Е. Р. Алексеев
О. В. Чеснокова
Е. А. Рудченко

Scilab

решение инженерных
и математических задач

CD

библиотека alt linux

В серии: Библиотека ALT Linux

Scilab

Решение инженерных и математических задач

Е. Р. Алексеев,
О. В. Чеснокова,
Е. А. Рудченко

Москва
ALT Linux; БИНОМ. Лаборатория знаний
2008

УДК 004.67
ББК 22.1
А47

Алексеев Е. Р.

А47 Scilab: Решение инженерных и математических задач / Е. Р. Алексеев, О. В. Чеснокова, Е. А. Рудченко. — М. : ALT Linux ; БИНОМ. Лаборатория знаний, 2008. — 260 с. : ил. ; 8 с. цв. вклейки.— (Библиотека ALT Linux).

ISBN 978-5-94774-890-1

Книга посвящена свободно распространяемому математическому пакету Scilab. Описаны графические возможности пакета (построение графиков и диаграмм), возможности программирования в среде пакета. Подробно рассмотрено решение математических задач (нелинейные уравнения и системы, задачи линейной алгебры, задачи оптимизации, дифференцирование и интегрирование, задачи обработки экспериментальных данных: интерполяция и аппроксимация, метод наименьших квадратов, обыкновенные дифференциальные уравнения и системы, уравнения в частных производных). К книге прилагается Live CD ALTLinux 4.0 Junior, содержащий программу Scilab 4.1.1.

Сайт книги: `http://books.altlinux.ru/altlibrary/scilab`

Книга адресована студентам и преподавателям математических и инженерных специальностей и научным сотрудникам.

УДК 004.67
ББК 22.1

По вопросам приобретения обращаться:
«БИНОМ. Лаборатория знаний»
(499) 157-52-72. E-mail: binom@Lbz.ru
http://www.Lbz.ru

ISBN 978-5-94774-890-1

Оглавление

Введение

Авторы давно хотели написать книгу, посвященную инженерным и научным расчетам с помощью свободно распространяемых программ. Мы остановили свой выбор на свободно распространяемой системе компьютерной математики Scilab. Scilab предназначен для выполнения инженерных и научных вычислений. По своим *возможностям* пакет Scilab сопоставим с известным математическим пакетом Mathcad, а по своему интерфейсу похож на пакет MATLAB. Однако при этом пакет Scilab — свободно распространяемая программа, а значит бесплатная для конечного пользователя. Существуют версии Scilab для различных операционных систем: для ОС Linux, ОС семейства Windows (в том числе и для MS Windows Vista) и даже для MacOS. На момент написания книги последней была версия пакета 4.1.1. Именно на базе ее и была написана книга. Последнюю версию пакета всегда можно скачать на официальном сайте программы `www.scilab.org`. К сожалению очень мало русскоязычной литературы, посвященной Scilab. Вообще отсутствуют книги на русском языке для пользователей, начинающих осваивать пакет. Написанная авторами книга призвана заполнить этот пробел. Авторы рекомендуют два русскоязычных сайта, посвященные пакету Scilab:

- сайты авторов книги `http://scilab.land.ru`, `http://teacher.dn-ua.com`, на которых можно найти большинство материалов книги в форматах *odt* [1] и *pdf*.
- сайт М. И. Павловой `http://www.csa.ru/~zebra/my_scilab/index.html`, с которого, собственно, и началось знакомство авторов книги с пакетом.

Много ссылок на литературу на разных языках можно найти на странице официального сайта Scilab `http://www.scilab.org/publications/index_publications.php?page=books.html`.

Входной язык системы Scilab позволяет не только использовать встроенные команды, но и разрабатывать собственные визуальные приложения.

[1] Файлы с расширением *odt* — текстовые документы в стандартном формате Open Document, их можно редактировать с помощью многих программ, шире всего для этого используется офисный пакет OpenOffice.org (`http://ru.openoffice.org`).

В книге на практических примерах рассмотрены основные принципы работы в Scilab. Книга состоит из тринадцати глав и приложения.

Первая глава является своеобразным введением в Scilab. В ней описаны основные возможностей пакета, системные требования к компьютеру и операционной системе, уделено внимание особенностям установки Scilab 4.1.1 на ПК. Кроме того, глава знакомит читателя со средой пакета и его главным меню.

Вторая глава посвящена основам работы в среде Scilab. Описана работа с переменными, основные операции и функции.

В третьей главе описана работа с массивами и матрицами в Scilab, рассматриваются возможности Scilab при решении задач линейной алгебры.

В четвёртой главе речь идет о графических возможностях Scilab. Подробно описаны возможности пакета для построения двумерных графиков в декартовой и полярной системах координат, а также графиков, заданных параметрически. Рассмотрены возможности форматирования двумерных графиков.

Возможности пакета для работы с трехмерными графиками описаны *в пятой главе*.

Шестая глава знакомит читателя с различными способами решения нелинейных уравнений и систем в Scilab.

В седьмой главе внимание уделено вычислительным методам интегрирования и дифференцирования, а в *восьмой главе* описано решение обыкновенных дифференциальных уравнений.

Девятая глава посвящена возможностям программирования в системе Scilab. Подробно описаны все операторы встроенного в Scilab языка программирования. Глава содержит большое количество примеров программ.

Десятая глава познакомит читателя с визуальным программированием. В ней изложены необходимые сведения об объектах и способах работы с ними.

В одиннадцатой главе описаны задачи обработки результатов эксперимента.

В двенадцатой главе речь идет о дифференциальных уравнениях в частных производных, описан метод сеток численного решения дифференциальных уравнений в частных производных и представлена его реализация в Scilab.

В тринадцатой главе вниманию читателя представлены задачи оптимизации функций одной и многих переменных без ограничений, а так же задачи линейного программирования.

В приложении приведены задания для самостоятельного решения.

Мы надеемся, что книгу можно будет использовать при изучении курсов информатики, вычислительной математики и др.

К книге прилагается CD диск.

Авторы благодарят компанию ALT Linux и лично Кирилла Маслинского за возможность издать книгу, посвященную пакету Scilab.

Е. Р. Алексеев, О. В. Чеснокова, Е. А. Рудченко
Донецк, сентябрь, 2007

Глава 1

Пакет Scilab. Начало работы

Scilab — это система компьютерной математики, которая предназначена для выполнения инженерных и научных вычислений, таких как:

- решение нелинейных уравнений и систем;
- решение задач линейной алгебры;
- решение задач оптимизации;
- дифференцирование и интегрирование;
- обработка экспериментальных данных (интерполяция и аппроксимация, метод наименьших квадратов);
- решение обыкновенных дифференциальных уравнений и систем.

Кроме того, Scilab предоставляет широкие возможности по созданию и редактированию различных видов графиков и поверхностей.

Несмотря на то, что система Scilab содержит достаточное количество встроенных команд, операторов и функций, отличительная ее черта — это гибкость. Пользователь может создать любую новую команду или функцию, а затем использовать ее наравне со встроенными. К тому же, система имеет достаточно мощный собственный язык программирования высокого уровня, что говорит о возможности решения новых задач.

1.1 Установка Scilab на ПК

Свободно распространяемую версию пакета вместе с полной документацией на английском языке можно получить на сайте программы `www.scilab.org`.

Существуют версии Scilab для операционных систем Windows и Linux. Они имеют некоторые отличия в названиях пунктов главного меню, но команды пакета в обеих версиях идентичны.

1.1.1 Установка Scilab под управлением Windows

Рассмотрим особенности установки пакета для операционной системы Windows. Для того, чтобы установить Scilab-4.1.1 на ПК, необходимо обратиться к одноименному выполняемому файлу, после чего начнет свою работу *Мастер установки*. В первом окне *Мастера установки* нужно выбрать язык (английский или французский) и нажать кнопку OK для продолжения установки. Следующее окно является информационным. Пользователь получает сообщение о том, что на его компьютер будет установлен пакет Scilab версии 4.1.1 и рекомендацию закрыть другие приложения перед установкой. Для перехода к третьему окну *Мастера установки* используют кнопку Next. В этом окне следует принять условия лицензионного соглашения (*I accept the agreement*) и нажать клавишу Next для продолжения. На следующем этапе пользователю будет предложено выбрать путь для установки пакета. По умолчанию это папка *C:\Program Files\scilab-4.1.1*. Другой путь можно установить при помощи кнопки Browse.... Кроме того, в этом окне выводится информация о количестве места на выбранном диске, требуемого для установки стандартного набора компонентов системы. Нажатие кнопки Next приведет к появлению диалогового окна, представленного на рис. 1.1. Здесь пользователю будет предложено выбрать один из четырех типов установки:

- установка по умолчанию (*Installation Default*);
- полная установка (*Full installation*);
- компактная установка (*Compact installation*);
- установка выбранных компонентов (*Custom installation*).

Далее будет приведен список компонентов, соответствующих выбранному типу установки. Переход к следующему окну *Мастера установки* осуществляется с помощью кнопки Next.

Следующее окно мастера установки сообщает пользователю о том, что после установки в меню *Пуск* будет создан ярлык, предназначенный для запуска Scilab. По умолчанию ему будет присвоено имя scilab-4.1.1. Изменить параметры ярлыка в меню Пуск можно при помощи кнопки Browse.... Нажатие кнопки

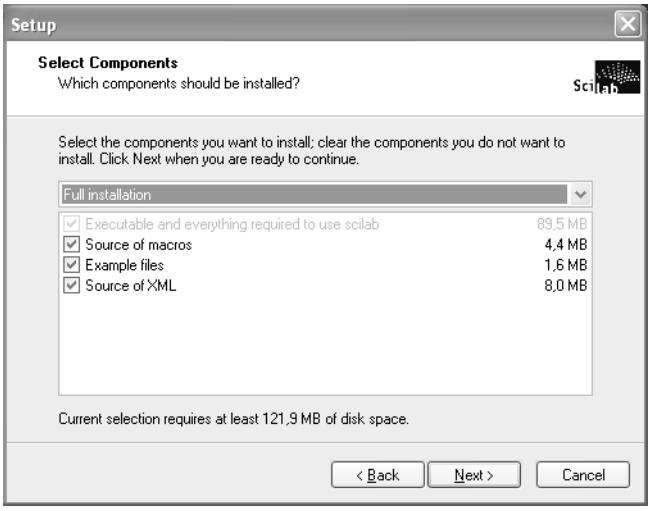

Рис. 1.1. Окно выбора компонентов для установки Scilab

Next приведет к появлению следующего окна, в котором *Мастер установки* предложит список дополнительных задач, доступных после установки. Для переходя к следующему шагу также необходимо выбрать Next.

Перед началом установки *Мастер* выдает краткий отчет о параметрах установки. Выбор кнопки Install приведет к началу процесса установки. Процесс установки Scilab заключается в копировании файлов системы на жесткий диск. Контролировать его пользователь может с помощью так называемого линейного индикатора. Завершается процесс установки появлением информационного окна. Нажатие кнопки Next приведет к последнему шагу, где у пользователя будет возможность установить единственную опцию *Launch scilab*. Если этот параметр активен, то Scilab запустится сразу после нажатия кнопки *Finish*, иначе запуск можно произвести из главного меню (*Пуск – Программы – scilab-4.1.1*) или с помощью ярлыка на рабочем столе (рис. 1.2).

Рис. 1.2. Ярлык для запуска Scilab

1.1.2 Установка Scilab под управлением Linux

Scilab входит в состав таких дистрибутивов, как ALT Linux 4.0 и Mandrivia 2007. Пакет есть в репозиториях Debian и Ubuntu. Для установки в другие дистрибутивы Linux необходимо загрузить текущую версию Scilab с сайта `http://www.scilab.org`, развернуть файл в какой либо каталог (например */usr/lib*) и, войдя в каталог Scilab с привилегиями суперпользователя *root*, выполнить команду `make`. После этого командой */usr/lib/Scilab 4/bin/scilab* можно запускать программу на выполнение.

1.2 Среда Scilab

После запуска Scilab на экране появиться *основное окно приложения*. Окно содержит *меню*, *панель инструментов* и *рабочую область*. Признаком того, что система готова к выполнению команды, является наличие знака приглашения `-->`, после которого расположен активный (мигающий) курсор. Рабочую область со знаком приглашения обычно называют *командной строкой*. *Ввод команд* в Scilab осуществляется с клавиатуры. Нажатие клавиши Enter заставляет систему выполнить команду и вывести результат (рис. 1.3).

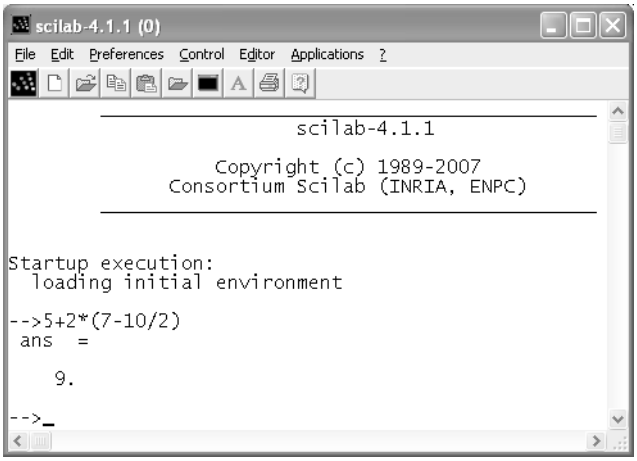

Рис. 1.3. Выполнение элементарной команды в Scilab

Понятно, что все выполняемые команды не могут одновременно находиться в поле зрения пользователя. Поэтому, просмотреть ту информацию, которая покинула видимую часть окна, можно, если воспользоваться стандартными средствами просмотра, например полосами прокрутки или клавишами перемещения курсора Page Up, Page Down.

Клавиши «Стрелка вверх» и «Стрелка вниз» также управляют курсором, однако в Scilab они имеют другое назначение. Эти клавиши позволяют вернуть в

командную строку ранее введенные команды или другую входную информацию, так как вся эта информация сохраняется в специальной области памяти. Так, если в пустой активной командной строке нажать клавишу ↑, то появится последняя вводимая команда, повторное нажатие вызовет предпоследнюю и так далее. Клавиша ↓ выводит команды в обратном порядке. Таким образом, можно сказать, что вся информация в рабочей области находится или в *зоне просмотра* или в *зоне редактирования*.

Важно знать, что в *зоне просмотра* нельзя ничего исправить или ввести. Единственная допустимая операция, кроме просмотра, это выделение информации с помощью мыши и копирование ее в буфер обмена, например, для дальнейшего помещения в командную строку.

Зона редактирования — это фактически командная строка. В ней действуют элементарные приемы редактирования: → — перемещение курсора вправо на один символ; ← — перемещение курсора влево на один символ; ***Home*** — перемещение курсора в начало строки; ***End*** — перемещение курсора в конец строки; ***Del*** — удаление символа после курсора; ***Backspace*** — удаление символа перед курсором.

Кроме того, существуют особенности *ввода команд*. Если команда заканчивается точкой с запятой «;», то результат ее действия не отображается в командной строке. В противном случае, при отсутствии знака «;», результат действия команды сразу же выводится в рабочую область (листинг 1.1).

Листинг 1.1. Использование «точки с запятой» в Scilab

```
-->2.7*3+3.14/2
 ans  =
    9.67
-->2.7*3+3.14/2;
-->
```

Текущий документ, отражающий работу пользователя с системой Scilab, содержащий строки ввода, вывода и сообщения об ошибках, принято называть *сессией*. Значения всех переменных, вычисленные в течение текущей сессии, сохраняются в специально зарезервированной области памяти, называемой рабочим пространством системы. При желании определения всех переменных и функций, входящих в текущую сессию можно сохранить в виде файла, саму сессию сохранить нельзя.

1.3 Основные команды главного меню Scilab

Главное меню системы содержит команды, предназначенные для работы с файлами, настройки среды, редактирования команд текущей сессии и получения справочной информации. Кроме того, с помощью главного меню можно создавать, редактировать, выполнять отладку и запускать на выполнение так называемые файлы-сценарии Scilab, а также работать с графическими приложениям пакета.

1.3.1 Работа с файлами

Пункт меню *File* предназначен для работы с файлами. Рассмотрим назначение представленных в нем команд[1]:

- *New Scilab* — открывает новое окно Scilab, фактически пакет запускается повторно;
- *Exec...* — запуск на выполнение созданной ранее Scilab-программы (файлы с расширением `sce` или `sci`);
- *Open* — открывает окно для загрузки созданного ранее файла, рисунка или модели;
- *Load* — открывает окно для загрузки файлов, информация в которых хранится в виде машинных кодов; при их открытии в память компьютера загружаются определенные ранее переменные и функции;
- *Save* — сохранение всех определенных в данной сессии переменных и функций в виде файла с расширением `sav` или `bin`;
- *Change Directory* — смена текущего каталога, выводит окно настройки путей файловой системы;
- *Get Change Directory* — выводит в командную строку имя текущего каталога;
- *Print Setup...* — выводит окно настройки параметров печати;
- *Print* — печать текущей сессии;
- *Exit* — выход из системы Scilab.

1.3.2 Редактирование команд текущей сессии

Пункт меню *Edit* содержит следующие команды:

- *Select All* — выделение всех команд текущей сессии;
- *Copy* — копирование выделенного объекта в буфер;
- *Paste* — вставка объекта из буфера;
- *Empty Clipboard* — очистка буфера обмена;
- *History* — группа команд, предназначенных для редактирования командной строки.

[1] Ниже описаны операции с файлами под управлением ОС Windows, в ОС Linux для выполнения операций с файлами необходимо выполнить команду *File – File Operations*.

1.3.3 Настройка среды

Команды настройки среды пакета представлены в меню *Preferences*:

- *Language* — предлагает выбрать из списка язык интерфейса (английский, французский);
- *Colors* — позволяет установить цвет шрифта (*Text*), цвет фона (*Background*) или цвета, принятые по умолчанию (*Default System Colors*);
- *Toolbar (F3)* — выводит или удаляет панель инструментов;
- *Files Association* — предлагает установить типы поддерживаемых файлов;
- *Choose Font* — выполняет настройки шрифта (гарнитура, начертание, размер);
- *Clear History* — очищает рабочее пространство;
- *Clear Command Window (F2)* — очищает рабочее окно;
- *Consol (F12)* — активизирует консольное приложение.

1.3.4 Справочная система

Команда главного меню *?* открывает доступ к справочной системе Scilab. В справочной системе информацию можно искать, воспользовавшись содержанием, в списке, упорядоченном по алфавиту, по ключевому слову или фразе.

С помощью команды *Scilab Demos* можно осуществить просмотр демонстрационных примеров.

1.3.5 Редактирование и отладка файлов-сценариев

Файл-сценарий — это список команд Scilab, сохраненный на диске. Для подготовки, редактирования и отладки файлов-сценариев служит специальный *редактор* SciPad, который можно вызвать, выполнив команду главного меню *Editor*. В результате работы этой команды будет создан новый файл-сценарий. По умолчанию он имеет имя `Untitled1.sce`.

Окно редактора файлов-сценариев выглядит стандартно, т.е. имеет заголовок, меню, панели инструментов, строку состояния.

Ввод текста в окно редактора файла-сценария осуществляется по правилам, принятым для команд Scilab. Рис. 1.4 содержит пример ввода команд для решения квадратного уравнения $3x^2 + 5x + 4 = 0$. Нетрудно заметить, что точка с запятой «;» ставится после тех команд, которые не требуют вывода значений.

Для *сохранения* введенной информации необходимо выполнить команду *File – Save* из меню редактора. Если информация сохраняется впервые, то появится окно *Save file As...*. Ввод имени в поле *File Name* и щелчок по кнопке *Save*

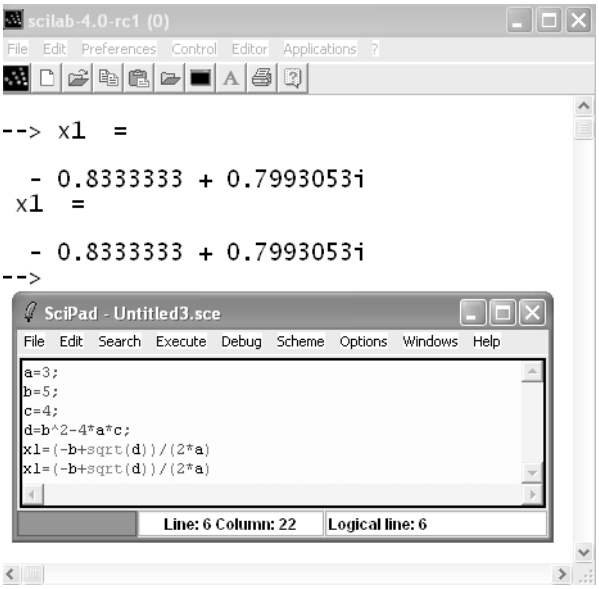

Рис. 1.4. Выполнение файла-сценария Scilab

приведет к сохранению информации, находящейся в окне редактора. Файлы-сценарии сохраняют с расширением `.sce`. *Открывает* ранее созданный файл команда главного меню *File – Open*.

Выполнить операторы файла-сценария можно несколькими способами:

- из меню редактора SciPad вызвать команду *Execute – Load into Scilab*;
- из главного меню Scilab вызвать команду *Exec* и указать имя файла-сценария.

Все эти действия приведут к появлению в рабочей области результатов вычислений команд файла-сценария (рис. 1.4).

Отметим, что редактор SciPad имеет возможность работы с множеством окон (пункт меню *Windows*), обладает принятыми для текстовых редакторов приемами редактирования (пункт меню *Edit*) и поиска (пункт меню *Search*). Кроме того, можно выполнить настройку среды редактора SciPad (пункт меню *Options*), вызвать справочную информацию (пункт меню *Help*) и осуществить отладку программы, набранной в редакторе (пункт меню *Debug*).

Выйти из режима редактирования можно, просто закрыв окно SciPad или выполнив команду *File – Exit*.

Глава 2

Основы работы в Scilab

2.1 Текстовые комментарии

Текстовый комментарий в Scilab — это строка, начинающаяся с символов //. Использовать текстовые комментарии можно как в рабочей области, так и в тексте файла-сценария. Строка после символов // не воспринимается как команда, и нажатие клавиши ***Enter*** приводит к активизации следующей командной строки:

Листинг 2.1. Пример использования комментария

```
-->//6+8
-->
```

2.2 Элементарные математические выражения

Для выполнения простейших *арифметических операций* в Scilab применяют следующие операторы: + сложение, - вычитание, * умножение, / деление слева направо, \ деление справа налево, ^ возведение в степень.

Вычислить значение арифметического выражения можно, если ввести его в командную строку и нажать клавишу ***Enter***. В рабочей области появится результат:

Листинг 2.2. Пример арифметического выражения

```
--> 2.35*(1.8-0.25)+1.34^2/3.12
ans =
4.2180
```

Если вычисляемое выражение *слишком длинное*, то перед нажатием клавиши ***Enter*** следует набрать три или более точек. Это будет означать продолжение командной строки:

Листинг 2.3. Выражение, расположенное на нескольких строках

```
--> 1+2+3+4+5+6....
7+8+9+10+....
+11+12+13+14+15
ans =
   120
```

Если символ точки с запятой «;» указан в конце выражения, то результат вычислений не выводится, а активизируется следующая командная строка:

Листинг 2.4. Использование «;»

```
--> 1+2;
--> 1+2
ans =
3
```

2.3 Переменные в Scilab

В рабочей области Scilab можно определять *переменные*, а затем использовать их в выражениях. Любая переменная до использования в формулах и выражениях должна быть определена. Для *определения переменной* необходимо набрать имя переменной, символ «=» и значение переменной. Здесь знак равенства — это *оператор присваивания*, действие которого не отличается от аналогичных операторов языков программирования. Т.е., если в общем виде оператор присваивания записать как

```
имя_переменной = значение_выражения
```

то в **переменную**, имя которой указано слева, будет записано **значение выражения**, указанного справа.

Имя переменной не должно совпадать с именами встроенных процедур, функций и встроенных переменных системы и может содержать до 24 символов. Система различает большие и малые буквы в именах переменных. Т.е. `ABC`, `abc`, `Abc`, `aBc` — это имена разных переменных. Выражение в правой части оператора присваивания может быть числом, арифметическим выражением, строкой символов или символьным выражением. Если речь идет о символьной или строковой переменной, то выражение в правой части оператора присваивания следует брать в одинарные кавычки.

Если символ «;» в конце выражения отсутствует, то в качестве результата выводится имя переменной и ее значение. Наличие символа «;» передает управление

следующей командной строке. Это позволяет использовать имена переменных для записи промежуточных результатов в память компьютера:

Листинг 2.5. Примеры определения переменных

```
-->//-----------------------------------------------
-->//Присваивание значений переменным a и b
--> a=2.3
a =
2.3000
--> b=-34.7
b =
-34.7000
-->//Присваивание значений переменным x и y,
-->//вычисление значения переменной  z
--> x=1;y=2; z=(x+y)-a/b
z =
3.0663
-->//Сообщение об ошибке - переменная c не определена
--> c+3/2
??? Undefined function or variable 'c'.
-->//-----------------------------------------------
-->//Определение символьной переменной
--> c='a'
c =
a
 -->//Определение строковой переменной
--> h='мама мыла раму'
h =мама мыла раму
```

Для очистки значения переменной можно применить команду

```
clear имя_переменной;
```

которая отменяет определения всех переменных данной сессии. Далее приведены примеры применения этой команды:

Листинг 2.6. Пример использования команды `clear`

```
-->//Определение переменных x и y
--> x=3; y=-1;
-->//Отмена определения переменной x
--> clear x
-->//Переменная x не определена
--> x
??? Undefined function or variable 'x'.
```

```
-->//Переменная y определена
--> y
y =
    -1
-->//Определение переменных a и b
-->a=1;b=2;
-->//Отмена определения переменных a и b
-->clear;
-->//Переменные a и b не определены
-->a
  !--error 4
undefined variable : a
-->b
  !--error 4
undefined variable : b
```

2.4 Системные переменные Scilab

Если команда не содержит знака присваивания, то по умолчанию вычисленное значение присваивается специальной *системной переменной* `ans`. Причем полученное значение можно использовать в последующих вычислениях, но важно помнить, что значение `ans` изменяется после каждого вызова команды без оператора присваивания:

Листинг 2.7. Пример работы с переменной `ans`

```
--> 25.7-3.14
ans =
22.5600
--> //Значение системной переменной равно 22.5600
--> 2*ans
ans =
45.1200
--> //Значение системной переменной увеличено вдвое
--> x=ans^0.3
x =
    3.1355
--> ans
ans =   45.1200
--> //После использования в выражении значение
--> //системной переменной не изменилось и равно 45.1200
```

Результат последней операции без знака присваивания хранится в переменной `ans`. Другие *системные переменные* в Scilab начинаются с символа `%`:

`%i` — мнимая единица ($\sqrt{-1}$);

`%pi` — число $\pi = 3.141592653589793$;

`%e` — число $e = 2.7182818$;

`%inf` — машинный символ бесконечности (∞);

`%NaN` — неопределенный результат ($0/0$,∞/∞ и т. п.);

`%eps` — условный ноль `%eps=2.220E-16`.

Все перечисленные переменные можно использовать в математических выражениях:

Листинг 2.8. Использование встроенных переменных

```
-->a=5.4;b=0.1;
-->F=cos(%pi/3)+(a-b)*%e^2
F  =     39.661997
```

Далее показан пример неверного обращения к системной переменной:

Листинг 2.9. Неправильное обращение к переменной `%pi`

```
-->sin(pi/2)
      !--error 4
undefined variable : pi
```

2.5 Ввод вещественного числа и представление результатов вычислений

Числовые результаты могут быть представлены с плавающей (например, `-3.2E-6`, `-6.42E+2`) или с фиксированной (например, `4.12`, `6.05`, `-17.5489`) точкой. Числа в формате с плавающей точкой представлены в экспоненциальной форме `mE±p`, где `m` — мантисса (целое или дробное число с десятичной точкой), `p` — порядок (целое число). Для того, чтобы перевести число в экспоненциальной форме к обычному представлению с фиксированной точкой, необходимо мантиссу умножить на десять в степени порядок.

Например,

`-6.42E+2` $= -6.42 \cdot 10^2 = -642$ `3.2E-6` $= 3.2 \cdot 10^{-6} = 0.0000032$

При вводе вещественных чисел для отделения дробной части используется точка. Примеры ввода и вывода вещественных чисел:

Листинг 2.10. Примеры определения вещественных чисел

```
-->0.123
 ans  =   0.123
-->-6.42e+2
 ans  =   - 642.
-->3.2e-6
 ans  =   0.0000032
```

Рассмотрим пример вывода значения системной переменной π и некоторой переменной q, определенной пользователем:

Листинг 2.11. Вывод вещественных чисел

```
-->%pi
 %pi  =
    3.1415927
-->q=0123.4567890123456
 q  =
    123.45679
```

Нетрудно заметить, что Scilab в качестве результата выводит только восемь значащих цифр. Это формат вывода вещественного числа по умолчанию. Для того, чтобы контролировать количество выводимых на печать разрядов, применяют команду `printf` с заданным форматом, который соответствует правилам, принятым для этой команды в языке C:

Листинг 2.12. Вывод вещественных чисел с использованием функции `printf`

```
-->printf("%1.12f",%pi)
3.141592653590
-->printf("%1.15f",%pi)
3.141592653589793
-->printf("%1.2f",q)
123.46
-->printf("%1.10f",q)
123.4567890123
-->//По умолчанию 6 знаков после запятой
-->printf("%f",q)
123.456789
```

2.6 Функции в Scilab

Все *функции*, используемые в Scilab, можно разделить на два класса:

- *встроенные*;
- *определенные пользователем.*

В общем виде *обращение к функции* в Scilab имеет вид:

```
имя_переменной = имя_функции(переменная1 [,переменная2, ...])
```

где `имя_переменной` — переменная, в которую будут записаны результаты работы функции; этот параметр может отсутствовать, тогда значение, вычисленное функцией, будет присвоено системной переменной `ans`;

`имя_функции` — имя встроенной или ранее созданной пользователем функции;

`переменная1, переменная2, ...` — список аргументов функции.

2.6.1 Элементарные математические функции

Пакет Scilab снабжен достаточным количеством всевозможных встроенных функций, знакомство с которыми будет происходить в следующих разделах. Здесь приведем только элементарные математические функции, используемые чаще всего (табл. 2.1).

Таблица 2.1. Элементарные математические функции

Функция	***Описание функции***
Тригонометрические	
`sin(x)`	синус числа x
`cos(x)`	косинус числа x
`tan(x)`	тангенс числа x
`cotg(x)`	котангенс числа x
`asin(x)`	арксинус числа x
`acos(x)`	арккосинус числа x
`atan(x)`	арктангенс числа x
Экспоненциальные	
`exp(x)`	Экспонента числа x
`log(x)`	Натуральный логарифм числа x
Другие	
`sqrt(x)`	корень квадратный из числа x
`abs(x)`	модуль числа x
`log10(x)`	десятичный логарифм от числа x
`log2(x)`	логарифм по основанию два от числа x

Пример вычисления значения выражения $z = \sqrt{\left|\sin\left(\frac{x}{y}\right)\right|} \cdot e^{x^y}$:

Листинг 2.13. Вычисление математического выражения

```
-->x=1.2;y=0.3;
-->z=sqrt(abs(sin(x/y)))*exp(x^y)
 z  =
    2.5015073
```

2.6.2 Функции, определенные пользователем

В первой главе мы уже упоминали о *файлах-сценариях* и даже создавали небольшую программу, которая решала конкретное квадратное уравнение. Но в эту программу невозможно было передать входные параметры, т.е. это был обычный список команд, воспринимаемый системой как единый оператор.

Функция, как правило, предназначена для неоднократного использования, она имеет входные параметры и не выполняется без их предварительного задания. Рассмотрим несколько способов *создания функций* в Scilab.

Первый способ — это применение оператора `deff`, который в общем виде можно записать так:

```
deff('[имя1,...,имяN] = имя_функции(переменная_1,...,переменная_M)',
                 'имя1=выражение1;...;имяN=выражениеN')
```

где `имя1,...,имяN` — список выходных параметров, то есть переменных, которым будет присвоен конечный результат вычислений,

`имя_функции` — имя с которым эта функция будет вызываться,

`переменная_1,...,переменная_M` — входные параметры.

Далее приведен самый простой способ применения оператора `deff`. Здесь показано, как создать и применить функцию для вычисления выражения $z = \sqrt{\left|\sin\left(\frac{x}{y}\right)\right|} \cdot e^{x^y}$ (значение этого выражения уже было вычислено в листинге 2.13):

Листинг 2.14. Пример определения функции пользователя с помощью оператора `deff`

```
-->deff('z=fun1(x,y)','z=sqrt(abs(sin(x/y)))*exp(x^y)');
-->x=1.2;y=0.3;z=fun1(x,y)
 z  = 2.5015073
```

Рассмотрим пример создания и применения функции, вычисляющей площадь треугольника со сторонами a, b и c по формуле Герона

$$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}, \text{ где } p = \frac{a+b+c}{2}.$$

Листинг 2.15. Функция, вычисляющая площадь треугольника по формуле Герона

```
-->deff('S=G(a,b,c)','p=(a+b+c)/2;S=sqrt((p-a)*(p-b)*(p-c))');
-->G(2,3,3)
 ans  = 1.4142136
```

В следующем листинге приведен пример создания и применения функции, с помощью которой можно найти корни квадратного уравнения вида $ax^2 + bx + c = 0$ по формулам $D = b^2 - 4ac$; $x_{1,2} = \frac{-b \pm \sqrt{D}}{2}a$:

Листинг 2.16. Пример функции решения квадратного уравнения

```
-->deff('[x1,x2]=korni(a,b,c)','d=b^2-4*a*c;
                x1=(-b+sqrt(d))/2/a;x2=(-b-sqrt(d))/2/a');
-->[x1,x2]=korni(-2,-3,5)
 x2  = 1.
 x1  = 2.5
```

Второй способ создания функции это применение конструкции вида:

```
function[имя1,...,имяN]=имя_функции(переменная_1,...,переменная_M)
тело функции
endfunction
```

где `имя1,...,имяN` — список выходных параметров, то есть переменных, которым будет присвоен конечный результат вычислений; `имя_функции` — имя с которым эта функция будет вызываться, `переменная_1, ...,переменная_M` — входные параметры.

Все имена переменных внутри функции, а также имена из списка входных и выходных параметров воспринимаются системой как *локальные*, т.е. считаются определенными только внутри функции.

Вообще говоря, функции в Scilab играют роль подпрограмм. Поэтому целесообразно набирать их тексты в редакторе и сохранять в виде отдельных файлов. Причем имя файла должно обязательно совпадать с именем функции. Расширение файлам-функциям обычно присваивают *sci* или *sce*.

Обращение к функции осуществляется так же, как и к любой другой встроенной функции системы, т.е. из командной строки. Однако функции, хранящиеся в отдельных файлах, должны быть предварительно загружены в систему, например, при помощи оператора `exec(имя_файла)` или командой главного меню *File – Exec...*, что, в общем, одно и то же.

В качестве примера рассмотрим следующую задачу.

Задача 2.1.
Решить кубическое уравнение.

Кубическое уравнение

$$ax^3 + bx^2 + cx + d = 0 \tag{2.1}$$

после деления на a принимает канонический вид:

$$x^3 + rx^2 + sx + t = 0\,, \tag{2.2}$$

где

$$r = \frac{b}{a}\,, \qquad s = \frac{c}{a}\,, \qquad t = \frac{d}{a}\,.$$

В уравнении (2.2) сделаем замену

$$x = y - \frac{r}{3}$$

и получим следующее приведенное уравнение:

$$y^3 + py + q = 0\,, \tag{2.3}$$

где

$$p = \frac{(3s - r^2)}{3}\,, \qquad q = 2\frac{r^3}{27} - \frac{rs}{3} + t\,.$$

Число действительных корней приведенного уравнения (2.3) зависит от знака дискриминанта $D = \left(\frac{p}{3}\right)^3 + \left(\frac{q}{2}\right)^3$ (табл. 2.2).

Таблица 2.2. Количество корней кубического уравнения

Дискриминант	***Количество действительных корней***	***Количество комплексных корней***
$D \geqslant 0$	1	2
$D < 0$	3	–

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

$$y_1 = u+v, \qquad y_2 = \frac{-(u+v)}{2} + \frac{(u-v)}{2} i\sqrt{3}, \qquad y_3 = \frac{-(u+v)}{2} - \frac{(u-v)}{2} i\sqrt{3}\,. \tag{2.4}$$

Здесь

$$u = \sqrt[3]{\frac{-q}{2} + \sqrt{(D)}}, \qquad v = \sqrt[3]{\frac{-q}{2} - \sqrt{(D)}}\,.$$

Далее представлен список команд, реализующий описанный выше способ решения кубического уравнения:

Листинг 2.17. Решение кубического уравнения

```
function [x1,x2,x3]=cub(a,b,c,d)
r=b/a;
s=c/a;
t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=-(u+v)/2+(u-v)/2*%i*sqrt(3);
y3=-(u+v)/2-(u-v)/2*%i*sqrt(3);
x1=y1-r/3;
x2=y2-r/3;
x3=y3-r/3;
endfunction
//Вызов функции и вывод результатов ее работы:
-->exec('C:\Scilab\scilab-4.1.1\cub.sce');
-->disp('exec done');
Warning :redefining function: cub
 exec done
-->[x1,x2,x3]=cub(3,-20,-3,4)
 x3  =
    0.3880206
 x2  =
  - 0.5064407
 x1  =
    6.7850868
```

Глава 3

Массивы и матрицы в Scilab. Решение задач линейной алгебры

Для работы с множеством данных удобно использовать массивы. Например, можно создать массив для хранения числовых или символьных данных. В этом случае вместо создания переменной для хранения каждого данного достаточно создать один массив, где каждому элементу будет присвоен порядковый номер.

Таким образом, *массив* — множественный тип данных, состоящий из фиксированного числа элементов. Как и любой другой переменной, массиву должно быть присвоено имя.

Переменную, представляющую собой просто список данных, называют *одномерным массивом*, или *вектором*. Для доступа к данным, хранящимся в определенном элементе массива, необходимо указать *имя массива* и *порядковый номер* этого элемента, называемый *индексом*.

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать *двумерные массивы* (матрицы). Для доступа к данным, хранящимся в таком массиве, необходимо указать имя массива и два индекса: первый должен соответствовать номеру строки, а второй — номеру столбца, в которых хранится необходимый элемент.

Значение *нижней границы индексации* в Scilab равно единице. Индексы могут быть только целыми положительными числами.

3.1 Ввод и формирование массивов и матриц

Задать одномерный массив в Scilab можно следующим образом:

```
name=Xn:dX:Xk
```

где `name` — имя переменной, в которую будет записан сформированный массив, `Xn` — значение первого элемента массива, `Xk` — значение последнего элемента массива, `dX` — шаг, с помощью которого формируется каждый следующий элемент массива, т.е. значение второго элемента составит `Xn+dX`, третьего `Xn+ dX+dX` и так далее до `Xk`.

Если параметр `dX` в конструкции отсутствует, это означает, что по умолчанию он принимает значение, равное единице, т.е. каждый следующий элемент массива равен значению предыдущего плюс один:

```
name=Xn:Xk
```

Переменную, заданную как массив, можно использовать в арифметических выражениях и в качестве аргумента математических функций. Результатом работы таких операторов являются массивы:

Листинг 3.1. Примеры работы с массивами

```
--> Xn=-3.5;dX=1.5;Xk=4.5;
--> X=Xn:dX:Xk
X =
-3.5000 -2.0000 -0.5000 1.0000 2.5000 4.0000
--> Y=sin(X/2)
Y =
-0.9840   -0.8415   -0.2474   0.4794   0.9490   0.9093
--> A=0:5
A =
0 1 2 3 4 5
--> 0:5
ans =
0  1  2 3 4 5
--> ans/2+%pi
ans =
3.1416   3.6416   4.1416   4.6416   5.1416   5.6416
```

Еще один способ задания векторов и матриц в Scilab — это их *поэлементный ввод*.

Так, для *определения вектора-строки* следует ввести имя массива, а затем после знака присваивания, в квадратных скобках через пробел или запятую, перечислить элементы массива:

```
name=[x1 x2 ... xn]     или     name=[x1, x2, ..., xn]
```

Пример ввода вектора-строки:

Листинг 3.2. Определение вектора-строки

```
--> V=[1 2 3 4 5]
V =
1 2 3 4 5
--> W=[1.1,2.3,-0.1,5.88]
W =
1.1000 2.3000 -0.1000 5.8800
```

Элементы *вектора-столбца вводятся* через точку с запятой:

```
name=[x1; x2; ...; xn]
```

Пример ввода вектора-столбца:

Листинг 3.3. Определение вектора-столбца

```
--> X=[1;2;3]
X =
1
2
3
```

Обратиться к элементу вектора можно, указав имя массива и порядковый номер элемента в круглых скобках:

```
name(индекс)
```

Например:

Листинг 3.4. Пример обращения к элементу массива

```
--> W=[1.1,2.3,-0.1,5.88];
--> W(1)+2*W(3)
ans = 0.9000
```

Ввод элементов матрицы также осуществляется в квадратных скобках, при этом элементы строки отделяются друг от друга пробелом или запятой, а строки разделяются между собой точкой с запятой:

```
name=[x11, x12, ..., x1n; x21, x22, ..., x2n; ...;
                  xm1, xm2, ..., xmn;]
```

Обратиться к элементу матрицы можно, указав после имени матрицы, в круглых скобках через запятую, номер строки и номер столбца на пересечении которых элемент расположен:

```
name(индекс1, индекс2)
```

Далее приведен пример задания матрицы и обращение к ее элементам:

Листинг 3.5. Пример обращения к элементам матрицы

```
--> A=[1 2 3;4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
--> A(1,2)^A(2,2)/A(3,3)
ans =  3.5556
```

Кроме того, матрицы и векторы можно формировать, составляя их из ранее заданных матриц и векторов:

Листинг 3.6. Пример конкатенации матриц

```
--> v1=[1 2 3]; v2=[4 5 6]; v3=[7 8 9];
--> //Горизонтальная конкатенация векторов-строк:
--> V=[v1 v2 v3]
V = 1 2 3 4 5 6 7 8 9
-->//Вертикальная конкатенация векторов-строк,
-->//результат матрица:
--> V=[v1; v2; v3]
V =
1 2 3
4 5 6
7 8 9
-->//Горизонтальная конкатенация матриц:
--> M=[V V V]
M =
1 2 3 1 2 3 1 2 3
4 5 6 4 5 6 4 5 6
7 8 9 7 8 9 7 8 9
-->//Вертикальная конкатенация матриц:
--> M=[V;V]
M =
1 2 3
4 5 6
7 8 9
1 2 3
4 5 6
7 8 9
```

Важную роль при работе с матрицами играет знак двоеточия «:». Указывая его вместо индекса при обращении к массиву, можно получать доступ к группам его элементов. Например:

Листинг 3.7. Примеры использования операции «:»

```
-->//Пусть задана матрица A
--> A=[5 7 6 5; 7 10 8 7;6 8 10 9;5 7 9 10]
--> //Выделить из матрицы A второй столбец
--> A(:,2)
ans =
7
10
8
7
--> //Выделить из матрицы A третью строку
--> A(3,:)
ans = 6 8 10 9
--> //Выделить из матрицы A подматрицу M
--> M=A(3:4,2:3)
M =
8 10
7 9
--> //Удалить из матрицы A второй столбец
--> A(:,2)=[]
A =
5 8 10
7 7 9
6 10 9
5 9 10
--> //Удалить из матрицы A третью строку
--> A(3,:)=[]
A =
5 8 10
7 7 9
5 9 10
--> //Представить матрицу M в виде вектора-столбца
--> v=M(:)
v =
8
7
10
9
--> //Выделить из вектора v элементы со второго по четвертый
```

```
--> b=v(2:4)
b =
7
10
9
--> //Удалить из массива b второй элемент
--> b(2)=[];
```

3.2 Действия над матрицами

Для работы с матрицами и векторами в Scilab предусмотрены следующие операции:

`+` — сложение;

`-` — вычитание[1];

`'` — транспонирование[2];

`*` — матричное умножение[3];

`*` — умножение на число;

`^` — возведение в степень[4];

`\` — левое деление[5];

`/` — правое деление[6];

`.*` — поэлементное умножение матриц;

`.^` — поэлементное возведение в степень;

`.\` — поэлементное левое деление;

`./` — поэлементное правое деление.

[1]Операции сложения и вычитания определены для матриц одной размерности или векторов одного типа, т.е. суммировать (вычитать) можно либо векторы-столбцы, либо векторы-строки одинаковой длины.

[2]Если в некоторой матрице заменить строки соответствующими столбцами, то получится транспонированная матрица.

[3]Операция умножения вектора на вектор определена только для векторов одинакового размера, причем один из них должен быть вектором-столбцом, а второй вектором-строкой. Матричное умножение выполняется по правилу «строка на столбец» и допустимо, если количество строк во второй матрице совпадает с количеством столбцов в первой. Кроме того, переместительный закон на произведение матриц не распространяется.

[4]Возвести матрицу в n-ю степень значит умножить ее саму на себя n раз. При этом целочисленный показатель степени может быть как положительным, так и отрицательным. В первом случае выполняется алгоритм умножения матрицы на себя указанное число раз, во втором умножается на себя матрица, *обратная к данной.*

[5]$(\mathbf{A}\backslash\mathbf{B}) \Rightarrow (\mathbf{A}^{-1}\mathbf{B})$, операция может быть применима для решения матричного уравнения вида $\mathbf{A}\cdot\mathbf{X}=\mathbf{B}$, где $\mathbf{X}$ — неизвестный вектор.

[6]$(\mathbf{B}/\mathbf{A}) \Rightarrow (\mathbf{B}\cdot\mathbf{A}^{-1})$, используют для решения матричных уравнений вида $\mathbf{X}\cdot\mathbf{A}=\mathbf{B}$.

Пример действий над матрицами:

Листинг 3.8. Примеры матричных операций

```
-->A=[1 2 0;-1 3 1;4 -2 5];
-->B=[-1 0 1;2 1 1;3 -1 -1];
-->//Вычислить (AT+B)2 - 2A(0.5BT-A)
-->(A'+B)^2-2*A*(1/2*B'-A)
 ans  =
    10.    8.     24.
    11.    20.    35.
    63.  - 30.    68.
--> //Решить матричные уравнения A•X=B и X•A=B.
-->A=[3 2;4 3];
-->B=[-1 7;3 5];
-->//Решение матричного уравнения AX=B:
-->X=A\B
 X  =
  - 9.     11.
    13.  - 13.
-->//Решение матричного уравнения XA=B:
-->X=B/A
 X  =
  - 31.    23.
  - 11.    9.
-->//Проверка
-->X*A-B
 ans  =
    0.    0.
    0.    0.
```

Кроме того, если к некоторому заданному вектору или матрице применить математическую функцию, то результатом будет новый вектор или матрица той же размерности, но элементы будут преобразованы в соответствии с заданной функцией:

Листинг 3.9. Пример применения функции к массиву

```
--> x=[0.1 -2.2 3.14 0 -1];
--> sin(x)
ans =
0.0998 -0.8085 0.0016 0 -0.8415
```

3.3 Специальные матричные функции

Для работы с матрицами и векторами в Scilab существуют специальные функции. Рассмотрим наиболее часто используемые из них.

Функции определения матриц:

- `matrix(A [,n,m])` — преобразует матрицу `A` в матрицу другого размера;

Листинг 3.10. Использование функции `matrix`

```
-->D=[1 2;3 4;5 6];
-->matrix(D,2,3)
ans  =
1. 5. 4.
3. 2. 6.
-->matrix(D,3,2)
ans  =
1. 2.
3. 4.
5. 6.
-->matrix(D,1,6)
ans  =
1. 3. 5. 2. 4. 6.
-->matrix(D,6,1)
ans  =
1.
3.
5.
2.
4.
6.
```

- `ones(m,n)` — создает матрицу единиц из `m` строк и `n` столбцов[1];

Листинг 3.11. Использование функции `ones`

```
-->ones(1,3) //Формируется вектор-строка
ans  =
1. 1. 1.
-->ones(2,2) //Формируется квадратная матрица
 ans  =
1. 1.
1. 1.
-->m=3; n=2;
```

[1] Результатом работы функции `ones(n1,n2...,nn)` будет многомерная матрица единиц.

```
-->X=ones(m,n) //Формируется матрица размерности m на n
 X  =
1. 1.
1. 1.
1. 1.
-->M=[1 2 3;4 5 6]
M  =
1. 2. 3.
4. 5. 6.
-->//Формируется матрица Y, состоящая из единиц,
-->//той же размерности, что и матрица M
-->Y=ones(M)
 Y  =
1. 1. 1.
1. 1. 1.
```

- `zeros(m,n)` — создает *нулевую матрицу*[1] из `m` строк и `n` столбцов[2];

Листинг 3.12. Использование функции `zeros`

```
-->zeros(3,2)
 ans  =
0. 0.
0. 0.
0. 0.
-->M=[1 2 3 4 5];
-->Z=zeros(M)
 Z  =
0. 0. 0. 0. 0.
```

- `eye(m,n)` — формирует *единичную матрицу*[3] из `m` строк и `n` столбцов;

Листинг 3.13. Примеры использования функции `eye`

```
-->eye(3,3)
 ans  =
1. 0. 0.
0. 1. 0.
0. 0. 1.
-->eye(5,1)
 ans  =
```

[1] В нулевой матрице все элементы равны нулю.

[2] Результатом работы функции `zeros(n1,n2...,nn)` будет многомерная матрица нулей.

[3] В единичной матрице элементы главной диагонали равны единице, а все остальные — нулю.

```
1.
0.
0.
0.
0.
-->m=3; n=4;
-->E=eye(m,n)
 E  =
1. 0. 0. 0.
0. 1. 0. 0.
0. 0. 1. 0.
-->M=[0 1;2 3];
-->//Формируется единичная матрица E
-->//той же размерности, что и матрица M
-->E=eye(M)
 E  =
1. 0.
0. 1.
-->//Функцию можно использовать без параметров eye().
-->//В этом случае задается матрица с неопределенными
-->//размерами, которые будут определены после суммирования
-->//с другой, определенной ранее, матрицей.
-->M=[1 2;3 4;5 6]; E=eye();
-->A=E+M
 A  =
2. 2.
3. 5.
5. 6.
-->M-E
 ans  =
0. 2.
3. 3.
5. 6.
```

- `rand(n1,n2,...nn[,fl])` — формирует многомерную матрицу *случайных чисел*. Необязательный параметр `p` — это символьная переменная, с помощью которой можно задать тип распределения случайной величины (`'uniform'` — равномерное, `'normal'` — гауссовское); `rand(m,n)` — формирует матрицу `m` на `n` случайных чисел; `rand(M)` — формирует матрицу случайных чисел, размер которой совпадает с размером матрицы `M`; результат функции `rand()` — случайное скалярное число;

Листинг 3.14. Примеры использования функции `rand`

```
-->rand(2,2)//Матрица 2 на 2 случайных чисел
 ans  =
0.2113249 0.0002211
0.7560439 0.3303271
--> R=rand(2,2,2)//Многомерный массив случайных чисел
R(:,:,1) =
0.9355 0.4103
0.9169 0.8936
R(:,:,2) =
0.0579 0.8132
0.3529 0.0099
-->rand()//Случайное число
 ans  =
0.6653811
```

- `sparse([i1 j1;i2 j2;...;in jn],[n1,n2,...,nn])` — формирует разреженную матрицу[1]. Для создания матрицы такого типа необходимо указать индексы ее ненулевых элементов — `[i1 j1,i2 j2,...,in jn]`, и их значения — `[n1,n2,...,nn]`. Индексы одного элемента отделяются друг от друга либо пробелом, либо запятой, а пары индексов — соответственно точкой с запятой, значения элементов разделяются запятыми. При попытке просмотреть матрицу подобного типа пользователю будет предоставлено сообщение о ее размерности, а также значения ненулевых элементов и их местоположение в матрице;

- `full(M)` — вывод разреженной матрицы `M` в виде таблицы;

Листинг 3.15. Использование функций `sparse` и `full`

```
-->A=sparse([1 3;3 2;3 5],[4,5,6])
 A  =
(    3,    5) sparse matrix
(    1,    3)        4.
(    3,    2)        5.
(    3,    5)        6.
-->full(A)
 ans  =
0. 0. 4. 0. 0.
0. 0. 0. 0. 0.
0. 5. 0. 0. 6.
```

[1] Разреженная матрица — матрица, большинство элементов которой нули.

