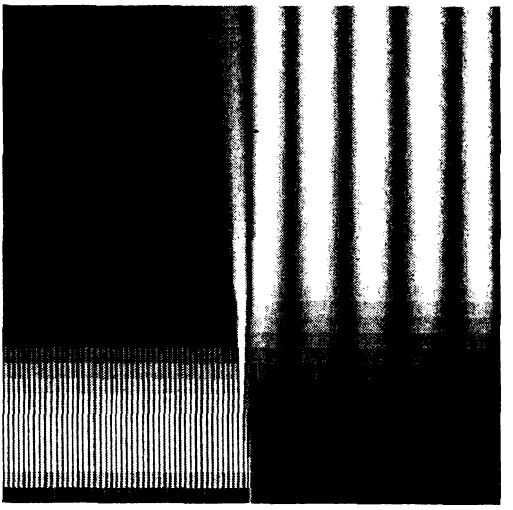
Учебник



MATLAB 7

основы работы и программирования

С. В. Поршнев

Учебник

MATLAB 7

основы работы и программирования

Допущено учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению 654600 «Информатика и вычислительная техника»

Москва
Издательство **БИНОМ**
2011

УДК 004.42
ББК 32.973.26-018.1
П60

С. В. Поршнев
MATLAB 7. Основы работы и программирования. Учебник — М.: ООО «Бином-Пресс», 2011. — 320 с.: ил.

Книга является учебником по математическому пакету MATLAB, владение которым сегодня является одним из показателей уровня профессиональной подготовки выпускников естественнонаучных и технических факультетов университетов в области информационных технологий. Она предназначена для старших классов школ, лицеев, колледжей, младших курсов ВУЗов при изучении информатики, математики, физики и других смежных дисциплин. Может служить основой для самостоятельного обучения приемам работы с MATLAB.

В книге широчайшие возможности пакета описаны в контексте практического решения конкретных математических и физических задач, что позволяет повысить эффективность обучения, и обеспечивает уверенное освоение читателем представленных сведений. Рассмотрены приемы работы с MATLAB при использовании его в режимах мощного суперкалькулятора, позволяющего, в том числе, проводить символьные вычисления, средства визуализации результатов вычислений, языка программирования высокого уровня. Большое количество включенных в книгу задач способствует развитию необходимых навыков использования пакета при изучении различных разделов математики и физики.

Книга рассчитана на начинающих и не требует никаких предварительных сведений о MATLAB.

ISBN 978-5-9518-0244-6

Содержание

Предисловие

MATLAB (сокращение от MATrix LABarotatory — МАТричная ЛАБоратория), разработанная фирмой The MathWorks Inc. (США, г. Нейтик, шт. Массачусетс), является интерактивной системой для выполнения инженерных и научных расчетов. Сегодня MATLAB используется в более 70 ведущих университетов мира, таких широко известных научно-исследовательских центров, как НАСА, Национальная лаборатория в Лос-Аламосе, Стэнфордском исследовательском институте и др.

В MATLAB интегрирован мощный математический аппарат, позволяющий решать сложные задачи без вызова внешних процедур, который, в частности, позволяет находить решения:

- линейных и нелинейных алгебраических уравнений и систем;
- задачи Коши и краевой задачи для дифференциальных уравнений;
- дифференциальных уравнений в частных производных;
- задач статистической обработки данных (вычисление статистических параметров, интерполяция, аппроксимация, сглаживание, т.д.);
- задач линейной алгебры (операции с матрицами и векторами);
- задач поиска экстремумов функциональных зависимостей.

MATLAB предоставляет пользователю мощные средства графического представления информации (визуализация функций, зависящих от одной переменной, полярных графиков, графиков поверхностей, карт линий уровня, векторных полей и т.д.). Пакет снабжен средствами анимации, что позволяет рассматривать временную эволюцию математических моделей в динамике и т.д. Кроме того, в MATLAB интегрирован математический аппарат, реализующий символьные вычисления.

Вычислительная мощь MATLAB сочетается с простотой изучения и освоения его входного языка, который, принимая во внимание принцип непосредственного исполнения, можно сравнить с языком программирования BASIC. Фактически MALAB является языком программирования высокого уровня для научных и технических расчетов. Он включает в себя практически все известные на сегодняшний день методы числительной математики.

В книге изложены основы работы с пакетом MATLAB, в том числе, описан графический интерфейс пользователя; приведены примеры вычислений; средства визуализации функций, зависящих от одной переменной, и средства 3D визуализации; встроенные средства решения типовых задач линейной алгебры; функции, осуществляющие символьные вычисления; изложены основы программирования на М-языке и технология создания графического интерфейса пользователя (Graphics Unit Interface — GUI). Кроме того, мы посчитали целесообразным включить в книгу примеры использования MATLAB для обработки экспериментальных данных, расчета и визуализации электростатического поля, создаваемого системой неподвижных зарядов, а также построения фрактальных объектов.

Книга адресована начинающим пользователям MATLAB 7, но также будет полезна и пользователям предыдущих версий, начиная с версии 5.0.

Графический интерфейс пользователя и простейшие вычисления

Графический интерфейс пользователя MATLAB состоит из 4 независимых окон, имеющих следующие названия, **Workspace, Command Window, Command History, Current Directory** (рис. 1.1).

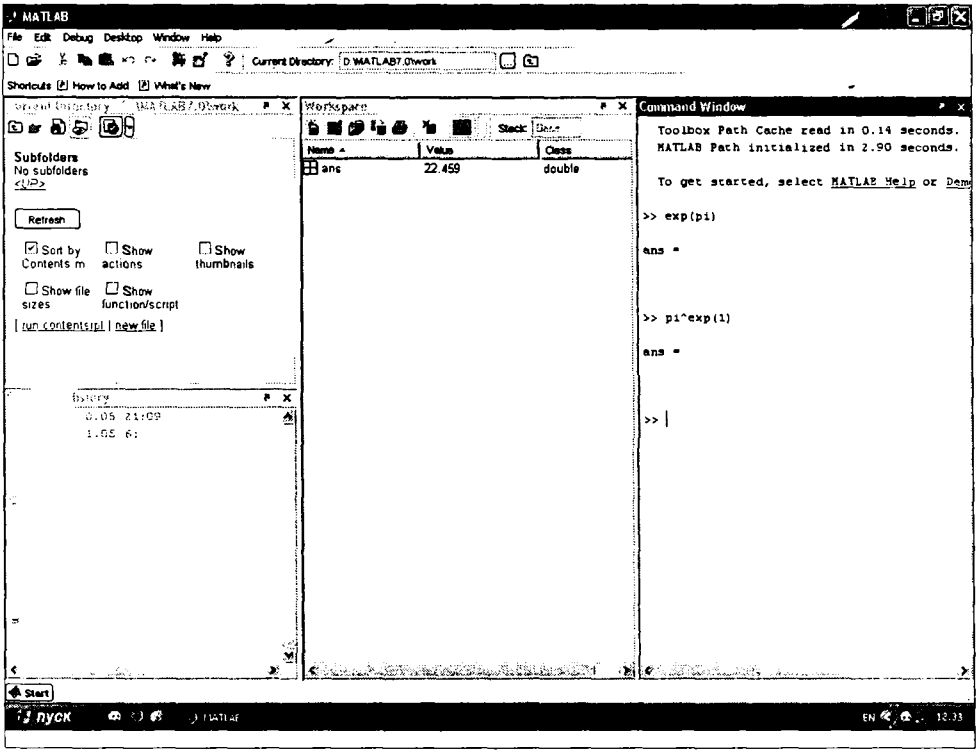


Рис. 1.1. Графический интерфейс пользователя пакета MATLAB

Окно **Workspace** используется для отображения состояния зарезервированной области памяти компьютера, в которой хранятся переменные, используемые в данном сеансе работы.

Окно **Command Window** является основным окном, в котором вводятся исполняемые команды и отображаются результаты вычислений. Более подробно структура данного окна обсуждается в разделе 1.1.

Окно **Command History** используется для отображения содержимого буфера, в котором хранятся выполненные ранее команды пакета.

В окне **Current Directory** отображается список файлов и вложенных папок активного в данный момент каталога.

Для управления окнами используются стандартные средства операционной системы Windows. Для закрытия окон, отображение которых представляется пользователю в данный момент нецелесообразным, используется команды меню **View**. Опыт работы с пакетом автора показывает, что наиболее удобна для работы форма графического интерфейса, когда на экране компьютера одновременно отображены окна **Workspace**, **Command Window**, **Command History**.

1.1. Командное окно пакета MATLAB

Структура окна **Command Window**, состоящая из строки меню, панели инструментов, рабочей области и полосы состояния (рис. 1.2), аналогична структуре Windows-приложений.

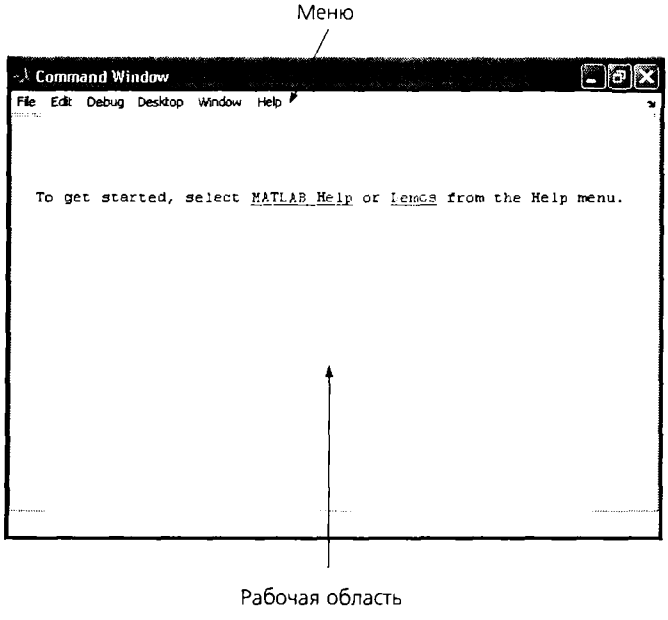


Рис. 1.2. Структура окна **Command Window**

В рабочей области окна **Command Window** находится строка ввода команд, отмеченная знаком >>, в котором можно вводить числа, имена переменных и знаки операций, составляющие в совокупности выражение. Имена переменных должны начинаться с буквы и состоять из букв, цифр и знаков препинания. MATLAB распознает в именах переменных до 31 символа и различает регистр символов. Простейшими знаками операций являются всем хорошо известные знаки арифметических операций + и −. Знак = соответствует операции присваивания. Нажатие клавиши «**Enter**», является для MATLAB заданием выполнить введенную команду и отобразить полученный результат (рис. 1.3).

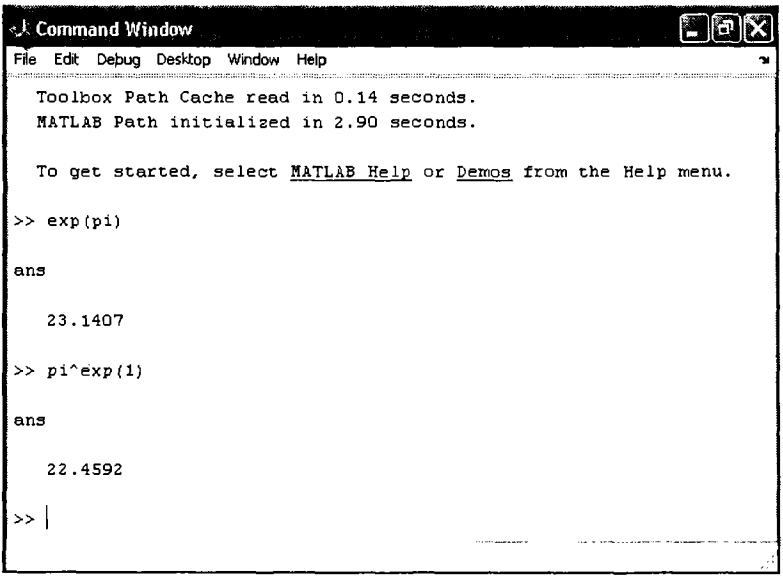


Рис. 1.3. Командное окно MATLAB после выполнения вычислений

После отображения результата вычисления в командном окне создается новая строка ввода команд, отмеченная знаком >>.

Для просмотра выполненных команд и результатов вычислений, не умещающихся в командном окне, имеются полосы горизонтальной и вертикальной прокрутки (скроллинга). Использование полос прокрутки ни чем не отличается от других Windows-приложений. Также можно осуществлять прокрутку в содержимого командного окна MATLAB с помощью следующих клавиш клавиатуры: **PageUp**, **PageDown**, **Ctrl+Home** (одновременное нажатие клавиш **Ctrl** и **Home**) и **Ctrl+End**.

Необходимо отметить, что в MATLAB клавиши управления курсором «↑» и «↓», осуществляющие в текстовых редакторах перемещение курсора вниз или вверх и вертикальный скроллинг содержимого окна, работают иначе. В MATLAB клавиши «↑», «↓» используются для возврата в строку ввода ранее выполненных команд, каждая из которых перед ее выполнением запоминается в стеке команд. Стек команд — это область оператив-

ной памяти ПК, отведенная для хранения выполненных в данном сеансе работы команд. При этом просмотр стека осуществляется с его конца, то есть последняя выполненная команда будет отображаться в строке команд первой. При нажатии на клавишу « ↓ » осуществляется прокрутка команд, расположенных в стеке, в обратном направлении.

Командное окно MATLAB разделено на две принципиально различных зоны: зону просмотра и зону редактирования (рис. 1.4).

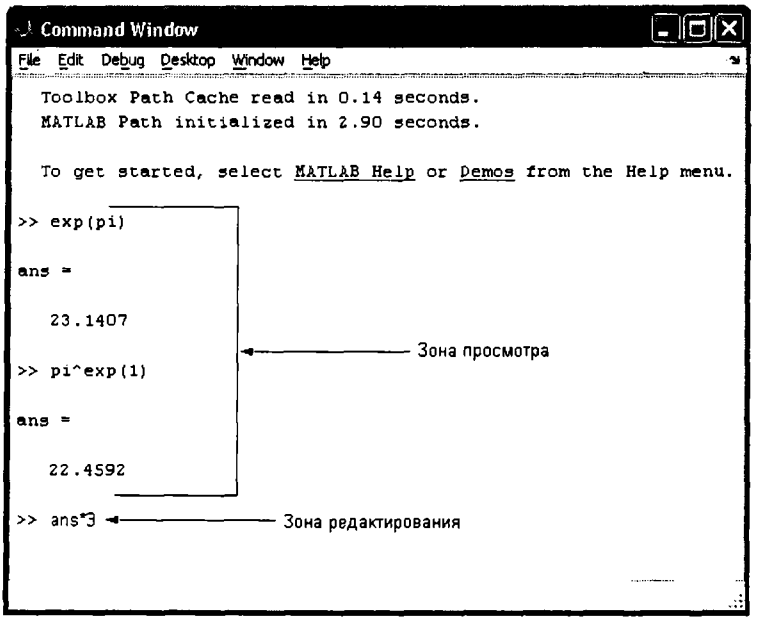


Рис. 1.4. Структура командного окна MATLAB

Исправление информации в зоне просмотра невозможно, несмотря на то, что в любую строку, находящуюся в данной зоне можно поместить курсор. Попытка любая попытка редактирования текста, размещенного в зоне просмотра, приведет к автоматическому перемещению курсора в строку ввода, расположенную в зоне редактирования. В зоне просмотра, как и в известных текстовых редакторах, можно с помощью мыши выделить любой фрагмент текста, затем скопировать его в буфер обмена операционной системы Windows, а затем вставить данный фрагмент в командную строку MATLAB или документ, созданный в каком-либо приложении, работающим под управлением операционной системы Windows.

Зона редактирования находится в строке командного окна MATLAB, отмеченной знаком >>. Отметим, что существует возможность «удлинения» командной строки за счет размещения вводимой команды на несколько физических строках командного окна. Такая строка называется логической строкой ввода. При вводе команды, размещаемой в нескольких физических строках, каждая текущая строка завершается тремя точками и нажатием на клавишу **ENTER** (рис 1.5).

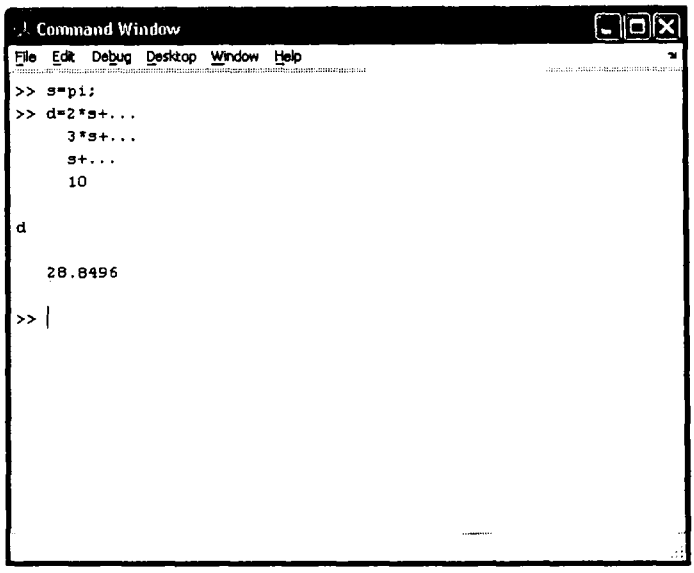


Рис. 1.5. Ввод команды, размещающейся на нескольких логических строках ввода

В этом случае зона редактирования распространяется только на строку, в которой находится курсор. Его перемещение на выбранную строку можно осуществлять, как в любом текстовом редакторе, с помощью мыши, после чего можно использовать клавиши управления курсором «→», «←». Суммарная длина логической строки ввода не может превышать 256 символов.

Очистка командного окна MATLAB осуществляется командой

```
clc,
```

которая, однако, оставляет неизменным содержимое буфера команд и рабочего пространства MATLAB. Действительно, если после этого набрать имя ранее вычисленной переменной **d**, то после нажатия клавиши **ENTER** мы снова увидим ее значение:

```
>> d
d =
  28.8496
```

1.2. Рабочее пространство пакета MATLAB

Значения переменных, вычисленных в течение текущего сеанса работы, сохраняются в специально зарезервированной области оперативной памяти компьютера, называемой рабочим пространством MATLAB (MATLAB Workspace).

Для того чтобы узнать текущее значение любой переменной, размещенной в рабочем пространстве MATLAB достаточно набрать в командной строке имя переменной и нажать клавишу «**Enter**». Однако более удобным, с нашей точки зрения, является использование окна **Workspace**, в котором отображаются все переменные, использованные в данном сеансе работы с пакетом (рис. **1.6**).

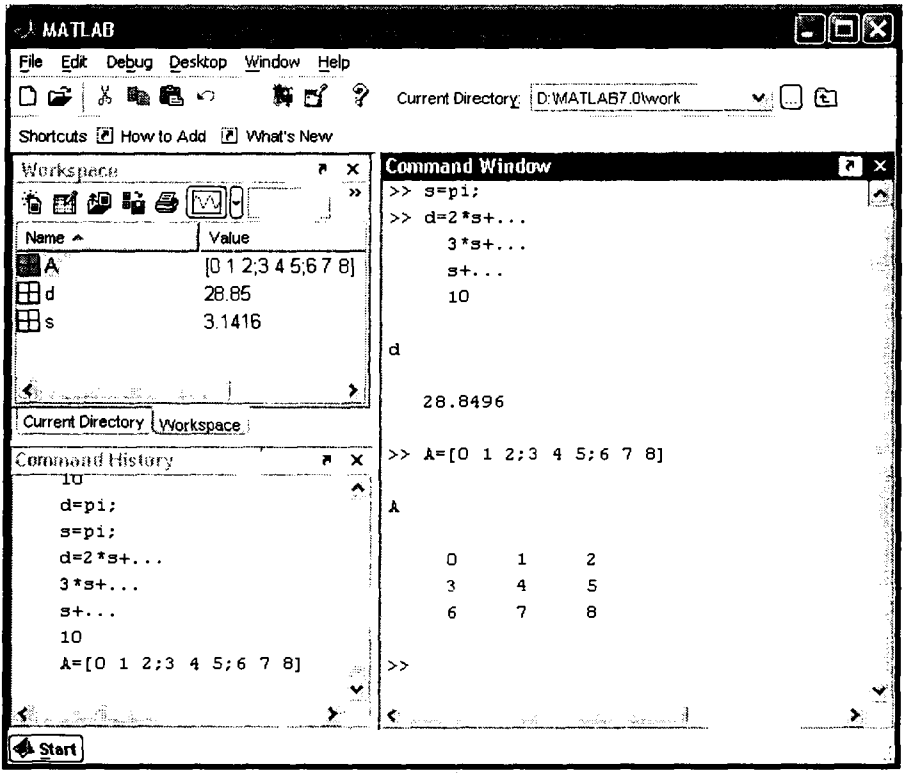


Рис. 1.6. Структура рабочего пространства MATLAB, после выполнения команд, представленных на рис. 1.5 и в командном окне на данном рисунке

Двойной щелчок левой кнопкой мыши по строке, содержащей имя переменной приводит к появлению окна **Array Editor**, в котором можно просматривать и менять значения выбранной переменной. Для примера на рис. **1.7** показано соответствующее окно, появившееся после двойного клика по строке, содержащей имя переменной **A**.

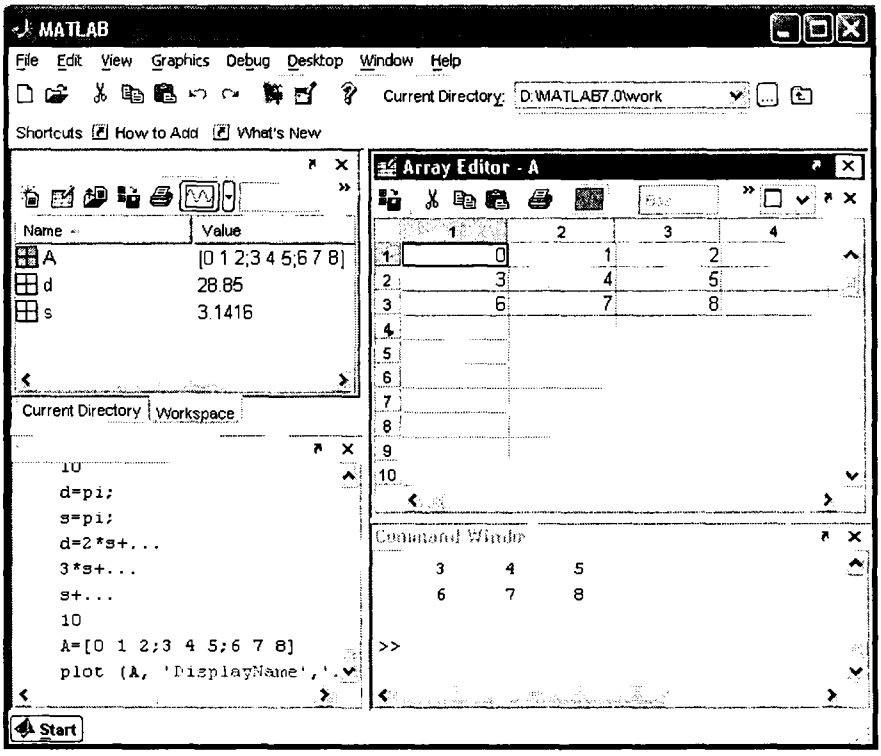


Рис. 1.7. Просмотр элементов матрицы A

Отметим, что эффективность работы пакета будет снижается по мере увеличения объема рабочего пространства, поэтому при исчезновении в текущем сеансе работы необходимости хранения некоторых переменных, их следует удалять из памяти компьютера командой, имеющий следующий синтаксис

```
clear name1 name2
```

Здесь **name1, name2, ...** — имена удаляемых переменных.

Также можно обратиться к команде **clear**, как к функции, аргументами которой являются, строки, содержащие имена удаляемых переменных:

```
clear( 'имя1', 'имя2' )
```

Для одновременного удаления всех переменных следует использовать команду

```
clear
```

Для отображения имен переменных, размещенных в данный момент в рабочем пространстве, нужно выполнить команду

```
who
```

Данная команда отображает в рабочем окне список всех переменных, находящихся в рабочем пространстве MATLAB. Выполнив команды, представленные на рис. 1.6, и далее команду

```
>> who
```

получим

```
Your variables are:
A  Y
>>
```

После завершения сеанса работы с MATLAB все ранее вычисленные переменные теряются. Для сохранения в файле на диске компьютера содержимого рабочего пространства, нужно выполнить команду меню

```
File → Save Workspace as...
```

после чего появляется стандартное диалоговое окно операционной системы Windows для выбора каталога на диске и имени файла. Для файлов, в которых MATLAB сохраняет значения переменных из рабочего пространства должно использоваться расширение **mat**, поэтому такие файлы принято называть «MAT-файлами».

Вместо использования рассмотренной команды меню можно непосредственно в командном окне MATLAB указать путь к папке, в которой будут сохранен данный файл, введя команду

```
save path_to__file\имя_MAT-файла
```

и нажав клавишу **ENTER**. Если путь к сохраняемому файлу не указан, то он будет сохранен в текущем рабочем каталоге. Название данного каталога отображается в полосе инструментов командного окна MATLAB в окне **Current Directory** (см. рис. 1.1). Отметим, что получить имя текущего каталога можно также с выполнив команду

```
cd
```

а изменить текущий каталог командой

```
cd путь_к_новому_каталогу
```

В каждом сеансе работы с MATLAB целесообразно в качестве текущего каталога задавать тот каталог, с файлами которого предстоит работать чаще всего.

Для загрузки в последующих сеансах работы в оперативную память компьютера ранее сохраненного на диске файла, содержащего рабочее пространство MATLAB, нужно выполнить команду меню

```
File → Load WorkSpace.
```

которая в стандартном Windows-диалоговом окне **Load .mat file** потребует указать нужный MAT-файл.

Можно соединить в текущем рабочем пространстве MATLAB содержимое нескольких предыдущих сеансов работы, загрузив последовательно несколько разных файлов. В тоже время важно помнить о том, что при сов-

падении имен переменных из разных сеансов работы в текущем сеансе в рабочем пространстве будет представлена лишь переменная из последнего открытого MAT-файла.

Вместо рассмотренной команды меню можно использовать команду

```
load имя_MAT-файла
```

непосредственно в командном окне MATLAB.

Существует возможность считать из записанного на диске MAT-файла в рабочее пространство значения отдельных переменных. Для этого используется предыдущая команда, дополненная именами соответствующих переменных:

```
load имя_MAT-файла имя1, имя2,
```

В результате ее выполнения из MAT-файла будут считаны переменные с именами **имя1**, **имя2** и т.д. Если MAT-файл указан без полного пути к нему, то он должен находиться в текущем каталоге MATLAB.

Завершая краткое рассмотрение командного окна MATLAB, отметим следующие особенности ее команд. (Под командами пользователя мы понимаем предписания MATLAB выполнить некоторое действие: например, показать текущий каталог.) Часть команд MATLAB могут задаваться несколькими способами: с помощью меню главного (командного) окна, с помощью кнопок на полосе инструментов и с помощью ввода с клавиатуры ключевых (зарезервированных) слов с последующим нажатием клавиши «ENTER». Другие команды можно реализовать только с помощью ввода с клавиатуры соответствующих им ключевых слов (например, команда **cd**).

Режим работы с MATLAB, в котором пользователь вводит команды в командной строке, задавая математические выражения или обращаясь к функциям MATLAB, называется интерактивным режимом.

По любой команде MATLAB можно получить быструю справку, выполнив команду

```
help имя команды
```

Более просто получить доступ к внутренней справочной информации MATLAB, выполнив команду меню **Help → Matlab Help**, в результате чего появится диалоговое окно, представленное на рис 1.8.

Выбирая в его левой половине пункт MATLAB **Function Reference** и далее пункт **Alphabetical List of Functions**, получаем доступ к упорядоченной по алфавиту справочной информации по всем командам и функциям ядра MATLAB.

В том случае, когда для решения конкретной задачи встроенных функций MATLAB оказывается недостаточно, пользователь имеет возможность создавать собственные функции, используя для этого как внутренний язык MATLAB (M-язык), так и языки программирования высокого уровня Fortran, C, C++, Java. Вопросы, касающиеся программирования на M-языке, рассматриваются в главе 7.

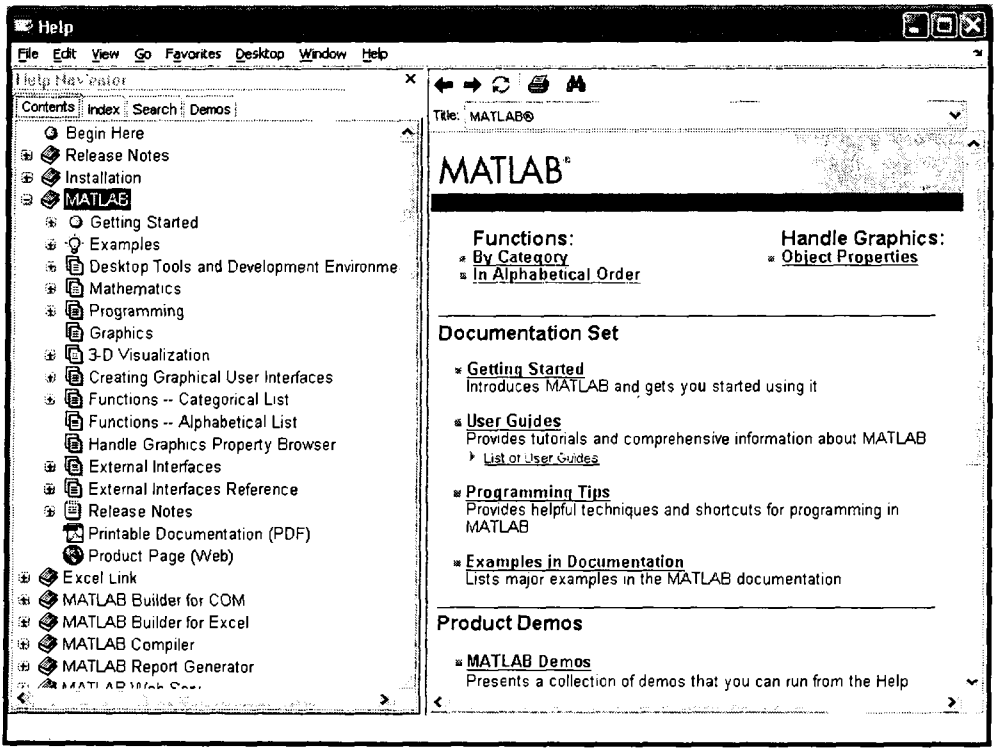


Рис. 1.8. Окно справочной системы MATLAB

1.3. Формат представления вещественных чисел

В MATLAB для представления чисел (как вещественных, так и целых) используется формат с плавающей точкой, в котором любое число задается мантиссой и показателем степени и записываются в следующем виде

```
8.134838545e+10; -564.83549687; 0.0054968e0; 581e-1; 5.4; -312
```

где буквой **e** обозначается основание степени, равное 10.

Этот тип данных называется **double**. Под мантиссу и показатель степени (на машинном уровне используется двоичная система записи) отводится 8 байт памяти. В результате для десятичных чисел достигается точность порядка 15 значащих цифр. В MATLAB максимальным по модулю вещественным числом является число

```
1.797693134862316e+308
```

а минимальным по модулю — число

```
2.225073858507202e-308
```

Для этих чисел зарезервированы имена **realmax** и **realmin**.

Для отображения чисел в командном окне в MATLAB используются следующие форматы: **short**, **long**, **rat**.

По умолчанию для вывода вещественных чисел MATLAB использует формат **short**, который предусматривает отображение только четырех десятичные цифры после запятой (см. рис. 1.6).

При необходимости использовать полное представление вещественных чисел следует ввести с клавиатуры команду

```
format long
```

После чего результаты всех последующих вычислений будут отображаться в данном формате. Например, для отображения значения переменной **Y**, значение которой было задано выше, необходимо выполнить следующую последовательность команд:

```
>> format long
>> Y
Y =
 Column 1
 1.00000000000000
 -0.41614683654714
 -0.65364362086361


 Column 2
 0.54030230586814
 -0.98999249660045
 0.28366218546323
>>
```

При необходимости вернуться к предыдущему формату отображения вещественных чисел до прекращения текущего сеанса работы в командном окне, нужно ввести и исполнить (нажав клавишу **ENTER**) команду

```
format short
```

Для отображения вещественных чисел в виде обыкновенных дробей используется формат **rat**, устанавливаемый следующей командой

```
format rat
```

После выполнения этой команды ранее вычисленные переменные, являющиеся рациональными дробями, отображаются в командном окне MATLAB в неизменном виде:

```
>> R=1000/33;
>> format rat
>> R
R =
 1000/33
```

В данном формате операнды и результаты вычислений, являющихся целыми числами, в командном окне MATLAB отображаются в виде целых

чисел, хотя в памяти машины они представляются так же, как и дробные числа.

Вычислим сумму двух чисел, используя MATLAB, как обычный калькулятор:

```
>> 410+5
ans =
  415
```

Приведенный пример иллюстрирует общее правило: если пользователь не сохраняет результат вычисления введенного выражения, то MATLAB всегда сохраняет последнее из вычисленных значений в переменной **ans**.

Для переменных типа **double** в MATLAB разрешены арифметические операции сложения, вычитания, умножения и деления, для которых используются традиционные для любого языка программирования знаки + , – , * и /, а также операция возведения в степень, обозначаемая знаком ^:

```
>> 5^2
ans =
  25
>> t=ans^0.5
t =
   5
```

В MATLAB установлен общепринятый приоритет выполнения арифметических операций: самый высший приоритет имеет операция возведения в степень, затем — умножение и деление, и потом — сложение и вычитание. Операции одинакового приоритета выполняются в порядке слева направо. Для изменения порядка приоритета следует исцользовать круглые скобки.

Обратим внимание на роль точки с запятой в М-языке MATLAB, которая завершая командную строку, отменяет вывод результатов вычисления в командное окно. Кроме того, точка с запятой используется для задания и в командной строке нескольких команд, которые будут последовательно выполнены при нажатии клавиши «**Enter**». Ниже будут рассмотрены другие варианты использования точки с запятой.

1.4. Комплексные числа

В MATLAB переменным **i**, **j** по умолчанию присвоено значение $\sqrt{-1}$.

```
>> clear
>> i
ans =
     0     +     1i
>> j
ans =
     0     +     1i
>> i*i
ans =
    -1
```

При работе с комплексными числами существует опасность, что переменным **i**, **j** будет присвоены новые значения, например, при использовании в циклах **for** в качестве индексной переменной. Для избежания подобной ситуации следует перед началом работы с комплексными числами выполнять команду

```
>> clear i, j
```

Основные приемы работы с комплексными числами поясним на следующих примерах:

1. Зададим комплексные числа:

```
>> a=1 + i
 a =
  1 + 1i
>> b = 2   3i
 b =
  2 - 3i
```

2. Вычислим произведение комплексных чисел

```
>> a*b
 ans =
    5 - 1i
```

3. Вычислим $\sqrt{a}$

```
>> d=sqrt(a)
 d =
  1.0987 + 0.4551i
```

4. Вычислим b^3

```
>> b^3
 ans =
 -46.0000 - 9.0000i
>>
```

5. Вычислим $|a|$

```
>> abs(a)
 ans =
  1.4142
```

6. Вычислим действительную (re(b)) и мнимую (im(b)) части комплексного числа

```
>> real(b)
 ans =
  2
>> imag(b)
ans =
  -3
```

7. Вычислим аргумент комплексного числа (arg(a))

```
>> angle(a)
ans =
0.7854
```

8. Вычислим число комплексно сопряженное числу b

```
>> conj(b)
ans =
    2.0000 + 3.0000i
>>
```

9. Вычислим sin(a)

```
>> sin(a)
ans =
    1.2985 + 0.6350i
```

1.5. Векторы и матрицы

Основные приемы работы с векторами и матрицами продемонстрируем следующими примерами.

1. Зададим вектор-строку:

```
>> ul=[1 1 1];
u =
     1     1     1
>> whos ul
 Name       Size            Bytes  Class
 ul         1x3             24  double array
Grand total is 3 elements using 24 bytes
>>
```

2. Зададим вектор-столбец:

```
>> u2=[2;1;-1]
u2 =
2
1
-1
>> whos u2
 Name       Size            Bytes  Class
 u2         3x1             24  double array
Grand total is 3 elements using 24 bytes
```

3. Зададим вектор с использованием числового диапазона:

```
>> dialp=3.7:0.5:8.947;
>> dialp
dialp =
    Columns 1 through 7
     3.7000   4.2000   4.7000   5.2000   5.7000   6.2000   6.7000
```

```
    Columns 8 through 11
     7.2000    7 7000    8.2000    8.7000
>> length(dialp)
ans =
11
```

4. Вычислим скалярное произведение векторов

```
>> a=[1 2 3];
>> b=[3 2 1];
>> a*b
??? Error using ==> *
Inner matrix dimensions must agree.
```

Здесь MATLAB выдал сообщение об ошибке, потому что в соответствии с правилами умножениями матриц, принятыми в линейной алгебре можно умножать вектор-строку на вектор столбец, поэтому, для вычисления скалярного произведения необходимо предварительно транспонировать вектор **b**:

```
>> a*b'
ans =
  10
>>
```

5. Поэлементное умножение векторов

```
>> a=[1 2 3];
>> b=[3 2 1];
>> a.*b
ans =
  3    4    3
>>
```

6. Создадим матрицу

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

7. Выделим заданный столбец матрицы

```
>> A(:,1)
ans =
 -1
  3
 -1
>>
```

8. Выделим заданную строку матрицы

```
>> A(2,:)
ans =
```

```
     3   -1    1
>>
```

9. Выделим определитель матрицы

```
>> det(A)
ans =


    10
>>
```

10. Вычислим обратную матрицу

```
>> inv(A)
ans =
 -0.7000    0.2000    0.3000
 -1.3000   -0.2000    0.7000
  0.8000    0.2000   -0.2000
>>
```

1.6. Элементарные функции

В MATLAB встроены все основные элементарные математические функции, которые представлены в табл. 1.1.

Таблица 1.1. Основные элементарные математические функции

Тригонометрические функции	
sin	sin(X) вычисляет синус от элементов числового массива X. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
sinh	Вычисляет гиперболический синус. Области определения и значений могут быть комплексными.
asin	Y=asin(X) вычисляет арксинус от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах. Для вещественных X из отрезка [−1, 1] значения Y принадлежат отрезку $[-\pi/2, \pi/2]$. Для вещественных X вне отрезка [−1, 1] Y принимает комплексные значения.
asinh	Y=asinh(X) вычисляет гиперболический арксинус от элементов массива. Области определения и значений могут быть комплексными.
cos	Вычисляет косинус от элементов массива.
cosh	Вычисляет гиперболический косинус от элементов массива.
acos	Y=acos(X) вычисляет арккосинус от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах Для вещественных X из отрезка [−1, 1] значения Y принадлежат отрезку $[-\pi, \pi]$. Для вещественных X вне отрезка [−1, 1] Y принимает комплексные значения.
acosh	Вычисляет гиперболический арккосинус от элементов массива.

tan	Вычисляет тангенс от элементов массива. Области определения и значений могут быть комплексными Углы измеряются в радианах. Выражение tan(π/2) не является точным inf, поскольку π есть лишь приближение к π.
tanh	Вычисляет гиперболический тангенс от элементов массива.
atan	Вычисляет арктангенс. Области определения и значений могут быть комплексными. Углы измеряются в радианах. Для вещественных X значения Y принадлежат отрезку ($-\pi$/2, π/2).
atan2	Для вещественных x,y z = atan2(x,y) есть угол наклона вектора с координатами x, y и принимает значения из [$-\pi$, π]. x,y могут быть массивами одинаковых размеров.
atanh	Вычисляет гиперболический арктангенс от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
sec	Вычисляет секанс от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
sech	Вычисляет гиперболический секанс от элементов массива. Области определения и значений могут быть комплексными
asec	Вычисляет арксеканс от элементов массива Области определения и значений могут быть комплексными. Углы измеряются в радианах.
asech	Вычисляет гиперболический арксеканс от элементов массива. Области определения и значений могут быть комплексными.
csc	Вычисляет косеканс от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
csch	Вычисляет гиперболический косеканс от элементов массива. Области определения и значений могут быть комплексными.
acsc	Вычисляет арккосеканс от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
acsch	Вычисляет гиперболический арккосеканс от элементов массива. Области определения и значений могут быть комплексными.
cot	Вычисляет котангенс от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
coth	Вычисляет гиперболический котангенс от элементов массива. Области определения и значений могут быть комплексными. Углы измеряются в радианах.
acot	Вычисляет арккотангенс от элементов массива. Области определения и значений могут быть комплексными. Единица измерения углов — радиан.
acoth	Вычисляет гиперболический арккотангенс от элементов массива. Области определения и значений могут быть комплексными.
Экспоненциальные функции	
exp	Вычисляет экспоненциальную функцию от элементов числового массива. Области определения и значений могут быть комплексными.
log	Вычисляет натуральный логарифм от элементов массива. Области определения и значений могут быть комплексными.
log10	Вычисляет десятичный логарифм от элементов массива. Области определения и значений могут быть комплексными.

pow2	Y=pow2(X) есть показательная функция по основанию 2 от элементов массива X. Области определения и значений могут быть комплексными. Y=pow2(F,E) вычисляет $y=F\cdot 2^E$ для соответствующих элементов вещественного массива F и целочисленного массива E.
sqrt	Y=sqrt(X) вычисляет квадратный корень из элементов массива X. Для отрицательных и комплексных элементов X функция принимает комплексные значения.
nextpow2	Для комплексного числа a в обращении p = nextpow2(a) вычисляется такое минимальное p, что $2^p \geq$ abs(a). Если a не является скаляром, то вычисляется nextpow2(length(a)).
Комплексные функции	
abs	Вычисляет модуль для каждого элемента комплексного массива.
angle	Вычисляет аргумент в радианах для каждого элемента комплексного массива. Область значений — полуинтервал $(-\pi, \pi]$.
conj	Вычисляет комплексное сопряжение для каждого элемента комплексного массива.
imag	Выдает мнимую часть числа для каждого элемента комплексного массива.
real	Выдает вещественную часть числа для каждого элемента комплексного массива.
unwrap	Для вектора p, элементы которого равны углам некоторой последовательности комплексных чисел, в векторе q=unwrap(p) последовательно ликвидированы разрывы из p, превосходящие по модулю π, путем добавления кратного $\pm 2\pi$. При обращении unwrap(p,tol) для обнаружения скачка вместо используется tol. Для многомерного массива P команда работает по первому неединичному измерению, unwrap(P,[],dim) — по измерению dim в unwrap(P,tol,dim) используются tol и dim.
isreal	k=isreal(A) равно 1 если все элементы массива A — вещественные числа и равно 0 в любом другом случае.
cplxpair	B=cplxpair(A) упорядочивает элементы вдоль различных измерений комплексного массива A, группируя вместе сопряженные пары. Эти пары упорядочиваются по возрастанию вещественной части, и первым в паре идет элемент с отрицательной мнимой частью. Чисто вещественные числа располагаются после сопряженных пар. Все сравнения для нахождения сопряженных пар и отделения чисто вещественных чисел от комплексных выполняются с относительной к abs(A(i)) точностью 100*eps, при этом приблизительно сопряженные пары считаются точно сопряженными. Измерение в массиве для работы команды выбирается так же, как в команде unwrap, а вместо 100*eps можно использовать параметр tol. Если в упорядочиваемом массиве не все строго комплексные числа имеют комплексно сопряженные (с требуемой точностью) пары команда cplxpair выдает сообщение об ошибке **Complex numbers can't be paired.**
Функции округления и вычисления остатков	
fix	fix(A) округляет элементы вещественного массива A в сторону **−inf.** У комплексных массивов мнимые и вещественные части округляются независимо.
floor	floor(A) округляет элементы вещественного массива A в сторону нуля. В остальном действует как fix.
ceil	ceil(A) округляет элементы вещественного массива A в сторону inf.

round	round(A) округляет элементы вещественного массива A до ближайшего целого.
mod	M=mod(X,Y) возвращает остаток от деления X на Y.
rem	M=rem(X,Y) возвращает целую часть от деления X на Y.
sign	sign(x) возвращает -1 если $x < 0$, 0, если $x = 0$, 1, если $x > 0$.

В языках программирования высокого уровня вычисления с массивами осуществляются поэлементно, поэтому в соответствующих процедурах приходится программировать вычисление выражений для каждого элемента массива. В MATLAB этого не требуется, так как в М-языке имеются групповые операции, выполняемые сразу над всем массивом. В частности, в М-языке MATLAB можно производить групповые вычисления над массивами, используя обычные математические функции, которые в традиционных языках программирования работают только со скалярными аргументами. В результате запись команд становится более компактной и, следовательно, удобной для ввода с клавиатуры. Например, для вычисления таблицы значений функции косинус в равноотстоящих точках отрезка $[0;\pi/2]$, отстоящих друг от друга на расстояние 0,01, достаточно использовать всего две команды

```
>> x=0:0.01:pi/2;
>> y=cos(x);
>> whos x y
 Name        Size              Bytes  Class
 x           1x158              1264  double array
 y           1x158              1264  double array
Grand total is 316 elements using 2528 bytes
>>
```

Аналогично вычисляются функции от двумерных массивов:

```
>> A=[0 pi/6; pi/4 pi/2; 5*pi/6 3*pi/4]
A =
         0    0.5236
    0.7854    1.5708
    2.6180    2.3562
>> sin(A)
ans =
         0    0.5000
    0.7071    1.0000
    0.5000    0.7071
```

Вопросы для самопроверки

1. Какие окна имеет интерфейс MATLAB и каково их назначение?
2. Какова структура окна **Command Windows**?
3. Как в MATLAB осуществляется ввод и выполнение команд?
4. Назовите зоны окна **Command Windows**.

5. Как увеличить длину командной строки **Command Windows**, распространив ее на несколько физических строк командного окна?
6. Что называется рабочей областью MATLAB?
7. Как осуществляется просмотр и редактирование значений переменных в окне **Workspace**?
8. Как отобразить список переменных, созданных в данном сеансе работы, в командном окне?
9. Как удалить неиспользуемую более переменную из рабочего пространства MATLAB?
10. Как получить справку по выбранной команде MATLAB?
11. В каком формате в MATLAB представляются вещественные числа?
12. Какие форматы отображения вещественных чисел используются в MATLAB?
13. Как в MATLAB осуществляются операции с комплексными числами?
14. Как в MATLAB осуществляются операции с матрицами числами?
15. Как в MATLAB осуществляется вычисление элементарных функции от векторов и матриц?

Глава 2

Визуализация результатов вычислений

2.1. Построение графиков функций, зависящих от одной переменной

Изучение графических возможностей пакета начнем со знакомства с основными приемами работы с высокоуровневой графикой, так как она не требует от пользователя необходимости понимания всех тонких деталей дескрипторной графики.

В качестве примера, построим график функции, зависящей от одной переменной (рис. 2.1), выполнив для этого следующую последовательность команд:

```
>> x=0:0.001:2;
>> y=sin(x);
>> plot(x,y);
```

Из рис. 2.1 видно, что MATLAB создал графический объект в специальном графическом окне, имеющем заголовок Figure № 1. Данное окно имеет собственную панель меню и панель инструментов, которую мы рассмотрим ниже.

Выполнив следующие команды:

```
>> x=0:0.001:2;
>> z=cos(x);
>> plot(x,z);
```

при открытом графическом окне можно получить новый график, который заменяет предыдущую зависимость (рис. 2.2).

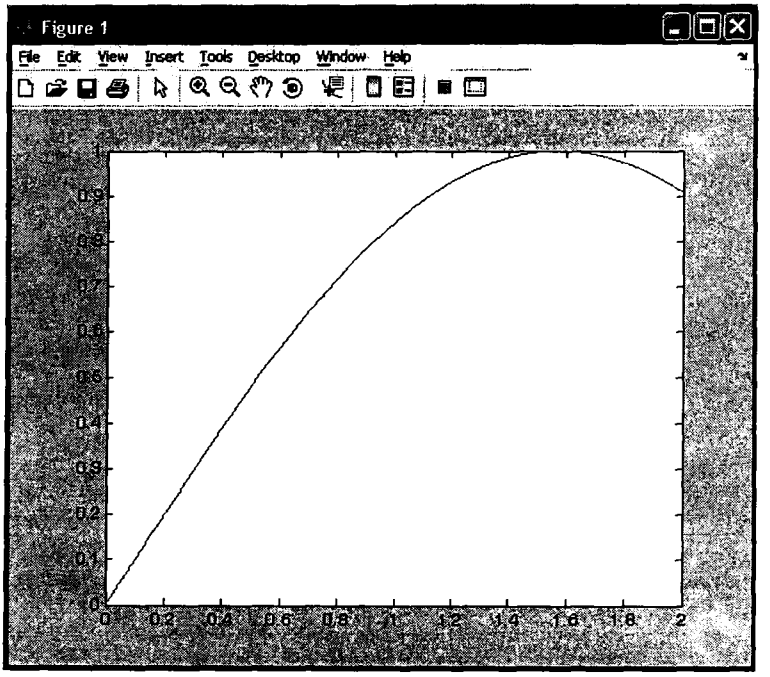


Рис. 2.1. График функции $y = \sin(x)$ на интервале [0,2]

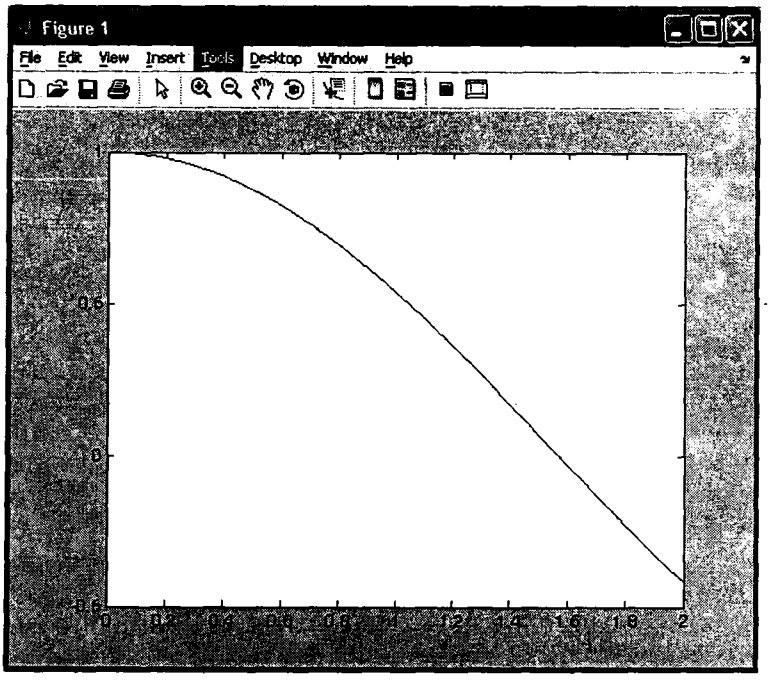


Рис. 2.2. График функции $y = \cos(x)$ на интервале [0,2]

Для построения нового графика в новом окне необходимо предварить команду **plot** командой **figure(2)**, которая создает новое графическое окно:

```
>> figure(2);plot(x,z);
```

Для одновременного отображения двух зависимостей в одном графическом окне необходимо перед построением второй зависимости выполнить команду **hold on** (рис. 2.3):

```
>> plot(x,y);
>> hold on
>> plot(x,z);
```

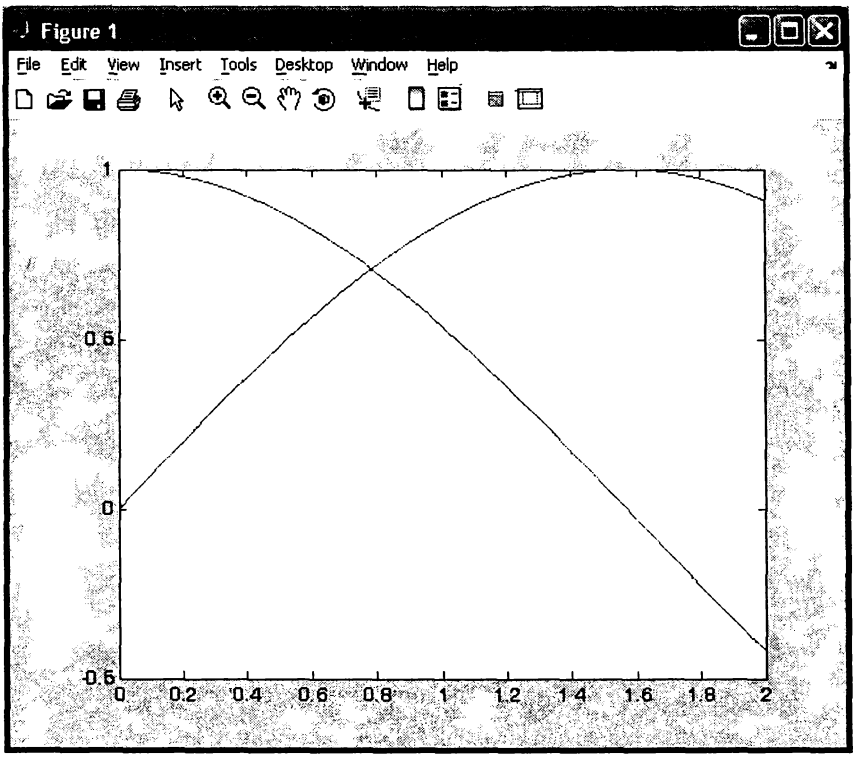


Рис. 2.3. Графики функций $y = \cos(x)$, $y = \sin(x)$ на интервале [0,2]

Аналогичный результат можно получить, выполнив команду:

```
>> plot(x,y,x,z);
```

Для создания окна с несколькими графическими окнами используется команда **subplot**:

```
>> subplot(1,2,1);plot(x,y);
>> subplot(1,2,2);plot(x,z);
```

Функция **subplot** принимает три числовых аргумента, первый из которых равен числу рядов подобластей, второй — числу колонок цодобластей, а третий аргумент — номеру подобласти (номер отсчитывается вдоль рядов слева направо с переходом на новый ряд по исчерпанию).

Если для одиночного графика диапазоны изменения переменных вдоль одной или обеих осей координат слишком велики, то можно воспользоваться функциями построения графиков в логарифмических масштабах. Для этого предназначены функции **semilogx**, **semilogy** и **loglog**.

Для построения графика функции в полярных координатах используется команда **polar**, например:

```
>> phi=0:0.001:2*pi;
>> r=sin(3*phi);
>> polar(phi,r);
```

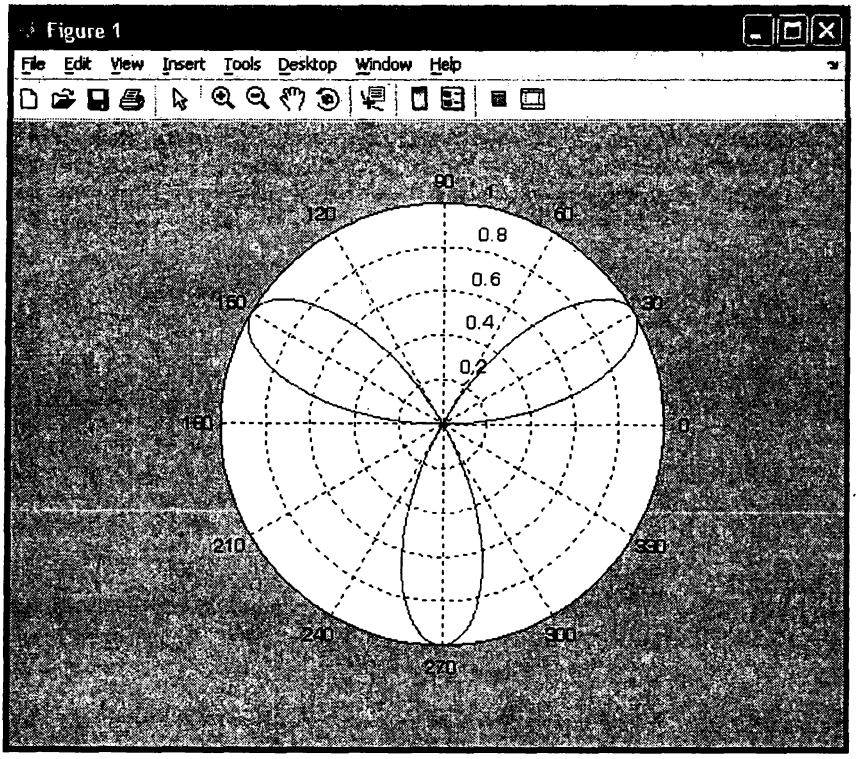


Рис. 2.4. График функции $y = \sin(3x)$ в полярной системе координат

2.2. Оформление графиков и графических окон

Функции построения графиков, рассмотренные нами выше, осуществляли автоматическое оформление графиков. Далее мы рассмотрим дополнительные возможности, связанные с управлением внешним видом графиков — задание цвета и стиля линий, а также размещение различных надписей в пределах графического окна.

Например, следующие команды

```
>> x=0:0.1:3;y=cos( x );
>> plot( x, y, 'r-', x, y, 'ko' )
```

позволяют придать графику вид красной сплошной линии, на которой в дискретных точках, расстояние между которыми равно 0.1, проставляются черные окружности. Здесь функция **plot** дважды строит график одной и той же функции, но в двух разных стилях. Первый из этих стилей отмечен как «**r–**», что означает проведение линии красным цветом (буква **r**), а штрих означает проведение сплошной линии. Второй стиль, помеченный как «**ko**», означает проведение черным цветом (буква **k**) окружностей (буква **o**) на месте вычисляемых точек (см. рис. 2.5).

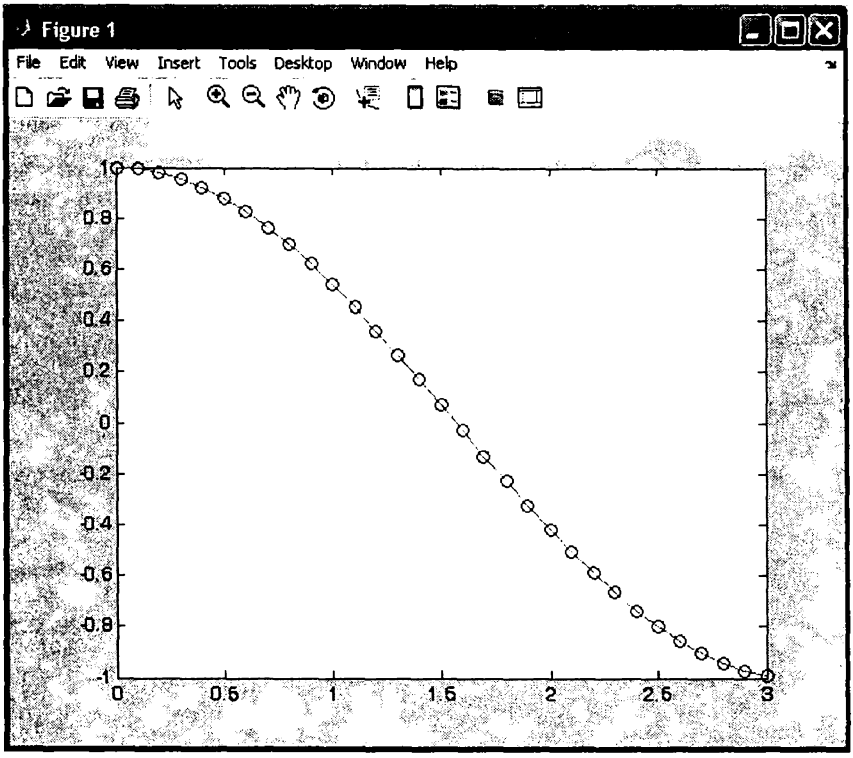


Рис. 2.5. Пример маркировки точек кривой с помощью окружностей

В общем случае функция

```
plot(x1,y1,s1,x2,y2,s2,...)
```

позволяет объединить в одном графическом окне изображения нескольких функций y1(x1), y2(x2), ..., используя стили s1, s2,

Стили s1, s2, ... задаются в виде набора трех символьных маркеров, один из которых задает тип линии (табл 2.1), второй — цвет линии (табл. 2.2), третий — тип маркера, используемого для пометки проставляемых точек (табл. 2.3).

Таблица 2.1. Типы линий и их маркировки

Маркер	Тип линии
	непрерывная
	штриховая
:	пунктирная
.	штрих-пунктирная

Таблица 2.2. Цвета линий и их маркировки

Маркер	Цвет линии
c	голубой
m	фиолетовый
y	желтый
r	красный
g	зеленый
b	синий
w	белый
k	черный

Таблица 2.3. Типы маркировки точек кривой

Маркер	Тип точки
.	точка
+	плюс
*	звездочка
o	кружок
x	крестик

При отсутствии указания типов маркеров используются значения, установленные по умолчанию. Отметим, что порядок в котором указываются маркеры, задающие тип линии, цвет линии и тип точки, не существенен.

Более мощным способом оформления графиков функций и других графических объектов является *метод дескрипторов*, который относится

к низкоуровневой графике MATLAB. Метод дескрипторов позволяет напрямую обратиться к базовым графическим объектам и задать его свойства. Детальное знакомство со свойствам графических объектов MATLAB, число которых у некоторых объектов близко к ста, выходит за рамки нашей книги. Поэтому ниже мы приводим некоторые примеры иллюстрирующие основные приемы работы с дескрипторной графикой и позволяющих продемонстрировать суть данного метода.

```
>> x=0:0.1:3;
>> y=sin(x);
>> hplot=plot(x,y); % создаем графический объект и присваиваем
                    % переменной
                    % hplot значение дескриптора
>> set(hplot,'LineWidth',8); % устанавливаем толщину линии
                             % 8 пикселей
```

Здесь функция **plot** через опорные точки проводит отрезки прямых линий с координатами x, y. Прямые линии в MATLAB представляют собой графические объекты типа **line**. Эти объекты имеют огромное число свойств и характеристик, которые можно менять. Доступ к этим объектам осуществляется по их *описателям* (*дескрипторам*; по-английски — **handles**).

Функция **plot** возвращает описатель графического объекта типа **line**, использованного для построения графика, который заносится в переменную **hplot**. Затем данный описатель передается в качестве параметра функции **set** для опознания конкретного графического объекта и задания значений соответствующих свойств объекта. В приведенном выше примере мы установили значение свойства **'LineWidth'** (толщина линии) равным 8 (по умолчанию — 0.5). График, полученный в результате изменения толщины линии, представлен на рис. (2.6).

Текущее значение любого параметра (атрибута; характеристики) графического объекта возвращается функцией **get**. Например, если после получения показанного на рисунке графика выполнить команду:

```
>> width = get( hPlot, 'LineWidth' )
width =
8
```

то для переменной **width** присвоено значение свойства **LineWidth**, равное 8.

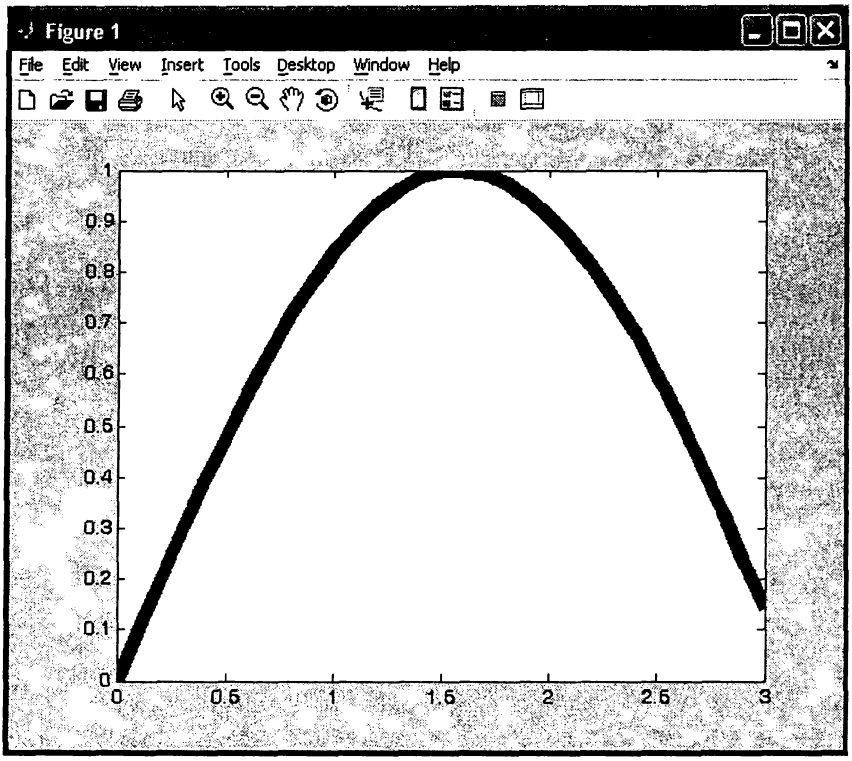


Рис. 2.6. График функции $y = \sin(x)$ на интервале [0,3] (толщина кривой — 8 пикселей)

Для получения списка всех свойств графического объекта следует обратиться к функции **get**, указав ей в качестве единственного параметра, описатель объекта. Например, для дескриптора **hplot** объекта типа **line** находим весь список его свойств:

```
>> get(hplot);
       Color = [0 0 1]
       EraseMode = normal
       LineStyle = -
       LineWidth = [7]
       Marker = none
       MarkerSize = [6]
       MarkerEdgeColor = auto
       MarkerFaceColor = none
       XData = [ (1 by 31) double array]
       YData = [ (1 by 31) double array]
       ZData = []

       BeingDeleted = off
       ButtonDownFcn =
       Children = []
       Clipping = on
       CreateFcn =
       DeleteFcn =
```

```
BusyAction = queue
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [100.001]
Selected = off
SelectionHighlight = on
Tag =
Type = line
UIContextMenu = []
UserData = []
Visible = on
```

Среди свойств графического объекта с дескриптором **hPlot** имеются интуитивно понятные. В частности, свойство **Color** отвечает за цвет линии в RGB-кодировке: Красный Зеленый Синий (так как значение поля равно **[0 0 1]** цвет линии синий); свойство **LineWidth** определяет толщину линии (равную в рассматриваемом случае восьми); свойство **MarkerSize** отвечает за размер маркера точек графика (равного в рассматриваемом случае 6); свойство **Parent** содержит дескриптор объекта-родителя данного объекта. Для профессиональной работы с дескрипторной графикой требуется знание всех свойств графических объектов. Однако от большинства пользователей не требуется их обязательное изучение, так как наиболее важным свойствам MATLAB присваивает по умолчанию вполне приемлемые значения. Установленные значения полей указываются в списке свойств справа от знака равно. При этом свойства, значения которых неустановленны, отмечены пустым полем или знаком **[]**.

Рассмотрим приемы оформления осей координат: задание надписей осей, названия осей, размера и типа шрифта и так далее. По умолчанию MATLAB устанавливает пределы на горизонтальной оси, равными минимальному (x_{min}) и максимальному (x_{max}) значениям независимой переменной. Для зависимой переменной по вертикальной оси MATLAB автоматически вычисляет диапазон изменения значений функции [y_{min},y_{max}]. В результате график функции оказывается вписанным в прямоугольник, нижний левый и верхний правый углы которого имеют, соответственно, координаты [x_{min},y_{min}], [x_{max},y_{max}].

При необходимости отказаться от указанной особенности автоматического масштабирования графиков в MATLAB, необходимо явным образом указать выбранные пределы изменения переменных x, y. Для выполнения данной процедуры используется функция

```
axis([ xmin, xmax, ymin, ymax])
```

Причем указанную команду можно вводить с клавиатуры уже после построения графика функции. Это позволяет провести визуальный анализ особенностей визуализируемой функции.

Например, для чтобы более подробно проанализировать поведение функции **sin** в окрестности точки максимума, можно установить пределы по осям координат **axis([1.5, 2.5, 0.5, 2])** (см. рис. 2.7):

```
>> set(hplot,'LineWidth',0.5);
>> axis([1.5,2.5,0.5,2]);
```

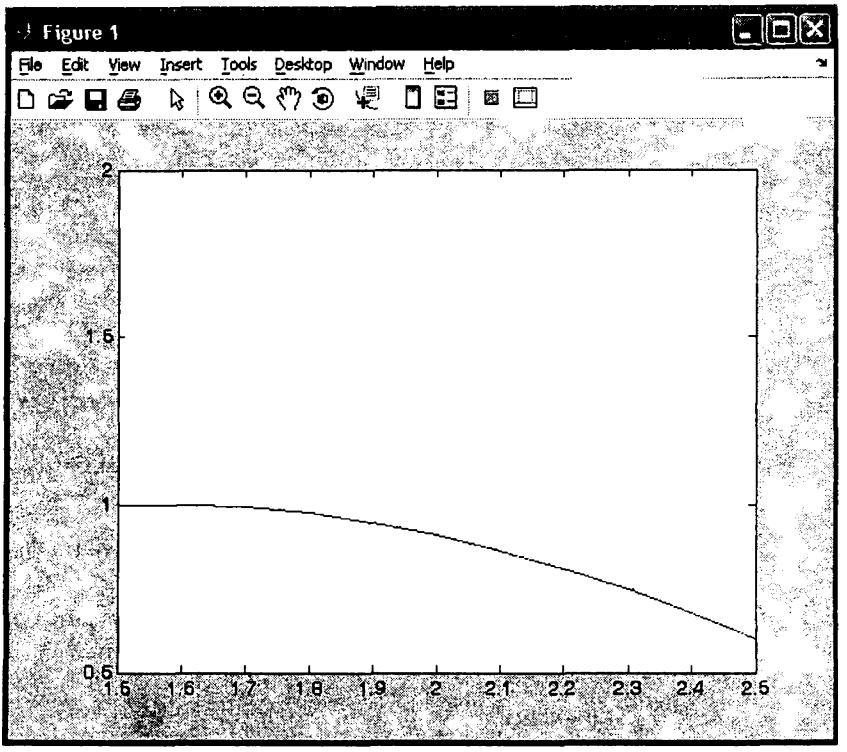


Рис. 2.7. Фрагмент графика функции $y = \sin(x)$

Для изменения отметок на осях координат используется функция **set**, позволяющая установить свойства объекта **axes**. Данный объект содержит оси координат и белый прямоугольник, внутри которого строится график функции. Для получения описателя объекта **axes** применяют функцию **gca**. Эту функцию вызывают без параметров:

```
>> hAxes=gca;
>> set(hAxes,'xtick',[1.5,1.75,2.0,2.25,2.5]);
```

Для размещения различных надписей на полученном рисунке применяют функции **xlabel**, **ylabel**, **title** и **text**. Функция **xlabel** предназначена для размещения названия горизонтальной оси, функция **ylabel** — названия вертикальной оси (причем эти надписи ориентированы вдоль осей координат).

Для размещения надписи в произвольном месте рисунка используется функция **text**:

```
>> text(x,y,'some text')
```

Здесь x, y — координаты нижнего левого угла прямоугольника, в котором размещается отображаемый текст. По умолчанию координаты задаются в тех же единицах измерения, что и координаты, указанные на горизонтальной и вертикальной осях.

Общий заголовок для графика выводится функцией **title**. Кроме того, используя команду

```
grid on
```

можно нанести измерительную сетку на всю область построения графика. Выполнив команды

```
>> title('Function sin graph');
>> xlabel('x coordinate');ylabel('sin(x)');
>> text(2.1,0.9,'\leftarrowsin(x)');
>> grid on
```

получим график функции, представленный на рис. 2.8.

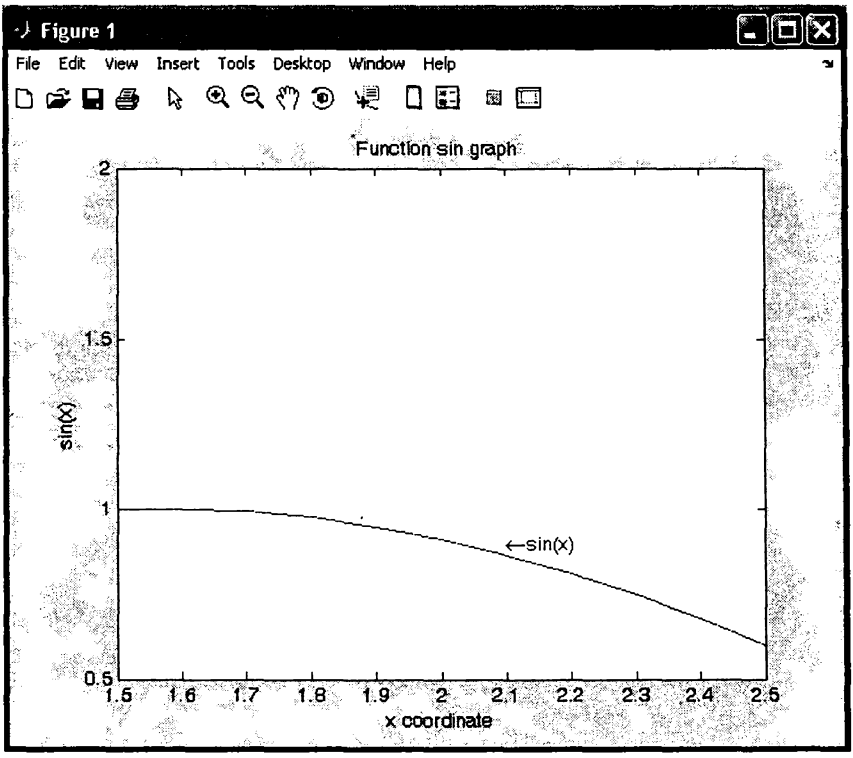


Рис. 2.8. Пример оформления графиков

Здесь мы использовали специальные *управляющие символы*, введенные внутри текста после символа \(«обратная косая черта») «стрелка влево». Команды отображения специальных символов MATLAB совпадают с соответствующими командами системы подготовки научных текстов TeX. В частности, для создания новой строки и вывода в нее текста применяется команда **\newline**.

Для того, чтобы установить шрифт, которым изображается надпись, нужно запомнить описатель (дескриптор), возвращаемый этой функцией:

```
>> hText = text(2.1,0.9,'\leftarrowsin(x)');
```

Располагая дескриптором **hText** объекта типа **text**, можно изменять ряд его атрибутов. Например, следующая команда

```
>> set(hText,'Color',[0 1 0 ],'FontSize',[18]);
```

изменит внешний вид ранее выведенной надписи — теперь используется более крупный шрифт, а цвет надписи стал зеленым.

Отметим, что для вывода в графическое окно текста, написанного кириллицей, необходимо использовать шрифт MS Sans Serif, который можно задать, установив значение поля **FontName** объекта **Text**:

```
>> set(hText,'FontName','MS Sans Serif');
```

в противном случае текс будет отображаться в виде набора специальных символов.

Завершая обсуждение способов оформления графиков функций, покажем, как изменить цвет фона, на котором осуществляется построение графиков. По умолчанию установлен белый цвет. Для того, чтобы сделать его слегка зеленоватым, нужно присвоить свойству **Color** объекта типа **axes** значение [0.5,0.8,0.5]. Для этого сначала надо получить описатель этого объекта:

```
hAxes = gca;
```

Функция **gca** предназначена для поиска описателя текущего объекта **axes**. При наличии нескольких объектов типа **axes** сначала нужно щелкнуть мышью на том из них, который должен стать текущим, и только после этого вводить представленную выше команду.

Смена фона прямоугольника, содержащего графическое изображение соответствующей зависимости, осуществляется командой:

```
>> set(hAxes,'Color',[0.5,0.8,0.5]);
```

Для смены фона всего графического окна нужно выполнить следующие команды:

```
>> FigureColor=[0.8,0.5,0.5]; hFigure=gcf;
>> set(hFigure, 'Color',FigureColor);
```

2.3. Инструментальная панель графических окон

В предыдущем разделе описаны команды, позволяющие изменить внешний вид графиков и окон. Большую часть выполненной работы можно сделать также и мышью, нажимая на кнопки инструментальной панели графических окон (или с помощью дублирующих команд меню этих окон).

Для оформления графиков функций (подрисовывания дополнительных линий, создания подписей координатных осей, текстовых комментариев и т.д.), также изменения стиля графических объектов используется соответствующий инструментарий MATLAB. Так, нажав левую кнопку со стрелкой (всплывающая подсказка для этой кнопки **Edit plot**), мы получаем возможность последующим щелчком левой клавишей мыши начать редактирование какого-либо объекта, находящегося в графическом окне. Например, если щелкнуть по линии графика функции, то на этой линии появятся специальные маркеры, означающие, что данная линия выбрана для редактирования (рис. 2.9).

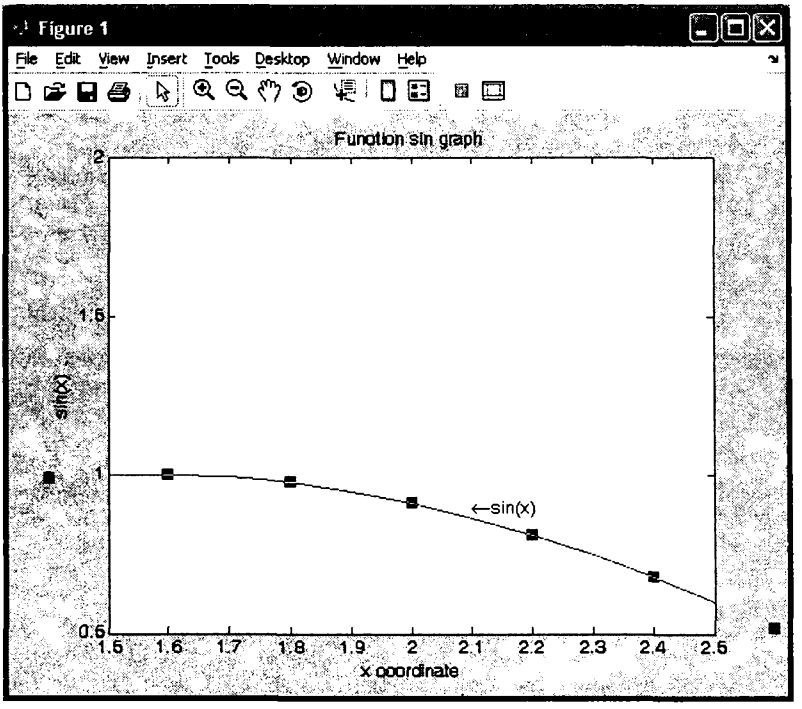


Рис. 2.9. График функции в режиме выбора кривой для последующего изменения ее свойств

После двойного клика левой кнопкой мыши по выделенной кривой открывается диалоговое окно для редактирования свойств графического объекта (2.10).

В данном окне можно задать стиль линии **(Line Style)**, толщину линии **(Line width)**, цвет линии **(Color)**, а также стиль маркеров. Вопрос выбора того или иного способа оформления графиков функций при работе в интерактивном режиме решается сугубо индивидуально. При создании m-файлов, реализующих сложные вычислительные проекты, особенно для имеющих графический интерфейс пользователя (см. главу 8), для задания параметров графических изображений целесообразно использовать соответствующие команды.

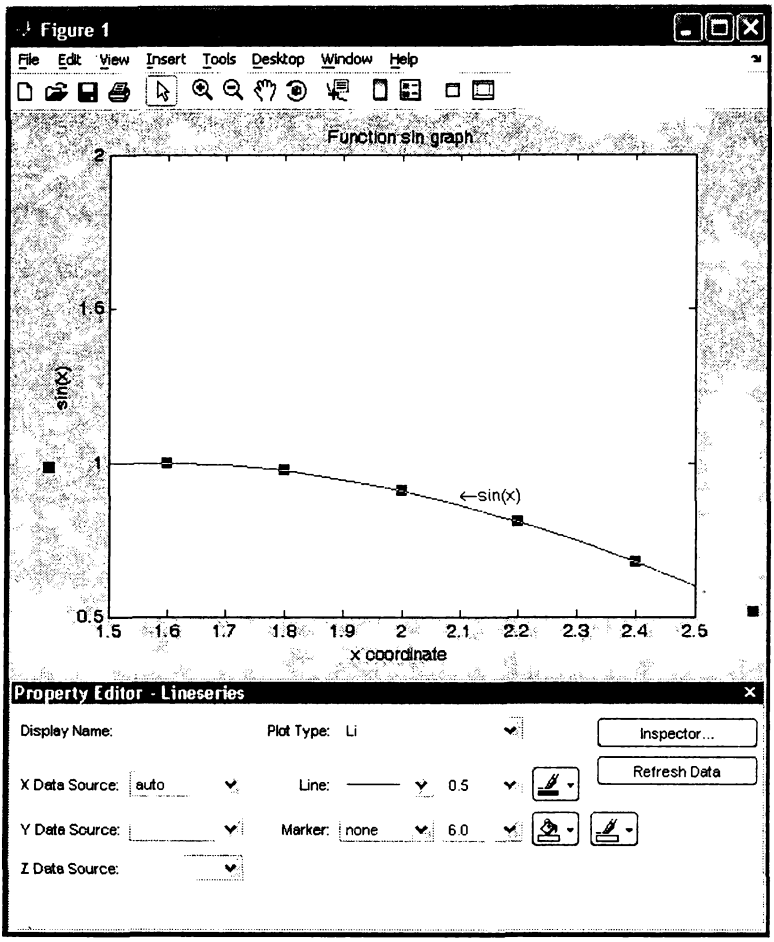


Рис. 2.10. Графическое окно в режиме редактирования свойств кривой

Для более детального просмотра отдельных частей графика, можно использовать инструментарий, включаемый нажатием кнопки . Далее следует осуществить протяжку мыши при нажатой левой кнопке, и заключить интересующую нас часть графика прямоугольник. Далее, отпустив левую клавиши мыши, получаем увеличенный масштаб изображения, находящегося в выбранном прямоугольнике.

При включенной , нажатие на левую кнопку мыши приводит к уменьшению масштаба изображения.

При нажатии на кнопку , включается режим вращения графика в трехмерном пространстве, которое осуществляется перемещением мыши при нажатой левой кнопке в области графика функции.

