе меню (рис. 12.6).



Рис. 12.6. Изменение структуры меню

Вид меню в работающем приложении **mygui** приведен на рис. 12.7.



**Рис. 12.7.** Стока меню приложения  
после изменения структуры меню

Итак, назначение кнопок панели инструментов редактора меню следующее.

- **Move Selected item Backward.** — перевод пункта на один уровень вверх в иерархии. Для меню эта кнопка недоступна, поскольку они расположены на самом верхнем уровне.
- **Move Selected item Forward.** — перевод пункта на один уровень вниз в иерархии.
- **Move Selected item Down** — сдвиг меню или пункта меню вниз с сохранением уровня. Сдвиг меню вниз означает, что в строке меню работающего приложения оно переместится вправо. Сдвиг пункта меню приводит к перемещению в пределах текущего меню.
- **Move Selected item Up** — сдвиг меню или пункта меню вверх с сохранением уровня. Сдвиг меню вверх означает, что в строке меню работающего приложения оно переместится влево. Сдвиг пункта меню приводит к перемещению в пределах текущего меню.

Аналогичные операции могут быть осуществлены в работающем приложении. Поскольку меню (объект `Uimenu`) является потомком графического окна, пункты меню (также объекты `Uimenu`) есть потомки меню, а подпункты — потомки пунктов, то для реорганизации меню следует найти указатели на нужные объекты и воспользоваться их свойствами `Parent` и `Children` (поиск объектов с заданными свойствами описан в разд. "Управление объектами, копирование, поиск, скрытые указатели" главы 9).

## Контекстное меню объектов

Объекты, в том числе и созданные в ходе работы приложения, могут иметь собственное контекстное меню, которое активизируется щелчком правой кнопкой 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 контекстного меню в редакторе

Перейдите к вкладке **Context Menus** в редакторе меню и нажмите кнопку создания контекстного меню (рис. 12.2), в навигаторе меню появляется строка для меню. Задайте ему тэг `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 кнопки **New MenuItem**, той же, 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. 12.8.

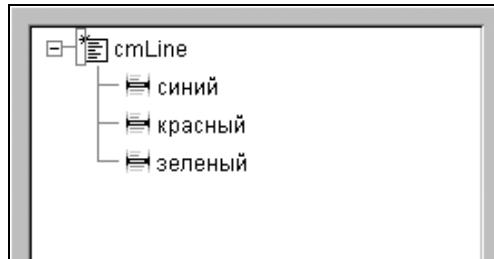


Рис. 12.8. Отображение контекстного меню в навигаторе объектов

Флаги **Separator above this item**, **Check this item** и **Enable this item** имеют тот же самый смысл, что и для элементов меню приложения, конструирование которого мы обсудили выше.

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 кнопку **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

Графические объекты MATLAB имеют 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 его свойство `UIContextMenu` в значение указателя на меню `cmline` сразу после создания линии. Этот указатель записан в поле `cmline` структуры `handles`. В нашем приложении линия графика рисуется в подфункциях обработки события `Callback` кнопок **Построить** и **Из файла**. Указатель на линию записывается в поле `Line` структуры `handles`, после чего обновленная структура сохраняется при помощи функции `guidata` (см. листинги 11.4 и 12.2). Итак, после этих строк в двух подфункциях `btnPlot_Callback` и `btnFromFile_Callback` следует обратиться к свойству `UIContextMenu` созданного объекта — линии:

```
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 обеспечивает изменение цвета линии — очевидно, следует запрограммировать событие `Callback` каждого пункта.

## Программирование контекстного меню

Обработка событий `Callback` пунктов контекстного меню производится аналогично программированию меню приложения. В редакторе меню перейдите последовательно к подфункциям обработки событий `Callback` пунктов контекстного меню, и в каждой из них установите свойство `Color` линии в соответствующее значение (листинг 12.7). Запрограммированное и связанное с линией контекстное меню разрешает быстрый доступ пользователя к цвету линии.

### Листинг 12.7. Программирование событий `Callback` пунктов контекстного меню

```
function cmlineBlue_Callback(hObject, eventdata, handles)
% Задание синего цвета линии
set(handles.Line, 'Color', 'b')
% ----

function cmlineRed_Callback(hObject, eventdata, handles)
% Задание красного цвета линии
```

```
set(handles.Line, 'Color', 'r')
%
function cmLineGreen_Callback(hObject, eventdata, handles)
% Задание зеленого цвета линии
set(handles.Line, 'Color', 'g')
```

Пункты контекстного меню являются объектами `Uimenu`, следовательно, к ним применимы подходы для пунктов меню, рассмотренные нами выше. В частности, каждый пункт контекстного меню может быть снабжен флагом, который включается при первом выборе пункта и сбрасывается при последующем. Пункты меню могут быть доступны или нет и разделены линиями.

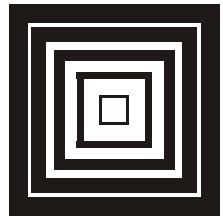
В нашем приложении `mygui` осталось обеспечить согласованную работу контекстного меню со списком **Цвет линии** с именем `pmColor`. Выбор цвета из меню должен приводить не только к изменению цвета линии, но и к появлению соответствующей строки в раскрывающемся списке. В каждую подфункцию обработки события `Callback` пункта контекстного меню следует добавить операторы, устанавливающие нужное значение (1, 2 или 3) свойства `Value` раскрывающегося списка (листинг 12.8).

#### Листинг 12.8. Согласование работы контекстного меню и списка для выбора цвета

```
function cmLineBlue_Callback(hObject, eventdata, handles)
% Задание синего цвета линии
set(handles.Line, 'Color', 'b')
% Установка строки "синий" в раскрывающемся списке
set(handles.pmColor, 'Value', 1)
%
function cmLineRed_Callback(hObject, eventdata, handles)
% Задание красного цвета линии
set(handles.Line, 'Color', 'r')
% Установка строки "красный" в раскрывающемся списке
set(handles.pmColor, 'Value', 2)
%
function cmLineGreen_Callback(hObject, eventdata, handles)
% Задание зеленого цвета линии
set(handles.Line, 'Color', 'g')
% Установка строки "зеленый" в раскрывающемся списке
set(handles.pmColor, 'Value', 3)
```

Совершенно аналогично контекстное меню создается и для других элементов приложения, например, осей или графического окна. В качестве упражнения снабдите оси контекстным меню, которое позволяет добавить или удалить линии сетки и стереть линию графика функции.

В этой и предыдущих двух главах мы сосредоточились в основном на назначении события `Callback` и его программировании для элементов управления различных типов. Исключение составляет только разд. "Изменение размеров приложения" главы 11, в котором обсуждалось согласованное изменение размеров и положения элементов управления при уменьшении или увеличении окна приложения. Для этого нам пришлось обратиться к событию `ResizeFcn` графического окна. В главе 13 мы рассмотрим еще ряд событий, которые оказываются полезными при написании собственных приложений с графическим интерфейсом.



# Глава 13

## Программирование событий

В предыдущих главах мы рассмотрели программирование события `Callback` элементов интерфейса, которое возникает, например, при нажатии на кнопку или установке флага. Кроме того, мы обсудили программирование события `ResizeFcn` графического окна, происходящего при изменении размеров приложения. В этой главе мы опишем события графических объектов и продемонстрируем способы программирования обработки некоторых из них на простых примерах, включая создание приложения с графическим интерфейсом без среды GUIDE. Мы также уделим внимание свойствам графических объектов, полезным при написании собственных приложений, например, задание формы курсора в пределах окна или получение текущих координат в графическом окне или осях.

### События графических объектов

Три события могут быть связаны со всеми графическими объектами:

- `ButtonDownFcn` — возникает при щелчке левой кнопкой мыши по объекту или не далее чем в пяти пикселях от объекта;
- `CreateFcn` — возникает при создании объекта;
- `DeleteFcn` — возникает при удалении объекта.

Событию `ButtonDownFcn` мы уделим достаточное внимание ниже при создании приложения для рисования многоугольных областей при помощи мыши.

Событие `CreateFcn` полезно для инициализации некоторых данных или предварительной настройки создаваемого объекта. Например, если при запуске приложение должно считать данные из файла или занести значения в некоторые переменные, то эти действия следует запрограммировать в файл-функции обработки события `CreateFcn` графического окна приложения.

Перед удалением объекта часто целесообразно произвести некоторые действия, например, вывести модальное диалоговое окно с подтверждением об удалении или сохранить данные (создание модальных диалоговых окон описано в *главе 12*).

Графическое окно поддерживает еще ряд событий кроме трех вышеперечисленных и `ResizeFcn`, обработка которого рассмотрена в *главе 11*:

- `CloseRequestFcn` — возникает перед закрытием окна;
- `KeyPressFcn` — возникает при нажатии пользователем клавиши;
- `WindowButtonDownFcn` — возникает при нажатии пользователем кнопки мыши;
- `WindowButtonMotionFcn` — возникает при движении мышью в пределах окна;
- `WindowButtonUpFcn` — возникает, когда пользователь отпускает кнопку мыши.

Следует иметь в виду, что событие `KeyPressFcn` возникает только в том случае, если ни один из элементов управления (например, флаг или переключатель) не выделен. Аналогично, `WindowButtonDownFcn` появляется только, если при нажатии кнопки мыши курсор находится в свободной области окна, незанятой другими элементами управления.

При обработке этих событий оказываются необходимыми следующие свойства графического окна:

- `CurrentCharacter` — содержит символ последней нажатой клавиши;
- `SelectionType` — содержит информацию о типе щелчка мышью (двойной, правой или левой кнопкой);
- `CurrentPoint` — координаты курсора мыши в момент последнего нажатия кнопки мыши в пределах окна. В качестве единиц измерения берутся установленные в свойстве `Unit` графического окна.

В качестве примера программирования ряда событий графического окна рассмотрим создание приложения, предназначенного для вывода ASCII-кода символов при нажатии на клавиши.

## Приложение для получения ASCII-кода символа

Наша цель — написать приложение, которое при нажатии на клавишу выводит в графическое окно символ и его ASCII-код, например, нажатие на `<t>` должно приводить к появлению строки "t = 116". В среде GUIDE соз-

дайте окно приложения с размещенным в ней объектом Static Text, который мы будем использовать для вывода информации. В инспекторе свойств **Property Inspector** присвойте текстовому объекту тэг `txtWin` и удалите содержимое свойства `String`.

Создайте теперь заготовку для подфункции обработки события `KeyPressFcn` графического окна, например, при помощи его всплывающего меню. Как уже было сказано, это событие возникает при нажатии пользователем одной из клавиш. Подфункция `figure1_KeyPressFcn` должна считывать значение свойства `CurrentCharacter` графического окна, преобразовывать его в ASCII-код, формировать строку с символом, знаком равно и кодом и выводить ее. Для получения ASCII-кода служит функция `double`, которая возвращает число. Поэтому придется применить еще функцию `num2str` перед заполнением строки с результатом (листинг 13.1).

### Листинг 13.1. Обработка события `KeyPressFcn` графического окна

```
function figure1_KeyPressFcn(hObject, eventdata, handles)
% Считывание текущего символа
ch = get(hObject, 'CurrentCharacter');
% Получение ASCII-кода текущего символа
code = double(ch);
% Преобразование числового значения в строковое
strcode = num2str(code);
% Считывание текущей текстовой информации объекта Static Text
str=get(handles.txtWin,'String');
% Добавление нового символа и его кода
str = [str, ' ', ch,'=', strcode];
% Вывод обновленной информации
set(handles.txtWin, 'String', str);
```

Сохраните приложение, например, с именем `asciicode`, и запустите его — при нажатии на клавишу выводится ASCII-код соответствующего символа. Заметьте, что нажатие на `<Ctrl>`, `<Alt>` или `<Shift>` приводит к выводу только знака равно, а сочетание клавиш `<Ctrl>` или `<Alt>` с символьной обрабатывается неверно. Этот недостаток приложения вы сможете устраниТЬ при чтении следующего раздела, в котором рассматриваются способы вызова подфункций обработки событий.

## Как вызываются подфункции обработки событий

При работе в среде GUIDE у нас не возникало такого вопроса. Выбор пункта меню с нужным событием приводил к автоматическому созданию заголовка подфункции, в которой требовалось реализовать желаемый алгоритм. Обсудим теперь способы обработки событий более подробно.

Вернитесь к приложению `asciicode` и в инспекторе свойств **Property Inspector** посмотрите значение свойства `KeyPressFcn` графического окна. Оно является строкой:

```
adciicode('figure1_KeyPressFcn', gcbf, [], guidata(gcbf))
```

обеспечивающей вызов основной функции `asciicode` приложения от нескольких аргументов. Основная функция имеет заголовок

```
function varargout = adciicode(varargin)
```

и может быть вызвана с произвольным числом входных и выходных аргументов (интерфейс таких функций описан в разд. "Файл-функции с переменным числом аргументов" главы 8).

Функция `asciicode` определяет, что она была вызвана с входными аргументами и при помощи `str2fun` формирует обращение к подфункции `figure1_KeyPressFcn` от трех входных аргументов:

- `hObject` принимает значение `gcbf`, т. е. указателя на графическое окно, событие которого в данный момент времени выполняется;
- `eventdata` становится равным пустому массиву;
- в `handles` передается структура данных приложения, которую возвращает функция `guidata`.

Допустим альтернативный способ вызова подфункции обработки события. Значением свойства `KeyPressFcn` графического окна может быть указатель на требуемую функцию или имя М-файла. По умолчанию предполагается, что функция обработки события вызывается с двумя обязательными входными аргументами: первый аргумент есть указатель на объект, событие которого выполняется, а второй аргумент содержит информацию о событии. В случае события `KeyPressFcn` второй аргумент является структурой с полями `Character`, `Modifier` и `Key`. Смысл этих полей следующий:

- `Character` — отображаемый на экране символ;
- `Modifier` — массив ячеек, который может содержать одну или несколько строк ('`control`', '`shift`' или '`alt`') если дополнительно была нажата одна или несколько функциональных клавиш;

- Key — символ, если была нажата символьная или цифровая клавиша, или строка с названием клавиши, например: 'subtract' для <->, 'multiply' для <\*>.

Учет содержимого этой структуры позволит расширить возможности нашего приложения `asciicode`. Однако в нем для обработки события `KeyPressFcn` предусмотрен вызов функции `figure1_KeyPressFcn` с тремя входными аргументами, причем третий (структура `handles`) используется для получения указателя на текстовый объект. Поэтому мы не будем изменять ее интерфейс, а при указании значения свойству `KeyPressFcn` графического окна применим следующий способ вызова по ссылке — сформируем массив ячеек

```
{@figure1_KeyPressFcn, handles}
```

Его первая ячейка содержит указатель на подфункцию обработки события, а вторая — значение дополнительного аргумента. Этот массив ячеек мы зададим в качестве значения свойства `KeyPressFcn` в подфункции `asciicode_OpeningFcn` (листинг 13.2), которая автоматически выполняется после запуска приложения и перед выводом его окна на экран. Заметьте, что в инспекторе свойств **Property Inspector** значение для `KeyPressFcn` уже недействительно, т. к. оно изменится после запуска приложения.

#### Листинг 13.2. Модифицированная подфункция `asciicode_OpeningFcn`

```
function asciicode_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Следующую строку мы добавили для установки свойства KeyPressFcn в
% значение массива структур с указателем на подфункцию обработки события
% KeyPressFcn и дополнительным аргументом со структурой данных приложения
set(handles.figure1, 'KeyPressFcn', {@figure1_KeyPressFcn, handles})
guidata(hObject, handles);
```

Теперь входной аргумент `eventdata` подфункции `figure1_KeyPressFcn` является структурой с полями `Character`, `Modifier` и `Key`, смысл которых мы только что разобрали. Пока можно вывести их значения в командное окно для проверки работы приложения, а потом учесть их в алгоритме обработки нажатия клавиши. Для этого просто добавьте в подфункцию `figure1_KeyPressFcn` присваивание значений полей некоторым переменным, например, `character`, `modifier` и `key` (листинг 13.3).

**Листинг 13.3. Вывод данных структуры eventdata в подфункции figure1\_KeyPressFcn**

```
ch=eventdata.Character
mdf=eventdata.Modifier
key=eventdata.Key
```

Приложение asciicode работает так же, как и со стандартным способом вызова подфункции обработки события KeyPressFcn, однако теперь входной аргумент eventdata подфункции figure1\_KeyPressFcn содержит подробную информацию о событии. Используйте ее для устранения следующего недостатка приложения — при нажатии на клавиши <Ctrl>, <Shift> и <Alt> без символьных клавиш выводиться ничего не должно, а при сочетании одной или нескольких из них с символьной клавишей об этом выдается сообщение, например: "shift+ctrl+h=8".

Очевидно, что если символьная клавиша не была нажата, то требуется выйти из подфункции. Иначе следует проверить, была ли нажата одна или несколько функциональных клавиш <Ctrl>, <Shift> и <Alt>, т. е. выяснить длину массива ячеек mdf, извлечь из него строки с названиями нажатых клавиш и сформировать текстовый результат. Один из возможных вариантов реализации этого алгоритма приведен в листинге 13.4.

**Листинг 13.4. Обработка нажатия на клавишу с учетом структуры event**

```
function figure1_KeyPressFcn(hObject, eventdata, handles)
% Считываем отображаемый на экране символ
ch = eventdata.Character;
% Считываем массив ячеек, содержащий строки с нажатыми
% функциональными клавишами
mdf = eventdata.Modifier;
% Считываем символ клавиши
key = eventdata.Key;
% Проверка, была ли нажата символьная клавиша
if length(ch) == 0
 % Символьная клавиша нажата не была, выходим из подфункции
 return
end
% Получение ASCII-кода символа
code = double(ch);
```

```
% Преобразование числового значения в строку
strcode = num2str(code);

% Считывание содержимого текстового объекта Static Text
str = get(handles.txtWin, 'String');

% Проверка, была ли нажата функциональная клавиша
if isempty(mdf)

 % Была нажата только символьная клавиша
 % Формируем строку для вывода
 str = [str, ' ', ch, '=', strcode];
 % Выводим результат
 set(handles.txtWin, 'String', str);
 % Завершаем работу подфункции
 return
end

% Была нажата одна или несколько функциональных клавиш
switch length(mdf)

 case 1
 % Была нажата одна функциональная клавиша, считываем ее название
 fkey = mdf{1};
 % Формируем строку для вывода
 str = [str, ' ', fkey, '+', key, '=', strcode];
 case 2
 % Были нажаты две функциональные клавиши, считываем их названия
 fkey1 = mdf{1};
 fkey2 = mdf{2};
 % Формируем строку для вывода
 str = [str, ' ', fkey1, '+', fkey2, '+', key, '=', strcode];
 case 3
 % Были нажаты три функциональные клавиши, считываем их названия

 fkey1 = mdf{1};
 fkey2 = mdf{2};
 fkey3 = mdf{3};
 % Формируем строку для вывода
 str = [str, ' ', fkey1, '+', fkey2, '+', fkey3, '+', ...
```

```
key, '=' , strcode];
end
% Выводим результат
set(handles.txtWin, 'String', str);
```

Следующий раздел посвящен примеру приложения, использующему событие `ButtonDownFcn`, которое происходит при щелчке мышью по объекту.

## Событие `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. В качестве примера напишем приложение для рисования многоугольных областей при помощи мыши, в котором щелчок мыши по осям приводит к появлению новой вершины (маркера) и соединению ее с предыдущей отрезком.

## Событие `ButtonDownFcn` осей

Для того чтобы понять способ обработки события `ButtonDownFcn`, начните с создания более простого приложения, в котором щелчок мышью по осям приводит к появлению маркера.

В среде GUIDE на заготовке окна приложения разместите оси с именем `axes`. Заданное по умолчанию автоматическое масштабирование осей в нашем примере нежелательно, т. к. при появлении первого маркера пределы осей изменятся. Следовательно, имеет смысл установить свойства осей `XLimMode` и `YLimMode` в значение `manual`. Каждый щелчок мышью должен добавлять маркер на оси, поэтому свойство осей `NextPlot` должно иметь значение `'add'`. Проделайте эти действия в инспекторе свойств **Property Inspector**.

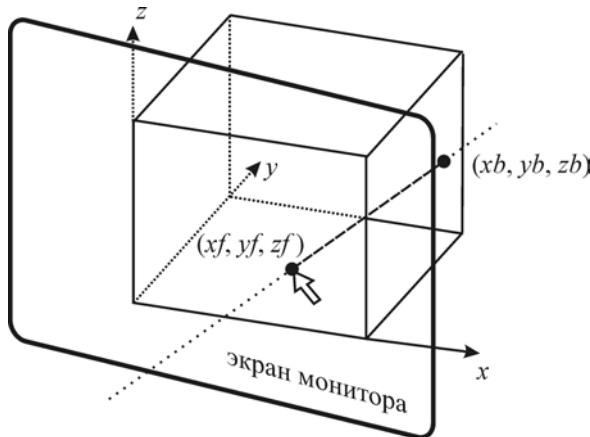
Перейдите к подфункции `axes_ButtonDownFcn` обработки события `ButtonDownFcn` осей. В ней требуется определить координаты точки, в которой был произведен щелчок мышью, и вывести в нее маркер.

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. 13.1), координаты которых хранятся в матрице

$$\begin{bmatrix} xb & yb & zb \\ xf & yf & zf \end{bmatrix}.$$

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.



**Рис. 13.1.** Координаты точек, хранящиеся в CurrentPoint

Итак, алгоритм подфункции `axes_ButtonDownFcn` простой — считаются координаты точки, например, из второй строки матрицы, и в нее выводится маркер (листинг 13.5).

#### Листинг 13.5. Обработка события ButtonDownFcn осей

```
function axes_ButtonDownFcn(hObject, eventdata, handles)
% Получение матрицы с координатами точки
Coord = get(hObject, 'CurrentPoint')
% Определение абсциссы и ординаты
x = Coord(2, 1);
y = Coord(2, 2);
% Вывод маркера
plot(x, y, 'o')
```

Сохраните приложение с именем `myplot` и запустите его — щелчок мышью по осям приводит к появлению маркера.

Вернемся теперь к исходной задаче рисования многоугольной области. При первом возникновении события `ButtonDownFcn` осей мы должны вывести маркер, а при каждом последующем необходимо соединять текущую вершину с предыдущей отрезком, образуя границу многоугольника. По всей видимости, представление всей границы одним графическим объектом даст определенные преимущества по сравнению с набором объектов для каждой стороны многоугольника (так и окажется впоследствии). При запуске приложения мы инициализируем линию (объект `Line`) при помощи функции `line` и установим ее свойства `XData` и `YData` в значения пустых массивов (см. разд. "Объекты `Rectangle` и `Line`, блок-схемы и диаграммы" главы 9).

При возникновении каждого события `ButtonDownFcn` осей будем добавлять к этим массивам абсциссу и ординату новой точки, увеличивая число отрезков ломаной линии.

В предыдущем разделе мы обсуждали способ инициализации в подфункции обработки события `OpeningFcn`, которое возникает после запуска приложения и перед появлением его на экране. Создайте в подфункции `myplot_OpeningFcn` линию, сохранив указатель на нее в поле `Line` структуры `handles` (не забудьте сохранить измененную структуру `handles`) и измените обработку события `ButtonDownFcn` осей (листинг 13.6).

#### Листинг 13.6. Инициализация объекта `Line` и обработка события `ButtonDownFcn` осей

```
function varargout = myplot_OpeningFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% Создание линии с пустыми массивами данных
% и круглыми маркерами в вершинах, запись указателя на нее в поле Line
% структуры handles
handles.Line = line('XData', [], 'YData', [], 'Marker', 'o');
% Сохранение модифицированной структуры handles
guidata(hObject, handles);

function axes_ButtonDownFcn(hObject, eventdata, handles)
% Получение матрицы с координатами точки
Coord = get(hObject, 'CurrentPoint')
% Определение абсциссы и ординаты
x = Coord(2, 1);
y=Coord(2,2);
```

```
% Считывание абсцисс предыдущих точек
X = get(handles.Line, 'XData');
% Считывание ординат предыдущих точек
Y = get(handles.Line, 'YData');
Добавление координат новой точки
X = [X x];
Y = [Y y];
% Модификация значений свойств XData и YData линии
set(handles.Line, 'Xdata', X, 'Ydata', Y)
```

Теперь приложение myedit позволяет рисовать ломаную линию, но имеет один существенный недостаток — при попытке замкнуть границу многоугольника ничего не происходит. Точно так же не удается разместить новую вершину на уже нарисованной границе. Очевидно, что при щелчке мышью по линии возникает событие `ButtonDownFcn`, которое пока не запрограммировано.

## Событие `ButtonDownFcn` линии

Основное отличие в программировании события `ButtonDownFcn` для линии по сравнению с тем же событием осей, состоит в том, что в среде GUIDE заготовка приложения не содержит объект линии, она создается только после запуска приложения. Поэтому необходимо, во-первых, в редакторе М-файлов самостоятельно создать заголовок соответствующей подфункции и запрограммировать ее и, во-вторых, связать эту подфункцию с линией. Назовем подфункцию обработки события `ButtonDownFcn` для линии `Line_ButtonDownFcn` и будем считать, что она имеет три входных аргумента (так же как и стандартные подфункции): `hObject`,  `eventdata` и `handles`. Алгоритм функции `Line_ButtonDownFcn` не отличается от обработки события `ButtonDownFcn` осей (листинг 13.7).

### Листинг 13.7. Подфункция обработки события `ButtonDownFcn` линии

```
function Line_ButtonDownFcn(hObject, eventdata, handles)
Coord = get(handles.axes, 'CurrentPoint')
x = Coord(2, 1); y = Coord(1, 2);
X = get(handles.Line, 'XData')
Y = get(handles.Line, 'YData')
X = [X x]
Y = [Y y]
set(handles.Line, 'Xdata', X, 'Ydata', Y)
```

Теперь надо связать подфункцию `Line_ButtonDownFcn` с событием `ButtonDownFcn` линии, для чего свойство линии `ButtonDownFcn` установим в значение массива ячеек `{@Line_ButtonDownFcn, handles}`.

Мы обсуждали вызов подфункций обработки события при помощи указателя в разд. "Как вызываются подфункции обработки событий" этой главы.

Операцию связывания следует выполнить сразу после создания объекта линии в подфункции `myplot_OpeningFcn`, которая может реализоваться в соответствии с листингом 13.8.

#### Листинг 13.8. Связь подфункции `Line_ButtonDownFcn` с событием `ButtonDownFcn` линии

```
function myplot_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Создание линии с пустыми массивами данных
% и круглыми маркерами в вершинах, запись указателя на нее в поле Line
% структуры handles
handles.Line = line('XData', [], 'YData', [], 'Marker', 'o')
% Связь события ButtonDownFcn линии с подфункцией Line_ButtonDownFcn
set(handles.Line, 'ButtonDownFcn', {@Line_ButtonDownFcn, handles})
% Сохранение модифицированной структуры handles
guidata(hObject, handles);
```

После проделанных изменений приложение `myplot` позволяет рисовать произвольные замкнутые многоугольные области.

Рассмотренные выше способы программирования событий наводят на мысль, что приложения с графическим интерфейсом могут быть созданы без обращения к среде визуального программирования GUIDE.

## Создание приложений с GUI без среды GUIDE

В качестве примера создадим файл-функцию `simplegui`, вызов которой приводит к появлению окна приложения с двумяарами осей. Под каждой парой осей расположена кнопка **Построить** для вывода на нее столбцовой диаграммы некоторых данных (рис. 13.2).

В основной функции `simplegui` следует разместить операторы, создающие графическое окно, оси и кнопки с заданными свойствами. Среди этих свойств следует задать те, которые содержат указатели на подфункции обработки событий. Сами подфункции размещаются в одном М-файле после

основной функции. Мы приводим листинг 13.9 файл-функции simplegui вместе с подфункциями, снабженный подробными комментариями.

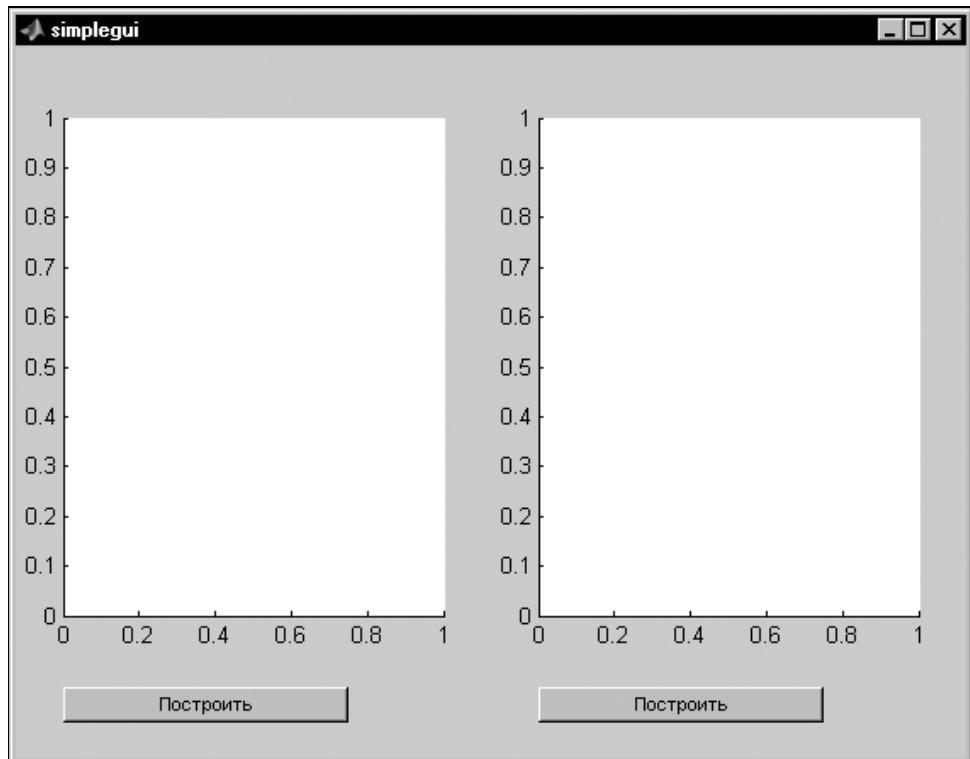


Рис. 13.2. Окно приложения simplegui

#### Листинг 13.9. Программирование приложения с GUI в одном М-файле

```
function simplegui
% Основная функция, создающая окно приложения и его компоненты
% Указатели на все графические объекты заносятся в структуру handles

% Создание графического окна без меню с заголовком "simplegui"
handles.F = figure('MenuBar', 'none', 'Name', 'simplegui', ...
 'NumberTitle', 'off');

% Связь события CloseRequestFcn с подфункцией Close
set(handles.F, 'CloseRequestFcn', {@Close,handles})
```

```
% Создание левой пары осей
handles.A1 = axes('Position', [0.05 0.2 0.4 0.7]);
% Создание правой пары осей
handles.A2 = axes('Position', [0.55 0.2 0.4 0.7]);
% Создание левой кнопки "Построить" и связь ее события Callback
% с подфункцией PlotBtn1
handles.PlotBtn1 = uicontrol('Style', 'pushbutton', 'Units', ...
 'Normalized', 'Position', [0.05 0.05 0.3 0.05], ...
 'String', 'Построить', 'Callback', {@PlotBtn1,handles});
% Создание правой кнопки "Построить" и связь ее события Callback
% с подфункцией PlotBtn2
handles.PlotBtn2 = uicontrol('Style', 'pushbutton', 'Units', ...
 'Normalized', 'Position', [0.55 0.05 0.3 0.05], ...
 'String', 'Построить', 'Callback', {@PlotBtn2,handles});

function PlotBtn1(hObject, eventdata, handles)
% Подфункция обработки события Callback левой кнопки

 % Делаем текущими левые оси
 axes(handles.A1)
 % Выводим на них вертикальную столбцовую диаграмму вектора
 bar(rand(1, 4))

function PlotBtn2(hObject, eventdata, handles)
% Подфункция обработки события Callback правой кнопки

 % Делаем текущими правые оси
 axes(handles.A2)
 % Выводим на них горизонтальную столбцовую диаграмму вектора
 barh(rand(1, 4))

function Close(hObject, eventdata, handles)
% Подфункция обработки события CloseRequestFcn окна приложения

 % Вывод модального диалогового окна с подтверждением закрытия
 q = questdlg('Закрыть приложение', 'simplegui', 'Да', 'Нет', 'Нет');
 % Проверка выбора пользователя
 if strcmp(q, 'Да')
```

```
% Пользователь подтвердил закрытие приложения,
% удаляем окно приложения
delete(handles.F)
end
```

## Свойства объектов, полезные при программировании событий

Программирование событий различных объектов может потребовать привлечения ряда их свойств. Например, выше мы рассмотрели использование свойства `CurrentPoint` осей для получения координат точки, в которой был сделан щелчок мыши. Свойство `CurrentCharacter` графического окна позволило нам определить символ нажатой клавиши. Перечислим еще ряд свойств графических объектов, к которым приходится прибегать при программировании приложений с графическим интерфейсом.

### Прерывание обработки событий

Если обработка некоторого события занимает достаточно много времени, то имеет смысл позаботиться о том, как на его выполнение повлияет возникновение нового события. Каждый графический объект имеет два свойства `BusyAction` и `Interruptible`, установка значений которых позволяет выбрать различные способы согласования событий. Для запрета прерывания исполняемого события объекта следует установить его свойство `Interruptible` в значение '`'off'`'. Тогда выполнение обработки следующего события определяется значением свойства `BusyAction`. Если оно принимает значение '`'cancel'`', то обработка следующего события отменяется, а если '`'queue'`' (по умолчанию), то следующее событие ставится в очередь.

По умолчанию свойство `Interruptible` принимает значение '`'on'`', при этом обработка события может прерываться при наличии в выполняемой функции одной из команд `drawnow`, `figure`, `getframe`, или `pause`. Для различных объектов имеются свои особенности при использовании свойств `Interruptible` и `BusyAction`, которые приведены в справочной системе (см. описание свойств графических объектов в разд. **MATLAB: Handle Graphics Property Browser** и **MATLAB: Creating Graphical User Interface: Interrupting Executing Callbacks**).

## Изменение формы курсора

Тип указателя мыши задается свойством `Pointer` графического окна, которое может принимать значения:

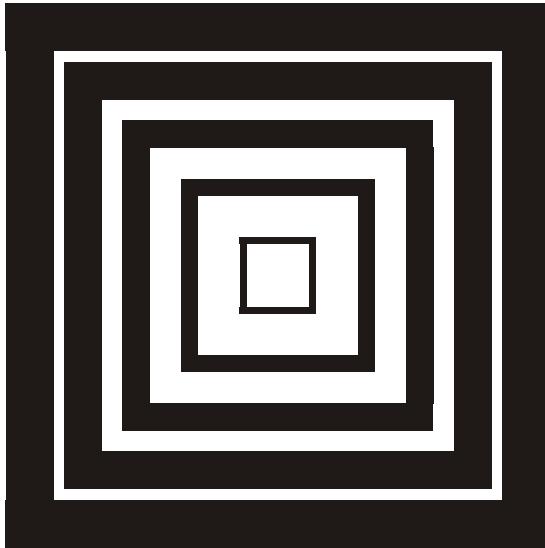
- `crosshair` — перекрестие;
- `arrow` — обычная стрелка (по умолчанию);
- `watch` — песочные часы;
- `ibeam` — вертикальный курсор.

Перечень всех возможных значений приведен в справочной системе MATLAB. Значение `custom` позволяет определить собственную квадратную пиктограмму размером 16 пикселов для курсора мыши в матрице и задать ее в качестве значения свойства `PointerShapeCData`. Единица означает черный цвет, двойка — белый, а прозрачность пикселя достигается установкой `Nan` в соответствующие элементы матрицы. Последовательность команд, приведенная в листинге 13.10, приводит к курсору мыши в виде квадратной прозрачной рамки.

### Листинг 13.10. Генерация указателя мыши в виде квадратной прозрачной рамки

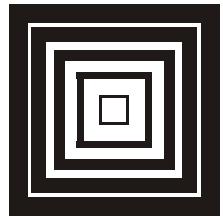
```
P = ones(16);
MP(2:15,2:15) = NaN;
set(gcf, 'Pointer', 'custom')
set(gcf, 'PointerShapeCData', MP)
```

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. 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 И РЕШЕНИЕ  
ПРИКЛАДНЫХ ЗАДАЧ



## Глава 14

# Решение задач математической физики

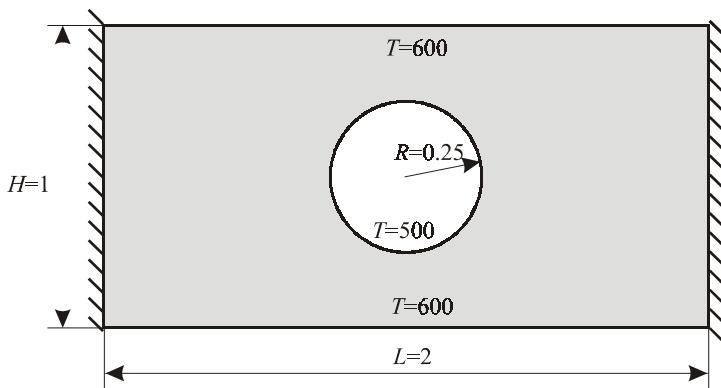
Большинство задач механики, теплопроводности, течения жидкости, электростатики и электродинамики сводятся к дифференциальным уравнениям в частных производных в областях сложной формы. Данная глава посвящена описанию возможностей Partial Differential Equations Toolbox (PDE Toolbox), предназначенного для 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, PDE Toolbox 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. 14.1.



**Рис. 14.1.** Область и граничные условия

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)

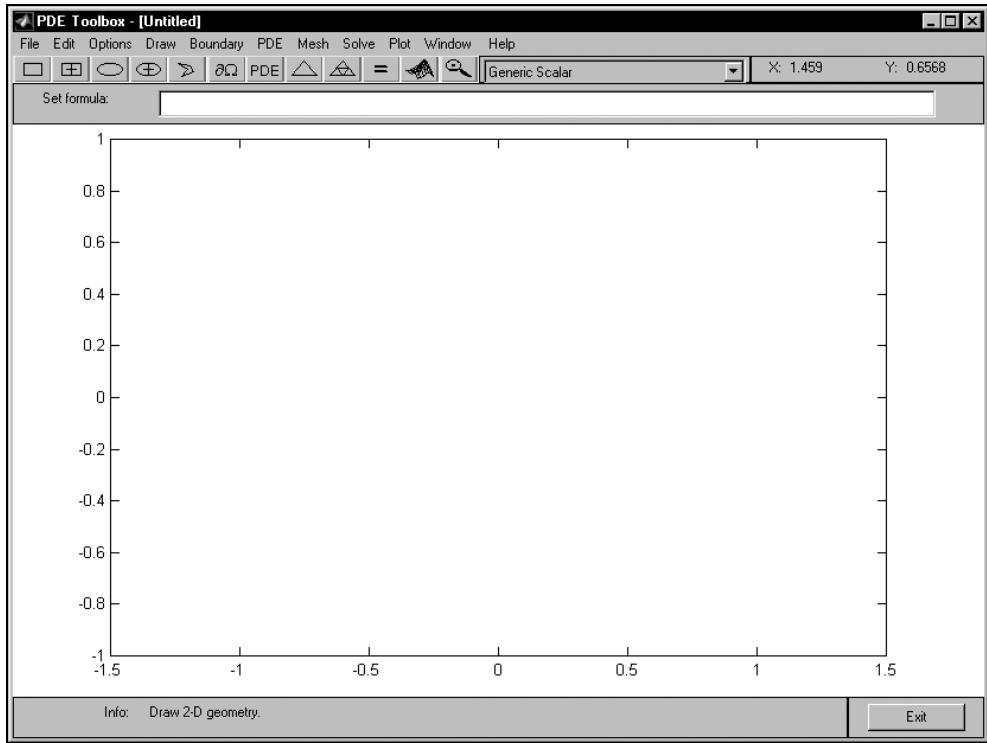
$$n \cdot k \nabla T = 0;$$

на верхней и нижней границе  $T = 600$ , а на окружности  $T = 500$ . Размерности единиц не указываются.

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. 14.2.



**Рис. 14.2.** Среда pdetool

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.

Конструирование области, приведенной на рис. 14.1, включает в себя создание прямоугольника и круга заданных размеров и вычитание круга из прямоугольника. Удобно расположить область так, чтобы центр прямоугольника совпал с началом координат, тогда координаты нижнего левого угла

прямоугольника будут  $(-1, -0.5)$ , а его высота и ширина 1 и 2 соответственно. Центр круга также совпадает с началом координат.

Начните с создания прямоугольника. Выберите в меню **Draw** пункт **Rectangle/square** или воспользуйтесь соответствующим инструментом (его пиктограмма содержит прямой угол). 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. 14.3), предназначенного для установки заданных размеров и положения объекта.

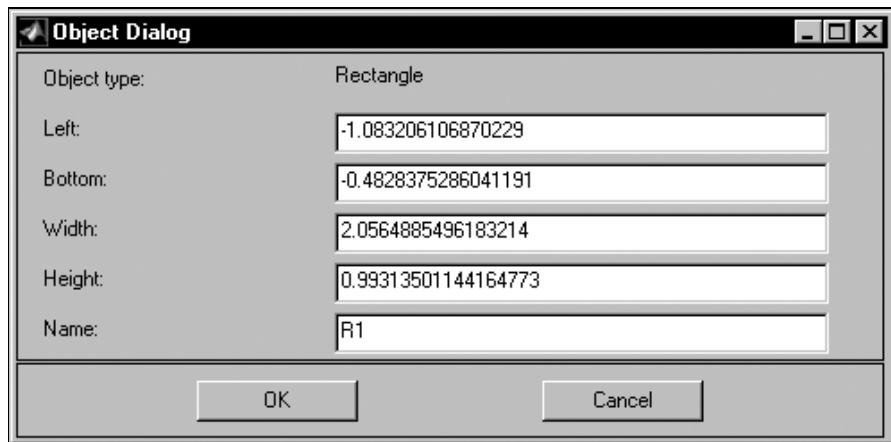


Рис. 14.3. Диалоговое окно **Object Dialog**

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 прямоугольников 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.

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.

### Примечание

Перекрестие на пиктограмме инструмента, или слово "centered" в названии пункта меню **Draw** означают, что соответствующий объект рисуется на осях из центра при нажатой левой кнопке мыши.

Поместите курсор мыши в начало координат осей и нарисуйте окружность при помощи мыши с одновременным удержанием **<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. 14.1, 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 — сре-да **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**. На экране теперь отобразится 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. 14.4. 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.

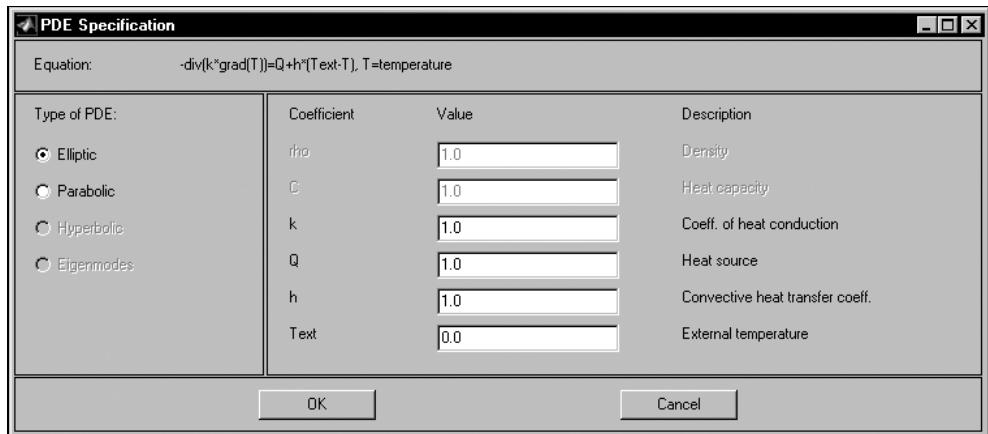


Рис. 14.4. Диалоговое окно **PDE Specification**

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  $Q$  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.

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. 14.5).

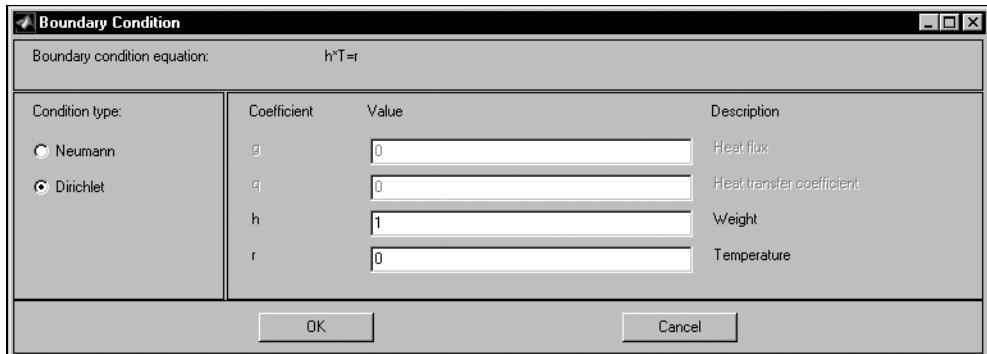


Рис. 14.5. Диалоговое окно **Boundary Condition**

Граничное условие предлагается в общем виде, выбор коэффициентов позволяет получить требуемый частный случай. На верхней границе прямоугольника задано значение температуры, равное 600 (см. рис. 14.1). Это граничное условие Дирихле. Убедитесь, что на панели **Condition type** окна **Boundary Condition** включен переключатель **Dirichlet**, соответствующий 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. 14.1). Равенство нулю потока есть частный случай условий Неймана. Объедините две грани-

цы 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  $n \cdot k \cdot \text{grad}(T) + q \cdot T = g$ , отвечающего конвективному теплообмену через границу с окружающей средой. Очевидно, что для получения теплоизолированных границ требуется задать коэффициенты  $q$  и  $g$  равными нулю.

### Примечание

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 производятся 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. 14.6).

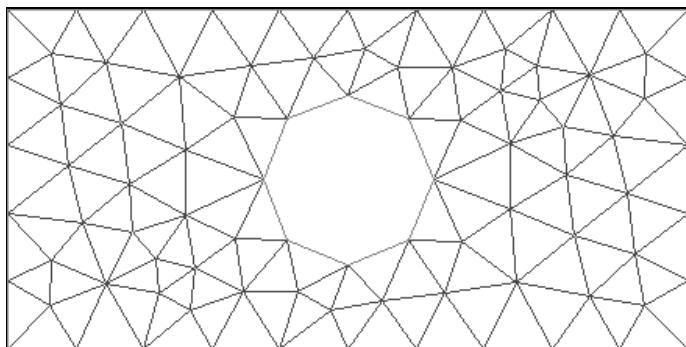


Рис. 14.6. Начальная триангуляция

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.

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. 14.7).

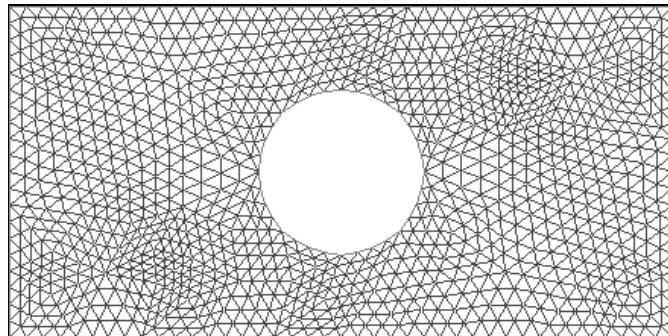


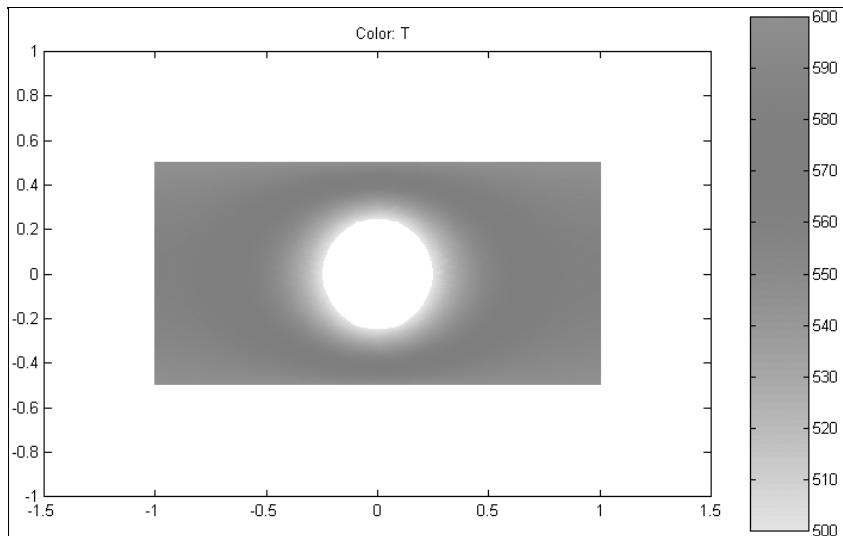
Рис. 14.7. Расчетная триангуляция

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. 14.8).

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 М-файле из пункта **Save as...** (или **Save** для сохранения в существующем М-файле с тем же именем) меню **File**. М-файл содержит функции PDE Toolbox, которые вызыва-

ются 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.



**Рис. 14.8.** Графическое представление результата

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" 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 PDE Toolbox

Интерфейс среды pdetool облегчает доступ пользователя к функциям, входящим в состав PDE Toolbox. 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 PDE Toolbox 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 PDE Toolbox. В следующих разделах приведены типы граничных задач для уравнений в частных производных, решение которых может быть найдено с использованием PDE Toolbox. Область обозначена через  $\Omega$ , 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:

$$-\nabla \cdot (c\nabla u) + au = f \text{ в } \Omega.$$

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. 14.1), коэффициенты уравнения и граничных условий были постоянны, однако PDE Toolbox 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 PDE Toolbox. 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 **Elliptic** на панели **Type of PDE**. 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, приведенному 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. 14.9), которое позволяет выбрать тип и вид визуализируемого результата.

Верхняя панель окна **Plot Selection** организована в виде таблицы, левая колонка которой содержит флаги, соответствующие способу визуализации результатов. Поле **Property** состоит из раскрывающихся списков, предназначенных для выбора отображаемой функции от приближенного решения. Опция **user entry** делает доступной строку ввода в колонке **User entry**. Введи-

те в ней выражение для вычисления погрешности  $u - (x.^2 + y.^2).^2$ . Нажатие кнопки **Plot** приводит к отображению заливого 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.

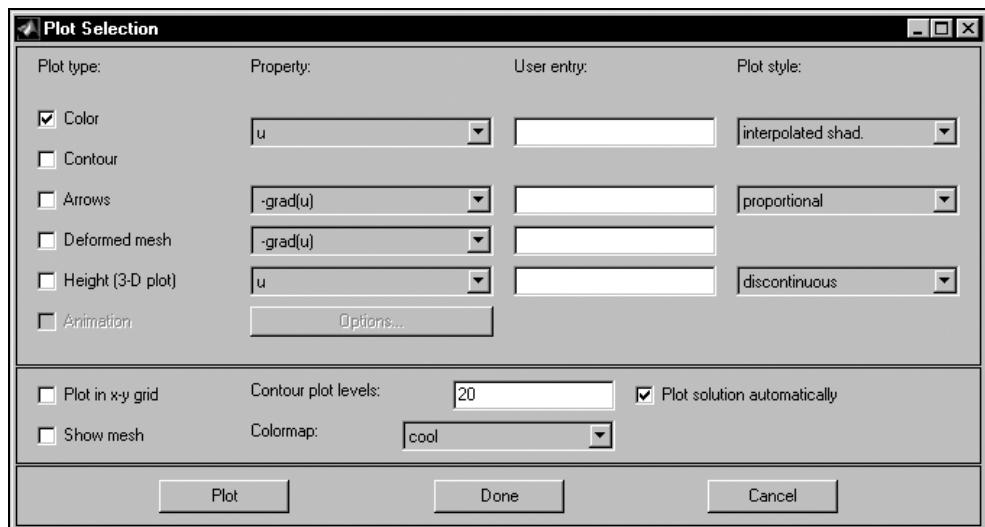


Рис. 14.9. Диалоговое окно **Plot Selection**

Переменные коэффициенты уравнения задаются 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 PDE Toolbox, выглядят следующим образом:

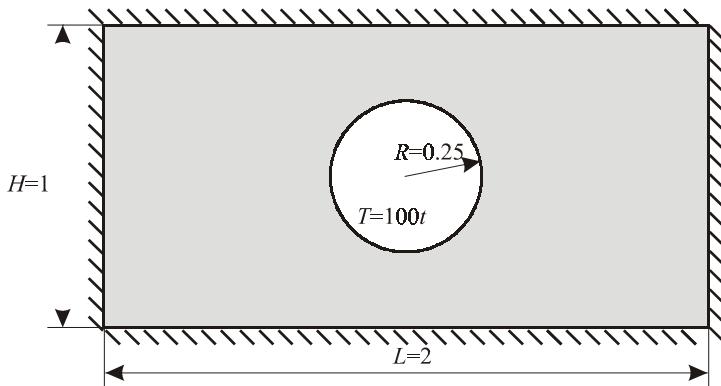
$$d \frac{\partial u}{\partial t} - \nabla \cdot (c \nabla u) + au = f; \quad d \frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c \nabla u) + au = f.$$

Коэффициенты  $d$ ,  $c$ ,  $a$  и правая часть  $f$  могут зависеть как от  $x$  и  $y$ , так и от времени  $t$ . Границные 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  $y$ . PDE Toolbox позволяет найти решение только в ограниченной области  $\Omega$ .

Среда pdetool предоставляет пользователю возможность получения 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

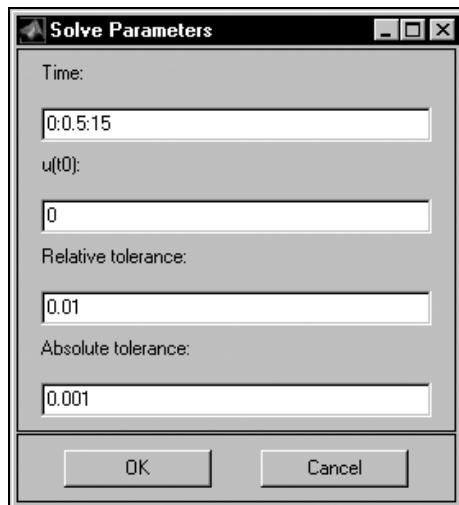
В качестве примера рассмотрим нестационарную задачу о распределении тепла в области, изображенной на рис. 14.10. Границы прямоугольника теплоизолированы, а края отверстия подвергаются нагреву, температура изменяется линейно со временем. Внутри области нет распределенных источников тепла, в начальный момент времени температура равна нулю во всей области. Нас будет интересовать изменение температуры на протяжении пятнадцати секунд.



**Рис. 14.10.** Задача  
о нестационарном распределении температуры

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  $T_{ext}$  обнулите. На границах прямоугольника поставьте условия равенства нулю потока тепла, а для окружности введите формулу  $100*t$ , переменная  $t$  используется для обозначения времени.

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. 14.11) 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. 14.11.** 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.

## Примечание

Часто требуется найти решение не в равноотстоящие моменты времени. Большой 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. 14.12.



Рис. 14.12. 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

PDE Toolbox 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  $\Omega$ . 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 PDE Toolbox, позволяют решить систему дифференциальных уравнений произвольной размерности. Среда `pdetool` оперирует только с системой второго порядка:

$$-\nabla \cdot (c_{11} \nabla u_1) - \nabla \cdot (c_{12} \nabla u_2) + a_{11}u_1 + a_{12}u_2 = f_1;$$

$$-\nabla \cdot (c_{21} \nabla u_1) - \nabla \cdot (c_{22} \nabla u_2) + a_{21}u_1 + a_{22}u_2 = f_2;$$

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

$$n \cdot (c_{11} \nabla u_1) + n \cdot (c_{12} \nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1;$$

$$n \cdot (c_{21} \nabla u_1) + n \cdot (c_{22} \nabla u_2) + q_{21}u_1 + q_{22}u_2 = g_2$$

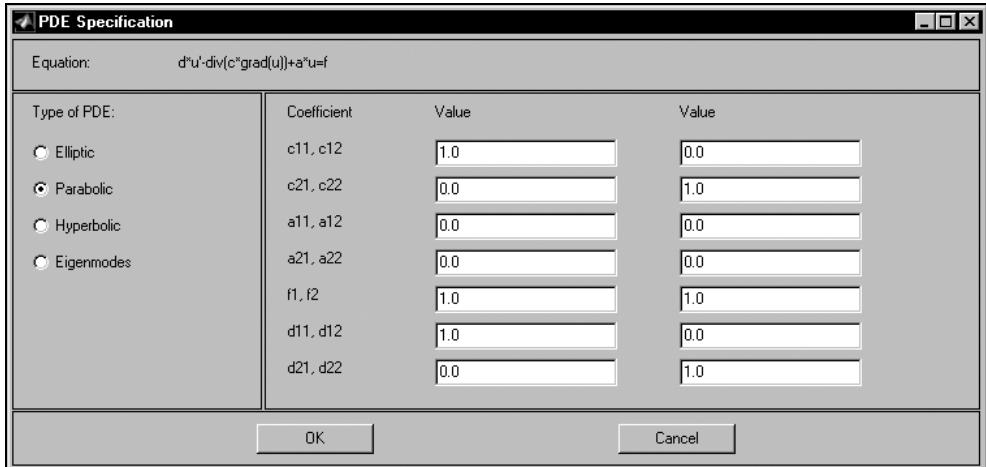
или смешанные граничные условия

$$h_{11}u_1 + h_{12}u_2 = r_1;$$

$$n \cdot (c_{11}\nabla u_1) + n \cdot (c_{12}\nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1 + h_{11}\mu;$$

$$n \cdot (c_{21}\nabla u_1) + n \cdot (c_{22}\nabla u_2) + q_{21}u_1 + q_{22}u_2 = g_2 + h_{12}\mu.$$

Нестационарные задачи, описываемые системами дифференциальных уравнений, также могут быть решены в PDE Toolbox. 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. 14.13.** Диалоговое окно **PDE Specification**, настроенное на решение системы параболических уравнений

Диалоговое окно **PDE Specification** позволяет задать коэффициенты стационарных и нестационарных систем (рис. 14.13) 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 компонент  $[u_1(x, y), u_2(x, y)]$ . Раскрывающиеся списки диалогового окна **Plot Selection** предоставляют возможность визуализировать первую или вторую компо-

ненты решения (они обозначены через  $u$  и  $v$ ), вывести график заданной пользователем функции от компонент решения (опция **user entry**) или отобразить решение векторным полем (флаг **Arrows**).

## Параметры триангуляции и управление процессом решения

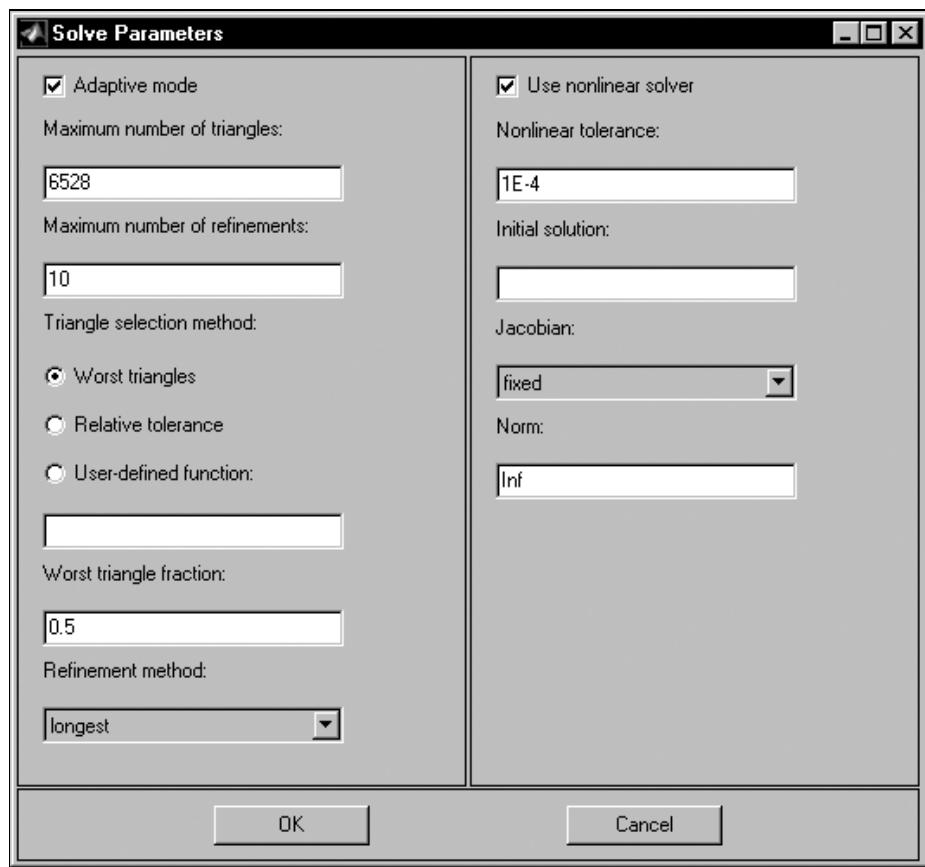
Начальная триангуляция области достаточно грубая, дальнейшее уменьшение шага сетки производится выбором пункта **Refine Mesh** меню **Mesh** или нажатием соответствующей кнопки на панели инструментов среды pdetool. Диалоговое окно **Mesh Parameters**, открывающееся при переходе к пункту **Parameters...** меню **Mesh**, предназначено для задания параметров триангуляции, в частности, максимальная длина стороны элементов вводится в строке **Maximum edge 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. 14.14 (для наглядности включены флаги вверху каждой панели).

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 triangles** 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 однотипной строке следует ввести погрешность;
- по критерию пользователя (переключатель **User-defined function**), дополнительная информация содержится в справочной системе MATLAB по PDE Toolbox.



**Рис. 14.14.** Диалоговое окно **Solve Parameters**  
для эллиптического уравнения

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 MATLAB по PDE Toolbox.

# 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 <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 мелкого объекта. Вид окна **Object Dialog** зависит от типа объекта, например, прямоугольник и квадрат задаются координатами левого нижнего угла, шириной и высотой (см. рис. 14.3). 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 **Object Dialog**, соответствующее многоугольнику, приведено на рис. 14.15. Раскрывающийся спи-

сок **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.

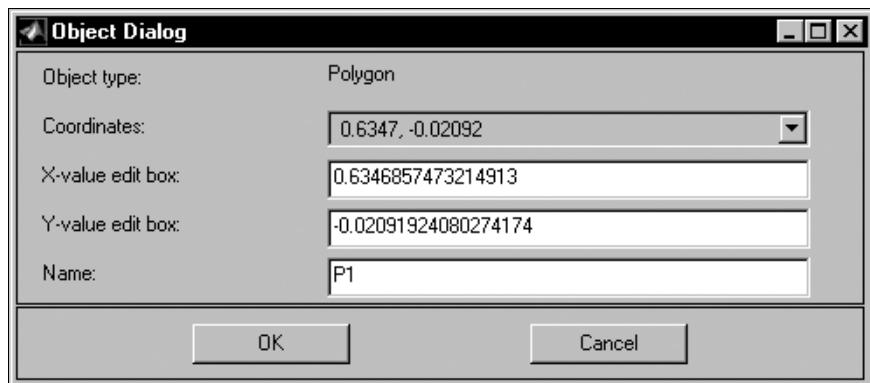


Рис. 14.15. Диалоговое окно **Object Dialog** со свойствами многоугольника

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. 14.16, нарисуйте прямоугольник R1 и три круга C1, C2, C3 и введите в строку **Set formula** выражение

R1 = (C1 + C2 + C3) и нажмите <Enter>.

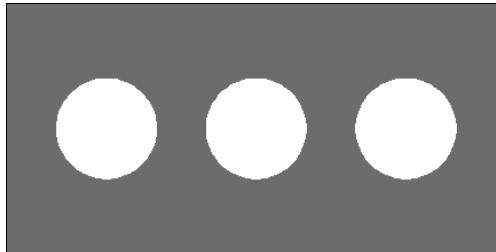


Рис. 14.16. Прямоугольник с отверстиями

В режиме рисования на области присутствуют все созданные примитивы, даже если они вычтены из других примитивов. Для отображения реальной геометрии области перейдите в режим дифференциального уравнения (пункт **PDE Mode** в меню **PDE**).

### Примечание

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 14.17. Очевидно, что следует вычесть из прямоугольника четыре небольших квадрата, целиком расположенных внутри прямоугольника, вершины которых совпадают с вершинами прямоугольника.

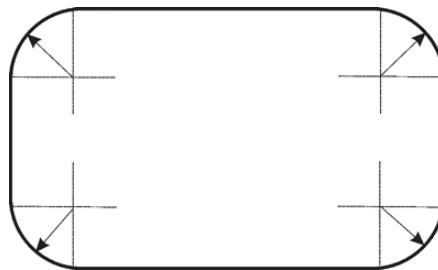


Рис. 14.17. Прямоугольник со скругленными углами

Для получения скругленных углов осталось объединить полученную область с четырьмя кругами, центры которых расположены в ближайших к центру прямоугольника вершинах удаленных квадратов, а радиусы равны стороне этих квадратов.

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.

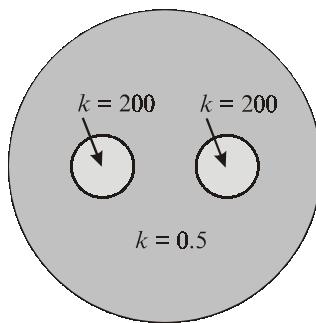
Объединение объектов позволяет просто задать геометрию области, однако следует иметь в виду, что при объединении сохраняются внутренние границы. Если область однородна, то следует удалить ненужные границы при помощи пункта **Remove Subdomain Border** меню **Boundary**, предварительно выделив их в режиме 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

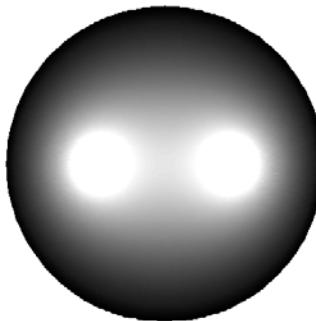
Исследуем стационарное распределение тепла в сечении теплоизолирующей оболочки с двумя нагревающимися стержнями, коэффициенты теплопроводности оболочки и стержней различны (рис. 14.18). Внешняя граница оболочки поддерживается при температуре ноль градусов, в оболочке нет источников тепла, а плотность 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.

Выберите подходящую триангуляцию и решите задачу. В окне среды **pdetool** отображается искомое распределение температуры в сечении рассматриваемой конструкции (рис. 14.19).



**Рис. 14.18.** Задача о распределении температуры в теплоизолирующей оболочке



**Рис. 14.19.** 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 PDE Toolbox

Среда `pdetool` с графическим интерфейсом пользователя предназначена лишь для облегчения доступа к ядру PDE Toolbox — 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

Имеются два способа определения геометрии области, в которой решается задача. Первый заключается в параметрическом задании частей границ области 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. 14.20. Данное окно предлагает со-

хранить 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 **OK** и займитесь изучением содержимого экспортированных массивов.

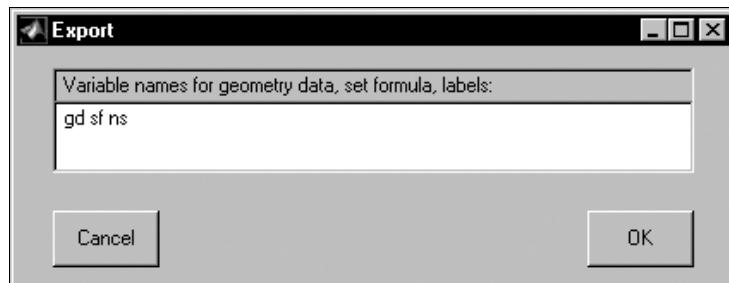


Рис. 14.20. Диалоговое окно Export

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 определяет принадлежность определенному геометрическому примитиву. Всего допустимы четыре типа геометрических

примитивов: круг, многоугольник, прямоугольник и эллипс. Первый элемент, 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.

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. Каждый столбец матрицы декомпозиционной геометрии отвечает элементу границы. 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
```

Очевидно, что прямоугольник представляется четырьмя отрезками. Двойка в первом элементе столбца означает, что столбец соответствует отрезку, второй и третий элементы 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(d1)` 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 М-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 `d1`. Убедитесь, что в глобальной переменной `d1` содержится матрица декомпозиционной геометрии области, создание которой описано выше. В командном окне выполните `wgeom(d1, 'recgeom')`. В текущем каталоге MATLAB появился файл `recgeom.m`, структура которого приведена в листинге 14.1.

#### Листинг 14.1. 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
```

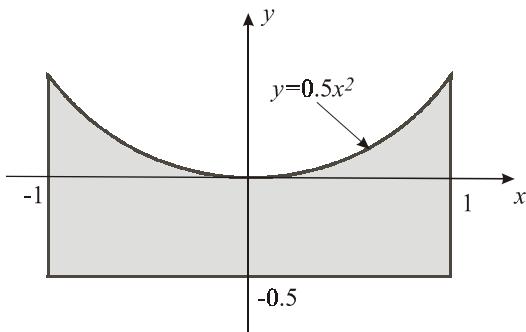
```
% boundary segment 3
ii = find(bs == 3);
if length(ii)
x(ii) = (-1 - (1))*(s(ii) - d(1, 3)) / (d(2, 3) - d(1, 3)) + (1);
y(ii) = (-0.5 - (-0.5))*(s(ii) - d(1, 3)) / (d(2, 3) - d(1, 3)) + (-0.5);
end

% 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 элементе 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. 14.21. Область является криволинейным четырехугольником с верхней границей, задаваемой параболой  $y = 0.5x^2$ . Нижние углы прямые.



**Рис. 14.21.** 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  $dl = decsg(gd, sf, ns)$ .
4. Сгенерируйте файл-функцию mydomain с декомпозиционной геометрией области, используя вызов `wgeom(dl, 'mydomain')`.
5. Откройте файл mydomain.m в редакторе М-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  $y$  у 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 =
```

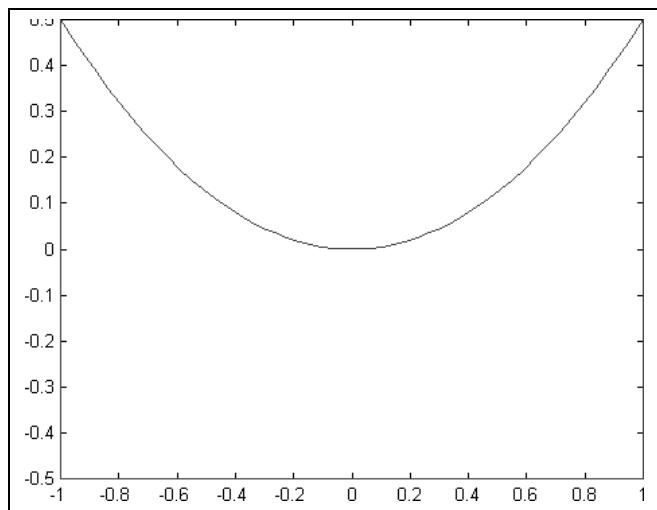
```
0.5000
```

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 14.2.

#### Листинг 14.2. 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. 14.21. Постройте геометрию области при помощи `pdegplot`. Вызов `pdegplot('mydomain')` приводит к появлению графического окна с границей (рис. 14.22).



**Рис. 14.22.** Геометрия области (`pdegplot ('mydomain')`)

Вышеописанный подход позволяет просто создать файл-функцию с декомпозиционной геометрией области, изменения автоматически сгенерированную файл-функцию для нарисованной заготовки в среде pdetool.

Наличие матрицы декомпозиционной геометрии или файл-функции открывает доступ ко многим возможностям PDE Toolbox, 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 нумерацией узлов.

### Примечание

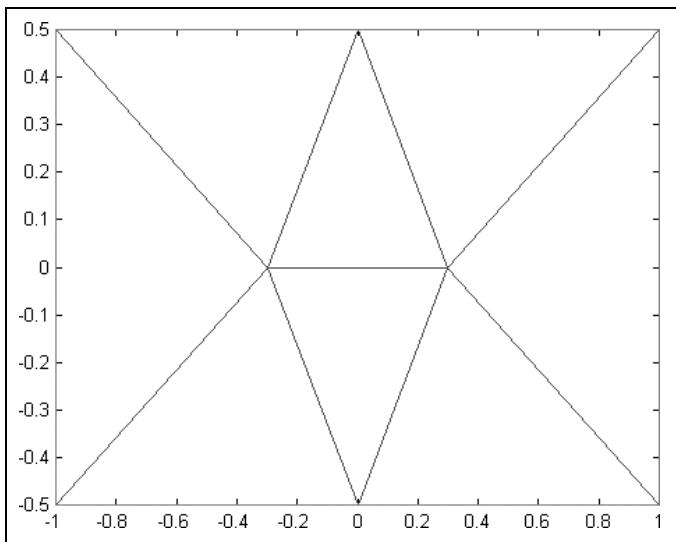
Понять структуру массивов `p`, `e` и `t` проще всего следующим образом. В среде `pdetool` следует инициализировать сетку, например, для прямоугольника, причем лучше установить достаточно крупный размер стороны элемента в диалоговом окне **Mesh Parameters**. Пронумеровать узлы и элементы позволяет выбор пунктов **Show Node Labels** и **Show Triangle Labels** меню **Mesh**. Далее необходимо экспорттировать сетку в глобальные массивы при помощи пункта **Export Mesh...**, их содержимое можно посмотреть в рабочей среде.

Функция `initmesh` может содержать дополнительные аргументы, задаваемые параметрами: название свойства, значение. Например, свойство `Hmax` означает максимально допустимое значение длины стороны треугольного элемента.

Полученную сетку можно отобразить в отдельном окне при помощи `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. 14.23.



**Рис. 14.23.** Сетка с максимально допустимой длиной стороны элемента, равной единице

Функция `refinemesh` служит для уменьшения шага сетки. Первым входным аргументом является матрица декомпозиционной геометрии, далее задаются массивы с информацией о триангуляции, в выходных аргументах воз врачаются 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. 14.23 и рис. 14.24).

```
[p1, e1, t1] = refinemesh(dl, p, e, t);
pdemesh(p1, e1, t1)
```

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.

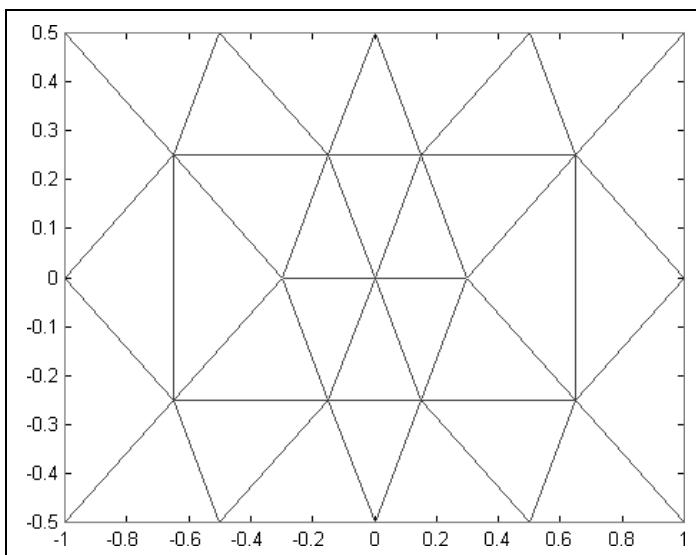


Рис. 14.24. Сетка с уменьшенным шагом (`refinemesh`)

## Границные условия и коэффициенты уравнения

Границные условия в PDE Toolbox 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 хра-

нения матрицы граничных условий и, одновременно, предлагается сохранить декомпозиционную геометрию в `g`. Получение декомпозиционной геометрии из матрицы геометрии области при помощи функции `decsq` 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 PDE Toolbox, 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  $b$ , 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. PDE Toolbox обладает несколькими солверами для нахождения решения различных типов задач:

- `assemPDE` — для эллиптических уравнений и систем;
- `parabolic` — для параболических уравнений и систем;
- `hyperbolic` — 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 метода 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 матрица 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 вектор  $u$  значений решения в узлах сетки, координаты которых хранятся в  $p$ . 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 PDE Toolbox. Использование всех возможностей `assemPDE` требует понимания того, как учитываются граничные условия при вычислении матриц и векторов правых частей элементов. Файл `pde.pdf` содержит разд. "The Finite Element Method" с необходимой информацией.

Солверы `parabolic` и `hyperbolic` сразу выдают матрицу с решением. Входные аргументы данных солверов такие же, как у `assempe`; 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)
```

Выходным аргументом `pdeeig` являются матрица, каждый столбец которой содержит значения собственной функции в узлах сетки, определяемых массивом `r`, и `l` — вектор собственных значений.

Солвер `pdenonlin`, предназначенный для решения нелинейных стационарных задач, имеет те же входные параметры, что и `assempe`

```
[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 PDE Toolbox.

### Примечание

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, принятными 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 `pdeplot` 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. 14.1).

*Таблица 14.1. Дополнительные параметры pdeplot*

| Свойство              | Значение                                                                                              |
|-----------------------|-------------------------------------------------------------------------------------------------------|
| <code>xydata</code>   | Вектор со значениями решения в узлах для построения двумерного графика                                |
| <code>xystyle</code>  | <code>off</code> , <code>flat</code> , <code>interp</code> (по умолчанию) — способы заливки контуров  |
| <code>contour</code>  | <code>on</code> , <code>off</code> (по умолчанию) — отображение или скрытие контурных линий           |
| <code>zdata</code>    | Вектор со значениями решения в узлах для построения трехмерного графика                               |
| <code>colormap</code> | <code>cool</code> , <code>hot</code> , <code>gray</code> , <code>bone</code> , ... — цветовые палитры |
| <code>mesh</code>     | <code>on</code> , <code>off</code> (по умолчанию) — отображение конечноэлементной сетки               |
| <code>colorbar</code> | <code>off</code> , <code>on</code> (по умолчанию) — вывод шкалы соответствия цвета и значения         |

Таблица 14.1 (окончание)

| Свойство | Значение                                                                          |
|----------|-----------------------------------------------------------------------------------|
| 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 PDE Toolbox.

## 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 М-файла с декомпозиционной геометрией `tug.m` и граничными условиями `tubg.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 М-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 14.3 содержит текст основной файл-функции `modelexam` и подфункции `exsol`, снабженный подробными комментариями.

### Листинг 14.3. Решение модельной задачи

```
function modelexam(gfile, bfile, err)
% Файл-функция находит решение модельной граничной
% задачи Дирихле для эллиптического дифференциального
```

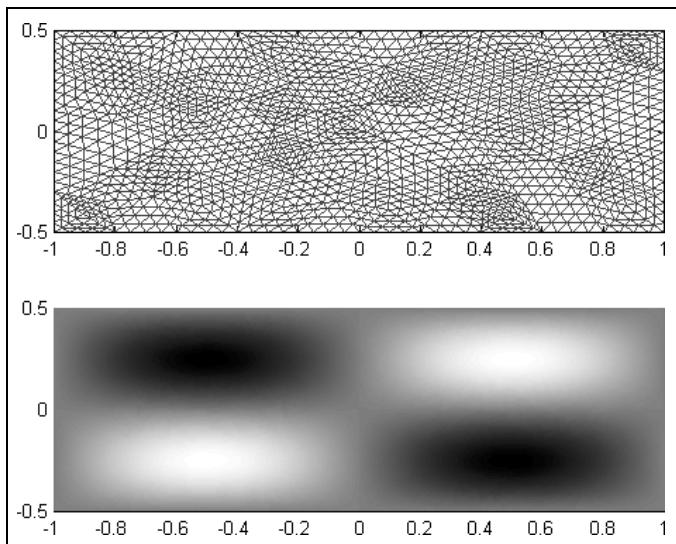
```
% уравнения в прямоугольной области.
% Использование:
% modelexam('файл с декомп. геом.', 'файл с гран. услов.', err)

% 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 хранятся в строках матрицы p
 ueh = 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 - ueh', 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('туг', 'тув', 1.0e-02)` приводит к появлению графического окна, в которое выводятся расчетная сетка и заливкой цветом контурный график решения модельной задачи с требуемой точностью (рис. 14.25).