- `hypermat(D[,V])` — создание многомерной матрицы с размерностью, заданной вектором `D` и значениями элементов, хранящихся в векторе `V` (использование параметра `V` необязательно);

Листинг 3.16. Использование функции `hypermat`

```
-->//Пример создания матрицы M,
-->//состоящей из трех матриц размерности два на два,
-->//каждый элемент матрицы - член последовательности
-->//целых чисел от 0 до 11.
-->M=hypermat([2 2 3],0:11)
 M  =
(:,:,1)
0. 2.
1. 3.
(:,:,2)
4. 6.
5. 7.
(:,:,3)
8. 10.
9. 11.
```

- `diag(V[,k])` — возвращает квадратную матрицу с элементами `V` на главной или на `k`-й диагонали[1]; функция `diag(A[,k])`, где `A` — ранее определенная матрица, в качестве результата выдаст вектор-столбец, содержащий элементы главной или `k`-ой диагонали матрицы `A`;

Листинг 3.17. Использование функции `diag`

```
--> V=[1,2,3];
--> diag(V)//Диагональная матрица, V на главной диагонали
ans = 1  0 0
      0  2 0
      0  0 3
-->//Диагональная матрица,
-->//V на первой диагонали (выше главной)
--> diag(V,1)
ans = 0  1 0 0
      0  0 2 0
      0  0 0 3
      0  0 0 0
-->//Диагональная матрица,
-->//V на первой диагонали (ниже главной)
```

[1] В диагональной матрице все элементы нули, кроме элементов главной диагонали.

```
--> diag(V,-1)
ans = 0  0 0 0
      1  0 0 0
      0  2 0 0
      0  0 3 0
--> A=[-1 2 0 ;2 1 -1 ;2 1 3]
A =
-1 2  0
 2 1 -1
 2 1  3
--> diag(A) //Главная диагональ матрицы A
ans =
    -1
     1
     3
```

- `cat(n, A, B, [C, ...])` — объединяет матрицы `A` и `B` или все входящие матрицы, при `n=1` по строкам, при `n=2` по столбцам; то же что `[A; B]` или `[A, B]`;

Листинг 3.18. Использование функции `cat`

```
--> A=[1 2;3 4]; B=[5 6 ;7 8];
--> cat(2,A,B)//Объединение матриц
ans =
1 2 5 6
3 4 7 8
--> cat(1,A,B) //Объединение матриц
ans =
1 2
3 4
5 6
7 8
```

- `tril(A[,k])` — формирует из матрицы `A` нижнюю треугольную матрицу[1], начиная с главной или с `k`-й диагонали;

Листинг 3.19. Использование функции `tril`

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

[1] Матрица называется нижней треугольной, если все элементы, расположенные выше главной диагонали, равны нулю.

```
7 8 9
-->//Нижняя треугольная матрица, начиная с главной диагонали
--> tril(A)
ans =
1 0 0
4 5 0
7 8 9
--> tril(A,0)//Тоже что и tril(A)
ans =
1 0 0
4 5 0
7 8 9
--> tril(A,1)//Нижняя треугольная матрица,
--> //начиная с первой диагонали (выше главной)
ans =
1 2 0
4 5 6
7 8 9
--> tril(A,-2) )//Нижняя треугольная матрица,
--> //начиная со второй диагонали (ниже главной)
ans =
0 0 0
0 0 0
7 0 0
```

- `triu(A[,k])` — формирует из матрицы `A` верхнюю треугольную матрицу[1], начиная с главной или с `k`-й диагонали;

Листинг 3.20. Использование функции `triu`

```
--> A=[1 2 3;4 5 6;7 8 9];
--> triu(A)//Верхняя треугольная матрица
ans =
1 2 3
0 5 6
0 0 9
--> triu(A,2) )//Верхняя треугольная матрица,
--> //начиная со второй диагонали (выше главной)
ans =
0 0 3
0 0 0
0 0 0
```

[1]Матрица называется верхней треугольной, если все элементы, расположенные ниже главной диагонали, равны нулю.

```
--> triu(A,-1) )//Верхняя треугольная матрица,
--> //начиная с первой диагонали (ниже главной)
ans =
1 2 3
4 5 6
0 8 9
```

- `sort(X)` — выполняет упорядочивание массива `X`; если `X` — матрица, сортировка выполняется по столбцам;

Листинг 3.21. Использование функции `sort`

```
-->b=[2 0 1]; sort(b) //Сортировка по убыванию
 ans  =
    2.    1.    0.
-->-sort(-b) //Сортировка по возрастанию
 ans  =
    0.    1.    2.
-->A=[1 2 0;-1 3 1;4 -2 5];
-->sort(A) //Сортировка матрицы
 ans  =
    5.    2.    0.
    4.    1.  - 1.
    3.    1.  - 2.
```

Функции вычисления различных числовых характеристик матриц:

- `size(V[,fl])` — определяет размер массива `V`; если `V` — двумерный массив, то `size(V,1)` или `size(V,'r')` определяют число строк матрицы `V`, а `size(V,2)` или `size(V,'c')` — число столбцов;

Листинг 3.22. Использование функции `size`

```
-->M=[1 2;3 4;5 6;7 8];
-->[n,m]=size(M)
 m  =
2.
 n  =
4.
-->size(M,1)
 ans  =
4.
-->size(M,2)
 ans  =
2.
```

- `length(X)` — определяет количество элементов массива `X`; если `X` — вектор, его длину; если `X` — матрица, вычисляет общее число ее элементов;

Листинг 3.23. Использование функции `length`

```
--> V=[-1 0 3 -2 1 -1 1];//Вектор-строка
--> length(V)//Длина вектора
ans =
    7
-->[1 2 3;4 5 6];//Матрица
-->length(ans)//Количество элементов матрицы
 ans  =
    6.
```

- `sum(X[,fl])` — вычисляет *сумму* элементов массива `X`, имеет необязательный параметр `fl`. Если параметр `fl` отсутствует, то функция `sum(X)` возвращает скалярное значение, равное сумме элементов массива. Если `fl='r'` или `fl=1`, что то же самое, то функция вернет строку, равную поэлементной сумме столбцов матрицы `X`. Если `fl='c'` или `fl=2`, то результатом работы функции будет вектор-столбец, каждый элемент которого равен сумме элементов строк матрицы `X`. Частный случай применения функции `sum` — это вычисление *скалярного произведения векторов*[1];

Листинг 3.24. Примеры использования функции `sum`

```
-->M=[1 2 3;4 5 6;7 8 9];
-->Y=sum(M) //Сумма элементов матрицы
Y  = 45.
-->S1=sum(M,1) //Сумма элементов матрицы по столбцам
S1  =
12 15 18
-->S2=sum(M,2) // Сумма элементов матрицы по строкам
S2  =
6
15
24
--> V=[-1 0 3 -2 1 -1 1];
--> sum(V) //Сумма элементов вектора
ans = 1
-->//Частный случай. Вычисление скалярного произведения
--> a=[1 2 3];b=[2 0 1];
--> sum(a.*b)
ans = 5
```

[1] Скалярное произведение вычисляется по формуле $\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$.

- `prod(X[,fl])` — вычисляет *произведение* элементов массива `X`, работает аналогично функции `sum`;

Листинг 3.25. Использование функции `prod`

```
-->prod(M)
ans  = 362880.
-->p1=prod(M,1)
p1  =
28 80 162
-->p2=prod(M,2)
p2  =
6
120
504
--> V=[1,2,3];
--> prod(V) //Произведение элементов вектора
ans = 6
```

- `max(M[,fl])` — вычисляет *наибольший элемент* в массиве `M`, имеет необязательный параметр `fl`. Если параметр `fl` отсутствует, то функция `max(M)` возвращает максимальный элемент массива `M`; если `fl='r'`, то функция вернет строку максимальных элементов столбцов матрицы `M`; если `fl='c'`, то результатом работы функции будет вектор-столбец, каждый элемент которого равен максимальному элементу соответствующих строк матрицы `M`. Функция `[x, nom]=max(M[,fl])` вернет значение максимального элемента `x` и его номер в массиве `nom`;

Листинг 3.26. Использование функции `max`

```
-->M=[5 0 3;2 7 1;0 4 9];
-->max(M)
 ans  =
9.
-->max(M,'r')
 ans  =
    5.    7.    9.
-->max(M,'c')
 ans  =
    5.
    7.
    9.
-->[x,nom]=max(M)
 nom  =
```

```
    3.    3.
 x  =
9.
```

- `min(M[,fl])` — вычисляет *наименьший элемент* в массиве `M`, работает аналогично функции `max`;

Листинг 3.27. Использование функции `min`

```
-->A=[5 10 3 2 7 1 25 4 0];
-->[x,nom]=min(A)
 nom  =
    7.
 x  =
    25.
```

- `mean(M[,fl])` — вычисляет *среднее значение* массива `M`; если `M` двумерный массив, то `mean(M,1)` или `mean(M,'r')` определяют среднее значение строк матрицы `M`, а `mean(M,2)` или `mean(M,'c')` — среднее значение столбцов;

Листинг 3.28. Использование функции `mean`

```
-->mean(M)
 ans  =
    3.4444444
-->mean(M,1)
 ans  =
    2.3333333    3.6666667    4.3333333
-->mean(M,2)
 ans  =
    2.6666667
    3.3333333
    4.3333333
```

- `median(M[,fl])` — вычисляет *медиану*[1] массива `M`, работает аналогично функции `mean`;

Листинг 3.29. Использование функции `median`

```
-->M=[5 0 3;2 7 1;0 4 9];
-->median(M)
 ans  = 3.
-->median(M,1)
 ans  =
```

[1]Значение, которое делит массив на две части.

```
    2.    4.    3.
-->median(M,2)
 ans  =
    3.
    2.
    4.
```

- `det(M)` — вычисляет *определитель* квадратной матрицы `M`;

Листинг 3.30. Использование функции `det`

```
-->M=[1 0 2;3 2 1;0 3 1];
-->det(M)
 ans  = 17.
-->Z=[1 2 2;0 1 3;2 4 4];
-->det(Z)
 ans  = 0.
```

- `rank(M[,tol])` — вычисление *ранга матрицы* `M`[1] с точностью `tol`.

Листинг 3.31. Использование функции `rank`

```
-->M=[1 0 2;3 2 1;0 3 1];
-->rank(M)
 ans  = 3.
-->Z=[1 2 2;0 1 3;2 4 4];
-->rank(Z)
 ans  = 2.
```

- `norm(M[,fl])` — вычисление *нормы* квадратной матрицы `M`; тип нормы определяется необязательной строковой переменной `fl`, по умолчанию `fl=2`. Функции `norm(M)` и `norm(M,2)` эквивалентны и вычисляют вторую норму матрицы `M`[2]. Первая норма[3] определяется функцией `norm(M,1)`. Функции `norm(M,'inf')` и `norm(M,'fro')` вычисляют соответственно бесконечную[4] и евклидову нормы[5]. Если `V` — вектор, то результатом работы функции `norm(V,1)` будет сумма модулей всех элементов вектора `V`. С помощью функции `norm(V,2)` можно вычислить модуль вектора `V`[6]. Значение `norm(V,'inf')` равно модулю максимального элемента вектора по модулю;

[1] *Ранг матрицы* — максимальное число линейно независимых строк.
[2] *Вторая норма матрицы* — ее наибольшее сингулярное значение.
[3] *Первая норма матрицы* — наибольшая сумма по столбцам.
[4] Бесконечная норма — наибольшая сумма по строкам.
[5] Евклидова норма — корень из суммы квадратов всех элементов матрицы.
[6] Модуль вектора — корень квадратный из суммы квадратов его элементов.

Листинг 3.32. Использование функции `norm`

```
-->M=[1 0 2;3 2 1;0 3 1];
-->norm(M,1)
 ans  = 5.
-->norm(M,2)
 ans  =
    4.5806705
-->norm(M,'inf')
 ans  = 6.
-->norm(M,'fro')
 ans  = 5.3851648
-->X=[5 -3 4 -1 2];
-->norm(X,1)
 ans  = 15.
-->sum(abs(X))//То же, что и norm(X,1)
 ans  = 15.
-->norm(X,2)
 ans  = 7.4161985
-->sqrt(sum(X^2)) //То же, что и norm(X,2)
 ans  = 7.4161985
-->norm(X,'inf')
 ans  =
    5.
-->max(abs(X))//То же, что и norm(X,'inf')
 ans  =
    5.
```

- `cond(M)` — вычисляет *число обусловленности*[1] матрицы `M` по второй норме;

Листинг 3.33. Использование функции `cond`

```
-->A=[5 7 6 5;7 10 8 7;6 8 10 9;5 7 9 10];
-->cond(A)
 ans  =
    2984.0927
```

[1] Число обусловленности равно произведению нормы исходной матрицы на норму обратной.

Функции, реализующие численные алгоритмы решения задач линейной алгебры:

- `spec(M)` — вычисляет *собственные значения* и *собственные векторы*[1] квадратной матрицы `M`.

Листинг 3.34. Использование функции `spec`

```
-->M=[3 -2;-4 1]
 M  =
3. - 2.
- 4. 1.
-->spec(M) //Собственные числа матрицы
 ans  =
- 1.
  5.
//X - собственные векторы,
-->соответствующие собственным значениям из матрицы Y.
-->[X,Y]=spec(M)
 Y  =
! - 1.    0  !
!   0     5. !
 X  =
!   0.4472136  - 0.7071068 !
!   0.8944272    0.7071068 !
```

- `inv(A)` — вычисляет *матрицу, обратную*[2] к `A`[3];

Листинг 3.35. Использование функции `inv`

```
-->//Пример вычисления обратной матрицы.
-->A=[1 2 3 5;0 1 3 2;4 2 1 1;2 3 0 1];
-->inv(A)
 ans  =
!   0.0285714  - 0.1428571    0.3428571  - 0.2 !
! - 0.1428571    0.2142857  - 0.2142857    0.5 !
! - 0.2          0.5          0.1        - 0.1 !
!   0.3714286  - 0.3571429  - 0.0428571  - 0.1 !
```

[1] Любой ненулевой вектор $\mathbf{x}$, принадлежащий некоторому векторному пространству, для которого $\mathbf{Ax} = L\mathbf{x}$, где L — некоторое число, называется собственным вектором матрицы $\mathbf{A}$; L — соответствующим ему собственным значением матрицы A.

[2] Обратной матрицей по отношению к данной называется матрица того же типа, которая будучи умноженной как слева, так и справа на данную матрицу, в результате даст единичную матрицу. Т.е. при умножении `A` на `inv(A)` слева должна получиться единичная матрица.

[3] Вычисление основано на методе LU-разложения.

```
-->//При умножении обратной матрицы на исходную,
-->//получилась матрица, близкая к единичной.
-->inv(A)*A
 ans  =
1. - 1.110D-16 0. 0.
0.   1.    - 5.551D-17 5.551D-17
0.   0. 1. 1.388D-17
0.   0. 6.939D-17 1.
-->//При попытке обратить вырожденную матрицу
-->//(определитель равен или близок к нулю)
-->//пользователь получит сообщение об ошибке.
-->B=[1 2 3;1 4 5;1 6 7];
-->inv(B)
!--error    19
Problem is singular
```

- `pinv(A[,tol])` — вычисляет *псевдообратную матрицу*[1] для матрицы `A` с точностью `tol` (необязательный параметр);

Листинг 3.36. Использование функции `pinv`

```
-->pinv(A)
 ans  =
  0.0285714 - 0.1428571   0.3428571 - 0.2
- 0.1428571   0.2142857 - 0.2142857   0.5
- 0.2   0.5   0.1 - 0.1
  0.3714286 - 0.3571429 - 0.0428571 - 0.1
```

- `linsolve(A,b)` — решает систему линейных алгебраических уравнений вида $\mathbf{A} \cdot \vec{x} - \vec{b} = 0$.

Листинг 3.37. Пример использования функции `linsolve`

```
-->//Решение системы линейных уравнений
-->//{x1+2x2-7=0; x1+x2-6=0}.
-->//Свободные коэффициенты вводятся как вектор-столбец
-->//и с учетом знаков.
-->A=[1 2;1 1];b=[-7;-6];
-->x=linsolve(A,b)
 x  =
5.
1.
```

[1] Если для матрицы `pinv(A[,tol])=X` выполняется условие `A*X*A=A`, `X*A*X=X`, а матрицы `X*A` и `A*X` эрмитовы, то вычисляемая матрица `X` *псевдообратная* к `A`. Вычисление базируется на методе сингулярного разложения, и все сингулярные значения матрицы, меньшие `tol`, приравниваются нулю.

```
-->//Результатом операции A*x+b является вектор, достаточно
-->//близкий к нулю, это значит, что система решена верно.
-->A*x+b
 ans  =
   1.0D-14 *
- 0.6217249
  0.0888178
-->//Решение системы {x1+x2-1=0; x1+x2-3=0}
-->A=[1 1;1 1]; b=[-1;-3];
-->//Система не имеет решений:
-->linsolve(A,b)
WARNING:Conflicting linear constraints!
 ans  =
     []
-->//Решение системы {3x1-x2-1=0; 6x1-2x2-2=0}.
-->//В случае, когда система имеет бесконечное
-->//множество решений, SCILAB выдаст одно из них.
-->A=[3 -1;6 -2];
-->b=[-1;-2];
-->x=linsolve(A,b)
 x  =
0.3
   - 0.1
-->//Проверка неверна
-->A*x+b
 ans  =
   1.0D-15 *
- 0.1110223
- 0.2220446
```

- `rref(A)` — осуществляет приведение матрицы `A` к треугольной форме, используя метод исключения Гаусса;

Листинг 3.38. Пример использования функции `rref`

```
--> A=[3 -2 1 5;6 -4 2 7;9 -6 3 12]
A =
3 -2 1 5
6 -4 2 7
9 -6 3 12
--> rref(A)
ans =
1.0000 -0.6667 0.3333 0
0   0 0 1.0000
0   0 0 0
```

- `lu(M)` — выполняет треугольное разложение матрицы `M`[1];

Листинг 3.39. Использование функции `lu`

```
-->A=[2 -1 5;3 2 -5;1 1 -2]
 A  =
2. - 1. 5.
3.   2.    - 5.
1.   1.    - 2.
-->[L,U]=lu(A)
 U  =
3.   2. - 5.
0. - 2.3333333   8.3333333 !
0.   0.   0.8571429 !
 L  =
0.6666667 1. 0.
1. 0. 0.
0.3333333    - 0.1428571 1.
-->LU=L*U
 LU  =
2. - 1. 5.
3.   2.    - 5.
1. 1. - 2.
```

- `qr(M)` — выполняет разложение матрицы `M`[2] на ортогональную и верхнюю треугольную матрицы;

Листинг 3.40. Использование функции `qr`

```
-->[Q,R]=qr(A)
 R  =
- 3.7416574  - 1.3363062    1.8708287
  0.         - 2.0528726    7.0632734
  0.           0.           0.7811335
 Q  =
- 0.5345225    0.8350668    0.1301889
- 0.8017837  - 0.4523279  - 0.3905667
0.2672612  - 0.3131501    0.9113224

-->Q*R
 ans  =
```

[1] $\mathbf{M} = C \cdot \mathbf{L} \cdot \mathbf{U}$, где $\mathbf{L}$ и $\mathbf{U}$ — соответственно нижняя и верхняя треугольные матрицы, все четыре матрицы квадратные и одного порядка. Такие вычисления называют LU-разложением.

[2] $\mathbf{M} = \mathbf{Q} \cdot \mathbf{R}$, где $\mathbf{Q}$ — ортогональная матрица, а $\mathbf{R}$ — верхняя треугольная матрица. Этот процесс называют QR-разложением.

```
2.  - 1.    5.
3.    2.  - 5.
1.    1.  - 2.
```

- `svd(M)` — выполняет *сингулярное разложение*[1] размером $n \times m$; результатом работы функции может быть либо сингулярное разложение, либо вектор, содержащий сингулярные значения матрицы;

Листинг 3.41. Использование функции `svd`

```
-->[U,S,V]=svd(A)
 V  =
- 0.1725618    0.9641403  - 0.2016333
- 0.3059444    0.1421160    0.9413825
  0.9362801    0.2241352    0.2704496
 S  =
7.8003308    0.           0.
0.           3.6207331    0.
0.           0.           0.2124427
 U  =
  0.5951314    0.8028320    0.0357682
- 0.7449652    0.5678344  - 0.3501300
- 0.3014060    0.1817273    0.9360180
-->U*S*V'
 ans  =
2.  - 1.    5.
3.    2.  - 5.
1.    1.  - 2.
-->s=svd(A)
 s  =
7.8003308
3.6207331
0.2124427
```

- `kernel(M[,tol[,fl]])` — определение *ядра матрицы*[2] `M`, параметры `tol` и `fl` являются необязательными. Первый задает точность вычислений, второй — используемый при вычислении алгоритм и принимает значения `'qr'` или `'svd'`.

[1] $\mathbf{M} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V^T}$, где $\mathbf{U}$ и $\mathbf{V}$ — ортогональные матрицы размером $m \times m$ и $n \times n$ соответственно, а $\mathbf{S}$ — диагональная матрица, на диагонали которой расположены сингулярные числа матрицы $\mathbf{M}$

[2] Ядро матрицы — это множество векторов $\mathbf{X}$. Поиск ядра матрицы сводится к решению однородной системы линейных уравнений $\mathbf{AX} = 0$. Если при вызове функции `X=kernel(A)` матрица $\mathbf{X}$ окажется непустой, то действительно $\mathbf{AX} = 0$.

Листинг 3.42. Использование функции `kernel`

```
-->A=[4 1 -3 -1;2 3 1 -5;1 -2 -2 3]
 A  =
4.    1.  - 3.  - 1.
2.    3.    1.  - 5.
1.  - 2.  - 2.    3.
-->X=kernel(A)
 X  =
0.3464102
0.5773503
0.4618802
0.5773503
```

3.4 Символьные матрицы и операции над ними

В Scilab можно задавать символьные матрицы, то есть матрицы, элементы которых имеют строковый тип. При этом необходимо помнить, что строковые элементы должны быть заключены в двойные или одинарные кавычки.

Листинг 3.43. Формирование символьных матриц

```
-->M=['a' 'b';'c' 'd']
 M  =
a  b
c  d
-->P=['1' '2';'3' '4']
 P  =
1  2
3  4
```

Символьные матрицы можно складывать (результат сложения — конкатенация соответствующих строк) и транспонировать:

Листинг 3.44. Операции над символьными матрицами

```
-->M+P
 ans  =
a1  b2
c3  d4
-->M'
 ans  =
a  c
b  d
```

Кроме того, операции сложения и умножения можно проводить над отдельными элементами символьных матриц при помощи функций `addf(a,b)` и `mulf(a,b)`:

Листинг 3.45. Использование функций `addf` и `mulf`

```
-->addf(M(1,1),P(2,2))
 ans  =
 a+4
-->mulf(M(1,2),P(2,1))
 ans  =
 b*3
```

3.5 Решение систем линейных алгебраических уравнений

Система m уравнений с n неизвестными вида:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1\,, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2\,, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

называется *системой линейных алгебраических уравнений* (СЛАУ), причем x_j — неизвестные, a_{ij} — коэффициенты при неизвестных, b_i — свободные коэффициенты ($i = 1 \ldots m$, $j = 1 \ldots n$). Система из m линейных уравнений с n неизвестными может быть описана при помощи матриц: $A \cdot x = b$, где x — вектор неизвестных, A — матрица коэффициентов при неизвестных или матрица системы, b — вектор свободных членов системы или вектор правых частей. Совокупность всех решений системы $(x_1, x_2, \ldots, x_n)$ называется множеством решений или просто *решением системы*.

Задача 3.1.
Решить СЛАУ при помощи правила Крамера:

$$\begin{aligned} 2x_1 + x_2 - 5x_3 + x_4 &= 8\,, \\ x_1 - 3x_2 - 6x_4 &= 9\,, \\ 2x_2 - x_3 + 2x_4 &= -5\,, \\ x_1 + 4x_2 - 7x_3 + 6x_4 &= 0\,. \end{aligned}$$

Правило Крамера заключается в следующем. Если определитель $\Delta = \det A$ матрицы системы из n уравнений с n неизвестными $A \cdot x = b$ отличен от нуля, то система имеет единственное решение $x_1, x_2, \ldots, x_n$, определяемое по формулам Крамера: $x_i = \Delta_i/\Delta$, где Δ_i — определитель матрицы, полученной из матрицы системы A заменой i-го столбца столбцом свободных членов b. Текст файла-сценария с решением задачи по формулам Крамера:

Листинг 3.46. Текст файла-сценария решения СЛАУ методом Крамера

```
//Матрица коэффициентов:
A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6];
b=[8;9;-5;0];  //Вектор свободных коэффициентов
A1=A;A1(:,1)=b; //Первая вспомогательная матрица
A2=A;A2(:,2)=b; //Вторая вспомогательная матрица
A3=A;A3(:,3)=b; //Третья вспомогательная матрица
A4=A;A4(:,4)=b; //Четвертая вспомогательная матрица
D=det(A); //Главный определитель
//Определители вспомогательных матриц:
d(1)=det(A1); d(2)=det(A2); d(3)=det(A3); d(4)=det(A4);
x=d/D //Вектор неизвестных
P=A*x-b //Проверка
```

Результаты работы файла-сценария:

Листинг 3.47. Вызов файла-сценария решения СЛАУ методом Крамера

```
-->exec('C:\scilab-4.1.1\kramer.sce');disp('exec done');
 x  =
    3.
  - 4.
  - 1.
    1.
 P  = 1.0D-14 *
     0.1776357
        0.
   - 0.0888178
     0.1554312
 exec done
```

Задача 3.2.
Решить СЛАУ из задачи 3.1 методом обратной матрицы.

Метод обратной матрицы: для системы из n линейных уравнений с n неизвестными $A \cdot x = b$, при условии, что определитель матрицы A не равен нулю, единственное решение можно представить в виде $x = A^{-1} \cdot b$.

Текст файла-сценария и результаты его работы:

Листинг 3.48. Решение СЛАУ с использованием функции `inv`

```
//Матрица и вектор свободных коэффициентов системы:
A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6];b=[8;9;-5;0];
```

```
x=inv(A)*b //Решение системы
//Результаты работы файла-сценария:
--> x  =
3.
- 4.
- 1.
1.
```

Задача 3.3.
Решить систему линейных уравнений методом Гаусса:

$$\begin{aligned} 2x_1 - x_2 + 5x_3 &= 0\,, \\ 3x_1 + 2x_2 - 5x_3 &= 1\,, \\ x_1 + x_2 - 2x_3 &= 4\,. \end{aligned}$$

Решение системы линейных уравнений при помощи *метода Гаусса* основывается на том, что от заданной системы переходят к эквивалентной системе, которая решается проще, чем исходная система.

Метод Гаусса состоит из двух этапов. Первый этап — это прямой ход, в результате которого расширенная матрица системы путем элементарных преобразований (перестановка уравнений системы, умножение уравнений на число, отличное от нуля, и сложение уравнений) приводится к ступенчатому виду. На втором этапе (обратный ход) ступенчатую матрицу преобразовывают так, чтобы в первых n столбцах получилась единичная матрица. Последний, $n+1$ столбец этой матрицы содержит решение системы линейных уравнений. Далее приведен текст файла-сценария и результаты его работы:

Листинг 3.49. Использование функции `rref` для решения СЛАУ

```
//Матрица и вектор свободных коэффициентов системы:
A=[2 -1 1;3 2 -5;1 3 -2]; b=[0;1;4];
//Приведение расширенной матрицы к треугольному виду:
C=rref([A b]);
//Определение размерности расширенной матрицы:
[n,m]=size(C);  //m- номер последнего столбца матрицы C
//Выделение последнего столбца из матрицы C:
x=C(:,m)  //x - решение системы
//Результаты работы программы:
--> x  =
    0.4642857
    1.6785714
    0.75
```

Глава 4

Построение двумерных графиков

В этой главе читатель может познакомиться с графическим аппаратом Scilab для построения двумерных графиков. Двумерными будем считать такие графики, в которых положение каждой точки задается двумя величинами.

4.1 Функция plot

Рассмотрение графиков начнем с простейших функций вида $y = f(x)$, для построения которых в Scilab существует функция `plot`. В предыдущих версиях Scilab (по третью версию Scilab включительно) функция `plot` предназначена для построения графика одной функции $y = f(x)$. Обращение к ней имеет вид:

```
plot(x,y,[xcap,ycap,caption])
```

Здесь `x` — массив абсцисс; `y` — массив ординат; `xcap, ycap, caption` — подписи осей X, Y и графика соответственно.

Задача 4.1.
Построить график функции $y = \sin(\cos(x))$ с помощью функции `plot`.

Пусть x изменяется на интервале $[-2\pi; 2\pi]$ с шагом 0,1. Сформируем массив X. Вычисляя значение функции $y = \sin(\cos(x))$ для каждого значения массива X, создадим массив Y. Затем воспользуемся функцией `plot(x,y)` для построения кривой и выведем с ее же помощью подписи координатных осей `'X'`, `'Y'`, а также имя графика `'plot function y=sin(cos(x))'` (см. листинг 4.1, рис. 4.1).

Листинг 4.1. Построение графика функции $y = \sin(\cos(x))$ с помощью функции `plot`

```
x=-2*%pi:0.1:2*%pi;
y=sin(cos(x))';
plot(x,y,'X','Y','plot function y=sin(cos(x))');
```

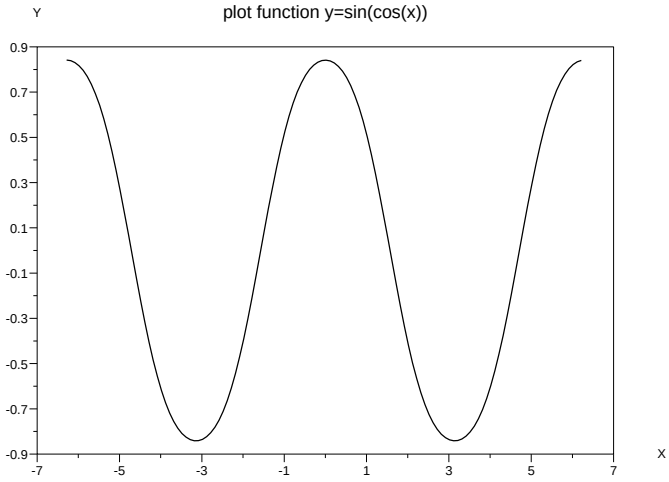

Рис. 4.1. График функции $y = \sin(\cos(x))$

В простейшем случае обращение к функции имеет вид `plot(y)`, в качестве массива `x` выступает массив номеров точек массива `y`. На листинге 4.2 и на рис. 4.2 представлен пример построения графика функции $y = f(i)$.

Листинг 4.2. Построение графика функции вида $y = f(i)$, где `j` — номер точки в массиве `y`

```
y=[1 2 3 -2 4 5 -1 6 9 11 0 -2 5];
plot(y);
```

Такой синтаксис функции `plot` позволяет строить графики нескольких функций.

Задача 4.2.
Построить графики функций $y = \sin(\cos(x))$, $z = \cos(\sin(x))$, $v = e^{\sin(x)}$, $t = e^{\cos(x)}$ в одних координатных осях.

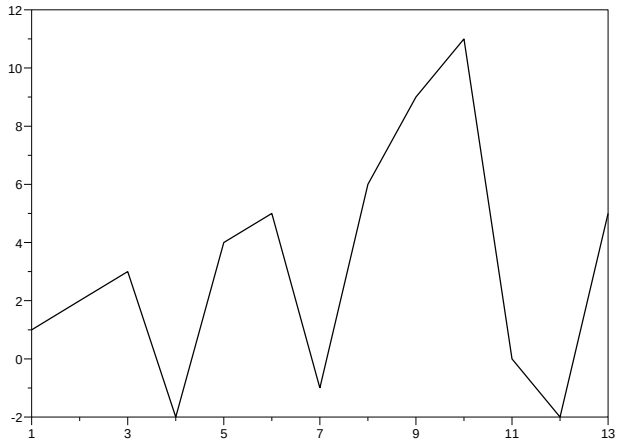

Рис. 4.2. График функции $y = f(i)$

Допустим, что x принадлежит интервалу $[-2\pi; 2\pi]$ и изменяется с шагом 0,1. Создадим массив X. Поскольку x является аргументом для всех четырех функций, его в обращении к функции `plot` можно не указывать. Также необязательно формировать для каждой функции свой массив значений. Достаточно указать в квадратных скобках через точку с запятой их математические выражения, и эти массивы автоматически будут созданы как промежуточный этап построения кривых функций (см. листинг 4.3 и рис 4.3).

Листинг 4.3. Построение графиков нескольких функций в одних координатных осях с помощью команды `plot` в Scilab 3

```
x=-2*%pi:0.1:2*%pi;
plot([sin(cos(x));cos(sin(x));exp(sin(x));exp(cos(x))]);
```

Как видно из рис. 4.3, функция `plot` в Scilab 3 не позволяет построить полноценные графики нескольких функций. Поэтому в Scilab 4 она была значительно модифицирована, и ее возможности теперь значительно расширены. Простейшее же обращение к функции `plot` в четвертой версии приложения осталось прежним `plot(x,y)`.

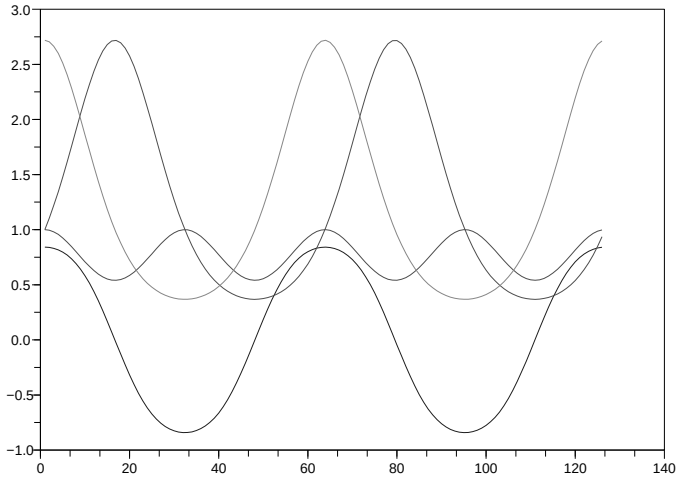

Рис. 4.3. Пример построения графиков нескольких функций в Scilab 3

4.2 Построение нескольких графиков в одной системе координат

При простейшем обращении к функции `plot(x,y)` создается окно с именем *Scilab Graphic (0)*, в котором будет построен график функции `y(x)` на заданном интервале. Если же повторно обратиться к функции `plot`, будет создано новое графическое окно, и в нем будет построен новый график.

Для построения нескольких графиков в одной системе координат можно обратиться к функции `plot` следующим образом:

```
plot(x1,y1,x2,y2,...xn,yn)
```

где `x1, y1` — массивы абсцисс и ординат первого графика;

`x2, y2` — массивы абсцисс и ординат второго графика;

`...`

`xn, yn` — массивы абсцисс и ординат n-ого графика.

Задача 4.3.
Построить в одних координатных осях графики функций $y = \sin(\frac{x}{2})$, $z = \cos(x)$ и $v = \exp(\cos(x))$.

Определим интервал изменения x — [-6,28;6,28], шаг — 0,02. Теперь сформируем массивы значений функций Y, Z, V.

Для построения заданных кривых в одних координатных осях необходимо в качестве аргументов функции `plot` попарно, через запятую, указать имя массива первого аргумента и имя массива первой функции, имя массива второго аргумента и имя массива второй функции и т. д. В нашем случае обращение к функции `plot` будет иметь вид `plot(x,y,x,z,x,v)` (листинг 4.4 и рис. 4.4).

Листинг 4.4. Построение графиков нескольких функций в одних координатных осях с помощью команды `plot` в Scilab 4

```
x=-6.28:0.02:6.28;y=sin(x/2);
z=cos(x);v=exp(cos(x));
plot(x,y,x,z,x,v);
```

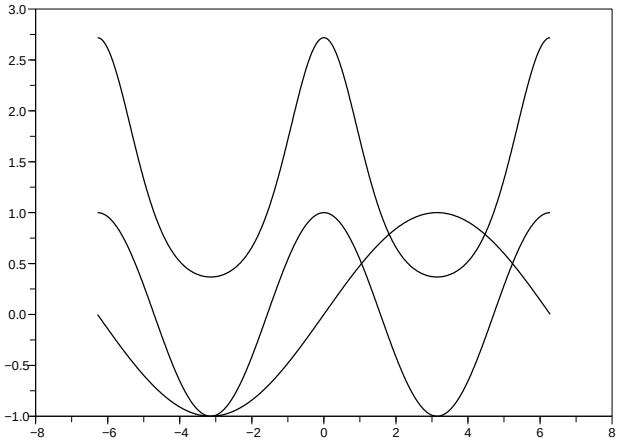

Рис. 4.4. Построение графиков нескольких функций с помощью функции `plot`

Построить несколько графиков в одном окне можно и с помощью короткой записи функции `plot(x,y)`, но перед обращением к функциям `plot(x2,y2)`, `plot(x3,y3)`, `...`, `plot(xn,yn)` вызвать команду `mtlb_hold('on')`, она заблокирует режим создания нового окна.

Задача 4.4.
Построить в одних координатных осях графики функций $y = \sin(\frac{x}{2})$, $z = \cos(x)$ и $v = \exp^{(\cos(x))}$, используя команду `mtlb_hold('on')`.

Как и в предыдущей задаче, прежде всего определяем интервал и шаг изменения x [-6,28;6,28], 0,02 и формируем массивы значений функций Y, Z, V. Однако применять будем краткую форму обращения к функции `plot` — `plot(x,y)`, которая поочередно создаст первый, второй и третий график.

Для того, чтобы каждый раз при выполнении функции `plot` не создавалось новое графическое окно, перед ней будем выполнять команду `mtlb_hold('on')` (листинг 4.5 и рис. 4.5).

Листинг 4.5. Построение графиков нескольких функций в одних координатных осях с помощью команды `mtlb_hold('on')`

```
x=-6.28:0.02:6.28;
y=sin(x/2); z=cos(x); v=exp(cos(x));
plot(x,y);
mtlb_hold(' on');
plot(x,z);
mtlb_hold(' on');
mtlb_hold(' on');
plot(x,v);
```

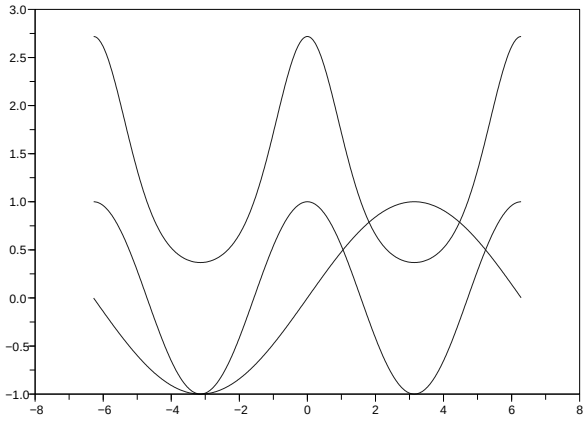

Рис. 4.5. Пример использования команды `mtlb_hold('on')`

Обратите внимание, что при построении графиков первым способом (см. задачу 4.3) Scilab автоматически изменяет цвета кривых, изображаемых в одной системе координат.

4.3 Построение нескольких графиков в одном графическом окне

В Scilab 4 можно выводить несколько графиков в одном окне, не совмещая их в одних координатных осях. Например, если графическое окно должно содержать 4 самостоятельных графика, все окно разбивается на 4 области, а затем в каждую из них выводится график функции.

Для формирования области в графическом окне служит команда `plotframe`:

```
plotframe(rect, tics [,grid, title, x-leg, y-leg, x, y, w, h])
```

где `rect` — вектор `[xmin, ymin, xmax, ymax]`, который определяет границы изменения x и y-координат области;

`tics` — вектор `[nx, mx, ny, my]`, который определяет количество линий сетки по оси X (`mx`) и Y (`my`), величины `nx` и `ny` должны определять число подинтервалов по осям X и Y;

`grid` — логическая переменная, которая определяет наличие `(%t)` либо отсутствие координатной сетки `(%f)`. Этот параметр следует указывать для обеих осей, например, `[%t,%t]`;

`bound` — логическая переменная, которая при значении `true` позволяет игнорировать параметры `tics(2)` и `tics(4)`.

`title` — заголовок, который будет выводится над графической областью;

`x-leg, y-leg` — подписи осей графика X и Y;

`x, y` — координаты верхнего левого угла области в графическом окне, `w` — ширина, `h` — высота окна. Значения `x`, `y`, `w`, `h` измеряются в относительных единицах и лежат в диапазоне $[0, 1]$.

После определения области в нее можно вывести график функции с помощью команды `plot`.

Задача 4.5.
Построить графики функций $y = \sin(2x)$, $z = \cos(3x)$, $u = \cos(\sin(2x))$, $v = \sin(\cos(3x))$ в одном графическом окне, каждый в своей системе координат.

Допустим, что x изменяется на интервале $[-10 : 10]$ с шагом 0,01. Сформируем массивы значений функций Y, Z, U, V.

Используя параметр `rect` как самостоятельную команду, задаем шаблон размера координатных осей каждой области построения кривой. Теперь ось X будет ограничена минимальным и максимальным значением x (зависит от конкретной функции), а ось Y для всех областей ограничивается значениями y -1 и 1.

Командой `tics` указываем, что на всех выводимых графиках, во всех областях на оси абсцисс, должно быть 11 основных и по 2 промежуточных деления, на оси ординат — 5 основных и по 10 промежуточных делений.

Для создания областей внутри графического окна используем функцию `plotframe` со всеми параметрами: прорисовыванием сетки комбинацией значений `(%t)` и `(%f)`, выводом подписи графика и координатных осей, а также разметкой каждой из областей — массив, в котором первые два числа — координаты верхнего левого угла, а последние два — ширина и высота области.

Для формирования нового графика после каждого вызова функции `plotframe` выполняем функцию `plot(x,y)` (листинг 4.6 и рис. 4.6).

Листинг 4.6. Построение графиков нескольких функций в одном графическом окне, но каждого в своих координатных осях с помощью команды `plotframe`

```
x=[-10:0.01:10];
y=sin(2*x); z=cos(3*x); u=cos(sin(2*x)); v=sin(cos(3*x));
rect=[min(x),-1,max(x),1];
tics=[2,11,10,5];
plotframe(rect,tics,[%t,%t],["Function y=sin(2x)",...
"X","Y"], [0,0,0.5,0.5]);
plot(x,y);
plotframe(rect,tics,[%f,%f],["Function y=cos(3x)",...
"X","Y"], [0.5,0,0.5,0.5]);
plot(x,z);
plotframe(rect,tics,[%f,%f],["Function y=cos(sin(2x))",...
 X","Y"], [0,0.5,0.5,0.5]);
plot(x,u);
plotframe(rect,tics,[%t,%t],["Function y=sin(cos(3x))",...
"X","Y"], [0.5,0.5,0.5,0.5]);
plot(x,v);
```

Еще одним способом изображения нескольких графиков в одном окне является использование функции `subplot`. Она также разделяет графическое окно на несколько отдельных областей, однако, по мнению авторов, имеет более простой синтаксис. Обращение к ней имеет вид:

`subplot(m,n,p)` или `subplot(mnp)`

Выполнение функции приводит к тому, что графическое окно разбивается на m окон по вертикали и n окон по горизонтали, текущим окном становится окно с номером p.

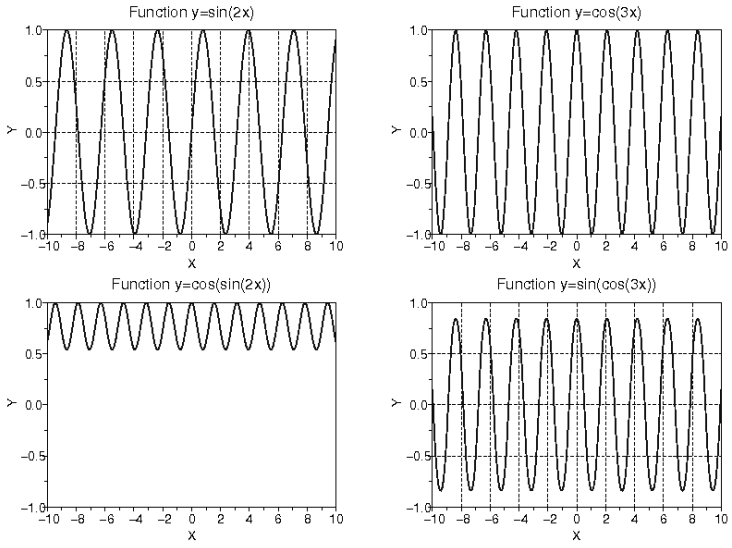

Рис. 4.6. Графики четырех функций в одном графическом окне

Задача 4.6.
Построить графики функций $y = \sin(x)$, $z = \cos(x)$, $u = \cos(\sin(x))$, $v = \sin(\cos(x))$, $w = \exp(\sin(x))$ и $r = \exp(\cos(x))$ в одном графическом окне, каждый в своей системе координат, используя команду `subplot`.

Пусть x изменяется на интервале $[-10 : 10]$ с шагом 0,01. Сформируем массивы значений функций Y, Z, U, V, W, R.

С помощью функции `subplot` разбиваем графическое окно на заданное количество областей. Определимся, что в каждом столбце по вертикали должно быть 3, а по горизонтали 2 области для вывода графиков.

Третье число в записи функции `subplot` указывает, в которую из областей (счет ведется по порядку — слева направо и сверху вниз) выводится график, формируемый функцией `plot(x,y)` (листинг 4.7 и рис. 4.7).

Листинг 4.7. Построение графиков нескольких функций в одном графическом окне, но каждого в своих координатных осях с помощью команды `subplot`

```
x=[-10:0.01:10];
y=sin(x); z=cos(x);
u=cos(sin(x)); v=sin(cos(x));
w=exp(sin(x)); r=exp(cos(x));
subplot(3,2,1);
```

```
plot(x,y);
subplot(3,2,2);
plot(x,z);
subplot(3,2,3);
plot(x,u);
subplot(3,2,4);
plot(x,v);
subplot(3,2,5);
plot(x,w);
subplot(3,2,6);
plot(x,r);
```

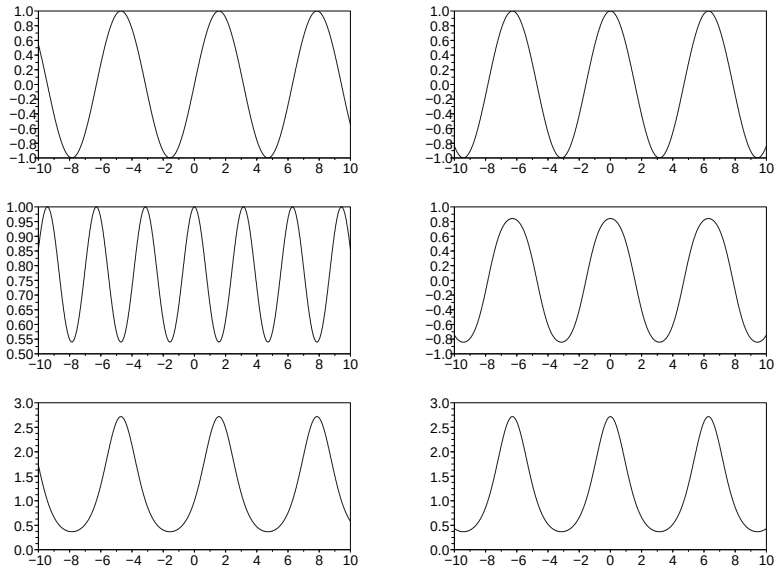

Рис. 4.7. Графики шести функций в одном графическом окне

4.4 Функция plot2d

Следующей функцией, которая может быть использована для построения двумерных графиков, является функция `plot2d`. В общем виде обращение к функции имеет вид:

```
plot2d([logflag],x,y',[key1=value1,key2=value2,...,keyn=valuen]
```

где `logflag` — строка из двух символов, каждый из которых определяет тип осей (`n` — нормальная ось, `l` — логарифмическая ось), по умолчанию — «`nn`»;

`x` — массив абсцисс;

`y` — массив ординат или матрица, каждый столбец которых содержит массив ординат очередного графика — в случае, если необходимо построить графики нескольких функций `y1, y2, ..., yn`, когда все они зависят от одной и той же переменной x. При этом количество элементов в массиве x и y должно быть одинаковым. Если x и y — матрицы одного размера, то каждый столбец матрицы y отображается относительно соответствующего столбца матрицы x;

`keyi=valuei` — последовательность значений свойств графика `key1=value1, key2=value2, ..., keyn=valuen`, определяющих его внешний вид. Возможные значения свойств графика будут подробно описаны ниже.

Следует отметить, что вовсе не обязательно использовать полную форму записи функции `plot2d` со всеми ее параметрами. В простейшем случае к ней можно обратиться кратко, как и к функции `plot`.

Задача 4.7.
Построить график функции $y = \sin(x)$.

Допустим, что x изменяется на интервале $[-2\pi; 2\pi]$ с шагом 0,1. Сформируем массив X. Создавать массив Y необязательно, следует лишь в качестве аргумента функции `plot2d` указать математическое выражение функции (см. листинг 4.8 и рис. 4.8).

Листинг 4.8. Пример простейшего использования функции `plot2d` для построения графика

```
x=[-2*%pi:0.1:2*%pi];
plot2d(sin(x));
```

Используя функцию `plot2d`, можно также построить несколько графиков в одних координатных осях.

Задача 4.8.
Построить графики функций $y = \sin(x)$, $y_1 = \sin(2x)$, $y_2 = \sin(3x)$ в одних координатных осях.

Сформируем массив X, приняв, что x изменяется на интервале $[0; 2\pi]$ с шагом 0,1. Для построения кривых в одних координатных осях воспользуемся функцией `plot2d(x,y)`. Однако в качестве массива Y в квадратных скобках поочередно укажем математические выражения заданных функций, разделяя их пробелами (см. листинг 4.9 и рис. 4.9).

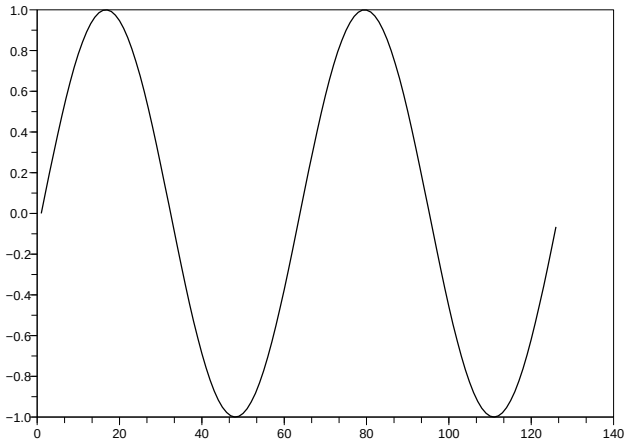

Рис. 4.8. График функции $y = \sin(x)$

Листинг 4.9. Построение графиков нескольких функций в одних координатных осях с помощью команды `plot2d`

```
x=[0:0.1:2*%pi]';
plot2d(x,[sin(x) sin(2*x) sin(3*x)]);
```

4.5 Оформление графиков при помощи функции plot

Установить желаемый вид и цвет графика можно, используя полную форму обращения к функции `plot`:

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

где `x1, x2, ..., xn` — массивы абсцисс графиков;

`y1, y2, ..., yn` — массивы ординат графиков;

`s1, s2, ..., sn` — строка, состоящая из трех символов, которые определяют соответственно цвет линии, тип маркера и тип линии графиков (см. табл. 4.1–4.3), в строке могут использоваться один, два или три символа одновременно в любой желаемой комбинации.

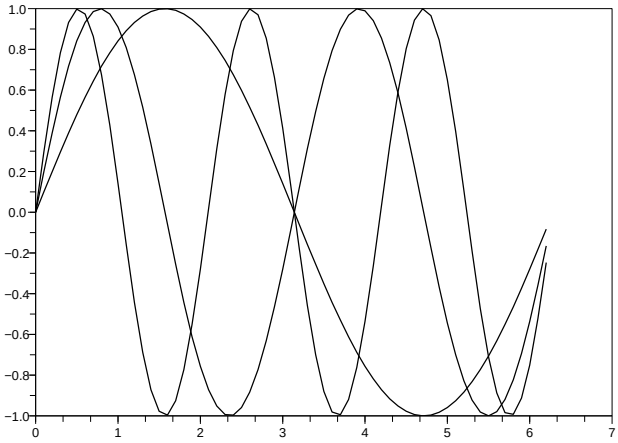

Рис. 4.9. Графики функций $y = \sin(x)$, $y = \sin(2x)$, $y = \sin(3x)$

Таблица 4.1. Символы, определяющие цвет линии графика

Символ	**Описание**
`y`	желтый
`m`	розовый
`c`	голубой
`r`	красный
`g`	зеленый
`b`	синий
`w`	белый
`k`	черный

Таблица 4.2. Символы, определяющие тип маркера

Символ	Описание
.	точка
o	кружок
x	крестик
+	знак «плюс»
*	звездочка
s	квадрат
d	ромб
v	треугольник вершиной вниз
^	треугольник вершиной вверх
<	треугольник вершиной влево
>	треугольник вершиной вправо
p	пятиконечная звезда

Таблица 4.3. Символы, определяющие тип линии графика

Символ	Описание
-	сплошная (по умолчанию)
:	штрих, чередующийся с двумя точками
-.	штрих, чередующийся с одной точками
--	штриховая

В качестве примера построим графики функций $y = \sin(\frac{x}{2})$, $z = \cos(x)$ и $v = e^{(\cos(x))}$ в одних координатных осях и независимо определим внешний вид каждого графика. Пусть график функции $y = \sin(\frac{x}{2})$ будет штриховым, $z = \cos(x)$ — черного цвета, с маркером в виде звездочки, $v = \exp^{(\cos(x))}$ — штриховым, красного цвета, с маркером в виде точки (см. листинг 4.10 и рис. 4.10).

Листинг 4.10. Изменение типа, цвета линии графика, вывод точек (маркеров) на линии графика с помошью функции `plot`

```
x=-6.28:0.2:6.28;
y=sin(x/2);
z=cos(x);
v=exp(cos(x));
plot(x,y,'--');
mtlb_hold('on');
plot(x,z,'k*');
mtlb_hold('on');
plot(x,v,'r.--');
```

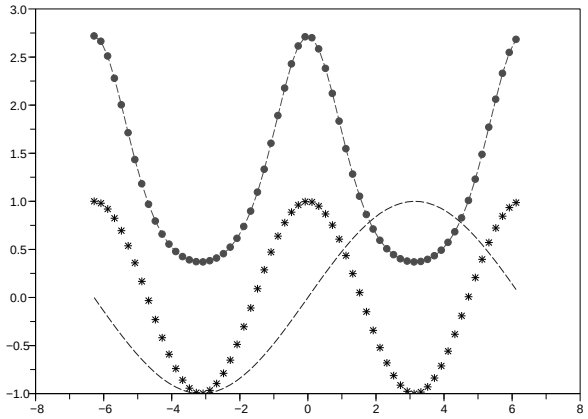

Рис. 4.10. Управление свойствами линии графика при помощи функции `plot`

Чтобы график проще «читался», удобно выводить сетку — дополнительные оси для показателя X и показателя Y. В Scilab это можно сделать с помощью команды `xgrid(color)`, где `color` определяет `id` цвета линии сетки. Если оставить скобки пустыми, по умолчанию будет прорисована сетка черного цвета.

Построим в одних координатных осях синусоиду с линией красного цвета и косинусоиду с линией синего цвета на заданном интервале, а затем выведем сетку (см. листинг 4.11 и рис. 4.11).

Листинг 4.11. Вывод сетки графика с помощью команды `xgrid`

```
x=-10:0.01:10;
y=sin(cos(x));
z=cos(sin(x));
plot(x,y,'r',x,z,'b');
xgrid();
```

Заголовок графика, построенного функцией `plot`, можно вывести командой `xtitle`:

```
xtitle(title, xstr, ystr)
```

где `title` — название графика;

`xstr` — название оси X;

`ystr` — название оси Y.

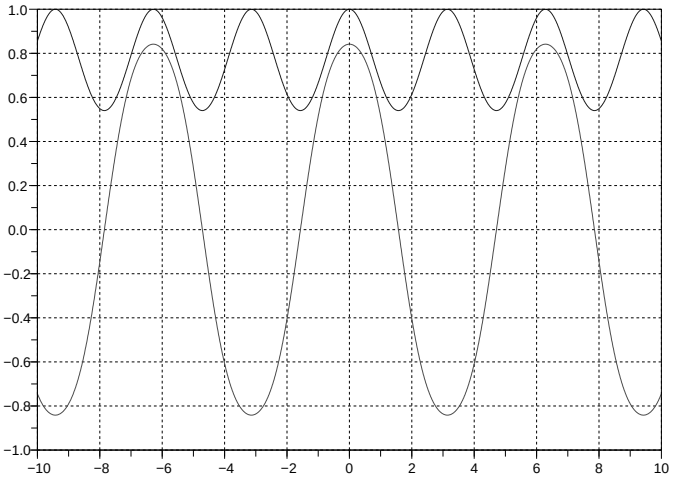

Рис. 4.11. Команда `xgrid`

Воспользуемся предыдущим примером и добавим к графику заголовок `Grafic y=f(x)` и подписи к координатным осям `X` и `Y` (см. листинг 4.12 и рис. 4.12).