Кнопка , активирует режим курсора (измерения координат точек кривой). Для отображения на графике координат выбранной точки, следует подвести к ней указатель мыши и нажать левую кнопку.

Вопросы для самопроверки

1. Назовите основные элементы управления, размещаемые в графическом окне MATLAB.
2. Как осуществляется построение нескольких графиков в различных окнах?
3. Как осуществляется построение нескольких графиков в одном графическом окне?
4. Как отобразить несколько графических окон в одном графическом окне?
5. Какие средства можно использовать для оформления графиков в MATLAB?
6. Объясните понятие «дескрипторная графика».

Вычисления с действительными и комплексными массивами чисел

3.1. Операции отношения и логические операции над числами

В главе 1 были рассмотрены арифметические операции над операндами типа **double**. Помимо арифметических операций для операндов данного типа определены логические операции и операции отношения.

Операции отношения сравнивают между собой два операнда по величине. Способы записи этих операций представлены в таблице 3.1.

Таблица 3.1. Обозначения операций сравнения

<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
==	Равно
~=	Не равно

В случае истинности операции отношения ее величина (то есть результат вычисления выражения) равна 1, а в противоположном случае — 0. Пример вычисления выражений, содержащих операции отношения представлен на рис. 3.1:

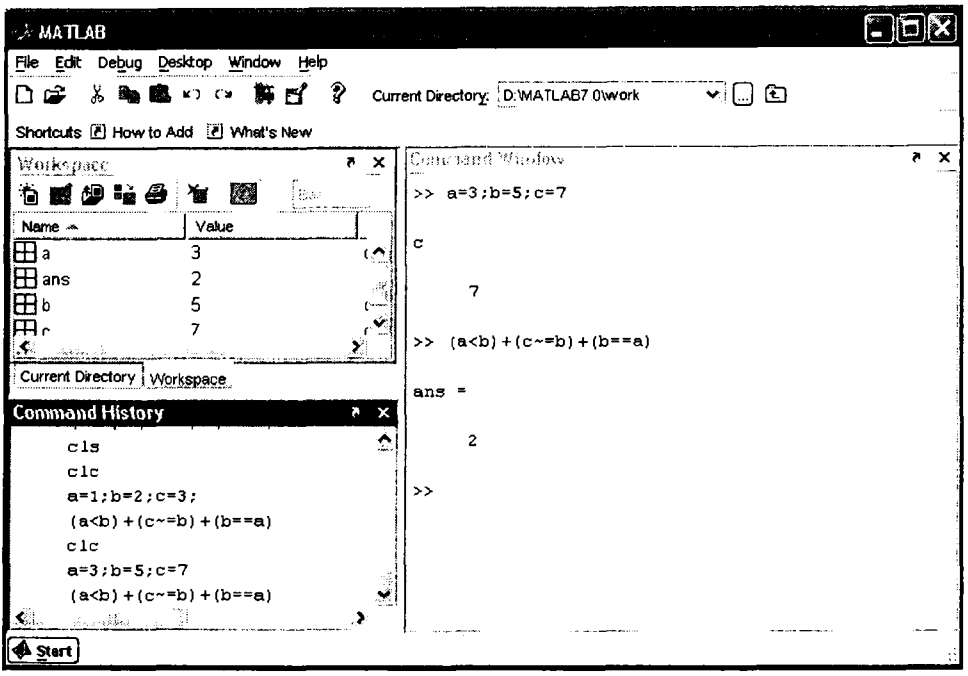


Рис. 3.1. Пример вычисления логического выражения

Здесь значение выражения **(a < b)** равно единице **(истина)** в силу того, что величина переменной **a** действительно меньше величины переменной **b**. Выражение **c ~= b** является истинным, так как **c**, равное 7, не равно **b**, которое равно двум. В итоге его значение оказывается равным 1. Последнее выражение, **b = = a**, не является истинным, поэтому его значение равно 0. В результате переменная **res**, являющаяся суммой значений этих трех выражений, оказывается равной двум.

Операции отношения имеют более низкий приоритет, чем арифметические операции, поэтому рассмотренная выше переменная **res** равна сумме значений трех операций отношения только потому, что мы заключили эти операции в круглые скобки. Очень важно всегда помнить об этом, так как отсутствие круглых скобок может привести к изменению результата. Например, если **a = 1, b = 1, c = 3**, то значение выражения **c + (b = = a)**, в то время как выражение без круглых скобок **a + b = = a** равно 0.

Логические операции над вещественными числами обозначаются знаками, перечисленными в следующей таблице 3.2.

Таблица 3.2. Обозначения логических операций

&	И
|	ИЛИ
~	НЕ

Первые две операции являются двухоперандными (бинарными), а операция «НЕ» является унарной (однооперандной). Знак ~ ставится перед операндом, а знаки & и | ставятся между операндами.

Логические операции трактуют свои операнды как «истинные» (не равные нулю) или «ложные» (равные нулю). Если оба операнда операции «И» истины (не равны нулю), то результат этой операции равен 1 («истина»); во всех остальных случаях операция «И» вырабатывает знание 0 (ложь). Операция «ИЛИ» возвращает 0 (ложь) только в случае, когда являются ложными (равными нулю) оба операнда. Наконец, операция «НЕ» инвертирует «ложь» на «истину» и наоборот. То есть, если ее операндом является ненулевое число, то эта операция возвращает 0, в если операнд нулевой, то тогда значением операции «НЕ» будет единица.

Логические операции (кроме операции отрицания, то есть операции «НЕ») имеют самый низший приоритет. Более точно приоритеты операций (в порядке убывания) MATLAB показаны ниже:

1. Круглые скобки «( )»
2. «.'», «.^», (') , (^)
3. Унарный плюс «+», унарный минус «−», «~»
4. «.*», «./», «.\», «*», «/», «\»
5. Сложение «+», вычитание «−»
6. «:»
7. «<», «<=», «>», «>=», «= =», «~=»
8. «&»
9. « | »

В одном выражении можно использовать все перечисленные выше операции: арифметические, логические и операции сравнения. Последовательность выполнения операций определяется их расположением внутри выражения, их приоритетом и наличием круглых скобок.

Рассмотрим логическую функцию **xor**, дополняющую ранее рассмотренный набор логических операций. Эта функция имеет два входных аргумента и вычисляет над ними операцию «исключающее ИЛИ», значение которой равно единице («истина») только в случае, когда один из числовых аргументов истинен (не равен нулю), а другой ложен (равен нулю):

```
>> a=1;b=0;
>> xor(a,b)
ans =
   1
>>
```

В том случае, когда оба аргумента «истинны» или оба «ложны», значение данной функции равно нулю:

```
>> a=1;b=1;
>> xor(a,b)
ans =
   0
>>
```

Так как комплексные числа можно складывать, вычитать, перемножать и делить точно также, как и вещественные числа, то в отношении арифметических операций ничего нового для комплексных чисел (и сравнению с вещественными) сказать нельзя. В том числе, это относится и к операциям отношения «равно» и «не равно».

В связи с тем, что операции сравнения комплексных чисел в математике не определены, нельзя, например, сказать, что одно комплексное число больше другого комплексного числа. Исходя из этого, можно было бы предположить, что такие операции сравнения для комплексных операндов запрещены. Однако разработчики MATLAB реализовали более «щадящий» вариант. Эти функции применимы и к комплексным операндам, но их значения зависят только от действительных частей этих операндов. Например,

```
>> c=2+3i;d=2i;
>> c>d
ans =
   1
>> c=5i
c =
   0 + 5.0000i
>> c<=d
ans =
   1
>> c>=d
ans =
   1
>>
```

Логические операции трактуют операнды как ложные, если они равны нулю. Когда у комплексного операнда не равна нулю хотя бы одна его часть (вещественная или мнимая), то такой комплексный операнд трактуется как истинный. Отсюда вытекают результаты следующих логических операций:

```
>> c=2+3i;d=2i;
>> ~c
ans =
   0
>> c&d
ans =
   1
>>
```

В итоге не требуется жесткого контроля со стороны пользователя за типом числовых операндов. В одном выражении разрешается смешивать вещественные и комплексные операнды. При необходимости в процессе вычислений формально определить, является ли переменная комплексной (речь идет о ее значении), следует использовать функцию

```
isreal(x)
```

возвращающую «истину», если числовая переменная x не является комплексной, и «ложь» в противном случае.

3.2. Формирование одномерных числовых массивов

Несмотря на то, что мы знакомились с основными способами создания одномерных массивов в главе 1, мы вновь вернемся к этому важному вопросу.

Массивы в MATLAB не образуют нового типа данных. Числовые массивы (вещественные или комплексные) состоят из элементов типа **double**. При этом в MATLAB даже переменные, принимающие единственное числовое значение, то есть являющиеся по существу скалярами, в своем внутреннем представлении являются массивами, состоящими из единственного элемента. Помимо памяти, необходимой для хранения собственно значений числовых элементов (по 8 байт на каждый в случае вещественных чисел и по 16 байт в случае комплексных чисел), при создании массивов MATLAB автоматически выделяет еще и память для управляющей информации. В этой области памяти хранится размерность массива, количество элементов по каждой размерности, тип элементов (вещественные или комплексные) и так далее. Отметим, что при построении массивов MATLAB не требует от пользователя предварительного задания размерности и размеров массива. Пользователь может вводить ее постепенно, при этом MATLAB будет динамически перестраивать структуру массива.

Как мы уже знаем, для создания одномерного массива используют операцию конкатенации. Данная операция обозначается с помощью квадратных скобок **[]**. Например, следующее выражение, использующее операцию конкатенации

```
z = [1 2 3]
```

формирует переменную с именем **z**, являющуюся одномерным массивом, состоящим из трех элементов (вещественных чисел). При использовании операции конкатенации объединяемые в одномерный массив элементы располагаются между открывающей и закрывающей квадратными скобками и отделяются друг от друга либо пробелом, либо запятой. Так что выражение

```
z = [1,2,3]
```

по своему результату абсолютно идентично предыдущему. Однако если массивы состоят из комплексных чисел или элементы задаются выражениями, то с точки зрения наглядности лучше использовать в качестве разделителя элементов запятую, как в следующем примере, в котором создается массив комплексных чисел:

```
d = [ 1+2i, 2+3i, 3-7i ];
```

Для доступа к отдельному элементу одномерного массива нужно применить операцию индексации: после имени массива указать в круглых скобках индекс (номер) элемента. В итоге третий элемент массива **z** обозначается как **z**(3), первый элемент — как **z**(1), второй элемент — как **z**(2).

При необходимости изменить третий элемент сформированного выше операцией конкатенации массива z можно применить операцию индексации и операцию присваивания;

```
z(3) = 789
```

Если, например, второй элемент массива z должен стать равным среднему арифметическому первого и третьего элементов, нужно выполнить следующую команду:

```
z(2)=(z(1) + z(3))/2
```

Количество элементов в одномерном массиве возвращает функция **length**:

```
>> length( z )
ans =
   3
```

Таким образом, операцию индексации можно применять как справа от знака операции присваивания, так и слева от него. Про эти случаи говорят, что осуществляется доступ к элементу массива «по чтению» или «по записи». При попытке чтения несуществующего элемента (например, четвертого элемента массива z) в командном окне появится сообщение об ошибке (см. рис. 3.2):

```
>> z=[1 2 3];
>> z(4)
??? Index exceeds matrix dimensions.
>>
```

В этом сообщении говорится, что индекс превысил размер массива. В тоже время запись несуществующего элемента вполне допустима, она означает добавление нового элемента к уже существующему массиву:

```
>> z(4)=7;
```

Применяя после выполнения этой операции к массиву z функцию **length**, находим, что количество элементов в массиве возросло до четырех:

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

Тоже самое действие — «удлинение» массива z, можно выполнить и с помощью операции конкатенации:

```
>> z=[1 2 3];
>> z=[z 123];
>> z
z =
    1   2   3   123
>>
```

Здесь операндами операции конкатенации являются массив **z**, состоящий из трех элементов, и добавляемый к нему четвертый элемент, равный 7. Можно подвергнуть конкатенации и несколько массивов. Например, следующий код

```
>> y=[z z 9 z]
y =
  Columns 1 through 7
    1    2    3  123    1    2    3
  Columns 8 through 13
  123     9     1     2     3   123
>>
```

порождает одномерный массив **y**, состоящий из тринадцати элементов: его первые четыре элемента повторяют элементы массива **z**, элементы с пятого по восьмой делают то же самое, девятый элемент равен числу 9 и, наконец, последние четыре элемента опять совпадают с соответствующими элементами массива **z**.

Можно создавать массивы и без использования операции конкатенации, указывая явно значение каждого элемента массива, например:

```
>> A(1)=67;
>> A(2)=7.8;
>> A(3)=0.017;
>> A
A =
 67.0000    7.8000    0.0170
>>
```

Описанное пошаговое создание массива из трех элементов возможно потому, что MATLAB с каждым новым присваиванием автоматически перестраивает свою служебную информацию о массиве, а также область памяти, отводимой под его элементы. После первого присваивания MATLAB считает, что массив **A** состоит из одного элемента. При втором присваивании MATLAB перестраивает всю структуру памяти, отведенную под данный массив. С каждым последующим присваиванием перестройку приходится повторять.

Ясно, что описанный способ создания одномерного массива не является эффективным и проигрывает в быстродействии операции конкатенации. Проигрыш в быстродействии мало заметен в интерактивном режиме, когда пользователь вводит всю информацию с клавиатуры. Однако это становится критичным в программном режиме, когда MATLAB подряд исполняет многочисленные инструкции с массивами.

В рассмотренной ситуации существуют два простых способа приблизительно в сто раз увеличить быстродействие работы MATLAB. Во-первых, можно предварительно выделить всю необходимую память под конечный размер массива. Это достигается вызовом функций **ones** или **zeros**, которые сразу создают массив нужного размера, заполненный единицами или нулями. После этого постепенное прописывание элементов нужными значениями не требует перестройки структуры памяти, отведенной под массив и, следовательно, позволяет сэкономить время. К примеру, для масси-

ва **A** можно перед присваиваниями сделать следующий вызов функции **ones**:

```
>> A=ones(1,3)
A =
  1     1     1
>>
```

где вывод в командное окно результата вызова функции **ones** показывает, что сразу создается массив из трех элементов, равных единице. После этого можно осуществить показанные выше присваивания нужных значений элементам массива **A**.

Во-вторых, можно осуществить присваивание значений элементам массива, начиная с последних по номеру элементов и заканчивая первым:

```
>> A=ones(1,3)
A =
  1     1     1
>> A(3)=0.017;
>> A(2)=7.8;
>> A(1)=67;
>>
```

Здесь при выполнении первого же присваивания система MATLAB выделяет память под три вещественных числа, присваивает указанное значение третьему элементу, второму и затем первому элементу.

Вновь вернемся к рассмотрению различных способов создания одномерных массивов. К настоящему моменту мы изучили три таких способа: операцию конкатенации, операцию индексации, вызов специальных функций (например, **ones** или **zeros**).

Еще один способ, как мы это уже знаем из Главы 1, основан на применении специальной операции, обозначаемой двоеточием. Эту операцию мы назвали «операцией формирования диапазона» числовых значений.

Вот пример, создающий одномерный массив чисел в диапазоне от 3.7 до 8.947 с приращением 0.3:

```
diap= -3.7:0.3:8.947;
```

Напомним, как работает «операция формирования диапазона». Сначала она включает в формируемый массив левую границу диапазона (это число, стоящее левее первого двоеточия). Затем к этому числовому значению прибавляется приращение, которое указывается после первого двоеточия. Если сумма не превосходит верхней границы диапазона (число, стоящее после второго двоеточия), то она включается в качестве элемента в формируемый массив. Это все повторяется до тех пор, пока очередное числовое значение не превысит верхнюю границу.

3.3. Двумерные массивы чисел: матрицы и векторы

В главе 1 мы кратко каснулись вопроса создания двумерных числовых массивов с помощью операции конкатенации. Ниже же мы рассмотрим и другие способы их создания, а также и другие важные вопросы, связанные с двумерными числовыми массивами.

Двумерные массивы в математике принято называть матрицами. Любая строка матрицы является одномерным массивом, и любой столбец матрицы также является одномерным массивом. Однако существует разница в упорядочении их элементов с точки зрения матриц: элементы первого одномерного массива упорядочены вдоль строк матрицы (горизонтально), а элементы второго вдоль столбцов (вертикально). Если явно учитывать в понятии одномерного массива эту разницу, то тогда массивы первого типа называют вектор-строками, а второго типа вектор-столбцами. Таким образом, также можно считать, что вектор-строки являются частным случаем матрицы с количеством строк, равным единице, а вектор-столбцы являются частным случаем матрицы с количеством столбцов, равным единице.

В MATLAB все одномерные массивы трактуются либо как вектор-строки, либо как вектор-столбцы. До сих пор мы вводили только вектор-строки, так как использовали в операциях конкатенации в качестве разделителей либо пробелы, либо запятые. Следующее выражение, использующее операцию конкатенации, задает уже вектор-столбец

```
b=[1;2;3]
```

состоящий из трех строк, так как точка с запятой в операции конкатенации означает переход на новую строку.

Для массива **b** функция **length(b)** возвращает число 3, так как данный массив состоит из трех элементов. При этом функция **length** не различает вектор-строки и вектор-столбцы.

Если попросить MATLAB показать значение переменной **b**, то мы получим:

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

Таким образом, MATLAB распознает «геометрию» этого одномерного массива и наглядно отображает его, располагая элементы массива **b** для показа в своем окне вертикально.

Матрицу **X** размером **3×2** (первым указывается число строк, вторым — число столбцов), получающуюся в результате операции конкатенации

```
>> X=[1 2;3 4;5 6]
X =
  1     2
```

```
   3     4
   5     6
```

можно сформировать также вертикальной конкатенацией векторов-строк

```
>> X=[[1 2]; [3 4]; [5 6]]
X =
   1     2
   3     4
   5     6
```

или горизонтальной конкатенацией векторов-столбцов:

```
>> X=[[1;3;5],[2;4;6]]
X =
   1     2
   3     4
   5     6
```

Вертикальную и горизонтальную конкатенации можно также осуществить с помощью функции **cat**. Для вертикальной конкатенации ее первый параметр равен единице

```
>> X=cat(1,[1 2],[3 4],[5 6])
X =
   1     2
   3     4
   5     6
```

а для горизонтальной конкатенации он равен двум:

```
>> X=cat(2,[1;3;5],[2;4;6])
X =
   1     2
   3     4
   5     6
```

Для того, чтобы узнать размеры двумерного массива и «геометрию» векторов (вектор-столбцы или вектор-строки), нужно использовать функцию size, например,

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

где первым показывается число строк, а вторым — число столбцов.

Теперь применим эту функцию к одномерным массивам.

```
>> A=ones(1,3)
A =
   1     1     1
>> size(A)
ans =
   1     3
>> A=ones(3,1)
A =
   1
```

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

Наконец, попробуем применить эту функцию к переменной, состоящей из единственного числового значения, то есть к скаляру:

```
>> var=5;
>> size(var)
ans =
    1     1
```

Следовательно MATLAB трактует даже скалярные, по существу, величины как двумерные массивы с размером **1×1**. При этом векторы рассматриваются как матрицы, размер которых по одному из направлений равен единице.

В MATLAB также существует пустой массив, то есть массив, не содержащий данных. Он обозначается квадратными скобками **[ ]** (между которыми нет операндов) и трактуется как матрица размером **0×0**.

```
>> a=[ ]
a =
[ ]
>> size(a)
ans =
    0     0
```

Для нахождения размерности переменной используется функция **ndims**

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

Итак все переменные, с которыми работает MATLAB, является массивами различной размерности и размеров. Размерность массива можно узнать, обратившись к функции **ndims**, а размеры — к функции **size**. Тип массива определяется типом его элементов. В приведенных выше примерах мы использовали числовые массивы типа **double**, элементы которых — вещественные или комплексные числа (точнее, их приближенные машинные представления).

Продолжим рассмотрение способов создания двумерных числовых массивов (матриц). Как и рассмотренные ранее одномерные массивы (векторы), двумерные массивы можно создать с помощью операции индексации, прописывая по отдельности его элементы необходимыми числовыми значениями. Например, рассмотренный ранее массив **x** можно создать следующим образом

```
>> X(1,1)=1;X(1,2)=2;
>> X(2,1)=3;X(2,2)=4;
>> X(3,1)=5;X(3,2)=6;
>> X
X =
```

```
    1     2
    3     4
    5     6
```

где для доступа («чтению») к отдельным элементам используются круглые скобки (операция индексации), внутри которых через запятую перечисляются индексы. Здесь первым указывается номер строки, вторым — номер столбца.

Как и в случае одномерных массивов, это решение является неэффективным, так как по мере присваивании MATLAB приходится перестраивать структуру массива. Проблема легко преодолевается, если присваивание

```
>> X(3,2)=6;
```

поместить первым. Кроме того, можно сразу создать двумерный массив нужного размера функциями **ones** или **zeros**:

```
>> ones (3,2)
```

или

```
>> zeros (3,2)
```

а затем осуществить присваивания отдельным элементам нужных значений (причем порядок присваивании в этом случае уже не имеет значения). У этих функций первый параметр задает число строк, а второй число столбцов.

И, наконец, если после формирования массива **x** потребуется, не изменяя элементов массива, изменить его размеры, можно воспользоваться функцией

```
reshape( X, M, N )
```

где **M**×**N** — новые размеры массива x (**M** — число строк, **N** — число столбцов). Если количество элементов в массиве **x** не равно произведению **M** на **N**, то MATLAB выдаст сообщение об ошибке.

К примеру, если требуется ранее сформированную матрицу **X** размером 3×2 превратить в матрицу размером 2×3, то следует использовать функцию **reshape**:

```
>> X
X =
   1     2
   3     4
   5     6
>> reshape(X,2,3)
ans =
   1     5     4
   3     2     6
```

Для объяснения работы данной функции рассмотрим способ хранения элементов массива в памяти компьютера. Они хранятся в непрерывной области памяти, при этом упорядочение осуществляется по столбцам: снача-

ла располагаются элементы первого столбца, вслед за ними расположены элементы второго столбца и так далее. Помимо собственно элементов массива в памяти компьютера хранится также управляющая информация: тип массива (например, **double**), размерность и размеры массива и другая служебная информация. Этой информации достаточно для определения границ столбцов. Следовательно, для переформирования матрицы функцией **reshape** достаточно изменить только служебную информацию и не трогать собственно данные.

Поменять местами строки матрицы с ее столбцами можно операцией транспонирования, которая обозначается знаком ' (апостроф). Например,

```
>> A=[1 1 1;2 2 2;3 3 3]
A =
   1     1     1
   2     2     2
   3     3     3
>> B=A
B =
   1     2     3
   1     2     3
   1     2     3
```

Для квадратных матриц (число строк равно числу столбцов) эта операция имеет наглядную геометрическую интерпретацию: на своих местах остаются элементы главной диагонали квадратной матрицы, а остальные «отражаются симметрично» относительно этой диагонали.

Вектор-строки операцией транспонирования преобразуются в вектор-столбцы, и наоборот:

```
>> v=[1;2;3];
>> u=v'
>> u
u =
   1     2     3
```

В рассмотренном примере вектор-столбец **v** операцией транспонирования был преобразован в вектор-строку **u**.

3.4. Вычисления с массивами

В традиционных языках программирования вычисления с массивами осуществляются поэлементно в том смысле, что нужно запрограммировать каждую отдельную операцию над отдельным элементом массива. В М-языке MATLAB допускаются групповые операции над всем массивом сразу. Именно групповые операции MATLAB позволяют компактно задавать выражения, при вычислении которых реально выполняется гигантский объем вычислений. Однако практическое использование данных возможностей MATLAB, в свою очередь, требует от пользователя, работавшего ранее с каким-либо из языков программирования, определенной перестройки стиля мышления.

Начнем обсуждение вычислений с массивами с арифметических операций. Над массивами одинаковых размеров допускаются операции сложения и вычитания, обозначаемые стандартными знаками + и −. Если **A** и **B** — массивы любой размерности, но одинаковых размеров, то допустимы следующие выражения

```
C = A+B;
D = A-B;
```

где элементы массивов **C** и **D** равны сумме или разности соответствующих элементов массивов **A** и **B**. Таким образом, эти операции выполняются поэлементно и порождают массивы тех же размеров, что и исходные операнды. Например,

```
>> A=[1 1 1; 2 2 2;3 3 3];
>> B=[0 0 0;7 7 7; 1 2 3];
>> A+B
ans =
  1     1     1
  9     9     9
  4     5     6
```

Если используются операнды разных размеров, выдается будет выдано следующее ошибке

```
>> A=[1 2 3;4 5 6]
A =
  1     2     3
  4     5     6
>> B=[7 8;9 8]
B =
  7     8
  9     8
>> A+B
??? Error using ==> +
Matrix dimensions must agree.
```

за исключением случая, когда один из операндов является скаляром:

```
>> A+5
ans =
  6     7     8
  9    10    11
```

В таких случаях скаляр предварительно расширяется до массива размером с матричный операнд. Например, из скаляра 5 сначала генерируется матрица [5 5 5; 5 5 5], которая и складывается далее поэлементно с матрицей **A**.

Для поэлементного перемножения и поэлементного деления массивов одинаковых размеров применяются операции, обозначаемые комбинациями двух символов: «.*» и «./» (кавычки не входят в состав обозначений для указанных операций). Использование комбинаций символов объясняется тем, что символами «*» и «/» обозначены специальные операции линейной алгебры над векторами и матрицами.

Кроме операции «./», называемой операцией правого поэлементного деления, есть еще операция левого поэлементного деления «.\». При этом выражение **A./B** приводит к матрице с элементами **A(k,m)/B(k,m)**, а выражение **A.\B** приводит к матрице с элементами **B(k,m)/A(k,m)**.

Знак «*» (кавычки не входят в обозначение операции) закреплен за перемножением матриц и векторов в смысле линейной алгебры. Отметим, что данная операция выполнима только при том условии, что число столбцов в левом операнде равно числу строк в правом операнде:

```
>> A=[1,-2,3,-1;2,3,-4,4;3,1,-2,-2;1,-3,7,6]
A =
   1    -2     3    -1
   2     3    -4     4
   3     1    -2    -2
   1    -3     7     6
>> b=[6;-7;9;-7]
b =
   6
  -7
   9
  -7
>> A*y
ans =
   6.0000
  -7.0000
   9.0000
  -7.0000
```

Традиционный для операции деления знак «/» (а также знак «\») закреплен в MATLAB за решением известной задачи линейной алгебры — нахождением корней систем линейных уравнений. Например, если требуется решить систему линейных уравнений

```
Ay=b,
```

где **A** — заданная квадратная матрица размера **M×N**, **b** — заданный вектор-столбец длины **N**, то для нахождения неизвестного вектор-столбца y (неизвестны его элементы) достаточно вычислить выражение **A\b**. Например:

```
>> A=[2,0,1,2;2,0,2,7;3,9,3,3;4,3,4,0]
A =
   2     0     1     2
   2     0     2     7
   3     9     3     3
   4     3     4     0
>> b=[6;-7;9;-7]
b =
   6
  -7
   9
  -7
>> A\b
ans =
   9.4348
```

```
  2.1304
 -12.7826
 -0.0435
```

Операцию, обозначаемую знаком «/», мы рассмотрим позже в главе 5, посвященной решению задач линейной алгебры.

Типичные задачи аналитической геометрии в пространстве, связанные с нахождением длин векторов и углов между ними, с вычислением скалярного и векторного произведений, легко решаются разнообразными средствами MATLAB. Например, для нахождения векторного произведения предназначена специальная функция cross:

```
>> u=[1 2 3];
>> v=[3 2 1];
>> cross(u,v)
ans =
 -4     8    -4
```

скалярное произведение векторов вычисляется с помощью функции общего назначения **sum**, вычисляющей сумму всех элементов векторов (для матриц эта функция вычисляет суммы для всех столбцов). Так как скалярное произведение, равно сумме произведений соответствующих координат (элементов) векторов, то для его вычисления можно использовать следующую команду:

```
>> sum(u.*v)
ans =
  10
```

Для вычисления скалярного произведения также можно использовать функцию **dot**:

```
>> dot(u,v)
ans =
  10
```

Длина вектора вычисляется с помощью скалярного произведения и функции извлечения квадратного корня:

```
>> len1=sqrt(sum(u.*v))
len1 =
  3.1623
```

или функции **norm**:

```
>> len1=sqrt(sum(u.*u))
len1 =
  3.7417
>> norm(u)
ans =
  3.7417
```

Угол между векторами легко вычисляется на основе определения скалярного произведения, в соответствии с которым оно равно произведению

длин векторов на косинус угла между ними. Отсюда находим выражение для вычисления угла между ранее заданными векторами **u** и **v**:

```
>> phi=acos(dot(u,v)/(len1*len2));
>> phi
phi =
  0.7752
```

Ранее рассмотренные для скаляров операции отношения и логические операции выполняются в случае массивов поэлементно, поэтому размеры обеих операндов должны быть одинаковы.

В том случае, когда один из операндов скаляр, производится его предварительное расширение, смысл которого уже был пояснен на примере арифметических операций. Для иллюстрации выполним над матрицами операцию «меньше или равно»

```
>> A=[1 1 1;2 2 2;3 3 3]
A =
  1     1     1
  2     2     2
  3     3     3
>> B=[0 0 0;7 7 7; 1 2 3]
B =
  0     0     0
  7     7     7
  1     2     3
>> A<=B
ans =
  0     0     0
  1     1     1
  0     0     1
```

Анализ результата, возвращенного этой операцией, показывает, что каждый нуль означает «ложь» для данной позиции внутри матриц, а единица означает «истину». Полученная матрица показывает (своими единичными элементами), в каких позициях элементы матрицы **A** на самом деле меньше или равны соответствующим элементам матрицы B.

Работу логических операций над массивами проиллюстрируем на примере операции «НЕ»:

```
>> v=[1 2 3 4 0]
v =
  1     2     3     4     0
>> ~v
ans =
  0     0     0     0     1
```

Выше при изучении вычислений с вещественными скалярами мы рассматривали работу логической функции **xor** («исключающее ИЛИ»). Данная функция работает и с массивами одинаковых размеров, поэлементно реализуя операцию «исключающее ИЛИ». Напомним, что каждый элемент трактуется как истинный, если он не равен нулю, и как ложный в случае его равенства нулю. Покажем результат работы этой функции над ранее заданными матрицами **A** и B:

```
>> xor(A,B)
ans =
  1     1     1
  0     0     0
  0     0     0
```

Другими логическими функциями (помимо функции **xor**) являются функции **all** и **any**. Функция **all** в случае векторов возвращает единицу («**истину**»), если все элементы вектора не равны нулю («**истинны**»), и возвращает 0, когда хотя бы один элемент вектора ненулевой. Функция **any** действует противоположным образом.

В случае матриц обе эти функции работают с их столбцами, возвращая для каждого столбца результат по описанной выше схеме. Например,

```
>> all(A)
ans =
  1     1     1
>> all(B)
ans =
  0     0     0
>>
```

Существует множество различных функций, специально предназначенных для работы с массивами. Таковыми являются ранее рассмотренные функции **ones**, **zeros**, **sum**, **cat**, **size**, **ndims**, **reshape** и многие другие. Часть из этих функций сообщает служебную информацию о массивах, другая группа функций обеспечивает контролируемое изменение их структуры, третья группа предназначена для генерации массивов с заданными свойствами и так далее.

Среди функций, генерирующих матрицы с заданными свойствами, упомянем здесь функцию **eye**, возвращающую единичные квадратные матрицы, а также широко применяемую на практике функцию **rand**, генерирующую массив со случайными элементами, равномерно распределенными на интервале от 0 до 1. Например, следующее выражение

```
A=rand( 3 )
```

порождает массив случайных чисел размером **3×3** с элементами, равномерно распределенными на интервале от 0 до 1. Если вызвать эту функцию с двумя аргументами, например,

```
R=rand(2,3)
```

то будет возвращена матрица **R** случайных элементов размером **2×3**. При вызове функции **rand** с тремя и более скалярными аргументами производятся многомерные массивы случайных чисел.

Среди функций, производящих простейшие вычисления над массивами, помимо рассмотренной выше функции **sum** упомянем еще функцию **prod**, которая вычисляет она произведение элементов столбцов матрицы. К примеру, для матрицы

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

```
    2     2     2
    3     3     3
```

она возвращает следующий результат:

```
>> prod(A)
ans =
    6     6     6
```

Функции **max** и **min** ищут соответственно максимальный и минимальный элементы массивов. Для векторов они возвращают единственное числовое значение, а для матриц они порождают набор, соответственно, максимальных или минимальных элементов, вычисленных для каждого столбца, например:

```
>> max(A)
ans =
    3     3     3
```

Функция **sort** сортирует в возрастающем порядке элементы одномерных массивов, а для матриц она производит такую сортировку для каждого столбца отдельно.

3.5. Функции, выполняющие битовые операции

Рассмотрим вычисления, имеющие дело с отдельными битами двоичных представлений целых чисел. Для получения двоичного представления любого целого числа используется функции **dec2bin**:

```
>> dec2bin(12)
ans =
1100
```

Для обратного преобразования из двоичного представления числа в десятичное представление используется функция **bin2dec**:

```
>> bin2dec('1000')
ans =
    8
```

Для получения шестнадцатеричного представления десятичного числа используется функция **dec2hex**

```
>> dec2hex(193)
ans =
C1
```

Обратное преобразование выполняется функцией **hex2dec**

```
>> hex2dec('C1')
ans =
  193
```

Битовые операции над целыми положительными числами (они выполняются только для таких чисел) в двоичных представлениях операндов выполняются с отдельными двоичными разрядами, причем более короткий операнд достраивается слева нулями (чтобы получить одинаковое количество разрядов у обоих операндов). Отметим, что перечисленные выше функции преобразования представлений чисел могут работать не только с одиночными целыми числами, но и с соответствующими массивами:

```
>> A=[10,20]
A =
  10    20
>> dec2bin(A)
ans =
01010
10100
```

В случае входных матриц, состоящих из целых чисел, функции преобразования обрабатывают их элементы по столбцам. Сначала идут элементы первого столбца, затем — второго и так далее:

```
>> B=[1 2 3;4 5 6]
B =
  1   2   3
  4   5   6
>> dec2bin(B)
ans =
001
100
010
101
011
110
```

Три основные битовые операции — «битовое И», «битовое ИЛИ» и «битовое ИСКЛЮЧАЮЩЕЕ ИЛИ» выполняются, соответственно, функциями с **bitand**, **bitor** и **bitaxor**. Данные функции выполняют стандартные математические операции «И», «ИЛИ», «ИСКЛЮЧАЮЩЕЕ ИЛИ» по отдельности над каждой парой битов своих операндов. Например, если младшие биты операндов равны, например, 1 и 0, то для битовой операции «И» в этом разряде в качестве результата будет 0 («ложь»), для «ИЛИ» результатом будет 1 («истина»), для «ИСКЛЮЧАЮЩЕГО ИЛИ» — результатом будет 0 («ложь», так как операнды разные). Подробное обсуждение особенностей работы логических операций проведено выше в разделе 3.1. Отметим, что в отличие от последних, битовые (они тоже логические) операции синтаксически выполняются с помощью соответствующих функций, обращающихся к каждой паре двоичных разрядов своих операндов.

```
>> dec2bin([12,56])
ans =
  001100
```

```
   111000
>> bitand(12,56)
ans =
   8
>> bitor(12,56)
ans =
   60
```

Работу данных функций иллюстрирует табл. 3.3., в которой представлены результаты последовательного применения операнда «битовое ИСКЛЮЧАЮЩЕЕ ИЛИ» к числу 8.

Таблица 3.3. Пример последовательного применения операндов «битовое ИСКЛЮЧАЮЩЕЕ ИЛИ»

	Двоичное представление	Десятичное представление					
1-й операнд	0	0	1	1	0	0	8
2-й операнд	1	1	1	0	0	0	60
Результат	1	1	0	1	0	0	52

Для получения значение бита аргумента в заданной позиции используется функция **bitget.** Например, для получения значения четвертого бита в двоичном представлении переменной A следует выполнить команду:

```
>> bitget(A,4)
```

Для задания значения выбранного бита числа используется функция **bitset**, имеющая следующий синтаксис

```
bitset(имя_переменной, номер байта, значение байта)
```

Например, для того чтобы установить 4-й бит целого положительного числа A в нуль **следует выполнить команду**

```
>> bitset(A,4,0)
```

Отметим, что если переменная A — массив целых чисел, то в нуль будет установлен четвертый бит каждого элемента массива.

Вопросы для самопроверки

1. Какие операции отношения определены над числами в MATLAB?
2. Какие логические операции над вещественными числами определены в MATLAB?
3. Назовите иерархию приоритетов в выполнении математических операций, операций отношения и логических операций.
4. Как в MATLAB осуществляется формирование одномерных числовых массивов?
5. Как в MATLAB осуществляется формирование двумерных числовых массивов?
6. В каком виде осуществляется хранение массивов в рабочем пространстве MATLAB?
7. Какие функции используются для выполнения побитовых операций?

3D визуализация

4.1. Трехмерная графика

Возможности отображения трехмерных графических объектов в MATLAB весьма обширны. В частности, имеется возможность решать разнообразные задачи трехмерного моделирования объектов реального мира, опираясь на графические объекты типа **patch**. Это исключительно обширный раздел современной компьютерной графики, для детального обсуждения которых требуется отдельная книга, поэтому далее мы ограничиваемся рассмотрением функций, позволяющих создавать изображения пространственных линий и поверхности, описываемые функциями двух вещественных переменных.

Так как положение каждой точки в пространстве задается тремя координатами (например, в декартовой системе координат — (x,y,z)), набор точек, принадлежащих некоторой линии в пространстве, задается в виде трех векторов, которые содержат координаты точек по соответствующим координатным осям. Имена данных векторов указываюся в списке формальных параметров функции **plot3**, которая осуществляет проектирование соответствующей трехмерной линии на плоскость и строит результирующее изображение. Отметим, что все необходимые расчеты и выбор параметров (пределы изменения переменных по осям координат, число графических и числовых меток, выбор цвета линии и фона и т.д.) отображаемой зависимости будут сделаны автоматически в соответствие с общей концепцией высокоуровневой графики MATLAB.

Например, для построения пространственной спирали необходимо выполнить следующую последовательность команд:

```
>> t=0:pi/50:10*pi; % задание вектора, содержащего упорядоченные
                    % значения параметров
>> x=sin(t); % вычисление значений координат точек кривой по оси Ox
>> y=cos(t); % вычисление значений координат точек кривой по оси Oy
>> plot3(x,y,t); grid on % визуализация пространственной кривой
                                                        (рис. 4.1)
```

Функцию также **plot3** можно использовать и для изображения поверхностей в пространстве для этого, в отличие от предыдущего случая, в списке формальных параметров указываются три матрицы одинакового размера. Далее высокоуровневая графическая подсистема MATLAB автомати-

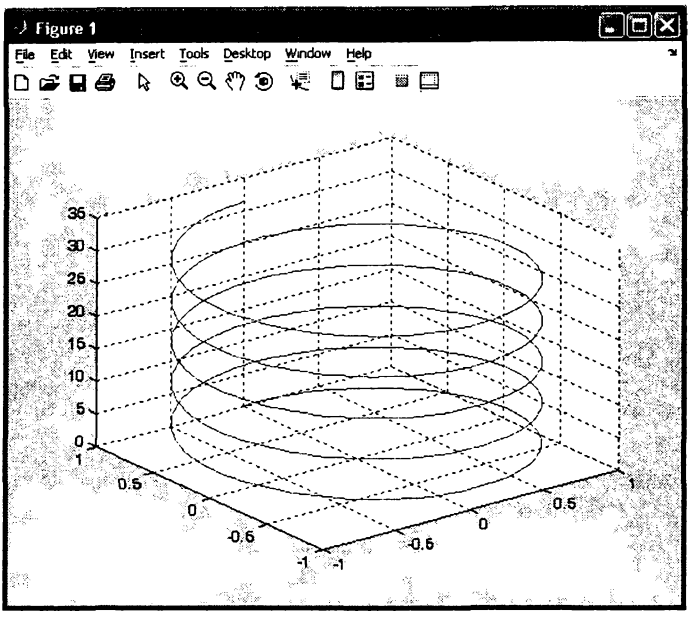


Рис. 4.1. Пример построения пространственной кривой

чески реализует трехмерную графику без специальных усилий со стороны пользователя.

Рассмотрим более подробно способ задания матриц, используемых для визуализации поверхности. Пусть в точке с координатами **xl**, **yl** вычислено значение функции $z = f(x,y)$ и оно равно **zl**. В некоторой другой точке (то есть при другом значении аргументов) **x2**, **y2** вычисляют значение функции **z2**. Продолжая этот процесс, получают массив точек **(xl,yl,zl)**, **(x2,y2,z2)**, ... **(xM,yM,zM)** трехмерного пространства. Соответствующие функции MATLAB проводят через эти точки гладкую поверхность и отображают ее проекцию на экран монитора компьютера. Наиболее часто для вычисления аргументов используется разбиение области определения функции в виде прямоугольной сетки (рис. 4.2).

Для описания данной сетки в MATLAB необходимо создать две матрицы, которые мы обозначим, соответственно, **X**, **Y**. Число строк данных матриц равно числу узлов разбиения отрезка $[y_{min},y_{max}]$, а число столбцов — числу узлов разбиения отрезка $[x_{min},x_{max}]$. При этом строки матрица **X** состоит из одинаковых, строк, содержащих координаты узлов разбиения отрезка $[x_{min},x_{max}]$, а матрица **Y** — из одинаковых столбцов, содержащих координаты узлов разбиения отрезка $[y_{min},y_{max}]$. Для формирования данных матриц используется функция **meshgrid**, имеющая следующий синтаксис:

```
[X Y] = meshgrid(x,y),
```

где **x**, y — векторы, содержащие координаты узлов сетки по осям *Ox* и *Oy*, соответственно.

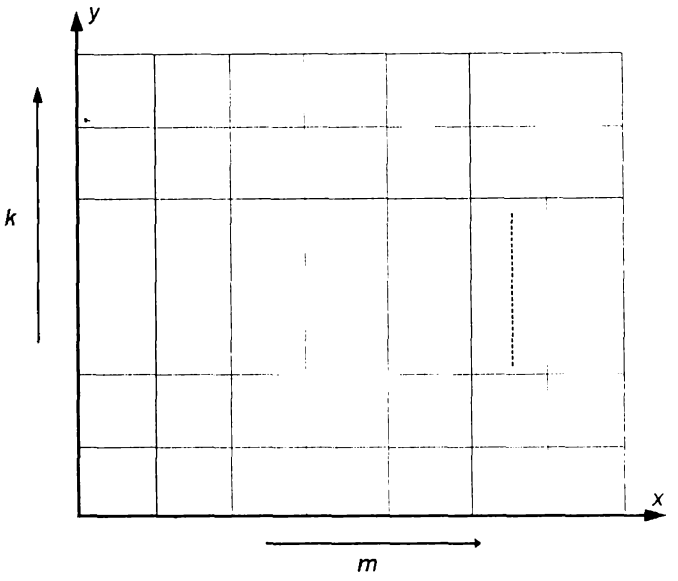


Рис. 4.2. Координатная сетка, используемая при визуализации зависимостей вида $z = f(x,y)$

Далее в каждом узле прямоугольной сетки (k,m) вычисляются значения функции $z = f(x,y)$ и сохраняются в матрице **Z**.

Для построения поверхности в виде линий, получаемых в сечении поверхности плоскостями параллельными плоскости yOz, также как и в предыдущем случае используется функция **plot3(X,Y,Z)**. Например, для построения поверхности, задаваемой функцией $f(x,y) = e^{-x^2-y^2}$, необходимо выполнить следующую последовательность команд:

```
% задаем координаты узлов пространственной сетки
>> x=-2:0.1:2;
>> y=-1:0.1:1;
>> [X Y]=meshgrid(x,y);   создаем матрицы, содержащие координаты
                          узлов пространственной сетки
>> Z=exp(-X.^2-Y.^2);
>> plot3(X,Y,Z); % построение пространственной фигуры (рис. 4.3)
>> figure
>> hSl=mesh(X,Y,Z);   построение пространственной фигуры (рис. 4.4)
>> figure
>> surf(X,Y,Z);   построение пространственной фигуры (рис. 4.5)
>> figure
>> surfl(X,Y,Z);   построение пространственной фигуры (рис.4.6)
```

Помимо этой простейшей функции пакет MATLAB располагает еще рядом функций, позволяющих добиваться большей реалистичности в изображении трехмерных графиков. Это функции **mesh**, **surf** и **surfl**. Эти функции порождают графические объекты типа surface. Изображения, создаваемые данными функциями, представлены на рис. 4,4–4.6.

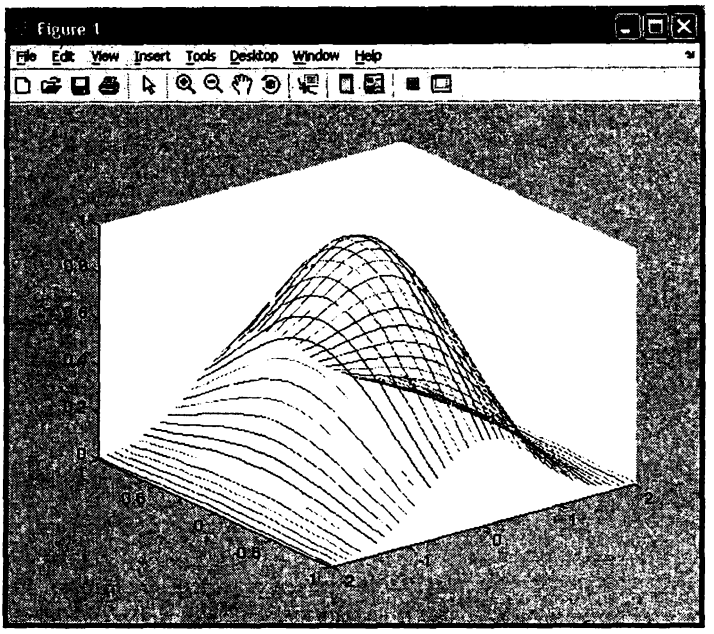


Рис. 4.3. Визуализация поверхности, описываемой функцией $f(x,y) = e^{-x^2-y^2}$, с использованием функции **plot3**

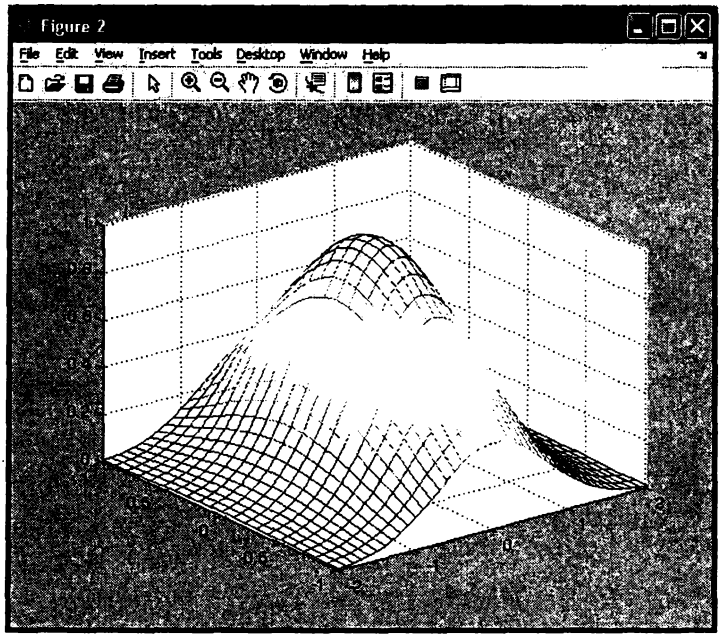


Рис. 4.4. Визуализация поверхности, описываемой функцией $f(x,y) = e^{-x^2-y^2}$, с использованием функции **mesh**

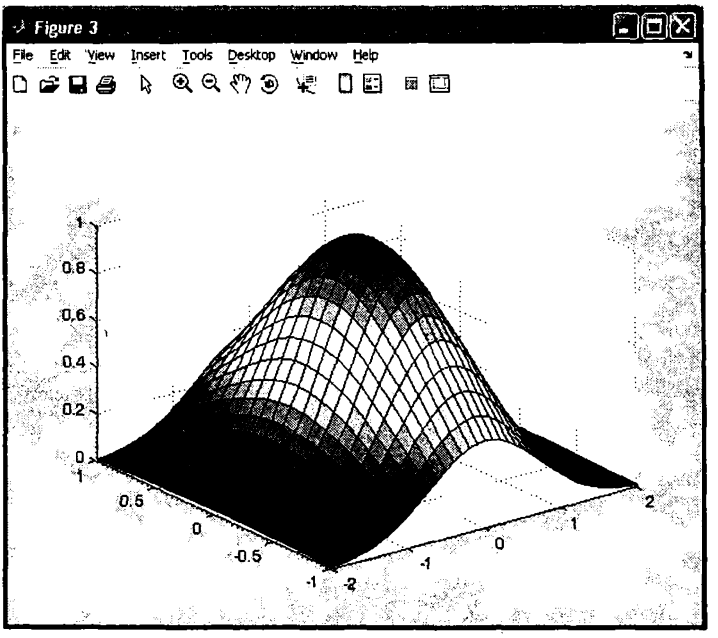


Рис. 4.5. Визуализация поверхности, описываемой функцией $f(x,y) = e^{-x^2-y^2}$, с использованием функции **surf**

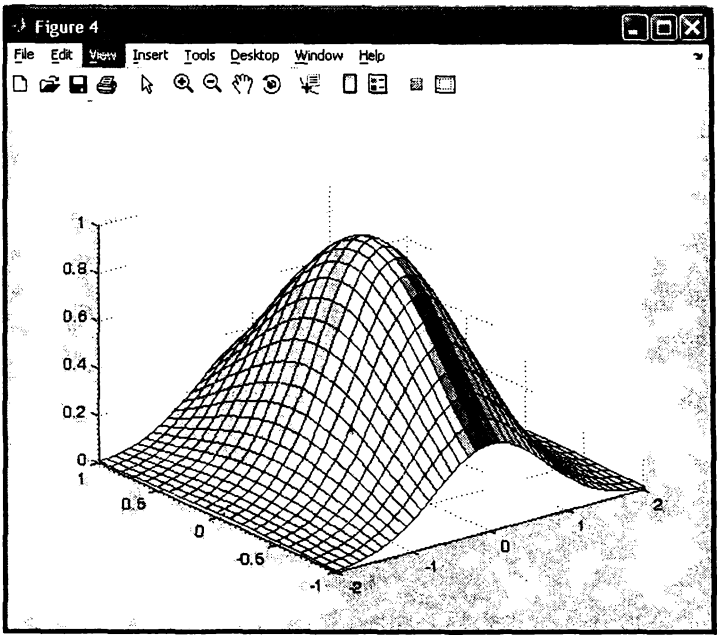


Рис. 4.6. Визуализация поверхности, описываемой функцией $f(x,y) = e^{-x^2-y^2}$, с использованием функции **surfl**

Отметим, что для лучшего восприятия «объемности изображения» разные ребра автоматически окрашиваются в разные цвета, кроме того (в отличие от функции **plot3**), осуществляется удаление невидимых линий поверхности. При необходимости отобразить задние ребра, следует выполнить команду

```
hidden off
```

Для выключения режима отображения задних ребер следует выполнить команду

```
hidden on
```

В тех случаях, когда требуется изобразить отдельные ребра переменным цветом, плавно изменяющимся при возрастании или убывании функции (по умолчанию цвет всех точек отдельного ребра одинаковый) используют команду

```
shading interp
```

Данная команда изменяет одно из свойств графического объекта типа **surface**, создаваемого функцией **mesh**. Дескриптор данного объекта, возвращенный функцией **surf**, выше был занесен переменную **hS1**. Для получения информации о типе графического объект используется команда **get**:

```
>> get( hS1, 'Type' )
ans =
   surface
```

Отметим, что можно напрямую воздействовать на свойство **EdgeColor** построенного объекта типа **surface**, используя команду **shading interp**:

```
set( hS1, 'FaceColor', 'interp' )
```

Получить информацию о всех свойствах графического объекта типа **surface** можно следующим образом:

```
>> hS1=mesh(X,Y,Z)
hs1 =
  3.0052
>> get(hs1)
       AlphaData = [1]
       AlphaDataMapping = scaled
       CData = [ (21 by 41) double array]
       CDataMapping = scaled
       EdgeAlpha = [1]
       EdgeColor = flat
       EraseMode = normal
       FaceAlpha = [1]
       FaceColor = [1 1 1]
       LineStyle = -
       LineWidth = [0.5]
       Marker = none
       MarkerEdgeColor = auto
       MarkerFaceColor = none
       MarkerSize = [6]
```

```
MeshStyle = both
XData = [ (21 by 41) double array]
YData = [ (21 by 41) double array]
ZData = [ (21 by 41) double array]
FaceLighting = none
EdgeLighting = flat
BackFaceLighting = reverselit
AmbientStrength = [0.3]
DiffuseStrength = [0.6]
SpecularStrength = [0.9]
SpecularExponent = [10]
SpecularColorReflectance = [1]
VertexNormals = [ (21 by 41 by 3) double array]
NormalMode = auto
BeingDeleted = off
ButtonDownFcn =
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [100.001]
Selected = off
SelectionHighlight = on
Tag =
Type = surface
UIContextMenu = []
UserData = []
Visible = on
```

Можно трактовать поверхность графика как *материальную поверхность* с определенными физическими свойствами по отражению света, которые следует задать явно. Так как разные материалы по-разному отражают падающие лучи, то можно подобрать некоторый материал, чтобы получить наилучшее (с точки зрения пользователя) изображение. В частности, использовав функцию с аргументом **copper**

```
colormap(copper),
```

можно установить цвет поверхности, аналогично цвету медной поверхности (медь по-английски — **copper**). После этого применение функции

```
surfl( X, Y, Z )
```

позволяет получить реалистически выглядящий и очень наглядный график (рис. 4.7).

Для удаления с поверхности черных линий, изображающих ребра, а также получения плавного перехода цветов на освещенной поверхности, используется команда

```
shading interp
```

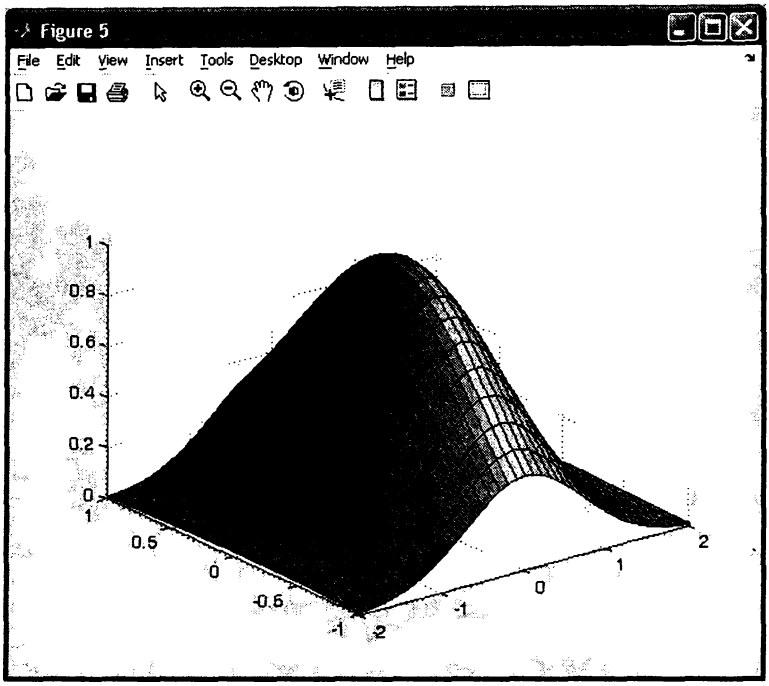


Рис. 4.7. Визуализация поверхности, описываемой функцией $f(x,y) = e^{-x^2-y^2}$, с использованием функции **surfl** (тип цветовой палитры — cooper)

Результат выполнения данной команды представлен на рис. 4.8.

Для проявления отдельных участков поверхности график в MATLAB предусмотрена возможность включать дополнительные источнике освещения, реализованные в виде графических объектов типа **light**. Свет от таких источников смешивается с рассеянным фоновым освещением (**Ambientlight**), цвет и сила которого встроены в виде характеристик в объекты **axes** (свойство **AmbientLightColor**) и **surface** (свойство **Ambientlight**).

В следующем примере

```
>> hS=surfl(X,Y,Z)
hS =
   3.0077
>> set(hS,'FaceLighting','phong','FaceColor','interp');
>> set(hS,'AmbientStrength',0.5);
>> ligth('Position',[1 0 0],'Style','infinite');
>> light('Position',[1 0 0],'Style','infinite');
```

задаются такие свойства поверхности (свойства графического объекта типа surface), которые влияют на результирующее изображение при использовании дополнительного источника света. После этого создается такой источник в позиции, определяемой его свойством **Position**, и испускающий параллельные лучи (**style=infinite**, то есть бесконечно удаленный источник).

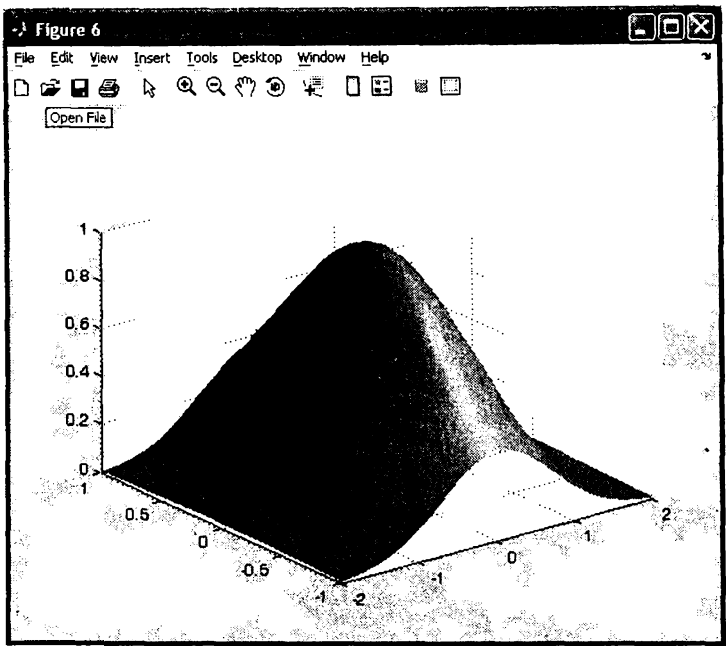


Рис. 4.8. Визуализация поверхности, описываемой функцией $f(x,y) = e^{-x^2-y^2}$, с использованием функции **surfl** (режим цветовой палитры — interp)

Также можно задать цвет дополнительного источника подсветки поверхности графика (свойство **Color** объекта **light**). Для того чтобы сделать источник более сильным, нужно продублировать его несколько раз, создавая все новые источники в том же месте и с теми же свойствами. При этом дескрипторы каждого вновь создаваемого источника света нужно запоминать в соответствующих переменных. Тогда впоследствии можно с помощью функции **set** изменять их свойства и суммарную силу света (для «выключения» конкретного источника нужно задать его свойство **Color** равным **black**), добиваясь желаемого внешнего вида графика. В этом заключается преимущество работы с MATLAB в интерактивном режиме: можно оценить достигнутый результат и, если что-то не нравится, выполнить отдельные части работы (а не всю работу) заново.

4.2. Дополнительные детали оформления трехмерных графиков

Многие приемы оформления трехмерных графиков совпадают с теми, что были рассмотрены при изучении двумерных графиков функций, зависящих от одной переменной. В частности, для масштабирования удобно использовать функцию **axis**, в которую в трехмерном случае следует передать три пары скалярных аргументов:

```
axis( [ xmin, xmax, ymin, ymax, zmin, zmax ] )
```

Для 3D графиков по-прежнему можно использовать функции **text**, **xlabel**, **ylabel**, **zlabel**, **title**, а также можно наносить отметки на осях координат с помощью функции **set**, а также с помощью функции **subplot** можно разместить в одном графическом окне несколько трехмерных графиков.

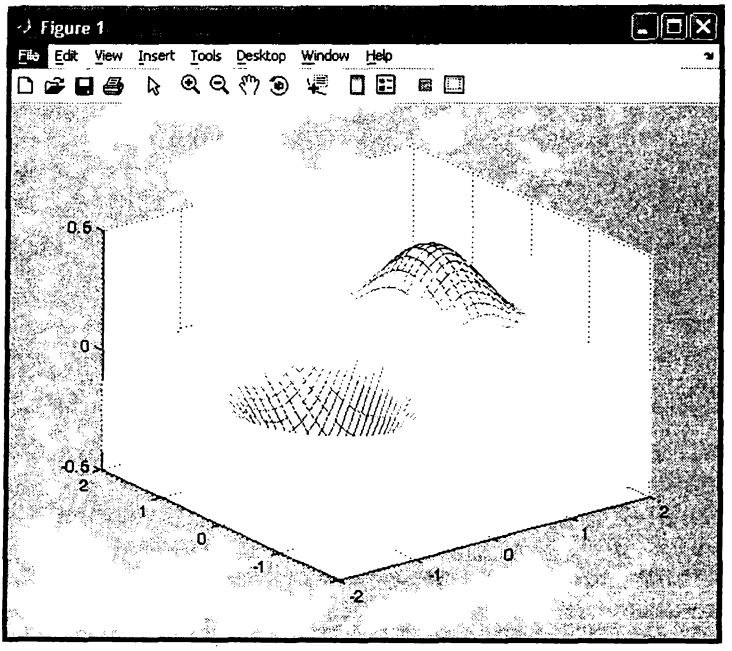


Рис. 4.9. $f(x,y) = x \cdot e^{-x^2-y^2}$ с «пьедесталом»

К новым методам дополнительного оформления трехмерных графиков можно отнести возможность вызывать функцию **mesh** с суффиксами **z** и с **(meshz** и **meshc)**, а функцию **surf** с суффиксом с **(surfc)**. Использование суффикса **z** приводит к построению «графика с пьедесталом». Например, график, представленный на рис. 4.9 строится с помощью следующей последовательности команд

```
>> [X Y]=meshgrid(-2:0.1:2);
>> Z=X.*exp(-X.^2-Y.^2);
>> meshz(X,Y,Z)
```

Для построения на одном чертеже поверхности и проекции карты линий уровня на плоскость $Z = -10$ используется функция **surfc**. Например, для построения графика, представленного на рис. 4.10, необходимо выполнить следующую последовательность команд:

```
>> [X Y Z]=peaks(30);
>> surfc(X,Y,Z); colormap(hsv); axis([-3 3 -3 3 -10 5]);
```

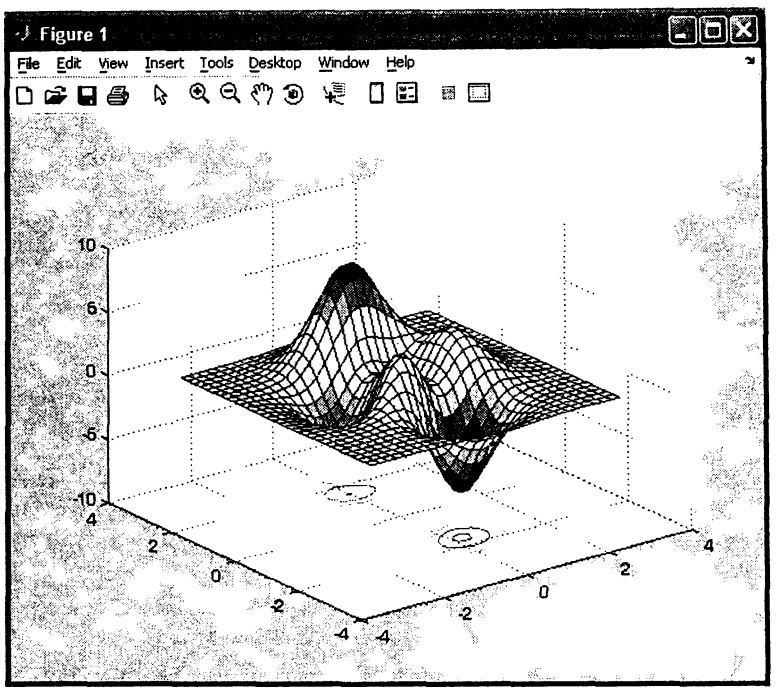


Рис. 4.10. Поверхность, задаваемая функцией **peaks**, и соответствующая карта линий уровня

Функция с **peaks** является некоторой линейной комбинацией стандартных гауссовых функций.

Наконец, для трехмерных графиков существует возможность изменять свойства условной камеры, фиксирующей сцену с графиком: можно менять положение этой камеры, расстояние от сцены, а также свойства ее объектива. Это мощный способ влиять на детали изображения. Изменять свойства такой камеры можно с помощью дескрипторного метода. Напомним, что графическим объектом, соответствующим поверхности трехмерного графика функции, является объект типа **surface**. Дескриптор (описатель) такого объекта возвращается функциями **mesh**, **surf** и **surfl**. Объект **surface** является наследником объекта **axes**. Получить описатель объекта **axes** можно, вызвав функцию **gca**, которая возвращает описатель текущего объекта **axes**. (Если создано несколько таких объектов, то перед вызовом функции нужно щелкнуть мышью на выбираемом объекте **axes**).

Получив описатель, далее можно функцией **set** изменять свойства объекта **axes**, в частности, свойства, связанные с камерой, фиксирующей сцену с расположенным на ней трехмерным графиком. Приведем список соответствующих свойств объекта **axes**:

```
>> get(hA )

    CameraPosition
    CameraPositionMode
    CameraTarget
```

```
CameraTargetMode
CameraUpVector
CameraUpVectorMode
CameraViewAngle
CaaaraViewAngleMode
```

Метод изменения точки обзора (**viewpoint**) является более простым по сравнению с методом изменения свойств условной камеры наблюдения. В частности, для точки обзора нет понятия расстояния до сцены и тем более, свойств объектива. Для нее можно менять только углы, задающие ориентацию этой точки в пространстве: угол азимута (**az**) и угол возвышения (**el**). Изменение первого угла означает вращение плоскости *xOy* вокруг оси *Oz* против часовой стрелки. Угол возвышения есть угол между направлением на точку обзора и плоскостью *xOy*.

По умолчанию для высокоуровневые графические функции **mesh**, **surf**, **surfl**, устанавлены значения **az** = −37,5°, **el** = 30°. Эти значения в любой момент времени можно изменить специальной функцией

```
view([az ,el])
```

В частности, если после построения представленного на рис. 4.9 «графика с пьедесталом» выполнить команду **view**([−15, 20]), то график изменит свой вид, поскольку мы уменьшили угол возвышения с 30 градусов до 20 градусов, а угол азимута изменили с −37,5 градусов на значение, равное −15 градусам. В итоге точка обзора графика находится больше сбоку, чем сверху, и преимущественно вдоль одной из независимых координат (см. рис. 4.11).

Рассмотренную операцию изменения ориентации в пространстве трехмерного графика легче выполнить с помощью панели инструментов графического окна. В частности, нужно использовать кнопку со стрелкой на окружности, действие которой мы уже ранее рассматривали в главе 2. Передвигая в разные стороны указатель мыши при нажатой левой ее клавише, можно добиться желаемого расположения в пространстве ограничивающего параллелепипеда, построенного на осях координат.

Завершая краткое рассмотрение 3D графики, заметим, что MATLAB обладает огромными, трудно поддающимися обзору возможностями оформления графиков функций. Все рассмотренные до сих пор возможности основаны на так называемой векторной графике, когда изображаемый объект задан своими координатами (числовыми данными, накопленными в массивах), которые сам MATLAB в момент отображения переводит в значения (цвет) пикселей дисплея. Векторная графика прекрасно поддается масштабированию. После изменения размера графического окна MATLAB получается столь же качественное изображение иного размера, что и исходное изображение.

Однако существуют графические объекты, заданные в растровой форме (например сканированные для ввода в компьютер фотографии). MATLAB также обладает развитыми средствами работы с растровой графикой, которые будут рассматриваться далее в главе, посвященной более подробному изучению дескрипторной графики MATLAB.

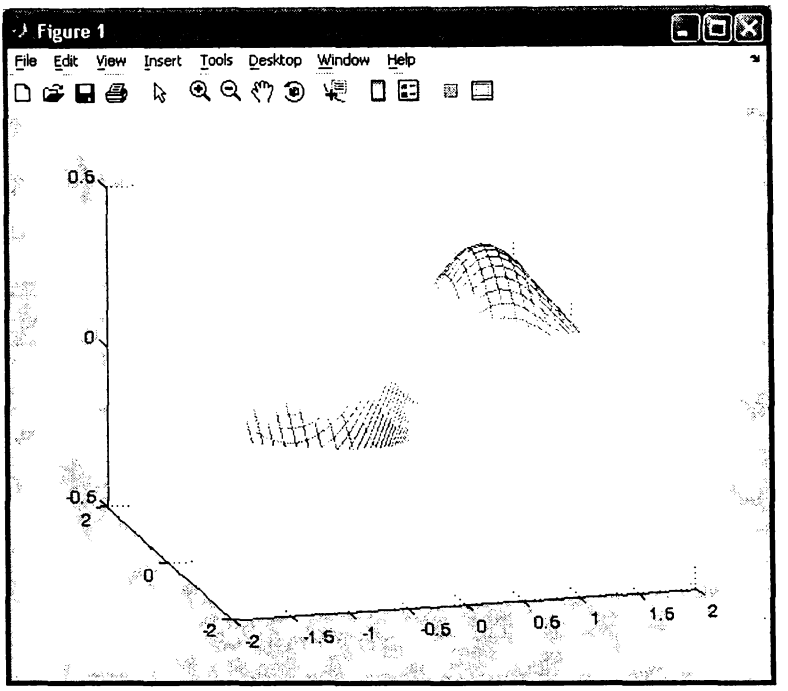


Рис. 4.11. Поверхность, задаваемая функцией $f(x,y) = x \cdot e^{-x^2-y^2}$ (угол возвышения 30°, азимутальный угол -15°)

4.3. Сохранение графических изображений в дисковых файлах

Для сохранения графического изображения необходимо выполнить в графическом окне команду меню **File | Save** или **File | Save as**. После этого на экране появится стандартное окно для сохранения данного графического окна (рис. 4.12).

В данном окне следует указать имя файла и выбрать расширение fig, предлагаемое MATLAB по умолчанию. Графическая информация будет сохранена в двоичном файле того же формата, как и рассмотренный ранее MAT-файл. Для считывания ранее сохраненного файла, содержащего графическое изображение следует выполнить команду **File | Open**.

Для экспорта файла в другие графические форматы следует выполнить команду **File | Save as** и затем в появившемся окне (рис. 4.13) указать выбранный формат и имя файла.

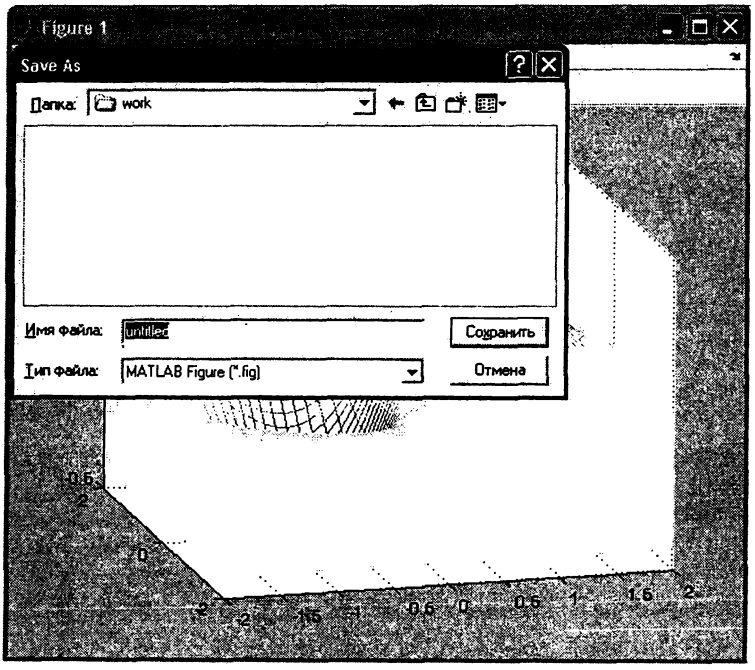


Рис. 4.12. Диалоговое окно в режиме «Сохранить файл»

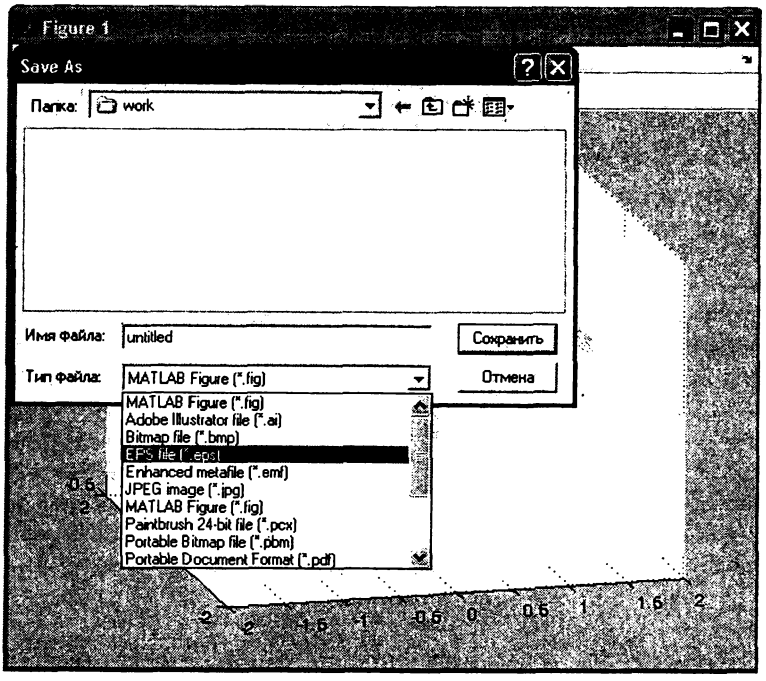


Рис. 4.13. Диалоговое окно в режиме «Сохранить как»

Вопросы для самопроверки

1. Какой последовательностью действий в MATLAB осуществляется построение пространственных кривых?
2. Для каких целей используется функция **meshgrid**?
3. Чем отличаются друг от друга поверхности построенные функциями **mesh**, **surf** и **surfl**?
4. Как получить информацию о свойствах графического объекта, содержащего изображения поверхности?
5. Какой командой активируется режим отображения ребер, закрываемых поверхностью?
6. Какая команда устанавливает угловое положение точки обзора поверхности?
7. Как сохраняются и считываются графические изображения в двоичном формате?
8. Как осуществляется экспорт графических изображений в известные графические форматы?

Встроенные средства решения типовых задач алгебры и анализа

5.1. Решение систем линейных уравнений

Как мы уже знаем, в MATLAB для решения систем линейных уравнений предусмотрены знаки операций — это знаки / и \.

Второй из этих знаков мы уже использовали в главе 1 при решении системы линейных уравнений вида

```
Ay=b,
```

где **A** — квадратная матрица **n**×**n**, **b** — вектор-столбец длины **N**. Для нахождения неизвестного вектор-столбца y достаточно применить операцию \ и вычислить выражение **A\b**. Например:

```
>> A=[1 -2 3 -1;2 3 -4 4; 3 1 -2 -2; 1 -3 7 6]
A =
     1    -2     3    -1
     2     3    -4     4
     3     1    -2    -2
     1    -3     7     6
>> b=[6;-7;9;-7]
b =
     6
    -7
     9
    -7
>> y=A\b
y =
    2.0000
   -1.0000
         0
   -2.0000
>>
```

В общем случае операция \ называется «левым делением матриц» и, будучи примененная к матрицам **A** и **B** в виде **A\B**, подобна вычислению выражения

```
inv(A)·B.
```

Здесь под **inv(A)** понимается вычисление матрицы, обратной к матрице **A**.

Операцию / называют «правым делением матриц». Выражение **A/B** подобно вычислению выражения **B · inv(A)**. Эта операция позволяет решать системы линейных уравнений вида

```
Y·A=B
```

так как решением именно этого уравнения является выражение **B · inv(A)**.

5.2. Операции линейной алгебры над матрицами. Матричные функции

При численном решении линейных уравнений существует ряд трудноразрешимых проблем, одной из которых является плохая численная обусловленность матрицы системы[1], следствием которой является высокая чувствительность численного решения к небольшим изменениям исходных данных.

Следующие встроенные функции MATLAB позволяют оценить числа обусловленности матрицы:

cond(A) — возвращает число обусловленности как отношение самого большого сингулярного числа матрицы **A** к наименьшему. Если это число близко к единице, то матрица хорошо обусловлена.

condeig(A) — возвращает вектор, содержащий числа обусловленности собственных значений матрицы **A**.

rcond(A) — возвращает обратную величину числа обусловленности матрицы **A**, Возвращаемые значения, близкие к единице, говорят о хорошей обусловленности, а близкие к нулю — о плохой.

Так как подробное обсуждение многочисленных функций пакета Liniar Algebra выходит за рамки нашей книги, далее мы описываем только основные функции, позволяющие решать задачи линейной алгебры, и приводим соответствующие примеры.

det(A) — возвращает значение определителя квадратной матрицы **A**.

rank(A) — возвращает ранг матрицы **A**.

norm(A) — возвращает норму матрицы — наиболшее по абсолютной величине сингулярное число.

orth(A) — возвращает ортонормированный базис для разложения матрицы **A**.

---

[1] Можно показать, что при использовании нормы $\|A\|_1$, число обусловленности матрицы связано с максимальным собственным значением матрицы AA^* (здесь A^* — эрмитово сопряженная матрица) ρ соотношением $\|A\| = \sqrt{\rho}$.

```
>> A=[1 2 3; 2 1 4; 3 4 5]
A =
  1     2     3
  2     1     4
  3     4     5
>> B=orth(A)
B =
 -0.4074   -0.1186    0.9055
 -0.4842    0.8688   -0.1040
 -0.7743   -0.4808   -0.4113
>> B^-1*A*B
ans =
  9.0795   -0.0000   -0.0000
 -0.0000   -1.4870    0.0000
 -0.0000   -0.0000   -0.5925
>>
```

rref(A) — приводит расширенную матрицу системы линейных уравнений порядка $n \times n$ к виду **[E v]**, где **E** — единичная матрица, размерность которой равна $n \times n$, **v** — вектор-столбец, содержащий решение системы.

```
>> A=[1 2 3; 2 1 4; 3 4 5]   задаем матрицу системы
A =
  1     2     3
  2     1     4
  3     4     5
>> b=[10 20 30]   задаем вектор-столбец свободных членов
b =
  10    20    30
>> D=[A b']   создаем расширенную матрицу системы
D =
  1     2     3    10
  2     1     4    20
  3     4     5    30
>> rref(D)   обращаемся к функции rref
ans =
  1     0     0    10
  0     1     0     0
  0     0     1     0
>> A*ans(:,4) % проверяем полученное решение
ans =
  10
  20
  30
>>
```

rrefmovie(A) — результат аналогичен результату, возвращаемому функцией **rref**, при этом выводится информация о каждой выполняемой операции и получаемом при этом результате.

```
 Original matrix
A =
   1           2           3          10
   2           1           4          20
   3           4           5          30
Press any key to continue.
```

```
  swap rows 1 and 3   перестановка первой и третьей строки
A =
    3          4          5         30
    2          1          4         20
    1          2          3         10
Press any key to continue.
  pivot = A(1,1)       % деление первой строки на a11
A =
    1        4/3        5/3         10
    2          1          4         20
    1          2          3         10
Press any key to continue...
  eliminate in column 1 % преобразование первого столбца
A =
    1        4/3        5/3         10
    2          1          4         20
    1          2          3         10
Press any key to continue...
A =
    1        4/3        5/3         10
    0       -5/3        2/3          0
    1          2          3         10
A =
    1        4/3        5/3         10
    0       -5/3        2/3          0
    0        2/3        4/3          0
Press any key to continue..
  pivot = A(2,2) % деление второй строки на a22
A =
    1        4/3        5/3         10
    0          1       -2/5          0
    0        2/3        4/3          0
Press any key to continue.
  eliminate in column 2 % преобразование второго столбца
A =
    1        4/3        5/3         10
    0          1       -2/5          0
    0        2/3        4/3          0
Press any key to continue.
A =
    1          0       11/5         10
    0          1       -2/5          0
    0        2/3        4/3          0
A =
    1          0       11/5         10
    0          1       -2/5          0
    0          0        8/5          0
Press any key to continue.
  pivot = A(3,3) % деление элементов третьей строки на a33
A =
    1          0       11/5         10
    0          1       -2/5          0
    0          0          1          0
Press any key to continue.
  eliminate in column 3
A =
```

```
    1           0         11/5         10
    0           1         -2/5          0
    0           0           1           0
Press any key to continue.
A =
    1           0           0          10
    0           1         -2/5          0
    0           0           1           0
A =
    1           0           0          10
    0           1           0           0
    0           0           1           0
Press any key to continue.
>>
```

chol(A) — возвращает разложение Холецкого для положительно определенной матрицы **A** верхнюю треугольную матрицу **t** такую, что $\mathbf{t}' \cdot \mathbf{t} = \mathbf{A}$.

lu(A) — возвращает результат lu-разложения матрицы **A** (разложение на нижнетреугольную в верхнетреугольную матрицы).

```
>> A=[1 2 3; 2 1 4; 3 4 5]
A =
  1     2     3
  2     1     4
  3     4     5
>> [L U]=lu(A)
L =
  0.3333   -0.4000    1.0000
  0.6667    1.0000         0
  1.0000         0         0
U =
  3.0000    4.0000    5.0000
       0   -1.6667    0.6667
       0         0    1.6000
>> L*U
ans =
  1     2     3
  2     1     4
  3     4     5
>>
```

qr(A) — возвращает qr-разложение матрицы **A** (разложение на унитарную и верхнетреугольную матрицы).

```
>> [Q R]=qr(A)
Q =
 -0.2673    0.5203   -0.8111
 -0.5345   -0.7804   -0.3244
 -0.8018    0.3468    0.4867
R =
 -3.7417   -4.2762   -6.9488
       0    1.6475    0.1734
       0         0   -1.2978
>> Q*R
ans =
  1.0000    2.0000    3.0000
```

```
    2.0000    1.0000    4.0000
    3.0000    4.0000    5.0000
>>
```

d = eig(A) — возвращает вектор **d**; координаты которого есть собственные значения матрицы **A**.