**Рис. 14.25.** Расчетная сетка и решение модельной задачи

PDE Toolbox содержит несколько демонстрационных файлов: `pdedemo1.m`—`pdedemo8.m` с решением различных задач. Данные примеры позволяют самостоятельно разобраться в возможностях PDE Toolbox. М-файл `pdedemos.m` является файл-программой с интерфейсом командной строки, облегчающей доступ к каждому из примеров.

## Функции PDE 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 MATLAB по PDE Toolbox.

## 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 полуосами `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.

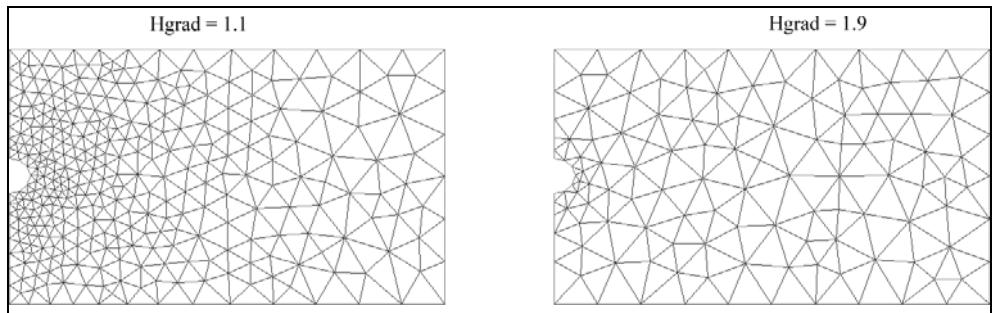
## Геометрия области и триангуляция

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 М-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. 14.26).

Ряд параметров `initmesh` предназначен для визуализации этапов алгоритма построения триангуляции и улучшения ее качества, подробная информация об их использовании приведена в документации к PDE Toolbox. Кроме того, функция `jigglemesh` реализует алгоритм улучшения триангуляции.

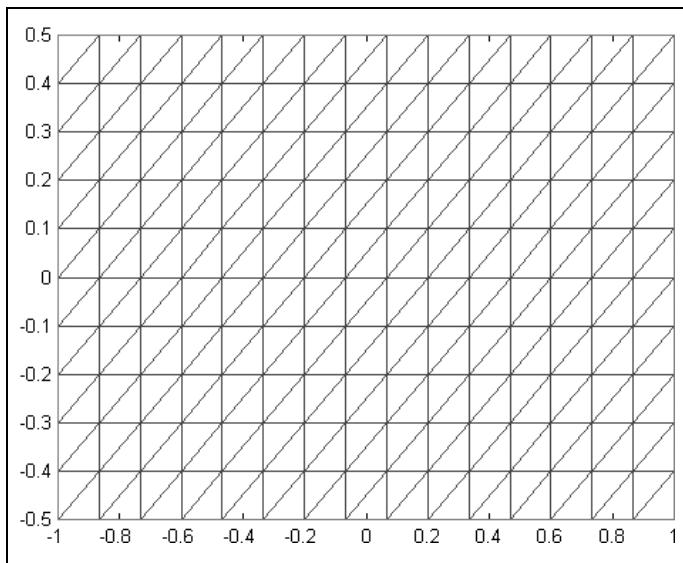


**Рис. 14.26.** Влияние параметра `Hgrad` на размеры элементов

- Функции `poimesh(gl, nx, ny)`, `poimesh('mygeom', nx, ny)` служат для инициализации *регулярной сетки* на *прямоугольной* области, декомпозиционная геометрия которой задается матрицей или М-файлом. Число узлов по каждой из координат задается в `nx` и `ny`. Последовательность команд для прямоугольника, декомпозиционная геометрия которого описана файл-функцией `myg`

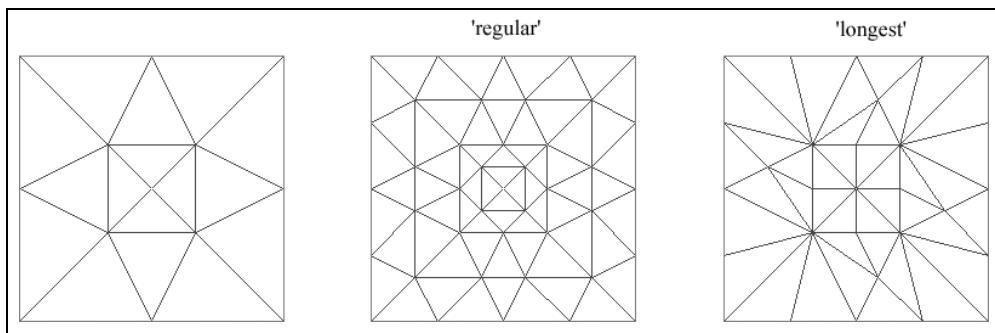
```
[p, e, t] = poimesh('myg', 15, 10);
pdemesh(p,e,t)
```

приводит к регулярной сетке, изображенной на рис. 14.27. Функция `poimesh` предназначена только для прямоугольных областей.

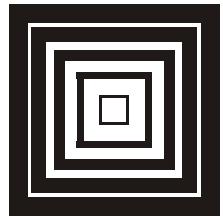


**Рис. 14.27.** Регулярная сетка в прямоугольнике

□ `[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 измельчения сетки делением наибольшей части треугольника на две части, по умолчанию используется '`regular`'. Исходная и преобразованные каждым из описанных способов сетки приведены на рис. 14.28.



**Рис. 14.28.** Способы уменьшения шага сетки



## Глава 15

# Разреженные матрицы

Матрицы, содержащие достаточно большое число нулевых элементов, называются *разреженными*. Разреженные матрицы 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 размещения в памяти разреженных матриц и алгоритмов выполнения операций над ними существенно снижает затраты компьютерных ресурсов и уменьшает время вычислений. Строго говоря, матрица является разреженной, если специальные алгоритмы в сочетании с компактной схемой хранения ее элементов дают выигрыш по сравнению с обычными методами, применяемыми к полностью заполненным матрицам.

## Работа с разреженными матрицами

Разреженные матрицы содержат значительное число нулевых элементов по сравнению с ненулевыми. При большой размерности матрицы хранить все элементы, включая нулевые, нет смысла. Достаточно записать в память только ненулевые значения и информацию об их положении в матрице. Очевидно, эффект будет достигаться лишь тогда, когда для запоминания значений ненулевых элементов и дополнительных данных потребуется меньше памяти и время выполнения алгоритмов, использующих эти данные, будет меньше времени выполнения операций над полностью заполненными матрицами. В качестве примера рассмотрим одну из возможных схем хранения, реализованную в MATLAB.

## Схема хранения

Разреженные матрицы в MATLAB помещаются в специальные переменные типа `double array (sparse)`. Ненулевые элементы матриц хранятся по столбцам сверху вниз, для каждого элемента запоминается номер строки и

столбца, соответствующие его положению в исходной полностью заполненной матрице.

Задайте из командной строки прямоугольную матрицу размера пять на шесть:

```
>> 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];
```

Компактное представление AN полностью заполненной матрицы A позволяет получить функция sparse:

```
>> 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 столбец 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. 15.1.

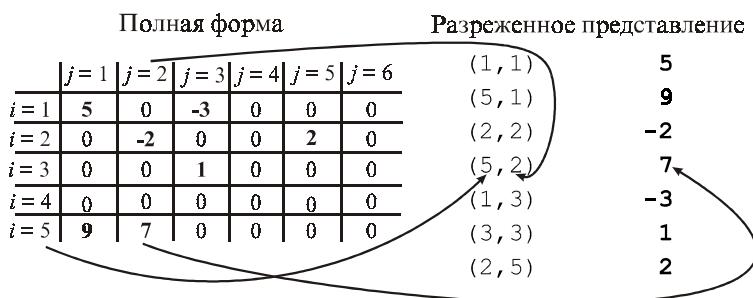


Рис. 15.1. Схема хранения

Обратите внимание, что вышеописанная схема существенно экономит память, выделяемую под хранение матрицы. Действительно, под массив A от-

водится  $5 \cdot 6 = 30$  ячеек с вещественными числами, каждая размера 8 байт, 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 посмотреть информацию о массивах в окне **Workspace** браузера рабочей среды или воспользоваться командой whos:

```
>> whos A AN
```

| Name | Size | Bytes | Class                 |
|------|------|-------|-----------------------|
| A    | 5x6  | 240   | double array          |
| AN   | 5x6  | 112   | double array (sparse) |

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 вернуться к обычному представлению 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 заполненную 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 double array (sparse).

## Создание разреженных матриц

Функция sparse предоставляет возможность непосредственного создания массива типа double array (sparse), соответствующего разреженной матрице. Обращение к sparse выглядит следующим образом:

```
AN = sparse(irow, jcol, nzer, m, n)
```

Здесь irow — вектор строчных индексов ненулевых элементов матрицы, а jcol — вектор столбцевых индексов ненулевых элементов матрицы, которые помещены в вектор nzer. В последних аргументах m и n задаются разме-

ры исходной полностью заполненной 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 `full(AN)`. Обход по столбцам в заполненной матрице 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];
>> nzer = [5 -3 -2 2 1 9 7];
>> AN = sparse(irow, jcol, nzer, 5, 6);
```

Более того, можно просматривать элементы разреженной матрицы в произвольном порядке, важно только правильно указать номера строковых и столбцевых индексов ненулевых элементов 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. соответствующие им пары с номерами строк и столбцов совпадают, 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. В этом можно убедиться при помощи функции `find`, которая допускает вызов не только от массивов типа `double array`, но и `double array (sparse)` и возвращает строчные и столбцевые индексы ненулевых элементов и их значения:

```
>> 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];
>> AN = sparse(irow, jcol, nzer, 5, 6);
>> [ir, jr, nz] = find(AN)
```

Интерфейс функции `sparse` поддерживает обращение к ней с переменным числом входных аргументов. Если размеры матрицы `m` и `n` не указаны, то в качестве них будут использоваться максимальные значения компонентов векторов `irow` и `jcol`. 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 размером 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 строковые и столбцевые индексы ненулевых элементов, а в третьем — их значения. Содержимое текстового файла `spmatr.dat` для матрицы `A` из данного раздела приведено в листинге 15.1.

#### Листинг 15.1. Текстовый 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
```

Считайте данные из файла `spmatr.dat` в некоторый массив и преобразуйте их в компактную форму хранения:

```
>> s = load('spmatr.dat');
>> A = spconvert(s);
```

## Примечание

Элементы матрицы могут быть комплексными. Использование `sparse` аналогично случаю вещественных матриц, вектор `nzer` в функции `sparse` должен содержать комплексные элементы. Импортирование комплексной матрицы из файла имеет одну особенность — файл должен иметь четыре столбца, причем первые два содержат строчные и столбцевые индексы ненулевых элементов матрицы, а третий и четвертый — их вещественную и мнимую части.

Матрицы, ненулевые элементы которых расположены на главных или побочных диагоналях, часто хранят по диагоналям, записывая их в столбцы вспомогательной матрицы меньшего размера. Например, матрицу

$$A = \begin{pmatrix} 5 & 0 & -3 & 0 & 0 \\ 1 & 3 & 0 & -1 & 0 \\ 0 & 1 & 7 & 0 & -2 \\ 0 & 0 & 1 & 22 & 0 \\ 0 & 0 & 0 & 1 & 8 \end{pmatrix}$$

можно упаковать следующим образом. Занесите в столбцы матрицы  $B$  диагонали  $A$ , начиная с нижней. Недостающие элементы побочных диагоналей следует дополнить нулями, помещая их в конец столбца для нижних диагоналей (поддиагоналей) и в начало столбца для верхних диагоналей (наддиагоналей):

```
>> 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];
```

Для перехода от формы хранения матрицы по диагоналям к компактной форме служит функция `spdiags`. Аргументами `spdiags` являются: массив  $B$  с диагональными элементами, вектор  $d$  с номерами диагоналей и размеры формируемой матрицы:

```
>> A = spdiags(B, d, 5, 5);
```

### Примечание

Поскольку диагонали могут быть разной длины, то функция `spdiags` использует не все элементы столбцов матрицы  $B$ . При заполнении поддиагоналей выбираются элементы, начиная сверху столбца, а при заполнении наддиагоналей — снизу.

Убедитесь в правильности результата, выведите заполненную и разреженную формы матрицы  $A$ . Указание в качестве входного аргумента  $B$  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 `spru` выводит шаблон матрицы в 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

Представление матрицы, хранящейся в массиве  $A$ , в полной форме имеет очевидную связь с соответствующими векторами  $irow$ ,  $jcol$  и  $nzer$ . Для нахождения значения элемента исходной матрицы  $A$  с произвольными индексами  $i$  и  $j$  в ее компактном представлении требуется произвести поиск вспомогательного индекса  $k$ , удовлетворяющего условию:  $i = irow(k)$ ,  $j = jcol(k)$ . Если такого номера  $k$  нет, то элемент матрицы нулевой, иначе  $A(i, j) = A(irow(k), jcol(k)) = nzer(k)$ . MATLAB производит поиск автоматически, позволяя получать доступ к элементам матриц, хранящихся

в массивах типа `double array (sparse)`, 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 скрыта от пользователя. Данные операции производятся над разреженными матрицами так же, как над обычными, и в результате получается разреженная матрица, хранящаяся в массиве типа `double array (sparse)`.

### Примечание

Работа с разреженными массивами `double array (sparse)` с точки зрения пользователя не отличается от работы с обычными массивами `double array`, хотя на самом деле используются другие алгоритмы. Выполнение разных алгоритмов для полностью заполненных матриц и разреженных достигается за счет того, что `sparse` является разновидностью (атрибутом) `double 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 разреженной матрице, если хотя бы один из множителей записан в компактной форме. Результатом поэлементного деления разреженной матрицы на полностью заполненную является разреженная матрица. Значения математических функций от разреженных матриц также являются разреженными, например, `sin(AN)`. Математические функции применяются поэлементно, так же как и к обычным матрицам. Обратите внимание на то, что после вычисления функции от разреженной матрицы или выполнения других операций с разреженными матрицами можно потерять преимущества компактного хранения и дальнейшей обработки этих матриц специальными алгоритмами. Рассмотрим пример вычисления функции `cos` от ранее введенной матрицы `AN`, после чего преобразуем ее в полностью заполненную матрицу и сравним память, требуемую для хранения обоих массивов.

```
>> CAN=cos(AN);
>> whos CAN
 Name Size Bytes Class
 CAN 5x6 388 double array (sparse)
Grand total is 30 elements using 388 bytes
```

```
>> ca = full(CAN);
>> whos ca
 Name Size Bytes Class
 ca 5x6 240 double array
Grand total is 30 elements using 240 bytes
```

В результате оказалось, что для хранения разреженной матрицы требуется больше памяти. Это связано с тем, что после вычисления косинуса матрица стала полностью заполненной, но хранится в форме разреженной матрицы, т. е. с избыточной информацией о номерах строчных и столбцовых индексов каждого элемента.

В том случае, когда необходимо вычислить значения только от ненулевых элементов, следует использовать spfun. Входными аргументами spfun являются указатель на файл-функцию или встроенную математическую функцию (или ее имя, или inline-функция) и массив типа double array (sparse) с компактной формой хранения матрицы.

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. Рассмотрим пример с ранее построенной матрицей A :

```
>> gn = sparse(A)
```

```
>> max(gn)
```

```
ans =
```

|        |    |
|--------|----|
| (1, 1) | 5  |
| (1, 2) | 3  |
| (1, 3) | 7  |
| (1, 4) | 22 |
| (1, 5) | 8  |

```
>> min(gn)
```

```
ans =
```

|        |    |
|--------|----|
| (1, 3) | -3 |
| (1, 4) | -1 |
| (1, 5) | -2 |

Результат функции min объясняется тем, что в первых столбцах минимальные элементы нулевые.

Конструирование блочных разреженных матриц и индексация при помощи вектора значений индексов не отличаются от случая полностью заполненных матриц.

Структура разреженной матрицы определяет наиболее удобный способ ее формирования. Справочная система MATLAB содержит пример генерации симметричной разреженной матрицы, соответствующей разностной аппроксимации второй производной. На главной 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 (см. разд. **Mathematics: Sparse Matrices: Creating Sparse Matrices** справочной системы MATLAB).

Информацию о разреженной матрице можно получить при помощи нескольких функций MATLAB. Как было упомянуто выше, 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`.

В приложениях часто встречаются ленточные матрицы, все ненулевые элементы которых расположены достаточно близко к главной диагонали. Важная характеристика ленточной матрицы  $B$  — ширина ее ленты  $b$ , которая определяется как

$$b = \max_{a_{i,j} \neq 0} |i - j|.$$

Возникает вопрос, как найти ширину ленты матрицы, хранящейся в компактной форме, к примеру в массиве `BN`. Очевидно, что для вычисления ширины ленты достаточно обратиться к функции `find` и `max`:

```
[irow, jcol] = find(BN);
b = max(irow - jcol);
```

При программировании собственных алгоритмов полезно убедиться, что матрица является разреженной, т. е. массивом типа `double array (sparse)`. 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 прямым и итерационными методами, нахождение собственных чисел и векторов. Алгоритмы, реализованные в файл-функциях, учитывают разреженность матриц, благодаря чему являются очень эффективными. Данный раздел посвящен обзору основных алгоритмов, приведен пример использования профайлера MATLAB для получения временных затрат алгоритмов.

### Факторизация матриц

MATLAB предлагает несколько классических способов факторизации разреженных матриц: полное и неполное разложение Холецкого,  $LU$ -разложение,  $QR$ -разложение. Факторизация разреженных матриц имеет свои особенности — следует стремиться к тому, чтобы получить как можно более разреженные множители, входящие в разложение. Продемонстрируем технику разложения разреженных матриц на примере разложения Холецкого симметричной положительно определенной матрицы

$$B = \begin{pmatrix} 4 & 1 & 0 & 0 & 1 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 1 \\ 1 & 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 1 & 1 & 4 \end{pmatrix}.$$

При помощи функции `sparse` создайте массив `BN` с разреженной матрицей  $B$ .

```
>> irow = [1 1 1 2 2 2 3 3 4 4 5 5 5 6 6 6];
>> jcol = [1 2 5 1 2 3 2 3 4 6 1 5 6 4 5 6];
>> nzer = [4 1 1 1 4 1 1 4 4 1 1 4 1 1 1 4];
>> BN = sparse(irow, jcol, nzer, 6, 6)
```

Найдите разложение Холецкого  $R^T R = B$  данной матрицы, используя функцию `chol`, которая возвращает верхнюю треугольную матрицу  $R$  в компактной форме.

```
>> R = chol(BN);
```

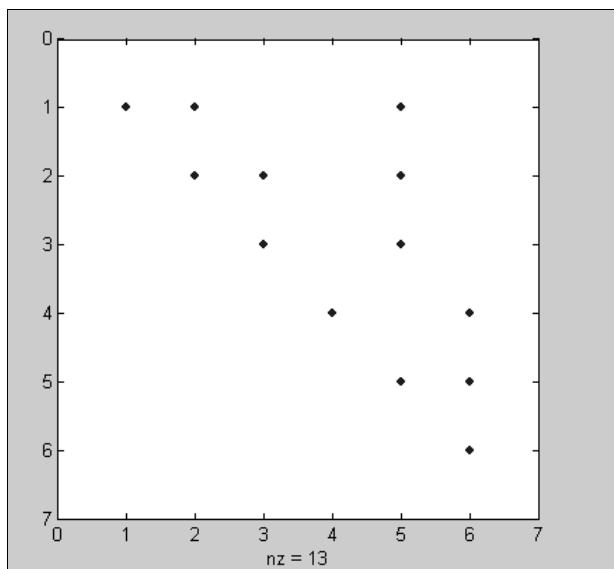
Отобразите шаблон матрицы  $R$ , используя команду `spy`, получающееся расположение ненулевых элементов приведено на рис. 15.2. Матрица  $R$ , входящая в разложение, имеет довольно большой разброс ненулевых элементов относительно главной диагонали. На практике стремятся свести

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 номеров строк и столбцов. В рассматриваемом примере получается следующий вектор

```
>> rind = symrcm(BN)
```

```
rind =
```

```
3 2 1 5 6 4
```



**Рис. 15.2.** Разложение Холецкого без переупорядочения

Сама матрица  $B$ , 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  $B$  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 |

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  $\text{nz}$  в графическом окне с шаблоном, в матрице  $R$  (рис. 15.3) свидетельствует об эффективности предварительных перестановок в исходной матрице с целью уменьшения ширины ленты.

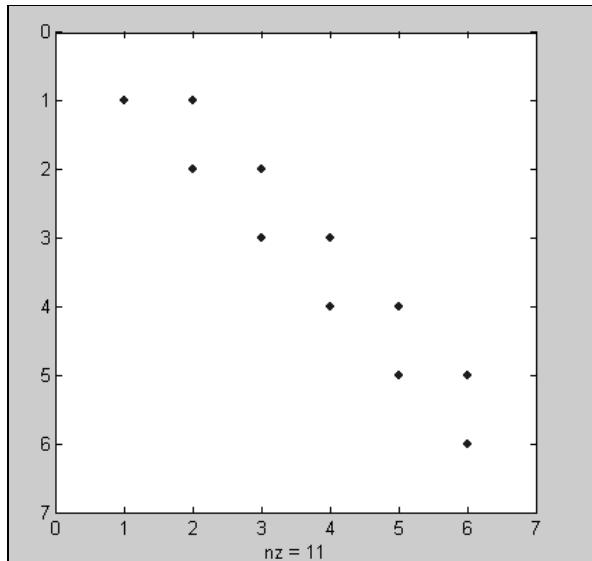


Рис. 15.3. Разложение Холецкого с переупорядочением

Заметьте, что поскольку в исходной матрице были произведены перестановки, то множитель  $R$  разложения Холецкого соответствует измененной матрице. Для факторизации исходной матрицы следует выполнить обратные перестановки. Проверьте это, вычислив:

```
>> (R(rind, rind))' * R(rind, rind)
```

Перед разложением Холецкого  $B = R^T R$  разреженных матриц имеет смысл попытаться уменьшить ширину ленты факторизуемой матрицы за счет сим-

метричной перестановки строк и столбцов, поскольку разложение не приводит к образованию новых ненулевых элементов вне ленты матрицы, т. е. ширина ленты матрицы  $R^T + R$  не больше ширины ленты  $B$ . Заметим, что функция `symrcm` основана на алгоритме Катхилла—Макки с обратным упорядочением, который уменьшает не только ширину ленты матрицы, но и ее *профиль*. Профильная схема является обобщением ленточной — допускается переменная ширина ленты в каждой строке матрицы, а количество всех элементов матрицы, входящих в такую ленту переменной ширины (оболочку), называется профилем. Разложение Холецкого сохраняет профиль  $R^T + R$  такой же, как у исходной матрицы, т. е. новые ненулевые элементы вне оболочки не появляются.

Выполните разложение Холецкого матрицы, приведенной ниже, без предварительного уменьшения профиля, затем примените алгоритм Катхилла—Макки с обратным упорядочением для уменьшения профиля и сравните количество ненулевых элементов в множителях разложения Холецкого.

$$B = \begin{pmatrix} 7 & 1 & 1 & 1 & 1 \\ 1 & 7 & 0 & 0 & 0 \\ 1 & 0 & 7 & 0 & 0 \\ 1 & 0 & 0 & 7 & 0 \\ 1 & 0 & 0 & 0 & 7 \end{pmatrix}.$$

Предварительное упорядочение строк и столбцов с целью уменьшения профиля или ширины ленты симметричной матрицы не гарантирует минимальное заполнение множителя разложения Холецкого, поскольку возможно появление достаточно большого количества ненулевых элементов внутри ленты в множителях разложения. Более эффективное упорядочение производят функции `symamd` и `symmmad`, основанные на алгоритме минимальной степени. Сравнение `symmmad` и `symrcm` приведено в справочной системе MATLAB на примере матрицы размера 60 (см. описание функции `symmmad`, к которому легко перейти из разд. **MATLAB Functions: Functions – By Category: Sparse Matrices** с гиперссылками на функции. Несколько примеров расположено также в разд. **MATLAB: Matrices: Sparce Matrices**. Для перехода к нему следует выбрать вкладку **Demo** окна справочной системы MATLAB).

### Примечание

Алгоритмы Катхилла—Макки и минимальной степени являются эвристическими. Они не дают гарантии, что найденные перестановки строк и столбцов приведут к наименьшей ширине ленты, профилю или заполнению множителей разложения Холецкого. Однако для многих практических важных задач они обеспечивают хорошие результаты и широко используются в вычислительной практике.

Мы рассмотрели примеры разложения Холецкого симметричных положительно определенных матриц. Переходим теперь к обзору остальных функций, предназначенных для факторизации разреженных несимметричных матриц.

*LU*-разложение матрицы находится при помощи функция `lu`. Если разреженная матрица  $A$  хранится в компактной форме в массиве `AN`, то обращение  $[L, U, P, Q] = \text{lu}(AN)$  приводит к записи в выходные аргументы соответствующих матриц разложения — нижней унитреугольной  $L$ , верхней треугольной  $U$  и матриц перестановок  $P$  и  $Q$  таких, что  $PAQ = LU$ . Все выходные аргументы функции `lu` принадлежат типу `double array (sparse)`. Матрица  $P$  соответствует перестановкам строк, обеспечивающим устойчивость вычислительного процесса, а  $Q$  — перестановкам столбцов, уменьшающим заполнение множителей разложения.

При *LU*-разложении требуется найти компромисс между двумя требованиями: вычислительной устойчивостью и, по возможности, сохранением разреженности множителей разложения. Для обеспечения вычислительной устойчивости производится перестановка строк так, чтобы на диагонали текущей строки разложения оказался достаточно большой по модулю элемент. По умолчанию строки переставляются, если модуль диагонального элемента в 10 раз меньше, чем максимальный модуль элементов того же столбца, лежащих ниже диагонали. В этом случае алгоритм `lu` пытается одновременно минимизировать заполнение множителей. Функция `lu` позволяет установить другие значения этого коэффициента, для чего ее следует вызвать с дополнительным вторым входным аргументом (по умолчанию он равен 0.1, что соответствует десятикратному превосходству). Второй аргумент может принимать значения от 0 до 1, причем чем он меньше, тем более разреженными могут получиться множители *LU*-разложения, возможно, за счет потери вычислительной устойчивости. Напротив, увеличение значения второго аргумента улучшает вычислительную устойчивость, однако может повлечь существенное возрастание количества ненулевых элементов в матрицах  $L$  и  $U$ .

Кроме *LU*-разложения в матричных вычислениях применяются и другие способы факторизации матриц, в частности, *QR*-разложение. Для заданной матрицы  $A$  требуется найти верхнюю треугольную матрицу  $R$  и ортогональную  $Q$  (т. е.  $Q^T Q$  — единичная матрица) такие, чтобы выполнялось равенство  $A = QR$ . Эта задача решается при помощи функции `qr`. Отметим, что функции `lu` и `qr` позволяют выполнить соответствующие разложения прямоугольных матриц и комплексных матриц.

Функции `luinc` и `cholinc` вычисляют неполное *LU*-разложение и неполное разложение Холецкого, которые могут использоваться для ускорения сходимости итерационных методов при решении системы линейных уравнений с исходной матрицей. Особенности применения `qr`, `luinc` и `cholinc` подробно описаны в справочной системе MATLAB.

Функции `lu`, `chol` и `qr` применимы и к матрицам, хранящимся в полной форме в массивах типа `double array`, однако если матрица разрежена, то для эффективной факторизации следует привлечь компактную форму записи в массивах `double array (sparse)`. Все алгоритмы MATLAB для разреженных матриц, хранящихся в массивах типа `double array (sparse)`, работают с учетом структуры матрицы, обеспечивая значительное ускорение вычислений по сравнению с обычными матрицами. Профайлер MATLAB позволяет убедиться в вышесказанном на примере *LU*-разложения.

## Профайлер

Профайлер MATLAB служит для получения информации о временных затратах на выполнение команд или М-файлов и является удобным средством для оптимизации алгоритмов. Профайлер помогает выявить блок операторов, работа которых занимает неоправданно большое время. Статистика временных затрат представляется в виде подробного отчета или наглядной столбцовой диаграммой. Выбор пункта **Profiler** в меню **Desktop** рабочей среды либо пункта **Open Profiler** меню **Tools** редактора М-файлов приводит к появлению окна профайлера. Ускорению программ в MATLAB посвящена глава 23.

Для использования средств профайлера следует указать имя М-файла в строке ввода раскрывающегося списка **Run this code** (если М-файл содержит файл-программу) или занести в нее вызов функции (если М-файл содержит файл-функцию) и нажать кнопку **Start Profiling**. После завершения работы в окне профайлера отображается собранная и обработанная статистика по выполнению этого файла.

Используйте профайлер для того, чтобы установить преимущество использования разреженной структуры матрицы на примере разложения Холецкого. Задана симметричная матрица  $S$  размера  $n = 2000$  с большим числом нулевых элементов

$$S = \begin{bmatrix} 3 & -1 & 0 & \cdots & 0 & 1 \\ -1 & 3 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 3 & -1 \\ 1 & 0 & 0 & \cdots & -1 & 3 \end{bmatrix}.$$

Наша задача состоит в сравнении временных затрат на разложение Холецкого матрицы  $S$ , записанной в компактной форме в массиве `SN` и в обычном массиве `S` типа `double array`. При конструировании массива `SN` типа `double array (sparse)` учтите, что матрица  $S$  отличается от трехдиагональной только двумя элементами, поэтому сначала сформируйте трехдиагональную

матрицу при помощи функции `spdiags`, а затем измените значения элементов  $S_{n,1}$  и  $S_{1,n}$ . Файл-программа `prof_sp`, приведенная в листинге 15.2, обеспечивает заполнение массива `SN`.

### Листинг 15.2. Файл-программа `prof_sp` для создания симметричной матрицы $S$

```
n = 2000; % dimension of matrix
e = ones(n, 1); % define unit column
% compose three-diagonal matrix in compact form
SN = spdiags([-e 3*e -e], -1:1, n, n);
% change two elements in matrix by 1
SN(1, n) = 1;
SN(n, 1) = 1;
% factorization sparse matrix
RN = chol(SN);
```

#### Примечание

В примере использованы комментарии на английском языке, поскольку в версии 7 пакета MATLAB профайлер не поддерживает альтернативные кодовые страницы.

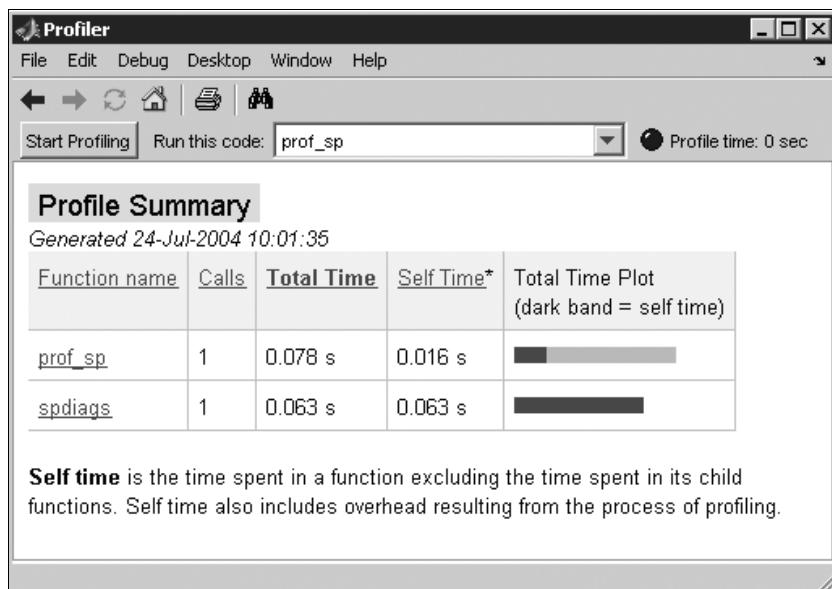


Рис. 15.4. Окно профайлера после выполнения М-файла `prof_sp.m`

Откройте окно профайлерса, наберите `prof_sp` в строке ввода раскрывающегося списка **Run this code** и нажмите кнопку **Start Profiling**. После завершения работы M-файла в окне профайлерса, представленном на рис. 15.4, появляется таблица с накопленной статистикой. Поля таблицы содержат: имена функций (**Function name**), количество вызовов (**Calls**), суммарное время работы файла для всех вызовов, включая выполнение вызываемых функций (**Total Time**), чистое время работы функции для всех вызовов (**Self Time**) и столбцовую диаграмму относительных затрат времени на выполнение вызываемых M-файлов, в которой более темным цветом выделено чистое время (**Total Time Plot**).

Интерфейс окна профайлерса достаточно прозрачный, поэтому отметим самое существенное. Для получения детальной информации о работе файл-программы `prof_sp` щелкните по гиперссылке с ее именем в таблице. Окно профайлерса принимает вид, изображенный на рис. 15.5.

### Примечание

Вид окна профайлерса может немного отличаться от приведенного на рис. 15.5, поскольку его содержимое определяется ранее сделанными установками флагов, о которых пойдет речь далее.

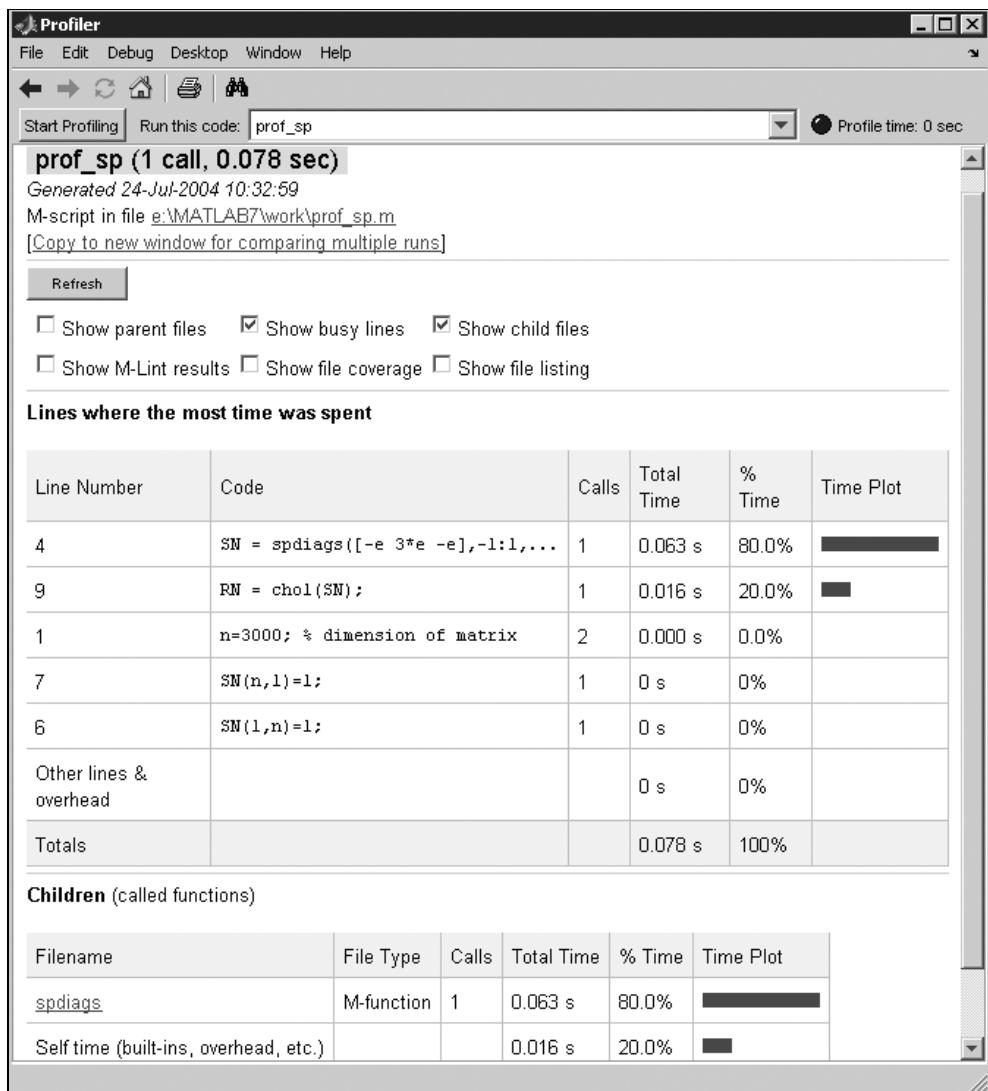
### Примечание

На панели инструментов профайлерса расположены кнопки **Back**, **Forward**, **Refresh**, предназначенные для листания страниц с информацией о работе M-файлов, так же как и в любом браузере.

Окно условно состоит из нескольких частей. В верхней части окна размещена общая информация и две гиперссылки. Щелчок по гиперссылке с именем файла приводит к его открытию в редакторе M-файлов. Гиперссылка **Copy to new window for comparing multiple runs** позволяет создать копию окна профайлерса для сравнения результатов при дальнейшем тестировании M-файла. Далее расположены флаги, определяющие содержимое окна, и кнопка **Refresh**, используемая для изменения его вида после переустановки флагов. Флаг **Show M-Lint results** отвечает за вывод информации средства M-Lint. О диагностике M-файлов с помощью M-Lint см. главы 5 и 21.

Содержимое окна на рис. 15.5 определяется установкой двух флагов. Флаг **Show busy line** управляет отображением информации о текущем файле. Табличная часть содержит временные затраты на выполнение операторов, упорядоченные в порядке убывания суммарного времени. Флаг **Show child files** обеспечивает вывод краткой информации о вызываемых функциях в разделе **Children**, расположенном ниже таблицы. Имена M-файлов, являющиеся одновременно гиперссылками, предназначены для перехода к страницам со сведениями о них. В данном случае в разделе **Children** находится ссылка на единственный M-файл `spdiags.m`. Хотя наша файл-программа обращается

еще и к функции `chol`, но ее имя не выводится, поскольку `chol` является встроенной функцией со скрытым кодом. Флаг **Show parent files** обеспечивает вывод раздела **Parents** с таким же содержанием, что и раздел **Children**, только с информацией о М-файле, из которого произошло обращение к текущему. В приводимом примере флаг отключен.



**Рис. 15.5.** Окно профайлера с таблицей о затратах времени при выполнении М-файла prof\_sp.m

В разд. "Подфункции" главы 5 описано, как при помощи специальной функции `exist` получить информацию о том, является ли некоторая функция встроенной или нет.

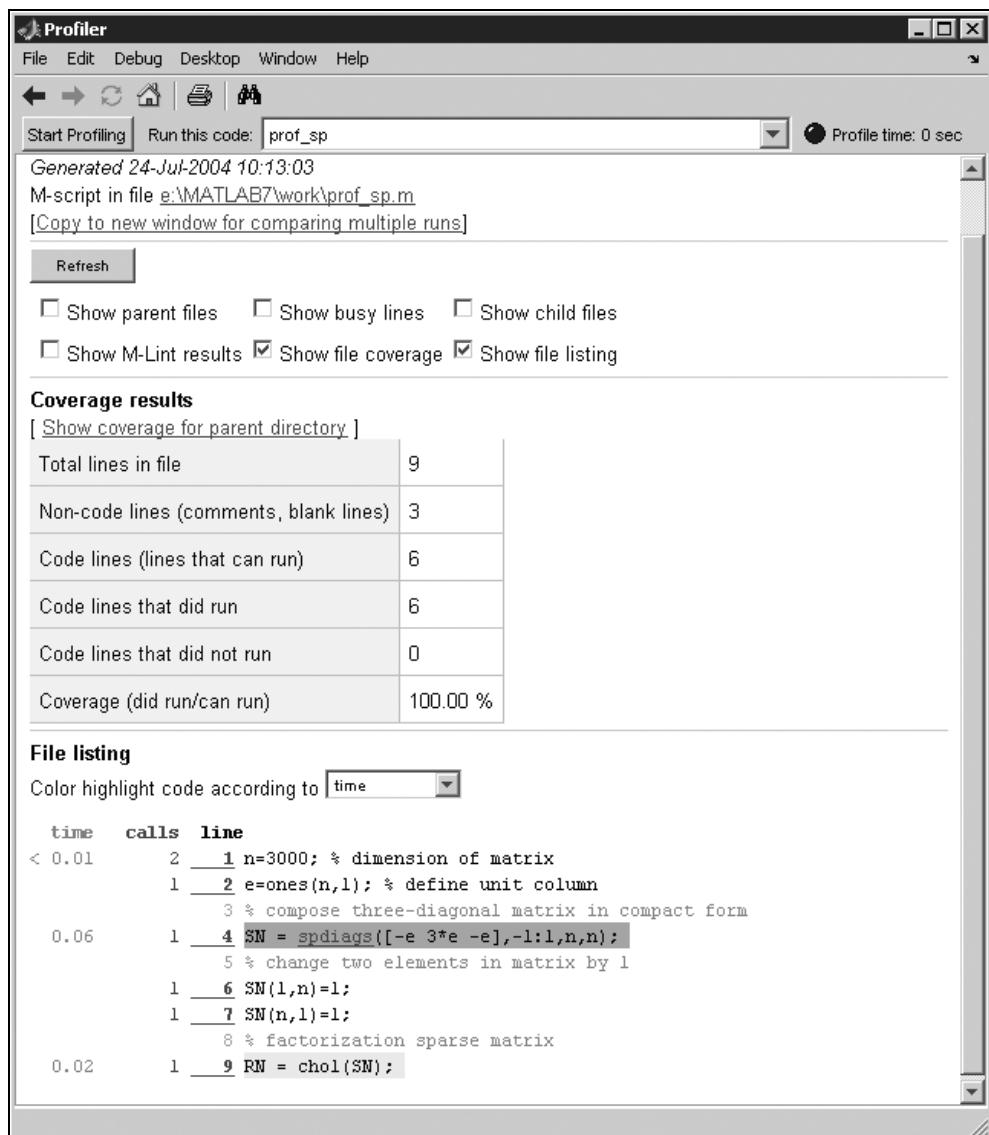


Рис. 15.6. Окно профайлера с листингом М-файла prof\_sp.m

На рис. 15.6 приводится окно профайлера, если установлены другие флаги: **Show file coverage** и **Show file listing**. Первый из них обеспечивает вывод суммарной информации в раздел **Coverage results** о текстовых строках файла, включая их общее количество, сколько из них комментариев, сколько исполняемых строк, сколько строк выполнились при текущем вызове профайлера и сколько строк не выполнялось.

После раздела **Coverage results** сформирован раздел **File listing**, за отображение которого отвечает флаг **Show file listing**. Этот раздел содержит листинг файл-программы `prof_sp`. Листинг расцвечен специальным образом, определяемым значением в списке **Color highlight code according to** (по умолчанию значение **time**). В приведенном примере выделены цветом операторы, выполнение которых потребовало наибольшего времени. Можно выбрать и другие критерии выделения цветом: по числу вызовов оператора (**numcalls**), выполнялся или нет оператор (**coverage** или **noncoverage**), по наличию диагностики средства M-Lint (**mlint**), или вообще отказаться от выделения цветом (**none**).

Слева от каждого оператора указаны время его выполнения (столбец **time**), число обращений к нему (столбец **calls**) и гиперссылка с номером строки программы (столбец **line**), щелчок по которой приводит к переходу к этой строке в редакторе M-файлов.

Разложение Холецкого разреженной матрицы на нашем компьютере при помощи специального алгоритма, реализованного в функции `chol`, потребовало 0.016 с (на разных компьютерах может получиться разное время счета). Сравните теперь это время со временем разложения матрицы, хранящейся в полной форме. Перед внесением изменений в файл-программу `prof_sp` удобно сделать копию окна профайлера при помощи гиперссылки вверху окна так, как было описано выше. Приведите массив `SN` к полной форме при помощи функции `full`, сохраните его в `S` и вызовите от него функцию `chol`. Запустите из профайлера на выполнение файл-программу `prof_sp` и посмотрите на существенную разницу во времени по сравнению с разложением Холецкого разреженной матрицы. На нашем компьютере (6.734 с) результаты различались примерно в 400 раз, и это для не очень больших размеров матрицы, поскольку в приложениях встречаются матрицы, размеры которых исчисляются миллионами. Кроме того, 0.125 с составило время преобразования разреженной матрицы в обычную. Очевидно, нет смысла комментировать эти результаты.

Среда профайлера является удобной оболочкой для всесторонней оценки работы программы, поиска неэффективных блоков и их оптимизации. Профайлер может быть открыт не только при помощи меню рабочей среды или редактора M-файлов, но и командой `profile` с опцией `viewer`

```
>> profile viewer
```

## Примечание

Если к моменту выполнения команды `profile viewer` предыдущие статистические данные не очищены, то будет открыто окно профайлера с суммарной собранной статистической информацией, о которой см. далее в этом разделе.

Мы упомянули об этой возможности не только для того, чтобы описать альтернативный способ вызова профайлера. Функция `profile` имеет достаточно большой набор опций, позволяющих оценивать характеристики программы, причем ее возможности шире, чем те, которые предоставляет среда профайлера. Например, по завершении сбора информации генерируется отчет в формате HTML, который открывается в установленном по умолчанию браузере Web-страниц и может быть сохранен.

Для активизации профайлера предназначен параметр `on`. Команда

```
>> profile on
```

приводит к началу сбора статистики временных затрат. Однако при использовании параметра `on` будут учитываться только обращения к подфункциям и внешним программам или функциям, за исключением встроенных. В нашем примере встроенной функцией является `chol`. Для получения дополнительной информации о работе всех вызываемых функций следует использовать еще опцию `-detail` со значением `builtin` (по умолчанию используется `mex`):

```
>> profile on -detail builtin
```

После запуска профайлера следует выполнять программы, операторы или вызывать функции, время работы которых подлежит определению:

```
>> profile on -detail builtin
>> prof_sp
>> profile off
```

Команда `profile off` приостанавливает сбор информации. Команда `profile resume` возобновляет работу профайлера с сохранением ранее накопленной статистики, а команда `profile clear` возобновляет работу профайлера без сохранения статистики. Отметим, что команда повторной активизации профайлера приводит к очистке статистики. Приведенные команды целесообразно использовать в своих программах для определения эффективности работы ее отдельных блоков.

Вывод отчета, изображенного на рис. 15.7, в браузер осуществляется командой `profsave` (либо командой `profile viewer` в окно профайлера). Таблица содержит информацию о времени работы каждой из функций, выполняемых после инициализации профайлера и до создания отчета. Обратите внимание, что команда `profile` с опцией `-detail builtin` выводит сведения обо всех встроенных функциях, которые были вызваны в процессе работы про-

граммы `prof_sp`. Этим обстоятельством можно пользоваться для получения набора встроенных функций, используемых в алгоритмах различных файл-функций MATLAB.

В столбце **Total Time** записано общее время, а в **Self Time** — время работы операторов функции без учета временных затрат на вызов из нее других встроенных функций. Детальную информацию о каждой функции можно получить, перейдя по ссылке с именем функции.

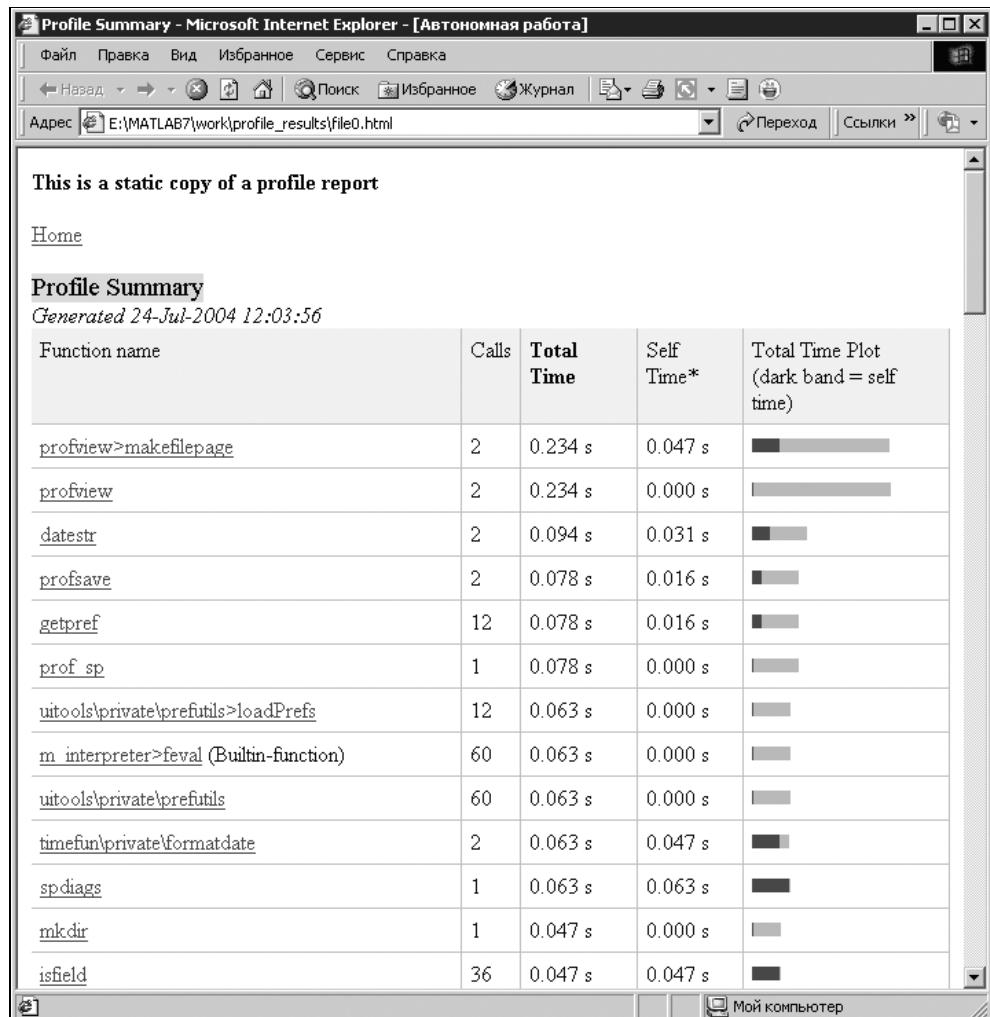


Рис. 15.7. Суммарный отчет о временных затратах

Отчет можно получить в виде совокупности файлов формата HTML в заданном подкаталоге (например, 'html\_report') текущего рабочего каталога MATLAB, создаваемом командой `profsave(stat_data, 'html_report')`. Предварительно структура с именем `stat_data` должна быть создана при помощи `stat_data = profile('info')`.

## 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  $Ax = b^{(1)}, \dots, Ax = b^{(k)}$  векторы правых частей записываются в столбцы некоторой вспомогательной матрицы  $B$ . В общем случае находится решение системы

$$AX = B,$$

где  $B = \begin{bmatrix} b^{(1)} & \dots & b^{(k)} \end{bmatrix}$ .

Каждый столбец матрицы  $X$  содержит решение соответствующей системы линейных алгебраических уравнений.

Операция  $A \backslash B$  реализует решение системы уравнений с разреженной матрицей  $A$  с использованием специальных алгоритмов, учитывающих структуру матрицы (применение обратной косой черты для решения систем общего вида было описано в главе 6).

В приложениях 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 ме-

тода предобусловленных сопряженных градиентов. Предобусловливатель строится на основе неполного разложения Холецкого исходной матрицы (см. разд. **Mathematics: Sparse Matrices: Simultaneous Linear Equations**).

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 `rcg` вместо предобусловливателя. Гибкость интерфейса `rcg` предоставляет вычислителю широкие возможности для проведения исследований с целью написания эффективных солверов для определенных классов задач.

Частичная алгебраическая проблема собственных значений — простая  $Ax = \lambda x$  или обобщенная  $Ax = \lambda Bx$  — 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:

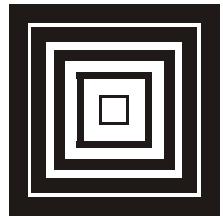
```
lambda = eigs(A, k, 0)
```

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 алгебраической проблемы собственных значений (см. разд. *"Собственные числа и векторы матрицы, функции матриц"* главы 6).

При решении обобщенной задачи 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 16

## Оптимизация

В состав MATLAB входит Optimization Toolbox, предназначенный для решения линейных и нелинейных оптимизационных задач. Функции этого Toolbox реализуют основные алгоритмы оптимизации, причем понимание алгоритма позволяет настроить выбранную функцию на эффективное решение поставленной задачи, что продемонстрировано на примере системы нелинейных уравнений. Optimization Toolbox не имеет приложений с графическим интерфейсом. Последний раздел данной главы содержит пример приложения, облегчающего доступ к нужной функции Toolbox и управление вычислительным процессом.

### Optimization Toolbox

При чтении главы 6 вы использовали ряд функций Optimization Toolbox для решения уравнений (`fzero`), минимизации функции одной переменной на отрезке (`fminbnd`) и минимизации функций нескольких переменных без ограничений на независимые переменные (`fminsearch`). Данный раздел посвящен более сложным задачам с ограничениями, которые могут быть решены с использованием вычислительных функций Optimization Toolbox. Интерфейс этих функций достаточно гибкий, все они допускают обращение с переменным числом входных и выходных аргументов в зависимости от данных задачи и искомых величин. Кроме того, большинство функций Toolbox позволяют исследование задач, зависящих от одного или нескольких параметров.

### Линейное и нелинейное программирование

#### Линейное программирование

Задача линейного программирования состоит в нахождении вектора  $x$ , который минимизирует целевую линейную функцию

$$f^T x, \quad (16.1)$$

где  $f$  — вектор коэффициентов, и удовлетворяет заданным линейным ограничениям: неравенствам

$$Ax \leq b \quad (16.2)$$

и равенствам

$$A_{eq}x = b_{eq}. \quad (16.3)$$

Кроме того, могут быть поставлены двусторонние покомпонентные ограничения в векторной форме

$$lb \leq x \leq ub. \quad (16.4)$$

В задачах оптимизации могут быть заданы не все типы ограничений, например, ограничения-равенства могут отсутствовать.

Для решения задач линейного программирования предназначена функция `linprog`. Первым аргументом `linprog` всегда является вектор  $f$ , далее задаются матрица  $A$  и вектор  $b$ . При наличии ограничений в виде равенств дополнительными аргументами могут быть  $A_{eq}$  и  $b_{eq}$ , 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. 16.1). 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 16.1. 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  $A$ , векторы  $b$  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.

$$f^T x = 44 \cdot x_1 + 35 \cdot x_2 + 100 \cdot x_3,$$

$$A = \begin{bmatrix} -4 & -6 & -15 \\ -2 & -2 & 0 \\ -5 & -3 & -4 \\ -7 & -3 & -12 \end{bmatrix}, \quad b = \begin{bmatrix} -250 \\ -60 \\ -100 \\ -220 \end{bmatrix}, \quad lb = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Для решения задачи составьте файл-программу `ration`. При вызове `linprog` вместо неиспользуемых аргументов (нет ограничений в виде равенств) задайте пустые массивы, обозначаемые в MATLAB квадратными скобками. Верхнее ограничение вида  $x \leq ub$  отсутствует, а функция `linprog` поддерживает обращение с переменным числом входных аргументов, поэтому седьмой входной аргумент не нужен.

Листинг 16.1 содержит операторы файл-программы `ration`.

#### Листинг 16.1. 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

$$\frac{1}{2} x^T H x + f^T x, \quad (16.5)$$

а ограничения в общем случае совпадают с ограничениями (16.2—16.4) в задаче линейного программирования. Для решения задач квадратичного программирования предназначена функция `quadprog`. 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 отсутствующих ограничений задаются пустые массивы.

Проиллюстрируем использование функции `quadprog` на примере задачи Марковица об определении состава инвестиционного портфеля рискованных ценных бумаг. Инвестор предполагает вложить свободные денежные средства в рыночные активы (ценные бумаги, акции) с целью получения дохода в будущем периоде. Для уменьшения рисков он выбрал для вложения четыре различных акции, которые обозначим  $A_1, A_2, A_3, A_4$ . Перед инвестором стоит задача определить, какую часть своих средств вложить в каждый актив так, чтобы получить желаемую доходность портфеля с наименьшим риском. Портфель определяется вектором долей от суммы инвестиций для покупки акций:

$$x = (x_1, x_2, x_3, x_4)^T, \quad x_1 + x_2 + x_3 + x_4 = 1, \quad x_i \geq 0 \quad (i=1, 2, 3, 4).$$

Риск оценивается как величина среднеквадратического отклонения ожидаемой доходности. Будем считать, что инвестор каким-либо способом произвел оценку ожидаемой доходности для каждой ценной бумаги, т. е. построил вектор доходностей  $y = (y_1, y_2, y_3, y_4)^T$  и вычислил матрицу ковариации  $V$ .

Если вектор  $y$  и матрица  $V$  известны, то доходность портфеля и его дисперсия вычисляются по формулам:

$$Y_p = y^T x = x^T y; \quad \sigma_p^2 = x^T V x.$$

Формальная постановка задачи Марковица приводит к задаче квадратичного программирования: найти минимум функции

$$J(x) = x^T V x \quad (16.6)$$

при ограничениях:

$$y^T x = a, \quad (16.7)$$

$$e^T x = 1, \quad e^T = (1, 1, 1, 1) \quad (16.8)$$

$$x \geq 0. \quad (16.9)$$

Величина желаемой доходности портфеля  $a$  должна быть не меньше минимальной и не больше максимальной доходности выбранных для инвестирования ценных бумаг. Неравенство (16.9) — одностороннее покомпонентное ограничение типа (16.4). Ограничения (16.7) и (16.8) — это ограничения вида (16.3), поэтому их следует объединить в одно, построив матрицу

$$A_{eq} = \begin{pmatrix} y^T \\ e^T \end{pmatrix}$$

и вектор

$$b_{eq} = \begin{pmatrix} a \\ 1 \end{pmatrix}.$$

Пусть заданы следующие значения для матрицы  $V$ , вектора  $y$  и желаемой доходности портфеля  $a$ :

$$V = \begin{bmatrix} 102.0 & 27.1 & -52.3 & 66.5 \\ 27.1 & 148.8 & 42.1 & -66.4 \\ -52.3 & 42.1 & 246.5 & 56.9 \\ 66.5 & -66.4 & 56.9 & 272.3 \end{bmatrix}; \quad y = \begin{bmatrix} 11.3 \\ 13.2 \\ 16.1 \\ 17.4 \end{bmatrix}; \quad a = 15.$$

Тогда в обозначениях, принятых для описания функции quadprog, построим матрицы и вектора, связанные с ограничениями:

$$A_{eq} = \begin{bmatrix} 11.3 & 13.2 & 16.1 & 17.4 \\ 1 & 1 & 1 & 1 \end{bmatrix}; \quad b_{eq} = \begin{bmatrix} 15 \\ 1 \end{bmatrix}; \quad lb = [0 \ 0 \ 0 \ 0]^T.$$

Для решения задачи о формировании портфеля с фиксированной доходностью составьте файл-программу `risk_asset`. Возможный вариант исходного текста файл-программы `risk_asset` приведен в листинге 16.2.

### Листинг 16.2. Файл-программа `risk_asset`

```
% Задание матрицы коэффициентов целевой функции
V = [102.0 27.1 -52.3 66.5;
 27.1 148.8 42.1 -66.4;
 -52.3 42.1 246.5 56.9;
 66.5 -66.4 56.9 272.3];

% Задание ограничений типа равенств
Aeq = [11.3 13.2 16.1 17.4;
 1 1 1 1];

beq = [15; 1];

% Задание ограничений снизу на переменные
lb = [0; 0; 0; 0];

% Решение и вывод результата в командное окно
x = quadprog(V, [], [], [], Aeq, beq, lb)
```

Выполнение файла-программы `risk_asset` даст ответ о распределении денежных средств в выбранные для вложений активы:

```
>> risk_asset
Warning: Large-scale method does not currently solve this problem
formulation, switching to medium-scale method.
> In C:\MATLAB65\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.

x =
0.0626
0.4359
0.1439
0.3575
```

При этом в командное окно вывелоось сообщение о применении Medium-scale алгоритма вместо Large-scale, принятого по умолчанию (см. разд. "Параметры оптимизации" данной главы).


**Примечание**


Обобщение приведенного примера, интерпретация результатов и специальные средства пакета MATLAB для решения задач такого класса приведены в главе 18.

## Нелинейное программирование

Optimization Toolbox позволяет решать ряд оптимизационных задач, в которых минимизируемая функция нелинейна, и к линейным ограничениям добавляются нелинейные. Общая постановка задачи нелинейного программирования такова: требуется разыскать

$$\min f(x) \quad (16.10)$$

среди всех векторов  $x$ , удовлетворяющих системе линейных ограничений (16.2—16.4) и дополнительных неравенств и равенств

$$c(x) \leq 0; \quad c_{eq}(x) = 0. \quad (16.11)$$

Решение поставленной задачи производится при помощи функции `fmincon`. Основное отличие интерфейса `fmincon` от `linprog` и `quadprog` состоит в том, что нелинейные ограничения  $c(x) \leq 0$  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 или количество вызовов исследуемой функции в процессе решения, меньше нуля — решение не найдено. Первый входной аргумент `fun` является указателем на файл-функцию (или ее именем), вычисляющую минимизируемую функцию  $f(x)$ , которая может зависеть от нескольких параметров. Значения параметров, в случае их наличия, передаются в последних аргументах `P1, P2, ...` начиная с 11-ой позиции в списке входных аргументов.

Исследование математических функций, зависящих от некоторых параметров, описано в разд. "Дополнительные возможности исследования функций" 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 нулевыми векторами и единичными матрицами.

гументов `fmincon` квадратными скобками (пустым массивом). Начальное приближение к решению указывается в `x0`. Список входных аргументов `fmincon` содержит управляющую структуру `options`, предназначенную для задания опций вычислительных алгоритмов. Для большинства функций Toolbox Optimization набор опций существенно больше, чем для рассмотренных в главе 6 функций `fzero`, `fminbnd` и `fminsearch`, поэтому изучению данного вопроса посвящено несколько разделов этой главы (см. разд. "Параметры оптимизации" и "Решение большой системы нелинейных уравнений" данной главы).

Нелинейные ограничения (16.11) (неравенства и равенства) программируются в файл-функции, указатель на нее (или ее имя) указывается в аргументе `nonlcon`. Входным аргументом `nonlcon` является вектор `x`, соответствующий искомому вектору `x`, а двумя выходными аргументами — векторы `c` и `ceq` левых частей нелинейных ограничений `c` и `ceq`.

Последние *идущие подряд* входные аргументы функции `fmincon` могут быть опущены, если они не используются. Например, при отсутствии нелинейных ограничений и параметров применяется следующий вызов `fmincon`:

```
x = fmincon(fun, x0, A, b, Aeq, beq, lb, ub)
```

Поскольку в данном случае управляющая структура `options` не задана, то вычисления будут производиться с принятыми по умолчанию опциями, узнать о которых можно при помощи функции `optimset`. Для этого достаточно вызвать ее от строки с именем вычислительной функции Optimiza-tion Toolbox, например, выполнение команды

```
>> optimset('fmincon')
```

приводит к выводу информации о всех настройках алгоритма функции `fmincon`, в том числе и точности  $10^{-6}$  по аргументу и функции.

Найдите решение следующей простой задачи (очевидно, что решение — нулевой вектор):

$$\min 3(x_1)^2 + 2(x_2)^2; \quad (x_1)^2 + (x_2)^2 \leq 1.$$

Обратите внимание, что имеется только одно нелинейное ограничение в виде неравенства, которое следует преобразовать к виду  $c(x) \leq 0$ , перенеся единицу в левую часть. Написание файл-функции, вычисляющей  $3(x_1)^2 + 2(x_2)^2$ , не представляет труда (листинг 16.3). При программировании нелинейного ограничения учтите, что соответствующая файл-функция возвращает два вектора левых частей нелинейных ограничений (листинг 16.4). В рассматриваемом примере первый вектор состоит только из одной компоненты, а второй должен быть пустым, поскольку нет ограничений в виде равенств.

**Листинг 16.3. Минимизируемая функция**

```
function f = myfun(x)
% Вычисление минимизируемой функции
f = 3*x(1)^2 + 2*x(2)^2;
```

**Листинг 16.4. 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 Optimization Toolbox. 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. Область допустимых значений независимых переменных может задаваться как линейными, так и нелинейными ограничениями, в общем случае они совпадают с ограничениями в задаче нелинейного программирования, описанной выше.

### Задача о достижении границы

В задаче о достижении границы задана вектор-функция  $F(x)$  и два вектора  $w$  и  $g$ . Требуется найти  $x$  из области допустимых значений, для которого величина  $\gamma$  будет минимальной при выполнении условий  $F_i(x) - w_i \gamma \leq g_i$  для всех компонент  $F(x)$ ,  $w$  и  $g$ .

Поставленная задача решается при помощи функции `fgoalattain`, обращение к которой в достаточно общем случае имеет вид:

```
x = fgoalattain(fun, x0, g, w, A, b, Aeq, beq, lb, ub, nonlcon, options, P1, P2, ...)
```

Входной аргумент `fun` является указателем на файл-функцию (или ее именем), вычисляющей вектор-функцию  $F(x)$ . Аргументы `g` и `w` — векторы, длина которых совпадает с числом значений вектор-функции. Использование остальных аргументов аналогично `fmincon`. Интерфейс функции `fgoalattain`, так же как и других функций Optimization Toolbox, допускает вызов с переменным числом входных и выходных аргументов в зависимости от условий, определяющих область допустимых значений, и требуемого результата. Так, например, в случае нелинейных ограничений и необходимости вычислить не только  $x$ , но  $F(x)$  и  $\gamma$ , следует указать три выходных аргумента, а неиспользуемые входные сделать пустыми массивами:

```
[x, F, gamma] = fgoalattain(fun, x0, g, w, [], [], [], [], [], nonlcon)
```

Четвертый дополнительный выходной аргумент функции `fgoalattain` служит для получения информации о вычислительном процессе. Положительное его значение свидетельствует об успешном завершении вычислений, ноль — вычисления остановлены из-за достижения максимального числа итераций или превышения максимально возможного числа вызовов иссле-

дуемой функции, а отрицательное значение сигнализирует о том, что решение не найдено.

В справочной системе MATLAB по Optimization Toolbox приведен пример конструирования регулятора с обратной связью с привлечением функции fgoalattain (см разд. **Optimization Toolbox: Tutorial: Multiobjective Examples**).

## Минимизация функции с полубесконечными ограничениями

Постановка задачи заключается в поиске минимума функции нескольких переменных (16.10) в области, задаваемой в общем случае линейными ограничениями (16.2—16.4), нелинейными (16.11) и полубесконечными ограничениями вида

$$K_i(x, w_i) \leq 0, \quad i=1, \dots, n, \quad (16.12)$$

где  $w_i$  — векторы дополнительных переменных размерности не более двух, компоненты которых изменяются в заданном интервале.

Для решения этой задачи предназначена функция fseminf со следующим интерфейсом:

```
x =
fseminf(fun, x0, ntheta, seminfcon, A, b, Aeq, beq, lb, ub, options,
P1, P2, ...)
```

причем могут быть указаны дополнительные выходные аргументы. Во второй из них запишется значение функции в точке минимума, а в третий — информация о вычислительном процессе. Смысл значений третьего выходного аргумента тот же самый, что и для функции fmincon, рассмотренной выше.

Входные аргументы функции fseminf: fun, x0, A, b, Aeq, beq, lb, ub предназначены для указания минимизируемой функции, начального приближения, матриц и векторов, определяющих линейные ограничения типа неравенств и равенств (16.2—16.4). Обсудим назначение параметров, непосредственно связанных с нелинейными ограничениями (16.11) и полубесконечными (16.12). Количество ограничений вида (16.12) задается в ntheta, а все нелинейные ограничения, включая полубесконечные, программируются в файл-функции, указатель на которую (или ее имя) передается в seminfcon. Эта файл-функция должна иметь специальный интерфейс и структуру (листинг 16.5).

### Листинг 16.5. Общий вид файл-функции для полубесконечных ограничений

```
function [c, ceq, K1, K2, ..., Kntheta, S] = sem_con(x, S)
% Проверка, был ли инициализирован массив шагов S
```

```
% для дополнительных переменных
if isnan(S(1, 1))
 % Это первый вызов sem_con, следует инициализировать массив S
 S(1, 1) = ...; S(1, 2) = ...;
 ...
 S(ntheta ,1) = ...; S(ntheta, 2) = ...;
end
% Генерация сетки значений дополнительных переменных
% Если дополнительная переменная является вектором с двумя компонентами,
% то следует использовать meshgrid
w1 = ...;
...
wntheta = ...;
% Вычисление левых частей полубесконечных ограничений (16.12)
% в узлах сетки
K1 = ...;
...
Kntheta = ...;
% Вычисление левых частей нелинейных ограничений
% (равенств и неравенств) типа (16.11)
c = ...
ceq = ...
```

Для заданного  $x$  файл-функция `sem_con` должна возвращать значения вектор-функций  $c(x)$  и  $c_{eq}(x)$ , входящих в левые части нелинейных ограничений (16.11). При отсутствии ограничений этого типа следует присвоить `c` и `ceq` пустые массивы в теле файл-функции. Список выходных аргументов `sem_con` включает в себя также значения левых частей  $K_i(x, w_i)$  полубесконечных ограничений (16.12), вычисленных для некоторого набора значений каждой из переменных  $w_i$ . Эти значения выбираются на сетке с постоянным шагом, причем шаг для  $w_i$  записан в  $i$ -ой строке массива `s`. Если  $w_i$  — скаляр, то выходной аргумент, соответствующий  $K_i(x, w_i)$ , будет вектором и шаг определяется значением `s(i, 1)`, а элемент `s(i, 2)` не используется. В случае, когда аргумент  $w_i$  функции  $K_i(x, w_i)$  является вектором длины 2, для  $w_i$  генерируется сетка значений с шагами `s(i, 1)` и `s(i, 2)` по каждой из компонент при помощи функции `meshgrid`.

В ходе вычислений функция `fseminf` обращается к `sem_con` с рекомендуемыми значениями шагов, но при первом вызове `sem_con` массив `S` должен быть инициализирован. Первый вызов `sem_con` легко отличить от последующих, поскольку входной аргумент `S` содержит `NaN` (не числа). Поэтому в начале файл-функции `sem_con` размещен условный оператор `if`, который позволяет инициализировать массив шагов.

Рассмотрим на простом примере минимизацию нелинейной функции с полубесконечными ограничениями. Пусть требуется найти минимум функции:

$$f_1(x_1, x_2) = x_1^2 + 2x_2^2$$

при отсутствии линейных и нелинейных ограничений, но с заданным полубесконечным ограничением вида:

$$K_1(x_1, x_2, w_1) = \frac{\cos(x_1 w_1)}{x_2 w_1} - 0.1 \leq 0.$$

Для решения поставленной задачи требуется написать файл-функции для вычисления  $f(x_1, x_2)$  и  $K_1(x_1, x_2, w_1)$ . Целевая функция программируется просто (листинг 16.6). При создании файл-функции для полубесконечного ограничения следуйте приведенным выше соглашениям относительно ее интерфейса и алгоритма. В нашем примере всего одно полубесконечное ограничение, причем дополнительная независимая переменная  $w_1$  является скаляром. В случае возникновения затруднений обратитесь к листингу 16.7.

#### Листинг 16.6. Минимизируемая функция решения задачи с полубесконечными ограничениями

```
function f = myfuns(x)
% целевая функция
f = x(1)^2 + 2*x(2)^2;
```

#### Листинг 16.7. Функция вычисления ограничений решения задачи с полубесконечными ограничениями

```
function [c, ceq, K1, S] = sem_con(X, S)
if isnan(S(1, 1))
 % Инициализация шага сетки 0.2 для скалярной переменной w1
 % (вторая компонента не будет использоваться)
 S = [0.2 0];
end
```

```
% Генерация сетки на отрезке [0, 100] для полубесконечных ограничений
w1 = 1:S(1, 1):100;
% Вычисление полубесконечного ограничения в узлах сетки
K1 = cos(w1*X(1))./(w1*X(2)) - 0.1;
% Нелинейные ограничения отсутствуют, задаем пустой массив
c = [];
ceq = [];
```

Для решения задачи осталось задать начальное приближение

```
>> x0 = [0.5; 0.2];
```

и вызвать функцию `fseminf`, установив число полубесконечных ограничений в 1:

```
>> [x, fval] = fseminf(@myfun, x0, 1, @sem_con)
x =
 1.3512
 2.1785
fval =
 11.3175
```

## Минимаксная задача

Постановка минимаксной задачи заключается в нахождении

$$\min_x \max_{i=1, 2, \dots, n} \{f_i(x)\}$$

для нелинейной вектор-функции  $f(x) = (f_1(x), f_2(x), \dots, f_n(x))$  при ограничениях (16.2—16.4) и (16.11) тех же самых, что и в рассмотренной выше задаче нелинейного программирования. Для решения данной задачи предназначена функция `fminimax`, применение которой в достаточно общем случае выглядит следующим образом:

```
x = fminimax(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon, options, P1, P2, ...)
```

Первый входной аргумент `fminimax` является файл-функцией, возвращающей вектор значений  $\{F_i(x)\}$ . Смысл остальных входных аргументов такой же, как и для `fmincon`, применение которой обсуждалось выше. Интерфейс функции `fminimax`, как и большинства функций Optimization Toolbox, допускает переменное число входных и выходных аргументов, причем в качестве неиспользуемых входных аргументов указывается пустой массив. Во

втором дополнительном выходном аргументе fminimax возвращаются значения вектор-функции  $f(x)$ , в третьем — максимальное из них, а четвертый выходной аргумент содержит информацию о причине останова вычислений. Смысл его значений тот же, что и для функции fmincon.

Рассмотрим модельную задачу для демонстрации обращения к функции fminimax, в которой надо найти независимые переменные  $x_1$  и  $x_2$  из условия:

$$\min_{\{x_1, x_2\}} \max_{\{f_i\}} \{f_1(x_1, x_2), f_2(x_1, x_2), f_3(x_1, x_2)\},$$

где

$$f_1(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 2)^2 - 2, \quad f_2(x_1, x_2) = x_1^{1.3} + 2x_2^{0.9} - 1,$$

$$f_3(x_1, x_2) = 1.7x_1 + x_2 - 3.5.$$

при линейных ограничениях

$$x_1 \geq 0, \quad x_2 \geq 0, \quad 3x_1 + 2x_2 \leq 2.2,$$

и нелинейном ограничении

$$-e^{x_1} + e^{x_2} = 1.5.$$

Для решения задачи напишите файл-функцию для вычисления минимизируемой вектор-функции, файл-функцию для задания нелинейных ограничений и файл-программу, реализующую обращение к fminimax. В листингах 16.8—16.10 приведены тексты программ.

#### Листинг 16.8. Минимизируемая функция решения задачи о минимаксе

```
function f = vectfun3(x)
% Вычисление минимизируемой вектор-функции

f(1) = (x(1) - 1)^2 + (x(2) - 2)^2 - 2.1;
f(2) = x(1)^1.3 + 2*x(2)^0.9 - 1;
f(3) = 1.7*x(1) + x(2) - 0.25;
```

#### Листинг 16.9. Функция вычисления ограничений в задаче о минимаксе

```
function [c, ceq] = nonlinear(x)
% Задание нелинейных ограничений

% ограничений-неравенств нет
c = [];
```

```
% ограничения-равенства
ceq(1) = -exp(x(1)) + exp(x(2)) - 0.5;
```

**Листинг 16.10. Файл-функция test\_mm решения задачи о минимаксе**

```
% начальное приближение
x0 = [1; 0];
% задание ограничений-равенств
A = [3 2];
b = 2.2;
% Задание ограничений снизу на переменные
lb = [0; 0];
[x, fval] = fminimax(@vectfun3, x0, A, b, [], [], lb, [], @nonlinear)
```

При вызове файл-функции test\_mm получается результат:

```
x =
0.2048
0.5466
fval =
0.6448 0.2885 0.6448
```

## Решение нелинейных уравнений

Optimization Toolbox позволяет решить системы нелинейных уравнений, общий вид которых задается при помощи нелинейной вектор-функции:  $F(x)=0$ , а покомпонентная запись имеет вид

$$\begin{cases} F_1(x_1, x_2, \dots, x_n) = 0; \\ F_2(x_1, x_2, \dots, x_n) = 0; \\ \dots \\ F_m(x_1, x_2, \dots, x_n) = 0. \end{cases}$$

Первым аргументом fsolve является указатель на файл-функцию (или ее имя), вычисляющую  $F(x)$ . Данная файл-функция принимает на входе вектор аргументов и возвращает вектор значений  $F(x)$ . Начальное приближение указывается во втором аргументе fsolve.

В достаточно общем случае вызов `fsolve` выглядит следующим образом:

```
x = fsolve(fun, x0, options, P1, P2, ...)
```

где `options` — управляющая структура, а необязательные аргументы `P1, P2, ...` — это значения параметров, от которых может зависеть левая часть системы уравнений. Если в списке входных аргументов `fsolve` всего два входных аргумента, то в командное окно выводится предупреждение, связанное с тем, что в версии 2.0 Optimization Toolbox алгоритм `fsolve` был модифицирован. Для подавления этого предупреждения следует вызывать `fsolve` с управляющей структурой, которую можно сформировать, например, со всеми принятыми по умолчанию настройками при помощи `optimset`:

```
x = fsolve(fun, x0, optimset('fsolve'))
```

Функция `fsolve` может быть вызвана с несколькими выходными аргументами. Второй выходной аргумент содержит значения вектор-функции для найденного решения, а третий — сведения о работе алгоритма. Отрицательное значение третьего аргумента говорит о том, что решение не найдено из-за расходности вычислительного процесса. Ноль означает досрочное прерывание вычислений при достижении максимально допустимого числа итераций или обращений к вектор-функции системы. Нахождение решения с заданной точностью подтверждается положительным значением третьего выходного аргумента `fsolve`. Решите систему из двух нелинейных уравнений с двумя неизвестными

$$\begin{cases} x_1(2 - x_2) - \cos x_1 \cdot e^{x_2} = 0; \\ 2 + x_1 - x_2 - \cos x_1 - e^{x_2} = 0. \end{cases} \quad (6.13)$$

Файл-функция, соответствующая левой части системы, программируется просто (листинг 16.11).

#### Листинг 16.11. Файл-функция, вычисляющая левую часть системы уравнений (6.13)

```
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], optimset('fsolve'))
```

```
Optimization terminated successfully:
```

```
Relative function value changing by less than OPTIONS.TolFun
```

```
x =
 0.7391 0.4429
f =
 1.0e-011 *
 -0.4702 -0.6404
```

Мы рассмотрели пример, в котором число неизвестных совпадает с числом уравнений системы. Алгоритмы, заложенные в функцию `fsolve`, допускают несовпадение числа неизвестных и уравнений. В качестве упражнения решите системы:

$$\begin{cases} x_1^2 - 2x_2^2 = -1; \\ -3x_1^2 + x_2^2 = -2; \\ x_1^3 + x_2^3 = 2; \end{cases} \quad \begin{cases} x_1^2 - 2x_2^2 + x_3 = -1; \\ -3x_1^2 + x_2^2 + x_3 = -2. \end{cases}$$

При исследовании первой системы задавайте различные начальные приближения и убедитесь, что возможно найти приближенное решение, достаточно близкое к точному  $x_1 = x_2 = 1$ . Для второй системы выбор начального приближения определяет, для какого точного решения разыскивается приближенное — сравните, например, начальные приближения  $[1\ 1\ 1]$  и  $[0\ 1\ 1]$ .

Алгоритмы, реализованные в функции `fsolve`, основаны на минимизации некоторых критериев, связанных с исходной системой нелинейных уравнений, в частности, суммы квадратов компонент вектор-функции. Решением считается та точка, в которой значение критерия мало, что не гарантирует существования корней системы в окрестности найденной точки. В этом легко убедиться на простом примере — примените функцию `fsolve` для системы из одного уравнения  $x^2 + 10^{-6} = 0$ , не имеющего вещественных корней. Используйте для сравнения `fzero` с некоторым начальным приближением к корню. Функция `fzero` пытается отделить интервал, на границах которого непрерывная функция  $x^2 + 10^{-6}$  имеет разные знаки для гарантии существования корня. Разумеется, такого интервала не существует и работа `fzero` прерывается.

Одним из алгоритмов функции `fsolve` является метод наименьших квадратов, который также используется для решения ряда оптимизационных задач, в частности, подбора параметров.

## 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 нулю. По существу, при помощи метода наименьших квадратов решается задача 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 возвращаются норма невязки (`resnorm`), вектор невязки (`flag`) и информация о завершении процесса вычислений (`flag`). Положительное значение переменной `flag` подтверждает сходимость итерационного процесса к решению, а нулевое значение предупреждает о превышении максимально допустимого числа итераций или вычислений квадрата нормы невязки, установленных по умолчанию.

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

$$\min_x \|Cx - d\|^2$$

с линейными ограничениями на решение (16.2—16.4).

## 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 Optimization Toolbox основано на методе наименьших квадратов, который в данном случае состоит в нахождении минимума выражения

$$\frac{1}{2} \sum_{i=1}^n [F(a_1, a_2, a_3, a_4, xdat_i) - ydat_i]^2$$

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 и в самом простом случае имеет вид:

```
x = lsqcurvefit(fun, a0, xdat, ydat)
```

где `fun` — указатель на файл-функцию (или ее имя), которая вычисляет  $F(a_1, a_2, a_3, a_4, x)$ . Список ее входных аргументов должен содержать массив для передачи значений параметров в файл-функцию и независимую переменную `x`.

Дополнительно могут быть поставлены ограничения на параметры, которые в векторной записи имеют вид  $lb \leq a \leq ub$ . 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  $F = a_1 e^{a_2 x} + a_3 \sin a_4 x$ , а 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 16.12). Параметры в `fitfun` передаются через вектор `a`.

#### Листинг 16.12. Файл-функция, зависящая от вектора параметров и аргумента

```
function y = fitfun(a, x)
y = a(1)*exp(a(2)*x) + 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 16.13, реализует вышеописанные действия, в результате получается график, изображенный на рис. 16.1. В командное окно выводится вектор найденных значений параметров:

```
a =
1.0959 1.1678 2.7003 3.7505
```

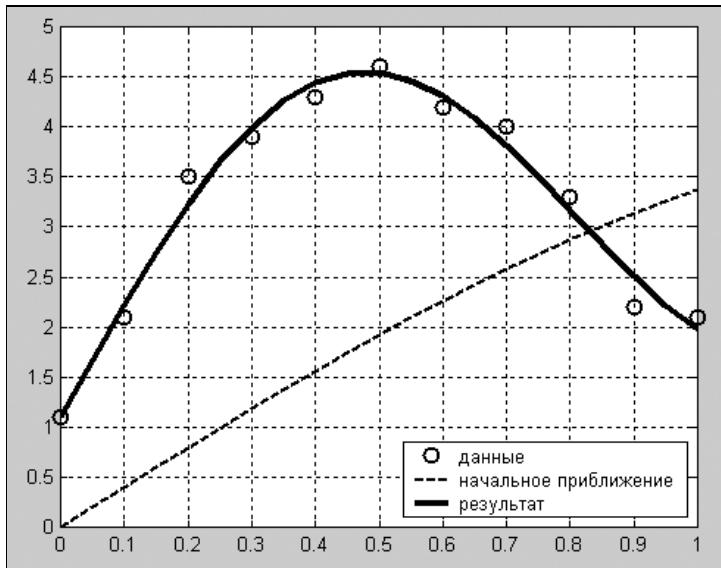


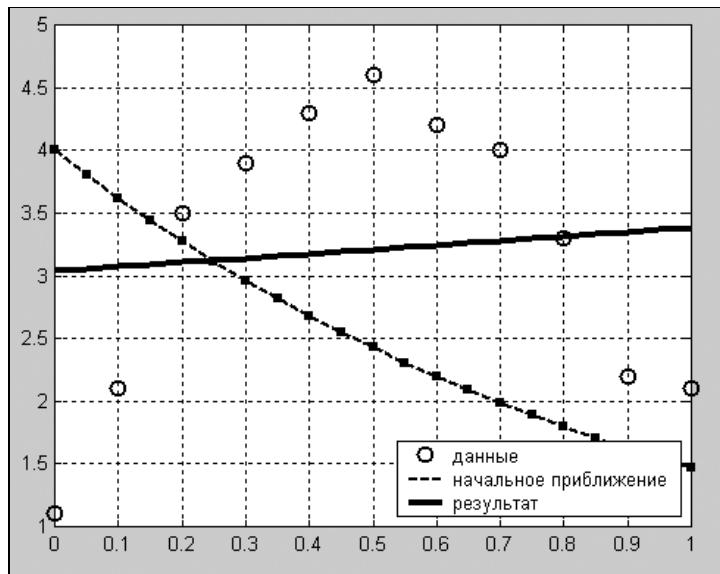
Рис. 16.1. Результат подбора параметров

### Листинг 16.13. 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  $a_0 = [4.0 \ -1.0 \ 0.0 \ 0.0]$ , то подбор параметров не приведет к хорошему результату (рис. 16.2).