Листинг 4.12. Вывод заголовка графика и подписей координатных осей с помощью команды `xtitle`

```
x=-10:0.01:10;
y=sin(cos(x));
z=cos(sin(x));
plot(x,y,'r',x,z,'b');
xgrid();
xtitle('Grafic y=f(x)','X','Y');
```

В случаях, когда в одной координатной плоскости изображаются графики нескольких функций, как в нашем примере, возникает необходимость в «легенде». Ее можно вывести с помощью команды `legend`:

```
legend(leg1, leg2, ..., legn, [pos], [boxed])
```

где `leg1` — имя первого графика, `leg2` — имя второго графика, `legn` — имя n-го графика;

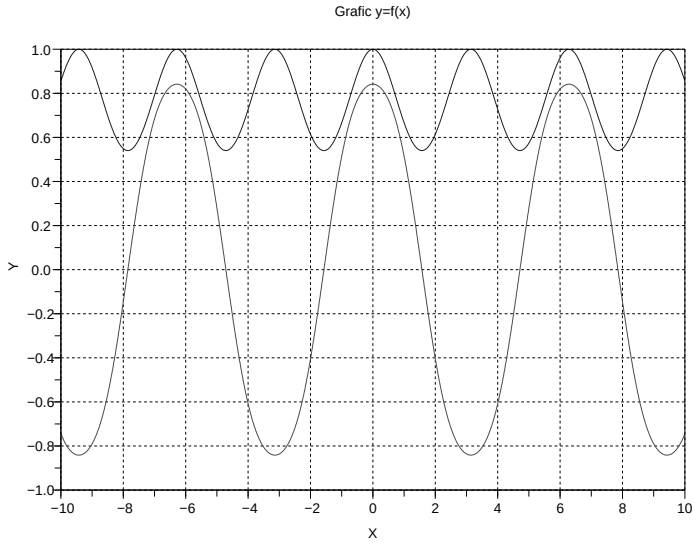

Рис. 4.12. Команда `xtitle`

`pos` — месторасположение легенды: 1 — в верхнем правом углу (по умолчанию), 2 — в верхнем левом углу, 3 — в нижнем левом углу, 4 — в нижнем правом углу, 5 — определяется пользователем после изображения графика;

`boxed` — логическая переменная, которая определяет, прорисовывать (значение по умолчанию — `%t`) или нет (значение `%f`) рамку вокруг легенды.

Выведем на графике предыдущего примера легенду в левом нижнем углу, отменив построение рамочки вокруг легенды (см. листинг 4.13 и рис. 4.13).

Листинг 4.13. Вывод легенды графика и определение ее свойств с помощью команды `legend`

```
x=-10:0.01:10;
y=sin(cos(x)); z=cos(sin(x));
plot(x,y,'r',x,z,'b');
xgrid();
xtitle('Grafic y=f(x)','X','Y');
legend('sin(cos(x))','cos(sin(x))',3,%f);
```

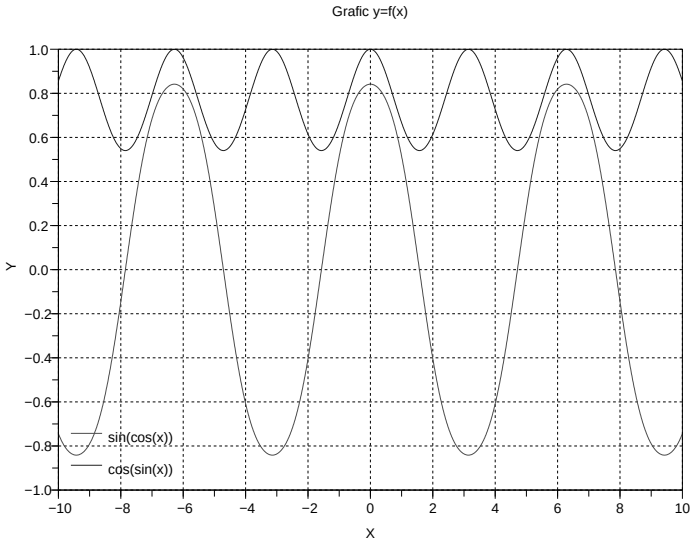

Рис. 4.13. Команда `legend`

4.6 Оформление графиков при помощи функции plot2d

В параграфе 4.4 упоминалось, что полная форма обращения к функции `plot2d` дает возможность пользователю самостоятельно определять внешний вид графика — за это в ней отвечает параметр `keyn=valuen`.

Напомним, что полная форма обращения к функции `plot2d` имеет вид:

```
plot2d([logflag],x,y',[key1=value1,key2=value2,...,keyn=valuen]
```

Возможны следующие значения параметра `keyn=valuen`:

- `style` — определяет массив числовых значений цветов графика. Количество элементов массива совпадает с количеством изображаемых графиков. Можно воспользоваться функцией `color`, которая по названию (`color("имя цвета")`) или коду rgb (`color(r,g,b)`) цвета формирует нужный *id (код)* цвета.

  Полный перечень всех доступных при форматировании оттенков с их RGB-id можно найти в статье встроенной справочной системы Scilab «*Color_list*». Однако следует учесть, что в статье перед id-цвета опущены «0.».

  В качестве примера построим в одних координатных осях графики функций $y = \sin(x)$ и $y = \cos(x)$, для синусоиды с помощью параметра `style` определим имя цвета — красный («red»), а для косинусоиды — *id* зеленого цвета (0,176,0) (см. листинг 4.14).

Полученные графики представлены на рис. 4.14.

Листинг 4.14. Изменение цвета линии графика с помощью параметра `style` функции `plot2d`

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x);cos(x)];
plot2d(x,y',style=[color("red"),color(0,176,0)]);
```

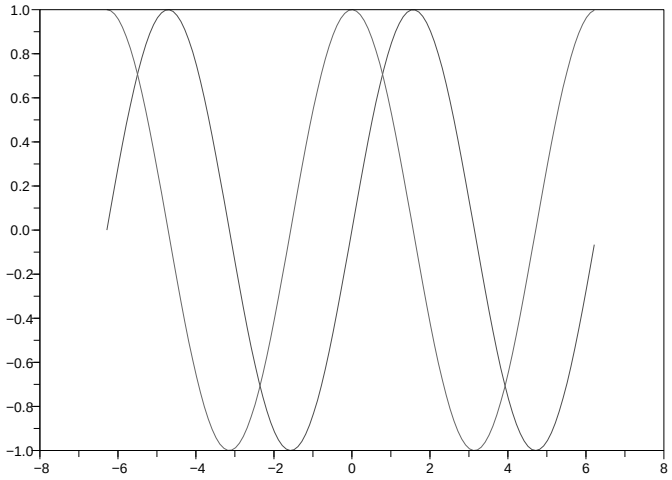

Рис. 4.14. Параметр `style` функции `plot2d`

- `rect` — значение параметра `keyn=valuen` функции `plot2d` — это вектор `[xmin, ymin, xmax, ymax]`, определяющий размер окна вокруг графика.

  Здесь `xmin, ymin` — положение верхней левой вершины;

  `xmax` — ширина окна;

  `ymax` — высота окна.

Применим параметр `rect` к предыдущему примеру, установив для него следующие значения [-8,-2,8,2] (см. листинг 4.15 и рис. 4.15). Вследствие того, что ось Y продлилась от [-1:1] до [-2:2], а ось X осталась без изменений, визуально кажется, будто график сжался по оси Y.

Листинг 4.15. Определение размеров окна вокруг графика с помощью параметра `rect` функции `plot2d`

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x);cos(x)];
plot2d(x,y',style=[color("red"),color(0,176,0)],
 rect=[-8,-2,8,2]);
```

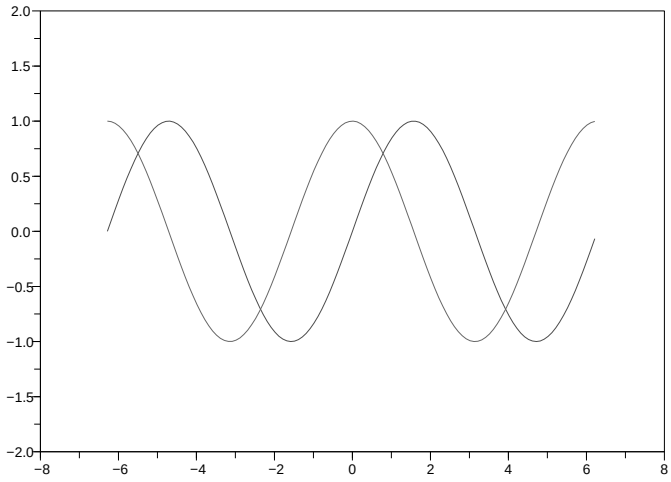

Рис. 4.15. Параметр `rect` функции `plot2d`

- `axesflag` — значение параметра `keyn=valuen` функции `plot2d` — определяет наличие рамки вокруг графика. Необходимо выделить следующие базисные значения этого параметра:

  0 — нет рамки;

  1 — изображение рамки, ось y слева (по умолчанию);

  3 — изображение рамки, ось y справа;

  5 — изображение осей, проходящих через точку (0,0).

Задача 4.9.
Построить графики функций $y = \sin(x)$ и $y_1 = \cos(x)$, используя 4 базисные значения параметра `axesflag`, в одном графическом окне при помощи функции `subplot`.

Напомним, что функция *subplot* позволяет выводить несколько графиков в одном графическом окне, не совмещая их в одной системе координат (см. параграф 4.3).

Разметим графическое окно на 4 области. Действие параметра `axesflag=0` будет выведено в верхнюю левую, `axesflag=1` — в верхнюю правую, `axesflag=3` — в нижнюю левую, а `axesflag=5` — в нижнюю правую (см. листинг 4.16 и рис. 4.16).

Листинг 4.16. Вывод рамки вокруг графика и определение размера координатных осей с помощью параметра `axesflag` функции `plot2d`

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
subplot(2,2,1)
plot2d(x,y',style=[color("red"), color("blue")], axesflag=0);
subplot(2,2,2)
plot2d(x,y',style=[color("red"), color("blue")], axesflag=1);
subplot(2,2,3)
plot2d(x,y',style=[color("red"), color("blue")], axesflag=3);
subplot(2,2,4)
plot2d(x,y',style=[color("red"), color("blue")], axesflag=5);
```

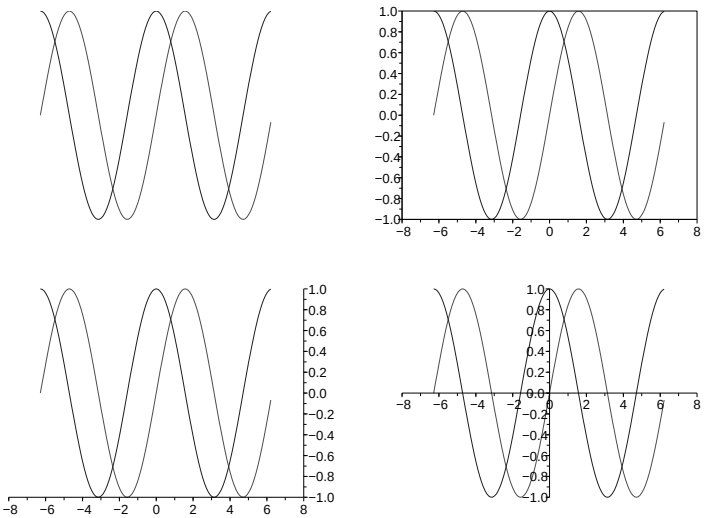

Рис. 4.16. Параметр `axesflag` функции `plot2d`

- Для того, чтобы определить число основных и промежуточных делений координатных осей, в Scilab существует параметр `nax`. Если параметр `axesflag=1` (по умолчанию), это массив из четырех значений: `[nx, Nx, ny, Ny]`.

  Здесь `Nx (Ny)` — число основных делений с подписями под осью X (Y);

  `nx (ny)` — число промежуточных делений.

Задача 4.10.
Построить графики функций $y = \sin(x)$ и $y_1 = \cos(x)$. Модифицировать масштаб координатных осей графика.

Сформируем массив X, приняв, что x изменяется в диапазоне [-8:8] с шагом 0,1, затем совместно сформируем массивы значений заданных функций с помощью следующей записи: `y=[sin(x); cos(x)]`.

С помощью функции `plot2d` построим кривые функций $y = \sin(x)$ и $y_1 = \cos(x)$, установив значение параметра `nax=[4,9,3,6]`. Таким образом, ось X будет разбита 9 основными делениями (засечками), каждое основное — 4 промежуточными, а ось Y — соответственно 6 и 3 (см. листинг 4.17 и рис. 4.17).

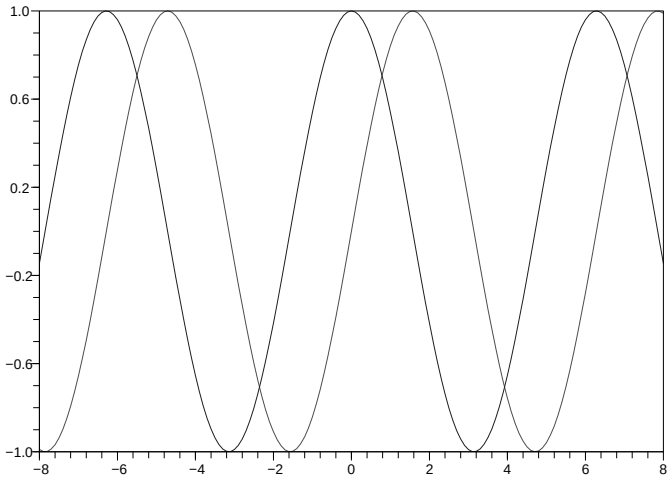

Рис. 4.17. Параметр `nax` функции `plot2d`

Листинг 4.17. Нанесение на координатные оси графика основных и промежуточных делений с помощью параметра `nax` функции `plot2d`

```
x=[-8:0.1:8];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"),color("blue")],axesflag=1,...
nax=[4,9,3,6]);
```

- `leg` — значение параметра `keyn=valuen` функции `plot2d` — строка, определяющая легенды для каждого графика:

  ```
  "leg1@leg2@leg3@...@legn"
  ```

  где `leg1` — легенда первого графика,

  `..., legn` — легенда n-го графика.

  В качестве примера используем предыдущую задачу. Построим графики функций $y = \sin(x)$ и $y_1 = \cos(x)$ с пересечением осей X и Y в точке (0,0) — значение параметра `axesflag=5`, выведем легенду с подписями для обеих кривых (см. листинг 4.18 и рис. 4.18).

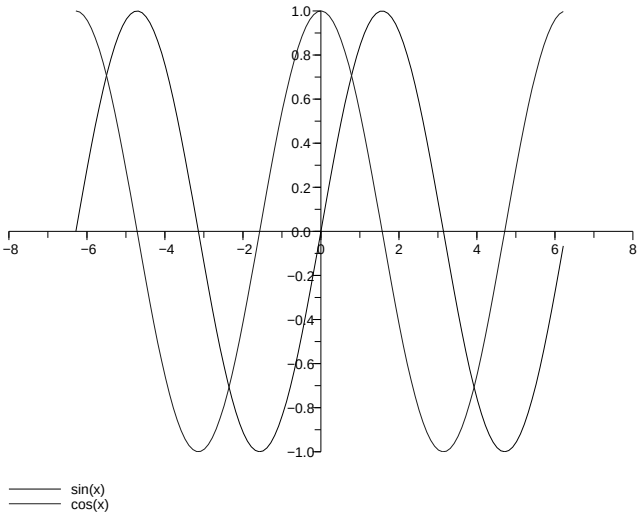

Рис. 4.18. Параметр `leg` функции `plot2d`

Листинг 4.18. Вывод легенды графика с помощью параметра `leg` функции `plot2d`

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',axesflag=5, leg="sin(x)@cos(x)");
```

4.7 Построение точечных графиков

Функцию `plot2d` можно использовать для построения точечных графиков. В этом случае обращение к функции имеет вид:

```
plot2d(x,y,d)
```

где `d` — отрицательное число, определяющее тип маркера (см. табл. 4.4).

Таблица 4.4. Числа, определяющие тип маркера

Число	Описание
-0	точка
-1	плюс
-2	крестик
-3	плюс, вписанный в окружность
-4	закрашенный ромб
-5	незакрашенный ромб
-6	треугольник вершиной вверх
-7	треугольник вершиной вниз
-8	плюс, вписанный в ромб
-9	кружок
-10	звездочка
-11	квадрат
-12	треугольник вершиной вправо
-13	треугольник вершиной влево
-14	пятиконечная звезда

Задача 4.11.
Построить точечный график функции $y = \sin(x)$ с типом маркера «плюс, вписанный в ромб».

Определив диапазон изменения x, сформируем массивы значений X и Y.

При построении кривой с помощью функции `plot2d` укажем аргумент -8, определяющий тип маркера «плюс, вписанный в ромб» (см. листинг 4.19 и рис. 4.19).

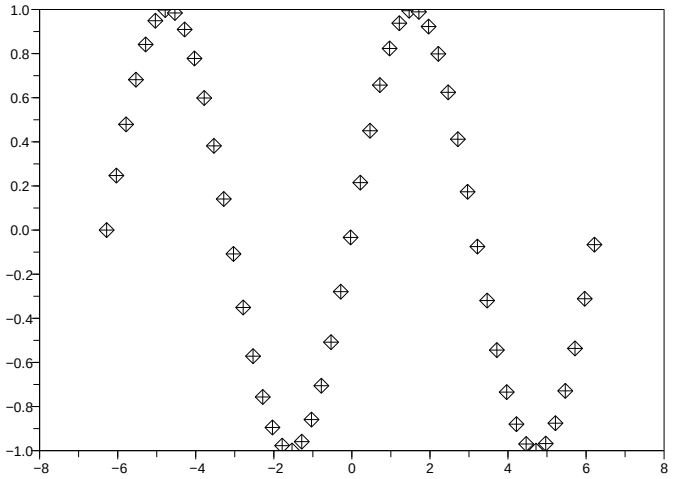

Рис. 4.19. Точечный график функции $y = \sin(x)$

Листинг 4.19. Построение точечных графиков в Scilab

```
x=[-2*%pi:0.25:2*%pi];
y=sin(x);
plot2d(x,y,-8);
```

4.8 Построение графиков в виде ступенчатой линии

Для изображения графика в виде ступенчатой линии в Scilab существует функция `plot2d2(x,y)`. Она полностью совпадает по синтаксису с функцией `plot2d`. Главное отличие состоит в том, что X и Y могут быть независимыми друг от друга функциями, важно лишь, чтобы массивы X и Y были разбиты на одинаковое количество интервалов.

Задача 4.12.
Имеются детальные наблюдения за ростом народонаселения на планете за период с 1947 по 2006 год, млн. чел. Построить график, отражающий динамику процесса на основании данных 1947, 1958, 1970, 1980, 1999 и 2006 годов.

Поэлементно введем массивы X и Y и воспользуемся функцией `plot2d2(x,y)` (см. листинг 4.20 и рис. 4.20).

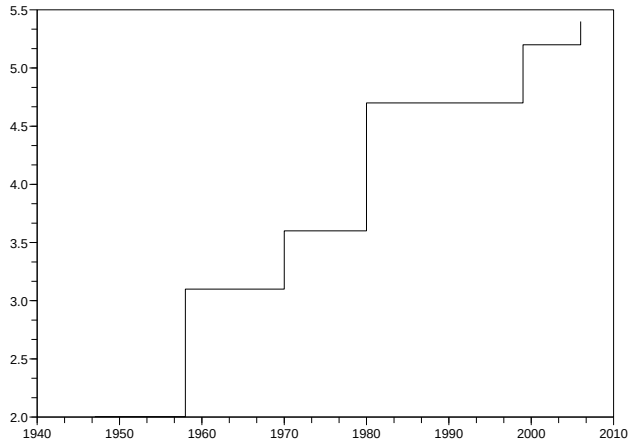

Рис. 4.20. Пример построения ступенчатого графика при помощи функции `plot2d2`

Листинг 4.20. Построение графиков в виде ступенчатой линии в Scilab

```
x=[1947 1958 1970 1980 1999 2006];
y=[2.003 3.1 3.6 4.7 5.2 5.4];
plot2d2(x,y);
```

4.9 Построение графиков в полярной системе координат

Полярная система координат состоит из заданной фиксированной точки O — полюса, концентрических окружностей с центром в полюсе и лучей, выходящих из точки O, один из которых OX — полярная ось.

Расположение любой точки M в полярных координатах можно задать положительным числом $\rho = OM$ (полярный радиус), и числом φ, равным величине угла XOM (полярный угол).

В Scilab для формирования графика в полярной системе координат необходимо сформировать массивы значений полярного угла и полярного радиуса, а затем обратиться к функции `polarplot`:

`polarplot(fi,ro,[key1=value1,key2=value2,...,keyn=valuen])`,

где `fi` — полярный угол;

`ro` — полярный радиус;

`keyn=valuen` — последовательность значений свойств графика (подробнее см. параграф 4.4).

Задача 4.13.
Построить полярные графики функций $\rho = 3\cos(5\varphi)$ и $\rho_1 = 3\cos(3\varphi)$.

Определив диапазон и шаг изменения полярного угла, формируем массивы `fi, ro`.

Поочередно строим заданные кривые с помощью функции `polarplot`, при этом для линии графика функции `ro` установим красный цвет, а для функции `ro1` — синий (см. листинг 4.21, рис. 4.21).

Листинг 4.21. Пример построения графиков функций в полярной системе координат

```
fi=0:0.01:2*%pi;
ro=3*cos(5*fi);ro1=3*cos(3*fi);
polarplot(fi,ro,style=color("red"));
polarplot(fi,ro1,style=color("blue"));
```

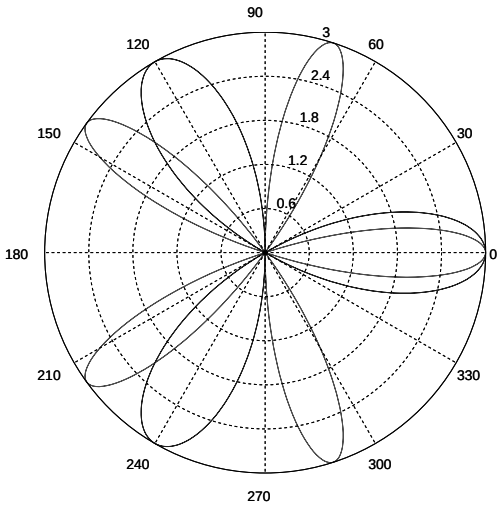

Рис. 4.21. Полярные графики функций $\rho = 3\cos(5\varphi)$, $\rho_1 = 3\cos(3\varphi)$

4.10 Построение графиков функций, заданных в параметрической форме

Задание функции $y(x)$ с помощью равенств $x = f(t)$ и $y = g(t)$ называют параметрическим, а вспомогательную величину t — параметром.

Для построения графика функции, заданной параметрически, необходимо определить массив t, определить массивы $x = f(t)$, $y = g(t)$ и построить график функции $y(x)$, используя функции `plot(x,y)` либо `plot2d(x,y)`.

Задача 4.14.
Построить график строфоиды.

Напомним, что строфоида представляет собой алгебраическую кривую третьего порядка, которая в общем виде задается уравнением:

$$x^2(a + x) = y^2(a - x) \tag{4.1}$$

Представим это уравнение с помощью параметра t:

$$\begin{cases} X(t) = \dfrac{(t^2 - 1)}{(t^2 + 1)} \\ Y(t) = \dfrac{t \cdot (t^2 - 1)}{(t^2 + 1)} \end{cases} \tag{4.2}$$

Зададим массивы t, x и y и построим график с помощью функции `plot(x,y)` (см. листинг 4.22, рис. 4.22).

```
t=-5:0.01:5;
x=(t.^2-1)./(t.^2+1);
y=t.*(t.^2-1)./(t.^2+1);
plot(x,y);
```

Листинг 4.22. Построение строфоиды с помощью функции `plot`

Задача 4.15.
Построить график полукубической параболы.

Полукубическая парабола — это алгебраическая кривая второго порядка, которая в общем виде может быть описана следующим уравнением:

$$y^m = A + Bx + Cx^2 + \ldots + Nx^n \tag{4.3}$$

Приведем это уравнение к параметрической форме:

$$\begin{cases} x(t) = 0.5t^2 \\ y(t) = 0.3t^3 \end{cases} \tag{4.4}$$

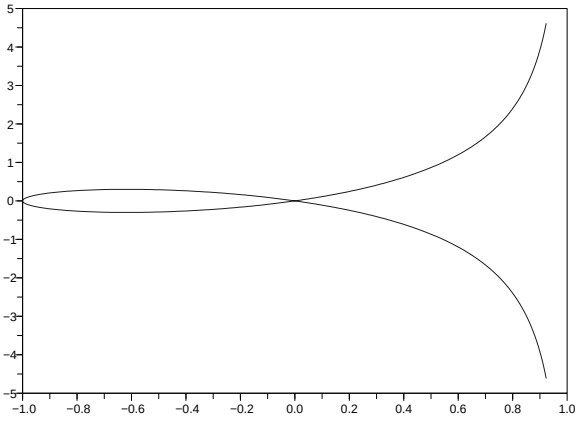

Рис. 4.22. График строфоиды

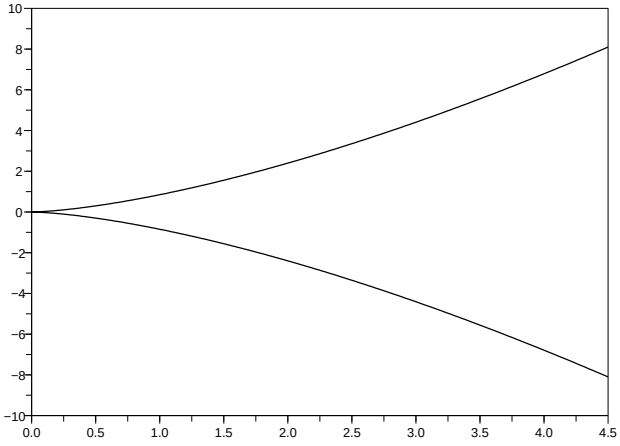

Рис. 4.23. График полукубической параболы

Как и в примере со строфоидой, t-параметр определяем как массив, а x и y как зависимые от него величины. Однако теперь для построения графика обратимся к функции `plot2d(x,y)` (см. листинг 4.23, рис. 4.23).

Листинг 4.23. Построение полукубической параболы с помощью функции `plot2d`

```
t=-3:0.01:3;
x=0.5*t.^2;
y=0.3*t.^3;
plot2d(x,y);
```

4.11 Режим форматирования графика

В Scilab внешний вид графика можно изменять, используя возможности графического окна, в котором он отображается. Переход к режиму форматирования осуществляется командой *Edit – Figure properties* меню графического окна.

Возможности форматирования мы рассмотрим на примере построения графиков функций $y_1 = \sin(2x)$ и $y_2 = \sin(3x)$ на интервале $[0; 2\pi]$ с шагом 0,1. Сформируем массив x и воспользуемся функцией `plot2d` (см. листинг 4.24, рис. 4.24).

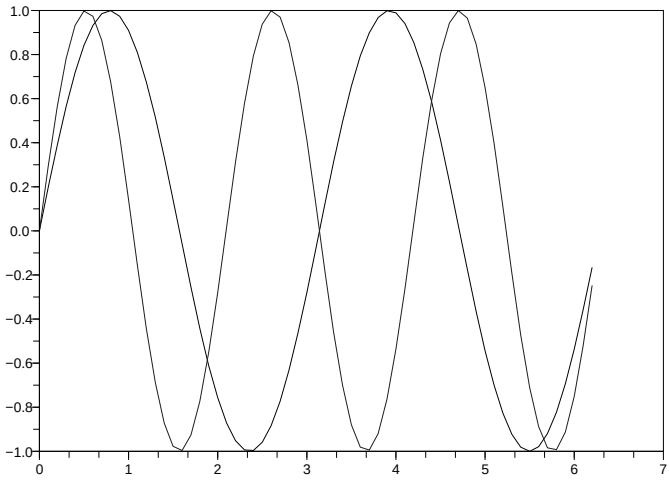

Рис. 4.24. Графики функций $y_1 = \sin(2x)$ и $y_2 = \sin(3x)$

Листинг 4.24. Построение графиков функций $y_1 = \sin(2x)$ и $y_2 = \sin(3x)$ с помощью функции `plot2d`

```
x=[0:0.1:2*%pi]';
plot2d(x,[sin(2*x) sin(3*x)]);
```

Командой меню графического окна *Edit – Figure properties* вызываем окно форматирования полученного графика *Figure Editor* (см. рис. 4.25).

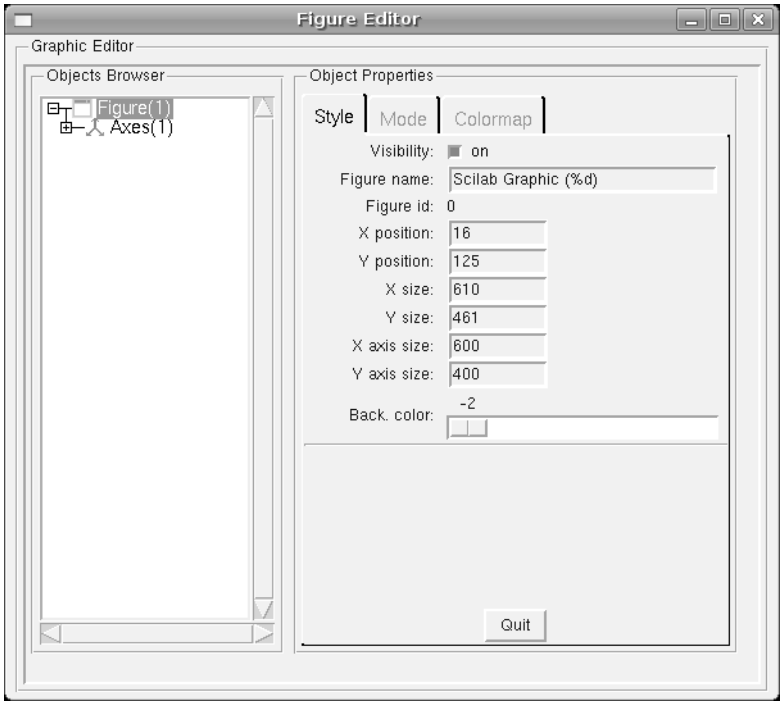

Рис. 4.25. Окно форматирования графика *Figure Editor*

Левая часть окна — *Object Browser* — это поле просмотра объектов, доступных для форматирования. Щелчок по объекту *Figure(1) (Графическое окно)* делает его активным, а в правой области окна — *Object Properties* — появляются свойства активного объекта, которые могут быть изменены.

Первоначально в поле *Object Browser* всегда отображаются два объекта: *Figure (Графическое окно)* и его дочерний объект *Axes (Оси)*. Значок «плюс» возле объекта указывает на то, что он содержит объекты более низкого порядка.

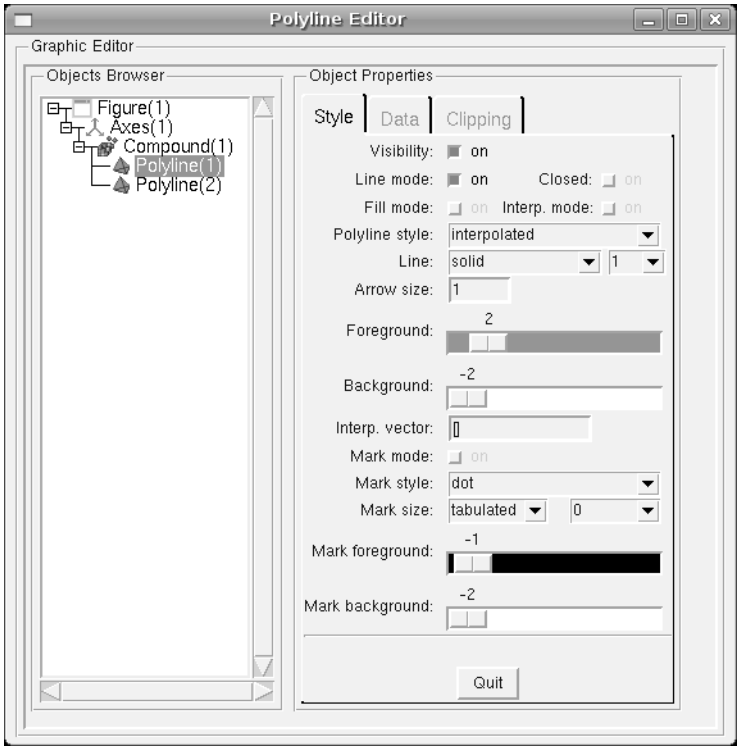

Рис. 4.26. Иерархия объектов в поле *Object Browser*

Если щелкнуть по значку «плюс» у объекта *Axes (Оси)*, появится объект — *Compound(1) (Группа)* также со значком «плюс». Объект *Compound(1)* содержит построенные в одних координатных осях графики функций $y_1 = sin(2x)$ и $y_2 = sin(3x)$ — соответственно *Polyline(2)* и *Polyline(1)* (см. рис. 4.26).

4.11.1 Форматирование объекта Figure (Графическое окно)

Напомним, что объект *Figure* — это графическое окно и собственно график, отображаемый в нем. Для изменения свойств графического окна необходимо выделить *Figure(1)* в поле *Object Browser* окна форматирования.

На рис. 4.26 изображена закладка *Style* окна форматирования *Figure Editor* для объекта *Figure(1)*. Здесь можно изменить значения следующих свойств:

Visibility (отображение графика) — переключатель, принимающий значения «on» и «off». По умолчанию установлено состояние «on» — график выводится на экран.

Figure name (имя графика) — это последовательность символов, которые выводятся в строке заголовка графического окна. По умолчанию графическому окну присваивается имя *Scilab Graphic (%d)*, где *%d* — это порядковый номер графика *(Figure id)*. Для первого графического окна *Figure id* равен 0, для второго — 1, для третьего — 2 и т. д. Однако вы можете ввести любое желаемое имя. Например, заменить *Scilab Graphic (%d)* на *Grafic y=f(x)* и нажать клавишу ***Enter***. Заголовок окна будет изменен (см. рис. 4.27).

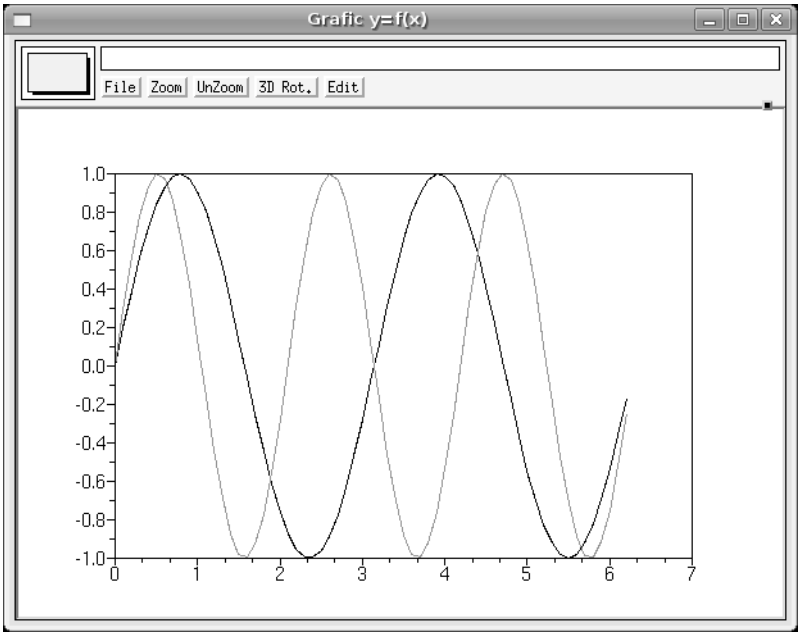

Рис. 4.27. Изменение заголовка графического окна в окне форматирования

X position, Y position — эти поля определяют положение графического окна на мониторе в пикселях по горизонтали и вертикали соответственно. Точка с координатами [0;0] — верхний левый угол экрана.

X size, Y size — это соответственно ширина и высота графического окна в пикселях.

X axis size, Y axis size — эти значения определяют размер осей X и Y.

Back. color (цвет фона) — каждому положению ползунка соответствует свой номер цвета (RGB-id). Доступны 35 оттенков (от -2 — белый до 32 — желто-горячий).

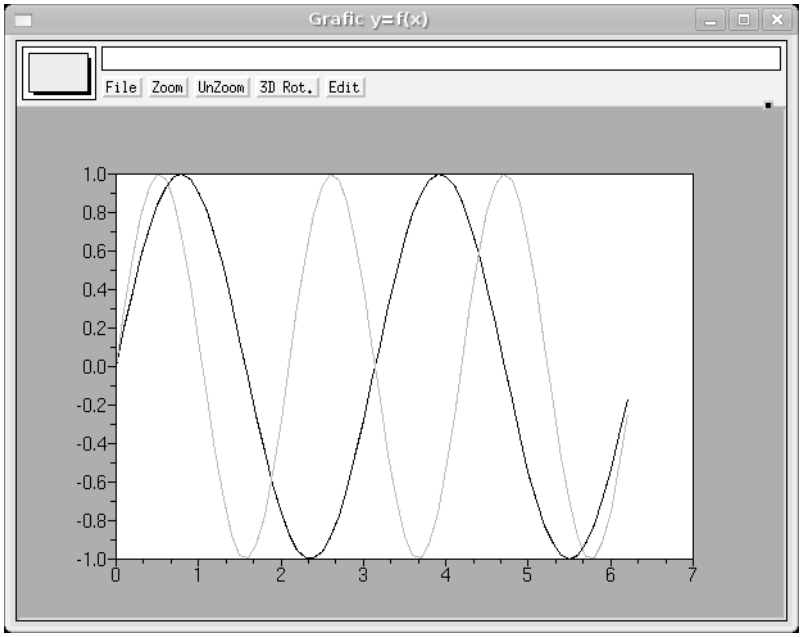

Рис. 4.28. Изменение фона графика в окне форматирования *Figure Editor*

Установим ползунок в положение 15. Цвет фона станет зеленым (см. рис. 4.28).

Цветовая палитра может быть изменена пользователем на закладке *Colormap* (см. рис. 4.29).

Устанавливая значения для красного (*RED*), зеленого (*GREEN*) и синего (*BLUE*) цветов, можно изменять каждый оттенок независимо, не вызывая изменений других цветов в палитре. Например, вектор [0 0 0] задает черный цвет, [0.230 0.230 0.250] — лазурный, [0.85 0.107 0.47] — малиновый.

В строке *Colormap (Nx3 double array)* можно задать RGB-id для всей палитры.

Полный перечень всех доступных при форматировании оттенков с их RGB-id можно найти в статье встроенной справочной системы Scilab «*Color_list*». Однако следует учесть, что в статье перед id-цвета опущены «0.».

На закладке *Mode* в области *Object Properties* для объекта *Figure* (см. рис. 4.30) можно установить следующие свойства:

Auto resize — свойство, которое позволяет изменять размер графика. Когда этот режим включен (положение переключателя «on» — по умолчанию), мы можем изменять размер графического окна, перетаскивая его границы с помощью мыши, и при этом автоматически будет изменяться размер графика, отображаемого в окне. В выключенном положении график будет сохранять свои размеры (см. рис. 4.31).

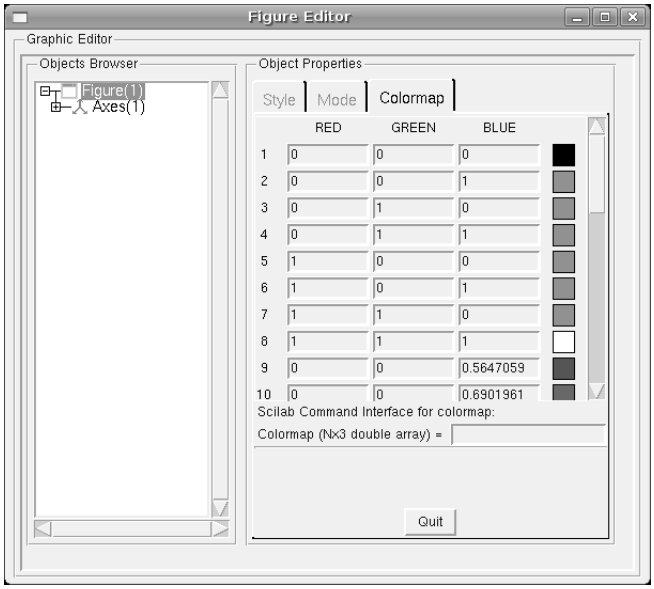

Рис. 4.29. Изменение палитры цветов на закладке *Colormap* окна форматирования *Figure Editor*

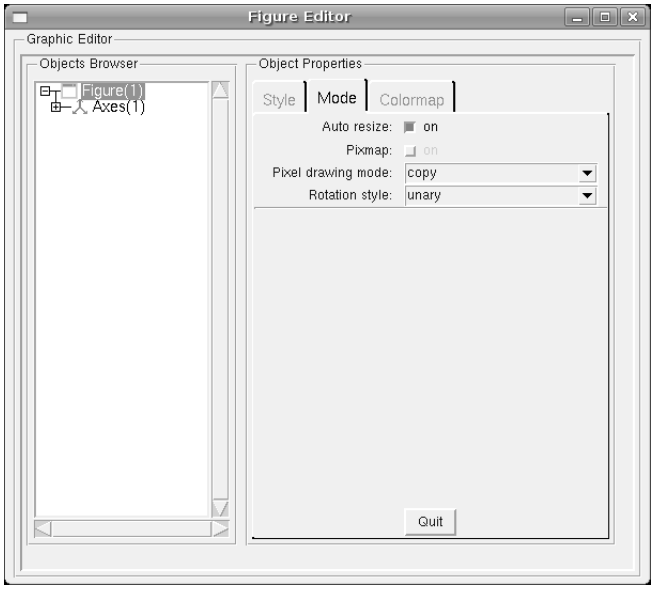

Рис. 4.30. Закладка Mode окна форматирования *Figure Editor*

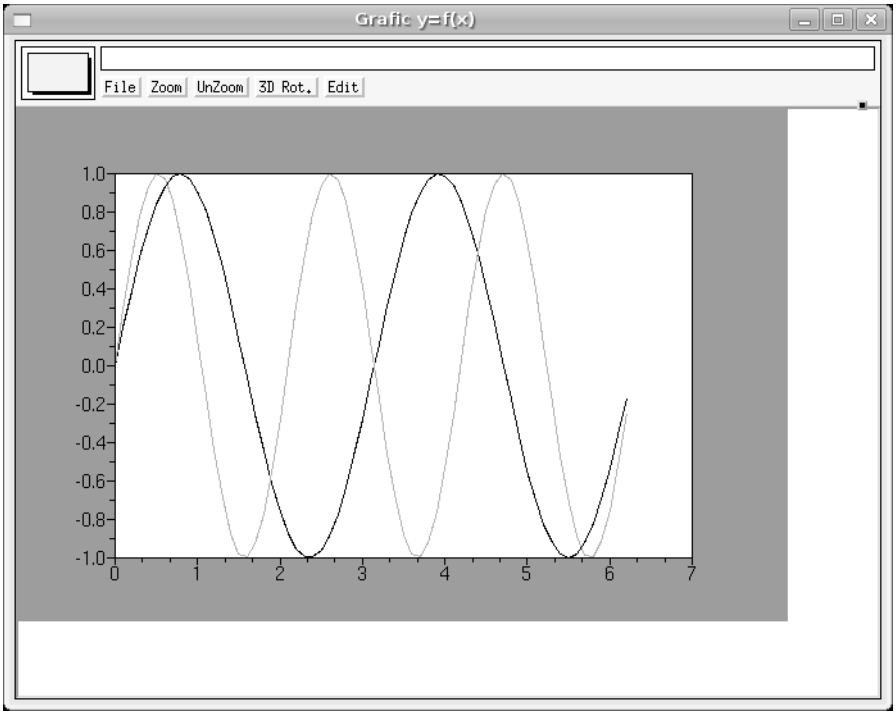

Рис. 4.31. Использование режима *Auto resize*

Pixmap — режим растрового изображения. В выключенном положении (по умолчанию) изображение формируется непосредственно на экране. При включенном режиме (положение переключателя «on») график создается как растровое изображение и направляется в графическое окно командой `show_pixmap()`. Следует отметить, что режим *Pixmap* используется при создании анимированных графиков для сглаживания переходов между кадрами.

Pixel drawing mode — свойство, которое определяет способ формирования изображения на экране. По умолчанию установлен режим «*copy*». В этом случае точно выполняется требуемая операция построения графика. Однако часто необходимо нанести изображение на уже существующее, при этом цвет вновь построенного графика должен четко выделяться. Для этого существует набор режимов: «*clear*», «*and*», «*andReverse*», «*andInverted*», «*noop*», «*xor*», «*or*», «*nor*», «*equiv*», «*invert*», «*orReverse*», «*copyInverted*», «*orInverted*», «*nand*», «*set*».

Результат использования значения «*invert*» режима *Pixel drawing mode* представлен на рис. 4.32.

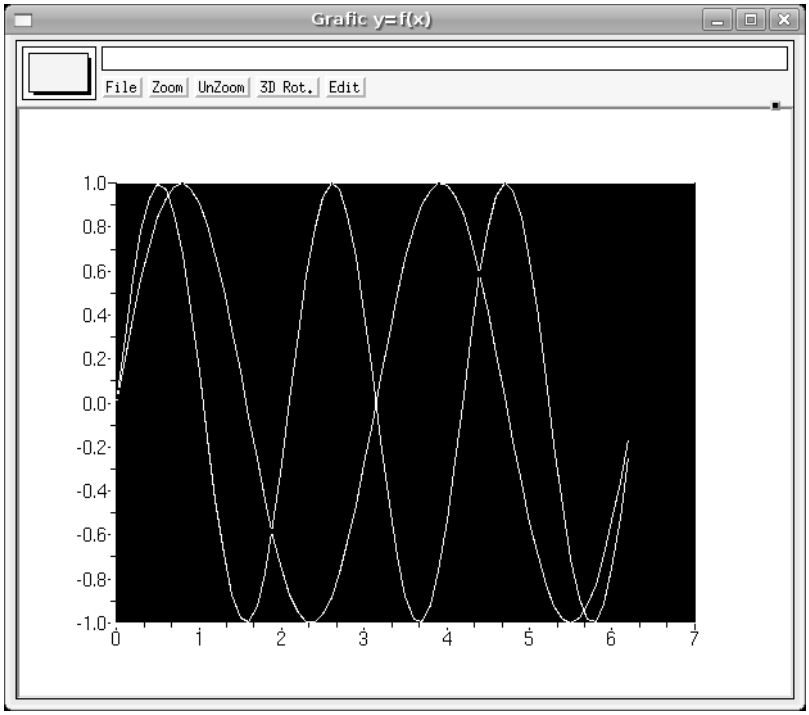

Рис. 4.32. Режим «*invert*» свойства *Pixel drawing mode* на вкладке *Mode*

Rotation style — это свойство применимо лишь к трехмерным графикам. Режим по умолчанию «*unary*» предназначен для вращения выделенных графиков, при включенном режиме «*multiple*» вращаются все трехмерные графики.

4.11.2 Форматирование объекта Axes (Оси графика)

Для изменения свойств объекта *Axes (Оси графика)* необходимо выделить его в поле *Object Browser* окна форматирования. В области *Object Properties* доступные для модификации свойства будут сгруппированы на нескольких закладках. Закладки X, Y и Z идентичны, с той лишь разницей, что позволяют устанавливать желаемый внешний вид соответственно для осей X, Y и Z. Поэтому мы рассмотрим лишь закладку X (см. рис. 4.33).

На закладке X все свойства разделены на две области: *Label Options (Свойства подписей осей)* и *Axis Options (Свойства осей)*. В области *Label Options* можно установить:

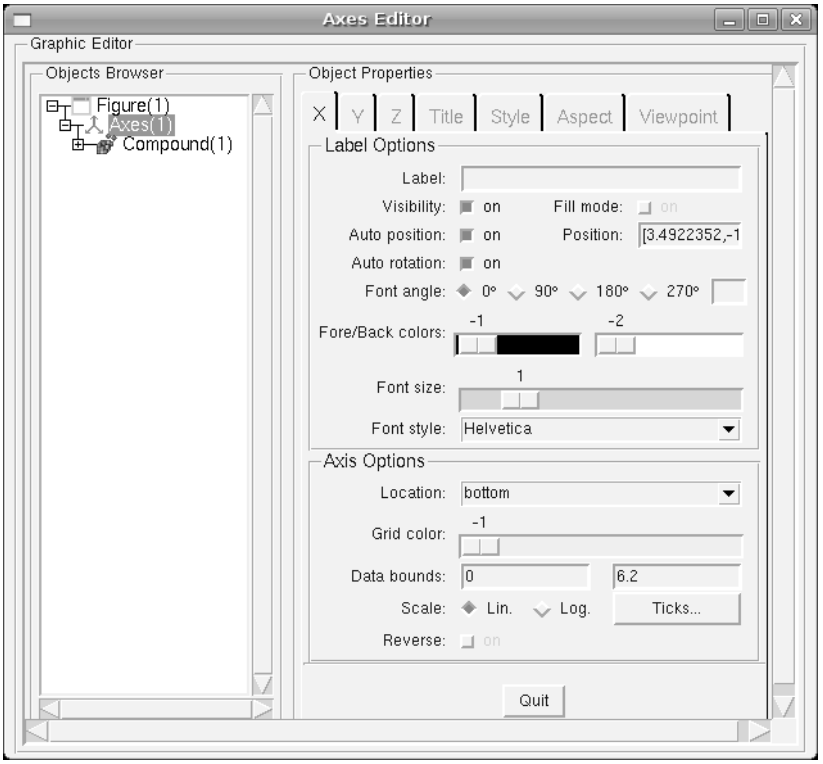

Рис. 4.33. Закладка *X* окна форматирования свойств объекта *Axes*

Label — собственно подпись оси — любая последовательность символов;

Visibility — видимость — переключатель, принимающий значения «on» и «off». По умолчанию оси графика выводятся на экран (положение «on»).

Fill mode — режим заливки — переключатель, принимающий значения «on» и «off» (по умолчанию). Для того, чтобы определить цвет фона вокруг подписи оси, необходимо установить состояние «on».

Auto position — автоматическое определение положения подписи оси графика. По умолчанию установлено значение «on» — подпись выводится внизу, по центру оси. Однако положение подписи можно определить и самостоятельно, для этого в поле *Position* задаются координаты в виде вектора $[x, y]$. При этом переключатель *Auto position* автоматически примет значение «off».

Auto Rotation — режим автоматического вращения подписи оси. По умолчанию этот режим отключен (состояние переключателя «off»).

Font angle — угол поворота подписи оси. Можно установить одно из предлагаемых значений: 0, 90, 180 и 270 градусов, а также любой произвольный угол поворота надписи в последнем поле (см. рис. 4.33).

Fore/Back colors — цвет символов и цвет фона подписи оси соответственно – устанавливаются при помощи ползунка, каждому положению которого отвечает определенный цвет. Всего доступно 35 цветов. Подробнее см. параграф 4.6.

Font size — размер символов подписи оси, возможны значения от 0 до 6. По умолчанию для шрифта установлен размер 1.

Font style — стиль начертания символов подписи оси. По умолчанию установлен стиль *Helvetica* (рубленый).

В области *Axis Options* можно изменить:

Location — расположение оси графика. Для оси X возможны следующие значения этого свойства: *bottom* — снизу, *top* — сверху, *middle* — посередине; а для оси Y: *left* — слева, *right* — справа, *middle* — посередине.

Grid color — цвет линий сетки графика, устанавливаемый с помощью ползунка. В положении -1 линии сетки графика отсутствуют, в положении 0 выводятся черные линии, кроме того доступны еще 32 цвета. Для того, чтобы отображались линии сетки для осей X и Y, необходимо установить свойство *Grid color* и на закладке X, и на закладке Y.

Data bounds — ограничение данных. Для каждой оси можно уменьшить диапазон исходных данных, по которым формируется график, сделав его более детальным.

Scale — масштаб оси графика. Существует два автоматических режима: *lin* (линейный) и *log* (логарифмический). Нажатие на кнопку *Ticks (Засечки)* приводит к появлению окна модификации деления оси *Edit Axes Ticks*. Это окно представлено на рис. 4.34.

С его помощью можно установить следующие свойства засечек координатных осей:

Visibility — отображение — переключатель, принимающий значения «on» и «off». По умолчанию засечки на оси графика выводятся на экран (положение «on»).

Auto ticks — режим автоматического деления оси, по умолчанию также включен (значение переключателя «on»). Однако существует возможность самостоятельно определить шаг, с которым будет разбита ось, его нужно ввести в поле *Steb by* и нажать ***Enter***. При этом переключатель *Auto ticks* автоматически примет значение «off».

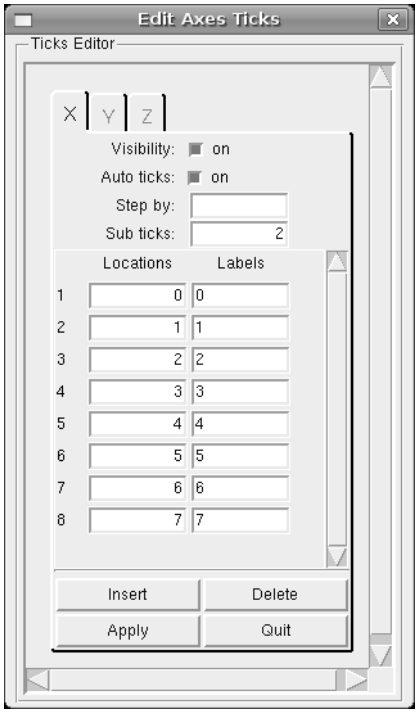

Рис. 4.34. Окно *Edit Axes Ticks*

Sub ticks — промежуточные засечки. В этом поле нужно ввести число засечек, которые будут выводиться между основными делениями оси. Следует отметить, что промежуточные засечки не подписываются.

В окне *Edit Axes Ticks* формируется таблица основных засечек (без засечек *Sub ticks*). Первый столбец *Locations* задает положение засечки, а второй *Labels* — подпись засечки.

Для удобства редактирования таблицы окно снабжено кнопками Insert, Delete, Apply, Quit. Кнопка Insert позволяет вставить в окно готовую таблицу засечек (либо ее фрагмент) посредством буфера обмена. Вставка производится начиная с позиции активной ячейки. Кнопка Delete позволяет удалять не только активную ячейку, но и всю строку, которой она принадлежит. Кнопка Apply подтверждает изменения, а Quit служит для выхода из окна *Edit Axes Ticks*.