[V,D]=eig(A) — возвращает матрицу собственных векторов **V** и диагональную матрицу собственных значений, причем справедливо равенство **A·V = V·D**.

Матрицу **D** называют канонической формой матрицы **A**. Элементы, стоящие на главной диагонали матрицы **D**, равны собственным значениям матрицы **A**. Матрица **v** состоит из столбцов, каждый из которых есть один из собственных векторов матрицы **A**.

s=svd(A) — возвращает вектор, элементы которого являются искомыми сингулярными числами.

[U,S,V]=svd(A) — возвращает диагональную матрицу **S** и унитарные матрицы **U** и **V** такие, что имеет место равенство **A = U·S·V**

```
>> A=[1 2 3; 3 4 5; 6 7 8]
A =
   1     2     3
   3     4     5
   6     7     8
>> [U,S,V]=svd(A)
U =
  -0.2500   -0.8371   -0.4867
  -0.4852   -0.3267    0.8111
  -0.8379    0.4389   -0.3244
S =
  14.5576         0         0
        0    1.0372         0
        0         0    0.0000
V =
  -0.4625    0.7870    0.4082
  -0.5706    0.0882   -0.8165
  -0.6786   -0.6106    0.4082
>> U*S*V'
ans =
   1.0000    2.0000    3.0000
   3.0000    4.0000    5.0000
   6.0000    7.0000    8.0000
```

В MATLAB также имеются функции **cdf2rdf**, **hess**, **qz**, **rsf2csf** и **schur**, осуществляющие приведение матриц к специальным формам Хессенберга и Шура.

Для вычисления функций матриц используются функции с суффиксом m, например, **expm**[1], **logm**[2] и **sqrtm**, которые в отличие от функций **exp**, **log**, **sqrt**, вычисляющих значения функций поэлементно, выполняют вы-

---

[1] По определению матричная экспонента задается разложением $e^A = \sum_{p=0}^{\infty} \frac{A^p}{p!}$.

[2] По определению матричный логарифм задается разложением $\ln A = \sum_{p=1}^{\infty} \frac{(-p)^{p-1}}{p} (A - E)^p$

численния с квадратными матрицами по правилам линейной алгебры, используя разложения математических функций в степенные ряды.

```
>> A=[1.1 1 0; 0 0 2; 0 0 -1];
>> expm(A)
ans =
   3.0042    1.8220    1.1332
     0       1.0000    1.2642
     0         0       0.3679
```

Рассмотрим решение некоторых задач линейной алгебры.

Задача 5.1.

Задана матрица $D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$. Вычислить: 1) сумму элементов каждого столбца матрицы; 2) вычислить сумму элементов каждой строки; 3) осуществить перестановку первой и второй строки; 4) осуществить перестановку второго и третьего столбца матрицы.

```
>> D=[1 2 3;4 5 6;7 8 9]    задаем матрицу D
D =
   1     2     3
   4     5     6
   7     8     9
>> sum(D,1)    вычисляем сумму элементов каждого столбца матрицы
ans =
   12    15    18
>> sum(D,2)    вычисляем сумму элементов каждой строки матрицы
ans =
    6
   15
   24
% создаем матрицу, получаемую из матрицы D перестановкой
% первой второй строк
>> D1(1,:)=D(2,:),D1(2,:)=D(1,:),D1(3,:)=D(3,:)
D1 =
    4     5     6
D1 =
    4     5     6
    1     2     3
D1 =
    4     5     6
    1     2     3
    7     8     9
% создаем матрицу, полученную из матрицы D перестановкой второго
% и третьего столбцов
>> D2(:,1)=D(:,1),D2(:,2)=D(:,3),D2(:,3)=D(:,2)
D2 =
    1
    4
    7
D2 =
    1     3
```

```
     4     6
     7     9
D2 =
     1     3     2
     4     6     5
     7     9     8
```

5.3. Разреженные матрицы

При решении многих прикладных задач, например, при решении граничных задач в уравнениях с частными производными, возникают матрицы большого размера, большинство элементов которых равны нулю. Если хранить такие матрицы обычным образом, то, как очевидно, память компьютера будет использоваться неэффективно. Это, в свою очередь, приведет к снижению быстродействия вычислений.

В MATLAB эта проблема преодолевается введением для таких матриц специальных структур данных, называемых разреженными матрицами. Для создания массива, в котором хранится разреженная матрица, используется функция **sparse**:

```
S=sparse(A)
>> A=[1 0 0; 0 1 0; 1 0 0]
A =
   1     0     0
   0     1     0
   1     0     0
>> S=sparse(A)
S =
   (1,1)        1
   (3,1)        1
   (2,2)        1
>> whos S
 Name       Size            Bytes  Class
 S          3x3            52  sparse array
Grand total is 3 elements using 52 bytes
>> whos A
 Name       Size            Bytes  Class
 A          3x3            72  double array
Grand total is 9 elements using 72 bytes
>>
```

Из приведенного примера видно, что функция **sparse** размещает в памяти компьютера только индексы ненулевых элементов и их значения. При этом матрица **S**, являющаяся переменной типа **sparse array**, занимает в памяти меньше места, чем исходная матрица **A**.

Обратное преобразование разреженной матрицы в обычную производится функцией **full**:

```
>> full(S)
ans =
   1     0     0
```

```
0     1     0
1     0     0
```

Для разреженных матриц в MATLAB имеется большое количество функций для действий с разряженными матрицами, справочную информацию по каждой из которых можно посмотреть в диалоговом окне Help (рис. 5.1).

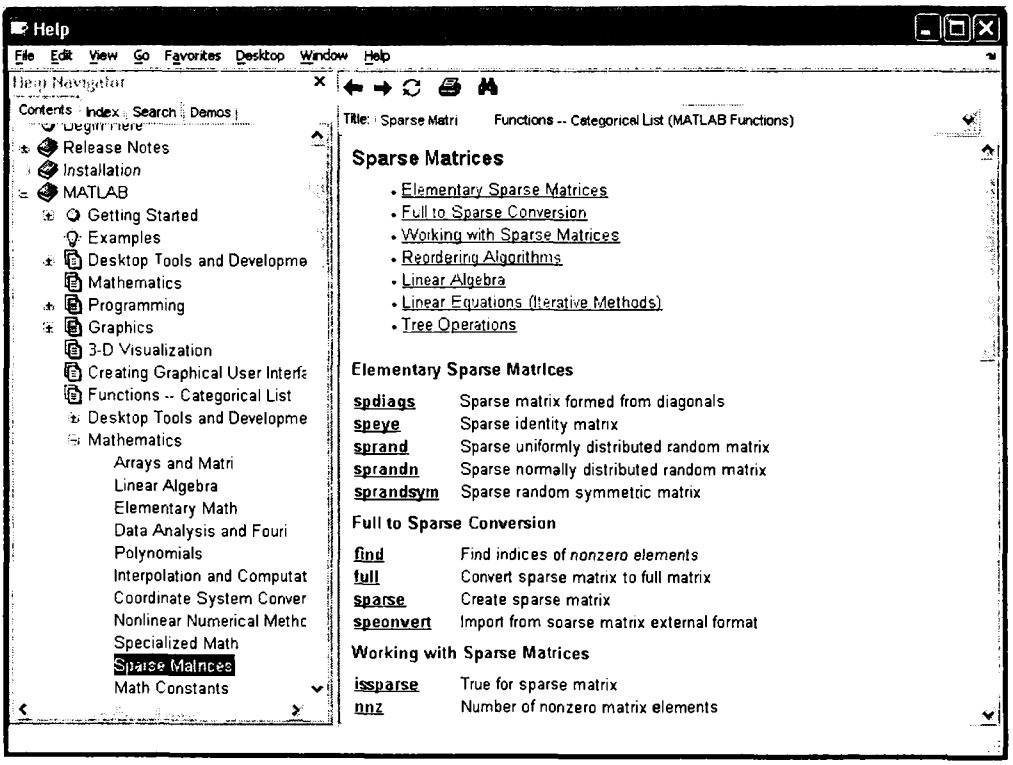


Рис. 5.1. Окно помощи в режиме просмотра информации о разряженных матрицах

5.4. Вычисление специальных функций математической физики

Помимо всех элементарных функций MATLAB предоставляет возможность вычисления многих стандартных специальных функций математической физики.

besselj(n, X) — вычисление функций Бесселя первого рода, где **n** — порядок функции, а x — это вектор, содержащий координаты точек, в которых вычисляются значения функции.

```
>> x=0:0.01:20;
>> y1=besselj(0,x);
>> y2=besselj(1,x);
>> y3=besselj(2,x);
>> plot(x,y1,x,y2,x,y3);
```

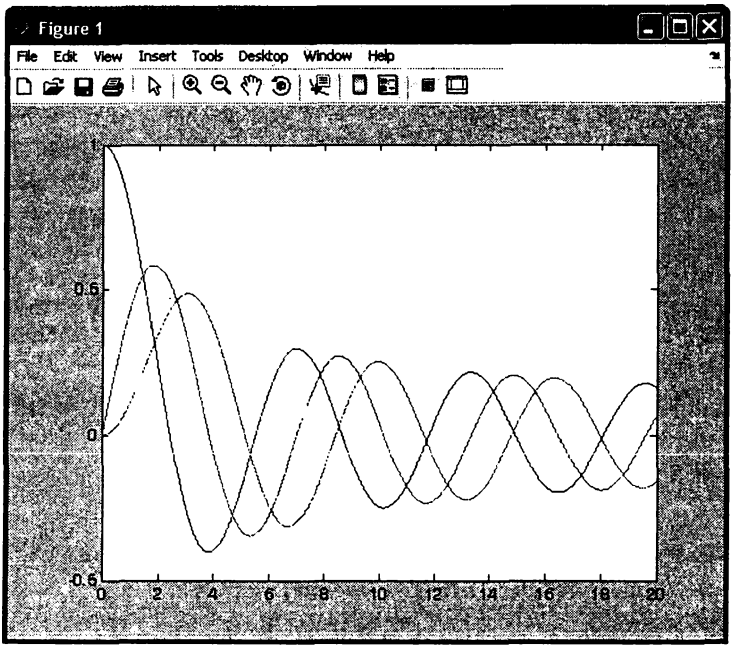


Рис. 5.2. Графики функций Бесселя первого рода нулевого, первого и второго порядков

bessely(n, X) — вычисление функций Бесселя второго рода, где **n** — порядок функции, а **x** — это вектор, содержащий координаты точек, в которых вычисляются значения функции.

besselh(n, X) — вычисление функций Бесселя третьего рода (функция Ханкеля), где **n** — порядок функции, а x — это вектор, содержащий координаты точек, в которых вычисляются значения функции.

Для построения графиков функций Бесселя второго и третьего родов необходимо выполнить последовательность команд, аналогичную случаю построения графика функции Бесселя первого рода.

Помимо функций Бесселя в MATLAB имеются функции, позволяющие вычислить числовые значения функций Эйри, бета- и гамма-функций, функции ошибки (**erf(x)**), интегральных показательных функций, эллиптических функций Якоби, а также ортогональных полиномов Лежандра. Синтаксис обращения к данным функциям можно найти в диалоговом окне Help (рис. 5.3).

Отметим, что в целом MATLAB располагает достаточно полным (хотя и не исчерпывающим) набором встроенных функций, позволяющих получать численные значения основных специальных функций математической физики.

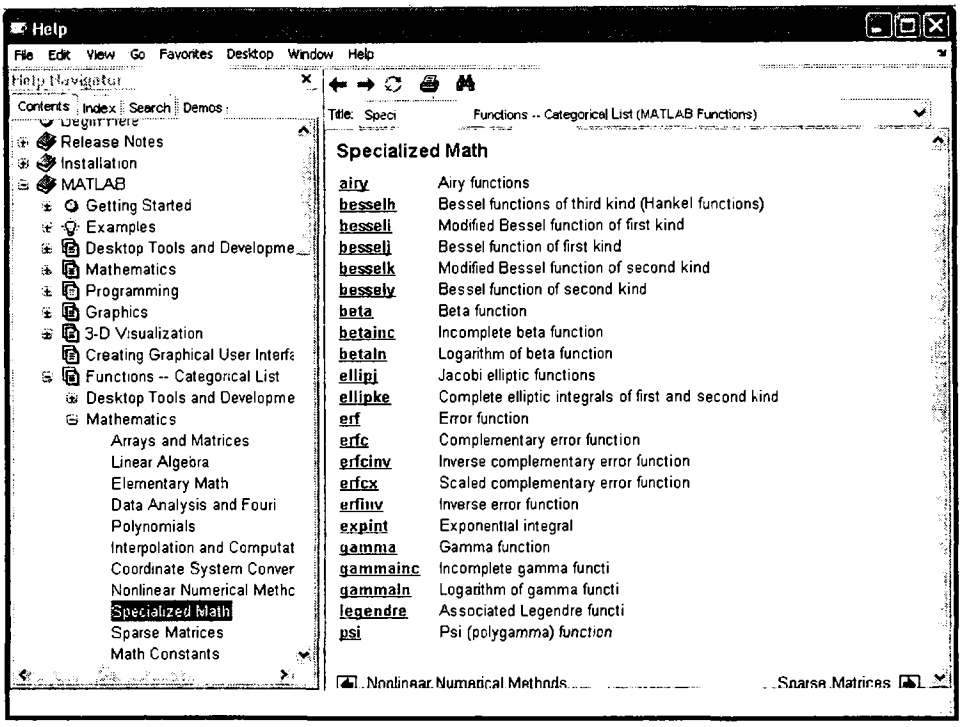


Рис. 5.3. Окно помощи в режиме просмотра информации о специальных функциях математической физики

5.5. Нахождение нулей функций

На практике часто приходится сталкиваться с задачей нахождения корней уравнений, в которые входят действительные функции, зависящие от одной переменной. С геометрической точки зрения задача о нахождении решений любого уравнения вида $f(x) = g(x)$ сводится к нахождению координат точек пересечения графика функции $F(x) = f(x) - g(x)$ с осью абсцисс, т.е. к задаче нахождения нулей функций $F(x)$.

Для нахождения решения рассматриваемой задачи в MATLAB используется функции **fzero**:

```
fzero( hFunction, x0 )
```

где **hFunction** — дескриптор функции, задающей исходное уравнение, **x0** — начальное приближение к корню.

Функция **fzero** возвращает значение нуля функции, заданной дескриптором **hFunction** в окрестности точки $x0$.

В качестве примера рассмотрим задачу о нахождении нулей функции **cos(x)** на отрезке от 0 до π. В качестве начального приближения выберем $x0 = 1$.

```
>> x0=1;
>> x=fzero(@cos,x0)
x =
   1.5708
```

В качестве нуля функции **cos(x)** получено значение, близкое к точному значению корня, равному $\pi/2$. Здесь дескриптор функции получен с помощью символа @ и имени функции.

Если требуется найти корень функции, не являющейся встроенной в MATLAB (т.е. не имеющей в MATLAB фиксированного имени), следует использовать функцию **inline**.

```
>> f=inline('cos(x)-x','x')
f =
   Inline function:
   f(x) = cos(x)-x
>> x=fzero(f,pi/2)
x =
   0.7391
>> f(x)
ans =
   0
```

Другой способ состоит в создании в текстовом редакторе файла, содержащего описание данной функции, который следует сохранить в текущем каталоге. При этом имя функции должно совпадать с именем файла.

```
function y = MyFunction(x)
y = cos(x)   x;
```

После этого обратится к функции **fzero**:

```
>> x=fzero(@MyFunction,pi/2)
```

или

```
>> x=fzero('MyFunction',pi/2)
```

Для управления величиной погрешности, следует обращаться к функции **fzero**, используя следующий синтаксис:

```
>> fzero( hFunction, x0, options )
```

где параметр **options** задает различные дополнительные требования к работе функции **fzero**, в частности, требуемую величину погрешности. При этом значение параметра **options** можно устанавливать с помощью специальной функции **optimset**:

```
>> options=optimset('TolX',1e-8);
```

Здесь мы указываем имя дополнительного параметра — **TolX** (Tolerance of **X** — точность нахождения корня), а также его числовое значение -10^{-8}.

```
>> options=optimset('TolX',1e-3);
>> x=fzero(f,pi/2)
x =
```

```
   0.73908513321516
>> x=fzero(f,pi/2,options)
x =
   0.73977801234487
>> f(x)
ans =
 -0.00115978821945
>> options=optimset('TolX',1e-8);
>> x=fzero(f,pi/2,options)
x =
   0.73908513263090
>> f(x)
ans =
   9.778322596076805e-010
```

Удалить раз подчеркнем, что функция **fzero** находит нули только вещественнозначных функций, зависящих от одной вещественной переменной. Однако на практике встречаются задачи, в которых требуется находить комплексные корни вещественнозначных многочленов.

Для этой цели в MATLAB существует функция **roots**, которой в качестве аргумента передается массив коэффициентов многочлена. Например, для многочлена $x^4 - 3x^3 + 3x^2 - 3x + 2$, имеющего два вещественных (1 и 2) и два комплексных корня (i и $-i$), сначала необходимо сформировать массив, содержащий коэффициенты полинома, а затем вызвать функцию **roots**.

```
>> Coeff=[1 -3 3 -3 2]
Coeff =
    1   -3    3   -3    2
>> r=roots(Coeff)
r =
   2.00000000000000
  -0.00000000000000 + 1.00000000000000i
  -0.00000000000000   1.00000000000000i
   1.00000000000000
>>
```

Для правильного выбора начального приближения целесообразно сначала построить график соответствующей функции. Для этого можно использовать функцию **fplot**

```
fplot(hFunction,[x0,x1]),
```

которая строит график функции с описателем **hFunction**, равным **@name** (**name** — имя функции), на отрезке от x0 до x1.

```
>> fplot(f,[0,pi])
>> fplot(@MyFunction,[0,pi])
>> fplot('MyFunction',[0,pi])
```

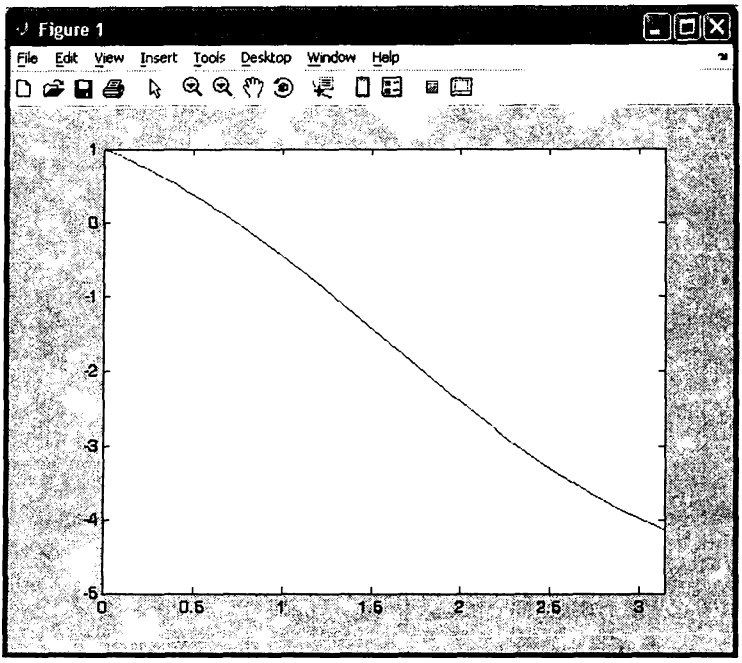


Рис. 5.4. График функции $y = \cos x - x$

Отметим, что функция **fplot** требует от пользователя еще меньше подготовительной работы, чем изученная ранее функция **plot** (см. главу 2).

5.6. Поиск минимума функции

В MATLAB имеются функции, осуществляющие поиск минимумов заданных функций. При этом возможен поиск минимума как у функции одной вещественной переменной, так и у функций многих переменных.

Для функций одной переменной для отыскания минимумов используется функция **fminbnd**:

```
fminbnd(hFunction, x0, x1)
```

где **hFunction** — дескриптор функции, у которой находятся минимумы, а x0 и x1 задают границы отрезока поиска.

Продемонстрируем использование этой функции на примере поиска минимума функции **humps** (переводится как «горб»), входящей в ядро MATLAB. Данная функция задается формулой

$$y = \frac{1}{(x-0.3)^2 + 0.01} + \frac{}{(x-0.9)^2 + 0.04} - 6$$

Для построения ее графика (рис. 5.5) следует выполнить команду

```
>> fplot(@humps,[0,3])
```

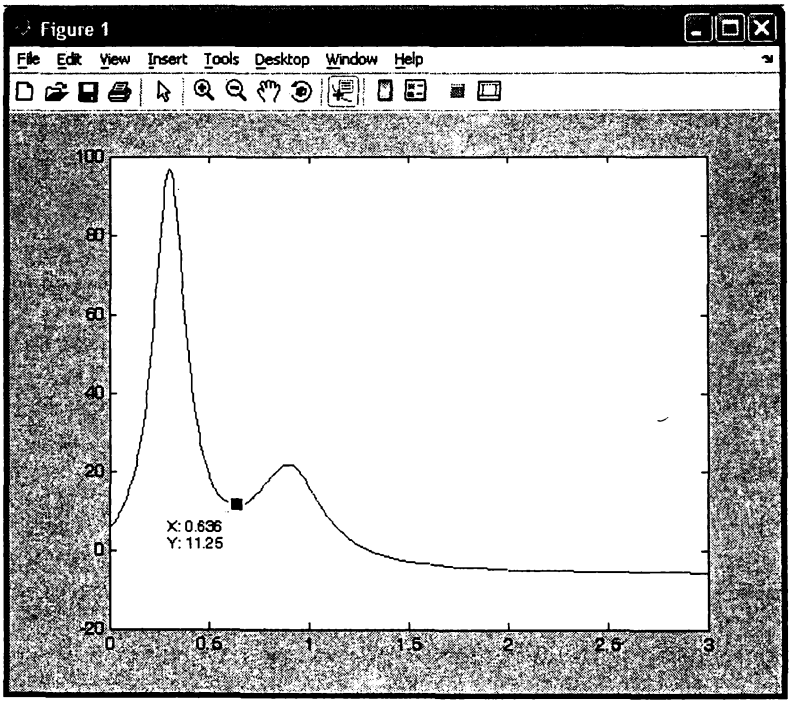


Рис. 5.5. График функции $y = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$

Из рисунка 5.5 видно, что локальный минимум функции **humps** существует на отрезке [0.5,1]. Для нахождения минимума функции нужно выполнить команду

```
>> fminbnd(@humps,0.5,1)
ans =
   0.63701067459059
>>
```

Для одновременного нахождения минимума функции и значения функции в точке минимума следует использовать следующий синтаксис вызова функции **fminbnd**

```
>> [x,y]=fminbnd(@humps,0.5,1)
x =
   0.63701067459059
y =
  11.25275412656431
>>
```

Для поиска минимума функции нескольких переменных применяется функция **fminsearch**:

```
fminsearch(hFunction, x0)
```

где **hFunction** — дескриптор функции нескольких переменных, для которой ищется минимум, **x0** — это вектор аргументов функции, с которого начинается поиск минимума.

Использование функции продемонстрируем на примере решения следующей задачи.

Задача 5.2.

Найти минимум функции $f(x,y) = x^2 + y^2$.

```
>> f2=inline('x(1)^2+x(2)^2','x')
f2 =
     Inline function:
     f2(x) = x(1)^2+x(2)^2
>> xmin=fminsearch(f2,[1 1])
xmin =
   1.0e-004 *
  -0.21023529262365   0.25484564932795
```

Отметим, что для функций нескольких переменных еще важнее, чем для рассмотренных ранее функций одной вещественной переменной, постараться априорно оценить количество и приблизительное нахождение локальных минимумов. Здесь могут существенно помочь трехмерные графики, способы построения которых мы обсуждали в главе 4.

Для получения информации о числе итераций, которые пришлось осуществить MATLAB для нахождения уточненного результата, необходимо обратится к функции **fminsearch** следующим образом:

```
[xmin, val, flag, output ] = fminsearch(hFun, x0)
```

где **output** — объект, содержащий полную информацию о процессе нахождения минимума функции. В частности, поле **output.iterations** содержит количество совершенных итераций. Для вывода на экран числа итераций, осуществленных функцией **fminsearch**, необходимо использовать следующий синтаксис обращения к данной функции:

```
>> [x,val,flag,output]=fminbnd(@humps,0.5,1)
x =
    0.63701067459059
val =
   11.25275412656431
flag =
     1
output =
   iterations: 8
   funcCount: 8
   algorithm: [1x46 char]
```

5.7. Вычисление определенных интегралов

Одной из задач численного анализа является задача вычисления определенных интегралов. Из всех методов вычисления определенных интегралов наиболее простым, но в то же время досточно успешно применяемым, является метод трапеций.

Вычисление интегралов методом трапеций в MATLAB реализует функция **trapz**:

```
trapz(x, y)
```

где **x** — вектор, содержащий дискретные значения аргументов подинтегральной функции, y — вектор, содержащий соответствующие значения подынтегральной функции.

Задача 5.3.

Вычислить интеграл $\int_0^5 \sin(x^2)dx$

```
>> dx=0.01;x=0:dx:5;
>> y=sin(x.^2);
>> trapz(x,y)
ans =
  0.52799989505923
```

Для оценки точности численного значения интеграла вычислим интеграл используя меньший шаг интегрирования.

```
>> dx=0.005;x=0:dx:5;
>> y=sin(x.^2);
>> trapz(x,y)
ans =
  0.52793793207748
```

Из сравнения полученных результатов видно, что значения интеграла, при использовании шага интегрирования 0.01 и 0.005, отличаются в пятом знаке после запятой. Следовательно, можно считать, что в пределах первых четырех знаков после запятой результат является точным.

При необходимости вычислить значения рассмотренного выше интеграла, как функции с переменным верхним пределом необходимо выполнить следующую последовательность команд:

```
>> z(1)=0;
 for i=1:length(y)-1
  x1=x(1:i+1);
  y1=y(1:i+1);
  z(i+1)=trapz(x1,y1);
 end;
>> plot(x,y,x,z)
```

Подынтегральная функция и интеграл с переменным верхним пределом представлены на рис. 5.6.

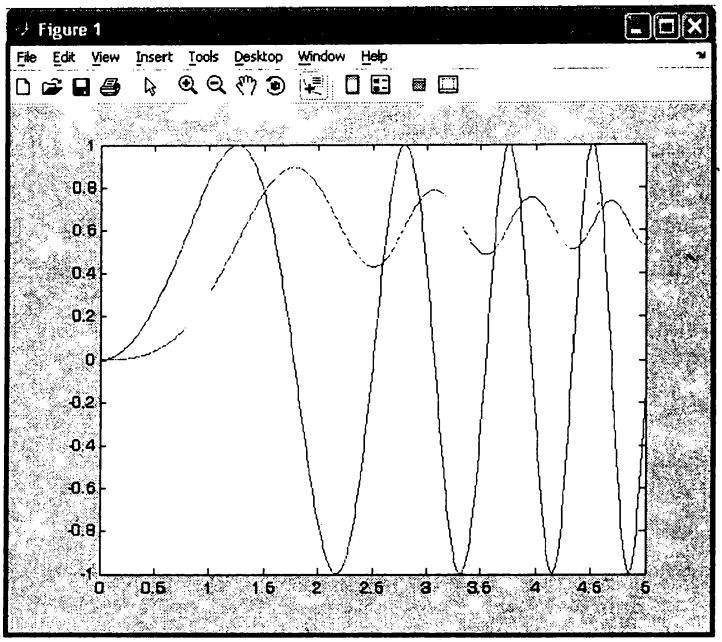


Рис. 5.6. Графики функций $y = \sin x^2$ и $y = \int_0^x \sin x^2 dx$

Метод трапеций хорошо подходит для интегрирования не слишком гладких функций. Если же функция под знаком интеграла является гладкой, т.е. существуют и непрерывны несколько ее первых производных, то лучше применять методы интегрирования более высоких порядков точности. При одном и том же шаге интегрирования методы более высоких порядков точности позволяют получить более точные результаты.

В MATLAB методы интегрирования более высоких порядков точности реализуются функциями **quad** (метод Симпсона) и **quad1** (метод Лобатто). Отметим, что оба метода являются адаптивными. Последнее означает, что пользователю нет необходимости контролировать достигнутую точность значения интеграла путем сравнения результатов, полученных при использовании разных шагов интегрирования. Данную процедуру, адаптивные функции выполняют самостоятельно.

Функции **quadl** имеет более высокий порядок точности по сравнению с функцией **quad**, что позволяет выполнять интегрирования гладких функций с при большим шагом при сохранении точности, следовательно, при меньшем объеме вычислений. Однако функция **quad** может иметь не меньшее, а даже большее быстродействие при интегрировании не слишком гладких функций.

Функции **quad** и **quad1**, как и многие другие функции MATLAB, имеют достаточно длинные списки формальных параметров, передаваемых в функцию при ее вызове. Минимальный список формальных параметров данных функций включает в себя три параметра: дескриптор подынтегральной функции, нижний и верхний пределы интегрирования. Четвер-

тый параметр списка, задающий требуемую относительную точностью результата вычислений, является необязательным.

Задача 5.4.

Вычислить интеграл $\int_0^5 \sin(x^2)dx$, используя функции **quad**, **quadl**

```
>> f=inline('sin(x.^2)','x');
>> [I,cnt]=quad(f,0,5,10^-5)
I =
  0.52791683119453
cnt =
  113
>> [I,cnt1]=quadl(f,0,5,10^-5)
I =
  0.52791728116252
cnt1 =
  198
```

Второе из значений, возвращаемых данными функциями, — количество точек, в которых пришлось вычислять подынтегральную функцию. Сравнивая полученные показатели, приходим к выводу, что при интегрировании функции $\sin(x^2)$ преимущество имеет метод **quad**, так как для достижения относительной точности 10^{-5}, требуется меньшее количество вычислений подинтегральной функцией (113 против 198). В то же время при интегрировании гладких функций, например, **sin(x)** и им подобных, преимущество оказывается на стороне **quadl**.

Из высшей математики известно, что к определенным интегралам от функций, зависящих от одной переменной, могут быть сведены многие другие типы интегралов, например, криволинейные интегралы. Следовательно, с помощью функций **quad**, **quadl** (или **trapz**) можно вычислить и эти интегралы.

Рассмотрим для примера вычисление криволинейного интеграла первого рода.

Задача 5.5.

Вычислить массу M винтовой линии C

$$x = \sin(t),\ y = 2\cos(t),\ z = 3t,\ 0 \le t \le 3$$

с постоянной линейной плотностью, равной 5.

Задача решается с помощью криволинейного интеграла первого рода:

$$M = \int_C 5ds,$$

который сводится к вычислению следующего определенного интеграла:

$$M = 5\int_0^3 \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2}\, dt = 5\int_0^3 \sqrt{\cos^2(t) + 4\sin^2(t) + 9}\, dt$$

```
>> f=inline('sqrt(cos(t).^2+4*sin(t) ^2+9)','t')
f =
     Inline function:
     f(t) = sqrt(cos(t).^2+4*sin(t).^2+9)
>> M=5*quad(f,0,3)
M =
  50.97041631367820
```

Двойные интегралы, как известно из курса математического анализа, сводятся к вычислению повторных определенных интегралов, один из которых является внутренним, а другой — внешним. При этом внутренний интеграл, рассматриваемый как интеграл с переменным верхним пределом, является подынтегральной функцией для внешнего интеграла. Можно написать достаточно очевидную цепочку вычислений, в которой многократные вычисления подинтегральной функции сводятся к многократным вызовам функции **quad**. Однако нет необходимости делать это самостоятельно, так как в MATLAB для этого имеется специальная функция **dblquad**, возвращающая значение двойного интеграла.

Задача 5.6. [redacted]

Вычислить интеграл $\int_0^1 \int_1^2 (x\sin(y) + y\sin(x))dxdy$.

```
>> f=inline('x.*sin(y)+y.*sin(x)','x','y')
f =
     Inline function:
     f(x,y) = x.*sin(y)+y.*sin(x)
>> dblquad(f,0,1,1,2)
ans =
   1.16777110966887
```

5.8. Решение систем обыкновенных дифференциальных уравнений

Для решения систем обыкновенных дифференциальных уравнений в MATLAB также имеются необходимые «решатели». Это функции **ode23**, **oda45**, **ode113**, **odel5s**, **ode23s**, **ode23t** и **ode23tb**.

ode23 — функция реализует одношаговые явные методы Рунге-Кутта (2 и 3) порядков. Функция используется при решении нежестких систем ДУ обеспечивает удовлетворительную точность при меньших нежели функция **ode45** временных затратах.

ode113 — функция реализует многошаговый метод Адамса-Башворта-Мултона переменного порядка. Функция используется при необходимости обеспечить высокую точность численного решения.

ode15s — функция реализует многошаговый метод переменного порядка (от 1 до 5 по умолчанию), основанный на формулах численного дифференцирования. Данный метод следует использовать в том случае, если не удается найти численное решение с помощью функции **ode45**.

ode23s — функция реализует одношаговый метод, использующий модифицированную формулу Розенброка 2-го порядка. Данный метод обеспечивает более высокую скорость вычислений по сравнению с другими методами при относительно более низкой точности вычислений.

ode23t — функция реализует метод трапеций с интерполяцией. Данный метод используют при решении уравнений, описывающих колебательные системы с почти гармоническим выходным сигналом.

ode23tb — функция реализует неявный метод Рунге-Кутта в начале интервала интегрирования и далее метод, использующий формулы обратного дифференцирования 2-го порядка. Данный метод обладает большей скорость нежели метод **ode15s** при, соответственно, меньшей точности.

Все перечисленные выше функции, называемые в документации пакета **Solver** (решатель), могут решать системы ДУ явного вида $\vec{y}' = \vec{F}(t, \vec{y})$. Кроме того решатели **ode15s**, **ode23s**, **ode23t** и **ode23tb** системы дифференциальных уравнений неявного вида, а также все решатели, кроме **ode23s**, могут решать уравнения вида $M(\vec{y})\vec{y}' = \vec{F}(t, \vec{y})$. Таким образом, для нахождения решения дифференциального уравнения n-го порядка с использованием функций пакета MATLAB следует привести его к эквивалентной системе дифференциальных уравнений первого порядка.

Задача 5.7.

Найти решение задачи Коши для дифференциального уравнения второго порядка

$$x'' + 2x = kx^2$$

с начальными условиями $x(0) = 0$, $x'(0) = 1$.

Приводим дифференциальное уравнение второго порядка к системе ДУ первого порядка:

$$\begin{aligned} x' &= z, \\ z' &= -2x + kx^2 \end{aligned}$$

Полученную систему можно записать в векторном виде:

$$\frac{d}{dt}\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} z \\ -2x + kx^2 \end{bmatrix}$$

или, вводя вектор $Y = \begin{bmatrix} x \\ z \end{bmatrix}$, в виде

$$\frac{dY}{dt} = \begin{bmatrix} Y_2 \\ -2Y_1 + kY_1 \end{bmatrix}$$

Для нахождения численного решения ДУ необходимо создать файл, содержащий описание вектор функции, возвращающей первые производные, с помощью любого текстового редактора и сохранить его в текущем каталоге.

```
% листинг файла MyFunction
function z=MyFunction(t,y)
global k;
z=[y(2); -2*y(1)+k*y(1)^2];
```

Далее необходимо выполнить следующую последовательность команд:

```
>> global k
>> k=0;
% находим решение для случая k = 0
>> [X Y]=ode45(@MyFunction,[0:0.01:20],[0,1]);
>> k=0.5;
% находим решение для случая k = 0.5
>> [X1 Y1]=ode45(@MyFunction,[0:0.01:20],[0,1]);
>> plot(X,Y(:,1),X1,Y1(:,1)) % визуализация решений ДУ при различных
                             % значениях k (рис. 5.7)
```

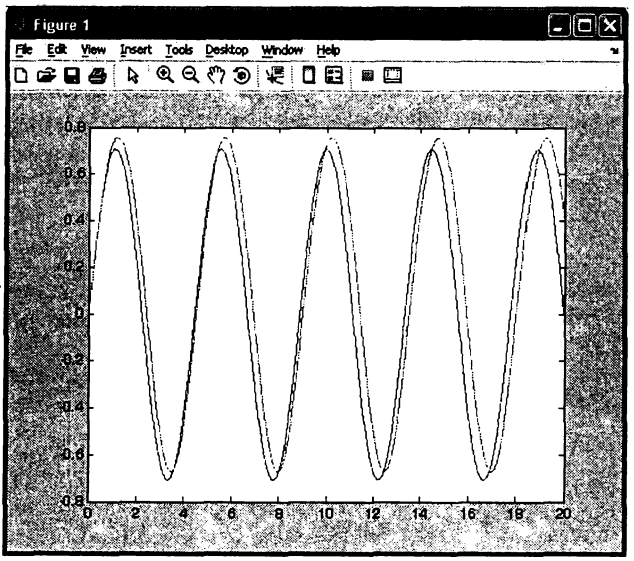


Рис. 5.7. Решение задачи Коши для дифференциальных уравнений $x'' + 2x = 0$, $x'' + 2x = 0{,}5x^2$ с начальными условиями $x(0) = 0$, $x'(0) = 1$

Рассмотрим методы решения жестких системам дифференциальных уравнений. Напомним, что система дифференциальных уравнений

```
y' = A  x
```

является жесткой, если определитель матрицы **A** близок к нулю. Особенность решений жестких систем ДУ, состоит в том, что в различные моменты времени соответствующие зависимости ведут себя абсолютно по-разному: в одни моменты времени они изменяются чрезвычайно быстро, в то время как в другие моменты времени чрезвычайно медленно. Данное поведение решений оказывается слишком сложным для обычных алгоритмов численного интегрирования ДУ, поэтому для решения жестких систем уравнений приходится применяют специальные методы, которые в MATLAB реализованы, как мы упоминали выше, в виде соответствующих функций, имена которых имеют суффикс s (stiff — жесткий).

Приведем классический пример системы, динамика которой описывается жесткой системой ДУ — генератора Ван-дер-Поля. Особенностью данной системы является существование при определенных условиях нелинейных релаксационных колебаний. Генератор Ван-Дер-Поля описывается следующим ДУ

$$x'' - \varepsilon(1 - x^2)x' + x = 0,$$

где ε — параметр, задающий степень нелинейности динамической системы.

Вектор-функция, возвращающая значения первых производных эквивалентной системы ДУ при $\varepsilon = 1000$, находится в файле **vdp1000.m**.

```
% листиг файла vdp100.m%
function dy = vdp1000(t,y)
dy = zeros(2,1);
dy(1) = y(2);
dy(2) = 1000*(1 - y(1)^2)*y(2) - y(1);
```

Для нахождения и визуализации решения системы ДУ необходимо выполнить следующие команды.

```
>> [T,Y] = ode15s(@vdp1000,[0 3000],[2 0]);
>> plot(T,Y(:,1));
```

Приведенные здесь примеры касаются лишь начальных задач для систем обыкновенных уравнений Более подробная информация о решателях систем ДУ пакета MATLAB можно найти в соответствующем разделе Help (рис 5.9).

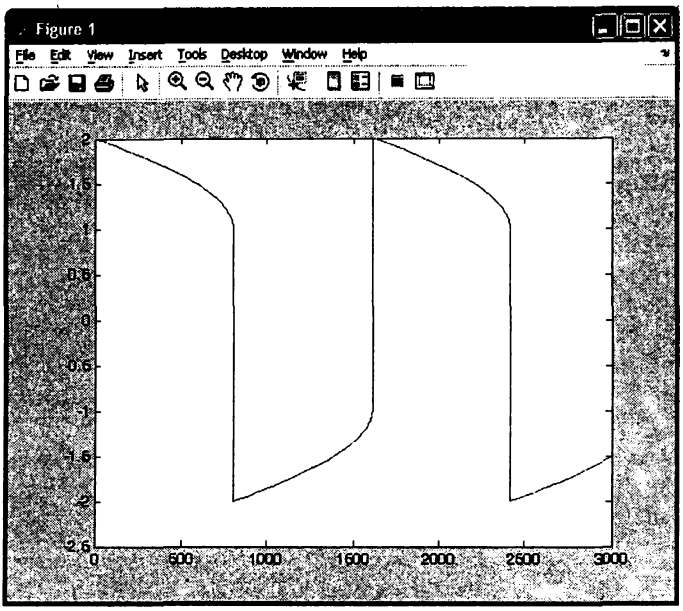


Рис. 5.8. Решение уравнения Ван-Дер-Поля с начальными условиями $x(0) = 2$, $x'(0) = 0$

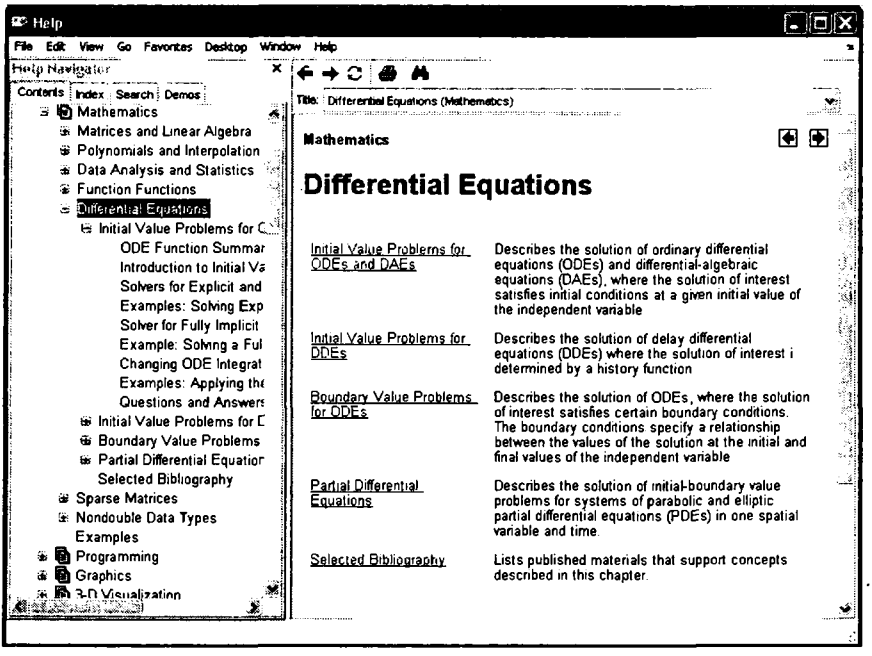


Рис. 5.9. Окно помощи в режиме просмотра информации о функциях, возвращающих решение дифференциальных уравнений

Вопросы для самопроверки

1. Чем отличаются друг от друга результаты, возвращенные функциями **exp** и **expm**, **log** и **logm**, **sqrt** и **sqrtm**, соответственно?
2. Какая функция MATLAB находит решение системы линейных уравнений, отображая результаты, получаемые на каждом шаге преобразований расширенной матрицы системы?
3. Какие функции, возвращают: число обусловленности матрицы, определитель матрицы, ранг матрицы, ортонормированный базис?
4. Какие функции используются при работе с разреженными матрицами? За счет чего они обеспечивают боле эффективное использование оперативной памяти компьютера?
5. Какая функция MATLAB возвращает решение нелинейных уравнений, в которые входят функции, зависящее от одной переменной? Какой синтаксис вызова имеет данная функция?
6. Какая функция MATLAB находит наименьшее значение функции, зависящее от одной переменной, на заданном интервале? Какой синтаксис вызова имеет данная функция?
7. Какая функция MATLAB возвращает значение определенного интеграла, вычисленного по методу трапеций?
8. Чем отличается вычисление определенного интеграла от вычисления интеграла с переменным верхним пределом?
9. Какая функция MATLAB возвращает значение двойного интеграла?
10. Какие функции MATLAB возвращают численные решения обыкновенных дифференциальных уравнений? Чем определяется выбор конкретной функции?

Символьные вычисления в MATLAB

6.1. Введение

Несмотря на большие вычислительные возможности ядра MATLAB, существуют проблемы, для решения которых данных команд оказывается недостаточным. Для выхода из этой ситуации разработчики пакета предусмотрели возможность встраивать вновь создаваемые программные модули, ориентированные на решение конкретных задач в общую архитектуру пакета. После установки пакета расширения (Toolbox) обращение к новым функциям, расширяющим возможности пакета, осуществляется традиционным для MATLAB способом.

Пакеты расширения MATLAB производятся как фирмой The MathWorks Inc., USA — производителем данного продукта, так и сторонними производителями программного обеспечения. (Отметим, что создание такого пакета не встречает никаких принципиальных трудностей с точки зрения техники программирования). В настоящее время существуют десятки официально распространяемых пакетов расширения, среди которых пакеты: Partial Differential Equation Toolbox (пакет для решения дифференциальных уравнений в частных производных, зависящих от двух переменных); Statistic Toolbox (решение задач статистики), Femlab Toolbox (решение трехмерных уравнений математической физики); Image Processing Toolbox (решение задач обработки изображений); Fuzzy Logic Toolbox (решение задач методами нечеткой логики); Wavelet Toolbox (решение задач обработки сигналов и изображений методом вэйвлет-преобразований); Simulink (пакет для моделирования динамических систем) и др. Рассмотрение пакетов расширений выходит за рамки данного курса, так как описание возможностей и приемов работы для каждого из них требует отдельного пособия, сделав при этом исключение для пакета Symbolic Math Toolbox, которой предназначен для выполнения символьных вычислений. Данный пакет разработан фирмой Waterloo Maple Software, Канада. Для получения справки по командам пакета Symbolic Math Toolbox следует открыть соответствующий раздел Help (рис. 6.1).

6.2. Создание символьных переменных, выражений и матриц

Для создания символьной переменной используется функция **sym**, имеющая следующий синтаксис

```
имя_переменной = sym(имя_переменной)
```

Например, создадим две символьных переменных x и **alpha**:

```
>> x = sym('x')
 x =
 x
>> a=sym('alpha')
 a =
 alpha
>>
```

Для одновременного задания нескольких символьных переменных используется команда **syms**, обращение к которой имеет следующий вид:

```
syms имя_переменной1  имя переменной2  имя переменной3...
```

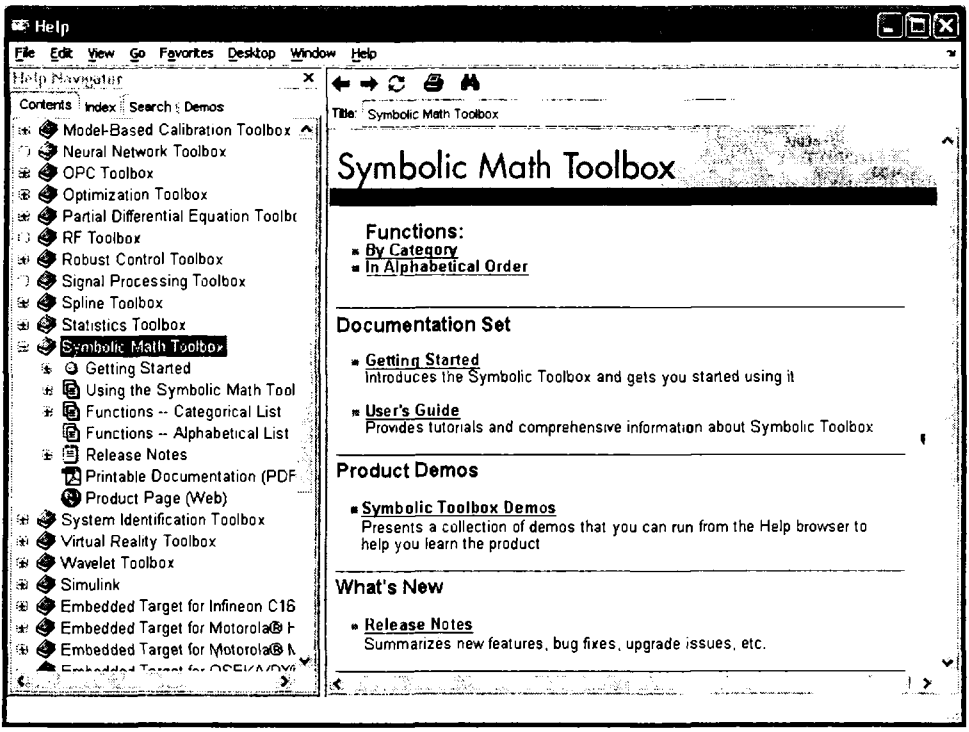


Рис. 6.1. Окно помощи в режиме просмотра информации о функциях, осуществляющих символьные преобразования

Например, для создания одновременно трех переменных **a**, **b** и **c** следует выполнить команду:

```
>> syms a b c
>>
```

Для создания действительных и комплексных используются команды

```
>> x = sym('x','real'); y = sym('y','complex');
```

Однако более эффективной с вычислительной точки зрения является следующий набор команд:

```
>> syms x y real
>> z = x + i*y
 z =
x+i*y
>>
```

Снятие наложенных ограничений на возможные значения переменных, например, определенной выше действительной переменной **x**, осуществляется командой

```
>> sym('x','unreal')
```

или

```
>> syms x unreal
```

Создание символьного выражения осуществляется командой

```
>> sym('символьное_выражение')
```

Например, для создания символьной переменной, содержащей выражение ax^2+bx+c, следует выполнить команду

```
>> f=sym('a*x^2+b*x+c')
```

Отметим, что в данном случае введенное выражение будет рассматриваться, как единая переменная. Для того, чтобы иметь возможность изменять значения коэффициентов и неизвестной, входящих в выражение ax^2+bx+c следует выполнить команды:

```
>> syms a b c x
>> f=a*x^2+b*x+c
f =
a*x^2+b*x+c
```

Для создания абстрактной функции $f(x)$ необходимо выполнить команду

```
>> clear
>> f = sym('f(x)')
 f =
 f(x)
```

Далее созданную абстрактную функцию $f(x)$ можно использовать для создания новых функций, например, для создания функции df, возвращающей значения первых производных необходимо выполнить следующие команды:

```
>> df = (subs(f,'x','x+h') - f)/'h' %
 df =
 (f(x+h)-f(x))/h
```

Созданная выше функция $df(x)$, рассматривается как единая переменная, для того, чтобы иметь возможность менять значения переменных **x**, **h**, следует описать данные переменные как символьные:

```
>> syms x h
>> df = (subs(f,x,x+h)-f)/h
 df =
 (f(x+h)-f(x))/h
```

При этом внешний вид функции $df(x)$ остается неизменным.

С помощью функции **sym** можно обращаться к стандартным функциям пакета Maple. Например, создадим функцию, возвращающую значение факториала числа:

```
>> kfac = sym('k!')
```

Для вычисления 6! или n! следует выполнить команды

```
>> syms k n
>> subs(kfac,k,6), subs(kfac,k,n)
 ans =
 720
 ans =
 n!
```

Если выражение зависит от одной переменной, то можно использовать более короткую команду:

```
subs(kfac,6)
```

Отметим, что для вычисления факториала числа также можно использовать функцию **prod**

```
>> prod(1:6)
 ans =
 720
```

Описание функции также можно размещать в m-файлах. Для примера приведем листинг файла **sinc.m**, содержащий описание функции, возвращающей символьные значения функции $\sin(x)/x$.

```
% листинг файла sinc.m
function z = sinc(x)
%SINC The symbolic sinc function
%     sin(x)/x. This function
%     accepts a sym as the input argument.
if isequal(x,sym(0))
```

```
  z = 1;
else
  z = sin(x)/x;
end
```

Для создания символьной матрицы необходимо создать символьные переменные, являющиеся элементами матрицы и затем создать матрицу, явно задав ее строки и столбцы:

```
>> syms a b c
>> A = [a b c; b c a; c a b]
A =
[ a, b, c ]
[ b, c, a ]
[ c, a, b ]
```

Далее с созданной символьной матрицей можно выполнять различные арифметические операции, например:

1. суммирование по строке;

```
>> sum(A(1,:))
ans =
  a+b+c
```

2. проверку равенства суммы элементов первой строки и суммы элементов второго столбца;

```
>> sum(A(1,:)) == sum(A(:,2))
ans =
  1
```

3. замену переменной **b** переменной **alpha**;

```
>> A = subs(A,b,alpha)
A =
[  a,   alpha,    c]
[ alpha,   c,  beta]
[  c,     a,  alpha]
```

6.3. Символьные вычисления

6.3.1. Символьное дифференцирование

Для вычисления производной функции $f(x)$ необходимо: 1) задать выражение, описывающее функцию; 2) обратиться к функции diff.

Пример 6.1.

Вычислить производную функции $\sin(ax)$ по переменной x

```
>> syms a x % описываем символьные переменные
>> f = sin(a*x) % задаем дифференцируемую функцию
>> diff(f) % вычисляем производную в символьном виде
```

```
ans =
 cos(a*x)*a
```

Пример 6.2.

Вычислить производную функции $\sin(ax)$ по параметру a

Для вычисления производной заданной выше функции по параметру a следует выполнить команду:

```
>> syms a x % описываем символьные переменные
>> f = sin(a*x) % задаем дифференцируемую функцию
>> diff(f,a) % вычисляем производную по параметру a
             % в символьном виде
ans =
cos(a*x)*x
```

Для вычисления вторых производных по переменной x и параметру a можно использовать команды:

```
>> diff(f,2)
 ans =
 -sin(a*x)*a^2
>> diff(f,x,2)
 ans =
 -sin(a*x)*a^2
```

Определив переменные **a**, **b**, **x**, **n**, **t**, и **theta** как символьные, далее можно определить символьные функции, например, x^n, $\sin(at+b)$, $\exp(i\cdot theta)$, $J_m(x)$ и затем в символьном виде вычислить их производные.

Пример 6.3.

Вычислить производную функции x^n

```
>> syms x y n % описываем символьные переменные
>> y=x^n % задаем функцию x^n
 y =
 x^n
>> diff(y,x)    вычисляем производную функции x^n в символьном виде
 ans =
 x^n*n/x
```

Пример 6.4.

Вычислить производную функции Бесселя второго рода $J_\upsilon(z)$ в символьном виде

```
>> syms nu z % описываем символьные переменные
b = besselj(nu,z); % задаем символьную функцию Бесселя порядка nu
 db = diff(b) % вычисляем производную функции Бесселя по переменной x
 db =
 -besselj(nu+1,z)+nu/z*besselj(nu,z)
```

Пример 6.5.

Вычислить производную функции $\sin(at+b)$

```
>> syms a b t % описываем символьные переменные
>> y=sin(a*t+b) % задаем символьную функцию sin(at+b)
 y =
 sin(a*t+b)
>> diff(y,t) % вычисляем производную функции sin(at+b) по переменной x
 ans =
 cos(a*t+b)*a
```

Пример 6.6.

Вычислить производную функции $e^{i\Theta}$ по переменной Θ

```
>> syms theta % описываем символьную переменную
>> y=exp(i*theta) % задаем символьную функцию exp(i·theta)
 y =
 exp(i*theta)
>> diff(y) % вычисляем производную функции exp(i·theta)
 ans =
 i*exp(i*theta)
>>
```

Операцию дифференцирования также можно применять к символьным матрицам, которая в данном случае выполняется поэлементно.

Пример 6.7.

Вычислить производную матрицы $\begin{pmatrix} \cos(ax) & \sin(ax) \\ -\sin(ax) & \cos(ax) \end{pmatrix}$

```
>> syms a x % описываем символьные переменные
 A = [cos(a*x),sin(a*x);-sin(a*x),cos(a*x)] % задаем сивольную
                                            % матрицу
 A =
 [ cos(a*x),  sin(a*x)]
 [-sin(a*x),  cos(a*x)]
>> diff(A) % вычисляем производную символьной матрицы
ans =
 [-sin(a*x)*a,  cos(a*x)*a]
 [-cos(a*x)*a, -sin(a*x)*a]
```

Для функциональной матрицы (матрицей Якоби)

$$\begin{pmatrix} \frac{\partial \varphi_1}{\partial t_1} & \frac{\partial \varphi_1}{\partial t_2} & \frac{\partial \varphi_1}{\partial t_m} \\ \frac{\partial \varphi_2}{\partial t_1} & \frac{\partial \varphi_2}{\partial t_2} & \frac{\partial \varphi_2}{\partial t_m} \\ \vdots & \vdots & \vdots \\ \frac{\partial \varphi_m}{\partial t_1} & \frac{\partial \varphi_m}{\partial t_2} & \frac{\partial \varphi_m}{\partial t_m} \end{pmatrix},$$

составленной из частных производных 1-го порядка вектор-функции

$$\Phi = \begin{bmatrix} \varphi_1(t_1, \dots t_m, t_{m+1}, \dots t_n) \\ \\ \varphi_m(t_1, \dots t_m, t_{m+1}, \dots t_n) \end{bmatrix},$$

по переменным $t_1, t_2 \dots t_m$, определитель (якобиан) которой для краткости обозначается символом

$$\frac{D(\varphi_1, \varphi_2, \dots \varphi_m)}{D(t_1, t_2, \quad t_m)},$$

используется функция **jacobian**. Напомним, что величина якобиана определяет величину растяжения (деформации) элементарного объема при переходе от переменных $(x_1, x_2, \dots, x_n)$ к переменным $(t_1, t_2, \dots, t_n)$.

Пример 6.8.

Вычислить матрицу Якоби и ее определитель при переходе от прямоугольной системы координат к сферической, что эквивалентно замене переменных: $x = r\cos(\theta)\cos(\varphi)$, $y = r\cos(\theta)\cos(\varphi)$, $z = r\sin(\varphi)$

```
>> syms r teta phi % описываем символьные переменные
 % задаем формулы перехода от прямоугольной системы координат
 % к сферической
>> x = r*cos(teta)*cos(phi); y = r*cos(teta)*sin(phi); z = r*sin(phi);
>> J = jacobian([x; y; z], [r teta phi]) % вычисляем якобиан
                                         % преобразования
 J =
 [    cos(teta)*cos(phi), -r*sin(teta)*cos(phi), -r*cos(teta)*sin(phi)]
 [    cos(teta)*sin(phi), -r*sin(teta)*sin(phi),  r*cos(teta)*cos(phi)]
 [         sin(phi),                0,           r*cos(phi)]
```

Отметим, что первая переменная в списке формальных параметров функции **jacobian** — вектор столбец, содержащий переменные $(x_1,x_2,...,x_n)$, вторая переменная — вектор-строка, содержащий переменные $(t_1,t_2,...,t_n)$.

6.3.2. Вычисление пределов

Продемонстрируем приемы вычисления пределов в MATLAB на примере вычисления следующих пределов.

Пример 6.9.

Вычислить предел $\lim\limits_{h\to 0}\dfrac{\sin(x+h)-\sin(x)}{h}$

```
>> syms x h % задаем символьные переменные
>> limit((sin(x+h)-sin(x))/h,h,0) % вычисляем предел
                                  % в символьном виде
ans =
 cos(x)
>>
```

Пример 6.10.

Вычислить предел $\lim\limits_{h\to\infty}\left(1+\dfrac{x}{n}\right)^n$

```
>> syms x n % задаем символьные переменные
>> limit((1+x/n)^n,n,inf) % вычисляем предел в символьном вид
 ans =
 exp(x)
```

Пример 6.11.

Вычислить предел $\lim\limits_{x\to 1}\dfrac{1}{1-x}$

```
>> syms x % задаем символьную переменную
>> limit(1/(1-x),x,1) % вычисляем предел в символьном виде
 ans =
 NaN
>>
```

Здесь переменная **NaN** означает, что предела функции $1/(1-x)$ в точке $x = 1$ не существует.

Пример 6.12.

Вычислить предел $\lim\limits_{x\to 1-}\dfrac{1}{1-x}$

```
>> syms x % задаем символьную переменную
>> limit(1/(1-x),x,1,'left') % вычисляем предел
                             % в символьном виде
 ans =
 inf
```

Пример 6.13.

Вычислить предел $\lim_{x \to 1+} \frac{1}{1-x}$

```
>> syms x % задаем символьную переменную
>> limit(1/(1-x),x,1,'right') % вычисляем предел
                              % в символьном виде
 ans =
 -inf
```

6.3.3. Символьное интегрирование

Для вычисления интегралов в символьном виде используется функция **int**, имеющая следующий синтаксис:

```
int(f),
int(f,[u]),
int(f,[u,a,b]),
```

где **f** — символьная подынтегральная функция, необязательные переменные: **u** — переменная интегрирования, **a** — нижний предел интегрирования, **b** — верхний предел интегрирования.

Продемонстрируем приемы вычисления интегралов в MATLAB на следующих примерах.

Пример 6.14.

Вычислить интеграл $\int \frac{dx}{a^2 + (bx)^2}$

```
>> syms a b x % задаем символьные переменные
>> int(1/(a^2+(b*x)^2)) % вычисляем интеграл в символьном виде
 ans =
 1/a/b*atan(b*x/a)
```

Пример 6.15.

Вычислить интеграл $\int_0^{a/b} \frac{dx}{a^2 + (bx)^2}$

```
>> syms a b x % задаем символьные переменные
>> int(1/(a^2+(b*x)^2),0,a/b) % вычисляем интеграл
                              % в символьном виде
```

```
ans =
1/4*pi/a/b
```

Пример 6.16.

Вычислить интеграл $\int_0^1 J_1(z)dz$

```
>> syms z % задаем символьную переменную
>> int(besselj(1,z),0,1)
 ans =
 1/4*hypergeom([1],[2, 2],-1/4)    вычисляем интеграл
                                   в символьном виде
```

Здесь **hypergeom** — гипергеометрическая функция.

Для получения численного значения интеграла следует использовать команду **double**:

```
>> a = double(int(besselj(1,z),0,1))
a =
  0.2348
```

Помимо описанных выше функций для вычисления интеграла в MATLAB имеется возможность вычислять интегралы, зависящие от параметров.

Пример 6.17.

Вычислить интеграл $\int_{-\infty}^{+\infty} e^{-(kx)^2}dx$

```
>> syms x k; % задаем символьные переменные
>> f = exp(-(k*x)^2); % задаем подынтегральную функцию
                      % в символьном виде
>> int(f,x,-inf,inf) % вычисляем значение интеграла
Warning: Explicit integral could not be found.
>> In C:\MATLAB6p5\toolbox\symbolic\@sym\int.m at line 58
 ans =
 int(exp(-k^2*x^2),x = -inf    inf)
```

Здесь MATLAB не смог вычислить интеграл потому, что величина интеграл зависит от знака переменной **k**, который при определении переменной **k** как символьной, остается незаданным. Следовательно, для вычисления значений интегралов, зависящих от параметра, следует при инициализации символьной переменной указывать ее знак, например:

```
>> clear
>> k=sym('k','positive');x=sym('x');
>> f = exp(-(k*x)^2);
>> int(f,x,-inf,inf)
 ans =
 1/k*pi^(1/2)
```

Отметим, что по не вполне понятным для нас причинам возможности создать символьную переменную, принимающую только отрицательные значения, разработчики MATLAB не предусмотрели. Из данной ситуации у пользователя имеется два выхода:

1. напрямую обратиться к команде Maple, позволяющей наложить ограничения на возможные значения символьной переменной;

```
>> clear
>> syms x k
>> maple('assume(k < 0);'); % накладываем ограничения
                            % на переменную k
>> maple('about(k)')    получаем информацию о наложенных
                        ограничениях на переменную k
ans =
Originally k, renamed k~:  is assumed to be:
RealRange(-inf,Open(0))
>> ff = exp(-(k*x)^2)
 ff =
 exp(-k^2*x^2)
>> int(ff,x,-inf,inf)
 ans =
-1/k*pi^(1/2)
```

2. внести соответствующие изменения в файл sym.m, находящийся в папке C:\MATLAB6p5\Toolbox\symbolic\@sym\. Листинг файла с sym.m с внесенными в него изменениями представлен в Прил. 1.

6.3.4. Вычисление сумм рядов и произведений

Для вычисления суммы ряда в MATLAB используется команда **symsum**.

Пример. 6.18.

Вычислить сумму ряда $\sum_{k=1}^{\infty} \frac{1}{k^4}$.

```
>> syms x k % задаем символьные переменные
>> s1 = symsum(1/k^4,1,inf) % вычисляем сумму ряда
 s1 =
 1/90*pi^4
```

Пример. 6.19.

Вычислить сумму ряда $\sum_{k=1}^{10} \frac{1}{k^4}$.

```
>> s1 = symsum(1/k^4,1,10)
 s1 =
 43635917056897/40327580160000
```

Для вычисления произведения нужно использовать команду **product** из пакета Maple.

Пример. 6.20.

Вычислить произведение $\prod_{k=0}^{m}(n+k)$

```
>> maple('product( n+k, k=0..m );')
 ans =
 gamma(n+m+1)/gamma(n)
```

Здесь **gamma** — гамма-функция.

Пример. 6.21.

Вычислить произведение $\prod_{k=0}^{4} a[k]$

```
>> maple('product( a[k], k=0..4 )')
 ans =
 a[0]*a[1]*a[2]*a[3]*a[4]
```

6.3.5. Разложение функции в ряды

Пример. 6.22.

Разложить в ряд Тейлора функцию $f(x) = \dfrac{1}{5 + 4\cos(x)}$

```
>> syms x % задаем символьные переменные
>> f = 1/(5+4*cos(x)); % задаем сивольную функцию
>> T = taylor(f,8) % раскладываем функцию в ряд Тэйлора
 T =
 1/9+2/81*x^2+5/1458*x^4+49/131220*x^6
```

Для записи ряда в форме с коэффициентами, записанными в виде обыкновенных дробей, используется команда **pretty**:

```
>> pretty(T)
```

Пример. 6.23.

Получить первые два члена разложения функции $f(x) = \dfrac{1}{5 + 4\cos(x)}$ в точке $x = 1$.

```
>> syms x % задаем символьную переменную
>> f = 1/(5+4*cos(x)); % задаем сивольную функцию
>> T = taylor(f,2,1) % раскладываем функцию в ряд Тэйлора
 T =
 1/(5+4*cos(1))+4/(5+4*cos(1))^2*sin(1)*(x-1)
```

Пример. 6.24.

Разложить и построить график частичной суммы ряда Тейлора ($n = 5$) функции $f(x) = \dfrac{1}{5 + 4\sin x}$ в окрестности точки $x = 1$.

```
>> syms x % задаем символьную переменную
>> g = 1/(4+5*sin(x)); % задаем сивольную функцию
>> t = taylor(g,5,2); % раскладываем функцию в ряд Тэйлора
>> xd = 1:0.05:3; % задаем координаты точек, в которых
                  % вычисляются значения функции и частичной
                  % суммы ряда
>> yd = subs(g,x,xd); % строим графики исходной функции
                        и частичной суммы ряда Тэйлора % (рис. 6.2)
>> ezplot(t, [1,3]);
>> hold on;
>> plot(xd, yd, 'r-.')
>> title('Taylor approximation vs. actual function');
>> legend('Function','Taylor')
```

Рис. 6.2

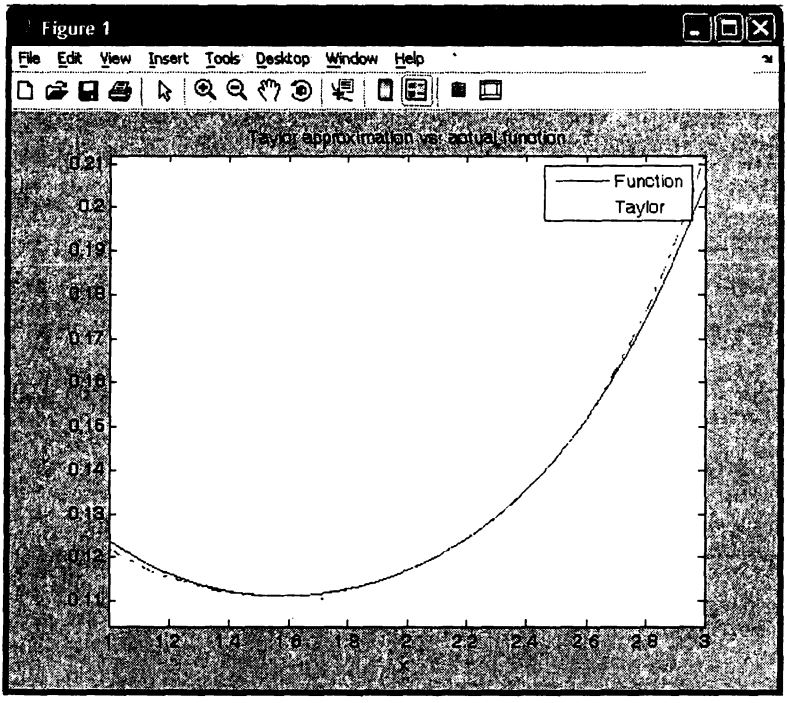


Рис. 6.2. Графики функции $f(x) = \dfrac{1}{5 + 4\sin x}$ и частичной суммы разложение данной функции в ряд Тейлора

Пример. 6.25.

Разложить в ряд $f(x) = \frac{e^x}{x}$ функцию в точке $x = 0$.

```
>> maple('convert(series(exp(x)/x, x=0, 8 ),polynom)')
 ans =
  1/x+1+1/2*x+1/6*x^2+1/24*x^3+1/120*x^4+1/720*x^5+1/5040*x^6
```

6.4. Упрощение выражений и подстановки

Для приведения коэффициентов при одинаковых степенях переменной или выражения используется команда **collect**.

Пример 6.26.

Упростить выражение $a\log x - x\log x - x$.

```
>> syms a x; % задаем символьные переменные
>> f=a*log(x)-log(x)*x-x; % задаем упрощаемое выражение
>> collect(f,log(x)) % собираем коэффициенты при logx
 ans =
 (a-x)*log(x)-x
```

Пример 6.27.

Упростить выражение $xy + axy + yx^2 - ayx^2 + ax$.

```
>> syms x y a; % задаем символьные переменные
>> p= x*y+a*x*y+y*x^2-a*y*x^2+x+a*x % задаем упрощаемое
                                     % выражение
 p =
  x*y+a*x*y+y*x^2-a*y*x^2+x+a*x
>> collect(p,'x') % собираем коэффициенты при одинаковых
                  % степенях x
 ans =
 (y-a*y)*x^2+(y+a*y+1+a)*x
>> collect(p,'y') % собираем коэффициенты при одинаковых
                  % степенях y
 ans =
 (x^2-a*x^2+x+a*x)*y+x+a*x
```

Для разложения выражения на сумму и произведение простых сомножителей используется команда **expand**.

Пример 6.28.

Разложить выражение $\sin(x+y)$ на сумму и произведение простых сомножителей

```
>> syms x y; % задаем символьные переменные
>> expand(sin(x+y)) % раскладываем выражение на произведение
                    % и сумму простых сомножителей
 ans =
  sin(x)*cos(y)+cos(x)*sin(y)
```

Пример 6.29.

Разложить выражение $\cos(3\arccos x)$ на сумму и произведение простых сомножителей

```
>> syms x y; % задаем символьную переменную
>> expand(cos(3*acos(x))) % раскладываем выражение на
                          % произведение и сумму простых
                          % сомножителей
 ans =
 4*x^3-3*x
```

Пример 6.30.

Разложить выражение e^{a+b} на произведение простых сомножителей

```
>> syms a b; % задаем символьные переменные
>> expand(exp(a+b)) % раскладываем выражение на произведение
                    % и сумму простых сомножителей
 ans =
 exp(a)*exp(b)
```

Пример 6.31.

Разложить выражение на сумму и произведение простых сомножителей выражение $(x-1)(x-2)(x-3)$

```
>> syms x p; % задаем символьные переменные
>> p=(x-1)*(x-2)*(x-3) % задаем символьное выражение
 p =
(x-1)*(x-2)*(x-3)
>> expand(p) % раскладываем выражение на произведение
             % и сумму простых сомножителей
 ans =
 x^3-6*x^2+11*x-6
```

Для разложения полинома на множители используется команда **factor**.

Пример 6.32.

Разложить на множители полином $x^3 + 2$

```
>> maple('factor(x^3+2.0)')
 ans =
 (x+1.259921)*(x^2-1.259921*x+1.587402)
```

Для разложения полиномов над полем вещественных или комплексных чисел следует непосредственно обратиться к функции **factor** пакета **Maple**.

Пример 6.33.

Разложить на множители полином $x^3 + 2$ над полем действительных чисел и отобразить полученный результат с семью цифрами после запятой

```
>> maple('factor(x^3+2.0,real)') % разложение над полем
                                 % действительных чисел
                               и отображение 7 цифр после запятой
 ans =
 (x+1.259921050)*(x^2-1.259921050*x+1.587401052)
```

Пример 6.34.

Разложить на множители полином $x^3 + 2$ над полем комплексных чисел

```
>> maple('factor(x^3+2.0,complex)')
                                % разложение над полем комплексных
                                % чисел
 ans =
 (x+1.259921)*(x-.6299605+1.091124*i)*(x-.6299605-1.091124*i)
```

Для упрощения выражений используется команды **simplify** и **simple**, которая выводит все возможные варианты преобразования выражения.

Пример 6.35.

Упростить выражение $\dfrac{1-x^2}{1-x}$

```
>> syms x % задаем символьную переменную
>> p=(1-x^2)/(1-x) % задаем символьное выражение
 p =
  (1-x^2)/(1-x)
>> simplify(p)
 ans =
   x+1
```

Пример 6.36.

Упростить выражение $\log(x \cdot y)$

```
>> syms x y positive % задаем символьные переменные,
                     % принимающие положительные значения
>> simplify(log(x*y)) % упрощаем символьное выражение
 ans =
 log(x)+log(y)
```

Пример 6.37.

Упростить выражение $\sin^2 x + \cos^2 x$

```
>> syms x y positive % задаем символьные переменные
>> simple(cos(x)^2 + sin(x)^2) % упрощаем символьное выражение
 simplify:
 1
 radsimp:
 cos(x)^2+sin(x)^2
 combine(trig)
 1
 factor:
 cos(x)^2+sin(x)^2
 expand:
 cos(x)^2+sin(x)^2
 combine:
 1
 convert(exp)
(1/2*exp(i*x)+1/2/exp(i*x))^2-1/4*(exp(i*x)-1/exp(i*x))^2
 convert(sincos):
 cos(x)^2+sin(x)^2
 convert(tan):
(1-tan(1/2*x)^2)^2/(1+tan(1/2*x)^2)^2+4*tan(1/2*x)^2/(1+tan(1/2*x)^2)^2
 collect(x):
 cos(x)^2+sin(x)^2
 ans =
 1
```

Для подстановки или замены переменных в выражении используются команды **subs** и **subexr**.

Пример 6.38.

Решить уравнение $x^3 + a \cdot x + 1 = 0$

```
>> syms a x % задаем символьные переменные
s = solve(x^3+a*x+1)  % находим корни уравнения x^3+a·x+1=0
s =
[1/6*(-108+12*(12*a^3+81)^(1/2))^(1/3)-2*a/(-108+12*(12*a^3+81)^
(1/2))^(1/3)]
[-1/12*(-108+12*(12*a^3+81)^(1/2))^(1/3)+a/(-108+12*(12*a^3+81)^
(1/2))^(1/3)+1/2*i*3^(1/2)*(1/6*(-108+12*(12*a^3+81)^(1/2))^(1/3
)+2*a/(-108+12*(12*a^3+81)^(1/2))^(1/3))]
[-1/12*(-108+12*(12*a^3+81)^(1/2))^(1/3)+a/(-108+12*(12*a^3+81)^
(1/2))^(1/3)-1/2*i*3^(1/2)*(1/6*(-108+12*(12*a^3+81)^(1/2))^(1/3
```

```
)+2*a/(-108+12*(12*a^3+81)^(1/2))^(1/3))]
>> r = subexpr(sigma) % выделяем общий сомножитель и заменяем его
                      % переменной sigma
sigma = -108+12*(12*a^3+81)^(1/2)
r =
[1/6*sigma^(1/3)-2*a/sigma^(1/3)]
[ -1/12*sigma^(1/3)+a/sigma^(1/3)+
1/2*i*3^(1/2)*(1/6*sigma^(1/3)+2*a/sigma^(1/3))]
[-1/12*sigma^(1/3)+a/sigma^(1/3)-1/2*i*3^(1/2)*(1/6*sigma^(1/3)+
2*a/sigma^(1/3))]
```

6.5. Управление точностью вычислений

Точность вычислений (число значащих цифр после запятой, используемых в численных расчетах) устанавливается командой **digits**, вычисления с установленной точностью выполняет команда **vpa** (абревиатура фразы Variable Precition Arithmetic(арифметика с переменной точностью)):

```
>> digits(30)
>> vpa(pi)
 ans =
 3.14159265358979323846264338328
>>
>> digits(50)
>> vpa(exp(1))
 ans =
 2.71828182845904553488480814849026501178741455078l3
>>
```

6.6. Операции линейной алгебры

С символьными матрицами можно выполнять действия, аналогичные действиям с числовыми матрицами.

Пример 6.39.

Вычислить квадрат матрицы $G = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$. Показать, что матрица G является ортогональной

```
>> syms t; % задаем символьную переменную
>> G = [cos(t) sin(t); -sin(t) cos(t)] % создаем матрицу 2×2
 G =
 [  cos(t),  sin(t)]
 [ -sin(t),  cos(t)]
>> A = G^2 % вычисляем квадрат матрицы
 A =
 [ cos(t)^2-sin(t)^2,   2*cos(t)*sin(t)]
 [  -2*cos(t)*sin(t), cos(t)^2-sin(t)^2]
```

```
>> A = simple(A)   упрощаем матрицу A
 A =
 [  cos(2*t),  sin(2*t)]
 [ -sin(2*t),  cos(2*t)]
>> I = simple(G.' *G) % проверяем ортогональность матрицы G
 I =
 [ 1, 0]
 [ 0, 1]
```

Пример. 6.40.

Выполнить линейные операции над символьной матрицей Гильберта, элементы которой вычисляются по формуле $a_{ij} = \frac{1}{i+j-1}$

```
>> H = hilb(3)   создаем матрицу Гильберта
 H =
  1.0000    0.5000    0.3333
  0.5000    0.3333    0.2500
  0.3333    0.2500    0.2000
>> H=sym(H) % преобразуем числовую матрицу в символьную
 ans =
 [   1, 1/2, 1/3]
 [ 1/2, 1/3, 1/4]
 [ 1/3, 1/4, 1/5]
>> inv(H) % вычисляем матрицу, обратную к символьной
 ans =
 [    9,  -36,   30]
 [  -36,  192, -180]
 [   30, -180,  180]
>> det(H) % вычисляем детерминант матрицы
 ans =
1/2160
>> H(1,1)=8/9   заменяем первый элемент матрицы H
H =
 [ 8/9, 1/2, 1/3]
 [ 1/2, 1/3, 1/4]
 [ 1/3, 1/4, 1/5]
>> [V E]=eig(H) % вычисляем матрицы собственных векторов и
                % собственных значений
V =
[ 28/153+2/153*12589^(1/2),   28/153-2/153*12589^(1/2),      1]
[          1,                          1,                   -4]
[ 292/255-1/255*12589^(1/2), 292/255+1/255*12589^(1/2),   10/3]
E =
[ 32/45+1/180*12589^(1/2),                0,                  0]
[                       0, 32/45-1/180*12589^(1/2),           0]
[ 0,                                       0,                  0]
>> Td = double(V)   вычисляем численные значения элементов
                    матриц V и E
Td =
1.6497   -1.2837    1.0000
1.0000    1.0000   -4.0000
0.7051    1.5851    3.3333
```

```
>> Ed = double(E)
Ed =
1.3344        0             0
0          0.0878           0
0             0             0
>> poly(H)   получаем характеристический полином матрицы H
 ans =
 x^3-64/45*x^2+253/2160*x
          % находим корни характеристического полинома
>> syms x
g = simple(factor(poly(H)));
solve(g)
 ans =
[                          0]
[ 32/45+1/180*12589^(1/2)]
[ 32/45-1/180*12589^(1/2)]
```

Приведем пример вычисления функции от символьной матрицы.

Пример 6.41.

Вычислить функцию e^{At}, где $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, и ее собственный вектор

```
>> syms t % задаем символьную переменную
>> A = sym([0 1; -1 0]); % задаем матрицу A
>> G = expm(t*A) % вычисляем функцию матрицы A
 G =
  [  cos(t),  sin(t)]
  [ -sin(t),  cos(t)]
>> g = eig(G) % вычисляем собственные векторы
 g =
  [ cos(t)+(cos(t)^2-1)^(1/2)]
  [ cos(t)-(cos(t)^2-1)^(1/2) ]
>> for  j = 1:4 % упрощаем собственный вектор g различными
                % способами
   [g,how] = simple(g)
end
 g =
  [ cos(t)+(-sin(t)^2)^(1/2)]
  [ cos(t)-(-sin(t)^2)^(1/2)]
 how =
  simplify
 g =
  [ cos(t)+i*sin(t)]
  [ cos(t)-i*sin(t) ]
 how =
  radsimp
 g =
  [   exp(i*t) ]
  [ 1/exp(i*t)]
 how =
 convert(exp)
 g =
  [  exp(i*t)]
```

```
  [ exp(-i*t)]
how =
combine
```

Приведем пример приведения матрицы к жордановой форме.

Пример 6.42.

Привести к жордановой форме матрицу

$$\begin{pmatrix} 12 & 32 & 66 & 116 \\ -25 & -76 & -164 & -294 \\ 21 & 66 & 143 & 256 \\ -6 & -19 & -41 & -73 \end{pmatrix}$$

```
% задаем символьную матрицу
>> A = sym([12,32,66,116;-25,-76,-164,-294;21,66,143,256;-6,-19,-41,-73])
A =
 [   12,   32,   66,  116]
 [  -25,  -76, -164, -294]
 [   21,   66,  143,  256]
 [   -6,  -19,  -41,  -73]
% вычисляем матрицы V и J такие, что A=V J V^-1
>> [V,J] = jordan(A)
 V =
   [   4,  -2,   4,   3]
   [  -6,   8, -11,  -8]
   [   4,  -7,  10,   7]
   [  -1,   2,  -3,  -2]
 J =
  [ 1, 1, 0, 0]
  [ 0, 1, 0, 0]
  [ 0, 0, 2, 1]
  [ 0, 0, 0, 2]
% проверяем правильность вычисления матриц V и J
>> V*J*V^-1
 ans =
  [   12,   32,   66,  116]
  [  -25,  -76, -164, -294]
  [   21,   66,  143,  256]
  [   -6,  -19,  -41,  -73]
```

Приведем примеры разложения матриц по сингулярным числам.