**Рис. 16.2.** Результат подбора параметров при плохом начальном приближении

## Параметры оптимизации

Задачи оптимизации условно разделены в Optimization Toolbox 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 кото-

рых выходит за рамки данной книги. Впрочем, краткая информация об алгоритмах содержится в справочной системе MATLAB по Toolbox. Последняя версия документации в формате PDF доступна на сайте производителя MATLAB. 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 входного аргумента функции Optimization Toolbox, 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 структуру с информацией о текущих опциях вычислительного алгоритма данной функции. Обращение к `optimset` без входных аргументов позволяет отобразить в командном окне все возможные значения каждого из параметров. Подробная информация о назначении каждого параметра приведена в справочной системе MATLAB (см. разд. **Optimization Toolbox: Function Reference: Optimization Parameters**, где приведена таблица со всеми параметрами и описаны функции Toolbox).

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 уравнений не должна быть недоопределенной, т. е. число уравнений не может превосходить число неизвестных.

Такие параметры, как `JacobPattern`, `LevenbergMarquardt`, `MaxPCGIter`, `TolPCG`, `PrecondBandWidth`, соответствуют Large-scale алгоритмам. Получающиеся в процессе вычислений системы линейных уравнений решаются итерационным методом — методом предобусловленных сопряженных градиентов (PCG). По умолчанию выбирается диагональный предобусловливатель, т. е. ширина верхней полуленты равна нулю. В ряде случаев возможно добиться ускорения сходимости итерационного процесса за счет увеличения ширины ленты предобусловливателя. Параметр `PrecondBandWidth` служит для задания ширины верхней полуленты ленты в исходной матрице, на основе которой строится предобусловливатель. Максимальное число итераций в методе сопряженных градиентов определяется значением `MaxPCGIter`, а критерий останова — `TolPCG`.

Задание матрицы Якоби 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 матрица Якоби системы. Соответствующий пример разобран ниже. Вместо аналитического вычисления матрицы Якоби левой части системы нелинейных уравнений можно задать его шаблон, т. е. расположение ненулевых элементов, в `JacobPattern`, тогда в процессе вычислений только ненулевые элементы матрицы Якоби будут аппроксимироваться конечными разностями, что значительно ускорит вычисления.

## Примеры

Следующие разделы посвящены применению некоторых опций, управляющих вычислительным процессом, на примере решения большой системы

нелинейных уравнений. Разобрано создание приложения с графическим интерфейсом пользователя, облегчающее использование возможностей Optimization Toolbox.

## Решение системы нелинейных уравнений

Рассмотрим пример решения системы нелинейных уравнений, в котором задание аналитических выражений для элементов матрицы Якоби существенно уменьшает время вычислений. Необходимо решить приведенную ниже систему нелинейных уравнений  $F(x)=0$  для  $n=1000$ .

$$\begin{cases} 2x_1^2 - x_2 = 1; \\ -x_1 + 2x_2^2 - x_3 = 1; \\ \dots \\ -x_{n-1} + 2x_n^2 = 1. \end{cases} \quad (6.14)$$

Файл-функция `largesys` вычисляет значения компонент  $F_2(x), \dots, F_{n-1}(x)$  вектор-функции в цикле `for` (листинг 16.14).

### Листинг 16.14. Файл-функция, вычисляющая левую часть системы уравнений (6.14)

```
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, как мы увидим, ее решение может занять значительное время, если не принимать во внимание разреженную структуру исходной системы. В данном примере вычисление матрицы Якоби дает существенный выигрыш во времени. Матрица Якоби решаемой системы

$$J = \left( \frac{\partial F_i}{\partial x_j} \right)_{i,j=1, 2, \dots, n}$$

имеет достаточно простую структуру — она является сильно разреженной (трехдиагональной):

$$J = \begin{bmatrix} 4x_1 & -1 & 0 & 0 \\ -1 & 4x_2 & 0 & 0 \\ & \ddots & & \\ 0 & 0 & 4x_{n-1} & -1 \\ 0 & 0 & -1 & 4x_n \end{bmatrix}.$$

Напомним, что разреженные матрицы создаются при помощи функции `sparse` (работа с разреженными матрицами описана в главе 15).

Очевидно, что матрица Якоби представляется суммой  $J = D + Dl + Dl^T$  разреженных матриц  $D$  и  $Dl$ , где

$$D = \begin{bmatrix} 4x_1 & 0 & 0 & 0 \\ 0 & 4x_2 & 0 & 0 \\ & \ddots & & \\ 0 & 0 & 4x_{n-1} & 0 \\ 0 & 0 & 0 & 4x_n \end{bmatrix}; \quad Dl = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ & \ddots & & \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Дополните файл-функцию `largesys` нахождением разреженного представления для матрицы Якоби, которая вычисляется и возвращается во втором выходном аргументе, если файл-функция вызывается с двумя выходными аргументами. 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 `largesysj` приведен в листинге 16.15.

### Листинг 16.15. Файл-функция `largesysj`, возвращающая матрицу Якоби

```
function [F, J] = largesysj(x)
% Файл-функция для вычисления левой части системы нелинейных уравнений
% и матрицы Якоби

% Вычисление компонент вектор-функции F
n = length(x);
F = rand(n,1);
F(1) = 2*x(1)^2 - x(2) - 1;

% Вычисление компонент вектор-функции F
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 матрицы Якоби
if nargin > 1
 % Матрица Якоби 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 разреженной матрицы Якоби
 J = Diag + D1 + D1';
end

```

Решение системы нелинейных уравнений оформите в файл-программе (листинг 16.16), в которой задаются число переменных и вектор начального приближения и используется `fsolve` для поиска корней. Наша цель состоит в исследовании эффективности применения явных формул для элементов матрицы Якоби, поэтому сначала обратитесь к `fsolve` со значением '`'off'`' параметра `Jacobian` (оно установлено по умолчанию). Заметьте, что в этом случае `fsolve` будет вызывать файл-функцию `largesysj` с одним выходным аргументом и матрица Якоби вычисляться не будет. Затем модифицируйте управляющую структуру, установив параметр `Jacobian` в значение в '`'on'`', и снова примените `fsolve`. Для контроля за временем счета удобно задействовать встроенные функции `tic` и `toc`.

#### Листинг 16.16. Файл-программа для решения большой системы нелинейных уравнений

```

n = 1000; % число переменных
x0 = ones(1, n); % начальное приближение

% Решение системы без использования явных формул
% для элементов матрицы Якоби

```

```

options = optimset('Display', 'iter', 'Diagnostics', 'on');
tic
x = fsolve(@largesysj, x0, options);
toc
% Решение системы с использованием явных формул
% для элементов матрицы Якоби
options = optimset(options, 'Jacobian', 'on');
tic
x = fsolve(@largesysj, x0, options);
toc

```

Выполнение операторов листинга 16.16 свидетельствует об эффективности использования явных формул для вычисления элементов матрицы Якоби по сравнению с аппроксимацией их конечными разностями — время счета отличается примерно в 50 раз. Поскольку параметр `Diagnostics` имеет значение '`on`', то в командное окно сначала выводится информация о количестве неизвестных, способе вычисления частных производных и применяемом алгоритме (в данном случае — метод доверительной области). Работа `fsolve` сопровождается отображением сведений о каждом шаге вычислительного процесса (опция `Display` установлена в '`iter`'). Сведения представлены в виде таблицы:

| Iteration | Func-count | f(x)         | Norm of step | First-order optimality | Trust-region radius |
|-----------|------------|--------------|--------------|------------------------|---------------------|
| 1         | 1          | 998          |              | 3                      | 1                   |
| 2         | 2          | 871.767      | 1            | 2.38                   | 1                   |
| 3         | 3          | 567.03       | 2.5          | 2.13                   | 2.5                 |
| 4         | 4          | 36.9426      | 6.25         | 0.932                  | 6.25                |
| 5         | 5          | 0.0498932    | 1.87805      | 0.0273                 | 15.6                |
| 6         | 6          | 6.87172e-008 | 0.0643166    | 3e-005                 | 15.6                |
| 7         | 7          | 1.31898e-019 | 7.56621e-005 | 4e-011                 | 15.6                |

Столбик `Iteration` содержит номер итерации; `Func-count` — число вызовов функции; `f(x)` — сумма квадратов значений левых частей уравнений системы для текущего приближения; `Norm of step` — норма шага на текущей итерации; `First-order optimality` — бесконечная норма градиента, вычисленного для текущего приближения; `Trust-region radius` — радиус доверительной области.

Данный раздел описывает только решение системы нелинейных уравнений методом доверительных областей при помощи `fsolve`. В функции `fsolve` реализованы и другие алгоритмы решения, которые выбираются при по-

мощи опций NonlEqnAlgorithm, LargeScale, LineSearchType управляющей структуры. Перед решением задач в Optimization Toolbox полезно обратиться к информации о настройках алгоритмов выбранной функции. Эти сведения могут быть почерпнуты из справочной системы Toolbox. Функции и допустимые параметры для них описаны в разделе **Optimization Toolbox: Function Reference**, задание параметров — в **Optimization Toolbox: Function Reference: Optimization Parameters**, а сами алгоритмы — в **Optimization Toolbox: Standard Algorithms** и **Optimization Toolbox: Large-Scale Algorithms**.

Мы рекомендуем также изучить примеры решения оптимизационных задач, приведенные в справочной системе по Optimization Toolbox.

## Пример приложения с GUI

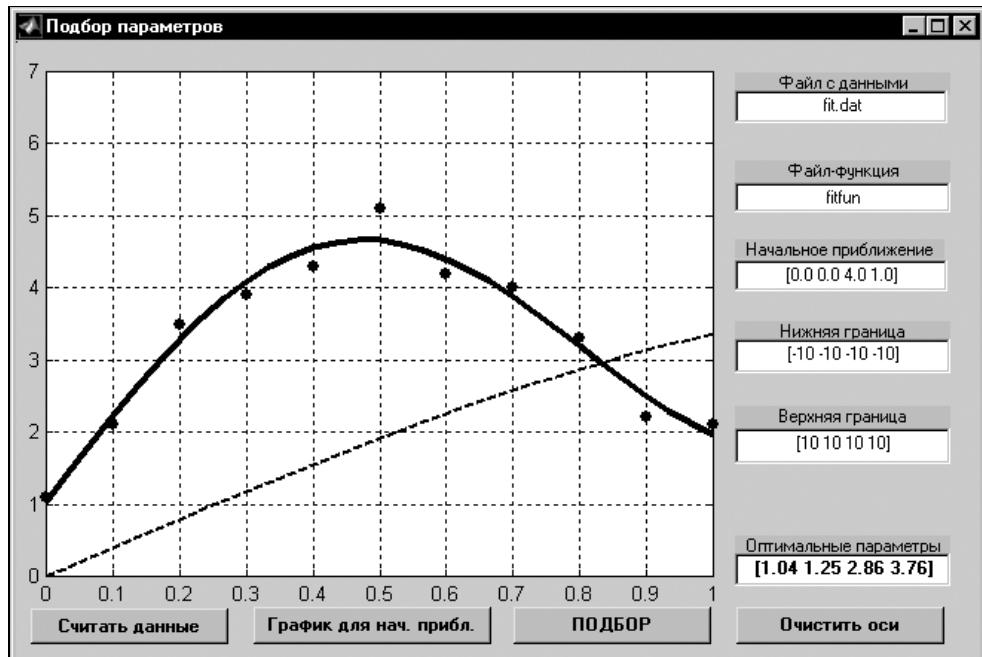
Optimization Toolbox не имеет приложений с графическим интерфейсом пользователя для доступа к функциям оптимизации. Решение оптимизационных задач значительно упростилось бы при наличии соответствующего приложения. Текущий раздел посвящен написанию приложения с графическим интерфейсом для решения задачи 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 16.13, для различных данных и функций не очень удобно — требуется изменять имя файла и файл-функции. При выборе начального приближения и задании допустимых интервалов для параметров также необходимо вносить изменения в файл-программу. Создание приложения с графическим интерфейсом позволит существенно экономить время при подобной многократной обработке набора 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. 16.3, 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 при-

ближения 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. 16.3.** Окно приложения **Подбор параметров**

Создание вышеописанного приложения **Подбор параметров** не представляет большого труда. Основные вопросы, связанные с программированием приложений с графическим интерфейсом пользователя, разобраны в части III книги. Подфункции обработки событий содержатся в листинге 16.17. Имена основных объектов, расположенных в окне приложения, помещены в табл. 16.2.

**Таблица 16.2.** Имена объектов приложения **Подбор параметров**

| Объект                               | Имя      |
|--------------------------------------|----------|
| Кнопка <b>Считать данные</b>         | LoadBtn  |
| Кнопка <b>График для нач. прибл.</b> | GuessBtn |

Таблица 16.2 (окончание)

| Объект                        | Имя         |
|-------------------------------|-------------|
| Кнопка ПОДБОР                 | FitBtn      |
| Кнопка Очистить оси           | ClearBtn    |
| Область Файл с данными        | DataFileEdt |
| Область Файл-функция          | FunEdt      |
| Область Начальное приближение | GuessEdt    |
| Область Нижняя граница        | LBEdt       |
| Область Верхняя граница       | UBEdt       |
| Область Оптимальные параметры | ParEdt      |

#### Листинг 16.17. Подфункции обработки событий

```

function LoadBtn_Callback(hObject, eventdata, handles)
 % 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);
 hold on
 % Очистка области ввода значений параметров "Оптимальные параметры"
 set(handles.ParEdt, 'String', '')
 end

```

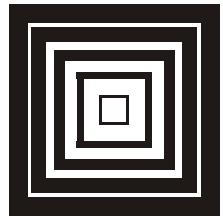
```
function GuessBtn_Callback(hObject, eventdata, handles)
 % Считывание вектора начального приближения для параметров из
 % области ввода "Начальное приближение"
 ParStr = ['par0=', get(handles.GuessEdt, 'String'), ';'];
 % Выполнение строки, запись начального приближения в вектор par0
 eval(ParStr)
 % 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', ';')
 y = feval(FunName, par0, x);
 % Отображение графика для начального приближения
 plot(x, y, '--b')

function FitBtn_Callback(hObject, eventdata, handles)
 % 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 областей 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 области 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 строку "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));
y = feval(FunName, par, x);
Hplot = plot(x, y, '-g');
set(Hplot(1), 'LineWidth', 2)

function ClearBtn_Callback(hObject, eventdata, handles)
% Очистка осей
cla;
```

Усовершенствуйте приложение, добавив элементы управления для задания параметров для управления вычислительным процессом: точности, максимально допустимого числа итераций и обращений к функции, выбора алгоритма. Перед вызовом `lsqcurvefit` следует выяснить выбор пользователя и сформировать подходящую структуру `options`. Схожие приложения с графическим интерфейсом могут быть созданы и для решения других задач оптимизации.



## Глава 17

# Символьные вычисления

В состав MATLAB входит Symbolic Math Toolbox, предназначенный для вычислений в символьном 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 Symbolic Math Toolbox 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.

## Символьные переменные и функции

Объектно-ориентированный подход, реализованный в MATLAB, позволил сделать работу с символьными выражениями простой и удобной. Если вы освоили работу с арифметическими выражениями, то символьные преобразования не должны вызвать затруднений.

## Определение переменных и функций и работа с ними

Символьные переменные и функции являются объектами класса `sym object`, в отличие от числовых переменных, которые содержатся в массивах `double array`. Символьный объект создается при помощи функции `syms`. Команда

```
>> syms x a b
```

создает три символьные переменные `x`, `a` и `b`. Размер памяти, отводимый по умолчанию под символьные переменные, достаточно большой — посмот-

рите информацию об определенных только что переменных в окне **Work-space** браузера рабочей среды или вызовите команду `whos`:

```
>> 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 символьных 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)
```

$$\frac{(\sin(x) + a)^2 (\cos(x) + b)^2}{|a + b|^{1/2}}$$

Определенная функция `f` также является символьной переменной типа `sym object`, в чем несложно убедиться при помощи браузера переменных.

Имеющиеся символьные переменные и функции позволяют образовывать новые символьные 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)
```

$$\frac{(\sin(x) + a)^2 (\cos(x) + b)^2 (\exp(-y) + 1)}{|a + b|^{1/2} \exp(y)}$$

Символьную 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
```

### Примечание

Для объявления символьных переменных может быть использована функция `sym`. 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 символьные переменные задаются командой `syms` с опцией `unreal`. Опция `real` означает, что переменные интерпретируются как вещественные. Убедитесь, что результат символьных вычислений зависит от того, какие символьные 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 символьных переменных `p` и `q`.

Использование одних и тех же операторов в символьных и числовых выражениях возможно благодаря тому, что пакет MATLAB является объектно-

ориентированной системой. Мы не будем описывать идеологию объектно-ориентированного программирования, а поясним сказанное на примере операции сложения. Для сложения чисел и числовых массивов в MATLAB зарезервирован знак +, который приводит к вызову встроенной функции plus, расположенной в подкаталоге \toolbox\matlab\ops\ основного каталога MATLAB. Проверьте, что  $1.3 + 2.9$  и plus(1.3, 2.9) приводят к одинарному результату. Для символьных переменных и функций используется файл-функция с тем же именем plus, находящаяся в подкаталоге \toolbox\symbolic\@sym\ основного каталога MATLAB. Изучите ее содержимое, открыв файл plus.m в редакторе MATLAB (не вносите в него изменений!). В начале оба ее входных аргумента приводятся к символьному типу данных при помощи функции sym, что позволяет складывать символьную переменную с числовой и получить символьный результат. Далее проверяется совпадение размеров входных аргументов, которые могут быть массивами. Последняя строка содержит вызов функции maple, служащей для обращения к соответствующей функции или оператору из ядра пакета Maple — в данном случае символьному сложению.

Говоря на языке объектно-ориентированного программирования, числовые переменные типа double array образуют *класс со своими методами* (в том числе plus). Для класса символьных объектов sym метод plus переопределен, MATLAB определяет по типу аргумента соответствующий метод класса и выполняет его.

### Примечание

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 разд. **MATLAB: Classes and Objects**.

## Матрицы и векторы

Символьные 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 результате образуются символьные 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]
>> F = A.*B
F =
[a*e, b*f]
[g*c, d*h]
```

Конструирование блочных символьных матриц не отличается от числовых, требуется следить за размерами соответствующих блоков:

```
>> D = [C A; B C]
D =
[a*e + b*g, a*f + b*h, a, b]
[c*e + d*g, c*f + d*h, c, d]
[e, f, a*e + b*g, a*f + b*h]
[g, h, c*e + d*g, c*f + d*h]
```

Обращение к элементам символьных матриц и векторов производится при помощи индексации, в том числе двоеточием и вектором со значениями индексов:

```
>> d2 = D(2, :)
d2 =
[c*e + d*g, c*f + d*h, c, d]
>> d = d2([1 3 4])
d =
[c*e + d*g, c, d]
```

Для удаления строк или столбцов символьных матриц используется пустой массив

```
>> D(1:2, :) = []
D =
[e, f, a*e + b*g, a*f + b*h]
[g, h, c*e + d*g, c*f + d*h]
```

## Вычисления с символьными переменными

Символьные операции позволяют находить точные значение выражений или значения со сколь угодно большой точностью. Для преобразования значения числовой переменной в символьную служит функция `sym`. Введите массив типа `double array`

```
>> A = [1.3 -2.1 4.9
 6.9 3.7 8.5];
```

и образуйте соответствующий ему символьный массив

```
>> B = sym(A)
B =
[13/10, -21/10, 49/10]
[69/10, 37/10, 17/2]
```

При переходе от числовых к символьным выражениям по умолчанию используется запись чисел в виде рациональной дроби. Возможны и другие формы представления чисел в символьном виде, которые определяются значением второго входного аргумента функции `sym` (см. разд. **MATLAB: Symbolic Math Toolbox: Function Reference** справочной системы).

Создайте вектор-столбец

```
>> c = [3.2; 0.4; -2.1];
```

и занесите в вектор `d` его символьное представление

```
>> d = sym(c);
```

Умножьте матрицу `B` на вектор `d` — результат является символьной переменной, причем все вычисления проделаны над рациональными дробями.

```
>> e = B*d
e =
[-697/100]
[571/100]
```

Использование рациональных дробей при выполнении символьных вычислений означает, что всегда получается точный результат, не содержащий погрешность округления. Убедиться в вышесказанном можно на простом 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 e
>> 1.0e+10 + 1.0e-10
ans =
1.000000000000000e+010
```

Запишите каждое из чисел в символьные переменные и снова вычислите сумму

Рациональная дробь является точным значением суммы. Разумеется, символьные вычисления требуют больших затрат компьютерных ресурсов по сравнению с обычными.

Вычисления с рациональными дробями позволяют получить значение символьного выражения с любой степенью точности, т. е. найти сколь угодно много значащих цифр результата. Для вычисления символьных выражений предназначена функция `vpa`:

```
>> c = sym('sqrt(2)');
>> cn = vpa(c)
cn =
1.4142135623730950488016887242097
```

По умолчанию удерживается 32 значащие цифры. Второй дополнительный входной параметр `ura` служит для задания точности:

```
>> cn = vpa(c, 70)
cn =
1.414213562373095048801688724209698078569671875376948073176679737990732
```

## Примечание

Второй аргумент `vra` задает удерживаемое число значащих цифр только для данного вызова `vra`. Для глобальной установки применяется функция `digits`, во входном аргументе которой указывается требуемое количество цифр.

Важно понимать, что выходной аргумент функции `vra` является символьной переменной:

```
>> whos cn
```

| Name | Size | Bytes | Class      |
|------|------|-------|------------|
| cn   | 1x1  | 266   | sym object |

Для перевода символьных переменных в числовые, т. е. переменные типа `double array`, используется функция `double`:

```
>> cnum = double(cn)
cnum =
1.41421356237310
```

## Графическое представление функций

Визуализация символьной функции одной переменной осуществляется при помощи `ezplot`. Самый простой вариант использования `ezplot` состоит в указании символьной 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. 17.1), что автоматически создается соответствующий заголовок. По умолчанию в качестве отрезка, на котором строится график, принимается промежуток пересечения области определения функции и интервала  $[-2\pi, 2\pi]$ . 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 символьной функции, зависящей от двух аргументов:

```
>> z = sym('x^2 + y^3');
>> ezplot(z, [-2 1 -3 4])
```

В данном случае выводится линия, на которой исследуемое выражение равно нулю.

### Примечание

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  $4$ .

При помощи `ezplot` возможно так же отображение параметрически заданных функций.

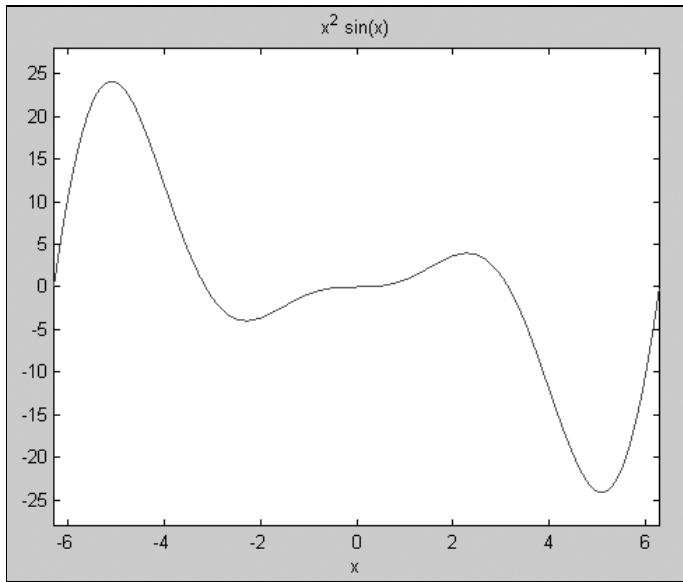


Рис. 17.1. График символьной функции

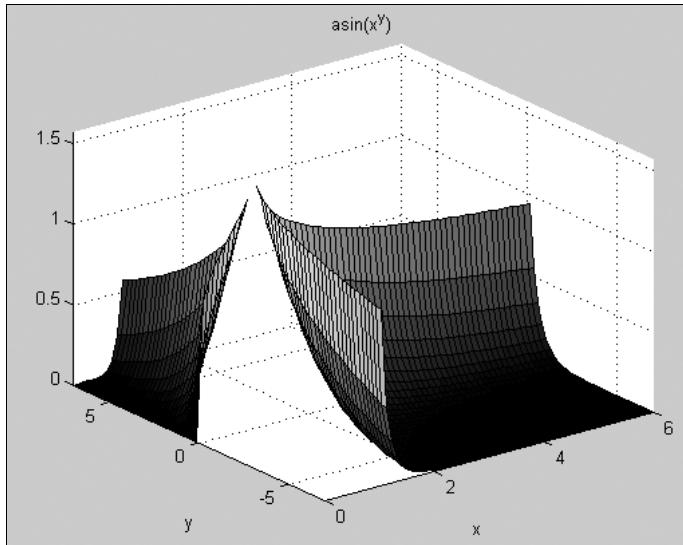


Рис. 17.2. График символьной функции двух переменных

Symbolic Math Toolbox предоставляет пользователю целый набор средств для визуализации символьных функций: `ezmesh`, `ezmeshc`, `ezplot`, `ezplot3`,

`ezpolar`, `ezsrf`, `ezsrfc`. Функция `ezsrf` отображает график символьной функции только для допустимых значений аргументов, остальные значения отбрасываются, что позволяет исследовать область определения функции двух переменных. Например, обращение

```
>> ezsrf('asin(x^y)', [0 6 -7 7])
```

приводит к графику, изображенному на рис. 17.2.

## Упрощение, преобразование и вычисление выражений

Сложные алгебраические и тригонометрические выражения нередко могут быть приведены к эквивалентным путем упрощения. Symbolic Math Toolbox имеет ряд сервисных функций, предназначенных для различных преобразований символьных выражений. Пользователь может производить как стандартные операции над полиномами, так и использовать более общий алгоритм, предназначенный для упрощения выражений, которые содержат встроенные символьные 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)
```

$$(x + a)^4 + (x - 1)^3 - (x - a)^2 - ax + x - 3$$

Преобразуйте `p` к виду, содержащему степени `x` с соответствующими коэффициентами:

```
>> pc = collect(p);
```

```
>> pretty(pc)
```

$$x^4 + (1 + 4a)x^3 + (-4 + 6a)x^2 + (4 + a + 4a)x + a - 4 - a$$

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)
```

$$a^4 + 4ax^3 + (-1 + 6x^2)a^2 + (4x^4 + x^3)a + x^4 + (x - 1)^3 - x^2 - 3 + x$$

Функция `expand` представляет полином суммой степеней без приведения подобных слагаемых:

```
>> pe = expand(p);
>> pretty(pe)
 4 3 2 2 3 4 3 2
x + 4ax + 6x a + 4xa + a + x - 4x + 4x - 4 + ax - a
```

Аргументом `expand` может быть не только полином, но и символьное выражение, содержащее тригонометрические, экспоненциальную и логарифмическую функции, например

```
>> f = sym('sin(arccos(3*x)) + exp(2*log(x))');
```

```
>> fe = expand(f);
```

```
>> pretty(fe)
```

$$(1 - 9x^{1/2})^2 + x^2$$

Символьные 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)
```

$$1/4 (2x + 1) (2x - 1) (x + 6) (x + 4) (x + 3)$$

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 =
```

$$2 \quad 3 \quad 5 \quad 11 \quad 17 \quad 41$$

выводит в командное окно аналогичный результат, но переменная `s` является символьной, а `s1` — вещественной, поскольку для объектов класса `sym` метод `factor` переопределен.

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 символное возвведение 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 символьной функции, переменной, подлежащей замене, и выражением, которое следует подставить вместо переменной. Функция `subs`, в частности, облегчает ввод громоздких символьных 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)
```

$$\frac{(\exp(x) + \exp(-x))^2 + (\sin(x) + \cos(x))^2}{(\exp(x) + \exp(-x))^2 - (\sin(x) + \cos(x))^2} + \frac{(\exp(x) + \exp(-x))^4}{(\sin(x) + \cos(x))^4}$$

Заметим, что необязательно было производить подстановку сначала для `a` и затем для `b`. Одновременная замена переменных соответствующими выражениями требует указания массивов ячеек в фигурных скобках (работа с массивами ячеек описана в разд. "Массивы структур и массивы ячеек" главы 8).

Последовательность команд

```
>> f = sym('(a^2 + b^2)/(a^2 - b^2) + a^4/b^4');
>> f=subs(f, {'a', 'b'}, {'(exp(x) + exp(-x))', '(sin(x) + cos(x))'});
>> pretty(f)
```

обеспечивает тот же самый результат.

Подстановка вместо переменной ее числового значения приводит к вычислению символьной 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 символьным 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 Symbolic Math Toolbox. Разумеется, следует учитывать, что далеко не все задачи имеют аналитическое решение.

## Задачи линейной алгебры

Функции Symbolic Math Toolbox выполняют в символьном виде операции с матрицами и векторами. Все задачи, описанные в главе 6, могут быть решены в символьной форме, причем соответствующие функции имеют те же названия, что и функции для численного решения, поскольку соответствующие методы переопределены для класса `sym`.

Вычисление определителя в символьной 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 символьного 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 A<sup>-1</sup> 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
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 символьных матриц могут быть не только символьными переменными, но и выражениями, и рациональными числами. В качестве примера найдите собственные числа матрицы Гильберта пятнадцатого порядка, используя сначала символьное представление элементов матрицы, а затем сравните их с результатом численного алгоритма. Определите временные затраты этих способов, сгенерировав отчет при помощи профайлера (см. разд. "Профайлер" главы 15).

Файл-программа для нахождения собственных чисел матрицы Гильберта двумя способами и замера времени приведена в листинге 17.1.

### Листинг 17.1. 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 символьном 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  $N_1(x, y)$ ,  $N_2(x, y)$ ,  $N_3(x, y)$ , каждая из которых равна единице в одной из вершин треугольника, а в двух остальных обращается в ноль (рис. 17.3). Вершины треугольника имеют координаты  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ . 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 17.2 содержит файл-программу basis для нахождения формул, по которым вычисляются требуемые функции. Обратите внимание, что Symbolic Math Toolbox позволяет генерировать выражения в виде, пригодном для занесения в собственную программу на Fortran или С. В файл-программе basis использована функция ccode, разработчикам программ на Fortran следует использовать fortran вместо ccode.

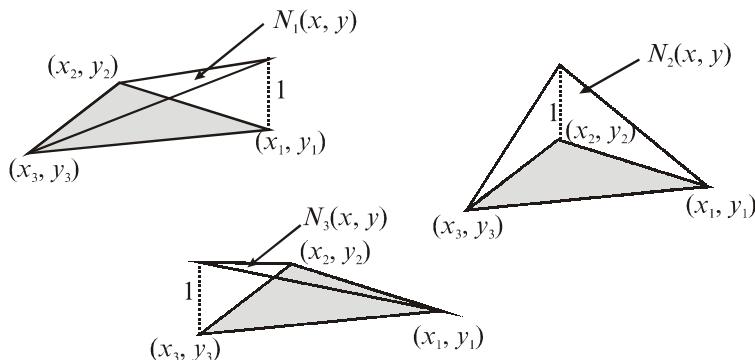


Рис. 17.3. Линейные базисные функции

**Листинг 17.2. 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 символьной 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 С-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 С-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 С-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 С. Ниже приведен результат только для первой базисной функции:

$$\begin{aligned} & -x_3 y_2 + x_2 y_3 - x y_3 + x y_2 + y x_3 - y x_2 \\ & \hline \\ & x_3 y_1 + x_2 y_3 - x_2 y_1 - x_3 y_2 - x_1 y_3 + x_1 y_2 \\ t0 = & (-x_3 y_2 + x_2 y_3 - x y_3 + x y_2 + y x_3 - y x_2) / (x_3 y_1 + x_2 y_3 - x_2 y_1 - x_3 y_2 - \\ & x_1 y_3 + x_1 y_2); \\ & \dots \end{aligned}$$

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)
```

$$1 - x + x^2 - x^3 + x^4 - x^5$$

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 символьная функция определена от нескольких переменных:

```
>> syms y
>> g = sym('1/(x + y)');
>> tg = taylor(g, 7, y);
>> pretty(tg)
```

$$\frac{1}{x} - \frac{y}{x^2} + \frac{y^2}{x^3} - \frac{y^3}{x^4} + \frac{y^4}{x^5} - \frac{y^5}{x^6} + \frac{y^6}{x^7}$$

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 математические функции, например:

```
>> pretty(taylor('exp(x)', 4, x, 1/2))

$$\begin{aligned} &\exp(1/2) + \exp(1/2)(x - 1/2) + 1/2 \exp(1/2)(x - 1/2)^2 \\ &+ 1/6 \exp(1/2)(x - 1/2)^3 \end{aligned}$$

```

В состав Symbolic Math Toolbox 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. 17.4.

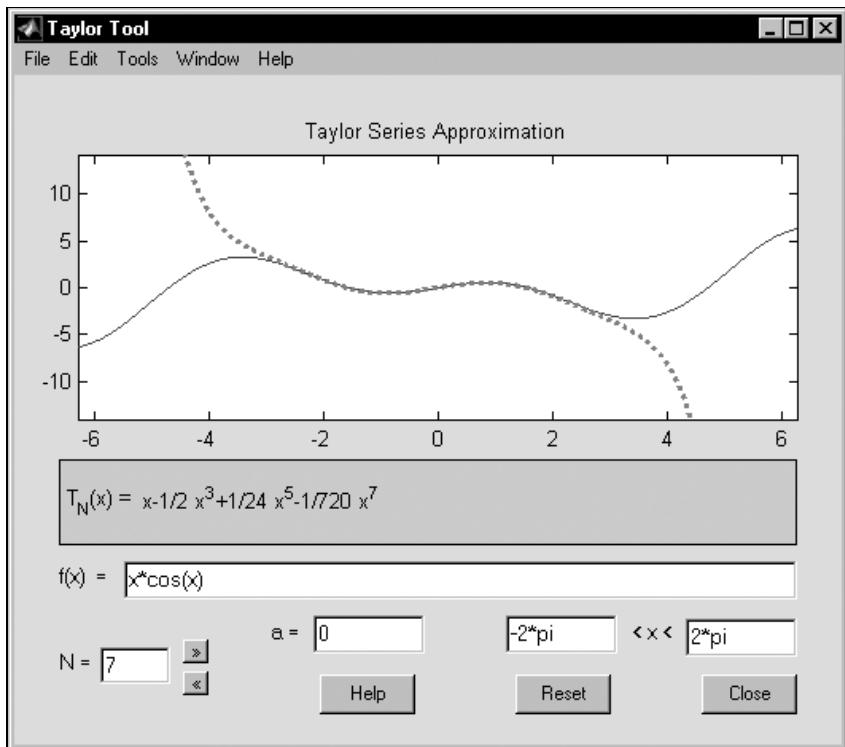


Рис. 17.4. Приложение `taylortool`

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 символьных выражений для сумм, в том числе и бесконечных, позволяет осуществить функция `symsum`. Обращение к `symsum` в общем виде предполагает задание четырех аргументов: слагаемого в символьной 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
```

Возможно суммирование слагаемых, зависящих не только от индекса, но и от некоторой символьной 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 Symbolic Math Toolbox предназначен для решения задач дифференциального и интегрального исчисления. Функция `limit` находит предел функции в некоторой точке, включая плюс или минус бесконечность. Первым входным аргументом `limit` является символьное выражение, вторым — переменная, а третьим — точка, в которой разыскивается предел. Пусть, например, требуется вычислить

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

Для получения ответа определите `a` и `x`, как символьные 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

$$\lim_{x \rightarrow 0_+} (10 + x)^{1/x}; \quad \lim_{x \rightarrow 0_-} (10 + x)^{1/x}.$$

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

$$\frac{d}{dx} \operatorname{arctg} x = \lim_{h \rightarrow 0} \frac{\operatorname{arctg}(x + h) - \operatorname{arctg} x}{h};$$

```
>> syms h x
>> L = limit((atan(x + h) - atan(x)) / h, h, 0);
>> pretty(L)
```

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

Вычисление производных любого порядка проще производить при помощи функции `diff`. Символьная запись функции указывается в первом входном аргументе, переменная, по которой производится дифференцирование — во втором, а порядок производной — в третьем. Применение `diff` для вычис-

ления производной в предыдущем примере, разумеется, приводит к эквивалентному результату:

```
>> P = diff('atan(x)', x, 1);
>> pretty(P)
```

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

Напишите файл-функцию `tangent` для исследования скорости роста функций. Входными аргументами `tangent` являются строка с символьным 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. 17.5.

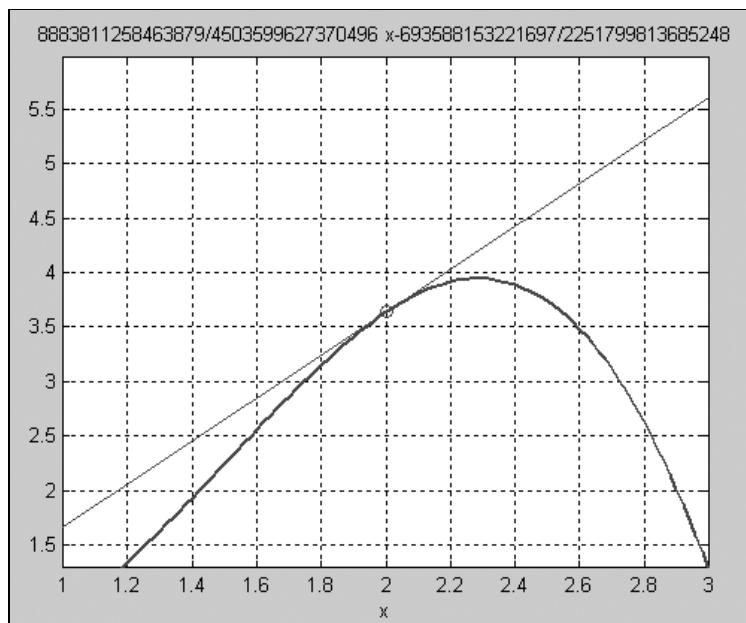


Рис. 17.5. Графики функции и касательной

Алгоритм файл-функции включает:

1. Определение символьной функции по строке при помощи `sym`.
2. Нахождение производной.
3. Формирование символьного 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 символьной 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 17.3 за дополнительной информацией в случае возникновения затруднений при программировании.

#### Листинг 17.3. Файл-функция `tangent`

```
function tangent(funstr, X0)
% Файл-функция для построения касательной
% к графику функции funstr в точке X0
% funstr – строка с символьным выражением функции
% Использование:
% tangent('exp(x)', 0)

% Задание символьной функции
syms x
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 символьное 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 символьного выражения для первой производной
k = diff(f, x, 1);
% Вычисление коэффициента касательной
K = subs(k, 'x', X0)
% Символьное 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 ту 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

```

Символьное интегрирование является значительно более сложной задачей, чем дифференцирование. Symbolic Math Toolbox позволяет работать как с неопределенными интегралами, так и с определенными. Неопределенные интегралы от символьных функций вычисляются при помощи `int`, в качестве входных аргументов указываются символьная 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 всегда может выполнить интегрирование. 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:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

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 символьном 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
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

$$\int \int_{c g}^{d b} y \sin x \, dx dy.$$

Определите символьные 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)

```

$$\frac{1}{2} \left( -\cos(b) + \cos(a) \right) (d^2 - c^2)$$

Аналогичным образом в символьном виде вычисляются любые кратные интегралы.

## 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)]
```

$$\frac{1}{(10 + 6 i \sqrt{111})^{1/2}}$$

$$\frac{1}{2} \frac{1}{3}$$

$$\frac{1}{(10 + 6 i \sqrt{111})^{1/3}}$$

Корни уравнения записываются в символьный массив `r`. Проверьте, что корни найдены верно, вычислите значение `f` от первого корня. Используйте функцию `simplify` для преобразования результата:

```
>> simplify(subs(f, 'x', r(1)))
ans =
0
```

Допустимо использование символьных 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 + e');
>> 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 символьных 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 символьное представление решения (работе со структурами данных посвящен разд. "Массивы структур и массивы ячеек" главы 8).

Решите систему нелинейных уравнений

$$\begin{cases} ax_1^2 + x_1x_2 + 1 = 0; \\ x_1^2 + bx_2 = 0 \end{cases}$$

и подстановкой проверьте полученные пары корней (при необходимости используйте `simplify` для упрощения результата):

```
>> syms x1 x2
>> f1 = sym('a*x1^2 + x1*x2 + 1');
>> f2 = sym('x1^2 + b*x2');
>> s = solve(f1, f2, x1, x2);
>> r1 = subs(f1, {x1,x2}, {s.x1(1), s.x2(1)});
>> simplify(r1)
>> r2 = subs(f2, {x1,x2}, {s.x1(1), s.x2(1)});
>> r1 = subs(f1, {x1,x2}, {s.x1(2), s.x2(2)});
>> simplify(r1)
>> r2 = subs(f2, {x1,x2}, {s.x1(2), s.x2(2)});
```

Итак, найденные пары корней точно удовлетворяют системе уравнений.

Если аналитическое решение невозможно, то возвращаются численные значения корней. Решите приведенную ниже систему уравнений и сравните ответ с приближенным решением, полученным при помощи `fsolve` в разд. "Решение нелинейных уравнений" главы 16:

$$\begin{cases} x_1(2 - x_2) = \cos x_1 \cdot e^{x_2}; \\ 2 + x_1 - x_2 = \cos x_1 + e^{x_2}. \end{cases}$$

Определите две символьные 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 символьные векторы:

```
>> s.x1
ans =
[.73908513321516064165531208767387]
[.67179203946718009639325125311984 - 1.3390702694949181314704735832283*i]
```

```
>> s.x2
ans =
[
 -lambertw(exp(2))+2
]
log(.67179203946718009639325125311984 - 1.3390702694949181314704735832283*i)
```

Функция `solve` нашла две пары решений: вещественные `s.x1(1)`, `s.x2(1)` и комплексные `s.x1(2)`, `s.x2(2)`. Первая компонента вектора `s.x2` выражена через функцию Ламберта  $w = L(x)$ , которая определена как зависимость решения  $x$  трансцендентного уравнения  $we^w = x$  от параметра  $w$ , входящего в уравнение. Вещественное решение совпадает с решением, полученным в главе 16 при помощи `fsolve`. Кроме того, функция `solve` нашла комплексные корни, решая систему нелинейных уравнений в символьном виде.

Интерфейс функции `solve` допускает использование символьных 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 символьными функциями не является обязательным. Входными аргументами `solve` могут быть строки с уравнениями, заключенные в апострофы, причем не обязательно переносить все слагаемые в левую часть. Независимые переменные можно так же не указывать, например, обращение

```
>> [x1, x2] = solve('x1*(2 - x2) = cos(x1)*exp(x2)', '2 + x1 - x2 = cos(x1) + exp(x2)')
```

аналогично приведенному выше.

## Решение дифференциальных уравнений и систем

Symbolic Math Toolbox 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

$$y' + y^2 = x^{-2}; \quad y(0.5) = -1.$$

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)
```

$$\frac{1/2 \tanh(-\sqrt{5} \log(x) + 1/2 \log(\frac{x^{1/2} - 1}{x^{1/2} + 1}))}{x}$$

Теперь решите уравнение Риккати численно, создайте файл-функцию `rikkaty` (листинг 17.4), 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 17.4. 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 осиях:

```
>> ezplot(y)
>> hold on
>> plot(X, Y, 'o')
>> grid on
```

Проверка (рис. 17.6) показывает совпадение аналитического и численного решений.

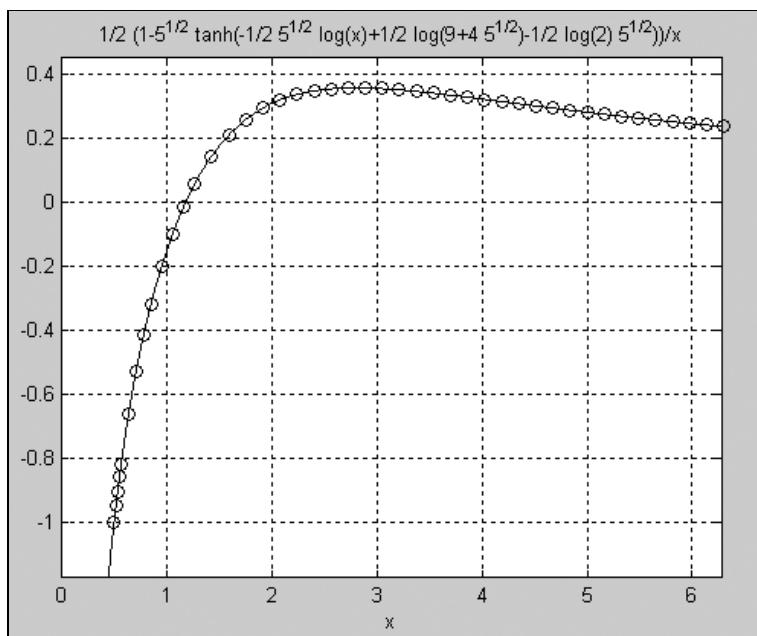


Рис. 17.6. Точное и приближенное  
решение уравнения Риккати

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 независимой переменной является `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 каче-

стве выходного аргумента вектор значений, каждый элемент которого становится символьной переменной:

```
>> [f, g] = dsolve('Df = exp(-g)', 'Dg = exp(-f)');
```

Выведите полученное решение в командное окно:

```
>> pretty(f)
```

$$\frac{\exp(t C_1 + C_2 C_1) - 1}{\log(\frac{C_1}{C_1})}$$

```
>> pretty(g)
```

$$\frac{\exp(t C_1 + C_2 C_1) C_1}{-\log(\frac{\exp(t C_1 + C_2 C_1) C_1}{\exp(t C_1 + C_2 C_1) - 1})}$$

Произведите проверку, подставив найденные символьные 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 `LegendreQ` при помощи встроенной справки приводит к сообщению об отсутствии однотипных M-файлов. Функции Symbolic Math Toolbox на самом деле реализуют интерфейс между средой MATLAB и библиотекой основных функций Maple. Функции `LegendreP` и `LegendreQ` 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)`

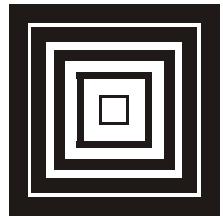
`LegendreQ(v, u, x)`

...

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 возможностей, предоставляемых Symbolic Math Toolbox пользователю, который желает использовать ресурсы вычислительного ядра Maple при работе в MATLAB с символьными выражениями. Если вы имеете опыт работы в Maple, то существенную пользу принесет Extended Symbolic Math Toolbox. Данное расширение основного Symbolic Math Toolbox позволяет оперировать как со всеми функциями Maple (исключая графические), так и создавать и выполнять приложения, написанные на языке программирования, встроенным в Maple.



## Глава 18

# Работа со сплайнами в Spline Toolbox

Область применения сплайнов весьма широка: они используются при аппроксимации одномерных и многомерных данных и функций, во многих других вычислительных задачах, а также в двумерной и трехмерной графике. Средства Spline Toolbox позволяют конструировать одномерные и многомерные сплайны (как интерполяционные, так и сглаживающие), удовлетворяющие различным условиям гладкости, и применять их для решения разнообразных задач. Входящие в состав Toolbox приложения с графическим интерфейсом служат для получения наглядного представления о поведении сплайнов и экспериментов с ними.

## Сплайны и формы их представления

Рассмотрим сначала задачу приближения табличной функции одной переменной с помощью сплайнов. Напомним основные определения и термины. Для решения задачи должны быть заданы две последовательности: точки  $x_1, x_2, \dots, x_N$  (сетка) и значения аппроксимируемой функции в этих точках  $y_1, y_2, \dots, y_N$  (сеточная функция). Сетка считается упорядоченной, т. е.  $x_i < x_{i+1}$ . Если аппроксимирующая функция проходит через заданные точки, то она называется интерполирующей, а иначе — сглаживающей. Функция  $S(x)$  называется сплайном степени  $k-1$  (порядка  $k$ ), если на каждом из отрезков  $[x_i, x_{i+1}]$ , где  $i=1, 2, \dots, N-1$ , эта функция является полиномом степени не выше  $k-1$ . Точки  $x_i$  ( $i=2, \dots, N-1$ ) называются точками разрыва, в них полиномы "сшиваются" так, что сохраняется непрерывность до производной некоторого порядка  $d_i < k-1$  (в разных точках условия гладкости могут быть различными). Используются две основные формы пред-

ставления сплайна: кусочно-полиномиальная (*pp*-форма) и *B*-форма, выражающая сплайн через базисные сплайны.

### Примечание

В русскоязычной литературе вместо термина "точки разрыва", как правило, используется термин "узлы". Мы будем придерживаться терминологии, принятой в пакете MATLAB.

## Кусочно-полиномиальная форма (*pp*-форма)

На каждом участке  $[x_j, x_{j+1}]$  с номером  $j$  приближающая функция  $S(x)$  представляется в виде полинома

$$P_j(x) = \sum_{i=0}^{k-1} a_i^{(j)} (x - x_j)^i.$$

*pp*-форма удобна для вычисления значений сплайна, его дифференцирования, интегрирования и других операций с ним. Построение сплайнов в *pp*-форме, т. е. определение множества коэффициентов  $a_i^{(j)}$ , сопряжено с решением системы линейных уравнений и изменение любого значения в табличных данных требует пересчета всех коэффициентов.

## *B*-форма (разложение по базисным сплайнам)

*B*-форма опирается на понятие базисных сплайнов. Несколько менее очевидный смысл их представления (по сравнению с *pp*-формой) компенсируется тем, что внесение изменений в табличные данные требует меньшего объема вычислений для перестройки сплайна. Сплайн в *B*-форме представляется суммой

$$S(x) = \sum_{j=1}^n B_{j,k}(x) a_j,$$

где  $k$  — порядок сплайна;  $B_{j,k}(x)$  — базисные сплайны (*B*-сплайны);  $a_j$  — массив коэффициентов;  $k$  — порядок сплайна. Отметим, что базисный сплайн  $B_{j,k}(x)$  положителен и отличен от нуля только в промежутке  $(t_j, t_{j+k})$ , ко-

торый называется носителем сплайна. Также выполняется условие нормировки:  $\sum_{j=1}^n B_{j,k} = 1$  для  $x \in [t_k, t_{k+1}]$ . Каждый  $j$ -ый базисный сплайн, в свою очередь, является кусочно-полиномиальной функцией степени меньше  $k$  с точками разрыва  $t_j, \dots, t_{j+k}$ . Последовательность точек  $t_1 \leq t_2 \leq \dots \leq t_{n+k}$  называется узлами  $B$ -формы.

Для обеспечения единственности сплайна заданных условий гладкости в точках разрыва и значений сеточной функции, как правило, недостаточно, поэтому дополнительно задают еще условия на концах рассматриваемого отрезка  $[x_1, x_N]$  — граничные (краевые) условия.

## Интерполяционные сплайны

Мы уже обсуждали аппроксимацию функций одной и нескольких переменных средствами MATLAB (см. разд. "Интерполирование и сглаживание" главы 6).

Обратимся теперь к рассмотрению возможностей, предоставляемых Spline Toolbox, и обсудим построение интерполяционных сплайнов, отличающихся порядками, формой представления и граничными условиями. Интерполяционный сплайн  $S(x)$  строится таким образом, чтобы для таблично заданной функции  $y$  выполнялись условия интерполяции:  $S(x_i) = y(x_i)$  для  $i = 1, \dots, N$ .

## Построение кубического сплайна

Рассмотрим сначала кубическую сплайн-интерполяцию средствами Spline Toolbox, поскольку кубические сплайны наиболее широко используются для приближения функций. При аппроксимации сплайнами существенную роль играют краевые условия.

## Стандартные краевые условия

Подготовим данные для приближения. Зададим последовательность равноотстоящих точек на промежутке  $[-1, 1]$  с шагом 0.2 и поместим их в массив  $x$ ,

введем inline-функцию, затем вычислим значения функции  $y(x) = e^{-x^2}$  в этих точках и запишем их в массив  $y$ :

```
>> x = -1:0.2:1;
>> fun = inline('exp(-x.^2)');
>> y = fun(x);
```

Построим теперь кубический интерполяционный сплайн при помощи функции `csapi`:

```
>> s = csapi(x, y)
```

```
s =
 form: 'pp'
 breaks: [1x11 double]
 coefs: [10x4 double]
 pieces: 10
 order: 4
 dim: 1
```

Функция `csapi` конструирует кубический интерполяционный сплайн, т. е. сплайн третьей степени (четвертого порядка), с так называемыми условиями "отсутствия узла" или "запрета стыка" (not-a knot condition), т. е. условиями неразрывности третьей производной сплайна в точках  $x_2$  и  $x_{N-1}$ . Требование непрерывности третьей производной в точке  $x_2$  фактически означает, что на участках  $[x_1, x_2]$  и  $[x_2, x_3]$  имеется единый интерполяционный полином  $P_l(x)$  и при этом выполняются условия  $P_l(x_i) = y_i$  при  $i=1, 2, 3$ .

Аналогичная ситуация на участках  $[x_{N-2}, x_{N-1}]$  и  $[x_{N-1}, x_N]$ . Таким образом, вместо  $N-1$  полинома мы будем иметь  $N-3$ . Входными аргументами функции `csapi` являются массивы точек разрыва и значений аппроксимируемой функции. В выходном аргументе возвращается структура  $s$ , содержащая описание кусочно-полиномиальной формы сплайна (работа со структурами описана в разд. "Простые структуры" главы 8).

При работе в Spline Toolbox умение оперировать структурами не является обязательным. Полученную структуру  $s$  можно использовать в качестве входного аргумента других функций Toolbox для построения графика сплайна, вычисления его значений, интегрирования, дифференцирования и ряда других действий.

## Операции над сплайнами

Сконструировав сплайн и записав информацию о нем в структуру, вы можете использовать целый набор функций Spline Toolbox для получения различных характеристик сплайнов и их визуализации. Каждая функция вызывается от структуры с описанием сплайна, например,

```
>> fnplt(s)
```

приводит к выводу графика сплайна. Отобразите исходные данные на этих же осях для проверки правильности результата. Для вычисления сплайна в некотором наборе точек служит функция fnval:

```
>> Y = fnval(s, [0.1 0.2])
```

```
Y =
```

```
0.9900 0.9608
```

Интегрирование и дифференцирование осуществляют, соответственно, функции fnint и fnder, которые возвращают структуру с описанием первообразной или производной требуемого порядка от заданного сплайна, например,

```
>> s2 = fnder(s, 2);
```

находит вторую производную. Вызов fnder с одним входным аргументом (структурой с описанием сплайна) приводит к вычислению первой производной. Наличие первообразной сплайна, которая получается с помощью функции fnint, позволяет интегрировать табличные функции — достаточно найти разность значений функции fnint в граничных точках. Проинтегрируйте сплайн s для нашего примера по его области определения и сравните с интегралом от функции  $y(x)$ :

```
>> S = fnint(s);
>> I = fnval(S, x(end)) - fnval(S, x(1))
I =
1.4936
>> quadl(fun, x(1), x(end))
ans =
1.4936
```

Для поиска минимального значения сплайна и его корней предназначены, соответственно, функции fnmin и fnzeros. Если они вызываются с одним входным аргументом (структурой с описанием сплайна), то поиск производится на всей области определения сплайна. Для вычисления корней или минимального значения на определенном промежутке следует задать его границы во втором дополнительном входном аргументе — векторе. Функ-

ция fnmin возвращает минимальное значение, а при обращении к ней с двумя выходными аргументами еще и абсциссу минимума:

```
>> [mval ,mx] = fnmin(s, [-1 0.5])
mval =
 0.3679
mx =
 -1
```

Выходным аргументом функции fnzeros является матрица из двух строк, число столбцов которой совпадает с числом корней сплайна. Содержимое столбца содержит информацию о корне — элементы столбца интерпретируются как границы интервала, которые могут: совпадать (если сплайн принимает в этой точке малое значение), быть примерно равны (если сплайн меняет знак на этом интервале) и отличаться друг от друга (если сплайн равен нулю на всем интервале).

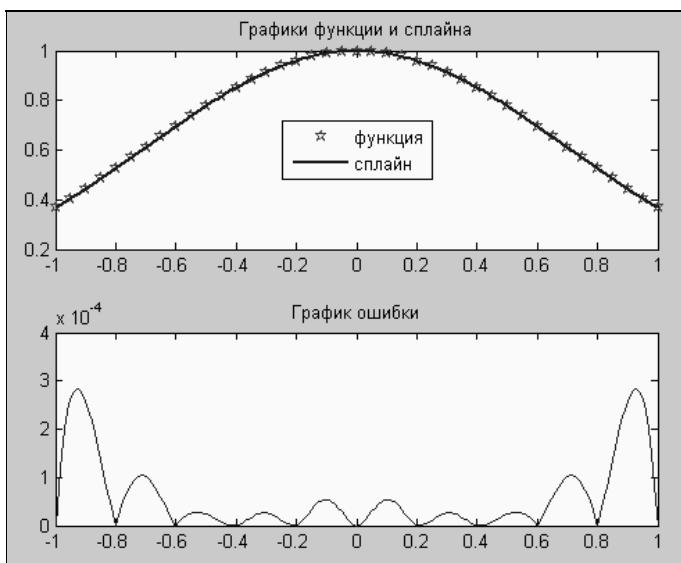
Все доступные операции над сплайнами описаны в справочной системе MATLAB по Toolbox (см. разд. **Spline Toolbox: Functions — Categorical List: Operators**).

Воспользуемся ими для вычисления погрешности.

Для получения наглядной информации о поведении сплайна и погрешности аппроксимации функции построим график сплайна и исходной функции на одних осях, а на других — график погрешности:

```
>> subplot(2, 1, 1)
>> x = -1:0.05:1;
>> y = fun(x);
>> plot(x, y, 'rp')
>> hold on
>> fnplt(s)
>> title('Графики функции и сплайна')
>> legend('функция', 'сплайн', 0)
>> xe = -1:0.01:1;
>> ye = abs(fnval(s, xe) - fun(xe));
>> subplot(2, 1, 2)
>> plot(xe, ye)
>> title('График ошибки')
```

Получающийся результат приведен на рис. 18.1.



**Рис. 18.1.** Представление кубического сплайна и погрешности интерполяции для функции  $y = e^{-x^2}$

Интерфейс функции `csapi` позволяет вместо конструирования сплайна в виде структуры вычислить его значения в заданных точках. Создадим вектор с абсциссами точек

```
>> x1 = -0.9:0.2:0.9;
```

и укажем его при обращении к `csapi` в качестве третьего входного аргумента:

```
>> value = csapi(x, y, x1);
```

В вектор `value` занесены значения кубического сплайна в заданных точках. Обращение `csapi(x, y, x1)` дает тот же результат, что и `fnval(csapi(x, y), x1)`.

## Построение сплайна для вектор-функции

Spline Toolbox позволяет конструировать сплайны для вектор-функций. Для построения, например, кубического сплайна используется рассмотренная выше функция `csapi`. В случае вектор-функций она вызывается от второго входного аргумента — матрицы, строки которой содержат значения функции в узлах.

Продемонстрируем приближение табличных одномерных данных, каждому значению аргумента которых соответствует два значения функции. Эти зна-

чения являются координатами точек, лежащих на кривой, которая называется "улиткой Паскаля" и задается параметрическими зависимостями:

$$x = a \cdot \cos^2 t + b \cdot \cos t, \quad y = a \cdot \cos t \cdot \sin t + b \cdot \sin t,$$

где  $0 \leq t \leq 2\pi$ ,  $a > 0$ ,  $b > 0$ .

Листинг 18.1 содержит текст файл-функции `ulitka2`, которая строит улитку Паскаля при помощи кубических сплайнов.

### Листинг 18.1. Файл-функция для построения улитки Паскаля

```
function ulitka2(a, b)
% Построение улитки Паскаля кубическими сплайнами

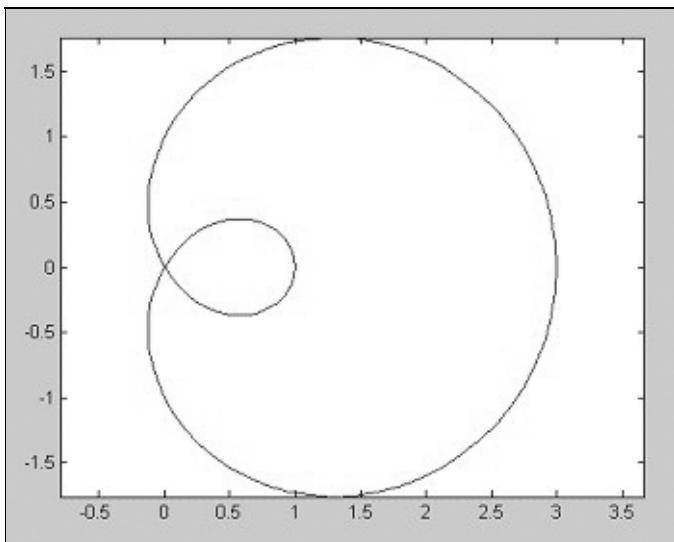
% Задание шага изменения параметра
h = pi/20;
% Задание значений параметра
t = 0:h:2*pi;
% Вычисление таблицы значений
x = a*cos(t).*cos(t) + b*cos(t);
y = a*cos(t).*sin(t) + b*sin(t)
% Формирование матрицы со значениями табличной функции
R = [x ; y];
% Создание сплайна
pp = csapi(t ,R);
% Вывод графика сплайна на оси с одинаковым масштабом
figure
fnplt(pp)
axis equal
```

### Вызов файл-функции

```
>> ulitka2(2, 1)
```

приводит к появлению улитки Паскаля (рис. 18.2).

Входными аргументами файл-функции `ulitka2` являются геометрические параметры улитки Паскаля  $a$  и  $b$ . Меняя эти величины, можно получить различные формы улитки. Исходными данными для построения сплайна являются равноотстоящие точки с шагом  $h$ , помещенные в  $t$ , и соответствующие им координаты  $x, y$  точек на кривой — координаты радиус-вектора улитки  $r$ . Функция `csapi` строит кусочно-полиномиальную форму кубического сплайна, который затем отображается графически при помощи `fnplt`.



**Рис. 18.2.** Улитка Паскаля, построенная с помощью применения кубического сплайна, при значениях параметров  $a = 2$ ,  $b = 1$

### Примечание

Для некоторых функций, в том числе и для рассмотренной `csapi`, происходит усреднение значения функции, если приближаемая функция многозначна, т. е. таблица содержит несколько одинаковых значений аргумента.

## Произвольные краевые условия

Как уже отмечалось выше, функция `csapi` строит сплайн с условиями "отсутствия узла". Эти условия дают наиболее точное представление интерполируемой функции, если нет информации об ее производных на границах рассматриваемого отрезка. При наличии каких-либо дополнительных данных о приближаемой функции, зачастую, целесообразно поставить другие краевые условия. Для этого применяется функция `csape`. Она допускает различные варианты обращения. Первый способ

```
s = csape(x, y)
```

приводит к построению кусочно-полиномиальной формы интерполяционного кубического сплайна с краевыми условиями, заданными по умолчанию (ниже мы скажем, что это за условия). Вызов `csape` с третьим входным аргументом

```
s = csape(x, y, cond)
```

позволяет выбирать краевые условия из довольно обширного списка, представленного в табл. 18.1. При этом может потребоваться расширение массива  $u$  для указания значений производной сплайна на границах отрезка. Возможно также задание собственных граничных условий, но тогда параметр  $conds$  должен быть не текстовой строкой, а вектором-строкой из двух элементов. При таком способе можно определить различные краевые условия. При этом значение первого элемента соответствует левой границе отрезка, а второго — правой. В массив  $u$  при этом добавляются дополнительные значения в начало и конец массива. Приняты следующие правила: если значение элемента вектора строки  $conds$  равно 1, то дополнительный элемент в  $u$  задает значение первой производной, если в  $conds$  элемент равен 2, то дополнительный элемент в  $u$  задает значение второй производной. В случае периодических условий  $conds$  задается массивом [0 0] и обращения  $s1 = csape(x, y, 'periodic')$  и  $s1 = csape(x, y, [0 0])$  приводят к одиаковому результату. Строковое значение 'variational' параметра  $conds$  эквивалентно заданию вектора [2 2] одновременно с записью дополнительных нулевых элементов в массив  $y$ . Если  $conds(j)$  (для  $j = 1$  и/или  $j = 2$ ) не задается или принимает значения, отличные от 0, 1, 2, то  $conds(j)$  принимается равным 1, а соответствующее значение в массиве  $u$  принимается по умолчанию следующим образом. Значением по умолчанию для дополнительных элементов  $u$  является производная кубического интерполянта, построенного по ближайшим четырем точкам.

Таблица 18.1. Параметры функции *csape*

| Значения параметра <i>conds</i> | Пояснения                                                                                                                                                 | Добавочные значения в массиве <i>y</i>                                                                        |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| 'complete'<br>или<br>'clamped'  | Условие закрепления. Совпадение тангенсов углов наклона (производных) на границах отрезка $[x_1, x_N]$ со значениями, заданными в расширенном массиве $y$ | Значения производных интерполируемой функции в $x_1$ и $x_N$ по умолчанию принимаются такими, как в "default" |
| 'not-a-knot'                    | Условие "отсутствия узла", описанное выше                                                                                                                 | Не требуется-<br>ся/игнорируется                                                                              |
| 'periodic'                      | Условие периодичности. Совпадение значений первой и второй производных на границах промежутка $[x_1, x_N]$                                                | Не требуется                                                                                                  |

Таблица 18.1 (окончание)

| Значения параметра <code>conds</code> | Пояснения                                                                                                                 | Добавочные значения в массиве <code>y</code>                                             |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| 'second'                              | Совпадение значений вторых производных на границах отрезка $[x_1, x_N]$ со значениями, заданными в массиве <code>y</code> | Значения в точках $x_1$ и $x_N$ вторых производных по умолчанию предполагаются равными 0 |
| 'variational'                         | Устанавливает вторые производные в точках $x_1$ и $x_N$ равными 0                                                         | Игнорируется                                                                             |

Рассмотрим пример приближения с помощью сплайнов периодической функции  $y = \cos(2\pi x)$  с последующим вычислением разности ее первых производных на концах  $x_1 = 0$ ,  $x_N = 1$

```
>> x = 0:0.05:1; y = cos(2*pi*x);
>> s = csapi(x, y);
>> s1 = csape(x, y, 'periodic');
>> diff(fnval(fnder(s), [0 1]))
ans =
 -0.0665
>> diff(fnval(fnder(s1), [0 1]))
ans =
 -3.7007e-016
```

В приведенном примере задаются исходные данные и строятся сплайны в *pp*-форме: `s` с условиями "отсутствия узла", и `s1` — с периодическими условиями. Затем вычисляется разность значений производных на концах промежутка, т. е. через период, равный 1. Здесь используются следующие функции Spline Toolbox. `fnder` позволяет вычислить первую производную сплайна. Результатом `fnder(s)` является структура с полями, аналогичными `s`. Функция `fnval` вычисляет значения первой производной сплайна в точках, указанных во втором ее входном аргументе. В данном случае он является массивом из двух элементов `[0 1]`, т. е. вычисления проводятся в точках  $x_1 = 0$ ,  $x_N = 1$ . В завершение с помощью функции MATLAB `diff` вычисляется разность полученных значений первой производной. Затем те же действия по вычислению разности значений первых производных в начале и конце периода проводятся для структуры `s1`. Из полученных резуль-

татов видно, что в первом случае, т. е. при наложении условий "отсутствия узла", разность полученных значений производной в начале и конце периода довольно велика по сравнению со вторым случаем задания периодических граничных условий при построении сплайна.

Чтобы еще раз подчеркнуть важность корректного задания граничных условий, вычислим упоминавшуюся разность значений первой производной сплайна, вторые производные которого на границах отрезка обращаются в ноль, т. е. "естественного сплайна":

```
>> s2 = csape(x, [0 y 0], [2 2]);
>> diff(fnval(fnder(s2), [0 1]))
ans =
 1.1490
```

Как видно, здесь результат получился существенно хуже. Таким образом, искусственное наложение условий  $y''(x_1)=0$ ,  $y''(x_N)=0$  при построении сплайна может привести к ощутимой погрешности. Нетрудно также проверить, что вызовы `s2 = csape(x, y, 'variational')` и `s2 = csape(x, [0 y 0], [2 2])` эквивалентны.

Другие примеры содержатся в справочной системе MATLAB по Toolbox (см. разд. **Spline Toolbox: Functions – Categorical List: Construction of Splines**).

## Использование сплайнов в B-форме

Обратимся теперь к работе со сплайнами в *B*-форме, которая была определена выше (см. разд. *"B-форма (разложение по базисным сплайнам)"* этой главы).

Напомним, что сплайн порядка  $k$  с узлами  $t_1, \dots, t_{n+k}$  в *B*-форме представляется суммой *B*-сплайнов  $B_{i,k}$ , каждый из которых отличен от нуля на интервале  $[t_i, t_{i+k}]$ . Узлы в последовательности могут совпадать и количество повторений узла определяет гладкость соединения между собой кусочных полиномов, составляющих *B*-сплайн. При формировании массива узлов должно выполняться следующее правило: *количество повторений узла в последовательности узлов равно разности между порядком сплайна и количеством условий гладкости (непрерывности функции и ее производных)*, накладываемых в данной точке. Следовательно,  $k$ -кратный узел в некоторой точке соответствует отсутствию условий непрерывности вообще, а простой узел предполагает непрерывность функции и ее производных до порядка  $k-2$  включительно.

В результате может быть реализовано приближение функций с различной степенью гладкости. *B*-форма может иметь разрывы в производных не только во внутренних, но и в граничных узлах. Принято выбирать начальные и конечные узлы так:  $t_1 = \dots = t_k = x_1$ ,  $t_{n+1} = \dots = t_{n+k} = x_N$ . Последовательность узлов может быть построена вручную или с помощью функции `augknt`. При обращении `augknt(x, k)` генерируется последовательность, у которой первый и последний узлы имеют кратность  $k$ :

```
>> knots = augknt([1 2 3 4 5], 4)
```

```
knots =
```

```
Columns 1 through 7
```

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|

```
Columns 8 through 11
```

|   |   |   |   |  |  |  |
|---|---|---|---|--|--|--|
| 5 | 5 | 5 | 5 |  |  |  |
|---|---|---|---|--|--|--|

Второй способ вызова `augknt(x, k, mults)` позволяет с помощью параметра `mults` указать желаемую кратность каждого внутреннего узла. При этом если `mults` является постоянной, то все внутренние узлы имеют одинаковую кратность.