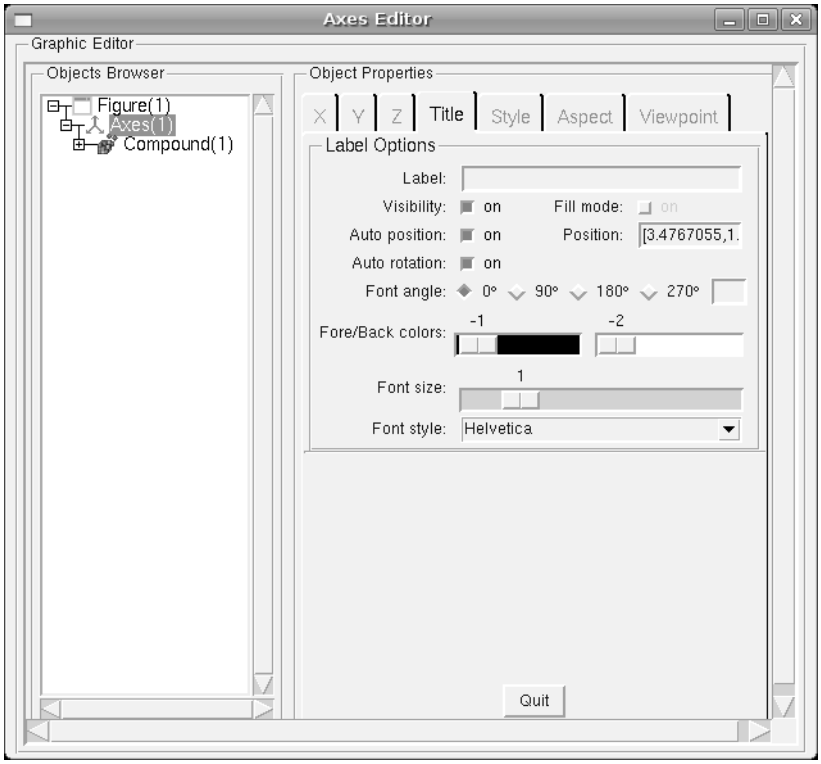

Рис. 4.35. Закладка *Title* окна форматирования *Axes Editor*

Последняя опция на закладке X — это переключатель *Reverse*. Если установить его в положение «on», график зеркально отобразится относительно оси Y. Если же включить этот режим на закладке Y, график будет зеркально отражен относительно оси X.

Закладка *Title* окна форматирования осей *Axes Editor* предназначена для изменения свойств названия графика. Она содержит лишь одну область — *Label Options*, идентичную области *Label Options* закладок *X, Y* и *Z* (см. рис. 4.35).

Проиллюстрируем возможности изменения свойств координатных осей графика с помощью закладок *X, Y, Title* окна форматирования осей *Axes Editor*.

Построим в одних координатных осях графики функций $y_1 = \sin(2x)$ и $y_2 = \sin(3x)$ на интервале $[0; 2\pi]$ с шагом 0,1.

Выведем подписи для оси X «`os absciss`» и для оси Y «`os ordinat`», установим для подписей стиль шрифта *Helvetica Bold* и размер 3. Для обеих осей ползунок *Grid Color* установим в положение 1.

Определим для оси X размещение *middle*, а на закладке Y включим режим *Reverse*. Выведем заголовок графика «`Grafic y=f(x)`», определив стиль шрифта

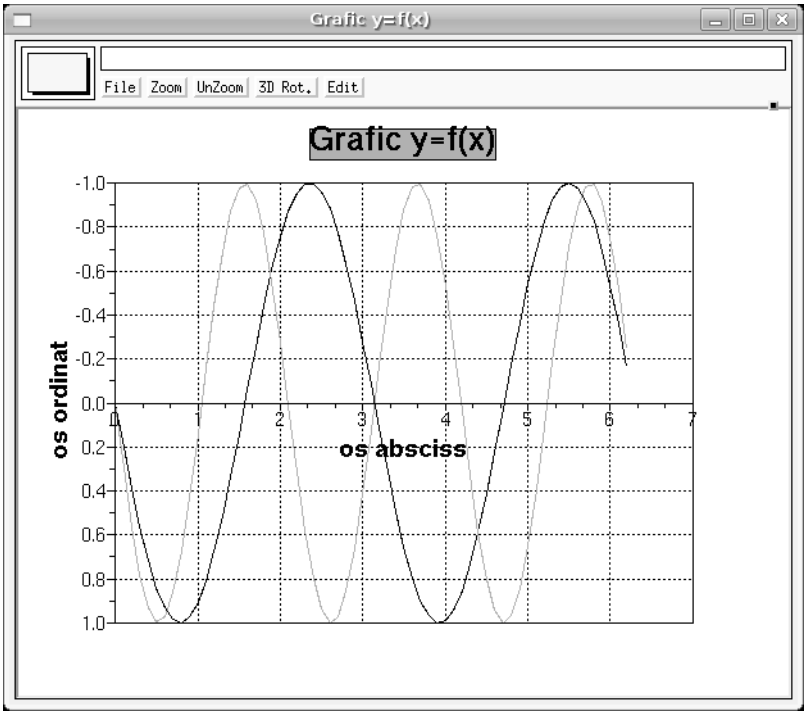

Рис. 4.36. Форматирование осей графика средствами закладок *X, Y, Title*

Helvetica Bold, размер символов 4. Включив режим заливки, установим ее цвет — желтый (положение ползунка 7).

В итоге сформируется графическое окно, представленное на рис. 4.36.

Закладка *Style* окна форматирования осей графика *Axes Editor* (см. рис. 4.37) предоставляет возможность изменять следующие свойства линии оси и подписей засечек:

Visibility — отображение — переключатель, принимающий значения «on» (по умолчанию) и «off». В положении «off» график вообще не выводится в окно.

Font style — стиль начертания символов подписей засечек на оси. По умолчанию установлен стиль *Helvetica*.

Font color — ползунок, каждое положение которого определяет цвет символов подписей засечек. По умолчанию установлен в положении -1 — черный цвет.

Font size — размер символов подписей засечек на оси, возможны значения от 0 до 6. По умолчанию для шрифта установлен размер 1.

Рис. 4.37. Закладка Style окна форматирования осей *Axes Editor*

Fore. color — ползунок, каждое положение которого определяет цвет собственно координатной оси. По умолчанию установлен в положении -1 — черный цвет.

Back. color — ползунок, каждое положение которого определяет цвет заливки фона графика. По умолчанию установлен в положении -2 — белый цвет.

Thickness — толщина линии координатной оси, определяемая ползунком с положениями от 1 до 30. По умолчанию для толщины линии установлено значение 1.

Line style — стиль начертания линии. Возможно 6 режимов: *solid* — сплошная линия, остальные режимы — вариации пунктиров.

В качестве примера отформатируем графики функций предыдущего примера средствами закладки *Style*. Установим стиль шрифта подписи засечек на осях *Helvetica Bold*, размер шрифта — 2, устновим голубой цвет фона графика (положение ползунка *Back. color* 12) и толщину линий координатных осей 2.

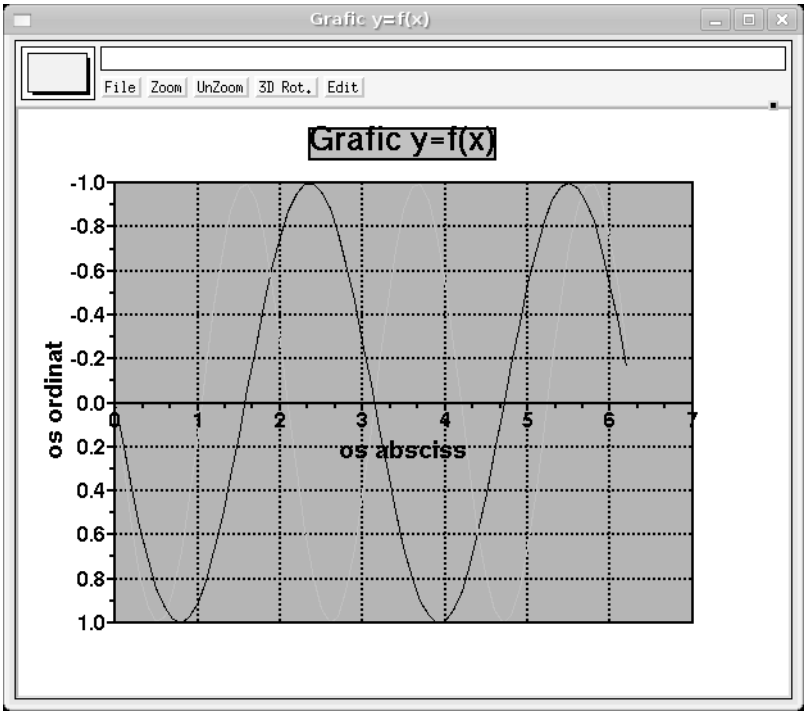

Рис. 4.38. Форматирование осей графика средствами закладки *Style*

На экран будет выведено графическое окно, представленное на рис. 4.38.

Закладка *Aspect* окна форматирования *Axes Editor* (см. рис. 4.39) позволяет изменять следующие свойства:

Auto clear — если переключатель установлен в положение «on», графическое окно будет автоматически очищаться каждый раз перед построением нового графика. Если же этот режим отключен (по умолчанию), графики будут накладываться в одних координатных осях в соответствии с режимом *Auto scale*.

Auto scale — режим обновления границ координатных осей графика. В состоянии переключателя «on» (по умолчанию) новый график изменит границы предыдущего графика, чтобы сформироваться на всем заданном интервале, но в том же масштабе, что и предыдущий график. При отключенном режиме *Auto scale* новый график будет построен в пределах осей предыдущего графика и, возможно, будет отражать лишь часть заданного интервала.

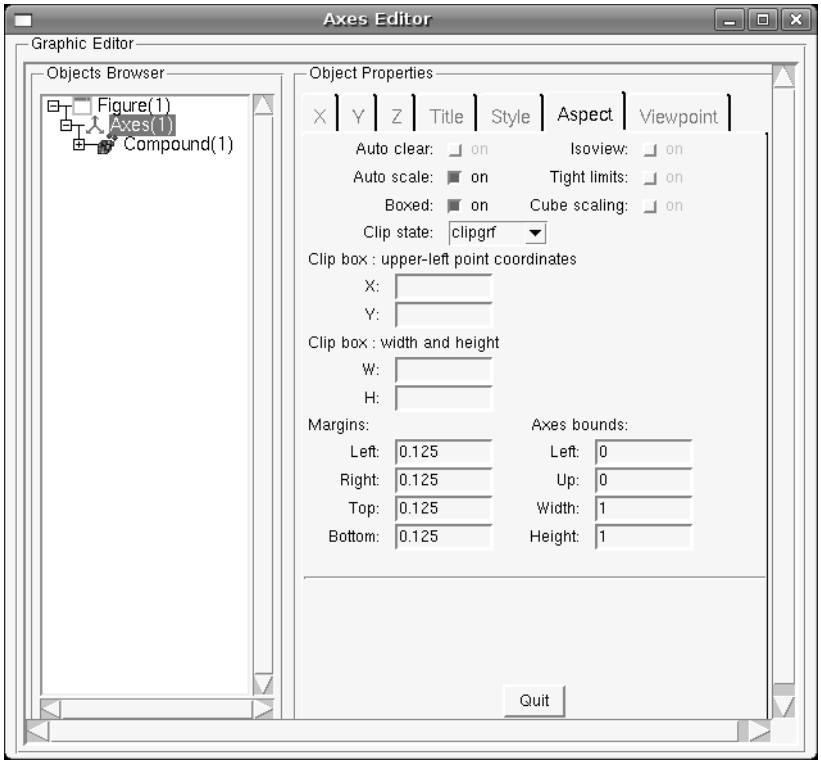

Рис. 4.39. Закладка Aspect окна форматирования осей *Axes Editor*

Воспользуемся предыдущими примерами и проиллюстрируем действие режима *Auto scale*. Построим графики функций $y = \sin(x)$ и $y_1 = \cos(x)$ на интервале $(-2\pi; 2\pi)$, а затем график функции $y_2 = \sin(3x)$ на интервале $(0; 2\pi)$. Обратите внимание, что интервал третьего графика гораздо уже. Поскольку по умолчанию режим *Auto scale* включен, оси будут изменены так, что оба графика сформируются полностью на заданных интервалах (см. листинг 4.25, рис. 4.40).

Листинг 4.25. Пример использования режима автоматического масштабирования координатных осей *Auto scale*

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y');
x=[0:0.1:2*%pi];
plot2d(sin(3*x));
```

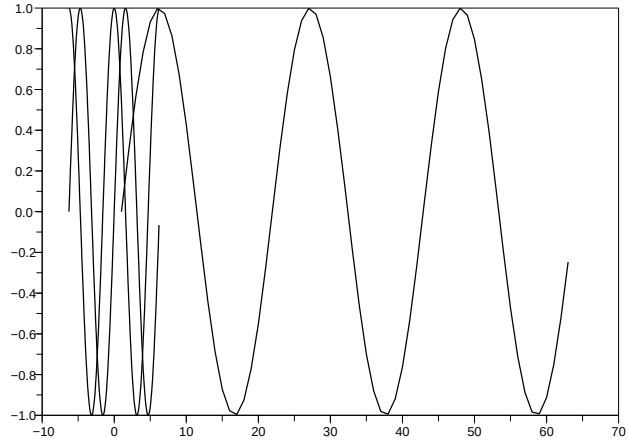

Рис. 4.40. Построение графиков с включенным режимом *Auto scale*

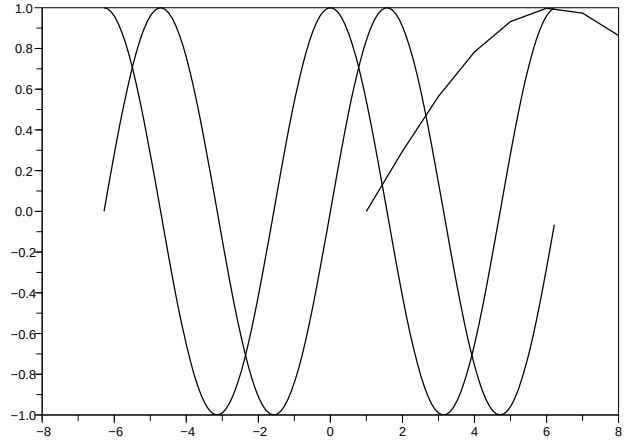

Рис. 4.41. Построение графиков с отключенным режимом *Auto scale*

Теперь построим первые два графика, отключим режим *Auto scale*, а затем построим третий график. Он будет сформирован лишь частично (см. листинг 4.26, рис. 4.41).

Листинг 4.26. Построение графиков нескольких функций в одном графическом окне при выключенном режиме *Auto scale*

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y');
//отключаем режим Auto scale
x=[0:0.1:2*%pi];
plot2d(sin(3*x));
```

Переключатель ***Boxed*** на закладке *Aspect* окна форматирования *Axes Editor* определяет, ограничивать график прямоугольной рамкой (положение «on» по умолчанию) или выводить только координатные оси (положение «off»).

Isoview — это свойство используется для того, чтобы установить одинаковый масштаб для всех осей графика. По умолчанию установлено состояние переключателя «off».

Tight limits — если этот режим включен, оси графика изменяются таким образом, чтобы точно соответствовать значению свойства *Data bounds* закладок *X*, *Y* и *Z*. При значении «off» (по умолчанию) оси могут увеличить исходный интервал, чтобы было проще выбрать масштаб оси и нанести на нее засечки.

Cube scaling — эта опция применима лишь к трехмерным графикам. При состоянии переключателя «on» исходные данные ограничиваются так, чтобы поверхность поместилась в куб размером 1. Это позволяет нагляднее изобразить 3D-график в тех случаях, когда масштаб координатных осей слишком разнится от одной оси к другой. По умолчанию установлено состояние переключателя «off».

Clip state — режим кадрирования (обрезки) графика. Возможно одно из следующих состояний переключателя: «*off*» означает, что создаваемое изображение не кадрируется; «*clipgrf*» (по умолчанию) — от создаваемого изображения обрезается область, находящаяся вне границ осей; «*on*» — от создаваемого изображения обрезается область, находящаяся вне границ, заданных свойством *Clip box*.

Clip box — прямоугольная область, которая будет отображаться после обрезки изображения. Вначале в полях *X* и *Y* задаются координаты верхней левой точки прямоугольника *(upper-left point coordinates)*, затем ширина и высота — поля *W* и *H*.

Margins — это свойство устанавливает расстояние от границы графического окна до области графика: *Left* (левый край), *Right* (правый), *Top* (верхний), *Bottom* (нижний). Значение должно находиться в интервале [0 : 1]. По умолчанию каждому полю присвоено значение 0.125.

Axes bounds — это свойство задает часть графика, которая будет выводиться в координатных осях. *Left* и *Up* определяют положение верхний левый угол, *Width* и *Height* — ширину и высоту фрагмента графика. Значение должно находиться в интервале [0 : 1]. По умолчанию отображаемый фрагмент задается матрицей [0 0 1 1].

Закладка *View Point* окна форматирования осей графика *Axes Editor* (см. рис. 4.42) позволяет установить лишь одно свойство — угол, под которым наблюдатель видит график. По умолчанию установлены значения углов поворота наблюдателя *(Rotation angles)* 0 и 270.

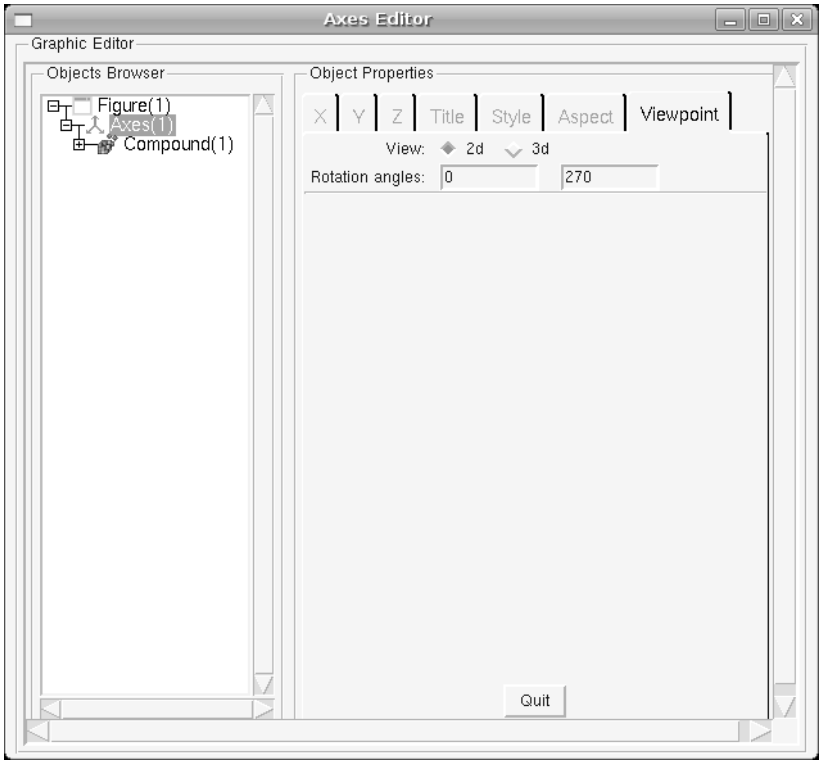

Рис. 4.42. Закладка *View Point* окна форматирования *Axes Editor*

4.11.3 Форматирование объекта Polyline (Линия графика)

Для перехода к форматированию линии графика необходимо выбрать объект *Polyline* в поле просмотра объектов *Object Browser* окна *Polyline Editor*. Доступные для изменения свойства в области *Object Properties* сгруппированы на трех закладках: *Style, Data, Clipping*.

Закладка *Style* окна форматирования *Polyline Editor* (см. рис. 4.43) позволяет установить значения следующих свойств:

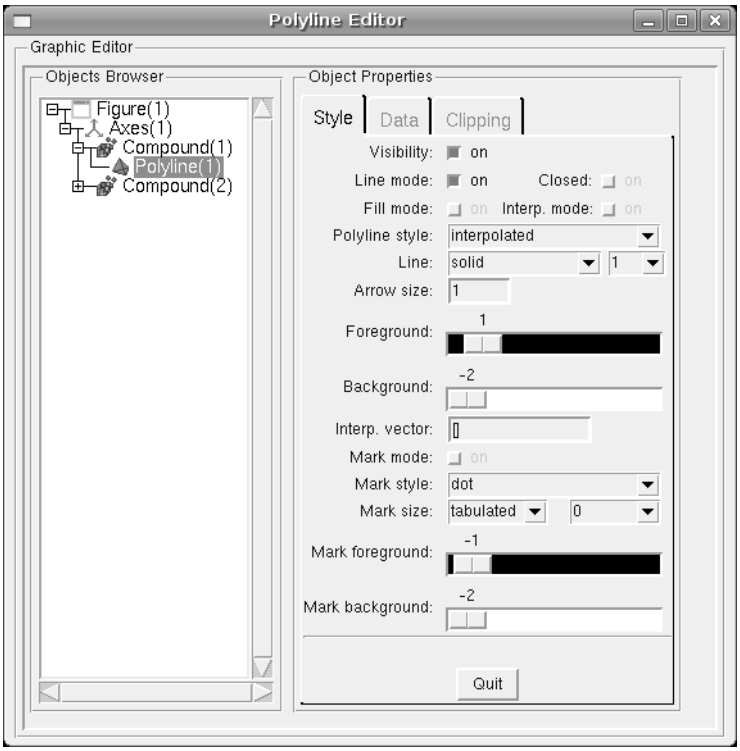

Рис. 4.43. Закладка *Style* окна форматирования *Polyline Editor*

Visibility — отображение — переключатель, принимающий значения «on» (по умолчанию) и «off». В положении «off» линия графика не отображается в окне.

Fill mode — режим заливки — переключатель, принимающий значения «on» и «off» (по умолчанию). Для того, чтобы определить цвет фона области, которую ограничивает кривая, переключатель необходимо установить в состояние «on».

Closed — если включить это свойство, линия графика станет замкнутой.

Polyline style — стиль отображения графика. Возможны следующие значения: *interpolated* — сплошная плавная линия; *staircase* — ступенчатая линия; *barplot* — полосчатые области; *arrowed* — линия, состоящая из последовательности стрелок, размер стрелки можно установить в поле *Arrow size*; *filled* — закрашенные области; *bar* — полосчатые области, ограниченные сплошной плавной линией (см. рис. 4.44).

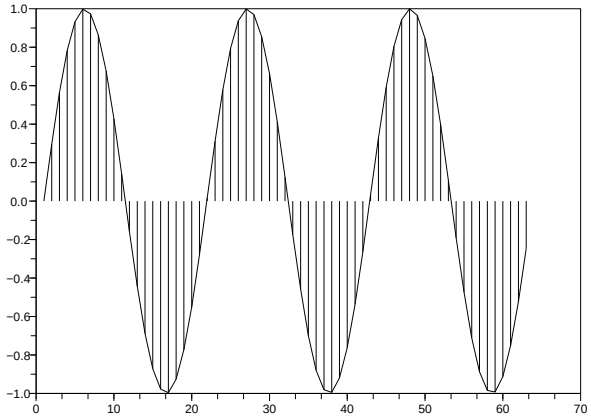

Рис. 4.44. График функции $y = \sin(3x)$. Стиль отображения линии графика — *bar*

Line — стили начертания линии графика. Доступны 6 стилей: *solid* — сплошная, остальные — вариации пунктирной линии. Здесь же из списка можно выбрать желаемую толщину кривой: от 1 до 30.

Foreground и ***Background*** — свойства, устанавливающие соответственно цвет линии графика и заливку области, которая ограничивается кривой, при этом переключатель *Fill mode* должен быть установлен в положение «on».

На рис. 4.45 представлен график функции $y = \sin(3x)$ со значением *Foreground* 1, *Background* 6, включенным переключателем *Fill mode* и режимом *Closed*.

Interp color vector — вектор, определяющий заливку каждого сегмента графика.

Mark mode — режим, позволяющий строить точечные графики (положение переключателя «on»). По умолчанию это свойство отключено.

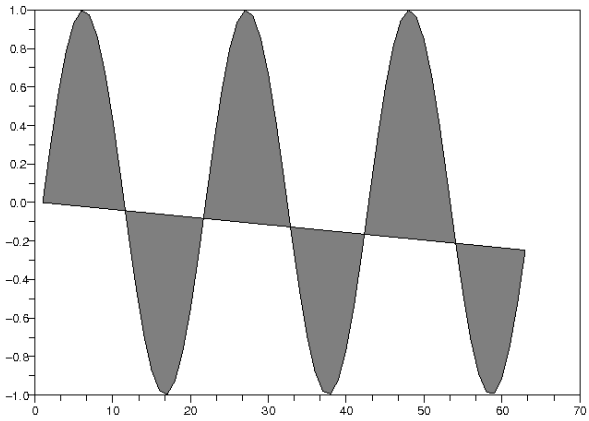

Рис. 4.45. Использование свойств *Foreground*, *Background* и режима *Closed*

Mark style — стиль маркера — возможны следующие значения: *dot* — точка; *plus* — знак «плюс»; *cross* — крестик; *star* — плюс, вписанный в окружность; *filled diamond* — закрашенный ромб; *diamond* — ромб; *triangle up* — треугольник вершиной вверх; *triangle down* — треугольник вершиной вниз; *diamond plus* — плюс, вписанный в ромб; *circle* — кружок; *asterisk* — звездочка; *square* — квадрат; *triangle right* — треугольник вершиной вправо; *triangle left* — треугольник вершиной влево; *pentagram* — пятиконечная звезда.

Mark size — размер маркера — устанавливаемые значения могут изменяться от 0 до 30pt.

Mark foreground — ползунок, каждое положение которого определяет цвет заливки маркера.

На рис. 4.46 представлен точечный график функции $y = \sin(3x)$ с типом маркера *filled diamond*, размер маркера 1, значение цвета заливки маркера *Mark foreground* 5.

Закладка *Data* окна форматирования *Polyline Editor* позволяет уточнить область данных, по которым строится графика. В поле *Data field* первоначально указывается текущий диапазон, в нашем случае это 2 массива типа *Double*, в каждом из них 63 значения — *[63x2 double array]* (см. рис. 4.47). Однако в этом списке можно выбрать строку *Edit data* и отредактировать таблицу исходных данных.

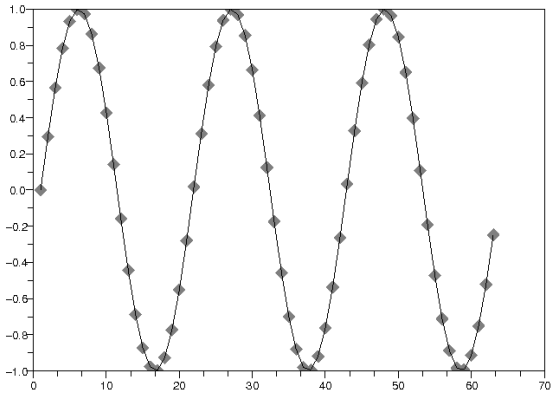

Рис. 4.46. Форматирование точечного графика функции $y = \sin(3x)$

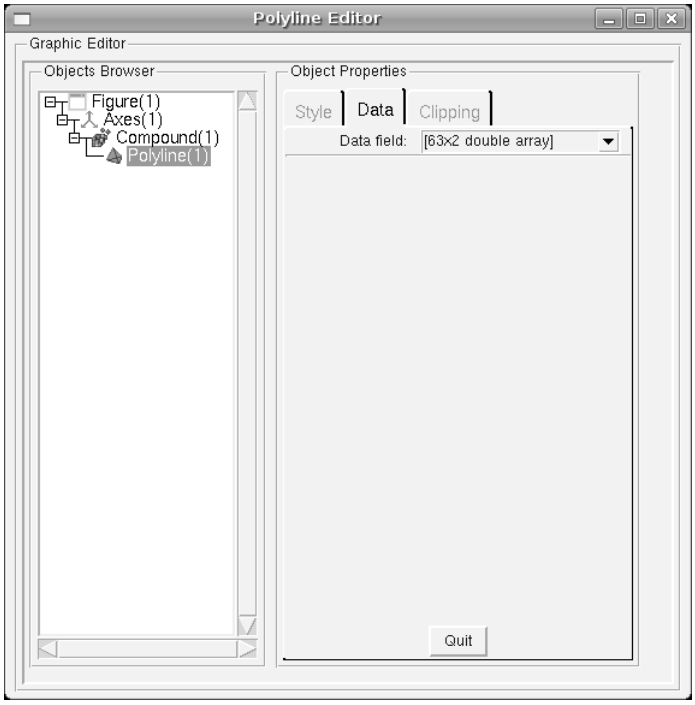

Рис. 4.47. Закладка *Data* окна форматирования *Polyline Editor*

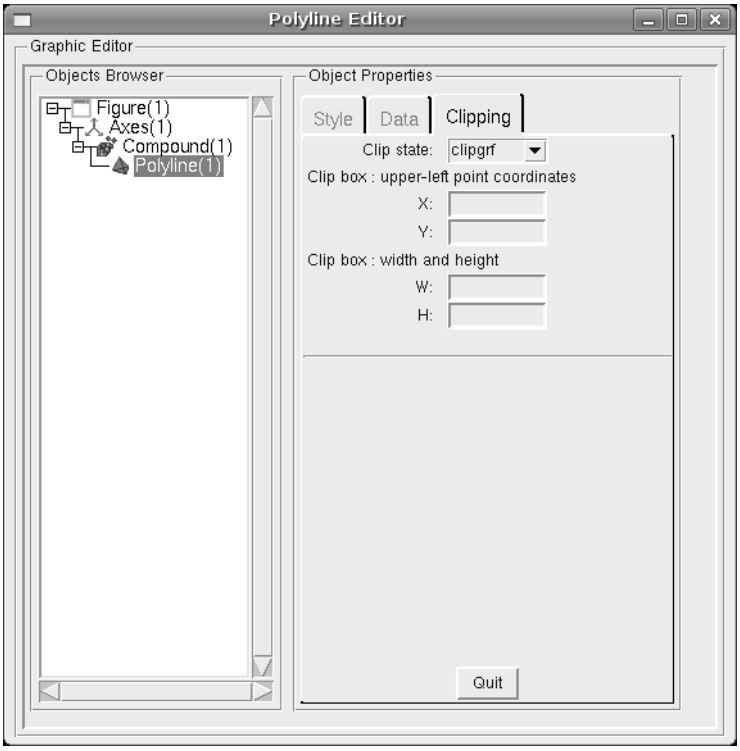

Рис. 4.48. Закладка *Clipping* окна форматирования *Polyline Editor*

Закладка *Clipping (Обрезка)* окна форматирования *Polyline Editor* позволяет установить границы прямоугольной области — *Clip box (Кадр)*, которая останется видимой после обрезки изображения (см. рис. 4.48).

Напомним, что в полях X и Y следует указать x, y координаты верхнего левого угла кадра, а в полях W, H — его ширину и высоту. Режим *Clip state* также может принимать одно из значений: «*off*» — означает, что создаваемая графика не кадрируется; «*clipgrf*» (по умолчанию) — от создаваемой графики обрезается область, находящаяся вне границ осей; «*on*» — от создаваемой графики обрезается область, находящаяся вне границ, заданных свойством *Clip box*.

Глава 5

Построение трехмерных графиков в Scilab

В настоящей главе будут рассмотрены основные возможности Scilab по созданию трехмерных графиков — объемных и пространственных. При этом к трехмерным отнесем все графики, положение каждой точки которых задается тремя величинами.

В целом процесс построения графика функции вида $Z(x, y)$ можно разделить на три этапа:

1. Создание в области построения графика прямоугольной сетки. Для этого формируются прямые линии, параллельные координатным осям x_i и y_j, где
$$x_i = x_0 + ih, \quad h = \frac{x_n - x_0}{n}, \quad i = 0, 1, \ldots, n\,,$$
$$y_j = y_0 + jh, \quad h = \frac{y_k - y_0}{k}, \quad j = 0, 1, \ldots, k\,.$$
2. Вычисление значений функции $z_{ij} = f(x_i, y_j)$ во всех узлах сетки.
3. Обращение к функции построения трехмерных графиков.

5.1 Функции plot3d и plot3d1

В Scilab поверхность можно построить с помощью функций `plot3d` или `plot3d1`. Их отличие состоит в том, что `plot3d` строит поверхность и заливает ее одним цветом, а `plot3d1` — поверхность, каждая ячейка которой имеет цвет, зависящий от значения функции в каждом соответствующем узле сетки (см. рис. I).

Обращение к функциям следующее:

```
plot3d(x,y,z,[theta,alpha,leg,flag,ebox][keyn=valuen]),
plot3d1(x,y,z,[theta,alpha,leg,flag,ebox][keyn=valuen]),
```

здесь `x` — вектор-столбец значений абсцисс;

`y` — вектор-столбец значений ординат;

`z` — матрица значений функции;

`theta, alpha` — действительные числа, которые определяют в градусах сферические координаты угла зрения на график. Попросту говоря, это угол, под которым наблюдатель видит отображаемую поверхность;

`leg` — подписи координатных осей графика — символы, отделяемые знаком `@`. Например, `'X@Y@Z'`.

`flag` — массив, состоящий из трех целочисленных параметров: `[mode,type,box]`. Здесь

`mode` — устанавливает цвет поверхности (см. табл. 5.1). По умолчанию равен 2 — цвет заливки синий, прямоугольная сетка выводится.

Таблица 5.1. Значения параметра `mode`

Значение	**Описание**
> 0	поверхность имеет цвет «mode», выводится прямоугольная сетка
0	выводится прямоугольная сетка, заливка отсутствует (белый цвет)
< 0	поверхность имеет цвет «mode», отсутствует прямоугольная сетка

`type` — позволяет управлять масштабом графика (см. табл. 5.2), по умолчанию имеет значение 2;

Таблица 5.2. Значения параметра `type`

Значение	**Описание**
0	применяется способ масштабирования, как у ранее созданного графика
1	границы графика указываются вручную с помощью параметра `ebox`
2	границы графика определяют исходные данные

`box` — определяет наличие рамки вокруг отображаемого графика (см. табл. 5.3). По умолчанию равен 4.

Таблица 5.3. Значения параметра `box`

Значение	**Описание**
0 и 1	нет рамки
2	только оси, находящиеся за поверхностью
3	выводится рамка и подписи осей
4	выводится рамка, оси и их подписи

`ebox` — определяет границы области, в которую будет выводиться поверхность, как вектор `[xmin, xmax, ymin, ymax, zmin, zmax]`. Этот параметр может использоваться только при значении параметра `type=1`.

`keyn=valuen` — последовательность значений свойств графика
`key1=value1, key2=value2, ..., keyn=valuen`,
таких как толщина линии, ее цвет, цвет заливки фона графического окна, наличие маркера и др. (см. параграф 4.6).

Таким образом, функции `plot3d` (`plot3d1`) в качестве параметров необходимо передать прямоугольную сетку и матрицу значений в узлах сетки.

Задача 5.1.
Построить график функции $Z = \sin(t) \cdot \cos(t)$.

Создадим массив значений аргумента t. Вычислим значения функции и запишем их в массив Z.

Обратите внимание, что при обращении к функции `plot3d` в качестве параметров X и Y, задающих прямоугольную сетку, дважды указан параметр t, поскольку обе функции — и sin, и cos — зависят от одной переменной — t (см. листинг 5.1, рис. 5.1).

Листинг 5.1. Построение графика функции $Z = \sin(t) \cdot \cos(t)$ с помощью функции `plot3d`

```
t=[0:0.3:2*%pi]';
Z=sin(t)*cos(t');
plot3d(t,t,Z);
```

Теперь немного усложним задачу. Построим поверхность, уравнение которой задается двумя независимыми переменными.

Задача 5.2.
Построить график функции $Z = 5y^2 - x^2$.

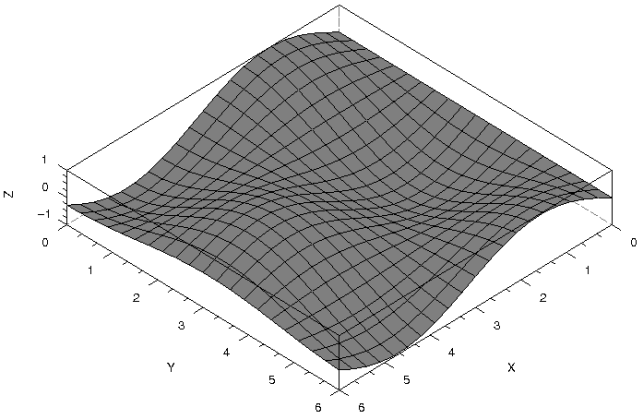

Рис. 5.1. График функции $Z = \sin(t) \cdot \cos(t)$

Прежде всего зададим массивы X и Y.

Затем сформируем матрицу значений функции $Z(x_i, y_j)$, используя оператор цикла `for`. Здесь i — параметр цикла, который будет перебирать все значения массива X, а j — параметр цикла, который будет сопоставлять каждому значению массива X по очереди все значения массива Y.

Таким образом, сначала будут вычислены все значения функции Z при меняющемся Y (от первого до последнего значения в массиве) и первом значении массива X. Затем — при втором значении массива X и т. д.

Напомним, здесь `length` — определяет количество элементов массива X (Y) (см. главу 2).

Наконец, для построения поверхности обратимся к функции `plot3d1` (см. листинг 5.2, рис. II).

Листинг 5.2. Форматирование матрицы значений функции $Z = 5y^2 - x^2$ командой `length` и построение ее графика с помощью функции `plot3d`

```
x=[-2:0.1:2];
y=[-3:0.1:3];
for i=1:length(x)
for j=1:length(y)
z(i,j)=5*y(j)^2-x(i)^2;
end
end
```

```
plot3d1(x',y',z,-125,51);
colorbar(-3,3)
```

Как видно из примера, использование лишь функции `plot3d` для графического изображения показателей, зависящих от двух независимых переменных, достаточно сложно. В Scilab существует несколько команд, призванных облегчить процедуру создания прямоугольной сетки — это `genfac3d` и `eval3dp`.

Простейшей из них по синтаксису является функция `genfac3d`:

```
[xx,yy,zz]=genfac3d(x,y,z)
```

Здесь `xx, yy, zz` — результирующая матрица размером $(4, n-1 \times m-1)$, где `xx(:,i), yy(:,i)` и `zz(:,i)` — координаты каждой из ячеек прямоугольной сетки;

`x` — вектор x-координат размера m;

`y` — вектор y-координат размера n;

`z` — матрица размера (m, n) значений функции $Z(x_i, y_j)$.

Задача 5.3.
Построить график функции $Z = \sin(t) \cdot \cos(t)$.

Определим массив параметра t и вычислим значения функции $Z = \sin(t) \cdot \cos(t)$. Прямоугольную сетку создадим при помощи команды `genfac3d` (см. листинг 5.3).

Листинг 5.3. Создание прямоугольной сетки графика командой `genfac3d` и построение ее графика с помощью функции `plot3d`

```
t=[0:0.3:2*%pi]';
z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d(xx,yy,zz);
```

Для формирования графика обратимся к функции `plot3d` (см. рис. 5.1).

Недостатком команды `genfac3d` является то, что она все-таки не упрощает работу с функцией `plot3d`, если поверхность задается функцией от двух переменных. В таком случае необходимо использовать команду `eval3dp`:

```
[Xf,Yf,Zf]=eval3dp(fun,p1,p2)
```

`Xf,Yf,Zf` — – результирующая матрица размером $(4, n-1 \times m-1)$, где `xx(:,i)`, `yy(:,i)` и `zz(:,i)` — координаты каждой из ячеек прямоугольной сетки;

`fun` — функция, определенная пользователем, которая задает трехмерный график;

`p1` — вектор размера m;

`p2` — вектор размера n.

Проиллюстрируем действие команды `eval3dp` следующим примером.

Задача 5.4.
Построить график, заданный следующими уравнениями: $x = p_1 \cdot \sin(p_1) \cdot \cos(p_2)$, $y = p_1 \cdot \cos(p_1) \cdot \cos(p_2)$, $z = p_1 \cdot \sin(p_2)$.

Прежде всего, определим массивы значений параметров p_1 и p_2. Далее создадим функцию `scp`, которая задает график.

Напомним, что функции в Scilab создаются при помощи команды `deff`:

```
deff([s1,s2,...]=newfunction(e1,e2,...)'
```

где `s1,s2,...` — список выходных параметров, т.е. переменных, которым будет присвоен конечный результат вычислений;

`newfunction` — имя создаваемой функции, оно будет использоваться для ее вызова;

`e1,e2,...` — входные параметры.

Обратите внимание, что команда `deff` записана в три строки только для удобства чтения листинга (см. листинг 5.4).

Листинг 5.4. Создание прямоугольной сетки графика командой `eval3dp` и построение ее графика с помощью функции `plot3d`

```
p1=linspace(0,2*%pi,10);
p2=linspace(0,2*%pi,10);
deff("[x,y,z]=scp(p1,p2)",["x=p1.*sin(p1).*cos(p2)";
"y=p1.*cos(p1).*cos(p2)";
"z=p1.*sin(p2)"]);
[Xf,Yf,Zf]=eval3dp(scp,p1,p2);
plot3d(Xf,Yf,Zf);
```

Теперь сформируем прямоугольную сеть при помощи команды `eval3dp` и построим график, обратившись к функции `plot3d` (см. рис. III).

В Scilab также существуют несколько других функций для построения поверхностей. Они имеют более простой синтаксис Matlab, однако, по мнению авторов, не всегда могут заменить функцию `plot3d`.

5.2 Функции meshgrid, surf и mesh

Для формирования прямоугольной сетки впервые в Scilab 4.0 появилась функция `meshgrid`. Обращение к ней имеет вид:

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

здесь `(x, y [z])` — массивы 2 (3) исходных параметров X, Y (Z), указываемые через запятую;

`[X, Y [Z]]` — матрицы в случае 2 и массивы в случае 3 входных величин.

После формирования сетки вывести в нее график можно с помощью функции `surf` либо `mesh`. Так же, как и в случае с функциями `plot3d` и `plot3d1`, `surf` строит поверхность, заливая каждую ячейку цветом, который зависит от конкретного значения функции в узле сетки, а `mesh` заливает ее одним цветом.

Таким образом, `mesh` является полным аналогом функции `surf` со значением параметров `Color mode=индекс белого цвета` в текущей палитре цветов и `Color flag=0`.

Обращение к функциям имеет вид:

```
surf([X,Y],Z,[color,keyn=valuen])
mesh([X,Y],Z,[color,])
```

здесь `X,Y` — массивы, задающие прямоугольную сетку;

`Z` — матрица значений функции;

`color` — матрица действительных чисел, устанавливающих цвет для каждого узла сети;

`keyn=valuen` — последовательность значений свойств графика
`key1=value1, key2=value2, ..., keyn=valuen`,
определяющих его внешний вид (см. параграф 4.6).

Конечно, в том случае, если прямоугольная сетка была построена командой `meshgrid`, необходимости указывать параметры `X,Y` нет. В самом простейшем случае к функции `surf` можно обратиться так — `surf(z)`.

Задача 5.5.
Построить график функции $Z = 5y^2 - x^2$ с помощью команды `mesh`.

С помощью команды `meshgrid` создадим прямоугольную сетку. Здесь $-2:2$ определяет положение прямых, параллельных оси X, а $-3:3$ — оси Y.

После формирования сетки вычислим значения функции Z во всех узлах и обратимся к функции `mesh` для построения графика (см. листинг 5.5, рис. 5.2).

Листинг 5.5. Создание прямоугольной сетки графика командой `meshgrid` и построение ее графика с помощью функции `mesh`

```
[x y]=meshgrid(-2:2,-3:3);
z=5*y.^2-x.^2;
mesh(x,y,z);
```

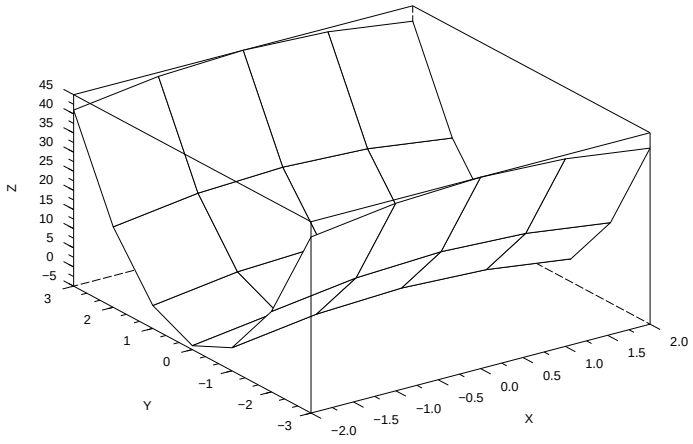

Рис. 5.2. График функции $Z = 5y^2 - x^2$, построенный командой `mesh`

Как видно из рисунка 5.2, сетка, построенная с шагом 1, слишком редкая, а вычисленных значений функции в узлах недостаточно для изображения плавного графика. Поэтому зачастую лучше самостоятельно указывать шаг формирования прямоугольной сетки при вызове команды `meshgrid`.

Задача 5.6.
Построить график функции $Z = 5y^2 - x^2$ с помощью команды `surf`.

Создадим с помощью команды `meshgrid` прямоугольную сетку, указывая шаг, через который будут построены параллельные координатным осям линии — 0,1 для обеих осей. В этом случае сетка будет плотнее, а график более плавным, чем в предыдущем примере.

Далее вычисляем значения функции Z и вызываем `surf` для построения поверхности (см. листинг 5.6, рис. IV).

Листинг 5.6. Создание прямоугольной сетки графика командой `meshgrid` и построение ее графика с помощью функции `surf`

```
[x y]=meshgrid(-2:0.1:2,-3:0.1:3);
z=5*y.^2-x.^2;
surf(x,y,z)
```

На первый взгляд может показаться, что для функции $Z = 5y^2 - x^2$ `plot3d1` и `surf` построили разные поверхности (см. рис. II и рис. IV). Однако это не так. Различие обусловлено использованием по умолчанию функцией `surf` режима масштабирования `Cube scaling` (см. рис. 5.3). Если его отключить, `surf` выведет изображение, идентичное построенному с помощью функции `plot3d` (см. рис. V).

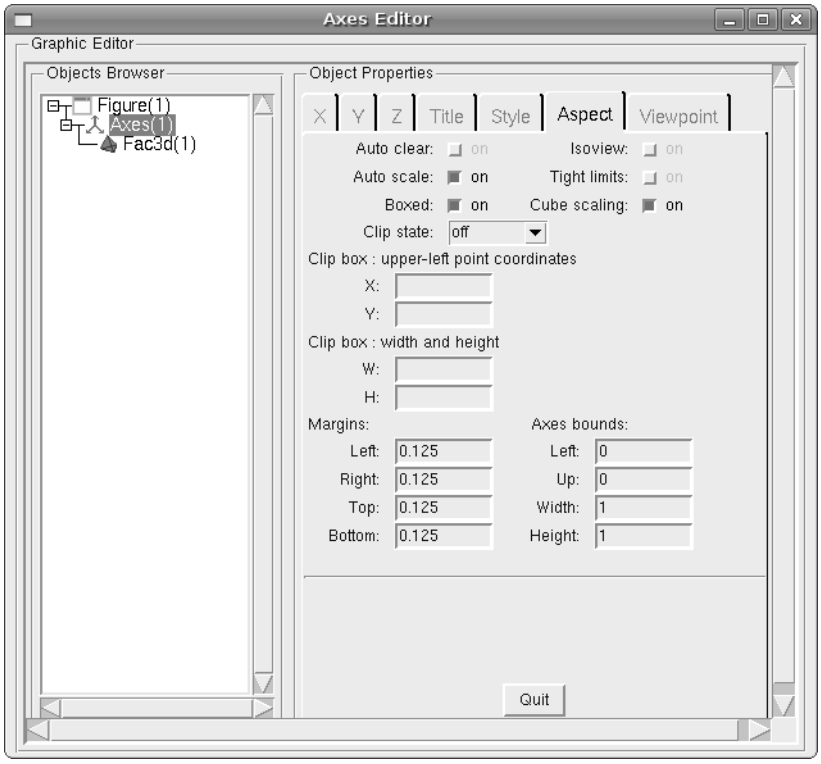

Рис. 5.3. Режим *Cube scaling* окна форматирования *Axes Editor*

В Scilab можно построить графики двух поверхностей в одной системе координат, для этого, как и для двумерных графиков, следует использовать команду `mtlb_hold('on')`, которая блокирует создание нового графического окна при выполнении команд `surf` или `mesh`.

Задача 5.7.
Построить график функции

$$\begin{cases} z(x,y) = (3x^2 + 4y^2) - 1\,, \\ z_1(x,y) = -(3x^2 + 4y^2) - 1\,. \end{cases}$$

Сформируем плотную прямоугольную сетку с помощью команды `meshgrid`.

Пусть $z(x,y) = (3x^2+4y^2)-1$ и $z_1(x,y) = -(3x^2+4y^2)-1$. Вычислим значения функций во всех узлах сетки.

Поверхность $Z = +(3x^2+4y^2)-1$ построим с помощью команды `surf`, каждая ее ячейка будет залита цветом, зависящим от значения функции в узле сетки. Далее вызовем команду `mtlb_hold('on'),` которая заблокирует создание нового графического окна, и с помощью команды `mesh` построим поверхность $Z1 = -(3x^2+4y^2)-1$ в одних координатных осях с $Z = +(3x^2+4y^2)-1$, при этом будет выведена прямоугольная сетка, а ячейки залиты белым цветом (см. листинг 5.7, рис. VI).

Листинг 5.7. Построение графиков нескольких функций в одном графическом окне с помощью команды `mtlb_hold('on')`

```
[x y]=meshgrid(-2:0.2:2,-2:0.2:2);
z=3*x.^2+4*y.^2-1;
z1=-3*x.^2-4*y.^2-1;
surf(x,y,z);
mtlb_hold('on');
mesh(x,y,z1);
```

5.3 Функции plot3d2 и plot3d3

Функции `plot3d2` и `plot3d3` являются аналогами функции `plot3d`, поэтому имеют такой же синтаксис:

```
plot3d2(x,y,z,[theta,alpha,leg,flag,ebox][keyn=valuen]),
plot3d3(x,y,z,[theta,alpha,leg,flag,ebox][keyn=valuen])
```

Эти функции предназначены для построения поверхности, которая задается набором граней. Т. е. если функция `plot3d` по входным данным сможет построить лишь отдельно стоящие друг от друга плоские грани, то `plot3d2 (plot3d3)` проинтерпретирует взаимное расположение этих граней в виде цельного геометрического тела.

Отличие функций `plot3d2` и `plot3d3` сходно с различием действия функций `plot3d` и `plot3d1`, а также `surf` и `mesh`. `Plot3d2` строит поверхность, при этом выводит сетку и заливает все ячейки одним из цветов, по умолчанию — синим.

`Plot3d` также выводит сетку, однако оставляет все ячейки без заливки (т. е. белыми)

Задача 5.8.
Построить сферу

$$\begin{cases} x(u,v) = \cos(u)\cos(v)\,, \\ y(u,v) = \cos(u)\sin(v)\,, \\ z(u,v) = \sin(u) \end{cases}$$

при помощи функции `plot3d2`.

При построении графиков поверхностей, заданных параметрически — $x(u,v)$, $y(u,v)$ и $z(u,v)$ — необходимо сформировать матрицы X, Y и Z одинакового размера. Для этого массивы u и v должны иметь одинаковый размер. После этого следует выделить два основных вида представления x, y и z в случае параметрического задания поверхностей:

1. Если x, y и z можно представить в виде $f(u) \cdot g(v)$, то соответствующие им матрицы X, Y и Z следует формировать в виде матричного умножения $f(u)$ на $g(v)$.

2. Если x, y и z можно представит в виде $f(u)$ или $g(v)$, то в этом случае матрицы X, Y и Z следует записывать в виде $f(u) \cdot \mathrm{ones}(\mathrm{size}(v))$ или $g(v) \cdot \mathrm{ones}(\mathrm{size}(u))$ соответственно.

Отметим также, что здесь `linspace` — функция, возвращающая массив с линейным приращением значений в заданном диапазоне. Например, `u=linspace(-%pi/2,%pi/2,40)` значит, что параметр `u` линейно изменяется в диапазоне $[-2\pi; 2\pi]$. Число 40 устанавливает, что массив должен содержать ровно 40 значений, по умолчанию их 100 (см. листинг 5.8).

Листинг 5.8. Построение сферы с помощью функции `plot3d2`

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d2(X,Y,Z);
```

Построенная функцией `plot3d2` сфера представлена на рис. 5.4.
Теперь посмотрим, как эту же задачу выполнит функция plot3d.

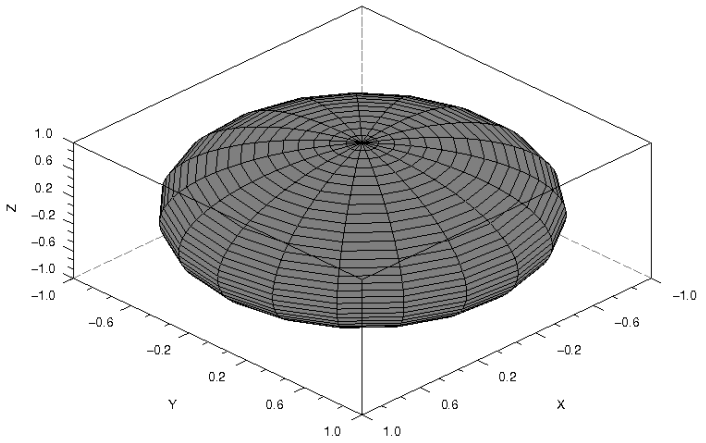

Рис. 5.4. График сферы, построенный функцией `plot3d2`

Задача 5.9.
Построить сферу

$$\begin{cases} x(u,v) = \cos(u)\cos(v)\,, \\ y(u,v) = \cos(u)\sin(v)\,, \\ z(u,v) = \sin(u) \end{cases}$$

с помощью функции `plot3d`.

Определим параметры u и v, вычислим значения функций x, y, z, как и в предыдущем примере (см. листинг 5.9). Однако для построения графика обратимся к функции `plot3d`. Получим следующее изображение (см. рис. VII).

Листинг 5.9. Построение сферы с помощью функции `plot3d`

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d(X,Y,Z);
```

Проиллюстрируем действие функции `plot3d3` на этом же примере.

Задача 5.10.
Построить сферу

$$\begin{cases} x(u,v) = \cos(u)\cos(v)\,, \\ y(u,v) = \cos(u)\sin(v)\,, \\ z(u,v) = \sin(u) \end{cases}$$

при помощи функции `plot3d3`.

Определим диапазоны изменения параметров u и v, как и предыдущих примерах, лишь уменьшив количество значений для массива u с 40 до 20 — так график будет выглядить менее перегруженным.

Вычислим значения функций x, y, z и для изображения графика обратимся к функции `plot3d3` (см. листинг 5.10). Обратите внимание, что заливка ячеек полученной сферы отсутствует (см. рис. 5.5).

Листинг 5.10. Построение сферы с помощью функции `plot3d3`

```
u = linspace(-%pi/2,%pi/2,20);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d3(X,Y,Z);
```

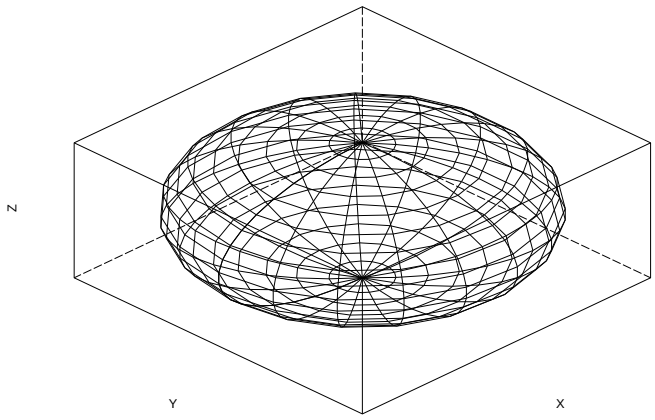

Рис. 5.5. График сферы, построенный функцией `plot3d3`

5.4 Функции param3d и param3d1

Для построения параметрической кривой в Scilab существует команда `param3d`:

```
param3d(x,y,z,[theta,alpha,leg,flag,ebox]).
```

Проиллюстрируем возможности функции `param3d` следующими примерами.