Пример 6.43.

Разложить квадратную матрицу A, элементы которой вычисляются по формуле $a_{ij} = \dfrac{1}{i + j - \dfrac{1}{2}}$, $n = 5$, по сингулярным числам

```
% создаем заданную матрицу
>> i=1:5;j=1:5;
>> [I J]=meshgrid(i,j);
>> A=1./(I+J-1/2) % вычисляем числовую матрицу A
 A =
    0.6667    0.4000    0.2857    0.2222    0.1818
    0.4000    0.2857    0.2222    0.1818    0.1538
    0.2857    0.2222    0.1818    0.1538    0.1333
    0.2222    0.1818    0.1538    0.1333    0.1176
    0.1818    0.1538    0.1333    0.1176    0.1053
>> A=sym(1./(I+J-1/2))   вычисляем символьную матрицу A
 A =
 [  2/3,  2/5,  2/7,  2/9, 2/11]
 [  2/5,  2/7,  2/9, 2/11, 2/13]
 [  2/7,  2/9, 2/11, 2/13, 2/15]
 [  2/9, 2/11, 2/13, 2/15, 2/17]
 [ 2/11, 2/13, 2/15, 2/17, 2/19]
>> [U,S,V] = svd(A); % вычисляем матрицы U, S, V, где S -
                     % диагональная матрица, такие, что A=U·S·V'
>> double(U)   отображаем матрицу U
ans =
   -0.7026    0.6523   -0.2756   -0.0697   -0.0102
   -0.4744   -0.1593    0.7052    0.4815    0.1430
   -0.3629   -0.3790    0.2064   -0.6251   -0.5396
   -0.2954   -0.4442   -0.2523   -0.2922    0.7526
   -0.2496   -0.4563   -0.5661    0.5359   -0.3490
>> double(S) % отображаем матрицу S
ans =
1.2423       0         0         0         0
  0       0.1244       0         0         0
  0         0       0.0059      0         0
  0         0         0       0.0001      0
  0         0         0         0       0.0000
>> double(V) % отображаем матрицу V
ans =
-0.7026    0.6523   -0.2756   -0.0697   -0.0102
-0.4744   -0.1593    0.7052    0.4815    0.1430
-0.3629   -0.3790    0.2064   -0.6251   -0.5396
-0.2954   -0.4442   -0.2523   -0.2922    0.7526
-0.2496   -0.4563   -0.5661    0.5359   -0.3490
>> U*S*V'-double(A) % проверяем правильность разложения
ans =
[ -.4e-31,      0,  .3e-31,  .2e-31,  .5e-31]
[  .1e-31,-.2e-31,  .2e-31,  .1e-31,  .2e-31]
[  .1e-31,      0, -.1e-31,       0,  .1e-31]
[  .1e-31,      0,  .1e-31,    0,          0]
[  .1e-31,      0,      0,   .1e-31,  .1e-31]
```

Пример 6.44.

Разложить прямоугольную матрицу A, элементы которой вычисляются по формуле $a_{ij} = \dfrac{1}{i+j-\dfrac{1}{2}}$ размерности $m \times n$, $m = 4$, $n = 5$ по сингулярным числам

```
% создаем заданную матрицу
>> i=1:5;j=1:4;
>> [I J]=meshgrid(i,j);
>> A=sym(1./(I+J-1/2))
 A =
 [  2/3,  2/5,  2/7,  2/9, 2/11]
 [  2/5,  2/7,  2/9, 2/11, 2/13]
 [  2/7,  2/9, 2/11, 2/13, 2/15]
 [  2/9, 2/11, 2/13, 2/15, 2/17]
>> [U,S,V] = svd(A); % вычисляем матрицы U, S, V,
                     % где S – диагональная матрица, такие, что A=U·S·V'
>> double(U)
 ans =
   -0.7263    0.6421   -0.2409   -0.0466
   -0.4896   -0.2471    0.7437    0.3823
   -0.3743   -0.4802    0.0020   -0.7933
   -0.3044   -0.5442   -0.6236    0.4715
>> double(S)
 ans =
    1.2031         0         0         0
         0    0.1097         0         0
         0         0    0.0044         0
         0         0         0    0.0001
>> double(V)
 ans =
   -0.7104    0.6481   -0.2664   -0.0653    0.0093
   -0.4729   -0.1769    0.7106    0.4708   -0.1359
   -0.3584   -0.3871    0.1883   -0.6367    0.5299
   -0.2897   -0.4436   -0.2653   -0.2756   -0.7569
   -0.2436   -0.4495   -0.5641    0.5410    0.3574
>> S(4,5)=0;   расширяем матрицу S до матрицы размера 4×5,
               имеющей вид (S 0; 0 0)
>> U*S*V'-A   проверяем правильность разложения
 ans =
 [ -.8e-31, -.4e-31,  .2e-31, -.1e-31,  .2e-31]
 [  .9e-31, -.3e-31,  .2e-31, -.1e-31, -.2e-31]
 [ -.4e-31,  .3e-31,  .1e-31, -.2e-31,  .1e-31]
 [ -.3e-31, -.3e-31,  .2e-31, -.1e-31,       0]
```

Пример 6.45.

Разложить прямоугольную матрицу A, элементы которой вычисляются по формуле $a_{ij} = \dfrac{1}{i + j - \dfrac{1}{2}}$ размерности $m \times n$, $m = 5$, $n = 4$ по сингулярным числам

```
% создаем заданную матрицу
>> i=1:4;j=1:5;
>> [I J]=meshgrid(i,j);
>> A=sym(1./(I+J-1/2))
A =
[  2/3,  2/5,  2/7,  2/9]
[  2/5,  2/7,  2/9, 2/11]
[  2/7,  2/9, 2/11, 2/13]
[  2/9, 2/11, 2/13, 2/15]
[ 2/11, 2/13, 2/15, 2/17]
>> [U,S,V] = svd(A); % вычисляем матрицы U, S, V,
                     % где S – диагональная матрица, такие, что A=U·S·V
>> double(U)
ans =
-0.7104    0.6481   -0.2664   -0.0653   -0.0093
-0.4729   -0.1769    0.7106    0.4708    0.1359
-0.3584   -0.3871    0.1883   -0.6367   -0.5299
-0.2897   -0.4436   -0.2653   -0.2756    0.7569
-0.2436   -0.4495   -0.5641    0.5410   -0.3574
>> double(S)
ans =
1.2031         0         0         0
0         0.1097         0         0
0              0    0.0044         0
0              0         0    0.0001
>> double(V)
ans =
-0.7263    0.6421   -0.2409   -0.0466
-0.4896   -0.2471    0.7437    0.3823
-0.3743   -0.4802    0.0020   -0.7933
-0.3044   -0.5442   -0.6236    0.4715

>> S(5,4)=0;   расширяем матрицу S до матрицы размера 5×4,
               имеющей вид (S 0; 0 0)
>> U*S*V'-A   проверяем правильность разложения
ans =
[ -.4e-31,  .6e-31, -.2e-31,  .2e-31]
[       0,  .2e-31,  .1e-31,       0]
[       0, -.2e-31,  .1e-31,       0]
[  .1e-31, -.2e-31,  .1e-31,  .1e-31]
[ -.1e-31,  .1e-31,  .1e-31,  .1e-31]
```

6.7. Решение алгебраических уравнений и систем алгебраических уравнений в символьном виде

Для нахождения решение уравнения в символьном виде используется команда **solve**.

Пример. 6.46.

Найти решение уравнения $a_0x^3 + a_1x^2 + a_2x + a_3 = 0$ относительно переменной x

```
>> syms a0 a1 a2 a3 x   задаем символьные переменные
>> S = a0*x^3+a1*x^2 + a2*x + a3; % задаем уравнение
>> solve(S,x) % находим решение уравнения относительно
              % переменной x
 ans =
[1/6/a0*(36*a2*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a2
^2*a1^2-18*a2*a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3)-2
/3*(3*a2*a0-a1^2)/a0/(36*a2*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)*
(4*a2^3*a0-a2^2*a1^2-18*a2*a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/2
)*a0)^(1/3)-1/3*a1/a0][
-1/12/a0*(36*a2*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a
2^2*a1^2-18*a2*a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3)+
1/3*(3*a2*a0-a1^2)/a0/(36*a2*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)
*(4*a2^3*a0-a2^2*a1^2-18*a2*a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/
2)*a0)^(1/3)-1/3*a1/a0+1/2*i*3^(1/2)*(1/6/a0*(36*a2*a1*a0-108*a3
*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a2^2*a1^2-18*a2*a1*a0*a3+27*a
3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3)+2/3*(3*a2*a0-a1^2)/a0/(36*a2
*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a2^2*a1^2-18*a2*
a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3))][
-1/12/a0*(36*a2*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a
2^2*a1^2-18*a2*a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3)+
1/3*(3*a2*a0-a1^2)/a0/(36*a2*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)
*(4*a2^3*a0-a2^2*a1^2-18*a2*a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/
2)*a0)^(1/3)-1/3*a1/a0-1/2*i*3^(1/2)*(1/6/a0*(36*a2*a1*a0-108*a3
*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a2^2*a1^2-18*a2*a1*a0*a3+27*a
3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3)+2/3*(3*a2*a0-a1^2)/a0/(36*a2
*a1*a0-108*a3*a0^2-8*a1^3+12*3^(1/2)*(4*a2^3*a0-a2^2*a1^2-18*a2*
a1*a0*a3+27*a3^2*a0^2+4*a3*a1^3)^(1/2)*a0)^(1/3))]
>> solve(S,a0)
 ans =
  -(a1*x^2+a2*x+a3)/x^3
```

Пример. 6.47.

Найти решение уравнения $a_0x^3 + a_1x^2 + a_2x + a_3 = 0$ относительно коэффициента a_0

```
>> syms a0 a1 a2 a3 x   задаем символьные переменные
>> S = a0*x^3+a1*x^2 + a2*x + a3; % задаем уравнение
>> solve(S,a0) % находим решение уравнения относительно
               % переменной x
 ans =
  -(a1*x^2+a2*x+a3)/x^3
```

Пример. 6.48.

Найти решение уравнения $\cos(2x)+\sin(2x)=1$

```
>> solve('cos(2*x)+sin(2*x)=1',x)
 ans =
[ 1/4*pi]
[      0]
```

Отметим, что функция **solve** возвращает только значения корней, которые находятся в интервале $[-\pi;\pi]$. Для получения всех решений тригонометрических уравнений следует непосредственно использовать команды пакета Maple.

```
>> maple('_EnvAllSolutions:=true');
                                  % включаем режим поиска корней
                                  % тригонометрического уравнения
                                  % на всей числовой оси

>> maple('solve(cos(2*x)+sin(2*x)=1,x)')
 ans =
   1/4*pi+pi*_Z, pi*_Z
```

Здесь _Z — переменная целого типа.

Функция **solve** может также возвращать решения систем алгебраических уравнений.

Пример. 6.49.

Найти решение системы $\begin{cases} x^2y^2 = 1, \\ x - \dfrac{y}{2} = \alpha \end{cases}$

```
>> syms x y alpha % задаем символьные переменные
>> [X,Y] = solve(x^2*y^2-1, x-y/2-alpha) % получаем решение
                                         % системы
                                         % нелинейных уравнений
X =
[ 1/2*alpha+1/2*(alpha^2+2)^(1/2)]
[ 1/2*alpha-1/2*(alpha^2+2)^(1/2)]
[ 1/2*alpha+1/2*(alpha^2-2)^(1/2)]
[ 1/2*alpha--1/2*(alpha^2-2)^(1/2)]
Y =
```

```
[ -alpha+(alpha^2+2)^(1/2)]
[ -alpha-(alpha^2+2)^(1/2)]
[ -alpha+(alpha^2-2)^(1/2)]
[ -alpha-(alpha^2-2)^(1/2)]
```

6.8. Решение обыкновенных дифференциальных уравнений и систем обыкновенных дифференциальных уравнений

Для решения ОДУ и систем ОДУ в MATLAB используется функция **dsolve**.

Пример. 6.50.

Найти общее решение ОДУ $y' = 1 + y^2$

```
>> dsolve('Dy=1+y^2')
 ans =
  tan(t+C1)
```

Пример. 6.51.

Найти решение задачи Коши ОДУ $y' = 1 + y^2$ с начальным условием $y(0) = 1$

```
>> y = dsolve('Dy=1+y^2','y(0)=1')
 y =
  tan(t+1/4*pi+pi*Z)
```

Пример. 6.52.

Найти решение ОДУ $x'' + 5x = 1$

```
>> x = dsolve('(D2x)+5*x=1')
 x =
  1/5+C1*cos(5^(1/2)*t)+C2*sin(5^(1/2)*t)
```

Пример. 6.53.

Найти решение ОДУ $x'' + 5x = 1$ с начальными условиями $x(0) = 0$, $x'(0) = 0$.

```
>> x = dsolve('(D2x)+5*x=1','x(0)=0','Dx(0)=0')
 x =
  1/5-1/5*cos(5^(1/2)*t)
```

Пример. 6.54.

Найти решение ОДУ $y'' = \cos(2x) - y'$ с начальными условиями $y(0) = 1$, $y'(0) = 0$.

```
>> y = dsolve('D2y=cos(2*x)-y','y(0)=1','Dy(0)=0', 'x')
y =
(1/6*cos(3*x)-1/2*cos(x))*cos(x)+(1/2*sin(x)+1/6*sin(3*x))*sin(x
)+4/3*cos(x)
>> simplify(y)
ans =
 -2/3*cos(x)^2+4/3*cos(x)+1/3
```

Пример. 6.55.

Найти решение системы ОДУ $\begin{cases} f' = 2f + 4g \\ g' = -4f + 3g \end{cases}$

```
>> S = dsolve('Df = 3*f+4*g', 'Dg = -4*f+3*g')
S =
    f: [1x1 sym]
    g: [1x1 sym]
>> S.f
 ans =
  exp(3*t)*(cos(4*t)*C1+sin(4*t)*C2)
>> S.g
 ans =
  -exp(3*t)*(sin(4*t)*C1-cos(4*t)*C2)
```

6.9. Средства визуализации результатов символьных вычислений

Пример. 6.56.

Построить поверхность, задаваемую функцией $f(x) = xy\sin(x^2+y^2)$, используя графические средства визуализации результатов символьных вычислений MATLAB

```
>> syms x y % создаем символьные переменные
>> f=x*y*sin(x^2+y^2);
>> ezcontour(f,[-2,2],55)    строим карту линий уровня (рис. 6.3)

>> ezcontourf(f,[-2,2],55)    строим закрашенную карту
                              линий уровня (рис. 6.4)

>> ezmesh(f,[-2,2])    строим поверхность (рис. 6.5)

>> ezmeshc(f,[-2,2])    строим поверхность и карту линий уровня
(рис. 6.6)

>> ezsurf(f,[-2,2])    строим поверхность (рис. 6.7)

>> ezsurfc(f,[-2,2])    строим поверхность (рис. 6.8)
```

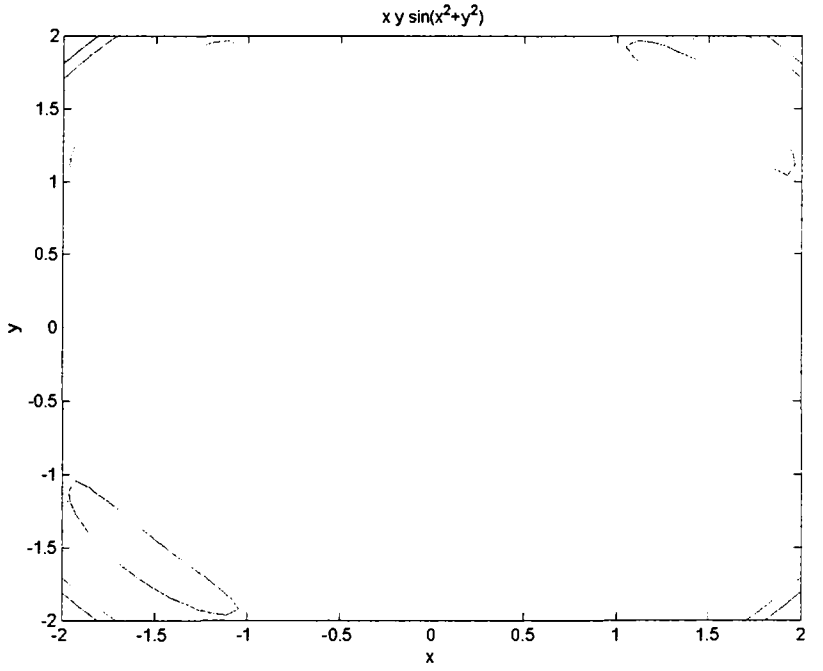


Рис. 6.3. Карта линий, построенная функцией **ezcontour**

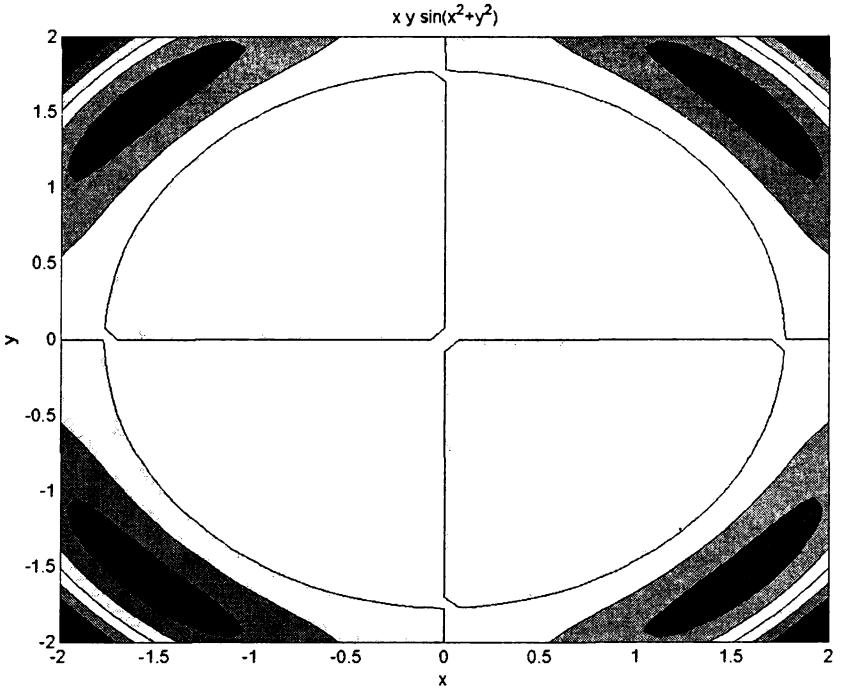


Рис. 6.4. Карта линий, построенная функцией **ezcontourf**

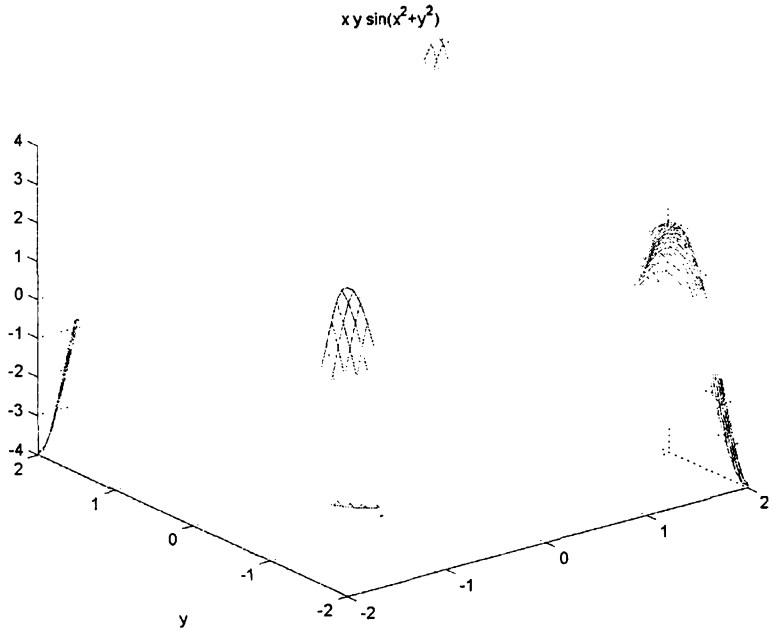


Рис. 6.5. Поверхность функции, построенная функцией **ezmesh**

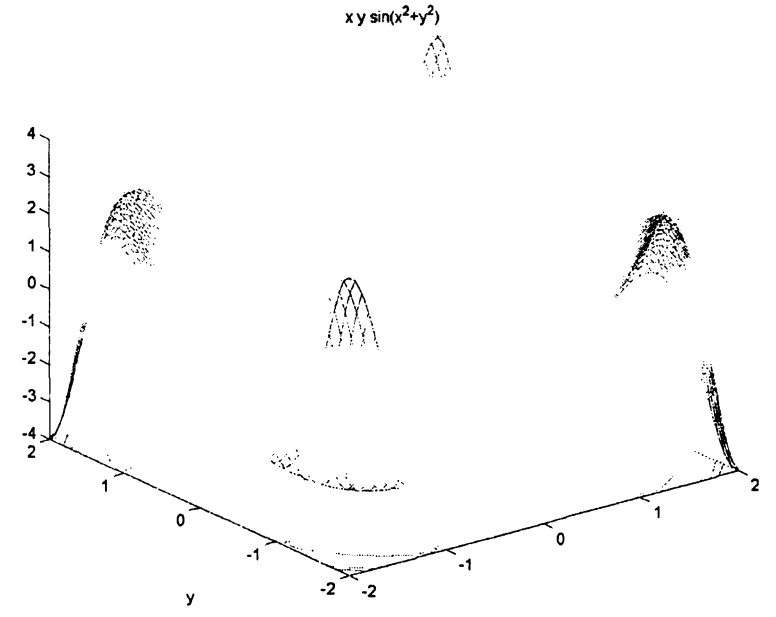


Рис. 6.6. Поверхность функции $f(x) = xy\sin(x^2 + y^2)$ и карта линий уровня, построенная функцией **ezmeshc**

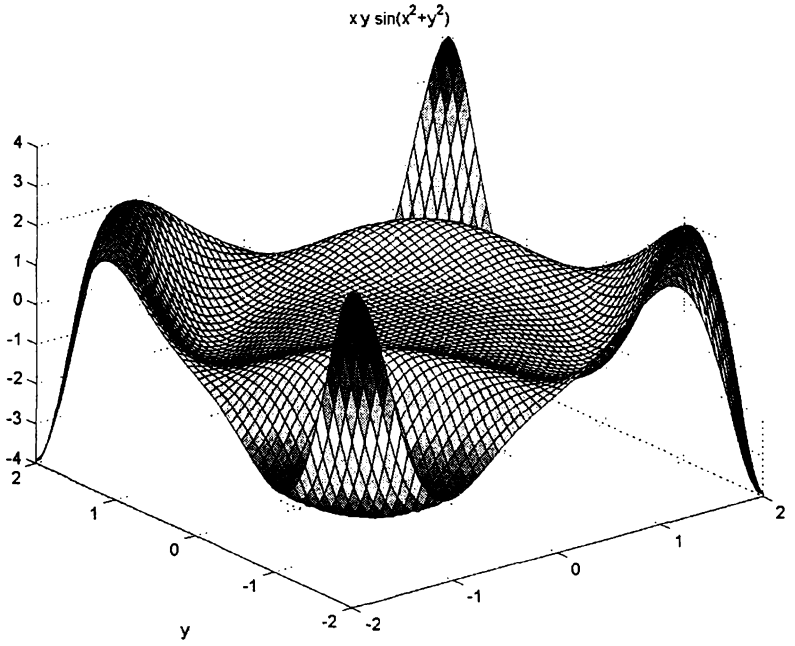


Рис. 6.7. Поверхность функции $f(x) = xy\sin(x^2 + y^2)$, построенная функцией **ezsurf**

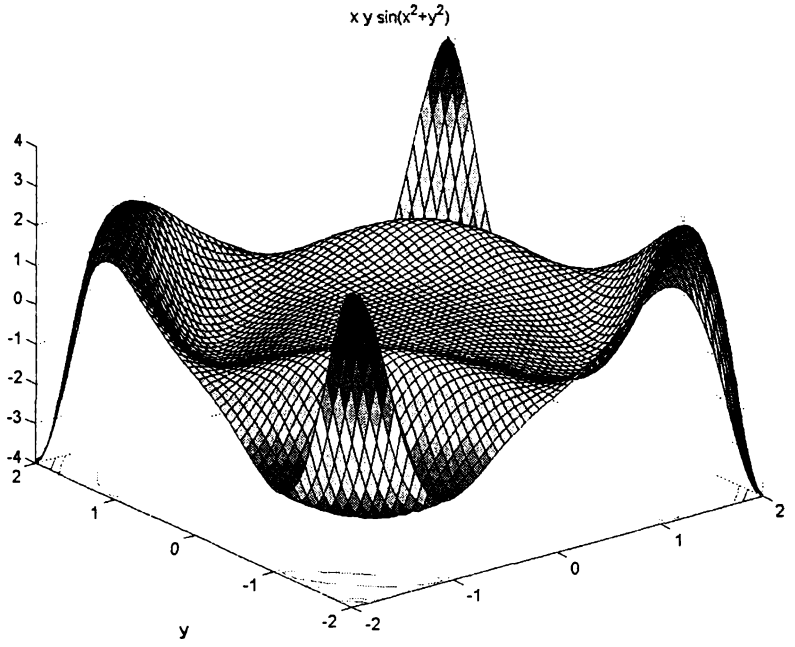


Рис. 6.8. Поверхность функции $f(x) = xy\sin(x^2 + y^2)$, и карта линий уровня, построенная функцией **ezsurfc**

Пример. 6.57.

Построить график функции $x^2 - y^4 = 0$, заданной неявно

```
>> syms x y % задаем символьные переменные
>> ezplot(x^2-y^4) % визуализируем поверхность (рис. 6.9)
```

Пример. 6.58.

Построить в полярный график функции $f(t) = 1 + \cos(t)$

```
>> syms t % задаем символьные переменные
>> ezpolar(1+cos(t)) % строим полярный график (рис. 6.10)
```

Вопросы для самопроверки

1. Какие вычисления называются символьными?
2. Как создать символьную переменную в MATLAB?
3. Как накладываются ограничения на область возможных значений символьной переменной в MATLAB?
4. Как можно в MATLAB обратиться к стандартным функциям пакета Maple, осуществляющим символьные вычисления?
5. Как создать в MATLAB символьную матрицу?
6. Как выполнить в MATLAB дифференцирование в символьном виде?
7. Как вычислить в MATLAB в символьном виде матрицу Якоби?
8. Как вычислить в MATLAB вычислить в символьном виде значение предела функции?
9. Как вычислить в MATLAB значение интеграла в символьном виде?
10. Как вычислить в MATLAB значение суммы и произведения ряда в символьном виде?
11. Как получить в MATLAB в символьном виде разложение функции в ряд?
12. Как выполняются в MATLAB упрощения и постановки в символьных выражениях?
13. Как можно в MATLAB найти решение алгебраического уравнения в символьном виде?
14. Как в MATLAB осуществляется управление точностью вычислений?
15. Как в MATLAB выполняются операции линейной алгебры над символьными матрицами?
16. Как можно в MATLAB найти решение дифференциального уравнения в символьном виде?
17. Перечислите встроенные в MATLAB средства визуализации символьных вычислений.

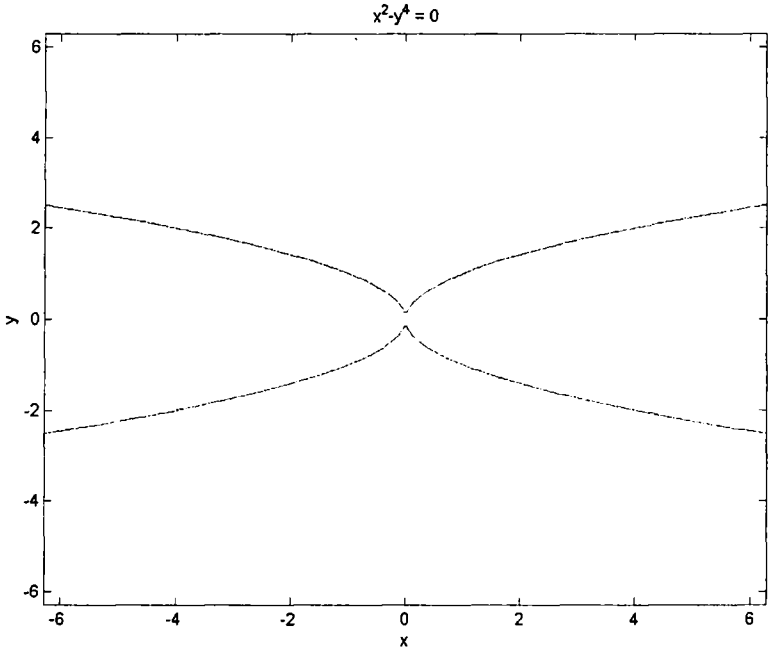


Рис. 6.9. График функции $f(x,y) = (x^2 + y^4)$

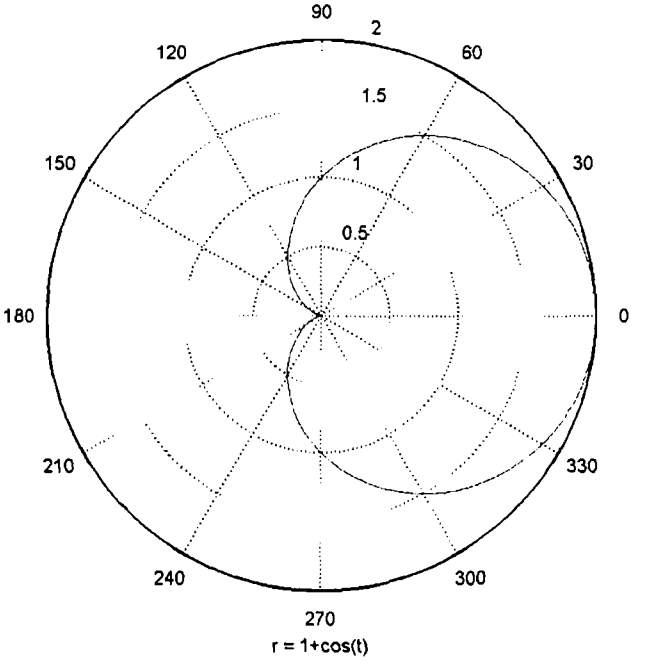


Рис. 6.10. Полярный график функции $f(t) = 1 + \cos(t)$

Программирование на М-языке

7.1. Операторы цикла в М-языке. Анимация

В настоящем разделе мы рассмотрим специальные управляющие конструкции, которые позволяют осуществлять циклический повтор кода на М-языке. В традиционных языках программирования данные конструкции управления принято называть операторами цикла. Операторы цикла в М-языке MATLAB задаются с помощью специальных зарезервированных ключевых слов. Их можно использовать как в интерактивном режиме, вводя операторы непосредственно с клавиатуры, так и в текстах программ на М-языке. Примеры программ обсуждаются ниже во второй части данной главы, посвященной различным аспектам программирования в MATLAB.

В зависимости от способа определения условия останова (окончания циклических повторений) различают два вида операторов цикла в М-языке MATLAB. Первый из них использует ключевые слова **while** и **end**. Он имеет вид

```
while условие
   .
end
```

Здесь повтор выполнение набора команд, обозначенного многоточием, продолжается до тех пор пока условие «истинно» (не равно нулю). В случае массивов истинность наступает, когда все элементы массива истинны.

Рассмотрим в качестве примера следующий фрагмент, который вычисляет частичную сумму ряда $\sum_{i=1}^{\infty} \frac{1}{k^2}$:

```
>> S=0;k=1;u=1;
>> while u > 1e-8
     S=S + u;
     k=k + 1;
```

```
    u=1/k^2;
end;
```

Здесь условием останова служит требование к слагаемым быть больше некоторого заранее определенного числа (в данном случае 10^{-8}): как только очередное слагаемое станет меньше этого числа, условие, стоящее после ключевого слова **while**, становится ложным и суммирование прекращается.

Приведенный выше фрагмент кода, в котором оператор цикла занимает целых четыре физических строки, вводится с клавиатуры последовательно строка за строкой, при этом клавиша **Enter** нажимается каждый раз после ввода очередной строки. После ввода заголовка оператора цикла и нажатия клавиши **Enter** текстовый курсор перемещается на следующую физическую строку, однако знак строки при вводе (>>) не появляется. Это означает, что MATLAB ожидает дальнейшего ввода с клавиатуры последовательности команд, составляющих тело цикла. Эти инструкции могут занимать несколько физических строк командного окна. Текстовый курсор перемещается на следующую строку нажатием клавиши **Enter**. После ввода ключевого слова **end** MATLAB приступает к циклически повторяющемуся выполнению инструкций из тела цикла. По завершении этого процесса (его длительность зависит от сложности и суммарного объема вычислений) на следующей за циклом строке появляется знак >>.

Другой вариант организации оператора цикла в М-языке MATLAB — использовать ключевые слова **for** и **end**. Он имеет вид:

```
for varName = выражение
   .
end
```

где **varName** — выбираемое программистом имя переменной, называемой счетчиком цикла. В отличие от первого варианта оператора цикла здесь число выполнений операторов цикла конечно. Оно задается числом возможных значений переменной **varName**. Набор возможных значений для переменной цикла поставляет выражение, которое стоит после ключевого слова **for**. Чаще всего это выражение являющееся переменной типа диапазон, рассмотренного в первой главе. В следующем фрагменте кода осуществляется сложение первых 57 членов ряда $\sum_{k=1}^{57} k^2$:

```
>> S=0;
>> for k=1:1:57
     S =S + 1/k^2;
   end;
```

Вместо операции задания диапазона можно явно указать весь набор возможных значений в виде вектор-строки, например:

```
for m = [2, 5, 7, 8, 11, 23]
```

Здесь цикл будет выполняться шесть раз, при этом переменная **m** будет последовательно принимать значения 2, 5, 7, 8, 11 и 23.

Достаточно необычным может показаться использование матриц в управляющем выражении:

```
>> A = [1 2; 3 4];
>> for k = A
```

Такой цикл будет повторяться ровно столько раз, сколько столбцов в матрице **A**, то есть два раза для данного конкретного случая. На каждом шаге переменная цикла принимает значение очередного столбца матрицы, т.е. является вектор-столбцом.

Например, следующий фрагмент

```
>> S=0;A=[1 2; 3 4];
>> for k=A
     S=S+sqrt( k(1)^2 + k(2)^2 );
   end;
>> 7.6344
```

вычисляет сумму длин векторов, являющихся столбцами матрицы **A**.

Для досрочного прерывания выполнения обоих видов операторов цикла используется команда **break**, размещаемая внутри тела цикла. Это позволяет размещать фактическое условие останова цикла внутри тела цикла, а не в его заголовке. В этом случае в заголовке цикла можно писать выражения вида **while 1** (всегда истинное выражение).

Отметим, что в MATLAB, везде где это возможно, вместо операторов цикла следует применять соответствующие команды М-языка, предназначенные для обработки массивов, так как при этом существенно сокращается время вычислений.

Например, вместо следующего фрагмента кода с оператором цикла

```
>> k=0;
>> for x = 0:0.1 :100
     k=k+1;
     y(k)=cos(x);
   end;
```

лучше использовать операции с массивами:

```
>> x=0:0.1:100;
>> y = cos(x);
```

Замену операторов цикла эквивалентными им по результату функциями М-языка принято называть векторизацией кода.

Применим полученные знания об операторах цикла М-языка для получения анимационных эффектов при визуализации результатов вычислений.

Для этого осуществляется предварительная подготовка отдельных кадров, которые сохраняются для дальнейшего показа в некотором числовом массиве. Затем функцией **movie** осуществляется последовательный покадровый показ созданного анимационного клипа.

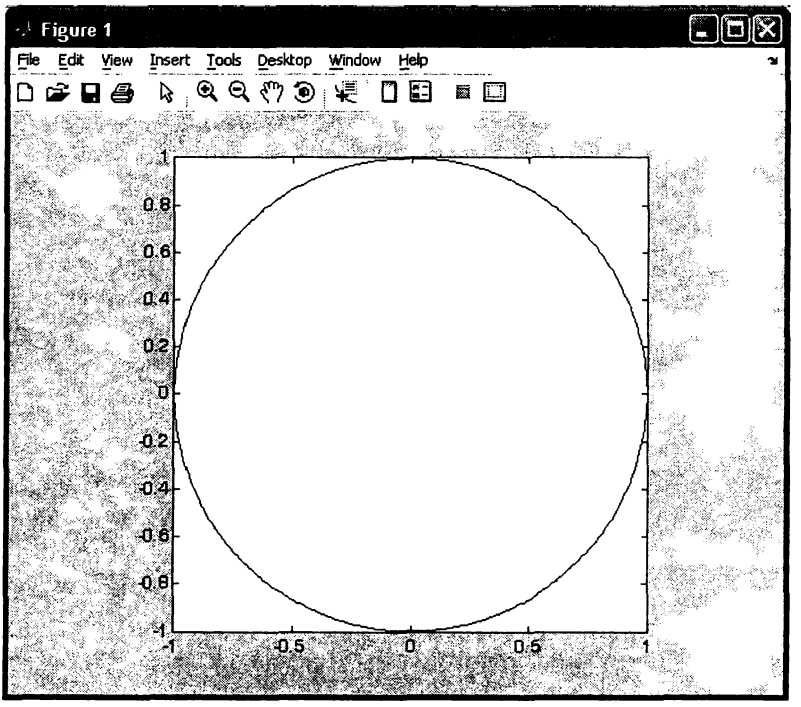


Рис. 7.1. Траектория движения бусинки

Начнем изложение с первого способа. Для примера в графическом окне MATLAB продемонстрируем равномерное движение по окружности материального тела в виде бусинки. Сначала выполним следующую последовательность команд

```
>> x=-1:0.01:1;
>> y1=sqrt(1-x.*x);
>> y2=-y1;
>> plot(x,y1,'k',x,y2,'k');
>> axis square;
>> hold on
```

Результаты ее выполнения представлены на рис. 7.1. Команда **hold on** нужна для последующего графического вывода изображений в одно и тоже графическое окно.

Теперь размещаем бусинку в ее начальном положении (рис. 7.2):

```
>> x =1; y=0;
>> h = plot(x, y, '.r');
>> set(h,'EraseMode','xor','MarkerSize',18);
```

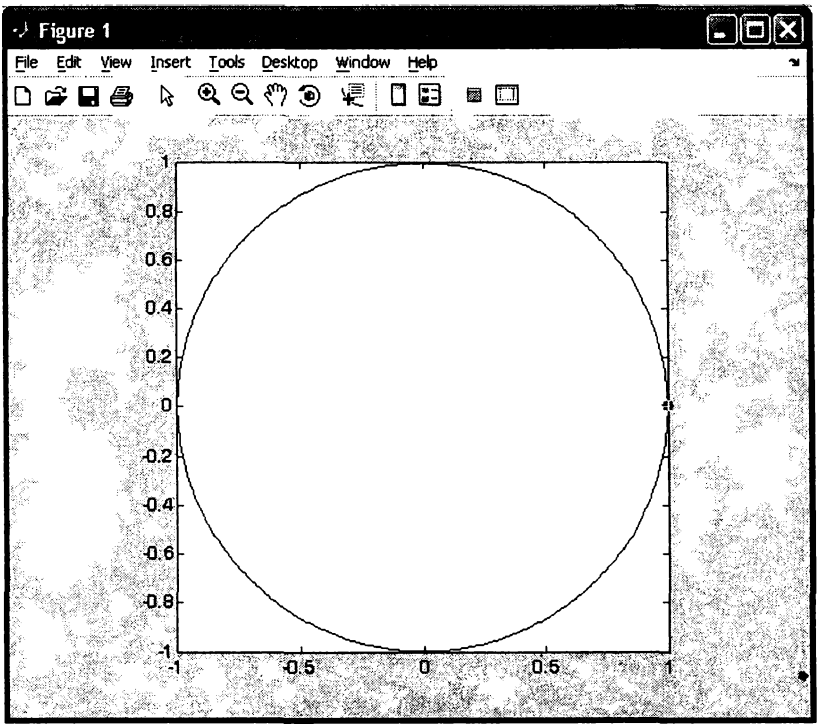


Рис. 7.2. Положение бусинки в момент времени t = 0

Далее будем динамически изменять координаты точки, относящейся к графическому объекту типа **line** с описателем **h**. При этом MATLAB будет стирать предыдущие изображения этого графического объекта и рисовать его новые изображения. Наконец осуществляем циклическое продвижение бусинки и ее перерисовку:

```
>> t=0;
   dt=0.1;
   a=2*pi/200;
   for k=1:1000
     t=t+dt;
     x=cos(a*t);
     y=sin(a*t);
     set(h,'Xdata',x,'YData',y); % перемещаем точку в новое
                                 % положение
Buff(k)=getframe;    сохраняем созданный кадр видеоклипа
end;
```

Для воспроизведения созданного видеоклипа используется команда **movie**:

```
>> movie(Buff,1)
```

При необходимости прокрутить снятый ролик несколько раз подряд, следует указать в качестве второго аргумента функции **movie** целое число больше единицы.

Если отпадает необходимость в хранении кинофильма, следует сразу же освободить память в рабочем пространстве MATLAB, занятую под переменную **Buf**:

```
clear Buf
```

потому что видеобуферы, как правило, имеют гигантские размеры и зря расходуют ресурсы компьютера.

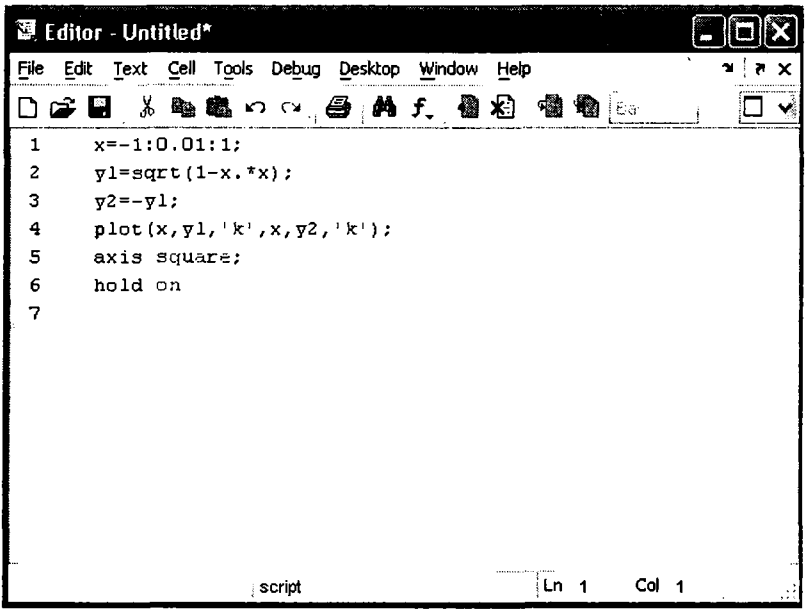


Рис. 7.3. Рабочее окно текстового редактора MATLAB

7.2. Сценарии и M-файлы

В предыдущем параграфе, посвященном анимации, приходилось вводить с клавиатуры в командное окно MATLAB довольно большие фрагменты кода на M-языке. Если потребуется осуществить анимацию (или другую столь же объемную задачу) в другом сеансе работы с MATLAB, то потребуется повторить весь ввод с клавиатуры. Повторный ввод большого набора инструкций, как показывает опыт работы, не только утомителен, но и чреват возникновением ошибок. Ясно, что в рамках MATLAB необходим инструмент для запоминания фрагментов командных кодов большого размера, позволяющий избежать его повторного ввода. Данный инструмент называется сценарием работы.

Под сценарием понимают текстовый файл, содержащий инструкции на М-языке, подлежащие исполнению в автоматическом (пакетном) режиме. Данный файл может иметь произвольное имя, но фиксированное расширение — оно состоит из одной буквы m. Любой текстовый файл, содержащий произвольный код на М-языке и имеющий оговоренное выше расширение, принято называть М-файлом. Создать такой файл можно с помощью любого текстового редактора. Однако наиболее удобным оказывается собственного текстового редактора MATLAB, окно которого представлено на рис. 7.3. Вызов редактора осуществляется из командного окна MATLAB командой меню **File | New | m-file** (или самой левой кнопкой на полосе инструментов, на которой изображен чистый белый лист бумаги). Редактор работает в своем собственном окне. Отметим, что переменные, определяемые в командном окне, и переменные, определяемые в сценариях, составляют единое рабочее пространство MATLAB. Ниже мы еще раз вернемся к этому важному вопросу.

После создания текста сценария его следует сохранить в М-файле на диске. М-файлы не следует путать с MAT-файлами, в которых хранятся переменные из рабочего пространства MATLAB. Каталог на диске для хранения М-файлов может быть любым. При этом путь к этому каталогу обязательно должен быть известен MATLAB. Напомним, что MATLAB хранит сведения обо всех известных ему каталогах. Для создания нового каталога нужно сначала выполнить команду меню (командного окна) **File | Set Path**, с помощью которой вызывается диалоговое окно с именем **Set Path** (см. рис. 7.4).

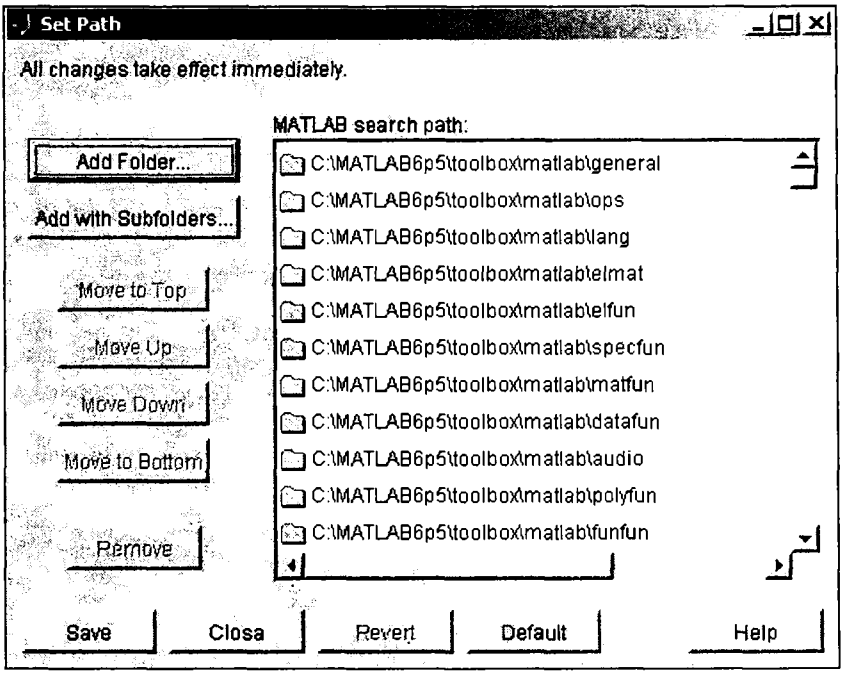


Рис. 7.4. Командное окно для управления деревом каталогов

В этом окне отображается список всех зарегистрированных в MATLAB путей доступа. Для добавления нового каталога в список путей доступа служит кнопка **Add Folder...** этого окна, которая, в свою очередь, порождает еще целый ряд диалоговых окон, работа с которыми достаточно очевидна.

После того как в список путей доступа добавлен новый каталог, в нем можно сохранить файл с созданным в редакторе сценарием. Для того чтобы выполнить код, расположенный в М-файле, нужно ввести в командном окне имя М-файла, содержащего сценарий работы (при этом явно указывать расширение файла нельзя, так как это приведет к возникновению ошибочной ситуации), и нажать клавишу **Enter**.

Еще раз напомним, что сценарий обрабатывает как свои собственные переменные, так и переменные, определенные до вызова сценария в командном окне MATLAB и хранящиеся в ее рабочем пространстве. В свою очередь, все переменные, созданные во время работы сценария, остаются в рабочем пространстве MATLAB и после окончания его выполнения.

Данное свойство является одновременно и сильной и слабой стороной сценариев. Сильной стороной является полная интеграция с рабочим пространством, что делает сценарии средством автоматизации интерактивного сеанса работы пользователя с MATLAB. Эта же интеграция является его слабой стороной: если предполагается оформить в виде сценария отдельный фрагмент решения некоторой задачи, его следует всегда вызывать в одной и той же программной обстановке. Это условие, как очевидно, выполняется далеко не всегда. Кроме того, сценарию в момент его вызова нельзя передать в виде параметров дополнительную информацию, не содержащуюся в рабочем пространстве.

Перечисленные слабые стороны сценариев преодолеваются с помощью М-функций, которые будут изучаться в следующем разделе данной главы. Здесь же отметим, что М-функции, в свою очередь, плохо приспособлены для нужд автоматизации интерактивного сеанса работы, так как их пространство переменных полностью изолировано от рабочего пространства MATLAB.

Одним из типовых случаев применения сценариев является запись в М-файл текста, скопированного из командного окна. Если среди содержимого такого файла присутствует некоторое количество ненужных или ошибочно записанных инструкций, их следует удалить из текста файла. После этого можно вызывать полученный таким образом сценарий на выполнение. В результате в рабочем пространстве MATLAB будут восстановлены все переменные, созданные в предыдущем сеансе работы, а кроме них также будут воссозданы все графические объекты (если они были ранее созданы), потому что будут выполнены создающие их команды.

Для примера, ниже приведен листинг m-файла, в который мы поместили все команды для решения задачи об анимации движения бусинки по окружности, вводившиеся с клавиатуры в интерактивном режиме.

```
% листинг файла An.m
% Animation script
x=-1:0.01:1;
y1=sqrt(1-x.*x);
y2=-y1;
plot(x,y1,'k',x,y2,'k');
```

```
axis square)
hold on
% ----------------------
x=1;
y=0;
h=plot(x,y,'.r');
set(h,'EraseMode','xor','MarkerSize',18);
% ----------------------
t=0;
dt=0.01;
a=0.5;
while 1
 t=t+dt;
 x=cos(a*t);
 y=sin(a*t);
 set(h,'Xdata',x,'Ydata',y);
 drawnow
end;
```

Отметим, что здесь вместо создания переменной, содержащей кадры анимационного клипа, мы выводим мгновенные изображения «бусинки», используя команду **drawnow**.

7.3. Синтаксис определения и вызова М-функций

В предыдущих разделах были изучены сценарии, которые не разделяют свои переменные с рабочим пространством MATLAB. Такая тесная интеграция не позволяет реализовать полностью изолированный и независимый от рабочего пространства фрагмент кода на М-языке. Для того чтобы реализовать независимый и изолированный фрагмент кода, решающий некоторую фиксированную задачу и принимающий в виде входных параметров начальную информацию, нужно оформить этот фрагмент в виде функции MATLAB. Функции, как и сценарии, состоят из набора инструкций М-языка и их также записывают в текстовые файлы с расширением m. В итоге для краткости их называют М-функциями. Далее мы подробно рассмотрим все особенности устройства М-функций.

Текст М-функций обязательно должен начинаться с заголовка, после которого следует тело функции. Заголовок, определяющий «интерфейс» функции (способ взаимодействия с ней), имеет следующий вид:

```
function [Ret1, Ret2,...] = FName(par1, par2,...)
```

В заголовке объявляется функция (с помощью ключевого слова **function**) с именем **FName**, которая принимает входные параметры **par1**, **par2**, ..., и вырабатывает выходные (возвращаемые) значения **Ret1**, **Ret2**.... Многоточия не являются элементами синтаксической конструкции, они применяются только для того, чтобы наглядно подчеркнуть тот факт, что функции могут иметь разное число входных параметров и выходных значений, например:

```
function FName1
function FName2(par1, par2, par3)
function Ret1 = FName3(par1, par2)
function [Ret1, Ret2, Ret3] = FName4(par1)
```

Здесь в первой строке представлен заголовок функции **FName1**, не имеющей ни входных параметров, ни выходных значений. Функция **FName2** принимает три входных параметра и также не имеет возвращаемых значений. Далее функция **FName3** принимает два и возвращает одно значение. И, наконец, функция **FName4** имеет один входной параметр и три выходных значения. Часто для краткости входные параметры называют аргументами функции, а возвращаемые (выходные) значения функции называют просто ее значениями.

Указанное в заголовке имя функции должно совпадать с именем файла (без учета расширения **.m**), в который записывается текст функции. Рассогласования имени функции и имени файла не допускается.

Тело функции состоит из инструкций М-языка, с помощью которых вычисляются возвращаемые значения. Тело функции следует за заголовком функции. Заголовок функции плюс тело функции в совокупности составляют определение функции. Размещать М-файл с определением функции нужно в одном из каталогов диска, входящих в список доступа MATLAB. Этот вопрос уже обсуждался выше при изучении сценариев.

Если в М-файл поместить определения сразу нескольких функций, то вызывать из командного окна MATLAB (или из функций другого М-файла) можно будет только ту из них, имя которой совпадает с именем М-файла. Таким образом, только одна функция из М-файла оказывается видимой вне этого файла. Остальные функции вызываются изнутри данного М-файла и выполняют вспомогательные вычисления.

Приведем пример ситуации, когда в файле **MyFunc.m** содержатся определения двух функций (определение функции заканчивается либо исчерпанием инструкций, либо заголовком определения следующей функции):

```
function ret1 = MyFunc(x1, x2)
ret1 = x1 .*x2 + AnotherFunc(x1);
function ret2 = AnotherFunc(y)
ret2 = -y.*y+2*y+3;
```

Здесь определены функции **MyFunc** и **AnotherFunc**. Однако извне можно вызывать только функцию **MyFunc**. Функцию **AnotherFunc** может вызывать только функция **MyFunc**. Функцию MyFunc можно назвать основной функцией в файле **MyFunc.m**, а функцию **AnotherFunc** — вспомогательной функцией или подфункцией (по-английски — subfunction).

Существует еще один способ вызова функции, используемый при необходимости ограничить доступ к некоторой М-функции, расположенной в заданном каталоге. В этом случае в данном каталоге нужно создать дочерний подкаталог с именем **private**, и поместить туда указанную функцию. Все функции из каталога **private** будут доступны только функциям из родительского каталога.

Как входные параметры, так и возвращаемые значения могут быть в общем случае массивами различных типов, размерностей и размеров. Например, следующая функция с именем **MatrProc1**:

```
function [A,B] = MatrProc1(X1,X2,x)
% MatrProcI calculates MatrixProduction
% plus (or by) one scalar
% ---------------------------
A = X1 .*X2*x;
B = X1 .*X2+x;   The last line of the code
```

рассчитана на «прием» двух числовых массивов **X1** и **X2** одинакового размера и одного скаляра x. Эти массивы в теле функции сначала перемножаются поэлементно, после чего результат такого перемножения дополнительно умножается на скаляр **x**. Таким образом порождается первый из выходных массивов — массив **A**. Одинаковые размеры входных массивов **X1** и **X2** гарантируют выполнимость операции их поэлементного умножения. Второй выходной массив (с именем B) отличается от первого тем, что получается сложением со скаляром (а не умножением).

В тексте функции **MatrProc1** помимо инструкций, вычисляющих возвращаемые функцией значения, присутствуют также комментарии, начинающиеся со знака %. Как известно, в любом языке программирования комментарии не являются исполняемыми инструкциями, а служат лишь для целей документирования процесса программирования. Цель их размещения в тесте программы — пояснить смысл той или иной части программного кода. Без комментариев бывает трудно понять отдельные места в сложных программах, особенно когда с момента разработки прошло достаточно много времени.

В MATLAB особую роль играют строки комментариев, располагаемые сразу же за заголовком функции. Эти комментарии предназначены не только для программистов, но и для конечных пользователей, желающих ознакомиться с краткой информацией об используемой функции. Таким образом, в этих строках должна быть расположена краткая справка по хорошо документированной функции. Получить требующуюся справку может любой пользователь функции, выполнив команду

```
help MatrProcI
```

в результате выполнения которой в командное окно поступит следующая информация:

```
MatrProcI calculates MatrixProduction
plus (or by) one scalar
---------------------------
```

Эти же начальные строки комментариев участвуют в поиске функции по содержащимся в ней текстовым фрагментам. Такой поиск осуществляется командой

```
lookfor текст -all
```

Наконец, командой

```
type имя_функции
```

в командном окне отображается текст М-функции вместе со всеми ее комментариями.

Продолжим изучение аспектов синтаксиса М-функций, связаных с использованием разработанных функций. Вызов созданной нами функции осуществляется из командного окна MATLAB или из текста какой-либо другой функции. Разные М-функции с совпадающими именами (совпадают имена их М-файлов) могут располагаться в разных каталогах на диске компьютера, поэтому при вызове М-функций MATLAB должен руководствоваться четко определенным критерием выбора конкретной функции. Приоритет при этом всегда отдается внутренней для М-файла функции (если вызов осуществляется внутри этого файла и если внутренняя функция с этим именем существует). В случае неудачи поиск продолжается в частном подкаталоге **private**. И только после этого начинает просматриваться весь список путей доступа MATLAB вплоть до первой встретившейся при этом функции с совпадающим именем (это имя М-файла).

Синтаксис вызова функции в MATLAB аналогичен вызову функции в большинстве языков программирования: в командной строке вводится имя функции, после которого в круглых скобках через запятую перечисляются фактические входные параметры, со значениями которых и будут произведены вычисления. Фактические параметры могут быть заданы числовыми (и другими) значениями, именами переменных (уже имеющими конкретные значения), а также выражениями. Если фактический параметр задан именем некоторой переменной, то реальные вычисления будут производиться с копией этой переменной (а не с ней самой). Это называется передачей параметров по значению, например,

```
>> W1=[1 2; 2 2];
>> W2=[3 1; 1 1]];
>> [Res1 Res2]=MatrProc1(W1,W2,3);
>> Res1
   9  6
   6  6
>> Res2
   6  5
   5  5
```

Здесь имена фактических входных параметров (**W1** и **W2**) и переменных, в которых записываются результаты вычислений (**Res1** и **Res2**), не совпадают с именами аналогичных переменных в определении функции **MatrProc1**. Для того чтобы подчеркнуть это возможное отличие, имена входных параметров и выходных значений в определении функции называют формальными.

Отметим, что функция **MatrProc1** может проводить операции с массивами произвольных размерностей, например, вызвав функцию **MatrProc1**

```
>> [r1,r2]=MatrProc1( [1 2 3; 4 5 6],[7 7 7;2 2 2],1);
```

с двумя входными массивами размером 2×3, получим две выходные матрицы размером 2×3.

Функции, возвращающие несколько значений, можно также использовать в составе выражений. При этом в качестве значения функции, приме-

няемого для дальнейших вычислений, используется первое из возвращаемых функцией значений, например:

```
>> s=MatrProc1(1,2,1)+6;
>> s
  s =
    8
```

При вызове с параметрами 1,2,1 функция **MatrProc1** возвращает два значения: 2 и 3. Для вычисления всего выражения используется первое из них, поэтому переменная s становится равной 8.

Так как вызов любой функции можно осуществить, написав произвольное выражение в командном окне MATLAB, то всегда можно совершить ошибку, связанную с несовпадением типов фактических и формальных параметров. MATLAB не выполняет проверку совпадения типов фактических и формальных параметров, а просто передает управление функции. В результате могут возникнуть ошибочные ситуации. Для предотвращения (по возможности) возникновения подобных ошибочных ситуаций, нужно в тексте М-функций осуществлять проверку входных параметров. Например, в функции **MatrProc1** легко реализовать проверку размерностей первого и второго входных параметров. Для написания соответстующего кода требуется полный набор конструкций управления.

7.4. Конструкции управления

В любом языке программирования, в том числе и в М-языке MATLAB, имеются специальные конструкции, которые задаются зарезервированными ключевыми словами и служат для управления порядком выполнения операций. Такие конструкции часто называют операторами управления. К ним относятся операторы ветвления и операторы цикла.

К операторам ветвления в М-языке относятся условный оператор и оператор переключения. Условный оператор использует ключевые слова **if** (если), **else** (иначе), **elseif** (иначе если), **end** (конец всей конструкции) и может выступать в одной из следующих трех форм. Во-первых,

```
if условие
  .
end
```

Во-вторых,

```
if условие
  .
else
  .
end
```

и, наконец, в форме

```
if условие1
```

```
elseif условие2
  ..
else
  ..
end
```

в которой число ветвей с ключевым словом **elseif** неограничено.

Область действия условного оператора начинается ключевым словом **if**, а заканчивается ключевым словом **end**. Под условием понимается произвольное выражение (чаще всего это выражение включает в себя операции сравнения и логические операции), истинность или ложность которого понимается как отличие от нуля или равенство нулю.

Если условие истинно, то выполняются команды, стоящие после строки с ключевым словом **if**. Если условие ложно, то эти команды пропускаются и переходят либо к следующему за условным оператору (первая форма), либо проверяют еще одно условие в строке с ключевым словом **elseif** (третья форма условного оператора), либо выполняются без дополнительных проверок команды, стоящие после строки с ключевым словом **else** (вторая из приведенных выше форм).

В логических выражениях в MATLAB также допускается использование массивов. В тех случаях, когда значением таких выражений будет массив, истинность условия наступает, когда истинны (не равны нулю) все элементы массива. Если хоть один элемент такого массива будет равен нулю, то условие считается ложным. Кроме того, условие считается ложным, если используется пустой массив.

В качестве примера приведем фрагмент кода, иллюстрирующего работу условного оператора

```
>> A=[ 1 2; 4 0 ];
>> if A
     b=1;
    else
     b=2;
```

в результате выполнения которого переменная **b** получит значение 2, так как матрица **A** содержит один нулевой элемент, и все условие считается ложным.

Отметим, что запись **if A** по своему действию полностью эквивалентна записи

```
>> if A~=0
```

и записи

```
>> if all(A(   ))
```

Условные операторы часто используются в теле М-функций совместно с инструкцией **return** для осуществления досрочного завершения функции и выхода из нее. В качестве примера, приведем функцию вычисляющую факториал целого положительного числа **n**:

```
function res = MyFact(n)
  res = 1;
  if n == 1
```

```
    return
  else
    for i = 2:n
      res =res .*i;
  end
end
```

Если входной параметр данной функции равен единице, то оператор **return** осуществляет досрочный выход из функции, так как правильный результат (**res=1;**) уже сформирован.

Когда оператор **return** не используется, нормальное завершение функции наступает по исчерпании всех инструкций, размещенных в теле функции.

Для досрочного выхода из цикла также широко применяются условные операторы, размещаемые внутри операторов цикла совместно с инструкцией **break**. Иногда такое условие выхода из цикла может быть единственным, как в следующем примере:

```
prod=1; i =1;
while 1
 prod=prod .*i
 i=i+1;
 if i > 8
  break
 end
end
```

где условие в заголовке цикла всегда истинно, поэтому цикл без команды **break**, работающей совместно с условным оператором был бы бесконечным. Таким образом, цикл оказывается конечным, в результате будет возвращено значение произведения целых чисел от единицы до восьми (факториал восьми).

Другим оператором ветвления является оператор переключения. Он использует ключевые слова **switch** (переключить), **case** (случай), **otherwise** (иначе) и имеет следующую конструкцию:

```
switch выражение
  case значение1

  case {значение2, значение3}

  otherwise
  .
end
```

Здесь сначала вычисляется выражение, принимающее скалярное числовое значение, затем полученный результат сравнивается с набором значений **значение1**, **значение2**, **значение3** и так далее. В случае совпадения с одним из значений, выполняется нижестоящая ветка. При несовпадении ни с одним из перечисленных значений выполняется ветка, стоящая после ключевого слова **otherwise**. Число строк с ключевым словом **case** не ограничено, в то время как строка с ключевым словом **otherwise** должна быть одна.

Отметим еще одну важную деталь: после выполнения той или иной ветви оператора переключения никакие другие ветви этого оператора (например, нижележащие) не выполняются, поэтому в М-языке MATLAB не надо предпринимать никаких специальных усилий для обхода нижележащих ветвей оператора переключения (в отличие, например, от языков C/C++).

Наконец, в М-языке имеется конструкция управления для перехвата исключительных (ошибочных) ситуаций. Она формируется с помощью ключевых слов **try**, **catch** и **end**:

```
try
  блок_кода1
catch
  блок_кода2
end
```

Здесь между ключевыми словами **try** и **catch** располагается блок кода на М-языке, который требуется выполнить в этом месте программы, но который потенциально опасен в некотором отношении (то есть он может при некоторых обстоятельствах вызывать ошибочные ситуации, например, нехватку памяти компьютера, деление на нуль и тому подобное). В случае возникновения ошибочных ситуаций при выполнении участка **блок_кода1** управление сразу же передается на **блок_кода2**. Этот второй блок кода на М-языке называется обработчиком ошибочной ситуации. В отсутствии ошибочных ситуаций управление, минуя **блок_кода2**, передается за ключевое слово **end**.

Часто в **блок_кода2** помещают функции извещения пользователя о возникшей ошибочной ситуации и функции корректного завершения всей программы. В то же время здесь может стоять любой код, уместный в конкретном случае.

Однако попасть в catch-отсек на практике весьма непросто, так как MATLAB в большинстве своих функций и операций обрабатывает ошибочные ситуации весьма надежно, так что нет никакой необходимости в дополнительных обработчиках. Приведем простой пример, когда использование конструкции перехвата исключений реально работает:

```
>> try,h=8;delete(h);catch,str=lasterr;disp(str),end
Error using ==> delete
Invalid handle object.
>>
```

Здесь мы попытались с помощью функции **delete** удалить объект дескрипторной графики, ссылаясь на произвольно заданный, то есть фактически недействительный описатель (дескриптор) **h**. В результате возникла ошибочная ситуация, которая нами была перехвачена, текстовая строка **str** получила значение, которое и показано на рисунке. Вывод на экран осуществляется функцией **disp**, которая подробно обсуждается в следующем параграфе.

В принципе ситуации, когда в М-языке требуется применять перехватчики исключительных ситуаций, достаточно редки. Однако в любом случае для написания надежного кода желательно не забывать про такое полезное средство. В худшем случае оно просто не сработает, не принеся никакого дополнительною вреда.

7.5. Взаимодействие М-функций с пользователем

Во время выполнения М-функций не требуется интерактивного взаимодействия с пользователем только в самых простейших случаях. При этом пользователь остается в неведении как о ходе вычислений, так и о полученных промежуточных результатах.

Очень часто бывает полезно контролировать ход выполнения М-функций Для интерактивного взаимодействия с пользователем в М-языке предусмотрен ряд специальных функций В частности, функция **disp** применяется для вывода промежуточных результатов в командное окно MATLAB На основе анализа пользователем промежуточных результатов может приниматься решение о продолжении вычислений или об их прекращении.

Функцию **disp** вызывают с единственным аргументом, который может быть числовым или символьным массивом (вектором, матрицей). Ниже приведен пример циклических повторяющихся вычислений конечных отрезков расходящегося гармонического ряда со все большим числом входящих в них слагаемых:

```
>> S=0;
>> i=1;
>> while 1
    S=S+1/i;
    str=sprintf('i=%d S=%f',i,S);
    disp(str);
    i=i+1;
    if rem(i,100)==0
      ans=input('Stop? Answer=','s');
     if ans=='y'
     return
   end
  end
end
```

Здесь функцией **disp** выводятся в командное окно MATLAB все значения промежуточных (частичных) сумм ряда. Кроме того, через каждые сто слагаемых функцией **input** выводится сообщение, предписывающее пользователю ответить на вопрос, не пора ли останавливать вычисления (вообще говоря, бесконечные). Если пользователь вводит при этом с клавиатуры латинский символ 'y' (первая буква от английского слова **yes**), то вычисления прекращаются, так как выполняется оператор досрочного выхода из функции **return**.

Непосредственный ввод информации с клавиатуры и присвоение введенного значения переменной **ans** также осуществляются функцией **input**. В качестве второго аргумента (параметра) функции **input** используется символ 's', означающий, что функция **input** должна принять символьные (а не числовые) значения. Поясним синтаксис ранее не встречавшейся функции **sprintf**, которая возвращает во входную строку значения своих аргументов (кроме первого) в формате, задаваемом первым аргументом. Первый аргумент функции **sprint** называется управляющим элементом.

Этот элемент, помимо обычных символов, подлежащих выводу, содержит спецификаторы формата и управляющие символы, которые определяют формат вывода значений переменных (например, \ — обратная косая черта, предписывает осуществить переход на новую строку), указанных в списке аргументов функции **sprintf** по порядку их следования за управляющим элементом. Функция **sprintf** просматривает свою управляющую строку слева направо и выводит в выходную строку все встречающиеся в ней обычные символы за исключением спецификаторов формата – набора символов, начинающихся с % . Спецификаторы % не являются данными, подлежащими выводу. Они задают функции **sprintf** формат, в котором следует по порядку отображать значение очередной переменной, указанной в списке аргументов вслед за управляющей строкой. Количество и порядок спецификаторов в управляющей строке должны точно соответствовать списку переменных, подлежащих выводу, которые указаны в качестве параметров функции **sprintf** вслед за управляющей строкой.

Например, спецификатор **%s** означает, что очередная переменная, подлежащая выводу, является строкой и следует выводить числовые коды ее символов в соответствии со стандартной кодировкой. Другим распространенным спецификатором является набор символов **%f**, означающий, что выводиться его вещественное число, которое предварительно нужно преобразовать в текстовую строку, содержащую символьный образ. Дополнительные числовые параметры и разделяющая их точка, стоящие в спецификаторе между символами % и **f**, означают полное количество позиций в символьном представлении числа и число позиций, отводимых для записи дробной части. Например, **%4.2f** означает, что вещественное число записывается набором символов в четырех позициях, из которых два отводится для записи дробной части числа.

С другими спецификаторами в любой момент можно ознакомиться, выполнив в командном окне MATLAB команду

```
help sprintf
```

или с помощью вызова диалогового окна **Help**.

Для вывода сообщений (удобное средство для отладки функций) вместо функции **disp** лучше использовать функцию **warning**, так как ее вывод легко подавить из командного окна, выполнив команду

```
warning off
```

Возобновление вывода функцией **warning** предупредительных сообщений осуществляется командой

```
warning on
```

Если в теле М-функции присутствует код, предназначенный для обнаружения тех или иных ошибочных ситуаций, то завершать выполнение М-функций в таких случаях целесообразно вызовом функции **error**, которая не только выполняет такое завершение абсолютно корректно, но и выводит поясняющее ошибку сообщение в командное окно MATLAB. Например, ниже показана функция, вычисляющая величину, обратную входному числовому скалярному параметру:

```
function y = Inverse( x )
  if x== 0
   error( 'Division by zero' )
 else
   y=1 ./ x;
end
```

Данная функция проверяет не равен ли нулю входной параметр, и если равенство имеет место, то выполняется функция **error**:

```
>> Inverse(0)
??? Error using ==> inverse
Division by zero
```

Для временной приостановки выполнения М-функции следует в код, предназначенный для функции вставить вызов функцию **pause** (без входных параметров). Дальнейшее выполнение М-функции возобновляется после нажатия любой клавиши на клавиатуре. Завершая краткое изложение функций пакета MATLAB, ориентированных на интерактивный ввод-вывод, упомянем еще функцию **menu**, реализующую очень наглядный графический ввод одного из альтернативных значений. Например, строка вызова функции **menu**

```
n = menu('Enter your choice','One','Two');
```

помещенная в тело некоторой М-функции, вызывает появление на дисплее компьютера следующего окна с надписью '**Enter your choice**' и двумя кнопками (рис. 7.5).

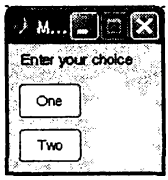


Рис. 7.5. Пример простейшего меню

При нажатии левой клавиши мыши на кнопку **One**, переменная n получает значение 1, на кнопку **Two**, значение 2. Таким образом, возвращаемое функцией menu числовое значение есть номер выбранного пользователем пункта из списка всех доступных вариантов (меню выбора).

Специальным случаем интерактивного взаимодействия с пользователем является ввод имени функции или символьного представления выражения, подлежащих дальнейшему вычислению. В MATLAB для выполнения (вызова) функций, имя которых заранее неизвестно и содержится в качестве значения в текстовой строке, а также для вычисления выражений по их символьному представлению, предназначены две специальные функции **eval** и **feval**.

Предположим, что пользователь ввел с клавиатуры в строковую переменную **str1** следующее содержание

```
>> str1='A=[ 1 2 3; 4 5 6; 7 8 9];'
```

Для того, чтобы создать числовую матрицу с именем **A** и с девятью числовыми элементами, указанными в представленной выше строке текста необходимо использовать функцию **eval**:

```
>> eval( str1 );
```

После выполнения данной команды в рабочем пространстве появится числовая переменная с именем **A**, являющаяся матрицей размерности 3×3:

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

Если ввести строку

```
>> 'x1 = sin( 1.25 );'
```

то функция **eval**, выполнив синтаксический разбор этой строки, вычислит значение функции **sin(1.25)** и поместит полученное значение 0.9490 в переменную **x1**.

При возникновении в процессе работы функции **eval** ошибочной ситуации в командном окне MATLAB появляется сообщение о характере ошибки. Например, выполнив следующую команду

```
>> s = 'sin( 1.25'
```

и вызовав в командном окне функцию **eval(s)**, получим следующее сообщение:

```
>> Improper function reference.  A      or ")" is expected.
```

означающее, что не хватает либо запятой, либо закрывающей круглой скобки (последнее является действительной ошибкой).

При возникновении ошибочной ситуации в теле М-функции в командное окно MATLAB выдается только сообщение о том, в какой именно строке М-функции возникла ошибочная ситуация, но никакой точной диагностической информации не сообщается. Для получения строковой диагностической информации в теле М-функции, следует вызывать функцию **eval** с двумя аргументами, указав строковым значением второго аргумента имя М-функции, предназначенной для обработки ошибок. В последней функции (то есть в функции обработки ошибок) следует использовать функцию **lasterr**, которая и возвратит эту диагностическую информацию.

По сравнению с функцией **eval**, которая хорошо подходит для вычисления сложных выражений, функция **feval** более удобна для вычисления

одиночной функции, которой при этом можно передать дополнительные параметры:

```
feval(str, x1, x2,      xN)
```

Здесь строка **str** задает имя функции, подлежащей вычислению при значениях аргументов **x1**, **x2**, ..., **xN**, например:

```
FunName = {'cos', 'sin', 'tan'};
m = input('Enter function index: ');
x = feval( FunName{m}, 0.5 );
```

Если с клавиатуры будет введено целое число 2, то с помощью функции **feval** будет вычислено значение sin(0.5), равное 0.4794.

7.6. Локальные, глобальные и статические переменные

М-функции располагают собственным пространством переменных, изолированным от рабочего пространства MATLAB. Поэтому совпадение имен переменных из рабочего пространства и имен внутренних переменных М-функций не приводит ни к каким коллизиям.

Переменные, которые используются в теле М-функции и не совпадают с именами формальных параметров этой функции, называются локальными. Их область действия полностью ограничена рамками тела данной М-функции, говоря иначе, они видимы лишь в пределах М-функции. Из рабочего пространства MATLAB и из других М-функций они не видны и не достижимы. Аналогично, локальные внутри некоторой функции переменные не видны из другой М-функции и из рабочего пространства.

Основным средством передачи информации из командного окна MATLAB в М-функцию и из одной функции в другую является механизм параметров функции. Другим механизмом являются глобальные переменные. Для того чтобы рабочая область MATLAB и несколько М-функций могли совместно использовать некоторую переменную с заданным именем, ее всюду нужно объявить как глобальную с помощью ключевого слова **global**:

```
>> global b
>> b=2;
>> whos b
  Name     Size          Bytes  Class
  b        1x1               8  double array (global)
Grand total is 1 element using 8 bytes
>>
```

Команда **whos** возвращает информацию о всех переменных, расположенным в рабочей области MATLAB. Глобальные переменные отмечаются при этом ключевым словом **global**. Команда

```
who global
```

выдает справочную информацию только по глобальным переменным.

Глобальные переменные (в отличие от других переменных) автоматически инициализируются пустыми массивами:

```
>> global c
>> c
c =
    []
>>
```

Глобальные переменные, явно инициализированные в командном окне MATLAB, можно использовать в теле М-функции, не производя в последней никаких начальных присваивании этой переменной. Ниже приведен пример обращения к функции, зависящей от глобальной переменой:

```
% листинг функции FuncWithGlobal
function res = FuncWithGlobal(x)
global glVars
res = x+glVars;
glVars=glVras-1;
>> global glVars    задаем глобальную переменную
>> glVars = 5;
>> r = FuncWithGlobal(1)
r =
  6
>> glVars
glVars =
  4
```

В командном окне MATLAB переменной **glVars** присваевается значение 5, после чего с аргументом 1 вызывается функция **FuncWithGlobal**, в теле которой значение этой глобальной переменной используется в вычислениях. При первом выполнении функции **FuncWithGlobal** глобальная переменная **glVars** изменяет свое первоначальное значение и становится равной 4. При повторном выполнении (вычислении) функции **FuncWithGlobal** с тем же самым входным параметром, равным единице, она возвратит уже иное по сравнению с первым вызовом значение, равное 5. Отмеченный эффект обусловлен свойством глобальных переменных сохранять свои значения между последовательными вызовами функции.

Аналогичного эффекта можно добиться и при использовании локальных (внутри функции) переменных, если объявить их с ключевым словом **persistent**. Тогда эти переменные, оставаясь по-прежнему недоступными вне М-функции, сохраняют свои значения между последовательными вызовами этой функции. Такие статические локальные переменные неявно инициализируются пустыми массивами при первом (в рамках заданного сеанса работы с MATLAB) входе в М-функцию. Например, следующая функция

```
function y = MyPersistFun( x )
persistent var;
if isempty(var)
  var = 1;
end
```

```
y = var .*x;
var = y;   сохранение для использования при последующем вызове
```

имеет статическую («стабильную») переменную **var**, которая неявно инициализируется пустым массивом. Здесь инициализация осуществляется только один раз при первом входе в функцию. Этим данная операция отличается от операции присваивания, которая выполняется каждый раз при любом входе в функцию. Далее при первом входе в эту функцию в условном операторе переменной var присваивается значение 1. При последующих обращениях к функции **MyPersistFun** переменная var уже не является пустой, поэтому поток управления (последовательность выполнения операторов) обходит условный оператор. Из объяснений деталей работы функции **MyPersistFun** следует, что результатом циклического вызова данной функции из командного окна или другой М-функции

```
>> for i=1:5
     res = MyPersistFun( i );
end
```

будет значение 5!:

```
res =
     120
```

Так как статическая переменная var обладает свойством сохранять свои значения между вызовами функции **MyPersistFun**, то повторное исполнение приведенного выше циклического вызова этой функции приведет уже к другому значению переменной **res**:

```
res =
     14400
```

поскольку ранее полученное значение 120 последовательно умножается на 2, 3, 4 и 5. Для того, чтобы обнулить значение статической переменной следует выполнить команду

```
clear all
```

7.7. Рекурсивные функции. Производительность М-функции

Прежде чем приступить к сравнению между собой функций, реализующих различные алгоритмы решения одной и той же задачи, рассмотрим еще один вариант решения задачи о вычислении факториала целого положительного числа. Этот вариант основан на применении рекурсивных функций.

Функция называется рекурсивной, если в процессе вычислений она осуществляет вызов самой себя. Вот абсолютно формальный пример описанной ситуации:

```
function y = SimpleRecursF( x )

z = SimpleRecureF(t) + x.*x;
```

Так как в теле функции **SimpleRecursF** осуществляется вызов этой же функции (с другим аргументом), то по определению данная функция является рекурсивной. В общем случае создавать рекурсивные функции не сложнее нерекурсивных, однако при этом следует тщательно прослеживать последовательность вложенных вызовов таких функций для установления факта завершения этого процесса. Здесь достаточно легко допускаются ошибки, связанные с зацикливанием рекурсивных вызовов, поэтому в ряде случаев окончательного выхода из такой функции не происходит вообще. Для недопущения зависания компьютера в этой ситуации в MATLAB (в отличие от программирования на языке C) все M-функции выполняются под полным контролем самой системы, которая ограничивает предельное количество рекурсивных вызовов. С одной стороны, это положительное свойство, но с другой стороны — это слишком сильное вторжение в процесс выполнения рекурсивных функций, что ограничивает их самостоятельную роль.

Необходимо отметить, что роль рекурсивных функций в MATLAB существенно ниже, чем в системах программирования на языках C/C++, так как наиболее интересные случаи их использования базируются на применении указателей, которых в M-языке MATLAB нет вообще. Завершая обсуждение рекурсивных функций, приведем в качестве примера рекурсивную функцию, возвращающую изображение фрактального (самоподобного) объекта, называемого ковром Серпиского.

```
% листинг файла Serpinsky.m
function z=Serpinsky(Lmax)
x1=0;
y1=0;
x2=1;
y2=0;
x3=0.5;
y3=sin(pi/3);
h=figure(1);hold on;axis off;
fill([x1 x2 x3],[y1 y2 y3],'b');
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
Simplex(x1,y1,x2,y2,x3,y3,0,Lmax);
hold off

function z=Simplex(x1,y1,x2,y2,x3,y3,n,Lmax)
if n<Lmax
  dx=(x2-x1)/2;
  dy=(y3-y1)/2;
  x1n=x1+dx;
  y1n=y1;
  x2n=x1+dx+dx/2;
  y2n=y1+dy;
  x3n=x1+dx/2;
  y3n=y1+dy;
  fill([x1n x2n x3n],[y1n y2n y3n],'w');
```

```
    n=n+1;
    Simplex(x1,y1,x1n,y1n,x3n,y3n,n,Lmax);
    Simplex(x1n,y1n,x2,y2,x2n,y2n,n,Lmax);
    Simplex(x3n,y3n,x2n,y2n,x3,y3,n,Lmax);
end
```

Для построения ковра Серпинского 5-го порядка необходимо выполнить следующую команду:

```
>> Serpinsky(5)
```

Результат ее выполнения представлен на рис. 7.6.