```
>>knots = augknt([1 2 3 4], 3, 2)
```

```
knots =
```

```
Columns 1 through 7
```

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|

```
Columns 8 through 10
```

|   |   |   |  |  |  |
|---|---|---|--|--|--|
| 4 | 4 | 4 |  |  |  |
|---|---|---|--|--|--|

Для задания желаемой кратности в каждом внутреннем узле следует сформировать подходящий массив `mults` со значениями кратности:

```
>>knots = augknt([1 2 3 4], 3, [1 2])
```

```
knots =
```

```
Columns 1 through 7
```

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 3 | 3 | 4 |
|---|---|---|---|---|---|---|

```
Columns 8 through 9
```

|   |   |  |  |  |  |  |
|---|---|--|--|--|--|--|
| 4 | 4 |  |  |  |  |  |
|---|---|--|--|--|--|--|

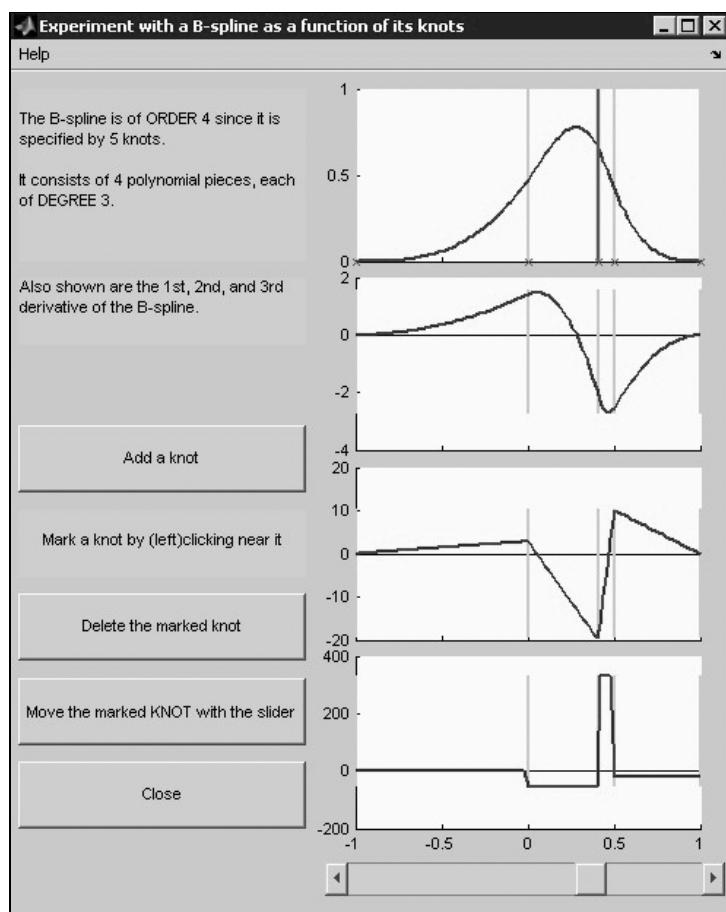
### Примечание

Следует отметить, что вне базового промежутка  $[t_1, t_{n+k}]$  *B*-форма дает нулевое значение, в то время как *pp*-форма слева от базового промежутка даст значение, определяемое кусочным полиномом на первом участке  $P_1$ , а справа от базового участка — полиномом на последнем участке  $P_{N-1}$ .

Для изучения влияния узлов на вид и порядок *B*-сплайна в Spline Toolbox включено приложение с графическим интерфейсом пользователя `bspligui`. После вызова

```
>> bspligui
```

откроется окно **Experiment with a B-spline as a function of its knots** (рис. 18.3), в котором изображен *B*-сплайн и его первые три производные. С помощью кнопок **Add a knot** (Добавить узел), **Mark a knot by (left) clicking near it** (Отметить узел щелчком мыши), **Delete the marked knot** (Удалить помеченный узел), **Move the marked knot with the slider** (Передвинуть помеченный узел с помощью бегунка) можно экспериментировать с количеством узлов, их положением и кратностью. В верхней части окна появляется информация о порядке *B*-сплайна и количестве полиномиальных кусков, составляющих его.



**Рис. 18.3.** Эксперименты с *B*-сплайнами с помощью функции `bspligui`

Рассмотрим пример интерполяции сплайнами функции, имеющей разрыв первой производной в некоторой точке. Пусть функция задана следующим образом:

$$y(x) = \begin{cases} (x+1)^2 & |x < 0; \\ (x-1)^2 & |x \geq 0. \end{cases}$$

Построим сплайн третьего порядка (квадратичный). В качестве значений аргумента выберем следующие:  $x = [-1 -0.25 0 0.25 1]$ . Рассматриваемая функция имеет разрыв первой производной в точке  $x=0$ . Для построения сплайн-функции может быть использована функция `spapi` в следующей форме: `spapi(knots, xx, yy)`. Здесь массив `knots` содержит узлы, необходимые для построения сплайна, количество их равно  $n+k$ , где  $k$  — порядок сплайна;  $n$  — количество значений в массиве `xx`. В массиве `yy` записаны значения функции от соответствующих элементов массива `xx`.

### Предупреждение

Если при использовании функции `spapi` некоторое значение в массиве `xx` повторяется  $r$  раз, то соответствующими элементами массива `yy` должны быть значения функции и ее первых  $(r-1)$  производных.

В нашем примере для определения сплайна единственным образом требуется дополнительно к значениям функции задать значения производных в каких-либо точках слева и справа от точки разрыва  $x=0$ . Следующая последовательность команд реализует поставленную задачу

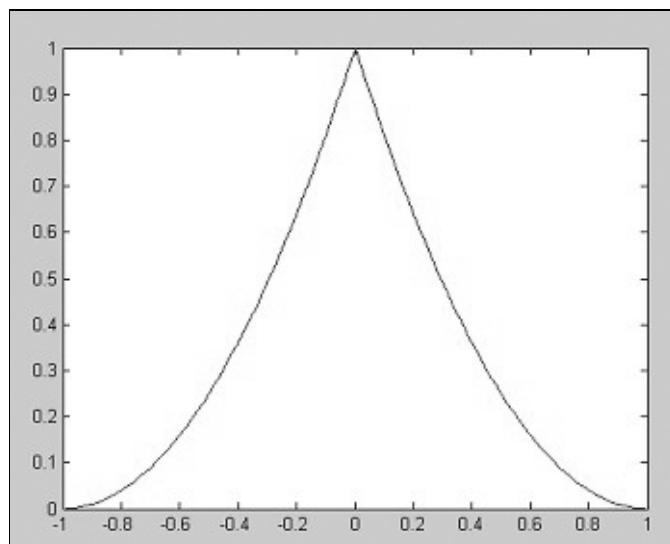
```
>> xx = [-1 -0.5 -0.5 0 0.5 0.5 1];
>> yy = [0 0.25 1 1 0.25 -1 0];
>> knots = [-1 -1 -1 -0.5 0 0 0.5 1 1 1];
>> s = spapi(knots, xx, yy)
>> fnplt(s)
```

Согласно приведенному выше правилу построения последовательности узлов, в массиве `knots` значение узла  $t=0$  повторяется дважды, т. к. в этом узле нарушается непрерывность первой производной (порядок сплайна равен трем, в узле  $t=0$  только одно условие — непрерывность функции). В массиве `xx` значения  $-0.5$  и  $0.5$  повторяются дважды, значит, в массиве значений функции `yy` им должны соответствовать значения самой функции и ее первой производной. Отметим, что дополнительные значения производных должны быть взяты с разных сторон от точки разрыва функции.

В командное окно выводится содержимое структуры `s` с информацией о построенном сплайне: получена  $B$ -форма с десятью узлами, количеством коэффициентов, равным семи, сплайн третьего порядка размерности один.

```
s =
 form: 'B-'
 knots: [1x10 double]
 coefs: [1x7 double]
 number: 7
 order: 3
 dim: 1
```

Результат работы функции `fnpkt` представлен на рис. 18.4.



**Рис. 18.4.** Построение сплайн-функции с разрывом производной с помощью функции `spapi`

Обращение к `spapi` в форме `sp = spapi(k, x, y)` позволяет строить сплайны различных порядков. В качестве первого аргумента указывается порядок интерполяционного сплайна. Следующими аргументами являются массивы `x` и `y`, представляющие собой входные данные для построения сплайна — массив координат точек разрыва и значений функции соответственно. В выходном аргументе (структуре) возвращается информация о сплайне. При таком вызове функции `spapi` подходящая последовательность узлов формируется автоматически с помощью функции `aprknt(x, k)`.

Рассмотрим построение сплайнов разных порядков по одним и тем же данным. В первом случае строится сплайн третьего порядка (квадратичный), а во втором — второго порядка. При этом, как следует из приведенных результатов, формируются разные массивы узлов:

```
>> x = 0:4;
>> y = x.*exp(-x.*x);
>> s2 = spapi(3, x, y)
>> fnplt(s2, '-', 1)
>> hold on
>> s1 = spapi(2, x, y)
>> fnplt(s1, '+', 2)
s2 =
 form: 'B-'
 knots: [0 0 0 1.5000 2.5000 4 4 4]
 coefs: [1x5 double]
 number: 5
 order: 3
 dim: 1
s1 =
 form: 'B-'
 knots: [0 0 1 2 3 4 4]
 coefs: [0 0.3679 0.0366 3.7023e-004 4.5014e-007]
 number: 5
 order: 2
 dim: 1
```

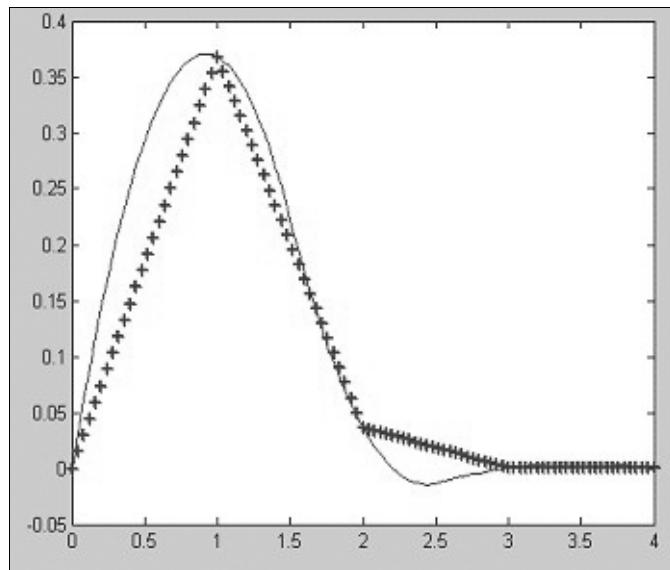
Графики первого и второго сплайнов приведены на рис. 18.5. Сплошной линией показан сплайн третьего порядка, а крестиками — второго.

Поскольку в сгенерированных массивах узлов `knots` каждый внутренний узел встречается лишь один раз, то это означает, что наложено максимально возможное для данного сплайна количество условий гладкости в каждом узле. В этом можно убедиться, отобразив график первой и второй производных сплайна:

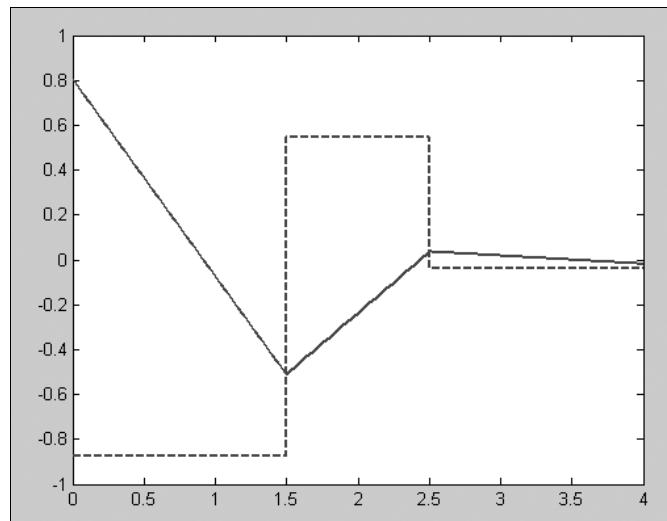
```
>> fnplt(fnder(s2), '-')
>> hold on
>> fnplt(fnder(s2, 2), '--')
```

Для сплайна третьего порядка максимально возможное количество условий в узле — непрерывность самой функции и ее первой производной, о чём

свидетельствует рис. 18.6 (сплошная линия — график первой производной, а штриховая — второй).



**Рис. 18.5.** Сплайны разных порядков, построенные по одним и тем же данным



**Рис. 18.6.** Графики первой и второй производных для сплайна третьего порядка

## Сглаживающие сплайны

На практике часто значения аппроксимируемой функции  $y_i$  известны с некоторой погрешностью  $\bar{y}_i = y_i + \varepsilon_i$ , где  $i = 1, \dots, N$ , например, являются данными эксперимента. Поэтому построение интерполяционного сплайна, т. е. точное удовлетворение этим данным становится неэффективным, особенно в тех случаях, когда погрешность в данных может быть велика. В этом случае применяются сглаживающие сплайны. Сглаживающий сплайн сообщает минимум функционалу:

$$J(s) = p \sum_{i=1}^N w_i (y_i - s(x_i))^2 + (1-p) \int_{x_1}^{x_N} \lambda(t) (s^{(m)}(t))^2 dt,$$

где  $p \in [0, 1]$  — параметр;  $w_i$  — неотрицательные весовые множители, которые нередко выбираются равными 1;  $\lambda(t)$  — неотрицательная весовая функция, как правило, полагается постоянной и равной единице. Для кубического сплайна  $m = 2$ .

Сгенерируем данные со случайным возмущением ("шумом"), а затем по ним — кубический сглаживающий сплайн с помощью функции `csaps` (`x, noisy`), первым параметром которой является массив координат, а вторым — значений возмущенной функции.

### Примечание

Данные с "шумом" получены путем добавления к значениям функции из предыдущего примера случайной составляющей с помощью следующих команд `x = 0:0.1:3; y = 2*x.*exp(-x.*x); noisy = y + .2*(rand(size(x)) - .5)`. Для повторяемости результатов переменные записаны в файл `set18.mat` и содержатся на прилагаемом к книге компакт-диске. Для их использования указанный файл следует скопировать в текущий каталог (способы считывания и записи двоичных данных разобраны в главе 1, а установка текущего каталога в разд. "Установка путей" главы 5).

Следующая последовательность команд приводит к построению сглаживающего сплайна и его графическому отображению. На рис. 18.7 сплошной линией показан сглаживающий сплайн, а кружками — исходные данные

```
>> load set18.mat
>> scs = csaps(x, noisy);
>> fnplt(scs)
>> hold on
```

```
>> plot(x, noisy, 'ro')
>> legend('сплайн', 'данные')
>> hold off
```

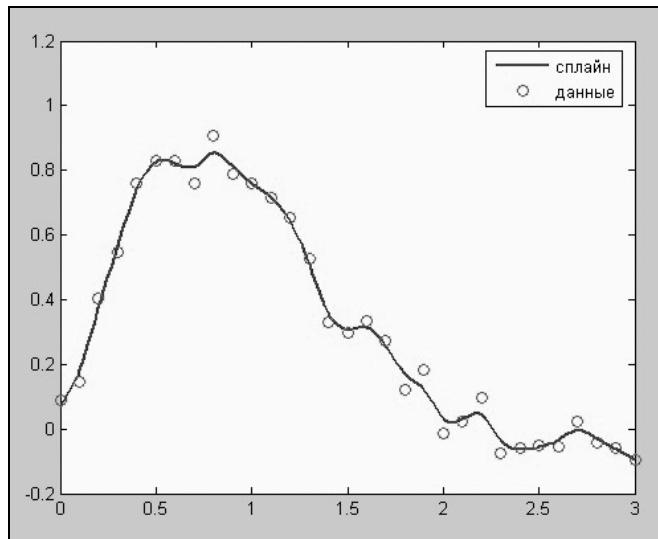
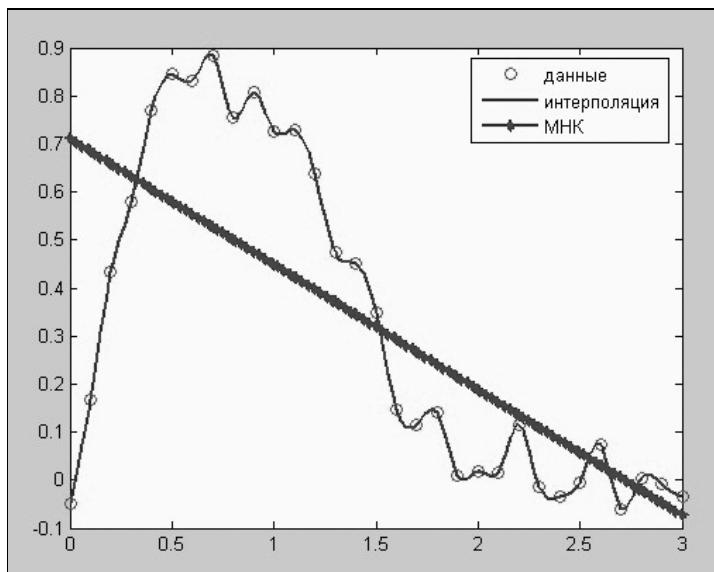


Рис.18.7 Кубический сглаживающий сплайн

Чтобы "регулировать", насколько близко сглаживающий сплайн должен подходить к исходным данным, можно использовать функцию `csaps` в другой форме `csaps(x, y, p)`, где  $p$  — параметр сглаживания. Крайние значения  $p = 1$  и  $p = 0$  приводят, соответственно, к интерполяционному кубическому сплайну и полиному первой степени (прямой), построенному по методу наименьших квадратов.

```
>> load set18.mat
>> scs1 = csaps(x, noisy, 1)
>> scs0 = csaps(x, noisy, 0)
>> plot(x, noisy, 'ro')
>> hold on
>> fnplt(scs1)
>> fnplt(scs0, 'b*-')
>> legend('данные', 'интерполяция', 'МНК')
>> hold off
```

Структура `scs1` содержит  $pp$ -форму интерполяционного кубического сплайна, а `scs0` — данные для построения прямой по методу наименьших квадратов (МНК). Графики полученных сглаживающих сплайнов приведены на рис. 18.8.



**Рис. 18.8.** Случаи предельных значений параметра сглаживания

Существуют некоторые рекомендуемые значения параметра  $p$ , так, по умолчанию принимается значение

$$p = \left(1 + \frac{h^3}{6}\right)^{-1},$$

где  $h$  — усредненное расстояние между абсциссами данных. Можно проводить эксперименты, меняя значение  $p$ . Узнать текущее значение  $p$  позволяет обращение `[sp, p] = csaps(x, y)`.

Кроме того, сглаживающий сплайн можно строить с помощью функции `spaps`, которая возвращает сплайн в  $B$ -форме: `sp = spaps(x, y, tol)`. Два первых входных аргумента задают исходные данные, а параметр `tol` определяет допустимое отклонение построенного сплайна от исходных данных:

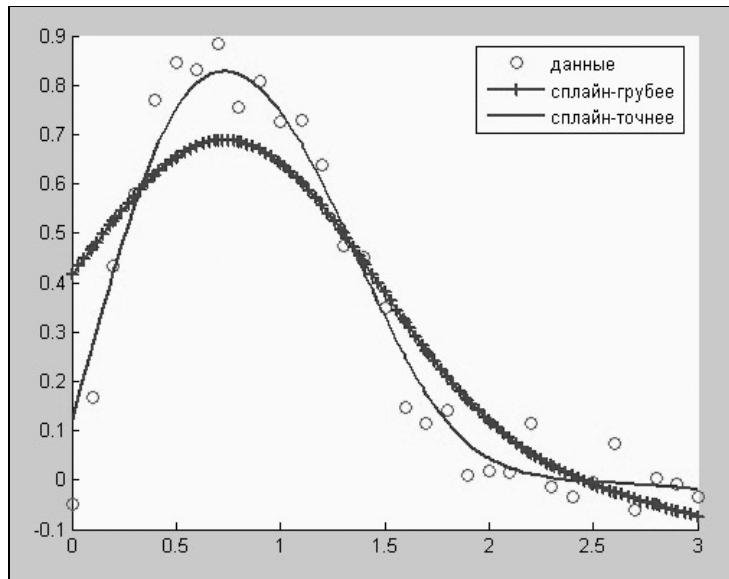
```
>> load set18.mat
>> scs1 = spaps(x, noisy, 0.05);
```

```

>> scsp2 = spaps(x, noisy, 0.01);
>> hold on
>> plot(x, noisy, 'ro')
>> fnplt(scsp1, 'b+-')
>> fnplt(scsp2)
>> legend('данные', 'сплайн-грубее', 'сплайн-точнее')

```

Приведенная последовательность команд приводит к конструированию двух сглаживающих сплайнов с разными допустимыми отклонениями от исходных данных (рис. 18.9).



**Рис. 18.9.** Сглаживающие сплайны при различных допустимых отклонениях от исходных данных

В Spline Toolbox имеется возможность построения  $B$ -формы сплайнов различных порядков, минимизирующих функционал:

$$J(s) = \sum_{i=1}^N w_i (y_i - s(x_i))^2,$$

т. е. применяется взвешенный метод наименьших квадратов в стандартной форме. Для этого используется функция `spap2`. При обращении к ней

`sp = spap2(knots, k, x, y)` весовые множители полагаются равными единице. Здесь `knots` — массив узлов, `x`, `y` — массивы исходных данных, `k` — порядок сплайна. Массив `y` может содержать скалярные, векторные или матричные данные. Для указания весовых множителей, отличных от 1, следует записать их в вектор `w`, длина которого совпадает с длиной `x`, и задать в пятом входном аргументе: `sp = spap2(knots, k, x, y, w)`.

### Примечание

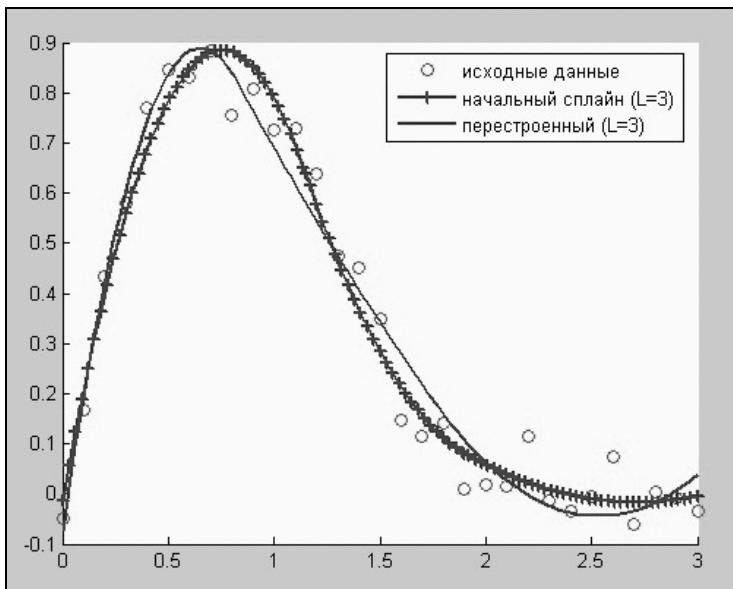
Данные в массиве `y` для функции `spap2`, соответствующие повторяющейся координате, усредняются.

Возможно избежать составления последовательности узлов, прибегнув к обращению `s = spap2(L, k, x, y)`. При этом последовательность узлов строится автоматически с помощью функции `aptknt`. Параметр `L` должен быть целым и положительным, он определяет количество участков для построения сплайна. Если  $L \geq (\text{length}(x) - k)$ , то находится интерполяционный сплайн, иначе — сглаживающий. В любом случае сплайн имеет  $k - 2$  непрерывных производных.

При построении сглаживающего сплайна точки разрыва определяются автоматически. С помощью функции `newknt(s)`, где `s` — структура с информацией о сконструированном сплайте, можно попытаться улучшить предложенную последовательность узлов и перестроить сплайн, снова обратившись к функции `spap2`: `s1 = spap2(newknt(s), k, x, y)`. Применение этих возможностей иллюстрируется следующими командами, выполнение которых приводит к сплайнам, представленным на рис. 18.10.

```
>> load set18.mat
>> s = spap2(3, 3, x, noisy);
>> hold on
>> plot(x, noisy, 'ro')
>> fnplt(s, 'b+-')
>> s1 = spap2(newknt(s), 3, x, noisy);
>> fnplt(s1, '-', 2);
>> legend('исходные данные', 'начальный сплайн', 'перестроенный')
>> hold off
```

Более подробно о возможностях функции `spap2` можно узнать в справочной системе MATLAB по Toolbox (см. разд. **Spline Toolbox: Functions – Categorical List: Construction of Splines**).



**Рис. 18.10.** Построение сплайна с помощью функции `spap2` и его коррекция

## Интерактивное построение кривых

В Spline Toolbox возможно построение кривых в интерактивном режиме. Для этого используется приложение с графическим интерфейсом `getcurve`. При его запуске

```
>> getcurve
```

открывается окно, в котором с помощью мыши требуется отметить точки на осях (рис. 18.11). Щелчок мышью вне осей означает завершение ввода и приводит к созданию кривой (рис. 18.12). При этом координаты точек записываются в стандартную переменную `ans` и выводятся в командное окно:

```
ans =
```

|         |         |         |        |        |
|---------|---------|---------|--------|--------|
| -0.7903 | -0.4032 | -0.0115 | 0.3848 | 0.5922 |
| -0.2193 | 0.2076  | -0.0029 | 0.2076 | 0.0088 |

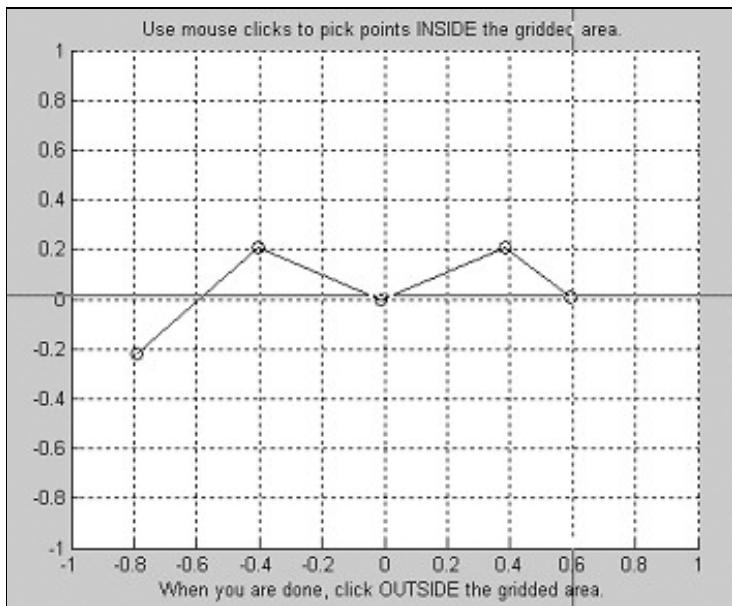


Рис. 18.11. Работа в окне функции `getcurve`

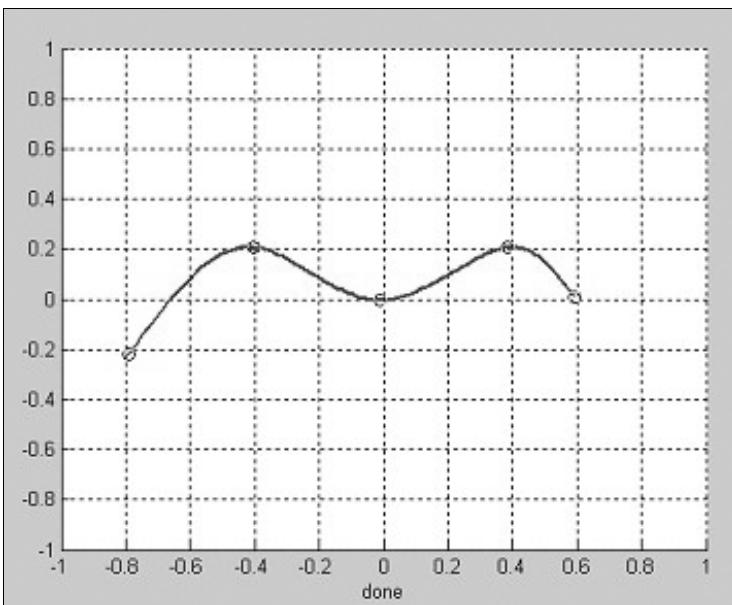


Рис. 18.12. Результат работы функции `getcurve`

## Приложение *splinetool*

В состав Spline Toolbox входит приложение с графическим интерфейсом пользователя *splinetool*, которое позволяет провести наглядные эксперименты по аппроксимации сплайнами. Пользователь может взять данные из имеющегося набора или задать собственные, а также выбрать:

- способ аппроксимации — интерполяционные сплайны или стягивающие сплайны;
- различные типы граничных условий;
- порядок используемого сплайна.

Рассмотрим работу с интерполяционными кубическими сплайнами. При обращении

```
>> splinetool
```

открывается окно приложения **Spline Tool**, приведенное на рис. 18.13, с кнопками для выбора данных.

Нажмите первую кнопку **Provide your own data or an m-file that does**, предназначенную для ввода собственных данных или имени М-файла, в котором они создаются. Откроется окно **Please provide data**, показанное на рис. 18.14, в котором введены данные для демонстрационного примера приближения функции  $\cos x$ .

Ознакомьтесь сначала с предлагаемым вариантом. Ничего не изменяя, нажмите кнопку **OK**. Откроется окно **Spline Tool** (рис. 18.15).

В окне **Spline Tool**, показанном на рис. 18.15, демонстрируется приближение функции  $\cos x$  интерполяционным кубическим сплайном с условиями "отсутствия узла". В правой части окна выводится графическая информация. На верхнем графике справа сплошной линией показан кубический сплайн, а кружками — исходные данные. На нижнем (вспомогательном) графике приведена ошибка, возникающая при замене исходной функции сплайном.

### Примечание

График ошибки может быть выведен, если известно аналитическое выражение аппроксимируемой функции. Если же функция табличная и задается массивом своих значений в точках разрывов, то и ошибка будет выводиться там же, т. е. при интерполировании она получится нулевой, с точностью до погрешности округления.

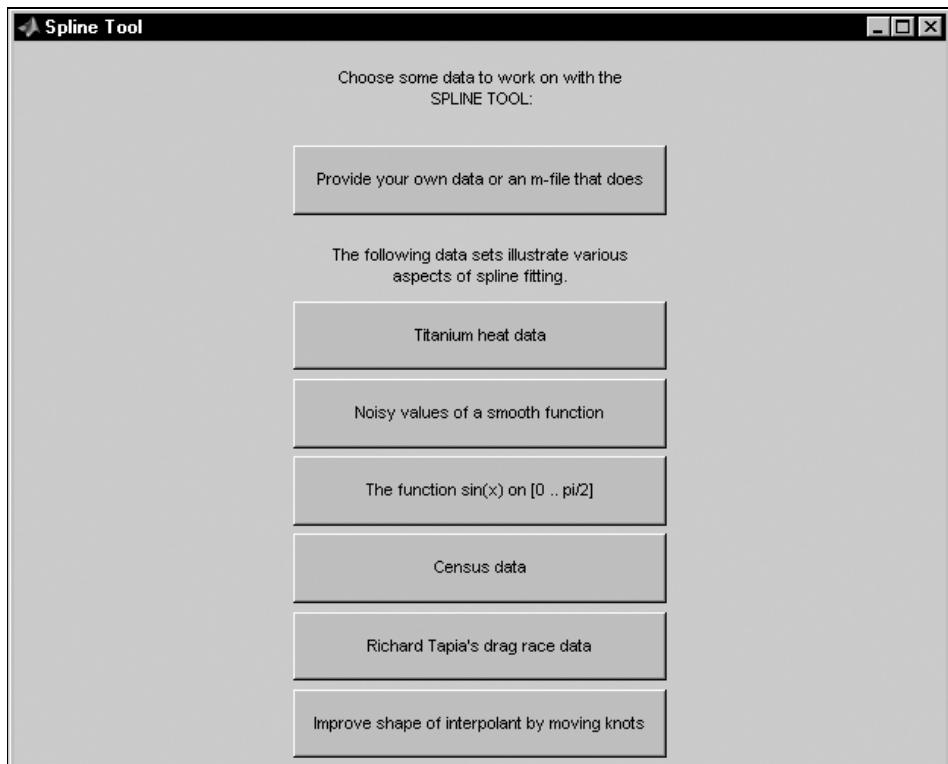


Рис. 18.13. Окно Spline Tool

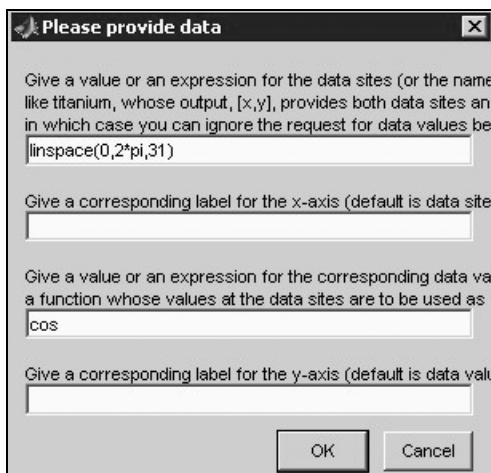
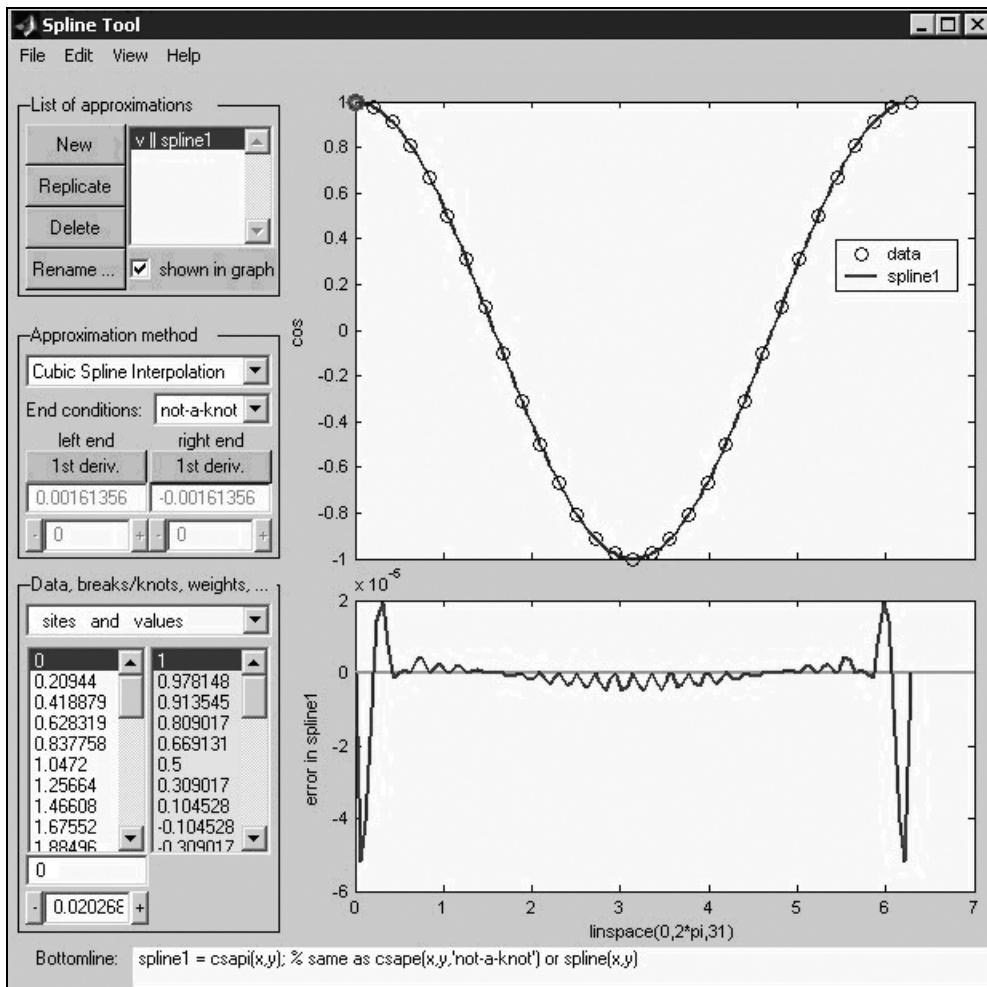


Рис. 18.14. Окно Please provide data



**Рис. 18.15.** Окно Spline Tool, демонстрирующее приближение функции  $\cos x$

На вспомогательном графике также могут быть построены первая и вторая производные сплайна. Выбор отображаемой информации на этом графике производится в меню View. По умолчанию осуществляется вывод графика ошибки (пункт Show Error). Для отображения графиков первой или второй производной следует выбрать, соответственно, пункты Show 1st Derivative или Show 2nd Derivative.

В расположенных слева от графиков полях предлагаются различные возможности для проведения экспериментов. Для кубического интерполяционного сплайна это эксперименты с изменением граничных условий. В поле

**Approximation method** находятся два списка: первый предлагает выбрать способ аппроксимации, а второй — задать граничные условия. В первом списке указывается тип сплайна, по умолчанию **Cubic Spline Interpolation** (Интерполяция кубическим сплайнам). Второй список **End conditions** служит для постановки граничных условий, по умолчанию это условия "отсутствия узла" (**not-a-knot**). График для этого случая уже выведен. Поле **Data**, **breaks/knots**, **weights** содержит данные и параметры приближения, которые могут быть изменены. В самой нижней части окна **Spline Tool** в строке **Bottomline** приведена запись функций Spline Toolbox, использовавшихся для построения сплайна. Можно вывести одновременно графики сплайнов с различными граничными условиями. Для создания нового сплайна и вывода его графика щелкните по кнопке **New** в поле **List of approximations**. Затем в поле **Approximation method** выберите из списка другой вид граничных условий **End conditions**, например, **second** (т. е. условия на вторые производные на концах). В областях ввода, расположенных под списком **End conditions**, задаются значения вторых производных на концах промежутка. В данном случае, поскольку функция  $\cos x$  задана аналитически, то вычисление вторых производных будет выполнено автоматически.

### Примечание

Для выбора порядка производной, значение которой необходимо задать, служат кнопки, расположенные над соответствующими областями ввода. Если на кнопке написано **2nd deriv.**, то нажатие на нее изменит надпись на **1st deriv.** и в отвечающую ей область ввода заносится значение первой производной.

Новый график сразу же выводится на экран. Текущий график обозначается жирной линией. Поэкспериментируйте самостоятельно с другими типами граничных условий, перечисленных в списке **End conditions**. Результаты работы можно вывести в отдельное графическое окно, выбрав в меню **File** пункт **Print to Figure**. Для экспорта исходных данных и результата аппроксимации в рабочую среду следует выбрать в меню **File** пункты **Export Data** и **Export Spline**. При выборе первого варианта открывается окно **Copy data to workspace**, а второго — **Copy spline to workspace**, в которых предлагается ввести имена для экспортируемых данных или согласиться с назначаемыми по умолчанию. Пункт **Save M-file** меню **File** предназначен для генерации М-файла с командами, которые реализуют создание присутствующих в данный момент на экране графиков, включая функции для построения соответствующих сплайнов.

Опишем приближение сплайнами произвольной функции, например,  $xe^{-x}$ , в приложении **splinetool**. Предварительно следует создать файл-функцию для вычисления исследуемой (листинг 18.2).

**Листинг 18.2. Файл-функция xexp**

```
function [y] = xexp(x)
y = x.*exp(-x)
```

После этого перейдите в основное окно **Spline Tool** приложения splinetool, выбрав в меню **File** пункт **Restart** и нажав **OK** в окне **Do you want to restart?**. В основном окне нажмите кнопку **Provide your own data or an m-file that does** и в появившемся окне **Please provide data** в первой строке ввода задайте вектор координат точек разрыва  $0:0.2:4$ , а в третьей — имя файл-функции **xexp** (рис. 18.16).

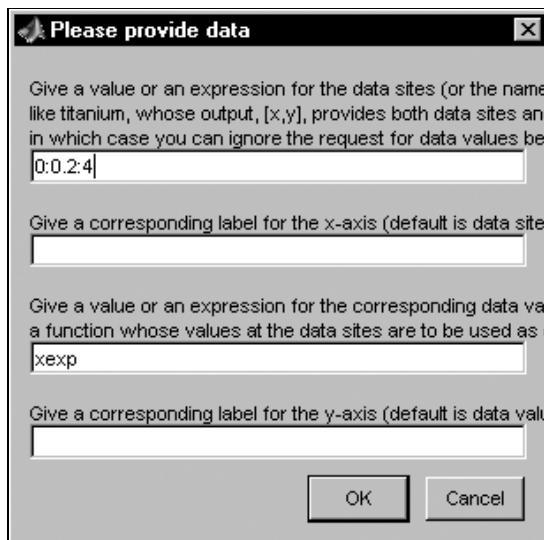


Рис. 18.16. Занесение данных в окно **Please provide data**

Результаты приближения приведены на рис. 18.17.

**Примечание**

Вид основного окна **Spline Tool** с результатами приближения функций приведен по версии Spline Toolbox 3.1.1, входящей в релиз R13, поскольку в период работы над книгой доступный релиз R14 с версией 3.2.1 содержал ошибку, связанную с построением графика погрешности для интерполяционного сплайна.

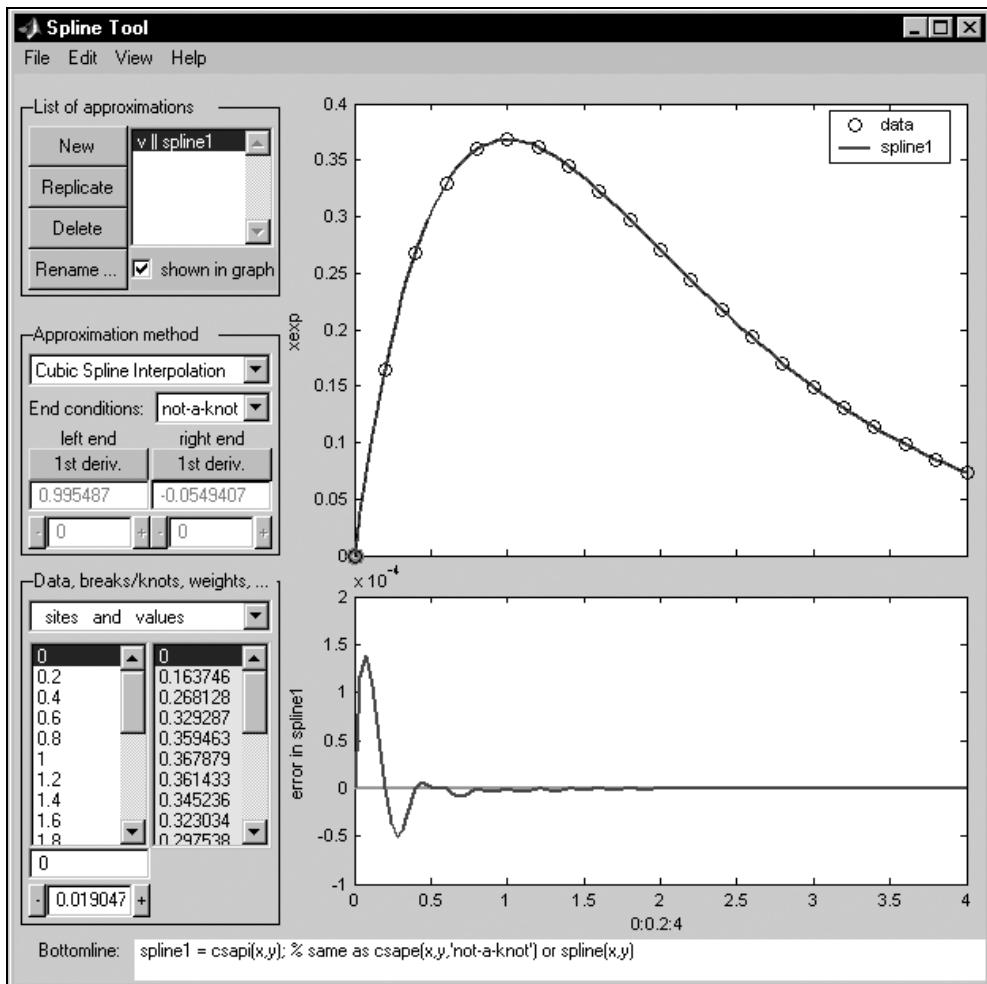


Рис. 18.17. Приближение кубическим сплайном функции  $x \cdot e^{-x}$

## Сплайны для поверхностей

В Spline Toolbox реализована работа с функциями многих переменных. При этом строится тензорное произведение сплайнов одной переменной. Для функций многих переменных реализованы обе формы представления — *pp*-форма и *B*-форма. Все функции, использовавшиеся для построения сплайнов, могут работать с сеточными данными любого количества переменных.

Поясним применение тензорного произведения на примере *pp*-формы для функции двух переменных. Пусть дана сеточная функция, определенная на прямоугольной области с сеткой узлов  $(x_i, y_k)$ , где  $i=1, \dots, n_1$ ,  $k=1, \dots, n_2$ .

Сеточная функция для  $z(x, y)$  задана двумерным массивом  $z_{i,k}$ . Тогда двумерный сплайн в виде тензорного произведения строится по формуле:

$$S(x, y) = g(x) \otimes h(y) = \sum_{i=0}^{k_1} \sum_{l=0}^{k_2} c_{i,l}^{(j,k)} (y - y_k)^l (x - x_j)^i, \quad x \in [x_j, x_{j+1}], \quad y \in [y_k, y_{k+1}],$$

где  $g(x)$  и  $h(y)$  — сплайн-функции одной переменной, которые могут отличаться порядком и количеством определяющих их промежутков. Коэффициенты  $c_{i,l}^{(j,k)}$  находятся либо из условий интерполяции  $S(x_i, y_k) = z_{i,k}$  ( $i=1, \dots, n_1$ ,  $k=1, \dots, n_2$ ) и требуемой гладкости, либо из условий сглаживания, аналогичных одномерному случаю.

В общем случае требуется аппроксимировать сеточную функцию *m*-переменных  $F(x_1, x_m)$ . Сохраним ту же нотацию для обязательных входных параметров функций Spline Toolbox, обозначая  $x$  — узлы сетки, а  $y$  — значения сеточной функции. В многомерном случае массив  $x = \{x_1, x_2, \dots, x_m\}$ , определяющий сетку, должен быть массивом ячеек, каждая ячейка которого содержит вектор с координатами узлов сетки по соответствующему направлению. Работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8.

Если массивы  $x_1, x_2, \dots, x_m$  имеют длины  $n_1, n_2, \dots, n_m$ , то в случае скалярной табличной функции массив  $y$  должен быть *m*-мерным с размерами  $n_1 \times n_2 \times \dots \times n_m$ . При аппроксимации вектор-функции с  $d$  компонентами массив  $y$  с табличными значениями —  $(m+1)$ -мерный размера  $d \times n_1 \times n_2 \times \dots \times n_m$ . В целом, приближение многомерных данных осуществляется аналогично одномерному случаю. В листинге 18.3 приведен пример файл-функции *surf\_pp* с подфункцией *rung*, в которой происходит обращение к *csapi*. Ее входные параметры сформированы с учетом приведенного выше правила.

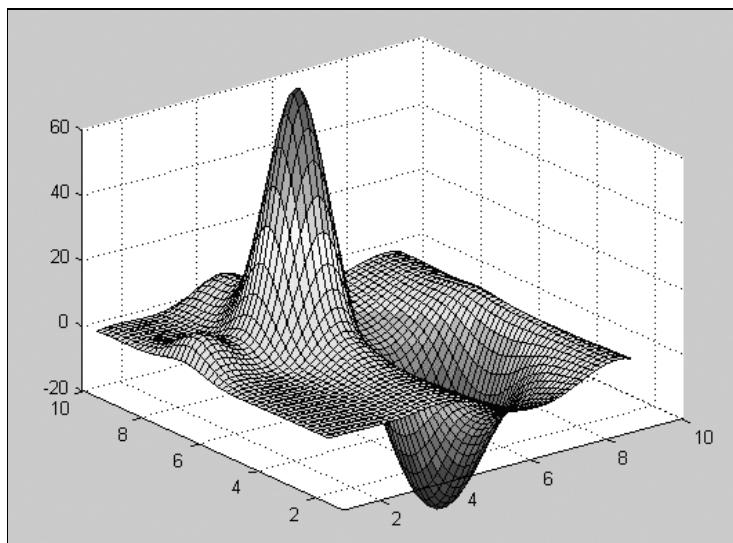
### Листинг 18.3. Файл-функция *surf\_pp* построения сплайна для поверхности

```
function surf_pp
% Построение поверхности при помощи сплайна
```

```
% Задание координат узлов сетки
x1 = 1:1:9;
x2 = 1.5:1:9.5;
% Вычисление значений функции в узлах
y = rung(80, x1, x2);
% Конструирование сплайна
bcs = csapi([x1, x2], y);
% Визуализация поверхности
figure
fnplt(bcs)
axis([1 10 1 10 -30 80]);

function u = rung(a, x, y)
% Подфункция для вычисления значений функции двух переменных
v1 = a*(1./(1 + 2.5*(x - 4).^2))'* (1./(1 + 1.5*(y - 6.5).^2));
v2 = -a*(1./(1 + 2.5*(x - 6.5).^2))'* (1./(1 + 0.5*(y - 5).^2));
u = v1 + v2;
```

Выполнение файл-функции `surf_pp` приводит к результату, представленному на рис. 18.18.



**Рис. 18.18.** Построение поверхности функцией `csapi`

Для получения  $B$ -формы тензорного произведения интерполяционных сплайнов предназначена функция `spapi`. Возможны два способа вызова функции:

```
s = spapi({k1, k2, ..., km}, x, y).
```

или

```
s = spapi({knot1, knot2, ..., knotm}, x, y).
```

Входные аргументы  $x$  и  $y$  задаются так, как описано выше в этом разделе. При первом способе обращения каждый из элементов  $k_j$  массива ячеек  $\{k_1, k_2, \dots, k_m\}$  определяет порядок сплайна по переменной  $x_j$ , и функция `spapi` автоматически строит подходящую последовательность узлов. При втором способе вызова предусматривается указание последовательности узлов  $knot_j$  по каждой переменной  $x_j$ , передаваемых в массиве ячеек  $\{knot_1, knot_2, \dots, knot_m\}$ .

Пример использования `spapi` для построения сплайнов разных порядков для аппроксимации функции двух переменных приведен в листинге 18.4. Достаточно редкая сетка выбрана для наглядной демонстрации зависимости гладкости поверхности от порядка сплайна (рис. 18.19).

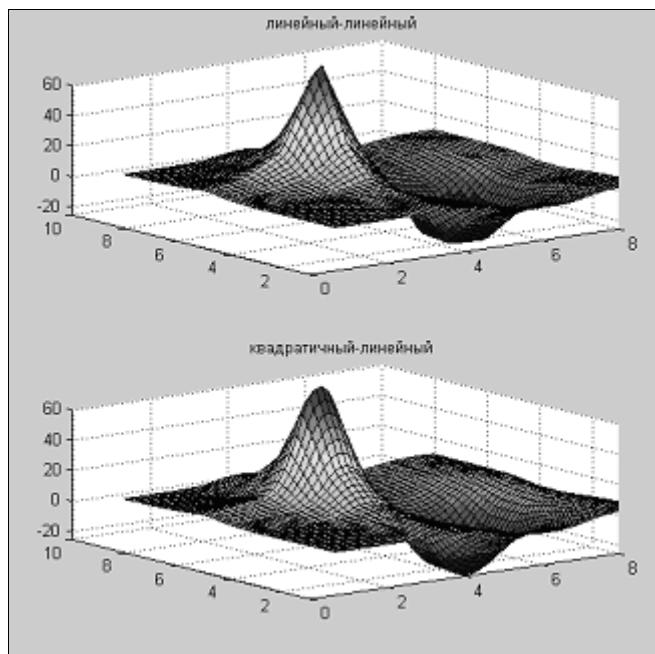
#### Листинг 18.4. Файл-функция `surf_B` для построения поверхности функцией `spapi`

```
function surf_B
% Демонстрация использования функции spapi

% Создание координат узлов сетки
x1 = 1:1:9;
x2 = 1.5:1:9.5;
% Вычисление функции в узлах
y = rung(80, x1, x2);
% Построение сплайна, линейного по каждой переменной
sp1 = spapi({2, 2}, {x1, x2}, y);
% Вывод графика поверхности
figure
subplot(2, 1, 1)
fnplt(sp1)
axis([0 8 1 10 -25 50])
title('линейный-линейный')
% Построение сплайна, квадратичного по x и линейного по y
sp2 = spapi({3, 2}, {x1, x2}, y);
```

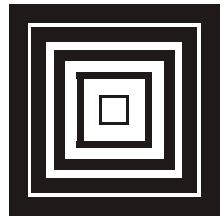
```
% Вывод графика поверхности
subplot(2, 1, 2)
fnplt(sp2)
axis([0 8 1 10 -25 50])
title('квадратичный-линейный')

function u = rung(a, x, y)
% Подфункция для вычисления значений функции двух переменных
v1 = a*(1./(1 + 2.5*(x - 4).^2))'* (1./(1 + 1.5*(y - 6.5).^2));
v2 = -a*(1./(1 + 2.5*(x - 6.5).^2))'* (1./(1 + 0.5*(y - 5).^2));
u = v1 + v2;
```



**Рис. 18.19.** Приближение поверхности сплайнами разных порядков функцией spapi

Возможности Spline Toolbox не ограничиваются рассмотренными в данной главе функциями и типами сплайнов. В частности, информация о рациональных сплайнах и сплайнах в *st*-форме доступна в справочной системе MATLAB по Toolbox (см. разд. **Spline Toolbox: Tutorial: NURBS and other rational splines** и **Spline Toolbox: Tutorial: The stform**, а также **Spline Toolbox: Functions – Categorical List: Construction of Splines**).



## Глава 19

# Приближение данных и подбор параметров в Curve Fitting Toolbox

При чтении предыдущих глав вы неоднократно встречали описание различных способов приближения функций и табличных данных как стандартными функциями MATLAB (`polyfit`, `interp1` и др.), так и средствами специализированных Toolbox. Предыдущая глава была посвящена Spline Toolbox, включающему набор функций и приложения с графическим интерфейсом для приближения сплайнами и оценки качества полученного результата. Часть функций Optimization Toolbox также позволяют решать задачи, связанные с аппроксимацией данных, например, нелинейную задачу подбора параметров (см. разд. "Подбор параметров" главы 16).

Часто процесс приближения состоит из нескольких этапов: предварительной обработки данных, выбора одного или нескольких методов приближения и анализа полученных результатов. Эта работа может быть проведена в Curve Fitting Toolbox, содержащем приложение с графическим интерфейсом Curve Fitting Tool и набор функций. Приложение Curve Fitting Tool позволяет:

- применить регрессионный анализ и другие методы для начальной фильтрации значений табличной функции, задавать весовые значения в узлах табличных данных;
- выбрать различные способы приближения, такие как: параметрические, связанные с подбором параметров моделей (одной из стандартных, например, полиномиальной, или определение собственной модели), сглаживающие и обычные сплайны;
- провести анализ результатов, включающий экстраполяцию, интегрирование и дифференцирование.

Выполнение каждого этапа сопровождается визуализацией результата. Кроме того, имеется возможность сгенерировать файл-функцию, содержащую обращения к функциям Toolbox, и использовать ее как независимое приложение или как часть собственного.

Мы рассмотрим работу с приложением Curve Fitting Tool, после освоения которого использование функций Toolbox не составит затруднений. Начнем изучение средств Toolbox с загрузки массивов данных для обработки, которые содержатся на компакт-диске в файле set19.mat. Скопируйте его в рабочий каталог и воспользуйтесь командой `load`

```
>> load set19.mat
```

либо соответствующими инструментами рабочей среды для чтения данных (способы считывания и записи двоичных данных разобраны в главе I).

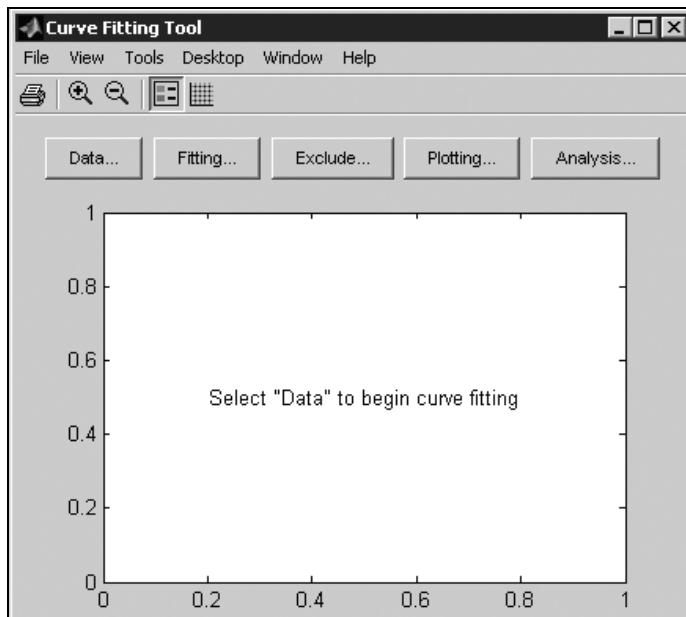
В рабочей среде должны появиться массивы `f1`, `f2`, `f3`, `f4` и `x`.

Естественно, если данные были сформированы в течение сеанса работы в MATLAB и хранятся в переменных рабочей среды, то предварительного считывания не требуется — они могут быть сразу обработаны средствами Curve Fitting Toolbox.

## Приложение Curve Fitting Tool и его средства

Приложение Curve Fitting Tool запускается командой `cftool`:

```
>> cftool
```



**Рис. 19.1.** Основное окно приложения Curve Fitting Tool

В результате открывается диалоговое окно **Curve Fitting Tool**, представленное на рис. 19.1.

Можно обратиться к функции `cftool`, задав в качестве параметров массивы узлов и значений функции, которую требуется аппроксимировать подбором параметров:

```
>> cftool(массив_узлов, массив_узловых_значений_функции)
```

В этом случае при открытии окна **Curve Fitting Tool** данные загружаются автоматически.

Набор кнопок, расположенных под панелью инструментов, позволяет выбрать один из инструментов Curve Fitting Toolbox. Нажатие на каждую кнопку приводит к появлению одноименного диалогового окна:

- Data** — для создания одного или нескольких множеств данных на основе переменных рабочей среды и их предварительной обработки до анализа;
- Fitting** — для выбора метода подбора параметров;
- Exclude** — для исключения части исходных данных из анализа;
- Plotting** — для указания графиков, выводимых в окне **Curve Fitting Tool**;
- Analysis** — для анализа результатов подбора параметров данных, включающего: экстраполяцию, интегрирование и дифференцирование.

Сеанс работы с приложением Curve Fitting Tool называется сессией. Сохранение, загрузка или начало новой сессии выполняются в меню **File**. Для сохранения результатов служит пункт **Save Session**, загрузки — **Load Session**. Если вы начинаете новую сессию, выбрав пункт **Clear Session**, то появляется запрос для подтверждения или отказа сохранения текущей сессии.

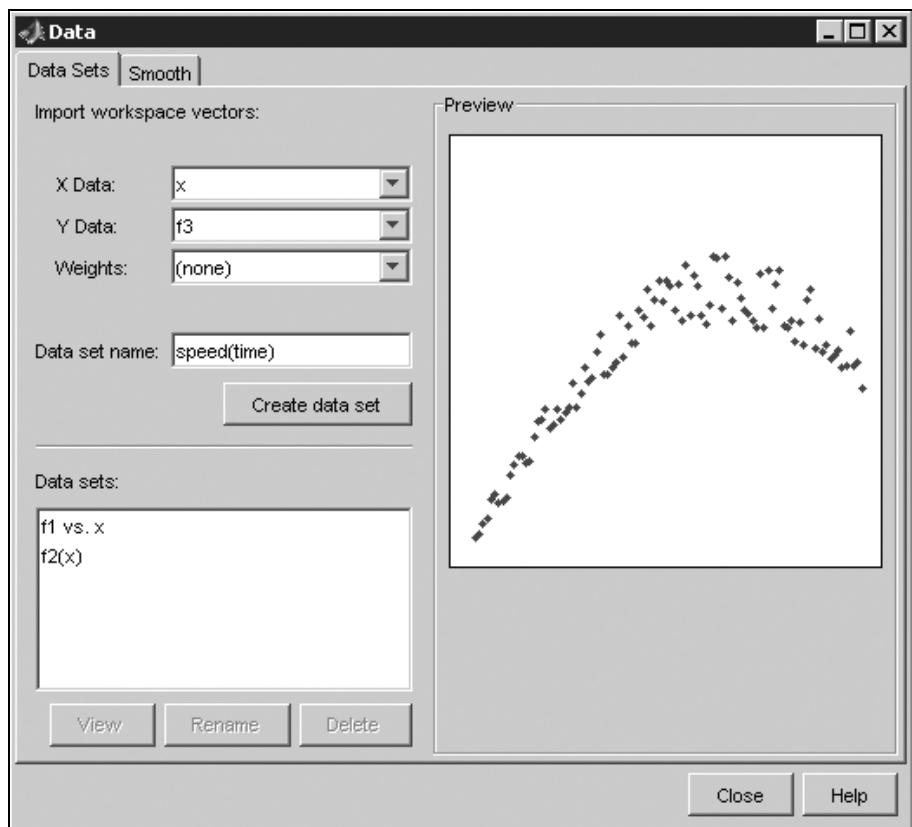
### Примечание

Для иллюстрации отдельных аспектов работы в Curve Fitting Tool мы будем начинать новую сессию, используя одни и те же массивы данных рабочей среды.

## Создание множества данных для приближения

Curve Fitting Toolbox оперирует с элементом данных, называемым множеством и содержащим значения табличной функции, узлов и весовых коэффициентов. Если весовые коэффициенты не заданы, они полагаются равными единице. Каждое множество снабжается именем. Для определения множества следует перейти к диалоговому окну **Data**, нажав одноименную кнопку в

основном окне **Curve Fitting Tool**. Вы получаете возможность определить данные для приближения (предполагаем, что в рабочую среду загружены массивы из файла set19.mat — массив узлов  $x$  и массивы табличных функций  $f_1, f_2, f_3, f_4$ ). В раскрывающихся списках **X Data**, **Y Data** и **Weights** окна **Data**, содержащих переменные рабочей среды, выбираются массивы данных (весовые коэффициенты можно не задавать). При этом в области **Preview** отображается выбранная табличная функция (рис. 19.2). До момента создания множества при помощи кнопки **Create data set** можно изменить массивы данных или стандартно генерируемое имя множества в поле **Data set name**.

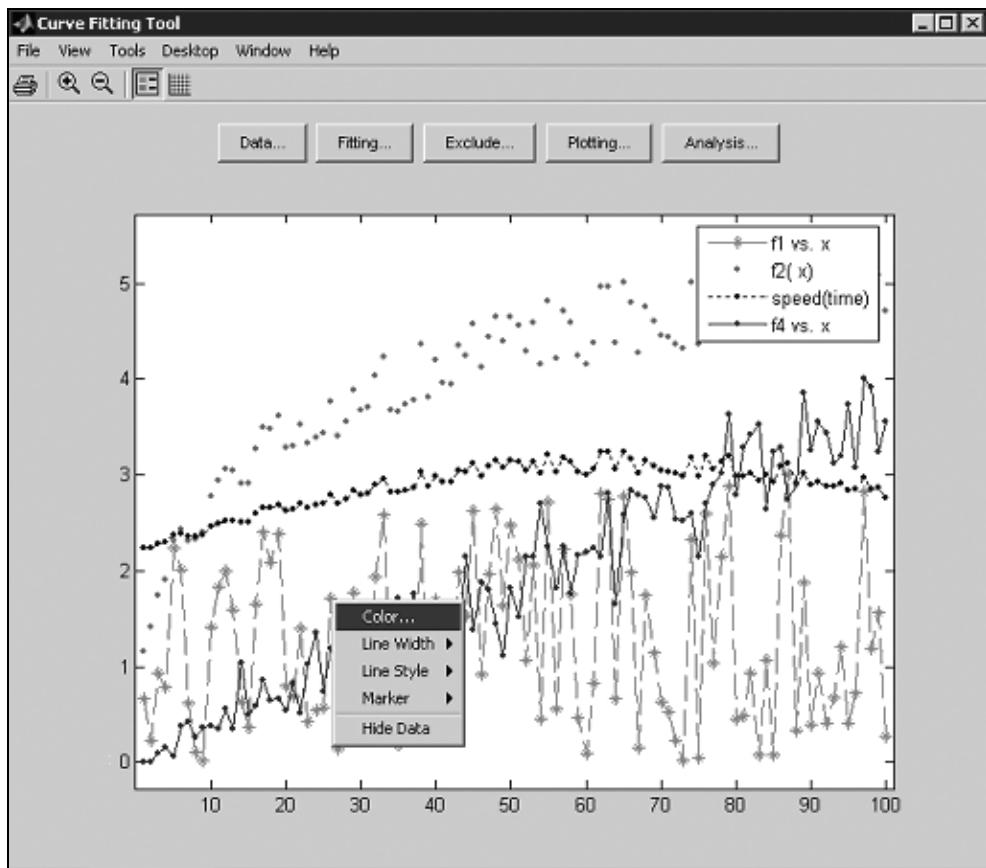


**Рис. 19.2.** Диалоговое окно **Data** для определения множества

После создания множеств их имена заносятся в поле **Data sets** и при выборе одного из них становятся доступны кнопки для работы с этими множествами: **View** (Графическое представление), **Rename** (Переименование), **Delete** (Удаление).

Создайте четыре множества из табличных функций, образованных парами  $(x, f1)$ ,  $(x, f2)$ ,  $(x, f3)$ ,  $(x, f4)$ , и дайте им, соответственно, имена  $f1$  vs.  $x$ ,  $f2(x)$ ,  $speed(time)$  и  $f4$  vs.  $x$ . Эти множества понадобятся нам для приближения и последующего анализа.

После закрытия диалогового окна **Data** в основном окне **Curve Fitting Tool**, приведенном на рис. 19.3, отображаются графики всех определенных множеств. Первоначально зависимости изображаются только точками, как для множества  $f2(x)$ . Использование контекстного меню (рис. 19.3) позволяет изменить стиль графиков данных, как это сделано для множеств  $f1$  vs.  $x$ ,  $speed(time)$  и  $f4$  vs.  $x$ .



**Рис. 19.3.** Диалоговое окно **Curve Fitting Tool**  
после определения множеств и контекстное меню  
изменения свойств графика

## Предварительная обработка данных

До начала приближения табличной функции иногда целесообразно предварительно обработать исходные данные. Приложение Curve Fitting Tool предоставляет две возможности: исключить часть табличных значений и произвести сглаживание табличной функции. Рассмотрим каждую из них.

### Исключение данных из таблицы

Для каждого множества можно сформулировать несколько правил. Каждое из них включает либо логические условия для исключения точек из таблицы, либо в таблице или на графике просто отмечаются те точки, которые не должны учитываться при приближении табличной функции. Каждому правилу присваивается идентификатор, по которому на него делается ссылка. Для доступа к диалогу формулирования правил исключения точек из таблицы значений в основном окне **Curve Fitting Tool** нажмите кнопку **Exclude**.

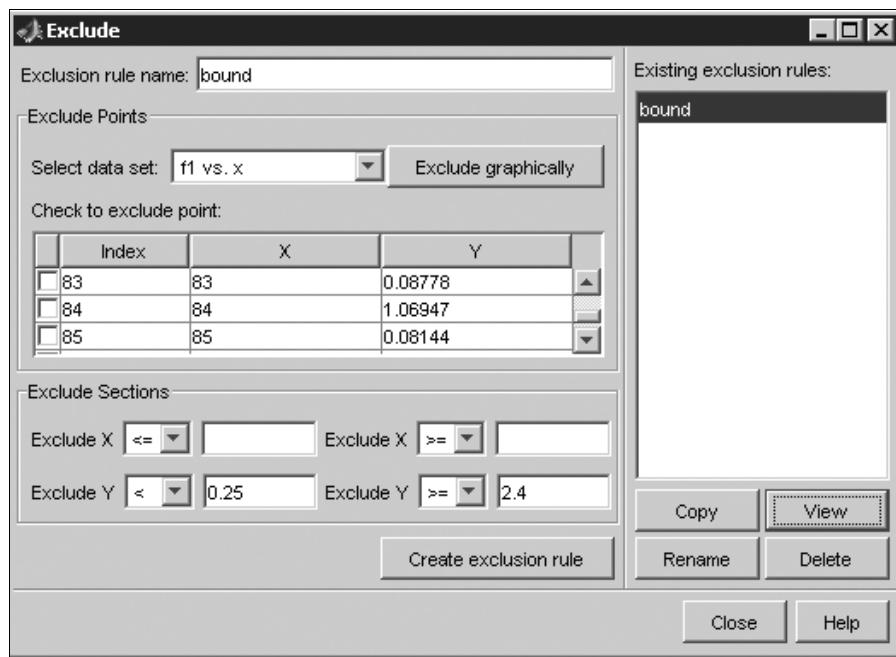


Рис. 19.4. Диалоговое окно **Exclude**  
для предварительной обработки данных

Рассмотрим пример для множества  $f1$  vs.  $x$ , созданного в предыдущем разделе. В диалоговом окне **Exclude** в области ввода **Exclusion rule name** задайте имя **bound**, правилу в списке **Select data set** выберите множество  $f1$  vs.  $x$  (рис. 19.4). Исключим, к примеру, из таблицы те значения, которые выходят за полуинтервал  $[0.25, 2.4]$ . Для этого в полях ввода **Exclude X** и **Exclude Y** определите верхнюю и нижнюю границы значений, как для узлов, так и для узловых значений табличной функции. Кроме этого, можно исключить отдельные точки вручную, отметив их в таблице **Check to exclude point**, или воспользоваться кнопкой **Exclude graphically** для выбора исключаемых точек на графике в появляющемся окне. Для завершения создания правила следует нажать кнопку **Create exclusion rule**. Имена определенных правил помещаются в окно **Exiting exclusion rules**.

Для просмотра некоторого правила надо выделить его имя в окне **Exiting exclusion rules** и нажать кнопку **View**. Обратитесь к только что созданному правилу **bound**. Появится окно просмотра правила, изображенное на рис. 19.5. Цветом выделены отбракованные точки.

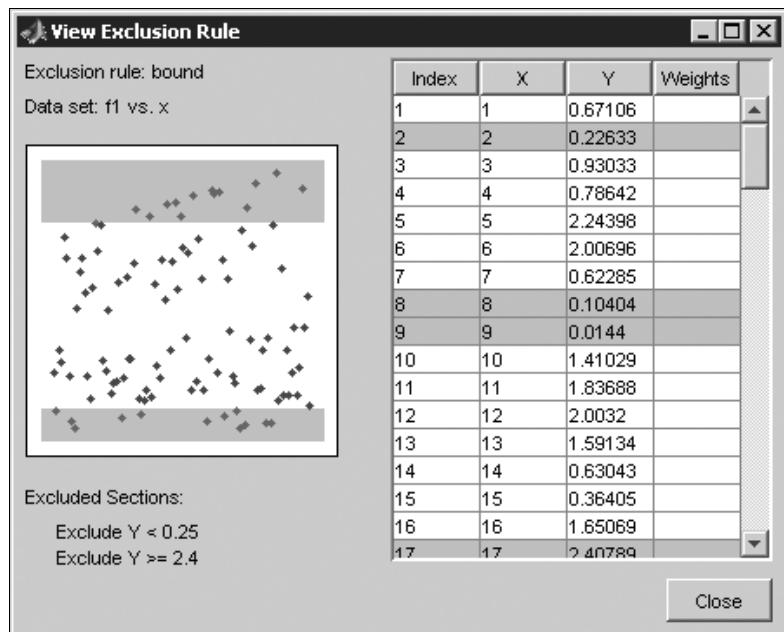


Рис. 19.5. Просмотр правила **bound**

Для одного множества можно сформировать несколько правил и использовать их при дальнейшей работе.

## Начальная фильтрация табличной функции

Несмотря на то, что это средство относится к предварительной обработке данных перед подбором параметров, его можно использовать и независимо для получения требуемого результата. Например, построение скользящей средней для табличной функции может рассматриваться как конечная цель приближения (характерно, в частности, для изучения динамики цен).

Для выполнения предварительного сглаживания (будем называть этот процесс фильтрацией, поскольку на этапе приближения также используются методы сглаживания) следует перейти в окне **Data** на вкладку **Smooth** (рис. 19.6).

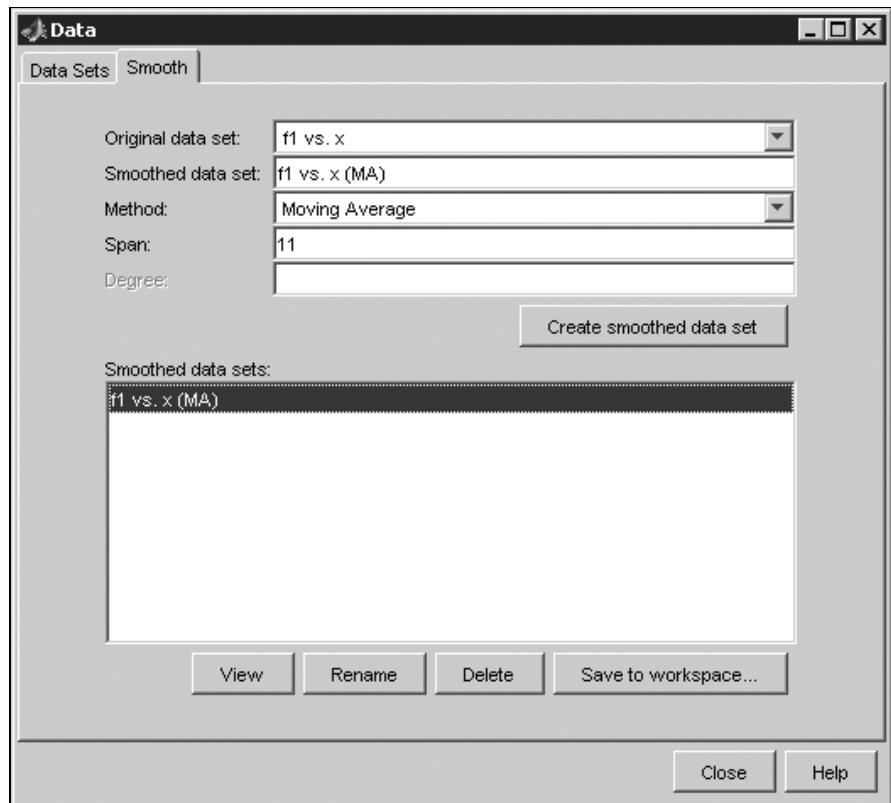


Рис. 19.6. Диалоговое окно **Data** для фильтрации табличных функций

В раскрывающемся списке **Original data set** следует выбрать имя ранее определенного множества, в поле ввода **Smoothed data set** можно изменить гене-

рированное имя для обработанного множества данных, которое будет получено в результате выполнения выбранной процедуры. Список **Method** позволяет указать метод фильтрации, а поля **Span** и **Degree** предназначены для ввода параметров, используемых в методах.

В приведенном на рис. 19.6 варианте создано одно дополнительное множество с именем `f1 vs. x(MA)`. Полученное множество имеет тот же статус, что и созданные ранее, поэтому для него строится график в основном окне приложения Curve Fitting Tool. Применение каждого метода фильтрации или одного с использованием разных наборов параметров сопровождается выводом соответствующего графика. Выбрав одно или несколько множеств для дальнейшего анализа, ненужные можно удалить.

Для предварительной фильтрации данных доступны следующие методы.

- **Moving Average** — метод скользящей средней: значения отфильтрованной табличной функции вычисляются как среднее арифметическое соседних точек, т. е. используется формула:

$$f_K = \frac{1}{2 \cdot n + 1} \sum_{i=K-n}^{K+n} y_i,$$

где величина  $n$  (должна быть нечетным числом) является параметром усреднения и задается в поле **Span**. Для граничных точек усреднение не производится, а для примыкающих к границам параметр усреднения берется наибольшим допустимым.

- **Lowess (linear fit)** — метод основан на линейном регрессионном анализе. На локальном промежутке, определяемом в поле **Span** параметром  $n$  аналогично со сглаживанием, для каждой точки вычисляется вес по формуле:

$$w_i = \left( 1 - \left| \frac{y_k - y_i}{\max_j |x_k - x_j|} \right|^3 \right)^3$$

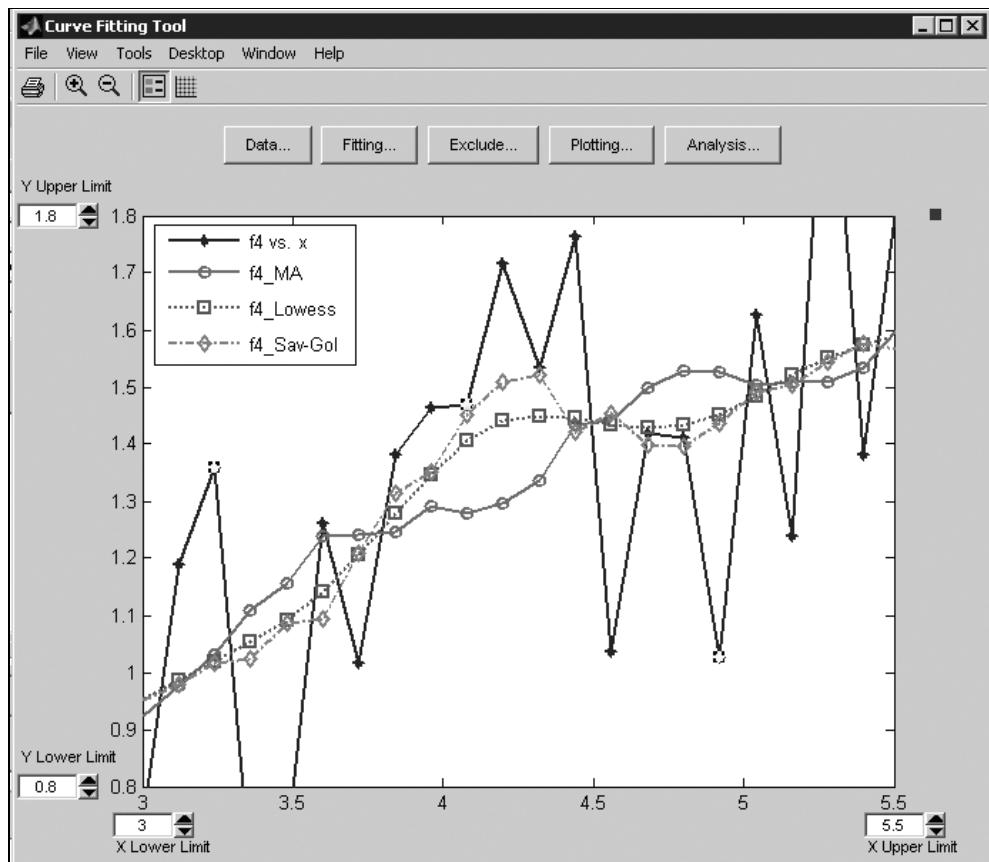
и строится полином первого порядка с применением техники линейного регрессионного анализа. В отличие от предыдущего метода для точек, прилежащих к граничным, параметр усреднения  $n$  не меняется.

- **Loess (quadratic fit)** — метод также использует технику линейного регрессионного анализа. Отличие от **Lowess** в том, что строится полином второго порядка.
- **Robust Lowess (linear fit)** и **Robust Loess (quadratic fit)** — методы робастной фильтрации. В отличие от базовых методов **Lowess** и **Loess**, они состоят из двух этапов: первый такой же, как в базовом методе. Затем в ка-

жной точке вычисляются компоненты вектора невязки  $r_i$  и находятся новые весовые коэффициенты:

$$w_i = \begin{cases} 0, & |r_i| \geq 6 \cdot M; \\ \left(1 - \left|\frac{r_i}{6 \cdot M}\right|^2\right)^2, & |r_i| < 6 \cdot M; \end{cases} \quad M = \text{median}(|r|).$$

После чего снова строится сглаживающий полином с применением техники линейного регрессионного анализа.



**Рис. 19.7.** Результаты фильтрации данных для табличной функции  $f_2$

□ **Savitzky-Golay** — метод фильтрации Савицкого—Голая. Для интервала усреднения, определяемого параметром  $n$  (нечетное число), методом наименьших квадратов без взвешивания строится сглаживающий полином степени, меньшей  $n$  (задается в поле **Degree**). Значение в центральной точке принимается за значение сглаженной табличной функции.

Приведенная характеристика методов фильтрации дает самое общее представление о них. Для понимания их возможностей и границ применимости следует ознакомиться со специальной литературой.

На рис 19.7 приведены примеры некоторых методов фильтрации с одинаковым параметром  $n=15$  для множества  $f4$  vs.  $x$ . В методе Савицкого—Голая используется полином второй степени. Для более наглядного различия результатов фильтрации представлена только часть данных из отрезка [3, 5.5]. Установка в меню **Tools** флага **Axis Limit Control** приводит к появлению списков для изменения пределов координатных осей и отображения точных их значений, что полезно при изменении масштаба инструментами **Zoom in** и **Zoom out**. Пункт **Default Axis Limit** меню **Tools** позволяет вернуться к исходным пределам при условии, что не один из инструментов **Zoom in** или **Zoom out** не выбран.

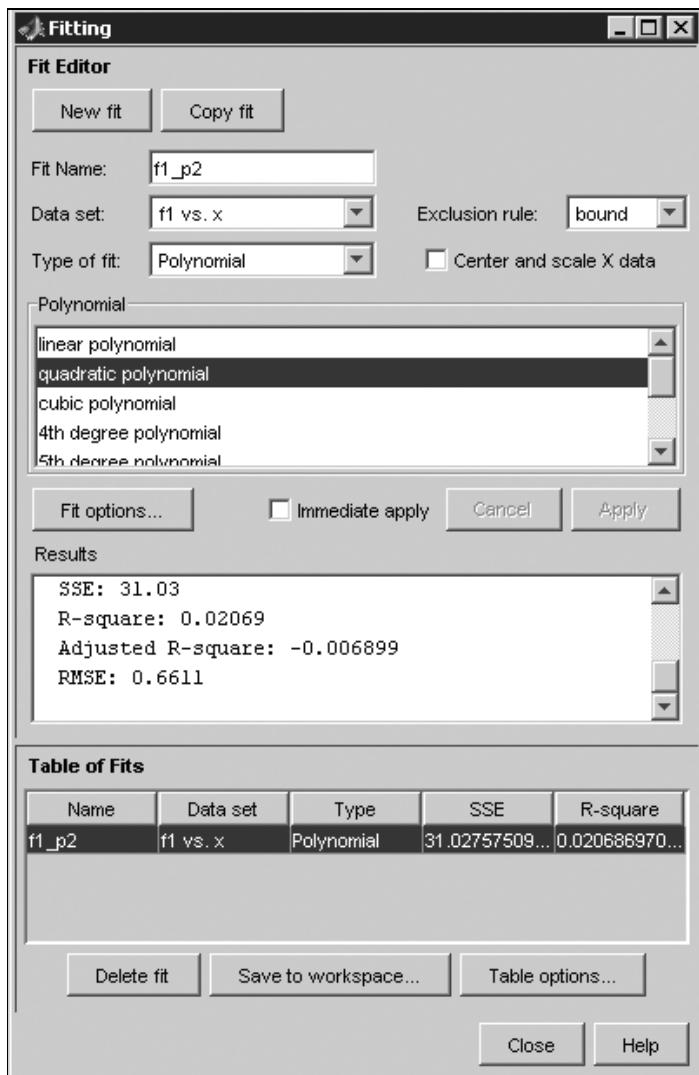
## Приближение табличных функций

Как мы уже упоминали, приложение Curve Fitting Tool позволяет использовать различные способы приближения данных — как параметрические, так и нет. Выбор любого из них (кроме интерполяции обычными сплайнами) означает решение соответствующей задачи оптимизации, поскольку алгоритмы аппроксимации данных Curve Fitting Toolbox основаны на методе наименьших квадратов. Имеется несколько возможностей: от простейшей минимизации суммы квадратов невязок и взвешенных невязок до выбора алгоритма, устойчивого по отношению к "выбросам" в данных. При этом доступны несколько методов решения возникающей оптимационной задачи, указание желаемой точности, границ интервалов для нахождения параметров и ряда других опций. Качество полученных приближений оценивается как визуально, так и по некоторым критериям, проверяемым средствами приложения Curve Fitting Tool. В этом разделе разбираются все эти вопросы.

## Создание приближений

Для перехода к приближению табличных функций следует в основном окне приложения Curve Fitting Tool нажать кнопку **Fitting**. Появляется одно-

именное диалоговое окно для управления процессом приближения данных (рис. 19.8), в котором для начала процесса следует нажать кнопку **New fit**, определить имя нового приближения и ввести его в поле **Fit Name** вместо предлагаемого по умолчанию (мы будем указывать имя табличной функции и аббревиатуру используемого метода).



**Рис. 19.8.** Диалоговое окно **Fitting** для подбора параметров

В раскрывающемся списке **Data set** выбирается определенное ранее множество, а тип приближения для него — в списке **Type of fit**. Имеется несколько возможностей: выбор одной из предопределенных параметрических моделей, в том числе линейной и отношением полиномов (искомые параметры являются коэффициентами полиномов), задание собственной модели и интерполяция сплайнами. Далее в этом разделе дана краткая характеристика типов реализованных приближений.

Выбор типа в списке **Type of fit** определяет содержимое поля, расположенного ниже и принимающего имя типа приближения. В этом поле могут отображаться: список предопределенных моделей, степень полинома или тип сплайна или средства для создания собственной модели. До выполнения приближения в списке **Exclusion rule** можно указать одно из определенных ранее правил для исключения части данных (см. разд. "Исключение данных из таблицы" данной главы).

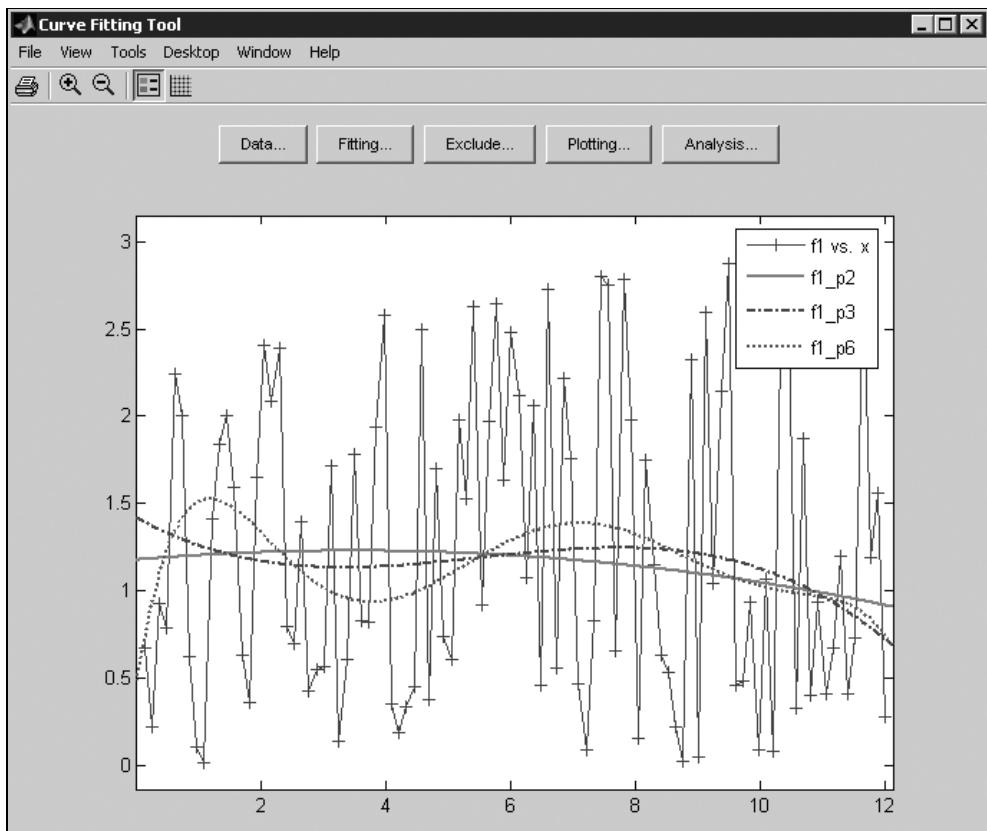


Рис. 19.9. Окно приложения Curve Fitting Tool после построения приближений

Щелчок по кнопке **Apply** приводит к вызову процедуры приближения и выводу результата в окно **Results**.

### Примечание

При установленном флаге **Immediate apply** вычисления начинаются автоматически после выбора модели.

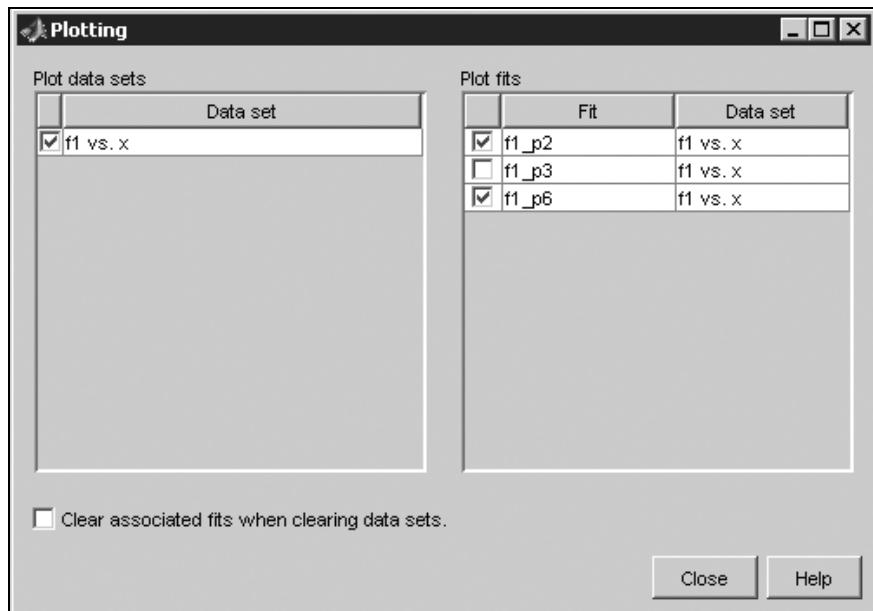
При чтении предыдущих разделов вы создали множество  $f1 \text{ vs. } x$  для загруженной табличной функции  $f1$  и правило  $\text{bound}$  для исключения части табличных данных. Аппроксимируйте  $f1$  полиномами второй, третьей и шестой степеней по методу наименьших квадратов (далее мы обсудим другие типы приближений). При создании каждого следующего приближения нажмите кнопку **New fit**, иначе текущее приближение заменится новым. Дайте имена приближениям в соответствии со степенью аппроксимирующего полинома  $f1_p2$ ,  $f1_p3$ ,  $f1_p6$ . Процесс приближения сопровождается выводом результатов в поле **Result** (рис. 19.8) и добавлением графиков приближений в окно **Curve Fitting Tool**. На рис. 19.9 приведены графики после изменения стилей линий из контекстного меню. Результаты вычислений содержат различные критерии, характеризующие качество приближения. В начале окна выводятся аналитический вид приближения и найденные значения параметров, включая границы доверительного интервала (этот вопрос кратко обсуждается в следующем разделе).

## Контроль качества приближений

Приложение Curve Fitting Tool предоставляет возможность оценить приближения с использованием графических средств. Для управления отображаемыми в основном окне **Toolbox** объектами следует воспользоваться кнопкой **Plotting**. В открывшемся окне (рис. 19.10) графики разбиты на две группы. В левой части перечислены графики множеств, а в правой — построенных приближений. В нашем случае имеется одно множество и три приближения. Сброс флага с левой стороны от имени множества или приближения позволяет скрыть соответствующий график. На рис. 19.10 представлено окно **Plotting**, в котором определено, что следует скрыть отображение приближения  $f1_p3$ . При необходимости восстановить изображение графика флаг надо установить.

### Примечание

Для скрытия графика можно воспользоваться контекстным меню в основном окне **Curve Fitting Tool** (см. рис. 19.3), однако восстановить изображение можно только в диалоговом окне **Plotting**.



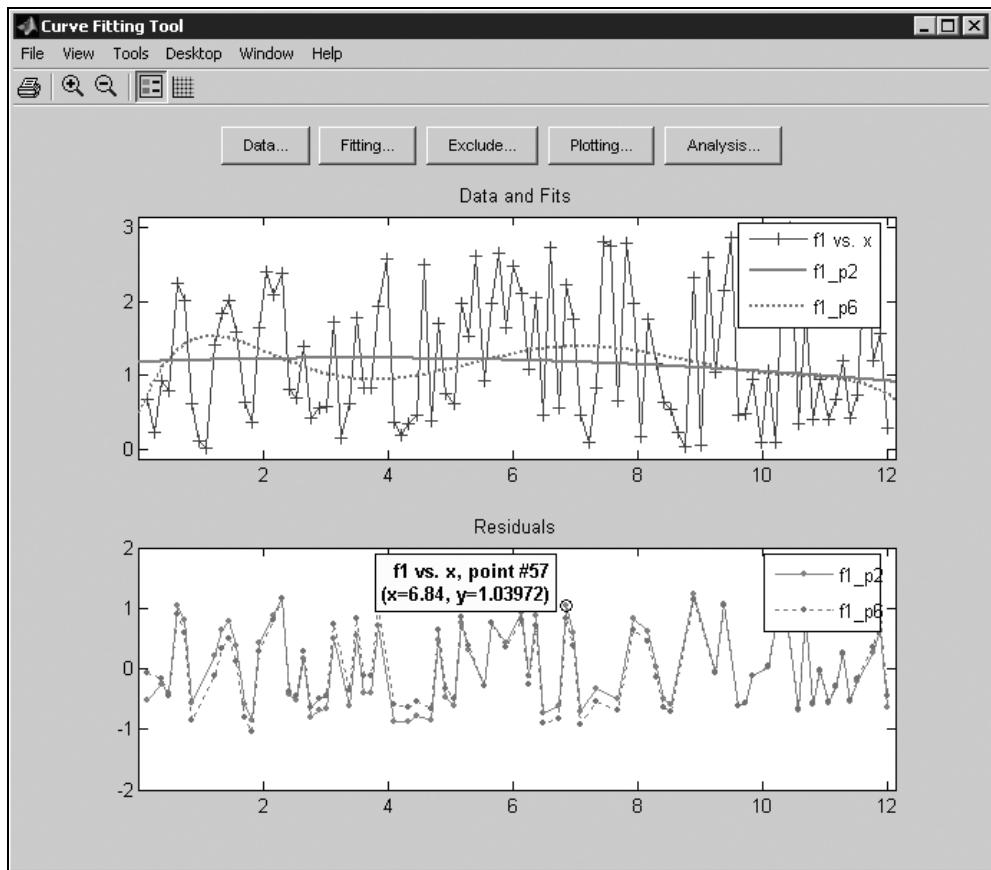
**Рис. 19.10.** Диалоговое окно Plotting

Установка флага **Clear associated fits when clearing data sets** запрещает выводить графики приближений, если сбрасывается флаг для отображения соответствующего множества, т. е. автоматически сбрасываются флаги у всех приближений для этого множества.

Оставьте два приближения  $f_1_{\_p2}$  и  $f_1_{\_p6}$  и проанализируйте невязки приближений. Для вывода графика невязки перейдите в основное окно **Curve Fitting Tool** и в меню **View** в пункте **Residuals** выберите **Line Plot**. При этом в окне приложения размещаются новые оси, содержащие графики невязки (рис. 19.11). При наведении мыши на точку графика и нажатии левой кнопки появляется информация об исходных данных (номер точки и ее координаты).

При нахождении параметров определяются также интервалы, которым они принадлежат с некоторой доверительной вероятностью, равной по умолчанию 95%. Значения доверительной вероятности выбираются в меню **View** в пункте **Confidence Level** окна приложения Curve Fitting Tools. Пункт **Prediction Bounds** служит для отображения доверительной полосы.

На рис. 19.12 доверительная полоса приведена для наглядности при доверительной вероятности 80%.



**Рис. 19.11.** Окно приложения **Curve Fitting Tool**  
с графиком ошибок для приближений

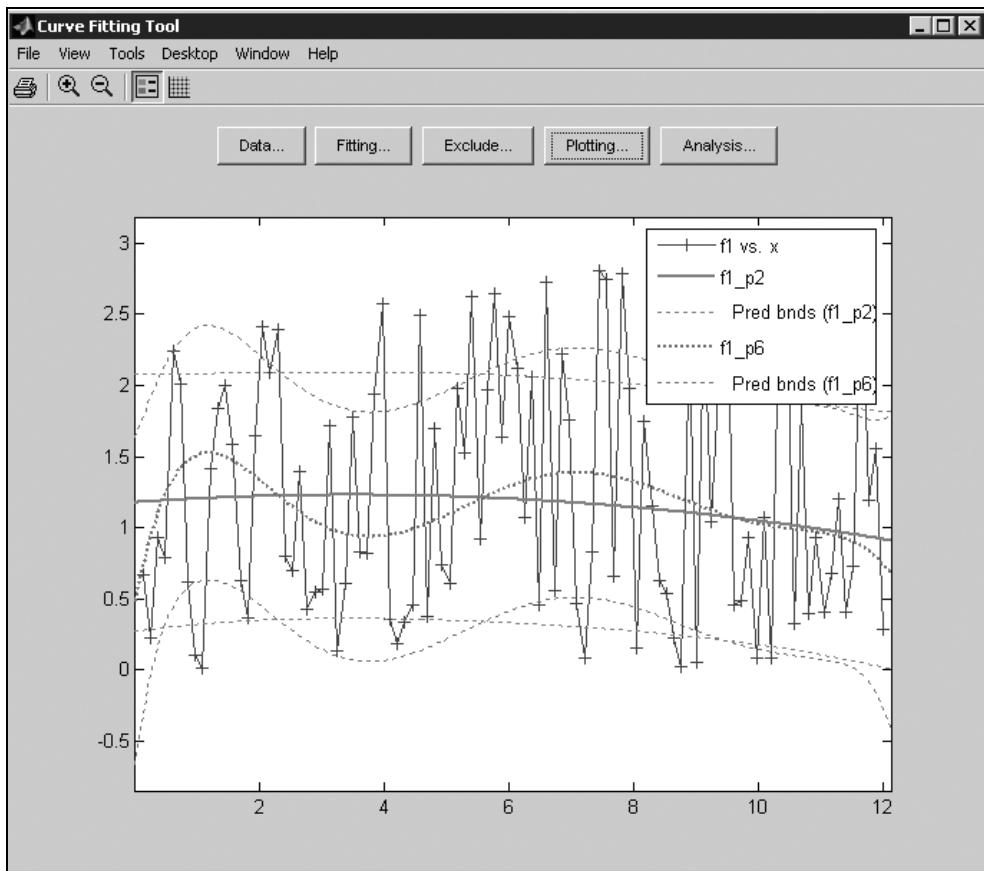
Кроме визуального контроля ошибки в пакете рассчитывается ряд критериев, выводимых в областях **Results** и **Table of Fits** окна **Fitting** (см. рис. 19.8). Рассмотрим эти критерии.

□ **SSE** (Sum of Squares due to Error) — величина, равная значению целевой функции, построенной на основе метода наименьших квадратов, которая вычисляется по формуле:

$$SSE = \sum_{i=1}^N w_i (y_i - \tilde{y}_i)^2,$$

где  $N$  — общее число точек, используемых для приближения;  $y_i$  — узловые значения исходной табличной функции;  $\tilde{y}_i$  — полученные узловые

значения для приближения;  $w_i$  — весовые коэффициенты (по умолчанию равны единице).



**Рис. 19.12.** Границы доверительного интервала.

□ **R-square** — метод использует оценку, вычисляемую следующим образом. Вычисляется величина SSR (Sum of Squares of the Regression) по формуле:

$$SSR = \sum_{i=1}^N w_i (\tilde{y}_i - \bar{y})^2,$$

где  $\bar{y}$  — среднее значение регрессии (математическое ожидание, если табличные значения рассматривать как выборку случайных величин). Затем рассчитывается  $SST = SSR + SSE$ .

Окончательно:

$$R\text{-square} = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}.$$

Значение **R-square** лежит в интервале  $[0, 1]$  и чем оно больше, тем лучше считается приближение.

- **Adjusted R-square** (в приложении используется сокращение **Adj R-sq**) — это скорректированный критерий **R-square**. Имеет смысл учитывать, если его значение положительно и не превосходит единицы. Он рассчитывается на основе числа степеней свободы, обозначаемого в пакете **DFE** и равного разности между числом точек  $N$ , используемых для построения приближения, и числом параметров  $P$ , определяющих приближение. Для полинома это число равно количеству его коэффициентов, т. е. на единицу больше степени. Если величину **DFE** обозначить  $L = N - P$ , то:

$$Adj\ R\ sq = 1 - \frac{SSE(N-1)}{SST(L)}.$$

- **RMSE** (Root Mean Squared Error) — по этому критерию вычисляется величина:

$$RMSE = \sqrt{\frac{SSE}{L}},$$

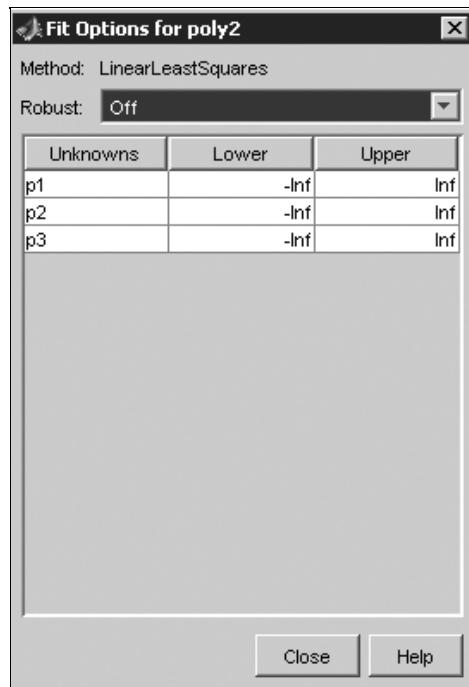
и ее близость к нулю соответствует хорошему приближению.

Желаемые критерии задаются в диалоговом окне **Table Options**, которое появляется после нажатия на кнопку **Table Options**. Имена флагов совпадают с именами критериев, **# Coeff**, которое соответствует ранее введенной величине  $P$ . Значения выбранных критериев отображаются в окне **Results** для приближения, выделенного в области **Table of Fits**.

## Типы аппроксимации для подбора параметров

В Curve Fitting Tool реализовано несколько используемых линейных и нелинейных параметрических моделей для приближения одномерных табличных функций. Тип модели задается в списке **Type of fit** окна **Fitting**, а в расположенному ниже поле выбирается одна из предопределенных моделей данного типа. Для выбранного типа аппроксимации табличных данных

можно определить ряд дополнительных опций, используя кнопку **Fit Options** для вызова соответствующего диалогового окна. Заголовок окна содержит название модели и информацию о количестве параметров, например, для полиномиального приближения второй степени это окно называется **Fit Options for poly2** и имеет вид, представленный на рис. 19.13. В приведенном примере при аппроксимации данных полиномом второго порядка можно указать ограничения на верхнюю и нижнюю границы значений каждого коэффициента в столбцах **Upper** и **Lower** таблицы. Раскрывающийся список **Robust** позволяет указать, надо ли использовать метод приближения, устойчивый по отношению к "выбросам" данных и выбрать: **LAR** (минимизация суммы модулей невязок) или **Bisquare** (итерационное уменьшение веса данных, удаленных от аппроксимирующей кривой). Выбор **Off** означает минимизацию суммы квадратов невязок. Полиномиальная модель является линейной, набор ее опций меньше, чем у нелинейных моделей, которые мы рассмотрим далее.



**Рис. 19.13.** Диалоговое окно  
для уточнения опций выбранного типа приближения

Библиотека параметрических моделей Curve Fitting Toolbox (доступная и в приложении Curve Fitting Tool) содержит следующие широко используемые предопределенные модели для аппроксимации табличных функций.

- **Polynomials** — подбираются коэффициенты полинома, степень которого может варьироваться от 1 до 9 включительно. Этот тип аппроксимации использовался в ранее приведенных примерах.
- **Exponentials** — в качестве аппроксимирующей функции выбирается одна из следующих:

$$y(x) = a e^{bx}; \quad y(x) = a e^{bx} + c e^{dx}.$$

- **Fourier** — данные приближаются отрезками ряда Фурье для  $1 \leq n \leq 8$ :

$$y(x) = a_0 + \sum_{i=1}^n (a_i \sin i\omega x + b_i \cos i\omega x).$$

- **Gaussian** — модель содержит следующий набор функций ( $1 \leq n \leq 8$ ):

$$y(x) = \sum_{i=1}^n \frac{a_i}{e^{\left(\frac{(x-b_i)}{c_i}\right)^2}}.$$

- **Power** — используются степенные функции:

$$y(x) = ax^b; \quad y(x) = a + bx^c.$$

- **Rationals** — табличная функция аппроксимируется отношением двух полиномов степеней не выше 5 (не обязательно равных):

$$y(x) = \frac{\sum_{i=0}^n p_i x^{n-i}}{\sum_{i=0}^m q_i x^{m-i}}, \quad q_0 = 1.$$

- **Sum of Sin Functions** — аппроксимирующая функция является отрезком следующего ряда ( $1 \leq n \leq 8$ ):

$$y(x) = \sum_{i=1}^n a_i \sin(b_i x + c_i).$$

- **Weibull** — подбираются параметры распределения Вебула:

$$y(x) = abx^{b-1} e^{-ax^b};$$

□ **Custom Equations** — пользователь Toolbox может сам определить вид параметрической модели. В следующем разделе поясняется, как это сделать.

Кроме параметрического приближения данных доступны также и другие способы:

□ **Smoothing Spline** — построение сглаживающего сплайна  $g(x)$ , минимизирующего функционал:

$$J(g) = \rho \sum_{i=1}^N w_i (y_i - g(x_i))^2 + (1-\rho) \int (g''(x))^2 dx,$$

где  $w_i$  — заданные веса (равные единице по умолчанию), а сглаживающий параметр  $\rho$  может изменяться от 0 до 1 ( $\rho=0$  соответствует интерполяционному кубическому сплайну, а  $\rho=1$  — приближению полиномом первой степени в смысле наименьших квадратов). По умолчанию параметр  $\rho$  выбирается равным  $1/(1+h^3/6)$ , где  $h$  — среднее расстояние между точками данных. В окне **Fit Options** для этой модели можно задать другое допустимое значение.

□ **Interpolant** — применяется один из следующих способов интерполирования:

- **Linear** — интерполяция кусочно-линейной функцией, т. е. отрезками прямых, соединяющих две соседние точки (интерполяционный сплайн первого порядка дефекта один);
- **Nearest neighbor** — интерполяция ступенчатой функцией, значение которой совпадает со значением табличной функции в ближайшем узле;
- **Cubic spline** — обычный кубический интерполяционный сплайн;
- **Shape-preserving** — кусочно-кубическая эрмитова интерполяция.

Перечисленные способы интерполирования мы обсуждали в разд. "Интерполяция сплайнами" главы 6.

### Примечание

При необходимости получить более детальное представление о методах приближений, используемых в пакете, следует воспользоваться специализированной литературой.

## Определение собственной параметрической модели

Вы можете определить собственную модель, зависящую от параметров, и с ее помощью выполнить приближение табличной функции, подбирая параметры средствами Curve Fitting Toolbox. Поясним это на примере приближения табличных функций  $f_2$  и  $f_3$  (соответствующие множества данных  $f_2(x)$ ,  $speed(time)$  вы создали в начале чтения этой главы) (см. разд. "Создание множества данных для приближения" данной главы).

Для создания собственной параметрической модели можно либо в меню **Tools** основного окна **Curve Fitting Tool** выбрать пункт **Custom Equation**, либо в диалоговом окне **Fitting** в раскрывающемся списке **Type of fit** указать тип приближения **Custom Equations** и щелкнуть по кнопке **New equation**. В обоих случаях открывается диалог **Create Custom Equation**, содержащий две вкладки для создания линейной параметрической модели (**Linear Equations**) и нелинейной (**General Equations**). Создайте две параметрические модели: одну линейную, а другую — нелинейную. Откройте диалоговое окно **Create Custom Equation** и перейдите на вкладку **General Equations**. В полях **Equation**, определяющих вычислительную формулу, запишите выражение для функции  $y(x) = a \cdot e^{b \cdot x} \cdot \sin(c \cdot x + d)$ , зависящей от четырех параметров, как показано на рис. 19.14. Дополнительно в таблице можно указать стартовые значения параметров, начиная с которых будет выполняться итерационный алгоритм поиска параметров, основанный на минимизации целевой функции (как правило, это взвешенная сумма квадратов невязок), и указать верхнюю и нижнюю границы изменения каждого параметра. В поле **Equation name** можно ввести собственное имя параметрической модели, но удобно оставить имя, генерированное MATLAB в виде формулы. Нажатие кнопки **OK** завершает создание модели, соответствующее выражение или имя добавляется в список **Custom Equations** окна **Fitting**.

Теперь определим линейную параметрическую модель, которая в общем случае записывается в виде конечной суммы:

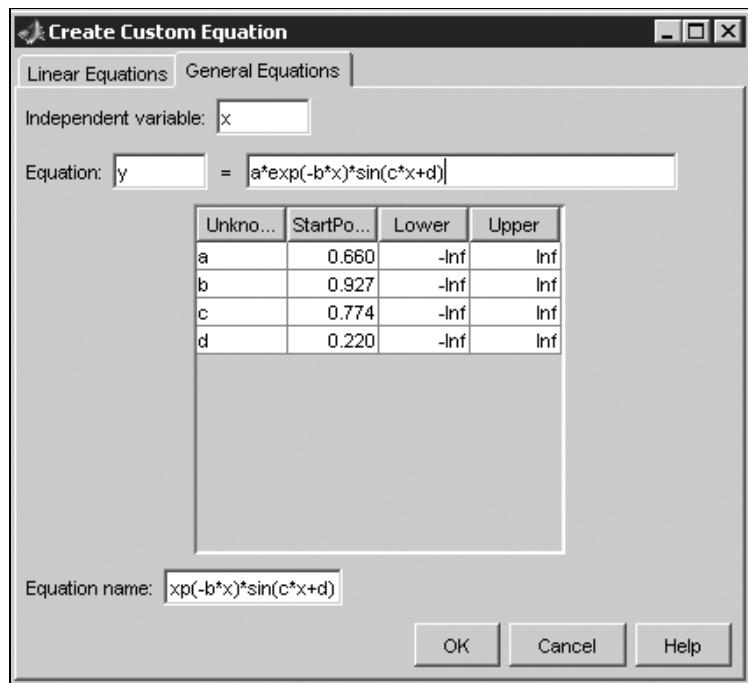
$$y(x) = a_0 + \sum_{i=1}^n a_i f_i(x),$$

где  $f_i$  — любая функция одного аргумента. Задайте следующую модель:

$$y(x) = a_0 + a_1 \cos x + a_2 e^x + a_3 \ln x.$$

Для этого перейдите на вкладку **Linear Equations**, в поле **Terms** введите функцию  $\cos(x)$ , нажмите кнопку **Add a term** и во вновь появившемся поле **Terms** введите  $e^x(x)$ , как показано на рис. 19.15. Аналогично задайте последнее слагаемое линейной параметрической модели (помните, что имя

функции для натурального логарифма  $\log$ ). Для включения в модель неизвестного аддитивного параметра (в нашем примере это  $a_0$ ) флаг **Unknown constant coefficient** должен быть установлен, его сброс приводит к отсутствию данного слагаемого в модели. После подтверждения нажатием кнопки **OK** будет создана параметрическая модель.



**Рис. 19.14.** Диалоговое окно

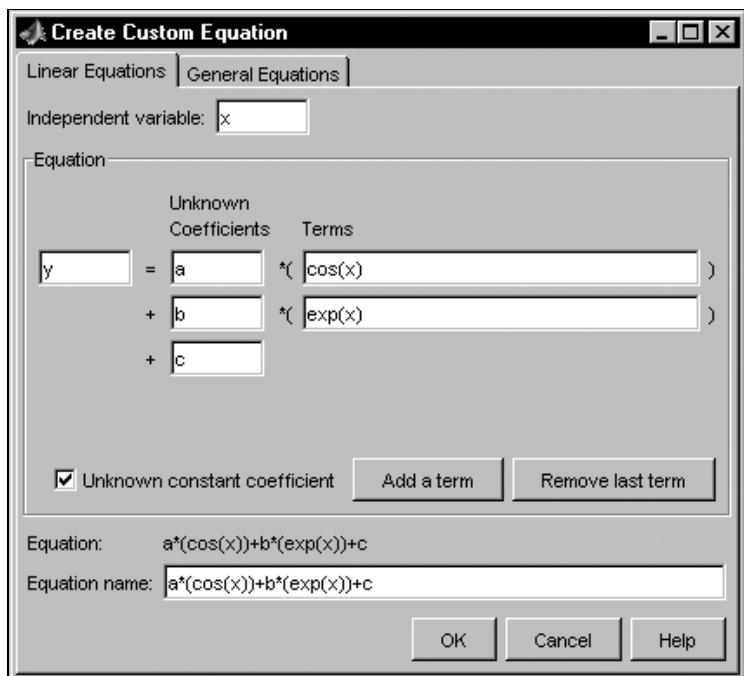
для определения параметрической функции общего вида

### ◀ Предупреждение ▶

При конструировании линейной модели не допускается использовать функции, зависящие от искомых параметров.

После создания собственной параметрической модели для нее определяется стандартный список опций в диалоговом окне **Fit options for custom** (к его заголовку также добавляется имя параметрической модели или описывающее ее выражение), которое появляется после выбора модели в окне **Fitting** и нажатия кнопки **Fit Options**. Наборы опций произвольной линейной модели

и рассмотренной выше полиномиальной (частного случая линейной модели) совпадают — имеется возможность выбора метода приближения в раскрывающемся списке **Robust** и задания границ параметров (см. разд. "Типы аппроксимации для подбора параметров" данной главы).



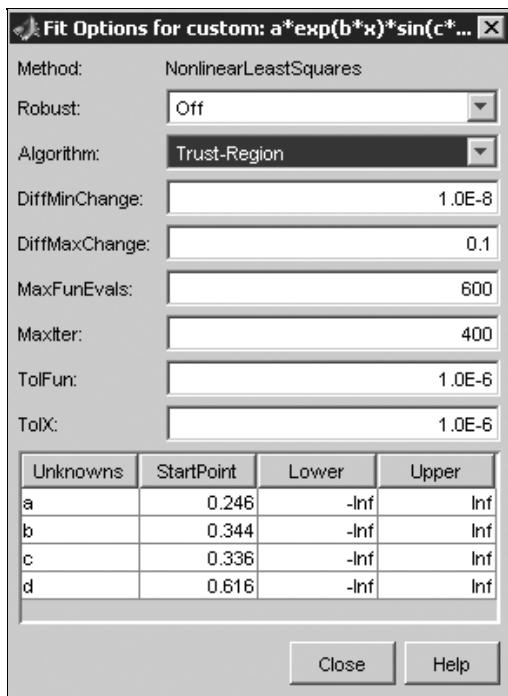
**Рис. 19.15.** Диалоговое окно  
для определения линейной параметрической функции

Нелинейные модели (предопределенные или собственные) допускают ряд дополнительных настроек, связанных с итерационным методом минимизации целевой функции — невязки приближения, построение которой зависит от выбора в списке **Robust**. Этот вопрос мы обсуждали при построении полиномиального приближения. Вид диалогового окна для настройки алгоритма поиска параметров нелинейной модели представлен на рис. 19.16. Предлагаемые опции имеют следующий смысл.

- **Algorithm** — метод минимизации целевой функции. Рекомендуется вначале применить метод доверительных областей (**Trust-Region**), установленный по умолчанию, а если не удалось получить удовлетворительное решение, то метод Левенберга—Марквардта (**Levenberg—Marquardt**) при

условии, что на искомые параметры нет ограничений. Метод Ньютона—Гаусса (**Gauss—Newton**) включен для учебных целей.

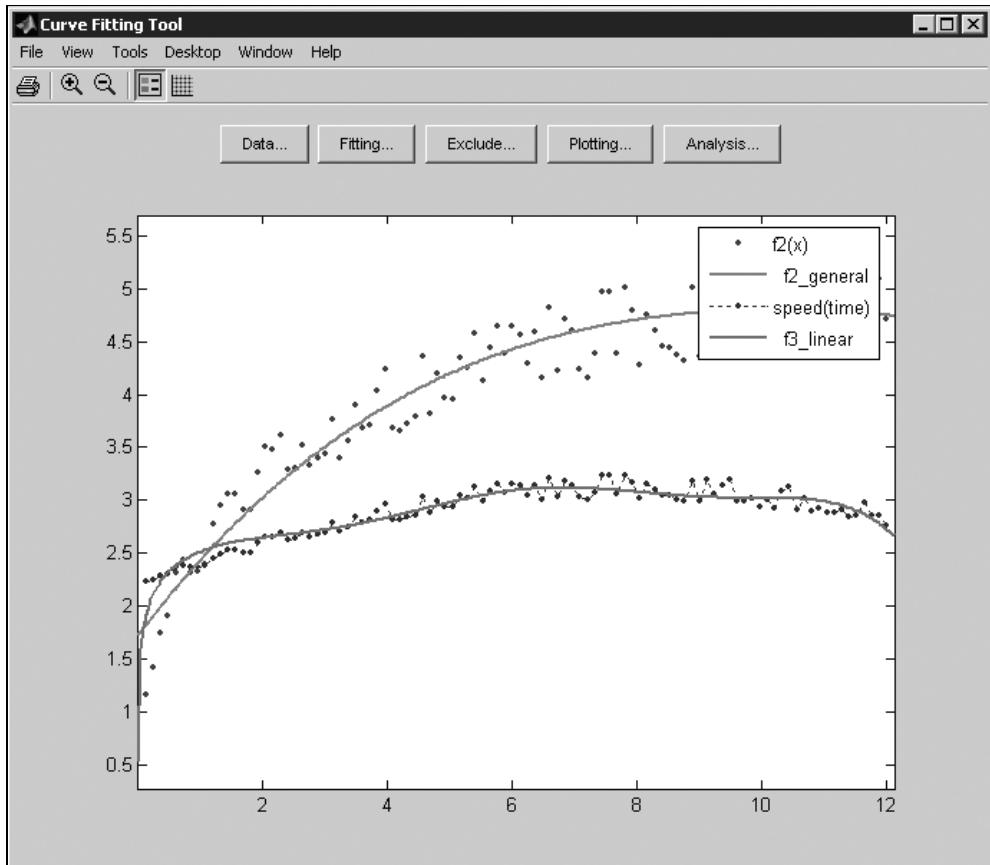
- DiffMinChange** и **DiffMaxChange** — минимальное и максимальное изменение коэффициентов матрицы Якоби при аппроксимации их конечными разностями (по умолчанию  $10^{-8}$  и 0.1 соответственно).
- MaxFunEvals** — максимальное количество вычислений значений функций, описывающей модель.
- MaxIter** — максимальное количество итераций.
- TolFun** — точность для завершения вычислений по значению функции.
- TolX** — точность для завершения вычислений по значению искомых параметров.



**Рис. 19.16.** Диалоговое окно  
для задания опций алгоритма поиска параметров

Выполните подбор параметров для множества `speed(time)`, используя созданную линейную параметрическую модель, а для множества `f2(x)` — не-

линейную. Для этого откройте диалоговое окно **Fitting** и постройте приближения так, как было описано в разд. "Создание приближений" данной главы. На рис. 19.17 представлены полученные результаты. Очевидно, что линейная параметрическая модель дает менее удовлетворительную аппроксимацию, несмотря на то, что разброс в табличной функции меньше. Мы намеренно привели этот пример, который показывает, что количество свободных параметров не всегда определяет качество приближения. Примените теперь нелинейную модель для приближения множества speed(time).



**Рис. 19.17.** Приближение данных с использованием собственных параметрических моделей

## Анализ построенных приближений

Построенное приближение имеет смысл подвергнуть не только визуальному анализу, но и получить ряд его численных характеристик. Мы уже обсуждали получение значений основных критериев качества приближения (см. разд. "Контроль качества приближений" данной главы).

Для дополнительного исследования приближения, включающего восполнение данных, дифференцирование и интегрирование табличной функции служит диалоговое окно **Analysis** (рис. 19.18), которое открывается нажатием одноименной кнопки в основном окне приложения Curve Fitting Tool. Средства диалогового окна **Analysis** позволяют использовать найденное приближение для получения информации о данных в некотором наборе точек. Имя приближения выбирается в раскрывающемся списке **Fit to analyze**, а вектор значений аргумента в списке ввода **Analyze at Xi =**, причем его элементы могут лежать и вне области определения табличной функции. После выбора одной из операций, доступных для проведения в окне **Analysis**, следует нажать кнопку **Apply** для ее выполнения. Перечислим эти операции.

- Восполнение значений табличной функции в заданном наборе точек при установленном флаге **Evaluate fit at Xi**. При этом способ получения результата определяется значением переключателя **Prediction or confidence bounds**:
  - **None** — вычисление по найденному приближению;
  - **For function** — вычисление границ области значений с учетом рассчитанных допустимых интервалов, определяемых уровнем доверительной вероятности (ее значение указывается в поле ввода **Level**), для параметров модели;
  - **For new observation** — при определении границ области значений предполагается наличие ошибок во входных данных, распределенных по нормальному закону с нулевым средним и постоянной дисперсией, оцененной по исходным данным.
- Вычисление производных от приближения. Можно определить первую и вторую производные, установив соответственно флаги **1st derivative at Xi** и **2st derivative at Xi**.
- Вычисление интегралов приближения. Верхними пределами являются узлы из набора **Xi**, а нижним либо наименьший узел (при установленном переключателе **Start from min(Xi)**), либо заданное значение, которое вводится при установке переключателя **Start from** в расположеное рядом поле. Интегрирование выполняется, если установлен флаг **Integrate to Xi**.

Для представления результатов в графическом виде в отдельном окне следует установить флаг **Plot results**, а для одновременного отображения исходных данных — флаг **Plot data set**. Графики изображаются на отдельных осях.

После задания всех параметров для проведения анализа и щелчка по кнопке **Apply** производятся расчеты и выводятся результаты. Создайте полиномиальное приближение 5-ой степени с именем *f2-analis-p5* для ранее введенной табличной функции *f2*, и проведите анализ приближения, например, для установок, показанных на рис. 19.18.

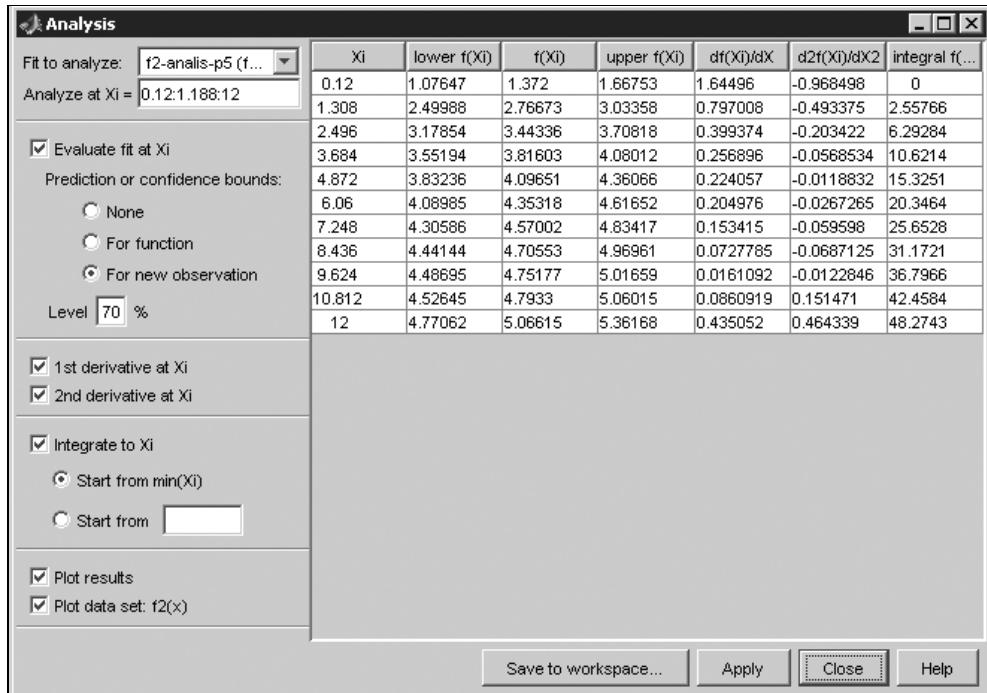


Рис. 19.18. Диалоговое окно **Analysis**  
с результатами

Графически результаты отображаются в окне редактора графиков как показано на рис. 19.19. Используя средства, рассмотренные в главе 4, вы можете должным образом оформить результаты.

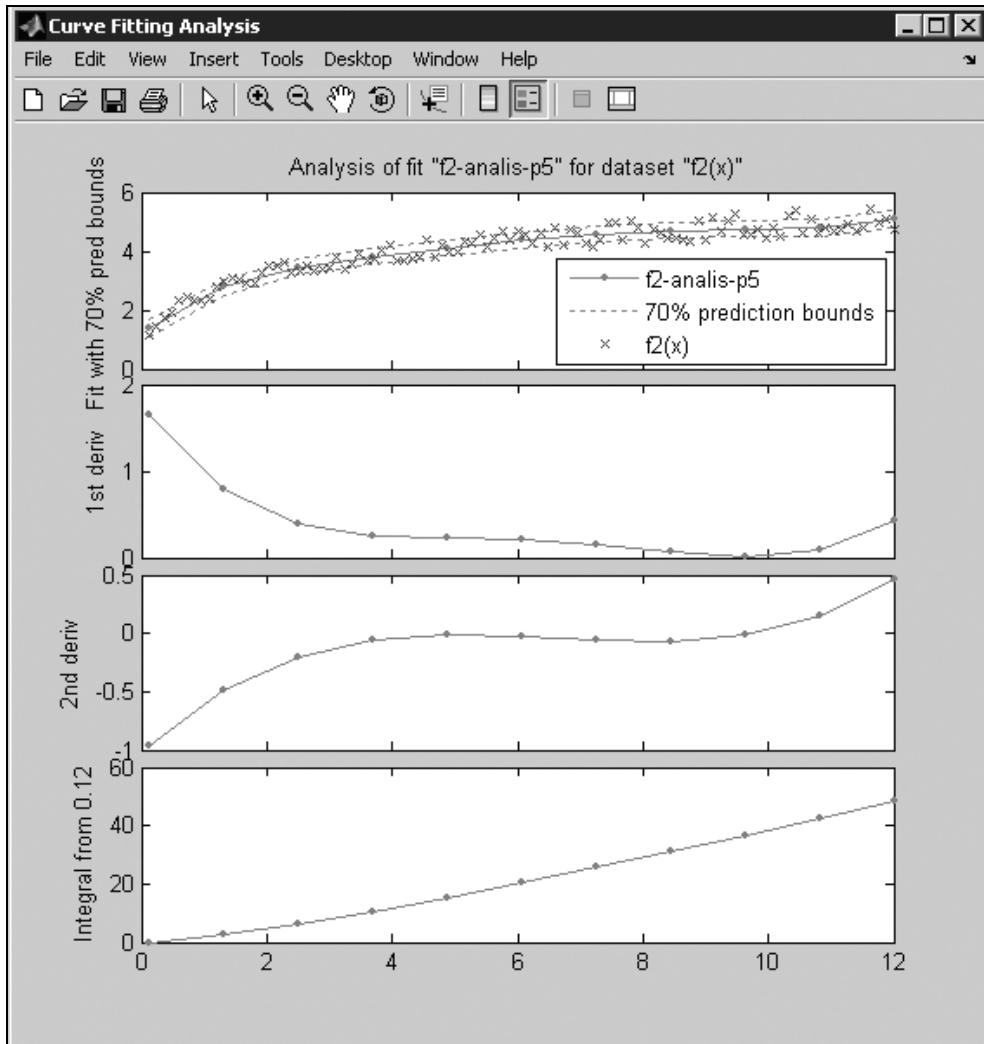
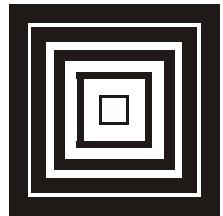


Рис. 19.19. Графическое представление результатов анализа



## Глава 20

# Решение экономических задач

Пакет MATLAB содержит ряд Toolbox, связанных с финансовыми и экономическими расчетами для реального бизнеса или обучения и исследований. Центральное место занимает Financial Toolbox позволяющий рассчитывать потоки платежей и их анализировать, в том числе, оценивать инвестиционные проекты, вычислять и анализировать цены, доходности и чувствительности отдельных финансовых активов и их портфелей, прогнозировать и оценивать экономические показатели, выявлять риски и управлять ими. Функции этого Toolbox реализуют основные алгоритмы финансовых вычислений. Financial Toolbox предоставляет множество различных функций, ориентированных на решение как конкретных задач, так и задач в общей постановке.

Financial Derivatives Toolbox является расширением Financial Toolbox и позволяет оценивать фиксированный доход производных финансовых инструментов и пакетов ценных бумаг на основе различных моделей, описывающих поведение процентных ставок в будущих периодах. Financial Time Series Toolbox представляет собой набор функций для анализа данных на различных финансовых рынках, близких по идеологии техническому анализу рынка ценных бумаг. В настоящей главе основное внимание уделено Financial Toolbox.

В этой главе рассматриваются приемы использования средств Financial Toolbox для решения модельных задач предметной области. Вопросы, связанные с адекватностью моделей, условий их применимости или справедливости гипотез относительно состояния финансовых рынков и аналогичные вопросы выходят за рамки настоящей книги. Более того, применение средств пакета в некоторых случаях может приводить к неверным результатам с точки зрения бизнес-приложений. Проблема анализа полученных данных и их дальнейшего применения лежит на пользователе пакета и требует дополнительного экономического образования.

**Примечание**

При использовании ряда функций Financial Toolbox, например, для анализа портфелей ценных бумаг, требуется, чтобы был установлен также Optimization ToolBox.

## Функции для работы с датами и временем

Многие функции работы с датами и временем, описываемые в этом разделе, определены, как в самом MATLAB, так и пакете Financial Toolbox. Несмотря на их большое количество, принципы использования легко продемонстрировать на ограниченном множестве (некоторых) основных функций.

### Представление времени и дат в MATLAB

Финансовые вычисления оперируют с интервалами времени между рассматриваемыми событиями, поэтому практически все они используют даты и время. В пакете MATLAB используется следующий способ внутреннего хранения времени. Текущий момент времени представляется в форме десятичного числа, при этом целая часть определяет номер дня после 01 января 0000 г. от рождества Христова, а дробная — время, отсчитанное от полуночи (00 часов 00 минут 00 секунд текущих суток). Для получения текущего момента времени во внутреннем формате можно использовать функцию now:

```
>> format long g
>> mom = now
mom =
732088.579040787
```

Таким образом, в момент обращения к функции был 732088-ой день от рождения Христова, а время составило 0.579040787 часть от 24 часов.

Такая форма удобна для вычислительных целей, поскольку дает возможность быстро найти интервал между разными моментами времени, но неприемлема для восприятия человеком. Общепринятое представление момента времени в пакете MATLAB обеспечивается в текстовой (строка символов) или векторной форме (массив, элементы которого являются составляющими момента времени: год, месяц, час и т. п.). Для получения текстовой формы предназначена функция datestr, обращение к которой имеет вид:

```
текстовый_формат = datestr(внутренний_формат, шаблон)
```

Первый входной параметр функции определяет значение момента времени во внутреннем представлении, а второй (может быть либо числом, либо строкой) — текстовую форму результата. В табл. 20.1 приведены некоторые значения этого параметра. В качестве примера представлен результат для ранее приведенного момента времени.

**Таблица 20.1. Шаблоны и текстовый формат даты**

| <b>Шаблон преобразования</b> |                        | <b>Текстовый формат</b> |
|------------------------------|------------------------|-------------------------|
| <b>Числовой</b>              | <b>Строковый</b>       |                         |
| 0                            | 'dd-mm-yyyy HH:MM:ss'  | 21-May-2004 13:53:49    |
| 1                            | 'dd-mm-yyyy '          | 21-May-2004             |
| 2                            | ' mm/dd/yy '           | 05/21/04                |
| 3                            | 'mmm'                  | May                     |
| 4                            | 'm'                    | M                       |
| 5                            | 'mm'                   | 05                      |
| 6                            | ' mm/dd '              | 05/21                   |
| 7                            | 'dd'                   | 21                      |
| 8                            | 'ddd'                  | Fri                     |
| 9                            | 'd'                    | F                       |
| 10                           | 'yyyy'                 | 2004                    |
| 11                           | 'yy'                   | 04                      |
| 12                           | 'mmmyy'                | May04                   |
| 13                           | 'HH:MM:SS'             | 13:53:49                |
| 14                           | 'HH:MM:SS PM'          | 1:53:49 PM              |
| 15                           | 'HH:MM'                | 13:53                   |
| 16                           | 'HH:MM:SS PM'          | 1:53 PM                 |
| 17*                          | 'QQ-yy'                | Q2-04                   |
| 18*                          | 'QQ'                   | Q2                      |
| 19                           | 'dd/mm'                | 21/05                   |
| 20                           | 'dd/mm/yy'             | 21/05/04                |
| 21                           | 'mmm.dd.yyyy HH:MM:ss' | May.21.2004 13:53:49    |

Таблица 20.1 (окончание)

| Шаблон преобразования |                       | Текстовый формат     |
|-----------------------|-----------------------|----------------------|
| Числовой              | Строковый             |                      |
| 22                    | 'mmm.dd.yyyy'         | May.21.2004          |
| 23                    | 'mm/dd/yyyy'          | 05.21.2004           |
| 24                    | 'dd/mm/yyyy'          | 21/05/2004           |
| 25                    | 'yy/mm/dd'            | 04/05/21             |
| 26                    | 'yyyy/mm/dd'          | 2004/05/21           |
| 27                    | 'QQ-yyyy'             | Q2-2004              |
| 28                    | 'mmmyyyy'             | May2004              |
| 29                    | 'yyyy-mm-dd'          | 2004-05-21           |
| 31                    | 'yyyy-mm-dd HH:MM:SS' | 2004-21-May 13:53:49 |

\*QQ — обозначает квартал.

Для получения векторной формы момента времени используется функция datevec, обращение к которой имеет вид:

векторный\_формат = datevec (внутренний\_формат или текстовый\_формат)

или

[yyyy, mmm, dd, hh, mm, ss] =

datevec (внутренний\_формат или текстовый\_формат).

Результат в первом случае заносится в массив размерности  $1 \times 6$ , во втором — в переменные (массивы  $1 \times 1$ ) с указанными именами (аббревиатура соответствует символьному формату полного текстового представления момента времени).

Функция datenum делает обратное преобразование момента времени, записанного в допустимой текстовой или векторной форме (double array  $1 \times 6$ , несколько double array  $1 \times 1$ ) во внутреннее представление и допускает разные варианты вызова:

внутренний\_формат = datenum(текстовый\_формат или векторный\_формат)

или

внутренний\_формат = datenum(yyyy, mmm, dd)

или

внутренний\_формат = datenum(yyyy, mmm, dd, hh, mm, ss)

Этой функцией следует пользоваться осторожно, поскольку она не проверяет допустимость числовых значений. Следующий пример это иллюстрирует (смоделирована ошибка, когда при обращении к функции переставлены месяц и день):

```
>> date1 = datenum(2004, 21, 05)
date1 =
 732560
>> date2 = datestr(date1,1)
date2 =
05-Sep-2005
```

Однако при использовании текстового аргумента проверка осуществляется:

```
> date1 = datenum('21/05/2004')
??? Error using ==> datevec
21 is too large to be a month.
```

Если год опущен, то используется значение из системной даты, установленной на компьютере. Еще одна особенность при использовании текстовой строки состоит в том, что допустимым является американский формат даты так, что 01/12 это 12 января, а не 1 декабря.

### Примечание

Ранее рассмотренная функция `now` позволяла получить внутреннее представление текущего системного момента времени. Для получения системной даты можно использовать две функции: `today` — дает системную дату во внутреннем формате и `date` — получает результат для системной даты в текстовом формате.

## Функции определения числа дней между датами

Очень часто при проведении экономических расчетов интересуют не конкретные моменты времени, а интервалы между датами. Однако число дней зависит от принятой для вычислений структуры календарного года. В типовых расчетах используется понятие *базовый период*, в качестве которого принимается один год с определенным числом дней в месяце. Базовый период может рассматриваться как реальный год с 365-ю или 366-ю календарными днями, как абстрактный год, состоящий из 12-ти месяцев по 30 дней в каждом и 360-ю днями в году или как комбинированный год. При различных расчетах при решении одной задачи могут использоваться разные

представления о структуре года, поэтому пакет использует несколько типов базовых периодов.

Для идентификации базового периода будем использовать флаг с именем базис (в справочной системе MATLAB — basis).

В табл. 20.2 приведены возможные значения этого флага и характеристики соответствующего базового периода.

*Таблица 20.2. Параметры базового периода*

| Значение флага basis | Число дней в году           | Число дней в месяце               |
|----------------------|-----------------------------|-----------------------------------|
| 0<br>(по умолчанию)  | Фактическое:<br>365 или 366 | Фактическое:<br>30, 31, 28 или 29 |
| 1                    | 360                         | 30                                |
| 2                    | 360                         | Фактическое:<br>30, 31, 28 или 29 |
| 3                    | 365                         | Фактическое:<br>30, 31, 28 или 29 |

В зависимости от выбранного базового периода интервалы между двумя датами могут отличаться. Для вычисления количества дней в некотором периоде можно использовать функцию daysdif:

число\_дней = daysdif(начальная\_дата, конечная\_дата, базис)

Входными аргументами являются даты (в текстовом виде или во внутреннем формате) и флаг, определяющий структуру года. Следующий пример это демонстрирует.

```
>> date1 = datenum('05/21/2004');
>> date2 = datenum('01/13/2003');
>> n_fact = daysdif(date1, date2, 0)
n_fact =
 -494
>> n_365 = daysdif(date1, date2, 3)
n_365 =
 -493
>> n_360 = daysdif(date1, date2, 1)
n_360 =
 -488
```

Отрицательное число дней означает, что конечная дата предшествует начальной.

### Примечание

Для вычисления количества дней между двумя датами можно использовать дополнительно три функции `daysact`, `days360`, `days365`, имеющие одинаковое обращение с двумя датами в качестве входных параметров, но использующие разную структуру года (соответственно из расчета фактического количества дней в году, 360 дней и 365 дней).

Кроме фактического количества дней между двумя датами интерес представляет количество рабочих дней между датами. Для этого сначала формируется массив дополнительных выходных дней в текстовом или числовом формате, кроме суббот и воскресений. Этот массив используется в качестве аргумента функции, учитывающей эти праздничные дни. Наиболее удобно использовать средства пакета. В файл-функции `holidays` (подкаталог ...\\toolbox\\finance\\calendar) содержится перечень нерабочих дат для NYSE (Нью-Йоркской фондовой биржи) с 01 января 1950 года по 31 декабря 2030 года во внутреннем формате записи дат. Для специфической настройки можно внести изменения в его текст и сохранить под тем же именем в рабочем каталоге, не изменяя исходного файла. В соответствии с правилом поиска М-файлов будет вызываться наша собственная файл-функция `holidays` (установка последовательности просмотра каталогов в MATLAB при обращении к М-файлам описана в главе 5).

Например, для 2004 г. в этой функции содержатся строки:

```
731947 ;... % 01-Jan-2004 New Year's Day
731965 ;... % 19-Jan-2004 Martin Luther King Jr. Day
731993 ;... % 16-Feb-2004 Washington's Birthday
732046 ;... % 09-Apr-2004 Good Friday
732098 ;... % 31-May-2004 Memorial Day
732133 ;... % 05-Jul-2004 Independence Day
732196 ;... % 06-Sep-2004 Labor Day
732276 ;... % 25-Nov-2004 Thanksgiving
732305 ;... % 24-Dec-2004 Christmas
```

Используя, например, редактор М-файлов, можно заменить их, предварительно вычислив даты праздников во внутреннем формате, следующими:

```
731947 ;... % 01-Jan-2004 Новый год
731948 ;... % 02-Jan-2004 Новый год
```

```
731953 ;... % 07-Jan-2004 Рождество
732000 ;... % 23-Feb-2004 День армии
732014 ;... % 08-Mar-2004 Женский день
732070 ;... % 03-May-2004 перенос 01-May
732071 ;... % 04-May-2004 перенос 02-May
732077 ;... % 10-May-2004 перенос 09-May
732112 ;... % 14-Jun-2004 перенос 12-Jun
732259 ;... % 08-Nov-2004 перенос 07-Nov
732294 ;... % 13-Dec-2004 перенос 12-Dec
```

Функция `holidays` формирует требуемый массив дней во внутреннем формате:

```
праздничные_дни = holidays(начальная_дата, конечная_дата)
```

### Примечание

Стандарты пакета не позволяют учесть российские особенности типа "черные субботы" и непредсказуемые переносы праздничных дней, если они приходятся на общевыеходные дни. В этих случаях, если рассматриваются разные варианты, после вычисления числа рабочих дней можно это число изменить "вручную".

После изменения файла `holidays.m` можно получить список праздничных дней между двумя датами для российского календаря:

```
>> rest_days = datestr(holidays('01-Jan-2004 ', '03-May-2004 '))
rest_days =
01-Jan-2004
02-Jan-2004
07-Jan-2004
23-Feb-2004
08-Mar-2004
03-May-2004
```

Функция `wrkdydif` вычисляет число рабочих дней с учетом праздников:

```
число_рабочих_дней =
 wrkdydif(начальная_дата, конечная_дата, праздничные_дни)
```

Применение других функций работы с датами и временем будут поясниться по мере необходимости.

## Расчеты денежных потоков

Вычисления, связанные с потоками платежей ( $CF$ ), используются при решении большинства экономических задач, при этом получаемые суммы считаются положительными, а выплачиваемые — отрицательными. Иногда различают входящий поток платежей ( $CIF$ ), т. е. получение средств, и исходящий поток платежей ( $COF$ ), т. е. оплата своих обязательств. Стоимость товаров и услуг, следовательно, и ценность денег, как их эквивалента, меняется с течением времени. Для проведения экономического анализа приходится пересчитывать цены или денежные суммы, приводя их к определенной дате (прошлой, текущей или будущей). При вычислении стоимости платежа в прошлом (по отношению к сроку платежа) — *текущей стоимости* ( $pv$ ), применяется *коэффициент дисконтирования*, рассчитываемый по *процентной ставке* ( $r$ ) или *по учетной (дисконтной) ставке* ( $d$ ).

При вычислении стоимости платежа в будущем — *будущей стоимости* ( $fv$ ), используется коэффициент наращивания средств, являющийся обратной величиной к коэффициенту дисконтирования.

Приведем некоторые формулы, используемые при расчетах денежных потоков. Знание этих формул не обязательно, важно правильно использовать функции пакета MATLAB. Однако понимание вопроса упрощается, если он математически формализован.

Годовая учетная ставка ( $d$ ) и годовая процентная ставка ( $r$ ) связаны соотношением:

$$d = \frac{r}{1+r}, \quad (20.1)$$

поэтому коэффициент наращивания или дисконтирования суммы можно выразить через любую величину. Будем применять чаще используемую процентную ставку.

Будущая стоимость ( $fv$ ) и текущая стоимость одного платежа ( $pv$ ) в момент времени  $t_k = n \cdot T + t$ , где  $T$  — базовый период и  $t < T$ , связаны с годовой процентной ставкой ( $r$ ) соотношением:

$$fv = pv \left(1 + r\right)^n \left(1 + \frac{t}{T} r\right). \quad (20.2)$$

Часто принимаются условия, что платеж производится в конце (или начале) базового периода. Поэтому  $t = 0$  и формула (20.2) упрощается. Если по формуле (20.2) вычисляется величина  $fv$ , то говорят о наращивании текущей суммы  $pv$ , если определяется величина  $pv$ , то говорят о дисконтиро-

вании суммы  $fV$ . Из формулы (20.2) может находиться также параметр  $r$ , который в этом случае обозначают  $y$ , и он имеет смысл *ставки доходности вложения*  $pv$  при условии получения дохода  $fV$ .

При начислении процентов  $m$  раз в году следует использовать *номинальную процентную ставку*  $r_0$  и вместо базового периода  $T$  период начисления  $T_0$ :

$$r = \frac{r_0}{m}, \quad T_0 = \frac{T}{m}, \quad (20.3)$$

количество начислений будет равно не  $n$ , а  $nm$ , и  $t < T_0$ .

Рассмотрим следующую модельную задачу, на которой проиллюстрируем применение функций пакета MATLAB. Инвестор оценивает следующий проект: на протяжении  $n$  лет он вкладывает в начале каждого  $i$ -го базового периода сумму  $P_{i-1}$ , а в конце каждого периода получает доход  $C_i$ . Текущая процентная ставка  $r$  (или учетная ставка  $d$ ). В частном случае некоторые величины  $P_i$  или  $C_i$  могут быть равны нулю. Надо произвести оценку такого проекта. Для упрощения модели считается, что конец предыдущего периода совпадает с началом следующего.

С учетом формулы (20.2) можно записать текущую нетто стоимость проекта в виде потока платежей:

$$npv = \sum_{i=1}^n \frac{C_i - P_i}{(1+r)^i} - P_0 = \sum_{i=0}^n \frac{CF_i}{(1+r)^i}. \quad (20.4)$$

Величины  $CF_i$  со своим знаком определяют поток платежей проекта.

Можно также использовать будущую стоимость потока платежей:

$$fv = \sum_{i=1}^n (C_i - P_i)(1+r)^{n-i} - P_0 (1+r)^n = \sum_{i=0}^n CF_i (1+r)^{n-i}. \quad (20.5)$$

Инвестор может оценить проект, используя разные критерии.

### Примечание

MATLAB позволяет использовать более сложные модели вычисления нетто стоимости потока платежей — нерегулярных потоков (не обязательно периодические), когда выплаты производятся в заранее известные сроки. Для этого нет необходимости выписывать формулы типа (20.4) и (20.5), нужно просто воспользоваться функциями MATLAB.

Если применить критерий *nir*, то надо убедиться, что нетто стоимость проекта положительна, т. е. проект не убыточен. Для этого можно воспользоваться функцией *pvvar*.

Можно руководствоваться аналогичным критерием, если вычислить будущую стоимость *fv* проекта и убедиться, что она не отрицательна. Функция *fvvar* позволяет произвести такие расчеты.

Часто используемым критерием является критерий *IRR*, при котором определяется норма внутренней доходности из условия, что текущая нетто стоимость потоков *CIF* (входящих) и *COF* (исходящих) проекта совпадает. При условии выплат и поступлений в конце (начале) года норма внутренней доходности вычисляется с помощью функции *irr*.

Рассмотрим конкретное наполнение модельной задачи: при текущей процентной ставке 10% анализируются два проекта, оба рассчитанные на 5 лет, начиная с конца 2004 г. Платежи происходят в последний рабочий понедельник месяца. Потоки платежей оцениваются по-разному для каждого проекта.

- Проект 1. Начальное вложение 100 млн руб. и получение ежегодных доходов, начиная с конца второго года в размерах соответственно: 47, 53, 30, и 20 млн руб.
- Проект 2. Начальное вложение 80 млн руб., дополнительное вложение 40 млн руб. в начале 4-го года и получение ежегодных доходов, начиная с конца первого года в размерах 25, 15, 20, 30, 80 млн руб. В последний год доход ожидается получить двумя равными поступлениями в конце каждого полугодия.

Представим данные в виде табл. 20.3 для более легкого восприятия потоков платежей по проектам, считая, что все расчеты производятся в последний понедельник месяца.

**Таблица 20.3. Параметры инвестиционных проектов**

| Дата поступления | Сумма поступления |          |
|------------------|-------------------|----------|
|                  | Проект 1          | Проект 2 |
| 27-Dec-2004      | -100              | -80      |
| 26-Dec-2005      | 0                 | 25       |
| 25-Dec-2006      | 47                | 15       |
| 31-Dec-2007      | 53                | -20      |
| 29-Dec-2008      | 30                | 30       |

Таблица 20.3 (окончание)

| Дата поступления | Сумма поступления |          |
|------------------|-------------------|----------|
|                  | Проект 1          | Проект 2 |
| 29-Jun-2009      | 0                 | 40       |
| 28-Dec-2009      | 20                | 40       |

Если не учитывать зависимость стоимости платежей от времени, то доход по обоим проектам равен 50 млн руб. Оценим эти проекты по критерию *prv*.

Составьте потоки платежей для каждого проекта с учетом их знаков и воспользуйтесь функцией *pvvar* для вычисления настоящей нетто стоимости проектов, которая вызывается:

```
prv = pvvar(поток_платежей, процентная_ставка, даты_платежей)
```

и функцией *fvvar* для определения будущей нетто стоимости потока платежей:

```
fv = fvvar(поток_платежей, процентная_ставка, даты_платежей)
```

Массив дат платежей может содержать либо числовые, либо текстовые данные. Примите в качестве дат платежей последний рабочий понедельник месяца.

Обращение к функции *xirr*, вычисляющей ставку внутренней доходности потока непериодических выплат и поступлений, имеет вид:

```
внутренняя_доходность =
```

```
xirr(поток_платежей, даты_платежей, начальное_приближение, ...
число_итераций)
```

Первых два входных параметра имеют тот же смысл, что и для функций *pvvar* и *fvvar*. Два последних параметра необязательны и задают начальное приближение (по умолчанию 0.1) для решения уравнения вида  $CIF = COF$  (совпадение текущей стоимости суммарных поступлений и платежей) и число итераций (по умолчанию 50) для решения этого уравнения.

Для учета дня расчета (последнего понедельника месяца) следует использовать функцию *lweekdate*, возвращающую дату во внутреннем формате и имеющую три числовых входных параметра: номер дня недели, год и месяц. В MATLAB дни недели нумеруются от 1 до 7, начиная с воскресенья.

В листинге 20.1 содержится файл-программа для расчета различных критериев по анализируемым проектам.

### Листинг 20.1. Файл-программа для оценки инвестиционных проектов

```
% задание денежных потоков
cf1 = [-100 0 47 53 30 20];
cf2 = [-80 25 15 -20 30 40 40];
% определение дат платежей первого проекта
for i = 1:6
date_flow(i) = lweekdate(2, 2004 + i, 12);
end
% определение дат платежей второго проекта
date_flow2 = date_flow1;
date_flow2(6) = lweekdate(2, 2009, 6);
date_flow2(7) = date_flow1(6);
% Сравнение по критерию прв
npv1 = pvvar(cf1, 0.1, date_flow1)
npv2 = pvvar(cf2, 0.1, date_flow2)
% Расчет будущей стоимости проектов
fv1 = fvvar(cf1, 0.1, date_flow1)
fv2 = fvvar(cf2, 0.1, date_flow2)
% Расчет внутренней ставки доходности проектов (критерий IRR)
int_rate1 = xirr(cf1, date_flow1, 0.13, 100)
int_rate2 = xirr(cf2, date_flow2, 0.13, 100)
```

В листинге по-разному учтены отсутствующие поступления по первому проекту на те даты, где по второму проекту они не нулевые. В первом случае платеж равен нулю, а дата задана, а во втором — пропущена дата. С одной стороны, это демонстрирует возможности использования функций, а с другой, имеет определенный смысл, который будет понятен при описании функции `payuni`. После выполнения файла-программы из листинга 20.1 получим следующие значения критериев:

```
npv1 =
11.5500
npv2 =
11.4774
fv1 =
18.6050
```

```
fv2 =
 18.4881
int_rate1 =
 0.1403
int_rate2 =
 0.1429
```

Проект 1 по критериям нетто стоимостей оказывается чуть лучше проекта 2 (особенно если провести сравнение в процентах), но по внутренней норме доходности второй проект кажется предпочтительнее.

Сравните эти проекты еще, заменив их эквивалентными проектами с равномерными ежегодными выплатами. Для чего воспользуйтесь функцией `payuni`:

```
поступление = payuni (поток_платежей, процентная_ставка)
```

Учтите, что во втором проекте надо первый платеж привести на конец года. Чтобы не вдаваться в тонкости процентных начислений, считайте, что его надо нарастить по ставке 10% в течение половины года, т. е. первый поток платежей следует предварительно модифицировать. И хотя очевидно, что этот показатель будет лучше для проекта 1, значения эквивалентных платежей дают дополнительный материал для анализа.