Задача 5.11.
Построить график линии, заданной параметрически:

$$\begin{cases} y = sin(t)\,, \\ y_1 = cos(t)\,, \\ y_2 = \dfrac{t}{7}\,. \end{cases}$$

Прежде всего, определим диапазон и шаг изменения параметра t.

Затем обратимся к функции `param3d`, передав ей математические выражения функций y, y_1 и y_2, а также углы в градусах, под которыми наблюдатель будет видеть формируемый график — 45 и 35 (см. листинг 5.11, рис. 5.6).

Листинг 5.11. Построение линии, заданной параметрически, с помощью функции `param3d`

```
t=[0:0.1:10*%pi];
param3d(sin(t),cos(t),t/7,45,35);
```

Задача 5.12.
Построить линию, заданную параметрически:

$$\begin{cases} x = t \cdot \sin(t)\,; \\ y = t \cdot \cos(t)\,; \\ z = \dfrac{t \cdot |t|}{(50 \cdot \pi)}\,. \end{cases}$$

Определив массив значений параметра t, вычислим значения X, Y и Z координат кривой.

Для построения графика используем команду `param3d`, установив углы обозрения наблюдателя 45 и 60 градусов. (см. листинг 5.12, рис. 5.7).

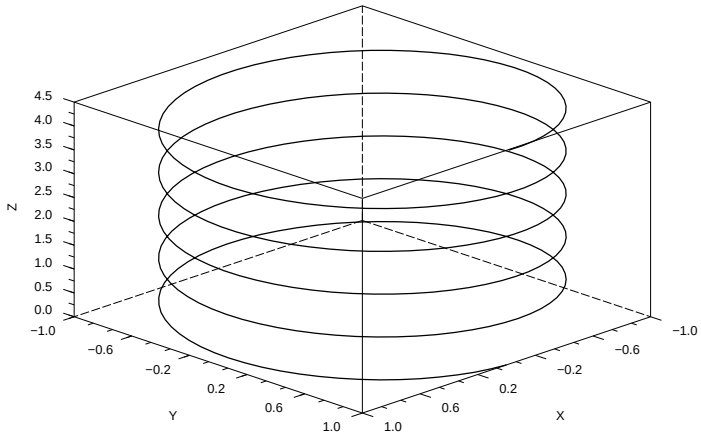

Рис. 5.6. График параметрической линии, построенный функцией `param3d`

Листинг 5.12. Построение линии, заданной параметрически, с помощью функции `param3d`

```
t=-50*%pi:0.1:50*%pi;
x=t.*sin(t);
y=t.*cos(t);
z=t.*abs(t)/(50*%pi);
param3d(x,y,z,45,60);
```

Для вывода нескольких параметрически заданных кривых в одних координатах в Scilab используется функция `param3d1`. Она имеет несколько отличный синтаксис:

```
param3d1(x,y,list(z,colors),[theta,alpha,leg,flag,ebox])
```

Здесь впервые появляется необходимость использования конструкции `list(z,colors)`, которая позволяет не только задавать Z-координату для каждой из кривых, но и устанавливать для них желаемый цвет. Рассмотрим это на примере.

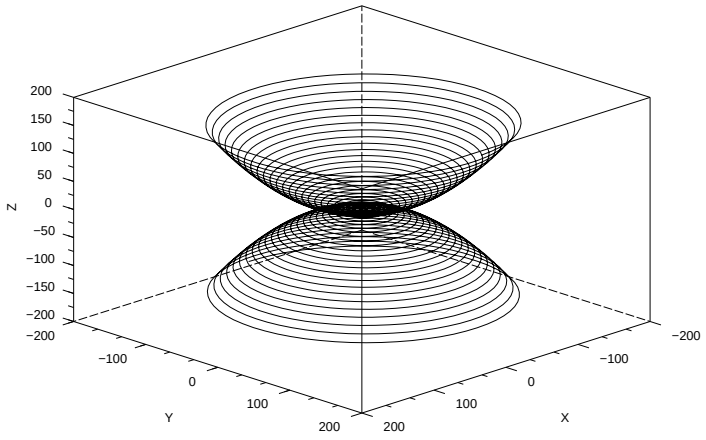

Рис. 5.7. График параметрической линии, построенный функцией `param3d`

Задача 5.13.
Построить графики линий, заданных параметрически:

$$\begin{cases} x = \sin(t)\,; \\ y = \sin(2t)\,; \\ z = t/10\,. \end{cases} \quad \text{и} \quad \begin{cases} x = \cos(t)\,; \\ y = \cos(2t)\,; \\ z = \sin(t)\,. \end{cases}$$

Зададим массив значений параметра t.

Для построения графиков линий в одной системе координат обратимся к функции `param3d1`. В качестве параметров в первых квадратных скобках передадим ей X и Y координаты первой кривой, а во вторых — второй. При помощи свойства `list` определяем Z-координаты и для первой кривой установим темно-синий цвет линии (9), а для второй — красный (5). Числа 35 и 45 — углы обозрения наблюдателя. Параметр `'X@Y@Z'` отвечает за вывод подписей осей графика (см. листинг 5.13, рис. VIII).

Листинг 5.13. Построение нескольких параметрически заданных линий с помощью команды `list` функции `param3d`

```
t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],...
list([t/10,sin(t)],[9,5]),35,45,"X@Y@Z");
```

Таблица 5.4. Значения параметра `mode`

Значение	Описание
0	Изолинии наносятся на поверхность $Z(x, y)$
1	Изолинии наносятся на поверхность и план, который задается уравнением $Z = z_{lev}$
2	Изолинии наносятся на двумерный график

5.5 Функция contour

В Scilab, кроме построения объемных графиков, также реализована возможность создания пространственных моделей объектов. На практике часто возникает необходимость построения карт в изолиниях значений показателя, где X, Y-координаты задают положение конкретной изучаемой точки на плоскости, а Z-координата — зафиксированную величину показателя в этой точке. Точки с одинаковыми значениями показателя соединяют так называемые изолинии — линии одинаковых уровней значений исследуемой величины.

Для построения изолиний в Scilab существует функция `contour`. Обращение к ней имеет вид:

```
contour(x,y,z,nz[theta,alpha,leg,flag,ebox,zlev])
```

Здесь `x, y` — массивы действительных чисел;

`z` — матрица действительных чисел – значения функции, описывающей поверхность $Z(x, y)$;

`nz` — параметр, который устанавливает количество изолиний. Если `nz` — целое число, то в диапазоне между минимальным и максимальным значениями функции $Z(x, y)$ через равные интервалы будут проведены `nz` изолиний. Если же задать `nz` как массив, то изолинии будут проводиться через все указанные в массиве значения;

`theta, alpha` — действительные числа, которые определяют в градусах сферические координаты угла обозрения наблюдателя. Попросту говоря, это угол, под которым наблюдатель видит отображаемую поверхность;

`leg` — подписи координатных осей графика — символы, отделяемые знаком `@`. Например, `'X@Y@Z'`.

`flag` — массив, состоящий из трех целочисленных параметров: `[mode,type,box]`. Здесь

`mode` — устанавливает способ и место нанесения линий уровня (см. табл. 5.4).

`type` — позволяет управлять масштабом графика (см. табл. 5.2), по умолчанию имеет значение 2;

`box` — определяет наличие рамки вокруг отображаемого графика (см. табл. 5.3). По умолчанию равен 4;

`ebox` — определяет границы области, в которую будет выводиться поверхность, как вектор `[xmin,xmax,ymin,ymax,zmin,zmax]`. Этот параметр может использоваться только при значении параметра `type=1`;

`zlev` — математическое выражение, которое задает план (горизонтальную проекцию заданной поверхности) для построения изолиний. По умолчанию совпадает с уравнением, описывающим плоскость, — в этом случае может не указываться.

Следует отметить, что функции `contour` уравнение поверхности $Z(x, y)$ удобнее передавать в качестве параметра как функцию, определенную пользователем.

Напомним, что функции в Scilab создаются при помощи команды `deff`:

```
deff('[s1,s2,...]=newfunction(e1,e2,...)
```

где `s1,s2,...` — список выходных параметров, т.е. переменных, которым будет присвоен конечный результат вычислений;

`newfunction` — имя создаваемой функции, оно будет использоваться для ее вызова;

`e1,e2,...` — входные параметры.

Второй способ создания функции - это применение конструкции вида:

```
function <lhs_arguments>=<function_name><rhs_arguments>
<тело_функции>
endfunction
```

где `lhs_arguments` — список выходных параметров;

`function_name` — имя создаваемой функции;

`rhs_arguments` — входные параметры.

Задача 5.14.
Построить линии уровня поверхности $Z = x \cdot \sin(x)^2 \cdot \cos(y)$.

Введем параметр t и определим массив его значений. При помощи команды function создадим функцию `my_surface` с входными данными x, y и выходными — z. В теле функции вычислим значения математического выражения, задающего поверхность.

Для построения изолиний обратимся к функции `contour` (см. листинг 5.14, рис. IX).

Листинг 5.14. Построение изолиний поверхности $Z = x \cdot \sin(x)^2 \cdot \cos(y)$ с помощью функции `contour`

```
t=linspace(-%pi,%pi,30);
function z=my_surface(x,y)
z=x*sin(x)^2*cos(y)
endfunction
contour(t,t,my_surface,10);
```

Этот пример показывает, что выполнение функции `contour` приводит к формированию линий одинаковых значений показателя и проецированию их на горизонтальную плоскость. Очевидно, что такое представление данных малоинформативно. Гораздо наглядней изображение изолиний поверхности и собственно поверхности в одном графическом окне.

Задача 5.15.
Построить поверхность $Z = \sin(x) \cdot \cos(y)$ и вывести изолинии в одном графическом окне.

Прежде всего, введем параметр t и сформируем массив его значений.

Создадим функцию `Surf`, обратившись к команде `deff`.

С помощью команды `rect` установим границы области построения графика в графическом окне для того, чтобы стало возможным совместить и поверхность, и спроецированные на горизонтальную плоскость изолинии поверхности.

Напомним, что при построении графика функции вида $Z(x, y)$ при помощи функции `plot3d` необходимо использовать оператор цикла `For`, формировать матрицу значений функции $z_{ij} = f(x_i, y_j)$. Чтобы избежать этого, воспользуемся командой `feval`.

Далее с помощью функции `plot3d` строим график поверхности $Z = \sin(x) \cdot \cos(y)$, устанавливая углы обзора наблюдателя, подписи для координатных осей. Определяем и массив `flag` [2,1,4]: 2 — цвет графика — синий, 1 — границы области построения графика определяются вручную (далее указан параметр `rect`, заданный выше), 4 — выводятся все оси и рамка вокруг графика.

Затем формируем изолинии, обратившись к функции `contour`, также устанавливаем углы обзора наблюдателя, подписи координатных осей, число формируемых изолиний 10 и значения массива `flag` [1,1,4]: 1 — режим вывода изолиний на отдельно построенный план, который задается тем же уравнением, что и поверхность ($Z = \sin(x) \cdot \cos(y)$), 1 — границы области построения графики

определяются вручную (далее указан параметр `rect`, заданный выше), 4 — выводятся все оси и рамка вокруг графика. Число −5 устанавливает положение горизонтальной плоскости с изолиниями — 5 единиц ниже графика поверхности.

С помощью команды `xtitle` выведем подпись для графика (см. листинг 5.15, рис. X).

Листинг 5.15. Построение поверхности $Z = \sin(x) \cdot \cos(y)$ (функция `plot3d`) и вывод ее изолиний (функция `contour`) в одном графическом окне командой `rect`

```
t=%pi*(-10:10)/10;
deff('[z]=Surf(x,y)','z=sin(x)*cos(y)');
rect=[-%pi,%pi,-%pi,%pi,-5,1];
z=feval(t,t,Surf);
plot3d(t,t,z,35,45,'X@Y@Z',[2,1,4],rect);
contour(t,t,z,10,35,45,'X@Y@Z',[1,1,4],rect, -5);
xtitle('plot3d and contour');
```

Однако и такое изображение поверхности и ее изолиний не всегда бывает удобным. Попробуем совместить график поверхности и ее линии уровня.

Задача 5.16.
Совместить график поверхности $Z = \sin(x) \cdot \cos(y)$ с ее изолиниями.

Как и в предыдущем примере, зададим массив значений параметра t, создадим функцию `Surf`, ограничим область для вывода графика внутри графического окна с помощью команды `rect`, а также вычислим значения функции $Z = \sin(x) \cdot \cos(y)$, выполнив команду `feval`.

При построении поверхности оставим все параметры без изменений, кроме углов обзора наблюдателя (установим 75 и 45) а также цвета заливки графика (установим значение параметра `mode` в массиве `flag` равным −19 — коричневый цвет).

При обращении к функции `contour` для совмещения поверхности и ее изолиний удалим значение параметра `location` −5 и установим для режима `mode` в массиве `flag` значение 0 — изолинии наносятся непосредственно на поверхность $Z = \sin(x) \cdot \cos(y)$.

С помощью команды `xtitle` также выводим подпись для графика (см. листинг 5.16, рис. XI).

Листинг 5.16. Пример срвмещения графика поверхности с ее изолиниями

```
t=%pi*(-10:10)/10; deff('[z]=Surf(x,y)','z=sin(x)*cos(y)');
rect=[-%pi,%pi,-%pi,%pi,-1,1];
z=feval(t,t,Surf);
```

```
plot3d(t,t,z,35,45,'X@Y@Z',[-19,1,4],rect);
contour(t,t,z+0.1,10,35,45,'X@Y@Z',[0,1,4],rect); ...
xtitle('plot3d and contour');
```

5.6 Функция contourf

В Scilab существует функция `contourf`, которая не просто изображает поверхность на горизонтальной плоскости в виде изолиний, но и заливает интервалы между ними цветом, в зависимости от конкретного уровня значений показателя.

Обращение к функции имеет вид:

```
contourf (x,y,z,nz,[style,strf,leg,rect,nax])
```

Здесь `x, y` — массивы действительных чисел;

`z` — матрица действительных чисел — значения функции, описывающей поверхность $Z(x, y)$;

`nz` — параметр, который устанавливает количество изолиний. Если `nz` — целое число, то в диапазоне между минимальным и максимальным значениями функции $Z(x, y)$ через равные интервалы будут проведены `nz` изолиний. Если же задать `nz` как массив, то изолинии будут проводиться через все указанные в этом массиве значения;

`style` — массив того же размера, что и `nz` — устанавливает цвет для каждого интервала уровней значений;

`strf` — строка, состоящая из трех чисел — «`csa`». Здесь `c` (Captions) устанавливает режим отображения подписей графика (см. табл. 5.5); `s` (Scaling) — режим масштабирования (см. табл. 5.6); `a` (Axes) — определяет положение осей графика (см. табл. 5.7).

`leg` — легенда графика, подпись каждой из кривых — символы, отделяемые знаком `@`. По умолчанию — « ».

`rect` — вектор `[xmin, ymin, xmax, ymax]`, который определяет границы изменения x и y координат графической области окна;

Таблица 5.5. Значение параметра `c` (Captions) строки `strf`

Значение	Описание
0	нет подписей
1	отображаются подписи, заданные параметров `leg`

Таблица 5.6. Значение параметра `s` (Scaling) строки `strf`

Значение	Описание
0	масштабирование по умолчанию
1	устанавливается параметром `rect`
2	масштаб зависит от минимального и максимального значения входных данных
3	выводятся изометрические оси, исходя из значений параметра `rect`
4	выводятся изометрические оси, исходя из входных данных
5	расширение осей для наилучшего вида, исходя из значений параметра `rect`
6	расширение осей для наилучшего вида, исходя из входных данных

Таблица 5.7. Значение параметра `a` (Axes) строки `strf`

Значение	Описание
0	нет осей
1	выводятся оси, ось Y слева
2	выводится рамка вокруг графика без делений
3	выводятся оси, ось Y справа
4	оси центрируются в графической области окна
5	оси выводятся таким образом, чтобы они пересекались в точке $(0;0)$.

`nax` — это массив из четырех значений `[nx, Nx, ny, Ny]`, определяющий число основных и промежуточных делений координатных осей графика. Здесь `Nx (Ny)` — число основных делений с подписями под осью X (Y); `nx (ny)` — число промежуточных делений.

Задача 5.17.
Построить изображение поверхности $Z = \sin(x) \cdot \cos(y)$ с помощью функции contourf.

Введем параметр t и создадим массив его значений, определим при помощи команды `deff` функцию `surf`.

Для наглядности приведем график поверхности $Z = \sin(x) \cdot \cos(y)$, построенный функцией `plot3d1`, и ее изображение на горизонтальной плоскости, сформированное функцией `contourf`, в одном графическом окне. С этой целью обра-

тимся к команде `subplot`, которой разобьем графическое окно на две области для вывода графиков.

Используя `feval`, вычислим значения функции $Z = \sin(x) \cdot \cos(y)$ и построим ее график при помощи `plot3d1`, указав углы обозрения наблюдателя — 80 и 15, а также, вызвав команду `xtitle`, выведем подпись графика — «plot3d1».

Теперь сформируем проекцию поверхности на горизонтальную плоскость посредством функции `contourf`. В качестве параметров передаем ей X, Y и Z-координаты, число изолиний (10), $10 : 20$ — массив, определяющий цвет каждого интервала между изолиниями, а также значения строки `strf="121"` (1 — режим отображения подписей; 2 — выбор масштаба зависит от минимального и максимального значения входных данных; 1 — режим отображения координатных осей, ось Y находится слева).

Обратите внимание, в этой задаче мы впервые создали шкалу цвета. Ее использование часто облегчает чтение графика. Для ее вывода в Scilab существует команда `colorbar (n, m)`, здесь `n` — минимальное значение диапазона,`m` — максимальное значение.

Выведем и для этого графика подписи осей и графика в целом — «contourf» — при помощи команды `xtitle` (см. листинг 5.17, рис. XII).

Листинг 5.17. Построение поверхности $Z = \sin(x) \cdot \cos(y)$ (функция `plot3d`) и ее изображения на горизонтальной поверхности (функция `contourf`) в одном графическом окне

```
t=-%pi:0.2:%pi;
deff('[z]=Surf(x,y)','z=sin(x)*cos(y)');
subplot(121);
z=feval(t,t,Surf);
plot3d1(t,t,z,80,15);
xtitle('plot3d1');
subplot(122);
contourf(t,t,z,10,10:20,strf="121");
colorbar(-%pi,%pi);
xtitle('contourf','X','Y');
```

5.7 Функция hist3d

Для построения трехмерных гистограмм в Scilab используется функция `hist3d`:

```
hist3d(f,[theta,alpha,leg,flag,ebox])
```

Здесь `f` — матрица $(m : n)$, задающая гистограмму $f(i, j) = F(x(i), y(j))$.

Параметры `theta,alpha,leg,flag,ebox` управляют теми же свойствами, что и у функции `plot3d`.

Задача 5.18.
Построить трехмерную гистограмму.

Для формирования матрицы входных данных воспользуемся командой `rand`. Напомним, чтобы создать матрицу размером `(m,n)`, необходимо использовать конструкцию `rand(m,n)` (см. листинг 5.18).

Листинг 5.18. Построение трехмерной гистограммы при помощи функции `hist3d`

```
hist3d(9.7*rand(10,10),20,35);
```

Полученная гистограмма изображена на рис. 5.8.

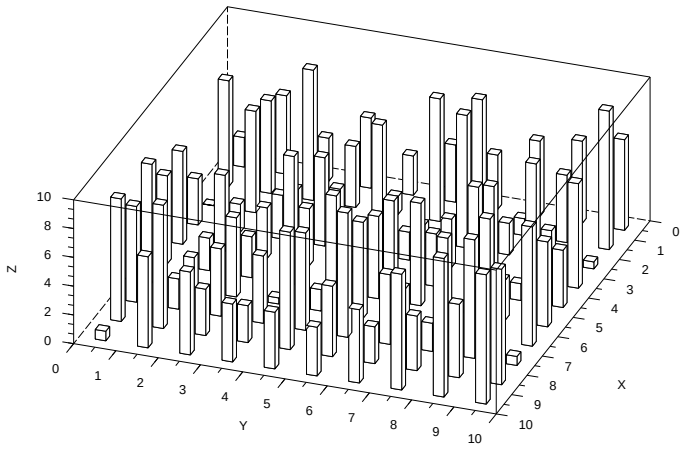

Рис. 5.8. Трехмерная гистограмма, построенная функцией `hist3d`

5.8 Примеры построения некоторых трехмерных графиков в Scilab

В этом параграфе мы рассмотрим приемы построения некоторых нестандартных трехмерных графиков средствами Scilab.

Прежде всего, научимся вырезать из графиков «ненужные» части.

Задача 5.19.
Построить поверхность $Z = \sin(t) \cdot \cos(t)$, вырезать из графика области, где $|Z| > 0.5$.

Сформируем массив значений параметра t, вычислим значения функции $Z = \sin(t) \cdot \cos(t)$ и запишем их в массив Z.

В массив Z_1 при помощи команды `find` запишем индексы тех элементов массива Z, чье модальное значение больше 0,5.

Далее мы будем использовать функцию `%inf`. Она предназначена для определения бесконечных элементов массива, поэтому запись `z(z1)=%inf*z1` приведет к тому, что `%inf*z1` объявит те элементы массива Z, чьи индексы содержатся в массиве Z_1, бесконечным величинами.

При формировании прямоугольной сетки для построения графика в узлах сетки, смежных с бесконечными элементами массива Z, все значения, разумеется, будут равны бесконечности, и функция `plot3d1` зальет соответствующие ячейки белым цветом. Таким образом, нам удастся создать эффект вырезания целых областей поверхности $Z = \sin(t) \cdot \cos(t)$ (см. листинг 5.19, рис. XIII).

Листинг 5.19. Пример «вырезания» из поверхности заданной области

```
t=linspace(-%pi,%pi,40);
z=sin(t)'*cos(t);
z1=find(abs(z)>0.5);
z(z1)=%inf*z1;
plot3d1(t,t,z);
```

Обратите внимание, что на ось Z автоматически будут нанесены подписи отметок, выше и ниже которых значения элементов массива Z были объявлены бесконечными — 0.5 и -0.5.

Теперь поставим перед собой другую задачу: построить геометрическое тело, полое внутри.

Задача 5.20.
Построение полой сферы.

Прежде всего, при помощи команды `deff` создадим функцию `sph`, которая задает сферу тремя параметрическими уравнениями X, Y, Z. Обратите внимание, что функция `sph` записана первыми тремя строками — для удобства (см. листинг 5.20):

```
deff('[x,y,z]=sph(alp,tet)',['x=r*cos(alp).*cos(tet)+...
orig(1)*ones(tet)';
'y=r*cos(alp).*sin(tet)+orig(2)*ones(tet);
'z=r*sin(alp)+orig(3)*ones(tet)']);
```

Далее задаем значения параметра `r`, вектора-строки `orig`, массивов `x` и `y`.

Уже известным нам способом, при помощи функции `%inf`, объявляем бесконечными величинами элементы массива x с индексами $(5 : 8)$ и $(30 : 35)$.

Таким образом мы добьемся того, что в графическое окно будет выведена сфера, имеющая две «шляпки», сверху и снизу (см. рис. XIV). Под ними элементы, объявленные бесконечными, образуют два «окошка», через которые мы сможем увидеть полость внутри сферы.

Саму же сферу построим при помощи команды `eval3dp` и функции `plot3d1`, для лучшего обзора всех деталей графика укажем углы поворота наблюдателя 35 и 15.

Листинг 5.20. Построение сферы с полостью внутри и «срезанными» полюсами

```
deff('[x,y,z]=sph(alp,tet)',['x=r*cos(alp).*cos(tet)+orig(1)*...
ones(tet)';
'y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)';
'z=r*sin(alp)+orig(3)*ones(tet)']);
r=1;orig=[0 0 0];
x=linspace(-%pi/2,%pi/2,40);
y=linspace(0,%pi*2,20);
x(5:8)=%inf*ones(5:8);
x(30:35)=%inf*ones(30:35);
[x1,y1,z1]=eval3dp(sph,x,y);
plot3d1(x1,y1,z1,35,15);
```

Ранее мы уже познакомились с возможностями функции `plot3d2`. Теперь рассмотрим еще несколько примеров ее применения.

Задача 5.21.
Построение ракушкообразного графика.

Такой график можно задать следующей системой уравнений:

$$\begin{cases} x = \cos(u) \cdot u \cdot \left(1 + \cos\dfrac{(v)}{2}\right); \\ y = \dfrac{u}{2} \cdot \sin(v); \\ z = (\sin(u) \cdot u) \cdot \left(1 + \cos\dfrac{(v)}{2}\right). \end{cases}$$

Зададим массивы значений параметров u, v. Вычислим значения функций x, y, z (см. листинг 5.21). Обратившись к функции `plot3d2`, получим график, представленный на рис. XV.

Листинг 5.21. Построение ракушкообразного графика

```
u = linspace(0,2*%pi,40);
v = linspace(0,2*%pi,20);
```

```
x= (cos(u).*u)'*(1+cos(v)/2);
y= (u/2)'*sin(v);
z= (sin(u).*u)'*(1+cos(v)/2);
plot3d2(x,y,z);
```

Задача 5.22.
Построение ленты Мебиуса при помощи функции `plot3d2`.

Лента Мебиуса — простейшая односторонняя поверхность с краем. Попасть из одной точки этой поверхности в любую другую можно, не пересекая края.

В общем параметрическом виде лента Мебиуса может быть представлена системой уравнений:

$$\begin{cases} x(u,v) = \left(1 + \dfrac{v}{2} \cdot \cos\left(\dfrac{u}{2}\right)\right) \cdot \cos(u)\,, \\ y(u,v) = \left(1 + \dfrac{v}{2} \cdot \cos\left(\dfrac{u}{2}\right)\right) \cdot \sin(u)\,, \\ z(u,v) = \dfrac{v}{2} \cdot \sin\left(\dfrac{u}{2}\right)\,. \end{cases}$$

Здесь u принадлежит интервалу $[0; 2\pi]$, а v — $[-1; 1]$. Эти формулы задают ленту Мебиуса ширины 1, чей центральный круг имеет радиус 1, лежит в плоскости xy с центром в $(0, 0, 0)$. параметр u пробегает вдоль ленты, в то время как v задает расстояние от края.

На листинге 5.22 предложен один из способов построения ленты Мебиуса, а ее график представлен на рис. 5.25.

Листинг 5.22. Построение ленты Мебиуса при помощи функции `plot3d2`

```
t=linspace(-1,1,20)';
x=linspace(0,%pi,40);
factor=2+t*cos(x);
X=factor*diag(cos(2*x));
Y=factor*diag(sin(2*x));
Z=t*sin(x);
plot3d2(X,Y,Z);
```

Задача 5.23.
Построение тора с узкой и широкой стороной при помощи функции `plot3d2`.

Тор — поверхность вращения в форме бублика, получаемая вращением окружности вокруг оси, лежащей в плоскости окружности и не пересекающей ее. Уравнение тора может быть задано параметрически в виде:

$$\begin{cases} x(u,v) = (R + r \cdot \cos(u) \cdot \cos(v))\,, \\ y(u,v) = (R + r \cdot \cos(u) \cdot \sin(v))\,, \\ z(u,v) = r \cdot \sin(u)\,, \end{cases}$$

Здесь u, v принадлежат интервалу $[0; 2\pi]$, а также R — расстояние от центра окружности до оси вращения, r — радиус окружности.

На листинге 5.23 приведен способ построения тора с узкой и широкой стороной подобно ленте Мебиуса, график тора избражен на рис. 5.26.

Листинг 5.23. Построение тора с узкой и широкой стороной при помощи функции `plot3d2`

```
x=linspace(0,2*%pi,40);
y=linspace(0,2*%pi,20)';
fact=1.5+cos(y)*(cos(x)/2+0.6);
X=fact*diag(cos(x));
Y=fact*diag(sin(x));
Z=sin(y)*(cos(x)/2+0.6);
plot3d2(X,Y,Z,);
```

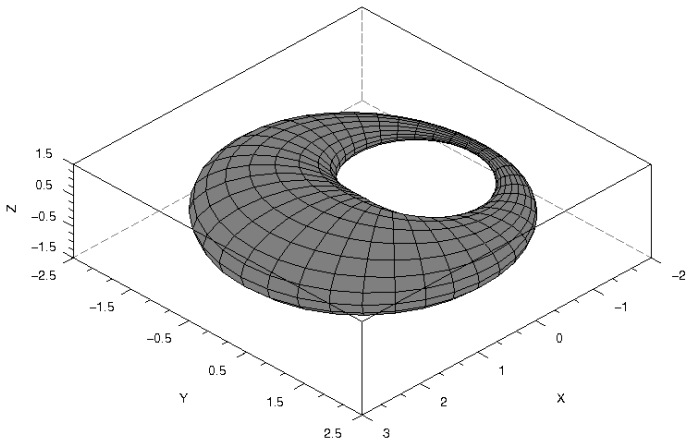

Рис. 5.9. Тор с узкой и широкой стороной

Задача 5.24.
Построение деформированного тора при помощи функции `plot3d2`.

Листинг 5.24 демонстрирует один из возможных способов построения вогнутого тора при помощи функции `plot3d2`, график тора представлен на рис. 5.27.

Листинг 5.24. Построение деформирования тора при помощи функции `plot3d2`

```
x=linspace(0,2*%pi,40);
y=linspace(0,2*%pi,20)';
factor=1.5+cos(y);
X=factor*cos(x);
Y=factor*sin(x);
Z=sin(y)*ones(x)+ ones(y)*cos(2*x);
plot3d2(X,Y,Z);
```

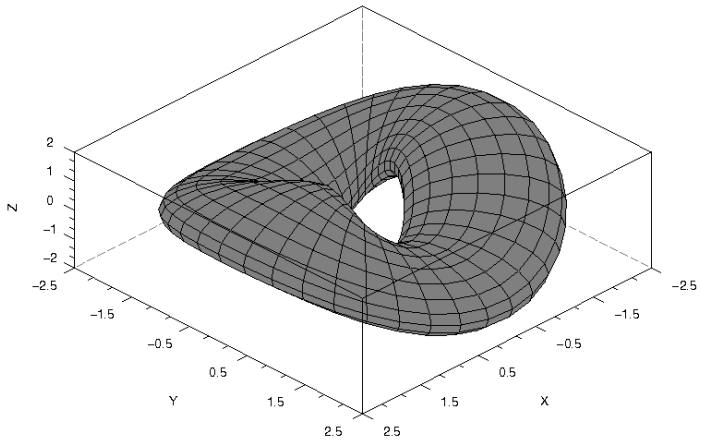

Рис. 5.10. Деформированный тор

Глава 6

Нелинейные уравнения и системы в SCILAB

Если нелинейное уравнение достаточно сложное, то отыскание его корней — процесс нетривиальный. Рассмотрим, какими средствами обладает Scilab для решения этой задачи.

6.1 Алгебраические уравнения

Любое уравнение $P(x) = 0$, где $P(x)$ — это многочлен, отличный от нулевого, называется *алгебраическим уравнением* или *полиномом*. Всякое алгебраическое уравнение относительно x можно записать в виде $a_0x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n = 0$, где $a_0 \neq 0$, $n \geqslant 1$ и a_i — коэффициенты алгебраического уравнения n–й степени. Например, линейное уравнение это алгебраическое уравнение первой степени, квадратное — второй, кубическое — третьей и так далее.

Решение алгебраического уравнения в Scilab состоит из двух этапов. Необходимо задать полином $P(x)$ с помощью функции `poly`, а затем найти его корни, применив функцию `roots`.

Итак, *определение полиномов* в Scilab осуществляет функция

```
poly(a, "x ["fl"]),
```

где `a` — это число или матрица чисел, `x` — символьная переменная, `fl` — необязательная символьная переменная, определяющая способ задания полинома. Символьная переменная `fl` может принимать только два значения — «`roots`» или «`coeff`» (соответственно «`r`» или «`c`»). Если `fl=c`, то будет сформирован полином с коэффициентами, хранящимися в параметре `a`. Если же `fl=r`, то значения

параметра a воспринимаются функцией как корни, для которых необходимо рассчитать коэффициенты соответствующего полинома. По умолчанию fl=r.

Следующий пример отражает создание полинома p, имеющего в качестве корня тройку, и полинома f с коэффициентом 3.

Листинг 6.1. Полиномы первой степени

```
-->p=poly(3,'x','r');
-->f=poly(3,'x','c');
-->p
 p  =
     - 3 + x
-->f
 f  =
      3
```

Далее приведены примеры создания более сложных полиномов.

Листинг 6.2. Использование функции poly

```
-->//Полином с корнями 1, 0 и 2
-->poly([1 0 2],'x')
 ans  =
           2   3
    2x - 3x + x
-->//Полином с коэффициентами 1, 0 и 2
-->poly([1 0 2],'x','c')
 ans  =
          2
    1 + 2x
```

Рассмотрим примеры *символьных операций* с полиномами:

Листинг 6.3. Примеры символьных операций с полиномами

```
-->p1=poly([-1 2],'x','c')
 p1  =
  - 1 + 2x
-->p2=poly([3 -7 2],'x','c')
 p2  =
               2
    3 - 7x + 2x
-->p1+p2 //Сложение
 ans  =
               2
    2 - 5x + 2x
```

```
-->p1-p2 //Вычитание
 ans  =
                2
4 + 9x - 2x
-->p1*p2 //Умножение
 ans  =
                  2    3
  - 3 + 13x - 16x + 4x
-->p1/p2 //Деление
 ans  =
      1
    -----
  - 3 + x
-->p1^2 //Возведение в степень
 ans  =
                2
   1 - 4x + 4x
-->p2^(-1) //Возведение в отрицательную степень
 ans  =
         1
   -----------
                2
   3 - 7x + 2x
```

Функция

```
roots(p)
```

предназначена для *решения алгебраического уравнения*. Здесь `p` — это полином, созданный функцией `poly` и представляющий собой левую часть уравнения $P(x) = 0$.

Решим несколько алгебраических уравнений.

Задача 6.1.
Найти корни полинома $2x^4 - 8x^3 + 8x^2 - 1 = 0$.

Для решения этой задачи необходимо задать полином p. Сделаем это при помощи функции `poly`, предварительно определив вектор коэффициентов V. Обратите внимание, что в уравнении отсутствует переменная x в первой степени, это означает, что соответствующий коэффициент равен нулю:

Листинг 6.4. Формирование полинома

```
-->V=[-1 0 8 -8 2];
-->p=poly(V,'x','c')
 p  =
          2    3    4
     1 + 8x - 8x + 2x
```

Теперь найдем корни полинома:

Листинг 6.5. Использование функции `roots`

```
-->X=roots(p)
 X  =
!   0.4588039 !
! - 0.3065630 !
!   1.5411961 !
!   2.306563  !
```

Графическое решение задачи[1], показанное на рис. 6.1, позволяет убедиться, что корни найдены верно.

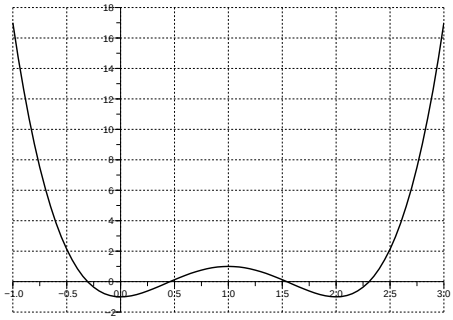

Рис. 6.1. Графическое решение задачи 6.1

Задача 6.2.
Найти корни полинома $x^3 + 0.4x^2 + 0.6x - 1 = 0$.

Решение этой задачи аналогично решению предыдущей, разница заключается в способе вызова необходимых для этого функций:

[1] Графическим решением уравнения $f(x) = 0$ является точка пересечения линии $f(x)$ с осью абсцисс.

Листинг 6.6. Решение задачи 6.2

```
-->roots(poly([-1 0.6 0.4 1],'x','c'))
 ans  =
!   0.7153636              !
! - 0.5576818 + 1.0425361i !
! - 0.5576818 - 1.0425361i !
```

Нетрудно заметить, что полином имеет один действительный (рис. 6.2) и два комплексных корня.

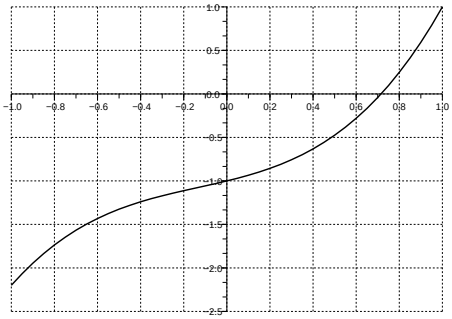

Рис. 6.2. Графическое решение задачи 6.2

Задача 6.3.
Найти решение уравнения $y(x) = 0$, если $y(x) = x^4 - 18x^2 + 6$.

Решение этой задачи представлено в листинге 6.7 и отличается от предыдущих лишь способом определения полинома. Графическое решение представлено на рис. 6.3.

Листинг 6.7. Решение задачи 6.3

```
-->x=poly(0,'x');
-->y=x^4-18*x^2+.6;
-->roots(y)
 ans  =
!   0.1827438 !
! - 0.1827438 !
! - 4.2387032 !
!   4.2387032 !
```

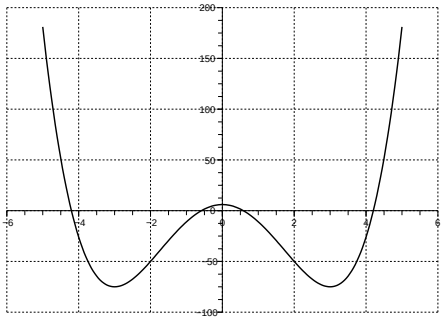

Рис. 6.3. Графическое решение задачи 6.3

6.2 Трансцендентные уравнения

Уравнение $f(x) = 0$, в котором неизвестное входит в аргумент трансцендентных функций, называется *трансцендентным уравнением*. К трансцендентным уравнениям принадлежат показательные, логарифмические и тригонометрические. В общем случае аналитическое решение уравнения $f(x) = 0$ можно найти только для узкого класса функций. Чаще всего приходится решать это уравнение *численными методами*.

Численное решение нелинейного уравнения проводят в два этапа. В начале отделяют *корни уравнения*, т.е. находят достаточно тесные промежутки, в которых содержится только один корень. Эти промежутки называют *интервалами изоляции корня*, определить их можно, изобразив график функции $f(x)$ или любым другим методом[1]. На втором этапе проводят уточнение отделенных корней, или, иначе говоря, находят корни с заданной точностью.

Для решения трансцендентных уравнений в Scilab применяют функцию

```
fsolve(x0,f)
```

где `x0` — начальное приближение, `f` — функция, описывающая левую часть уравнения $y(x) = 0$.

Рассмотрим применение этой функции на примерах.

Задача 6.4.
Найти решение уравнения $\sqrt[3]{(x-1)^2} - \sqrt[3]{x^2} = 0$.

[1]Методы определения интервала изоляции корня основаны на следующем свойстве: если непрерывная функции $f(x)$ на интервале $[a, b]$ поменяла знак, т.е. $f(a) \cdot f(b) < 0$, то она имеет на этом интервале хотя бы один корень.

Определим интервал изоляции корня заданного уравнения. Воспользуемся графическим методом отделения корней. Если выражение, стоящее в правой части уравнения, представить в виде разности двух функций $f(x) - g(x) = 0$, то абсцисса точки пересечения линий $f(x)$ и $g(x)$ — корень данного уравнения. В нашем случае $f(x) = \sqrt[3]{(x-1)^2}$, $g(x) = \sqrt[3]{x^2}$. На рис. 6.4 видно, что корень данного уравнения лежит в интервале $[0; 1]$.

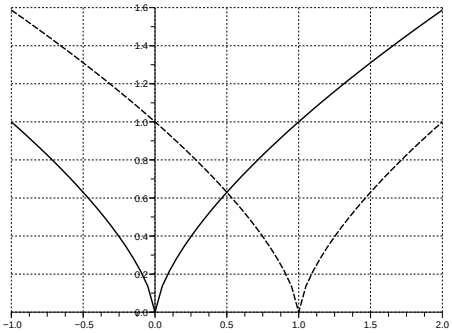

Рис. 6.4. Графическое решение задачи 6.4

Выберем ноль в качестве начального приближения, зададим функцию, описывающую уравнение и решим его:

Листинг 6.8. Решение задачи 6.4

```
-->deff('[y]=f1(x)','y1=((x-1)^2)^(1/3),y2=(x^2)^(1/3),y=y1-y2')
-->fsolve(0,f1)
 ans  =     0.5
```

Задача 6.5.
Найти корни уравнения $f(x) = e^x/5 - 2(x-1)^2$.

На рис. 6.5 видно, что график функции $f(x)$ трижды пересекает ось абсцисс, т.е. уравнение имеет три корня.

Последовательно вызывая функцию `fsolve` с различными начальными приближениями, получим все решения заданного уравнения:

Листинг 6.9. Решение задачи 6.5

```
-->deff('[y]=f(x)','y=exp(x)/5-2*(x-1)^2')
-->x(1)=fsolve(0,f);x(2)=fsolve(2,f);x(3)=fsolve(5,f);
-->x
```

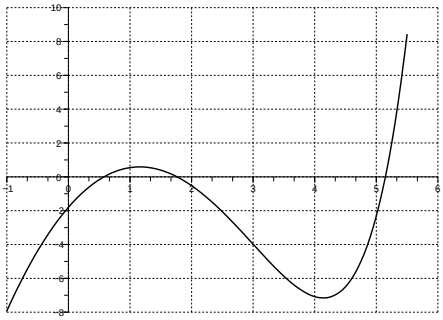

Рис. 6.5. Графическое решение задачи 6.5

```
 x  = !   0.5778406 !
      !   1.7638701 !
      !   5.1476865 !
```

Кроме того, начальные приближения можно задать в виде вектора, и тогда функцию можно вызвать один раз:

Листинг 6.10. Решение задачи 6.5 (альтернативный способ)

```
-->fsolve([0;2;5],f)
 ans  = !   0.5778406 !
        !   1.7638701 !
        !   5.1476865 !
```

Задача 6.6.
Вычислить корни уравнения $\sin(x) - 0.4x = 0$ в диапазоне $[-5\pi; 5\pi]$.

Решение задачи представлено в листинге 6.11.

Листинг 6.11. Решение задачи 6.6

```
-->deff('[y]=fff(x)','y=-0.4+sin(x)')
-->V=[-5*%pi:%pi:5*%pi]; X=fsolve(V,fff);
-->X //Множество решений
 X  = !-16.11948 -12.154854 -9.8362948 -5.8716685 -3.5531095
  0.4115168 2.7300758 6.6947022 9.0132611 12.977887 15.296446!
```

Задача 6.7.
Найти решение уравнения $y(x) = 0$, если $y(x) = x^5 - x^3 + 1$.

Нетрудно заметить, что заданное уравнение — полином пятой степени, который имеет один действительный корень (рис. 6.6) в интервале от –2 до –1.

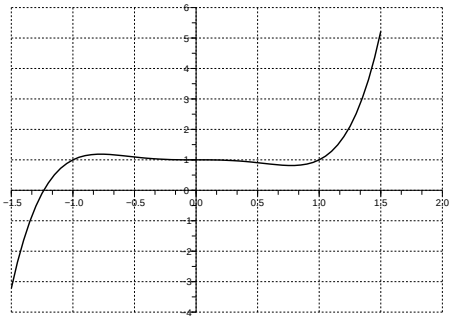

Рис. 6.6. Графическое решение задачи 6.7

Решим эту задачу при помощи функции `fsolve`:

Листинг 6.12. Решение задачи 6.7

```
-->deff('[f]=y(x)','f=x^5-x^3+1')
-->X=fsolve(-2,y)
 X  = 1.2365057
```

Теперь применим функцию `roots`:

Листинг 6.13. Решение задачи 6.7 с использованием функции `roots`

```
-->roots(poly([1 0 0 -1 0 1],'x','c'))
 ans  =
!   0.9590477 + 0.4283660i !
!   0.9590477 - 0.4283660i !
! - 0.3407949 + 0.7854231i !
! - 0.3407949 - 0.7854231i !
! - 1.2365057              !
```

Как видим, заданное уравнение, кроме действительного корня (листинг 6.12), имеет и мнимые (листинг 6.13). Поэтому для отыскания всех корней полинома лучше использовать функцию `roots`.

6.3 Системы уравнений

Если заданы m уравнений с n неизвестными и требуется найти последовательность из n чисел, которые одновременно удовлетворяют каждому из m уравнений, то говорят о *системе уравнений*. Для решения систем уравнений в Scilab также применяют функцию `fsolve(x0,f)`.

Задача 6.8.
Решить систему уравнений: $\{x^2 + y^2 = 1;\ x^3 - 1 = 0\}$.

Графическое решение системы (рис. 6.7) показывает, что она имеет две пары корней.

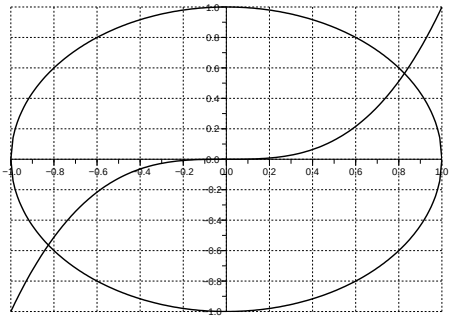

Рис. 6.7. Графическое решение системы уравнений

Окружность и гипербола пересекаются в точках $[0.8; 0.6]$ и $[-0.8; -0.6]$. Эти значения приблизительны. Для того чтобы уточнить их применим функцию `fsolve`, предварительно определив систему с помощью файл–функции:

Листинг 6.14. Решение задачи 6.8

```
function [y]=fun(x)
y(1)=x(1)^2+x(2)^2-1;
y(2)=x(1)^3-x(2);
endfunction
-->exec('C:\fun.sce'); disp('exec done'); exec done
-->fsolve([0.5 0.5],fun)
 ans  = 0.8260314    0.5636242
-->fsolve([-0.5 -0.5],fun)
 ans  =  - 0.8260314  - 0.5636242
```

Задача 6.9.
В данной задаче исследуется система из трех нелинейных уравнений с тремя неизвестными:

Листинг 6.15. Решение задачи 6.9

```
function [y]=fun(x)
y(1)=x(1)^2+x(2)^2+x(3)^2-1
y(2)=2*x(1)^2+x(2)^2-4*x(3)
y(3)=3*x(1)^2-4*x(2)+x(3)^2
endfunction
-->exec('D:\scilab 3\fun');disp('exec done'); exec done
-->fsolve([0.5 0.5 0.5],fun)//решение системы
ans  = !   0.7851969    0.4966114    0.3699228 !
```

Глава 7

Численное интегрирование и дифференцирование

В функциях интегрирования и дифференцирования в Scilab реализованы различные численные алгоритмы.

7.1 Интегрирование по методу трапеций

В Scilab *численное интегрирование по методу трапеций* реализовано с помощью функции `inttrap([x,]y)`. Эта функция вычисляет площадь фигуры под графиком функции $y(x)$, которая описана набором точек (x, y). Параметр x является необязательным. Для функции `inttrap(y)` элементы вектора x принимают значения номеров элементов вектора y.

Задача 7.1.

Вычислить определенный интеграл $\int_5^{13} \sqrt{2x-1}\,dx$.

Этот интеграл легко сводится к табличному $\int_5^{13} \sqrt{2x-1}\,dx = \frac{\sqrt{(2x-1)^3}}{3}$, поэтому вычислить его по формуле Ньютона–Лейбница[1] не составит труда:

[1] Вычисление определенного интеграла по формуле Ньютона-Лейбница:

$$\int_a^b f(x) = F(b) - F(a)\,.$$

Листинг 7.1. Точное решение задачи 7.1

```
-->a=5;b=13;
-->I=1/3*(2*b-1)^(3/2)-1/3*(2*a-1)^(3/2)
 I  =  32.666667
```

Теперь применим для отыскания заданного определенного интеграла *метод трапеций*[1]. Рассмотрим несколько вариантов решения данной задачи. В первом случае интервал интегрирования делится на отрезки с шагом один, во втором 0.5 и в третьем 0.1. Не трудно заметить, что чем больше точек разбиения, тем точнее значение искомого интеграла:

Листинг 7.2. Приближенное решение задачи 7.1 с использованием функции `inttrap`

```
-->x=a:b;y=sqrt(2*x-1);
-->inttrap(x,y)
 ans  =
    32.655571
-->h=0.5; x=a:h:b; y=sqrt(2*x-1);
-->inttrap(x,y)
 ans  =
    32.66389
-->h=0.1; x=a:h:b; y=sqrt(2*x-1);
-->inttrap(x,y)
 ans  =
    32.666556
```

Далее в листинге 7.3 приведен пример использования функции `inttrap` с одним аргументом. Как видим, в первом случае значение интеграла, вычисленного при помощи этой функции, неточно и совпадает со значением, полученным функцией `inttrap(x,y)` на интервале $[5; 13]$ с шагом 1. Т.е. мы нашли сумму площадей восьми прямолинейных трапеций с основанием $h = 1$ и боковыми сторонами, заданными вектором y. Во втором случае, при попытке увеличить точность интегрирования, значение интеграла существенно увеличивается. Дело в том, что, уменьшив шаг разбиения интервала интегрирования до 0.1, мы увеличили количество элементов векторов x и y, и применение функции `inttrap(y)` приведет к вычислению суммы площадей восьмидесяти трапеций с основанием $h = 1$ и боковыми сторонами, заданными вектором y. Таким образом, в первом и втором примерах вычисляются площади совершенно разных фигур.

[1] Для вычисления интеграла методом трапеций участок интегрирования разбивают на определенное количество равных отрезков, каждую из полученных криволинейных трапеций заменяют прямолинейной и вычисляют приближенное значение интеграла как сумму площадей этих трапеций.

Листинг 7.3. Использование функции `inttrap` с одним аргументом

```
-->x=a:b;y=sqrt(2*x-1);
-->inttrap(y)
 ans  =     32.655571
-->h=0.1;x=a:h:b;y=sqrt(2*x-1);
-->inttrap(y)
 ans  =     326.66556
```

7.2 Интегрирование по квадратуре

Методы трапеций являются частными случаями *квадратурных формул Ньютона–Котеса*, которые, вообще говоря, имеют вид

$$\int_a^b y\,dy = (b-a)\sum_{i=0}^{n} H_i y_i \tag{7.1}$$

где H_i — это некоторые константы, называемые постоянными Ньютона–Котеса.

Если для (7.1) принять $n = 1$, то получим *метод трапеций*, а при $n = 2$ — *метод Симпсона*. Эти методы называют квадратурными методами низших порядков. Для $n > 2$ получают *квадратурные формулы Ньютона–Котеса высших порядков*. Вычислительный алгоритм квадратурных формул реализован в Scilab функцией:

```
integrate(fun, x, a, b, [,er1 [,er2]])
```

где `fun` — функция, задающая подынтегральное выражение в символьном виде; `x` — переменная интегрирования, так же задается в виде символа; `a`, `b` — пределы интегрирования, действительные числа; `er1` и `er2` — параметры, отражающие абсолютную и относительную точность вычислений (действительные числа).

Задача 7.2.
Вычислить интеграл из задачи 7.1

Решение показано в листинге 7.4.

Листинг 7.4. Использование функции `integrate`

```
-->integrate('(2*x-1)^0.5','x',5,13)
 ans  =     32.666667
```

7.3 Интегрирование внешней функции

Наиболее универсальной командой интегрирования в Scilab является:

```
[I,err]=intg(a, b, name [,er1 [,er2]])
```

где `name` — имя функции, задающей подынтегральное выражение (здесь функция может быть задана в виде набора дискретных точек (как таблица) или с помощью внешней функции); `a` и `b` — пределы интегрирования; `er1` и `er2` — абсолютная и относительная точность вычислений (необязательные параметры).

Задача 7.3.
Вычислить интеграл из задачи 7.1

Решение показано в листинге 7.5.

Листинг 7.5. Использование функции `intg`

```
-->deff('y=G(x)','y=sqrt(2*x-1)'); intg(5,13,G)
 ans  =     32.666667
```

Задача 7.4.
Вычислить интеграл $\int_0^1 \frac{t^2}{\sqrt{(3+\sin(t))}}\,dt$.

Численное решение интеграла показано в листинге 7.6.

Листинг 7.6. Решение задачи 7.4

```
-->function y=f(t),y=t^2/sqrt(3+sin(t)),endfunction;
-->[I,er]=intg(0,1,f)
 er  =  1.933D-15
 I  =
        0.1741192
```

7.4 Приближенное дифференцирование, основанное на интерполяционной формуле Ньютона

Идея численного дифференцирования заключается в том, что функцию $y(x)$, заданную в равноотстоящих точках x_i $(i = 0, 1, \ldots, n)$ отрезка $[a, b]$ с помощью значений $y_i = f(x_i)$, приближенно заменяют интерполяционным полиномом Ньютона, построенном для системы узлов $x_0, x_1, \ldots, x_k$ $(k \leqslant n)$, и вычисляют производные $y' = f'(x)$, $y'' = f''(x)$ и т. д.

$$y'(x_0) = \frac{1}{h}\left(\Delta y_0 - \frac{\Delta^2 y_0}{2!} + \frac{\Delta^3 y_0}{3!} - \frac{\Delta^4 y_0}{4!} + \frac{\Delta^5 y_0}{5!} - \ldots\right), \qquad (7.2)$$

На практике приближенное дифференцирование применяют в основном для функций, заданных в виде таблицы.

В Scilab численное дифференцирование реализовано командой `dy=diff(y[,n])`, где `y` — значения функции $y(x)$ в виде вектора вещественных чисел, `n` — порядок дифференцирования. Результат работы функции — вектор вещественных чисел `dy`, представляющий собой разности порядка `n` интерполяционного полинома Ньютона $\Delta y, \Delta_2 y, \ldots, \Delta_k y$. Рассмотрим работу функции на примере.

Задача 7.5.
Найти $y'(50)$ функции $y = \lg(x)$, заданной в виде таблицы.

Решение данной задачи с комментариями представлено в листинге 7.7.

Листинг 7.7. Использование функции `diff`

```
-->h=5;x=50:5:65;
-->y=log10(x)
 y  =  1.69897    1.7403627    1.7781513    1.8129134
-->dy=diff(y)
 dy  = 0.0413927    0.0377886    0.0347621
-->dy2=diff(y,2)
 dy2  =  - 0.0036041  - 0.0030265
-->dy3=diff(y,3)
 dy3  =  0.0005777
-->//Приближенное значение y'(50) по формуле (7.2)
-->Y=(dy(1)-dy2(1)/2+dy3(1)/3)/h
 Y  =  0.0086775
-->//Значение y'(50) для lg'(x)=1/ln(10)/x
-->1/log(10)/x(1)
 ans  =     0.0086859
-->//Приближенное значение y'(x), x=50,55,60 (7.2)
-->Y=(dy-dy2(1:$-1)/2+dy3(1:$-2)/3)/h
 Y  =   0.0086389    0.0079181    0.0073128
-->//Значение y'(x), x=50,55,60, для lg'(x)=1/ln(10)/x
-->(1/log(10))./x(1:$-1)
 ans  =  0.0086859    0.0078963    0.0072382
```

7.5 Вычисление производной функции в точке. Приближенное вычисление частных производных

Более универсальной командой дифференцирования является команда

```
g=numdiff(fun,x)
```

здесь `fun` — имя функции, задающей выражение для дифференцирования. Функция должна быть задана в виде `y=fun(x [, p1, p2,..., pn])`, где `x` — переменная, по которой будет проводится дифференцирование. Если параметры `p1`, `p2`, ..., `pn` присутствуют в описании функции, то они должны быть обязательно определены при вызове, например, так: `g=numdiff(list(fun,p1,p2,...pn),x)`.