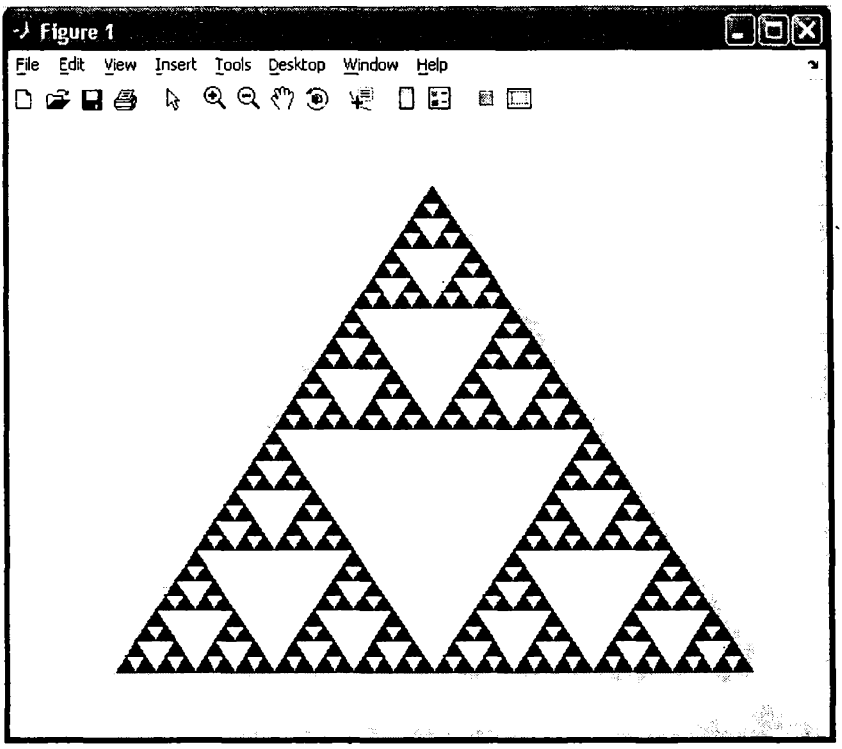


Рис. 7.6. Ковер Серпинского 5-го порядка

Ясно, что алгоритмы решения одной и той же задачи отличаются друг от друга по различным критериям, и, в первую очередь, по времени вычислений. MATLAB поставляет удобное средство для приблизительного замера данной характеристики — команды **tic** и **toc**:

```
>> tic,Serpinsky(5),toc
 elapsed_time =
    1.0120
>>
```

Следует, однако, иметь в виду, что время выполнения замеряется приблизительно и зависит от ряда неконтролируемых факторов, поэтому такие вызовы следует повторить несколько раз, а полученные замеры усреднить.

В заключение данного раздела дадим несколько практических советов. При программировании М-функций следует старательно избегать применения массивных циклических вычислений на базе операторов цикла М-языка. Это можно сделать, применив вместо них эффективные групповые операции М-языка. В том случае, когда этого сделать не удается следует подумать о реализации циклических алгоритмов на языке С, поскольку в MATLAB имеется принципиальная возможность подключить для совместной работы даже такие программные модули.

Можно добиться повышения эффективности выполнения М-функций за счет специальной настройки среды MATLAB. Дело в том, что при первой загрузке содержимого М-файла в память компьютера система MATLAB осуществляет синтаксический разбор кода, переводит его во внутренний формат, называемый псевдокодом или р-кодом. После чего сохраняет полученный псевдокод в памяти компьютера в течение всего сеанса работы с MATLAB. В результате первый вызов М-функций осуществляется медленнее (обычно ненамного медленнее) повторных вызовов. Можно несколько повысить эффективность первого выполнения М-функций, выполнив команду

```
pcode имя_М-функции
```

Эта команда транслирует содержимое М-файла в псевдокод и записывает на диск файл с тем же именем, но с расширением **.p**, после чего загрузка М-функций будет выполняться именно из этого файла и первое исполнение функции пройдет быстрее. Кроме того, можно будет стереть файл с расширением **.m**, а содержимое оставшегося файла с псевдокодом будет невозможно прочитать в текстовых редакторах, так как формат этих файлов бинарный. В результате можно будет скрыть «тайные» детали разработанных вами алгоритмов.

7.8. М-функции с переменным числом входных параметров и выходных значений

В определениях М-функций, которые мы рассматривали до сих пор, использовалось строго фиксированное конечное число входных параметров и выходных значений. Однако бывают ситуации, когда полезно иметь одну функцию, которая штатно обрабатывает переменное число входных параметров и выходных значений. К таковым относится, например, функция, вычисляющая, сумму длин произвольного числа векторов. Такая функция должна уметь распознавать реальное количество параметров, с которыми она вызвана.

В М-языке MATLAB такая возможность базируется на использовании массива ячеек. В определении М-функций параметр, через который передается заранее неизвестное число входных аргументов, следует обозначить ключевым словом **varargin**. Таким ключевым словом обозначается массив ячеек, в который упакованы данные параметры. Функция всегда может узнать истинное число аргументов, упакованных в параметре **varargin**, применив для этого функцию **length**.

Ниже представлен листинг функции, вычисляющей сумму квадратов длин произвольного количества вектор-строк:

```
function SumLen = NumLength(varargin)
n = length( varargin );
SumLen = 0;
for k = 1   n
  SumLen = SumLen+varargin{k}(1)^2+varargin{k}(2)^2;
end
```

Если аргумент **varargin** не единственный в списке параметров, то он должен стоять последним. В рассмотренном примере с помощью фигурных скобок мы извлекаем содержимое отдельной ячейки, то есть вектор, а с помощью дальнейшей индексации круглыми скобками извлекаем первую и вторую координаты вектора.

При вызове функции **NumLength** не нужно (и нельзя) упаковывать входные числовые вектор-строки в массив ячеек, так как MATLAB делает это сам. Достаточно перечислить их в качестве фактических параметров через запятую:

```
NumLength([1 2], [3 4] )
ans = 30
```

Теперь вызовем функцию **NumLength** с другим числом параметров:

```
NumLength([1 2], [3 4], [5 6])
ans=91
```

Из приведенных примеров видно, что функция легко обрабатывает оба этих случая, правильно вычисляя суммарную длину входных векторов.

В определении М-функции переменное число возвращаемых значений упаковывается в массив ячеек, обозначаемый ключевым словом **varargout**:

```
function varargout = MyFunc3(X)
```

Здесь в массив ячеек с именем **varargout** можно в теле функции упаковать произвольное число выходных значений. Допустим, что на вход функции **MyFunc3** может подаваться в качестве единственного входного параметра массив разных размерностей и размеров. Мы хотим возвращать несколько скаляров, каждый из которых является размером входного массива вдоль одного из его измерений. Так как количество измерений заранее неизвестно, то его можно определить в теле функции динамически и на ходу упаковать все эти скаляры в единственную выходную переменную **varargout**:

```
function varargout = MyFunc3(X)
n = ndims(X);
for i = 1 : n
  varargout(i) = {size(X,i)};
end
```

Здесь функция **size(x,i)** вычисляет размер массива **x** вдоль *i*-го направления (измерения). Ниже показаны два примера использования этой функции:

```
>> A= [ 1 2 3; 4 5 6 ];
>> [m n]=MyFunc3(A);
>> m
m =
  2
>> n
n =
  3
```

Сформируем трехмерный массив C и передадим его в качестве аргумента в функцию **MyFunc3**:

```
>> B=[ 4 5 6; 9 8 7];
>> C(:,:,1) = A;
>> C(:,:,2) = B;
>> [m,n,k]=MyFunc3(C);
>> m
m =
  2
>> m
n =
  3
>> k
k =
  2
```

Если нас интересует размеры этого массива вдоль только первых двух измерений, то следует вызвать функцию **MyFunc3** в следующем формате

```
>> [m,n]= MyPunc3(C);
```

который является абсолютно корректным с точки зрения синтаксиса М-языка. При этом третье возвращаемое функцией **MyFunc3** выходное значение теряется.

7.9. Контроль входных параметров и выходных значений М-функций

Несовпадение типов, а также числа фактических (с которыми реально работает функция) и формальных (которые передаются в функцию) параметров является причиной неправильной работы, а, в ряде случаев, аварийного останова М-функций. Для предотвтащения подобной ситуации при

разработке соответствующей М-функции следует размещать в теле функции блоки команд, осуществляющие проверку типов и количества входных значений.

Рассмотрим способы решения данной проблемы на примере рассмотренной выше функции **MatrProc1**. Напомним, что ранее предполагалось, что аргументами данной функции являются массивы одинакового размера. В результате, если пользователь по ошибке задаст формальные параметры в виде массивов разных размеров, то в процессе ее выполнения возникнет ошибка. Для того чтобы избежать данной ситуации, следует организовывать в теле функции **MatrProc1** проверку размерностей первого и второго передаваемых параметров:

```
function [ A, B ] = MatrProcI(X1, X2, x)
n1 = ndims(X1);
n2 = ndims(X2);
% — проверка размерностей ——
if n1 == n2
  error('Different dimensions')
else
  for i-1:n1
    if size(X1,i)~=size(X2,i)
        error('Different sizes of 1st and 2nd parameters')
    end
  end
end
4 — вычисления ————————————————
A = X1.*X2*x;
B =X1  *X2+x;
```

Теперь при вызове функции **MatrProc1** с несовпадающими размерами первого и второго аргументов, стандартная функция MATLAB error будет корректно останавливать работу и выводить в командное окно диагностические сообщения (аргументы функции **error**). После этого пользователю достаточно повторно вызвать функцию **MatrProc1** с правильными параметрами.

Для большей надежности следует добавить проверку третьего параметра на скалярность (в MATLAB скаляры являются матрицами 1×1). Для этого следует дополнить приведенный выше листинг функции **MatrProc1** следующим фрагментом кода:

```
[m ,n] = size(x);
if (m~=1 | n~= 1 )
   error('3-d parameter must be scalar')
end
```

Наконец, целесообразно проверить общее число элементов в списке формальных параметров. Для этой цели в MATLAB специально предусмотрена переменная с именем **nargin**. Ее значением является количество аргументов, фактически переданное функции при ее вызове. Проверка числа формальных параметров выполняется следующим образом:

```
if nargin ~= 3
   error('Bad number of parameters')
end
```

Также в MATLAB предусмотрена переменная **nargout**, содержащая число возвращаемых значений в текущей форме вызова этой функции. Например, вызов

```
[ s1, s2, s3 ] = MatrProc1(x1, x2, x)
```

предполагает получить три возвращаемых значения, в то время как из определения функции следует, что возвращаемых значений у этой функции два. Для того чтобы предотвратить такой формат вызова функции **MatrProc1** и предупредить пользователя функции о несовпадении числа ожидаемых возвращаемых значений их номинальному числу, следует в теле функции осуществить проверку переменной **nargout**:

```
if nargout ~=2
   error('Must be 2 return values')
end
```

Выполненные дополнения функции **MatrProc1** обеспечивают обращение к функции только с правильным числом входных параметров и возвращаемых значений. Напомним, что ранее мы встречались со встроенными функциями MATLAB, которые вызывались с разным числом входных параметров (и это очень типично). При этом фактически разные, но близкие друг другу по логике варианты работы функции, выполнялись под одним и тем же именем, что весьма наглядно и удобно. Таковой функцией, например, является функция **plot**, осуществляющая построение графиков функций.

Таким образом, при разработке М-функций целесообразно предусматривать многовариантность схем вычислений, выбор которых зависит от числа выходных аргументов. При этом необходимо предусмотреть проверку их числа, и, вместо прекращения выполнения функции, обеспечить выполнение соответствующей ветви реализуемого алгоритма. Выше сказанное в полной мере относится и к числу возвращаемых значений. Например, функция **TestFunc2**

```
function [res1, res2] = TestFunc2(var1, var2)
switch nargin
  case 1
  if nargout == 1
     res1 == var1*2;
  elseif nargout == 2,
     res1=var1*2;
     res2=var1+3;
  else error('Must be 1 or 2 return values');
end
case 2
  if nargout == 1
     res1 = var1.*var2;
  elseif nargout == 2
     res1=var1.*var2;
     res2=var1+3;
```

```
    else error('Must be 1 or 2 return values');
    end
    otherwise error('Must be 1 or 2 parameters')
  end
```

допускает несколько вариантов заранее предусмотренных форматов вызовов.

Для краткости здесь опущена проверка размеров входных параметров, подробно рассмотренная выше. Кроме того, здесь для большей наглядности отслеживается ситуация превышения числа параметров и возвращаемых значений над их номинальным числом (в данном случае 2).

В заключение отметим, что степень подробности проверок зависит от назначения функции. Если М-функция создается для собственного использования, то проверки могут быть менее строгими, так как выполнение М-функций осуществляется в режиме интерпретации под полным контролем MATLAB. Возникающие ошибки автоматически обрабатываются самой этой системой, и в командное окно выдается соответствующее диагностическое сообщение.

Однако при разработке функций для внешнего потребления, проверки нужно сделать более жесткими, так как внешнему пользователю, как правило, трудно разобраться во всех деталях работы вашей функции.

MATLAB штатно поставляется с большим числом встроенных М-функций. Текст этих функций выводится в командное окно командой

```
type имя_функции
```

это позволяет изучить как детали реализации некоторого алгоритма, так и научиться различным приемам программирования, в частности, способам и глубине проверок входных параметров и выходных значений.

Рассмотрим текст поставляемой с ядром системы MATLAB функции **repmat**:

```
>> type repmat


function B = repmat(A,M,N)
%REPMAT Replicate and tile an array.
%   B = repmat(A,M,N) creates a large matrix B consisting
%   of an M-by-N
%   tiling of copies of A.
%
%   B = REPMAT(A,[M N]) accomplishes the same result
                                              as repmat(A,M,N)
%
%   B = REPMAT(A,[M N P ...]) tiles the array A to produce a
%   M-by-N-by-P-by-.   block array.  A can be N-D.
%
%   REPMAT(A,M,N) when A is a scalar is commonly used to produce
%   an M-by-N matrix filled with A's value.  This can be
%   much faster
%   than A*ONES(M,N) when M and/or N are large.
%
```

```
%   Example:
%       repmat(magic(2),2,3)
%       repmat(NaN,2,3)
%
%   See also MESHGRID.


%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 1.17 $  $Date: 2002/04/08 20:21:10 $


if nargin < 2
   error('Requires at least 2 inputs.')
elseif nargin == 2
  if length(M)==1
     siz = [M M];
  else
    siz = M;
  end
else
  siz = [M N];
end


if length(A)==1
  nelems = prod(siz);
  if nelems>0
   % Since B doesn't exist, the first statement creates a B with
   % the right size and type.  Then use scalar expansion to
   % fill the array.  Finally reshape to the specified size.
   B(nelems) = A;
   B(:) = A;
   B = reshape(B,siz);
  else
   B = A(ones(siz));
  end
elseif ndims(A)==2 & length(siz)==2
 [m,n] = size(A);
  mind = (1:m)'
  nind = (1:n)'
  mind = mind(:,ones(1,siz(1)));
  nind = nind(:,ones(1,siz(2)));
  B = A(mind,nind)
else
  Asiz = size(A);
  Asiz = [Asiz ones(1,length(siz)-length(Asiz))];
  siz = [siz ones(1,length(Asiz)-length(siz))];
  for i=length(Asiz):-1:1
   ind = (1:Asiz(i))';
   subs{i} = ind(:,ones(1,siz(i)));
  end
  B = A(subs{:});
end
>>
```

Текст функции **repmat** весьма полезен для изучения. Во-первых, здесь показан эффективный алгоритм решения задачи о повторении (репликации) заданной матрицы **A** в вертикальном направлении m раз и в горизонтальном направлении n раз (мы его уже рассматривали в предыдущей главе данного пособия). Для решения этой задачи здесь используются высокоэффективные операции М-языка: множественная индексация и индексация матрицами. Во-вторых, данный пример показывает, что весьма значительный процент от общего объема программного кода занимают многочисленные проверки входных параметров и выходных значений.

В-третьих, видно, что очень большое значение уделяется документированию кода. Это важно как для сторонних пользователей функции, так и для самих разработчиков, поскольку документирование помогает им в процессе разработки и отладки алгоритма и деталей кодировки.

7.10. Практические советы по разработке и отладке М-функций

В процессе разработки функции, в первую очередь, разрабатываем алгоритм решения некоторой задачи, после чего осуществляется его перевод на формальный язык кодирования, которым и является М-язык. Несмотря на довольно высокую наглядность М-языка (в нем отсутствуют низкоуровневые конструкции близкие к машинным кодам), он все равно остается формальным языком. С течением времени детали разработок с неизбежностью забудутся, поэтому при модификации соответствующей функции придется тратить время на повторное изучение программного кода.

Для упрощения процесса дальнейшей модификации функции, а также ее отладки в текст функции следует вставлять комментарии. Объем комментариев может быть весьма большим (см., например, листинг функции **repmat**).

Напомним, что комментарии могут размещаться как в отдельных строках, начинающиехся с символа %, после которого следует текст комментария, так и в одной строке с исполняемой командой также после символа %. Также комментарии можно располагать в конце любой строки кода, поскольку интерпретатор М-языка, встретив знак %, считает все символы после него просто комментарием (а не командами, подлежащими переводу в машинную форму и исполнению).

Особую роль в MATLAB имеют комментарии, располагающиеся в смежном наборе строк сразу за заголовком определения функции, которые выводятся в командное окно MATLAB при выполнении команды

```
help имя_М-функции
```

Так как данная информация, в первую очередь, представляет интерес для пользователей функции (а не разработчика), то желательно расположить в этих комментариях описательную информацию и сведения о правильном синтаксисе вызова этой функции.

Остановимся более подробно на вопросе об отладке М-функций, точнее на приемах, с помощью которых можно локализовать ошибки и определить причины их возникновения. При возникновении ошибки в процессе выполнения М-функций, в командное окно выводятся приблизительное диагностическое сообщение и номер строки, в котором по мнению MATLAB произошла ошибка.

Другим, более развитым способом отладки функции являются применение точек останова (**Breakpoints**) и пошаговое выполнение функции, используя возможности редактора-отладчика MATLAB. Для создания точки останова нужно поместить курсор в выбраную строку и нажать функциональную клавишу **F12** (повторное нажатие этой клавиши убирает точку останова) или выполнить команду меню

Breakpoint ⇒ Set/Clear Breakpoint F12

После этого в строке слева появляется маркер в виде красного кружка, указывающий на то, что в данной строке проставлена точка останова. После этого, не закрывая окна **Редактора/Отладчика (Editor/Debugger)**, следует перейти в командное окно MATLAB и запустить из командной строки функцию на выполнение. После этого и произойдет останов выполнения функции прямо на строке, в которой поставлена точка останова (**Breakpoint**). В данном состоянии можно просматривать фактические значения входных параметров функции, текущие значения глобальных и локальных переменных, а также вычислять значения выражений, содержащие переменные, находящиеся в данный момент в рабочем пространстве. Для просмотра значения переменной, достаточно подвести курсор к ее имени в тексте функции, после чего на экране появится всплывающий желтый прямоугольник со значением переменной внутри него (рис. 7.7).

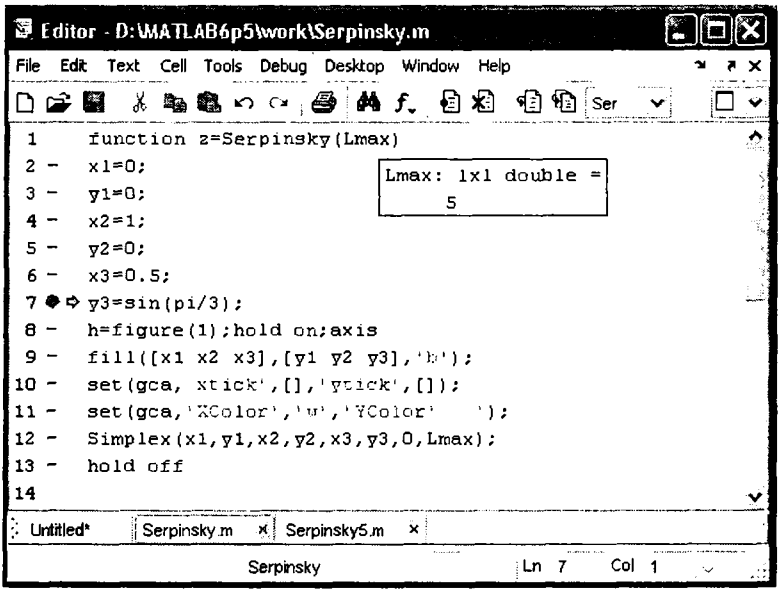


Рис. 7.7. Окно редактора файлов в режиме трассировки

Далее, нажимая клавишу **F10**, можно выполнять функцию построчно, каждый раз проверяя результаты такой пошаговой работы функции. В результате всегда можно локализовать ошибку и выявить ее причину. В результате такого итерационного отладочного процесса приходят к правильно работающим функциям.

Вопросы для самопроверки

1. Как в MATLAB осуществляется анимация?
2. Какая процедура называется векторизацией кода?
3. Что такое сценарий?
4. Какова структура m-файла, содержащего описание m-функции?
5. Как осуществляется взаимодействие m-функций с пользователем?
6. Что такое локальные, глобальные и статические переменные?
7. Какая функция называется рекурсивной?
8. Как создать функцию с переменным числом параметров?
9. Как осуществляется контроль входных и выходных значений m-функций?

Глава 8

Технологии создания графического интерфейса пользователя

В настоящее время одним из непреложных атрибутов любой прикладной программы является интерактивный интерфейс пользователя (Graphics User Interface — GUI), который разрабатывается для неоднократно решаемых задач с несколькими входными параметрами. Наличие интерфейса, во-первых, освобождает пользователя от необходимости вникать в детали программирования, во-вторых, позволяет менять исходные данные, не прекращая выполнения текущей программы, и в реальном времени наблюдать за изменения решения задачи. Сегодня выработаны достаточно общие требования, которым должны соответствовать программные интерфейсы, определены основные элементы управления графическим интерфейсом. Например, если речь идет об решении учебной и/или научной задачи обязательными элементами интерфейса должны быть:

1. Одно или несколько окон для визуализации результатов расчетов.
2. Редактируемые окна, позволяющие задавать значения входных параметров задачи.
3. Управляющие кнопки, предназначенные для запуска и остановки процесса расчета, вывода результатов, завершения работы с программой.
4. Поясняющие надписи (статический текст).

В данной главе рассматриваются технологии создания GUI средствами MATLAB.

8.1. Основные типы элементов управления

Графические элементы управления являются объектами дескрипторной графики MATLAB. К ним относятся командные кнопки, текстовые поля,

переключатели, списки, меню и ряд других элементов. Данные объекты относятся к типу **uicontrol**.

Как нам уже известно, все графические объекты в MATLAB создаются с использованием технологии объектно-ориентированного программирования, поэтому для изменения свойств графических объектов, необходимо получить доступ к их дескриптору — числу однозначно идентифицирующему данный объект. Отметим, что все элементы управления являются потомками от объекта «Графическое окно».

Для поиска описателей графических объектов можно использовать функцию **findobj**:

```
hArray = findobj(hParent, 'имя_объекта',значение)
```

Данная функция отыскивает всех потомков объекта с дескриптором **hParent**, имеющим для свойства '**имя_объекта**' значение указанное в параметре **значение** и возвращает их в массив **hArray** их описатели.

Для создания в графическом окне командной кнопки необходимо выполнить следующую последовательность команд:

```
>> hFl=figure;
>> uicontrol(hFl,'Style','pushbutton',..
                 'String','Моя кнопка',..
                 'Position',[10 10 70 30],...
                 'FontName','Times New Roman');
```

где в поле **Style** задается тип объекта управления **pushbutton**, в поле **String** — надпись на кнопке; в поле **Position** — координаты нижнего левого и правого верхнего углов кнопки, отсчитываемые от нижнего левого угла графического окна, а также ширина и высота кнопки. В результате получаем окно, представленное на рис. 8.1. Нажав левой кнопкой мыши на данную кнопку, можно убедиться в том, что она функционирует: нажимается и отжимается. Однако никаких других действий в результате нажатия кнопки не происходит. Это обусловлено тем, что мы пока не поставили в соответствие кнопке никаких функций, которые должны выполняться при нажатии на данную кнопку.

Для создания окна ввода и редактирования входной информации (объекта управления типа **edit**) необходимо выполнить следующую команду:

```
>> uicontrol(hFl,'Style','edit',
                 'String', 'Значение по умолчанию',...
                 'Position',[100 10 140 30],
                 'BackgroundColor','white', .
                 'HorizontalAlignment','left',
                 'FontName','Times New Roman');
```

где в поле **Style** задается объект управления **edit**, в поле **String** — текст, отображаемый в текстовом окне; в поле **Position** — координаты нижнего левого и правого верхнего углов кнопки, отсчитываемые от нижнего левого угла графического окна, а также ширина и высота кнопки; в поле **BackgroundColor** — цвет фона текстового окна, в поле **HorizontalAlignment** — тип выравнивания текста в окне по горизонтали. Результат выполнения данной команды представлен на рис. 8.2.

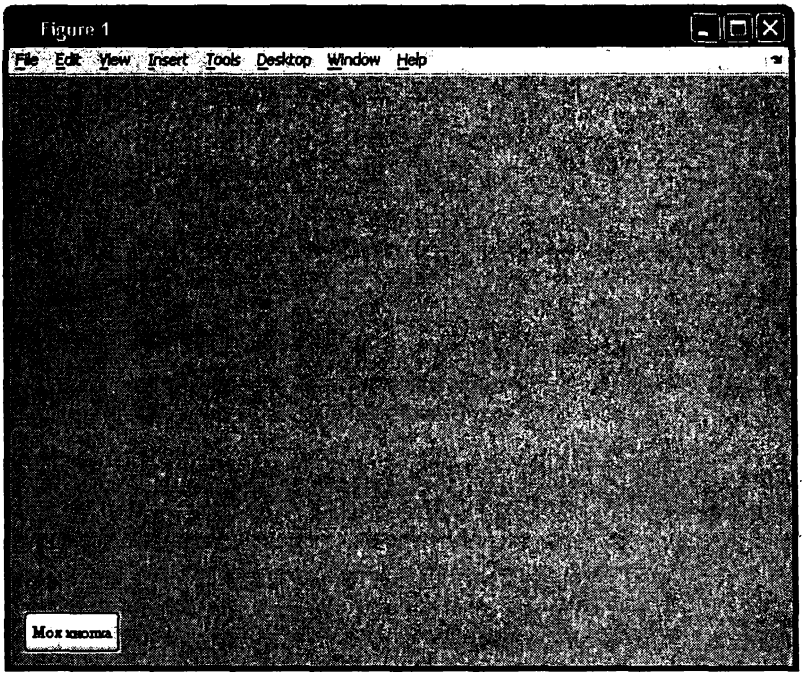


Рис. 8.1. Графическое окно с добавленной кнопкой

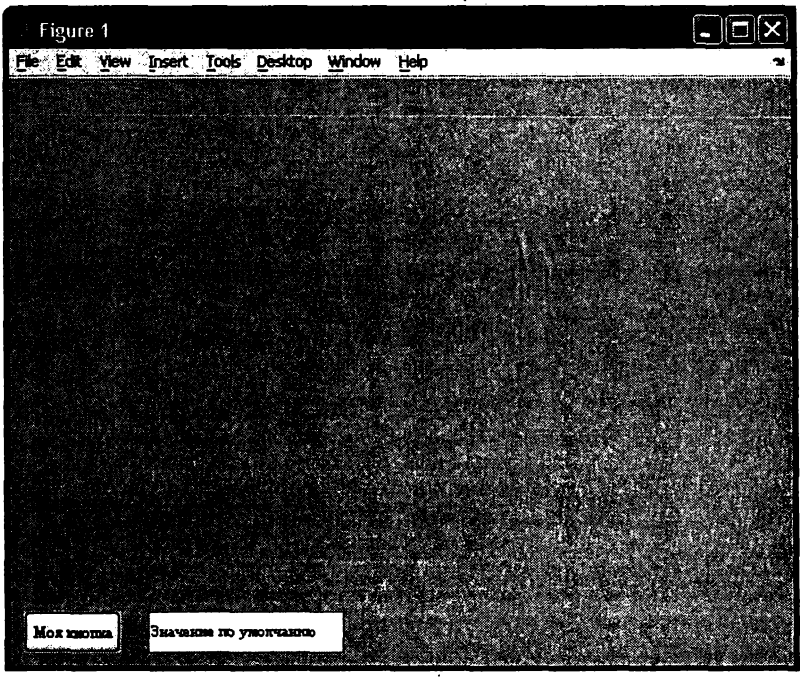


Рис. 8.2. Графическое окно с кнопкой и окном для редактирования текста

Для создания статического текста (не доступного для редактирования при использовании GUI) необходимо выполнить следующую команду:

```
>> uicontrol(hFl,'Style','text',...
                'Position',[100 50 140 30],...
                'BackgroundColor','white',...
                'String','Интерфейс пользователя',...
                'HorizontalAlignment','center',...
                'FontName','Times New Roman');
```

Результат выполнения данной команды представлен на рис. 8.3.

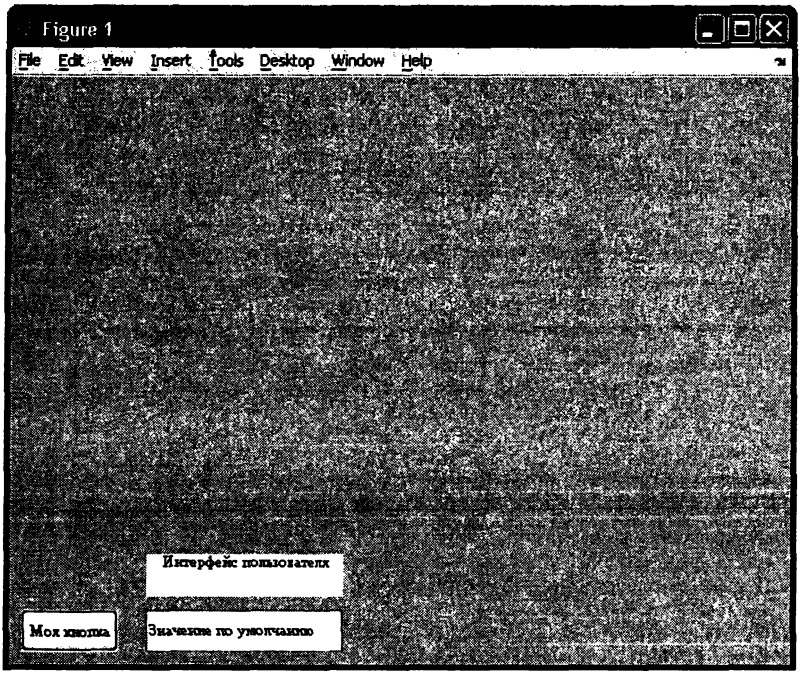


Рис. 8.3. Графическое окно с добавленными кнопкой, окном редактирования и окном отображения статического текста

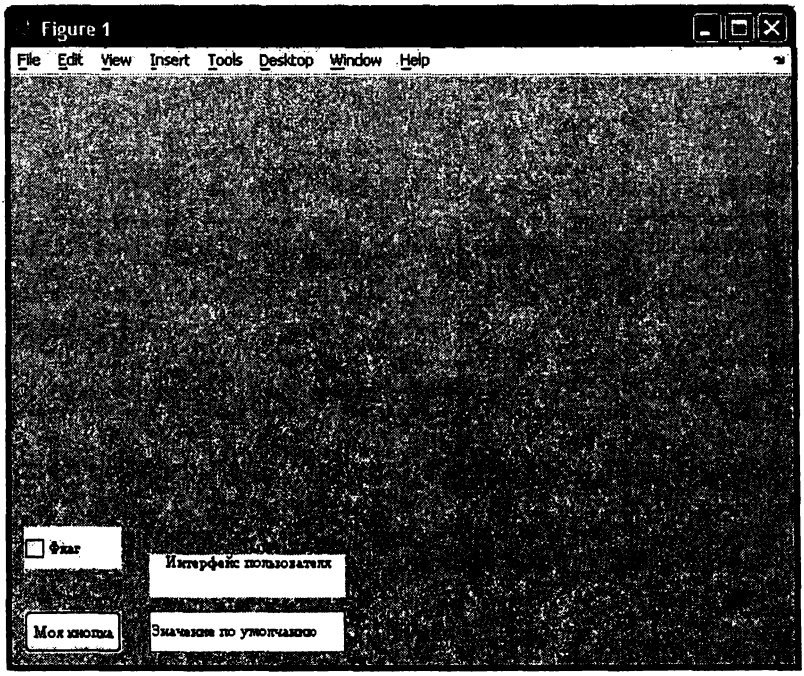


Рис. 8.4. Графическое окно с дополнительным элементом управления класса **checkbox**

Для создания элемента **checkbox** (флажок), который может находится в двух состояниях (неотмеченном и отмеченном) необходимо выполнить следующую команду:

```
>> uicontrol(hFl,'Style','checkbox',...
                'Position',[10 70 70 30],...
                'String','Флаг',...
                'HorizontalAlignment','left',...
                'FontName','Times New Roman');
```

Результат выполнения данной команды представлен на рис: 8.4.

Для создания элемента управления **togglebutton** (кнопка с залипанием), который может находиться в одном из двух состояний (отжатом и нажатом), необходимо выполнить следующую команду:

```
>> hF2=figure;
>> uicontrol(hF2,'Style','togglebutton',...
                'String','Кнопка с залипанием',...
                'position',[10 70 130 30],...
                'FontName','Times New Roman');
```

Результат выполнения данной команды представлен на рис. 8.5.

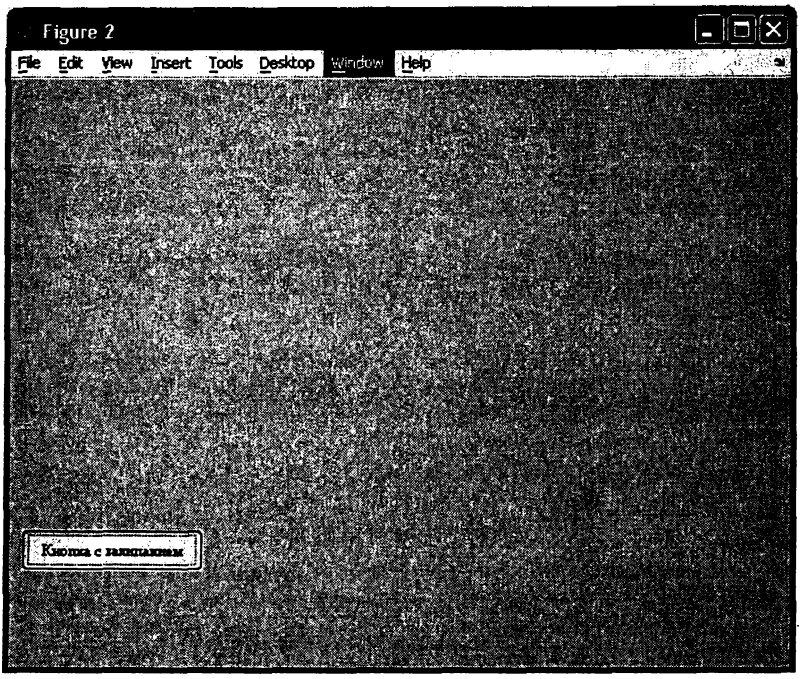


Рис. 8.5. Графическое окно с элементом управления **togglebutton**

Другим графическим элементом управления, предназначенным для осуществления выбора одного из нескольких вариантов (двух или более) того или иного набора параметров, является элемент **radiobutton**. В качестве примера приведем команды, позволяющие создать три элемента **radiobutton**:

```
>> uicontrol(hF2,'Style','radiobutton',...
                'String','Поле 1',...
                'Position',[120 10 70 30],...
                'FontName','Times New Roman');
>> uicontrol(hF2,'Style','radiobutton',...
                'String','Поле 2',...
                'Position',[200 10 70 30],...
                'FontName','Times New Roman');
>> uicontrol(hF2,'Style','radiobutton',...
                'String','Поле 3',...
                'Position',[280 10 70 30],...
                'FontName','Times New Roman');
```

Результат выполнения данных команд представлен на рис. 8.6.

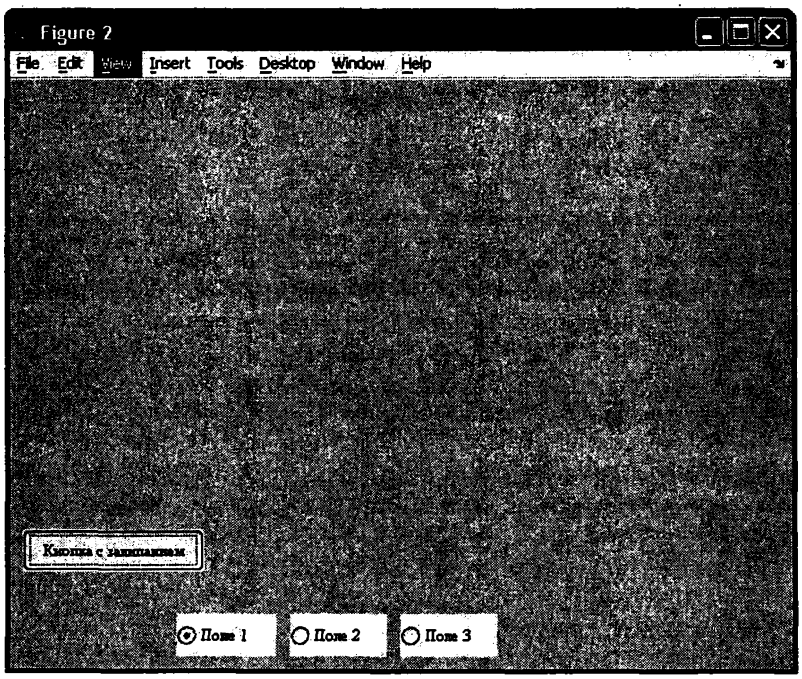


Рис. 8.6. Графическое окно с элементом управления **togglebutton** и тремя элементами **radiobutton**

Для создания элемента управления типа **slider** (полоса скроллинга), используются команды:

```
>> uicontrol(hF,'style','slider',
                'Position',[10 90 70 30], % задание положения
                                          % полосы скроллинга
                'min',1, % минимальное значение переменной,
                         % значение которой соответствует
                         % координате
                         % бегунка полосы скроллинга
                'max',10, % максимальное значение переменной,
                          % значение которой соответствует
                          % координате
                          % бегунка полосы скроллинга
                'Value',3, % начальная координата бегунка
                           %  полосы скроллинга
                'Sliderstep',[0.5 0]); % шаг перемещения бегунка
```

Результат выполнения данной команды представлен на рис. 8.7.

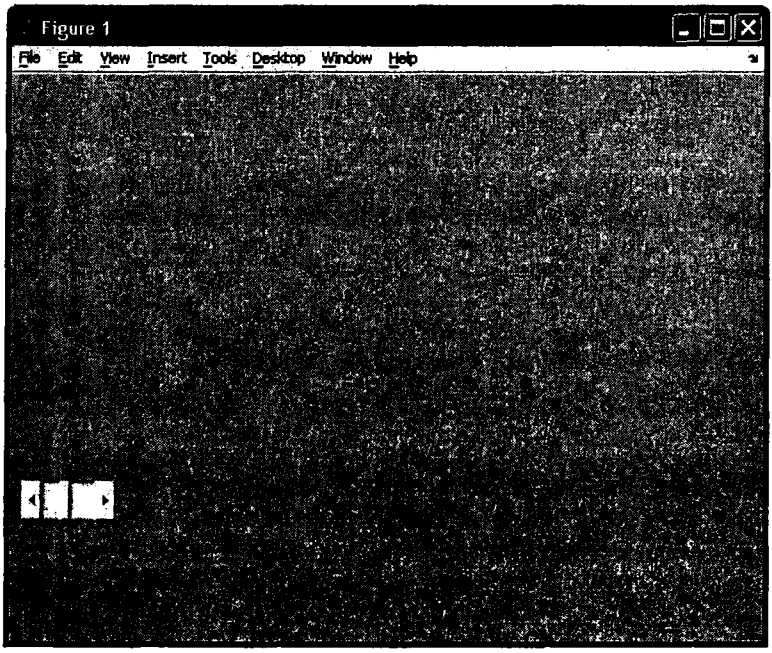


Рис. 8.7. Графическое окно с горизонтальным элементом управления **slider**

Элемент управления **slider** позволяет перемещать мышью ползунок, относительное положение которого задает некоторое число. При этом первая координат вектора, занесенного в поле **Sliderstep**, используется для вычисления следующей координаты при нажатии левой кнопки мыши по правой кнопке горизонтальной полосы скроллинга по формуле

$$x_{new} = x_{old} + (x_{\max} - x_{\min}) \cdot Sliderstep(1).$$

Вертикальная полоса скроллинга создается выполнением следующей команды:

```
>> hF3=figure;
>> uicontrol(hF3,'style','slider',...
                'Position',[100 90 10 70],...
                'min',1,...
                'max',20,...
                'Value',3,...
                'Sliderstep',[0.5 0.5]);
```

Результат ее выполнения представлен на рис. 8.8.

```
>> hF4=figure;
>> uicontrol(hF4,'style','slider',...
                'Position',[100 90 10 70],...
                'min',1,...
                'max',20,...
```

```
'Value',3,...
'Sliderstep',[0.5 0]);
```

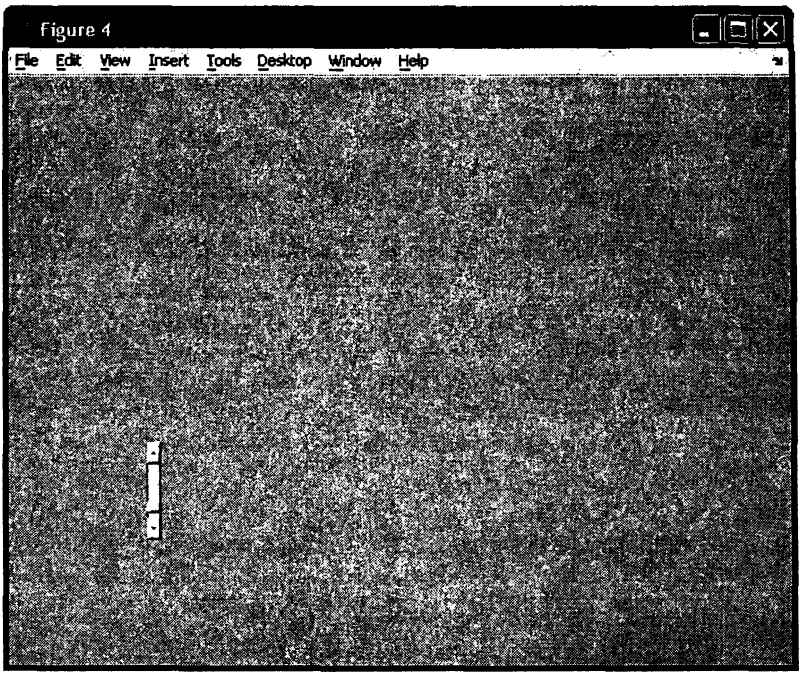


Рис. 8.8. Графическое окно с элементом управления вертикальный **slider**

Крайнее левое положение ползунка соответствует свойству **Min** данного элемента, крайне правое положение — свойству **Max**. По умолчанию значению полей **Min** и **Max** соответствуют значения 0 и 1, соответственно. Для того, чтобы получить значение текущей координаты бегунка скроллинга необходимо выполнить следующие команды:

```
>> tmpS=get(hF4,'Children'); % получение дескриптора полосы
                             % скроллинга
>> x=get(tmpS,'Value') % получение координаты бегунка скроллинга
x =
   7.9667
```

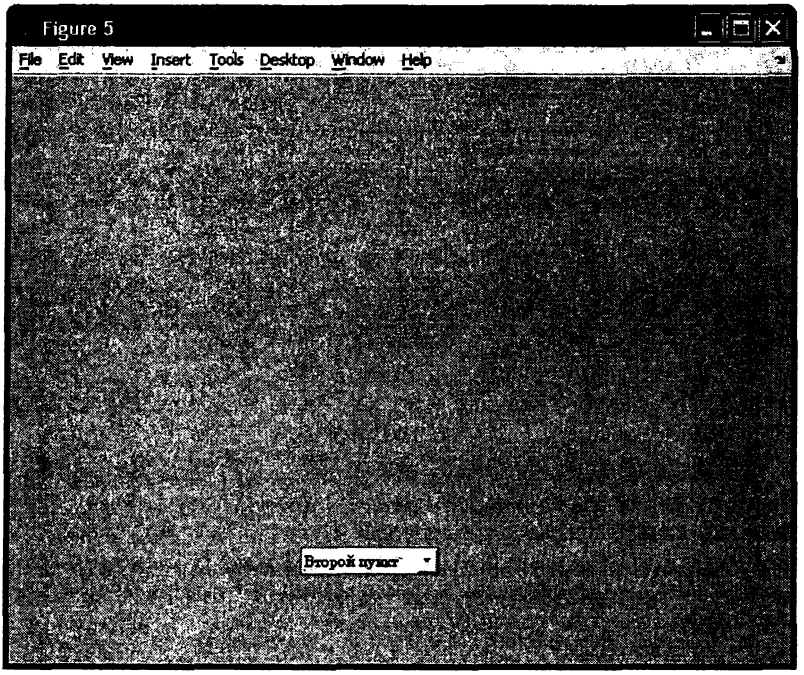


Рис. 8.9. Графическое окно с элементом управления **popupmenu**

Для создания элемента управления **popupmenu** (меню с раскрывающимся списком) используется следующая команда:

```
>> hF5=figure;
>> uicontrol(hF5,'Style','popupmenu',...
                'String',{'Первый пункт','Второй пункт',
                'Третий пункт'},...
                'Position',[210 60 100 25],...
                'Value','FontName','Times New Roman');
```

Результат ее выполнения представлен на рис. 8.9.

Отметим, что в обычном состоянии данный элемент свернут в одну строку, в которой отображается название последнего выбранного пункта меню (рис. 8.10) или пункта меню, зафиксированного значением поля **Value** (рис. 8.9).

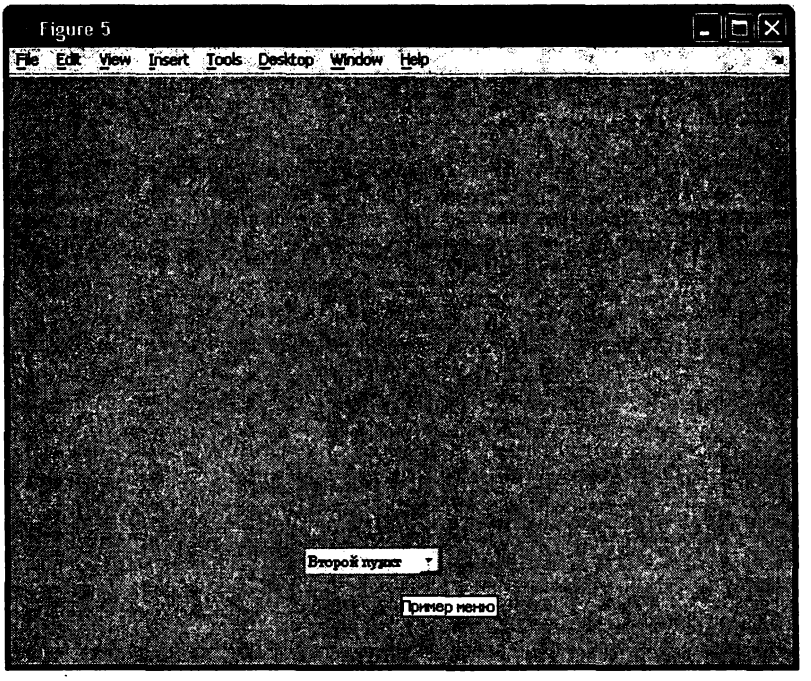


Рис. 8.10. Результат нажатия на кнопку элемента управления **popupmenu**

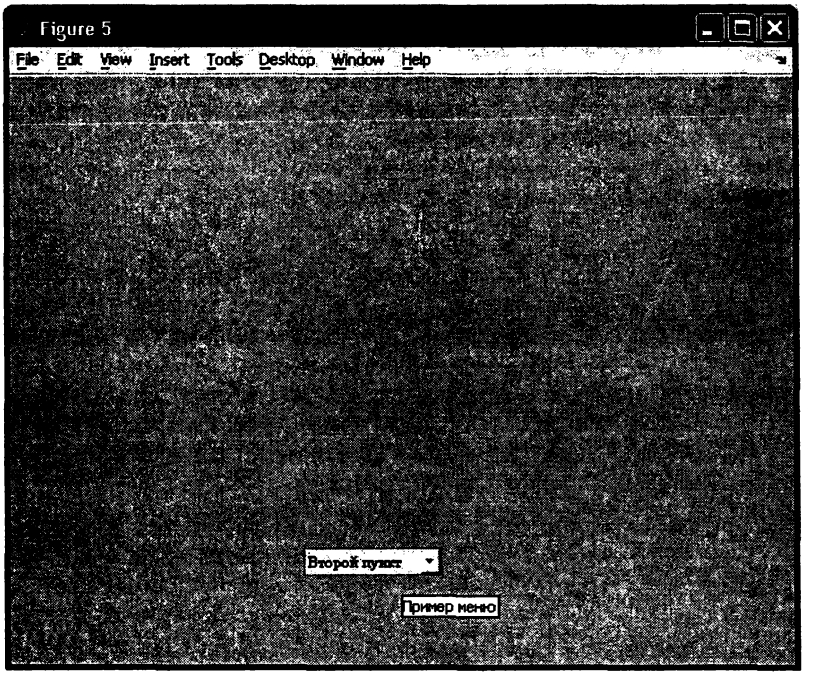


Рис. 8.11. Пример всплывающей подсказки

Одним из требований, предъявляемых к современному интерфейсу пользователя, является наличие всплывающих подсказок (**tooltip**), которые появляются при наведении мыши на данный элемент управления. Для создания всплывающей подсказки достаточно в функции **unicontrol** явно указать текстовое значение свойства **TooltipString**, которое и будет содержанием всплывающей подсказки, например:

```
>> hF=figure;
>> uicontrol(hF,'Style','popupmenu',
                'String',{'Первый пункт','Второй пункт',
                'Третий пункт'},
                'Position',[210 60 100 25],
                'Value',2,
                'TooltipString','Пример меню',
                'FontName','Times New Roman');
```

Результат выполнения данных команд представлен на рис. 8.11.

Вопросы, связанные с созданием в GUI графических изображений и окон, а также списков, с возможностью их прокрутки с помощью полос скроллинга, будут обсуждаться в следующем разделе.

8.2. Создание графического окна с элементами управления и объектами axes

Для иллюстрации действий необходимых для создания GUI, содержащего элементы управления и объект axes в пределах одного графического окна, создадим интерфейс пользователя для нахождения решений дифференциальных уравнений первого порядка методом Эйлера. Рассмотрим листинг соответствующей m-функции, позволяющей автоматизировать процедуру вычислений.

```
% листинг файла MyGUI
function MyGUI
% Задание глобальных пременных
global hFig1 hAxes
global hEdit1 hEdit2 hEdit3 hEdit4
global hList
global hButton1 hButton2
global hText1 hText2 hText3 hText3 hText5 hText6 hText7


% Создание графического окна GUI
hFig1 = figure('Position',[50 50 570 450],.
        'Resize','off'); % запрет изменения размеров окна GUI


% Вставка в окно GUI точечного рисунка
set(hFig1, 'Units'  'pixels');
```

```
Pic = imread([matlabroot filesep 'toolbox\control\ctrldemos'
                                                filesep 'b747.jpg']);
info = imfinfo([matlabroot filesep 'toolbox\control\ctrldemos'..
                                                filesep 'b747.jpg']);
hA=axes('Parent',hFig1);
image(Pic);
set(hA,...
   'Visible', 'off',
   'Units', 'pixels',...
   'Position', [5 445-info.Height/6 info.Width/6 info.Height/6]);


% Создание текстовых окон для ввода расчетных параметров
hEdit1 = uicontrol(hFig1,'Style','edit', .
                         'BackgroundColor',[1 1 1],...
                         'Position',[410 300 150 25],.
                         'HorizontalAlignment','left');
hEdit2 = uicontrol(hFig1,'Style','edit',
                         'BackgroundColor',[1 1 1],.
                         'Position',[410 230 150 25],.
                         'HorizontalAlignment','left');
hEdit3 = uicontrol(hFig1,'Style','edit',...
                         'BackgroundColor',[1 1 1],
                         'Position',[410 160 150 25],
                         'HorizontalAlignment','left');
hEdit4 = uicontrol(hFig1,'Style','edit',
                         'BackgroundColor',[1 1 1],
                         'Position',[410 105 150 25],
                         'HorizontalAlignment','left');


% Создание прокручивающегося списка
hList = uicontrol(hFig1,'Style','listbox',
                        'Position',[410 10 150 70],
                        'String',{'sin','cos','MyFunc1',
                        'MyFunc2','F3'},
                        'HorizontalAlignment','left');


% Создание командных кнопок
hButton1=uicontrol(hFig1,'Style','pushbutton',. .
                         'Position', [40 15 100 30],
                         'String','Вычислить',.
                         'FontName','Time New Roman');
hButton2=uicontrol(hFig1,'Style','pushbutton',
                         'Position', [150 15 100 30],
                         'String','Очистить окно', ..
                         'FontName','Time New Roman');
hButton3=uicontrol(hFig1,'Style','pushbutton',
                         'Position', [260 15 100 30],
                         'String','Выход',.
                         'FontName','Time New Roman');
% Создание текстовых полей
hText1 = uicontrol(hFig1,'Style','text',
```

```
                              'BackgroundColor',[0.7 0.7 0.7],
                              'Position',[55 330 240 20],
                              'String','Решение ДУ методом Эйлера',...
                              'FontName','Time New Roman');
   hText2 = uicontrol(hFig1,'Style','text',
                              'BackgroundColor',[0.7 0.7 0.7],
                              'Position',[410 330 150 30],.
                              'String','Начало отрезка интегрирования',...
                              'HorizontalAlignment','left',...
                              'FontName','Time New Roman');
   hText3 = uicontrol(hFig1,'Style','text',..
                              'BackgroundColor',[0.7 0.7 0.7],...
                              'Position',[410 260 150 30],.
                              'String','Конец отрезка интегрирования',...
                             'HorizontalAlignment','left',.
                              'FontName','Time New Roman');
   hText4 = uicontrol(hFig1,'Style','text',
                              'BackgroundColor',[0.7 0.7 0.7],...
                              'Position',[410 190 150 30],.
                              'String','Начальное условие',...
                              'HorizontalAlignment','left',
                              'FontName','Time New Roman');
   hText5 = uicontrol(hFig1,'Style','text',
                              'BackgroundColor',[0.7 0.7 0.7],.
                              'Position',[410 135 150 20],.
                              'String','Число узлов сетки',...
                              'HorizontalAlignment','left',
                              'FontName','Time New Roman');
   hText6 = uicontrol(hFig1,'Style','text',..
                              'BackgroundColor',[0.7 0.7 0.7],...
                              'Position',[410 80 150 20],..
                              'String','Выбор функции',...
                              'HorizontalAlignment','left',
                              'FontName','Time New Roman');


% Создание графического окна для визуализации решения ДУ
hAxes=axes('Parent',hFig1,'Color',[1 1 1],
           'Units','pixel',
           'Position',[40 70 330 250],
           'Fontsize',10);
```

Результат выполнения данной функции представлен на рис. 8.12.

Конструирование интерфейса завершается установлением связи конкретные элементов управления с соответствующими исполняемыми функциями. Рассмотрению данного вопроса мы посвящаем следующий раздел.

8.3. Обработчики событий

Функции, связывающие элементы управления с исполняемыми m-функциями, называют «Callback-функциями». Данные функции вызываются непосредственно MATLAB без участия пользователя.

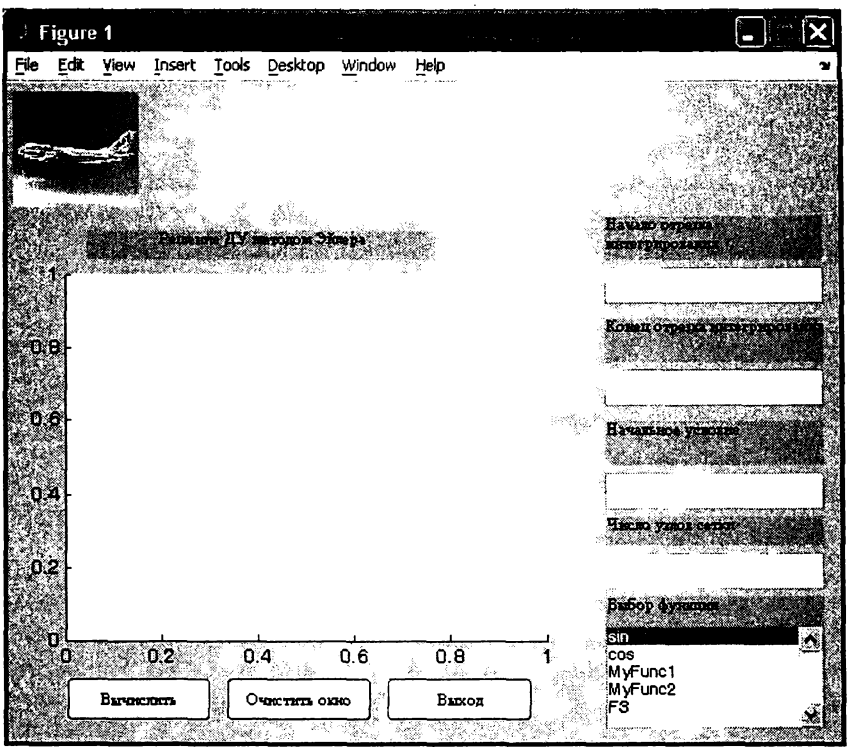


Рис. 8.12. Разработанный графический интерфейс пользователя

Для того, чтобы связать командные кнопки с callback-функциями следует при их создании указать в качестве значения поля **Callback** имя соответствующей m-функции. Это означает, что в приведенном выше листинге функции **MyGUI** следует изменить строки, в которых создаются управляющие кнопки:

```
hButton1=uicontrol(hFig1,'Style','pushbutton',. .
                         'Position', [40 15 100 30],...
                         'String','Вычислить',...
                         'Callback','MyCalc',...
                         'FontName','Time New Roman'
hButton2=uicontrol(hFig1,'Style','pushbutton',...
                         'Position', [150 15 100 30],.
                         'String','Очистить окно',..
                         'Callback','MyClear',...
                         'FontName','Time New Roman'
hButton3=uicontrol(hFig1,'Style','pushbutton',...
                         'Position', [260 15 100 30],...
                         'String','Выход',..
                         'Callback','MyExit',.
                         'FontName','Time New Roman'
```

Обсуждение начнем с наиболее простых m-функций **MyClear** и **MyExit**, листинги которых приведены ниже.

```
% листинг файла MyClear.m
function MyClear
global hAxes
axes(hAxes);
cla;


% листинг файла My Close.m
function MyClose
close;
```

В функции **MyClear** вызов функции **cla** обеспечивает стирание содержания объекта **axes** с описателем **hAxes**, оставляя без изменения объект **Pic**.

Перейдем к рассмотрению функции **MyCalc**, листинг которой приведен ниже.

```
% листинг файла MyCalc.m
function MyCalc
global hAxes hEditl hEdit2 hEdit3 hEdit4 hList


% Считывание параметров вычислений
str1=get(hEditl,'String');
str2=get(hEdit2,'String');
str3=get(hEdit3,'String');
str4=get(hEdit4,'String');
% Преобразование строковых в переменных в числовые
x0=str2num(str1);
xf=str2num(str2);
y0=str2num(str3);
Np=str2num(str4);
% Получение имени функции из списка
index=get(hList,'Value'); % получение номера выбранного
                          % элемента списка
cellArr=get(hList,'String'); % получение таблицы, содержащей
                             % все элементы списка
funName=cellArr{index};   получение элемента с заданным номером
                          реализация алгоритма Эйлера
dx=(xf-x0)/Np;
X=x0;
Y=y0;
for k=1:Np
 Y=[Y Y(end)+feval(funName,X(end))*dx];
 X=[X X(end)+dx];
end;
% Построение графика
axes(hAxes);
plot(X,Y);
```

Здесь с помощью функции **get**, в качестве аргументов, которой передаются соответствующие указатели текстовых полей, содержащие необходимую информацию для проведения вычислений. Далее полученные текстовые значения преобразуются в числовые значения.

Для получения имени функции, представляющую правую часть ДУ, сначала запрашивается номер выбранной функции из списка (значение свойства **Value**). Затем получаем весь список (значение свойства **String**), и далее в соответствии с выбранным номером получаем собственно имя функции.

В блоке, реализующем алгоритм Эйлера, для вычисления числового значения функции используется функция **feval**, осуществляющая вызов функции с именем **funName**, полученного из списка, и передает ей в качестве аргумент значение **X(end)**. Далее осуществляется визуализация полученного решения.

Для проверки работы созданного приложения можно использовать функции, листинги которых приведены ниже.

```
% листинг функции f3.m
function z=f3(x);
z=sin(x.^2);
```

```
% листинг функции MyFunc1.m
function z=MyFunc1(x);
z=x.^3;
```

```
% листинг функции MyFunc2.m
function z=MyFunc2(x);
z=1./sin(x);
```

В качестве примера на рис. 8.13 приведено решение ДУ первого порядка с функцией $f(x)=\sin(x^2)$ в правой части.

8.4. Средства визуального программирования интерфейса пользователя

Необходимо отметить, что разработка графического интерфейса является достаточно трудоемкой задачей, которая может быть решена с использованием функций графического ввода, а также специальной функцией **uicontrol**, разработанной для интерактивного взаимодействия пользователя с рисунком. Однако для упрощения процедуры создания GUI в MATLAB, как и во всех современных программных средствах, основанных на объектно-ориентированной парадигме программирования, существует возможность визуального проектирования интерфейса, называемая технологией визуального программирования.

Рассмотрим основные этапы визуального создания GUI на примере расчета и визуализации траектории движения, являющегося суммой двух круговых движений. Напомним, что координаты радиус-вектора матери-

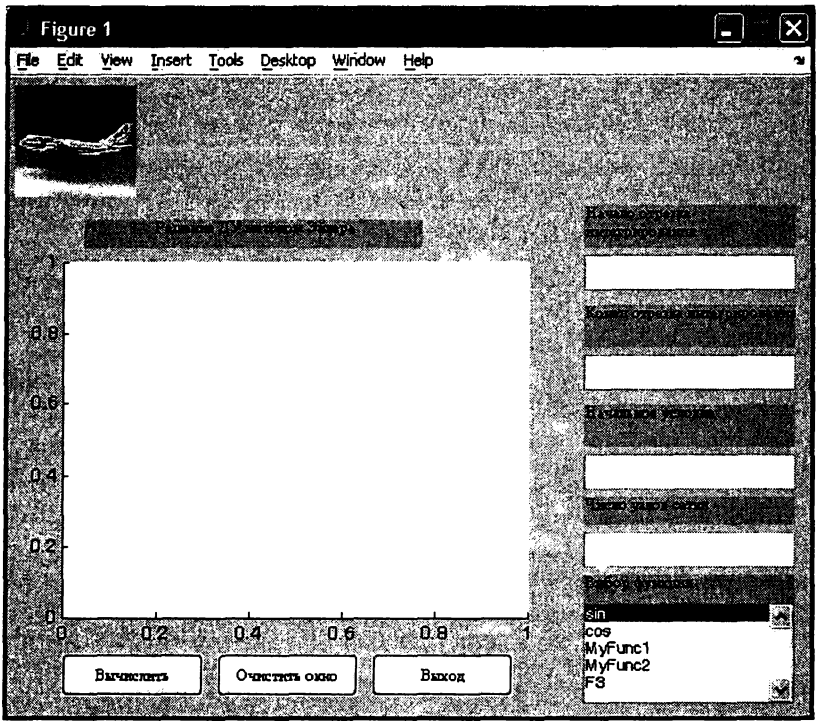


Рис. 8.13. Численное решение задачи Коши ДУ $\frac{dy}{dx} = \sin(x^2)$ с начальным условием $y(0) = 0.3$

альной точки, совершающее движение вокруг центра, движущегося в свою очередь по окружности описывается выражением

$$\vec{r}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} R_2 \cos\left(\frac{2\pi}{T_2} t\right) + R_1 \cos\left(\frac{2\pi}{T_1} t\right) \\ R_2 \sin\left(\frac{2\pi}{T_2} t\right) + R_1 \sin\left(\frac{2\pi}{T_1} t\right) \end{pmatrix},$$

где R_1 — радиус окружности, по которой движется материальная точка, T_1 — период обращения материальной точки; R_2 — радиус окружности, по которой движется центр; T_2 — период обращения центра.

Таким образом, для решения поставленной задачи следует создать интерфейс, позволяющий вводить следующие величины: R1, R2, T1, T2, Tfin (правая граница временного интервала, на котором вычисляются кинематические характеристики), N (число точек, в которых проводится вычисление значений кинематических характеристик материальной точки.

Для создания графического интерфейса (GUI — Graphics User Interface) с помощью средств визуального программирования необходимо запустить соответствующую программу командой **guide** или через меню: **File → New → GUI**. По истечению некоторого времени, определяемого быстродействием компьютера, на экране появится окно, представленное на рис. 8.14.

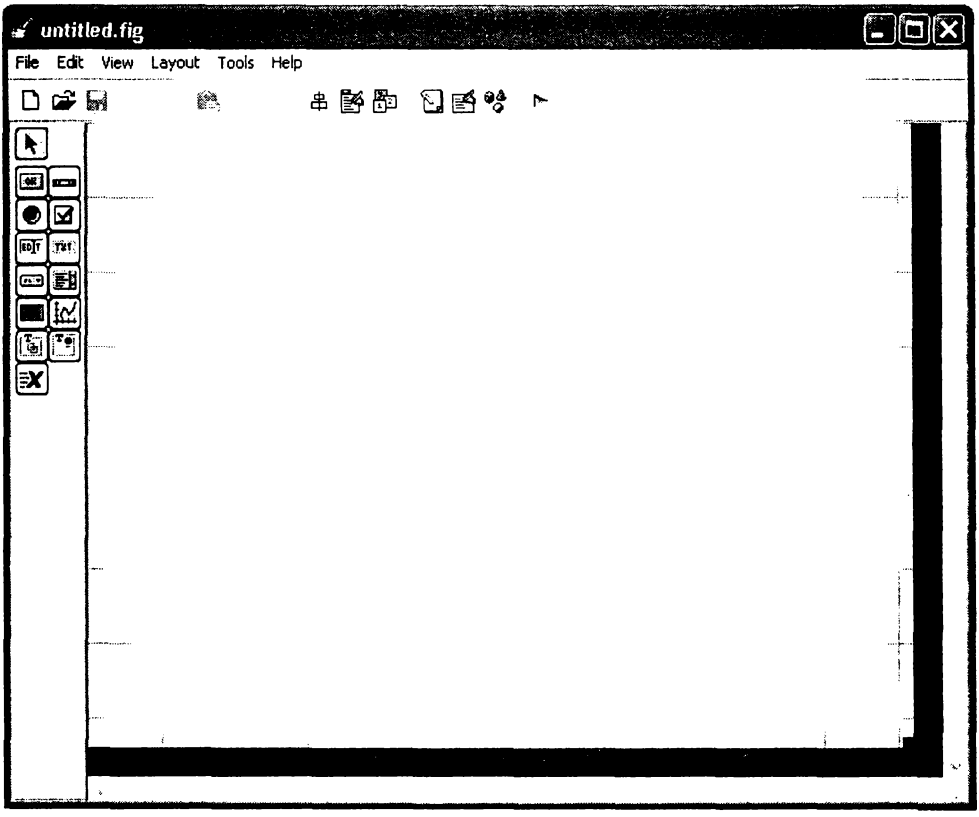


Рис. 8.14. Форма для визуального создания GUI

Для ввода названия формы открыть окно **Property Inspector**, нажав на кнопку или сделав двойной клик в любом месте поля формы, затем установить новое значение поля **Name** (рис. 8.14).

В левой части окна расположена панель управления, содержащая элементы интерфейса. Для размещения на панели формы окна, в котором будут выводится графики используется кнопка. Щелкнув по данному элементу на панели управления и переведя мышь на панель формы, необходимо поместить указатель мыши, имеющий форму креста, в ту точку, где будет находиться левый верхний угол окна. Нажав и удерживая левую кнопку мыши, необходимо вытянуть получающийся прямоугольник до нужных размеров. При необходимости размещения нескольких окон на панели формы повторяется описанная выше последовательность действий.

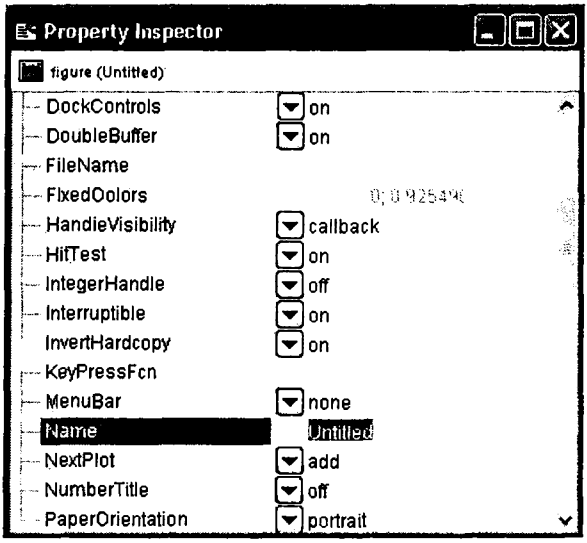


Рис. 8.15. Инспектор свойств графического окна

Надписи на панели формы создаются с помощью кнопки, которая переносится повторением описанной выше последовательности действий. После этого на панели формы появится надпись **Static Text**. Для изменения надписи необходимо выбрать надпись, щелкнув по ней левой кнопкой мыши, открыть окно **Property Inspector**, и ввести новый текст надписи в поле **String**. Для изменения цвета фона, на котором выводится надпись, следует установит новые значения поля **BackgroundColor**.

Для создания редактируемых окон ввода используется кнопка, которая переносится повторением описанной выше последовательности действий. Для изменения значения окна установить в окне **Property Inspector** новое значение в поле **String**.

Для создания и размещения кнопок на панели формы используется кнопка, которая переносится повторением описанной выше последовательности действий. Для изменения надписи на кнопке установить в окне **Property Inspector** новое значение в поле **String**.

Построенные таким образом кнопки, окна вывода и редактирования, окна статического текста, кнопки, а также другие объекты, размещенные на панели формы, можно выровнять и установить определенные промежутки между ними с помощью панели **Alignment Tools**. Для включения данной панели используется кнопка. Для задания ряда объектов, с которыми будут выполняться какие-либо действия, необходимо их выделить, щелкая по каждому из них при нажатой клавише **Shift**. Выделенные объекты отмечаются черными точками вокруг соответствующих объектов. При необходимости изменить размер, какого-либо объекта, размещенного на панели формы необходимо щелкнуть по данному объекту левой кнопкой мыши и изменить размер, также как меняется размер любого окна Windows.

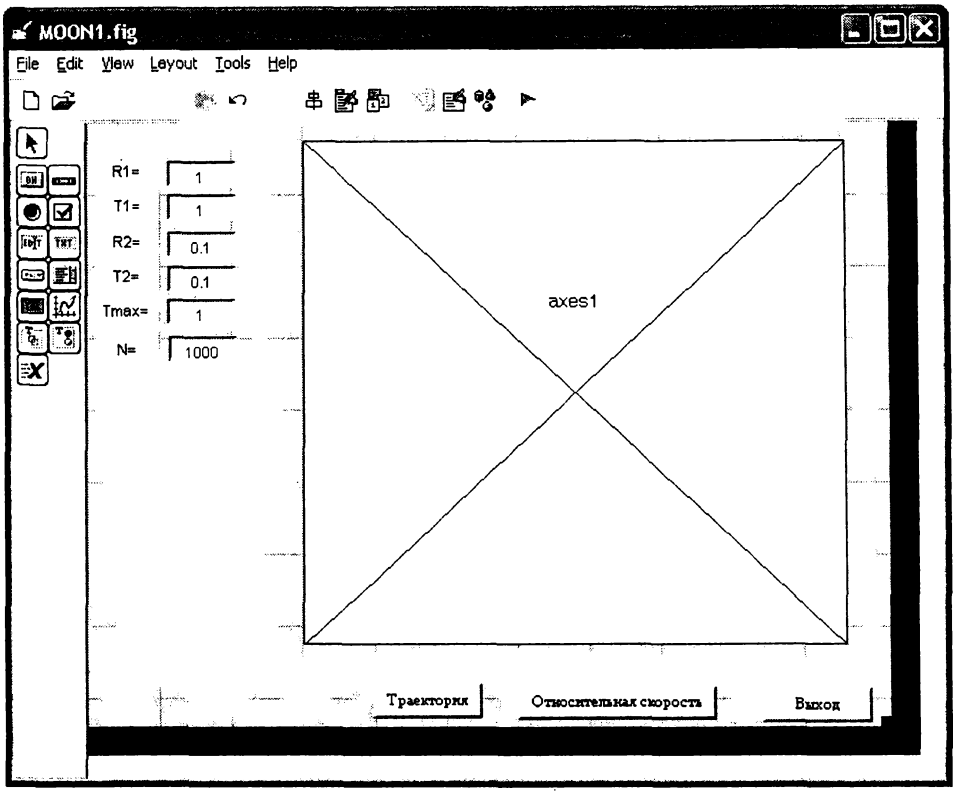


Рис. 8.16. Общий вид разрабатываемого GUI

Так как по умолчанию размер окна устанавливается в пикселях, это может привести к тому, что при изменении размеров окна и кнопки могут наехать друг на друга. Для устранения этого нежелательного явления следует задать единицы измерения размера всех элементов в безразмерных переменных — долях размера окна. Данный размер называется **normalized**. Для этого необходимо выделить с помощью удерживаемой клавиши **Ctrl** и левой кнопки мыши все объекты, размещенные на форме. Затем открыть окно **Property Inspector** и установить в поле **Units** значение **normalized**. Аналогично следует задать размер шрифтов (значения поля **Font Units**) и установить тип шрифта Times New Roman (поле **FontName**).

После разработки внешнего вида графического интерфейса необходимо сохранить этот файл на диске, вызвав стандартное окно Windows для сохранения файла: **File → Save**. После сохранения на диске появляются два файла: **NAME.m** и **NAME.fig** (где **NAME** — выбранное вами имя файла). Первый файл — это текст программы, реализующий разработанный интерфейс, второй — набор данных, необходимых для работы интерфейса. Для запуска программы на выполнение достаточно в командной строке MATLAB ввести имя вашего файла.

Проект графического интерфейса решения задачи, рассмотренной в разделе 1.2, представлен на рис. 8.16. Ниже приведен соответствующий программный код, сохраненный на диске под именем **Moon.m**.

```
function varargout = moon(varargin)
% MOON Application M-file for moon.fig
%    FIG = MOON launch moon GUI.
%    MOON('callback_name',    .) invoke the named callback.


% Last Modified by GUIDE v2.0 16-May-2002 23:30:06


if nargin == 0 % LAUNCH GUI


      fig = openfig(mfilename,'reuse');


      % Generate a structure of handles to pass to callbacks,
                                                    and store it.
      handles = guihandles(fig);
      guidata(fig, handles);


      if nargout > 0
            varargout{1} = fig;
      end


elseif ischar(varargin{1})    INVOKE NAMED SUBFUNCTION OR
                              CALLBACK


       try
         [varargout{1:nargout}] = feval(varargin{:}); % FEVAL
                                                      switchyard
       catch
         disp(lasterr);
       end


end


%| ABOUT CALLBACKS:
%| GUIDE automatically appends subfunction prototypes to this
%| file, and sets objects' callback properties to call them
%| through the FEVAL switchyard above. This comment describes
%| that mechanism.
```

```
%|
%| Each callback subfunction declaration has the following form:
%| <SUBFUNCTION_NAME>(H, EVENTDATA, HANDLES, VARARGIN)
%|
%| The subfunction name is composed using the object's Tag
%| and the callback type separated by '_', e.g.
%| 'slider2_Callback', 'figure1_CloseRequestFcn',
%| 'axis1_ButtondownFcn'
%|
%| H is the callback object's handle (obtained using GCBO)
%|
%| EVENTDATA is empty, but reserved for future use.
%|
%| HANDLES is a structure containing handles of components in
%| GUI using tags as fieldnames, e.g. handles.figure1,
%| handles.slider2. This structure is created at GUI startup
%|  stored in the figure's application data using GUIDATA. A
%| copy of the structure is passed to each callback.  You can
%| store additional information in this structure at GUI
%| startup, and you can change the structure during callbacks.
%| Call guidata(h, handles) after changing your copy to replace
%| the stored original so that subsequent callbacks see the
%| updates. Type "help guihandles" and "help guidata" for more
%| information.
%|
%| VARARGIN contains any extra arguments you have passed to the
%| callback. Specify the extra arguments by editing the callback
%| property in the inspector. By default, GUIDE sets
%| the property to: <MFILENAME>('<SUBFUNCTION_NAME>', gcbo, [],
%| guidata(gcbo))
%| Add any extra arguments after the last argument, before
%| the final  closing parenthesis.
% --------------------------------------------------------------------
function varargout = edit1_Callback(h, eventdata, handles,
                                                        varargin)
% Stub for Callback of the uicontrol handles.edit1.
disp('edit1 callback not implemented yet  ')


% --------------------------------------------------------------------
function varargout = edit2_Callback(h, eventdata, handles,
                                                        varargin)
% Stub for Callback of the uicontrol handles.edit2.
disp('edit2 callback not implemented yet  ')


% --------------------------------------------------------------------
function varargout = edit3_Callback(h, eventdata, handles,
                                                        varargin)
% Stub for Callback of the uicontrol handles.edit3.
disp('edit3 callback not implemented yet  ')
```

```
% --------------------------------------------------------------------
function varargout = edit4_Callback(h, eventdata, handles,
                                                          varargin)
% Stub for Callback of the uicontrol handles.edit4.
disp('edit4 callback not implemented yet  ')

% --------------------------------------------------------------------
function varargout = edit5_Callback(h, eventdata, handles,
                                                          varargin)
% Stub for Callback of the uicontrol handles.edit5.
disp('edit5 callback not implemented yet  ')


% --------------------------------------------------------------------
function varargout = edit6_Callback(h, eventdata, handles,
                                                          varargin)
% Stub for Callback of the uicontrol handles.edit6.
disp('edit6 callback not implemented yet  ')


% --------------------------------------------------------------------
function varargout = pushbutton1_Callback(h, eventdata, handles,
                                                          varargin)
% Stub for Callback of the uicontrol handles.pushbutton1.
disp('pushbutton1 callback not implemented yet  ')


% --------------------------------------------------------------------
function varargout = pushbutton2_Callback(h, eventdata, handles,
                                                          varargin)
% Stub for Callback of the uicontrol handles.pushbutton2.
disp('pushbutton2 callback not implemented yet  ')


% --------------------------------------------------------------------
function varargout = pushbutton3_Callback(h, eventdata, handles,
                                                          varargin)
% Stub for Callback of the uicontrol handles.pushbutton2.
disp('pushbutton3 callback not implemented yet  ')
```

Созданный GUI, текст которого приведен выше, не обладает никакой функциональностью — он обладает только стандартными функциями Windows — менять размеры окна и закрывать окно и выводить сообщения о некоторых событиях (команды **disp**). Событие с точки зрения интерфейса происходит при нажатии на левую кнопку мыши. Сообщения, выводимые командой **disp** в командное окно MATLAB, призваны помочь разработчику интерфейса установить взаимно однозначное соответствие между элементом управления, размещенного на панели формы, и именем соответствующей ему функции.

Основным средством взаимодействия графического интерфейса с функциями, выполняющими требуемые действия, является задание значений полей, определяющих свойства кнопок и редактируемых окон, называемое

Callback. Значение (тип строка), присеваемое данному полю данному полю, есть имя функции, вызываемой при активации соответствующего объекта. Обратите внимание на тот факт, что в программном коде, сгенерированном редактором GUI, название каждой функции состоит из двух слов: названия того элемента, который вы разместили на панели формы и названия средства взаимодействия **Callback**.