```
>> c1f1 = payuni(c1f1, 0.1)
c1f1 =
 3.0525
>> cf20 = cf2(1:5);
>> cf20(6) = 40 + 40*(1 + 0.5*0.1);
>> c1f2 = payuni(cf20, 0.1)
c1f2 =
 3.0346
```

Инвестор может ограничиться проведенными расчетами для принятия решения или провести дальнейший анализ на основе дополнительной информации о проектах. Мы не ставим своей задачей обосновывать выбор проекта для реализации.

Параметрами чувствительности к процентной ставке для одностороннего потока платежей (например, только поступлений от проекта) являются дюрация Маколея (или просто дюрация) и выпуклость. Обозначим текущую стоимость положительного потока платежей  $P$ . Если  $P$  рассматривать как достаточно гладкую функцию процентной ставки  $r$ , то при малых изменениях величины  $r$  на  $\delta r$  будет справедливо разложение цены в ряд Тейлора:

$$P(r + \delta r) = P(r) + \Delta P(r) = P(r) + P'(r)\delta r + \frac{1}{2}P''(r)(\delta r)^2 + o((\delta r)^3). \quad (20.6)$$

Удерживая два члена в приращении  $\Delta P(r)$ , можно выразить  $\Delta P(r)$  через модифицированную дюрацию  $D_m(r)$ , выпуклость  $V(r)$  и значение  $P(r)$ :

$$\Delta P(r) = P(r) \left\{ -D_m(r)\delta r + \frac{1}{2}V(r)(\delta r)^2 \right\} \quad (20.7)$$

и, таким образом, зная модифицированную дюрацию и выпуклость потока платежей, легко оценить изменение стоимости  $\Delta P(r)$  потока платежей при изменении ее доходности на величину  $\delta r$ . Если значение  $\delta r$  невелико, то можно ограничиться всего одним линейным членом. Модифицированная дюрация  $D_m(r)$  выражается через дюрацию Маколея  $D(r)$  и ставку  $r$  формулой:

$$D_m(r) = \frac{D(r)}{(1+r)}. \quad (20.8)$$

Дюрация (Маколея) и выпуклость рассчитываются через параметры потока платежей. Так, дюрация, имеющая смысл средневзвешенного времени полного платежа, определяется как:

$$D(r) = \sum_{i=1}^n t_i w_i, \quad w_i = \frac{pv(CF)_i}{P(r)}, \quad P(r) = \sum_{i=1}^n pv(CF_i), \quad t_i = i. \quad (20.9)$$

В формулах (20.9) текущая стоимость отдельного платежа потока, дисконтированная по ставке  $r$  с учетом времени платежа, обозначена  $pv(CF_i)$ , а для того, чтобы подчеркнуть временной фактор для времени  $i$ -го поступления, использовано обозначение  $t_i$ . Первая из формул (20.9) остается справедливой и при несовпадении платежей с началом или концом периода. Используя обозначения в формулах (20.9), выпуклость этого потока выражается следующим образом:

$$V(r) = \frac{1}{(1+r)^2} \left( \sum_{i=1}^n t_i^2 w_i + D(r) \right). \quad (20.10)$$

Для вычисления параметров чувствительности потока платежа к процентной ставке  $r$  предназначены функции `cfdur` и `cfconv`, имеющие одинаковые входные параметры:

`[дюрация, мод_дюрация] = cfdur(поток, доходность)`

и

`выпуклость = cfconv(поток, доходность)`

где поток — массив последовательных платежей в конце каждого периода и доходность — ставка доходности для одного периода. Дюрация и модифицированная дюрация измеряются в количестве периодов, а выпуклость — в периодах в квадрате (для согласования размерностей). Посчитайте дюрацию и выпуклость поступлений по двум ранее рассмотренным проектам на начало инвестирования. Для этого сначала модифицируйте потоки, а затем примените описанные функции.

Для второго потока поступлений в качестве периода приходится выбирать полгода (поскольку в последний год доход планируется получать дважды). Поэтому следует в поток добавить нулевые поступления в середине каждого предыдущего года. Дюрация будет рассчитана в количестве шести месячных периодов, т. е. в два раза превышающем количество 12-месячных (годовых) периодов. Значит и для первого денежного потока следует вычислять дюрацию в той же размерности. То же самое справедливо и для выпуклости. При расчете дюрации для первого потока в полугодиях надо задать нулевые поступления в середине каждого года. Остальные суммы дохода приведены в табл. 20.3 (выплаты по проекту учитывать не следует, т. к. они имеют противоположную направленность). Для второго денежного потока только в последнем году присутствует полугодовое поступление. При вычислениях для полугодового периода начислений надо учесть и то, что доходность (с точностью до величин малого порядка) в два раза меньше.

Результаты вычисления также дают дополнительную информацию для сравнения проектов:

```
>> cf11 = [0 0 0 47 0 53 0 30 0 20];
>> dur1 = cfdur(cf11, 0.1/2)
>> cx1 = cfconv(cf11, 0.1/2)

dur1 =
 6.1118

cx1 =
 42.9530

>> cf21 = [0 25 0 15 0 0 0 30 40 40];
>> dur2 = cfdur(cf21, 0.1/2)
>> cx2 = cfconv(cf21, 0.1/2)

dur2 =
 6.9540

cx2 =
 58.9449
```


**Примечание**

Казалось бы, можно воспользоваться очевидным фактом, что один год состоит из двух полугодий и, естественно, число полугодий при вычислении дюрации в два раза больше, чем число лет. Однако если после расчетов перевести полугодия в годы и сравнить с полученной дюрацией для исходного денежного потока в годах, то результаты будут отличаться на величину порядка  $O((dr)^2)$ , поскольку при дисконтировании применяются нелинейные множители. Проверьте это для рассматриваемого примера.

Из расчета видно, что второй поток чувствительнее, чем первый, к изменению процентной ставки (для него дюрация и выпуклость больше), и значит, имеет больший риск, связанный с возможными ее изменениями в период осуществления проекта.

## Расчеты по обслуживанию кредитов

Рассмотрим функции пакета для обслуживания долгов и кредитов. Пусть требуется принять решение о погашении долга 150 тыс. руб. в течение трех лет. На кредитном рынке процентная ставка по долгосрочному кредиту составляет 20% годовых. Рассмотрим схему погашения равными ежегодными платежами. При равных выплатах 20% от суммы составляют проценты по сумме оставшегося долга, а остальная часть идет в счет погашения суммы займа. В табл. 20.4 приведен расчет структуры таких выплат. Для иллюстрации частичного погашения кредита срочными платежами (одинаковыми суммами за год) возьмем сумму платежа, равную 70 тыс. руб. Хотя такой суммы недостаточно для полного погашения ссуды, но зато все расчеты прозрачны, и можно продемонстрировать более широкие возможности функций пакета.

*Таблица 20.4. Динамика погашения займа равными выплатами.*

| Год          | Остаток по займу, тыс. руб. | Платеж по сумме займа, тыс. руб. | Проценты по остатку, тыс. руб. | Полный платеж за год, тыс. руб. |
|--------------|-----------------------------|----------------------------------|--------------------------------|---------------------------------|
| 1            | 150                         | 40                               | 30                             | 70                              |
| 2            | 110                         | 48                               | 22                             | 70                              |
| 3            | 62                          | 57,6                             | 12,4                           | 70                              |
| <b>ИТОГИ</b> | 4,4                         | 145,6                            | 64,4                           | 210                             |

Для анализа можно использовать несколько функций, например: `amortize`, `annurate`, `payrep`. Многие параметры этих функций имеют одинаковый смысл, поэтому пояснение дается только, когда они встречаются первый раз.

Вся структура погашения кредита может быть вычислена с помощью функции `amortize`:

```
[за_кредит, за_проценты, долг, платеж] =
 amortize(ставка, периодов, сумма, остаток, момент)
```

Первых три входных аргумента обязательны и задают:

- ставка — процентную ставку по кредиту за один период;
- периодов — полное число периодов расчета по займу;
- сумма — сумма полученных заемщиком денежных средств.

Два последних параметров необязательны и в нашем случае не требуются. Их значение следующее:

- момент (по умолчанию 0) — флаг, равный 1 при выплатах в начале периода и 0 — в конце;
- остаток (по умолчанию 0) — та сумма, которая останется неоплаченной, если задается отрицательная величина (по табл. 20.4 это -4.4) или сумма, которую будет должен кредитор заемщику после погашения долга (например, если кредитором выступает банк и у заемщика открыт счет, то эта сумма будет зачислена на клиентский счет).

Первых три выходных параметра являются одномерными массивами размерности входного параметра `периодов` и содержат:

- `за_кредит` — платежи по сумме кредита за каждый период;
- `за_проценты` — выплаты по начисленным процентам за остаток кредита в каждый период;
- `долг` — оставшийся долг после текущего платежа;
- `платеж` — сумму общей разовой выплаты.

Вычислим эту структуру, для примера, по данным табл. 20.4:

```
[SC, SI, B, Payment] = amortize(0.2, 3, 150, -4.4)
```

`SC` =

|         |         |         |
|---------|---------|---------|
| 40.0000 | 48.0000 | 57.6000 |
|---------|---------|---------|

`SI` =

|         |         |         |
|---------|---------|---------|
| 30.0000 | 22.0000 | 12.4000 |
|---------|---------|---------|

`B` =

|          |         |        |
|----------|---------|--------|
| 110.0000 | 62.0000 | 4.4000 |
|----------|---------|--------|

```
Payment =
70.0000
```

Можно вычислить величину процентной ставки платежа за один период, воспользовавшись функцией `annurate` с тем же смыслом входных и выходных параметров:

```
ставка = annurate(периодов, платеж, сумма, остаток, момент)
```

Например, для платежа 70 тыс. руб. имеем ставку по кредиту меньше 20% поскольку остается неоплаченный долг 4.4 тыс. руб.:

```
>> annurate(3, 70, 150)
```

```
ans =
```

```
0.1891
```

Расчет только суммы одноразового платежа дает функция `rauper` с теми же параметрами:

```
платеж = rauper(ставка, периодов, сумма, остаток, момент)
```

Например, чтобы погасить полностью кредит, остаток надо положить нулем или опустить (значение по умолчанию 0):

```
>> rauper(0.2, 3, 150)
```

```
ans =
```

```
71.2088
```

Пересчитайте структуру обслуживания рассматриваемого займа со срочным платежом 71.2088 тыс. руб. и убедитесь, что кредит полностью покрывается.

Рассмотрим еще функции, позволяющие определить будущие накопления, например, в банке при периодических вкладах. Функция `annuterm` позволяет определить, за какое время накопится требуемая сумма:

```
периодов = annuterm(ставка, платеж, сумма, вклад, момент)
```

Значения одноименных параметров имеют тот же смысл, что и ранее описанных. Значение параметра `сумма` — величина вклада на начало периода накопления, а `вклад` — сумма вклада на конец всего периода.

Например, рассчитаем срок, когда была бы накоплена сумма 150 тыс. руб. из предыдущего примера, если бы не брался кредит, а ежегодно вносился вклад в размере 50 тыс. руб. при ставке 10% годовых (заметим, что на финансовых рынках ставки кредитования выше ставок вложения):

```
> annuterm(0.1, 50, 0, 150)
```

```
ans =
```

```
2.7527
```

Еще одна функция бывает полезна для планирования будущих выплат сумм, когда известно число периодов и сумма одного взноса:

```
вклад = fvfix(ставка, периодов, платеж, сумма, момент)
```

При ежеквартальных вкладах по 15 условных единиц и годовой ставке 10% в течение трех лет накопления без начальной суммы составят:

```
>> fvfix(0.1/4, 3*4, 15)
```

```
ans =
```

```
206.9333
```

### Примечание

В приведенных выше функциях период начисления необязательно совпадает с базовым периодом (календарным годом по умолчанию). Для расчета применяется формула начисления по сложным процентам с использованием номинальной процентной ставки  $r_0$  (20.3). Поэтому число периодов и норма начисления должны быть заданы с учетом этого факта.

## Расчеты по долговым ценным бумагам

Для целей иллюстрации возможностей MATLAB будем различать только бескупонные и купонные облигации, хотя сегмент рынка долговых ценных бумаг включает финансовые инструменты, называемые по разным причинам иначе (векселя, зеро, бонны, ноты и др.). Более того, не будем делать различия между государственными, банковскими и корпоративными бумагами и вникать в нюансы обращения рыночных активов.

Пакет MATLAB предоставляет большое количество функций для анализа рынка облигаций, детальное обсуждение которых требует отдельного изложения. Здесь рассмотрим примеры типовых расчетов.

### Дисконтные активы

Все финансовые инструменты, доход по которым выплачивается при погашении актива по номиналу, будем называть бескупонными облигациями или зеро, хотя это и не совсем корректно. Для привлекательности бескупонных облигаций их доходность должна быть больше банковских процентных ставок для накопления вложений, и тем больше, чем больше срок обращения. Бескупонные облигации погашаются по номиналу и размещаются со скидкой (*дисконтом*), поэтому при их анализе часто применяется учетная ставка, а в именах функций используется префикс или постфикс *disc*:

`discrate`, `fvdisc`, `prdisc`, `ylddisc`. Все эти функции вычисляют одну из величин, входящих в формулу (20.6), через другие:

$$P = \frac{S}{\left(1 + \frac{t}{T}y\right)} = S \left(1 - \frac{t}{T}d\right), \quad (20.11)$$

где  $P$  — цена;  $S$  — номинальная стоимость или будущая стоимость;  $y$  — доходность к погашению;  $d$  — годовая учетная (дисконтная) ставка;  $T$  — базисный период;  $t$  — число дней до погашения или реализации.

### Примечание

В зависимости от того, какое значение рассчитывается, параметры могут иметь разный смысл, понятный пользователю, например, цена может быть рыночной или "справедливой" (теоретической). Более того, под рыночной ценой расчётчик может также понимать следующие величины: средневзвешенная за некоторый период, цена последней сделки, цена закрытия торговой сессии на выбранной им бирже и т. п. Дополнительно этими функциями можно пользоваться и для расчетов при начислении по простым процентам, задавая нужные входные параметры.

К перечисленным ранее функциям можно обратиться следующим образом:

`дисконт` = `discrate`(`дата`, `погашение`, `номинал`, `цена`, `базис`)

`стоимость` = `fvdisc`(`дата`, `реализация`, `цена`, `дисконт`, `базис`)

`цена` = `prdisc`(`дата`, `погашение`, `номинал`, `дисконт`, `базис`)

`доходность` = `ylddisc`(`дата`, `погашение`, `номинал`, `цена`, `базис`)

Входные и выходные параметры этих функций имеют следующий смысл:

- `дата` — дата, на которую производится расчет;
- `погашение` — дата погашения ценной бумаги;
- `реализация` — дата реализации ценной бумаги до срока погашения;
- `номинал` — номинальная стоимость актива;
- `цена` — цена актива на дату расчета;
- `дата, доходность` — годовая норма доходности финансового инструмента;
- `стоимость` — будущая стоимость актива при заданной учетной ставке;
- `базис` — флаг, определяющий структуру календарного года (необязательный параметр, 0 по умолчанию).

Рассмотрим небольшой пример на использование функций этого ряда. При этом возьмем актив со сроком обращения меньше одного базового периода, на котором более рельефно проявляются особенности вычислений с использованием формулы (20.11). Инвестор имеет пакет абстрактных бескупонных облигаций номиналом  $S = 100\ 000$ , выпущенных в обращение 10 декабря 2003 г. со сроком погашения 10 октября 2005 г., которые он планирует реализовать до даты погашения. Облигации были размещены по цене  $P_0 = 87\ 000$ , т. е. со скидкой 13% (курсовая стоимость 87%). Воспользуйтесь функцией `discrate` для вычисления годовой дисконтной ставки для актива:

```
>> d0 = discrate('10-dec-2003', '10-oct-2004', 100000, 87000, 0)
d0 =

```

0.1560

Расчетная годовая дисконтная ставка больше 13%, поскольку она учитывает фактор времени (12 месяцев в году по сравнению с 10 месяцами обращения облигации). Такая большая скидка при размещении финансового инструмента выбрана для большей наглядности результатов вычислений.

На рынке облигаций обычно оперируют с курсовыми стоимостями актива, выраженным в процентах от номинала (будем обозначать эти величины  $K$ ). Пусть текущая дата 7 июля 2004 г., и на рынке уровень цен следующий: цена спроса (лучшая котировка на покупку — *bid price*)  $K_b = 95.5\%$ , цена предложения (лучшая котировка на продажу — *ask price*)  $K_a = 96.5\%$  и цена последней сделки  $K_m = 96\%$ . Требуется оценить, насколько рыночные цены отличаются от справедливой (в процентах от номинала), найти доходность и дисконтную ставку для каждой цены на момент расчета.

Для решения этой задачи напишите функцию, в которой определите следующие входные параметры: дату реализации и рыночную цену. В этом случае можно решать такую задачу неоднократно для разных дат и уровней цен. Воспользуемся линейной моделью справедливой цены  $P_t$ , определяемой из условия, что она линейно растет от цены размещения  $P_0$  в момент выпуска актива в обращение до номинала в момент погашения:

$$P_t = P_0 + \frac{S - P_0}{T_0} (T_0 - t) = S - \frac{S - P_0}{T_0} t,$$

где  $T_0$  — период обращения с момента выпуска до момента погашения;  $t$  — число дней, оставшихся до погашения. Получить эту цену можно с использованием функции `prdisc`. Пример функции для оценки рассматриваемого финансового инструмента приведен в листинге 20.2.

**Листинг 20.2. Файл-функция profit\_zero расчета по дисконтной облигации**

```

function [yld, disc, prof] = profit_zero(c_date, c_price)
%задание постоянных параметров для облигации
M_Date = '10-oct-2004';
I_Date = '10-dec-2003';
p0 = 87000;
face = 100000;
b = 0;
%расчет исходной ставки дисконта
d0 = disbrate(I_Date, M_Date, face, p0, b);
% расчет справедливой цены
pt = prdisc(c_date, M_Date, 100000, d0, b);
%вычисление рыночной цены актива
price = face*c_price/100;
%доходность
yld = 100*ylddisc(c_date, M_Date, face, price, b);
%годовая дисконтная ставка
disc = 100*disbrate(c_date, M_Date, face, price, b);
%отклонение от справедливой цены в процентах от номинала
prof = 100*(price - pt)/face;

```

Для проверки функции profit\_zero воспользуйтесь программой из листинга 20.3.

**Листинг 20.3. Файл-программа zero\_test**

```

format short g
now_date = '07-jul-2004';
Km = 96
Ka = 96.5
Kb = 95.5
[yldb, discdb, profdb] = profit_zero(now_date, Kb)
[yldask, discask, profask] = profit_zero(now_date, Ka)
[yldm, discm, profm] = profit_zero(now_date, Km)

```

После выполнения программы из листинга 20.3 получаются следующие оценки:

```
>> zero_test
yldbid =
 18.154
discbid =
 17.337
profbid =
 -0.45082
yldask =
 13.973
discask =
 13.484
profask =
 0.54918
yldm =
 16.053
discm =
 15.411
profm =
 0.04918
```

Отрицательное значение profbid показывает, что рыночная цена, соответствующая котировке на покупку актива, меньше справедливой цены.

### Примечание

В этом разделе рассмотрены функции, применяемые для краткосрочных активов, т. к. на практике формула (20.11) для начисления по простым процентам используется в течение одного базового периода. Для долгосрочных облигаций, когда используется методика начисления по сложным процентам, следует применять функции из следующего раздела, задавая нулевую купонную ставку.

## Купонные облигации

Обращение купонных облигаций предусматривает промежуточные выплаты их владельцу помимо номинальной стоимости в момент погашения. Дополнительные доходы определяются купонной ставкой и периодичностью вы-

платы доходов. При расчетах по купонным облигациям надо учитывать, что стоимость купона равномерно возрастает в межкупонный период. Поэтому полная стоимость ("грязная цена") облигации в момент, не совпадающий с датой купонного платежа, складывается из двух частей: накопленного купона к расчетной дате и чистой (или "очищенной") цены. Это важно учитывать, например, при уплате налога с дохода. Если речь идет о купле-продаже облигации, то стоимость накопленного купона является доходом продавца и не включается в доход покупателя.

Для иллюстрации возможностей пакета рассмотрим облигации с постоянной купонной ставкой и выплатой  $m$  раз в году. Будем всюду далее использовать следующие обозначения:

- $S$  — номинальная стоимость облигации (номинал), т. е. сумма, выплачиваемая эмитентом владельцу облигации при погашении;
- $c$  — годовая купонная ставка (десятичная дробь или %), определяет процент от номинала для общей суммы годовых дополнительных выплат по облигации, равную  $S_c$ ;
- $m$  — число выплат по купону в году (определяет периодичность, например,  $m=12$  — ежемесячно,  $m=4$  — ежеквартально,  $m=2$  — полугодовые выплаты,  $m=1$  — ежегодные);
- $c_m = \frac{c}{m}$  — процент выплаты по одному купону;
- $T_c$  — межкупонный период (число дней между двумя последовательными купонными платежами);
- $y$  — доходность облигации к погашению (стандартно эту величину обозначают YTM);
- $t$  — количество дней, оставшихся до ближайшего купонного платежа;
- $n$  — число оставшихся купонных платежей до погашения;
- $P$  — текущая цена облигации (так же как и ранее, может иметь различный смысл в зависимости от типа расчета).

Цена облигации  $P$  выражается формулой:

$$P = S \left( \sum_{i=1}^n \frac{c_m}{\left(1 + \frac{y}{m}\right)^{\tau_i}} + \frac{1}{\left(1 + \frac{y}{m}\right)^{\tau_n}} \right), \quad (20.12)$$

где  $i$  — номер купонного периода, отсчитываемый от расчетной даты;  $\tau_i$  — фактор времени для дисконтирования будущего платежа на заданный момент. Для простейшего случая расчета  $P$  на начало купонного периода и  $m=1$  временной фактор  $\tau_i = i$  (сравните с формулой (20.4) при  $y=r$ ). В общем случае, если  $T$  — число дней в базисном периоде, применяется формула

$$\tau_i = i - \frac{T_c - t}{T}. \quad (20.13)$$

В пакете MATLAB большинство функций ориентировано на американский стандарт SIA, при котором в качестве расчетного (с точки зрения процентных начислений — базового) берется половина календарного года. Поэтому формула (20.13) внутри функций пакета используется в модифицированном виде, и далее в этом разделе приводится краткое разъяснение этого вопроса. Основная часть функций MATLAB также ориентирована на указанный стандарт.

Большинство функций для расчета ценных бумаг с фиксированными платежами использует одни и те же параметры, для которых будем использовать одни и те же имена при вызове функций (в скобках указаны ранее введенные математические обозначения):

- цена ( $P$ ) — цена облигации;
- доходность ( $y$ ) — доходность к погашению;
- С\_дата — текущая или расчетная дата, относительно которой осуществляется расчет;
- М\_дата — дата погашения облигации;
- I\_дата — дата начала обращения облигации;
- F\_дата — дата первой купонной выплаты по облигации (используется для расчета дат купонных платежей);
- L\_дата — дата последней купонной выплаты по облигации (по умолчанию совпадает со значением М\_дата и используется для расчета дат купонных платежей, если не задана F\_дата);
- S\_дата — дата начала начислений по облигации (она используется для определения поступлений в будущем периоде, например, если планируется приобретение облигации, т. е. фактически для форвардного финансового инструмента — по умолчанию совпадает со значением С\_дата);
- номинал ( $S$ ) — номинальная стоимость, т. е. сумма, выплачиваемая при погашении облигации без учета купонного платежа;

- купон ( $c$ ) — годовая купонная ставка;
- периодов ( $m$ ) — число купонных периодов в календарном году;
- базис — флаг, задающий структуру календарного года;
- правило — флаг, задающий правило: 1 (по умолчанию) — учитывать последний день месяца, 0 — не учитывать. Если дата погашения является последним числом месяца длительностью менее 31 дня (30 или 28/29), и этот флаг установлен в 1, то даты платежей по купонам будут последними днями месяца. В противном случае выплаты приходятся на одно и тоже число месяца.

Для пояснения принципов использования функций возьмем модельную купонную облигацию номиналом 1000 у. е. с годовой купонной ставкой 8% (0.08) сроком обращения 1 год, ежеквартальной выплатой по купону, датой начала обращения 01 декабря 2003 г. и сроком погашения 30 ноября 2004 г. Для определенности будем использовать фактическую структуру года (базис = 0).

Функция `cfdates` позволяет найти даты платежей по купонам:

```
даты_выплат = cfdates(С_дата, М_дата, периодов, базис, правило, ...
 I_дата, F_дата, L_дата, S_дата)
```

Функция возвращает массив дат выплат по облигации во внутреннем формате следующих после расчетной даты. Входные параметры соответствуют ранее введенным обозначениям. Обязательными параметрами являются первых два, но, как правило, функция вызывается минимум с тремя параметрами.

В листинге 20.4 приведен пример использования функции `cfdates` для иллюстрации использования правила конца месяца. Для вычисления всех дат платежей в качестве расчетной даты можно взять любую дату в первом периоде, например, дату начала обращения. Для наглядности все параметры задаются до обращения к функции. В листинге функция используется дважды с разным значением флага правила.

#### Листинг 20.4. Файл-программа `bond_dates` расчета дат выплат по облигации

```
%Задание дат для расчета
I_Date = '01-dec-2003';
M_Date = '30-nov-2004';
%Число купонных платежей
m = 4;
%Фактическая структура года
b = 0;
```

```
% Учитывать правило последнего дня месяца
m_end = 1;
CFDates = cfdates(I_Date, M_Date, m, b, m_end);
Dates_1 = datestr(CFDates, 1)

% Не учитывать правило последнего дня месяца
m_end = 0;
CFDates = cfdates(I_Date, M_Date, m, b, m_end);
Dates_0 = datestr(CFDates, 1)
```

После выполнения файл-программы получим даты платежей по облигации:

```
Dates_1 =
29-Feb-2004
31-May-2004
31-Aug-2004
30-Nov-2004

Dates_0 =
29-Feb-2004
30-May-2004
30-Aug-2004
30-Nov-2004
```

Рассмотрим далее облигацию со сроками платежей, совпадающими с последними календарными днями месяца (массив Dates\_1). Возьмем для дальнейшей иллюстрации расчетную дату 15 мая 2004 г. Функции cpndaten и cpndatep находят ближайшую дату следующего или предыдущего платежа, а функции cpndaysn и cpndatep — число дней до ближайшей будущей и прошлой купонной даты. Функция cpncount позволяет найти число оставшихся купонных периодов. Все они имеют одинаковую форму вызова:

```
След_дата = cpndaten(С_дата, М_дата, периодов, базис, правило, ...
 I_дата, F_дата, L_дата)
Пред_дата = cpndatep(С_дата, М_дата, периодов, базис, правило, ...
 I_дата, F_дата, L_дата)
след_дней = cpndaysn(С_дата, М_дата, периодов, базис, правило, ...
 I_дата, F_дата, L_дата)
пред_дней = cpndaysp(С_дата, М_дата, периодов, базис, правило, ...
 I_дата, F_дата, L_дата)
Число_выплат = cpncount(С_дата, М_дата, периодов, базис, правило, ...
 I_дата, F_дата, L_дата)
```

Значения параметров и правила их использования такие же, как и для функции `cfdates`. Для рассматриваемого примера имеем:

```
>> next = datestr(cpndaten('15-may-2004', '30-nov-2004', 4))
next =
31-May-2004
prev = datestr(cpndatep('15-may-2004', '30-nov-2004', 4))
prev =
29-Feb-2004
days_N = cpndaysn('15-may-2004', '30-nov-2004', 4)
days_N =
 16
days_P = cpndaysp('15-may-2004', '30-nov-2004', 4)
days_P =
 76
periods = cpncount('15-may-2004', '30-nov-2004', 4)
periods =
 3
```

Теперь обратимся к функциям для расчетов стоимостных параметров. Наша модельная облигация, обладая годовой купонной ставкой 8%, должна приносить доход по купону – 20 у. е. Сведем параметры облигации в табл. 20.5.

*Таблица 20.5. Параметры модельной облигации*

| Дата платежа | Тип платежа         | Сумма платежа, у. е. |
|--------------|---------------------|----------------------|
| 01-dec-2003  | Размещение          |                      |
| 29-Feb-2004  | Купон 1             | 20                   |
| 15-May-2004  | Расчетная дата      |                      |
| 31-May-2004  | Купон 2             | 20                   |
| 31-Aug-2004  | Купон 3             | 20                   |
| 30-Nov-2004  | Купон 4 и погашение | 1020                 |

Ключевую роль при вычислениях по облигациям с фиксированными поступлениями играет функция `cfamounts`, которая рассчитывает параметры облигации, связанные с потоком выплат:

```
[выплаты, даты, факторы_времени, флаги] =
cfamounts(купон, С_дата, М_дата, периодов, базис, правило, ...
I_дата, F_дата, L_дата, S_дата, номинал)
```

Все входные параметры описаны ранее. Вместо необязательных входных параметров при обращении к функции cfamounts можно задать пустые массивы. Выходные параметры представляют собой массивы размерностью на единицу больше, чем число оставшихся купонных платежей. Они определяют:

- выплаты — величины платежей на дату выплаты;
- даты — расчетная дата и следующие за ней даты выплат;
- факторы\_времени — временные дисконтирования на расчетный день для каждой суммы;
- флаги — определяют тип каждого платежа.

Составьте функцию bond\_cf для расчетов стоимостных параметров модельной облигации, например так, как представлено в листинге 20.5.

#### Листинг 20.5. Файл-программа bond\_cf расчета платежей по облигации

```
I_Date = '01-dec-2003';
C_Date = '15-may-2004';
M_Date = '30-nov-2004';
b = 0;
m = 4;
c = 0.08;
S = 1000;
m_end = 1;
[paym, CFDates, d_fact, flags] = cfamounts(c, C_Date, M_Date, m, ...
b, m_end, [], [], [], [], S);
Dates_1 = datestr(CFDates, 1)
fprintf('\n payment \n');
fprintf('%12.3f ',paym);
fprintf('\n d_factors \n');
fprintf(' %10.5f',d_fact);
fprintf('\n flags \n');
fprintf(' %6.2f',flags);
```

После ее выполнения получим следующий результат:

```
Dates_1 =
15-May-2004
31-May-2004
31-Aug-2004
30-Nov-2004
```

```

payment
-16.522 20.000 20.000 1020.000
d_factors
0.00000 0.08743 0.58696 1.08743
flags
0.00 3.00 3.00 4.00

```

Моменты времени, на которые произведены расчеты, и суммы поступлений, кроме первого, очевидны (табл. 20.5). Значение первого платежа показывает сумму накопленного текущего купона на расчетную дату. Знак минус тоже имеет смысловую нагрузку. Если в расчетную дату происходит сделка купли-продажи, то эта сумма является доходом продавца (например, для учета налогообложения), но заплачена покупателем продавцу как часть цены. Она будет возмещена при погашении купона в ближайшую дату выплаты.

Элементы массива `d_factors` представляют величины  $\tau_i$ . Для пояснения расчета этих величин приведем табл. 20.6, где указана дата начала обращения облигации, необходимая для понимания расчета.

*Таблица 20.6. К расчету факторов времени для облигации*

| Даты в расчете | Смысл даты         | Число дней до платежа, $t_i$ | База расчета, $T_i$ | Фактор времени, $\tau_i$ | Формула расчета            |
|----------------|--------------------|------------------------------|---------------------|--------------------------|----------------------------|
| 01-Dec-2003    | Начало обращения   |                              |                     |                          |                            |
| 15-May-2004    | Расчетная дата     |                              |                     |                          |                            |
| 31-May-2004    | Купон 2            | 16                           | 183                 | 0.0874317                | $\tau_1 = \frac{t_1}{T_1}$ |
| 31-Aug-2004    | Купон 3            | 108                          | 184                 | 0.5869565                | $\tau_2 = \frac{t_2}{T}$   |
| 30-Nov-2004    | Погашение, купон 4 | 199                          | 183                 | 1.0874317                | $\tau_3 = 1 + \tau_1$      |

Число дней до платежа определяется стандартно, как интервал между двумя датами.

В качестве базы расчета (базы начисления) берется полугодовой период в соответствии со стандартом SIA. Поэтому продолжительность базы начисления не одинакова: она равна реальному числу дней в шести месяцах, заканчивающихся датой ближайшей выплаты по купону. Например, если бы ближайшая выплата состоялась 10 июня 2004 г., то полугодовой период начинался бы с 10 декабря 2003 г.

Первоначально факторы времени рассчитываются только в пределах начального полугодового периода, т. е. величины меньшие 1. Для нашего случая это первых два фактора. Для каждого следующего периода они циклически увеличиваются на единицу. В нашем случае фактор времени для 30 ноября 2004 г. определяется как первый фактор для начального полугодия плюс 1. Если бы потребовалось рассчитывать факторы времени далее, то формулы расчета были бы следующими:  $\tau_4 = 1 + \tau_2$ ,  $\tau_5 = 2 + \tau_1$ ,  $\tau_6 = 2 + \tau_2$ ,  $\tau_7 = 3 + \tau_1$  и т. д.

Флаги указывают на тип каждого вычисленного поступления: 0 — выплаты не производятся, 3 — получение купонной суммы, 4 — погашение по номиналу с купонным платежом (если дата погашения совпадает с датой купонного платежа).

Хотя функция `cfamounts` дает все необходимые величины для определения текущей нетто стоимости платежей (справедливой цены облигации), для ее вычисления лучше использовать функцию `bndprice`, не требующую знания расчетной формулы:

```
[цена, накопление] = bndprice(доходность, купон, С_дата, М_дата, ...
периодов, базис, правило, I_дата, F_дата, L_дата, S_дата, номинал)
```

Последний и четыре первых входных параметра являются обязательными. Для остальных можно задать пустые массивы, и они примут значения по умолчанию. Выходной параметр `цена` содержит очищенную цену облигации. Второй выходной параметр дает накопленную сумму на расчетную дату по текущему купону (это та же величина, что и первый элемент массива в параметре функции `cfamounts`, но с положительным знаком). Для модельной облигации (при условии, что ранее использованные переменные определены и доходность к погашению оценивается как 9%) имеем:

```
>> [price, accrue] = bndprice(0.09, c, C_Date, M_Date, m, b, ...
m_end, [], [], [], [], S)
price =
 995.2187
accrue=
 16.5217
```

На рынке облигаций важной является задача определения доходности к погашению по цене, смысл которой может быть разным, о чем говорилось ранее в этой главе. Для этого предназначена функция `yldbond`:

```
доходность = bndyield(цена, купон, С_дата, М_дата, периодов, ...
базис, правило, I_дата, F_дата, L_дата, S_дата, номинал)
```

Эта функция из уравнения (20.12) находит  $y$ .

Вычислите, какова доходность модельной облигации, если ее рыночная цена 975 у. е:

```
>> Yield = bndyield(975, c, С_Date, М_Date, m, b, m_end, [], [], ...
[], [], S)
Yield =
0.1303
```

Для инвестора с точки зрения принятия решений нужны показатели чувствительности цены облигации к изменению ее доходности: дюрация (либо модифицированная дюрация) и выпуклость потока поступлений в период обращения облигации (см. формулы (20.6—20.10), в которых  $r$  следует заменить на  $y$ ). Для вычисления дюрации и выпуклости потока платежей были рассмотрены функции `cfdur` и `cfconv`, но они предназначены для вычисления характеристик аннуитетов и не подходят для рыночных активов. Для облигаций можно использовать несколько разных функций с префиксом `bnd`, имеющих схожий интерфейс. Дюрацию при заданной доходности можно вычислить функцией `bnddury`:

```
[год_дюрация, пол_дюрация, мод_дюрация] =
bnddury(доходность, купон, С_дата, М_дата, периодов, базис, ...
правило, I_дата, F_дата, L_дата, S_дата, номинал)
```

Входные параметры имеют тот же смысл, что и для ранее описанных функций. Три выходных параметра рассчитывают разные дюрации:

- `год_дюрация` — дюрация Маколея при годовом расчетном периоде;
- `пол_дюрация` — дюрация Маколея при полугодовом расчетном периоде;
- `мод_дюрация` — модифицированная дюрация для полугодовой доходности.

Выпуклость рассчитывается с помощью функции `bndconvv` с теми же входными параметрами:

```
[год_выпуклость, пол_выпуклость] =
Bndconvv (доходность, купон, С_дата, М_дата, периодов, базис, ...
правило, I_дата, F_дата, L_дата, S_дата, номинал)
```

Выходные параметры `год_выпуклость` и `пол_выпуклость` дают значение выпуклости в расчете на годовой и полугодовой расчетные периоды. Для модельной облигации имеем:

```
>> [ModDur, YearDur, PerDur] = bnddury(0.09, c, C_Date, M_Date, ...
m, b, m_end, [], [], [], [], S)
ModDur =
0.5063
YearDur =
0.5290
PerDur =
1.0581
>> [YearConv, PerConv] = bndconvy(0.09, c, C_Date, M_Date, ...
m, b, m_end, [], [], [], [], S)
YearConv =
0.5040
PerConv =
2.0158
```

То, что полугодовая дюрация примерно в два раза больше годовой, а выпуклость соответственно в четыре, объясняется тем, что они являются коэффициентами при линейном и квадратичном слагаемых в представлении  $\Delta P(y)$  от  $\delta y$  (формула (20.7) при замене  $\delta r$  на  $\delta y$ ).

### Примечание

Все рассмотренные функции допускают в качестве исходных аргументов задавать массивы и тем самым одновременно рассчитывать параметры для пакета разных облигаций. Выходными параметрами также будут массивы, содержащие поэлементно характеристики каждой облигации.

## Портфельный анализ рисковых активов

В заключение раздела рассмотрим средства пакета для анализа портфелей ценных бумаг средствами пакета MATLAB. Основные функции портфельного анализа основаны на моделях Марковица и Тобина и их модификациях. Вначале кратко поясним суть вопроса. В главе 16 рассматривался иллюстративный пример для определения одной точки на эффективной границе Марковица.

В общей постановке задача портфельного инвестирования для инвестора формулируется так. Инвестор рассматривает возможность вложения капи-

тала в некоторый набор акций (рисковых активов), выбранный им из личных соображений, и желает составить портфель ценных бумаг так, чтобы его доходность была бы наибольшей при фиксированном риске или риск был бы наименьшим при выбранной доходности портфеля. В рассматриваемой модели за меру риска принимается среднеквадратическое отклонение доходности ценных бумаг или портфеля от ее математического ожидания. На выбор портфеля влияет также отношение инвестора к риску, т. е. насколько он готов увеличить рискованность вложения с целью получить большую доходность.

В простейшем случае поиск наилучшего портфеля состоит из двух шагов: построение эффективной границы Марковица с использованием функций `frontcon` или `portopt` и выбор портфеля с помощью функции `portalloc` в соответствии с отношением инвестора к риску и значениями процентных ставок для безрискового вложения и заимствования.

Для решения поставленной задачи инвестор должен сделать оценки ожидаемой доходности для каждой акции и матрицы ковариаций для выбранных акций на период инвестирования. Расчетная модель опирается на ряд гипотез, например, что активы сколь угодно делимы, в конце периода портфель реализуется, горизонт инвестирования для всех участников рынка одинаков, операционные издержки отсутствуют или ими можно пренебречь и ряд других. Соответственно при анализе полученных результатов следует делать соответствующие поправки.

Пусть инвестор рассматривает вопрос о составлении портфеля из  $n$  рисковых активов, ожидаемые доходности которых представлены вектором  $y$ , а их ковариации — матрицей  $V$ :

$$y = \{y_i\}_{i=1}^n, \quad V = \{v_{i,j}\}_{i=1, j=1}^{n,n}.$$

Портфель определяется вектором  $x = \{x_i\}$  ( $i = 1, \dots, n$ ), компоненты которого указывают долю средств для покупки акций, поэтому условие

$$\sum_{i=1}^n x_i = 1$$

означает вложения всех выделенных для инвестирования средств в портфель. Обычно предполагается, что инвестор не имеет возможности осуществлять короткие продажи (продажи без покрытия), т. е. продавать заимствованные бумаги. Тогда компоненты вектора  $x$  должны быть неотрицательными ( $x_i > 0$ ). Однако если такая возможность существует, то указанное ограничение не ставится, либо оно может быть сформулировано для каких-то отдельных ценных бумаг. Более того, MATLAB имеет средства

формировать более сложные ограничения на компоненты  $x_i$ , которые будут рассмотрены далее.

Ожидаемая доходность портфеля  $y_p$  и его среднеквадратическое отклонение (риск)  $\sigma_p$  определяются формулами:

$$y_p = x^T y, \quad \sigma_p = \sqrt{x^T V x}.$$

Если  $y_{\min}$  и  $y_{\max}$  — наименьшая и наибольшая доходность акций в портфеле, то  $y_p \in [y_{\min}, y_{\max}]$ . Если для каждого значения доходности из указанного интервала решить задачу, рассмотренную в главе 16 и построить график  $y_p(\sigma_p)$ , то получится выпуклая кривая (см. рис 20.1), называемая "путь" Марковица, часть которой, лежащая между точками 2 и 3, является эффективной границей (или эффективным фронтом). Каждой точке на этой границе  $(y_p^0, \sigma_p^0)$  соответствует оптимальный портфель в том смысле, что при фиксированной доходности  $y_p^0$  невозможно составить портфель с меньшим риском или при фиксированном риске  $\sigma_p^0$  невозможно найти портфель с большей доходностью. Точки, лежащие на участке 1 – 2, не отвечают оптимальному портфелю, поскольку при том же риске ( $\sigma_p$ ) есть портфель с большей доходностью для вышеприведенной точки на участке 2 – 3. В силу этого точка 2 ( $y_{\text{low}}$ ) определяет нижнее значение для доходности портфеля.

## Построение эффективной границы рисковых активов

Для большей наглядности возьмем модельный пример, в котором активы имеют существенный разброс параметров, не характерный для реального рынка. Все процентные величины будем задавать десятичными дробями, как того требуют функции пакета при их вызове.

Пусть имеется шесть рисковых активов. Поименуем их акции С1, Е1, Е2, Е3, М1, М2 и пронумеруем от 1 до 6, так, что 3-й финансовый инструмент это акции Е2. Для каждого актива инвестор оценил ожидаемую доходность  $y_i$ , риск  $\sigma_i$  (среднеквадратическое отклонение) и коэффициенты корреляции  $\rho_{ij}$ .

Запишем данные в матричной форме:

$$y = \begin{bmatrix} 0.108 \\ 0.136 \\ 0.144 \\ 0.151 \\ 0.187 \\ 0.191 \end{bmatrix}; \quad \sigma = \begin{bmatrix} 0.12 \\ 0.135 \\ 0.15 \\ 0.18 \\ 0.20 \\ 0.205 \end{bmatrix}; \quad \rho = \begin{bmatrix} 1 & 0.32 & 0.4 & -0.3 & 0.31 & -0.18 \\ 0.32 & 1 & 0.45 & 0.43 & 0.29 & 0.3 \\ 0.4 & 0.45 & 1 & 0.4 & 0.46 & -0.29 \\ -0.3 & 0.43 & 0.4 & 1 & 0.41 & 0.4 \\ 0.31 & 0.29 & 0.46 & 0.41 & 1 & 0.25 \\ -0.18 & 0.3 & -0.29 & 0.4 & 0.25 & 1 \end{bmatrix}.$$

Функции пакета для анализа портфелей используют матрицу ковариаций, а не коэффициентов корреляции. Для получения матрицы ковариаций воспользуйтесь функцией corr2cov:

```
A_ковариации = corr2cov(A_СрКвОтклонений, A_корреляции)
```

выходным параметром которой является матрица ковариаций случайных величин A\_ковариации, построенная по вектору среднеквадратических отклонений A\_СрКвОтклонений и матрице коэффициентов корреляции A\_корреляции.

Далее, при описании функций одинаковым параметрам будем давать одни и те же имена и пояснить их при первом использовании. Поскольку названия величин для портфелей и активов могут быть одноименными, будем снабжать параметры функций префиксом, например, A — для актива, П — для портфеля.

### Примечание

Если первоначально оценена матрица ковариаций, то никаких предварительных преобразований исходных данных не требуется.

Воспользуйтесь листингом 20.6 и сформируйте в рабочей среде для дальнейшего использования вектор доходностей и матрицу ковариаций для активов, выбранных для примера.

#### Листинг 20.6. Файл-программа Tesr\_portf подготовки данных для анализа портфеля рисковых активов

```
%Ожидаемые доходности активов
A_yld = [0.108 0.136 0.144 0.151 0.187 0.191];
%Среднеквадратические отклонения активов
A_dev = [0.12 0.135 0.15 0.18 0.20 0.205];
%Матрица коэффициентов корреляции
A_corr = [1 0.32 0.4 -0.3 0.31 -0.18;
```

```

0.32 1 0.45 0.43 0.29 0.3;
0.4 0.45 1 0.4 0.46 -0.29;
-0.3 0.43 0.4 1 0.41 0.4;
0.31 0.29 0.46 0.41 1 0.25;
-0.18 0.3 -0.29 0.4 0.25 1];

```

%вычисление матрица ковариаций активов

```
A_cov = corr2cov(A_dev, A_corr);
```

### Примечание

При конструировании собственных тестов помните, что матрица коэффициентов корреляции (и ковариаций) должна быть симметричной и положительно определенной. Такая проверка в функциях пакета MATLAB отсутствует, и, если вы ошибетесь при вводе данных, то можете получить результаты, противоречащие здравому смыслу.

Построение эффективной границы Марковица состоит в нахождении двух массивов точек для доходности и риска, а также для каждой пары "доходность-риск" векторы долей вложений в каждый актив. Для нахождения границы можно использовать две функции `frontcon` и `portopt`, отличающиеся только способом задания дополнительных ограничений на доли активов в портфеле. Поскольку функция `frontcon` использует `portopt`, то начнем с функции `portopt`, которая вызывается следующим образом:

```
[П_риск, П_доходности, П_доли] =
portopt (A_доходности, A_ковариации, П_число, П_доходности, ...
 ...
Множество_ограничений)
```

Аналогичное обращение к функции `frontcon`:

```
[П_риск, П_доходности, П_доли] =
frontcon(A_доходности, A_ковариации, П_число, П_доходности, ...
 ...
A_ограничения, Г_активов, Г_ограничения)
```

Обязательными входными параметрами являются два первых: вектор ожидаемых доходностей активов `A_доходности` и их матрица ковариаций `A_ковариации`. Два следующих параметра определяют точки для расчета эффективной границы, причем один из них должен быть пустым массивом, если определен второй. Если задан параметр `П_число`, определяющий число точек на эффективной границе, то рассчитываются равноотстоящие точки на интервале  $[y_{\text{low}}, y_{\text{max}}]$ . Если определить массив точек из интервала  $[y_{\text{low}}, y_{\text{max}}]$ , то будут рассчитаны точки эффективной границы для этих значений. Если опущены оба параметра или для них заданы пустые значения, то по умолчанию расчет проводится для десяти равноотстоящих точек

интервала  $[y_{\text{low}}, y_{\text{max}}]$ . Поскольку первоначально не известно значение  $y_{\text{low}}$ , то его можно рассчитать, обратившись к функции со значением  $\text{P\_число} = 2$ . Другой способ получения  $y_{\text{low}}$  продемонстрирован в приводимом ниже примере. Последний параметр `Множество_ограничений` для `portopt` представляет матричную форму дополнительных ограничений на весовые коэффициенты активов  $x_i$  в портфеле. Долевые ограничения на состав портфеля в `frontcon` разбиты на два вида: двусторонние на каждый вес  $x_i$  (для каждого актива в отдельности) и на сгруппированные активы по разным признакам. Параметр `Г_активов` определяет состав группы, а параметр `Г_ограничения` — задает ограничения для группы (задание дополнительных ограничений рассмотрено далее в разд. "Дополнительные ограничения при анализе портфелей" этой главы).

Выходные параметры у функций одинаковы и содержат характеристики портфелей на эффективной границе:

- `P_rиск` — вектор рисков для каждого портфеля (среднеквадратические отклонения доходности от ожидаемой);
- `P_доходности` — вектор ожидаемых доходностей для каждого портфеля;
- `P_доли` — массив, каждая строка которого представляет доли активов в портфеле.

Если не использовать дополнительных ограничений, вызов любой функции одинаков, поэтому рассмотрим функцию `portopt` для нахождения точек эффективной границы для модельной задачи. В нашем случае  $y_{\text{min}} = 10.8\%$  и  $y_{\text{max}} = 19.1\%$ . Попытаемся рассчитать четыре портфеля с доходностями от 12% с шагом в 2%:

```
>> P_Ret = [0.12 0.14 0.16 0.18];
>> [P_Risk, P_Ret, P_Ass] = portopt(A_yld, A_cov, [], P_Ret)
??? Error using ==> portopt
```

One or more requested returns are less than the return 0.131667 of the least risky portfolio

Полученное сообщение об ошибке информирует, что эффективного портфеля с доходностью, меньшей 0.131667, нет. Указанное значение есть  $y_{\text{low}}$  в наших обозначениях. Надо либо изменить вектор заданных доходностей портфелей, либо использовать другую форму вызова функции. Воспользуемся другим способом расчета четырех портфелей на эффективной границе:

```
>> [P_Risk, P_Ret, P_Ass] = portopt(A_yld, A_cov, 4)
P_Risk =
 0.0805
```

```

0.0884
0.1126
0.2050
P_Ret =
0.1317
0.1514
0.1712
0.1910
P_Ass =
0.5747 -0.0000 0.0487 0.2336 0.0000 0.1430
0.2592 -0.0000 0.3822 0.0020 0 0.3566
0.0000 -0.0000 0.4088 -0.0000 0.1411 0.4501
-0.0000 0 0 0 0 1.0000

```

Обратите внимание, что для нашего примера второй актив ( $E_1$ ) не вошел ни в один портфель. Последний портфель состоит только из одного актива с наибольшей доходностью, что вполне предсказуемо.

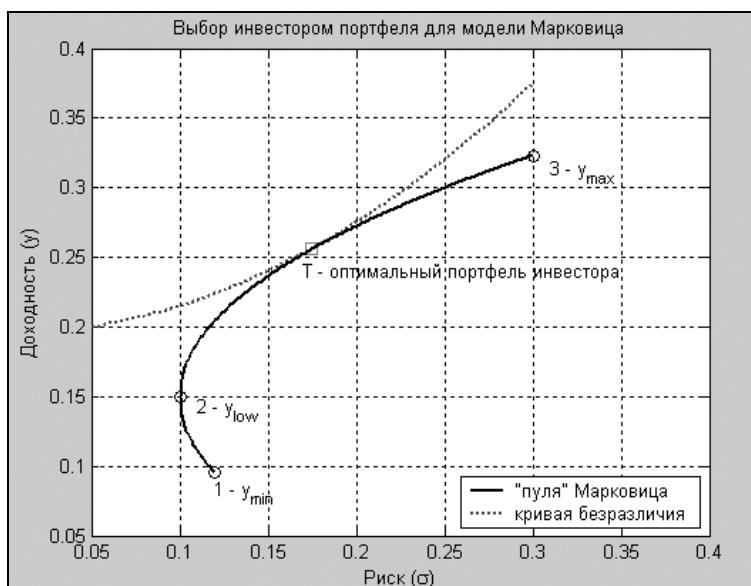
## Оптимальный выбор портфеля

Построение эффективной границы дает инвестору множество эффективных портфелей. Инвестор выбирает тот из них, который соответствует его отношению к риску: портфель с большей доходностью имеет больший риск. Моделирование отношения инвестора к риску в пакете MATLAB осуществляется с помощью функции полезности Неймана—Монгерштерна:

$$U(y_p, \sigma_p) = y_p - \frac{1}{2}a\sigma_p^2. \quad (20.14)$$

Постоянная  $a$  называется *индексом неприятия риска*. Уравнение  $U = \text{const}$  описывает линию безразличия, т. е. полезность для инвестора одинакова на всех точках этой кривой. Множество этих линий образуют непересекающееся семейство, при этом, чем больше полезность, тем выше (по отношению к доходности  $y_p$ ) и левее (по отношению к риску  $\sigma_p$ ) кривая расположена на графике. Инвестор стремится максимизировать значение функции (20.14) на множестве допустимых портфелей. Поэтому оптимальным для инвестора является портфель, отвечающий единственной точке эффективной границы, в которой она касается одной из линий безразличия (на рис. 20.1 эта точка обозначена  $T$ ). Эта линия безразличия, которую будем называть *рациональной*, отвечает наибольшей достижимой полезности, ибо малейшее ее увеличение приведет к отсутствию общих точек на допустимом множестве

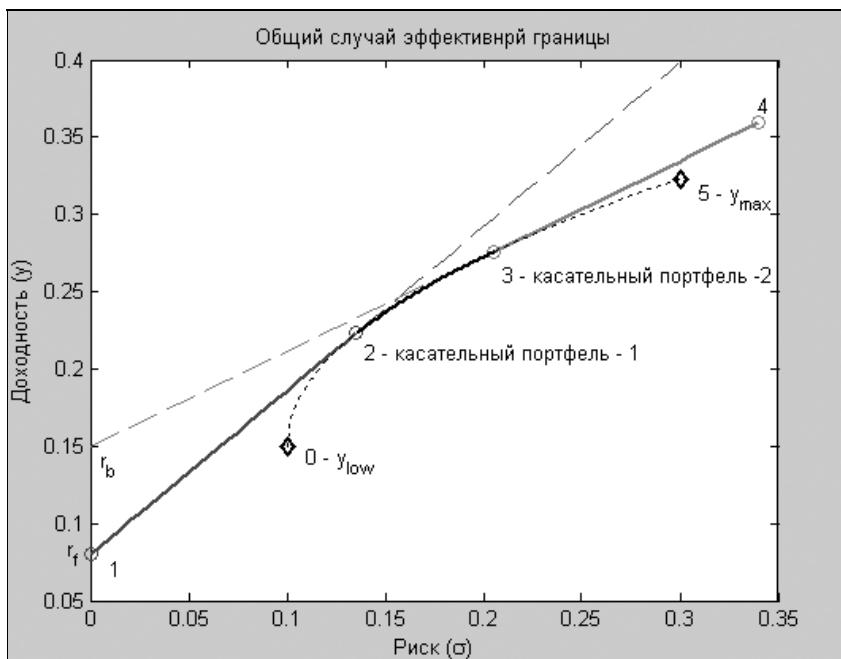
портфелей и выбранном уровне полезности. Точка 3 эффективной границы является угловой, поэтому условие касания линий (совпадение тангенсов углов наклона касательных прямых к кривой безразличия и эффективной границе) может не выполняться.



**Рис. 20.1.** Выбор оптимального портфеля на эффективной границе Марковица

Однако в пакете используется модифицированная модель инвестирования, когда инвестор может вкладывать средства не только в рисковые активы, но и в безрисковый актив, который является единственным для выбранного срока (горизонта) инвестирования, характеризуемым процентной ставкой  $r_f$ . Дополнительно инвестор может привлекать заемные средства по ставке  $r_b$  ( $r_b \geq r_f$ ), поэтому эффективная граница может состоять из трех частей (см. рис. 20.2, сплошная жирная линия):

1. Отрезок прямой 1 – 2, начинающийся в точке  $(0, r_f)$  и касающийся в точке 2 границы Марковица.
2. Луч 3 – 4 (4 — не конечная точка) есть часть касательной прямой к границе Марковица, исходящей из точки  $(0, r_b)$ .
3. Часть границы Марковица между точками 2 и 3.



**Рис. 20.2.** Эффективная граница при наличии безрисковых вложений и заимствований

В зависимости от того, какого участка эффективной границы будет касатьсяся кривая безразличия, инвестор сформирует свой портфель ценных бумаг. Точка касания будет единственной, поскольку линия безразличия является выпуклой функцией.

Если рациональная кривая безразличия соприкасается с границей  $2 - 3$ , то инвестору следует вложить все собственные средства в рисковый портфель, соответствующий точке соприкосновения, как в модели Марковица.

Если рациональная кривая безразличия соприкасается с какой-либо частью допустимой прямой (отрезком  $1 - 2$  или лучом  $3 - 4$ ), то получается коэффициент  $\xi$  вложения средств в касательный портфель, соответствующий точке касания прямой с границей Марковица:

$$\xi = \frac{\sigma_t}{\sigma_p^{(T)}}, \quad (20.15)$$

где  $\sigma_t$  — риск, отвечающий найденной точке на эффективной границе, а  $\sigma_p^{(T)}$  — риск касательного портфеля либо для точки  $2$ , либо для точки  $3$ .

Для луча 3 – 4 этот коэффициент больше 1, следовательно, инвестору следует занять недостающие средства по ставке  $r_b$  для увеличения объема второго касательного портфеля, соответствующего точке 3. Это можно интерпретировать как вложения с отрицательным весом  $(1-\xi)$  в безрисковый актив, т. е. заимствование.

Для отрезка 1 – 2 коэффициент  $\xi$  (20.15) меньше 1, поэтому инвестор должен разделить свой капитал и вложить часть  $(1-\xi)$  в безрисковый актив по ставке  $r_f$ , а часть  $\xi$  в рискованный портфель с долевым составом, отвечающим точке 2 (первый касательный портфель).

В последнем случае окончательный портфель будет содержать не только рисковые финансовые инструменты, но также и безрисковый актив, в то время как в первых двух — только рисковые ценные бумаги.

### Примечание

Возможны и другие варианты эффективной границы. Если  $r_f = r_h$  (модель Тобина), то эффективной границей является прямая (точки 2 и 3 совпадают). Если отсутствует возможность заимствования, то не будет луча 3 – 4, и эффективная граница состоит только из двух участков: отрезка 1 – 2 и части границы Марковица от точки 2 до конечной точки 5. Возможно, что отсутствует безрисковое вложение. Тогда можно считать  $r_f = 0$ , т. е. средства некуда вкладывать без риска, и точка на первом участке определяет часть средств для вложения в рисковый портфель, а остальную часть, соответствующую нулевой ставке "безрисковых вложений", следует вывести с рынка рисковых ценных бумаг.

Для расчета рационального портфеля предназначена функция `portalloc`:

```
[КП_риск, КП_доходность, КП_доли, коэффициент, ОП_риск, ...
ОП_доходность] = portalloc(П_риск, П_доходности, П_доли, ...
Бр_вложение, Бр_заимствование, Инд_риска)
```

Первые три входных параметра — это массивы характеристик портфелей на эффективной границе Марковица, рассчитанные с помощью ранее рассмотренных функций `frontcon` и `portopt`. Два следующих параметра — это ставка вложения в безрисковый актив `Бр_вложение` и процентная ставка безрискового заимствования `Бр_заимствование`. Последний параметр `Инд_риска` есть индекс неприятия риска инвестором. Рекомендуемые значения: 4 — для осторожного инвестора и 2 — для готового к риску. Чем больше это значение, тем меньший риск выбирается.

Выходные параметры дают характеристики касательного портфеля (`КП_риск`, `КП_доходность`, `КП_доли`), коэффициент  $\xi$  вложения средств в касательный портфель (коэффициент), риск и доходность оптимального портфеля (`ОП_риск`, `ОП_доходность`).

Если эта функция вызывается без выходных параметров, то она выводит графическое окно, в котором иллюстрируется положение оптимального портфеля на эффективной границе.

Фактически возможны три варианта, которые обсуждались ранее. Для того чтобы проанализировать возможные случаи работы функции `portalloc`, выполните файл-программу из листинга 20.7. Вначале она содержит команды программы из листинга 20.6 для формирования исходных данных рисковых активов. Далее с помощью функции находится эффективная граница Марковица и задаются общие значения для безрисковых ставок вложения и заимствования. Затем для трех разных индексов неприятия риска ищется оптимальный портфель. Чтобы получить и графическое, и численное представление результатов, функция вызывается дважды: без выходных параметров для построения графика, а потом выводятся численные значения характеристик оптимального портфеля.

### Листинг 20.7. Получение оптимального портфеля в зависимости от отношения инвестора к риску

```
%Ожидаемые доходности активов
A_yld = [0.108 0.136 0.144 0.151 0.187 0.191];
%Среднеквадратические отклонения активов
A_dev = [0.12 0.135 0.15 0.18 0.20 0.205];
%Матрица коэффициентов корреляции
A_corr = [1 0.32 0.4 -0.3 0.31 -0.18;
 0.32 1 0.45 0.43 0.29 0.3;
 0.4 0.45 1 0.4 0.46 -0.29;
 -0.3 0.43 0.4 1 0.41 0.4;
 0.31 0.29 0.46 0.41 1 0.25;
 -0.18 0.3 -0.29 0.4 0.25 1];
%вычисление матрицы ковариаций активов
A_cov = corr2cov(A_dev, A_corr);

%нахождение эффективной границы
[P_Risk, P_Ret, P_Ass] = portopt(A_yld, A_cov, 25)
%Процентные ставки безрисковых активов
R_Freeasset = 0.095;
R_Borrow = 0.13;
```

```
% Оптимальный портфель инвестора
% с высоким уровнем неприятия риска
Index_Risk = 7;
[TP_Risk1, TP_Ret1, TP_Ass1, R_Fraction1, ...
OP_Risk1, OP_Ret1] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow , Index_Risk)
port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk);

% Оптимальный портфель инвестора
% с низким уровнем неприятия риска
Index_Risk = 2;
[TP_Risk2, TP_Ret2, TP_Ass2, R_Fraction2, ...
OP_Risk2, OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow , Index_Risk)
port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk);

% Оптимальный портфель инвестора
% со среднем уровнем неприятия риска
Index_Risk = 3;
[TP_Risk3, TP_Ret3, TP_Ass3, R_Fraction3, ...
OP_Risk3, OP_Ret3] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk)
port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk);
```

В результате выполнения программы из листинга 20.7 получатся оптимальные портфели для инвесторов с разным уровнем неприятия риска. Для осторожных, не склонных к риску, инвесторов выбор оптимального портфеля представлен на рис. 20.3.

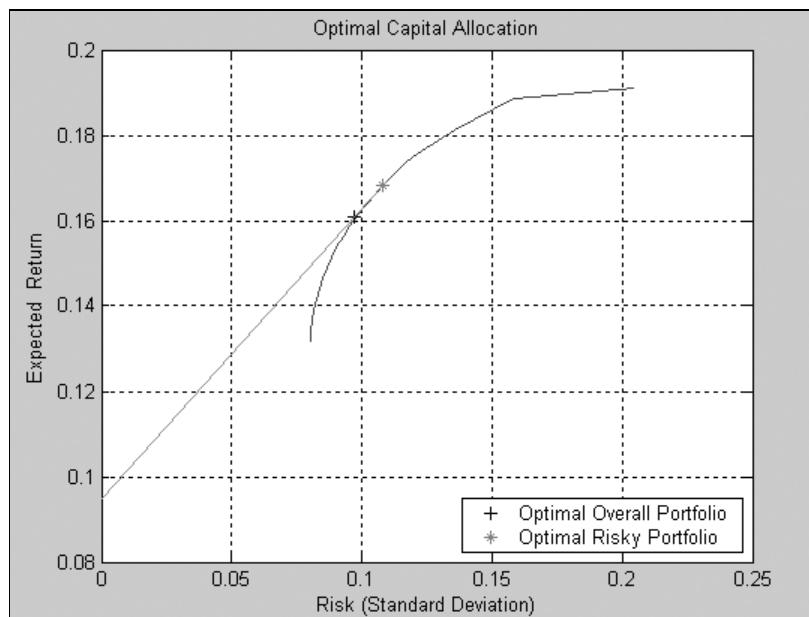
Оптимальный портфель лежит на отрезке, соединяющем безрисковый актив для вложения и первый касательный портфель. Численные результаты для этого случая таковы:

```
TP_Risk1 =
0.1078
TP_Ret1 =
0.1683
```

```

TP_Ass1 =
 0.0072 -0.0000 0.4636 -0.0000 0.0905 0.4387
R_Fraction1 =
 0.9003
OP_Risk1 =
 0.0971
OP_Ret1 =
 0.1609

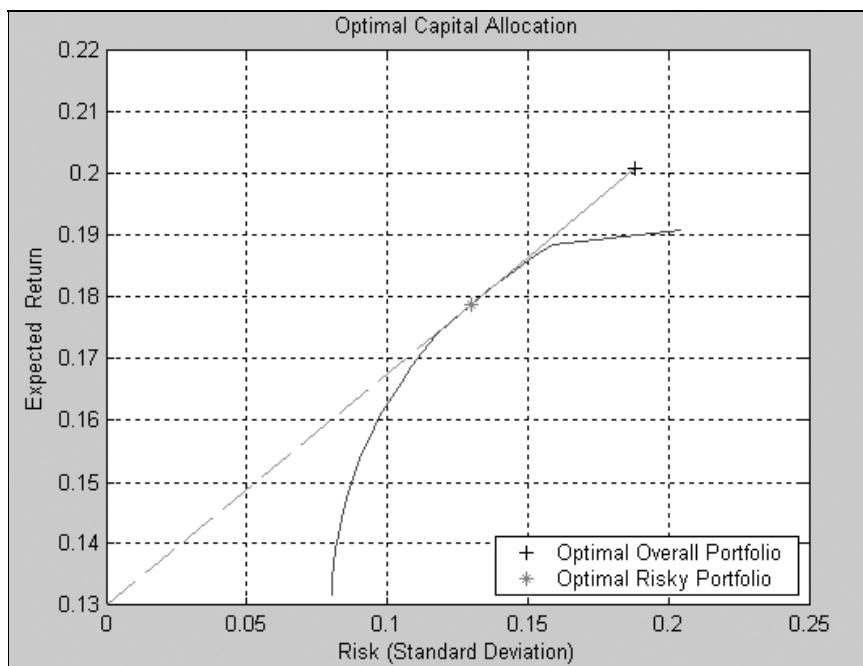
```



**Рис. 20.3.** Оптимальный портфель инвестора с высоким уровнем неприятия риска

Наиболее показательной величиной является параметр `R_Fraction1` (формула (20.15)), показывающий долю касательного портфеля в оптимальном. В данном случае он меньше единицы. Это показывает, что оптимальный портфель лежит на участке 1 – 2 эффективной границы в соответствии с обозначениями на рис. 20.2. Он также указывает инвестору, что 90% собственных средств надо вложить в касательный портфель, доли каждого актива представлены параметром `TP_Ass1`. Оставшиеся 10% следует вложить в безрисковый актив. Если величина `Index_Risk` будет больше, то и процент вложений в безрисковый актив будет больше. Параметры `OP_Ret1` и `OP_Risk1` дают расчетные значения для ожидаемой доходности и риска оптимального портфеля.

Оптимальный портфель инвестора, готового рискнуть для получения большей доходности, представлен на рис 20.4.



**Рис. 20.4.** Оптимальный портфель инвестора с низким уровнем неприятия риска

Оптимальный портфель лежит на линии 3 – 4 (в соответствии с обозначениями рис. 20.2), а рассчитанные величины таковы:

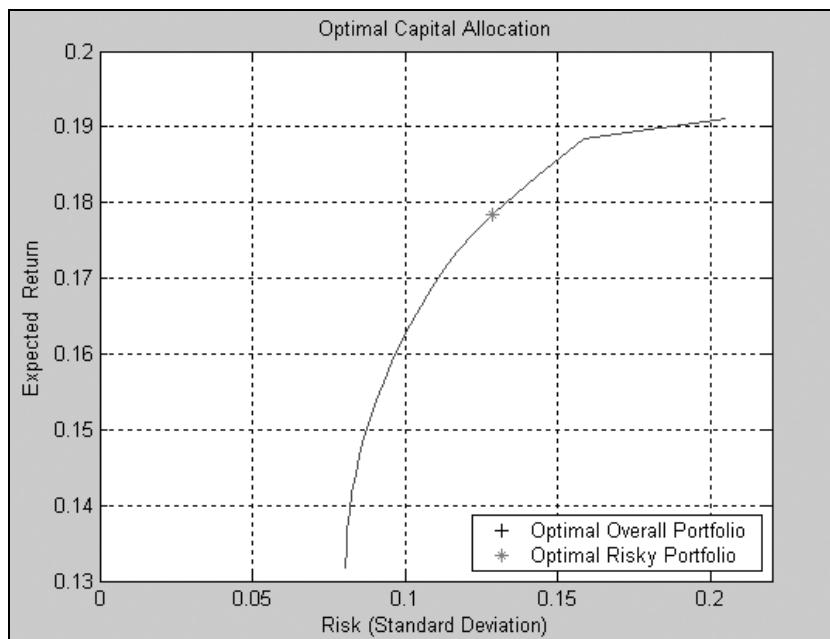
```

TP_Risk2 =
 0.1298
TP_Ret2 =
 0.1789
TP_Ass2 =
 0.0000 -0.0000 0.2340 -0.0000 0.2870 0.4790
R_Fraction2 =
 1.4503
OP_Risk2 =
 0.1882
OP_Ret2 =
 0.2009

```

Параметр `R_Fraction2` показывает, что инвестору следует увеличить собственные инвестиции на 45% за счет заемных средств. Эта величина будет тем большей, чем меньше индекс неприятия риска `Index_Risk`. Все средства вкладываются в касательный портфель с долями активов, которые определяются параметром `TP_Ass2`.

Инвестор со средним уровнем неприятия риска на нашем модельном рынке ценных бумаг выберет портфель на границе Марковица (на ее части 2 – 3 в обозначениях рис. 20.2), и функция отобразит это в виде, представленном на рис. 20.5.



**Рис. 20.5.** Оптимальный портфель инвестора со средним уровнем неприятия риска

Численные результаты для этого случая будут следующими:

```
TP_Risk3 =
0.1284
TP_Ret3 =
0.1783
TP_Ass3 =
0.0000 -0.0000 0.2461 -0.0000 0.2769 0.4770
```

```
R_Fraction3 =
1
OP_Risk3 =
0.1284
OP_Ret3 =
0.1783
```

Поскольку параметр `R_Fraction3` равен 1, то характеристики касательного портфеля и оптимального одинаковые.

### Примечание

Важно осознать, что отношение инвестора к риску определяется параметром коэффициент в функции `portalloc`, а не тем, на каком участке эффективной границы расположен оптимальный портфель. Рыночная конъюнктура может быть такой, что для всех инвесторов оптимальные портфели будут лежать на одном из трех основных участков эффективной границы и отличаться будут только долями. В теории считается, что осторожный инвестор имеет индекс неприятия риска в окрестности 4 и более, инвестор, готовый рисковать, вблизи 2 и менее. Инвестор со средним уровнем неприятия риска имеет этот параметр в окрестности 3.

## Дополнительные ограничения при анализе портфелей

При расчете эффективной границы мы обратили внимание, что второй актив не вошел ни в один портфель. Однако если инвестор выбрал для вложения этот рыночный инструмент, то он желает включить его в свой портфель, руководствуясь состоянием рынка. Функции пакета MATLAB дают пользователю возможность сформировать дополнительные ограничения на весовые коэффициенты для активов в портфеле. Любые дополнительные ограничения сужают множество допустимых портфелей и, следовательно, приводят к получению менее эффективного портфеля. Эти ограничения по-разному передаются в рассмотренные ранее функции `frontcon` и `portopt`. Для функции `frontcon` они передаются с использованием параметров `A_ограничения`, `G_активов`, `G_ограничения`. Простейшие двусторонние ограничения на активы формулируются для каждой ценой бумаги независимо и имеют вид:

$$L_i \leq x_i \leq U_i, \quad 0 \leq L_i \leq x_i \leq U_i \leq 1. \quad (20.16)$$

Для передачи этих ограничений функции `frontcon` в качестве параметра следует сформировать массив из двух строк, содержащих индивидуальные ограничения для каждой ценной бумаги. Пусть в рассматриваемом примере

инвестор предполагает в каждый актив вложить не менее 5% и не более 37% собственного капитала. Тогда ограничения на активы запишутся в массив как:

$$AC = \begin{bmatrix} 0.05 & 0.05 & 0.05 & 0.05 & 0.05 & 0.05 \\ 0.37 & 0.37 & 0.37 & 0.37 & 0.37 & 0.37 \end{bmatrix}. \quad (20.17)$$

Далее, в листинге 20.8, иллюстрируется использование этого массива.

Другой тип ограничений, который будем называть ограничениями на группу активов, задается для функции `frontcon` двумя параметрами  $\Gamma_{\text{активов}}$  и  $\Gamma_{\text{ограничения}}$ .

Первый из них определяет группы активов. Он представляет собой матрицу, состоящую из такого количества строк, сколько групп выделил в своем портфеле инвестор. Каждая строка состоит из нулей и единиц и содержит количество элементов, равное числу активов в портфеле. Значение 1 указывает, что ценная бумага входит в группу, а 0 — нет. Инвестор самостоятельно руководствуется признаками для включения финансовых инструментов в группу. Второй параметр  $\Gamma_{\text{ограничения}}$  является массивом, состоящим из двух столбцов размерности количества введенных групп для нижней и верхней границы суммы весов активов, входящих в группу. Если матрицу, задающую состав групп, обозначить  $G$ , нижние и верхние ограничения для каждой группы записать в векторы  $g_l$  и  $g_u$ , то математически эти условия запишутся в форме:

$$g_l \leq Gx \leq g_u. \quad (20.18)$$

Определим для ранее рассмотренного примера группы, сформированные по двум признакам:

- сектор экономики, к которому принадлежит эмитент;
- ориентация на внутренний или внешний рынки.

Будем считать, что ценные бумаги M1, M2 и M3 выпущены в обращение машиностроительными корпорациями, E1 и E2 — энергетическими, C1 — предоставляющими услуги связи. Эмитенты акций E2 и M3 ориентированы на внешние рынки. Пусть инвестор имеет намерения вкладывать средства в активы, руководствуясь следующими соображениями:

- для энергетических компаний вложения должны быть не менее 22% и не более 64%;
- для машиностроительных компаний вложения должны быть не менее 26% и не более 71%;
- для компаний, ориентированных на внутренний рынок, вложения должны быть не менее 14% и не более 56%;

- для компаний, ориентированных на внешний рынок, вложения должны быть не менее 25% и не более 84%.

Тогда для параметров  $\Gamma_{\text{активов}}$  и  $\Gamma_{\text{ограничения}}$  надо сформировать массивы, которые соответственно обозначим  $GA$  и  $BG$ , в виде:

$$GA = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad (20.19)$$

и

$$BG = \begin{bmatrix} 0.12 & 0.46 \\ 0.1 & 0.52 \\ 0.24 & 0.63 \\ 0.25 & 0.53 \end{bmatrix}. \quad (20.20)$$

Для определенности будем относить себя к инвестору с индексом неприятия риска 2. Рассчитайте эффективную границу при наличии дополнительных условий на вес активов в портфеле, воспользовавшись листингом 20.8. Вначале рассчитывается состав портфеля только с ограничениями вида (20.16), потом только с ограничениями типа (20.18) и окончательно со всеми условиями.

#### Листинг 20.8. Использование ограничений при поиске оптимального портфеля функцией `frontcon`

```
%Ожидаемые доходности активов
A_yld = [0.108 0.136 0.144 0.151 0.187 0.191];
%Среднеквадратические отклонения активов
A_dev = [0.12 0.135 0.15 0.18 0.20 0.205];
%Матрица коэффициентов корреляции
A_corr = [1 0.32 0.4 -0.3 0.31 -0.18;
 0.32 1 0.45 0.43 0.29 0.3;
 0.4 0.45 1 0.4 0.46 -0.29;
 -0.3 0.43 0.4 1 0.41 0.4;
 0.31 0.29 0.46 0.41 1 0.25;
 -0.18 0.3 -0.29 0.4 0.25 1];
%вычисление матрицы ковариаций активов
A_cov = corr2cov(A_dev, A_corr);
```

%Матрица для ограничений на активы

```
AC = [0.05 0.05 0.05 0.05 0.05 0.05;
 0.37 0.37 0.37 0.37 0.37 0.37];
```

%Данные для ограничений на группы активов

```
GA = [0 1 1 0 0 0;
 0 1 1 0 0 0;
 0 0 1 0 0 1;
 1 1 0 1 1 0];
```

```
BG = [0.12 0.46;
```

```
 0.1 0.52;
```

```
 0.24 0.63;
```

```
 0.25 0.53]; %0.25 0.52 при таких значениях - ошибка
```

%Процентные ставки безрисковых активов

```
R_Freeasset = 0.095;
```

```
R_Borrow = 0.13;
```

% Инвестор с низким уровнем неприятия риска

```
Index_Risk = 2;
```

%нахождение эффективной границы

```
[P_Risk, P_Ret, P_Ass] = frontcon(A_yld, A_cov, 5)
```

%Процентные ставки безрисковых активов

```
R_Freeasset = 0.095;
```

```
R_Borrow = 0.13;
```

%Инвестор с низким уровнем неприятия риска

```
Index_Risk = 2;
```

%нахождение эффективной границы с ограничениями на активы

```
[P_Risk, P_Ret, P_Ass] = frontcon(A_yld, A_cov, 25, [], AC);
```

%Оптимальный портфель инвестора с ограничением на активы

```
disp('Оптимальный портфель инвестора')
```

```
disp('с ограничениями на активы')
```

```
[CondA_TP_Risk2, CondA_TP_Ret2, CondA_TP_Ass2, CondA_R_Fraction2, ...
```

```
CondA_OP_Risk2, CondA_OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
```

```
R_Freeasset, R_Borrow, Index_Risk)
```

%нахождение эффективной границы с ограничениями на группы активов

```
[P_Risk, P_Ret, P_Ass] = frontcon(A_yld, A_cov, 25, [], [], GA, BG);
```

```
%Оптимальный портфель инвестора с ограничениями на группы активов
disp('Оптимальный портфель инвестора')
disp(' с ограничениями на группы активов ')
[CondG_TP_Risk2, CondG_TP_Ret2, CondG_TP_Ass2, CondG_R_Fraction2, ...
CondG_OP_Risk2, CondG_OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk)

%нахождение эффективной границы со всеми ограничениями
[CondA_TP_Risk2, CondA_TP_Ret2, CondA_TP_Ass2, R_Fraction2, ...
P_Risk, P_Ret, P_Ass] = frontcon(A_yld, A_cov, 25, [], AC, GA, BG);
%Оптимальный портфель инвестора со всеми ограничениями
disp('Оптимальный портфель инвестора')
disp('со всеми ограничениями')
[CondF_TP_Risk2, CondF_TP_Ret2, CondF_TP_Ass2, CondF_R_Fraction2, ...
CondF_OP_Risk2, CondF_OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk)
```

После выполнения программы из листинга 20.8 получите следующие результаты:

Оптимальный портфель инвестора

с ограничениями на активы

CondA\_TP\_Risk2 =

0.1266

CondA\_TP\_Ret2 =

0.1741

CondA\_TP\_Ass2 =

|        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|
| 0.0500 | 0.0500 | 0.1408 | 0.0500 | 0.3392 | 0.3700 |
|--------|--------|--------|--------|--------|--------|

CondA\_R\_Fraction2 =

1.3767

CondA\_OP\_Risk2 =

0.1743

CondA\_OP\_Ret2 =

0.1907

Оптимальный портфель инвестора

с ограничениями на группы активов

CondG\_TP\_Risk2 =

0.1380

```
CondG_TP_Ret2 =
 0.1818
CondG_TP_Ass2 =
 0.0000 0 0.1646 -0.0000 0.3700 0.4654
CondG_R_Fraction2 =
 1.3604
CondG_OP_Risk2 =
 0.1877
CondG_OP_Ret2 =
 0.2004
Оптимальный портфель инвестора
со всеми ограничениями
CondF_TP_Risk2 =
 0.1272
CondF_TP_Ret2 =
 0.1743
CondF_TP_Ass2 =
 0.0500 0.0500 0.1362 0.0508 0.3430 0.3700
CondF_R_Fraction2 =
 1.3686
CondF_OP_Risk2 =
 0.1741
CondF_OP_Ret2 =
 0.1906
```

По результатам вычислений легко увидеть, какие ограничения внесли изменения на формирование эффективной границы. Для этого надо проанализировать состав касательного портфеля. Как правило, инвестор проводит многовариантные расчеты для подбора удовлетворяющего его портфеля.

### Примечание

Поскольку многочисленные ограничения могут привести к отсутствию допустимых портфелей, рекомендуется поочередно вводить новые ограничения. В тестовом примере дополнительные условия выбраны так, чтобы избежать такой ситуации.

Для функции `portopt` формирование ограничений происходит иначе. Для нее вся совокупность ограничений передается параметром `Множест-`

во\_ограничений. Для того чтобы этот параметр определить, надо воспользоваться другой функцией пакета `portcons`, имеющей переменное число однотипных аргументов и вызываемой следующим образом для формирования одного типа ограничений:

Множество\_ограничений =

```
portcons('тип_ограничения', данные_1, ..., данные_N)
```

В табл. 20.7 представлены типы ограничений и требуемые для их формирования данные.

**Таблица 20.7. Дополнительные ограничения на активы**

| Тип ограничения | Назначение                                                         | Число элементов данных | Описание данных                                                                                                                                                                                                                                                                                                |
|-----------------|--------------------------------------------------------------------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Default         | Стандартные ограничения на веса:<br>$x_i \geq 0$<br>$\sum x_i = 1$ | 1                      | 1. число_активов (обязательный). Количество активов в портфеле — скаляр                                                                                                                                                                                                                                        |
| PortValue       | Нормировка портфеля                                                | 2                      | 1. коэффициент (обязательный), задает сумму весов активов в портфеле — скаляр.<br>2. число_активов (обязательный). Количество активов в портфеле — скаляр                                                                                                                                                      |
| AssetLims       | Ограничения на активы                                              | 3                      | 1. нижняя_граница (обязательный), вектор из $L_i$ в обозначениях формулы (20.16). Размерность — число_активов.<br>2. верхняя_граница (обязательный), вектор из $U_i$ в обозначениях формулы (20.16). Размерность — число_активов.<br>3. число_активов (необязательный), количество активов в портфеле — скаляр |

Таблица 20.7 (продолжение)

| Тип ограничения | Назначение                                                              | Число элементов данных | Описание данных                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------|-------------------------------------------------------------------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GroupLims       | Ограничения на группы активов.                                          | 4                      | <p>1. Г_активов (обязательный), матрица типа <math>G</math> в обозначениях формулы (20.18). Размерность — число_групп на число_активов.</p> <p>2. нижняя_граница (обязательный), вектор <math>g_l</math> в обозначениях формулы (20.18). Размерность — число_групп.</p> <p>3. верхняя_граница (обязательный), вектор <math>g_u</math> в обозначениях формулы (20.18). Размерность — число_групп.</p> <p>4. число_групп (необязательный), количество определенных групп — скаляр</p>                               |
| GroupComparison | Ограничения на сумму весов в двух множествах групп (сравнение попарное) | 5                      | <p>1. Множество_1_групп (обязательный), матрица типа <math>G</math> в обозначениях формулы (20.18). Размерность — число_групп на число_активов.</p> <p>2. Множество_2_групп (обязательный), матрица типа <math>G</math> в обозначениях формулы (20.18). Размерность — число_групп на число_активов.</p> <p>3. минимальное_отношение (обязательный), скаляр или вектор, определяющий минимальное отношение суммы весов в Множество_1_групп к сумме весов в Множество_2_групп. Размерность — 1 или число_групп.</p> |

Таблица 20.7 (окончание)

| Тип ограничения | Назначение                                                                                | Число элементов данных | Описание данных                                                                                                                                                                                                                                                                                                                                  |
|-----------------|-------------------------------------------------------------------------------------------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 |                                                                                           |                        | <p>4. максимальное_отношение (обязательный), скаляр или вектор, определяющий максимальное отношение суммы весов в Множество_1_групп к сумме весов в Множество_2_групп. Размерность — 1 или число_групп.</p> <p>5. Число_групп (необязательный), количество определенных групп в каждом множестве (одинаковое для каждого множества) — скаляр</p> |
| Custom          | <p>Произвольные линейные ограничения.</p> <p>Неравенства:<br/> <math>Ax \leq b</math></p> | 2                      | <p>1. матрица_A (обязательный), матрица коэффициентов матричного неравенства <math>Ax \leq b</math>. Размерность — число_ограничений на число_активов.</p> <p>2. вектор_b (обязательный), вектор правых частей матричного неравенства <math>Ax \leq b</math>. Размерность — число_ограничений</p>                                                |

Выходной параметр Множество\_ограничений представляет прямоугольную матрицу, число столбцов которой на 1 больше числа активов в портфеле. При построении выходной матрицы ограничений условия их совместимости не проверяются. Поясним это на следующем простейшем примере:

```
>> Restr_Set = portcons ('PortValue', 1.25, 3, 'Default', 3)
```

```
Restr_Set =
```

|         |         |         |         |
|---------|---------|---------|---------|
| 1.0000  | 1.0000  | 1.0000  | 1.2500  |
| -1.0000 | -1.0000 | -1.0000 | -1.2500 |
| 1.0000  | 1.0000  | 1.0000  | 1.0000  |
| -1.0000 | -1.0000 | -1.0000 | -1.0000 |
| -1.0000 | 0       | 0       | 0       |
| 0       | -1.0000 | 0       | 0       |
| 0       | 0       | -1.0000 | 0       |

Первые две строки являются условием нормировки суммы весов портфеля на величину 1.25. Следующие строки формируют неравенства, соответствующие ограничениям по умолчанию: нормировка суммы весов портфеля на 1 и  $x_i \geq 0$  ( $i = 1, 2, 3$ ). Только при дальнейшем использовании ограничений выяснится их несовместность. Для использования функции детальное знание структуры выходной матрицы не требуется.

Типы ограничений `AssetLims` и `GroupLims` соответствуют ранее рассмотренным ограничениям для функции `frontcon` и пояснений не требуют. Фактически и формирование ограничения типа `Custom` тоже прозрачно.

Поясним, как формируется ограничение типа `GroupComparison`. Для его создания надо определить два множества, состоящие из одинакового количества групп. Принцип их задания такой же, как и для ранее рассмотренного параметра `Г_активов` функции `frontcon`. Пусть для определенности минимальное\_отношение и максимальное\_отношение представляют собой векторы. Обозначим  $G_i^{(1)} = \{G_{ij}^{(1)}\}$  и  $G_i^{(2)} = \{G_{ij}^{(2)}\}$  ( $j = 1, \dots, n$  — номер актива) строки матриц `Множество_1_групп` и `Множество_2_групп`, определяющие сравниваемые группы (сравниваются группы с одинаковым порядковым номером во множествах). Пусть далее компоненты векторов `минимальное_отношение` и `максимальное_отношение` обозначены  $l_i$  и  $u_i$ , а вектора `число_активов` —  $n$ . Тогда составляется двойное неравенство:

$$l_i \leq \frac{\sum_{j=1}^n G_{ij}^{(1)} x_j}{\sum_{j=1}^n G_{ij}^{(2)} x_j} \leq u_i$$

или два ограничения-неравенства:

$$-\sum_{j=1}^n G_{ij}^{(1)} x_j + l_i \sum_{j=1}^n G_{ij}^{(2)} x_j \leq 0, \quad \sum_{j=1}^n G_{ij}^{(1)} x_j - u_i \sum_{j=1}^n G_{ij}^{(2)} x_j \leq 0.$$

В заключение в листинге 20.9 представим программу расчета тех же портфелей, что и в листинге 20.8, только с применением функции `portopt`. В качестве результатов выведем только доходность и риск оптимального портфеля, а также коэффициент вложения в касательный портфель.

#### Листинг 20.9. Использование ограничений при поиске оптимального портфеля функцией `portopt`

%Ожидаемые доходности активов

```
A_yld = [0.108 0.136 0.144 0.151 0.187 0.191];
```

```

%Среднеквадратические отклонения активов
A_dev = [0.12 0.135 0.15 0.18 0.20 0.205];
%Матрица коэффициентов корреляции
A_corr = [1 0.32 0.4 -0.3 0.31 -0.18;
 0.32 1 0.45 0.43 0.29 0.3;
 0.4 0.45 1 0.4 0.46 -0.29;
 -0.3 0.43 0.4 1 0.41 0.4;
 0.31 0.29 0.46 0.41 1 0.25;
 -0.18 0.3 0.29 0.4 0.25 1];
%вычисление матрицы ковариаций активов
A_cov = corr2cov(A_dev, A_corr);
%Процентные ставки безрисковых активов
R_Freeasset = 0.095;
R_Borrow = 0.13;
% Инвестор с низким уровнем неприятия риска
Index_Risk = 2;
%Процентные ставки безрисковых активов
R_Freeasset = 0.095;
R_Borrow = 0.13;
% Инвестор с низким уровнем неприятия риска
Index_Risk = 2;
%нахождение эффективной границы с ограничениями на активы
%Матрица для ограничений на активы
ACmax = [0.37 0.37 0.37 0.37 0.37 0.37];
ACmin = [0.05 0.05 0.05 0.05 0.05 0.05];
Restr_Set = portcons('Default', 6, 'AssetLims', ACmin, ACmax, 6);
[P_Risk, P_Ret, P_Ass] = portopt(A_yld, A_cov, 25, [], Restr_Set);
%Оптимальный портфель инвестора с
disp('Оптимальный портфель инвестора')
disp(' с ограничениями на активы ')
[CondA_TP_Risk2, CondA_TP_Ret2, CondA_TP_Ass2, CondA_R_Fraction2, ...
CondA_OP_Risk2, CondA_OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk);
CondA_R_Fraction2
CondA_OP_Risk2
CondA_OP_Ret2
%нахождение эффективной границы с ограничениями на группы активов
%Данные для ограничений на группы активов

```

```

GA = [0 1 1 0 0 0;
 0 1 1 0 0 0;
 0 0 1 0 0 1;
 1 1 0 1 1 0];
GBmax = [0.46 ; 0.52 ; 0.63 ; 0.53];
GBmin = [0.12 ; 0.1 ; 0.24 ; 0.25];
Restr_Set = portcons('Default', 6, 'GroupLims', GA, GBmin, GBmax);
[P_Risk, P_Ret, P_Ass] = portopt(A_yld, A_cov, 25, [], Restr_Set);
%Оптимальный портфель инвестора с ограничениями на группы активов
disp('Оптимальный портфель инвестора')
disp('с ограничениями на группы активов ')
[CondG_TP_Risk2, CondG_TP_Ret2, CondG_TP_Ass2, CondG_R_Fraction2, ...
CondG_OP_Risk2, CondG_OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk);
CondG_R_Fraction2
CondG_OP_Risk2
CondG_OP_Ret2
%нахождение эффективной границы со всеми ограничениями
Restr_Set = portcons('Default', 6, 'AssetLims', ACmin, ACmax, 'GroupLims', GA, GBmin, GBmax);
[P_Risk, P_Ret, P_Ass] = portopt(A_yld, A_cov, 25, [], Restr_Set);
%Оптимальный портфель инвестора со всеми ограничениями
disp('Оптимальный портфель инвестора')
disp('со всеми ограничениями ')
[CondF_TP_Risk2, CondF_TP_Ret2, CondF_TP_Ass2, CondF_R_Fraction2, ...
CondF_OP_Risk2, CondF_OP_Ret2] = port_alloc(P_Risk, P_Ret, P_Ass, ...
R_Freeasset, R_Borrow, Index_Risk);
CondF_R_Fraction2
CondF_OP_Risk2
CondF_OP_Ret2

```

После выполнения программы получим те же результаты, что и ранее (поскольку решалась та же задача).