Результат работы функции — матрица $g_{ij} = \frac{df_i}{dx_j}$.

Рассмотрим несколько примеров.

Задача 7.6.
Вычислить $f'(1)$, если $f(x) = (x+2)^3 + 5x$.

Листинг 7.8 содержит решение данной задачи.

Листинг 7.8. Решение задачи 7.6

```
-->function  f=my(x), f=(x+2)^3+5*x, endfunction;
-->numdiff(my,1)
 ans  = 32.
-->x=1;3*(x+2)^2+5
 ans  = 32.
```

Задача 7.7.
Вычислить $f(x)$ в точках 0, 1, 2, 3 для $f(x) = (x+2)^3 + 5x$.

Решение:

Листинг 7.9. Решение задачи 7.7

```
-->v=0:3;
-->numdiff(my,v)
 ans  =
    17.    0.     0.           0.
    0.     32.    0.           0.
    0.     0.     52.999999    0.
    0.     0.     0.           80.000002
-->function  f1=my1(x), f1=3*(x+2)^2+5, endfunction;
-->my1(v)
 ans  =     17.    32.    53.    80.
```

Задача 7.8.
Задана функция многих переменных $y(x_1, x_2, x_3) = x_1 x_2^{x_3} + x_1^2 x_3$. Вычислить $\frac{dy}{dx_1}$, $\frac{dy}{dx_2}$, $\frac{dy}{dx_3}$ в точке (1, 2, 3).

Решение задачи представлено в листинге 7.10.

Листинг 7.10. Решение задачи 7.8

```
-->function [Y]=f(X), Y=X(1)*X(2)^X(3)+X(1)^2*X(3),endfunction
-->X=[1 2 3];
-->numdiff(f,X)
 ans  =     14.    12.    6.5451775
-->//--------------------------------
-->function [Y]=f1(X),
Y(1)=X(2)^X(3)+2*X(1)*X(3),
Y(2)=X(1)*X(3)*X(2)^(X(3)-1),
Y(3)=x(1)*X(2)^X(3)*log(X(2))+X(1)^2,
endfunction
-->f1(X)
 ans  =
    14.
    12.
    6.5451774
```

Глава 8

Решение обыкновенных дифференциальных уравнений

Дифференциальным уравнением n-го порядка называется соотношение вида

$$H(t, x, x', x'', ..., x^{(n)}) = 0 \tag{8.1}$$

Решением дифференциального уравнения является функция $x(t)$, которая обращает уравнение в тождество.

Системой дифференциальных уравнений n-го порядка называется система вида:

$$\begin{array}{c} x_1' = f_1(t, x_1, x_2, \ldots, x_n) \\ x_2' = f_2(t, x_1, x_2, \ldots, x_n) \\ \cdots \\ x_n' = f_n(t, x_1, x_2, \ldots, x_n) \end{array} \tag{8.2}$$

Решение системы — вектор, который обращает уравнения системы (8.2) в тождества:

$$x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{pmatrix} \tag{8.3}$$

Дифференциальные уравнения и системы имеют бесконечное множество решений, которые отличаются друг от друга константами. Для однозначного определения решения требуется задать дополнительные *начальные* или *граничные условия*. Количество таких условий должно совпадать с порядком дифференциального уравнения или системы. В зависимости от вида дополнительных условий

в дифференциальных уравнениях различают: *задачу Коши* — все дополнительные условия заданы в одной (чаще начальной) точке интервала; *краевую задачу* — дополнительные условия указаны на границах интервала.

Большое количество уравнений может быть решено точно. Однако есть уравнения, а особенно системы уравнений, для которых точное решение записать нельзя. Такие уравнения и системы решают при помощи численных методов. Численные методы также применяют в том случае, если для уравнений с известным аналитическим решением требуется найти числовое значение при определенных исходных данных.

Для решения дифференциальных уравнений и систем в Scilab предусмотрена функция

```
[y,w,iw]=ode([type],y0,t0,t [,rtol [,atol]],f [,jac] [,w,iw])
```

для которой обязательными входными параметрами являются: `y0` — вектор начальных условий; `t0` — начальная точка интервала интегрирования; `t` — координаты узлов сетки, в которых происходит поиск решения; `f` — внешняя функция, определяющая правую часть уравнения или системы уравнений (8.2); `y` — вектор решений (8.3).

Таким образом, для того чтобы решить обыкновенное дифференциальное уравнение вида $\frac{dy}{dt} = f(t, y),\ y(t_0) = y_0$, необходимо вызвать функцию `y=ode(y0,t0,t,f)`.

Рассмотрим необязательные параметры функции `ode`:

`type` — параметр, с помощью которого можно выбрать метод решения или тип решаемой задачи, указав одну из строк: `adams` — применяют при решении дифференциальных уравнений или систем методом прогноза-коррекции Адамса; `stiff` — указывают при решении жестких задач; `rk` — используют при решении дифференциальных уравнений или систем методом Рунге — Кутта четвертого порядка; `rkf` — указывают при выборе пятиэтапного метода Рунге — Кутта четвертого порядка; `fix` — тот же метод Рунге — Кутта, но с фиксированным шагом;

`rtol, atol` — относительная и абсолютная погрешности вычислений, вектор, размерность которого совпадает с размерностью вектора `y`, по умолчанию `rtol=0.00001`, `atol=0.0000001`, при использовании параметров `rkf` и `fix - rtol=0.001`, `atol=0.0001`;

`jac` — матрица, представляющая собой якобиан правой части жесткой системы дифференциальных уравнений, задают матрицу в виде внешней функции вида `J=jak(t,y)`;

`w, iw` — векторы, предназначенные для сохранения информации о параметрах интегрирования, которые применяют для того, чтобы последующие вычисления выполнялись с теми же параметрами.

Рассмотрим использование функции на примере следующих задач.

Задача 8.1.
Решить задачу Коши $\frac{dx}{dt} + x = \sin(xt)$, $x(0) = 1.5$.

Перепишем уравнение следующим образом: $\dfrac{dx}{dt} = -x + \sin(xt)$, $x(0) = 1.5$.

График, моделирующий процесс, описанный заданным уравнением, представлен на рис. 8.1

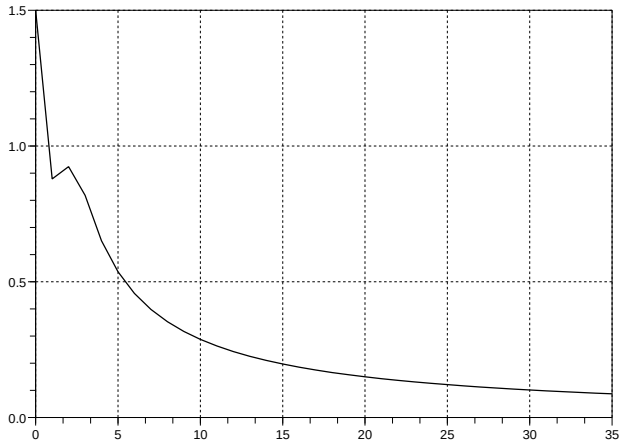

Рис. 8.1. Решение дифференциального уравнения из задачи 8.1

Далее представим его в виде внешней функции и применим функцию `y=ode(x0,t0,t,f)`, в качестве параметров которой будем использовать:

`f` — ссылка на предварительно созданную функцию $f(t, x)$;

`t` — координаты сетки;

`x0, t0` — начальное условие $x(0) = 1.5$;

`y` — результат работы функции.

Листинг 8.1. Решение задачи 8.1

```
-->function yd=f(t,x),yd=-x+sin(t*x),endfunction;
-->x0=1.5;t0=0;t=0:1:35;
-->y=ode(x0,t0,t,f);
-->plot(t,y)
```

Задача 8.2.
Решить задачу Коши

$$x' = \cos(xy)\,,$$
$$y' = \sin(x + ty)\,,$$
$$x(0) = 0, \quad y(0) = 0.$$

на интервале [0; 10].

Далее приведена функция, описывающая заданную систему обыкновенных дифференциальных уравнений, и команды Scilab, необходимые для ее численного и графического решения (рис. 8.2).

Листинг 8.2. Решение задачи 8.2

```
//Функция, описывающая систему дифференциальных уравнений
function dy=syst(t,y)
dy=zeros(2,1);
dy(1)=cos(y(1)*y(2));
dy(2)=sin(y(1)+y(2)*t);
endfunction
//Решение системы дифференциальных уравнений
x0=[0;0];t0=0;t=0:*0.1*:10;y=ode(x0,t0,t,syst);
//Формирование графического решения
plot(t,y)
```

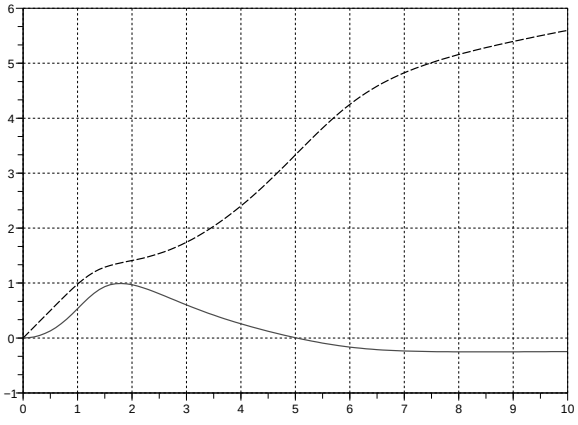

Рис. 8.2. Решение задачи 8.2

Задача 8.3.
Найти решение задачи Коши для следующей жесткой системы:

$$\frac{dX}{dt} = \begin{pmatrix} 119.46 & 185.38 & 126.88 & 121.03 \\ -10.395 & -10.136 & -3.636 & 8.577 \\ -53.302 & -85.932 & -63.182 & 54.211 \\ -115.58 & -181.75 & -112.8 & -199 \end{pmatrix} X; \qquad X(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Решение системы:

Листинг 8.3. Решение задачи 8.3

```
-->B=[119.46 185.38 126.88 121.03;-10.395 -10.136 -3.636 8.577;
-->-53.302 -85.932 -63.182 -54.211;-115.58 -181.75 -112.8 -199];
-->function dx=syst1(t,x), dx=B*x,endfunction
-->function J=Jac(t,y),J=B,endfunction
-->x0=[1;1;1;1]; t0=0; t=0:0.01:5;
-->y=ode("stiff",x0,t0,t,syst1,Jac);
-->plot(t,y); xgrid();
```

Графическое решение показано на рис. 8.3.

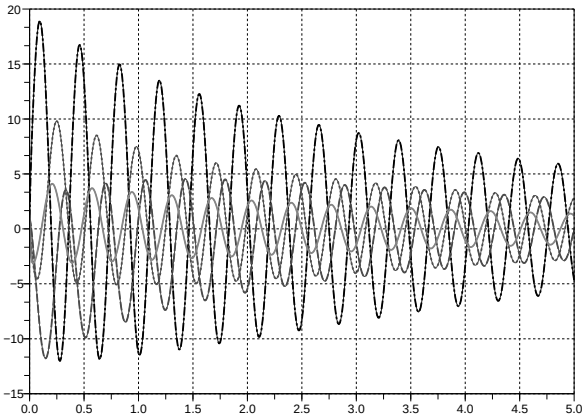

Рис. 8.3. Графическое решение задачи 8.3

Задача 8.4.
Решить нелинейную жесткую систему дифференциальных уравнений:

$$\left\{ \begin{array}{l} \dfrac{dx_1}{dt} = -7x_1 + 7x_2 \\ \dfrac{dx_2}{dt} = 157x_1 - 1.15x_2x_3 \\ \dfrac{dx_3}{dt} = 0.96x_1x_2 - 8.36x_3 \end{array} \right\}, \qquad X(0) = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}.$$

На рис. 8.4 показано решение системы на интервале $[0; 2]$.

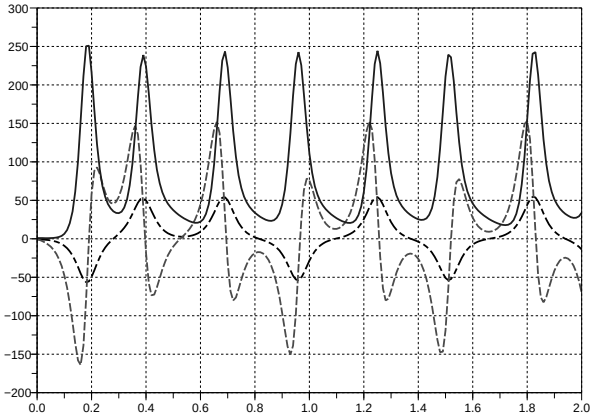

Рис. 8.4. Графическое решение задачи 8.4

Команды Scilab, необходимые для решения задачи:

Листинг 8.4. Решение задачи 8.4

```
function dx=syst2(t,x) //Функция задающая систему ОДУ
dx=zeros(3,1);
dx(1)=-7*x(1)+7*x(2);
dx(2)=157*x(1)+x(2)-1.15*x(1)*x(3);
dx(3)=0.96*x(1)*x(2)-8.36*x(3);
endfunction
-->//Решение ОДУ
-->x0=[-1;0;1]; t0=0; t=0:0.01:2;y=ode("stiff",x0,t0,t,syst2);
-->plot(t,y); xgrid();
```

Задача 8.5.
Решить следующую краевую задачу на интервале $[0.25; 2]$:

$$\frac{d^2x}{dt^2} + 4\frac{dx}{dt} + 13 = e^{\sin(t)}, \qquad x(0.25) = -1, \qquad x'(0.25) = 1.$$

Преобразуем уравнение в систему, сделав замену $y = \dfrac{dx}{dt}$:

$$\frac{dy}{dt} = -4y - 13x + e^{\sin(t)}, \qquad \frac{dx}{dt} = y, \qquad y(0.25) = 1, \qquad x(0.25) = -1.$$

Составим функцию вычисления системы и решим ее:

Листинг 8.5. Решение задачи 8.5

```
function F=FF(t,x)
F=[-4*x(1)-13*x(2)+exp(t);x(1)];
endfunction
-->//Решение системы дифференциальных уравнений
-->X0=[1;-1];t0=0.25;t=0.25:0.05:2;
-->y=ode("stiff",X0,t0,t,FF);
-->//Вывод графика решения
-->plot(t,y); xgrid();
```

График решения приведен на рис. 8.5.

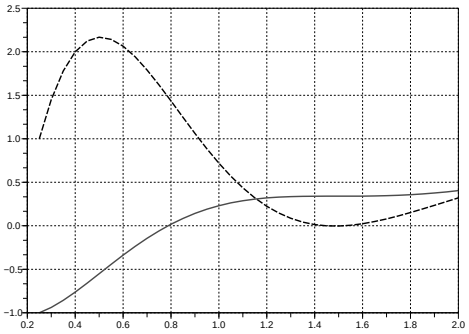

Рис. 8.5. Решение задачи 8.5

Глава 9

Программирование в Scilab

В Scilab встроен мощный язык программирования с поддержкой объектов. В этой главе будет описаны возможности структурного программирования, следующая глава будет посвящена визуальному программированию в среде Scilab.

Как уже отмечалось раннее, работа в Scilab может осуществляться не только в режиме командной строки, но и в так называемом программном режиме. Напомним, что для создания программы (программу в Scilab иногда называют сценарием) необходимо:

1. Вызвать команду *Editor* из меню (см. рис. 9.1).

2. В окне редактора *Scipad* набрать текст программы.

3. Сохранить текст программы с помощью команды *File – Save* в виде файла с расширением `sce`, например, `file.sce`.

4. После этого программу можно будет вызвать, набрав в командной строке `exec`, например, `exec("file.sce")`. Другие способы вызова — воспользоваться командой меню *File – Exec...* или, находясь в окне Scipad, выполнить команду *Execute – Load into Scilab* (`Ctrl+L`).

Программный режим достаточно удобен, так как он позволяет сохранить разработанный вычислительный алгоритм в виде файла и повторять его при других исходных данных в других сессиях. Кроме обращений к функциям и операторов присваивания, в программных файлах могут использоваться операторы языка программирования Scilab (язык программирования Scilab будем называть sci-языком).

Рис. 9.1. Окно Scipad с программой решения кубического уравнения

9.1 Основные операторы sci-языка

Изучение sci-языка начнем с функций ввода-вывода.

9.1.1 Функции ввода-вывода в Scilab

Для организации простейшего ввода в Scilab можно воспользоваться функциями

```
x=input('title');
```

или

```
x=x_dialog('title', 'stroka');
```

Функция `input` выводит в командной строке Scilab подсказку `title` и ждет, пока пользователь введет значение, которое в качестве результата возвращается в переменную `x`. Функция `x_dialog` выводит на экран диалоговое окно с именем `title`, после чего пользователь может щелкнуть OK, и тогда `stroka` вернется в качестве результата в переменную `x`, либо ввести новое значение вместо `stroka`,

которое и вернется в качестве результата в переменную x. На рисунке 9.2 представлено диалоговое окно, которое формируется строкой x=x_dialog('Input X', '5').

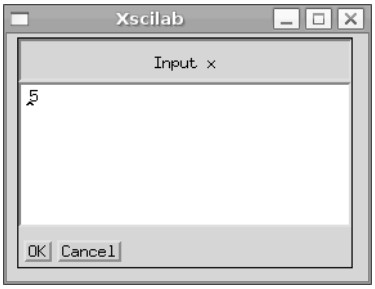

Рис. 9.2. Окно ввода

Функция input преобразовывает введенное значение к числовому типу данных, а функция x_dialog возвращает строковое значение. Поэтому при использовании функции x_dialog для ввода числовых значений возвращаемую ею строку следует преобразовать в число с помощью функции evstr. Можно предложить следующую форму использования функции x_dialog для ввода числовых значений:

```
x=evstr(x_dialog('title','stroka'));
```

Для вывода в текстовом режиме можно использовать функцию disp следующей структуры:

```
disp(b)
```

Здесь b — имя переменной или заключенный в кавычки текст.

9.1.2 Оператор присваивания

Оператор присваивания имеет следующую структуру

```
a=b
```

здесь a — имя переменной или элемента массива, b — значение или выражение. В результате выполнения оператора присваивания переменной a присваивается значение выражения b.

9.1.3 Условный оператор

Одним из основных операторов, реализующих ветвление в большинстве языков программирования, является условный оператор if. Существует обычная и расширенная формы оператора if в Scilab. Обычный if имеет вид

```
if условие
операторы1
else
операторы2
end
```

Здесь `условие` — логическое выражение, `операторы1`, `операторы2` — операторы языка Scilab или встроенные функции. Оператор `if` работает по следующем алгоритму: если условие истинно, то выполняются `операторы1`, если ложно — `операторы2`.

В Scilab для построения логических выражений могут использоваться условные операторы: `&`, `and` (логическое и), `|`, `or` (логическое или), `~`, `not` (логическое отрицание) и операторы отношения: `<` (меньше), `>` (больше), `==` (равно), `~=`, `<>` (не равно), `<=` (меньше или равно), `>=` (больше или равно).

Зачастую при решении практических задач недостаточно выбора выполнения или невыполнения одного условия. В этом случае можно, конечно, по ветке `else` написать новый оператор `if`, но лучше воспользоваться расширенной формой оператора `if`.

```
if условие1
операторы1
elseif условие2
операторы2 elseif условие 3
операторы3
...
elseif условие n
операторыn
else
операторы
end
```

В этом случае оператор `if` работает так: если `условие1` истинно, то выполняются `операторы1`, иначе проверяется `условие2`, если оно истинно, то выполняются `операторы2`, иначе проверяется `условие3` и т. д. Если ни одно из условий по веткам `else` и `elseif` не выполняется, то выполняются операторы по ветке `else`.

Задача 9.1.
В качестве примера программирования разветвляющегося процесса рассмотрим решение биквадратного уравнения $ax^4 + bx^2 + c = 0$.

Для решения биквадратного уравнения необходимо заменой $y = x^2$ привести его к квадратному и решить это уравнение. После этого для нахождения корней биквадратного уравнения необходимо будет извлечь корни из найденных значений y. Входными данными этой задачи являются коэффициенты биквадратного

уравнения a, b, c. Выходными данными являются корни уравнения x_1, x_2, x_3, x_4 или сообщение о том, что действительных корней нет.

Алгоритм состоит из следующих этапов:

1. Ввод коэффициентов уравнения a, b и c;
2. Вычисление дискриминанта уравнения d;
3. Если $d < 0$, определяются y_1 и y_2, в противном случае выводится сообщение «Корней нет».
4. Если $y_1 < 0$ и $y_2 < 0$, то вывод сообщения «Корней нет».
5. Если $y_1 \geqslant 0$ и $y_2 \geqslant 0$, то вычисляются четыре корня по формулам $\pm\sqrt{y_1}$, $\pm\sqrt{y_2}$ и выводятся значения корней.
6. Если условия 4) и 5) не выполняются, то необходимо проверить знак y_1.
7. Если y_1 неотрицательно, то вычисляются два корня по формуле $\pm\sqrt{y_1}$, иначе оба корня вычисляются по формуле $\pm\sqrt{y_2}$.

Программа решения биквадратного уравнения на sci-языке приведена на листинге 9.1, ее вызов и результаты — на листинге 9.2.

Листинг 9.1. Программа решения биквадратного уравнения

```
//Ввод значений коэффициентов биквадратного уравнения.
a=input('a=');
b=input('b=');
c=input('c=');
//Вычисляем дискриминант.
d=b*b-4*a*c;
//Если дискриминант отрицателен,
if d<0
//то вывод сообщения,
disp('Real roots are not present');
else
//иначе вычисление корней соответствующего
//квадратного уравнения.
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
//Если оба корня отрицательны,
if (x1<0)&(x2<0)
//вывод сообщения об отсутствии действительных корней.
disp('Real roots are not present');
//иначе, если оба корня положительны,
elseif (x1>=0)&(x2>=0)
```

```
//вычисление четырех корней.
disp('Four real roots');
y1=sqrt(x1);
y2=-y1;
y3=sqrt(x2);
y4=-y2;
disp(y1,y2,y3,y4);
//Иначе, если оба условия (x1<0)&(x2<0) и (x1>=0)&(x2>=0)
//не выполняются,
else
//то вывод сообщения
disp('Two real roots');
//Проверка знака x1.
if x1>=0
//Если x1 положителен, то вычисление двух корней биквадратного
//уравнения извлечением корня из x1,
y1=sqrt(x1);
y2=-y1;
disp(y1);
disp(y2);
//иначе (остался один вариант - x2 положителен),
//вычисление двух
//корней биквадратного уравнения извлечением корня из x2.
else
y1=sqrt(x2); y2=-y1;
disp(y1); disp(y2);
end
end end
```

Листинг 9.2. Решение биквадратного уравнения

```
-->exec("G:/Lecture Scilab EG/2/l1.sci");
a=-->-6
b=-->9
c=-->-1
Four real roots
0.3476307
1.1743734
- 0.3476307
0.3476307
```

Найти все корни биквадратного уравнения можно и без оператора `if`, воспользовавшись тем, что в Scilab определены операции над комплексными числами (см. листинг 9.3).

Листинг 9.3. Решение биквадратного уравнения

```
a=input('a=');
b=input('b=');
c=input('c=');
d=b*b-4*a*c;
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
y1=sqrt(x1);
y2=-y1;
y3=sqrt(x2);
y4=-y3;
disp(y1,y2,y3,y4);
```

Результат работы программы, представленной на листинге 9.3 представлены ниже (см. листинг 9.4).

Листинг 9.4. Комплексные корни биквадратного уравнения

```
-->a=3
-->b=8
-->c=-1
-1.6692213i
1.6692213i
-0.3458800
0.3458800
```

9.1.4 Оператор альтернативного выбора

Еще одним способом организации разветвлений является оператор альтернативного выбора `select` следующей структуры:

```
select параметр
case значение1 then операторы1
case значение2 then операторы2
...
else операторы
end
```

Оператор `select` работает следующим образом: если значение `параметра` равно `значениию1`, то выполняются `операторы1`, иначе, если `параметр` равен `значениию2`, то выполняются `операторы2`. В противном случае, если значение `параметра` совпадает со `значением3`, то выполняются `операторы3` и т. д. Если значение параметра не совпадает ни с одним из значений в группах `case`, то выполняются `операторы`, которые идут после служебного слова `else`.

Конечно, любой алгоритм можно запрограммировать без использования `select`, используя только `if`, но использование оператора альтернативного выбора `select` делает программу более компактной.

Рассмотрим использование оператора `select` на примере решения следующей задачи.

Задача 9.2.
Вывести на печать название дня недели, соответствующее заданному числу D, при условии, что в месяце 31 день и 1-е число — понедельник.

Для решения задачи воспользуемся условием, что 1-е число — понедельник. Если в результате остаток от деления заданного числа D на семь будет равен единице, то это понедельник, двойке — вторник, тройке — среда и так далее. Вычислить остаток от деления числа x на k можно по формуле $x - \mathrm{int}(x/k) * x$. Следовательно, при построении алгоритма необходимо использовать семь условных операторов.

Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта (см. листинг 9.5). Вызов программы представлен на листинге 9.6.

Листинг 9.5. Решение задачи 9.2

```
D=input('Enter a number from 1 to 31');
//Вычисление остатка от деления D на 7, сравнение его с числами
//от 0 до 6.
select D-int(D/7)*7
case 1  then disp('Monday');
case 2 then disp('Tuesday');
case 3 then disp('Wednesday');
case 4 then disp('Thursday');
case 5 then disp('Friday');
case 6 then disp('Saturday');
else
disp('Sunday');
end
```

Листинг 9.6. Вызов функции для решения задачи 9.2

```
-->exec('G:\Lecture Scilab EG\2\l2.sci');disp('exec done');
Enter a number from 1 to 31-->19
 Friday
```

Рассмотрим операторы цикла в Scilab. В sci-языке есть два вида цикла: оператор цикла с предусловием `while` и оператор `for`.

9.1.5 Оператор while

Оператор цикла `while` имеет вид

```
while условие
операторы
end
```

Здесь `условие` — логическое выражение; `операторы` будут выполняться циклически, пока логическое `условие` истинно.

Оператор цикла `while` обладает значительной гибкостью, но не слишком удобен для организации «строгих» циклов, которые должны быть выполнены заданное число раз. Оператор цикла `for` используется именно в этих случаях.

9.1.6 Оператор for

Оператор цикла `for` имеет вид

```
for x=xn:hx:xk
операторы
end
```

Здесь `x` — имя скалярной переменной — параметра цикла, `xn` — начальное значение параметра цикла, `xk` — конечное значение параметра цикла, `hx` — шаг цикла. Если шаг цикла равен 1, то `hx` можно опустить, и в этом случае оператор `for` будет таким.

```
for x=xn:xk
операторы
end
```

Выполнение цикла начинается с присвоения параметру стартового значения ($x = xn$). Затем следует проверка, не превосходит ли параметр конечное значение ($x > xk$). Если $x > xk$, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. Если же $x \leqslant xk$, то выполняются операторы в цикле (тело цикла). Далее параметр цикла увеличивает свое значение на hx ($x = x + hx$). После чего снова производится проверка значения параметра цикла, и алгоритм повторяется.

9.2 Обработка массивов и матриц в Scilab

Рассмотрим возможности sci-языка для обработки массивов и матриц. Особенностью программирования задач обработки массивов (одномерных, двумерных) на sci-языке является возможность как поэлементной обработки массивов (как в любом языке программирования), так и использование функций Scilab для работы массивами и матрицами.

Рассмотрим основные алгоритмы поэлементной обработки массивов и матриц и их реализацию на sci-языке.

9.2.1 Ввод-вывод массивов и матриц

Ввод массивов и матриц следует организовывать поэлементно. На листингах 9.7 и 9.8 приведены программы ввода элементов массивов и матриц на sci-языке.

Листинг 9.7. Ввод элементов массива

```
N=input('N=');
disp('Vvod massiva x');
for i=1:N
x(i)=input('X=');
end
disp(x);
```

Листинг 9.8. Ввод элементов матрицы

```
N=input('N=');
M=input('M=');
disp('Vvod matrici');
for i=1:N
for j=1:M
a(i,j)=input('');
end
end
disp(a);
```

Вывод массива (матрицы) можно организовать аналогичным образом в цикле или воспользоваться оператором `disp` для вывода массива (матрицы) целиком[1].

9.2.2 Вычисление суммы и произведения элементов массива (матрицы)

Рассмотрим алгоритм нахождения суммы, который заключается в следующем: вначале сумма равна 0 ($s = 0$), затем к s добавляем первый элемент массива и результат записываем опять в переменную s, далее к переменной s добавляем второй элемент массива и результат записываем в s и далее аналогично добавляем к s остальные элементы массива. При нахождении суммы элементов матрицы последовательно суммируем элементы всех строк.

[1]Напоминаем читателю, что если после имени массива (матрицы) не поставить точку с запятой, то массив (матрица) будет выведена в окне Scilab.

Алгоритм нахождения произведения следующий: на первом этапе начальное значение произведения равно 1 ($p = 1$). Затем p последовательно умножается на очередной элемент, и результат записывается в p и т. д. На листингах 9.9–9.12 представлены элементы программ, реализующие эти алгоритмы.

Листинг 9.9. Программа вычисления суммы элементов массива

```
//Записываем в переменную s число 0.
s=0;
//Перебираем все элементы массива
for i=1:length(x)
//накопление суммы
s=s+x(i);
end
```

Листинг 9.10. Программа вычисления произведения элементов массива

```
p=1;
for i=1:length(x)
p=p*x(i);
end
```

Листинг 9.11. Программа вычисления суммы элементов матрицы

```
s=0;
//Вычисляем количество строк N и столбцов M матрицы A.
[N,M]=size(A);
for i=1:N
for j=1:M
s=s+a(i,j);
end
end
disp(s);
```

Листинг 9.12. Программа вычисления произведения элементов матрицы

```
//Начальное значение произведение (p) равно 1.
p=1;
//Вычисляем количество строк N и столбцов M матрицы A.
[N,M]=size(A);
//Перебираем все строки матрицы.
for i=1:N
//Перебираем все столбцы матрицы.
```

```
for j=1:M
//Умножаем значение p на текущий элемент матрицы.
p=p*a(i,j);
end
end
```

9.2.3 Поиск максимального (минимального) элемента массива (матрицы)

Алгоритм решения задачи поиска максимума и его номера в массиве следующий. Пусть в переменной с именем `Max` хранится значение максимального элемента массива, а в переменной с именем `Nmax` — его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную `Max`, а в `Nmax` — его номер (1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную `Max`, а в переменную `Nmax` — текущее значение индекса `i`. На листинге 9.13 представлен фрагмент программы поиска максимума.

Листинг 9.13. Реализация алгоритма поиска максимума

```
//Записываем в Max значение первого элемента массива.
Max=a(1);
//Записываем в Nmax номер максимального элемента
//массива, сейчас это число 1.
Nmax=1;
//Перебираем все элементы массива, начиная со второго.
for i=2:N
//Если текущий элемент массива больше Max,
if x(i)>Max
//то текущий элемент массива объявляем максимальным,
Max=x(i);
//а его номер равен i.
Nmax=i;
end;
end;
```

Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в операторе `if` знак поменяется с «>» на «<».

На листинге 9.14 представлена программа поиска минимального элемента матрицы и его индексов: `Nmin` — номер строки, `Lmin` — номер столбца минимального элемента.

Обратите внимание, что при поиске минимального (максимального) элемента матрицы циклы по `i` и `j` начинаются с 1. Если написать с двух, то при обработке

элементов будет пропущена первая строка или первый столбец при сравнении $a_{i,j}$ с `min`.

Листинг 9.14. Программа поиска минимального элемента матрицы и его индексов

```
//Записываем в Min a(1,1), в Nmin и Lmin число 1.
Min=a(1,1); Nmin=1; Lmin=1;
for i=1:N
for j=1:M
//Если текущий элемент матрицы меньше Min,
if a(i,j)<Min
//то текущий элемент массива объявляем минимальным,
Min=a(i,j);
//а его индексы равны i и j.
Nmin=i;
Lmin=j;
end;
end;
end;
```

9.2.4 Сортировка элементов массива

Сортировка представляет собой процесс упорядочивания элементов массива в порядке возрастания или убывания их значений. Например, массив X из n элементов будет отсортирован[1] в порядке возрастания значений его элементов, если

$$X[1] \leqslant X[2] \leqslant \cdots \leqslant X[n]\,,$$

и в порядке убывания, если

$$X[1] \geqslant X[2] \geqslant \cdots \geqslant X[n]\,.$$

Рассмотрим наиболее известный алгоритм сортировки методом пузырька. Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Те же действия выполним для второго и третьего, третьего и четвертого, ..., i-го и $(i+1)$-го, ..., $(n-1)$-го и n-го элементов. В результате этих действий самый большой элемент станет на последнее (n-е) место. Теперь повторим данный алгоритм сначала, но последний (n-й) элемент рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на $(n-1)$-е место, далее следует повторить алгоритм до тех пор, пока не упорядочим массив.

[1] Зачастую алгоритм сортировки называют алгоритмом упорядочивания элементов массива.

Алгоритм сортировки по убыванию будет отличаться от приведенного заменой знака «>» на «<». На листинге 9.15 представлен фрагмент программы на sci-языке, реализующий алгоритм сортировки по возрастанию.

Листинг 9.15. Программа упорядочивания по возрастанию

```
x=[-3 5 7 49 -8 11 -5 32 -11];
for i=1:length(x)-1
for j=1:length(x)-i
//Сравниваем два соседних элемента, и если предыдущий меньше
//последующего,
if x(j)>x(j+1)
//то меняем их местами.
b=x(j);
x(j)=x(j+1);
x(j+1)=b;
end;
end;
end;
//Вывод упорядоченного массива.
disp(x);
```

9.2.5 Удаление элемента из массива

Необходимо удалить из массива x, состоящего из n элементов, m-й по номеру элемент. Для этого достаточно записать $(m+1)$-й элемент на место элемента с номером m, $(m+2)$-й — на место $(m+1)$-го, ..., n-й — на место $(n-1)$-го, после чего удалить последний n-й элемент. На листинге 9.16 приведен фрагмент программы, реализующий описанный алгоритм.

Листинг 9.16. Программа удаления m-го элемента из массива $x(n)$

```
x=[3 2 1 5 4 6 8 7];
disp(x);
n=length(x);
//Ввод номера удаляемого элемента.
m=input('m=');
//Сдвиг всех элементов, начиная с m-го на один влево.
for i=m:n-1
x(i)=x(i+1);
end;
//Удаление n-го элемента из массива.
x(:,n)=[];
//Уменьшение n на 1.
n=n-1;
//Вывод преобразрванного массива.
disp(x);
```

9.3 Работа с файлами в Scilab

Рассмотрим функции Scilab для работы с файлами[1].

9.3.1 Функция открытия файла mopen

Как и в любом другом языке программирования, работа с файлом начинается с его открытия. Для открытия файла в sci-языке предназначена функция `mopen`, которая имеет вид:

```
[fd,err]=mopen(file,mode)
```

`file` — строка, в которой хранится имя файла,

`mode` — режим работы с файлом:

- `'r'` — текстовый файл открывается в режиме чтения,
- `'rb'` — двоичный файл открывается в режиме чтения,
- `'w'` — открывается пустой текстовый файл, который предназначен только для записи информации;
- `'wb'` — открывается пустой двоичный файл, который предназначен только для записи информации;
- `'a'` — открывается текстовый файл, который будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- `'ab'` — открывается двоичный файл, который будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- `'r+'` — открывается текстовый файл, который будет использоваться в режиме чтения и записи;
- `'rb+'` — открывается двоичный файл, который будет использоваться в режиме чтения и записи;
- `'w+'` — создаваемый пустой текстовый файл предназначен для чтения и записи информации;
- `'wb+'` — создаваемый пустой двоичный файл предназначен для чтения и записи информации;
- `'a+'` — открываемый текстовый файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан;
- `'ab+'` — открываемый двоичный файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан.

[1] Авторы обращают внимание читателей, что функции работы с файлами в sci-языке аналогичны соответствующим функциям в языке C.

Функция `mopen` возвращает идентификатор открытого файла `fd` и код ошибки `err`. Идентификатор файла — имя (код), по которому описанные ниже функции будут обращаться к реальному файлу на диске. В переменной `err` возвращается значение 0 в случае удачного открытия файла. Если `err` $\neq 0$, то это значит, что файл открыть не удалось.

9.3.2 Функция записи в текстовый файла mfprintf

Функция записи в текстовый файл `mfprintf` имеет вид

```
mfprintf(f, s1, s2)
```

Здесь `f` — идентификатор файла (значение идентификатора возвращается функцией `mopen`), `s1` — строка вывода, `s2` — список выводимых переменных.

В строке вывода вместо выводимых переменных указывается строка преобразования следующего вида:

```
%[флаг][ширина][.точность][модификатор]тип
```
[1]

Значения параметров строки преобразования приведены в таблице 9.1.

В строке вывода могут использоваться некоторые специальные символы, приведенные в табл. 9.2.

9.3.3 Функция чтения данных из текстового файла mfscanf

При считывании данных из файла можно воспользоваться функцией `mfscanf` следующего вида:

```
A=mfscanf(f, s1)
```

Здесь `f` — идентификатор файла, который возвращается функцией `mopen`, `s1` — строка форматов вида

```
%[ширина][.точность]тип
```

Функция `mfscanf` работает следующим образом: из файла с идентификатором `f` считываются в переменную `A` значения в соответствии с форматом `s1`. При чтении числовых значений из текстового файла следует помнить, что два числа считаются разделенными, если между ними есть хотя бы один пробел, символ табуляции или символ перехода на новую строку.

При считывании данных из текстового файла пользователь может следить, достигнут ли конец файла, с помощью функции `meof(f)` (`f` — идентификатор файла), которая возвращает единицу, если достигнут конец файла, и ноль в противном случае.

[1] Параметры в квадратных скобках являются необязательными и могут отсутствовать.

Таблица 9.1. Значение параметров строки преобразования

Параметр	Назначение
Флаг	
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «-»
Пробел	Перед положительным числом выводится пробел, перед отрицательным — «-»
#	Выводится код системы счисления: 0 — перед восьмеричным, 0x (0X) — перед шестнадцатеричным числом.
Ширина	
n	Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции заполняются пробелами.
0n	То же, что и n, но незаполненные позиции заполняются нулями.
Точность	
ничего	Точность по умолчанию
n	Для типов e, E, f выводить n знаков после десятичной точки
Тип	
c	При вводе символьный тип char, при выводе один байт.
d, i	Десятичное со знаком
i	Десятичное со знаком
o	Восьмеричное int unsigned
u	Десятичное без знака
x, X	Шестнадцатеричное int unsigned, при x используются символы a-f, при X — A-F.
f	Значение со знаком вида [-]dddd.dddd
e	Значение со знаком вида [-]d.ddde[+|-]ddd
E	Значение со знаком вида [-]d.ddddE[+|-]ddd
g	Значение со знаком типа e или f в зависимости от значения и точности
G	Значение со знаком типа E или F в зависимости от значения и точности
s	Строка символов
Модификатор (типа)	
h	Для d, i, o, u, x, X короткое целое
l	Для d, i, o, u, x, X длинное целое

Таблица 9.2. Некоторые специальные символы

Символ	Назначение
\b	Сдвиг текущей позиции влево
\n	Перевод строки
\r	Перевод в начало строки, не переходя на новую строку
\t	Горизонтальная табуляция
\’	Символ одинарной кавычки
\’’	Символ двойной кавычки
\?	Символ «?»

9.3.4 Функция закрытия файла mclose

После выполнения всех операций с файлом он должен быть закрыт с помощью функции `mclose` следующей структуры:

```
mclose(f)
```

Здесь `f` — идентификатор закрываемого файла. С помощью функции `mclose('all')` можно закрыть сразу все открытые файлы, кроме стандартных системных файлов.

Пример создания текстового файла приведен на листинге 9.17.

Листинг 9.17. Создание текстового файла

```
//В текстовом файле abc.txt хранятся размеры матрицы N и M
//и сама матрица A(N,M)
//N - количество строк матрицы.
N=3;
//M- количество столбцов матрицы.
M=4;
A=[2 4 6 7; 6 3 2 1; 11 12 34 10];
//Открываем пустой файл abc.txt в режиме записи.
f=mopen('abc.txt','w');
//Записываем в файл abc.txt N и M, разделенные символом
//табуляции.
mfprintf(f,'%d\t%d\n',N,M);
for i=1:N
for j=1:M
//Записываем в файл abc.txt очередной элемент матрицы A.
mfprintf(f,'%g\t',A(i,j));
end
//По окончании записи очередной строки записываем в файл
```

```
//символ <<конец строки>>.
mfprintf(f,'\n');
end
mclose(f);
```

Созданный текстовый файл можно увидеть на рис. 9.3.

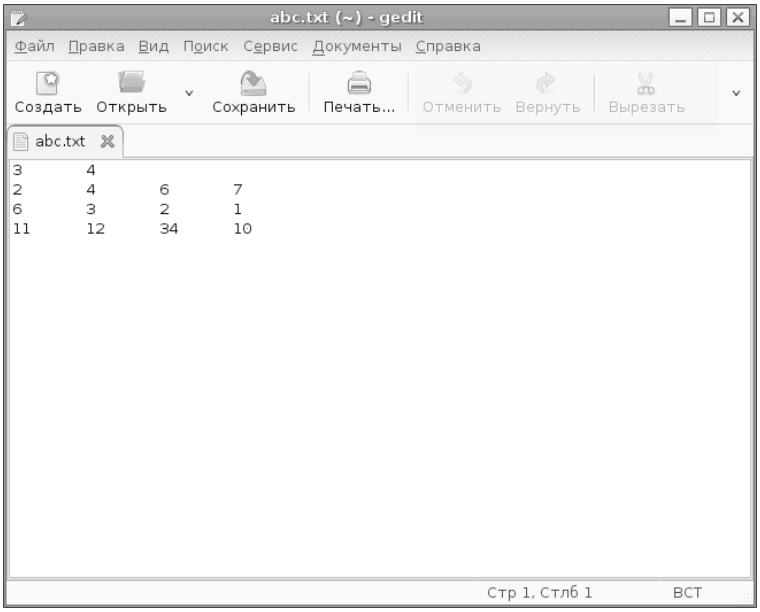

Рис. 9.3. Созданный текстовый файл

Программа чтения данных из этого текстового файла приведена на листинге 9.18.

Листинг 9.18. Чтение из текстового файла

```
f=mopen('abc.txt','r');
N=mfscanf(f,'%d');
M=mfscanf(f,'%d');
for i=1:N
for j=1:M
A(i,j)=mfscanf(f,'%g');
end
end
mclose(f);
```

Результаты работы файла-сценария, представленного на листинге 9.17, можно увидеть на листинге 9.19.

Листинг 9.19. Результат работы файла-сценария

```
N =
3.
M =
4.
A =
! 2. 4. 6. 7. !
! 6. 3. 2. 1. !
! 11. 12. 34. 10. !
```

9.4 Пример программы в Scilab

В качестве примера программы на sci-языке рассмотрим следующую задачу.

Задача 9.3.
Положительные числа из массива Y переписать в массив X, удалить из массива X элементы, меньшие среднего арифметического и расположенные после минимального элемента.

Программа решения задачи представлена на листинге 9.20. Использование переменной `min1` в программе обусловлено тем, что в Scilab есть встроенная функция `min`. Использование переменной с именем `min` не позволит использовать встроенную функцию `min`.

Листинг 9.20. Решение задачи 9.3

```
//Ввод количества элементов в массиве y.
N=input('N=');
disp('Vvod massiva Y');
//Цикл для ввода элементов в массиве y.
for i=1:N
y(i)=input('Y=');
end
disp(y);
//Переменная k содержит количество положительных элементов в
//массиве y, и как следствие, количество элементов в массиве x.
//Первоначально k=0.
k=0;
//Перебираем все элементы в массиве y.
for i=1:N
//Если текущий элемент положителен, то
if y(i)>0
//значение переменной k увеличиваем на 1,
k=k+1;
```

```
//а элемент массива y переписываем в x.
x(k)=y(i);
end;
end;
//вывод массива x.
disp(x);
//В переменной s будем хранить сумму элементов массива x, в
//переменной min1 - минимальное значение в массиве x, в Nmin
//- номер минимального элемента в массиве x.
s=x(1);
min1=x(1);
Nmin=1;
//Цикл для поиска минимума и суммы среди элементов массива x.
for i=2:k
//Накапливание суммы.
s=s+x(i);
//Поиск минимума.
if x(i)<min1
min1=x(i);
Nmin=i;
end;
end;
//Начиная с минимального, перебираем все элементы массива, и
//если
i=Nmin;
while i<=k
//текущий элемент меньше среднего арифметического,
if x(i)<s/k
//то удаляем текущий элемент. Обратите внимание, при удалении
//происходит сдвиг элементов на 1 влево, и поэтому не надо
//увеличивать i на 1 и переходить к следующему элементу после
//удаления.
for j=i:k-1
x(j)=x(j+1);
end;
//Уменьшение количества элементов в массиве на 1.
x(k)=[];
k=k-1;
else
//Если удаление не происходило, то переходим к следующему
//элементу массива.
i=i+1;
end;
end;
disp(x);
```

9.5 Функции в Scilab

В Scilab несложно оформлять собственные функции. Структура функции в Scilab следующая:

```
function [y1,y2,...,yn]=ff(x1,x2,...,xm)
операторы
endfunction
```

Здесь `x1`, `x2`, ..., `xm` — список входных параметров функции; `y1`, `y2`, ..., `yn` — список выходных параметров функции, `ff` — имя функции.

Если вызываемая функция находится не в текущем файле, то перед ее вызовом следует загрузить файл, в котором находится функция с помощью функции `exec` следующей структуры.

```
exec('file',-1).
```

Здесь `file` — имя файла на диске, в котором находится вызываемая функция.

Рассмотрим пример с использованием функций и файлов.

Задача 9.4.
В матрице $A(N, N)$ найти сумму элементов, расположенных на диагоналях матрицы, вычислить количество элементов, равных максимальному элементу матрицы. Число N и матрица A хранятся в текстовом файле *primer.txt* (см. рис. 9.4).

Для нахождения суммы (`summa`), максимума (`maximum`) и количества максимумов (`kolichestvo`) оформим функцию, заголовок ее будет иметь вид:

```
function [summa, maximum, kolichestvo]=matrica_A(N,N)
```

Текст функции приведен на листинге 9.21.

Листинг 9.21. Текст функции `matrica_A`

```
function [summa, maximum, kolichestvo]=matrica_A(A,N)
summa=0;
for i=1:N
//Суммируем элементы, расположенные на главной A(i,i) и
//побочной A(i,N+1-i) диагоналях
summa=summa+A(i,i)+A(i,N+1-i);
end
//Если количество строк в матрице нечетное, то есть элемент,
//который расположен одновременно на главной и побочной
//диагоналях, мы его просуммировали дважды: и как элемент на
//главной, и как элемент побочной диагонали, поэтому его надо
//вычесть из суммы.
```

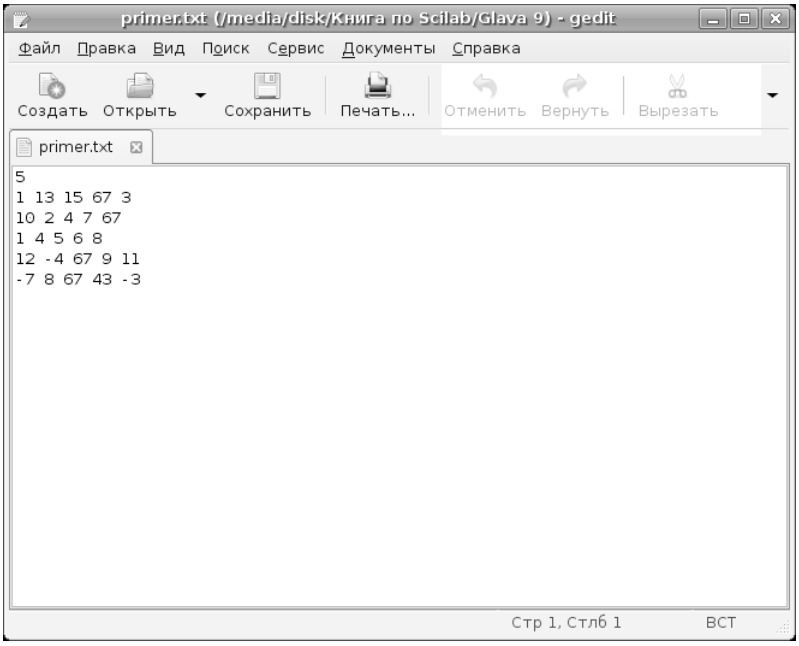

Рис. 9.4. Текстовый файл *primer.txt*

```
if (N-int(N/2)*2)==1
summa=summa-A(int(N/2)*2+1,int(N/2)*2+1);
end
//Элемент A(1,1) объявляем максимальным, количество максимумов
//равно 1.
maximum=A(1,1);kolichestvo=1;
for i=1:N
for j=1:N
//Если текущий элемент A(i,j) больше максимального, то его и
//объявляем максимальным, количество максимумов равно 1.
if A(i,j)>maximum
maximum=A(i,j);
kolichestvo=1;
//Если текущий элемент равен максимальному, то количество
//максимумов увеличиваем на 1.
elseif A(i,j)==maximum
kolichestvo=kolichestvo+1;
end
end
end
endfunction
```

Текст функции, предназначенной для чтения матрицы из файла и вызова функции `matrica_A`, приведен на листинге 9.22, результаты — на листинге 9.23.

Листинг 9.22. Вызов функции `matrica_A`

```
f=mopen('G:\primer.txt','r');
N=mfscanf(f,'%d');
for i=1:N
for j=1:N
B(i,j)=mfscanf(f,'%g');
end
end
mclose(f);
[s,m,k]=matrica_A(B,N);
```

Листинг 9.23. Результаты работы программы

```
-->exec("matrica_2.sci");
-->s
 s =
21.
-->m
 m =
67.
-->k
k =
4.
```

В этой части были рассмотрены основные возможности программирования в Scilab. В следующей главе будут рассмотрены возможности построения визуальных программ.

Глава 10

Создание графических приложений в среде Scilab

Scilab позволяет создавать не только обычные программы для автоматизации расчетов, но и визуальные приложения, которые будут запускаться в среде Scilab. Основным объектом в среде Scilab является графическое окно.

10.1 Работа с графическим окном

Для создания пустого графического окна служит функция `figure`.

```
F=figure();
```

В результате выполнения этой команды будет создано графическое окно с именем *objfigure1*. По умолчанию первое окно получает имя *objfigure1*, второе — *objfigure2* и т. д. Указатель на графическое окно[1] записывается в переменную `F`. Размер и положение окна на экране компьютера можно задавать с помощью параметра

```
'position',[x y dx dy],
```

где `x, y` — положение верхнего левого угла окна (по горизонтали и вертикали соответственно) относительно верхнего левого угла экрана;

`dx` — размер окна по горизонтали (ширина окна) в пикселях;

`dy` — размер окна по вертикали (высота окна) в пикселях.

[1] Под указателем мы будем понимать переменную, в которой хранится адрес окна или другого объекта.

Параметры окна можно задавать одним из двух способов.

1. Непосредственно при создании графического окна задаются его параметры. В этом случае обращение к функции figure имеет вид

```
F=figure('Свойство1', 'Значение1', 'Свойство2',
'Значение2', ..., 'Свойствоn', 'Значениеn')
```

здесь `'Свойство1'` — название первого параметра, `Значение1` — его значение, `'Свойство2'` — название второго параметра, `Значение2`[1] — значение второго параметра и т. д.

Например, с помощью команды

```
F=figure('position', [10 100 300 200]);
```

будет создано окно, представленное на рис. 10.1.

Рис. 10.1. Первое графическое окно

2. После создания графического окна с помощью функции

```
set(f,'Свойство','Значение')
```

устанавливается значение параметров; здесь `f` — указатель на графическое окно, `'Свойство'` — имя параметра, `'Значение'` — его значение.

Следующие две строки задают месторасположение и размер окна (рис. 10.2).

```
f=figure();
set(f,'position',[20,40,600,450])
```

[1] `Значениеi` будет использоваться в кавычках, если значением параметра является строка, если же значением параметра является число, то кавычки использовать не надо.

Рис. 10.2. Графическое окно определенного размера и месторасположения

Для изменения заголовка окна используется параметр `'figure_name', 'name'`, определяющий заголовок окна (`'name'`). На листинге 10.1 приведен пример создания окна с именем *FIRST WINDOWS* (см. рис. 10.3).

Листинг 10.1. Создание окна с именем *FIRST WINDOW*

```
f=figure();
set(f,'position',[20,40,600,450]);
set(f,'figure_name','FIRST WINDOW');
```

Это же окно можно получить с помощью одной строки

```
f=figure('position',[20,40,600,450],'figure_name','FIRST WINDOW');
```

Графическое окно можно закрыть с помощью функции

```
close(f),
```

здесь `f` — указатель на окно.

Удаляется окно с помощью функции

```
delete(f),
```

где `f` — указатель на окно.

Рис. 10.3. Окно с заголовком *FIRST WINDOW*

10.2 Динамическое создание интерфейсных элементов. Описание основных функций

В Scilab используется динамический способ создания интерфейсных компонентов. Он заключается в том, что на стадии выполнения программы могут создаваться (и удаляться) те или иные элементы управления (кнопки, метки, флажки и т. д.) и их свойствам присваиваются соответствующие значения.

Для создания любого интерфейсного компонента с заданными свойствами используется функция `uicontrol`, возвращающая указатель на формируемый компонент:

```
C=uicontrol(F, 'Style', 'тип_компонента', 'Свойство_1',
Значение_1, 'Свойство_2', Значение_2,...,
'Свойство_k',Значение_k);
```

Здесь `C` — указатель на создаваемый компонент;

`F` — указатель на объект, внутри которого будет создаваться компонент (чаще всего этим компонентом будет окно); первый аргумент функции `uicontrol` не является обязательным, и если он отсутствует, то родителем (владельцем) создаваемого компонента является текущий графический объект — текущее графическое окно;

`'Style'` — служебная строка `Style`, указывает на стиль создаваемого компонента (символьное имя);

`'тип_компонента'` — определяет, к какому классу принадлежит создаваемый компонент, это может быть `PushButton`, `Radiobutton`, `Edit`, `StaticText`, `Slider`, `Panel`, `Button Group`, `Listbox` или другие компоненты, это свойство будет указываться для каждого из компонентов;

`'Свойство_k', Значение_k` — определяют свойства и значения отдельных компонентов, они будут описаны ниже конкретно для каждого компонента.