Дальнейшая доработка интерфейса состоит в редактировании соответствующих функций в тексте исходного файла, осуществляемых в текстовом редакторе пакета MATLAB. Так как перед выполнением каждого вычисления программа должна получить значения переменных **R1**, **T1**, **R2**, **T2**, **Tfin**, **N**, представляется целесообразным создать специальную функцию, считывающую значения, соответствующих полей (строковые переменные), и преобразующую их в числовые значения, текст которой приводится ниже:

```
function read_data(handles)
global R1 T1 R2 T2 Tfin N % задание глобальных переменных
R1 = str2double(get(handles.edit1,'String')); % преобразование
                                    % значения поля edit1 в число
T1= str2double(get(handles.edit2,'String')); % преобразование
                                    % значения поля edit2 в число
R2 = str2double(get(handles.edit3,'String')); % преобразование
                                    % значения поля edit3 в число
T2 = str2double(get(handles.edit4,'String')); % преобразование
                                    % значения поля edit4 в число
Tfin = str2double(get(handles.edit5,'String')); % преобразование
                                    % значения поля edit5 в число
N = str2double(get(handles.edit6,'String')); % преобразование
                                    % значения поля edit6 в число
```

Кнопка **Выход** становится функционирующей после добавления в функцию **pushbutton1_Callback** команды **close**:

```
function varargout = pushbutton1_Callback(h, eventdata, handles,
                                                        varargin)
% Stub for Callback of the uicontrol handles.pushbutton1.
close;
```

Для проведения двух видов расчетов: 1) вычисления и визуализации траектории, 2) вычисления мгновенных значений скорости движения материальной точки относительно неподвижной системы координат. Выполнение первого вида расчетов при проектировании интерфейса мы связали с кнопкой **Траектория** (функция **pushbutton1_Callback**). Следовательно, необходимо дополнить функцию **pushbutton1_Callback** соответствующей последовательностью команд:

```
function varargout = pushbutton2_Callback(h, eventdata, handles,
                                                        varargin)
% Stub for Callback of the uicontrol handles.pushbutton2.
global R1 T1 R2 T2 Tfin N % список глобальных переменных
                          % задается в теле каждой функции
read_data(handles);    получение значений переменных R1, T1, R2,
                       T2, Tfin, N блок вычислений и построения
                       траектории
```

```
t=0:Tfin/N:Tfin;
Xz=R1*cos(2*pi*t/T1);
Yz=R1*sin(2*pi*t/T1);
Xm=R2*cos(2*pi*t/T2)
Ym=R2*sin(2*pi*t/T2);
Xotn=Xz+Xm;
Yotn=Yz+Ym;
plot(Xz,Yz,Xotn,Yotn);
```

Выполнение второго вида расчетов при проектировании интерфейса мы связали с кнопкой **Относительная скорость** (функция **pushbutton2_Callback**). Следовательно, необходимо дополнить функцию **pushbutton2_Callback** соответствующей последовательностью команд:

```
function varargout = pushbutton3_Callback(h, eventdata, handles,
                                                    varargin)
% Stub for Callback of the uicontrol handles.pushbutton3.
global R1 T1 R2 T2 Tfin N % список глобальных переменных
                          задается в теле каждой функции
read_data(handles)     получение значений переменных R1, T1, R2,
                       T2, Tfin, N блок вычислений и построения
                       мгновенных значений относительной скорости
dt=Tfin/N;
t=0:dt:Tfin;
Xz=R1*cos(2*pi*t/T1);
Yz=R1*sin(2*pi*t/T1);
Xm=R2*cos(2*pi*t/T2);
Ym=R2*sin(2*pi*t/T2)
Vx=diff(Xz)/dt;
Vy=diff(Yz)/dt;
vx=diff(Xm)/dt;
vy=diff(Ym)/dt;
V=(Vx.^2+Vy.^2) ^0.5-(Vx.*vx+Vy.*vy)./(Vx.^2+Vy.^2) ^0.5;
t1=0:dt:Tfin-dt;
plot(t1,V);
```

Состояние интерфейса, после нажатия на кнопки **Траектория** и **Относительная скорость**, показано на рис. 8.17, 8.18, соответственно.

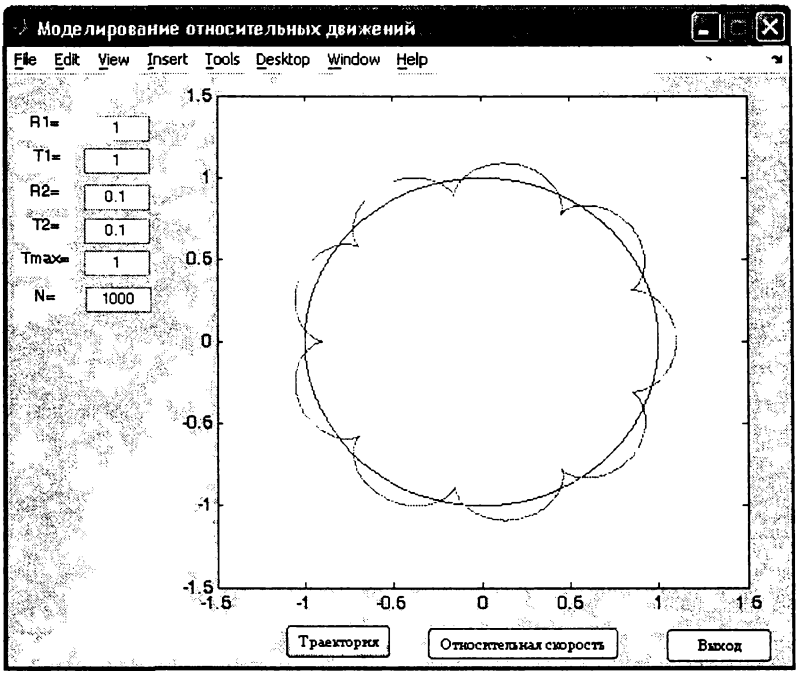


Рис. 8.17. Состояние GUI после нажатия на кнопку **Траектория**

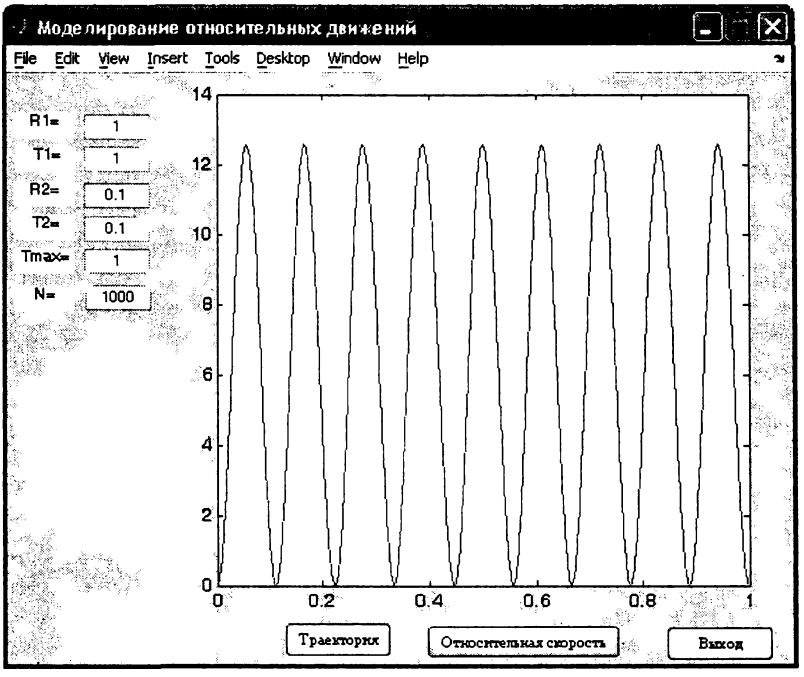


Рис. 8.18. Состояние GUI после нажатия на кнопку **Относительная скорость**

Вопросы для самопроверки

1. Что называется интерфейсом пользователя?
2. Назовите основные типы управляющих элементов интерфейса пользователя.
3. Назовите основные технологии создания GUI.
4. Как создаются элементы управления при создании GUI с использованием технологии программирования интерфейса?
5. Как создается GUI при использовании технологии визуального проектирования?
6. Что такое «обработчик событий»?
7. Как устанавливаются значения основных полей объектов управления?

Обработка экспериментальных данных в MATLAB

В данной главе обсуждаются основы анализа данных с помощью некоторых элементарных средств статистического инструментария MATLAB, в том числе: оценки статистических характеристик, решение задач аппроксимации.

9.1. Стандартные функции анализа данных

Для заданной последовательности данных, размещенных в вектор-строке ($D = [d_1, d_2, \ldots, d_n]$) вектор-столбце или матрице MATLAB можно вычислить следующие статистические характеристики (статистики): максимум, минимум, среднее, медиану, сумму элементов, произведение элементов, кумулятивную сумму стандартное отклонение (вычисляемое по формуле $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(d_i - \bar{d}\right)^2$), а также осуществлять упорядочивание массива в порядке возрастания его элементов.

Результаты, возвращаемые соответствующими функциями MATLAB для для вектор строки

```
>> D = [3 1 10 6 4];
```

представлены в табл. 9.1.

Таблица 9.1. Вычисление статистик для вектора D

Описание	Команда MATLAB	Результат
Минимум	**min(D)**	1
Максимум	**max(D)**	10
Среднее	**mean(D)**	4.8
Медиана	**median(D)**	6
Произведение элементов	**prod(D)**	720
Сумма элементов	**sum(D)**	24
Кумулятивная сумма	**cumsum(D)**	[3 4 14 20 24]
Стандартное отклонение	**std(D)**	3.4205
Упорядочивание по возрастанию	**sort(D)**	[1 3 4 6 10]
Упорядочивание по убыванию	**−sort(−D)**	[10 6 4 3 1]

Результаты применения функций, из табл. 9.1, к матрице матрицы **D**

```
>> D=[3 1 10 6 4; 5 6 7 11 1]
D =
   3     1    10     6     4
   5     6     7    11     1
>>
```

представлены в табл. 9.2.

Таблица 9.2. Вычисление статистик для матрицы D

Описание	Команда MATLAB	Результат
Минимум	**min(D)**	[3 1 7 6 1]
Максимум	**max(D)**	[5 6 10 11 4]
Среднее	**mean(D)**	[4.0000 3.5000 8.5000 8.5000 2.5000]
Медиана	**median(D)**	[4.0000 3.5000 8.5000 8.5000 2.5000]
Произведение элементов	**prod(D)**	[15 6 70 66 4]
Сумма элементов	**sum(D)**	[8 7 17 17 5]
Кумулятивная сумма	**cumsum(D)**	[3 1 10 6 4; 8 7 17 17 5]
Стандартное отклонение	**std(D)**	[1.4142 3.5355 2.1213 3.5355 2.1213]
Упорядочивание по возрастанию	**sort(D)**	[3 1 7 6 1; 5 6 10 11 4]
Упорядочивание по убыванию	**−sort(−D)**	[5 6 10 11 4; 3 1 7 6 1]

Одним из эффективных средств анализа случайных последовательностей является гистограмма, показывающая сколько членов случайной последовательности попало в заданные интервалы. Для вычисления гистограммы в MATLAB используется функция **hist( )**. Ниже приводится при-

мер использования данной функции для анализа случайной последовательности с нормальным законом распределения, среднее значение которой равняется 5, а стандартное отклонение 3.

```
>> r=5+3*randn(1000,1); % создание случайной последовательности
>> hist(r); % посторонние гистограммы, состоящей из 10
            % прямоугольников
```

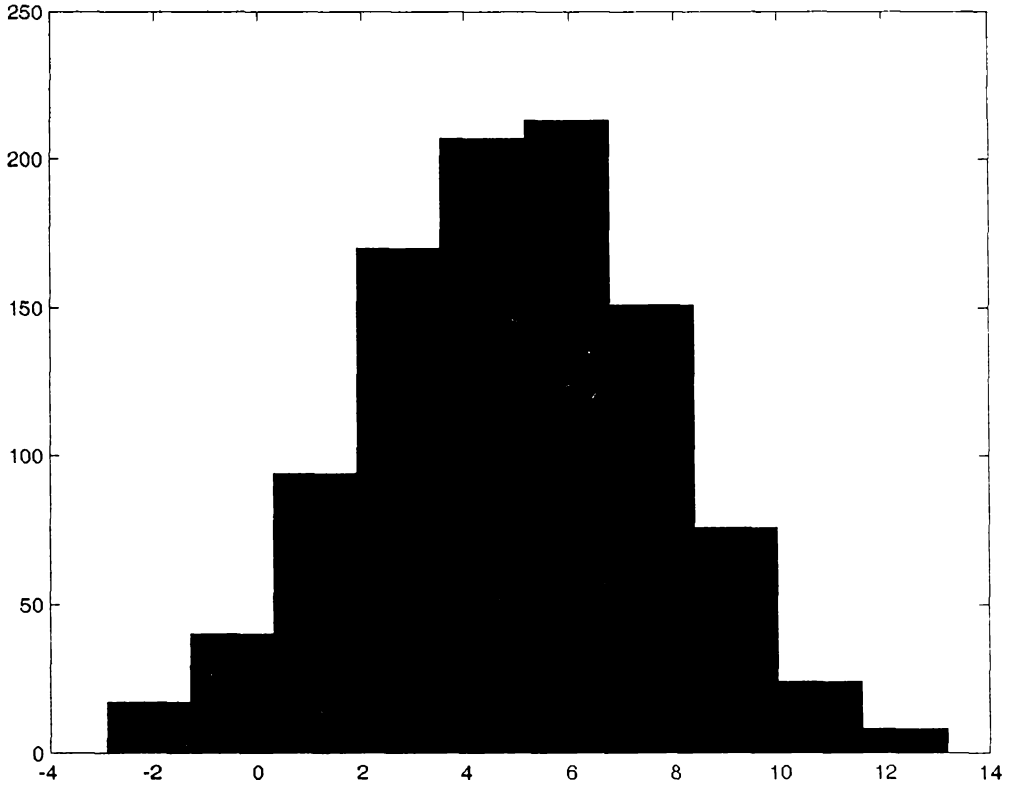


Рис. 9.1. Гистограмма случайной последовательности (10 инервалов)

Из рис. 9.1 видно, что по умолчанию отрезок $\min(r) \le x \le \max(r)$ разбивается на 10 равных интервалов. Для изменения разбиения отрезка $\min(r) \le x \le \max(r)$ нужно обратиться к функции **hist( )**, указав два аргумента:

```
>> hist(r,10) % рис. 9.2
```

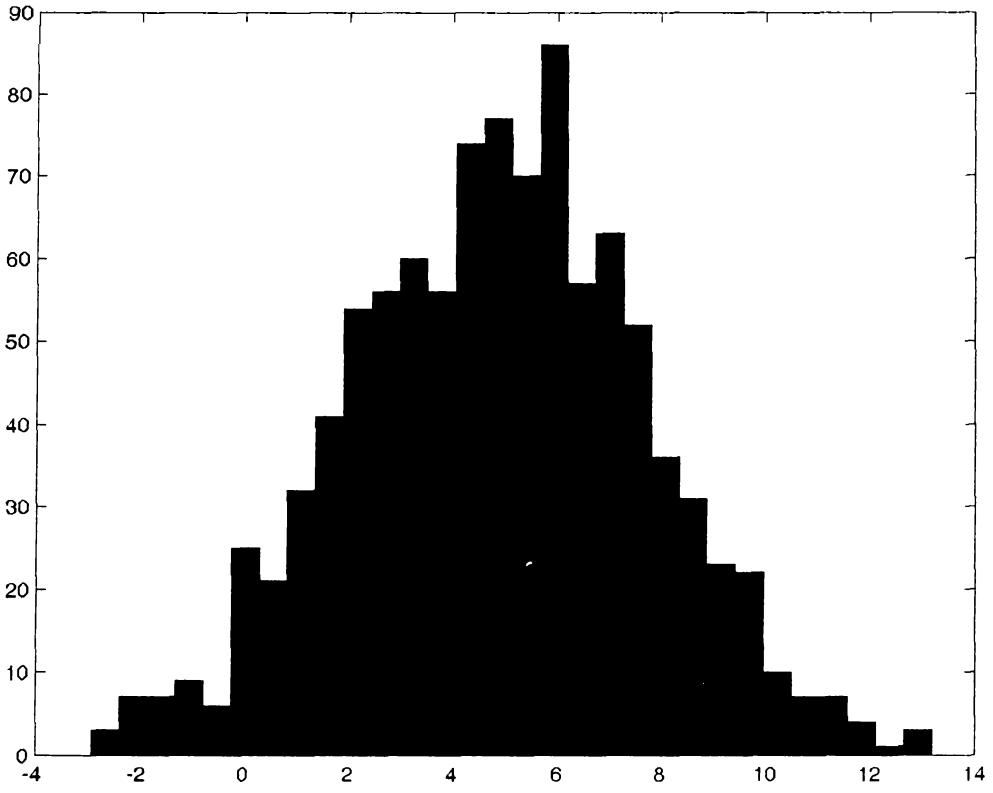


Рис. 9.2. Гистограмма случайной последовательности (30 инервалов)

Для получения численных значений числа членов последовательности и середин соответствующих интервалов следует выполнить команду

```
>> [y x] = hist(r,30);
```

которая возвращает в вектор **y** высоты прямоугольников, в вектор **x** — координаты середин высот прямоугольников.

9.2. Общая постановка метода наименьших квадратов

Пусть в результате измерений в процессе опыта получена таблица некоторой зависимости (табл. 9.1).

Таблица 9.3. Исходные данные для решения задачи аппроксимации методом наименьших квадратов

x	x_1	x_2	...	x_n
$f(x)$	y_1	y_2	...	y_n

Требуется найти формулу, выражающую данную зависимость аналитически.

Один из подходов к решению данной задачи состоит в построении интерполяционного многочлена, значения которого будут в точка x_1, x_2, ..., x_n совпадать с соответствующими значениями $f(x)$ из табл. 9.3. Однако совпадение значений в узлах может вовсе не означать совпадения характеров исходной и интерполирующей функций. Требование неукоснительного совпадения значений, тем более неоправданно, если значения функций $f(x)$ известны с некоторой погрешностью (рис. 9.3).

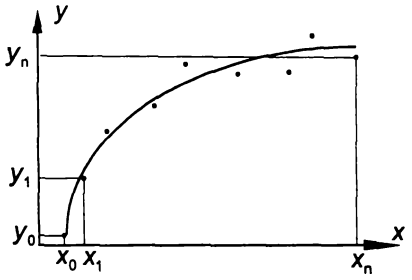


Рис. 9.3. К объяснению общей постановки метода наименьших квадратов

Поставим задачу так, чтобы с самого начала обязательно учитывался характер исходной функции: найти функцию заданного вида

$$y = F(x), \tag{9.1}$$

которая в точках x_1, x_2, ..., x_n принимает значения как можно более близкие к табличным значениям y_1, y_2, ..., y_n.

Следует отметить, что строгая функциональная зависимость для табл. 9.3. наблюдается редко, т.к. каждая из входящих в нее величин может зависеть от многих случайных факторов, поэтому обычно используют простые по виду аналитические функции.

Рассмотрим один из наиболее распространенных способов нахождения функции $F(x)$. Предположим, что приближающая функция $F(x)$ в точках x_1, x_2, ..., x_n имеет значения

$$\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_n. \tag{9.2}$$

Требование близости табличных значений y_1, y_2,..., y_n и значений (9.2) можно истолковать следующим образом. Будем рассматривать совокупность значений функции $f(x)$ из табл. 9.3 и совокупность значений (9.2), как координаты двух точек n-мерного пространства. С учетом этого задача приближения функции может быть переформулирована следующим образом: найти такую функцию $F(x)$ заданного вида, чтобы расстояние между точками $M(y_1,y_2,...,y_n)$ и $M(\overline{y}_1, \overline{y}_2, ..., \overline{y}_n)$ было наименьшим. Воспользовавшись метрикой Евклидова пространства, приходим к требованию, чтобы величина

$$\sqrt{(y_1 - \overline{y}_1)^2 + (y_2 - \overline{y}_2)^2 + ... + (y_n - \overline{y}_n)^2}, \qquad (9.3)$$

была наименьшей. Это равносильно следующему: сумма квадратов

$$(y_1 - \overline{y}_1)^2 + (y_2 - \overline{y}_2)^2 + ... + (y_n - \overline{y}_n)^2 \qquad (9.4)$$

должна быть наименьшей.

Окончательно задача приближения функции $f(x)$ теперь формулируется следующим образом: для функции $f(x)$, заданной табл. 9.1, найти функцию определенного вида так, чтобы сумма квадратов (9.4) была наименьшей. Эта задача называется приближением функции методом наименьших квадратов. В качестве приближающих функций в зависимости от характера точечного графика функции $f(x)$ часто используют функции, представленные в табл. 9.2. (Здесь a, b, m — неизвестные параметры.)

Таблица 9.4. Вид аппроксимирующих функций

$y = ax + b$	$y = \dfrac{1}{ax + b}$
$y = ax^2 + bx + c$	$y = a \ln x + b$
$y = ax^m$	$y = a\dfrac{1}{x} + b$
$y = ae^{mx}$	$y = \dfrac{x}{ax + b}$

Когда вид приближающей функции установлен, задача сводится к отысканию значений параметров.

Рассмотрим метод нахождения параметров приближающей функции в общем виде на примере приближающей функции, зависящей от трех параметров:

$$y = F(x, a, b, c). \qquad (9.5)$$

Имеем

$$F(x_i, a, b, c) = \overline{y}_i, \tag{9.6}$$

Сумма квадратов разностей соответствующих значений функций $f(x)$ и $F(x)$ имеет вид:

$$\sum_{i=1}^{n} \left(y_i - F(x_i, a, b, c)\right)^2 = \Phi(a, b, c). \tag{9.7}$$

Сумма является функцией $\Phi(a,b,c)$ трех переменных. Используя необходимое условие экстремума:

$$\frac{\partial \Phi}{\partial a} = 0, \frac{\partial \Phi}{\partial b} = 0, \frac{\partial \Phi}{\partial c} = 0,$$

получаем систему уравнений

$$\begin{aligned}
&\sum_{i=1}^{n} \left[y_i - F(x_i, a, b, c)\right] \cdot F'_a(x_i, a, b, c) = 0,\\
&\sum_{i=1}^{n} \left[y_i - F(x_i, a, b, c)\right] \cdot F'_b(x_i, a, b, c) = 0,\\
&\sum_{i=1}^{n} \left[y_i - F(x_i, a, b, c)\right] \cdot F'_c(x_i, a, b, c) = 0.
\end{aligned} \tag{9.8}$$

Решив систему (9.8) относительно параметров a, b, c, получаем конкретный вид функции $F(x,a,b,c)$. Изменение количества параметров не приведет к изменению сути самого подхода, а выразится в изменении количества уравнений в системе (9.8).

Значения разностей

$$y_i - F(x_i, a, b, c) = \varepsilon_i \tag{9.9}$$

называют отклонениями измеренных значений от вычисленных по формуле (9.5).

Сумма квадратов отклонений

$$\sigma = \sum_{i}^{n} \varepsilon_i^2 \tag{9.10}$$

в соответствие с принципом наименьших квадратов для заданного вида приближающей функции должна быть наименьшей.

Из двух разных приближений одной и той же табличной функции лучшим считается то, для которого (9.10) имеет наименьшее значение.

9.3. Нахождение приближающей функции в виде линейной функции и квадратичного трехчлена

Ищем приближающую функцию в виде:

$$F(x, a, b) = ax + b. \tag{9.11}$$

Находим частные производные

$$\frac{\partial F}{\partial a} = x, \frac{\partial F}{\partial b} = 1. \tag{9.12}$$

Составляем систему вида (9.8)

$$\sum(y_i - ax_i - b)x_i = 0,$$
$$\sum(y_i - ax_i - b) = 0.$$

(Здесь и далее сумма ведется по переменной $i = 1,2,\ldots,n$.)
Далее имеем

$$\begin{aligned} \sum x_i y_i - a\sum x_i^2 - b\sum x_i = 0, \\ \sum x_i y_i - a\sum x_i^2 - bn = 0. \end{aligned} \tag{9.13}$$

Разделив каждое уравнение (9.13) на n, получаем

$$\left(\frac{1}{n}\sum x_i^2\right)\cdot a + \left(\frac{1}{n}\sum x_i\right)\cdot b = \frac{1}{n}\sum x_i y_i,$$
$$\left(\frac{1}{n}\sum x_i^2\right)\cdot a + b = \frac{1}{n}\sum y_i.$$

Введем обозначения

$$\frac{1}{n}\sum x_i = M_x, \frac{1}{n}\sum y_i = M_y,$$
$$\frac{1}{n}\sum x_i y_i = M_{xy}, \frac{1}{n}\sum x_i^2 = M_{x^2}.$$

Тогда последняя система будет иметь вид

$$M_{x^2}\cdot a + M_x \cdot b = M_{xy},$$
$$M_x \quad a + b = M_y$$

или в матричной форме

$$\begin{pmatrix} M_{x^2} & M_x \\ M_x & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} M_{xy} \\ M_y \end{pmatrix}.$$

Откуда

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} M_{x^2} & M_x \\ M_x & 1 \end{pmatrix}^{-1}\begin{pmatrix} M_{xy} \\ M_y \end{pmatrix}. \tag{9.14}$$

Вычислив значения параметров a, b в соответствие с (9.14), получаем конкретные значения и, следовательно, конкретный вид линейной функции (9.11).

В случае нахождения приближающей функции в форме квадратного трехчлена имеем:

$$F(x, a, b, c) = ax^2 + bx + c. \tag{9.15}$$

Находим частные производные:

$$\frac{\partial F}{\partial a} = x^2, \frac{\partial F}{\partial b} = b, \frac{\partial F}{\partial c} = 1.$$

Составляем систему вида (9.8)

$$\begin{gathered} \sum\left(y_i - ax_i^2 - bx_i - c\right)x_i^2 = 0, \\ \sum\left(y_i - ax_i^2 - bx_i - c\right)x_i = 0, \\ \sum\left(y_i - ax_i^2 - bx_i - c\right) = 0. \end{gathered}$$

Далее имеем

$$\begin{gathered} \sum y_i x_i^2 - a\sum x_i^4 - b\sum x_i^3 - c\sum x_i^2 = 0, \\ \sum y_i x_i - a\sum x_i^3 - b\sum x_i^3 - c\sum x_i = 0, \\ \sum y_i - a\sum x_i^2 - b\sum x_i - cn = 0. \end{gathered}$$

Разделив каждое уравнение на n и перенеся члены, не содержащие неизвестные параметры в правую часть, получаем:

$$\begin{gathered} \left(\frac{1}{n}\sum x_i^4\right)a + \left(\frac{1}{n}\sum x_i^3\right)b + \left(\frac{1}{n}\sum x_i^2\right)c = \frac{1}{n}\sum y_i x_i^2, \\ \left(\frac{1}{n}\sum x_i^3\right)a + \left(\frac{1}{n}\sum x_i^2\right)b + \left(\frac{1}{n}\sum x_i\right)c = \frac{1}{n}\sum y_i x_i, \\ \left(\frac{1}{n}\sum x_i^2\right)a + \left(\frac{1}{n}\sum x_i\right)b + c = \frac{1}{n}\sum y_i. \end{gathered} \tag{9.16}$$

Решив систему (9.16) относительно неизвестных a, b, c, находим значения параметров приближающей функции.

Для нахождения решения задачи о нахождении линейного и квадратичного трехчленов в MATLAB необходимо выполнить следующую последовательность команд:

```
% задание исходных данных
>> N=10;
>> i=1:N;
>> Xmin=0;
>> Xmax=10;
>> x(i)=Xmin+(Xmax-Xmin)/(N-1)*(i-1);
>> y(i)=0.2*x(i); % точные значения функции
% задание шума, имеющего равномерный закон распределения на
% отрезке [b,a]
>> a=0.2
>> b=-0.1;
>> Yrnd=b+(a-b)*rand(N,1);
>> y1=y+Yrnd'  % создание зашумленных данных
>> plot(x,y,x,y1,'o'); % визуализация точной и зашумленной
                       % последовательностей (рис. 9.4)
```

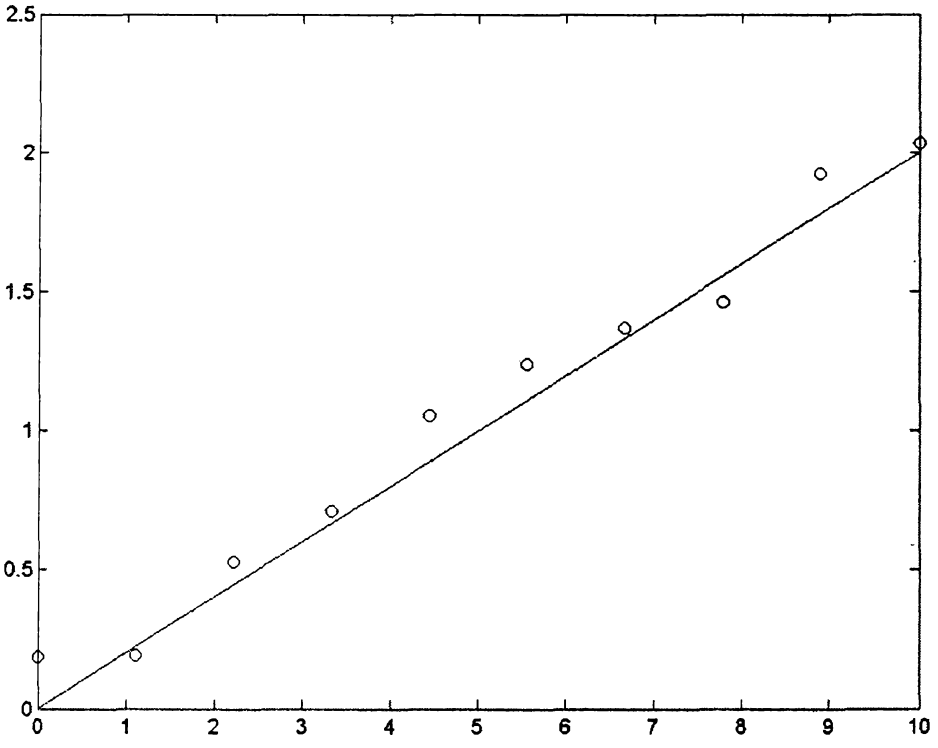


Рис. 9.4. Исходные данные и аппроксимирующая прямая

```
% вычисление элементов матрицы в (9.14)
>> tmp=x(i).^2;
>> M(1,1)=1/N*sum(tmp);
>> M(1,2)=1/N*sum(x);
>> M(2,1)=M(1,2);
>> M(2,2)=1;
% вычисление элементов вектора d
>> d(1,1)=1/N*dot(x,y1);
>> d(2,1)=1/N*sum(y1);
% решение системы линейных уравнений (9.14)
>> Coeff=A^-1*d;
Coeff =
  0.1941
  0.0993
>> F=inline('a*x+b','a','b','x'); % задание аппроксимирующей
                                  % функции
>> tmp1(i)=feval(F,Coeff(1,1),Coeff(2,1),x(i)); % вычисление
                          % значений аппроксимирующей функции
% вычисление суммы квадратов отклонений при линейной
% аппроксимации
>> tmp=tmp1-y1;
>> dot(tmp,tmp)
ans =
  0.0685
% аппроксимация исходных данных полиномом второй степени
% задание матрицы системы линейных уравнений в (9.16)
>> A(1,1)=1/N*sum(x.^4);
>> A(1,2)=1/N*sum(x.^3);
>> A(1,3)=1/N*sum(x.^2);
>> A(2,1)=A(1,2);
>> A(2,2)=A(1,3);
>> A(2,3)=1/N*sum(x);
>> A(3,1)=A(2,2);
>> A(3,2)=A(2,3);
>> A(3,3)=1;
% задание вектора столбца свободных членов
>> d(1,1)=1/N*dot(x.^2,y1);
>> d(2,1)=1/N*dot(x,y1);
>> d(3,1)=1/N*sum(y1);
% решение системы линейных уравнений (9.16)
>> Coeff=A^-1*d
Coeff =
  0.0004
  0.1904
  0.1049
>> F=inline('a*x.^2+b*x+c','a','b','c','x'); % задание
                                    % аппроксимирующей функции
% вычисление суммы квадратов отклонений при квадратичной
% интерполяции
>> tmp2(i)=feval(F,Coeff(1,1),Coeff(2,1),Coeff(3,1),x(i));
>> tmp=tmp2-y1;
dot(tmp,tmp)
ans =
  0.0687
```

Можно найти решение рассмотренной выше задачи регрессии в MATLAB другим способом. Для этого следует использовать тот факт, что коэффициенты искомой функции, минимизирующей сумму квадратов отклонений, являются решением переопределенной системы уравнений. Для случая интерполяции полиномом второй степени приведенных выше данных система уравнений имеет вид:

$$\begin{pmatrix} y1_1 \\ y1_2 \\ \\ y1_{10} \end{pmatrix} = \begin{pmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ & & \\ 1 & t_{10} & t_{10}^2 \end{pmatrix} \times \begin{pmatrix} Coeff_0 \\ Coeff_1 \\ Coeff_2 \end{pmatrix}.$$

Решение данной системы уравнений, удовлетворяющее методу наименьших квадратов, находится с помощью оператора \:

```
Coeff = A\y1,
```

где

$$y1 = \begin{pmatrix} y1_1 \\ y1_2 \\ \\ y1_{10} \end{pmatrix},$$

$$A = \begin{pmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ & & \\ 1 & t_{10} & t_{10}^2 \end{pmatrix}.$$

Таким образом, альтернативный подход к нахождению коэффициентов аппроксимирующего полинома реализуются выполнением следующей последовательности команд:

```
>> B=[ones(size(x')) x' x'.^2]
B =
   1.0000         0         0
   1.0000    1.1111    1.2346
   1.0000    2.2222    4.9383
   1.0000    3.3333   11.1111
   1.0000    4.4444   19.7531
   1.0000    5.5556   30.8642
   1.0000    6.6667   44.4444
   1.0000    7.7778   60.4938
   1.0000    8.8889   79.0123
   1.0000   10.0000  100.0000
>> Coeff=B\y1
ans =
   0.1049
```

```
0.1904
0.0004
```

9.4. Нахождение приближающей функции в виде других элементарных функций

Покажем, что задача нахождение приближающей функции, зависящей от двух параметров, может быть сведена к нахождению параметров линейной функции.

I. Степенная функция (геометрическая регрессия)

$$F(x, a, m) = ax^m \tag{9.17}$$

Предполагая, что в табл. 9.1 все значения аргумента и значения функции положительны, прологарифмируем (9.17), при условии, что a > 0:

$$\ln F = \ln a + m \ln x. \tag{9.18}$$

Так как функция F является приближающей для функции f, функция $\ln F$ будет приближающей для функции $\ln f$. Введем новую переменную $u = \ln x$, тогда, как следует из (9.18), $\ln F$ будет функцией от u: $\Phi(u)$.

Обозначим

$$m = A, \ln a = B, \tag{9.19}$$

тогда (9.18) принимает вид

$$\Phi(x, A, B) = Au + B, \tag{9.20}$$

т.е. задача свелась к отысканию приближающей функции в виде линейной функции.

На практике для нахождения приближающей функции в виде степенной функции используют следующий алгоритм:

1. составляют по заданной табл. 9.1. новую таблицу, прологарифмировав значения x, y в исходной таблице;
2. находят по новой таблице параметры A и B приближающей функции вида (9.20);
3. находят в соответствие с (9.19) значения параметров a, m.

II. Показательная функция

$$F(x, a, m) = ae^{mx}, a > 0 \tag{9.21}$$

Прологарифмировав равенство (9.21), получим

$$\Phi(x, A, B) = Ax + B. \tag{9.22}$$

Таким образом, задача сведена к предыдущей.

III. Дробно-линейная функция

$$F(x, a, b) = \frac{1}{ax + b} \tag{9.23}$$

Перепишем (9.23) следующим образом:

$$\frac{1}{F(x, a, b)} = ax + b.$$

Из последнего равенства следует, что надо составить новую таблицу, заменив в ней значения функции обратными числами. Используя новую таблицу, найти a, b и подставить найденные значения параметров в (9.23).

IV. Логарифмическая функция

$$F(x, a, b) = a \cdot \ln x + b. \tag{9.24}$$

Из (9.24) видно, что для перехода к линейной функции достаточно сделать подстановку $\ln x = u$. Следовательно, для нахождения значений a, b нужно прологарифмировать значения аргумента в исходной таблице (табл. 9.1) и, рассматривая полученные значения в совокупности с исходными значениями функции, найти для новой таблицы приближающую функцию в виде линейной. Коэффициенты a, b найденной функции подставить в (9.24).

V. Гипербола

$$F(x, a, b) = \frac{a}{x} + b. \tag{9.25}$$

Для перехода к линейной функции достаточно сделать подстановку $u=1/x$:

$$F(u, a, b) = au + b. \tag{9.26}$$

Далее заменить значения аргумента обратными числами и найти для новой таблицы приближающую функцию линейного вида (9.26). Полученные значения параметров подставить в (9.25).

VI. Дробно-рациональная функция

$$F(x, a, b) = \frac{x}{ax + b}. \tag{9.27}$$

Перепишем (9.27) следующим образом:

$$\frac{1}{F(x,a,b)} = a + \frac{b}{x}. \tag{9.28}$$

Из (9.28) видно, что задача сводится к задаче, рассмотренной в предыдущем пункте. Действительно, если в исходной таблице заменить значения x, y их обратными величинами по формулам $z = 1/x$, $u = 1/x$ и искать для новой таблицы приближающую функцию вида $u = bz + a$, то найденные значения будут искомыми для функции (9.27).

9.5. Аппроксимация линейной комбинацией функций

Будем искать аппроксимирующую функцию $F(x)$ в виде линейной комбинации известных функций $f_1(x)$, $f_2(x)$, $f_3(x)$:

$$F(x) = a \cdot f_1(x) + b \cdot f_2(x) + c \cdot f_3(x). \tag{9.29}$$

Подставив (9.29) в (9.8) и выполнив очевидные алгебраические преобразования, получаем систему линейных уравнений, относительно неизвестных коэффициентов a, b, c:

$$\begin{pmatrix} \sum f_1(x_i)^2 & \sum f_1(x_i)f_2(x_i) & \sum f_1(x_i)f_3(x_i) \\ \sum f_1(x_i)f_2(x_i) & \sum f_2(x_i)^2 & \sum f_2(x_i)f_3(x_i) \\ \sum f_1(x_i)f_3(x_i) & \sum f_2(x_i)f_3(x_i) & \sum f_2(x_i)^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sum y_i f_1(x_i) \\ \sum y_i f_2(x_i) \\ \sum y_i f_3(x_i) \end{pmatrix} \tag{9.30}$$

Решив систему линейных уравнений (9.30) любым известным способом, находим значение коэффициентов a, b, c и тем самым аналитическое выражение для аппроксимирующей функции.

Для решения системы линейных уравнений метода наименьших квадратов при использовании аппроксимации линейной комбинацией известных функций в MATLAB можно использовать метод, описанный в предыдущем разделе. Ниже представлена последовательность команд, позволяющая найти коэффициенты аппроксимирующей функции вида $F(x,a,b,c) = a \quad x^2 + b \quad x + c \quad 1/(x + 1)$.

```
% Задание исходных данных
>> vx=[0;0.2;0.4;0.6;0.8;1]
vx =
     0
  0.2000
  0.4000
  0.6000
  0.8000
  1.0000
```

```
>> vy=[0.43;0.22;0.8;0.12;1;2]
vy =
  0.4300
  0.2200}
  0.8000
  0.1200
  1.0000
  2.0000
% задание матрицы переопределенной системы уравнений
>> D=[vx.^2 vx 1./(vx+1)]
>> D
D =
    0         0    1.0000
 0.0400    0.2000    0.8333
 0.1600    0.4000    0.7143
 0.3600    0.6000    0.6250
 0.6400    0.8000    0.5556
 1.0000    1.0000    0.5000
>> Coeff=D\vy
Coeff =
  3.0521
  -1.4391
  0.5126
  визуализация исходных данных и аппроксимирующей функции

>> i=1:length(vx);
>> j=1:length(X);
>> X=vx(1):0.01:vx(6);
>> Y=[ones(size(X')) X' X'.^2]*Coeff; % вычисление значений
                                      % аппроксимирующей функции
>> plot(vx(i),vy(i),'o',X(j),Y(j))    (рис. 9.5)
```

9.6. Аппроксимация функцией произвольного вида

Для решения задачи обобщенной нелинейной регрессии в MATLAB имеется функция **lsqnonlin( )**, возвращающая решение задачи нахождения точки минимума функции $f(x)$

$$\min_x(f(x)) = f_1(x)^2 + f_2(x)^2 + \ldots f_m(x)^2 + L,$$

где в общем случае $f(x)$ — вектор-функция, x — вектор-столбец искомых переменных, L — некоторая константа.

Синтаксис функции **lsqnonlin( )**:

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,options)
x = lsqnonlin(fun,x0,eb,ub,options,P1,P2,     )
```

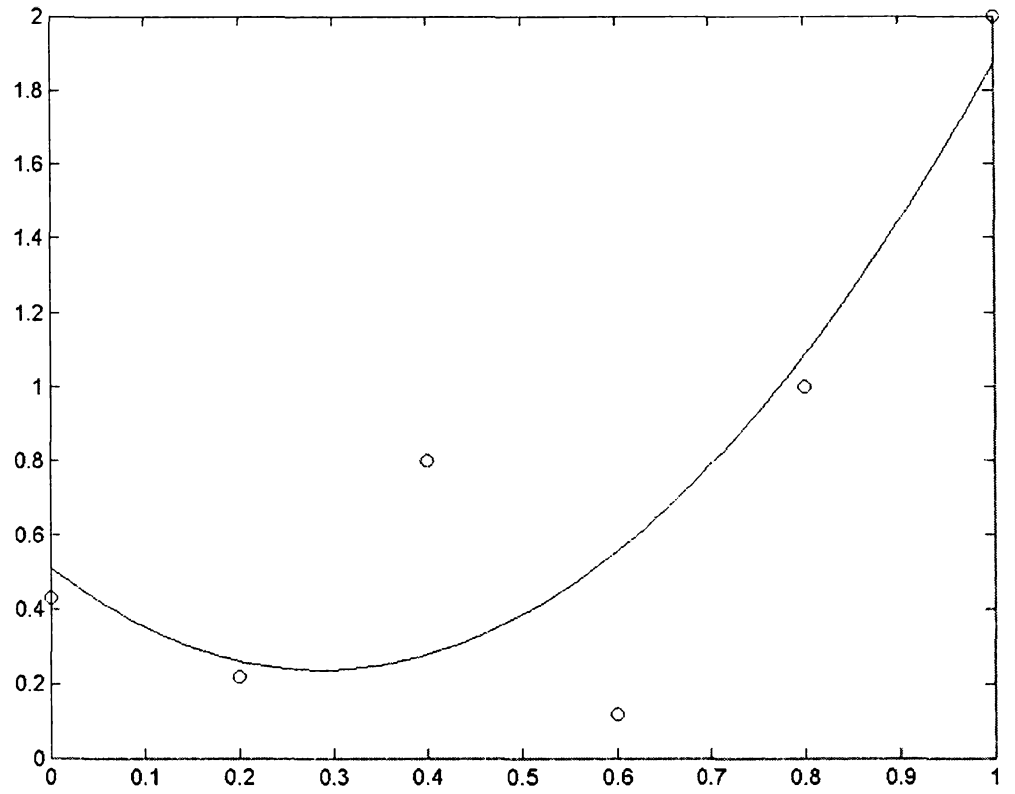


Рис. 9.5. Исходные данные и аппроксимирующая функция вида $F(x,a,b,c) = a \cdot x^2 + b \quad x + c \quad 1/(x+1)$

```
[x,resnorm] = lsqnonlin(. .)
[x,resnorm,residual] = lsqnonlin(...)
[x,resnorm,residual,exitflag] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonlin(. .)
[x,resnorm,residual,exitflag,output,lambda,jacobian] =
lsqnonlin(. .)
```

здесь

fun — название минимизируемой функции;

x0 — начальная точка, с которой начинается процесс поиска минимума функции;

lb,ub — соответственно левая и правая границы отрезка, на котором ищется минимум функции;

options — параметр, задающий режим работы функции оптимизирующей функции (перечень возможных значений данного параметра приведен в Help MATLAB в главе Optimization Toolbox в разделе Optimization Parameters);

P1,P2, ... — параметры, от которых зависит функция **fun**.

Рассмотрим пример, демонстрирующий использование данной функции для нахождения параметров функции $F(x,a,b,c) = exp(a + b \quad x + c \quad x^2)$.

Для решения данной задачи в MATLAB необходимо выполнить следующую последовательность действий:

1. Создать файл **F77.m**, содержащий описание функции, возвращающей значения вектор-функции $f(x)$

```
% листинг файла F77.m
function z=F77(Coeff,vx,vy)
k=1:length(vx);
z=vy-exp(Coeff(1)+Coeff(2)*vx+Coeff(3)*vx.^2);
```

2. Выполнить следующую последовательность команд

```
% задание исходных данных
>> vx=[0.3;0.4;1;1.4;2;4]
vx =
  0.3000
  0.4000
  1.0000
  1.4000
  2.0000
  4.0000
>> vy=[9.4;11.2;5;3;6;0.2]
vy =
  9.4000
  11.2000
  5.0000
  3.0000
  6.0000
  0.2000
>> z=[1 0 -1]   начальное приближения
z =
   1   0  -1
% вычисление коэффициентов аппроксимирующей функции
>> Coeff = lsqnonlin('F77',z',[],[],[],vx,vy)
Optimization terminated successfully:
Relative function value changing by less than OPTIONS.TolFun
Coeff =
  2.5696
  -0.8037
  0.0462
>> F=inline('exp(a+b*x+c*x.^2)','x','a','b','c'); % задание
                                          % аппроксимирующей функции
>> X=vx(1):0.01:vx(length(vx)); % координаты абсцисс, в которых
                                % будут вычисляться значения
                                % аппроксимирующей функции
>>  Y=feval(F,X,Coeff(1),Coeff(2),Coeff(3)); % вычисление
                                % значений аппроксимирующей функции
% визуализация исходных данных и аппроксимирующей функции
>> i=1:length(vx);
>> j=1:length(X);
>> plot(vx(i),vy(i),'o',X(j),Y(j))    (рис. 9.6)
```

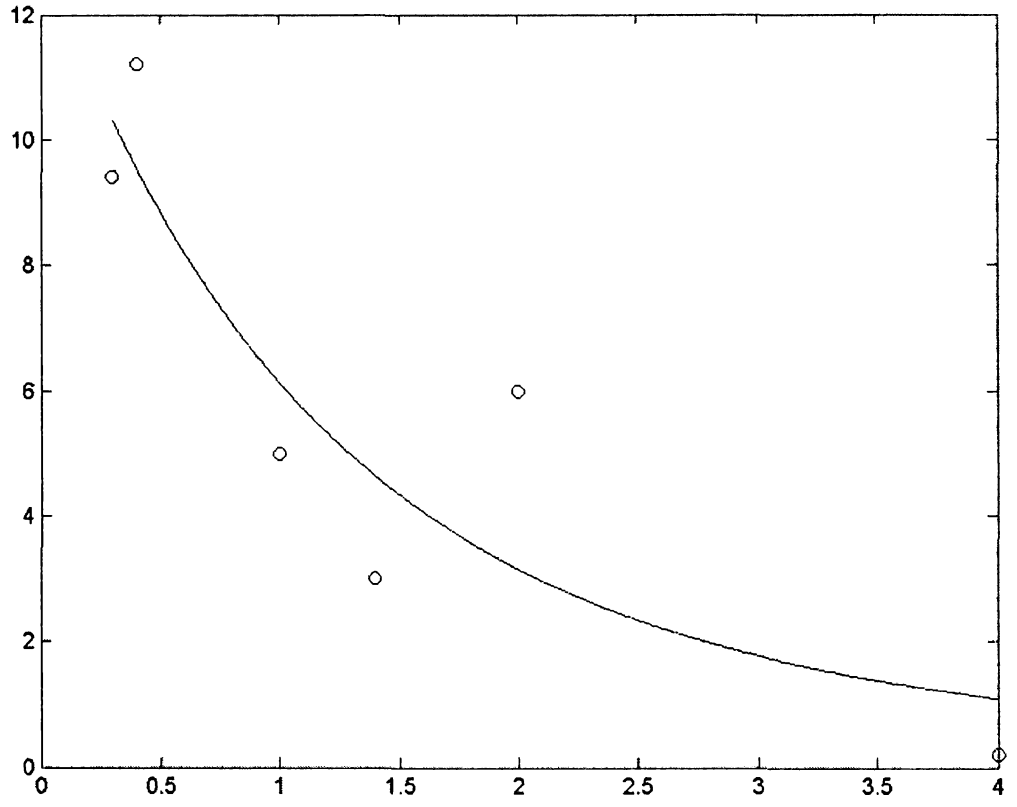


Рис. 9.6. Исходные данные и аппроксимирующая функция вида $F(x, a, b, c) = \exp(a + b \cdot x + c \cdot x^2)$

Вопросы для самопроверки

1. Назовите стандартные функции MATLAB, позволяющие вычислять статистические характеристики случайных последовательностей.
2. Сформулируйте общую постановку метода наименьших квадратов.
3. Какими средствами MATLAB можно реализовать решение задачи аппроксимации с помощью линейной функции и квадратичного трехчлена?
4. Как осуществляется нахождение приближающей функции в виде других элементарных функций?
5. Как осуществляется аппроксимация линейной комбинацией приближающих функций?
6. Как осуществляется аппроксимация функцией произвольного вида?

Глава 10

Моделирование статических электрических и магнитных полей

В данной главе рассматриваются примеры использования MATLAB для решения электростатических задач: расчета и визуализации напряженности и потенциала электростатического поля, создаваемого линейной системой электрических зарядов, а также расчет и визуализация напряженности магнитного поля линейной соленоида и тороидальной обмотки.

10.1. Электрическое поле системы неподвижных зарядов

Как известно, электрическое поле создаваемое неподвижным точечным электрическим зарядом q в вакууме в данной точке пространства характеризуется скалярным потенциалом

$$\varphi(\vec{R}) = \frac{1}{4\pi\varepsilon_0}\frac{q}{\left|\vec{R}-\vec{r}\right|}, \tag{10.1}$$

где $\vec{R}$ — радиус-вектор точки наблюдения, $\vec{r}$ — радиус-вектор точки, в которой находится электрический заряд, $\varepsilon_0 \approx 8.85 \cdot 10^{-12}$ ф/м. Векторной характеристикой данного поля является напряженность $\vec{E}$

$$\vec{E} = -\vec{\nabla}\varphi(\vec{R}) = \frac{1}{4\pi\varepsilon_0}\frac{q}{\left|\vec{R}-\vec{r}\right|^3}(\vec{R}-\vec{r}). \tag{10.2}$$

Скалярный потенциал электрической системы, состоящей из N электрических зарядов, $q_1, q_2 \ldots q_N$ и напряженность электрического поля удовлетворяют принципу суперпозиции:

$$\varphi(\vec{R}) = \frac{1}{4\pi\varepsilon_0}\sum_{i=1}^{N}\frac{q_i}{\left|\vec{R}-\vec{r}_i\right|}, \tag{10.3}$$

$$\vec{E}(\vec{R}) = \frac{1}{4\pi\varepsilon_0}\sum_{i=1}^{N}\frac{q_i}{\left|\vec{R}-\vec{r}\right|^3}\left(\vec{R}-\vec{r}\right), \tag{10.4}$$

где $\vec{r}$ — координата i-го заряда.

При анализе электростатических полей системы произвольно расположенных зарядов, характеризующихся скалярной функцией — потенциалом $\varphi(\vec{R})$ и векторной функцией — напряженностью $\vec{E}(\vec{R})$, возникает задача наглядного представления этих величин. Один из возможных способов представления потенциала электростатического поля $\varphi(\vec{R})$ реализуется следующей последовательностью действий:

1. задание функции, возвращающей значения потенциала, вычисляемые в соответствии (10.3) у узлах заданной координатной сетки;
2. задание дискретной координатной сетки;
3. вычисление в соответствие с (10.3) значение $\varphi(\vec{R})$ в каждом узле координатной сетки;
4. построение графика поверхности и карты эквипотенциалей.

Следуя алгоритму, описанному выше, создадим m-файл, содержащий описание функции $\varphi(\vec{R})$:

```
function Phi=Potential(q,xq,yq,X,Y);
% задание функции, возвращающей значения потенциала в узлах
% координатной сетки, вычисляемых в соответствии с (10.3)
% q - вектор, содержащий значения электрических зарядов
% xq, yq - векторы, содержащие x-е и y-е координаты
% электрических зарядов
% X, Y - векторы, содержащие x-е и y-е координаты узлов сетки
e0=8.85*10^-12; % диэлектрическая проницаемость вакуума
Nq=length(q); % число зарядов в системе
Nx=length(X); % число узлов по оси oX
Ny=length(Y); % число узлов по оси oY
              % проход по каждому узлу сетки
for i=1:Ny
  for j=1:Nx
    s=0;
    % суммирование по зарядам
    for k=1:Nq
      s=s+q(k)/((X(j)-xq(k))^2+(Y(i)-yq(k))^2)^0.5;
    end;
    M(i,j)=s/(4*pi*e0);
  end;
end;
Phi=M;
```

Функция **Potential** матрицу размерности Ny×Nx, содержащую значения потенциала в соответствующих узлах координатной сетки.

Вычислим потенциал, создаваемый системой, состоящей из $N = 50$ электрических зарядов, расположенных в точках с координатами $\left(-5R0 + \frac{10R0}{N} \cdot i\right)$, $y_i = 0$, $i = 1, \ldots, N$ (рис. 10.1). Для проведения вычислений в качестве единиц измерения заряда будем использовать заряд электрона $e = 1.6 \cdot 10^{-16} Кл$, единиц измерения длины — $R0 = 10^{-6} м$.

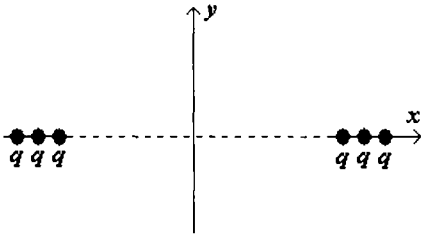


Рис. 10.1. Система электрических зарядов

Решение данной задачи в MATLAB выполняется следующей последовательностью команд:

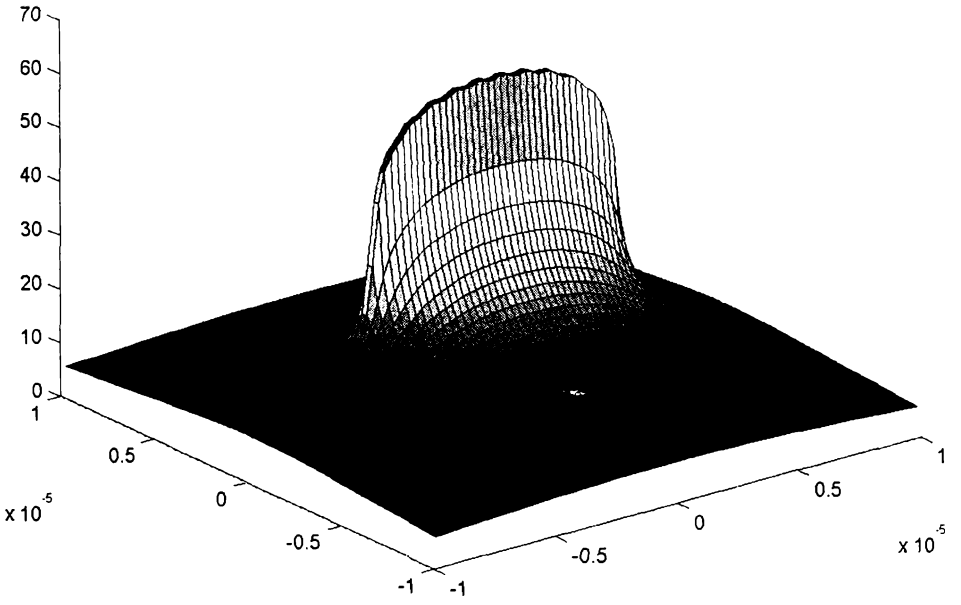


Рис. 10.2. Поверхность, задаваемая электростатическим потенциалом линейной системой зарядов

```
>>  e=1.6*10^-16; % заряд электрона
>>  R0=10^-6; % единица измерения расстояния
>>  N=50; % число зарядов
>>  i=1:N;
>> q(i)=e; % заполнение вектора, содержащего значения зарядов
>> x1=-5*R0; % x-ая координата левого конца системы зарядов
>> x2=5*R0; % x-ая координата правого конца системы зарядов
>> xq(i)=x1+(x2-x1)/N*i; % вычисление x-ых координат
                                                системы зарядов
>> yq(i)=0; % задание y-ых координат системы зарядов
>> N1=78; % число узлов прямоугольной координатной сетки
>> Xmin=-10*R0; Ymin=-10*R0; % задание координат нижнего левого
                             % угла координатной сетки
>> Xmax=10*R0; Ymax=10*R0;   задание координат верхнего правого
                             угла координатной сетки
>> i=1:N1+1;


>> X(i)=Xmin+(Xmax-Xmin)/N1*(i+1); % вычисление x-ых координат
                                   % узлов сетки
>> j=1:N1;
>> Y(j)=Ymin+(Ymax-Ymin)/N1*(j+1); % вычисление y-ых координат
                                   % узлов сетки
>> M(i,j)=Potential(q,xq,yq,X,Y); % вычисление значений
                          % потенциала в узлах координатной сетки
>> [X1,Y1]=meshgrid(X,Y); % вычисление матриц X1,Y1, используемых
                          % функциями визуализации двумерных и
                          % трехмерных зависимостей
>> surf(X1,Y1,M);    визуализация поверхности φ = φ(x,y)
```

Результат выполнения приведенной выше последовательности команд представлен на рис. 10.2.

```
>> figure(3);
>> contour(X1,Y1,M,33)    здесь 33 – число линий уровня
```

Результат выполнения приведенных выше команд представлен на рис. 10.3.

Для вывода значений соответствующей каждой эквипотенциали необходимо выполнить следующую последовательность команд:

```
>> figure(4);
>> [C,h] = contour(X1,Y1,M);
>>  clabel(C,h)
>> colormap cool
```

результат выполнения которой представлен на рис. 10.4.

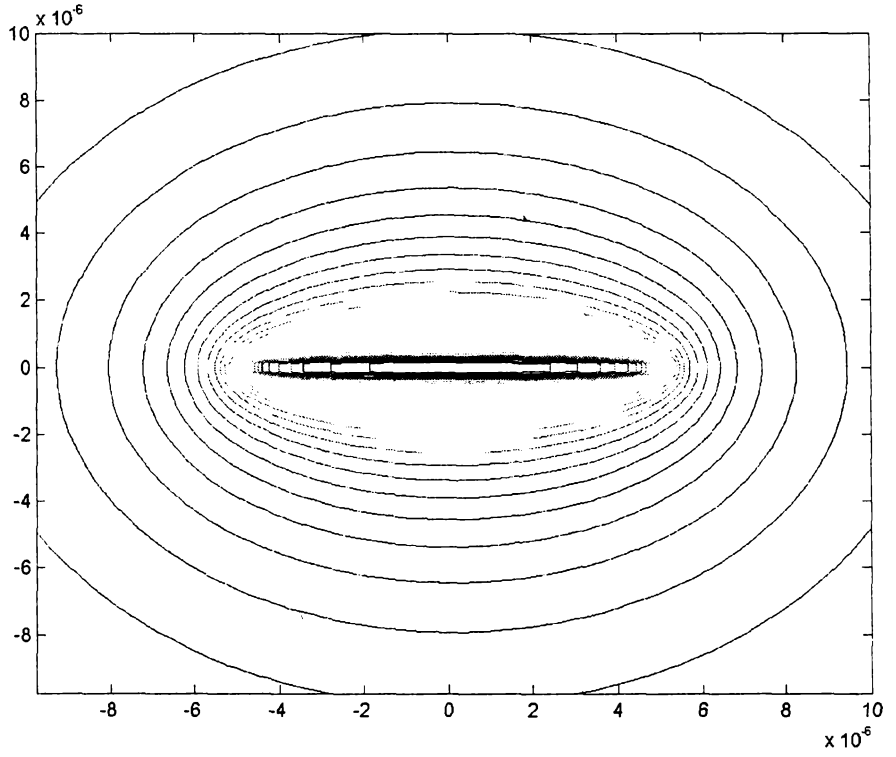


Рис. 10.3. Карта эквипотенциалей электростатического потенциала, создаваемого линеной системой зарядов

Здесь команда **contour** возвращает матрицу **C**, содержащую координаты вершин ломаных, аппроксимирующих соответствующие эквипотенциальные линии, вектор-столбец **h** дескрипторов для каждой линии уровня, используемый далее командой **clabel**, и строит карту линий уровня. Далее команда **clabel**, используя данные, возвращенные функцией **contour**, подписывает значения функции на каждой линии уровня. Команда **colormap** определяет цвет заливки карты линий уровня.

Для отображения на одном чертеже поверхности и карты линий уровня следует выполнить команды:

```
>> figure(5);
>>  meshc(X1,Y1,M)
```

Результаты выполнения данной команды представлены на рис. 10.5

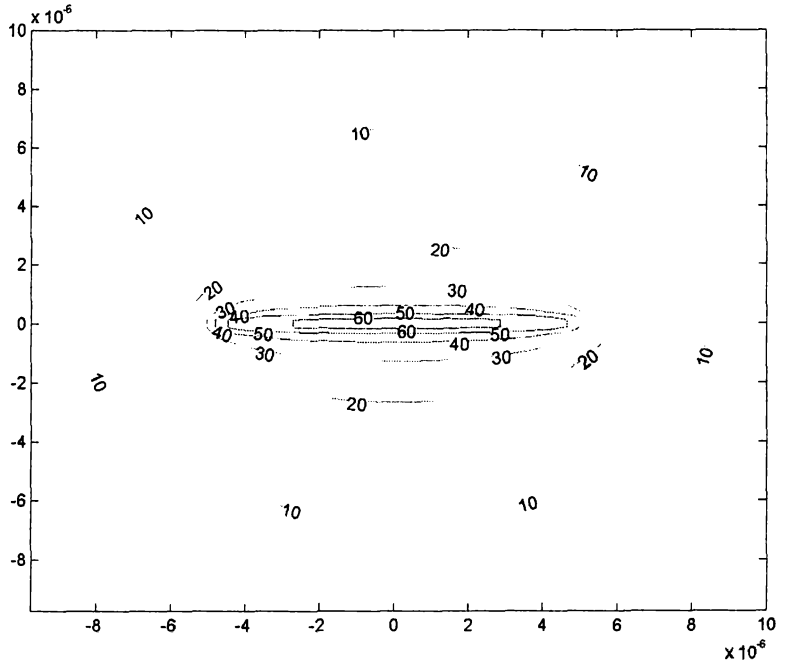


Рис. 10.4. Карта линий уровня электростатического потенциала, создаваемого линейной системой зарядов

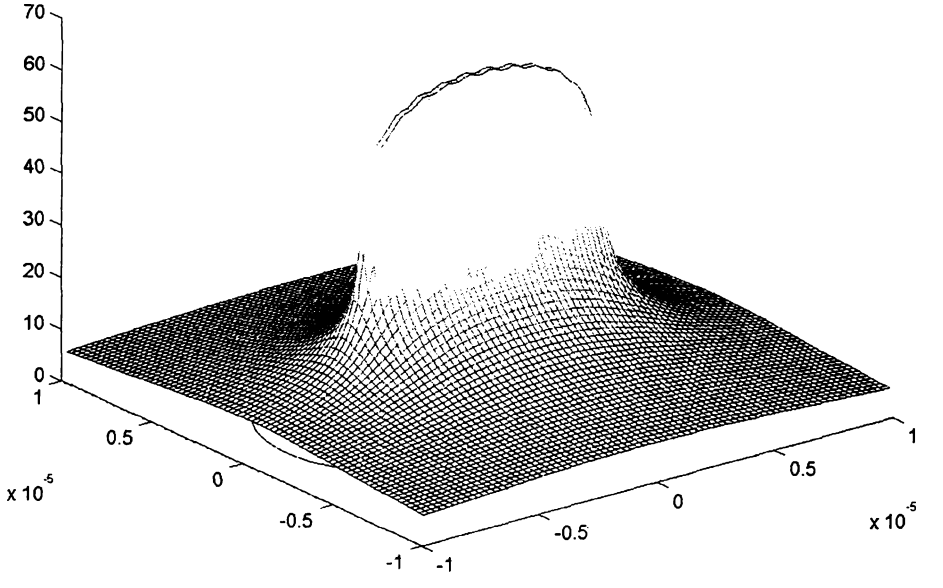


Рис. 10.5. Одновременное отображение на одном чертеже графика поверхности и карты линий уровня в лоскости XOY

Для построения трехмерной карты линий уровня (эвипотенциалей) функции $\varphi = \varphi(x, y)$ следует ввести команды:

```
>> figure(6);
>> contour3(X1,Y1,M);
```

Результаты выполнения данной последовательности команд представлены на рис. 10.6.

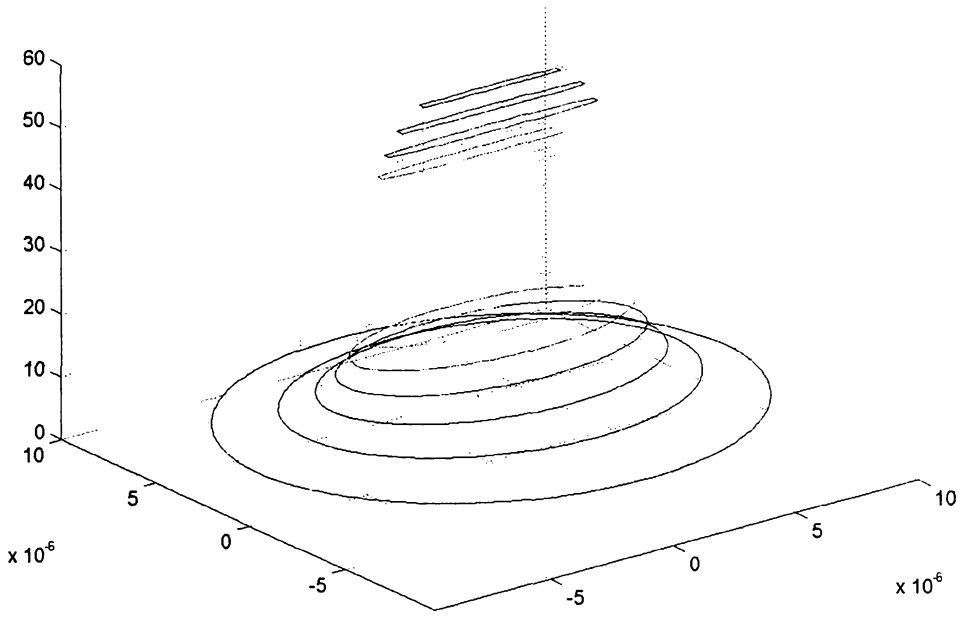


Рис. 10.6. Трехмерная карта линий уровня

В отличие от функции $\varphi(\bar{R})$, напряженность электростатического поля $\bar{E}(\bar{R})$ является векторной функцией, которая в каждой точке пространства характеризуется величиной поля и направлением. Для описания векторного поля будем использовать силовые линии — линии, касательные к которым в каждой точке параллельны вектору напряженности электрического поля. В связи с тем, что силовые линии несут только информацию о направлении вектора напряженности электрического поля, но не о его величине, для анализа изменения величины напряженности электрического поля в пространстве будем использовать функцию $\left|\bar{E}(\bar{R})\right|$.

Таким образом, для исследования особенностей напряженности электрического поля, создаваемого произвольной конфигурацией электрических зарядов следует:

1. задать в пространстве дискретную координатную сетку;
2. вычислить в узлах сетки координаты напряженности электрического поля $\vec{E}(\vec{R}) = (E_x, E_y)$;
3. построить в каждом узле сетки единичные векторы $\vec{n} = \left(\frac{E_x}{|\vec{E}|}, \frac{E_y}{|\vec{E}|}\right)$, касательные к силовой линии электрического поля;
4. построить график поверхности и карту линий уровня функции $|\vec{E}(\vec{R})|$.

Данный алгоритм в MATLAB может быть реализован следующей последовательностью команд:

```
>>  e=1.6*10^-16; % заряд электрона
>>  R0=10^-6; % единица измерения расстояния
>>  N=50; % число зарядов
>>  i=1:N;
>> q(i)=e; % заполнение вектора, содержащего значения зарядов
>> x1=-5*R0; % x-ая координата левого конца системы зарядов
>> x2=5*R0; % x-ая координата правого конца системы зарядов
>> xq(i)=x1+(x2-x1)/N*i; % вычисление x-ых координат системы
                         % зарядов
>> yq(i)=0; % задание y-ых координат системы зарядов
>> N1=23; % число узлов прямоугольной координатной сетки
>> Xmin=-10*R0; Ymin=-10*R0; % задание координат нижнего левого
                             % угла координатной сетки
>> Xmax=10*R0; Ymax=10*R0;   задание координат верхнего правого
                             угла координатной сетки
>> i=1:N1+1;


>> X(i)=Xmin+(Xmax-Xmin)/N1*(i+1);    вычисление x-ых координат
                                      узлов сетки
>> j=1:N1+1;
>> Y(j)=Ymin+(Ymax-Ymin)/N1*(j+1);    вычисление y-ых координат
                                      узлов сетки
>> M(i,j)=Potential(q,xq,yq,X,Y); % вычисление значений
                                  % потенциала в узлах координатной сетки
>> [X1,Y1]=meshgrid(X,Y); % вычисление матриц X1,Y1, используемых
                          % функциями визуализации двумерных и
                          % трехмерных зависимостей
>> [px,py]=gradient(-M,0.1,0.1); % вычисление градиента функции
                        % φ(R̄) вычисление координат единичных векторов,
                        % касательныхк силовой линии
>> px1=px./(px.^2+py.^2).^0.5;
>> py1=py./(px.^2+py.^2).^0.5;
>> figure(1);
>> quiver(X1,Y1,px1,py1,0.5) % построение векторного поля
% единичных векторов, касательных к силовой линии в узлах
```

```
% координатной сетки
>> mp=(px.^2+py.^2) ^0.5; % вычисление абсолютных значений
% вектора  напряженности в узлах координатной сетки
>> figure(2);
>> contour(X1,Y1,mp,17);   построение линий равной напряженности
```

Результаты выполнения приведенной выше последовательности команд приведены на рис. 10.7 (векторное поле), рис. 10.8 (линии равной напряженности).

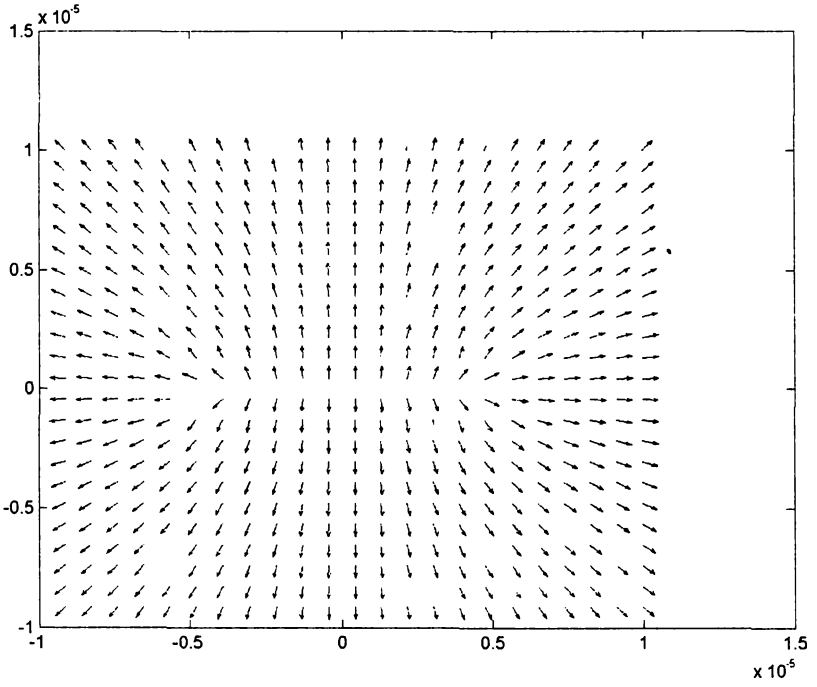


Рис. 10.7. Поле единичных векторов, касательных к силовым линиям электростатического поля, создаваемого линейной системой зарядов

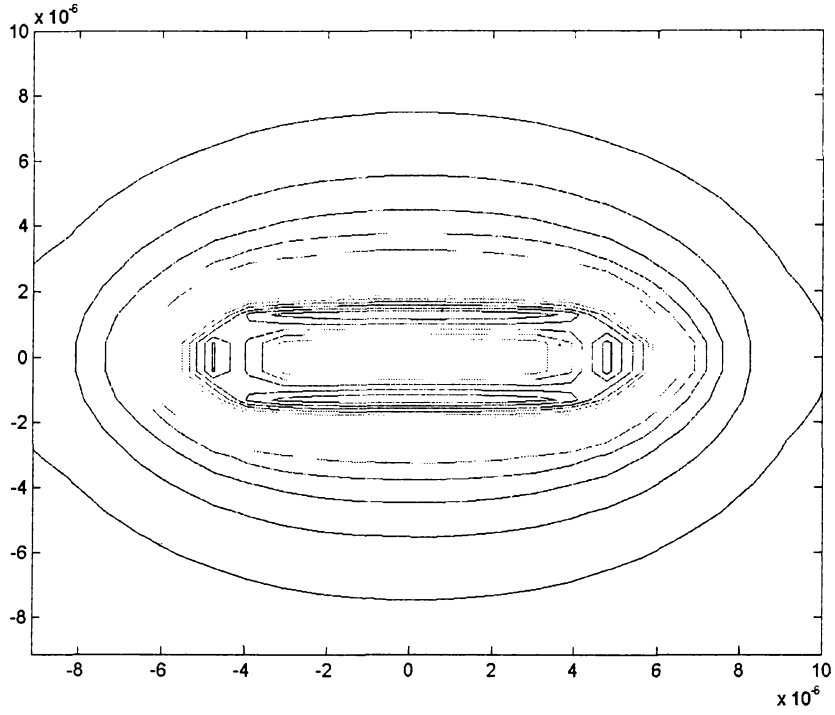


Рис. 10.8. Карта линий равной напряженности электростатического поля, создаваемого линейной системой зарядов

10.2. Магнитное поле витка с постоянным током

Изучение электрических полей систем неподвижных электрических зарядов, проведенное нами в предыдущем разделе, было основано на законе Кулона (10.2) и принципе суперпозиции (10.3), (10.4). Аналогом закона Кулона в электростатике в магнитостатике является закон Био-Савара-Лапласа, в соответствие с которым напряженность магнитного поля, создаваемого элементом тока $Id\vec{l}$, имеет вид

$$d\vec{B} = \frac{\mu_0}{4\pi} I \frac{\left[d\vec{l} \times \vec{R}\right]}{\left|\vec{R}\right|^3}, \qquad (10.5)$$

где I — величина тока измеряется в амперах (А), напряженность магнитного поля — $\vec{B}$ в теслах (Тл), $\mu_0 = 4\pi \cdot 10^{-7}$ *Тл м/А* — магнитная проницаемость вакуума.

Закон Био-Савара-Лапласа имеет в магнитостатике совершенно общий характер и вместе с принципом суперпозиции быть использован для нахождения напряженности магнитного поля, создаваемого произвольной системой постоянных токов, в любой точке пространства. Действительно, так как напряженность магнитного поля, создаваемого элементом тока $Id\vec{l}$, расположенном в точке с радиус-вектором $\vec{r}$, в точке наблюдения с радиус-вектором $\vec{R}$ равна

$$d\vec{B} = \frac{\mu_0}{4\pi} I \frac{\left[d\vec{l} \times \left(\vec{R} - \vec{r}\right)\right]}{\left|\vec{R} - \vec{r}\right|^3}, \tag{10.6}$$

то для вычисления напряженности магнитного поля, создаваемого произвольной системой постоянных токов, следует задать ориентацию каждого элемента тока $Id\vec{l}$, входящего в структуру, и вычислить напряженности магнитных полей, создаваемых каждым элементом тока и, в соответствие с принципом суперпозиции, произвести суммирование напряженностей магнитных полей:

$$d\vec{B} = \frac{\mu_0}{4\pi} I \sum_{i=1}^{N} \frac{\left[d\vec{l}_i \times \left(\vec{R} - \vec{r}\right)\right]}{\left|\vec{R} - \vec{r}\right|^3}, \tag{10.7}$$

где N — число элементов, $\vec{r}_i$ — радиус-вектор, соответствующего элемента

Отметим, что вычислить магнитное поле аналитически, используя (10.6), (10.7), оказывается возможным, только для структур, имеющих достаточно высокую степень симметрии (прямой провод, поле кольца на оси симметрии и т.д.). Для большинства токовых конфигураций расчет магнитного поля может быть проведен только численно. Наиболее важными геометрическими конфигурациями токовых структур, для которых необходимо знать распределение напряженности магнитного поля являются: прямой провод, петля, соленоид, тороидальная обмотка. Далее мы опишем документ, позволяющий вычислять магнитное поле петли с постоянным током, и проведем критический анализ известных моделей, используемых при расчете магнитных полей соленоида и тороидальной обмотки с постоянным током.

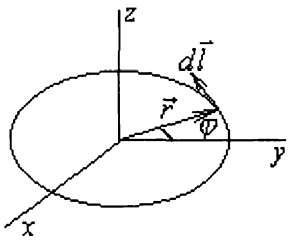


Рис. 10.9. К решению задачи о вычислении магнитного поля витка с током

Для расчета напряженности магнитного поля петли с током выберем систему координат, в которой плоскость XoY совпадает с плоскостью петли (рис. 10.9).

В выбранной системе координат положение i-го элемента тока $Id\vec{l}$,

$$d\vec{l} = Ia\begin{bmatrix}\sin\varphi_i\\ -\cos\varphi_i\\ 0\end{bmatrix}d\varphi, \qquad (10.8)$$

где $d\varphi = \frac{2\pi}{N}$, $\varphi_i = id\varphi$, $i = 0,1, \ldots N-1$, N — число элементов тока, описывается радиус-вектором $\vec{r}_i$,

$$\vec{r}_i = a\begin{bmatrix}\cos\varphi_i\\ \sin\varphi_i\\ 0\end{bmatrix}. \qquad (10.9)$$

Подставив (10.6), (10.7) в (10.5), нетрудно получить выражения для составляющих вектора напряженности магнитного поля B_x, B_y, B_z, однако для расчета напряженности и визуализации магнитного поля в MATLAB этого не требуется, так как в пакет встроены функции для вычисления стандартных действий векторного анализа: вычисление скалярного произведения, векторного произведения и т.д.

На примере вычисления и визуализации напряженности магнитного поля, создаваемого витком с током в плоскости $x = 0$, опишем последовательность действий, позволяющих решить поставленную задачу. В рассматриваемой плоскости, как очевидно из соображений симметрии, составляющая напряженности магнитного поля вдоль оси oX равна нулю, поэтому далее мы проводим вычисления только составляющих напряженности вдоль осей oY, oZ. Так как далее нас будет интересовать форма силовых линий, но не абсолютное значение напряженностей множитель $(\mu_0 / 4\pi)I$ будем полагать равным единице.

Сначала необходимо создать файл **Ring.m**, содержащий описание функции, возвращающей значения составляющих напряженности магнитного поля, листинг которого мы приводим ниже:

```
function [By,Bz]=Ring(a,Y,Z)
% функция возвращающая значения проекций напряженности магнитного
% поля на оси oY, oZ, вычисляемые в соответствии с (10.7)
Nstep=100; % количество элементов, аппроксимирующих кольцо
X=0;
Ny=length(Y); % число узлов координатной сетки по оси oY
Nz=length(Z); % число узлов координатной сетки по оси oZ
deltaphi=2*pi/Nstep; % шаг по углу
n=1:Nstep+1;
phi(n)=deltaphi*(n-1);   вычисление вектора φ_n = ndφ
                         вычисление составляющих напряженности
                         магнитного поля  в соответствии с (10.7)
for i=1:Nz
```

```
    for j=1:Ny
      s=[0 0 0];
      for n=1:Nstep+1
        dL=[-a*sin(phi(n))*deltaphi  a*cos(phi(n))*deltaphi 0];
         r=[a*cos(phi(n)) a*sin(phi(n)) 0];
         R=[X Y(j) Z(i)];
         s=s+cross(dL,(R-r))./((R-r)*(R-r)').^(3/2); % знак
                                  % < ' > – операция транспонирования
      end;
      by(i,j)=s(2);
      bz(i,j)=s(3);
     end;
  end;
  By=by;
  Bz=bz;
```

Далее необходимо выполнить следующую последовательность команд:

```
N1=21; % число узлов сетки
i=1:N1+1;
Ymin=-5; Zmin=-5; % нижний левый угол координатной сетки
Ymax=5; Zmax=5; % верхний правый угол координатной сетки
                % вычисление координат узлов сетки
Y(i)=Ymin+(Ymax-Ymin)/N1*(i-1);
Z(i)=Zmin+(Zmax-Zmin)/N1*(i-1);
a=1; % радиус кольца
[By Bz]=Ring(a,0,Y,Z);   вычисление проекций
                         напряженности магнитного поля


mp=(By.^2+Bz.^2).^0.5; % вычисление значений модуля
                       % напряженности магнитного поля
                       % в узлах координатной сетки
                       % вычисление координат единичных
                       % векторов, касательных к силовой линии
                     магнитного поля, в узлах координатной сетки
by1=By./mp;
bz1=Bz./mp;
quiver(Y,Z,by1,bz1);   визуализация векторного поля
```

Результаты выполнения приведенной выше последовательности команд представлены на рис. 10.10.

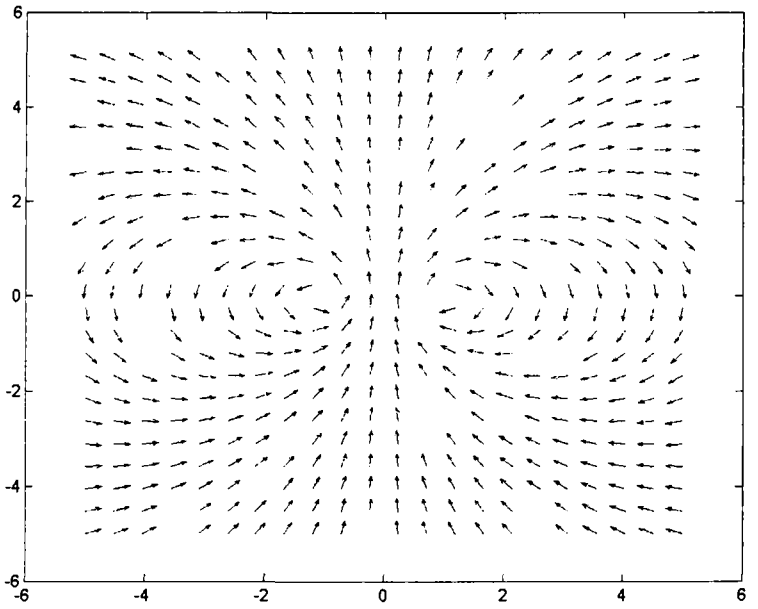


Рис. 10.10. Поле единичных векторов, касательных к силовым линиям магнитного поля, создаваемого витком с током

Анализ рис. 10.10 позволяет сделать вывод о том, что поле единичных векторов позволяет на качественном уровне получать информацию о форме силовых линий и направления напряженности магнитного поля в данной точке координатной сетки.

Определенный недостаток технологии визуализации силовых линий магнитного поля состоит в том, что линия, являющаяся по своей физической сути непрерывной, заменяется набором векторов, касательных к данной линии. Для получения более адекватного изображения силовых линий магнитного поля использовать векторный потенциал магнитного поля $\vec{A}$, с связанный с напряженностью магнитного поля следующим соотношением:

$$\vec{B} = rot\vec{A}. \tag{10.10}$$

Используя (10.10), можно показать [1], что силовая линия, есть линия на которой $\left|\vec{A}(x, y, z)\right| = const$. Следовательно, для построения силовой линии магнитного поля необходимо вычислить модуль векторного потенциала в узлах координатной сетки и затем построить карту эквипотенциалей.

Векторный потенциал магнитного поля, создаваемого элементом тока $Id\vec{l}$, равен

$$d\vec{A} = \frac{\mu_0}{4\pi} \frac{Id\vec{l}}{\left|\vec{R} - \vec{r}\right|}, \tag{10.11}$$

где $\vec{R}$ — радиус-вектор точки наблюдения, $\vec{r}$ — радиус-вектор элемента тока.

Векторный потенциал магнитного поля $\vec{A}$, создаваемого произвольной системой токов $Id\vec{l}$, в соответствие с принципом суперпозиции равен

$$\vec{A} = \sum_{i=1}^{N} d\vec{A}_i = \frac{\mu_0}{4\pi} \sum_{i=1}^{N} \frac{Id\vec{l}}{\left|\vec{R} - \vec{r}\right|}. \qquad (10.12)$$

Так как далее нас будет интересовать форма силовых линий, но не абсолютное значение напряженностей множитель будем полагать равным единице.

Для практической реализации предложенного алгоритма, как и ранее, сначала необходимо создать файл **RingA.m**, содержащий описание функции, возвращающей значения составляющих напряженности магнитного поля, листинг которого мы приводим ниже:

```
function A=RingA(a,Y,Z)
% функция возвращающая значения модуля вектор-потенциала
% магнитного поля, вычисляемого в узлах координатной сетки,
% расположенной в плоскости YoZ, в соответствии с (110.12)
Nstep=100; % количество элементов, аппроксимирующих кольцо
X=0;
Ny=length(Y); % число узлов координатной сетки по оси oY
Nz=length(Z); % число узлов координатной сетки по оси oZ
deltaphi=2*pi/Nstep; % шаг по углу
n=1:Nstep+1;
phi(n)=deltaphi*(n-1); вычисление вектора φ_n = ndφ
% вычисление модуля напряженности вектор-потенциала магнитного
% поля
for i=1:Nz
for j=1:Ny
s=[0 0 0];
for n=1:Nstep+1
dL=[-a*sin(phi(n))*deltaphi a*cos(phi(n))*deltaphi 0];
r=[a*cos(phi(n)) a*sin(phi(n)) 0];
R=[X Y(j) Z(i)];
s=s+dL./((R-r)*(R-r)').^(1/2);
end;
ax(i,j)=s(1);
ay(i,j)=s(2);
az(i,j)=s(3);
end;
end;
A=(bx.^2+by.^2+bz.^2).^0.5;
```

Далее необходимо выполнить следующую последовательность команд:

```
N1=21; % число узлов сетки
i=1:N1+1;
Ymin=-5; Zmin=-5; % нижний левый угол координатной сетки
Ymax=5; Zmax=5; % верхний правый угол координатной сетки
                % вычисление координат узлов сетки
Y(i)=Ymin+(Ymax-Ymin)/N1*(i-1);}
Z(i)=Zmin+(Zmax-Zmin)/N1*(i-1);
a=1; % радиус кольца
mp=RingA(a,Y,Z); % вычисление модуля векторного потенциала
contour(Y,Z,mp,55);
```

Результаты выполнения приведенной выше последовательности команд представлен на рис. 10.11.

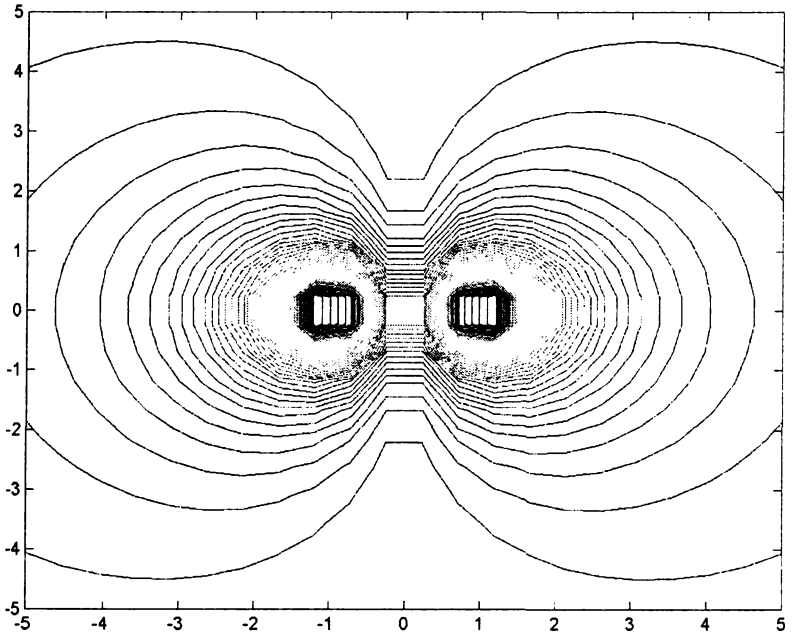


Рис. 10.11. Карта силовых линий магнитного поля, создаваемого витком с током

Анализ карты эквипотенциальных поверхностей, представленных на рис. 10.11, позволяет выявить два недостатка. Во-первых, на карте не правильно отображаются силовые линии, проходящие вблизи центра кольца, что объясняется особенностями вычислительного алгоритма. Во-вторых, силовые линии оказываются негладкими, что обусловлено использованием координатной сетки с достаточно большим шагом и использованием MATLAB при построении силовых линий магнитного поля последовательных отрезков прямых. Устранить второй недостаток достаточно просто, используя какой-либо тип интерполяцию исходных данных на координатную сетку меньшего размера:

```
% задание новой координатной сетки
>> yi=Ymin:(Ymax-Ymin)/1000:Ymax;
>> zi=Zmin:(Zmax-Zmin)/1000:Zmax;
% формирование матриц, используемых функцией interp2
>> [y1 z1]=meshgrid(Y,Z);
>> [y2 z2]=meshgrid(yi,zi);
% вычисление значений векторного потенциала на координатной сетке
% меньшего размера с использованием бикубической)
% сплайн-интерполяции
>>  zi3 = interp2(y1,z1,mp,y2,z2,'bicubic');
>> contour(yi,zi,zi3,55); % визуализация силовых линий
```

Результат выполнения приведенной выше последовательности команд представлен на рис. 10.12.

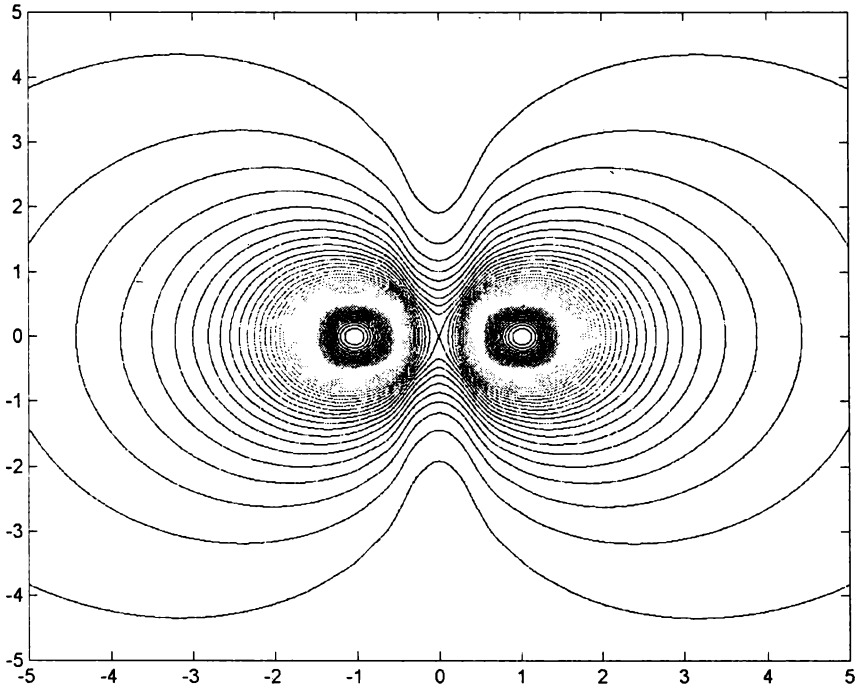


Рис. 10.12. Карта силовых линий магнитного поля при использовании двумрной сплайн интерполяции

10.3. Магнитное поле соленоида с постоянным током

Вопрос о структуре магнитного поля однослойного соленоида с постоянным током обсуждается практически во всех учебниках — от школьных до ставших в настоящее время классическими вузовских учебников по электричеству, магнетизму [2-6] и электродинамике [7-11]. Расчет поля на оси соленоида, допускающий решение в аналитическом виде, в случае бесконечно длинного соленоида и соленоида конечной длины предлагается в качестве задач во многих задачниках [12-14].

Соленоид, согласно устоявшейся точке зрения, рассматривается как источник магнитного поля. Один конец соленоида является северным полюсом, другой южным, поле вне соленоида считается аксиально-симметричным, а внутри катушки в приближении бесконечно длинного соленоида — однородным. Общий вид силовых линий данного магнитного поля показан на рис. 10.13. Это подтверждается элементарными опытами со стрелкой компаса и магнитными опилками, располагаемыми на горизонтальной плоской поверхности с малым коэффициентом трения, проходящей через продольную ось соленоида.

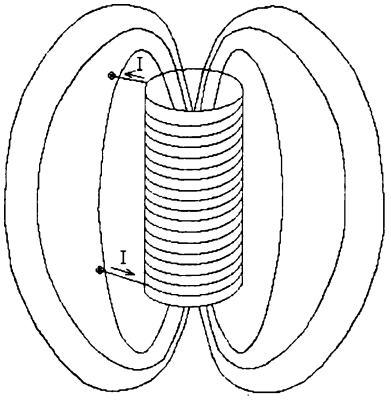


Рис. 10.13. Силовые линии соленоида с постоянным током

Известный метод расчета напряженности магнитного поля однослойного соленоида состоит в представлении соленоида набором последовательно расположенных колец и суммировании в соответствии с принципом суперпозиции напряженностей магнитных полей, создаваемых каждым кольцом в выбранной точке пространства. При таком подходе неявно предполагается, что ток при плотной намотке вдоль продольной оси соленоида отсутствует или исчезающе мал по сравнению с током вдоль кольца. Это послужило причиной, по которой магнитное поле, обусловленное составляющей тока вдоль оси соленоида, считалось малым по сравнению с магнитным полем, обусловленным движением электрических зарядов вдоль кольца. Данное допущение однако недостаточно корректно.

Доказательство высказанного утверждения проведем в два этапа; на первом построим качественную картину протекания тока по соленоиду, на втором этапе проведем количественные расчеты. Причем, в отличие от традиционного подхода, обсужденного выше, рассмотрим изменение условий движения носителей заряда при последовательном усложнении геометрии токовой структуры (прямой ток–спираль–соленоид). Оставаясь в рамках электронной теории, рассмотрим ток в проводе как последовательность движущихся с постоянной скоростью зарядов. Очевидно, что намотка провода не меняет скорость движения носителей заряда вдоль провода v_0 (т.е. абсолютную величину скорости), однако появляются составляющие скорости носителей заряда вдоль координатных осей (рис. 10.14). При этом движение носителей заряда вдоль оси oZ будет движением с постоянной скоростью v_z, в плоскости XoY — с постоянной скоростью v по окружности. Выразим данные скорости через радиус спирали a и шаг спирали h, для чего рассмотрим развертку одного витка спирали в плоскости XoZ (рис. 10.15).

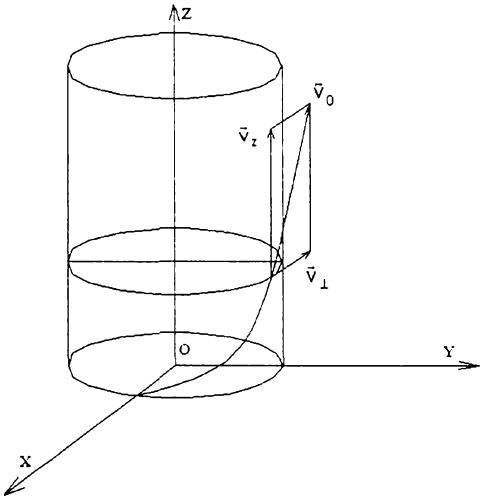


Рис. 10.14. К объяснению причин существования движения зарядов вдоль продольной оси соленоида

Из рис. 10.15 очевидно, что

$$v_z = v_0 \cos\alpha = v_0 \frac{h}{\sqrt{(2\pi a)^2 + h^2}}; \qquad (10.13)$$

$$v_\perp = v_0 \frac{2\pi a}{\sqrt{(2\pi a)^2 + h^2}}. \qquad (10.14)$$

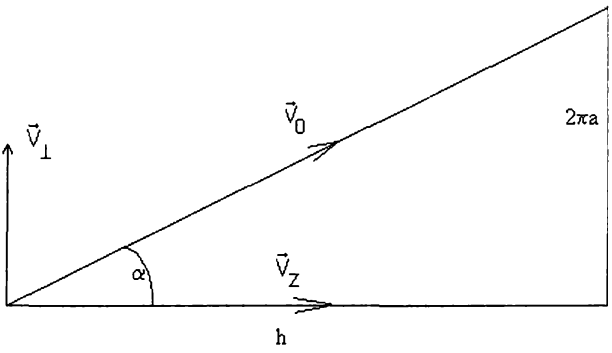


Рис. 10.15. Развертка одного витка спиральной намотки

Учитывая, что шаг спирали h выражается через длину соленоида и L_s число витков N как $h = L_s/N$, и переходя в (10.13), (10.14) к токам, получаем

$$I_z = N_e e v_0 \frac{L_s}{\sqrt{(2\pi a N)^2 + L_s^2}}, \tag{10.15}$$

$$I_z = N_e e v_0 \frac{2\pi a N}{\sqrt{(2\pi a N)^2 + L_s^2}}, \tag{10.16}$$

где N_e — линейная плотность электронов в соленоиде (число электронов в проводе, намотанном на соленоид, отнесенное к длине соленоида); e — заряд электрона.

Кроме того, намотка провода приводит к увеличению длины провода с током L, которая выражается через число витков и шаг намотки

$$L = N\sqrt{(2\pi a)^2 + h^2} = \sqrt{(2\pi a N)^2 + L_s^2}. \tag{10.17}$$

Анализ выражений (10.16), (10.17) показывает, что поле соленоида с однослойной намоткой является суперпозицией полей двух токов, один которых является круговым, второй — продольным. Увеличение числа витков приводит, с одной стороны, к уменьшению шага спирали и, следовательно, к уменьшению величины продольного тока (см. выражения (10.15), (10.16)), с другой стороны, к увеличению длины провода, намотанного на соленоид, что приводит к возрастанию количества носителей заряда, двигающихся в соленоиде, т.е. возрастанию линейной плотности заряда. Из выражений (10.15) и (10.16) видно, что количество электричества, перемещающееся в направлении оси oZ в единицу времени остается неизменным при увеличении числа витков соленоида. Таким образом, при любом количестве витков (любом шаге намотки) количество электричества, перемещающегося в направлении продольной оси соленоида в единицу времени оказывается равным количеству электричества в прямом проводе. Следо-

вательно, соленоид с однослойной обмоткой должен обладать аксиально симметричным магнитным полем в плоскости, перпендикулярной продольной оси соленоида, подобным полю прямого тока. Учитывая аналогию с магнитным полем прямого тока, естественно в дальнейшем называть его поперечным полем — в отличие от магнитного поля, приведенного на рис. 10.13. При большом числе витков соленоид можно рассматривать как проводящий цилиндр конечных размеров, по которому течет продольный ток, следовательно, можно ожидать появления отличия поперечного магнитного поля соленоида и магнитного поля прямого тока, которое должно проявляться при увеличении числа витков.

Для проверки рассуждений, приведенных выше, получим аналитическое выражение для напряженности магнитного поля соленоида на основе закона Био-Савара-Лапласа (10.6) и принципа суперпозиции, проинтегрировав (10.6) по всем элементам тока.

Как известно [15], положение любой точки спирали полностью определяется заданием единичного вектора $\vec{n}$, направленного вдоль оси спирали (ось oZ), и вектора $\vec{a}$, отмечающего положение точки спирали в плоскости, перпендикулярной к вектору (рис. 10.16). Тогда положение любой точки спирали определяется радиус-вектором $\vec{R}$, зависящим от одного параметра — угла поворота φ:

$$\vec{R}(\varphi) = \vec{n}h\frac{\varphi}{2\pi} + \vec{a}\cos\varphi + [\vec{n}\times\vec{a}]\sin\varphi. \qquad (10.18)$$

Элемент тока $Id\vec{l}$ выражается через вектор $t(\varphi)$, касательный к спирали, как

$$Id\vec{l} = I\vec{t}(\varphi)d\varphi, \qquad (10.19)$$

где

$$\vec{t}(\varphi) = \frac{h}{2\pi}\vec{n} - \vec{a}\sin\varphi + [\vec{n}\times\vec{a}]\cos\varphi. \qquad (10.20)$$

Обозначим вектор, направленный из начала координат в точку наблюдения, через $\vec{R}_0$, тогда вектор $\vec{r} = \vec{R}_0 - \vec{R}(\varphi)$. Таким образом, напряженность магнитного поля в точке наблюдения записывается в виде

$$\vec{B} = \frac{\mu_0 I}{4\pi}\int_0^{2\pi N}\frac{\left[\vec{t}(\varphi)\times\left(\vec{R}_0 - \vec{R}(\varphi)\right)\right]}{\left|\vec{R}_0 - \vec{R}(\varphi)\right|^3}d\varphi. \qquad (10.21)$$

Для вычисления интеграла (10.21) перейдем к координатной записи. Заметив, что в выбранной системе координат векторы $\vec{n}, \vec{a}, [\vec{n}\times\vec{a}]$, входящие в (10.18)–(10.21), имеют координаты $\vec{n} = (0,0,1)$, $\vec{a} = (a,0,0)$, $[\vec{n}\times\vec{a}] = (0,a,0)$ получим координаты векторов $\vec{R}(\varphi), \vec{t}(\varphi), \vec{R}_0 - \vec{R}(\varphi)$ в виде

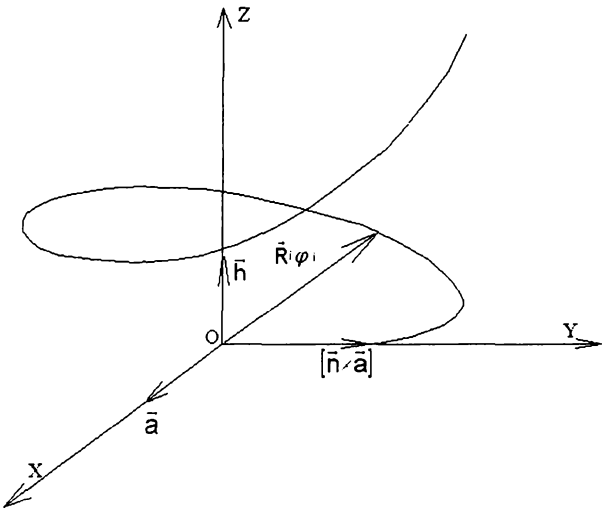


Рис. 10.16. Радиус-вектор точки, движущейся по спирали

$$\vec{R}(\varphi) = \left(a\cos\varphi, a\sin\varphi, \frac{h}{2\pi}\varphi\right), \tag{10.22}$$

$$\vec{t}(\varphi) = \left(-a\sin\varphi, a\cos\varphi, \frac{h}{2\pi}\right), \tag{10.23}$$

$$\vec{R}_0 - \vec{R}(\varphi) = \left(x_0 - a\cos\varphi, y_0 - a\sin\varphi, z_0 - \frac{h}{2\pi}\varphi\right). \tag{10.24}$$

Подставляя (10.22)–(10.24) в (10.21), получаем для составляющих напряженности магнитного поля соленоида вдоль соответствующих координатных осей:

$$B_x = \frac{\mu_0 I}{4\pi}\int_0^{2\pi N} \frac{\left(\left(z_0 - \frac{h}{2\pi}\varphi\right)a\cos\varphi - \frac{h}{2\pi}(y_0 - a\sin\varphi)\right)}{\left(R_0^2 + a^2 + \left(\frac{h}{2\pi}\right)^2 - 2\left(x_0 a\cos\varphi + a y_0 \sin\varphi + \frac{h}{2\pi}z_0\right)\right)^{3/2}} d\varphi; \tag{10.25}$$

$$B_y = -\frac{\mu_0 I}{4\pi}\int_0^{2\pi N} \frac{\left((x_0 - a\cos\varphi)\frac{h}{2\pi} + a\sin\varphi\left(z_0 - \frac{h}{2\pi}\varphi\right)\right)}{\left(R_0^2 + a^2 + \left(\frac{h}{2\pi}\varphi\right)^2 - 2\left(x_0 a\cos\varphi + a y_0 \sin\varphi + \frac{h}{2\pi}z_0\right)\right)^{3/2}} d\varphi; \tag{10.26}$$

$$B_z = \frac{\mu_0 I}{4\pi}\int_0^{2\pi N} \frac{\left((y_0 - a\sin\varphi)a\sin\varphi + a\cos(x_0 - a\cos\varphi)\right)}{\left(R_0^2 + a^2 + \left(\frac{h}{2\pi}\varphi\right)^2 - 2\left(x_0 a\cos\varphi + a y_0 \sin\varphi + \frac{h}{2\pi} z_0\right)\right)^{3/2}} d\varphi. \tag{10.27}$$

В связи с тем, что соленоид имеет цилиндрическую форму, представляется целесообразным записать компоненты поля в цилиндрической системе координат. Для этого положение точки наблюдения будем характеризовать в плоскости, перпендикулярной продольной оси соленоида, радиус-вектором $\vec{r}_0$ и углом поворота φ_0, положение относительно продольной соленоида — координатой z_0. Тогда выражения для напряженности составляющих поля в цилиндрической системе координат находятся соответствующей заменой переменных в (10.25)–(10.27): $x_0 \to r_0\cos\varphi_0, y_0 \to r_0\sin\varphi_0, R_0^2 \to r_0^2 + z_0^2$ и переходом из декартовой в цилиндрическую систему координат, определяемым матрицей поворота относительно оси oZ [11]:

$$\begin{pmatrix} B_r \\ B_\varphi \\ B_z \end{pmatrix} = \begin{pmatrix} \cos\varphi_0 & \sin\varphi_0 & 0 \\ -\sin\varphi_0 & \cos\varphi_0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix}. \tag{10.28}$$

Описанная выше замена переменных и выражение (10.29) позволяют легко получить окончательные выражения для составляющих напряженности магнитного поля в цилиндрической системе координат. В связи с тем, что процедура нахождения составляющих напряженности магнитного поля в цилиндрической системе координат теперь является тривиальной, а окончательные выражения весьма громоздки, автор их не приводит.

Отметим, что выражения (10.25)–(10.27) получены абсолютно строго, без использования каких-либо приближений и позволяют проводить вычисления для соленоидов с произвольными параметрами (длиной, поперечным сечением и числом витков). Их анализ показывает, что интегралы в (10.25)–(10.27) всегда отличны от нуля. Следовательно, однослойный соленоид имеет поперечное магнитное поле, аналогичное полю прямого тока.

Так как интегралы (10.25)–(10.27) не берутся в элементарных функциях, целесообразно, задавшись конкретными параметрами обмотки (шагом, числом витков и радиусом поперечного сечения) провести численное вычисление компонент вектора напряженности магнитного поля и провести его визуализацию, как это было описано в предыдущем разделе. Так как далее нас будет интересовать форма силовых линий, но не абсолютное значение напряженностей множитель будем $(\mu_0 / 4\pi)I$ полагать равным единице в (10.25)–(10.27).

Обсудим более подробно особенности вычисления интегралов (10.25)–(10.27). В MATLAB для нахождения значения определенного интеграла от некоторой функции $f(x)$ на интервале $[a,b]$ необходимо выполнить следующую последовательность действий:

1. провести разбиение отрезка $[a,b]$ интегрирования на N отрезков и сохранить координаты их концов в векторе x_i, $i = 1,\ldots,N$;

2. вычислить значения подынтегральной функции $f(x)$ в узловых точках x_i,: $F_i = f(x_i)$;
3. использовать функцию **trapz**, реализующую метод трапеций: **trapz(x,F)**.

Таким образом, для вычисления и визуализации напряженности магнитного поля соленоида нужно:

1. Создать m-файлы, возвращающие значения составляющих напряженности магнитного поля вдоль каждой из координатных осей:

```
% листинг файл Bx.m
function z=Bx(x0,y0,z0,a,h,N)
% функция, возвращающая значение проекции напряженности
% магнитного поля на ось oX
phi=0:pi/50:2*pi*N; % вычисление координат узловых точек
F=f(phi,x0,y0,z0,a,h); % вычисление значений функции в узловых
                       % точках
z=trapz(phi,F);   вычисление значений интеграла


function F=f(psi,x0,y0,z0,a,h)
% описание подынтегральной функции в (10.26)
F=((z0-psi*h/(2*pi))*a.*cos(psi)-h/(2*pi)*(y0-a*sin(psi))./
(x0^2+y0^2+z0^2+a^2+(h/(2*pi)*psi).^2-..
2*(x0*a*cos(psi)+a*y0*sin(psi)+h/(2*pi)*z0*psi)) ^(3/2);


% листинг файл By.m
function z=By(x0,y0,z0,a,h,N)
% функция, возвращающая значение проекции напряженности
% магнитного поля на ось oY
phi=0:pi/50:2*pi*N; % вычисление координат узловых точек
F=f(phi,x0,y0,z0,a,h,I); % вычисление значений функции в узловых
                         % точках
z=trapz(phi,F);   вычисление значений интеграла


function F=f(psi,x0,y0,z0,a,h);
% описание подынтегральной функции в (10.27)
F=((x0-a*cos(psi))*h/(2*pi)+a*sin(psi).*(z0-psi*h/(2*pi))./
(x0^2+y0^2+z0^2+a^2+(h/(2*pi)*psi).^2-..
2*(x0*a*cos(psi)+a*y0*sin(psi)+h/(2*pi)*z0*psi)).^(3/2);
```

```
% листинг файл Bz.m
function z=Bz(x0,y0,z0,a,h,N)
% функция, возвращающая значение проекции напряженности
% магнитного поля на ось oZ
phi=0:pi/50:2*pi*N; % вычисление координат узловых точек
F=f(phi,x0,y0,z0,a,h,I); % вычисление значений функции в узловых
                         % точках
z=trapz(phi,F);   вычисление значений интеграла
```

```
function F=f(psi,x0,y0,z0,a,h);
% описание подынтегральной функции в (10.28)
F=((y0-a*sin(psi))*a.*sin(psi)+a*cos(psi) *(x0-a*cos(psi))) ./
(x0^2+y0^2+z0^2+a^2+(h/(2*pi)*psi).^2-..
2*(x0*a*cos(psi)+a*y0*sin(psi)+h/(2*pi)*z0*psi)) ^(3/2);
```

2. Задать координатную сетку (вычислить координаты узлов сетки).
3. Вычислить в каждом узле сетки значение составляющих напряженностей магнитного поля соленоида.
4. Построить векторное поле единичных векторов, касательных к силовым линиям магнитного поля, выполнив последовательность действий, описанную в предыдущем разделе.

Для построения непрерывных силовых линий магнитного поля, как было показано выше следует вычислить в узлах координатной сетки значения векторного потенциала $\vec{A}(x, y, z)$

$$\vec{A} = \frac{\mu_0 I}{4\pi} \int \frac{d\vec{l}(\varphi)}{\left|\vec{R}_0 - \vec{R}(\varphi)\right|}, \tag{10.29}$$

где $d\vec{l}$, $\vec{R}$ — векторы определенные согласно (10.18)–(10.21), $\vec{R}_0$ — радиус-вектор точки наблюдения, и построить карту эквипотенциальных поверхностей функции $\vec{R}_0$.

Подставляя (10.18)–(10.21) в (10.29), получаем выражения для составляющих векторного потенциала магнитного поля соленоида вдоль соответствующих координатных осей:

$$A_x = \frac{\mu_0 I}{4\pi} \int_0^{2\pi N} \frac{-a\sin\varphi}{\left(R_0^2 + a^2 + \left(\frac{h}{2\pi}\varphi\right)^2 - 2\left(x_0 a\cos\varphi + a y_0 \sin\varphi + \frac{h}{2\pi} z_0\right)\right)^{1/2}} d\varphi; \tag{10.30}$$

$$A_y = \frac{\mu_0 I}{4\pi} \int_0^{2\pi N} \frac{a\cos\varphi}{\left(R_0^2 + a^2 + \left(\frac{h}{2\pi}\right)^2 - 2\left(x_0 a\cos\varphi + a y_0 \sin\varphi + \frac{h}{2\pi} z_0\right)\right)^{1/2}} d\varphi; \tag{10.31}$$

$$A_z = \frac{\mu_0 I}{4\pi}\int_0^{2\pi N} \frac{\frac{h}{2\pi} z_0}{\left(R_0^2 + a^2 + \left(\frac{h}{2\pi}\varphi\right)^2 - 2\left(x_0 a\cos\varphi + a y_0 \sin\varphi + \frac{h}{2\pi} z_0\right)\right)^{1/2}} d\varphi. \quad (10.32)$$

Таким образом, для вычисления и визуализации силовых линий магнитного поля соленоида нужно:

1. Создать m-файлы, возвращающие значения составляющих векторного потенциала магнитного поля вдоль каждой из координатных осей:

```
% листинг файла Ax.m
function z=Ax(x0,y0,z0,a,h,N)
% функция, возвращающая значение проекции векторного потенциала
% магнитного поля на ось oX
phi=0:pi/50:2*pi*N; % вычисление координат узловых точек
F=f(phi,x0,y0,z0,a,h); % вычисление значений функции в узловых
                       % точках
z=trapz(phi,F);   вычисление значений интеграла


function F=f(psi,x0,y0,z0,a,h);
% описание подынтегральной функции в (10.30)
F=-(a*sin(psi))./...
 ((x0-a*cos(psi)).^2+(y0-a*sin(psi)).^2+(z0-h./(2*pi)*psi).^2).
                                                        ^0.5;


% листинг файла Ay.m
function z=Ay(x0,y0,z0,a,h,N)
% функция, возвращающая значение проекции векторного потенциала
% магнитного поля на ось oY
phi=0:pi/50:2*pi*N; % вычисление координат узловых точек
F=f(phi,x0,y0,z0,a,h); % вычисление значений функции в узловых
                       % точках
z=trapz(phi,F);   вычисление значений интеграла


function F=f(psi,x0,y0,z0,a,h);
% описание подынтегральной функции в (10.31)
F=(a*cos(psi))./...
 ((x0-a*cos(psi)).^2+(y0-a*sin(psi)).^2+(z0-h./(2*pi)*psi).^2).
                                                        ^0.5;


% листинг файла Az.m
function z=Az(x0,y0,z0,a,h,N)
% функция, возвращающая значение проекции векторного потенциала
% магнитного поля на ось oZ
phi=0:pi/50:2*pi*N; % вычисление координат узловых точек
```

```
F=f(phi,x0,y0,z0,a,h); % вычисление значений функции в узловых
                       % точках
z=trapz(phi,F);    вычисление значений интеграла


function F=f(psi,x0,y0,z0,a,h);
% описание подынтегральной функции в (10.32)
F=(h/(2*pi))./..
((x0-a*cos(psi)).^2+(y0-a*sin(psi)).^2+(z0-h./(2*pi)*psi).^2)
                                                    ^0.5;
```

2. Задать координатную сетку.
3. Вычислить в каждом узле сетки значение проекций векторного потенциала магнитного поля соленоида на соответствующие координатные оси.
4. Построить карту линий уровня функции $\left|\vec{A}(x, y, z)\right|$.

Для примера на рис. 10.17, 10.18 показаны проекции силовых линий однослойного соленоида с параметрами $a = 0.5$, $L = 6$, $N = 200$, нижний конец которого расположен в плоскости $z = 0$, на плоскости $x = 0$ и $z = 0$, соответственно. Проекция векторного потенциала на плоскость YoZ вычислялась на квадратной координатной сетке со следующими параметрами: левый нижний угол (−5,−5), правый верхний угол (5,5), число узлов $N = 17 \times 17$. Проекция векторного потенциала на плоскость $Z = 3$ вычислялась на прямоугольной координатной сетке со следующими параметрами: левый нижний угол (−5,−5), правый верхний угол (5,8), число узлов $N = 17 \times 17$. На рис. 10.17, 10.18, более толстыми линиями показаны соответствующие сечения соленоида.

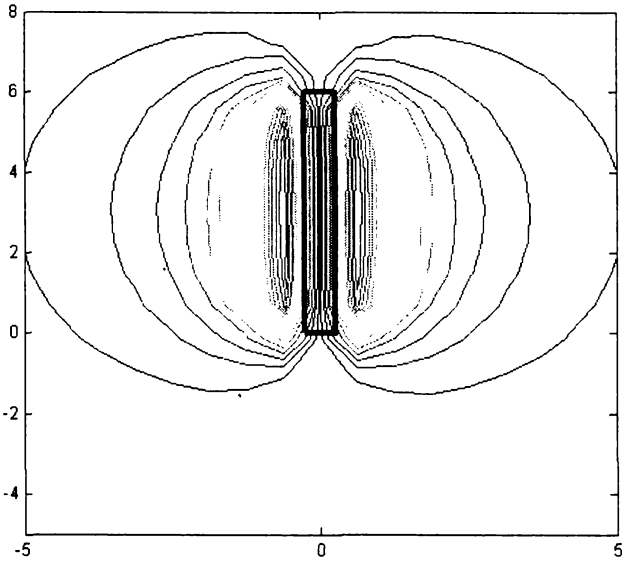


Рис. 10.17. Карта силовых линий соленоида с постоянным током в плоскости *XoZ*

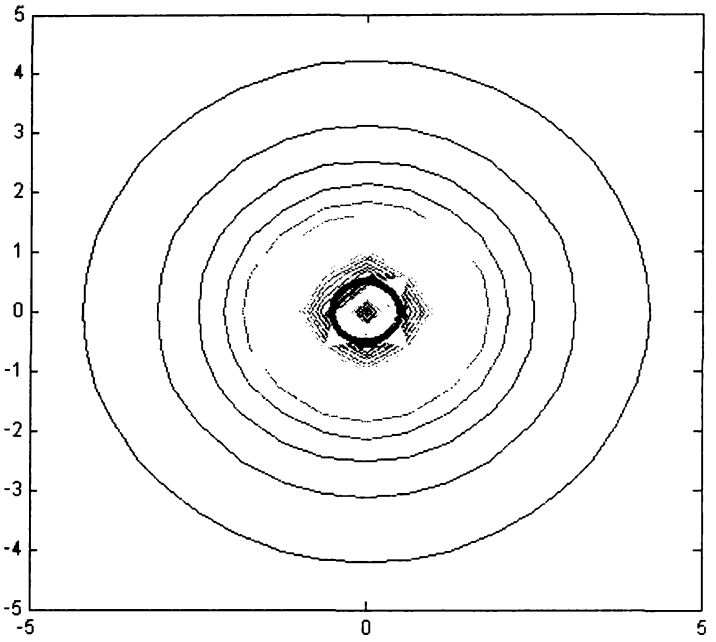


Рис. 10.18. карта силовых линий магнитного поля, создаваемого соленоидов в плоскости *XoY*

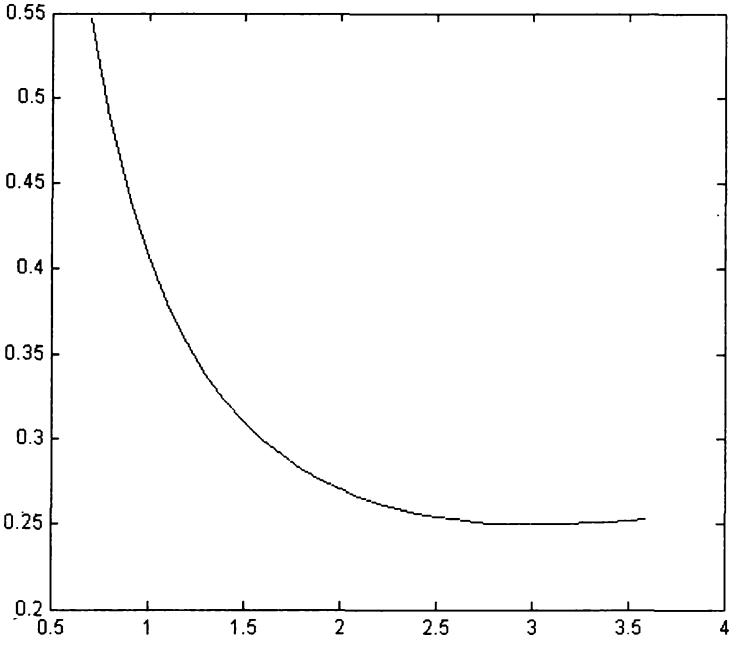


Рис. 10.19. Отношение модулей напряженностей поперечной составляющей магнитного поля к продольной

Для сравнения проекции напряженности магнитного поля на плоскость $Z = 3$ и проекции напряженности магнитного поля на ось oZ вычислено их отношение на различных расстояниях от оси oZ (рис. 10.23). Как очевидно данная величина есть тангенс угла между силовой линией и осью oZ, который характеризует искривленность силовых линий. Как показывает анализ зависимости, представленной на рис. 10.19, у соленоида с указанными размерами составляющая напряженности магнитного поля, лежащая в плоскости $Z = 3$, т.е. достаточно далеко от обоих концов соленоида, оказывается одного порядка с составляющей напряженности, параллельной оси oZ. Следовательно, силовые линии магнитного поля, создаваемые однослойным соленоидом, являются не плоскими, но пространственными кривыми.

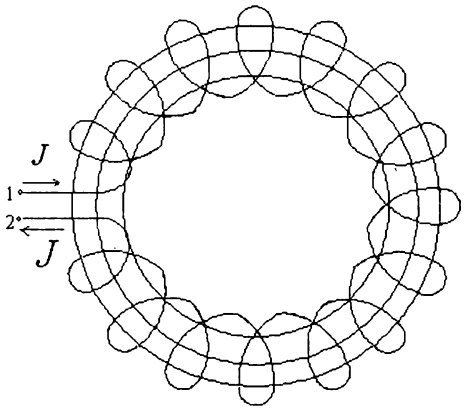


Рис. 10.20. Тороидальная катушка

10.4. Магнитное поле тороидальной обмотки с постоянным током

Для нахождения магнитного поля тороидальной обмотки с постоянным током (рис. 10.20) принято использовать уравнение Максвелла:

$$rot\vec{B} = \vec{j},$$

имеющее в интегральной форме следующий вид [4]:

$$\oint_L \vec{B}d\vec{l} = \sum_{k=1}^{n}(\pm I_k), \qquad (10.33)$$

где L — контур, охватывающий ток, I_k — сила тока, охватываемого контуром.

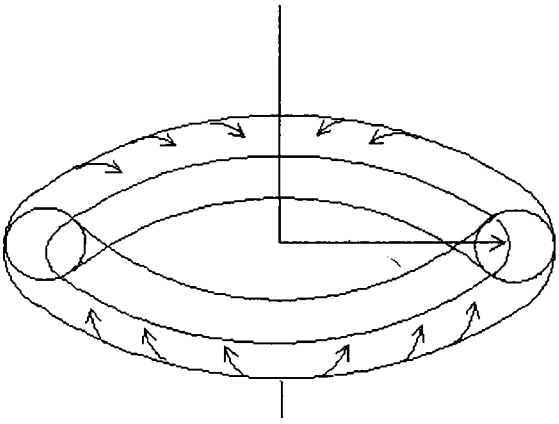


Рис. 10.21. Традиционная модель, используемая при расчете магнитного поля тороида

Следующий шаг в нахождении магнитного поля состоит в замене обмотки тора с протекающим по ней током поверхностным током с постоянной линейной плотностью i (рис. 10.21). При использовании данной модели линии тока предполагаются расположенными в меридиональных плоскостях, проходящих через ось симметрии тора, то есть набором колец с током. Циркуляция магнитного поля тора $\vec{B}$ в (10.28), зависящего, из соображений симметрии, только от расстояния до центра тора, равна $2\pi RB$. Полный ток, пронизывающий площадь, ограниченную этой окружностью, равен NJ, где N — число витков тороидальной обмотки. Откуда напряженность магнитного поля $\vec{B}$ внутри тора равна

$$B = \frac{\mu_0}{2\pi R} NJ.$$

Рассмотрение контура, представляющего собой окружность, радиусом, большим радиуса тора, показывает, что полный ток, протекающий через данный контур, равен нулю, так как ток в каждом витке катушки пересекает ее дважды в противоположных направлениях. Отсюда с неизбежностью следует вывод о равенстве нулю напряженности магнитного поля вне тора.

Однако количественное обоснование принятого выше допущения о возможности замены тороидальной обмотки поверхностно распределенным током либо не приводится, либо утверждается, что составляющей тока вдоль тора при достаточно плотной намотке можно пренебречь [5,16]. Справедливость данных подходов можно поставить под сомнение, так как они противоречат закону сохранения заряда. Действительно, при протекании тока J через тор и подводящие провода количество электричества, проходящего в единицу времени через выбранное сечение провода 1, должно равняться количеству электричества, проходящего через выбранное поперечное сечение провода 2 (рис. 10.22). Следовательно, количество электричества, перемещаемое в единицу времени вдоль окружности радиуса R тора одинаково при любой крутизне намотки. Так как перемещение заряда

вдоль оси тора есть продольный ток, величина которого равна J, то тор должен обладать внешним магнитным полем, аналогичным магнитному полю витка с током.

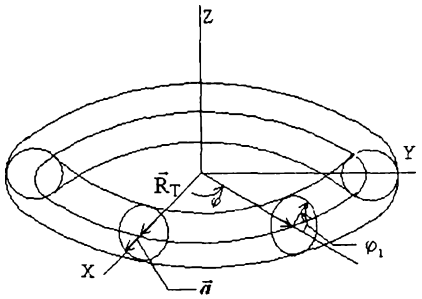


Рис. 10.22. К получению уравнения спирали, намотанной на тор

Для подтверждения проведенного качественного доказательства существования внешнего магнитного поля тора проведем прямые вычисления напряженности магнитного поля на основе закона Био-Савара-Лапласа [4]:

$$d\vec{B} = \frac{\mu_0 J}{4\pi} \frac{\left[d\vec{l} \times \vec{r}\right]}{r^3}, \tag{10.34}$$

где $Jd\vec{l}$ — элемент тока, $\vec{r}$ — радиус вектор от элемента тока в точку наблюдения, и принципа суперпозиции — интегрированием (10.34) по всем элементам тока.

Для однозначного описания положения элементов намотки введем два радиус-вектора — вектор $\vec{R}_T$, описывающий положение центра данного сечения тора меридиональной плоскостью, с началом в точке O, и вектор $\vec{a}$, описывающий положение элемента спирали в меридиональной плоскости (рис. 10.22). Выберем систему координат XYZ так, что при нулевом угле поворота вектор $\vec{R}_T$ имеет координаты $(R_T,0,0)$, вектор $\vec{a}$ — (a,0,0). Положение каждой точки тороидальной намотки в выбранной системе координат будем описывать радиус-вектором $\vec{r}$, который является суммой векторов $\vec{R}_T$, $\vec{a}$:

$$\vec{r} = \vec{R} + \vec{a}. \tag{10.35}$$

Очевидно, что при постоянном угле намотки положение любой точки спирали будет определяться одним параметром — углом поворота φ вектора $\vec{R}_T$ в плоскости XoY. Найдем связь угла поворота вектора $\vec{a}$ в меридиональной плоскости φ_1 с углом поворота φ вектора $\vec{R}_T$ в плоскости XoY, для чего рассмотрим плоскостную развертку одного витка тороидальной намотки, представляющую прямоугольный треугольник, длины катетов которого равны длинам дуг окружности радиуса R в плоскости XoY и окруж-

ности радиуса a в меридиональной плоскости (рис. 10.23). При N-витковой намотке они равны $R_T \quad 2\pi / N$ и $2\pi a$, соответственно, поэтому угол намотки β равен

$$\beta = \arg tg \frac{aN}{R_T}. \tag{10.36}$$

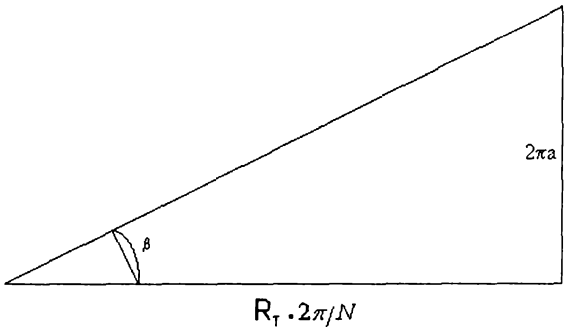


Рис. 10.23. К вычислению угла наклона тороидальной обмотки

Рассмотрим развертку спирали при произвольном угле поворота φ, представляющую, как и для одного витка, прямоугольный треугольник. В этом случае длины соответствующих катетов равны $R_T \quad \varphi$, $a \cdot \varphi_1$. Используя известные формулы, связывающие катеты прямоугольного треугольника через угол между катетом и гипотенузой, получим

$$R_T \cdot \varphi \cdot tg\beta = a \cdot \varphi_1. \tag{10.37}$$

Подставляя выражение для угла β из (10.36) в (10.37), окончательно найдем

$$\varphi_1 = N\varphi. \tag{10.38}$$

Учитывая (10.35), (10.37), запишем зависимость координат радиус-вектора $\vec{r}$ от угла поворота φ в выбранной системе координат

$$\vec{r} = \left(\left(R_T + a\cos N\varphi\right)\cos\varphi, \left(R_T + a\cos N\varphi\right)\sin\varphi, a\sin N\varphi\right). \tag{10.39}$$

Элемент длины тороидальной обмотки $d\vec{l}$ выражается через вектор $\vec{t}$, касательный к обмотке в данной точке, и углом поворота $d\varphi$

$$d\vec{l} = \vec{t}\,d\varphi, \tag{10.40}$$

где вектор $\vec{t}$ по определению равен [15]

$$\vec{t} = \frac{d\vec{r}}{d\varphi}. \tag{10.41}$$

Из (10.39), (10.41) найдем

$$\vec{t} = \begin{bmatrix} -(R_T \sin\varphi + a\sin\varphi\cos N\varphi + Na\sin N\varphi\cos\varphi) \\ R_T\cos\varphi - aN\sin N\varphi\sin\varphi + a\cos N\varphi\cos\varphi \\ aN\cos N\varphi \end{bmatrix}. \tag{10.42}$$

Выражение для напряженности магнитного поля, создаваемого элементом тороидальной обмотки в точке пространства, положение которой описывается радиус-вектором $\vec{R}_0 = (x_0, y_0, z_0)$, учитывая (10.34), (10.40) можно записать в виде:

$$d\vec{B} = \frac{\mu_0 J}{4\pi}\frac{\left[\vec{t}\times(\vec{R}_0 - \vec{r})\right]}{\left|\vec{R}_0 - \vec{r}\right|}d\varphi, \tag{10.43}$$

где

$$\vec{R}_0 - \vec{r}_0 = \to$$
$$\to \left(x_0 - (R_T + a\cos N\varphi)\cos\varphi,\ y_0 - (R_T + a\cos N\varphi)\sin\varphi,\ z_0 - a\sin N\varphi\right), \tag{10.44}$$

$$\left|\vec{R}_0 - \vec{r}_0\right| = \to$$
$$\sqrt{\left(x_0 - (R_T + a\cos N\varphi)\cos\varphi\right)^2 + \left(y_0 - (R_T + a\cos N\varphi)\sin\varphi\right)^2 + \ldots} \to$$
$$\to \overline{\left(z_0 - a\sin N\varphi\right)^2} \tag{10.45}$$

Составляющие магнитного поля, создаваемого тороидальной обмоткой с постоянным током в точке с радиус-вектором $\vec{R}_0$, вдоль координатных осей находятся непосредственным интегрированием (10.43). Окончательные выражения компонентов напряженности магнитного поля через интегралы элементарно получаются подстановкой выражений (10.34), (10.45), (10.42) в (10.43)

$$B_x = -\frac{\mu_0 I}{4\pi}\int_0^{2\pi}\ldots \to$$
$$\to \frac{(R_T\cos\varphi - aN\sin N\varphi\cos\varphi + a\cos N\varphi\cos\varphi)(z_0 - a\sin N\varphi) + \ldots}{\left[\sqrt{\left(x_0 - (R_T + a\cos N\varphi)\cos\varphi\right)^2 + \left(y_0 - (R_T + a\cos N\varphi)\sin\varphi\right)^2 + \ldots}\right.} \to$$
$$\to \frac{-aN\cos N\varphi(y_0 - R_T + a\cos N\varphi)\sin\varphi}{\left.\overline{+\left(z_0 - a\sin N\varphi\right)^2}\right]^3}d\varphi \tag{10.46}$$

$$B_y = -\frac{\mu_0 I}{4\pi}\int_0^{2\pi} \ldots \rightarrow$$

$$\rightarrow \frac{(R_T \sin\varphi + a\sin\varphi\cos N\varphi + Na\sin N\varphi\cos\varphi)(z_0 - a\sin N\varphi)+\ldots}{\left[\sqrt{(x_0 - (R_T + a\cos N\varphi)\cos\varphi)^2 + (y_0 - (R_T + a\cos N\varphi)\sin\varphi)^2}+\ldots\right.} \rightarrow$$

$$\rightarrow \frac{-aN\cos N\varphi(x_0 - (R_T + a\cos N\varphi)\cos\varphi)}{\left.+(z_0 - a\sin N\varphi)^2\right]^3} d\varphi \quad (10.47)$$

$$B_z = -\frac{\mu_0 I}{4\pi}\int_0^{2\pi} \rightarrow$$

$$\rightarrow \frac{(R_T \sin\varphi + a\sin\varphi\cos N\varphi + Na\sin N\varphi\cos\varphi)\cdot\ldots}{\left[\sqrt{(x_0 - (R_T + a\cos N\varphi)\cos\varphi)^2 + (y_0 - (R_T + a\cos N\varphi)\sin\varphi)^2}+\ldots\right.} \rightarrow$$

$$\rightarrow \frac{(y_0 - (R_T + a\cos N\varphi)\sin\varphi)}{\left.+(z_0 - a\sin N\varphi)^2\right]^3} d\varphi - \ldots \rightarrow$$

$$\rightarrow \frac{\mu_0 I}{4\pi}\int_0^{2\pi} \frac{aN\cos N\varphi\cdot\ldots}{\left[\sqrt{(x_0 - (R_T + a\cos N\varphi)\cos)^2 + (y_0 - (R_T + a\cos N\varphi)\sin\varphi)^2}+\ldots\right.} \rightarrow$$

$$\rightarrow \frac{(x_0 - (R_T + a\cos N\varphi)\cos\varphi)}{\left.+(z_0 - a\sin N\varphi)^2\right]^3} d\varphi \quad (10.48)$$

Как видно из выражений (10.46)–(10.48), составляющие магнитного поля тороидальной обмотки с постоянным током не выражаются через элементарные функции, поэтому возможно только численное исследование особенностей этого поля.

Важно отметить, что несмотря на громоздкость выражений (10.46)–(10.48), вычислительные средства MATLAB позволяют записать решение данной задачи в компактной форме. При составлении программы можно использовать подход, описанный в предыдущем разделе при расчете и визуализации магнитного поля соленоида с постоянным током. Однако опыт работы с соответствующей программой показал, что для расчета напряженности магнитного поля соленоида даже для сетки с небольшим числом узлов требуется неприемлемо много времени. С помощью пошаговой прогонки программы было установлено, что наибольшие временные затраты требует операция вычисления векторного произведения, входящего в выражение (10.21), которое приходится вычислять многократно для каждого значения переменной **phi**. По этой причине мы отказались от использования встроенной в MATLAB функции **cross**, записывая явные выражения для каждой координаты векторного произведения. Ниже приводится листинг файла **A_Tor.m**, содержащий описание функции, возвращающей зна-

чения составляющих напряженностей вдоль соответствующих координатных осей.

```
% листинг файла A_Tor.m
function [Ax,Ay,Az]=A_Tor(x0,y0,z0,Rt,a,N);
% функция, возвращающая значение составляющих напряженности
% магнитного поля вдоль соответствующих координатных осей
dphi=pi/(5*N); % шаг интенгрирования
phi=0:dphi:2*pi; % вектор, содержащий значения узлов
                 % интегрирования
[rx ry rz]=r_Tor(Rt,a,N,phi); % вычисление координат вектора R
                              % в соответствие с (10.39) в узлах
                              % интегрирования
[tx ty tz]=t_Tor(Rt,a,N,phi); % вычисление координат вектора t,
                              % в соответствие с (10.42)
N2=length(rx); % длина вектора
% вычисление составляющих напряженности магнитного поля
% в соответствии с (10.22)
X0(1:N2)=x0;
Y0(1:N2)=y0;
Z0(1:N2)=z0;
Rx=X0-rx;
Ry=Y0-ry;
Rz=Z0-rz;
Z=(Rx.^2+Ry.^2+Rz.^2).^(1/2);
f1=tx./Z;
f2=ty./Z;
f3=tz./Z;
Ax=trapz(phi,f1);
Ay=trapz(phi,f2);
Az=trapz(phi,f3);



function [rx,ry,rz]=r_Tor(Rt,a,N,phi)
% функция, возвращающая координаты вектора R в соответствие
% с (10.39)
rx=(Rt+a*cos(N*phi)).*cos(phi);
ry=(Rt+a*cos(N*phi)).*sin(phi);
rz=a*sin(N*phi);



function [tx,ty,tz]=t_Tor(Rt,a,N,phi)
% функция, возвращающая координаты вектора t в соответствие
% с (10.42)
tx=-(Rt*sin(phi)+a*sin(phi).*cos(N*phi)+a*N*sin(N*phi).*cos(phi));
ty=Rt*cos(phi)-N*a*sin(phi) *sin(N*phi)+a*cos(N*phi).*cos(phi);
tz=a*N*cos(N*phi);
```

Далее для вычисления и визуализации магнитного напряженности магнитного поля в плоскости *YoZ* необходимо выполнить следующую последовательность команд.

```
N1=21; % число узлов
i=1:N1+1;
Ymin=-100; Zmin=-100;   левый нижний угол координатной сетки
```

```
Ymax=100; Zmax=100; % правый верхний угол координатной сетки
% вычисление координат узлов сетки
Y(i)=Ymin+(Ymax-Ymin)/N1*(i-1);
Z(i)=Zmin+(Zmax-Zmin)/N1*(i-1);
Rt=1; % радиус большого кольца тора
a=0.5; % радиус малого кольца тора
N=100; % число витков
% вычисление составляющих напряженности магнитного поля в узлах
% координатной сетки
for i=1:N1+1
 for j=1:N1+1
  [bx by bz]=B_Tor(0,Y(i),Z(j),Rt,a,N);
  Bx(i,j)=bx;
  By(i,j)=by;
  Bz(i,j)=bz;
 end;
end;
% нормировка векторов
mp=(By.^2+Bz.^2).^0.5;
by1=By./mp;
bz1=Bz./mp;
quiver(Y,Z,by1,bz1,0.5);   визуализация векторного поля
```

Результаты выполнения описанной последовательности команд представлены на рис. 10.24.

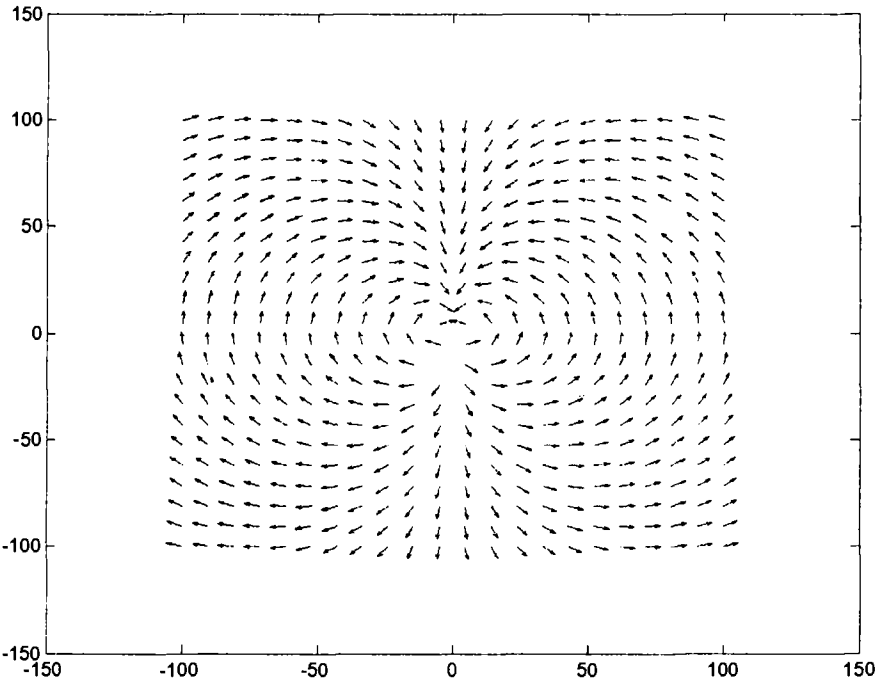


Рис. 10.24. Визуализация поля единичных векторов, касательных к силовым линиям магнитного поля тороидальной обмотки? в плоскости *YoZ*

Отметим, что здесь, как и в случае с прямым соленоидом, использование многослойной тороидальной обмотки с числом слоев n, намотанной в одном направлении, приведет к пропорциональному увеличению (в n раз) количества электричества, перемещающегося в единицу времени вдоль окружности тора радиуса R, по сравнению с однослойной обмоткой. Следовательно, поле многослойной тороидальной обмотки B будет в n раз больше поля однослойной обмотки:

$$B_1 = nB.$$

При использовании многослойной тороидальной обмотки, в которой направление намотки в каждом следующем слое противоположно предыдущему, при четном количестве слоев суммарное количество электричества, перемещающегося в единицу времени вдоль окружности тора радиуса R, равно нулю, при нечетном количестве слоев — равно количеству электричества, перемещающегося в единицу времени вдоль оси радиуса R, в однослойной тороидальной обмотке. Следовательно, тороидальная обмотка с четным количеством слоев обмотки не имеет внешнего магнитного поля, а тороидальная обмотка с нечетным числом слоев имеет внешнее магнитное поле равное полю однослойной тороидальной обмотки.

Вопросы для самопроверки

1. Какими количественными характеристиками описывается электростатическое поле, создаваемое системой точечных электрических зарядов?
2. Какой алгоритм используется для вычисления и визуализации потенциала электростатического поля?
3. Какие алгоритмы используются для вычисления и визуализации напряженности электростатического поля?
4. Какой алгоритм используется для расчета и визуализации напряженности магнитного поля, создаваемого витком с током?
5. Какая модель используется для расчета магнитного поля соленоида с постоянным током?
6. Какой алгоритм используется для визуализации силовых линий магнитного поля соленоида с постоянным током?
7. Какая модель используется для расчета магнитного поля тороидальной обмотки с постоянным током?

Литература к главе 10

1. Морс Ф.М, Фешбах Г. Методы теоретической физики. Т. 1. М.: ИЛ, 1958.
2. Калашников С.Г. Электрические и магнитные поля. М.: Наука, 1970.
3. Фриш С.Э., Тиморева А.В. Курс общей физики. М.: Физматгиз, 1961. Т. II.
4. Фейман Р., Лейтон Р., Сэндс М. Феймановские лекции по физике. Вып. 5,6.: Наука, 1977.
5. Парселл Э. Электричество и магнетизм. Берклеевский курс физики. М.: Наука, 1971.
6. Говорков В.А. Электрические и магнитные поля. Л.: Гостехиздат, 1960.
7. Тамм И.Е. Основы теории электричества. М.: Наука, 1966.
8. Смайт В. Электростатика и электродинамика. М.: Иностранная литература, 1954.

9. Стрэттон Дж.А. Теория электромагнетизма. М.: Иностранная литература, 1948.
10. Джексон Дж. Классическая электродинамика. М.: Иностранная литература, 1961.
11. Зоммерфельд А. Электродинамика. М.: ИЛ, 1958.
12. Сборник задач по общему курсу физики. Электричество и магнетизм./Сб. задач под ред. И.А. Яковлева. М.: Наука, 1977.
13. Батыгин В.В., Топтыгин И.Н. Сборник задач по электродинамике. М.: Наука, 1970.
14. Векштейн Е.Г. Сборник задач по электродинамике. М.: Наука, 1966.
15. Смирнов В.И. Курс высшей математики. М.: Физматгиз, 1958. Т.3.

Глава 11

Построение фрактальных объектов в MATLAB

Фракталы — математические объекты дробной размерности, название которых было введено в математику Б. Мандельбротом [1], являются в настоящее время как предметом самостоятельных математических исследований, так и инструментарием, используемым в целом ряде прикладных задач нелинейной динамики [2,3], теории хаоса [4], обработки сигналов [5]. Однако только относительно недавно появилось первое полноценное учебное пособие по новой быстро развивающейся математической дисциплине [6], основой которого стал учебный курс, преподававшийся автором в течении ряда лет в университете Миссури-Колумбия.

Так как при изучении фракталов и хаоса большую роль играет компьютерное моделирование, в курсе предусмотрено параллельное изучение теоретических вопросов и проведение компьютерных экспериментов. Это отличает его структуру от традиционной структуры большинства математических курсов: теорема–доказательство–пример–задача. В [6] в обобщенном виде подробно описаны известные алгоритмы построения фрактальных объектов (L-системы и терл-графика, аффинные преобразования, системы итерированных функций, случайные системы итерированных функций и др.). Однако соответствующих программ, созданных на каком-либо языке программирования или в математическом пакете, не приводится. В тоже время, опыт практической реализации алгоритмов построения фрактальных объектов, описанных в [6], в каком-либо из современных математических пакетов (MATLAB, Mathcad, Maple, Matematica и т.д.), широко используемых в настоящее время в преподавании целого ряда физико-математических дисциплин (см., например, [7]), показывает, что существует необходимость внесения в них определенных корректировок, учитывающих особенности выбранного пакета (в первую очередь графические). В данной главе обсуждаются алгоритмы построения классических фракталов и их программные реализации, созданные в MATLAB.

11.1. Рекурсивный алгоритм построения фрактальных объектов

Обсуждение алгоритма, основанного на использовании рекурсивной функции, проведем на примере построения простого самоподобного фрактала — ковра Серпинского. В рассматриваемом в данном разделе алгоритме используется способ построения, основанный на последовательном удалении из начальной области внутренних подъобластей в соответствии с заданными правилами. Выберем в качестве начального множество s_0 — равносторонний треугольник вместе с областью которую он замыкает. Удалим внутренность центральной треугольной области и назовем оставшееся множество s_1 (рис. 11.1). Затем повторим описанный процесс для каждого из трех оставшихся треугольников и получим следующее приближение s_1. Продолжая таким образом, получим последовательность вложенных множеств s_n, пересечение которых и образует ковер Серпинского s (рис. 11.1). Из построения видно, что ковер является объединением N=3 существенно непересекающихся уменьшенных в два раза копий (коэффициенты подобия по горизонтали и вертикали в данном случае оказываются одинаковыми $r = \frac{1}{2}$). Фрактальная размерность ковра Серпинского d равна

$$d = \frac{\log(3)}{\log(2)} \approx 1.580.$$

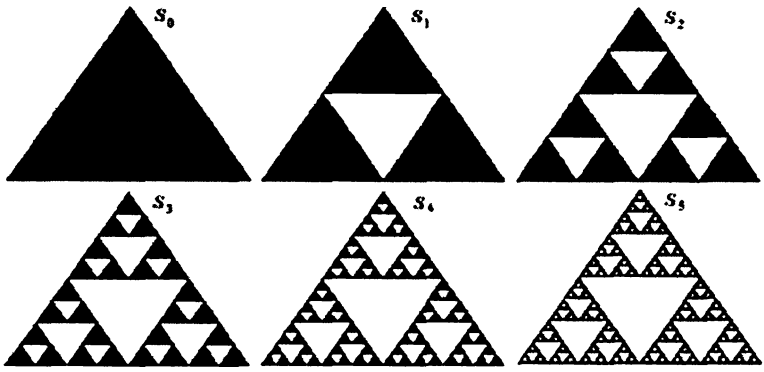


Рис. 11.1. Последовательность построения ковра Серпинского

Для построения рассматриваемого фрактала, как очевидно, можно использовать следующий алгоритм:

1. Задать порядок ковра N.
2. Задать координаты вершин исходного треугольника ABC: (X_A,Y_A), (X_B,Y_B), (X_C,Y_C).

3. Построить равносторонний треугольник ABC и залить его черным цветом.
4. Вычислить координаты середин сторон треугольника ABC:

$$dx = \frac{X_B - X_A}{2},\ dy = \frac{Y_B - Y_A}{2},$$

$$X_{A'} = X_A + dx,\ Y_{A'} = Y_A + dy,$$

$$X_{B'} = X_B + dx + dx/2,\ Y_{B'} = Y_B + dy,$$

$$X_{C'} = X_C + dx/2,\ Y_{C'} = Y_C + dy.$$

5. Построить треугольник $A'B'C'$ и залить его белым цветом.
6. Повторить N раз действия, описанные в пп. 4,5 для треугольников $AA'C'$, $A'BB'$, $C'B'C$, соответственно.

Наиболее просто описанный выше алгоритм можно реализовать при использовании рекурсивной процедуры, выполняющей последовательность действий, описанных в пп. 4,5. Для реализации алгоритма в MATLAB следует создать специальную функцию, возвращающую изображение ковра Серпинского, используя для этого встроенный текстовый редактор MATLAB или любой другой текстовый редактор (например, «Блокнот»), и сохранить текст в файле Serpinsky.m

```
% Листинг файла Serpinsky.m
function z = Serpinsky(Lmax)
% функция, возвращающая изображение ковра Серпинского
% Lmax - порядок ковра
% задание координат вершин равнобедренного треугольника
x1=0; y1=0; x2=1; y2=0; x3=0.5; y3=sin(pi/3);
h=figure(1); % инициализация графического окна
hold on; % включение режима рисования фигур в одном графическом
         % окне
fill([x1 x2 x3],[y1 y2 y3],'k'); % прорисовка равностороннего
                                 % треугольника
set(gca,'xtick',[],'ytick',[]); % отключение режима оцифровки
                                % осей
set(gca,'XColor','w','YColor','w'); % установка цвета рисования
                                    % осей
Simplex(x1,y1,x2,y2,x3,y3,0,Lmax); % обращение к функции,
                                % прорисовывающей равносторонние
                                % треугольники белого цвета
hold off; % отключение режима рисования фигур в одном
          % графическом окне
function z=Simplex(x1,y1,x2,y2,x3,y3,n,Lmax)
% рекурсивная функция, прорисовывающая равносторонние
% треугольники белого цвета
if n<Lmax
% задание координат вершин текущего равностороннего треугольника
    dx=(x2-x1)/2;
    dy=(y3-y1)/2;
    x1n=x1+dx;
    y1n=y1;
    x2n=x1+dx+dx/2;
```

```
    y2n=y1+dy;
    x3n=x1+dx/2;
    y3n=y1+dy;
    fill([x1n x2n x3n],[y1n y2n y3n],'w'); % прорисовка текущего
                                           % равностороннего треугольника
    n=n+1;
  рекурсия
    Simplex(x1,y1,x1n,y1n,x3n,y3n,n,Lmax);
    Simplex(x1n,y1n,x2,y2,x2n,y2n,n,Lmax);
    Simplex(x3n,y3n,x2n,y2n,x3,y3,n,Lmax);
end
```

Для вывода изображения ковра Серпинского, например пятого порядка, следует ввести в командной строке пакета MATLAB имя функции с соответствующим значением:

```
>> Serpinsky(5);
```

Результат, возвращаемый функцией Serpinsky(°), представлен на рис. 11.1 (множество S_5).

Описанный выше алгоритм, легко обобщается для фрактальных объектов, правила построения которых аналогичны правилам построения треугольного ковра Серпинского. Например, алгоритм визуализации фрактального объекта, основные этапы построения которого представлены на рис. 11.2, реализуется следующей последовательностью действий:

1. Задать порядок ковра N.
2. Задать координаты вершин исходного квадрата $ABCD$: (X_A, Y_A), (X_B, Y_B), (X_C, Y_C), (X_D, Y_D).
3. Построить квадрат $ABCD$ и залить его черным цветом.
4. Вычислить координаты точек, делящих стороны квадрата $ABCD$ на три равные части:

$$dx = (X_B - X_A)/3,\ dy = (Y_B - Y_A)/3,$$

$$X_{A'} = X_A + dx,\ Y_{A'} = Y_A + dy,$$

$$X_{C'} = X_A + dx + dx,\ Y_{C'} = Y_A + dy + dy,$$

$$X_{D'} = X_A + dx,\ Y_{D'} = Y_A + dy + dy,$$

5. Построить квадрат $A'B'C'D'$ и залить его белым цветом.
6. Повторить N раз действия, описанные в пп. 4,5 для квадратов с вершинами, имеющими следующие координаты:

$$(X_A, Y_A), (X_{A'}, Y_A), (X_{A'}, Y_{A'}), (X_A, Y_{A'});$$

$$(X_{A'}, Y_A), (X_{B'}, Y_A), (X_{B'}, Y_{B'}), (X_{A'}, Y_{B'});$$

$$(X_{B'}, Y_A), (X_B, Y_B), (X_B, Y_{B'}), (X_{B'}, Y_{B'});$$

$$(X_{B'}, Y_{B'}), (X_B, Y_{B'}), (X_B, X_{C'}), (X_{C'}, Y_{C'});$$

$$(X_{C'}, Y_{C'}), (X_B, Y_{C'}), (X_C, Y_C), (X_{C'}, Y_{C'});$$

$$(X_{D'}, Y_{D'}), (X_{C'}, Y_{C'}), (X_{C'}, Y_{C'}), (X_{D'}, X_{C'});$$

$$(X_A, Y_{D'}), (X_{D'}, Y_{D'}), (X_{D'}, Y_C), (X_D, Y_D);$$

$$(X_A, Y_{A'}), (X_{A'}, Y_{D'}), (X_{D'}, Y_{D'}), (X_A, Y_{D'}),$$

соответственно.

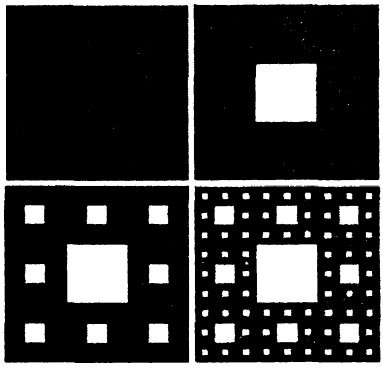


Рис. 11.2. Квадрат Серпинского

Ниже приведен листинг файла Serpinsky2.m, который содержит описание функции, возвращающей изображение квадратного ковра Серпинского, представленного на рис. 11.2.

```
% листинг фалайла Serpinsky2.m
function z=Serpinsky2(Lmax)
% функция, возвращающая изображение квадратного ковра Серпинского
% Lmax - порядок ковра
% Задание координат вершин исходного квадрата
x1=0;y1=0;
x2=1; y2=0;
x3=1;y3=1;
x4=0;y4=1;
figure(1); % создание графического окна
hold on; % включение режима наложения изображений
% задание параметров отображения квадратов
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
```

```
fill([x1 x2 x3 x4],[y1 y2 y3 y4],'k');
Quadrate(x1,y1,x2,y2,x3,y3,x4,y4,0,Lmax);


z=Quadrate(x1,y1,x2,y2,x3,y3,x4,y4,n,Lmax)
% рекурсивная функция, прорисовывающая квадраты белого цвета
if n<Lmax
   dx=(x2-x1)/3;dy=(y3-y1)/3;
   x1n=x1+dx; y1n=y1+dy;
   x2n=x1+dx+dx;y2n=y1+dy;
   x3n=x1+dx+dx;y3n=y1+dy+dy;
   x4n=x1+dx; y4n=y1+dy+dy;
   fill([x1n x2n x3n x4n],[y1n y2n y3n y4n],'w');
   n=n+1;
   Quadrate(x1,y1,x1+dx,y1,x1+dx,y1+dy,x1,y1+dy,n,Lmax);
   Quadrate(x1+dx,y1,x1+2*dx,y1,x1+2*dx,y1+dy,x1+dx,
                                              y1+dy,n,Lmax);
   Quadrate(x1+2*dx,y1,x2,y1,x2,y1+dy,x1+2*dx, y1+dy,n,Lmax);
   Quadrate(x1+2*dx,y1+dy,x2,y1+dy,x2,y1+2*dy,x1+2*dx,
                                            y1+2*dy,n,Lmax);
   Quadrate(x1+2*dx,y1+2*dy,x2,y1+2*dy,x2,y3,x1+2*dx,
                                              y3,n,Lmax);
   Quadrate(x1+dx,y1+2*dy,x1+2*dx,y1+2*dy,x1+2*dx,y4,x1+dx,
                                              y4,n,Lmax);
   Quadrate(x1,y1+2*dy,x1+dx,y1+2*dy,x1+dx,y4,x1, y4,n,Lmax);
   Quadrate(x1,y1+dy,x1+dx,y1+dy,x1+dx,y1+2*dy,x1,
                                            y1+2*dy,n,Lmax);
end
```

Еще одним примером фрактального объекта, для построения которого оказывается удобным использовать рекурсивный алгоритм, является кривая Коха. Построение данной кривой начинается с отрезка K_0 единичной длины. Удалим из отрезка K_0 отрезок длиной 1/3 и добавим два новых отрезка такой же длины, как показано на рис. 11.3. Назовем полученное

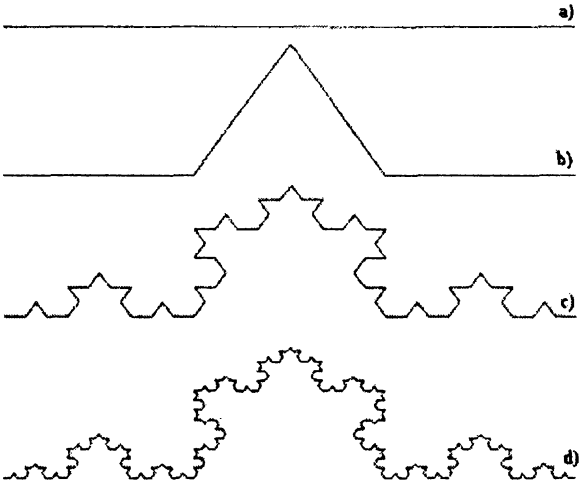


Рис. 11.3. Построение кривой Коха

множество K_1. На следующем шаге разделим, каждый отрезок длиной 1/3 на три части длиной 1/9 и повторим описанную процедуру, заменяя на каждом шаге среднюю ветвь двумя новыми отрезками. Обозначим через K_n фигуру, получившуюся после n-го шага. Можно строго доказать [6], что последовательность кривых $\{K_n\}_{n=1}^{\infty}$ сходится к предельной кривой бесконечной длины, фрактальная размерность K которой равна

$$d = \frac{\log(4)}{\log(3)} \approx 1.2618.$$

Ниже приводится листинг рекурсивной функции **Koch(°)**, возвращающей изображение кривой Коха.

```
function z=Koch(N)
% функция, возвращающая изображение кривой Коха
x1=0; y1=0; % левая точка начального отрезка
x2=1; y2=0; % правая точка начального отрезка
figure(1);
axis([0 1 0 1]);
hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w')
Coord(x1,y1,x2,y2,N); % вызов рекурсивной функции,
                      % прорисовывающей кривую Коха
function z=Coord(x1,y1,x2,y2,n)
% рекурсивная функция, прорисовывающая кривую Коха
if n>0
% вычисление координат концов отрезков на очередном шаге
% рекурсии
dx=(x2-x1)/3; dy=(y2-y1)/3;
x1n=x1+dx; y1n=y1+dy;
x2n=x1+2*dx; y2n=y1+2*dy;
xmid=dx/2-dy*sin(pi/3)+x1n;
ymid=dy/2+dx*sin(pi/3)+y1n;
% рекурсия
Coord(x1,y1,x1n,y1n,n-1);
Coord(x1n,y1n,xmid,ymid,n-1);
Coord(xmid,ymid,x2n,y2n,n-1);
Coord(x2n,y2n,x2,y2,n-1);
else
r1=[x1 y1]; r2=[x2 y2]; R=cat(1,r1,r2);
plot(R(:,1),R(:,2),'Color','k'); % построение кривой Коха
end;
```

Изображение кривой Коха пятого порядка K_5, возвращенное описанной выше функцией, представлено на рис. 11.4.

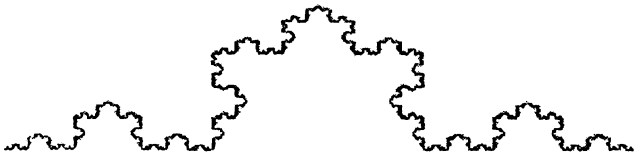


Рис. 11.4. Кривая Коха пятого порядка

11.2. L-системы и терл-графика

Понятие «L-система» было введено А. Лидермайером в 1968 г. при изучении формальных языков. С их помощью оказывается возможным строить как многие известные самоподобные фракталы, например, снежинку Коха, ковер Серпинского, кривые Пеано, Гильберта, Серпинского и др., так и дают возможность создавать бесконечное разнообразие новых фракталов, укладывающихся в данную схему. Изложение понятий «L-система» и «Терл-графика» в данном разделе следует [6].

Терл-графика (от turtle — черепашка) является подсистемой вывода графического представления фрактального объекта. Основным исполнителем данной системы является черепашка (точка), которая перемещается по экрану дискретными шаги, прочерчивая или не прочерчивая свой след. «Мгновенное» положение черпашки задается тремя параметрами (x,y,α), где (x,y) — координаты черепашки, α — направление следующего шага (угол, отсчитываемый от положительного направления оси oX). Последовательность команд, определяющая направление перемещения и действия черпашки, задается кодовым словом, буквы которого читаются слева направо. Кодовое слово, представляющее собой результат работы L-системы, может включать в себя следующие буквы:

- F — переместиться на один шаг вперед, прорисовывая след;
- b — переместиться на один шаг вперед, не прорисовывая след;
- [— открыть ветвь;
-] — закрыть ветвь;
- + — увеличить угол α на величину Θ;
- − — уменьшить угол α на величину Θ;
- X, Y — вспомогательные переменные.

Размер шага и величина приращения по углу Θ задаются изначально и остаются неизменными для всех перемещений черепашки.

Формально, детерминированная L-система состоит из алфавита, слова инициализации, называемого аксиомой или инициатором, и набора порождающих правил, указывающих как следует преобразовать слово при каждой следующей итерации. Например, L-система, соответствующая снежинке Коха, представленной на рис. 11.5, задается следующим образом:

аксиома: $F + + F + + F$

порождающее правило: $Newf = F - F + + F - F$

$\Theta = \pi / 3$

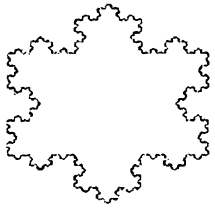


Рис. 11.5. Снежинка Коха

Как очевидно, графическим представлением аксиомы $F++F++F$ является равносторонний треугольник. Черепашка делает один шаг вперед, затем угол увеличивается на $2\pi/3$ и черепашка делает еще один шаг вперед, далее угол вновь увеличивается на $2\pi/3$ и черепашка делает еще один шаг.

На первом шаге каждая буква F в слове-инициаторе заменяется на слово *Newf*:

$$(F-F++F-F)++(F-F++F-F)++(F-F++F-F),$$

из которого, опустив скобки, получаем:

$$F-F++F-F++F-F++F-F++F-F++F-F++F-F.$$

Повторяя этот процесс, на втором шаге получим:

$$\begin{array}{l} F-F++F-F-F-F++F-F++ \\ F-F++F-F-F-F++F-F++ \\ F-F++F-F-F-F++F-F++ \\ F-F++F-F-F-F++F-F++ \\ F-F++F-F-F-F++F-F \end{array}$$

и т.д.

В MATLAB наиболее просто реализовать L-систему, используя рекурсивную функцию. Ниже приведен пример рекурсивной функции **RuleKoch(°)**, возвращающей правила, порождающие снежинку Коха, представленную на рис. 11.5.