```

>> Оптимальный портфель инвестора
 с ограничениями на активы
CondA_R_Fraction2 =
 1.3767

```

```
CondA_OP_Risk2 =
```

```
 0.1743
```

```
CondA_OP_Ret2 =
```

```
 0.1907
```

Оптимальный портфель инвестора  
с ограничениями на группы активов

```
CondG_R_Fraction2 =
```

```
 1.3604
```

```
CondG_OP_Risk2 =
```

```
 0.1877
```

```
CondG_OP_Ret2 =
```

```
 0.2004
```

Оптимальный портфель инвестора  
со всеми ограничениями

```
CondF_R_Fraction2 =
```

```
 1.3686
```

```
CondF_OP_Risk2 =
```

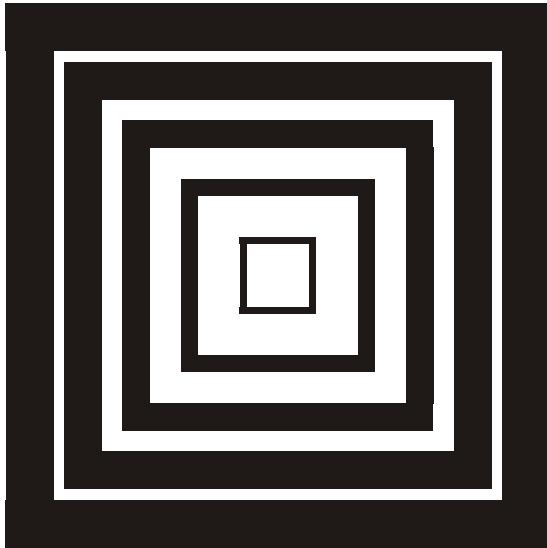
```
 0.1741
```

```
CondF_OP_Ret2 =
```

```
 0.1906
```

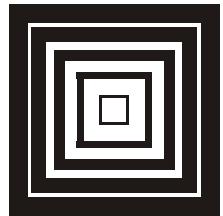
### Примечание

В заключение обратим ваше внимание на необходимость осторожно относиться к полученным численным результатам, даже если они вполне правдоподобны. Задача определения оптимального портфеля чувствительна к исходным данным и ошибкам вычислений.



## **ЧАСТЬ V**

# **Дополнительные возможности MATLAB**



## Глава 21

# Связь MATLAB и MS Office

Среда MATLAB допускает интегрирование с MS Word и MS Excel, которое позволяет достаточно просто подготовить отчет по результатам исследований в MATLAB и получить файл в одном из распространенных форматов, включая MS Word и MS Power Point. Отчеты могут являться документами MS Word, которые позволяют выполнить блоки команд MATLAB и сразу же вывести в документ текстовые или графические результаты. Сочетание этой возможности со средствами MS Word по оформлению документов образует удобную среду для разработки, например, интерактивных учебных пособий. Обработка данных облегчается при сочетании работы в MATLAB и MS Excel. Надстройка MS Excel Link, входящая в поставку MATLAB, снабжает пользователя MS Excel доступом ко всем функциям MATLAB, которые значительно расширяют возможности электронных таблиц.

## Публикация результатов работы

Для обмена информацией с другими пользователями с целью представления своей работы зачастую недостаточно предъявить соответствующий M-файл, даже снабженный подробными комментариями. Наглядная демонстрация должна включать формулы, блоки команд, графические и текстовые результаты в одном из распространенных форматов. Используя, например, MS Word, можно написать отчет по работе, сопроводив его комментариями, текстовыми данными, скопированными из командного окна, и рисунками, экспортированными из графических окон. Однако эту работу можно переложить на MATLAB и, затрачивая минимальные усилия, получить данный отчет в одном из форматов: документ MS Word, презентация PowerPoint, HTML или LaTeX.

Справочная система MATLAB содержит ссылку на видеодемонстрацию (на английском языке) процесса публикации результатов, которая длится около

5 мин. Для ее запуска перейдите к содержимому вкладки **Demos** интерактивной справочной системы MATLAB, и в разд. **MATLAB: Desktop Tools and Development Environment: Publishing M Code from the Editor** щелкните по гиперссылке **Run this demo**.

Мы не будем рассматривать все возможности для публикации результатов работы, а опишем только основные подходы. Прежде всего следует организовать расчеты в файл-программе, разбитой на ячейки. Работу в этом режиме мы рассматривали выше (см. разд. "Разбиение M-файла на ячейки" главы 5).

Предположим, что наша работа состояла в вычислении интеграла

$$\int_{-1}^1 |x|^{-\frac{1}{100}} dx.$$

Его точное значение составляет  $200/99$ , но прямое применение функции `quadl` не позволяет вычислить этот интеграл из-за особенности подынтегральной функции (численное интегрирование описано в разд. "Вычисление определенных интегралов" главы 6).

Мы разбиваем интеграл по отрезку  $[-1, 1]$  на два:  $[-1, 0]$  и  $[0, 1]$ , находим их значения при помощи `quadl` и затем складываем для получения ответа. В отчет желательно поместить формулы, графики подынтегральных функций, команды MATLAB и их результат. Для этого подготовьте M-файл `integral.m`, разбитый на ячейки в соответствии с листингом 21.1.

Заголовки ячеек станут разделами отчета. Если в ячейку требуется поместить текст, то строку с текстом следует начинать со знака %, т. е. закомментировать. Для получения в отчете жирного шрифта следует заключить текстовую строку символами звездочки, а моноширинного — символами вертикальной черты. Формулы набираются в формате LaTeX и окружаются двумя знаками доллара (примеры набора формул приведены в разд. "Оформление графика" главы 3 и "Вывод математических формул в формате LaTeX" главы 9).

### Примечание

Меню **Cell** редактора M-файлов содержит подменю **Insert Text Markup**. Его пункты служат для размещения в тексте M-файла заготовок для формул и оформления текста различными стилями.

Те операторы MATLAB, которые выводят нужную нам информацию в командное окно, мы не завершаем точкой с запятой для включения результатов в отчет.

**Листинг 21.1. М-файл для опубликования результатов работы**

```
% Calculation of the integral
% $\int_{-1}^1 |x|^{1/100} dx = \frac{200}{99}$
% First step. Introduce integrand.
fun = inline('abs(x).^(0.01)');
% Second step. Direct integration using |quadl|
I = quadl(fun, -1, 1)
%%
% *It is not very good idea!*
% Just look at integrand
fplot(fun, [-1 1])
%% Splitting
% $\int_{-1}^1 |x|^{1/100} dx = $
% $\int_{-1}^0 (-x)^{1/100} dx + \int_0^1 x^{1/100} dx$-
% First integral
% $\int_{-1}^0 (-x)^{1/100} dx$-
fplot(fun, [-1 0])
I1 = quadl(fun, -1, 0)
%% Second integral
% $\int_0^1 x^{1/100} dx$-
fplot(fun, [0 1])
I2 = quadl(fun, 0, 1)
%% Result
I = I1 + I2
Iex = 200/99
```

После создания и сохранения М-файла integral.m в редакторе М-файлов в меню **File** выберите пункт **Publish to html**. По завершении процесса создания отчета он открывается в окне браузера MATLAB. Отчет содержит оглавление, гиперссылки которого позволяют быстро перейти к нужному разделу. Все созданные файлы (документ в формате HTML и рисунки) размещены в подкаталоге html каталога, содержащего М-файл.

Для опубликования отчета в других форматах следует в меню **File** редактора М-файлов перейти к подменю **Publish to** и задать желаемый формат выбором одного из его пунктов.

Перечислим некоторые важные настройки процесса публикации, которые устанавливаются в диалоговом окне **Preferences**, появляющемся после выбо-

ра в меню редактора **File** пункта **Preferences**. В левой части окна перейдите к разделу **Publishing**. В правой части окна флаг **Evaluate code** должен быть установлен для автоматического выполнения ячеек М-файла при создании отчета, а за включение в отчет самих команд отвечает флаг **Display code in output**. Для задания графического формата для экспорта графиков следует перейти к разделу **Publishing images** и выбрать желаемый формат в раскрывающемся списке **Image file type**.

Описанная в этом разделе публикация отчета позволяет получить готовый документ в одном из широко распространенных форматов и внести в него необходимые дополнения по мере надобности. В следующем разделе мы рассмотрим другой способ — создание интерактивных документов MS Word (М-книг), которые не только содержат результаты работы, но и позволяют повторить их прямо из документа, открытого в MS Word.

## М-КНИГИ

М-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 (М-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, оперируя всеми средствами MS 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 М-книги

Перед началом работы над М-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 М-книги. Продолжение работы над существующими М-книгами и разработка новых не требуют повторных настроек. Разумеется, при переустановке MS Word или MATLAB придется произвести процесс настройки сначала.

Суть настройки состоит в том, что создается шаблон документа (М-книги) с необходимыми стилями форматирования и макросами. Создание М-книги в MS Word на основе этого шаблона приведет к встраиванию в интерфейс редактора MS Word средств для связи с MATLAB.

До начала настройки определите каталог, в который будет помещен шаблон для М-книги. В этот каталог следует поместить и копию шаблона normal.dot. Будем для определенности считать, что выполнена стандартная установка MS Office и создан каталог C:\Program Files\Microsoft Office\Templates, где будем размещать файлы шаблонов (далее этот каталог имеется просто Templates).

### Примечание

Для пакета MS Office 2000 или старше, установленного в ОС, базирующейся на сетевой платформе, файл шаблона normal.dot находится в каталоге пользователя: C:\Documents and Settings\имя\_пользователя\Application Data\ в подкаталоге Шаблоны или Templates (в зависимости от локализации версии).

Запустите MATLAB и наберите в командном окне `notebook('-setup')`. Запрашивается номер версии MS Word, установленной на вашем компьютере. Выберите нужную цифру и следуйте появляющимся инструкциям. При правильной установке программного обеспечения выводится сообщение о том, что после нажатия на любую клавишу появится диалоговое окно для указания пути к шаблону normal.dot. В каталоге Templates появился файл m-book.dot, являющийся шаблоном для создания М-книг.

Запустите MS Word и установите уровень безопасности, позволяющий подключать макросы. Для русифицированного пакета MS Office 2000 или старше соответствующее диалоговое окно открывается при выборе в меню **Сервис** пункта **Параметры**, в котором следует перейти к вкладке **Безопасность**.

Имеется несколько способов, позволяющих начать работу над новой М-книгой. Команда `notebook` приводит к появлению в MS Word нового файла, основанного на шаблоне m-book.dot. Если MS 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 m-book.dot, установить переключатель **документ** на панели **Создать** и нажать **OK**.

В MS Word создалось меню **Notebook**, предназначеннное для управления и редактирования интерактивной М-книги (рис. 21.1). Далее поясняется использование элементов этого меню.

В меню **Файл** добавился пункт **New M-book**; кроме того, всплывающее меню приобрело дополнительные пункты (рис. 21.2). В список стилей включены стили, определенные в m-book.dot: **AutoInit**, **Calc**, **Error**, **Input**, **NoGraph**, **Output**. По умолчанию используется стиль **Обычный**.

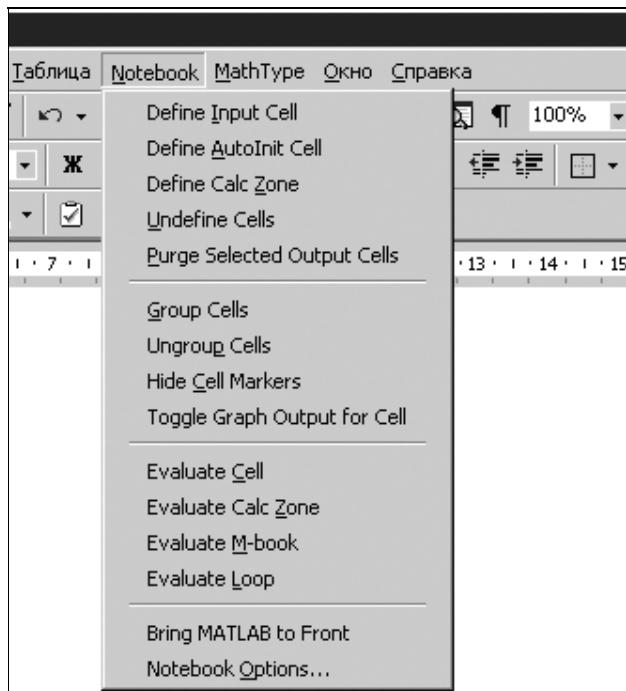


Рис. 21.1. Дополнительное меню Notebook в MS Word

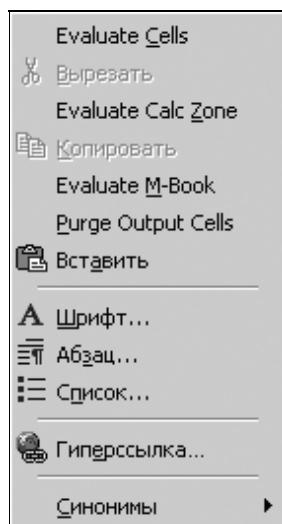


Рис. 21.2. Новый вид контекстного меню в MS Word

### Примечание

Вне зависимости от локализации версии MS 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 заключен 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.

## Группировка ячеек

Решение задачи линейной оптимизации о составлении рациона, описанное в главе 16, может быть наглядно продемонстрировано в М-книге. Сначала следует привести условие задачи и вспомогательную таблицу, а затем набрать операторы файл-программы ration (см. листинг 16.1) 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 21.2.

### Листинг 21.2. 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 21.2) 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 21.3).

### Листинг 21.3. 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.

```
x =
13.2143
16.7857
6.4286
```

Создание группы ячеек имеет ряд особенностей. Группа не должна содержать текст и другие объекты MS Word 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 М-книгой, создайте группу ячеек ввода с командами, обеспечивающими отображение круговой диаграммы полученного ре-

шения, и выполните ее (листинг 21.4). Соответствующая ячейка вывода содержит область графического окна с диаграммой (рис. 21.3). Важно понимать, что рисунок в ячейке вывода, заключенной в большие квадратные скобки, является объектом MS Word. Данное обстоятельство позволяет обращаться с ним, как с обычным рисунком, внедряемым в документ.

#### Листинг 21.4. Использование графических команд MATLAB

```
pie3(x)
HT = title('Пропорции продуктов в смеси');
set(HT, 'FontSize', 16)
[здесь размещается рис. 21.3]
```

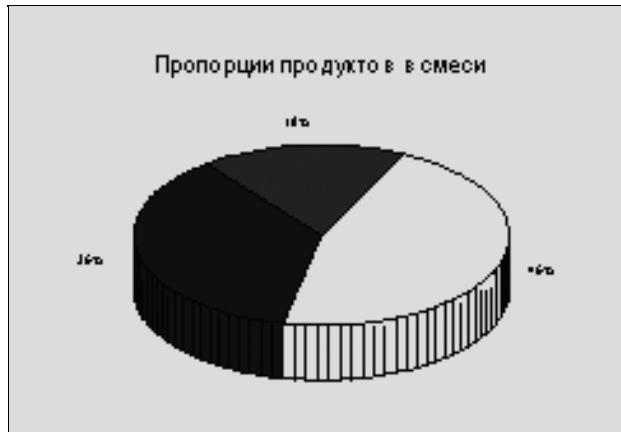


Рис. 21.3. Рисунок в ячейке вывода листинга 22.4, являющийся объектом MS Word

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 MS Word. Повторное выполнение команд, находящихся в соответствующих ячейках ввода, не нарушит расположения 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, приводит к останову вычислений. Для автоматического перехода к выполнению следующей ячейки необходимо установить флаг **Stop evaluating on error** в диалоговом окне **Notebook Options** (рис. 21.4), появляющемся при выборе одноименного пункта в меню **Notebook**.

Переменные различных разделов являются общими, к примеру, если в ячейке одного раздела переменной *x* было присвоено некоторое значение, то *x* можно использовать и в остальных разделах. Все переменные М-книги — глобальные. Более того, если в редакторе MS Word открыто несколько М-книг, то их переменные определены в одной рабочей среде.

Открытие М-книги в MS 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 команду *clear* для очистки рабочей среды. Для установки стиля **AutoInit** служит пункт **Define Autoinit Cell** меню **Notebook**.

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 графической информацией. Меню **Notebook** содержит пункт **Notebook Options**, выбор которого приводит к появлению одноименного диалогового окна, изображенного на рис. 21.4. 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.

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 раскрывающимся списке **Units**.

Ячейки вывода с окончательными результатами преобразовываются в текст выбором пункта **Undefine Cells** меню **Notebook**. Пользователь может переопределить стили шаблона *m-book.dot* так же, как и любого другого стиля,

выбрав в меню **Формат** пункт **Стили и форматирование** и произведя нужные установки в появившемся диалоговом окне.

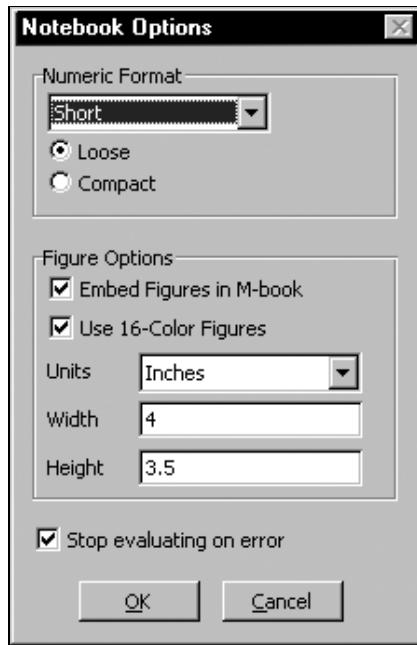


Рис. 21.4 Диалоговое окно Notebook Options

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.

## Совместная работа в MATLAB и MS Excel

Интегрирование MATLAB и MS Excel расширяет возможности электронных таблиц благодаря доступу к многочисленным функциям MATLAB для обработки данных, различных вычислений и визуализации результата. Надстройка excllink.xla реализует совместную работу в MATLAB и MS Excel. Для связи с MS Excel в MATLAB определен ряд специальных функций.

## Примечание

В состав MATLAB может входить также MATLAB Builder for MS Excel, который позволяет создавать COM-объекты для использования их в MS Excel, однако в следующих разделах мы ограничимся только обзором MS Excel Link.

## Конфигурирование MS Excel

Перед тем как настраивать MS Excel на совместную работу с MATLAB, следует убедиться, что MS Excel Link входит в установленную версию MATLAB. В подкаталоге exlink основного каталога MATLAB или подката-лога toolbox должен находиться файл с надстройкой exclink.xla. Запустите MS Excel и в меню **Сервис** выберите пункт **Надстройки**. Открывается диалоговое окно (рис. 21.5), содержащее информацию о доступных в данный момент надстройках.

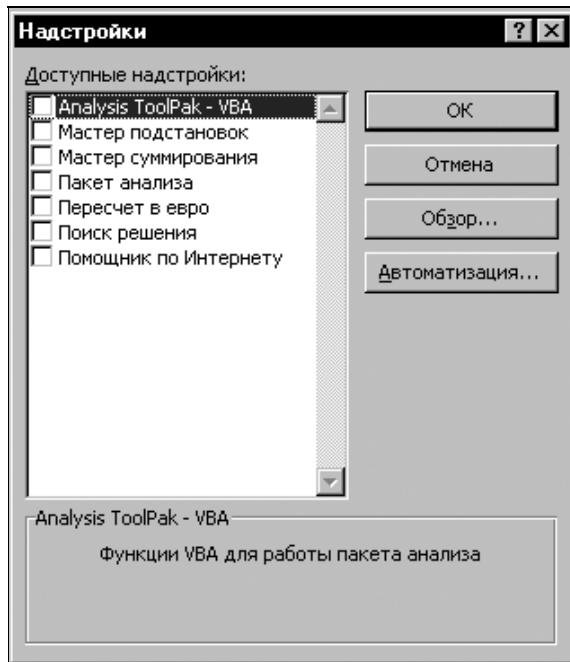


Рис. 21.5. Диалоговое окно MS Excel Надстройки

Используя кнопку **Обзор** укажите путь к файлу exclink.xla. В списке надстроек диалогового окна появилась строка **MS Excel Link 2.2 for use with**

**MATLAB** с установленным флагом. Нажмите **OK**, требуемая надстройка добавлена в MS Excel. Обратите внимание, что в MS Excel присутствует панель инструментов **MS Excel Link**, содержащая кнопки: **startmatlab**, **putmatrix**, **getmatrix**, **evalstring**. Данные кнопки реализуют основные действия, требуемые для осуществления взаимосвязи между MS Excel и MATLAB — обмен матричными данными и выполнение команд MATLAB из среды MS Excel.

При повторных запусках MS Excel надстройка excllink.xla подключается автоматически. Избежать подключения надстройки можно сбросом соответствующего флага в диалоговом окне **Надстройки**.

## Обмен данными между MATLAB и MS Excel

Запустите MS Excel, проверьте, что проделаны все необходимые настройки так, как описано в предыдущем разделе. Введите в ячейки с A1 по C3 матрицу (рис. 21.6), для отделения десятичных знаков используйте запятую в соответствии с требованиями MS 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. 21.6. Ввод матрицы в ячейки

Выделите на листе данные ячейки и нажмите кнопку **putmatrix**, появляется диалоговое окно MS Excel со строкой ввода, предназначеннной для определения имени переменной рабочей среды MATLAB, которую следует экспорттировать данные из выделенных ячеек MS 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);
```

## Примечание

Вызов `inv` для обращения матрицы, как и любой другой команды MATLAB, можно осуществить прямо из MS Excel. Нажатие на кнопку `evalstring`, расположенную на панели MS 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 MS Excel, сделайте текущей ячейку A5 и нажмите кнопку `getmatrix`. Появляется диалоговое окно со строкой ввода, в которой требуется ввести имя переменной, импортируемой в MS Excel. В данном случае такой переменной является `IM`. Нажмите **OK**, в ячейки с A5 по C7 заносятся элементы обратной матрицы.

MATLAB производит вычисления с двойной точностью. Если требуется отобразить в MS Excel все значащие цифры, то 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 MS 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 MS Excel, затем экспортируются в MATLAB, обрабатываются там некоторым образом, и результат импортируется в MS Excel. Пользователь переносит данные при помощи кнопок панели инструментов MS 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 MS Excel.

Наряду с числовыми массивами, объектами, подлежащими обмену между MATLAB и MS Excel, могут быть текстовые данные. Введите в ячейки некоторый текст (рис. 21.7).

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
```

```
mounth 4x3 1252 cell array
Grand total is 86 elements using 1252 bytes
```

|   | A       | B        | C         |
|---|---------|----------|-----------|
| 1 | January | February | March     |
| 2 | April   | May      | June      |
| 3 | July    | August   | September |
| 4 | October | November | December  |

**Рис. 21.7.** Ввод в ячейки текстовой информации

Оказывается, текстовая информация из ячеек MS Excel записывается в массив ячеек (*cell array*) MATLAB. Экспорт в MATLAB текста только одной ячейки рабочего листа MS 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 MS Excel приводит к заполнению прямоугольной области на рабочем листе, размеры которой совпадают с размерами импортируемого массива. Символьный массив импортируется в одну ячейку листа MS Excel.

Обмен данными между приложениями может быть осуществлен не только при помощи кнопок панели инструментов **MS Excel Link**, но и с использованием функций, определенных в надстройке MS Excel Link.

## Обращение к основным функциям **MS Excel Link**

Всего в MS Excel Link определено тринацать функций, распадающихся на две категории: функции для обмена данными между MATLAB и MS Excel и функции, предназначенные для установления связи между приложениями. Данные функции могут вызываться как из ячеек рабочего листа книги MS Excel, так и из приложений на Visual Basic. При обращении к функциям MS Excel Link из ячеек рабочего листа используется функциональная форма, т. е. аргументы заключаются в круглые скобки. Вызов функций из приложений на Visual Basic требует командной формы записи, в которой аргументы задаются через пробел после имени функции.

Для начала работы необходимы три основные функции, которые фактически дублируются кнопками панели инструментов **MS Excel Link**.

Функция *MLPutMatrix* служит для помещения данных из ячеек листа MS 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.

Обращение из MS 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 (рис. 21.6), затем поместите в ячейку E2 вызов функции `=MLPutMatrix("M"; A1:C3)`. Обращение к функции из ячейки рабочего листа начинается со знака "равно". Нажатие на <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 MS Excel с A5 по C7. Вид рабочего листа приведен на рис. 21.8, в ячейках с формулами отображаются нули, содержимое каждой из трех ячеек указано на рисунке отдельно.

|   |           |           |           |                         |
|---|-----------|-----------|-----------|-------------------------|
|   | E2        |           | =         | =MLPutMatrix("M";A1:C3) |
|   | A         | B         | C         | D                       |
| 1 | 5,5       | 1,6       | -0,8      |                         |
| 2 | 2,3       | 6,1       | 0,2       | 0                       |
| 3 | 0,1       | 0,4       | 3,9       |                         |
| 4 |           |           |           | 0                       |
| 5 | 0,204684  | -0,056631 | 0,044891  |                         |
| 6 | -0,077264 | 0,185865  | -0,025380 | 0                       |
| 7 | 0,002676  | -0,017611 | 0,257862  |                         |

=MLPutMatrix("M";A1:C3)  
 =MLEvalString("IM=inv(M)")  
 =MLGetMatrix("IM";"A5:C7")

Рис. 21.8. Содержимое рабочего листа

Пользователи, имеющие опыт программирования на Visual Basic, могут использовать функции MS Excel Link в своих программах. Листинг 21.5 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 `excllink.xls` при помощи пункта **Ссылки** меню **Сервис**.

### Листинг 21.5. 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 MS Excel Link могут оперировать с различным заданием области ячеек, в том числе и по имени. Подробные сведения содержатся в справочной системе MS Excel в разд. **Справочник по Visual Basic**. Несколько примеров, касающихся интегрирования MATLAB и MS Excel, включены в книгу ExlISamp.xls, которая находится в том же подкаталоге MATLAB, что и надстройка excllink.xla.

## Функции MS Excel Link

Как мы упоминали выше, имеется две группы функций MS Excel Link: функции для обмена данными между MATLAB и MS Excel и функции для связи между приложениями.

В MS Excel Link определено девять функций, обеспечивающих экспорт и импорт данных при совместной работе в MATLAB и MS Excel. Три из них: MLPutMatrix, MLEvalString и MLGetMatrix описаны в предыдущем разделе.

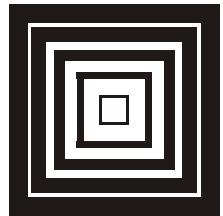
Функция matlabfcn вычисляет указанную в ней функцию MATLAB от массива данных, заданного диапазоном ячеек. Результат возвращается в ячейку, содержащую вызов matlabfcn. Если результат — массив, то следует прибегнуть к функции matlabsub, которая позволяет задать диапазон ячеек для вывода. Эти функции используются только в ячейках рабочего листа.

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 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 строки, так и столбцы, создаются новые строки. Если же размеры области ячеек не соответствуют массиву, то 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 (VBA) оказывается полезной функция `MLPutVar`, которая предназначена для экспорта значения переменной из процедуры на VBA в рабочую среду MATLAB. Первым аргументом `MLPutVar` является имя переменной MATLAB, а вторым — имя переменной в процедуре на VBA. Импорт данных из рабочей среды в переменную процедуры осуществляется функция `MLGetVar`. Порядок аргументов `MLGetVar` такой же, как и у `MLPutVar`. Очевидно, что две эти функции используются только при программировании на VBA.

Четыре функции: `matlabinit`, `MLAutoStart`, `MLClose`, `MLOpen`, обеспечивающие согласованную работу MATLAB и MS Excel, образуют вторую группу функций MS Excel Link. Функция `matlabinit` служит для инициализации MS Excel Link и запуска MATLAB. Для запуска только MATLAB следует обратиться к `MLOpen`. Функция `MLAutoStart` позволяет задать или отменить автоматический запуск MATLAB при открытии MS Excel. Для закрытия MATLAB вызывается `MLClose`.



## Глава 22

# Модернизация приложений с GUI версии 5.3

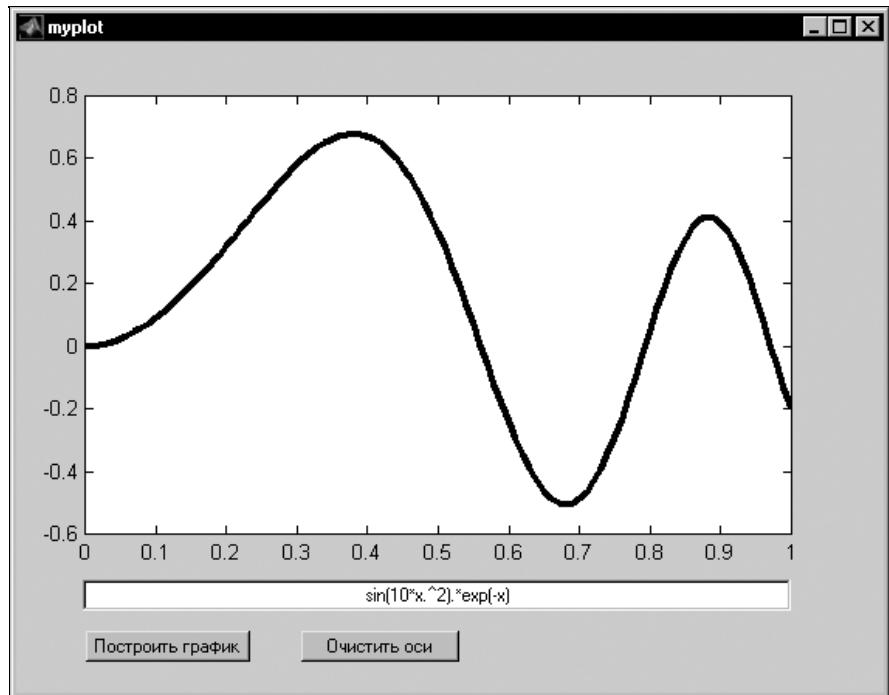
В этой главе описан процесс перевода приложения с графическим интерфейсом пользователя, которое было создано в среде GUIDE MATLAB 5.3, в формат, принятый в новых версиях MATLAB, включая и версию 7. Переход на новую версию MATLAB требует некоторой переработки старых приложений с графическим интерфейсом пользователя для того, чтобы их можно было использовать в последующих версиях. 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 М-файле, создаваемом программистом. Начиная с версии 6.0 в MATLAB принят другой формат хранения — с приложением связаны файлы с расширениями `fig` и `m`, причем М-файл создается автоматически в среде GUIDE 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. 22.1.

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 предназначе-

но для демонстрации способа модернизации приложений с графическим интерфейсом, написанных в MATLAB 5.3, для более новых версий. 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 файла-функции `myplotprog` содержится в листинге 22.1. Имена объектов окна приложения и вызовы `myplotprog` при возникновении событий `Callback` данных объектов приведены в табл. 22.1.



**Рис. 22.1.** Окно приложения `myplot`

**Таблица 22.1.** Имена объектов приложения `myplot`

| Объект                         | Имя      | Событие <code>Callback</code>         |
|--------------------------------|----------|---------------------------------------|
| Строка ввода                   | FunEdt   | <code>myplotprog('SetFun')</code>     |
| Кнопка <b>Построить график</b> | PlotBtn  | <code>myplotprog('PressPlot')</code>  |
| Кнопка <b>Очистить оси</b>     | ClearBtn | <code>myplotprog('PressClear')</code> |

**Листинг 22.1. 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 следующих. Обратитесь при необходимости к соответствующим разделам книги, посвященным дескрипторной графике и созданию приложений в MATLAB (использование дескрипторной графики описано в *главе 9, часть III* книги освещает вопросы написания приложений с GUI).

## Модернизация приложения

Приложение версии 5.3 с графическим интерфейсом пользователя, как правило, содержится в файлах с расширением mat и т. Пример такого приложения myplot приведен в предыдущем разделе. Данный раздел описывает основные этапы перевода файлов приложения в форматы FIG и M, свойственные более новым версиям MATLAB, и изменения связанные с приложением файл-функции обработки событий. Приложение приводится к такому виду, который обеспечивает дальнейшую работу над ним в среде GUIDE более новых версий MATLAB без каких-либо особенностей. Процесс мо-

дернизации не требует установленной версии 5.3, необходимо иметь только файлы приложения. Всюду дальше под MATLAB мы будем понимать версию старше 5.3.

## Сохранение приложения в формате FIG

Сохраните файлы myplot.m, myplot.mat и myplotprog.m в текущем каталоге MATLAB и запустите приложение, набрав `myplot` в командной строке MATLAB. Появляется окно приложения, причем оно функционирует так же, как и в 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 указать 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 и их свойства описаны в *главе 9*).

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 `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`, а новое имя. Прежние

имена приложений облегчат работу, особенно, если модернизации подлежит несколько часто используемых ранее приложений. Закройте среду GUIDE и перейдите к изучению содержимого текущего каталога.

В текущем каталоге MATLAB 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 22.2. Обратите внимание, что первым оператором 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 22.2. 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 удоб-

бен. 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 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 удобнее производить, если с приложением ассоциирован M-файл.

Сделайте доступными указатели на объекты так, как описано в предыдущем разделе, используя `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 приложение при помощи `guide(h)`.

Выберите в меню **Tools** среды GUIDE пункт **GUI Options**, появляется диалоговое окно **GUI 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 установить **Callback (GUI becomes Current Figure within Callbacks)**.

Сохраните приложение с именем `myplot.fig` в среде GUIDE, выбрав в меню **File** пункт **Save as**. Если вы ранее осуществляли 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 22.2), инициализирующую приложение в стиле версии 5.3. Поскольку требуется полностью отказаться от формата версии 5.3, то следует выбрать **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, в пункте **View Callbacks** выберите подпункт **Callback**. Обратите внимание, что в файле `myplot.m` появилась подфункция `FunEdt_Callback` обработки события `CallBack` строки ввода. Аналогичным образом последовательно создай-

те подфункции PlotBtn\_Callback и ClearBtn\_Callback, соответствующие кнопкам **Построить график** и **Очистить оси**. Разместите в теле каждой из этих подфункций обращение к myplotprog с соответствующим входным аргументом. В результате файл myplot.m должен иметь структуру, приведенную в листинге 22.3 (автоматически созданные комментарии не приводятся для экономии места). Запустите из среды GUIDE приложение и убедитесь, что оно работает правильно.

### Листинг 22.3. Содержимое сгенерированного файла myplot с подфункциями

```
function varargout = myplot(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
 'gui_Singleton', gui_Singleton, ...
 'gui_OpeningFcn', @myplot_OpeningFcn, ...
 'gui_OutputFcn', @myplot_OutputFcn, ...
 'gui_LayoutFcn', [], ...
 'gui_Callback', []);
```

```
if nargin && ischar(varargin{1})
 gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
 [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
 gui_mainfcn(gui_State, varargin{:});
end
```

```
function myplot_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
```

```
function varargout = myplot_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
function FunEdt_Callback(hObject, eventdata, handles)
myplotprog('SetFun')
```

```
function PlotBtn_Callback(hObject, eventdata, handles)
myplotprog('PressPlot')

function ClearBtn_Callback(hObject, eventdata, handles)
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.

Указатели на графические объекты содержатся в полях структуры handles, имена которых совпадают со значениями свойства Tag данных объектов. Обмен 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 11).

Подфункции обработки событий элементов интерфейса обновленного приложения myplot приведены в листинге 22.4.

#### Листинг 22.4. Подфункции обработки событий обновленного приложения myplot

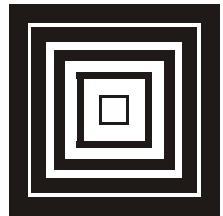
```
function FunEdt_Callback(hObject, eventdata, handles)
% Образование нового поля FunStr структуры handles и занесение в него
% содержимого строки ввода
handles.FunStr = get(hObject, 'String');
% Сохранение обновленной структуры handles
guidata(hObject, handles)

function PlotBtn_Callback(hObject, eventdata, handles)
% Задание вектора аргументов для построения графика
x = 0:0.01:1;
```

```
% Конструирование и выполнение строки для вычисления вектора значений
% аргумента
eval(['y = ' handles.FunStr ';'])
% Построение графика
plot(x, y, 'LineWidth', 3, 'Color', 'k')

function ClearBtn_Callback(hObject, eventdata, handles)
% очистка осей
cla
```

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 23

# Повышение производительности приложений MATLAB

Данная глава описывает основные принципы эффективной разработки приложений в MATLAB, которые позволяют снижать время счета и экономить память. На достаточно простом уровне объяснено использование программного интерфейса MATLAB API для вызова внешних процедур, написанных на других языках программирования.

## Ускорение работы М-файлов, экономия памяти

MATLAB интерпретирует команды, записанные в М-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.

## 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` (лис-

тинг 23.1), а второй — в файл-программе test2 (листинг 23.2). 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 23.1. 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 23.2. 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!

## Примечание

1. Значения времени, полученные на различных компьютерах, могут не совпадать с приведенными в книге, но соотношение остается приблизительно тем же.
2. Функции `tic` и `toc` служат для определения реального времени выполнения команд. Процессорное время возвращается функцией `cputime`, например: `T1 = cputime;` `test1;` `T2 = cputime - T1.`

Следующая задача имеет менее очевидное решение. Предположим, что требуется преобразовать элементы матрицы по формулам:  $A(i, j) = A(i, j) / (i * j)$ . Деление всех элементов матрицы  $A$  во вложенных циклах не представляет труда (листинг 23.3).

### Листинг 23.3. 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 произведением подходящего столбца и строки (листинг 23.4).

### Листинг 23.4. 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
```

Заметьте, что после поэлементного деления массив *M* не нужен — его следует удалить из памяти. Основные приемы работы с памятью описаны в следующем разделе.

## Экономия памяти

Один из самых очевидных способов экономии памяти заключается в удалении тех переменных, которые больше не понадобятся. Для этого существуют два способа. Во-первых, одна или несколько переменных выделяются в окне **Workspace** и удаляются при помощи <Delete> или одноименного пункта контекстного меню. Второй способ, пригодный в приложениях, состоит в использовании команды *clear*. Удаляемые переменные заносятся в список ее параметров, например: *clear hlp1 tmp3*. Если эти переменные были созданы в файл-программе или из командной строки, то они удаляются из основной среды MATLAB. Переменные файл-функции удаляются из среды файл-функции. Однако если они были объявлены как глобальные, то они останутся в памяти и будут доступны для остальных функций (работа с глобальными переменными описана в разд. "Подфункции" главы 5).

Для удаления глобальных переменных следует в качестве первого параметра *clear* указать *global*, например: *clear global alpha beta*.

Команда *clear* может быть вызвана и в функциональной форме: *clear('hlp1', 'tmp3')*, что удобно, если имена удаляемых переменных хранятся в некоторых строковых переменных.

Удаление переменных не всегда желательно. Возможно, что в одном блоке программы массив с вычисленными значениями не нужен, но он понадобится в другом ее блоке. Если этот массив занимает много памяти, то имеет смысл записать содержащиеся в нем данные на диск, удалить его из памяти и загружать по мере надобности. Для этого служат команды *save* и *load*. При чтении главы 1 мы применяли их для сохранения значений сразу всех переменных рабочей среды, а при чтении главы 2 — для сохранения числовых данных в текстовом файле и считывания. Обсудим возможности этих команд более подробно.

Как *save*, так и *load* могут быть вызваны в командной и функциональной формах. Например: *save filename A B* и *save(filename, 'A', 'B')*, где *filename* — имя файла, приводят к одинаковому результату. При функциональном способе вместо имени переменной в апострофах можно указывать строковую переменную, что иногда оказывается удобным. По умолчанию

переменные сохраняются в двоичных файлах с расширением mat, которые занимают меньше места, чем текстовые, и записываются быстрее.

### Примечание

Для того чтобы mat-файлы читались в версиях 6.x, следует в качестве последнего параметра save установить опцию -v6, например: `save('DataFile', 'A', '-v6')`.

Для записи значений переменных приложения можно использовать всего один файл. Если запись происходит в нескольких местах, т. е. значения добавляются в файл, то указывается опция -append, например:

```
A = rand(100);
save('DataFile', 'A')

B = A.^2;
save('DataFile', 'B', '-append');
clear('A', 'B')
```

Для последующей загрузки переменных применяется load, например, `load('DataFile', 'A')` приводит к появлению массива A в рабочей среде.

В приведенных примерах мы не указывали расширение файла данных — оно задавалось mat по умолчанию. Предположим, что переменные были записаны в файле с другим расширением, например, myvars.dat при помощи `save('myvars.dat', 'A', 'B')`. Тогда при считывании их значений необходимо принять во внимание следующее обстоятельство. Файл myvars.dat является двоичным, поскольку при сохранении не была задана опция -ascii. С другой стороны, load интерпретирует все файлы с расширением, отличным от mat, как текстовые. Поэтому обращение `load('myvars.dat', 'A', 'B')` приведет к ошибке — вместо этого следует задать дополнительный параметр -mat: `load('myvars.dat', 'A', 'B', '-mat')`.

Увеличения доступной памяти можно добиться за счет процедуры, которую принято называть "сборкой мусора". Для этого следует воспользоваться командой pack, которая размещает переменные MATLAB в памяти оптимальным образом. Дело в том, что во время сеанса работы данные не упорядочиваются в памяти и это часто приводит к наличию свободных несмежных фрагментов, каждого из которых недостаточно для размещения большого массива. Команда pack записывает переменные во временный файл в текущем каталоге (по умолчанию в pack.tmp), полностью освобождает память рабочей среды и затем снова загружает переменные, так что становится доступной непрерывная область памяти. При такой организации памяти в MATLAB предпочтительнее вводить первыми самые большие массивы (см. пример в разделе справочной системы **MATLAB: Programming**:

## Improving Performance and Memory Usage: Making Efficient Use of Memory: Working with Variables).

Существенной экономии памяти и времени счета можно добиться при учете структуры данных. Например, матрицы с большим количеством нулевых элементов следует хранить в разреженном виде (работа с разреженными матрицами описана в главе 15).

Не менее важно принимать во внимание и тип данных. Так, если данные содержат только целые значения, то имеет смысл использовать для их хранения целочисленные массивы подходящего типа (`int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`) в зависимости от максимального и минимального значений данных.

Числа 8, 16 или 32 в названии типа говорят о том, сколько бит занимает одна переменная данного типа. Допустимые значения переменных для каждого из типов приведены в справочной системе MATLAB (см., например, разд. **MATLAB: Programming: Data Types: Numeric Types**).

Для инициализации подходящего массива данных можно применить следующее обращение к одной из функций `ones`, `eye` или `zeros`:

```
>> i8A = ones(1000, 'int8');
>> whos i8A
Name Size Bytes Class
i8A 1000x1000 1000000 int8 array
Grand total is 1000000 elements using 1000000 bytes
```

Дальнейшие операции с массивом чисел типа `int8` сохраняют тип данных, что необходимо учитывать для избежания ошибок:

```
>> i8A(:, :) = 200;
>> whos i8A
Name Size Bytes Class
i8A 1000x1000 1000000 int8 array
Grand total is 1000000 elements using 1000000 bytes
>> i8A(1, 1)
ans =
 127
```

При работе с вещественными данными не всегда оправданно применение двойной точности, т. е. массивов типа `double`, которые используются по умолчанию. В ряде случаев для экономии памяти в два раза целесообразно ограничиться обычной точностью и преобразовать массив данных к типу `single` при помощи одноименной функции:

```
>> dA = ones(1000);
>> sA = single(dA);
```

```
>> whos dA sA
 Name Size Bytes Class
 dA 1000x1000 8000000 double array
 sA 1000x1000 4000000 single array
Grand total is 2000000 elements using 12000000 bytes
```

Вместо преобразования массива данных к типу *single* можно было сразу начать работу с этим типом, инициализировав нулевой массив:

```
>> sA = zeros(1000, 'single');
>> sA(:, :) = 0.33;
>> sA = sA/2;
>> sA = sA.^2;
>> sA = sin(sA);
>> whos sA
 Name Size Bytes Class
 sA 1000x1000 4000000 single array
Grand total is 1000000 elements using 4000000 bytes
```

Предварительная инициализация массивов, которую мы только что рассмотрели, позволяет сократить время расчетов и уменьшает фрагментацию памяти по сравнению с динамическим выделением памяти под создаваемый массив.

## Выделение памяти под массивы

Массивы среди MATLAB не требуют предварительного объявления в отличие от многих других языков программирования, а размеры массива динамически 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 одинаковым матрицам 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 400. 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 23.5 и 23.6).

#### Листинг 23.5. 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 23.6. 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 по-

нимать, 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
```

Анализ кода файл-программы `test5` при помощи M-Lint сопровождается появлением того же самого совета по предварительной инициализации массива (запуск M-Lint описан в разд. "Диагностика M-файлов" главы 5).

Принципиально иной способ ускорения выполнения приложений MATLAB за счет вызова внешних процедур на других языках программирования описан в следующем разделе.

## Связь MATLAB с другими языками программирования

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`. В среде Windows MEX-файл является динамически подключаемой библиотекой и имеет расширение `dll`. Генерация MEX-файлов из процедур Fortran требует установки подходящего компилятора, например, одного из: Digital Visual Fortran или Compaq Visual Fortran. Встроенный компилятор Fortran не входит в пакет MATLAB в отличие от компилятора Lcc для приложений на C.

По сравнению с подходами по повышению эффективности приложений MATLAB, которые мы рассмотрели выше, MEX-файлы предоставляют принципиально другой способ повышения быстродействия программ. Он состоит в написании внешней процедуры на Fortran или C, создании MEX-файла и обращении к нему из среды и приложений MATLAB. Простой пример такой внешней процедуры рассмотрен в последнем разделе этой главы.

## Конфигурирование MATLAB Compiler

Перед компиляцией программ необходимо произвести некоторые установки, связанные с выбором компилятора. Если компилятор не выбран, то по умолчанию используется встроенный Lcc для приложений на С. Приступим к конфигурированию MATLAB Compiler. Наберите в командной строке

```
>> mex -setup
```

### Предупреждение

Некоторые антивирусные программы могут конфликтовать с процедурой конфигурирования или компиляцией. Если после набора команды mex велось сообщение об ошибке, то следует временно отключить антивирусную программу.

Запускается программа с интерфейсом из командной строки и предлагается автоматический поиск 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?
```

Ведите **у** (подтверждение) на запрос об автоматическом поиске. Появляется список доступных компиляторов С и Fortran (который может отличаться от приведенного ниже):

```
Select a compiler:
```

```
[1] Compaq Visual Fortran version 6.1 in C:\Program Files\Microsoft
Visual Studio
```

```
[2] Lcc C version 2.4 in C:\MATLAB7\sys\lcc
```

```
[3] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft
Visual Studio
```

```
[0] None
```

```
Compiler:
```

Поскольку далее мы разберем использование внешних процедур на Fortran, то следует ввести номер соответствующего компилятора (в данном случае первый) и нажать <Enter>. Далее предлагается подтвердить сделанный выбор:

```
Please verify your choices:
```

```
Compiler: Compaq Visual Fortran 6.1
```

```
Location: C:\Program Files\Microsoft Visual Studio
```

```
Are these correct? ([y]/n):
```

Ведите **у** и нажмите <Enter>. Конфигурирование MATLAB Compiler завершено. Переийдите к изучению некоторых примеров, разобранных в следующих разделах.

## 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 90. Процедура `sum` складывает два вещественных числа `a` и `b` и записывает результат в `c` (листинг 23.7). Будем считать, что процедура `sum` содержится в файле `mysum.f90`.

### Листинг 23.7. Исходная процедура `sum` на Fortran 90 (файл `mysum.f90`)

```
subroutine sum(a, b, c)
! Процедура sum складывает два вещественных числа a и b
! и возвращает результат в c
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`:

1. `nrhs` — число *входных* аргументов, с которыми будет происходить обращение к MEX-функции `mysum` из среды MATLAB.
2. `prhs` — массив указателей на *входные* аргументы `mysum`.
3. `nlhs` — число *выходных* аргументов, с которыми будет происходить обращение к MEX-функции `mysum` из среды MATLAB.
4. `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 23.8, назовите файл с ней `mysumg.f90`. Итак, интерфейсная процедура имеет четыре параметра типа `integer*4`, два массива `plhs`, `prhs` и два скаляра `nlhs`, `nrhs`.

В `mexFunction` использованы две функции MATLAB API: `mxCreateDoubleMatrix` и `mxGetPr`, возвращающие значения типа `integer*4`. Функция `mxCreateDoubleMatrix` 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 будет иметь 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`.

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 единственным 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 23.7), занесен в переменную `c`.

Осталось обеспечить возвращение MEX-функцией в выходном аргументе значения переменной `c`. Предварительно необходимо создать массив размера один на один при помощи функции `mxCreateDoubleMatrix1` MATLAB API. Первыми двумя входными аргументами `mxCreateDoubleMatrix` является число строк и столбцов создаваемого массива, а третьим — ноль или единица. Ноль, указанный в третьем аргументе, соответствует вещественному массиву, а единица — комплексному. Функция `mxCreateDoubleMatrix` 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 переда-

чи 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 23.8. Интерфейсная процедура `mexFunction` (файл `mysumg.f90`)

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
! Интерфейсная процедура для sum
! Описание типов аргументов
integer nlhs, nrhs, plhs(*), prhs(*)
! Описание типов используемых функций из MATLAB API
integer mxCreateDoubleMatrix, mxGetPr
! Описание типов указателей на используемые переменные
integer a_pr, b_pr, c_pr
! Описание типов используемых переменных
real*8 a, b, c

! Получение указателей на первый и второй входные аргументы, с которыми
! вызывается MEX-функция, указатели хранятся в массиве prhs,
! используется функция MATLAB API
a_pr = mxGetPr(prhs(1))
b_pr = mxGetPr(prhs(2))
! Запись значений первого и второго входных аргументов с указателями
! 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)
! Вызов процедуры sum, которая заносит в переменную c сумму a и b
call sum(a, b, c)
! Создание выходного аргумента — вещественного массива размера один на
! один при помощи функции MATLAB API, выходной аргумент есть указатель
! на массив
plhs(1) = mxCreateDoubleMatrix(1, 1, 0)
c_pr = mxGetPr(plhs(1))
```

```
! Копирование значения с 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.f90`, и интерфейсной процедуры `mexFunction`, которая записана в файле `mysumg.f90`. Для создания MEX-функции служит команда `mex`. Напомним, что перед применением `mex` следует определить, каким компилятором Fortran будет пользоваться MATLAB для генерации MEX-функции (см. разд. "Конфигурирование MATLAB Compiler" данной главы).

После настройки MATLAB на использование компилятора Fortran можно приступить к окончательному этапу — созданию MEX-функции. Убедитесь, что файлы `mysum.f90` и `mysumg.f90` находятся в текущем каталоге MATLAB, и выполните команду `mex`, указав в качестве ее параметров имена файлов:

```
>> mex mysum.f90 mysumg.f90
```

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
```

После вызова MEX-функция остается в памяти. Если она не нужна, то имеет смысл выгрузить ее при помощи команды `clear`:

```
>> clear mysum
```

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:

1. Получение значений входных аргументов.
2. Вызов процедуры на Fortran.
3. 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 интерфейсной процедуре. Кроме функции `mxGetPr`, придется использовать `mxGetPi`, которая возвращает указатель на комплексную часть первого элемента массива. Получение и запись значения переменной по указателю производятся при помощи процедур `mxCopyPtrToComplex16` и `mxCopyComplex16ToPtr`, причем входным аргументом, кроме указателя на вещественную, является еще и указатель на комплексную часть. При создании массива функцией `mxCreateDoubleMatrix` следует задать единицу в качестве третьего аргумента, поскольку предполагается хранение комплексных чисел. Текст исходной процедуры на Fortran приведен в листинге 23.9, а отвечающая ей интерфейсная процедура в листинге 23.10.

### Листинг 23.9. Процедура сложения комплексных чисел (файл `mymsum.f90`)

```
subroutine sum(a, b, c)
! Процедура sum складывает два комплексных числа a и b
! и возвращает результат в c
complex*16 a, b, c
c = a + b
end
```

### Листинг 23.10. Интерфейсная процедура `mexFunction` (файл `mymsumg.f90`)

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
! Интерфейсная процедура для sum
! Описание типов аргументов
integer nlhs, nrhs, plhs(*), prhs(*)
! Описание типов используемых функций из MATLAB API
integer mxCreateFull, mxGetPr, mxGetPi
! Описание типов указателей на используемые переменные
integer a_pr, b_pr, c_pr, a_pi, b_pi, c_pi
! Описание типов используемых переменных
complex*16 a, b, c
```

```

! Получение указателей на вещественные и мнимые части первого и второго
! входных аргументов, с которыми вызывается MEX-функция, указатели
! 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))

! Запись значений первого и второго входных аргументов с указателями
! a_pr, a_pi и b_pr, b_pi в переменные a и b при помощи
! процедуры MATLAB API
call mxCopyPtrToComplex16(a_pr, a_pi, a, 1)
call mxCopyPtrToComplex16(b_pr, b_pi, b, 1)

! Вызов процедуры sum, которая заносит в переменную c сумму a и b
 call sum(a, b, c)

! Создание выходного аргумента – комплексного массива размера один на
! один при помощи функции MATLAB API, выходной аргумент есть указатель
! на массив
plhs(1) = mxCreateDoubleMatrix(1, 1, 1)
c_pr = mxGetPr(plhs(1))
c_pi = mxGetPi(plhs(1))

! Копирование значения с в массив с указателями
! 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.f90 и mycsumg.f90 и скомпилируйте MEX-функцию при помощи команды mex, рассмотренной в предыдущем разделе:

```
>> mex mycsum.f90 mycsumg.f90
```

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 =
```

## Обмен массивами данных

Передачу массивов данных в процедуру на Fortran и возвращение их в рабочую среду MATLAB проиллюстрируем на примере перемножения двух матриц. Создание процедуры для нахождения произведения двух матриц на Fortran не представляет труда, ее текст приведен в листинге 23.11.

### Листинг 23.11. Процедура перемножения матриц (файл `mymult.f90`)

```
subroutine mymult(n1, n2, n3, X, Y, Z)
! 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 i = 1, n1
 do j = 1, n3
 Z(i, j) = 0.0d0
 do k = 1, n2
 Z(i, j) = Z(i, j) + X(i, k)*Y(k, j)
 enddo
 enddo
enddo
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 23.12).

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 сов-

падение  $p$  и  $q$ . В случае, когда  $p$  не равно  $q$ , необходимо прекратить работу программы и вывести соответствующее сообщение в командное окно MATLAB. Процедура `mexErrMsgTxt` MATLAB API служит для останова по ошибке. Стока с сообщением об ошибке указывается в качестве входного аргумента `mexErrMsgTxt`.

### Примечание

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 разд. **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 `mxCreateDoubleMatrix`, записать указатель на него в первый элемент массива `plhs` и воспользоваться `mxGetPr` и `mxCopyReal8ToPtr`.

#### Листинг 23.12. Интерфейсная процедура (файл `mymultg.f90`)

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
! Интерфейсная процедура для mymult
! Описание типов аргументов
integer nlhs, nrhs
integer plhs(*), prhs(*)
! Описание типов используемых функций из MATLAB API
integer mxGetM, mxGetN, mxGetPr, mxCreateDoubleMatrix
! Описание типов указателей на используемые переменные
integer A_pr, B_pr, C_pr
! Переменные m, n, p, q используются для записи размеров матриц
integer m, n, p, q
! Предполагается, что матрицы не могут иметь размер, больший 50
real*8 A(50, 50), B(50, 50), C(50, 50)
```

```

! Получение размеров матриц — входных аргументов MEX-функции
m = mxGetM(prhs(1))
n = mxGetN(prhs(1))
p = mxGetM(prhs(2))
q = mxGetN(prhs(2))
! Проверка соответствующих размеров матриц
if (n.ne.p) then
 call mexErrMsgTxt('Matrix dimensions must agree!')
endif
! 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 matrix > 50')
endif
! Запись указателей на матрицы в соответствующие переменные A_pr и B_pr
A_pr = mxGetPr(prhs(1))
B_pr = mxGetPr(prhs(2))
! Занесение данных из матриц в массивы A и B
call mxCopyPtrToReal8(A_pr, A, m*n)
call mxCopyPtrToReal8(B_pr, B, p*q)
! Вызов процедуры multmtr, которая заносит в массив C произведение A*B
call multmtr(m, n, q, A, B, C)
! Создание выходного аргумента — вещественного массива размера m на q
! при помощи функции MATLAB API, выходной аргумент является указателем
! на массив
plhs(1) = mxCreateDoubleMatrix(m, q, 0)
C_pr = mxGetPr(plhs(1))
! Копирование значений C в массив с указателем C_pr
call mxCopyReal8ToPtr(C, C_pr, m*q)
end

```

Сгенерируйте MEX-функцию mymult, поместив файлы mymult.f90 и mymultg.f90 в текущий каталог MATLAB, и проверьте ее работу:

```

>> mex mymult.f90 mymultg.f90
>> 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 =

```

```
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 23.12, поскольку процедура `multmtr` (см. листинг 23.11) перемножает матрицы без ограничения на их размеры. Объявление динамических массивов в интерфейсной процедуре, приведенной в листинге 23.13, позволяет решить эту проблему. Создайте MEX-файл, используя улучшенную интерфейсную процедуру, и проверьте его работу.

#### Листинг 23.13. Интерфейсная процедура с динамическими массивами `mymultg2.f90`

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
integer nlhs, nrhs
integer plhs(*), prhs(*)
integer mxGetM, mxGetN, mxGetPr, mxCreateDoubleMatrix
integer A_pr, B_pr, C_pr
integer m, n, p, q
!
! 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 agree!')
endif
!
! Выделение памяти для записи значений первого входного
! аргумента MEX-функции в массив A
allocate(A(m, n))
A_pr = mxGetPr(prhs(1))
call mxCopyPtrToReal8(A_pr, A, m*n)
!
! Выделение памяти для записи значений второго входного
! аргумента MEX-функции в массив B
allocate(B(p, q))
B_pr = mxGetPr(prhs(2))
call mxCopyPtrToReal8(B_pr, B, p*q)
!
! Выделение памяти под результат процедуры multmtr
allocate(C(m, q))
call multmtr(m, n, q, A, B, C)
!
! Высвобождение памяти, занятой под массивы A и B
deallocate(A, B)
plhs(1) = mxCreateDoubleMatrix(m, q, 0)
C_pr = mxGetPr(plhs(1))
call mxCopyReal8ToPtr(C, C_pr, m*q)
!
! Высвобождение памяти, занятой под массив C
deallocate(C)
end
```

## Ускорение работы при использовании циклов

Использование внешних процедур в MATLAB позволяет ускорить работу приложений MATLAB, содержащих циклы, особенно вложенные. Например, MEX-функция `tumult` для перемножения матриц, описанная в предыдущем разделе, работает намного быстрее, чем аналогичная функция, написанная на встроенным языке программирования MATLAB. Проверьте это, создав файл-функцию `tumultfun`, которая так же как и `tumult`, находит произведение матриц в трех вложенных циклах.

### Примечание

Мы приводим этот пример только для демонстрации ускорения работы приложения с вложенными циклами. Оператор `*` все равно быстрее вы-

числит произведение матриц в MATLAB. Программирование внешней функции имеет смысл, если в MATLAB нет соответствующего оператора или функции и в циклах требуется обработать данные большого объема.

Текст файл-функции `mymultfun` приведен в листинге 23.14. Алгоритм очевиден, мы привели его только для того, чтобы подчеркнуть, что использованы рекомендации по ускорению работы файл-функций. Например, перед заполнением массива `C` он сначала инициализируется (см. разд. "Выделение памяти под массивы" данной главы).

#### Листинг 23.14. Файл-функция `mymultfun` для перемножения матриц

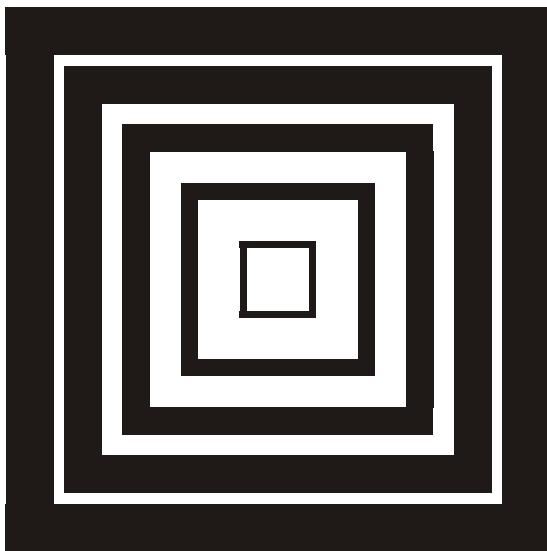
```
function C = mymultfun(A, B)
[m, n] = size(A);
[p, q] = size(B);
C = zeros(m, q);
for i = 1:m
 for j = 1:q
 C(i, j) = 0;
 for k = 1:n
 C(i, j) = C(i, j) + A(i, k)*B(k, j);
 end
 end
end
```

Сравните время работы MEX-функции `mymult` и файл-функции `mymultfun`:

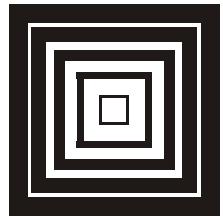
```
>> A = rand(400);
>> B = rand(400);
>> tic, C = mymultfun(A, B); toc
Elapsed time is 8.500000 seconds.
>> tic, C = mymult(A, B); toc
Elapsed time is 1.594000 seconds.
```

Для разных компьютеров временные затраты могут различаться, но соотношение остается примерно тем же — MEX-функция в несколько раз экономит время счета.

В заключение этой главы отметим, что вы имеете возможность вызывать функции MATLAB из собственных программ на Fortran или C (см. разд. **MATLAB: External Interfaces: Calling MATLAB from C and Fortran Programs** справочной системы MATLAB).



# ПРИЛОЖЕНИЯ



## 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 и Toolbox.

## Управление средой, файлами и переменными Получение справочной информации

- `demo` — получение доступа к демонстрационным файлам в окне **Help**.
- `doc` — запуск справочной системы по всем разделам MATLAB. В качестве параметра может быть указано имя функции или команды, для которой требуется получить справочную информацию:

`doc plot`

или имя Toolbox:

`doc toolbox\pde`

- `help` — вывод разделов встроенной справки в командное окно
  - `help elfun` — содержимое раздела `elfun` с информацией о встроенных элементарных математических функциях.  
В тексте названия функций и команд приведено заглавными буквами, но при работе надо набирать команды и функции строчными буквами.
  - `help floor` — описание функции `floor`.

- `help /` — информации об операторах и специальных символах.
  - `help syntax` — справка о синтаксисе команд и функций MATLAB.
- helpwin** — запуск краткой справочной системы по разделам MATLAB в окне **Help**.
- lookfor** — поиск M-файлов, в которых первая строка комментариев содержит заданный текст, например:
- ```
lookfor integral
```
- Просматриваются файлы, находящиеся в путях поиска MATLAB (подробнее про пути поиска написано в разд. "Установка путей" главы 5).
- ver** — вывод в командное окно версий MATLAB, Toolbox и дополнительных пакетов, входящих в установленную версию.
- version** — вывод в командное окно версии MATLAB.
- whatsnew** — вывод в окно **Help** информации о новшествах, отличающихся данной версию от предыдущей.

Управление средой MATLAB

- addpath** — добавление каталога в пути поиска MATLAB (подробнее про пути поиска написано в разд. "Установка путей" главы 5). 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.
- clc** — очистка содержимого командного окна.
- echo** — управление отображением строк выполняемого M-файла в командном окне.
- `echo on` — вывод строк.
 - `echo off` — подавление вывода строк.
- format** — определение формата вывода чисел в командное окно (см. разд. "Форматы вывода результата вычислений" главы 1).
- `format short` — формат с плавающей точкой с 4-мя цифрами после десятичной точки.

- `format long` — формат с плавающей точкой с 14-ю цифрами после десятичной точки (для одинарной точности — с 7-ю).
- `format short e` — экспоненциальный формат с 5-ю значащими цифрами.
- `format long e` — экспоненциальный формат с 15-ю значащими цифрами (для одинарной точности — с 8-ю).
- `format short g` — наиболее подходящий из экспоненциального формата и формата с плавающей точкой с 4-мя цифрами.
- `format long g` — наиболее подходящий из экспоненциального формата и формата с плавающей точкой с 15-ю цифрами (для одинарной точности — с 7-ю).
- `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.

□ **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" 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 предупреждения.
- **mlint** — анализатор кода М-файла (работа с графической оболочкой M-Lint описана в разд. "Диагностика M-файлов" главы 5).
 - `mlint('myprog.m')` — выводит информацию о замеченных ошибках и неточностях в М-файле и способах повышения его производительности.
 - `mlint({'myprog1.m'; 'myprog2.m'; 'myprog3.m'})` — одновременный анализ нескольких файлов (имена задаются в массиве ячеек, работа с массивами ячеек описана в разд. "Массивы структур и массивы ячеек" главы 8).
 - `info = mlint('myprog.m')` — запись информации о каждом обнаруженном замечании в массив структур с полями: `line`, `column` (для указания строки с замечанием и позиции в строке) и `message` — текст замечания (работа с массивами структур описана в разд. "Массивы структур и массивы ячеек" главы 8).
- **profile** — управление профайлером (основы работы с профайлером описаны в разд. "Профайлер" главы 15).

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 М- и МЕХ-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 столбцовой 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.
- **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` (работа со структурами описана в разд. "Простые структуры" главы 8).
- **which** — вывод пути к функции, например
- ```
>> which fzero
C:\MATLAB7\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` (см. разд. "Массивы структур и массивы ячеек" главы 8).

## Управление переменными

- **clear** — удаление переменных рабочей среды (см. разд. "Просмотр и удаление переменных, выбор имен переменных" главы 1).
- `clear A m R` — удаление переменных `A`, `m` и `R` из рабочей среды.
  - `clear global` — удаление всех глобальных переменных.

- `clear functions` — удаление всех откомпилированных М-функций.
  - `clear mex` — удаление ссылок на МЕХ-файлы и М-функции.
  - `clear all` — удаление переменных и ссылок на МЕХ-файлы и М-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 и восстановление рабочей среды" главы 1 и "Считывание и запись данных" главы 2).
- `mlock` — предотвращение удаления переменных работающего в данный момент М-файла командой `clear`.
- `munlock` — разрешение удаления переменных работающего в данный момент М-файла командой `clear`.
- `openvar` — запуск редактора М-файлов со специальным окном, позволяющим интерактивное редактирование значений переменных.  
`openvar A` — редактирование переменной `A` (см. разд. "Организация вывода текстовых результатов" главы 2).
- `pack` — создание виртуальной памяти на диске в файле `pack.tmp`, в которую заносятся значения всех имеющихся переменных, и очистка рабочей среды. Значения используемых переменныхчитываются из файла `pack.tmp`. Используется при обработке большого объема данных (см. разд. "Экономия памяти" главы 23).
- `save` — запись значений переменных на диск (см. разд. "Сохранение и восстановление рабочей среды" главы 1 и "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')` — М-файл MATLAB.
  - `saveas(h, 'filename.pcx')`
  - `saveas(h, 'filename.tif')`
- `size` — определение размеров массива (см., например, разд. "Ввод, сложение и вычитание векторов" главы 2).
- `who` — вывод имен всех определенных в рабочей среде переменных.
- `whos` — получение информации о переменных рабочей среды (см. разд. "Просмотр и удаление переменных, выбор имен переменных" главы 1).
- `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 (см. разд. "Удаление и очистка объектов" главы 9).
- `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 и восстановление 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 (работа с массивами структур описана в разд. "Массивы структур и массивы ячеек" главы 8).
- 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, использование:  
`[path, name, ext] = fileparts('имя файла')`
- fullfile** — конструирование полного имени файла из иерархии подкаталогов и имени файла, результат заносится в строковую переменную, использование:  
`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 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 М-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 М-файлов и открытие в нем `myfun.m`.
- `open('tymod.mdl')` — открытие модели `tymod.mdl` в среде Simulink.

Пользователь имеет возможность определить способ открытия файлов с другими расширениями, создав подходящую файл-функцию. Например, файлы с расширениями `tut` требуют наличия функции `openmy1`.

□ `pwd` — отображение имени текущего каталога.

`s = pwd` — запись в строковую переменную `s` имени текущего каталога.

□ `tempdir` — отображение имени каталога, предназначенного для хранения временных файлов.

`s = tempdir` — запись в строковую переменную имени каталога, предназначенного для хранения временных файлов.

□ `tempname` — отображение имени текущего временного файла.

`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 operandов может быть скаляром                                                                                                                                                                                                                                |
| $*$               | Перемножение скаляров или матриц подходящих размеров. Один из operandов может быть скаляром                                                                                                                                                                                                                                                                                       |
| <code>kron</code> | Кронекеровское или тензорное произведение матриц<br>$C = \text{kron}(A, B)$                                                                                                                                                                                                                                                                                                       |
| $/$               | Деление скаляров. Поэлементное деление матрицы на скаляр. Если оба операнда — матрицы, то $A/B = A * \text{inv}(B)$                                                                                                                                                                                                                                                               |
| $^$               | Возведение скаляра в степень. Вычисление степени квадратной матрицы                                                                                                                                                                                                                                                                                                               |
| $\backslash$      | Левое матричное деление. Если $A$ является квадратной матрицей, то $A \backslash B = \text{inv}(A) * B$ . Если $A$ — квадратная матрица размера $n$ , $B$ — вектор-столбец из $n$ элементов, то $X = A \backslash B$ содержит решение системы линейных уравнений $AX = B$ . Допускаются переопределенные и недоопределенные системы (см. разд. "Задачи линейной алгебры" главы 6) |
| $.*$              | 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 $C(i, j) = A(i, j) / B(i, j)$                                                                                                                                                                                                                                                                |
| $. \backslash$    | Поэлементное левое деление массивов одинаковых размеров, например, $C = A . \backslash B$ приводит к $C(i, j) = B(i, j) / A(i, j)$                                                                                                                                                                                                                                                |
| $. ^$             | Поэлементное возведение матрицы в степень, 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 ' $.$ ' 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**

| Оператор     | Назначение                   |
|--------------|------------------------------|
| >            | Больше: $a > b$              |
| $\geq$       | Больше или равно: $a \geq b$ |
| <            | Меньше: $a < b$              |
| $\leq$       | Меньше или равно: $a \leq b$ |
| $\equiv$     | Равно: $a \equiv b$          |
| $\sim\equiv$ | Не равно: $a \sim\equiv b$   |

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 И: $k \leq \text{length}(x) \&\& x(k) > 0$ , если первое условие $k \leq \text{length}(x)$ не выполняется, то второе $x(k) > 0$ не проверяется |
|              | Логическое ИЛИ: $a > b   a < c$                                                                                                                           |
| $\ $         | Логическое ИЛИ: $a > b    a < c$ , если первое условие $a > b$ выполняется, то второе $a < c$ не проверяется                                              |
| $\text{xor}$ | Логическое исключающее ИЛИ: $\text{xor}(a > b, a == c)$                                                                                                   |
| $\sim$       | Логическое отрицание: $\sim(a == 0)$                                                                                                                      |

## 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" этого приложения).

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` бит. Если выходной аргумент представляется числом бит, 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:<br><br><code>a = [1 2 3]; b = [1 + 2i, 3 - 9i]; c = [0.2; -3; -4; 8];</code><br><br><code>A = [1 2 3; 4 5 6; 7 8 9];</code> |

Таблица П4 (окончание)

| Символы | Назначение                                                                                                                                                                                                                                                                                                                                     |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | Конструирование блочных матриц так же производится при помощи квадратных скобок:<br>$M = [A \ B; \ C \ D];$<br>Пустые квадратные скобки используются для определения пустого массива и удаления строк или столбцов: $A(2, \ :) = [];$<br>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                                                                                                                                                                       |
| :       | 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 <code>cd</code>                                                                                                                                                                                                                                                                                         |
| ...     | Продолжение команды на следующей строке. Используется как при наборе в командной строке, так и в редакторе M-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 строки комментариев в M-файле                                                                                                                                                                                                                                                                                                           |
| % {     | Начало блока комментариев в M-файле                                                                                                                                                                                                                                                                                                            |
| % }     | Конец блока комментариев в M-файле                                                                                                                                                                                                                                                                                                             |

## Логические функции

□ **all** — проверка на наличие нулевого элемента в массиве.

$f = \text{all}(A)$  — 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.

$f = \text{any}(A)$  — 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.

$a = \text{exist}('name')$  — возвращает тип проверяемого объекта:

- 0, если name не существует;
- 1, если name является переменной рабочей среды;
- 2, если name — имя M-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, имеющемсь в путях поиска;
- 7, если name является именем каталога.

□ **find** — нахождение индексов и значений ненулевых элементов массива.

- $k = \text{find}(x)$  — в вектор k заносятся номера ненулевых элементов массива x. Если x является матрицей, то она трактуется как вектор, составленный из ее столбцов.
- $[i, j] = \text{find}(x)$  — в векторы i и j записываются индексы ненулевых элементов матрицы x, что удобно, например, при работе с разрезанными матрицами (см. разд. "Логическое индексирование" главы 2, "Логические операции с числами и массивами" главы 7 и "Работа с разрезанными матрицами" главы 15).
- $[i, j, v] = \text{find}(x)$  — 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 (работа с массивами ячеек описана в разд. "Массивы структур и массивы ячеек" главы 8).
- `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 (т. е. 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 (работа со структурами описана в разд. "Простые структуры" главы 8).
- `TF = isnfinite(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 (дескрипторной графике и указателям посвящена глава 9).
- `k = ishold` — возвращает логическую единицу, если `hold` установлено в `on`, т. е. при выводе графиков в текущие оси 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 (т. е. `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 (работа с разреженными матрицами описана в главе 15).
- `k = isstruct(S)` — возвращает логическую единицу, если `S` является структурой, и ноль — в противном случае (работа со структурами описана в разд. "Простые структуры" главы 8).

□ **`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 2).
- **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 встроенным 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.
- **continue** — переход к следующему шагу цикла `for` или `while`.
- **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 перманентных переменных 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 ряда вычислительных функций. Некоторые функ-

ции по умолчанию находят результат с меньшей точностью (см., например, разд. "Решение произвольных уравнений" главы 6).

- **eval** — выполнение содержимого строки или строковой переменной, как команды MATLAB (см. разд. "Формирование и исполнение команд, функция eval" главы 8), например:

```
>> eval('y = sin(x).*cos(x)')
```

или

```
>> 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*).