У существующего интерфейсного объекта можно изменить те или иные свойства с помощью функции `set`:

```
set(C,'Свойство_1',Значение_1, 'Свойство_2', Значение_2, ...,
'Свойство_k', Значение_k)
```

Здесь `C` — указатель на динамический компонент, параметры которого будут меняться. `C` может быть и вектором динамических элементов, в этом случае функция `set` будет задавать значения свойств для всех объектов `C(i)`;

`'Свойство_k', Значение_k` — изменяемые параметры и их значения.

Получить значение параметра компонентов можно с помощью функции `get` следующей структуры:

```
get(C,'Свойство')
```

Здесь `C` — указатель на динамический интерфейсный компонент, значение параметра которого необходимо узнать;

`'Свойство'` — имя параметра, значение которого нужно узнать.

Функция возвращает значение параметра.

Далее мы поговорим об особенностях создания различных компонентов.

10.2.1 Командная кнопка

Командная кнопка типа *PushButton* создается с помощью функции `uicontrol`, в которой параметру `'Style'` необходимо присвоить значение `'pushbutton'`. По умолчанию она не снабжается никакой надписью, имеет серый цвет и располагается в левом нижнем углу фигуры, Надпись на кнопке можно установить с помощью свойства `String` (см. листинг 10.2 и рис. 10.4).

Листинг 10.2. Создание окна с кнопкой

```
//Создаем окно
d=figure();
//Создаем кнопку, устанавливая свойство Style.
dbt=uicontrol(d,'Style','pushbutton');
//Изменяем надпись YES на кнопке
set(dbt,'String','YES');
```

Рис. 10.4. Кнопка *YES* в окне

Модифицируем программу создания кнопки, задав дополнительно значения некоторых свойств:

- месторасположение и заголовок окна;
- надпись на кнопке;
- месторасположение кнопки.

Текст программы приведен на листинге 10.3, а на рис. 10.5 можно увидеть окно, которое получилось в результате работы этой программы.

Листинг 10.3. Определение свойств кнопки

```
//Создаем окно.
f=figure();
//Определяем месторасположение окна.
set(f,'position',[0,0,250,100])
//Определяем имя (заголовок) окна.
set(f,'figure_name','Окно с кнопкой');
//Создаем кнопку (style - pushbutton), надпись на кнопке -
//Кнопка, позиция кнопки определяется параметром position.
Button=uicontrol('style','pushbutton','string','Кнопка',...
'position',[50,50,100,20]);
```

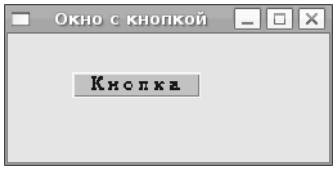

Рис. 10.5. Окно с кнопкой

Теперь при щелчке на кнопке вокруг ее надписи появляется пунктирный прямоугольник, свидетельствующий о том, что кнопка «находится в фокусе». Щелчок за пределами поверхности кнопки выведет ее из фокуса, и пунктирная рамка пропадет. Главным назначением командной кнопки является вызов функции, реагирующей на щелчок по кнопке.

Щелчок по кнопке генерирует событие `CallBack`, которое указывается как параметр функции `uicontrol`. Значением параметра `CallBack` является строка с именем функции, вызываемой при щелчке по кнопке. В этом случае функция `uicontrol` становится такой

```
Button=uicontrol('style', 'pushbutton', 'string', 'Button',
'CallBack', 'Function');
```

Здесь `Function` — имя вызываемой при наступлении события `CallBack` функции.

В качестве примера рассмотрим окно с кнопкой, при щелчке по которой появляется окно с графиком функции $y = sin(x)$ (см. листинг 10.4). После запуска этой программы появится окно, представленное на рис. 10.6, при щелчке по кнопке *Button* вызывается обработчик события — функция `gr_sin`, в результате появляется окно, изображенное на рис. 10.7.

Листинг 10.4. Пример кнопки с обработчиком события `CallBack`

```
f=figure();
set(f,'position',[0,0,250,100])
set(f,'figure_name','Grafik');
//Создаем кнопку, которая при щелчке по ней мышкой вызывает
//функцию gr_sin.
Button=uicontrol('style','pushbutton','string','Button',...
 'position',[50,50,100,20],'CallBack','gr_sin');
function y=gr_sin()
x=-5:0.2:5;
y=sin(x);
plot(x,y);
xgrid();
endfunction
```

Рис. 10.6. Окно программы

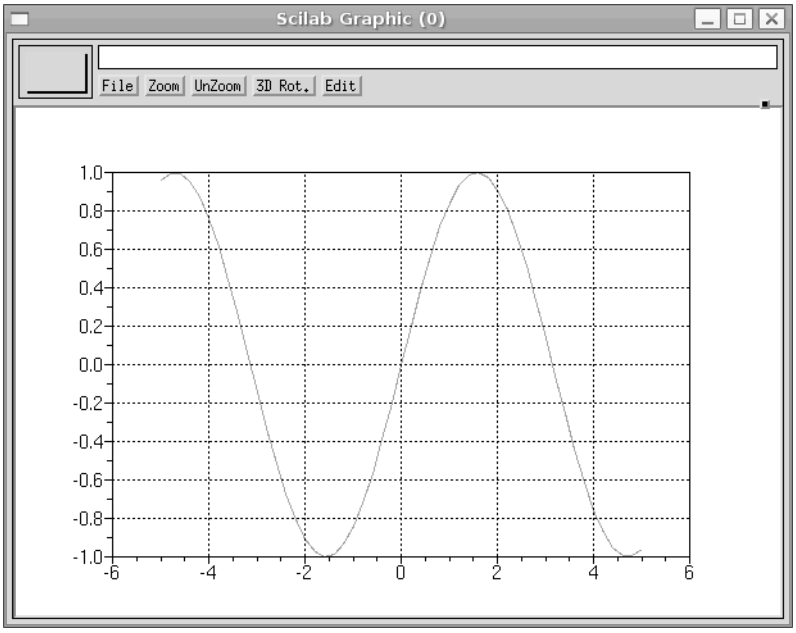

Рис. 10.7. График функции $y = sin(x)$

10.2.2 Метка

Следующим наиболее часто используемым компонентом является метка — текстовое поле для отображения символьной информации. Для определения метки значения параметра `'Style'` в функции `uicontrol` должно иметь значение `'text'`. Компонент предназначен для вывода символьной строки (или нескольких строк). Выводимый на метку текст — значение параметра `'String'` — может быть изменен только из программы.

Рассмотрим пример создания текстового поля (метки) с помощью функции `uicontrol` (см. листинг 10.5 и рис. 10.8):

Листинг 10.5. Создание метки

```
f=figure();
uicontrol('Style','text','Position',[10,130,150,20],'String',...
 'Metka');
```

Рис. 10.8. Окно с меткой

Одним из основных свойств метки является горизонтальное выравнивание текста, которое определяется свойством `HorizontalAlignment`. Это свойство может принимать одно из следующих значений:

`left` — выравнивание текста по левому краю;

`center` — выравнивание текста по центру (значение по умолчанию);

`right` — выравнивание по правому краю.

В качестве примера рассмотрим окно, содержащее 4 текстовых поля с разными значениями свойства `HorizontalAlignment`. Текст программы представлен в листинге 10.6, а окно с четырьмя метками — на рис. 10.9.

Листинг 10.6. Создание нескольких меток

```
hFig=figure();
set(hFig,'Position',[50,50,300,200]);
hSt1=uicontrol('Style','text','Position',[30,30,150,20],...
'String', 'Metka 1');
set(hSt1,'BackgroundColor',[1 1 1]);
set(hSt1,'HorizontalAlignment','left');
hSt2=uicontrol('Style', 'text', 'Position', [30,60,150,20],...
'HorizontalAlignment', 'center', 'BackgroundColor', [1 1 1],...
'String', 'Metka 2');
hSt3=uicontrol('Style','text','Position',[30,90,150,20],...
'HorizontalAlignment','right','BackgroundColor',[1 1 1],...
```

```
'String','Metka 3');
hSt4=uicontrol('Style','text','Position',[30,120,150,20],...
'BackgroundColor',[1 1 1],'String','Metka 4');
```

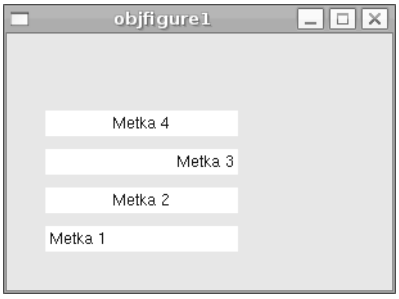

Рис. 10.9. Окно с несколькими метками

10.2.3 Компоненты Переключатель и Флажок

Рассмотрим еще два компонента — переключатель и флажок, которые позволяют переключаться между состояниями или выключать одно из свойств.

У флажка свойство `'Style'` принимает значение `'checkbox'`, у переключателя свойство `'Style'` должно быть установлено в `'radiobutton'`.

Индикатором альтернативных комбинаций является переключатель (Radiobutton), который также создается с помощью функции `uicontrol`. Пример создания переключателя представлен на листинге 10.7 и рис. 10.10.

Листинг 10.7. Создание переключателя

```
hFig=figure();
R=uicontrol('Style','radiobutton','String','name','value',1,...,
 'Position', [25,150,70,30]);
```

При создании переключателя должно быть задано его состояние (параметр `'value'`), переключатель может быть активен (значение `'value'` равно `1`) или нет (значение `'value'` равно `0`). Задать значение свойства `'value'` можно также и с помощью функции `set`. Например:

```
set(Rb,'value',0)
```

Получить значение свойства 'value' можно с помощью функции `get`.

Переключатель может реагировать на событие `'CallBack'`, и вызывать на выполнение определенную функцию. В этом случае создать кнопку можно с помощью вызова следующей функции `uicontrol`:

Рис. 10.10. Окно с переключателем

```
r1=uicontrol('Style','radiobutton','String','sin(x)','value',
0,'CallBack','F1');
```

Здесь `F1` — имя функции, которая будет вызываться при щечке по переключателю. Однако, когда Вы будете писать функцию — обработчик события `'CallBack'` переключателя, то следует помнить, что при щелчке по переключателю автоматически происходит смена его состояния.

Проиллюстрируем применение переключателей на примере программы, в которой с помощью переключателя можно выбрать функцию, график которой будет воспроизводиться в отдельном графическом окне при щелчке по кнопке *Plot* (см. листинг 10.8 и рис. 10.11).

Листинг 10.8. Пример работы с переключателями

```
//Создаем графическое окно.
hFig=figure('Position',[50,50,200,200]);
//Создание переключателей
hRb1=uicontrol('Style','radiobutton','String','sin(x)',...
'value',1, 'Position',[25,100,60,20]);
hRb2=uicontrol('Style','radiobutton','String','cos(x)',...
'value',1, 'Position',[25,140,60,20]);
//Создаем кнопку с именем Plot, которая с помощью обработчика
//Radio строит график функции в соответствии с положением
//переключателей.
Button=uicontrol('style','pushbutton','string','Plot',...
'position',[20,50,80,20],'CallBack','Radio');
//Создаем кнопку с именем Close, которая с помощью обработчика
//Final закрывает окно.
Button1=uicontrol('style','pushbutton','string','Close',...
'position',[20,25,80,20],'CallBack','Final');
//Функция Radio, реагирующая на щелчок по кнопке
function Radio()
```

```
newaxes;
x=-2*%pi:0.1:2*%pi;
if get(hRb1,'value')==1 //Если активна первая кнопка,
y=sin(x);
plot(x,y,'-r'); //то построение синусоиды
xgrid();
end;
if get(hRb2,'value')==1 //Если активна вторая кнопка,
y=cos(x);
plot(x,y,'-b'); //то построение косинусоиды
xgrid(); //Нанесение сетки на график
end;
endfunction
//Функция, отвечающая за кнопку Close и закрывающая окно.
function Final()
close(hFig);
endfunction
```

Рис. 10.11. Окно приложения

На рис. 10.11 представлено окно приложения. При щелчке по кнопке *Plot* будет создано диалоговое окно, в котором будут изображены графики выбранных с помощью переключателей функций (см. рис. 10.12). Если ни одна из функций не выбрана, то при щелчке по кнопке *Plot* ничего не происходит. Кнопка *Close* закрывает приложение. Изменение состояния переключателей происходит автоматически при щелчке по ним.

Компонент «флажок» используется для индикации неальтернативных комбинаций. Генерация события `'CallBack'` и автоматическое выделение кнопки происходят при щелчке на квадратике или сопровождающей его надписи. Если флажок включен, то значение свойства `'value'` равно 1. Щелчок по флажку автоматически изменяет состояние на противоположное. Использование флажка аналогично переключателю.

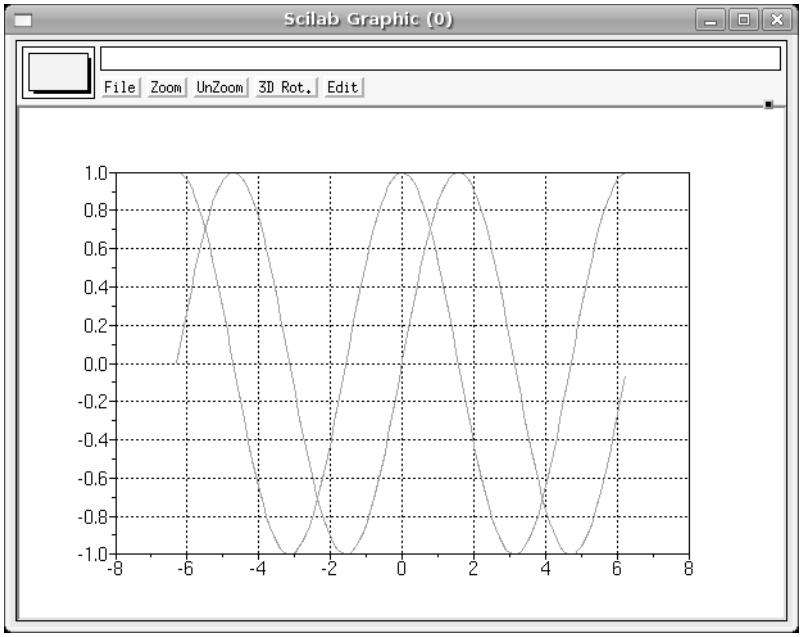

Рис. 10.12. Построенные в отдельном окне графики функций

10.2.4 Компонент окно редактирования

Интерфейсный элемент окно редактирования (у того компонента свойство `'Style'` должно принимать значение `'edit'`) может использоваться для ввода и вывода символьной информации. Текст, набираемый в окне редактирования, можно корректировать. При работе с компонентом можно использовать операции с буфером обмена. Процедура ввода, завершаемая нажатием клавиши *Enter*, генерирует событие `CallBack`.

Строка ввода определяется параметром `'String'`, который определяет находящийся в компоненте текст. Для нормального функционирования компонента этот параметр необходимо обязательно задавать при определении компонента с помощью функции `uicontrol`. Изменить значение этого свойства можно с помощью функции `set`, а считать его значение — с помощью функции `get`.

Вводимый текст может быть прижат к левому или правому краю окна ввода, если задать соответствующее значение свойства `HorizontalAlignment` (по аналогии с компонентом «Метка»). Если вводимый текст представляет собой числовое значение, которое должно быть использовано в работе программы, то содержимое свойства `'String'` переводится в числовой формат с помощью функции `eval` (можно было воспользоваться и функцией `evstr`) (будет рассмотрено далее на примере квадратного уравнения).

В качестве примера работы с несколькими компонентами рассмотрим следующую задачу.

Задача 10.1.
Написать программу решения квадратного или биквадратного уравнения. Выбор типа уравнения будем проводить с помощью компонента «Переключатель».

Программа с комментариями представлена на листинге 10.9.

Листинг 10.9. Решения квадратного или биквадратного уравнения

```
f=figure(); //Создание графического объекта.
//Устанавливаем размер окна.
set(f,'position',[0,0,700,300])
//Устанавливаем заголовок окна.
set(f,'figure_name','УРАВНЕНИЕ');
//Создание текстовых полей для подписей полей ввода
//коэффициентов.
//Подпись A=.
lab_a=uicontrol(f,'style','text','string','A=','position',...
[50, 250, 100, 20]);
//Подпись B=.
lab_b=uicontrol(f,'style','text','string','B=','position',...
[150, 250, 100, 20]);
//Подпись C=.
lab_c=uicontrol(f,'style','text','string','C=','position',...
[250, 250, 100, 20]);
//Поле редактирования для ввода коэффициента a.
edit_a=uicontrol(f,'style','edit','string','1','position',...
[50, 230, 100, 20]);
//Поле редактирования для ввода коэффициента b.
edit_b=uicontrol(f,'style','edit','string','2','position',...
[150, 230, 100, 20]);
//Поле редактирования для ввода коэффициента c.
edit_c=uicontrol(f,'style','edit','string','1','position',...
[250, 230, 100, 20]);
//Текстовое поле, определяющее вывод результатов.
textresult=uicontrol(f,'style','text','string','','position',...
[5, 80, 650, 20]);
//Флажок, отвечающий за выбор типа уравнения.
radio_bikv=uicontrol('style','radiobutton','string',...
'Биквадратное уравнение?', 'value',1,'position',...
[100,100,300,20]);
```

```
BtSolve=uicontrol('style','pushbutton','string','Решить',...
'CallBack', 'Solve','position',[50,50,120,20]);
BtClose=uicontrol('style','pushbutton','string','Закрыть',...
'CallBack', '_Close','position',[300,50,120,20]);
//Функция решения уравнения.
function Solve()
//Считываем значение переменных из текстовых полей и
//преобразовываем их к числовому типу.
a=eval(get(edit_a,'string'));
b=eval(get(edit_b,'string'));
c=eval(get(edit_c,'string'));
d=b*b-4*a*c;
//Проверяем значение флажка, если флажок выключен,
if get(radio_bikv,'value')==0
//то решаем квадратное уравнение,
if d<0
set(textresult,'string','Нет решения квадратного уравнения');
else
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
set(textresult,'string',sprintf
("2 корня квадратного уравнения\t x1=%1.2f\tx2=%1.2f",x1,x2));
end;
//если флажок включен,
else
//то решаем биквадратное уравнение.
if d<0
set(textresult,'string','Нет решения биквадратного уравнения');
else
  y1=(-b+sqrt(d))/2/a;
  y2=(-b-sqrt(d))/2/a;
  if(y1<0)&(y2<0)
set(textresult,'string','Нет решения биквадратного уравнения');
 elseif (y1>=0)&(y2>=0)
 x1=sqrt(y1);x2=-x1;x3=sqrt(y2);x4=-x3;
set(textresult,'string',sprintf("4 корня биквадратного...
уравнения \t x1=%1.2f\tx2=%1.2f\tx3=%1.2f\tx4=%1.2f",...
x1,x2,x3,x4));
 else
 if y1>=0
    x1=sqrt(y1);x2=-x1;
 else
    x1=sqrt(y2);x2=-x1;
 end;
```

```
 set(textresult,'string',sprintf
("2корня биквадратного уравнения\t x1=%1.2f\tx2=%1.2f",x1,x2));
 end;
end;
end
endfunction
// Функция закрытия окна.
function _Close()
close(f)
endfunction
```

На рисунке 10.13 представлено окно программы.

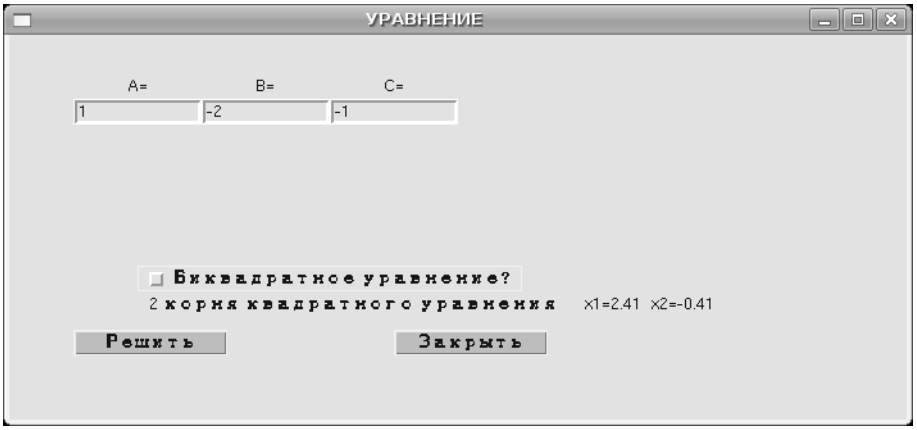

Рис. 10.13. Окно программы решения уравнения

Авторы рекомендуют читателю разобраться с этой несложной программой. Разобравшись с ней, Вы сможете понять механизм взаимодействия стандартных компонентов Scilab и обработчиков событий, чтобы затем использовать эти знания при разработке собственных визуальных приложений.

10.2.5 Списки строк

Интерфейсный компонент «список строк» в простейшем случае можно рассматривать как окно с массивом строк в нем. Если длина списка превышает высоту окна, то для перемещения по списку может использоваться вертикальная полоса прокрутки, которая генерируется автоматически.

Создание списка строк производится с помощью функции `uicontrol` при задании параметра `'Style'` — `'listbox'`. Рассмотрим это на простом примере (см. листинг 10.10 и рис. 10.14).

Листинг 10.10. Создание списка

```
//Создание графического окна.
f=figure();
//Создание  listbox
h=uicontrol(f,'style','listbox','position', [10 10 150 160]);
//Заполнение списка.
set(h, 'string', "строка 1|строка 2|строка 3");
set(h, 'value', [1 3]);
//Выделение item 1 и 3 в списке.
```

Рис. 10.14. Список с выделенными элементами

Список позволяет пользователю выбрать одну или несколько строк и в зависимости от выбора произвести то или иное действие.

Выбор строки осуществляется щелчком левой кнопки мыши в тот момент, когда курсор указывает на выбираемую строку. Одновременно с подсветкой строки ее номер заносится в свойство `'value'` и генерируется событие `'CallBack'`. Строки в списке нумеруются от 1. Для выбора разрозненных строк нужно зажать клавишу *Ctrl* и щелкать мышью по выделяемым строкам. При этом каждая выделяемая строка подсвечивается, а ее номер запоминается в векторе `'value'`. Для выбора группы подряд идущих строк можно нажать и удерживать клавишу *Shift*, а затем щелкнуть по первой и последней строке группы. Все промежуточные строки тоже будут выделены и все их номера запомнятся в векторе `'value'`.

С помощью рассмотренных в этой главе компонентов и встроенных функций Scilab можно создавать визуальные программы для решения инженерных и математических задая любой сложности.

Глава 11

Обработка экспериментальных данных

11.1 Метод наименьших квадратов

Метод наименьших квадратов позволяет по экспериментальным данным подобрать такую аналитическую функцию, которая проходит настолько близко к экспериментальным точкам, насколько это возможно.

Пусть в результате эксперимента были получены некоторые данные, отображенные в виде таблицы (табл. 11.1). Требуется построить аналитическую зависимость, наиболее точно описывающую результаты эксперимента.

Таблица 11.1. Экспериментальные данные

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\ldots$	x_n
y_i	y_1	y_2	y_3	y_4	y_5	y_6	y_7	$\ldots$	y_n

Идея метода наименьших квадратов заключается в том, что функцию $Y = f(x, a_0, a_1, \ldots, a_k)$ необходимо подобрать таким образом, чтобы сумма квадратов отклонений измеренных значений y_i от расчетных Y_i была наименьшей:

$$S = \sum_{i=1}^{n} (y_i - f(x_i, a_0, a_1, \ldots, a_k))^2 \to \min \tag{11.1}$$

Задача сводится к определению коэффициентов a_i из условия (11.1). Для реализации этой задачи в Scilab предусмотрена функция

```
[a,S]=datafit(F,z,c)
```

где `F` — функция, параметры которой необходимо подобрать; `z` — матрица исходных данных; `c` — вектор начальных приближений; `a` — вектор коэффициентов; `S` — сумма квадратов отклонений измеренных значений от расчетных.

Рассмотрим использование функции `datafit` на примере.

Задача 11.1.
В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P, Вт) от входного напряжения (U, В) для асинхронного двигателя (табл. 11.2).

Таблица 11.2. Зависимость потребляемой из сети мощности от входного напряжения.

U, В	132	140	150	162	170	180	190	200	211	220	232	240	251
P, Вт	330	350	385	425	450	485	540	600	660	730	920	1020	1350

Методом наименьших квадратов подобрать зависимость вида $P = a_1 + a_2U + a_3U^2 + a_4U^3$.

Далее приведено решение задачи с комментариями.

Листинг 11.1. Решение задачи 11.1

```
//Функция, вычисляющая разность между экспериментальными
//и теоретическими значениями.
//Перед использованием необходимо определить
//z=[x;y] - матрицу исходных данных - и
//c - вектор начальных значений коэффициентов,
//размерность вектора должна совпадать
//с количеством искомых коэффициентов.
function [zr]=G(c,z)
zr=z(2)-c(1)-c(2)*z(1)-c(3)*z(1)^2-c(4)*z(1)^3
endfunction
//Исходные данные
x=[1.32 1.40 1.50 1.62 1.70 1.80 1.90...
   2.00,2.11,2.20,2.32,2.40,2.51];
y=[3.30 3.50 3.85 4.25 4.50 4.85 5.40...
   6.00 6.60 7.30 9.20 10.20 13.50];
//Формирование матрицы исходных данных
z=[x;y];
//Вектор начальных приближений
c=[0;0;0;0];
//Решение задачи
```

```
[a,err]=datafit(G,z,c)
 S  =
    0.5287901
 a  =
  - 51.576664
    95.594671
  - 55.695312
    11.111453
```

Итак, в результате работы функции `datafit` была подобрана аналитическая зависимость вида $P = -51.577 + 95.595U - 55.695U^2 + 11.111U^3$, а сумма квадратов отклонений измеренных значений от расчетных составила 0.529.

Геометрическая интерпретация задачи (рис. 11.1):

Листинг 11.2. Построение графика подобранной зависимости

```
//Построение графика экспериментальных данных
plot2d(x,y,-4);
//Построение графика подобранной функции
t=1.32:0.01:2.51;
Ptc=a(1)+a(2)*t+a(3)*t^2+a(4)*t^3;
plot2d(t,Ptc);
```

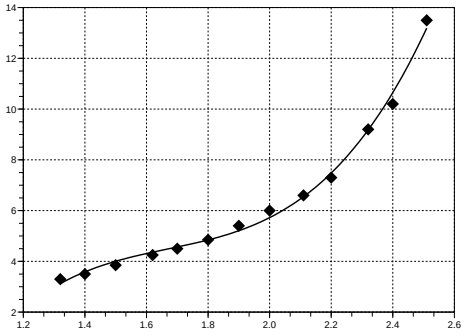

Рис. 11.1. Графическая интерпретация задачи 11.1

Задача 11.2.
К некоторым экспериментальным данным подобрать зависимость вида $Y = a_1 \cdot x^{a_2} + a_3$.

Далее представлено решение задачи и ее графическая интерпретация (рис. 11.2).

Листинг 11.3. Решение задачи 11.2

```
function [zr]=F(c,z)
zr=z(2)-c(1)*z(1)^c(2)-c(3);
endfunction
x=[10.1,10.2,10.3,10.8,10.9,11,11.1,11.4,12.2,13.3,13.8,...
      14,14.4,14.5,15,15.6,15.8,17,18.1,19];
y=[24,36,26,45,34,37,55,51,75,84,74,91,85,87,...
   94,92,96,97,98,99];
z=[x;y];
c=[0;0;0];
[a,S]=datafit(F,z,c);
t=10:0.01:19;
Yt=a(1)*t^a(2)+*a*(3);
plot2d(x,y,-3);
plot2d(t,Yt);
```

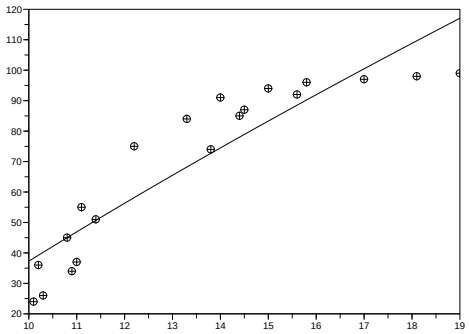

Рис. 11.2. Графическое решение задачи 11.2

Одной из наиболее часто используемых в методе наименьших квадратов функций является прямая, описываемая уравнением вида $y = a_1 + a_2x$, которая называется *линией регрессии* y на x. Параметры a_1 и a_2 являются *коэффициентами регрессии*. Показатель, характеризующий тесноту линейной связи между x и y, называемый *коэффициентом корреляции*, рассчитывается по формуле:

$$r = \frac{\sum_{i=1}^{n}(x_i - M_x)\cdot(y_i - M_y)}{\sqrt{\sum_{i=1}^{n}(x_i - M_x)^2 \sum_{i=1}^{n}(y_i - M_y)^2}}, \qquad Mx = \frac{\sum_{i=1}^{n} x_i}{n}, \qquad My = \frac{\sum_{i=1}^{n} y_i}{n} \tag{11.2}$$

Значение коэффициента корреляции удовлетворяет соотношению $-1 \leqslant r \leqslant 1$. Чем меньше отличается абсолютная величина r от единицы, тем ближе к линии регрессии располагаются экспериментальные точки. Если коэффициент корреляции близок к нулю, то это означает, что между x и y отсутствует линейная связь, но может существовать другая, нелинейная, зависимость.

Аналогом коэффициента корреляции r для нелинейных зависимостей является *индекс корреляции*, рассчитываемый по формуле:

$$R = \sqrt{1 - \frac{\sum_{i=1}^{n}(y_i - Y_i)^2}{\sum_{i=1}^{n}(y_i - M_y)^2}} \tag{11.3}$$

где y — экспериментальные значения, Y — значения, найденные методом наименьших квадратов, M_y — среднее значение y. Индекс корреляции по своему абсолютному значению колеблется в пределах от 0 до 1. При функциональной зависимости индекс корреляции равен 1. В случае отсутствии связи $R = 0$. Если коэффициент корреляции r является мерой тесноты связи только для линейной формы, то индекс корреляции R — и для линейной, и для криволинейной. При прямолинейной связи коэффициент корреляции по своей абсолютной величине равен индексу корреляции.

Для расчета *коэффициентов регрессии* в Scilab предназначена функция

```
a=regress(x,y)
```

где `x` и `y` — экспериментальные данные, `a` — вектор коэффициентов линии регрессии a_1 и a_2.

Рассмотрим работу этой функции на примере.

Задача 11.3.
В «Основах химии» Д. И. Менделеева приводятся данные о растворимости азотнокислого натрия $NaNO_3$ в зависимости от температуры воды. Число условных частей $NaNO_3$, растворяющихся в 100 частях воды при соответствующих температурах, представлено в табл. 11.3. Требуется определить растворимость азотнокислого натрия при температуре 32 градуса в случае линейной зависимости и найти коэффициент и индекс корреляции.

Таблица 11.3. Данные о растворимости $NaNO_3$ в зависимости от температуры воды.

0	4	10	15	21	29	36	51	68
66.7	71.0	76.3	80.6	85.7	92.9	99.4	113.6	125.1

Решение задачи:

Листинг 11.4. Решение задачи 11.3

```
//Экспериментальные данные
x=[0 4 10 15 21 29 36 51 68];
y=[66.7 71 76.3 80.6 85.7 92.9 99.4 113.6 125.1];
//Расчет коэффициентов регрессии
a=regress(x,y)
 a  =
    67.507794
    0.8706404
//Растворимость азотного натрия при температуре 32 градуса
-->t=32;a(1)+a(2)*t
 ans  = 95.368287
//Коэффициент корреляции (11.2)
r=sum((x-mean(x)).*(y-mean(y)))/...
  sqrt(sum((x-mean(x))^2)*sum((y-mean(y))^2))
 r = 0.9989549
//Индекс корреляции (11.3)
R=sqrt(1-sum((y-(a(1)+a(2)*x))^2)/sum((y-mean(y))^2))
 R = 0.9989549
```

Построение графика экспериментальных данных и линии регрессии (рис. 11.3):

Листинг 11.5. График подобранной линии регрессии

```
t=0:70; Yt=a(1)+a(2)*t;
plot2d(x,y,-5); plot2d(t,Yt);
```

Для расчета коэффициента корреляции в Scilab также предназначена встроенная функция `a=corr(x,y)`, где `x` и `y` — экспериментальные данные.

11.2 Интерполяция функций

Простейшая *задача интерполирования* заключается в следующем. На отрезке $[a;b]$ заданы точки $x_0, x_1, x_2, \ldots, x_n$ (всего $n+1$ точка), которые называют узлами

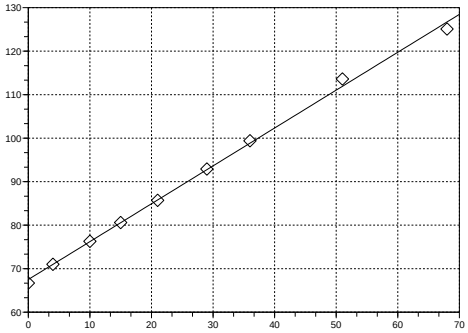

Рис. 11.3. Графическое решение задачи 11.3

интерполяции, и значения некоторой функции $f(x)$ в этих точках:

$$f(x_0) = y_0, \quad f(x_1) = y_1, \quad f(x_2) = y_2, \quad \ldots, \quad f(x_n) = y_n\,. \tag{11.4}$$

Требуется построить интерполирующую функцию $F(x)$, принадлежащую известному классу и принимающую в узлах интерполяции те же значения, что и $f(x)$:

$$F(x_0) = y_0, \quad F(x_1) = y_1, \quad F(x_2) = y_2, \quad \ldots, F(x_n) = y_n\,. \tag{11.5}$$

Для решения подобной задачи довольно часто используют сплайн-интерполяцию (от английского слова *spline* — рейка, линейка). Один из наиболее распространенных вариантов интерполяции — интерполяция кубическими сплайнами. Кроме того, существуют квадратичные и линейные сплайны.

В Scilab для построения *линейной интерполяции* служит функция

```
y=interpln(z,x)
```

где `z` — матрица исходных драных; `x` — вектор абсцисс; `y` — вектор значений линейного сплайна в точка `x`.

Далее приведен пример использования функции `interpln`. Здесь линейный сплайн применяется для решения задачи 11.1. Графическое решение задачи показано на рис. 11.5.

Листинг 11.6. Использование функции `interpln`

```
x=[132 140 150 162 170 180 190 200 211 220 232 240 251];
y=[330 350 385 425 450 485 540 600 660 730 920 1020 1350];
plot2d(x,y,-4);
z=[x;y];
```

```
t=132:5:252; ptd=interpln(z,t);
plot2d(t,ptd);
```

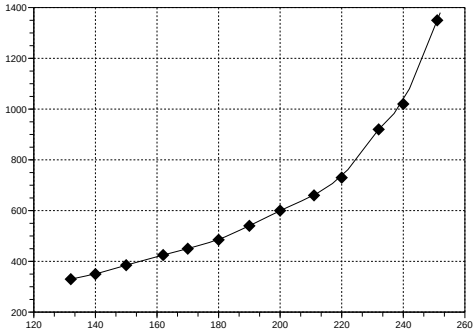

Рис. 11.4. Применение линейного сплайна для задачи 11.1

Построение *кубического сплайна* в Scilab состоит из двух этапов: вначале вычисляются коэффициенты сплайна с помощью функции `d=splin(x,y)`, а затем рассчитывается значения интерполяционного полинома в точке `y=interp(t,x,y,d)`.

Функция `d=splin(x,y)` имеет следующие параметры: `x` — строго возрастающий вектор, состоящий минимум из двух компонент; `y` — вектор того же формата, что и `x`; `d` — результат работы функции, коэффициенты кубического сплайна.

Для функции `y=interp(t,x,y,k)` параметры `x`, `y` и `d` имеют те же значения, параметр `t` — это вектор абсцисс, а `y` — вектор ординат, являющихся значениями кубического сплайна в точках `x`.

Задача 11.4.
Найти приближенное значение функции при заданном значении аргумента с помощью интерполяции кубическими сплайнами в точках $x_=0,702$, $x_2 = 0,512$, $x_3 = 0,608$. Функция задана таблично (табл. 11.4).

Таблица 11.4. Исходные данные к задаче 11.4

0.43	0.48	0.55	0.62	0.7	0.75
1.63597	1.73234	1.87686	2.03345	2.22846	2.35973

Задачу можно решить так:

Листинг 11.7. Решение задачи 11.4

```
x=[0.43 0.48 0.55 0.62 0.7 0.75];
y=[1.63597 1.73234 1.87686 2.03345 2.22846 2.35973];
plot2d(x,y,-4);//График экспериментальных данных
koeff=splin(x,y);
X=[0.702 0.512 0.608];
//Значение функции в заданных точках
Y=interp(X,x,y,koeff)
 Y  = 2.2335678    1.7969698    2.0057073
plot2d(X,Y,-3); //Нанесение точек на график
//Построение кубического сплайна
t=0.43:0.01:0.75;
ptd=interp(t,x,y,koeff);
plot2d(t,ptd);
```

Графическая часть задачи представлена на рис. 11.5.

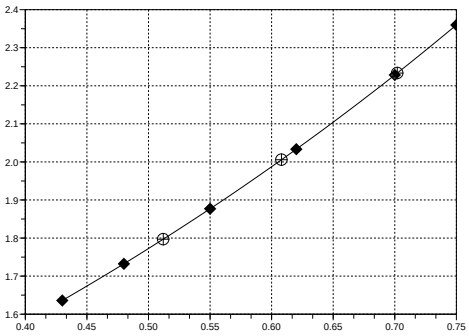

Рис. 11.5. Решение задачи 11.4

Глава 12

Решение дифференциальных уравнений в частных производных

Математические модели физических и иных процессов описываются с помощью дифференциальных уравнений в частных производных[1]. Аргументами функций этих уравнений являются пространственные координаты x, y, z и время t. Общие сведения о дифференциальных уравнениях в частных производных приведены в первом параграфе главы. В Scilab, как и в большинстве математических пакетов, нет средств для непосредственного решения уравнений математической физики. Однако возможностей пакета достаточно, для реализации метода сеток решения дифференциальных уравнений в частных производных. В последующих параграфах этой главы и описана реализация метода сеток для решения параболических, гиперболических и эллиптических уравнений в Scilab.

12.1 Общие сведения о дифференциальных уравнениях в частных производных

Линейным уравнением в частных производных второго порядка называется соотношение между функцией $u(x,y)$ (или $u(x,t)$) и ее частными производными вида [1]:

$$\begin{aligned} L(u) = A(x,y)\frac{\partial^2 u}{\partial x^2} + 2B(x,y)\frac{\partial^2 u}{\partial x \partial y} + C(x,y)\frac{\partial^2 u}{\partial y^2} + \\ + D(x,y)\frac{\partial u}{\partial x} + E(x,y)\frac{\partial u}{\partial y} + G(x,y)u(x,y) = F(x,y) \end{aligned} \tag{12.1}$$

[1] В литературе эти уравнения часто называют уравнениями математической физики.

Если переменная функция u зависит от x и t, то уравнение (12.1) может быть записано следующим образом:

$$L(u) = A(x,t)\frac{\partial^2 u}{\partial x^2} + 2B(x,t)\frac{\partial^2 u}{\partial x \partial t} + C(x,t)\frac{\partial^2 u}{\partial t^2} + \\ + D(x,t)\frac{\partial u}{\partial x} + E(x,t)\frac{\partial u}{\partial t} + G(x,t)u(x,t) = F(x,t) \tag{12.2}$$

В случае если в правой части уравнения $F = 0$, то уравнения (12.1)–(12.2) называются однородными, иначе — неоднородными [2].

Если $B^2 - 4AC < 0$, то уравнение (12.2) относится к классу *эллиптических уравнений*, если $B^2 - 4AC > 0$, то (12.2) это — *гиперболическое уравнение*, если $B^2 - 4AC = 0$ — *параболическое уравнение*. В случае, когда $B^2 - 4AC$ не имеет постоянного знака, то это — уравнение смешанного типа.

С помощью преобразования переменных x, y (или x, t) уравнение можно привести к виду, когда $B = 0$. В этом случае очень просто определяется тип уравнения. Если A и C имеют один и тот же знак, то уравнение (12.2) — *эллиптическое уравнение*, если разные, то *гиперболическое*, а если A или C равно 0, то уравнение относится к *параболическим* [2].

К классическим эллиптическим уравнениям относятся [1, 2]:

- уравнение *Лапласа* $\Delta u = 0$[1], которое используется для описания магнитных и стационарных тепловых полей;
- уравнение *Пуассона* $\Delta u = f$, которое применяется в электростатике, теории упругости и т. д.;
- уравнение *Гельмгольца* $\Delta u + cu = f$, описывающее установившиеся колебательные процессы.

Среди *гиперболических* уравнений можно выделить [3]:

- волновые уравнения: одномерное *волновое* уравнение $\dfrac{\partial^2 u}{\partial t^2} = a^2 \dfrac{\partial^2 u}{\partial x^2} + f(x,t)$, которое описывает вынужденные колебания струны; двумерное волновое уравнение $\dfrac{\partial^2 u}{\partial t^2} = a^2 \left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} \right) + f(x,y,t)$ описывает колебания мембраны[2].
- *телеграфное* уравнение $\dfrac{\partial^2 u}{\partial t^2} + \dfrac{RC + LG}{LC}\dfrac{\partial u}{\partial t} + \dfrac{RG}{LC}u - \dfrac{1}{LC}\dfrac{\partial^2 u}{\partial x^2} = 0$ описывает изменение потенциала u в линиях электропередачи; L, C, R, G — коэффи-

[1] В двумерном случае оператор Лапласа имеет вид $\Delta u = \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}$, в трехмерном — $\Delta u = \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2}$.

[2] При $f = 0$ уравнение описывает свободные колебания струны или мембраны.

циент самоиндукции, емкость, сопротивление, характеристика потерь на единицу длины линии.

К классическим *параболическим* уравнениям относится уравнение *теплопроводности* $\dfrac{\partial u}{\partial t} = a^2 \Delta u + f$.

Для нахождения единственного решения дифференциального уравнения в частных производных необходимо задать начальные и граничные условия. Начальными условиями принято называть условия, заданные в начальный момент времени t. Граничные условия задаются при различных значениях пространственных переменных. Для *эллиптических* уравнений задаются только граничные условия, которые можно разделить на три класса [3]:

- условие *Дирихле* $u_{(x,y,z)\in\Gamma} = \varphi(x,y,z)$, в этом случае на границе области Г, в которой ищется решение, задана некая непрерывная функция φ. В одномерном случае это условие принимает вид $u(0,t) = \varphi_1(t)$, $u(L,t) = \varphi_2(t)$, $(0,L)$ — интервал, на котором ищется решение одномерной задачи;
- условие *Неймана* $\left.\dfrac{\partial u}{\partial n}\right|_{(x,y,z)\in\Gamma} = \varphi(x,y,z)$, в этом случае на границе области задана производная по направлению n внешней нормали;
- смешанное условие $\left(\alpha u + \beta \dfrac{\partial u}{\partial n}\right)_{(x,y,z)\in\Gamma} = \varphi(x,y,z)$.

Для *параболических* уравнений, кроме граничных условий, необходимо определить одно начальное, которое может быть таким: $u(x,t_0) = \psi(x)$.

В случае гиперболических уравнений начальные условия могут быть следующими: $u(x,t_0) = \psi_1(x)$ и $\dfrac{\partial u(x,t_0)}{\partial t} = \psi_2(x)$.

Существуют аналитические методы решения уравнений в частных производных, такие так как метод Фурье (метод разделения переменных), в результате применения которых решение записывается в виде суммы бесконечного ряда довольно сложной структуры, и нахождение численного значения функции в конкретной точке представляет собой отдельную математическую задачу. Поэтому широкое распространение получили численные методы решения уравнений в частных производных.

12.2 Использование метода сеток для решения параболических уравнений в частных производных

Одним из наиболее распространенных численных методов решения уравнений является *метод сеток*[1] [1]. В методе сеток область Ω, в которой необходимо

[1] Метод сеток в литературе также называют методом конечных разностей.

найти решение уравнения, прямыми, параллельными осям $t = t_j$ и $x = x_i$, разобьем на прямоугольные области (см. рис. 12.1), где $x_i = x_0 + ih$, $h = \dfrac{x_n - x_0}{h}$, $i = 0, 1, 2, \ldots, n$, $t_j = t_0 + j\Delta$, $\Delta = \dfrac{t_k - t_0}{k}$, $j = 0, 1, \ldots k$. Точки, которые лежат на границе Γ области Ω, называются *внешними*, остальные точки — *внутренними*. Совокупность всех точек называется сеткой Ω_h^Δ, величины h и Δ — шагами сетки по x и t соответственно.

Идея метода сеток состоит в том, что вместо любой непрерывной функции $w(x,t)$ будем рассматривать дискретную функцию $w_i^j = w(x_i, t_j)$, которая определена в узлах сетки Ω_h^Δ, вместо производных функции будем рассматривать их простейшие разностные аппроксимации в узлах сетки. Таким образом, вместо системы дифференциальных уравнений в частных производных получим систему алгебраических уравнений. Чем меньше величины h и Δ, тем точнее получаемые алгебраические уравнения моделируют исходное дифференциальное уравнение в частных производных. В этом и последующих параграфах этой главы будет рассмотрен метод сеток для каждого из трех типов уравнений и его реализация в Scilab. Знакомство с численными методами решения дифференциальных уравнений в частных производных начнем с разностных схем решения параболических уравнений.

Разностные схемы решения параболических уравнений будем рассматривать на примере следующего одномерного уравнения (12.3):

$$\begin{array}{l} \dfrac{\partial u}{\partial t} = a^2 \dfrac{\partial^2 u}{\partial t^2} + f(x,t), \qquad 0 \leqslant x \leqslant L, \qquad 0 \leqslant t \leqslant T \\ u(0,t) = \mu(t), \qquad u(L,t) = \eta(t), \qquad 0 \leqslant t \leqslant T \\ u(x,0) = \varphi(x), \qquad 0 \leqslant x \leqslant L \end{array} \tag{12.3}$$

Построим сетку Ω_h^Δ (см. рис. 12.1). Для получения сеточного уравнения заменим производную $\dfrac{\partial^2 u}{\partial x^2}$ приближенной разностной формулой [1]:

$$\frac{\partial^2 u(x_i, t_j)}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}. \tag{12.4}$$

В этой и последующих формулах $u_{i,j}$ — это значение функции u в точке (x_i, t_j), $u_{i+1,j}$ — в точке (x_{i+1}, t_j), $u_{i-1,j}$ — в точке (x_{i-1}, t_j), $u_{i,j+1}$ — в точке (x_i, t_{j+1}) и $u_{i,j-1}$ — в точке (x_i, t_{j-1}).

Для замены $\dfrac{\partial u}{\partial t}$ можно воспользоваться одной из приближенных разностных формул [1]:

$$\frac{\partial u(x_i, t_j)}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta} \tag{12.5}$$

$$\frac{\partial u(x_i, t_j)}{\partial t} = \frac{u_{i,j} - u_{i,j-1}}{\Delta} \tag{12.6}$$

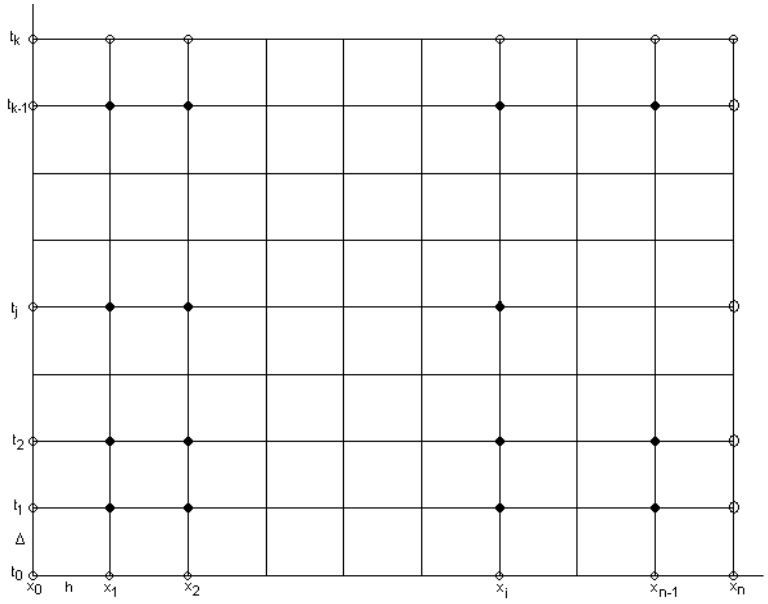

Рис. 12.1. Сетка Ω_h^Δ для области Ω с границей Γ

Кроме того, заменим начальные и граничные условия их разностной аппроксимацией:

$$u_{i,0} = \varphi(x_i) = \varphi_i, \qquad i = 0, 1, \dots n \tag{12.7}$$

$$u_{0,j} = \mu(t_j) = \mu_j, \qquad u_{n,j} = \nu(t_j) = \nu_j, \qquad j = 0, 1, \dots, k \tag{12.8}$$

Заменив частные производные в задаче (12.3) соотношениями (12.4) и (12.5) и учитывая условия (12.7)–(12.8), получим следующую вычислительную схему для расчета значений функции u в узлах сетки Ω_h^Δ:

$$u_{i,j+1} = \gamma u_{i,j-1} + (1 - 2\gamma)u_{i,j} + \gamma u_{i,j+1} + \Delta f_{i,j} \tag{12.9}$$

$$u_{0,j} = \mu_j, \qquad u_{n,j} = \nu_j, \qquad u_{i,0} = \varphi_i, \qquad \gamma = \frac{a^2\Delta}{h^2} \tag{12.10}$$

Это *явная двухслойная* разностная схема (см. рис. 12.2).

Учитывая, что на нулевом слое (при $i = 0$) все значения $u_{i,0}$, (как впрочем, и $u_{0,j}$, $u_{n,j}$) известны, по формуле (12.9) можно сначала явно рассчитать значения $u_{i,1}$, затем $u_{i,2}$ и так до $u_{i,k}$. Для устойчивости разностной схемы (12.9) значения шагов по t и x должны удовлетворять следующему условию:

$$\Delta \leqslant \frac{h^2}{2a^2} \tag{12.11}$$

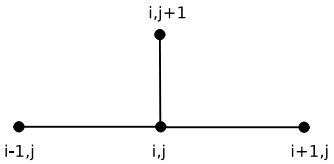

Рис. 12.2. Шаблон явной двухслойной разностной схемы

Рассмотрим решение параболического уравнения на примере следующей задачи.

Задача 12.1.
Решить параболическое уравнение, описывающее распределение температуры в стержне длиной L, начальная температура стержня задается произвольной функцией $\varphi(x)$. Температуры концов стержня равны $u(0,t) = U_1 = \text{const}$, $u(L,t) = U_2 = \text{const}$.

$$\begin{aligned} &\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \qquad a^2 = \frac{\lambda}{c\rho}, \qquad 0 < x < L, \qquad 0 < t < \infty, \\ &u(0,t) = U_1 = \text{const}, \qquad u(L,t) = U_2 = \text{const}, \qquad 0 < t < \infty, \\ &u(x,0) = \varphi(x), \qquad 0 < x < L, \end{aligned} \tag{12.12}$$

здесь a^2 — коэффициент температуропроводности, λ — коэффициент теплопроводности материала стержня, c — удельная теплоемкость, ρ — плотность.

Подпрограмма решения задачи 12.1 с помощью явной разностной схемы (12.9)–(12.10) в Scilab представлена на листинге 12.1[1].

Листинг 12.1. Подпрограмма решения задачи 12.1 с помощью явной разностной схемы

```
//Правая часть дифференциального уравнения.
function y=f(x,t)
y=sin(x*t)
endfunction
//Начальное условие
function y=fi(x)
y=exp(0.15*x)
endfunction
//Условие на левой границе
function y=myu(t)
y=1
```

[1] В листинге 12.1 и во всех последующих уже учтен тот факт, что массивы в Scilab нумеруются с 1.

```
endfunction
//Условие на правой границе
function y=nyu(x)
y=2.117
endfunction
function [u,x,t]=parabol(N,K,L,T,a)
//Функция решения параболического уравнения методом сеток с
//помощью явной разностной схемы. N - количество участков,
//на которые разбивается интервал по x (0,L); K - количество
//участков, на которые разбивается интервал по t (0,T); a -
//параметр дифференциального уравнения теплопроводности,
//Функция возвращает матрицу решений u и вектора x, t
//Вычисляем шаг по x
h=L/N;
//Вычисляем шаг по t
delta=T/K;
//Формируем массив x и первый столбец матрицы решений U
//из начального условия
for i=1:N+1
x(i)=(i-1)*h;
u(i,1)=fi(x(i));
end
//Формируем массив t, первую и последнюю строку
//матрицы решений
//U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
u(1,j)=myu(t(j));
u(N+1,j)=nyu(t(j));
end
gam=a^2*delta/h^2;
//Формируем матрицу решений u с помощью явной разностной схемы
//(12.9)
for j=1:K
for i=2:N
u(i,j+1)=gam*u(i-1,j)+(1-2*gam)*u(i,j)+gam*u(i+1,j)+delta*...
f(x(i),t(j));
end
end
end
```

Входными данными подпрограммы `parabol` решения задачи 12.1 являются: `N` — количество интервалов, на которые разбивается отрезок $(0, L)$; `K` — количество интервалов, на которые разбивается отрезок $(0, T)$; `L` — длина стержня, `T` —

интервал времени, a — параметр дифференциального уравнения. Функция возвращает три параметра: решение — сеточная функция u, определенная на сетке Ω_h^{Δ}, массивы x и t.

На листинге 12.2 представлено обращение к функции parabol для решения задачи 12.1 и построения графика решения, который изображен на рис. 12.3.

Листинг 12.2. Вызов функции parabol

```
[U,X,T]=parabol(50,200,5,3,0.4);
surf(X,T,U');
title('PARABOLIC EQUATION');
xlabel('X');
ylabel('T');
```

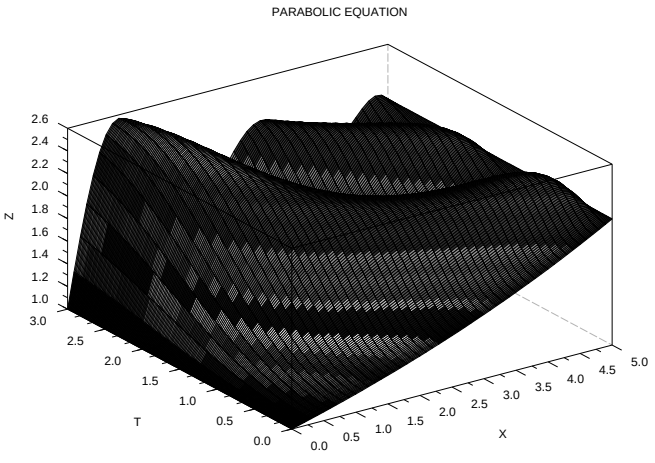

Рис. 12.3. График решения задачи 12.1 при $f(x,t) = sin(xt)$

При решении параболических уравнений с помощью явной разностной схемы основной проблемой является устойчивость решения и правильный выбор шага по t, удовлетворяющего соотношению (12.11).