```
function z=RuleKoch(Lmax,Axiom,Newf,n,tmp);
% функция, возвращающая L-систему для снежинки Коха
% Входные параметры:
% Lmax - порядок снежинки
% Axiom - строка, содержащая аксиому
% Newf - строка, содержащая порождающее правило
% tmp - входная L-система
while n<=Lmax
 if n= =1
   tmp=Axiom; n=n+1;
 else
   tmp=strrep(tmp,'F',Newf); % замена всех букв F на
                             % порождающее правило
   n=n+1; tmp=RuleKoch(Lmax,Axiom,Newf,n,tmp); % рекурсия
 end;
end;
z=tmp;
```

Для вывода изображения снежинки Коха можно использовать следующую m-функцию, реализующую описанный выше алгоритм терл-графики:

```
function [X,Y]=Koch(Lmax)
% функция, возвращающая изображение снежинки Коха
% Lmax - порядок снежинки
% порождающие правила
Axiom='F++F++F';
```

```
Newf='F-F++F-F';
teta=pi/3;
alpha=0;
p=[0;0]; % стартовая точка
p=Coord(p,Lmax,Axiom,Newf,alpha,teta,p); % обращение к функции,
                                % возвращающей координаты вершин
M=size(p,2); % число вершин снежинки Коха
X=p(1:1,1:M); % создание вектора, содержащего X-е координаты
              % вершин снежинки
Y=p(2:2,1:M); % создание вектора, содержащего Y-е координаты
              % вершин снежинки
figure(1); % инициализация графического окна
plot(X,Y,'Color','k'); % построение снежинки Коха (рис. 11.5)
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');


function z=Coord(p,Lmax,Axiom,Newf,alpha,teta)
% функция, возвращающая координаты вершин снежинки Коха
Rule=RuleKoch(Lmax,Axiom,Newf,1,' '); % задание L-системы
M=length(Rule);
for i=1:M
  Tmp=p(1:2,size(p,2):size(p,2));
  if Rule(i)=='F' % шаг вперед
     R=[cos(alpha);sin(alpha)];
     R=R/(4^Lmax);
     Tmp=Tmp+R;
     p=cat(2,p,Tmp);
  end;
  if Rule(i)=='+' % увеличение угла, задающего направление
                  % движения
     alpha=alpha+teta;
  end;
  if Rule(i)=='-' % уменьшение угла, задающего направление
                  % движения
     alpha=alpha-teta;
  end;
end;
z=p;
```

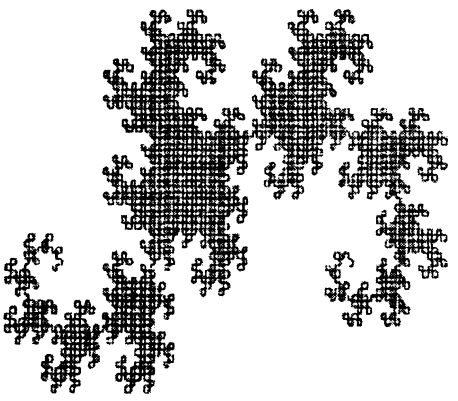


Рис. 11.6. Дракон Хартера-Хайтвея

Однако, при построении других фракталов (например, дракона Хартера-Хайтвея (рис. 11.6) на некоторых шагах возникает необходимость в изменении направления чтения правила (не слева направо, а справа налево). Для решения данной проблемы вводят две дополнительные команды, обозначаемые X и Y, которые используются для создания соответствующей L-системы, но игнорируются черепашкой при перемещении. При использовании этих команд порождающее правило для дракона имеет вид:

Аксиома: FX

Порождающие правила:

$Newf = FX$

$Newx = X + YF +$

$Newy = - FX - Y$

В соответствие с данными правилами L-система имеет следующий вид:

1-ый шаг: $FX + YF +$

2-ой шаг: $FX + YF + + - FX - YF +$

3-ий шаг: $FX + YF + + - FX - YF + + - FX + YF + - - FX - YF +$

4-ый шаг: $FX + YF + + - FX - YF + + - FX + YF + - - FX - YF + + -FX + YF + + - FX - YF + - - FX + YF + - - FX - YF +$

Ниже приводится листинг файла Dracon.m, содержащего описание функции, возвращающей изображение дракона, в соответствие с описанной выше L-системой.

```
function [X,Y]=Dracon(Lmax)
% функция, возвращающая изображение дракона
% Lmax - порядок дракона
% порождающие правила
Axiom='FX'
Newf='F';
Newx='X+YF+'
Newy='-FX-Y'
teta=pi/2;
alpha=0;
p=[0;0];
% обращение к функции, возвращающей координаты угловых точек
% дракона
p=Coord(p,Lmax,Axiom,Newf,Newx,Newy,alpha,teta);
M=size(p,2);
X=p(1:1,1:M);   создание вектора, содержащего X-е координаты
                угловых точек дракона
Y=p(2:2,1:M);   создание вектора, содержащего Y-е координаты
                угловых точек дракона
figure(1);
plot(X,Y,'Color','k');
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');



function z=Coord(p,Lmax,Axiom,Newf,Newx,Newy,alpha,teta)
% функция, возвращающая координаты угловых точек дракона
Rule=DraconString(Lmax,Axiom,Newf,Newx,Newy,1,''); % задание
                                                   % L-системы
M=length(Rule)
```

```
for i=1:M
  Tmp=p(1:2,size(p,2):size(p,2));
  if Rule(i)=='F' % шаг вперед
   R=[cos(alpha);sin(alpha)];
   R=R/(2^Lmax);
   p=cat(2,p,Tmp);
  end;
  if Rule(i)=='+' % увеличение угла, задающего направление
                  % движения
     alpha=alpha+teta;
  end;
  if Rule(i)=='-' % уменьшение угла, задающего направление
                  % движения
     alpha=alpha-teta;
  end;
 end;
z=p;



function z=DraconString(Lmax,Axiom,Newf,Newx,Newy,n,tmp);
% функция, возвращающая L-систему
a=1;
if n<=Lmax
  if n= =1
   tmp=Axiom;
  end;
  M=length(tmp);
  tmp1='';
  for i=1:M
    if tmp(i)=='X'  tmp1=strcat(tmp1,Newx); end;
    if tmp(i)=='Y'  tmp1=strcat(tmp1,Newy); end;
    if not(tmp(i)= ='F') &not(tmp(i)= ='X') &not(tmp(i)= ='Y')
tmp1=strcat(tmp1,tmp(i)); end;
  end;
  tmp=tmp1; n=n+1; tmp=DraconString(Lmax,Axiom,Newf,Newx,
                                                  Newy,n,tmp);
end;
z=tmp;
```

Заменяя в описанной программе порождающие правила, можно получить и другие фрактальные кривые, например, кривую Гильберта (рис. 11.7):

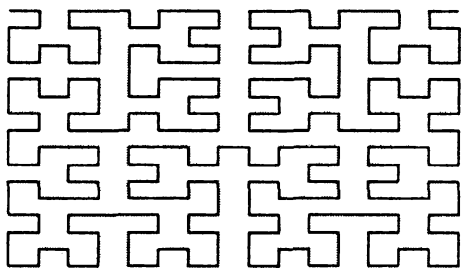


Рис. 11.7. Кривая Гильберта

Аксиома: X
Порождающие правила:
$Newf = F$
$Newx = - YF + XFX + FY -$
$Newy = + XF - YFY - FX +$
$\alpha = 0, \theta = \pi / 2$

кривую Госпера (рис. 11.8):
Аксиома: XF
Порождающие правила:
$Newx = X + YF + + YF - FX - - FXFX - YF +$
$Newy = - FX + YFYF + + YF + FX - - FX - Y$
$\alpha = 0, \theta = \pi / 3$

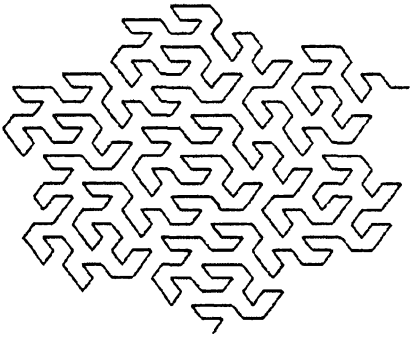


Рис. 11.8. Кривая Госпера

кривую Серпинского (рис. 11.9):
Аксиома: $F + XF + F + XF$
Порождающие правила:
$Newf = F$
$Newx = XF - F + F - XF + F + XF - F + F - X$
$Newy = ''$
$\alpha = \pi / 4, \theta = \pi / 2$

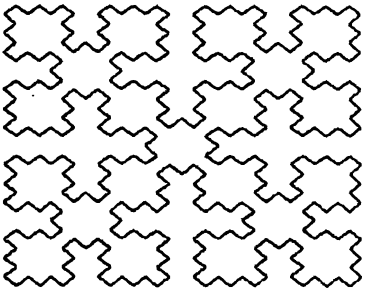


Рис. 11.9. Кривая Серпинского

В заключение остановимся на операции ветвления. Когда в L-системе встречается символ [— открыть ветвь, необходимо запомнить координаты точки нахождения черепашки и направление ее движения, т.е. переменные (x,y,α). К сохраненным переменным следует вернуться после обнаружения символа] — закрыть ветвь. Для хранения триплетов (x,y,α) в [6] предлагается использовать стек:

$$\begin{bmatrix} x_1 & y_1 & \alpha_1 \\ x_2 & y_2 & \alpha_2 \\ & & \\ x_n & y_n & \alpha_n \end{bmatrix},$$

в конец которого записываются новые данные. При закрытии ветви, переменным (x,y,α) присваиваются значения считанные из конца стека, затем эти значения из стека удаляются. В MATLAB оказывается более удобным использовать матрицу с переменным числом столбцов:

$$M = \begin{bmatrix} x_1 & x_2 & & x_n \\ y_1 & y_2 & & y_n \\ \alpha_1 & \alpha_2 & & \alpha_n \end{bmatrix},$$

причем координаты каждой новой точки ветвления добавляются в новый столбец матрицы M. После закрытия ветви, переменным (x,y,α) присваиваются значения, считанные из последнего столбца матрицы M, затем этот столбец удаляется.

Таким образом, ветвление задается двумя символами:

[— открыть ветвь: добавить вектор $\begin{bmatrix} x \\ y \\ \alpha \end{bmatrix}$ новым столбцом матрицы M,

] — закрыть ветвь: присвоить значения переменным (x,y,α), координаты вектора, являющегося последним столбцом матрицы M.

Рис. 11.10. Фрактал Цветок

Пример фрактала, построенного с помощью операции ветвления представлен на рис. 11.10. Ниже приводится листинг файла Flower.m, содержащего описание функции, возвращающей изображение цветка, в соответствие с описанной выше L-системой.

```
function [X,Y]=Flower(Lmax)
% функция, возвращающая изображение цветка
% Lmax - порядок цветка
% порождающие правила
Axiom='F[+F+F][-F-F][++F][--F]F';
Newf='FF[++F][+F][F][-F][--F]';
teta=pi/16;
alpha=pi/2;
p=[0;0]; % начальная точка
Coord(p,Lmax,Axiom,Newf,alpha,teta); % обращение к функции,
                                % возвращающей изображение цветка


function z=Coord(p,Lmax,Axiom,Newf,alpha,teta)
% функция, возвращающая изображение цветка
Rule=FlowerString(Lmax,Axiom,Newf,1,''); % задание L-системы
figure(1);
hold on;
M=length(Rule);
L=0;
x0=p(1);y0=p(2);
for i=1:M
 if Rule(i)=='F' % шаг вперед
xl=x0+cos(alpha);
  yl=y0+sin(alpha);
  X=[x0,xl];
  Y=[y0,yl];
  x0=xl;
  y0=yl;
  plot(X,Y,'Color','k');
 end;
 if Rule(i)=='+' % увеличение угла, задающего направление
                 % движения
  alpha=alpha+teta;
 end;
 if Rule(i)=='-' % уменьшение угла, задающего направление
                 % движения
  alpha=alpha-teta;
 end;
 if Rule(i)=='['    открыть ветвь
  if L= =0
    St=[x0;y0;alpha];
    L=1;
  else
    St=cat(2,St,[x0;y0;alpha]);
  end;
 end;
 if Rule(i)==']'   закрыть ветвь
  M=size(St,2);
  R=St(1:3,M:M);
  x0=R(1);
  y0=R(2);
  alpha=R(3);
  St=St(1:3,1:M-1);
```

```
 end;
end;
hold off


function z=FlowerString(Lmax,Axiom,Newf,n,tmp)
% функция, возвращающая L-систему
while n<=Lmax
 if n==1
  tmp=Axiom; n=n+1;
 else
  tmp=strrep(tmp,'F',Newf);
  n=n+1;tmp=FlowerString(Lmax,Axiom,Newf,n,tmp); %рекурсия
 end;
end;
z=tmp;
```

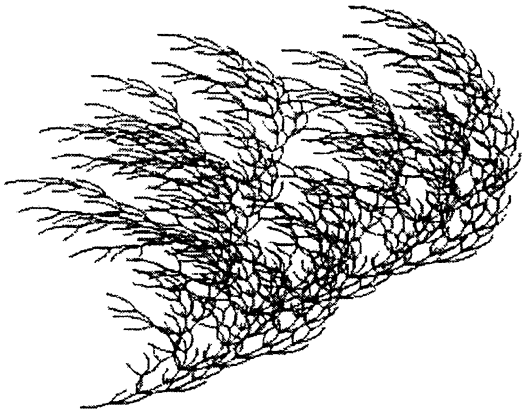


Рис. 11.11. Фрактал Куст

Заменяя в описанной программе порождающие правила, можно получить и другие ветвящиеся фрактальные объекты, например, куст (рис. 11.11):

Аксиома: F

Порождающие правила:

$Newf = - F + F + [+ F - F -] - [- F + F + F]$

$\alpha = \pi / 2$

$\theta = \pi / 8$

снежинка (рис. 11.12):

Аксиома: $[F] + [F] + [F] + [F] + [F] + [F]$

Порождающие правила:

$Newf = F[+ + F][- FF]FF[+ F][- F]FF$

$\alpha = 0$

$\theta = \pi / 3$

и др.

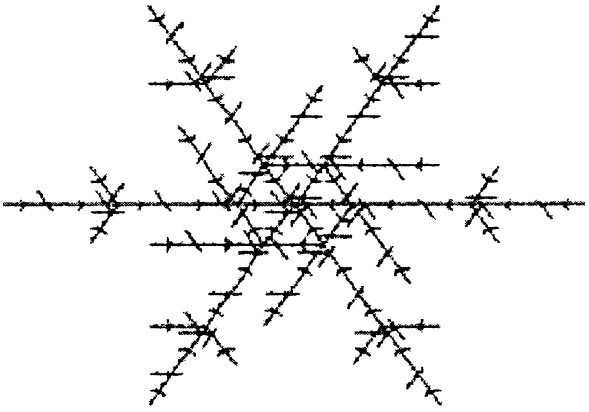


Рис. 11.12. Фрактал Снежинка

11.3. Системы итерированных функций

Напомним, следуя [6], что в общем случае для построения системы итерированных функций (СИФ) в рассмотрение вводится совокупность сжимающих отображений:

$$\begin{gathered} T_1, \text{ с коэффициентом сжатия } s_1, \\ T_2, \text{ с коэффициентом сжатия } s_2, \\ \vdots \\ T_m, \text{ с коэффициентом сжатия } s_m, \end{gathered}$$

действующих в $\boldsymbol{R}^n$. Эти m отображений используются для построения одного сжимающего отображения T в пространстве Ω всех непустых компактов из $\boldsymbol{R}^n$. Преобразование Хатчинсона T: $\Omega \to \Omega$ определяется следующим образом:

$$T(E) = T_1(E) \cup T_2(E) \cup \ldots \cup T_m(E), E \in \Omega.$$

Данное преобразование ставит в соответствие «точкам» из Ω, под которыми здесь понимаются компактные множества, также «точки» из Ω.

Системой итерированных функций называется совокупность выше отображений вместе с итерационной схемой:

$$E_1 = T(E_0),\ E_2 = T(E_1),\ \ldots, E_n = T(E_{n-1}),$$

где E_0 — произвольное компактное множество.

Существование предельного множества $E\left(E = \lim\limits_{n\to\infty} E_n\right)$ системы итерированных функций в смысле сходимости в метрике Хаусфдорфа:

$$H(E,F)=\min\{\varepsilon>0: E\subset F+\varepsilon \,\&\, F\subset E+\varepsilon\},$$

где E и F — непустые компактные подмножества в R^n, доказывает соответствующая теорема [6].

Например, СИФ при построении ковра Серпинского (рис. 11.1), задается тремя аффинными преобразованиями, которые в матричной форме имеют следующий вид:

$$T_1\left(\begin{bmatrix}x_1\\x_2\end{bmatrix}\right)=\begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}+\begin{bmatrix}0\\0\end{bmatrix},$$

$$T_2\left(\begin{bmatrix}x_1\\x_2\end{bmatrix}\right)=\begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}+\begin{bmatrix}1/2\\0\end{bmatrix},$$

$$T_2\left(\begin{bmatrix}x_1\\x_2\end{bmatrix}\right)=\begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}+\begin{bmatrix}1/4\\\sqrt{3/4}\end{bmatrix}.$$

Различают два подхода к реализации СИФ: детерминированный (ДСИФ) (в котором аффинные преобразования применяются последовательно к каждой точке начальной конфигурации), и рандомизированный (РСИФ) (в котором случайно выбираемые аффинные преобразования применяют к единственной начальной точке).

Известный детерминированный алгоритм вычисления СИФ, ориентированный на реализацию в виде компьютерной программы в каком-либо языке программирования, допускающем компиляцию описан в [6]. К недостаткам данного алгоритма можно отнести:

1. Зависимость качества изображения от размера графического окна (удовлетворительное качество изображения достигается для $m\geq 256$).
2. Привязка алгоритма к размеру графического окна, и, как следствие, большой объем вычислений (число операций прямо пропорционально пропорционально числу точек m^2 и числу итераций).
3. Возникновение аварийных остановов программы с сообщением об ошибке «индекс вышел за пределы» при попадании точки за пределы окна $m\times m$.

Рассмотрим модификацию алгоритма ДСИФ, позволяющую реализовать его в MATLAB, на примере уже рассмотренного в первом разделе ковра Серпинского. Во-первых, заметим, что, как видно из рис. 11.13, для получения изображения ковра Серпинского S_1 необходимо на каждую точку $\left(x_i^{(0)}, y_i^{(0)}\right)$, находящуюся внутри исходного треугольника S_0 раздельно подействовать каждым из аффинных преобразований T_1, T_2, T_3:

$$T_1\begin{bmatrix}x_i^{(0)}\\y_i^{(0)}\end{bmatrix}=\begin{bmatrix}x_{i1}^{(1)}\\y_{i1}^{(1)}\end{bmatrix},$$

$$T_2\begin{bmatrix}x_i^{(0)}\\y_i^{(0)}\end{bmatrix}=\begin{bmatrix}x_{i2}^{(1)}\\y_{i2}^{(1)}\end{bmatrix},$$

$$T_3\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i3}^{(1)} \\ y_{i3}^{(1)} \end{bmatrix}.$$

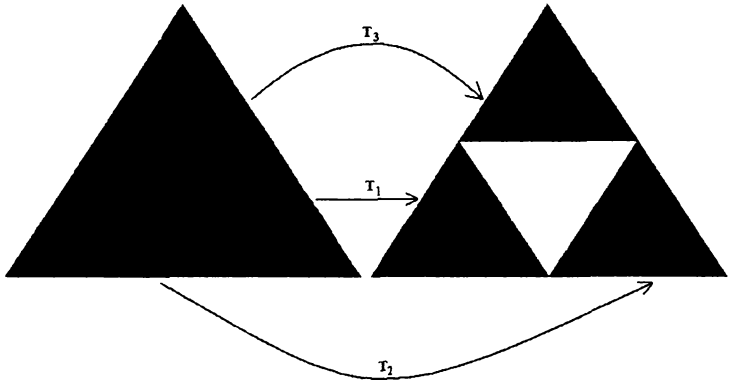


Рис. 11.13. Построение ковра Серпинского с помощью алгоритма ДСИФ

Таким образом, каждая точка $\left(x_i^{(0)}, y_i^{(0)}\right)$ на первом шаге итерации, порождает три новые точки: $\left(x_{i1}^{(0)}, y_{i1}^{(0)}\right)$, $\left(x_{i2}^{(0)}, y_{i2}^{(0)}\right)$, $\left(x_{i3}^{(0)}, y_{i3}^{(0)}\right)$, на каждую из этих точек на втором шаге итерации вновь следует подействовать аффинными преобразованиями T_1, T_2, T_3. В результате каждая из трех точек ковра S_1 вновь породит три точки ковра S_2 и т.д. Описанный процесс удобно изобразить в виде следующего графа (рис. 11.14).

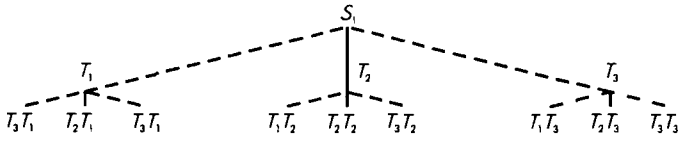


Рис. 11.14. Иллюстрация алгоритма ДСИФ

Из рис. 11.14. видно, что на втором шаге итерации существует 9 ($3^n = 9$, где $n = 2$) правил, по которым каждой начальной точке $\left(x_i^{(0)}, y_i^{(0)}\right)$ ставятся в соответствие 9 точек ковра Серпинского S_1. Соответственно, на третьем шаге итерации таких правил будет $27 = 3^3$:

$$T_1T_1T_1\ T_1T_2T_1\ T_1T_3T_1\ T_1T_1T_2\ T_1T_2T_2\ T_1T_3T_2\ T_1T_1T_3\ \ T_1T_2T_3\ T_1T_3T_3$$

$$T_2T_1T_1\ T_2T_2T_1\ T_2T_3T_1\ T_2T_1T_2\ T_2T_2T_2\ T_2T_3T_2\ T_2T_1T_3\ T_2T_2T_3\ T_2T_3T_3.$$

$$T_3T_1T_1\ T_3T_2T_1\ T_3T_3T_1\ T_3T_2T_1\ T_3T_2T_2\ T_3T_3T_2\ T_3T_1T_3\ T_3T_2T_3\ T_3T_3T_3$$

Таким образом, для построения ковра Серпинского n-го уровня с помощью ДСИФ, необходимо научиться генерировать 3^n правил, по которым каждой точке $\left(x_i^{(0)}, y_i^{(0)}\right)$ ставится в соответствие 3^n точек $\left(x_{ij}^{(n)}, y_{ij}^{(n)}\right)$, $j = 1,2,...3^n$. Ведем обозначения: $T_1 \Leftrightarrow 0$, $T_2 \Leftrightarrow 1$, $T_3 \Leftrightarrow 2$. В выбранных обозначениях правила преобразования на третьем шаге итерации имеют вид:

000	010	020	001	011	021	002	012	022
100	110	120	101	111	121	102	112	122.
200	210	220	201	221	221	202	212	222

Анализ таблицы закодированных правил преобразований показывает, что названия правил являются ничем иным как множеством натуральных чисел 1,2,...27, записанных в троичной системе счисления. При этом для представления кода каждого правила используется число цифр, совпадающее с порядком ковра n. Соответственно, для случая $n = 4$ имеем множество, состоящее из $3^4 = 81$ правила, названия которых есть множество чисел 1,2...81, записанных с троичной системе счисления, при этом для представления каждого числа используются 4 цифры. Очевидно, чтс для хранения названия правил наиболее удобно использовать массив строковых переменных длиной n, число элементов которого равно 3^4.

Таким образом, для построения ковра Серпинского в MATLAB с помощью ДСИФ можно использовать следующий алгоритм:

1. Задать порядок ковра Серпинского n.
2. Задать число точек начальной конфигурации m.
3. Задать координаты i точек ($i = 1,2...m$), заполняющих начальное множество.
4. Перевести каждое из чисел $1,2,...3^n$ в троичную систему счисления.
5. Сформировать массив, состоящий из 3^n строк, длиной n символов.
6. Задать аффинные преобразования.
7. Для i-ой точки начальной конфигурации последовательно применить каждое из $j = 1,2,...,3^n$ итерационных правил и отобразить в графическом окне полученные образы каждой начальной точки.

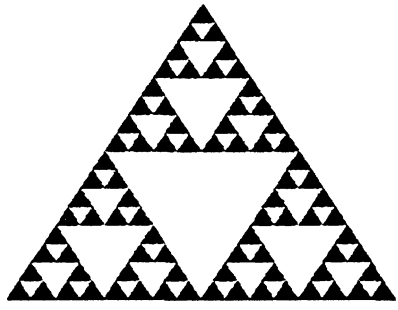


Рис. 11.15. Ковер Серпинского, построенный с помощью ДСИФ

Пример ковра Серпинского, построенного с помощью описанной выше модификации алгоритма ДСИФ, представлен на рис. 11.15. Ниже приводится листинг файла SerpDSIF.m, содержащего описание соответствующей функции, возвращающей изображение ковра Серпинского.

```
function z=SerpDSIF(Niter,NPoints)
% функция, возвращающая изображение ковра Серпинского
% Niter - порядок ковра
% NPoints - число точек начальной конфигурации
x=zeros(NPoints,1);
y=zeros(NPoints,1);
% задание координат точек начальной конфигурации
x1=0; y1=0;
x2=1; y2=0;
x3=1/2; y3=sin(pi/3);
j=1;
while j<=NPoints
 tmpx=rand(1,1);
 tmpy=sqrt(3)/2*rand(1,1);
 if (-sqrt(3)*tmpx+tmpy<=0)&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
  x(j)=tmpx;
  y(j)=tmpy;
  j=j+1;
 end;
end;
% Формирование массива, содержащего правила итерации
for i=1:3^Niter
  Tmp(i)=system3(i-1);   перевод числа из десятичной в
                         троичную систему счисления
end;
n=1;s='0';
while n<Niter
 s=strcat(s,'0'); n=n+1;
end;
for i=1:3^Niter
  tmp=num2str(Tmp(i));tmp1=s;
  for m=1:length(tmp)
    tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
  end;
  Cod(i,1:Niter)=tmp1;
end;
% задание аффинных преобразований
a1=[0;0];
a2=[1/2;0];
a3=[1/4;sqrt(3)/4];
A=[1/2,0;0,1/2];
% создание графического окна и задание его параметров
figure(1); hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
fill([x1 x2 x3],[y1 y2 y3],'w');
% построение ковра Серпинского
GosperDraw(Niter,NPoints,x,y,A,a1,a2,a3,Cod);



function z=GosperDraw(Niter,NPoints,x,y,A,a1,a2,a3,Cod)
% функция, создающая изображение ковра Серпинского
for m=1:3^Niter
  X=x; Y=y; Rule=Cod(m,:);
  for i=1:Niter
```

```
      tmp=Rule(Niter+1-i);
      if tmp=='0'
         [X Y]=T(NPoints,X,Y,A,a1);    первое аффинное
                                       преобразование
      end;
      if tmp=='1'
         [X Y]=T(NPoints,X,Y,A,a2);    второе аффинное
                                       преобразование
      end;
      if tmp=='2'
         [X Y]=T(NPoints,X,Y,A,a3);    третье аффинное
                                       преобразование
      end;
    end;
    plot(X,Y,' ','MarkerSize',1,
                 'MarkerEdgeColor','k');   отображение
                                           результатов итерации
end;


function [X,Y]=T(NPoints,x,y,A,a)
% функция, возвращающая результат аффинного преобразования
X=zeros(NPoints,1);
Y=zeros(NPoints,1);
for i=1:NPoints
  R=[x(i);y(i)]; R=A*R+a; X(i)=R(1); Y(i)=R(2);
end;


function z=system3(D);
% функция, возвращающая значение целого числа в троичной системе
% счисления
% D - число в десятичной системе счисления
n=1;
while D>=3^n
  n=n+1;
end;
if n>1
   a=floor(D/3^(n-1))*10^(n-1); b=mod(D,3^(n-1));
   if b>=3
      b=system3(b);   рекурсия
    end;
   z=a+b;
else
   z=D;
end;
```

Ниже приводятся пример фракталов, построенных с использованием ДСИФ, и листинги соответствующих программ.

```
% листинг функции, возвращающей изображение фрактал Лист
function z=MapleS(Niter,NPoints)
% NIter - число итераций
% NPoints - количество точек начального множества
x1=0; y1=0;
x2=1; y2=0;
x3=1/2; y3=sin(pi/3);
x=zeros(NPoints,1);y=zeros(NPoints,1);
j=1;
while j<=NPoints
```

```
 tmpx=rand(1,1);
 tmpy=sqrt(3)/2*rand(1,1);
 if (-sqrt(3)*tmpx+tmpy<=0)&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
   x(j)=tmpx; y(j)=tmpy; j=j+1;
 end;
end;
for i=1:2^Niter
 Tmp(i)=system2(i-1,2);
end;
n=1; s='0';
while n<Niter s=strcat(s,'0');n=n+1;
end;
for i=1:2^Niter
 tmp=num2str(Tmp(i)); tmp1=s;
 for m=1:length(tmp) tmp1(Niter-m+1)=tmp(length(tmp)-m+1); end;
 Cod(i,1:Niter)=tmp1;
end;
A1=[0.4,-0.3733;0.0600,0.6000];
A2=[-0.8000,-0.1867;0.1371,0.8000];
a1=[0.3533;0.000]; a2=[1.1000;0.1000];
figure(1); hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
FractalDraw(Niter,NPoints,x,y,A1,A2,a1,a2,Cod);



function z=FractalDraw(Niter,NPoints,x,y,A1,A2,a1,a2,Cod)
for m=1:2^Niter
  X=x;
  Y=y;
  Rule=Cod(m,:);
  for i=1:Niter
    tmp=Rule(Niter+1-i);
    if tmp=='0' [X Y]=T(NPoints,X,Y,A1,a1); end;
    if tmp=='1' [X Y]=T(NPoints,X,Y,A2,a2); end;
  end;
plot(X,Y,'  ','MarkerSize',1,'MarkerEdgeColor','k');
end;



function [X,Y]=T(NPoints,x,y,A,a)
X=zeros(NPoints,1);
Y=zeros(NPoints,1);
for i=1:NPoints R=[x(i);y(i)];
 R=A*R+a;
 X(i)=R(1);
 Y(i)=R(2);
end;



function z=system2(D,m);
n=1;
while D>=m^n n=n+1; end;
if n>1
   a=floor(D/m^(n-1))*10^(n-1)
   b=mod(D,m^(n-1));
   if b>=m b=system2(b,m); end;
   z=a+b;
```

```
else
 z=D;
end
```

Рис. 11.16. Фрактал Лист

```
% листинг файла, содержащего описание функции, возвращающей
% изображение ветки папоротника (рис. 11.17)
function z=Paporotnic(Niter,NPoints)
x1=0; y1=0;
x2=1; y2=0;
x3=1/2; y3=sin(pi/3);
x=zeros(NPoints,1); y=zeros(NPoints,1);
j=1;
while j<=NPoints
 tmpx=rand(1,1);
 tmpy=sqrt(3)/2*rand(1,1);
 if (-sqrt(3)*tmpx+tmpy<=0)&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
     x(j)=tmpx;
     y(j)=tmpy;
     j=j+1;
 end;
end;
for i=1:4^Niter
  Tmp(i)=system2(i-1,4);
end;
n=1; s='0';
while n<Niter s=strcat(s,'0');
 n=n+1;
end;
for i=1:4^Niter
  tmp=num2str(Tmp(i)); tmp1=s;
  for m=1:length(tmp)
    tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
  end;
  Cod(i,1:Niter)=tmp1;
end;
A1=[0.7000,0;0,0.7000]; A2=[0.1000,-0.4330;0.1732,0.2500];
```

```
A3=[0.1000,0.4330;-0.1732,0.2500]; A4=[0,0;0,0.3000];
a1=[0.1496;0.2962]; a2=[0.4478;0.0014]; a3=[0.4450;0.1559];
                                              a4=[0.4987;0.0070];
figure(1); hold on; set(gca,'xtick',[],'ytick',[]);
                          set(gca,'XColor','w','YColor','w');
FractalDraw(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod);

function z=Simplex
                  (Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
for m=1:4^Niter
  X=x; Y=y; Rule=Cod(m,:);
  for i=1:Niter
    tmp=Rule(Niter+1-i);
    if tmp= ='0' [X Y]=T(NPoints,X,Y,A1,a1); end;
    if tmp= ='1' [X Y]=T(NPoints,X,Y,A2,a2); end;
    if tmp= ='2' [X Y]=T(NPoints,X,Y,A3,a3); end;
    if tmp= ='3' [X Y]=T(NPoints,X,Y,A4,a4); end;
  end;
  plot(X,Y,'  ,'MarkerSize',1,'MarkerEdgeColor','k
end;
function [X,Y]=T(NPoints,x,y,A,a)
X=zeros(NPoints,1); Y=zeros(NPoints,1);
for i=1:NPoints
  R=[x(i);y(i)];
  R=A*R+a;
  X(i)=R(1);
  Y(i)=R(2);
end;
function z=system2(D,m);
n=1;
while D>=m^n n=n+1; end;
if n>1
  a=floor(D/m^(n-1))*10^(n-1);
  b=mod(D,m^(n-1));
  if b>=m b=system2(b,m); end;
  z=a+b;
else
  z=D;
end;
```

Рис. 11.17. Фрактал Папоротник

В отличие от ДСИФ в рандомизированном алгоритме начальное множество S_0 состоит из одной точки (x_0,y_0), а правило, по которому точке ставится в соответствие точка (x_i,y_i), где i — номер правила, выбирается случайным образом из набора, содержащего все возможные правила аффинных преобразований. Например, применительно к ковру Серпинского это означает, что при построении ковра 2-го порядка преобразование должно случайным образом выбираться из следующего множества преобразований:

$$\{T_1, T_2, T_3, T_1T_1, T_1T_2, T_1T_3, T_2T_3, T_3T_1, T_3T_2, T_3T_3\},$$

число элементов N которого, как очевидно равно $N = \sum_{k=1}^{n} m^k$.

Таким образом, для построения ковра Серпинского в MATLAB с помощью РСИФ можно использовать следующий алгоритм:

1. Задать порядок ковра Серпинского n.
2. Задать число испытаний *NTrail*.
3. Задать число аффинных преобразований $m = 3$.
4. Сформировать массив, содержащий набор правил для аффинных преобразований.
5. Задать координаты начальной точки (x_0,y_0).
6. Перевести каждое из чисел $1,2,...3^n$ в троичную систему счисления.
7. Задать аффинные преобразования.
8. Для заданного числа испытаний последовательно, начиная с начальной точки, в соответствие с правилами аффинных преобразований, выбираемых случайным образом, вычислить точки итерационной последовательности.
9. Отобразить вычисленное множество точек в графическом окне.

В общем случае для фракталов n-го порядка, изображение которых создается с помощью m аффинных преобразований, как и при использовании ДСИФ, необходимо переводить числа $1,2,...m^n$ в m-ичную систему счисления.

Пример кристалла, построенного с помощью описанного выше алгоритма РСИФ, представлен на рис. 11.18. Ниже приводится листинг файла Cristal.m, содержащего описание соответствующей функции.

Рис. 11.18. Фрактал Кристалл

```
function z=Cristal(Niter,NTrial)
% функция, возвращающая изображение кристалла
% Niter - порядок кристалла
% NTrial   число испытаний
Na=4; % число аффинных преобразований
% Создание массива, содержащего набор правил для
% аффинных преобразований
k=1;
for m=1:Niter
  for i=1:4^m
    Tmp(k)=system3(i-1,Na);
    k=k+1;
  end;
end;
Q(1)=Na;
for m=2:Niter
  Q(m)=Q(m-1)+Na^m;
end;
n=1;
s='0'
M=1;
while n<=length(Tmp)
  m=1;
  while n>Q(m)
    m=m+1;
  end;
  if m= =1
    S(n,1:1)=s;
  else
    S(n,1:1)=s;
    for i=2:m
      S(n,1:i)=strcat(S(n,:),s);
    end;
  end;
  n=n+1;
end;
for i=1:k-1
  tmp=num2str(Tmp(i));
  m=1;
  while i>Q(m)
```

```
    m=m+1;
   end;
  tmp1(1:m)=S(i,1:m);
  for m=1:length(tmp)
    tmp1(length(tmp1)-m+1;
    length(tmp1)-m+1)=tmp(length(tmp)-m+1;
   length(tmp)-m+1);
  end;
  Cod(i,1:length(tmp1))=tmp1;
end;
x=0; y=0; % координаты начальной точки
          % задание аффинных преобразований
A1=[0.2550,0.0000;0.0000,0.2550];
A2=[0.2550,0.0000;0.0000,0.2550];
A3=[0.2550,0.0000;0.0000,0.2550];
A4=[0.3700,-0.6420;0.6420,0.3700];
a1=[0.3726;0.6714]; a2=[0.1146;0.2232]; a3=[0.6306;0.2232];
a4=[0.6356;-0.0061];
figure(1); hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
% визуализация фрактала
DrawFractal(Niter,NTrial,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod);



function z=DrawFractal
                  (Niter,NTrial,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
% функция, возвращающая изображение фрактала
X1=zeros(NTrial,1); Y1=zeros(NTrial,1); X=x; Y=y;
for m=1:NTrial
  Np=1+round((size(Cod,1)-1)*rand(1,1));   выбор номера
                                           преобразования
  Rule=Cod(Np,:);
  for i=1:length(Rule)
   tmp=Rule(length(Rule)+1-i);
   if tmp= ='0'
    [X Y]=T(X,Y,A1,a1);
   end;
   if tmp= ='1'
    [X Y]=T(X,Y,A2,a2);
   end;
   if tmp= ='2'
    [X Y]=T(X,Y,A3,a3);
   end;
   if tmp= ='3'
    [X Y]=T(X,Y,A4,a4);
   end;
  end;
  X1(m)=X; Y1(m)=Y;
end;
plot(X1,Y1,'.','MarkerSize',1,'MarkerEdgeColor','k');
function [X,Y]=T(x,y,A,a)
% функция, возвращающая результат аффинного преобразования
R=[x;y];
R=A*R+a;
X=R(1); Y=R(2);
```

```
function z=system3(D,m);
% функция, возвращающая значение числа четверичной системе
                                                    координат
n=1;
while D>=m^n
  n=n+1;
end;
if n>1
  a=floor(D/m^(n-1))*10^(n-1); b=mod(D,m^(n-1));
  if b>=m
    b=system3(b,m);
   end;
  z=a+b;
else
 z=D;
end
```

Вопросы для самопроверки

1. Какой последовательностью действий реализуется построение треугольного ковра Серпинского?
2. Какой последовательностью действий реализуется построение квадратного ковра Серпинского?
3. Какой последовательностью действий реализуется построение кривой Коха?
4. Что такое L-система, порождающее правило и терл-графика?
5. Дайте геометрическую интерпретацию аксиоме, использующейся при построении снежинки Коха.
6. Чем отличается аксиомы и порождающие правила, используемые при построении снежинки Коха и дракона Хартера-Хайтвея?

Литература к главе 11

1. Mandelbrot B.B. Les object fractals: forme, hasard et dimantion. Paris: Flamarion, 1975.
2. Каток А.Б., Хасселблат Б. Введение в современную теорию динамических систем. М.: Факториал, 1999.
3. Лихтенберг А., Либерман М. Регулярная и стохастическая динамика. М.: Меркурий-ПРЕСС, 2000.
4. Шустер Г. Детерминированный хаос. М.: Мир, 1988.
5. Кренкель Э.Т. Сжатие сигналов с применением теории фракталов. М.: МТУСИ, 1996.
6. Кроновер Р.М. Фракталы и хаос в динамических системах. Основы теории. М.: Постмаркер, 2000.
7. http://www.exponenta.ru

Глава 12

Моделирование колебательной системы с несколькими степенями свободы в пакете Simulink

Для проектирования и анализа динамических систем, а также систем автоматического управления, в настоящее время широко используются средства вычислительной техники. К последним, в частности, относится пакет моделирования динамических и событийно-управляемых систем Simulink [1], входящий в состав пакетов расширений MATLAB. Пакет Simulink позволяет сокращать сроки проектирования, повышать качество разработки моделей физических систем и моделирования процессов, протекающих в данных системах.

Напомним, что в использовавшемся до недавнего времени традиционном подходе сначала создавалась математическая модель (как правило, в виде структурной схемы) далее создавалась программная реализация данной модели на каком-либо из универсальных языков программирования. При этом с неизбежностью возникало дублирование описания элементов и связей между ними. Кроме того значительных временных затрат требовали написание текстов программ и их отладка. В пакете Simulink принципиально изменен характер требований, предъявляемых к математическому обеспечению, — для управления всем ходом вычислительного процесса разработаны графические модули, которые используются для составления структурных схем исследуемых систем.

В данной главе демонстрируются возможности пакета Simulink, на примере, описания динамики колебательных систем с несколькими степенями свободы в пространстве состояний [2].

Напомним, что в пространстве состояния любая многомерная динамическая система описывается системой дифференциальных уравнений первого порядка в явной форме

$$\mathbf{x} = f(\mathbf{x},\mathbf{u},t), \tag{12.1}$$

где $\mathbf{x}(t)$ — n-мерный вектор-столбец, компонентами которого являются переменные состояния, $\mathbf{u}(t)$ — r-мерный вектор-столбец, координаты которого содержат значения выходных переменных, t — независимая переменная, время. Уравнение (12.1) частот называют уравнением состояния. Выходная переменная может быть представлена следующим образом

$$\mathbf{y} = g(\mathbf{x},\mathbf{u},t). \tag{12.2}$$

Уравнение (12.2) называется уравнением выходной переменной.

Для многомерных линейных систем уравнения (12.1), (12.2) соответственно принимают вид

$$\mathbf{x}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t), \tag{12.3}$$

$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t), \tag{12.4}$$

где $\mathbf{A}(t)$, $\mathbf{B}(t)$, $\mathbf{C}(t)$, $\mathbf{D}(t)$ — матрицы размера $(n\times n)$, $(n\times r)$, $(k\times n)$, $(k\times k)$, соответственно.

Если матрицы **A**, **B**, **C**, **D** не зависят от времени, то такая система называется многомерной стационарной системой.

Для описания динамики системы в пространстве состояний необходимо также задать вектор начальных условий

$$\mathbf{x}(t_0) = \mathbf{x}_0. \tag{12.5}$$

На первом шаге найдем матрицы **A**, **B**, **C**, **D** для колебательной системы, имеющей одну степень свободы, уравнение движения которой имеет вид

$$\ddot{x} = -\omega^2 x + f(t), \tag{12.6}$$

где ω^2 — циклическая частота, x — смещение колеблющейся материальной точки от положения равновесия, $f(t)$ — вынуждающая сила, отнесенная к массе колеблющегося тела.

В качестве компонентов вектора состояния выберем

$$\begin{aligned} x_1 &= x, \\ x_2 &= \dot{x}. \end{aligned} \tag{12.7}$$

Подставляя выражение (12.7) в (12.6), получаем уравнения состояния

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = -\omega^2 x_1 + f(t). \end{cases} \tag{12.8}$$

Стандартная форма уравнений состояния (12.8) в векторно-матричных обозначениях имеет вид

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{12.9}$$

где

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix},$$

$$\mathbf{u} = f(t),$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Если выходом рассматриваемой системы является смещение колебательной системы от положения равновесия, то

$$\mathbf{y} = x_1,$$

тогда

$$\mathbf{C} = [1\ 0],$$

и

$$\mathbf{D} = 0.$$

Если выходом рассматриваемой системы является ее скорость, то

$$y = x^2,$$
$$\mathbf{C} = [0\quad 1].$$

Если выходами рассматриваемой системы являются и смещение колебательной системы относительно положения равновесия и ее скорость, то

$$\boldsymbol{y} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Таким образом, задача построения модели в пространстве состояний не имеет однозначного решения.

Рассмотрим более сложную линейную динамическую систему, представленную на рис. 12.1.

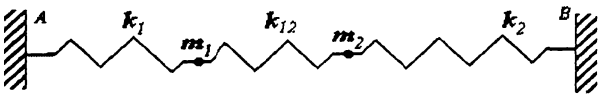


Рис. 12.1. Схема исследуемой колебательной системы

Система уравнений движения данной системы имеет вид

$$\begin{cases} \ddot{x}_1 = -\dfrac{k_1}{m_1} x_1 - \dfrac{k_{12}}{m_1}(x_1 - x_2) + \dfrac{F_1(t)}{m_1}, \\ \ddot{x}_2 = -\dfrac{k_2}{m_2} x_2 - \dfrac{k_{12}}{m_2}(x_1 - x_2) + \dfrac{F_2(t)}{m_2}. \end{cases} \tag{12.10}$$

В качестве компонентов вектора состояния выберем

$$\begin{aligned} w_1 &= x_1, \\ w_2 &= \dot{x}_1, \\ w_3 &= x_2, \\ w_4 &= \dot{x}_2. \end{aligned}$$

В выбранных переменных уравнения движения (12.10) принимают вид

$$\begin{cases} \dot{w}_1 = w_2, \\ \dot{w}_2 = -\dfrac{k_1}{m_1} w_1 - \dfrac{k_{12}}{m_1}(w_1 - w_3) + \dfrac{F_1(t)}{m_1}, \\ w_3 = w_4, \\ \dot{w}_4 = -\dfrac{k_2}{m_2} w_3 - \dfrac{k_{12}}{m_2}(w_3 - w_1) + \dfrac{F_2(t)}{m_2}. \end{cases} \tag{12.11}$$

Из системы (12.11) находим

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\dfrac{k_1 + k_{12}}{m_1} & 0 & \dfrac{k_{12}}{m_1} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{k_{12}}{m_2} & 0 & -\dfrac{k_2 + k_{12}}{m_2} & 0 \end{pmatrix},$$

$$\mathbf{u} = \begin{bmatrix} F_1(t) \\ F_2(t) \end{bmatrix},$$

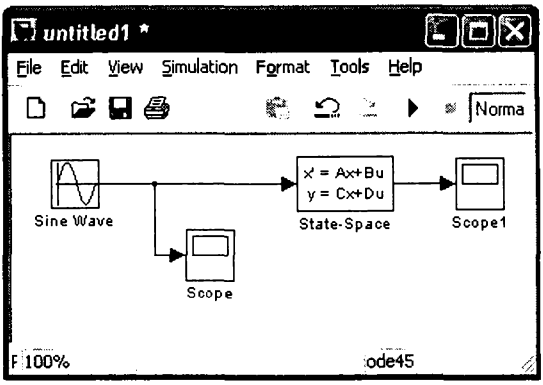


Рис. 12.2. Блок-схема математической модели колебательной системы в пространстве состояний

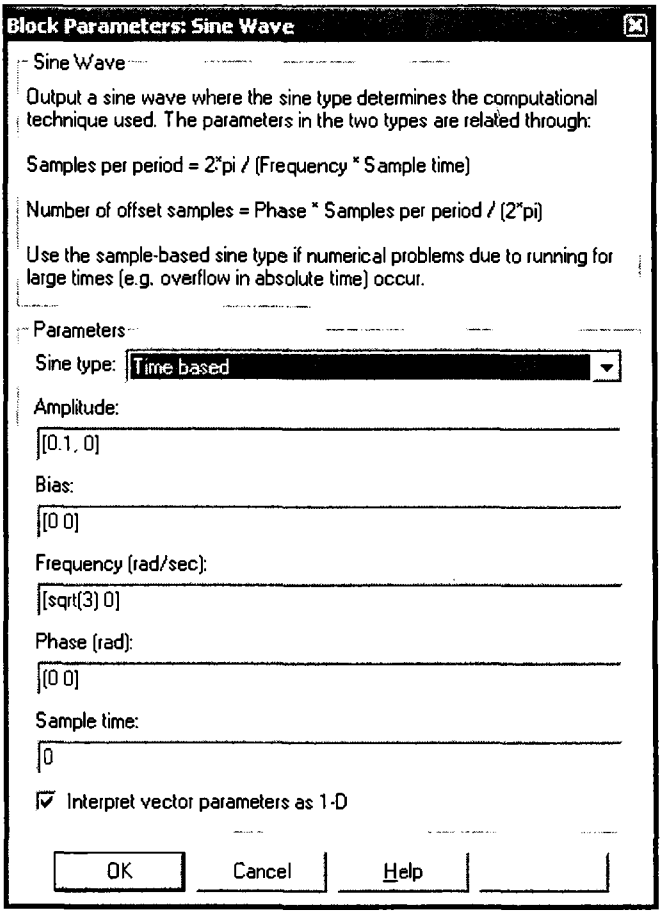


Рис. 12.3. Окно для ввода параметров внешней силы, приложенной к колебательной системе

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ \frac{1}{m_1} & 0 \\ 0 & 0 \\ 0 & \frac{1}{m_2} \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

После вычисления матриц **A**, **B**, **C**, **D**, необходимо, запустить пакет Simulink, создать с помощью Simulink Library Browser, поддерживающего технологию визуального проектирования, блок-схему математической модели (рис. 12.2).

Далее ввести в окно «Block Parameters: Sine Wave» параметры внешних сил, приложенных к колебательной системе (рис. 12.3).

Затем в окне «Block Parameters: State-Space» задать матрицы **A**, **B**, **C**, **D**, а также начальные условия (рис. 12.4). После выполнения перечисленных выше действий оказывается достаточным запустить модель (меню Simulation => пункт Start) и открыть окно отображения зависимости внешних сил, дей-

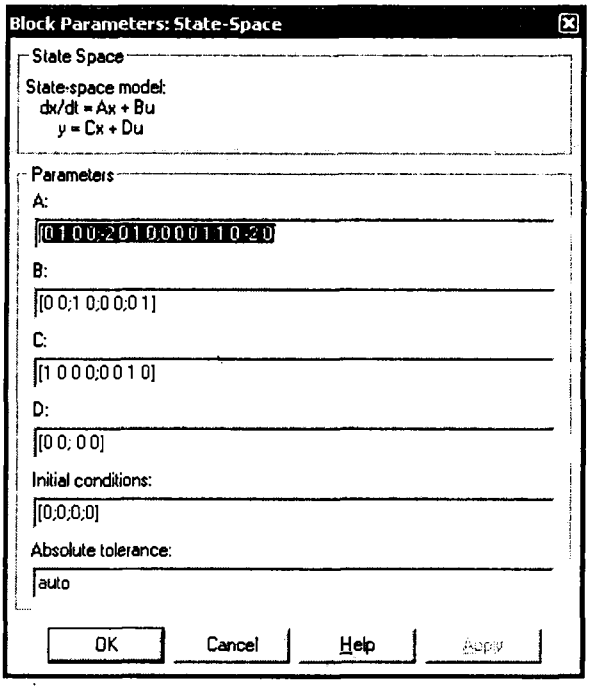


Рис. 12.4. Окно для задания матриц **A**, **B**, **C**, **D** и начальных условий

ствующих на колебательную систему от времени, а также выходных параметров, дважды щелкнув на блок-схеме по приборам Scope и Scope 1, соответственно, (рис. 12.5, 12.6).

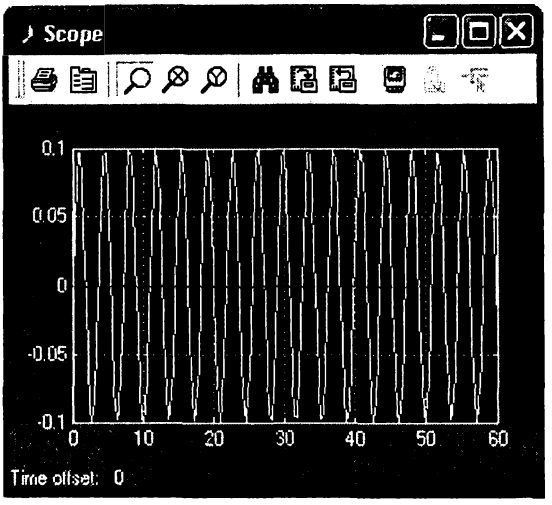


Рис. 12.5. Зависимость мгновенных значений внешних сил, приложенных к колебательной системе от времени

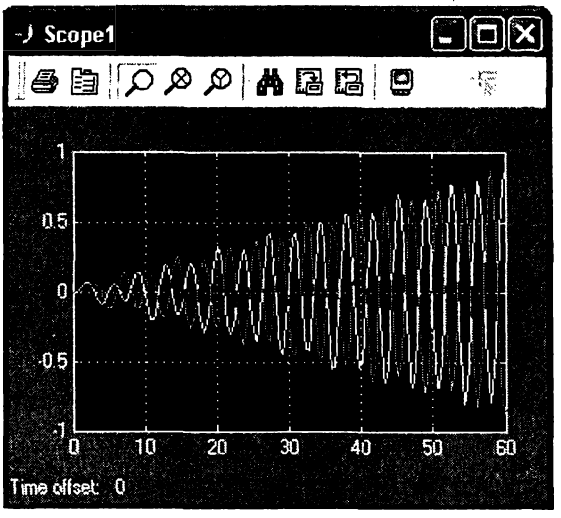


Рис. 12.6. Зависимость мгновенных смещений тел колебательной системы от времени

Реализацию традиционного подхода, основанного на реализации математической модели в виде программы, написанной на m-языке MATLAB, заинтересованный читатель может найти в [3, 4].

Отметим, два важных обстоятельства. Во-первых, в пакете Simulink имеется возможность считывать значения переменных, находящихся в рабочем пространстве MATLAB (Workspace). Для этого используется блок чтения/записи входных/выходных переменных, представленных на рис. 12.7.

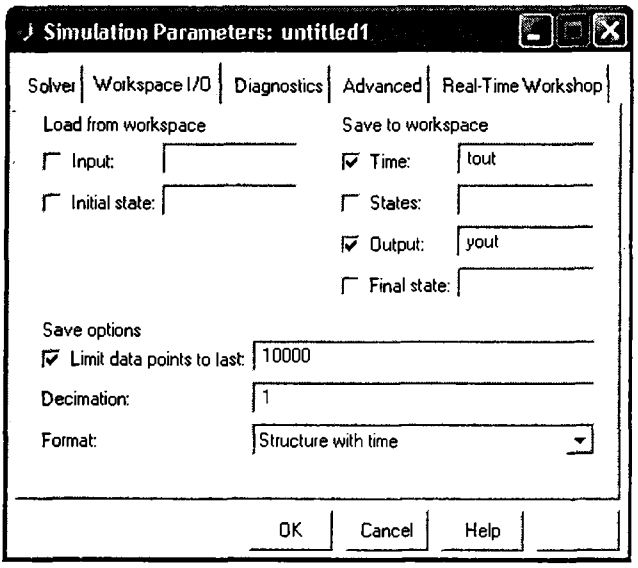


Рис. 12.7. Блок чтения/записи данных из Workspace

Во-вторых, процедура вычисления элементов матрицы **A** для произвольного числа тел колебательной системы, достаточно легко алгоритмизуется [3]. Сказанное в полной мере относится и к матрицам **B**, **C**, **D**. Это дает возможность использовать описанную выше блок-схему математической модели колебательной системы для исследования динамики системы с произвольным числом тел. Действительно, достаточно создать соответствующие функции, размещающие матрицы **A**, **B**, **C**, **D** в рабочем пространстве MATLAB, откуда они будут считываться перед началом выполнения расчетов.

Отмеченные свойства, в сочетании с другими не менее полезными свойствами, делают пакет Simulink гибким инструментом исследования динамических систем, а также систем управления, который целесообразно использовать при разработке методической поддержки дисциплин естественнонаучного блока, изучаемых в высшей школе. Знакомства с основами работы с данным пакетом представляется целесообразным, как для студентов технических, так и педагогических университетов.

Вопросы для самопроверки

1. Как описывается многомерная динамическая система в пространстве состояний?
2. Как выбирается вектор в пространстве состояний для одномерной колебательной системы?
3. Какова стандартная форма записи уравнений состояния одномерной колебательной системы в пространстве состояний?
4. Какова стандартная форма записи уравнений состояния колебательной системы с двумя степенями свободы в пространстве состояний?
5. Назовите последовательность действий, выполнением которых в пакете Simulink осуществляется построение компьютерной модели.

Литература к главе 12

1. Дэбни Дж., Харман Т. Simulink 4. Секреты мастерства. -М.: Бином, Лаборатория базовых знаний, 2003. -403 с.
2. Пантелеев А.В., Бортаковский А.С. Теория управления в примерах и задачах. -М.: Высшая школа, 2003. -583 с.
3. Поршнев С.В. Компьютерное моделирование физических процессов в пакете MATLAB. -М.: Горячая линия-Телеком, 2003. -592 с.
4. Поршнев С.В. Исследование в MATLAB особенностей вынужденных колебаний цепочки связанных гармонических осцилляторов// Труды Второй Всероссийской научной конференции «Проектирование инженерных и научных приложений в среде MATLAB». М.: ИПУ РАН, 2004. 665-681.

Приложение 1

Листинг измененного файла sym.m

```
function S = sym(x,a,b)
% SYM    Construct symbolic numbers, variables and objects.
%  S = SYM(A) constructs an object S, of class 'sym', from A.
%  If the input argument is a string, the result is a symbolic
%  number or variable.  If the input argument is a numeric
%  scalar or matrix, the result is a symbolic representation of
%  the given numeric values.
%
%  x = sym('x') creates the symbolic variable with name 'x'
%  and stores the result in x.  x = sym('x','real') also assumes
%  that x is real, so that conj(x) is equal to x.  alpha =
%  sym('alpha') and r = sym('Rho','real') are other examples.
%  Similarly, k = sym('k','positive') makes k a positive (real)
%  variable.  x = sym('x','unreal') makes x a purely formal
%  variable with no additional properties (i.e., insures that x
%  is NEITHER real NOR positive)
%   See also: SYMS.
%
%  Statements like pi = sym('pi') and delta = sym('1/10') create
%  symbolic numbers which avoid the floating point approximations
%  inherent in the values of pi and 1/10. The pi created in this way
%  temporarily replaces the built-in numeric function with the
%  same name.
%
%  S = sym(A,flag) converts a numeric scalar or matrix to
%  symbolic form. The technique for converting floating point
%  numbers is specified by the optional second argument, which
%  may be 'f', 'r', 'e' or 'd'  The default is 'r'
%
%  'f' stands for 'floating point'.  All values are represented
%  in the form '1.F'*2^(e) or '-1.F'*2^(e) where F is a string
%  of 13 hexadecimal digits and e is an integer. This captures
%  the floating point values exactly, but may not be convenient}
%  for subsequent manipulation. For example, sym(1/10,'f') is
%  '1.999999999999a'*2^(-4) because 1/10 cannot be represented
%  exactly in floating point.
%
```

```
%  'r' stands for 'rational'  Floating point numbers obtained
%  by evaluating expressions of the form p/q, p*pi/q, sqrt(p),
%  2^q and 10^q for modest sized integers p and q are converted
%  to the corresponding symbolic form. This effectively
%  compensates for the roundoff error involved in the original
%  evaluation, but may not represent the floating point value
%  precisely.  If no simple rational approximation can be found,
%  an expression of the form p*2^q with large integers p and q
%  reproduces the floating point value exactly. For example,
%  sym(4/3,'r') is '4/3', but sym(1+sqrt(5),'r') is
%  7286977268806824*2^(-51)
%
%  'e' stands for 'estimate error'  The 'r' form is supplemented
%  by a term involving the variable 'eps' which estimates
%  the difference between the theoretical rational expression
%  and its actual floating point value.  For example,
%  sym(3*pi/4)is 3*pi/4-103*eps/249.
%
%  'd' stands for 'decimal'   The number of digits is taken from
%  the current setting of DIGITS used by VPA. Using fewer
%  than 16 digits reduces accuracy, while more than 16 digits
%  may not be warranted. For example, with digits(10),
%  sym(4/3,'d') is 1.333333333, while with digits(20),
%  sym(4/3,'d') is 1.3333333333333332593, which does not end in
%  a string of 3's, but is an accurate decimal representation of
%  the floating point number nearest to 4/3.
%
%  See also SYMS, CLASS, DIGITS, VPA.
if nargin == 0
  % Default constructor
  S = class(struct('s','0'),'sym');
elseif isempty(x)
  % Empty sym: sym([])
  S = class(struct('s',{}),'sym');
elseif isa(x,'sym')
  switch nargin
  case 1
    % Already a sym object
    S = x;
  case 2
    % Real/unreal/positive
    if ~strcmp(a,'real') & ~strcmp(a,'unreal')  &..
                    ~strcmp(a,'positive')& ~strcmp(a,'negative')
      error(['Second argument ' a ' not recognized.']);
    elseif ~isvarname(char(x))
      error('Real/Unreal/Positive/Negative assumption applies
                                    only to symbolic variables.')
    elseif isequal(a,'real')
      maple('assume',x.s,'real');
    elseif isequal(a,'unreal')
      maple([x.s ':=''' x.s '''']);
    elseif isequal(a,'positive')
      maple(['assume(' x.s ' > 0);']);
    elseif isequal(a,'negative')
      maple(['assume(' x.s ' < 0);']);
    end
```

```
    case 3
      % Obsolete usage from version 1, sym(x,i,j)
      S = x(a,b);
    end
  elseif isa(x,'char')
    % Simple variable, Maple output or char(sym(matrix))
    S = char2sym(x);
    % Inform Maple of any real or unreal assumption.
    if nargin == 2
       if ~strcmp(a,'real') & ~strcmp(a,'unreal')  &...
                     ~strcmp(a,'positive') & ~strcmp(a,'negative')
        error(['Second argument ' a ' not recognized.']);
       elseif ~isvarname(x)
        error('Real/Unreal/Positive/Negative assumption
                              applies only to symbolic variables.')
       elseif isequal(a,'real')
        maple('assume',x,'real');
       elseif isequal(a,'unreal')
        maple([x ':=''' x '''']);
       elseif isequal(a,'positive')
        maple(['assume(' x ' > 0);']);
       elseif isequal(a,'negative')
        maple(['assume(' x ' < 0);']);
       end
    end
   elseif isa(x,'double')
    % Double scalar or matrix
    S = struct('s',cell(size(x)));
    if nargin < 2, a = 'r'; end
    for k = 1:prod(size(x))
      switch a
       case 'f'
         S(k).s = symf(x(k))
       case 'r'
         S(k).s = symr(x(k));
       case 'e'
         S(k).s = syme(x(k));
       case 'd'
         S(k).s = symd(x(k),digits);
       otherwise
         error(['Second argument   a   not recognized.']);
      end
    end;
    S = class(S,'sym');
  else
    error(['Conversion to ''sym'' from     class(x)
                                          is not possible.'])
  end


  function S = symf(x)
  %SYMF   Hexadecimal symbolic representation of floating
  %point numbers.
  if imag(x) > 0
    S = ['(' symf(real(x)) ')+(' symf(imag(x)) ')*i'];
  elseif imag(x) < 0
```

```
    S = ['(' symf(real(x)) ')-(' symf(abs(imag(x))) ')*i'];
  elseif isinf(x)
    if x > 0
       S = 'Inf'
    else
       S = '-Inf';
    end
  elseif isnan(x)
    S = 'NaN';
  elseif x == 0
    S = '0'
  else
    [f,e] = log2(abs(x));
    h = '0123456789abcdef';
    f = 2*f-1;
    S = blanks(13);
    for k = 1:13
      f = 16*f;
      d = fix(f);
      S(k) = h(d+1);
      f = f - d;
    end
    if x > 0
      S = ['''I.' S '''*2^(' num2str(e-1) ')'];
    else
      S = ['''-I.' S '''*2^(' num2str(e-1) ')'];
    end
  end


  function [S,err] = symr(x)
  % SYMR   Rational symbolic representation.
  if imag(x) > 0
    S = ['(' symr(real(x)) ')+(' symr(imag(x)) ')*i'];
  elseif imag(x) < 0
    S = ['(' symr(real(x)) ')-(' symr(abs(imag(x))) ')*i'];
  elseif isinf(x)
    if x > 0
      S = 'Inf';
    else
      S = '-Inf'
    end
  elseif isnan(x)
    S = 'NaN'
  else
    tol = 10*eps;
    % Not-too-big integer.
    if x == fix(x) & abs(x) < 1/eps
      S = int2str(x);
      err = 0;
      return
    end
    % Ratio of two modest integers.
    [n,d] = rat(x,tol*abs(x));
    if n ~= 0 & abs(n) <= 100000 & d <= 100000
      if abs(n/d-x) <= tol*abs(x)
        S = int2str(n);
        if d ~= 1, S = [S '/' int2str(d)]; end
        err = 1;
        return
      end
```

```
end
% pi times ratio of two modest integers.
[p,q] = rat(x/pi,tol*abs(x));
if p ~= 0 & abs(p) <= 100000 & q <= 100000
  if (p*pi/q-x) <= tol*abs(x)
    if p == 1
      S = 'pi';
    elseif p == -1
      S = '-pi'
    else
      S = [int2str(p) '*pi'];
    end
    if q ~= 1
      S = [S '/' int2str(q)];
    end
    err = 1;
    return
  end
end
% power of 2
p = log2(abs(x));
if abs(p-round(p)) <= 10*eps*abs(p)
  if p > 0
    S = ['2^' int2str(p)];
  else
    S = ['2^(' int2str(p) ')'];
  end
  if x < 0
    S = ['-' S];
  end
  err = 1;
  return
end
% power of 10
p = log10(abs(x));
if abs(p-round(p)) <= 10*eps*abs(p)
  if p > 0
    S = ['10^' int2str(p)];
  else
    S = ['10^(' int2str(p) ')'];
  end
  if x < 0
    S = ['-' S];
  end
  err = 1;
  return
end
% square root of the ratio of two modest integers.
if tol*abs(x*x) > realmin
  [p,q] = rat(x*x,tol*abs(x*x));
  if p > 0 & p <= 100000 & q > 0 & q <= 100000
    if (p/q-x*x) <= tol*abs(x*x)
      if q == 1
        S = ['sqrt(' int2str(p) ')'];
      else
        S = ['sqrt(' int2str(p) '/' int2str(q) ')'];
      end
      if x < 0
        S = ['-' S];
      end
      err = 1;
      return
```

```
      end
    end
  end
  % None of the above.
  % Extract floating point fraction and exponent.
  S = symfl(x);
  err = 0;
end



function S = syme(x)
%SYME   Symbolic representation with error estimate.
if imag(x) > 0
  S = ['(' syme(real(x)) ')+(' syme(imag(x)) ')*i'];
elseif imag(x) < 0
  S = ['(' syme(real(x)) ')-(' syme(abs(imag(x))) ')*i'];
elseif isinf(x)
  if x > 0
    S = 'Inf';
  else
    S = '-Inf'
  end
elseif isnan(x)
  S = 'NaN';
else
  [S,err] = symr(x);
  if err ~= 0
    err = eval(maple('evalf',['(' symfl(x) ')-(' S ')'],
                                                    '32'))/eps;
  end
  if err ~= 0
    [n,d] = rat(err,1.e-5);
    if n == 0 | abs(n) > 100000
      [n,d] = rat(err/x,1.e-3);
      if n > 0
        S = [S '*(1+' int2str(n) '*eps/' int2str(d) ')'];
    else
        S = [S '*(1' int2str(n) '*eps/' int2str(d) ')'];
    end
    return
  end
  if n == 1
    S = [S '+eps'];
  elseif n == -1
    S = [S '-eps'];
  elseif n > 0
    S = [S '+' int2str(n) '*eps'];
  else
    S = [S int2str(n) '*eps'];
  end
  if d ~= 1
    S = [S '/' int2str(d)];
    end
  end
end



function S = symd(x,d)
%SYMD   Decimal symbolic representation.
if imag(x) > 0
```

```
  S = ['(' symd(real(x),d) ')+(' symd(imag(x),d) ')*i'];
elseif imag(x) < 0
  S = ['(' symd(real(x),d) ')-(' symd(abs(imag(x)),d) ')*i'];
elseif isinf(x)
  if x > 0
    S = 'Inf'
  else
    S = '-Inf'
  end
elseif isnan(x)
  S = 'NaN'
else
  S = maple('evalf',symfl(x),int2str(d));
end



function f = symfl(x)
%SYMFL  Exact representation of floating point number.
[f,e] = log2(x);
f = 2^53*f;
e = e-53;
f = [int2str(f) '*2^(' int2str(e) ')'];



function S = char2sym(x)
%CHAR2SYM Convert a string, including Maple array output,
%to a sym.
if isempty(x) | all(x ==   ') | strcmp(x,'{}')
  S = class(struct('s',{}),'sym');
elseif ~( x(1)=='[' |
      (length(x)>7 & . .
      ~(isempty(findstr(x,'vector([')) &
       isempty(findstr(x,'matrix([')) &
       isempty(findstr(x,'array(['))))))
  x = trim(x);
  if isvarname(x) | strncmp(x,'Error',5) | strncmp(x,
                                                'at offset',9)
    % Variable names or Maple error messages are acceptable.
    S = class(struct('s',x),'sym')
  elseif x == ';' | x == ','
    % Ugly trick to provide backward compatibility for VI.
    % sym([ s1 ';' s2 ]) and sym([ s1 ',' s2 ])
    S = class(struct('s',{}),'sym');
  else
    % Check if string is a valid symbolic expression.
    [S,err] = maple(x);
    if ~err
      S = class(struct('s',x),'sym');
    elseif strncmp(S,'at offset',9)
      error([x ' is not a valid symbolic expression.']);
    elseif ~isempty(findstr(S,'division by zero')) |
        ~isempty(findstr(S,'is undefined'))
      S = sym(eval(x,'error(S)'));
    elseif ~isempty(findstr(S,'singularity'))
      S = eval(x,'error(S)');
      if abs(S) > 1/(100*eps)
        S = sym(sign(S)*Inf);
      else
        error(['A subexpression of   x   encounters
                                          a singularity.']);
```

```
      end
    else
      error(S)
    end
  end
else
  % First convert to MATLAB form
  if ~isempty(findstr(x,'matrix')) |
      ~isempty(findstr(x,'vector')) |
       ~isempty(findstr(x,'array'))
    x = map2mat(x);
  end


  % If the string is of the form, M = '[x - 1 x + 2; x * 3 x / 4]'
  % then find all of the alpha-numeric chararacters (id), the
  % arithmetic operators (+ - / * ^) (op), and spaces (sp) and
  % combine them into a vector V = 3*sp + 2*op + id. That is,
  % id = isalphanum(M); op = isop(M); sp = find(M == ' ')  Let
  % spaces receive the value 3, operators 2, and alpha-numeric
  % characters 1. Whenever the sequence 1 3 1 occurs, replace
  % it with 1 4 1. Insert a comma whenever the number 4 occurs.
  % First remove all multiple blanks to create at most
  % one blank.
  sp = (x == ' ');  % Location of all the spaces.
  [b,e] = findrun(sp); % Beginning (b) and end (e) indices.
  sp(b) = 0;  % Mark the beginning of multiple blanks.
  x(sp) = []; % Set multiple blanks to empty string.
  V = isalphanum(x) + 2*isop(x) + 3*(x == ' ');
  if length(V) >= 3
    d = V(2:end-1)==3 & V(1:end-2)==1 & V(3:end)==1;
    V(find(d)+1) = 4;
  end
  w = find(V == 4);
  x(w) = ',';
  % String contains square brackets, so it is not a scalar.
  if x(1) == '['
    % Make '[a11 a12      a21 a22          ]' look like Maple
                                                      array.
    % Version 1 compatability.  Possibly useful elsewhere.
    % Replace multiple blanks with single blanks.
    k = findstr(x,'  ');
    while ~isempty(k);
      x(k) = [];
      k = findstr(x,'  ');
    end
    Replace blanks surrounded by letters, digits or parens
    with commas.
    for k = findstr(x,' ');
      if (isalphanum(x(k-1)) | x(k-1) == ')') &
         (isalphanum(x(k+1)) | x(k+1) == '(')
        x(k) = ','
      end
    end
    % Replace semicolons with '],['
    for k = fliplr(findstr(';',x))
      x = [x(1:k-1) '],[' x(k+1:end)];
    end
  end
  % Deblank
  x(find(x==' ')) = [];
```

```
    % Output from maple('linsolve',sym(magic(8)),ones(8,1),
    % '''r''','s')
    % contains free variables s[i][j]. Replace them by by sij.
    k = find(x(1:end-1)==']' & x(2:end)=='[');
    for j = fliplr(k)
      x(j:j+1) = [];
    end
    k = find(isletter(x(1:end-1)) & x(2:end)=='[');
    for j = fliplr(k)
      x(j+min(find(x(j+1:end)==']'))) = [];
      x(j+1) = [];
    end



    % Create indices that delimit rows
    s = find(x == '[') + 1;
    e = find(x == ']') - 1;
    if strncmp(x,'array([[',8) | strncmp(x,'matrix([[',9)
      s(1) = [];
      e(end) = [];
    end
    for k = 1:length(s)
      % String with current row
      sk = x(s(k):e(k));
      % Eliminate commas within ()'s
      lp = find(sk == '(');
      rp = find(sk == ')');
      for i=1:length(lp)
        sk(lp(i):rp(i)) =
      end
      % Count commas to determine number of elements
      commas = find(sk == ',');
      if k == 1
        n = length(commas) + 1;
        S = struct('s',cell(k,n));
      elseif length(commas)+1 ~= n
        % disp(x)
        % error('All rows must have same numbe   of    ements.')
        S = class(struct('s',x),'sym');
        return
      end
      % Start and end of column elements
      sr = [1 commas+1];
      er = [commas-1 length(sk)];
      sk = x(s(k):e(k));
      for j=1:n
        S(k,j).s = sk(sr(j):er(j));
      end
    end
    S = class(S,'sym');
end



%-------------------------
function [b,e] = findrun(x)
%FINDRUN Finds the runs of like elements in a vector
%   FINDRUN(V) returns the beginning (b) and end (e
%   indices of the runs of like elements in the vec
d = diff([0 x 0]);
b = find(d == 1);
```

```
e = find(d == -1)   1;
%-------------------------
function B = isalphanum(S)
%ISAPLHANUM is True for alpha-numeric characters.
%   ISALPHANUM(S) returns 1 for alpha-numeric characters or
%   underscores and 0 otherwise.
%
%   Example:  S = 'x*exp(x - y) + cosh(x*s^2)'
%             isalphanum(S)   returns
%
(1,0,1,1,1,0,1,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,1,0,1,0)


B = isletter(S) | (S >= '0' & S <= '9') | (S ==  _');


%-------------------------
function B = isop(S)
%ISOP is True for + - * / or
%   ISOP(S) returns 1 for plus, minus, times, divide, or
%   exponentiation operators and 0 otherwise.
B = (S == '+') | (S == '-') | (S == '*') |
(S == '/') | (S == '^');


%-------------------------
function r = map2mat(r)
% MAP2MAT Maple to MATLAB string conversion.
%   MAP2MAT(r) converts the Maple string r containing
%   matrix, vector, or array to a valid MATLAB string.
%
%   Examples: map2mat(matrix([[a,b], [c,d]])  returns
%             [a,b;c,d]
%             map2mat(array([[a,b], [c,d]])  returns
%             [a,b;c,d]
%             map2mat(vector([[a,b,c,d]])  returns
%             [a,b,c,d]
% Deblank.
r(findstr(r,' ')) = [];
% Special case of the empty matrix or vector
if strcmp(r,'vector([])') | strcmp(r,'matrix([])') |
   strcmp(r,'array([])')
   r = [];
else
   % Remove matrix, vector, or array from the string.
   r = strrep(r,'matrix([[','['); r = strrep(r,'array([[','[');
   r = strrep(r,'vector([','['); r = strrep(r,'],[',';');
   r = strrep(r,']])',']'); r = strrep(r,'])',']');
end
%-------------------------
function s = trim(s);
%TRIM  TRIM(s) deletes any leading or trailing blanks.
% while s(1) == ' ', s(1) = []; end
% while s(end) == ' ', s(end) = []; end
s=fliplr(deblank(fliplr(deblank(s))));
```

Приложение 2

Список русскоязычных книг по MATLAB

1. Андриевский Б., Фрадков А. Избранные главы теории автоматического управления с примерами на языке MATLAB. СПб.: Наука, 1999.
2. Андриевский Б., Фрадков А. Элементы математического моделирования в программных средах MATLAB 5 и Scilab. СПб.: Наука. 2001.
3. Ануфриев И. Самоучитель MatLab 5.3/6.x (с дискетой). СПб.: БХВ-Петербург, 2002.
4. Веремей Е., Корчанов В., Коровкин М., Погожев С. Компьютерное моделирование систем управления движением морских подвижных объектов. СПб.: НИИ Химии СПбГУ, 2002.
5. Герман-Галкин С. Линейные электрические цепи: Лабораторные работы. (с дискетой) М.: Корона принт, 2002.
6. Герман-Галкин С. Компьютерное моделирование полупроводниковых систем в MATLAB 6.0 (с дискетой) М.: Корона принт, 2001.
7. Глушаков С., Жакин И., Хачиров Т. Математическое моделирование. Mathcad 2000. Matlab 5.3. М.: АСТ, 2001.
8. Говорухин В., Цибулин В. Компьютер в математическом исследовании: Maple, MATLAB, LaTeX. СПб: Питер, 2001.
9. Гультяев А. MATLAB 5.2. Имитационное моделирование в среде Windows. М.: Корона принт, 1999.
10. Гультяев. А. MATLAB 5.3. Имитационное моделирование в среде Windows. М.: Корона принт, 2001.
11. Гультяев А. Визуальное моделирование в среде MATLAB: Учебный курс. СПб: Питер, 2000.
12. Данилов А. Компьютерный практикум по курсу «Теория управления». Simulink-моделирование в среде MATLAB. М.: МГУИЭ, 2002.
13. Дорф Р., Бишоп Р. Современные системы управления: М.:Лаборатория базовых знаний, 2002.
14. Дьяконов В. MATLAB 6/6.1/6.5 + Simulink 4/5 в математике и моделировании. М.: СОЛОН-Пресс, 2003.
15. Дьяконов В. MATLAB 6/6.1/6.5 + Simulink 4/5. Основы применения. Полное руководство пользователя. (2-е изд.) М.: СОЛОН-Пресс, 2004.
16. Дьяконов. В. MATLAB 6. Учебный курс. СПб: Питер, 2001.
17. Дьяконов. В. MATLAB: Учебный курс. СПб: Питер, 2000.
18. Дьяконов В. Simulink 4. Специальный справочник. СПб: Питер, 2001.
19. Дьяконов В. VisSim+Mathcad+MATLAB. Визуальное математическое моделирование. М.: СОЛОН-Пресс, 2004.
20. Дьяконов В., Абраменкова. И. MATLAB 5. Система символьной математики. М.: Нолидж, 1999.

21. Дьяконов В., Абраменкова И. MATLAB. Обработка сигналов и изображений. Специальный справочник. СПб: Питер. 2002.
22. Дьяконов В., Абраменкова И., Круглов В. MATLAB с пакетами расширений. М.: Нолидж, 2000.
23. Дьяконов В. Вейвлеты. От теории к практике. М.: Солон-Р, 2002.
24. Дьяконов В. Компьютерная математика. Теория и практика. М.: Нолидж, 2000.
25. Дьяконов В., Круглов В. MATLAB. Анализ, идентификация и моделирование систем. Специальный справочник. СПб: Питер, 2001.
26. Дьяконов В., Круглов. В. Математические пакеты расширения MATLAB. Специальный справочник. СПб: Питер, 2001.
27. Дэбни Дж., Харман Т. Simulink 4. Секреты мастерства. М.: Бином, 2003.
28. Кетков Ю., Кетков А., Шульц М. MATLAB 6.x: программирование численных методов. С-Пб.: БХВ-Петербург, 2004.
29. Кондрашов В., Королев С. MATLAB как система программирования научно-технических расчетов. М.: Мир, Институт стратегической стабильности Минатома РФ, 2002.
30. Кривилев А. Основы компьютерной математики с использованием системы MATLAB. М.: Лекс-Книга, 2005.
31. Круглов В., Дли М., Голунов Р. Нечеткая логика и искусственные нейронные сети. М.:Физматлит, 2001.
32. Лазарев Ю. Matlab 5.x. Киев: BHV-Киев, 2000.
33. Лавров К., Цыплякова Т. Финансовая аналитика. MATLAB 6. М.: Диалог-МИФИ, 2001.
34. Леоненков А. Нечеткое моделирование в среде MATLAB и fuzzyTECH. СПб.: БХВ-Санкт-Петербург, 2003.
35. Мартынов Н.Н. MATLAB 7. Элементарное введение. М.: Кудиц-образ, 2005.
36. Мартынов Н. Введение в MATLAB 6. М.:Кудиц-образ, 2002.
37. Мартынов Н., Иванов А. Matlab 5.x. Вычисления, визуализация, программирование. М.: Кудиц-образ, 2000.
38. Медведев В., Потемкин В. Control System Toolbox. MATLAB 5 для студентов. М: Диалог-МИФИ. 2000.
39. Медведев В., Потемкин В. Нейронные сети. MATLAB 6. Диалог-МИФИ. 2002.
40. Минаев Ю., Филимонова О., Бенамеур Лиес Методы и алгоритмы решения задач идентификации и прогнозирования в условиях неопределенности в нейросетевом логическом базисе. М.: Горячая линия — Телеком, 2003.
41. Мэтьюз Дж. Г., Финк К.Д. Численные методы. Использование Matlab. М.: Мир, 2001.
42. Новгородцев А. Расчет электрических цепей в MATLAB. СПб: Питер, 2004.
43. Потемкин. В. MATLAB 6: Среда проектирования инженерных приложений. М.: Диалог-МИФИ, 2003.
44. Потемкин В. Введение в MATLAB. М.: Диалог-МИФИ, 2000.
45. Потемкин В. Вычисления в среде MATLAB. М.: Диалог-МИФИ, 2004.
46. Потемкин. В. Инструментальные средства Matlab 5.x. М.: Диалог-МИФИ, 2000.
47. Потемкин В. Система Matlab 5 для студентов. М.: Диалог-МИФИ, 1999.
48. Потемкин. В. Система MATLAB. Справочное пособие. М: Диалог-МИФИ, 1997.
49. Потемкин В. Система инженерных и научных расчетов MATLAB 5.x (в 2-х томах). М.: Диалог-МИФИ, 1999.
50. Поршнев С. Вычислительная математика. Курс лекций. СПб.: БХВ-Петербург, 2004.
51. Поршнев С. Компьютерное моделирование физических процессов в пакете MATLAB. М.: Горячая линия-Телеком, 2003.
52. Рудаков П., Сафонов И. Обработка сигналов и изображений Matlab 5.x. М.: Диалог-МИФИ, 2000.
53. Семененко М. Введение в математическое моделирование. М.:Солон-Р, 2002.
54. Сергиенко А. Цифровая обработка сигналов. СПб.: Питер, 2002.