- **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 (см. разд. "Функции от функций" главы 8), например:

```
[m, k] = feval('max', x).
```

- **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 (см. разд. "Условный оператор *if*" главы 7).

- `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.

□ **nargout** — определение количества выходных аргументов, с которыми вызвана файл-функция (см. разд. "Проверка выходных аргументов" главы 7).

- `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.

□ **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. Из массива ячеек `varargin` 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. В массив ячеек `varargout` 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).

- `r = input('Введите значение r')` — ожидание ввода пользователем значения `r` с клавиатуры и занесение введенного значения в `r`.

- `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:

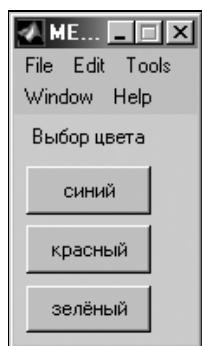
Выберите цвет:

- 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 М-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.
- **inferioro, superioro** — 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 с размером шесть на семь;
  - `C = calendar(y, m)` — занесение дат месяца с номером `m` года `y` в матрицу с размером 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. Примеры 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'.

**datem** — преобразование строки с датой в серийную дату (см. функцию `now`).

**datestr** — преобразование серийной даты (см. функцию `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 (см. разд. "Текстовые файлы" главы 8).

- `status = fclose(fid)` — закрытие файла с идентификатором `fid`. Если файл закрыт успешно, то `status=0`, если нет, то `status=-1`.
- `status = fclose('all')` — закрытие всех файлов, кроме файлов с идентификаторами 0,1 и 2.

□ **fopen** — открытие файла и получение информации о файлах (см. разд. "Текстовые файлы" главы 8).

- `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 (см. разд. "Открытие файла, считывание данных и закрытие файла" главы 8).

`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 (см. разд. "Запись в текстовый файл" главы 8).
- **fscanf** — чтение данных из текстового файла, записанных в определенном формате (см. разд. "Считывание информации из текстового файла" главы 8).
- **feof** — проверка достижения конца файла (см. разд. "Открытие файла, считывание данных и закрытие файла" главы 8).  
`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` байт 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  
`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 MS Excel: `rng = 'A5..D4'`.

### Примечание

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 `dlm` 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.

## Примечание

Нулевые значения элементов матрицы пропускаются при записи в текстовый файл, соответствующие разделители добавляются для сохранения табличной структуры данных.

- **textread** — импорт данных из текстового файла, имеющего табличную структуру.

## Примечание

Для импорта данных из различных форматов можно также выбрать в меню **File** рабочей среды пункт **Import Data** и в появившемся диалоговом окне **Import Wizard** выбрать способ импортирования.

[*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 переменных определяются 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;
- %pi — чтение *n* цифр целого числа;
- %f — чтение вещественного числа, соответствующий выходной аргумент является вещественным массивом;
- %nf — чтение *n* цифр вещественного числа;
- %p.*r*f — чтение *n* цифр вещественного числа, из них *r* после десятичной точки;
- %s — чтение строки, отделенной пробелами, символами табуляции или перевода строки (размечивающими 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;
- %pq — 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 квадратные скобки, 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
```

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

Работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8.

□ **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 в соответствующие элементы `A`;
  - ◊ `'prodofsize'` — значения элементов массива `A` равны количеству элементов в ячейках массива `C`.

Пример использования `cellfun`:

```
>> C = {ones(3, 5, 2) 'fsfh'; ...
 rand(10) strvcat('alsd', 'hhh')}

C =
 [3x5x2 double] 'fsfh'
 [10x10 double] [2x4 char]
```

```
>> A = cellfun('prodofsize', C)
```

```
A =
```

|     |   |
|-----|---|
| 30  | 7 |
| 100 | 8 |

- `A = cellfun('size', C, k)` — в массиве A возвращаются размеры содержащегося 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, содержащимо 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 (работа с массивами символов описана в разд. "Массивы строк" главы 8).

`C = cellstr(strmas)` — ячейки массива C образуются из строк массива символов strmas, пример:

```
>> strmas = ['aaaaaa'; 'bbbbbb'; 'ccccc'];
```

```
>> C = cellstr(strmas)
```

```
C =
```

```
'aaaaaa'
'bbbbbb'
'ccccc'
```

- **cell2struct** — преобразование массива ячеек в массив структур (работа с массивами структур описана в разд. "Массивы структур и массивы ячеек" главы 8).

`struct = cell2struct(c, fields, dim)` — содержимое ячеек массива с (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 с 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 с 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

Работа со структурами и массивами структур описана в разд. "Простые структуры" и "Массивы структур и массивы ячеек" главы 8.

□ **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` (работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8).

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 (работа с массивами символов описана в разд. "Массивы строк" главы 8, а с массивами ячеек в разд. "Массивы ячеек" главы 8).

□ **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. 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 (работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8).

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(C) = [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.

$Mu = \text{lin2mu}(y)$ — логарифмическое кодирование сигнала с амплитудой от -1 до 1 , записанного в вектор y . Элементами вектора Mu являются целые числа из диапазона $[0, 255]$.

□ **mu2lin** — обратное по отношению к **lin2mu** преобразование.

$y = \text{mu2lin}(Mu)$ — входной аргумент (вектор с целыми числами от 0 до 255) преобразуется в вектор y , элементы которого принадлежат интервалу $[-32124/32768, 32124/32768]$.

□ **sound** — воспроизведение звука.

- $\text{sound}(y, fs)$ — 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 $\text{size}(y) = [n \ 2]$ отвечает стереофоническому звуку.
- $\text{sound}(y)$ — воспроизведение сигнала с частотой дискретизации, равной по умолчанию 8192 Гц.
- $\text{sound}(y, fs, bits)$ — воспроизведение сигнала с частотой дискретизации fs и разрядностью $bits$.

□ **soundsc** — воспроизведение звука с предварительным масштабированием.

- $\text{sound}(y, fs)$, $\text{sound}(y)$, $\text{sound}(y, bits)$ — работают аналогично **sound**, но амплитуда сигнала предварительно масштабируется и приводится к отрезку $[-1, 1]$ так, чтобы звук был максимальной громкости без усечения амплитуды (т. е. без потери качества).
- $\text{soundsc}(y, \dots, slim)$ — проигрывание звукового сигнала с предварительным масштабированием к $[-1, 1]$ тех значений y , которые принадлежат отрезку, заданному вектором $slim = [\text{slow} \ \text{shigh}]$.

□ **wavread** — считывание звука из WAV-файла.

- $y = \text{wavread}(\text{filename})$ — в массив y заносятся амплитуды отсчетов (масштабированные к $[-1, 1]$) дискретизированного звука из WAV-файла с именем `filename`. Вектор y соответствует монофоническому звуку, а

матрица с двумя столбцами — стереофоническому, первому каналу отвечает $y(:, 1)$, второму — $y(:, 2)$.

- $[y, fs, nb]$ = wavread(filename) — в дополнительных выходных параметрах содержится информация о дискретизованном звуке (см. функцию 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, nb, 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, nb, 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 значения 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 (работа со структурами описана в разд. "Простые структуры" главы 8).

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)` (работа с массивами структур описана в разд. "Массивы структур" главы 8).

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 пикселях;
- `Height` — высота изображения в пикселях;
- `BitDepth` — глубина цвета (бит/пикセル);
- `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` формата `fmt` (см. функцию `imfinfo`). Размерность массива `A` зависит от типа изображения. Если файл содержит изображение в оттенках серого (функция `imfinfo` 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 `imfinfo` 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. функцию `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. функцию `imread`).
- `imwrite(A, MAP, filename, fmt)` — запись индексированных графических данных, содержащихся в матрицах A и MAP в файл с именем `filename` типа `fmt` (см. функцию `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 большие 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]`.

Операции со строками

См. разд. "Работа со строками" главы 8.

Обработка строк

- **deblank** — удаление пробелов в конце строки (см. разд. "Массивы строк" главы 8).
 - `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` (работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8).
- **findstr** — поиск подстроки в строке (см. разд. "Сервисные функции для работы со строками" главы 8).

`k = findstr(s1, s2)` — выходной аргумент `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 (см. разд. "Сервисные функции для работы со строками" главы 8).

`snew = lower(s)` — преобразование символов строки `s` в строчные буквы. Допускается применение функции `lower` к массиву ячеек, состоящих из строк (см. функцию `deblank`).
- **strcat** — сцепление строк (см. разд. "Ввод и сцепление строк" главы 8).

`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 (работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8).

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 (см. разд. "Сервисные функции для работы со строками" главы 8).

`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 (работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8).

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 (см. разд. "Массивы строк" главы 8).

- k = strmatch(str, MAS) — поиск в массиве символов или массиве ячеек из строк MAS строки, начинающейся с str. Входной 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 =
    [ ]
```

□ **strcmp** — сравнение первых n символов двух строк (см. разд. "Сервисные функции для работы со строками" главы 8).

flag = strcmp(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 = strcmp({'March', 'April', 'May'}, ...
    {'May', 'May', 'May'}, 2)
flag =
    1     0     1
```

□ **strrep** — замена в строке одной подстроки на другую (см. разд. "Сервисные функции для работы со строками" главы 8).

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##'
```

или

```
>> strnew = strrep({'March', 'April', 'May'}, {'Ma'}, {'##'})
strnew =
```

```
'##rch'      'April'      '##y'
```

или

```
>> 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. функцию deblank).

Преобразования "строка-число"

□ **char** — получение символа по ASCII-коду и 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: ;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ [ \]^_` 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 =
абвгдежзийклмнопрстуфхцчшъыъэую
```

Входным аргументом может быть массив ячеек из строк mas, в этом случае функция `char` образует из каждой строки ячейки mas строку символьного массива chmas (работа с массивами ячеек описана в разд. "Массивы ячеек" главе 8).

- `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 (см. разд. "Массивы строк" главы 8).

Пример:

```
>> chmas = char('AAAAAAAAAAAAA', ' ', 'BBBBBBBBBB')
chmas =
AAAAAAAAAAAAA
BBBBBBBBBB
```

Входные аргументы могут быть массивами символов:

```
>> chmas1 = char('AAA', 'BB');
>> chmas2 = char('CCC', 'DDDDDD');
>> chmas = char(chmas1 ,chmas2)
chmas =
AAA
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 (см. разд. "Простой пример, программа-калькулятор" главы 8).

- `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` (подробная информация о форматной записи в файл с примерами использования содержится в разд. "Текстовые файлы" главы 8).

Пример использования `sprintf`:

```
>> A = [1.1 3.2; 0.7 -4.2];
>> str = sprintf('a=%8.1d b=%8.1d\n c=%8.1d d=%8.1d', A)
str =
a = 1.1e+000 b = 7.0e-001
c = 3.2e+000 d = -4.2e+000
```

- `[str, errmsg] = sprintf(format, A)` — если при форматной записи произошла ошибка, то выходной аргумент `errmsg` содержит соответствующее сообщение.

□ **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 числовый массив.

- `a = str2double(str)` — из строки `str` извлекается число и заносится в переменную `a`. Стока `str` может содержать цифры, точку, знаки плюс или минус, символ `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` (работа с массивами ячеек описана в разд. "Массивы ячеек" главы 8).

Пример:

```
>> A = str2double({'-7' '3*i' 'FFF' '3.19'})
A =
    -7.0000          0 + 3.0000i        NaN            3.1900
```

□ **str2num** — преобразование массива символов в массив чисел (см. разд. "Простой пример, программа-калькулятор" главы 8).

`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 15).

Создание матриц и массивов

□ **blkdiag** — конструирование блочно-диагональных матриц.

`M = blkdiag(A, B, C)` — занесение в `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, $H = \text{invhilb}(n)$.
- **linspace** — генерация вектора, значения элементов которого изменяются с постоянным шагом (см., например, разд. "Решение граничных задач" главы 6).
 - $v = \text{linspace}(a, b)$ — в вектор v заносится 100 значений от a до b .
 - $v = \text{linspace}(a, b, n)$ — в вектор v заносится n значений от a до b .
- **logspace** — генерация вектора, значения элементов которого изменяются с постоянным шагом в логарифмической метрике (см., например, разд. "Перманентные переменные" главы 8).
 - $v = \text{logspace}(a, b)$ — в вектор v заносится 50 значений от 10^a до 10^b .
 - $v = \text{logspace}(a, b, n)$ — в вектор v заносится n значений от 10^a до 10^b .
- **magic** — создание "магического квадрата".

$M = \text{magic}(n)$ — 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.
 - $A = \text{ones}(n)$ — A содержит квадратную матрицу из единиц размера n .
 - $A = \text{ones}(m, n)$ — A содержит прямоугольную матрицу размера m на n , состоящую из единиц.
 - $A = \text{ones}(m, n, k)$ — 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, $P = \text{pascal}(n)$. 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.
 - $T = \text{toeplitz}(c, r)$ — создание несимметричной теплицевой матрицы при помощи вектор-столбца c и вектор-строки r .
 - $T = \text{toeplitz}(r)$ — 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 в матрицы, то возможны следующие варианты вызова `cat`:
 - $M = \text{cat}(1, A, B)$ — сцепление A и B вдоль первого измерения (массивы A и B расположены в столбик);
 - $M = \text{cat}(2, A, B)$ — сцепление A и B вдоль второго измерения (массивы A и B расположены в строку);
 - $M = \text{cat}(3, A, B)$ — образование трехмерного массива.
- **`diag`** — выделение диагонали и конструирование диагональной матрицы.
 - $A = \text{diag}(a)$ — создание диагональной матрицы A , на диагонали которой стоят элементы вектора a .
 - $A = \text{diag}(a, k)$ — создание диагональной матрицы A , на побочной k -ой диагонали которой стоят элементы вектора a .
 - $a = \text{diag}(A)$ — 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.
 - $M = \text{repmat}(A, m, n)$ — матрица M состоит из m блоков по вертикали и n по горизонтали, каждый блок является матрицей A .
 - $M = \text{repmat}(A, [m n])$ — аналогично $M = \text{repmat}(A, m, n)$.
 - $M = \text{repmat}(a, [m n p \dots])$ — конструирование многомерного блочного массива.
- **`reshape`** — изменение формы матрицы или массива.
 - $A = \text{reshape}(x, m, n)$ — формирование матрицы m на n из элементов массива x длины $m * n$. Элементы x выбираются последовательно, обра- зуя столбцы A .

- $A = \text{reshape}(x, m, n, p)$ — формирование трехмерного массива m на n на p из элементов массива x длины $m \times n \times p$. Аналогичным образом создаются многомерные массивы.
- **`rot90`** — поворот матрицы (см. разд. "Применение функций обработки данных к матрицам" главы 2).
- $B = \text{rot90}(A)$ — B образуется из A поворотом против часовой стрелки на 90° .
 - $B = \text{rot90}(A, k)$ — B образуется из A поворотом против часовой стрелки на 90° k раз.
- **`tril`** — выделение нижнего треугольника из матрицы (см. разд. "Создание матриц специального вида" главы 2).
- $L = \text{tril}(A)$ — B матрицу L тех же размеров, что и A , заносятся элементы нижнего треугольника A с диагональю.
 - $L = \text{tril}(A, k)$ — 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`).
- **`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 $w'' - zw = 0$.
- $w = \text{airy}(z)$ — функция Эйри первого порядка.
 - $w = \text{airy}(1, z)$ — производная функции Эйри первого порядка.
 - $w = \text{airy}(2, z)$ — функция Эйри второго порядка.
 - $w = \text{airy}(3, z)$ — 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

$$\text{besselh}(\nu, 1, z) * \exp(-i * z).$$
- `f = besselh(nu, 2, z, 1)` — то же самое, что

$$\text{besselh}(\nu, 2, z) * \exp(i * z).$$

□ **besselj, bessely** — функции Бесселя, являющиеся решениями дифференциального уравнения $z^2 y'' + zy' + (z^2 - \nu^2)y = 0$ для вещественных ν .

- `f = besselj(nu, z)` — функция Бесселя первого рода.
- `f = besselj(nu, z, 1)` — то же самое, что

$$\text{besselj}(\nu, z) * \exp(-\text{abs}(\text{imag}(z))).$$
- `f = bessely(nu, z)` — функция Бесселя второго рода.
- `f = bessely(nu, z, 1)` — то же самое, что

$$\text{bessely}(\nu, z) * \exp(-\text{abs}(\text{imag}(z))).$$

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` получает-

ся, если один из аргументов z или ν — вектор-строка, а второй — вектор-столбец.

□ **besseli, besselk** — модифицированные функции Бесселя, являющиеся решениями дифференциального уравнения $z^2 y'' + zy' + (z^2 + \nu^2)y = 0$ для вещественных ν .

- $f = \text{besseli}(\nu, z)$ — модифицированная функция Бесселя первого рода.
- $f = \text{besseli}(\nu, z, 1)$ — то же самое, что

$$\text{besseli}(\nu, z) * \exp(-\text{abs}(\text{real}(z))).$$
- $f = \text{besselk}(\nu, z)$ — модифицированная функция Бесселя второго рода.
- $f = \text{besselk}(\nu, z, 1)$ — то же самое, что

$$\text{besselk}(\nu, z) * \exp(-\text{abs}(\text{real}(z))).$$

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:

$$B(z, w) = \int_0^1 t^{z-1} (1-t)^{w-1} dt; \quad B_x(z, w) = \frac{1}{B(z, w)} \int_0^x t^{z-1} (1-t)^{w-1} dt.$$

- $f = \text{beta}(z, w)$ — вычисление бета-функции.
- $f = \text{beta}(x, z, w)$ — вычисление неполной бета-функции, x должен принадлежать отрезку $[0, 1]$.
- $f = \text{betaln}(z, w)$ — вычисление натурального логарифма от бета-функции с использованием более эффективного алгоритма, чем $\log(\text{beta}(z, w))$.

Аргументы w и z 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

$$u = \int_0^\phi \frac{d\Phi}{\sqrt{1 - m \sin^2 \Phi}}.$$

- `[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:

$$K(m) = \int_0^{\pi/2} \frac{d\phi}{\sqrt{1 - m \sin^2 \phi}}; \quad E(m) = \int_0^{\pi/2} \sqrt{1 - m \sin^2 \phi} d\phi.$$

- `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:

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt; \quad \operatorname{erfc} x = 1 - \operatorname{erf} x; \quad \operatorname{erfcx} x = e^{x^2} \operatorname{erfc} x.$$

- `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.

$$\Gamma(\alpha) = \int_0^{\infty} e^{-t} t^{\alpha-1} dt; \quad \Gamma_x(\alpha) = \frac{1}{\Gamma(\alpha)} \int_0^x e^{-t} t^{\alpha-1} dt.$$

`y = gamma(a), y = gammainc(x, a), y = gammaln(a).`

□ **legendre** — присоединенные функции Лежандра первого рода $P_n^m(x)$ и полуно нормированные присоединенные функции Лежандра $S_n^m(x)$, определяемые формулами

$$P_n^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_n(x); \quad S_n^m(x) = (-1)^m \sqrt{\frac{2(n-m)!}{(n+m)!}} P_n^m(x),$$

где $P_n(x)$ — полиномы Лежандра.

- `p = legendre(n, x)` — вычисление присоединенных функций Лежандра первого рода для всех $m = 0, 1, \dots, n$. 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 $m = 0, 1, \dots, n$, вычисленные для каждого элемента вектора `x`.
- `p = legendre(n, x, 'sch')` — вычисление полуно нормированных присоединенных функций Лежандра первого рода для всех $m = 0, 1, \dots, n$. Интерфейс аналогичен `legendre(n, x)`.

□ **pow2** — нахождение степеней двойки.

- $x = \text{pow2}(y)$ — в векторе x возвращается результат возведения числа 2 в степень, записанную в соответствующем элементе y .
- $x = \text{pow2}(f, e)$ — элементы вектора x вычисляются по формуле $x(i) = f(i) * 2^e(i)$.

□ **rat, rats** — приближение вещественных чисел отношением двух целых чисел (рациональной дробью).

- $[n, d] = \text{rat}(x, tol)$ — возвращает два целых числа n и d таких, что n/d приблизительно равно x (с точностью tol) в следующем смысле: $\text{abs}(n/d - x) \leq tol * \text{abs}(x)$. Если x — массив, то n и d являются массивами того же размера, содержащими соответствующие значения числителя и знаменателя для каждого элемента x .
- $[n, d] = \text{rat}(x)$ — использует по умолчанию точность $tol = 1.e-6 * \text{norm}(X(:), 1)$.
- $s = \text{rat}(x), s = \text{rat}(x, tol)$ — возвращают ответ в строковой переменной s .
- $s = \text{rats}(x, len)$ — использует rat для приближенного представления числа x рациональной дробью со строкой длиной len .

Преобразование координат

□ **cart2pol** — переход от декартовых координат к полярным или цилиндрическим.

- $[th, r] = \text{cart2pol}(x, y)$ — переход от декартовых координат к полярным по формулам $th = \text{atan2}(y, x), r = \sqrt{x.^2 + y.^2}$.
- $[th, r, z] = \text{cart2pol}(x, y, z)$ — переход от декартовых координат к цилиндрическим по формулам $th = \text{atan2}(y, x), r = \sqrt{x.^2 + y.^2}, z = z$.

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.

$[th, phi, r] = \text{cart2sph}(x, y, z)$ — переход от декартовых координат к сферическим по формулам:

- $th = \text{atan2}(y, x);$

- `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. функцию `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`.

Решение различных математических задач

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 15).

Матричный анализ

□ **`cond`** — нахождение числа обусловленности по отношению к различным матричным нормам (см. разд. "Системы с плохо обусловленными матрицами" главы 6, а также функцию `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)` (см. разд. "Системы с плохо обусловленными матрицами" главы 6).
 - **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 Гельдера 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 столбцовой норме.

□ **rref, rrefmovie** — нахождение приведенно-ступенчатой формы матрицы исключением по Гауссу—Жордану с выбором главного элемента.

- `R = rref(A)` — в R содержится приведено-ступенчатая форма A, при вычислениях элементы, меньшие `max(size(A)) * eps * norm(A, inf)`, полагаются равными нулю.

- `[R, jb] = rref(A)` — вектор jb содержит номера связанных компонент 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 $B = \text{inv}(D) * A * D$ суммы модулей элементов были примерно одинаковы. Балансировка не изменяет спектр матрицы. Использование:

$B = \text{balance}(A)$, $[D, B] = \text{balance}(A)$ — элементы D являются целыми степенями двойки. Если A есть симметричная матрица, то балансировки не происходит и $B = A$, $D = \text{eye}(\text{size}(A))$.

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 (см. разд. "Собственные числа и векторы матрицы, функции матриц" главы 6).

- $d = \text{eig}(A)$ — в векторе d возвращаются собственные значения матрицы A , т. е. $A^*u = d(i)*u$. Исходная матрица предварительно балансируется.
- $[V, D] = \text{eig}(A)$ — столбцы матрицы V являются собственными векторами A , D есть диагональная матрица, состоящая из собственных значений, $A^*U = D^*U$. Исходная матрица предварительно балансируется.
- $[V, D] = \text{eig}(A, 'nobalance')$ — то же, что и $[V, D] = \text{eig}(A)$, но без предварительной балансировки.
- $d = \text{eig}(A, B)$ — в векторе d возвращаются обобщенные собственные значения, являющиеся решением $A^*u = d(i)^*B^*u$, используется QZ-алгоритм.
- $[V, D] = \text{eig}(A, B)$ — дополнительно возвращаются обобщенные собственные векторы.

□ **gsvd** — обобщенное сингулярное разложение (см. также функцию **svd**).

- $[U, V, X, C, S] = \text{gsvd}(A, B)$ — 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 $\text{size}(A) = [m p]$,

`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 $m \geq p$ или $n \geq p$, 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 обобщенных 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, разложение 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 15).

□ **inv** — обращение матрицы (см. разд. "Обращение матриц" главы 6).

$B = \text{inv}(A)$ — возвращает матрицу, обратную к квадратной матрице A . В случае плохой обусловленности A выдается предупреждение.

Задание в качестве входного аргумента символьной матрицы приводит к поиску обратной к ней также в символьной форме (см. разд. "Задачи линейной алгебры" главы 17).

□ **lu** — LU-разложение квадратной матрицы.

- $[L, U] = \text{lu}(A)$ — возвращает верхнюю треугольную матрицу U и матрицу L , которая может быть сведена к нижней треугольной перестановками.
- $[L, U, P] = \text{lu}(A)$ — дополнительно возвращает матрицу перестановок P такую, что $P^*A = L^*U$.
- $[L, U] = \text{lu}(A, \text{tresh})$ — 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.

Функция **lu** применима к разреженным матрицам (см. разд. "Факторизация матриц" главы 15).

□ **lsqnonneg** — нахождение положительного решения системы линейных алгебраических уравнений (не обязательно с квадратной матрицей) методом наименьших квадратов (см. разд. "Метод наименьших квадратов" главы 16).

- $X = \text{lsqnonneg}(C, d)$ — возвращает вектор X с неотрицательными компонентами, которые минимизируют $\text{norm}(C^*X - d)$.
- $X = \text{lsqnonneg}(C, d, X0)$ — вектор $X > 0$ используется в качестве начального приближения.
- $X = \text{lsqnonneg}(C, d, X0, \text{options})$ — процесс вычислений управляется при помощи задания параметров в структуре **options** функцией **optimset** (см. разд. "Параметры оптимизации" главы 16).
- $[X, \text{resnorm}] = \text{lsqnonneg}(\dots)$ — в выходном аргументе **resnorm** возвращается квадрат нормы невязки $\text{norm}(C^*X - d)^2$.
- $[X, \text{resnorm}, \text{residual}] = \text{lsqnonneg}(\dots)$ — в выходном аргументе **residual** возвращается невязка $C^*X - d$.
- $[X, \text{resnorm}, \text{residual}, \text{exitflag}] = \text{lsqnonneg}(\dots)$ — значение единицы выходного аргумента **exitflag** свидетельствует об успешном на-

хождении 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`, $\text{lambda}(i) \leq 0$, если $X(i)$ приближенно равно нулю и $\text{lambda}(i) = 0$, 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 $\text{norm}(b - A*b) / \text{NORM}(b) < 1e-6$. 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 предобусловителя 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 (см. разд. "Обращение матриц" главы 6).

- `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 $A = Q*R$.

- $[Q, R, E] = \text{qr}(A)$ — дополнительно возвращает матрицу перестановок E , $A^*E = Q^*R$.
- $[Q, R] = \text{QR}(A, 0)$ — если $\text{size}(A) = [m n]$ и $m > n$, то возвращаются только первые n столбцов Q .
- $[Q, R, E] = \text{QR}(A, 0)$ — то же, что и $[Q, R] = \text{QR}(A, 0)$, но возвращается матрица перестановок E такая, что $Q^*R = A(:, E)$.

Вычисление функций от матриц

См. разд. "Вычисление математических функций от элементов матриц" главы 2.

- **expm** — матричная экспонента, использование: $F = \text{expm}(A)$.
- **funm** — вычисление произвольной функции от матрицы.
 - $F = \text{funm}(A, 'funname')$ — 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.
 - $[F, errest] = \text{funm}(A, 'funname')$ — выходной аргумент *errest* содержит грубую оценку результата.
- **logm** — матричный логарифм.
 - $L = \text{logm}(A)$ — 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. функцию *funm*).
 - $[L, esterr] = \text{logm}(A)$ — выходной аргумент *errest* содержит грубую оценку результата.
- **sqrtm** — квадратный корень из матрицы.
 - $X = \text{sqrtm}(A)$ — возвращает матрицу X такую, что $X^*X = A$. Если A является вырожденной матрицей, то выдается предупреждение.
 - $[X, resnorm] = \text{sqrtm}(A)$ — возвращает в *resnorm* относительную погрешность $\text{norm}(A - X^2, 'fro')/\text{norm}(A, 'fro')$.

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 16).

- `x = fsolve(funname, x0)` — возвращает решение, используя `x0` в качестве начального приближения.
- `x = fsolve(funname, x0, options)` — процесс решения управляет 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 $f(x, p1, p2, ...)$.

Применение `fsolve` для исследования функций, зависящих от параметров, может быть организовано при помощи вложенных и анонимных функций так, как описано в разд. "Исследование функций, зависящих от параметров" главы 6. Программированию вложенных функций посвящен разд. "Вложенные функции" главы 5.

- `[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 количества 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` (см. разд. "Решение произвольных уравнений" главы 6).

- `x = fzero(fun, x0)` — возвращает решение, используя `x0` в качестве начального приближения.

- `x = fzero(fun, [a b])` — возвращает решение на промежутке $[a, b]$, используя x_0 в качестве начального приближения. Предполагается, что $f(a) * f(b) < 0$.

Следующие варианты вызова аналогичны `fsolve`:

- `x = fzero(fun, x0, options)`
- `x = fzero(fun, x0, options, p1, p2, ...)`

Применение `fzero` для исследования функций, зависящих от параметров, описано в разд. "Исследование функций, зависящих от параметров" главы 6. Программированию вложенных функций посвящен разд. "Вложенные функции" главы 5.

- `[x, fval] = fzero(fun, ...)`
- `[x, fval, exitflag, output] = fzero(...)`

Использование `exitflag` несколько отличается от случая `fsolve`.

`[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, или 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** — символьное решение уравнений и систем (см. разд. "Решение уравнений и систем" главы 17).

- `r = solve(f)` — нахождение символьного решения уравнения, заданного строкой `f`. Входным аргументом может быть символьная 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 символьные функции. По умолчанию неизвестными переменными являются те, которые возвращают `findsym`. Поля структуры `r` содержат компоненты решения.
- `[r1, r2, ..., rn] = solve(f1, f2, ..., fn)` — решение записывается в символьные переменные `r1, r2, ..., rn`.

- $r = \text{solve}(f_1, f_2, \dots, f_n, t_1, t_2, \dots, t_n)$ — дополнительные входные аргументы t_1, t_2, \dots, t_n указывают на переменные, подлежащие определению.
- $[r_1, r_2, \dots, r_n] = \text{solve}(f_1, f_2, \dots, f_n, t_1, t_2, \dots, t_n)$ — решение записывается в символьные переменные r_1, r_2, \dots, r_n .

Интерполяция и приближение данных

Примеры, связанные с интерполяцией и приближением данных, приведены в разд. "Полиномы и интерполяция" главы 6. Интерполяция сплайнами в Spline Toolbox описана в главе 18, а в главе 19 обсуждается приближение при помощи параметрических моделей средствами Curve Fitting Toolbox.

- **polyfit** — приближение табличной функции одной переменной полиномом заданного порядка по методу наименьших квадратов (см. разд. "Приближение по методу наименьших квадратов" главы 6).
- **griddata** — приближение неравномерно распределенных трехмерных данных поверхностью, построенной на регулярной сетке.
 - $ZI = \text{griddata}(x, y, z, XI, YI)$ — 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.
 - $ZI = \text{griddata}(x, y, z, XI, YI, 'linear')$ — приближение линейными функциями.
 - $ZI = \text{griddata}(x, y, z, XI, YI, 'cubic')$ — приближение кубическими функциями.
 - $ZI = \text{griddata}(x, y, z, XI, YI, 'nearest')$ — приближение по ближайшему соседу.

□ **interp1, interp2, interp3, interpn** — интерполяция одномерных, двумерных, трехмерных и многомерных данных различными способами (см. разд. "Интерполяция сплайнами" и "Интерполяция двумерных и многомерных данных" главы 6).

□ **interpft** — одномерная интерполяция с использованием быстрого преобразования Фурье.

$y = \text{interpft}(x, n)$ — предполагается, что вектор x содержит равноотстоящие друг от друга на шаг dx элементы, $\text{length}(x) = m$. Находится преобразование Фурье от x , затем оно дополняется точками с шагом

$\text{dx}^* \text{m/n}$. Вычисляется обратное преобразование Фурье и возвращается в векторе y .

- **spline** — интерполяция кубическими сплайнами.

$y_i = \text{spline}(x, y, x_i)$ — табличные данные, заданные векторами x и y , интерполируются кубическими сплайнами. Возвращаемый вектор y_i содержит значения сплайна в абсциссах, определяемых вектором x_i .

Минимизация и оптимизация

Функции MATLAB, предназначенные для решения задач минимизации и оптимизации, входят в состав Optimization Toolbox (см. главу 16, где приведены формулировки основных задач и примеры использования функций).

- **fgoalattain** — решение задачи о достижении границы (см. разд. "Задача о достижении границы" главы 16).
- **fminbnd** — нахождение локального минимума функции одной переменной на заданном интервале (см. разд. "Нахождение экстремумов функций" главы 6).
- **fmincon** — решение задач нелинейного программирования (см. разд. "Нелинейное программирование" главы 16).
- **fminimax** — решение минимаксной задачи (см. разд. "Минимаксная задача" главы 16).
- **fminsearch** — поиск локального минимума функции нескольких переменных (см. разд. "Нахождение экстремумов функций" главы 6).
- **fminunc** — поиск минимума нелинейной функции без ограничения на переменные.
- **linprog** — решение задач линейного программирования (см. разд. "Линейное программирование" главы 16).
- **lsqcurvefit** — подбор параметров (см. разд. "Подбор параметров" главы 16).
- **lsqlin** — метод наименьших квадратов для решения систем линейных уравнений с линейными ограничениями на решение (см. разд. "Метод наименьших квадратов" главы 16).
- **lsqnonlin** — нелинейный подбор параметров.
- **optimget** — получение параметров, определяющих работу функций минимизации и оптимизации (см. разд. "Управление ходом вычислений" главы 6).

- **optimset** — задание параметров, управляющих минимизацией и оптимизацией (см., например, разд. "Управление ходом вычислений" главы 6 и "Решение системы нелинейных уравнений" главы 16).
- **quadprog** — решение задач квадратичного программирования (см. разд. "Квадратичное программирование" главы 16).

Дифференцирование и конечные разности

- **del2** — вычисление разностного аналога оператора Лапласа.

Предполагается, что матрица U содержит значения некоторой функции в точках сетки.

- $L = \text{del2}(U)$ — возвращается матрица со значениями для внутренних 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: $L = \text{del2}(U, h)$, $L = \text{del2}(U, hx, hy)$.

Генерация прямоугольной сетки производится при помощи функции **meshgrid** (см. разд. "Графики функций двух переменных" главы 2).

- $L = \text{del2}(U, hx, hy, hz, \dots)$ — аппроксимация оператора Лапласа и в многомерном случае. Для построения многомерных сеток предназначена **ndgrid**.

- **diff** — нахождение конечных разностей и символьное вычисление производных (символным вычислениям посвящена глава 17).

- $D = \text{diff}(X)$ — по вектору X строит вектор $D = [X(2) - X(1), \dots, X(n) - X(n - 1)]$, где $n = \text{length}(X)$, причем $\text{length}(D) = n - 1$. Если входной аргумент является матрицей, то происходит вычисление для каждого столбца X .
- $Dk = \text{diff}(X, k)$ — вычисление конечных разностей k -го порядка. Например, вычисление $k = 3 \text{ diff}(X, 3)$ и $\text{diff}(\text{diff}(\text{diff}(X)))$ приводят к одинаковым результатам.
- $f1 = \text{diff}(f)$ — нахождение первой производной в аналитическом виде от символьной функции f . По умолчанию в качестве независимой переменной выбирается результат **findsym(f)**. Выходной аргумент является символьной функцией.

- `f1 = diff(f, t)` — нахождение первой производной в аналитическом виде от символьной функции `f` по переменной `t`. Переменная `t` должна быть объявлена как символьная при помощи `sym` 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 17).

□ **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 компонент градиента функции нескольких переменных. Для построения многомерных сеток предназначена `ndgrid`.

Интегрирование

□ **dblquad** — вычисление двойных интегралов (см. разд. "Вычисление двойных интегралов" главы 6).

- `result = dblquad('fun', inmin, inmax, outmin, outmax)`
- `result = dblquad('fun', inmin, inmax, outmin, outmax, tol)`
- `result = dblquad('fun', inmin, inmax, outmin, outmax, tol, method)`

□ **int** — нахождение определенных и неопределенных интегралов в символьном виде (см. разд. "Пределы, дифференцирование и интегрирование" главы 17).

`int(f), int(f, t), int(f, a, b), int(f, t, a, b).`

□ **quad, quadl** — вычисление определенных интегралов по квадратурным формулам Симпсона и Ньютона—Котеса с автоматическим подбором шага интегрирования (см. разд. "Вычисление определенных интегралов" главы 6, где также обсуждается применение анонимных функций для интегрирования функций, зависящих от параметров).

- `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, ...)`

Решение дифференциальных уравнений и систем

Аппроксимация решения стационарных и нестационарных задач, описываемых дифференциальными уравнениями в частных производных, по методу конечных элементов реализована в функциях PDE Toolbox (см. главу 14).

- **dsolve** — аналитическое решение дифференциальных уравнений и систем (см. разд. "Решение дифференциальных уравнений и систем" главы 17).
- **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**.
- **dde23** — решение дифференциальных уравнений с запаздывающим аргументом. Задание параметров, управляющих вычислительным процессом, производится при помощи **ddeset**.
- **ode15i** — решение дифференциальных уравнений, не разрешенных относительно производной, и дифференциально-алгебраических уравнений.
- **bvp4c** — численное решение граничных задач для обыкновенных дифференциальных уравнений произвольного порядка и систем (см. разд. "Решение дифференциальных уравнений" главы 6).

Графика и визуализация данных

Интерактивная среда для построения и редактирования графиков

- **plottools** — запуск интерактивной среды со всеми ее компонентами (см. главу 4).
- **figurepalette** — управление окном **Figure Palette** с шаблонами графиков.
 - **figurepalette('show')** — отображение окна с шаблонами графиков для текущего графического окна.
 - **figurepalette('hide')** — скрытие окна с шаблонами графиков для текущего графического окна.
 - **figurepalette('toggle')** — переключение между режимами 'show' и 'hide' для текущего графического окна.
 - **figurepalette(hF, ...)** — то же самое, но для графического окна с указателем **hF**.

□ **plotbrowser** — управление окном браузера объектов **Plot Browser**.

- `plotbrowser('on')` — отображение окна браузера объектов для текущего графического окна.
- `plotbrowser('off')` — скрытие окна браузера объектов для текущего графического окна.
- `plotbrowser('toggle')` — переключение между режимами '`on`' и '`off`' для текущего графического окна.
- `plotbrowser(hF, ...)` — то же самое, но для графического окна с указателем `hF`.
- `propertyeditor` — управление окном редактора свойств графических объектов **Property Editor**. Использование аналогично `plotbrowser`.

Двумерные графики

Использование низкоуровневой графики для задания свойств графических объектов объясняется в главах 3 и 9.

□ **bar** — вертикальная столбцевая диаграмма матричных или векторных данных (см. разд. "Диаграммы и гистограммы" главы 3), примеры:

- `bar(rand(1, 10)), bar(rand(1, 10), 1.2), bar(rand(3, 4));`
- `bar(hA, ...)` — вывод диаграммы на оси с указателем `hA` вместо текущих;
- `h = bar(...)` — возвращает вектор указателей на созданные рисованные объекты **Barseries**;
- `h = bar('v6'...)` — возвращает вектор указателей на созданные базовые полигональные объекты (**Patch**) для совместимости с предыдущими версиями MATLAB (базовые и рисованные объекты описаны в разд. "Графические объекты" главы 9).

□ **barh** — горизонтальная столбцевая диаграмма матричных или векторных данных, примеры:

- `barh(rand(1, 10)), barh(rand(1, 10), 1.2), barh(rand(3, 4));`
- `bar(hA, ...), h = barh(...)` и `h = barh('v6'...)` — аналогично функции `bar`.

□ **comet** — анимированный график плоской линии (см. разд. "Анимированные графики" главы 3).

- `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 $p * \text{length}(Y)$, по умолчанию используется $p = 0.1$.
 - `comet(hA, ...)` — вывод анимированного графика на оси с указателем `hA` вместо текущих.
- **`ezplot`** — построение графика функции одной переменной, в том числе заданной неявно и параметрически с автоматическим подбором шага по аргументу и выводом заголовка графика (см. разд. "Графическое представление функций" главы 17).
- `ezplot(f)` — построение графика функции на отрезке $[-2\pi, 2\pi]$, где f — строка с исследуемой функцией, указатель на нее, `inline`-функция, либо анонимная функция. Задание анонимной функции или указателя требует использования поэлементных операций при определении функции.
- Пример для функции $f(x) = x^2 \sin x$:
- ```
>> f = @(x)x.^2.*sin(x);
>> ezplot(f)
```
- Функция может у быть задана неявно:  $f(x, y) = 0$ , например:
- ```
>> f = inline('x.^2 + y.^2-9');
>> ezplot(f)
```
- В этом случае по умолчанию график строится в квадрате $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$.
- `ezplot(x, y)` — вывод графика параметрически заданной функции $x(t), y(t)$:
- ```
>> ezplot('t*sin(t)', 't*cos(t)')
```
- Для параметрически заданных функций по умолчанию  $t \in [0, 2\pi]$ .
- `ezplot(f, [min, max])` — построение графика явной  $y = f(x)$  и неявной  $f(x, y) = 0$  функции на отрезке  $[min, max]$ .
  - `ezplot(f, [xmin, xmax, ymin, ymax])` — построение графика неявной функции  $f(x, y)$  в квадрате  $[xmin, xmax] \times [ymin, ymax]$ .
  - `ezplot(x, y, [tmin, tmax])` — построение графика параметрически заданной функции  $x(t), y(t)$  для значений параметра из отрезка  $[tmin, tmax]$ .

- `ezplot(hA, ...)` — вывод графика на оси с указателем `hA`.
- `h = ezplot(...)` — запись в `h` указателя на линию графика. Свойства линии могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

□ **`ezpolar`** — построение кривой в полярных координатах с автоматическим подбором шага по аргументу и выводом заголовка графика.

- `ezpolar(fi)` — построение кривой  $r = \phi(\theta)$  для  $\theta \in [0, 2\pi]$ , где `fi` — строка с исследуемой функцией, указатель на нее, `inline`-функция либо анонимная функция. Задание анонимной функции или указателя требует использования поэлементных операций при определении функции.
- `ezpolar(f, [a, b])` — то же, что и `ezpolar(fi)`, но для  $\theta \in [a, b]$ .
- `ezpolar(hA, ...), h = ezpolar(...)` — аналогично `ezplot`.

□ **`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`: '`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. функцию `caxis`), а затем происходит билинейная интерполяция цвета внутри многоугольной области, например:

`fill(X, Y, C)` — построение сразу нескольких многоугольников, число многоугольников равно столбцам матриц `X` и `Y` (предполагается, что матрицы одинаковых размеров). Третий 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:

```
fill(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).

- **fplot** — построение функции одной переменной с автоматическим подбором шага по аргументу (см. разд. "Файл-функции с одним входным аргументом" главы 5 и разд. "Более подробно о fplot" главы 6).
- **hist** — гистограмма матричных или векторных данных (см. разд. "Гистограммы векторных данных" главы 3).
  - `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.
  - `[n, xout] = hist(...)` — возвращает в векторе `n` число элементов, попавших в каждый из интервалов, а в векторе `xout` — границы 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`.
  - `hist(hA, ...)` — вывод гистограммы на оси с указателем `hA` вместо текущих.
- **loglog, semilogx, semilogy** — построение графиков в логарифмическом и полулогарифмическом масштабах. Используются так же, как `plot` (см. разд. "Графики в логарифмических масштабах" главы 3).
- **pie** — отображение данных в виде круговой диаграммы (см. разд. "Диаграммы векторных данных" главы 3).

- `pie(x)` — площадь сектора круговой диаграммы, отвечающего  $x(i)$ , пропорциональна  $x(i)/\text{sum}(x)$ . Если  $\text{sum}(x) < 1$ , то получается неполная круговая диаграмма.
- `pie(x, parts)` — ненулевые компоненты вектора `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.
- `pie(hA, ...)` — вывод круговой диаграммы на оси с указателем `hA` вместо текущих.
- `h = pie(...)` — возвращает вектор `h` с указателями на графические объекты `patch` и `text`, образующие круговую диаграмму.

Свойства каждого из графических объектов могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

□ **`pie3`** — построение объемной круговой диаграммы (см. разд. "Диаграммы векторных данных" главы 3 и пример в разд. "Управление объектами, копирование, поиск, скрытые указатели" главы 9).

□ **`plot`** — визуализация функций одной переменной, векторных и матричных данных (см., например, разд. "Построение графиков функции одной переменной" главы 2 и разд. "Графики в линейном масштабе" главы 3).

- `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)
```

Вторым аргументом `plot` может быть матрица, число строк или столбцов которой совпадает с длиной вектора `x`. При этом выводится несколько графиков. Пример:

```
>> x = -10:0.1:10;
>> y = [sin(x); cos(x); sin(x).^2; cos(x).^2];
>> 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 осиях, 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 (свойства линий описаны в разд. "Свойства линий и поверхностей" главы 9), пример:
 

```
>> plot(x, y, 'Marker', 'o', 'MarkerSize', 5, 'MarkerEdgeColor', 'g', 'MarkerFaceColor', 'y')
```
- `plot(hA, ...)` — вывод графиков на оси с указателем `hA` вместо текущих.
- `h = plot(...)` — возвращает вектор указателей на созданные рисованные объекты `Lineseries`.

Свойства каждого из графических объектов могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

- `h = plot('v6', ...)` — возвращает вектор указателей на созданные базовые объекты `Line` для совместимости с предыдущими версиями MATLAB (базовые и рисованные объекты описаны в разд. "Графические объекты" главы 9).

#### □ **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. функцию `plot`).

#### □ **stem** — визуализация данных в виде черенковой диаграммы.

- `stem(x)` — отображение зависимости элементов вектора `x` от его номеров.
- `stem(x, y)` — вывод значений массива `y` с абсциссами, указанными в векторе `x`, пример:

```
>> x = (0:0.1:10)';
>> y = [sin(x) cos(x)];
>> stem(x, y)
```

Число строк массива `y` должно равняться длине вектора `x`.

- `stem(x, y, 'r*--')` — вывод с указанием цвета и стиля линий и типа маркера.
- `stem(hA, ...)` — вывод на оси с указателем `hA`.
- `h = stem(...)` — возвращает вектор указателей на созданные рисованные объекты `Stemseries` (их число равно количеству столбцов в `y`).
- `h = stem('v6', ...)` — возвращает вектор указателей на созданные базовые объекты линии (их число равно количеству столбцов в `y`). Для совместимости с предыдущими версиями.

Базовые и рисованные объекты описаны в разд. "Графические объекты" главы 9.

## Трехмерные и контурные графики

Использование низкоуровневой графики для задания свойств графических объектов объясняется в главах 3 и 9.

□ **`bar3`** — вертикальная столбцевая трехмерная диаграмма матричных и векторных данных, пример: `bar3(rand(3, 4))`.

- `bar3(hA, ...)` — вывод диаграммы на оси с указателем `hA` вместо текущих.
- `h = bar3(...)` — возвращает вектор указателей на созданные полигональные объекты.

Свойства полигональных объектов могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

□ **`bar3h`** — горизонтальная столбцевая трехмерная диаграмма матричных и векторных данных, пример: `bar3h(rand(3, 4))`.

`bar3h(hA, ...)` и `h = bar3h(...)` — аналогично `bar3`.

□ **`comet3`** — анимированный график трехмерной линии (см. разд. "Анимированные графики" главы 3).

- `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`.
- `comet3(hA, ...)` — вывод анимированного графика на оси с указателем `hA` вместо текущих.

□ **contour** — построение линий уровня функции двух переменных (см. разд. "Контурные графики" главы 3).

- `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. функцию `contourc`) и указатель на построенные линии (рисованный объект `Contourgroup`) (см. разд. "Рисованные объекты (Plot Objects)" главы 9).
- `[C, h] = contour('v6'...)` — выходными аргументами являются матрица с информацией о линиях уровня (см. функцию `contourc`) и вектор указателей на построенные линии (полигональные объекты). Для совместимости с предыдущими версиями MATLAB.

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 задаются при помощи дополнительного строкового входного аргумента (см. функцию `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  $(x_1, y_1), (x_2, y_2)$  и т. д.

Способы задания входных аргументов `contourc` совпадают с `contour`.

- **`contourf`** — залитый цветами контурный график, использование аналогично функции `contour` (см. разд. "Контурные графики" главы 3).
- **`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.
  - `cylinder(hA, ...)` — вывод цилиндрической поверхности на оси с указателем `hA` вместо текущих.
  - $[X, Y, Z] = \text{cylinder}(\dots)$  — 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)`.
- **`ezcontour`** — отображение линий уровня функции двух переменных с автоматическим подбором сетки для их нахождения. Заголовок получаемого графика содержит выражение, задающее исследуемую функцию.
  - `ezcontour(fun)` — построение линий уровня функции `fun`, где `fun` — строка с исследуемой функцией, указатель на нее, inline-функция либо анонимная функция. Задание анонимной функции или указателя требует применения поэлементных операций при определении функции. По умолчанию считается, что область определения — квадрат  $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ . (Использование inline-функций и анонимных функций обсуждается в разд. "Встраиваемые и анонимные функции" главы 6. Обращение к функции по указателю описано в разд. "Файл-функции с одним входным аргументом" главы 5.)

Пример:

```
>> fun = inline('x^2*cos(y)+y^2*sin(x)');
>> ezcontour(fun)
```

- `ezcontour(fun, [x1 x2 y1 y2])` — областью определения считается прямоугольник  $[x_1, x_2] \times [y_1, y_2]$ .
- `ezcontour(..., n)` — построение линий уровня с использованием сетки с  $n$  узлами по каждому направлению.
- `ezcontour(hA, ...)` — вывод линий уровня на оси с указателем `hA`.
- `h = ezcontour(...)` — запись в вектор `h` указателей на созданные полигональные объекты.

Свойства полигональных объектов могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

- `ezcontourf` — получение залитых цветом контурных графиков функции двух переменных. Использование аналогично `ezcontour`.
- `ezmesh` — построение каркасной модели поверхности функции двух переменных. Использование аналогично `ezcontour`; кроме того, имеется возможность построения поверхности, заданной параметрически, и отображать функцию на круговой области определения. Вызов `ezmesh` с выходным аргументом приводит к записи в него указателя на созданную поверхность (см. разд. "Графическое представление функций" главы 17).

Пример отображения функции  $z(x, y) = x^2 - y^2$  на круговой области определения. Переменные  $x$  и  $y$  принадлежат кругу, вписанному в квадрат  $[-1, 1] \times [-1, 1]$ :

```
>> z = inline('x^2-y^2');
>> ezmesh(z, [-5.3 5.3], 'circ')
```

Пример построения параметрически заданной поверхности  $x(s, t) = s \cos t$ ,  $y(s, t) = s \sin t$ ,  $z(s, t) = s/2$ :

```
>> x = inline('s*cos(t)');
>> y = inline('s*sin(t)');
>> z = inline('s/2');
>> ezmesh(x, y, z)
```

По умолчанию значения параметров  $t$  и  $s$  принадлежат  $[-2\pi, 2\pi]$ . Для задания других интервалов следует указать их в четвертом входном аргументе — векторе  $[t1 t2 s1 s2]$ :

```
>> ezmesh(x, y, z, [-0.9*pi 0.9*pi -0.7*pi 0.8*pi])
```

или, если параметры принадлежат одному интервалу, то

```
>> ezmesh(x, y, z, [-0.6*pi 0.9*pi])
```

- **ezmeshc** — построение каркасной модели поверхности функции двух переменных и линий уровня на плоскости  $xy$ . Использование аналогично `ezmesh` (см. разд. "Графическое представление функций" главы 17).
- **ezplot3** — отображение параметрически заданной линии в трехмерном пространстве (см. разд. "Графическое представление функций" главы 17).
  - `ezplot3(x, y, z)` — построение параметрически заданной кривой  $x(t)$ ,  $y(t)$ ,  $z(t)$  для  $t$  из отрезка  $[0, 2\pi]$ , где  $x$ ,  $y$ ,  $z$  — строки с исследуемыми функциями, указатели на них, `inline`-функции либо анонимные функции. Задание анонимных функций или указателей требует использования поэлементных операций при определении функций.
  - `ezplot3(x, y, z, [tmin, tmax])` — то же самое, что и `ezplot3(x, y, z)`, но для значений параметра из отрезка  $[tmin, tmax]$ .
  - `ezplot3(..., 'animate')` — анимированный график, например:  
`>> ezplot3('sin(t)', 'cos(t)', 't', [0 100], 'animate')`
  - `ezplot3(hA, ...), h = ezplot3(...)` — аналогично `ezcontour`.
- **ezsurf** — построение каркасной модели поверхности функции двух переменных. Использование аналогично `ezmesh` (см. разд. "Графическое представление функций" главы 17).
- **ezsurfc** — построение каркасной модели поверхности функции двух переменных и линий уровня на плоскости  $xy$ . Использование аналогично `ezmesh`.
- **fill3** — рисование закрашенного цветом многоугольника в трехмерном пространстве (см. разд. "Графическое представление функций" главы 17).
  - `fill3(x, y, z, c)` — рисует закрашенный цветом многоугольник. Вершины многоугольника ( $x(i)$ ,  $y(i)$ ,  $z(i)$ ) содержатся в трех первых входных аргументах, причем  $\text{length}(x) = \text{length}(y) = \text{length}(z)$ . 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. функцию `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, 'PropertyName', 'PropValue', 'PropertyName', 'PropValue', ...)` — пары '`PropertyName`', '`PropValue`' 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 (см. разд. "Трехмерные графики функций" главы 3).

- `hidden on` — удаление невидимых частей.
- `hidden off` — отображение невидимых частей.
- `hidden` — переключение между режимами `on` и `off`.

□ `mesh` — построение каркасной поверхности (см. разд. "Графики функций двух переменных" главы 2 и разд. "Трехмерные графики функций" главы 3).

- `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 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(..., 'PropertyName', PropertyValue, 'PropertyName', PropertyValue)` — 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` (см. главу 9).

- **`meshc`** — построение каркасной поверхности вместе с линиями уровня на плоскости  $xy$ . Использование аналогично `mesh` (см. разд. "Трехмерные графики функций" главы 3).
- **`meshz`** — построение каркасной поверхности вместе с линиями уровня на поверхности. Использование аналогично `mesh`.
- **`plot3`** — построение линий в трехмерном пространстве (см. разд. "Построение параметрически заданных поверхностей и линий" главы 3).
  - `plot3(x, y, z)` — отображение линий, проходящей через точки с координатами  $(x(i), y(i), z(i))$ , где  $x$ ,  $y$  и  $z$  являются векторами

одинаковой длины. Дополнительный строковый аргумент (см. функцию `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  $x_1, y_1, z_1, x_2, y_2, z_2$  и т. д. могут быть либо векторами, либо матрицами одинаковых размеров, а строковые аргументы  $s_1, s_2$  и т. д. задают стиль линий.
- `plot3(x, y, z, 'PropName', 'PropValue', 'PropName', 'PropValue', ...)` — указание свойств линии каждого графика парами, содержащими название свойства и его значение (см. функцию `plot`).
- `h = plot3(...)` — выходной аргумент вектор `h` содержит указатели на все созданные линии (рисованные объекты `Lineseries`).

Свойства каждого из графических объектов могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

□ **`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'`'.
- `hA = slice(...)` — графический вывод производится на оси с указателем `hA`.
- `h = slice(...)` — в выходном аргументе возвращается вектор указателей на созданную поверхность.

Свойства каждой поверхности могут быть изменены в дальнейшем при помощи `set` (см. главу 9).

□ **`sphere`** — отображение сферы и генерация точек, лежащих на поверхности сферы.

- `sphere` — 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))`, пример:
- ```
◊ >> 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. функцию `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)`.

- **`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)` (см. разд. "Трехмерные графики функций" главы 3).

Управление способом заливки цветом ячеек каркасной поверхности производится при помощи команды `shading`, задаваемой после вызова `surf`. Параметры `faceted` (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 с 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 (см. разд. "Освещение объектов, объект Light (источник света)" главы 9).

`surface('PropertyName', 'PropertyValue', 'PropertyName', 'PropertyValue', ...)` — 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° . Высокоуровненные 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. функцию `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 с 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 с 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 $[1, \text{length}(\text{colormap})]$.

Цвет каждого узла каркасной сетки поверхности можно задавать напрямую, не привязываясь к цветовой палитре. 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`. Заначение '`flat`' приводит к постоянно-му 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 пикселя 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.

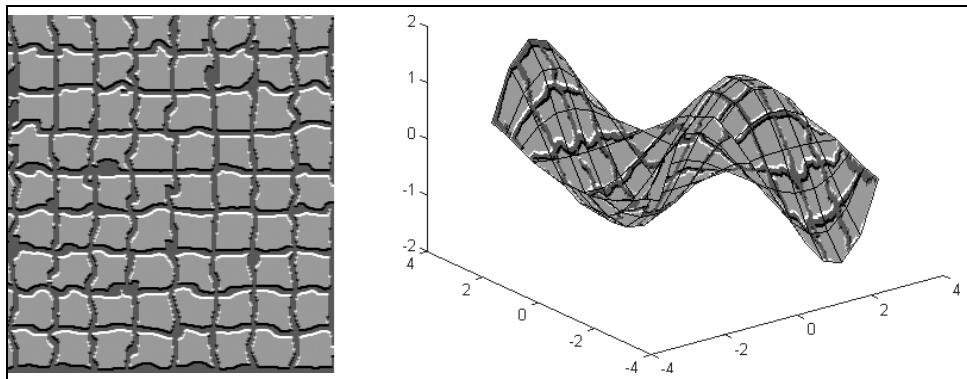


Рис. П2. Текстура и текстурированная поверхность

Стиль и толщина линий каркасной поверхности устанавливаются при помощи свойств `LineStyle` и `LineWidth`. Значением `LineStyle` может 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 `AmbientLightColor` осей (объекта `axes`).
- `BackFaceLighting` — определение способа частей поверхности в зависимости от направления нормали к поверхности и расположения наблюдателя. Значение `reverselit` (установленное по умолчанию) соответствует 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 ребра. 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)
```

- **surf** — построение залитой цветом каркасной поверхности и линий уровня на плоскости xy . Использование **surf** аналогично **surf** и **meshc** (см. разд. "Трехмерные графики функций" главы 3).
- **surfl** — построение освещенной поверхности (см. разд. "Построение освещенной поверхности" главы 3).
 - **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 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 векторных полей

- **compass** — отображение радиус-векторов (см. разд. "Визуализация векторных полей" главы 3).
 - **compass(hA, ...)** — построение графика на осях с указателем **hA**.
 - **h = compass(...)** — в вектор **h** записываются указатели на созданные объекты линии, свойства которых можно изменить при помощи **set** (см. главу 9).
 - **coneplot** — визуализация трехмерных векторных полей.
- Векторное поле задается двумя наборами трехмерных массивов x , y , z и U , V , W (все шесть массивов должны быть одинакового размера):
- x , y и z — координаты точек трехмерного пространства, из которых исходят векторы;
 - U , V и W — величины проекций векторов на оси x , y и z .

Функция `coneplot` позволяет указать начальные точки для построения векторов, которые не обязаны совпадать с точками, определенными массивами `X`, `Y` и `Z`. Координаты этих точек задаются массивами `Cx`, `Cy` и `Cz`.

- `hC = coneplot(X, Y, Z, U, V, W, Cx, Cy, Cz)` — визуализация векторного поля конусами, длина каждого конуса пропорциональна длине соответствующего вектора. Выходной аргумент `hC` содержит указатель на созданный полигональный объект (состоящий из конусов). Работа с полигональными объектами описана в главе 9.
- `hC = coneplot(X, Y, Z, U, V, W, Cx, Cy, Cz, s)` — то же самое, что и предыдущее обращение, но длина каждого конуса увеличивается в `s` раз (если `s` не указано, то оно принимается равным 1). Значение `s=0` отменяет автоматическое масштабирование.

Использование функции `coneplot`, как правило, требует установки некоторых свойств осей. Листинг П2 содержит пример визуализации векторного поля, заданного вектор-функцией

$$F(x, y, z) = \begin{bmatrix} u(x, y, z) \\ v(x, y, z) \\ w(x, y, z) \end{bmatrix}; \quad u(x, y, z) = \frac{x}{\sqrt{x^2 + y^2 + z^2}};$$

$$v(x, y, z) = \frac{y}{\sqrt{x^2 + y^2 + z^2}}; \quad w(x, y, z) = \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

на трехмерной сетке $(x_i, y_j, z_k)_{i,j,k=1,2,\dots,n}$, которая содержит $n=41$ узел по каждому из направлений:

$$x_i = 1 + \frac{i-1}{20}; \quad y_j = -1 + \frac{j-1}{20}; \quad z_k = \frac{k-1}{20}.$$

Точки, из которых исходят конусы, заданы на более редкой сетке $(cx_i, cy_j, cz_k)_{i,j=1,2,\dots,p; k=1,2,\dots,q}$, где $p=4$ и $q=8$, узлы которой не совпадают с $(x_i, y_j, z_k)_{i,j,k=1,2,\dots,n}$. Для получения значений вектор-функции на сетке $(cx_i, cy_j, cz_k)_{i,j=1,2,\dots,p; k=1,2,\dots,q}$ по умолчанию используется линейная интерполяция.

Листинг П2. Пример использования функции `coneplot`

```
% Задание границ области определения вектор-функции
xL = 1; xR = 3;
yL = -1; yR = 1;
```

```
zL = 0; zR = 2;
% Генерация матриц, содержащих координаты узлов мелкой сетки
[X, Y, Z] = meshgrid(xL:0.05:xR, yL:0.05:yR, zL:0.05:zR);
% Вычисление компонент вектор-функции на этой сетке
U = X./sqrt(X.^2 + Y.^2 + Z.^2);
V = Y./sqrt(X.^2 + Y.^2 + Z.^2);
W = Z./sqrt(X.^2 + Y.^2 + Z.^2);
% Генерация матриц, содержащих координаты узлов редкой сетки
[Cx, Cy, Cz] = meshgrid(xL:(xR - xL)/3:xR, yL:(yR - yL)/3:yR, ...
    zL:(zR - zL)/7:zR);
% Создание графического окна
figure
% Построение векторного поля с увеличением автоматически
% масштабированного конуса в 4 раза и запись указателя на полученный
% полигональный объект в переменную hC
hC = coneplot(X, Y, Z, U, V, W, Cx, Cy, Cz, 4)
% Задание зеленого цвета для граней полигонального объекта
% и скрытие ребер (выполняется для каждого конуса)
set(hC,'FaceColor','g','EdgeColor','none')
% Установка пределов осей, точно соответствующих границам
% изменения данных
axis tight
% Задание точки обзора
view(31,28)
% Добавление источника света
light
% Нанесение подписей к осям
xlabel('x')
ylabel('y')
zlabel('z')
```

Функция coneplot допускает другие варианты вызова.

- coneplot(U, V, W, Cx, Cy, Cz) — по умолчанию принимается, что $[X, Y, Z] = \text{meshgrid}(1:n, 1:m, 1:p)$, где m, n и p являются размерами массива U (т. е. $[m, n, p] = \text{size}(U)$).
- coneplot(..., 'quiver') — вместо конусов рисуются стрелки (см. ниже функцию quiver3).

- `coneplot(..., 'nearest')` или `coneplot(..., 'cubic')` — вместо линейной интерполяции применяется, соответственно, интерполяции по ближайшим соседям или кубическая (см. разд. "Интерполяция двумерных и многомерных данных" главы 6, а также описание функции `interp3` в справочной системе MATLAB).
 - `coneplot(X, Y, Z, U, V, W, 'nointerp')` — то же самое, что и `coneplot(X, Y, Z, U, V, W, Cx, Cy, Cz)`, но в качестве массивов `Cx`, `Cy` и `Cz` соответственно принимаются массивы `X`, `Y` и `Z` (интерполяция не требуется).
 - `coneplot(hA, ...)` — осуществляет графический вывод на оси с указателем `hA`.
 - `hC = coneplot(...)` — возвращает указатель на созданный полигональный объект. Этот указатель используется для изменения свойств полигонального объекта.
- **`feather`** — отображение векторов исходящими из равноотстоящих точек на одной прямой (см. разд. "Визуализация векторных полей" главы 3).
- `feather(hA, ...)` и `h = feather(...)` — аналогично `compass`.
- **`quiver`** — визуализация двумерного вектороного поля (см. разд. "Визуализация векторных полей" главы 3).
- `quiver(hA, ...)` — аналогично `compass`.
 - `h = quiver(...)` — возвращает указатель на рисованный объект `Quivergroup`, свойства которого могут быть изменены при помощи `set` (см. главу 9).
 - `h = quiver('v6', ...)` — возвращает вектор указателей на созданные базовые объекты `Line` для совместимости с предыдущими версиями MATLAB (базовые и рисованные объекты описаны в разд. "Графические объекты" главы 9).
- **`quiver3`** — визуализация вектор-функции от трех переменных, определенной на некоторой поверхности (см. разд. "Визуализация векторных полей" главы 3).
- `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. функцию `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` (см. главу 9), например:

```
>> h = quiver3(X, Y, Z, U, V, W)
>> set(h(1), 'Color', 'r')
>> set(h(2), 'Color', 'k')
```

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. Стока содержат 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
```

Функция `delaunay` основана на программе `Qhull` и допускает задание тех же самых параметров алгоритма триангуляции, что и `Qhull` (см. <http://www.qhull.org/>).

Функция `delaunay` используется, например, для задания треугольной сетки при визуализации функций, область определения которых не прямоугольная (см. функции `trimesh` и `trisurf` ниже).

□ **`trimesh`** — построение каркасной поверхности функции на произвольной (не обязательно прямоугольной) области. Область задается сеткой из треугольников. Сетка генерируется при помощи функции `delaunay` (см. выше).

- `trimesh(tri, x, y, z)` — построение каркасной модели поверхности на треугольной сетке с координатами узлов, заданных в векторах `x` и `y`, и значениями функции в векторе `z`. Матрица `tri` содержит информацию о триангуляции.

Поверхность графика функции является полигональным объектом (`Patch`) (свойства полигональных объектов и работа с ними описаны в разд. *"Объект Patch, цветовое оформление объектов"* главы 9).

- `trimesh(tri, x, y, z, c)` — указание цвета полигонального объекта в матрице `c`.

Задание свойств поверхности производится при помощи функции `set` или непосредственно во входных аргументах:

- `trimesh(..., 'PropertyName', 'PropertyValue', 'PropertyName', 'PropertyValue', ...)`.
- `h = trimesh(...)` — возвращает указатель на созданный полигональный объект.

Применение `set` для изменения свойств графических объектов рассматривается в главе 9.

Листинг П3 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 П3. 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)

```

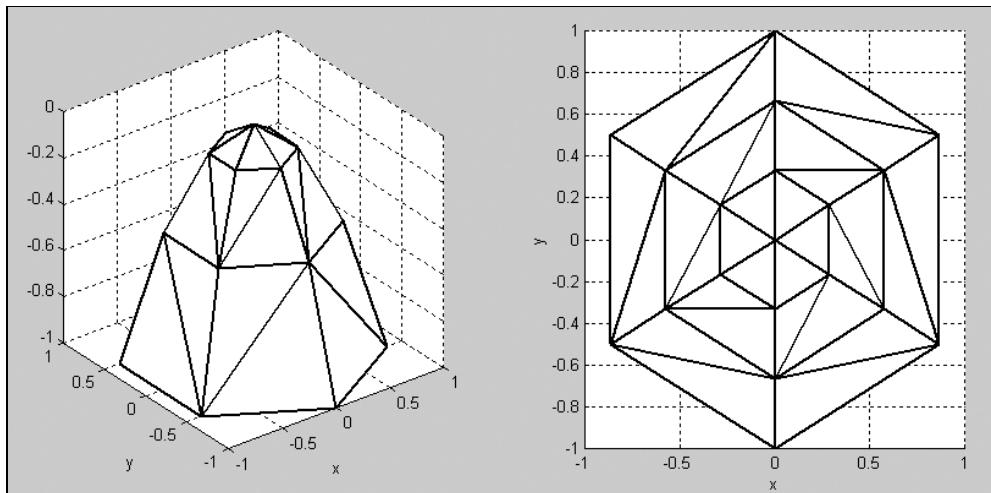


Рис. П3. График функции на треугольной сетке и сетка

- **trisurf** — построение каркасной закрашенной поверхности функции на произвольной (не обязательно 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 **h** на сами линии, являющиеся многоугольниками (графическими объектами типа **patch**). Данные аргументы инициализируются при соответствующем обращении к **contour**, **contourf** или **contour3** (см. 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 <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), рядом 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(..., 'PropertyName', 'PropertyValue', ...)` — 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 пункта (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-mm-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 (см. разд. "Оформление графиков" главы 3).

- `grid on` — отображение сетки на текущих осях.
- `grid off` — скрытие сетки на текущих осях.
- `grid` — если линии сетки отсутствуют, то они отображаются, а если присутствуют, то скрываются.

Полное управление сеткой осуществляется при помощи свойств `XGrid`, `YGrid`, `ZGrid`, `XMinorGrid`, `YMinorGrid` и `ZMinorGrid` координатных осей (см. разд. "Свойства осей" главы 9).

□ **gtext** — режим интерактивного размещения текстовых подсказок 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({'string1', 'string2', ...})` — указание в качестве входного аргумента вектор-строки ячеек из текстовых строк позволяет вводить многострочный текст.
- `gtext({'string1'; 'string2'; ...})` — указание в качестве входного аргумента вектор-столбца ячеек из текстовых строк позволяет разместить текст за несколько щелчков мышью.
- `gtext(..., 'PropertyName', PropertyValue, ...)` — пары входных аргументов определяют свойства добавляемого текста, как объекта `text` (свойства текстовых объектов описаны в разд. "Вывод текстовой информации" главы 9).

□ **hold** — управление выводом нескольких графиков в одно окно (см. разд. "Вывод нескольких графиков на одни оси" главы 3 и "Влияние команд `hold`, `cla`, `clf` и `reset` на свойства окна и осей" главы 9).

- `hold on` — каждый новый график добавляется в текущее графическое окно.
- `hold off` — каждый новый график отображается в текущем графическом окне, переписывая содержимое окна.
- `hold` — переключение между режимами добавления графика и выводом нового графика с потерей предыдущих.

□ **legend** — помещение легенды на график (см. разд. "Оформление графиков" главы 3).

- `legend(string1, string2, string3, ...)` — входные аргументы являются строками или строковыми переменными, содержащими пояснения для графических объектов, расположенных на текущих осях.
- `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 массиве строк : `legend(M)` или `legend(h, M)` (работа с массивами строк описана в разд. "Массивы строк" главы 9).

- `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(..., 'Location', location)` — задание положения легенды в векторе `location = [x y width height]`, где `x` и `y` — координаты нижнего левого угла в системе координат графического окна, а `width` и `height` — ширина и высота легенды. Единицы измерения определяются в нормализованных единицах (см. разд. "Размещение окон, осей и текста" главы 9).

Параметр `location` может принимать одно из предопределенных текстовых значений, например: '`North`', '`South`' и др. (см. разд. "Оформление графика" главы 3).

Произвольное положение легенды определяется в режиме редактирования графиков (см. главу 4).

Программное изменение положения и свойств объектов легенды производится при помощи обращения `[hLeg, hO, hP, txt] = legend(...)`, которое возвращает: указатель на оси легенды (`hLeg`), указатели на графические и текстовые объекты легенды (`hO`), указатели на объекты графика (`hP`), массив ячеек из текстовых строк легенды (`txt`). Свойства данных объектов могут быть изменены при помощи функции `set` (см. главу 9).

- `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 (см. разд. "Оформление графика" главы 3).
 - `title(str)` — текст, содержащийся во входном аргументе (строке или строковой переменной) помещается в графическом окне вверху осей. Текст может быть представлен в формате TeX.

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. Например, для использования интерпретатора LaTeX следует установить свойство `Interpreter` в значение '`latex`' (см. разд. "Вывод математических формул в формате LaTeX" главы 9):

```
>> h = title('$$f(x) = \int\limits_0^x t \sin t dt$$',
'Interpreter', 'latex')
```

- `h = title(...)` — возвращает указатель на создаваемый заголовок (текстовый объект), свойства которого могут быть изменены при помощи функции `set` (см. разд. "Вывод текстовой информации" главы 9).

□ `xlabel, ylabel` и `zlabel` — подписи к осям (см. разд. "Оформление графика" главы 3).

- `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, PropertyValue, ...)` — помещаемая подпись к оси (объект `text`) имеет свойства, определяемые парами входных аргументов (см. разд. "Вывод текстовой информации" главы 9).
- `h = xlabel(...)` — возвращает указатель на создаваемую подпись к оси (объект `text`), для изменения его свойств следует воспользоваться функцией `set` (см. главу 9).

Управление видом графика, камера

Несколько разделов книги посвящены описанию способов, предлагаемых MATLAB для изменения вида графиков (см. разд. "Поворот графика, изменение точки обзора" главы 3 и разд. "Обзор графиков и поверхностей" главы 4).

Ниже приведены 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. функцию `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 пикселях, 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 П4 содержит пример использова-

ния camorbit во вложенных циклах с целью осмотра поверхности со всех сторон.

Листинг П4. Вращение камеры вокруг объекта

```
figure  
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.
 - `cemode = 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 П5.

Листинг П5. 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 с координатами 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`.
- `cvemode = camva('mode')` — выходной аргумент `emode` 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 `ax`.

□ 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 П6.

Листинг П6. 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. функцию `viewmtx`).
- $T = \text{view}$ — получение текущей матрицы преобразования значений азимута и угла склонения.

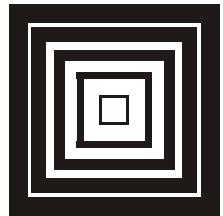
□ **viewmtx** — вычисление матрицы преобразования.

- $T = \text{viewmtx}(az, el)$ — возвращает матрицу ортогонального проектирования для отображения трехмерных объектов на плоскости (экране монитора) в соответствии с точкой обзора, определяемой азимутом и углом склонения (см. функцию `view`). Сама точка обзора на текущих осях не изменяется. Для получения матрицы проектирования, соответствующей текущему положению точки обзора, следует использовать обращение $T = \text{view}$.
- $T = \text{viewmtx}(az, el, phi)$ — 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, первые две компоненты которых, поделенные на четвертую, являются искомыми проекциями на плоскость экрана. Листинг П7 содержит пример изображения куба с различной перспективой.

Листинг П7. Изменение перспективы изображения

```
figure
% Задание координат вершин куба
x = [0 1 1 0 0 0 1 1 0 0 1 1 1 1 1 0 0];
y = [0 0 1 1 0 0 0 1 1 0 0 0 1 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
```



Приложение 2

Описание компакт-диска

К книге приложен компакт-диск, который содержит листинги программ и команды, приведенные в тексте. Диск организован следующим образом. В корневом каталоге находятся папки с именами `Work_XX`, где `XX` — номер главы. Каждая папка содержит М-файлы и файл `readme.txt`, в котором описано соответствие между именами М-файлов и номерами листингов программ данной главы.

Папки с номерами `Work_10`, `Work_11`, `Work_12`, `Work_13` и `Work_22` отвечают тем главам, которые посвящены созданию приложений с графическим интерфейсом пользователя. Изучая эти главы, вы будете программировать подфункции обработки событий и модифицировать их. Многие файлы содержат подфункции, предназначенные для вставки в основную функцию обработки событий. Поэтому имена М-файлов с подфункциями называются `listing_x.m`, где `x` — номер листинга в тексте. Это же замечание касается одного файла `listing_17.m` в папке `Work_16` с набором подфункций для приложения с графическим интерфейсом.

В каждый каталог `Work_XX` помещен файл `summaryXX.m`, представляющий из себя М-файл, разбитый на ячейки, который содержит основные с точки зрения авторов команды и функции из текста книги, не оформленные в виде листингов. Используя редактор М-файлов, их можно исполнять одновременно с чтением книги, не набирая в командной строке. Рекомендуется последовательное выполнение ячеек, т. к. команды ячейки могут использовать ранее созданные переменные среды (организация работы в М-файле, разбитом на ячейки, описана в разд. *"Разбиение М-файла на ячейки"* главы 5).

Другая возможность состоит в открытии файлов `summaryXX.m` текстовым редактором (можно изменить расширение для упрощения вызова на `txt`) и копировании исполняемого кода в командную строку MATLAB.

Для использования прилагаемых М-файлов следует либо скопировать их в текущий каталог MATLAB, либо скопировать целиком подкаталог и сделать его текущим (установка текущего каталога описана в разд. *"Установка путей"* главы 5, которая целиком посвящена работе с М-файлами и редактору М-файлов). После этого вы можете запустить М-файл из командной строки или редактора М-файлов или вызвать как файл-функцию (в зависимости от содержимого М-файла).

Список литературы

1. Ануфриев И. Е. Самоучитель MatLab 5.3/6.x. — СПб.: БХВ-Петербург, 2002. — 736 с.
2. Дьяконов В. П. Matlab 6/6.1/6.5+Simulink 4/5. Основы программирования: Руководство пользователя. — М.: Солон-Пресс, 2002. — 768 с.
3. Мэтьюз Д., Финк К. Численные методы. Использование MATLAB (3-е издание). — СПб.: Вильямс, 2001. — 720 с.
4. Чен К., Джиблин П., Ирвинг А. MATLAB в математических исследованиях. — М.: Мир, 2001. — 346 с.
5. Черных И. В. Simulink: среда создания инженерных приложений. — М.: Диалог-МИФИ, 2004. — 496 с.
6. Kwon Y.W. The Finite Element Method using MATLAB. — Boca Raton a. o.: CRC Press, 1997. — 519 p.