Для решения этой проблемы были предложены неявные разностные схемы [1]. Эти схемы абсолютно устойчивы, но алгоритм решения получаемого сеточного уравнения несколько сложнее, чем простой пересчет по формуле (12.9). Рассмотрим неявную разностную схему для параболического уравнения. Для построения неявной разностной схемы заменим частные производные в задаче (12.3) соотношениями (12.4), (12.6[1]) и с учетом условий (12.7)–(12.8) получим следующую

[1]Вместо (12.5) для явной схемы

вычислительную схему для расчета значений функции u в узлах сетки Ω_h^{Δ}.

$$\begin{gathered}\gamma u_{i-1,j} - (1+2\gamma)u_{i,j} + \gamma u_{i+1,j} = -u_{i,j-1} - \Delta f_{i,j}, \\ i = 1, 2, \ldots, n-1, \qquad j = 1, 2, \ldots k\end{gathered} \tag{12.13}$$

Соотношения (12.13) вместе с равенствами (12.10) — *неявная двухслойная* разностная схема (см. рис. 12.4). Схема (12.10), (12.13) не позволяет явно выписать решение, и для нахождения $u_{i,j}$ при каждом значении j необходимо решить трехдиагональную систему линейных алгебраических уравнений, для чего можно воспользоваться одним из итерационных методов (например методом Зейделя [1]) или методом прогонки [1]. Преобразуем систему (12.13) к следующему виду:

$$u_{i,j} = \frac{\gamma}{1+2\gamma}(u_{i-1,j} + u_{i+1,j}) + \frac{u_{i,j-1}}{1+2\gamma} + \frac{\Delta}{1+2\gamma} f(x_i, t_j) \tag{12.14}$$

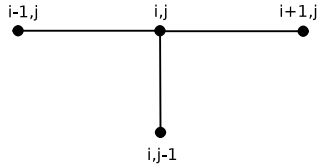

Рис. 12.4. Шаблон неявной двухслойной разностной схемы

Формула (12.14) позволит запрограммировать решение системы, получаемой с помощью неявной разностной схемы, одним из численных методов, например, с помощью метода Зейделя.

Решение параболического уравнения с помощью неявной разностной схемы (с помощью функции `neyavn` представлено на листинге 12.3.

Входными данными функции *neyavn* являются: `N` — количество участков, на которые разбивается интервал по x $(0, L)$; `K` — количество участков, на которые разбивается интервал по t $(0, T)$; `a` — параметр дифференциального уравнения теплопроводности; `eps` — точность решения СЛАУ (12.4) методом Зейделя[1]. Функция `neyavn` возвращает матрицу решений u задачи 12.1; массивы x и t; `r` — точность решения системы (12.4) методом Зейделя; количество итераций `k`.

[1]Выбор метода Зейделя при программировании трехдиагональной системы линейных алгебраических уравнений вида (12.13)–(12.14) — субъективный выбор авторов. Читатель может использовать и другой итерационный метод или специализированный метод решения подобных систем, типа метода прогонки.

Листинг 12.3. Решение задачи 12.1 с помощью неявной разностной схемы[1]

```
//Правая часть дифференциального уравнения.
function y=f(x,t)
y=sin(x*t)
//y=0;
endfunction
//Начальное условие
function y=fi(x)
y=exp(0.15*x)
endfunction
//Условие на левой границе
function y=myu(t)
y=1
endfunction
//Условие на правой границе
function y=nyu(x)
y=2.117
endfunction
function [u,x,t,r,k]=neyavn(N,K,L,T,a,eps)
//Функция решения параболического уравнения методом сеток с
//помощью неявной разностной схемы.
//Вычисляем шаг по x
h=L/N;
//Вычисляем шаг по t
delta=T/K;
//Формируем массив x и первый столбец матрицы решений U
//из начального условия
for i=1:N+1
x(i)=(i-1)*h;
u(i,1)=fi(x(i));
end
//Формируем массив t, первую и последнюю строку матрицы
//решений U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
u(1,j)=myu(t(j));
u(N+1,j)=nyu(t(j));
end
//Определяем матрицу ошибок R и заполняем ее нулями
R(N+1,K+1)=0;
```

[1] Читатель может самостоятельно сравнить результаты, полученные с помощью явной и неявной разностных схем.

```
//Вычисляем коэффициент gamma
gam=a^2*delta/h^2;
r=1;
k=0;
//Цикл while для организации итерационного процесса при
//решении системы уравнений (12.14) методом Зейделя с
//точностью eps
while r>eps
//Вычисление матрицы ошибок R во внутренних точках
//и пересчет значений u во внутренних точках при решениии СЛАУ
//(12.14) методом Зейделя
for i=2:N
for j=2:K+1
R(i,j)=abs(u(i,j)-gam/(1+2*gam)*(u(i-1,j)+u(i+1,j))-u(i,j-1)...
/(1+2*gam)-delta*f(x(i),t(j))/(1+2*gam));
u(i,j)=gam/(1+2*gam)*(u(i-1,j)+u(i+1,j))+u(i,j-1)...
/(1+2*gam)+delta*f(x(i),t(j))/(1+2*gam);
end
end
//Поиск максимума в матрице ошибок
r=R(1,1);
for i=1:N+1
for j=1:K+1
if R(i,j)>r
r=R(i,j);
end
end
end
//Увеличение количества итерации.
k=k+1;
end
disp(k)
endfunction
[U,X,T]=neyavn(50,200,5,3,0.4,0.1);
surf(X,T,U');
title('PARABOLIC EQUATION');
xlabel('X');
ylabel('T');
```

Графики решений с помощью явной и неявной разностных схем практически совпадают.

Обратите внимание, что функции, приведенные на листингах 12.1 и 12.3, позволяют решать уравнения вида (12.3) с различными функциями $f(x,t)$, $\mu(t)$, $\eta(t)$, $\varphi(x)$.

Использование неявной разностной схемы в случае, когда $f(x,t) \neq 0$, рассмотрим на примере еще одной задачи.

Задача 12.2.
Найти распределение температуры в стержне длиной L, начальная температура стержня задается произвольной функцией $f(x)$. Температуры концов стержня равны $u(0,t) = U_1 = \text{const}$, $u(L,t) = U_2 = \text{const}$. На боковой поверхности стержня происходит теплообмен по закону Ньютона со средой, температура которой равна u_0.

Начально-граничная задача, описывающая распределение температуры стержня, имеет вид:

$$\begin{aligned}
&\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} - h\,(u - u_0)\,,\\
&a^2 = \frac{\lambda}{c\rho}, \qquad h = \frac{\alpha p}{c\rho\sigma}, \qquad 0 < x < L, \qquad 0 < t < \infty,\\
&u(0,t) = U_1 = \text{const}, \qquad u(L,t) = U_2 = \text{const}, \qquad 0 < t < \infty,\\
&u(x,0) = \varphi(x), \qquad 0 < x < L
\end{aligned} \tag{12.15}$$

здесь α — коэффициент теплообмена, σ — площадь поперечного сечения стержня, p — периметр поперечного сечения стержня.

Построим сетку Ω_h^{Δ} $(x_i = ihx,\ h = \dfrac{L}{h},\ i = 0,1,2,\ldots,n)$, $(t_j = j\Delta,\ \Delta = \dfrac{T}{k},\ j = 0,1,\ldots k)$. Для получения сеточного уравнения заменим производные $\dfrac{\partial^2 u}{\partial x^2}$ и $\dfrac{\partial u}{\partial t}$ приближенными разностными формулами (12.4) и (12.6). Получится следующая неявная разностная схема (12.16)–(12.18):

$$\begin{aligned}
&u_{i,0} = \varphi(x_i), \qquad i = 0,1,\ldots,N,\\
&u_{0,j} = U_1, \qquad U_{N,j} = U_2, \qquad j = 0,1,\ldots,K
\end{aligned} \tag{12.16}$$

$$u_{i,j} = \frac{1}{1+2\gamma+\Delta h} u_{i,j-1} + \frac{\gamma}{1+2\gamma+\Delta h}\,(u_{i-1,j} + u_{i+1,j}) + \frac{\Delta h}{1+2\gamma+\Delta h} u_{0,}$$
$$i = 1,2,\ldots,N-1; \qquad j = 1,2,\ldots,k; \tag{12.17}$$

$$\gamma = a^2 \frac{\Delta}{hx^2} \tag{12.18}$$

Применение неявной разностной схемы для решения задачи 12.2 представлено на листинге 12.4.

Листинг 12.4. Функция neiav решения задачи 12.2 с помощью неявной разностной схемы

```
//Начальное условие
function y=fi(x)
y=exp(0.15*x)
endfunction
function [u,x,t,r,k]=neiav(N,K,L,T,a,h,U1,U2,u0,eps)
//Функция решения параболического уравнения методом сеток с
//помощью неявной разностной схемы.
//N - количество участков, на которые разбивается интервал по x
//(0,L); K - количество участков, на которые разбивается
// интервал по t (0,T);
//a, h - параметры диффренциального уравнения теплопроводности;
//eps - точность решения СЛАУ (7.17) методом Зейделя;
//U1 - температура на левом конце стержня;
//U2 - температура на правом конце стержня;
//Функция neiav возвращает:
//u - матрицу решений в узлах сетки, массив x, массив t,
//r - точность решения системы (12.17) методом Зейделя,
//k - количество итераций при решении системы методом Зейделя.
//Вычисляем шаг по x
hx=L/N;
//Вычисляем шаг по t
delta=T/K;
//Формируем массив x и первый столбец матрицы решений U
//из начального условия
for i=1:N+1
x(i)=(i-1)*hx;
u(i,1)=fi(x(i));
end
//Формируем массив t, первую и последнюю строку матрицы
//решений U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
u(1,j)=U1;
u(N+1,j)=U2;
end
//Определяем матрицу ошибок R и заполняем ее нулями
R(N+1,K+1)=0;
//Вычисляем коэффициент gamma
gam=a^2*delta/hx^2;
r=1;
k=0;
```

```
//Цикл while для организации итерационного процесса при решении
//системы уравнений (12.17) методом Зейделя с точностью eps
while r>eps
//Вычисление матрицы ошибок R во внутренних точках
//и пересчет значений u во внутренних точках при решениии СЛАУ
//(12.17) методом Зейделя
for j=2:K+1
for i=2:N
V=gam*(u(i-1,j)+u(i+1,j))/(1+2*gam+delta*hx)+u(i,j-1)/...
(1+2*gam+delta*hx)+delta*h*u0/(1+2*gam+delta*hx);
R(i,j)=abs(V-u(i,j));
u(i,j)=V;
end
end
//Поиск максимума в матрице ошибок
r=R(1,1);
for i=1:N+1
for j=1:K+1
if R(i,j)>r
r=R(i,j);
end
end
end
//Увеличение количества итераций
k=k+1;
end
endfunction
//Вызов функции решения задачи 12.2.
[U,X,T,R,K]=neiav(50,200,5,3,0.4,0.5,1,2.117,30,0.001);
//Построение графика функции
surf(X,T,U');
title('Example 12.2');
xlabel('X');
ylabel('T');
```

Входные и выходные данные функции `neiav` решения задачи 12.2 описаны в комментариях листинга 12.4. На рис. 12.5 представлены результаты решения задачи.

Для решения получаемого алгебраического уравнения методом Зейделя потребовалось 1079 итераций. Поэтому для уменьшения количества итераций имеет смысл попробовать ускорить итерационный процесс с помощью методов релаксации или градиентных методов решения систем алгебраических уравнений. Методика, изложенная в этом параграфе, может быть использована и при решении других параболических уравнений.

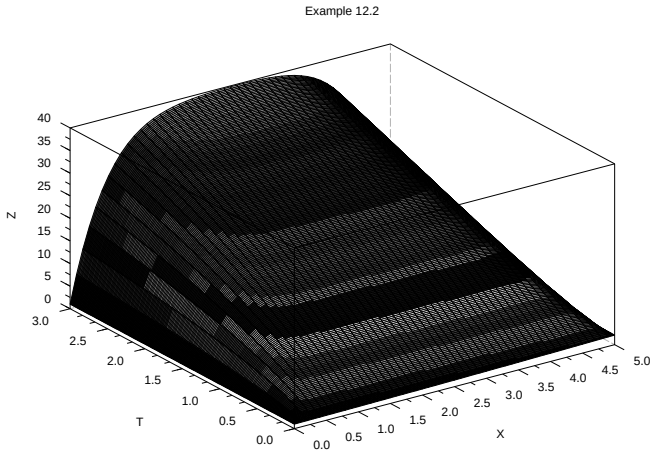

Рис. 12.5. График решения задачи 12.2

12.3 Использование метода сеток для решения гиперболических уравнений

Решение гиперболических уравнений также можно осуществить с помощью разностных схем. Разностные схемы решения одномерного гиперболического уравнения рассмотрим на примере следующего уравнения:

$$\begin{aligned} &\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \qquad 0 < x < L, \qquad 0 \leqslant t \leqslant T \\ &u(0,t) = \mu(t), \qquad u(L,t) = \eta(t), \qquad 0 \leqslant t \leqslant T \\ &u(x,0) = \varphi(x), \qquad \frac{\partial u(x,0)}{\partial t} = \psi(x), \qquad 0 \leqslant x \leqslant L \end{aligned} \tag{12.19}$$

Построим сетку Ω_h^{Δ} (см. рис. 12.1), в которой будем искать решение уравнения (12.19). Частную производную $\dfrac{\partial^2 u}{\partial x^2}$ заменим разностным соотношением (12.4), а производную $\dfrac{\partial^2 u}{\partial t^2}$ — соотношением (12.20) [1].

$$\frac{\partial^2 u(x_i, t_j)}{\partial t^2} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta^2} \tag{12.20}$$

Подставляя (12.20), (12.4), (12.5) в гранично-начальную задачу (12.19), получим следующую явную разностную схему решения уравнения:

$$\begin{array}{l} u_{i,j+1} = -u_{i,j-1} + \gamma(u_{i-1,j} + u_{i+1,j}) = (2-2\gamma)u_{i,j} + \Delta^2 f_{i,j} \\ i = 1,2,\ldots,N-1, j = 1,2,\ldots,K-1 \\ u_{i,0} = \varphi(x_i), \qquad \dfrac{u_{i,1}-u_{i,0}}{\Delta} = \Psi_i, \qquad i = 0,1,\ldots,N \\ u_{0,j} = \mu_j, u_{N,j} = \nu_j, \qquad j = 0,1,\ldots,K \\ \gamma = \dfrac{a^2\Delta^2}{h^2} \end{array} \tag{12.21}$$

которая устойчива при $\gamma < 1$ и по аналогии с разностной схемой (12.9)–(12.10) может быть легко запрограммирована в Scilab.

В качестве примера рассмотрим следующую начально-граничную задачу.

Задача 12.3.
Решить начально-граничную задачу

$$\begin{array}{l} \dfrac{\partial^2\omega}{\partial t^2} = a^2\dfrac{\partial^2\omega}{\partial t^2} + \sin(xt), \qquad 0 < x < L, \qquad t > 0, \\ \omega(0,t) = \varphi(0), \qquad \omega(L,t) = \varphi(L) \\ \omega(x,0) = \varphi(x), \qquad \omega_t(x,0) = \psi(x) \end{array} \tag{12.22}$$

На листинге 12.5 представлена функция ggg решения уравнения (12.22), а на рис. 12.6 — график полученного решения. Параметры функции ggg аналогичны рассмотренным ранее подпрограммам решения параболических уравнений.

Листинг 12.5. Функция ggg решения задачи 12.3 с помощью явной разностной схемы

```
function [u,x,t]=ggg(N,K,L,T,a)
//Функция решения гиперболического уравнения с помощью явной
//разностной схемы. Входные данные:
//N - количество участков, на которые разбивается интервал по
//x (0,L); K - количество участков, на которые разбивается
//интервал по t (0,T); a - параметр дифференциального
//уравнения теплопроводности. Выходные данные:
//u - матрица решений в узлах сетки, массив x, массив t,
//Вычисляем шаг по x
h=L/N;
//Вычисляем шаг по t
delta=T/K;
//Формируем массив x, первый и второй столбцы матрицы решений u
//из начального условия
```

```
for i=1:N+1
x(i)=(i-1)*h;
u(i,1)=fi(x(i));
u(i,2)=u(i,1)+delta*psi(x(i));
end
//Формируем массив t, первую и последнюю строку матрицы решений
//U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
end
//Формируем первую и последнюю строку матрицы решений U
//из граничных условий
for j=2:K+1
u(1,j)=0;
u(N+1,j)=fi(L);
end
gam=a^2*delta^2/h^2;
//Формируем матрицу решений u с помощью явной разностной схемы
//(12.22)
for j=2:K
for i=2:N
u(i,j+1)=-u(i,j-1)+gam*u(i-1,j)+(2-2*gam)*...
 u(i,j)+gam*u(i+1,j)+delta^2*f(x(i),t(j));
end
end
end
```

Для решения гиперболических уравнений можно построить и неявные схемы, однако в связи с тем, что у них нет таких преимуществ перед явными, как в параболических уравнениях, их имеет смысл использовать только когда нельзя построить явную (например, при смешанных условиях на границе области).

12.4 Использование метода сеток для решения эллиптических уравнений

Рассмотрим разностную схему для эллиптического уравнения в прямоугольной области $\Omega(R-b \leqslant x \leqslant R+b, -a \leqslant y \leqslant a)$ с граничными условиями Дирихле на границе Γ.

Задача 12.4.

$$\Delta u = \frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} - \frac{5}{x}\frac{\partial \Psi}{\partial x} = -2 \qquad (12.23)$$
$$\Psi_{(x,y)\in\Gamma} = 0$$

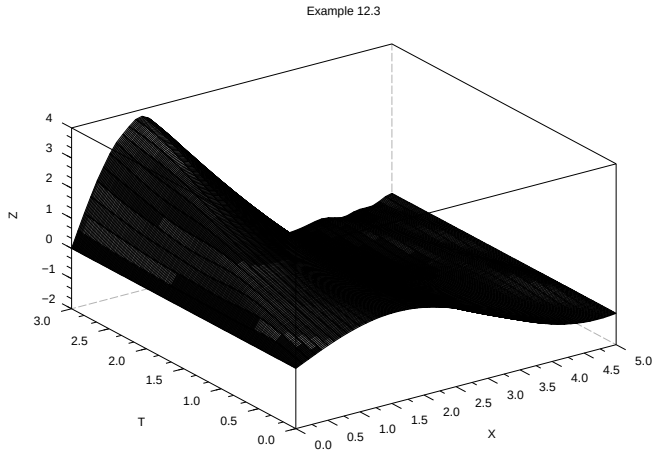

Рис. 12.6. График решения задачи 12.3

Построим сетку Ω_{hx}^{hy}, для чего проведем в области Ω прямые, параллельные осям $y = y_j$ и $x = x_i$, где $x_i = R - b + i \cdot hx$, $hx = \dfrac{2b}{n}$, $i = 0, 1, 2, \ldots, Nx$, $y_j = -a + j \cdot hy$, $hy = \dfrac{2a}{k}$, $j = 0, 1, \ldots, Ny$. Для построения разностного уравнения заменим частные производные и граничные условия следующими соотношениями:

$$\begin{aligned} \frac{\partial^2 \Psi(x_i, y_j)}{\partial x^2} &= \frac{\Psi_{i-1,j} - 2\Psi_{i,j} + \Psi_{i+1,j}}{hx^2} \\ \frac{\partial^2 \Psi(x_i, y_j)}{\partial y^2} &= \frac{\Psi_{i,j-1} - 2\Psi_{i,j} + \Psi_{i,j+1}}{hy^2} \end{aligned} \tag{12.24}$$

$$\begin{aligned} &\Psi_{i,0} = \Psi_{i,Ny} = 0, \qquad i = 0, 1, \ldots, Nx \\ &\Psi_{0,j} = \Psi_{Nx,j} = 0, \qquad j = 0, 1, \ldots, Ny \end{aligned} \tag{12.25}$$

С помощью соотношений (12.24)–(12.25) преобразуем эллиптическую краевую задачу к следующей системе разностных уравнений.

$$\begin{aligned} &\Psi_{i,j} = \frac{1}{A}\left(B_i \Psi_{i+1,j} + C_i \Psi_{i-1,j} + D(\Psi_{i,j-1} + \Psi_{i,j+1}) + 2\right) \\ &A = \frac{2}{hx^2} + \frac{2}{hy^2}, \qquad B_i = \frac{1}{hx^2} + \frac{5}{2hx\, x_i}, \qquad C_i = \frac{1}{hx^2} - \frac{5}{2hx\, x_i}, \qquad D = \frac{1}{hy^2} \\ &i = 1, 2, \ldots, Nx - 1; \qquad j = 1, 2, \ldots, Ny - 1 \\ &\Psi_{i,0} = \Psi_{i,Ny} = 0, \qquad i = 0, 1, \ldots, Nx \\ &\Psi_{0,j} = \Psi_{Nx,j} = 0, \qquad i = 0, 1, \ldots, Ny \end{aligned} \tag{12.26}$$

Эту систему можно решать итерационными методами (например методом Зейделя). В случае медленной сходимости итерационных процессов при решении сеточных уравнений, получаемых при аппроксимации гиперболических и эллиптических задач, имеет смысл попробовать заменить метод Зейделя градиентными методами (или методами релаксации). На листинге 12.6 представлено решение уравнения (12.23) сеточным методом, а на рис. 12.7 — график найденного решения.

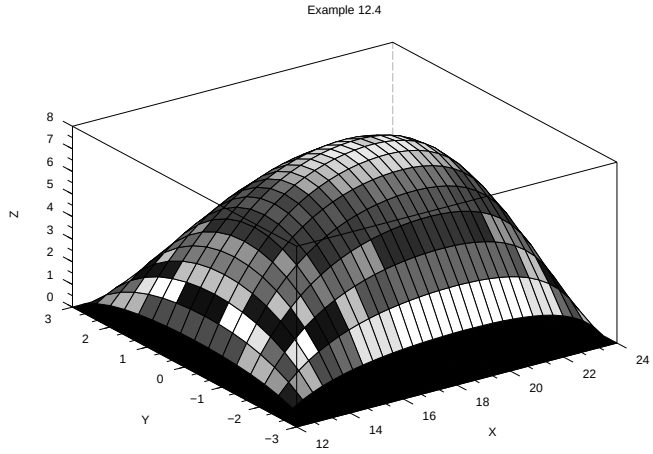

Рис. 12.7. График решения уравнения (12.23) сеточным методом

Листинг 12.6. Решение задачи 12.4

```
function [psi,x,y,k]=ellip(R,a,b,Nx,Ny,eps)
//Функция ellip решения задачи 12.4.
//Входные данные:
//R, a, b - значения, определяющие область решения задачи,
//Nx - количество участков, на которые разбивается интервал по
//x(R-b,R+b);
//Ny - количество участков, на которые разбивается интервал по
//y (-a,a);
//eps - точность решения уравнения (12.26) метоом Зейделя.
//Выходные данные:
//psi - матрица решений в узлах сетки, массив x, массив y,
//k - количество итерация при решении разностного уравнения
//(12.26) методом Зейделя.
//Вычисляем шаг по y
hy=2*a/Ny;
//Вычисляем шаг по x
```

```
hx=2*b/Nx;
//Формируем массив x, первый и последний столбцы матрицы
//решений psi из граничного условия
for i=1:Nx+1
x(i)=R-b+(i-1)*hx;
psi(i,1)=0;
psi(i,Ny+1)=0;
end;
//Формируем массив y, первую и последнюю строки матрицы
//решений psi из граничного условия
for j=1:Ny+1
y(j)=-a+(j-1)*hy;
psi(1,j)=0;
psi(Nx+1,1)=0;
end;
//Вычисляем коэффициенты разностного уравнения (12.26)
A=2/hy^2+2/hx^2;
D=1/hy^2;
for i=2:Nx+1
B(i)=1/hx^2+5/(2*hx*x(i));
C(i)=1/hx^2-5/(2*hx*x(i));
end
//Решение разностного уравнения (12.26) методом Зейделя с
//точностью eps
p=1;
k=0;
while p>eps
for i=2:Nx
for j=2:Ny
V=1/A*(B(i)*psi(i-1,j)+C(i)*psi(i+1,j)+D*(psi(i,j-1)...
+psi(i,j+1))+2);
R(i,j)=abs(V-psi(i,j));
psi(i,j)=V;
end
end
p=R(2,2);
for i=2:Nx
for j=2:Ny
if R(i,j)>p
p=R(i,j);
end
end
end
k=k+1;
```

```
end
endfunction
//Вызов функции решения задачи 12.4.
[PSI,X,Y,K]=ellip(18,3,6,32,16,0.01);
//Построение графика функции
surf(X,Y,PSI');
title('Example 12.4');
xlabel('X');
ylabel('Y');
```

Авторы надеются, что читатель, разобравшийся с решением уравнения (12.23), без проблем построит разностную схему и для других эллиптических уравнений.

Метод сеток позволяет решить широкий класс уравнений в частных производных. Однако при сложной геометрии области, уравнениях с переменными коэффициентами, сложными условиями на границе области использование этого метода нецелесообразно, в этих случаях можно использовать метод конечных элементов, который реализован в кроссплатформенном свободно распространяемом пакете *freefem*[4].

Глава 13

Решение задач оптимизации

В этой главе будут рассмотрены средства пакета Scilab для решения оптимизационных задач.

13.1 Поиск минимума функции одной переменной

В качестве простейшей оптимизационной задачи рассмотрим поиск локального минимума функции одной переменной.

Задача 13.1.
Найти минимум функции $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$.

Решение задачи начнем с построения графика функции (см. листинг 13.1 и рис. 13.1).

Листинг 13.1. Построение графика функции

```
x=-5:0.1:1;
y=x.^4+3*x.^3-13*x.^2-6*x+26;
plot(x,y);
xtitle('График функции f(x)=x^4+3*x^3-13*x^2-6*x+26','X','Y');
xgrid();
```

Из графика видно, что функция имеет минимум в районе точки -4. Для нахождения более точного значения минимума функции в Scilab служит функция

```
[f,xopt]=optim(costf,x0),
```

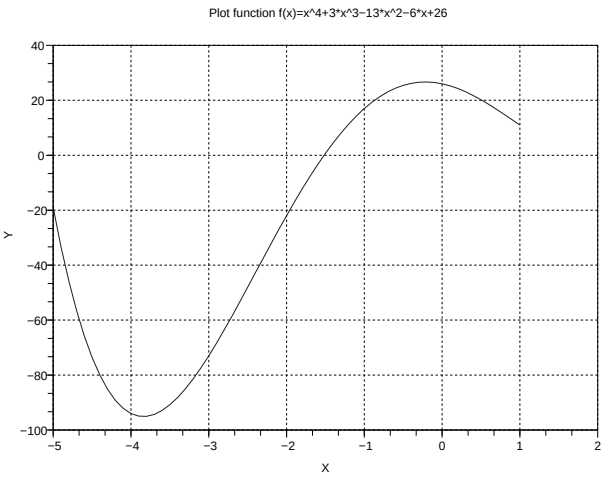

Рис. 13.1. График функции $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$

которая предназначена для поиска минимума любой функции, `x0` — вектор-столбец начальных приближений длиной n, функция `costf` определяет функцию, минимум которой ищется.

Функция возвращает минимум функции (`f`) и точку, в которой функция достигает этого значения (`xopt`).

Главной особенностью функции `optim` является структура функции `costf`, которая должна быть следующей:

```
function [f,g,ind]=costf(x,ind)
//Функция costf должна возвращать функцию f, ее градиент g.
//f - функция от вектора неизвестных x, минимум которой ищется
f=gg(x);
//g - градиент функции f (вектор частной производной f по x),
g=numdiff(gg,x);
endfunction
```

Для функции одной переменной в качестве `f` возвращается функция, минимум которой ищется, в качестве функции `g` — ее производная.

Если возвращаемое сформированной функцией `costf` значение `ind` равно 2, 3 или 4, то функция `costf` обеспечивает поиск минимума, т. е. в качестве результата функции `optim` возвращается `f` и `xopt`. Если `ind=1`, то в функциии `optim` ничего не считается, условие `ind<0` означает, что минимум $f(x)$ не может быть оценен, а `ind=0` прерывает оптимизацию.

Вообще говоря, значение параметра `ind` является внутренним параметром для связи между `optim` и `costf`, для использования `optim` необходимо помнить, что параметр `ind` должен быть определен в функции `costf`.

Таким образом, при использовании функции `optim` необходимо сформировать функцию `costf`, которая должна возвращать минимизируемую функцию `f` и ее градиент (производную).

Рассмотрим использование функции `optim` для поиска минимума $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$. Как видно из графика (см. рис. 13.1) минимум функции достигается в районе $x_{min} \approx 4$. На листинге представлено использование `optim` для поиска минимума функции одной переменной на примере $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$.

Листинг 13.2. Поиск минимума функции одной переменной

```
//Функция fi, в которой будет формироваться функция f и ее
//производная g.
function [f,g,ind]=fi(x,ind)
//Функция f, минимум которой ищется.
f=x^4+3*x^3-13*x^2-6*x+26
//Функция g - производная от функции f.
g=4*x^3+9*x^2-26*x-6
endfunction
//Начальное приближение точки минимума.
y0=-2;
//Для поиска точки минимума (xmin) и значения функции (fmin) в
//ней - вызов функции optim.
[fmin,xmin]=optim(fi,y0);
```

Ниже представлен результат поиска минимума функции одной переменной:

```
-->fmin
fmin =
  - 95.089413
-->xmin
xmin =
3.8407084
```

Аналогично можно найти минимум любой другой функции одной переменной, главной задачей является проблема правильного выбора точки начального приближения. Но это проблема не пакета Scilab, а математическая проблема.

13.2 Поиск минимума функции многих переменных

При нахождении минимума функции многих переменных функцию `costf` необходимо построить таким образом, чтобы входными данными в нее были значения вектора неизвестных `x` и параметра `ind`. Функция `costf` должна зависеть не от нескольких неизвестных, а от одного массива (вектора) неизвестных.

В случае функции многих переменных структура функции `costf` должна быть такой:

```
function [f,g,ind]=costf(x,ind)
//f - функция от вектора неизвестных x, минимум которой ищется
f=gg(x);
//g - градиент функции f (вектор частной производной)
g=numdiff(gg,x);
endfunction
```

В качестве примера рассмотрим поиск минимума функции Розенброка

$$f(x,y) = 100(y - x^2)^2 + (1 - x^2)^2$$

. График функции Розенброка представлен на рис. 13.2.

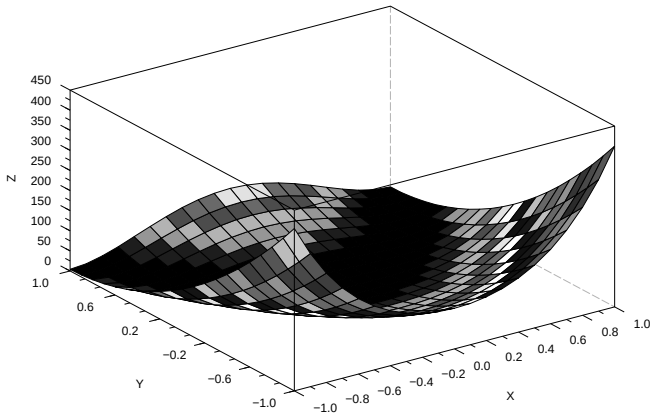

Рис. 13.2. График функции Розенброка

Как известно, функция Розеброка имеет минимум в точке $(1, 1)$, равный 0. Ввиду своей специфики функция Розеброка является тестовой для алгоритмов минимизации. Найдем минимум этой функции с помощью функции `optim` (листинг 13.3).

Листинг 13.3. Поиск минимума функции Розенброка

```
//Начальное приближение x0
x0=[-2;2]
//Функция Розенброка
function y=gg(x)
//Обратите внимание, здесь x - массив из двух неизвестных.
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
//Формирование функции cst, возвращающей функцию Розенброка и
//ее градиент.
function [f,g,ind]=cst(x,ind)
f=gg(x);
g=numdiff(gg,x);
endfunction
//Вызов функции optim
[f,xopt]=optim(cst,x0)
```

Ниже представлен результат поиска минимума функции Розенброка с помощью функции `optim`.

```
x0 =
-2.
2.
xopt =
 0.9999955
 0.9999910
f =
 2.010D-11
```

13.3 Решение задач линейного программирования

Еще одной часто встречающейся в практике оптимизационной задачей является задача линейного программирования. Знакомство с задачами линейного программирования начнем на примере задачи об оптимальном рационе.

Задача об оптимальном рационе. Имеется четыре вида продуктов питания: $П_1$, $П_2$, $П_3$, $П_4$. Известна стоимость единицы каждого продукта c_1, c_2, c_3, c_4. Из этих продуктов необходимо составить пищевой рацион, который должен содержать не менее b_1 единиц белков, не менее b_2 единиц углеводов, не менее b_3 единиц жиров. Причем известно, что в единице продукта $П_1$ содержится a_{11} единиц белков, a_{12} единиц углеводов и a_{13} единиц жиров и т. д. (см. таблицу 13.1).

Требуется составить пищевой рацион, чтобы обеспечить заданные условия при минимальной стоимости.

Таблица 13.1. Содержимое белков, углеводов и жиров в продуктах

Элемент	**белки**	**углеводы**	**жиры**
$П_1$	a_{11}	a_{12}	a_{13}
$П_2$	a_{21}	a_{22}	a_{23}
$П_3$	a_{31}	a_{32}	a_{33}
$П_4$	a_{41}	a_{42}	a_{43}

Пусть x_1, x_2, x_3, x_4 — количества продуктов $П_1$, $П_2$, $П_3$, $П_4$. Общая стоимость рациона равна

$$L = c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 = \sum_{i=1}^{4} c_ix_i \tag{13.1}$$

Сформулируем ограничение на количество белков, углеводов и жиров в виде неравенств. В одной единице продукта $П_1$ содержится a_{11} единиц белков, в x_1 единицах — $a_{11}x_1$, в x_2 единицах продукта $П_2$ содержится $a_{21}x_2$ единиц белка и т. д. Следовательно, общее количество белков во всех четырех типах продукта равно $\sum_{j=1}^{4} a_{j1}x_j$ и должно быть не меньше b_1. Получаем первое ограничение:

$$a_{11}x_1 + a_{21}x_2 + a_{31}x_3 + a_{41}x_4 \geqslant b_1 \tag{13.2}$$

Аналогичные ограничения для жиров и углеводов имеют вид:

$$a_{12}x_1 + a_{22}x_2 + a_{32}x_3 + a_{42}x_4 \geqslant b_2 \tag{13.3}$$

$$a_{13}x_1 + a_{23}x_2 + a_{33}x_3 + a_{43}x_4 \geqslant b_3 \tag{13.4}$$

Приняв во внимание, что x_1, x_2, x_3, x_4 — положительные значения, получим еще четыре ограничения:

$$x_1 \geqslant 0, \qquad x_2 \geqslant 0, \qquad x_3 \geqslant 0, \qquad x_4 \geqslant 0 \tag{13.5}$$

Таким образом, задачу о рациональном рационе можно сформулировать следующим образом: найти значения переменных x_1, x_2, x_3, x_4, удовлетворяющие системе ограничений (13.2)–(13.5), при которых линейная функция (13.1) принимала бы минимальное значение.

Задача об оптимальном рационе является задачей линейного программирования, функция (13.1) называется функцией цели, а ограничения (13.2)–(13.5) — системой ограничений задачи линейного программирования.

В задачах линейного программирования функция цели L и система ограничений являются линейными.

В общем случае задачу линейного программирования можно сформулировать следующим образом. Найти такие значения $x_1, x_2, \ldots, x_n$, удовлетворяющие системе ограничений (13.6), при которых функция цели L (13.7) достигает своего минимального (максимального) значения:

$$\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i, \qquad i = 1, \ldots, m, \qquad x_i \geqslant 0 \tag{13.6}$$

$$L = c_1x_1 + c_2x_2 + \ldots + c_nx_n = \sum_{i=1}^{n} c_ix_i \tag{13.7}$$

Для решения задач линейного программирования в Scilab предназначена функция `linpro` следующей структуры:

```
[x,kl,f]=linpro(c,A,b[,ci,cs][,k][,x0])
```

Здесь `c` — массив (вектор-столбец) коэффициентов при неизвестных функции цели, длина вектора n совпадает с количеством неизвестных x.

`A` — матрица при неизвестных из левой части системы ограничений, количество строк матрицы равно количеству ограничений m, а количество столбцов совпадает с количеством неизвестных n.

`b` — массив (вектор-столбец), содержит свободные члены системы ограничений, длина вектора m.

`ci` — массив (вектор-столбец) размерности n содержит нижнюю границу переменных ($ci_j \leqslant x_j$); если таковая отсутствует, указывают `[]`.

`cs` — массив (вектор-столбец) длиной n, содержит верхнюю границу переменных ($cs_j \geqslant x_j$); если таковая отсутствует, указывают `[]`.

`k` — целочисленная переменная, используется, если в систему ограничений кроме неравенств входят и равенства, в матрице они будут находиться в k первых строках, оставшиеся l строк займут неравенства, т.е. $m = k + l$.

`x0` — вектор-столбец начальных приближений длиной n.

Функция `linpro` возвращает массив неизвестных x, минимальное значение функции f и массив множителей Лагранжа kl.

Рассмотрим использование функции `linpro` на примере решения следующей задачи линейного программирования.

Задача 13.2.
Найти такие значения переменных x_1, x_2, x_3, x_4, при которых функция цели L

$$L = -x_2 - 2x_3 + x_4$$

достигает своего минимального значения и удовлетворяются ограничения:

$$\begin{aligned} &3x_1 - x_2 \leqslant 2 \\ &x_2 - 2x_3 \leqslant -1 \\ &4x_3 - x_4 \leqslant 3 \\ &5x_1 + x_4 \geqslant 6 \\ &x_1 \geq 0, x_2 \geqslant 0, \qquad x_3 \geqslant 0, \qquad x_4 \geqslant 0 \end{aligned}$$

Обратите внимание, что в четвертом ограничении присутствует знак «$\geqslant$». Для приведения системы ограничений виду (13.1) необходимо четвертое уравнение умножить на -1. Решение задачи 13.2 представлено на листинге 13.4.

Листинг 13.4. Решение задачи 13.2

```
c=[0;-1;-2;1];
A=[3 -1 0 0;0 1 -2 0; 0 0 4 -1; -5 0 0 -1];
b=[2;-1;3;-6];
ci=[0;0;0;0];
[x,kl,f]=linpro(p,A,b,ai,[])
```

Полученные значения представлены на листинге 13.5.

Листинг 13.5. Результаты решения задачи 13.2

```
f=
2.
kl=
0.
0.
0.
0.
0.0909091
1.0909091
0.5454545
0.4545455
x=
1.
1.
1.
1
```

Рассмотрим решение еще одной задачи линейного программирования.

Задача 13.3.
Туристическая фирма заключила контракт с двумя турбазами на одном из черноморских курортов, рассчитанными соответственно на 200 и 150 человек. Туристам для осмотра предлагаются дельфинарий в городе, ботанический сад и походы в горы.
Составить маршрут движения туристов так, чтобы это обошлось возможно дешевле, если дельфинарий принимает в день 70 организованных туристов, ботанический сад - 180, а в горы в один день могут пойти 110 человек.
Стоимость одного посещения выражается таблицей 13.2.

Таблица 13.2.

Турбаза	Дельфинарий	Ботанический сад	Поход в горы
1	5	6	20
2	10	12	5

Для решения задачи введем следующие обозначения:

x_1 — число туристов первой турбазы, посещающих дельфинарий;

x_2 — число туристов первой турбазы, посещающих ботанический сад;

x_3 — число туристов первой турбазы, отправляющихся в поход;

x_4 — число туристов второй турбазы, посещающих дельфинарий;

x_5 — число туристов второй турбазы, посещающих ботанический сад;

x_6 — число туристов второй турбазы, отправляющихся в поход.

Составим функцию цели, заключающуюся в минимизации стоимости мероприятий фирмы:

$$Z = 5x_1 + 6x_2 + 20x_3 + 10x_4 + 12x_5 + 5x_6.$$

Руководствуясь условием задачи, определим ограничения:

$$\begin{aligned} &x_1 + x_4 \leqslant 70\,; \\ &x_2 + x_5 \leqslant 180\,; \\ &x_3 + x_6 \leqslant 110\,; \\ &x_1 + x_2 + x_3 = 200\,; \\ &x_4 + x_5 + x_6 = 150\,. \end{aligned}$$

Кроме того, количество туристов, участвующих в мероприятиях, не может быть отрицательным: $x_1 \geqslant 0$, $x_2 \geqslant 0$, $x_3 \geqslant 0$, $x_4 \geqslant 0$, $x_5 \geqslant 0$, $x_6 \geqslant 0$.

Решение задачи представлено в листинге 13.6. В массиве `x` будут хранится значения x_1, x_2, x_3, x_4, x_5, x_6. В матрице коэффициентов `A` первые две строки отражают равенства системы ограничений, поэтому включен параметр `k=2`. Вектор `ci` представляет нижнюю границу неизвестных, т.е. отражает тот факт, что они не могут быть меньше нуля. Если присвоить результат вычислений функции вектору из трех компонент, то первая из них будет содержать вектор неизвестных `x`, вторая — множители Лагранжа `kl`, а третья — минимальное значение функции цели `f`.

Листинг 13.6. Решение задачи 13.3

```
Z=[5;6;20;10;12;5];
A=[1 1 1 0 0 0;
0 0 0 1 1 1;1 0 0 1 0 0;0 1 0 0 1 0;0 0 1 0 0 1];
b=[200;150;70;180;110];
ci=[0;0;0;0;0;0];
k=2;
[x,kl,f]=linpro(Z,A,b,ci,[],k);
```

Полученное в Scilab решение задачи представлено на листинге 13.7

Листинг 13.7. Результаты решения задачи 13.3

```
f=
 2120.
kl=
0.
0.
-20.
0.
-1.
0.
-6.
-11.
1.
0.
6.
x=
30.
170.
0.
40.
0.
110.
```

Авторы надеются, что разобрав примеры из этой главы, читатель сможет использовать Scilab для решения своих оптимизационных задач.

Этой главой мы завершаем первое знакомство с пакетом и надеемся, что свободно распространяемый пакет Scilab поможет читателю решать математические задачи, возникающие в его практической деятельности.

Глава 14

Задания для самостоятельной работы в Scilab

14.1 Задания по теме «Массивы и матрицы в Scilab. Решение задач линейной алгебры»

Задание 1.1. Решить систему линейных алгебраических уравнений, сделать проверку

1. $\begin{cases} -x_1 - x_2 - 2x_3 - 3x_4 = 2 \\ 3x_1 - x_2 - x_3 - 2x_4 = -8 \\ 2x_1 + 3x_2 - x_3 - x_4 = -12 \\ x_1 + 2x_2 + 3x_3 - x_4 = 8 \end{cases}$

2. $\begin{cases} x_1 - 2x_2 + 3x_3 - 2x_4 = -6 \\ x_1 + x_2 - 2x_3 - 3x_4 = -8 \\ 3x_1 - 2x_2 - x_3 + 2x_4 = 4 \\ 2x_1 + 3x_2 + 2x_3 + x_4 = 8 \end{cases}$

3. $\begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 = 5 \\ 2x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ 3x_1 + 2x_2 + x_3 + 2x_4 = 1 \\ 4x_1 + 3x_2 + 2x_3 + x_4 = -5 \end{cases}$

4. $\begin{cases} 0.1x_1 + 0.5x_2 + 0.3x_3 - 0.4x_4 = 2 \\ 0.3x_1 + 0.1x_2 - 0.2x_3 = 0.9 \\ 0.5x_1 - 0.7x_2 + 1x_4 = -0.9 \\ 0.3x_2 - 0.5x_3 = 0.1 \end{cases}$

5. $\begin{cases} 10x_2 + 30x_3 + 40x_4 = -50 \\ 10x_1 + 20x_3 + 30x_4 = -40 \\ 30x_1 + 20x_2 - 50x_4 = 120 \\ 40x_1 + 30x_2 + 50x_3 = 50 \end{cases}$

6. $\begin{cases} 0.3x_1 + x_2 + 1.67x_3 - 2.3x_4 = 4 \\ 3x_1 + 5x_2 + 7x_3 - x_4 = 0 \\ 5x_1 + 7x_2 + x_3 - 3x_4 = 4 \\ 7x_1 + x_2 + 3x_3 - 5x_4 = 16 \end{cases}$

7. $\begin{cases} 2x_1 + x_2 + 5x_3 + x_4 = 8 \\ 0.333x_1 - x_2 - 2x_4 = 3 \\ 2x_2 + x_3 + 2x_4 = -5 \\ x_1 + 4x_2 + 7x_3 + 6x_4 = 0 \end{cases}$

8. $\begin{cases} -x_1 + x_2 + x_3 + x_4 = 12 \\ 2x_1 + x_2 + 2x_3 + 3x_4 = 13 \\ 1.5x_1 + x_2 + 0.5x_3 + x_4 = 7 \\ 4x_1 + 3x_2 + 2x_3 + x_4 = -15 \end{cases}$

9. $\begin{cases} -2x_1 - x_2 + 3x_3 + 2x_4 = 40 \\ -x_1 + x_2 + x_3 + 0.6667x_4 = 20 \\ -3x_1 - x_2 - x_3 + 2x_4 = 60 \\ -3x_1 - x_2 + 3x_3 - x_4 = 60 \end{cases}$

10. $\begin{cases} 3x_1 - 6x_2 - 3x_3 + 3x_4 = 8 \\ 2x_1 - x_2 + x_3 + x_4 = 5 \\ x_1 + x_2 + 2x_3 + x_4 = -1 \\ x_1 - x_2 - x_3 + 3x_4 = 10 \end{cases}$

11. $\begin{cases} 20x_1 + 5x_2 + 5x_4 = -9 \\ x_1 - 3x_2 + 4x_3 = -7 \\ 3x_2 - 2x_3 - 4x_4 = 12 \\ x_1 + 2x_2 - x_3 + 3x_4 = 10 \end{cases}$

12. $\begin{cases} x_1 - 3x_2 + x_3 + x_4 = 11 \\ x_1 + 3x_2 + 5x_3 + 7x_4 = 12 \\ 3x_1 + 5x_2 + 7x_3 + x_4 = 0 \\ -5x_1 - 7x_2 - x_3 - 3x_4 = -4 \end{cases}$

13. $\begin{cases} 2x_1 + x_2 + x_3 - x_4 = 11 \\ 2x_1 + x_2 - 3x_4 = 2 \\ 3x_1 + x_3 + x_4 = -3 \\ 4x_1 - 4x_2 - 4x_3 + 10x_4 = 7 \end{cases}$

14. $\begin{cases} 2x_1 + x_3 + 4x_4 = 19 \\ x_1 + 2x_2 - x_3 + x_4 = 18 \\ 2x_1 + x_2 + x_3 + x_4 = 15 \\ 2x_1 - 2x_2 + 4x_3 + 2x_4 = -11 \end{cases}$

15. $\begin{cases} 5x_1 - 3x_2 - 7x_3 + 3x_4 = 1 \\ -x_2 - 3x_3 + 4x_4 = -5 \\ x_1 - 2x_3 - 3x_4 = -4 \\ 1.3333x_1 - x_2 - 1.6667x_3 = 13 \end{cases}$

Задание 1.2. Если возможно, вычислить матрицу, обратную к матрице D.

1. $D = 2(A^2 + B)(2B - A)$, где

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

2. $D = 3A - (A + 2B)B^2$, где

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

3. $D = 3A^2 - (A + 2B)B$, где

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

4. $D = (A - B^2 2)(2A + B^3)$, где

$$A = \begin{pmatrix} 5 & 2 & 0 \\ 10 & 4 & 1 \\ 7 & 3 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 3 & 6 & -1 \\ -1 & -2 & 0 \\ 2 & 1 & 3 \end{pmatrix}$$

5. $D = 2(A - B)(A^2 + B)$, где

$$A = \begin{pmatrix} 5 & 1 & 7 \\ -10 & -2 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 4 & 1 \\ 3 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix}$$

6. $D = (A - B)^2 A + 2B$, где

$$A = \begin{pmatrix} 5 & -1 & 3 \\ 0 & 2 & -1 \\ -2 & -1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 3 & 7 & -2 \\ 1 & 1 & -2 \\ 0 & 1 & 3 \end{pmatrix}$$

7. $D = (A^2 - B^2)(A + B^2)$, где

$$A = \begin{pmatrix} 7 & 2 & 0 \\ -7 & -2 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 0 & 2 & 3 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{pmatrix}$$

8. $D = 2(A - B)(A^2 + B)$, где

$$A = \begin{pmatrix} 5 & 1 & 7 \\ -10 & -2 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 4 & 1 \\ 3 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix}$$

9. $D = 2A - (A^2 + B)B$, где

$$A = \begin{pmatrix} 1 & 4 & 2 \\ 2 & 1 & -2 \\ 0 & 1 & -1 \end{pmatrix}, \qquad B = \begin{pmatrix} 4 & 6 & -2 \\ 4 & 10 & 1 \\ 2 & 4 & -5 \end{pmatrix}$$

10. $D = 2(A - 0,5B) + A^3 B$, где

$$A = \begin{pmatrix} 5 & 3 & -1 \\ 2 & 0 & 4 \\ 3 & 5 & -1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 4 & 16 \\ -3 & -2 & 0 \\ 5 & 7 & 2 \end{pmatrix}$$

11. $D = (A - B)A^2 + 3B$, где

$$A = \begin{pmatrix} 3 & 2 & -5 \\ 4 & 2 & 0 \\ 1 & 1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} -1 & 2 & 4 \\ 0 & 3 & 2 \\ -1 & -3 & 4 \end{pmatrix}$$

12. $D = 3(A^2 + B^2) - 2AB$, где

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

13. $D = 2A^3 + 3B(AB - 2A)$, где

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 5 & 3 & 1 \\ -1 & 2 & 0 \\ -3 & 0 & 0 \end{pmatrix}$$

14. $D = A(A^2 - B) - 2(B + A)B$, где

$$A = \begin{pmatrix} 2 & 3 & 1 \\ -1 & 2 & 4 \\ 5 & 3 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 7 & 13 \\ -1 & 0 & 5 \\ 5 & 13 & 21 \end{pmatrix}$$

15. $D = (2A - B)(3A + B) - 2A^2B$, где

$$A = \begin{pmatrix} 1 & 0 & 3 \\ -2 & 0 & 1 \\ -1 & 3 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 7 & 5 & 2 \\ 0 & 1 & 2 \\ -3 & -1 & -1 \end{pmatrix}$$

14.2 Задания по теме «Построение двумерных графиков»

Задание 2.1. Изобразите график функции $f(x)$.

1. $f(x) = \dfrac{1.2x^3 + x^2 - 2.8x - 1}{x^2 - 1}$.
2. $f(x) = \dfrac{1.9x^3 - 2.8x^2 - 1.9x + 1}{3x^2 - 1}$.
3. $f(x) = \dfrac{2x^2 - 5}{\sqrt{x^2 - 2}}$.
4. $f(x) = \dfrac{4.1x^3 - 3.25x}{4x^4 - 1}$.
5. $f(x) = \dfrac{x^2 - 11.5}{4x - 3}$.
6. $f(x) = \dfrac{2.3x^2 - 7}{\sqrt{3x^2 - 4}}$.
7. $f(x) = \sqrt[3]{(x - 4.5)^2(x + 2)}$.
8. $f(x) = \sqrt[3]{x^2(x - 4.7)}$.
9. $f(x) = \sqrt[3]{(x + 5)^2} - \sqrt[3]{(x - 7)^2}$.
10. $f(x) = \sqrt[3]{(x^2 - x - 2)^2}$.
11. $f(x) = \sqrt[3]{x^2(x + 3.5)^2}$.
12. $f(x) = \sqrt[3]{(x + 5)^2} - \sqrt[3]{x - 1}$.
13. $f(x) = \sqrt[3]{(3.5 + x)(x^2 + 6x + 6)}$.
14. $f(x) = \sqrt[3]{(4 + x)(x^2 + 2x + 1)}$.
15. $f(x) = \sqrt[3]{(x^2 - x - 6)^2}$.

Задание 2.2. Изобразите график функции в полярных координатах

1. $\rho(\varphi) = -2\mathrm{ctg}\,\varphi$.
2. $\rho(\varphi) = 2\cos 6\varphi$.
3. $\rho(\varphi) = 2^{\varphi} + 1$.
4. $\rho(\varphi) = 2\sqrt{\cos 2\varphi}$.
5. $\rho(\varphi) = 3\varphi + 2$.
6. $\rho(\varphi) = 3\varphi^2 + \varphi$.
7. $\rho(\varphi) = 2\sin 6\varphi$.
8. $\rho(\varphi) = 3^{\varphi}$.
9. $\rho(\varphi) = 2\mathrm{tg}\,3\varphi$.
10. $\rho(\varphi) = \dfrac{1}{\cos\frac{3}{\varphi}}$.
11. $\rho(\varphi) = \dfrac{2}{\sin\varphi} + 3$.
12. $\rho(\varphi) = 5\sin^2\dfrac{\varphi}{3}$.
13. $\rho(\varphi) = \dfrac{2}{\sin\varphi} + 1$.
14. $\rho(\varphi) = 5\sin\dfrac{\varphi}{3}$.
15. $\rho(\varphi) = \dfrac{3}{\varphi^2} + 1$.

14.3 Задания по теме «Построение трехмерных графиков»

Задание 3.1. Построить график, заданный системой уравнений

$$\begin{cases} x = \cos(u)\cdot u\cdot\left(1+\cos\dfrac{(v)}{2}\right); \\ y = \dfrac{u}{2}\cdot\sin(v); \\ z = (\sin(u)\cdot u)\cdot\left(1+\cos\dfrac{(v)}{2}\right). \end{cases}$$

при помощи функции `plot3d2`.

1. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 2\pi$
2. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 8\pi$
3. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 4\pi$
4. $0 \leqslant u \leqslant 8\pi\,, \quad 0 \leqslant v \leqslant 2\pi$
5. $0 \leqslant u \leqslant 4\pi\,, \quad 0 \leqslant v \leqslant 42\pi$
6. $0 \leqslant u \leqslant 8\pi\,, \quad 0 \leqslant v \leqslant 4\pi$
7. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 36\pi$
8. $0 \leqslant u \leqslant 8\pi\,, \quad 0 \leqslant v \leqslant 8\pi$
9. $0 \leqslant u \leqslant 4\pi\,, \quad 0 \leqslant v \leqslant 6\pi$
10. $0 \leqslant u \leqslant 72\pi\,, \quad 0 \leqslant v \leqslant 72\pi$
11. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 5\pi$
12. $0 \leqslant u \leqslant 4\pi\,, \quad 0 \leqslant v \leqslant 78\pi$
13. $0 \leqslant u \leqslant 3\pi\,, \quad 0 \leqslant v \leqslant 8\pi$
14. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 32\pi$
15. $0 \leqslant u \leqslant 2\pi\,, \quad 0 \leqslant v \leqslant 96\pi$

Задание 3.2. Изобразить линии, заданные параметрически:

$$\begin{cases} x(t) = \sin(t) \\ y(t) = \sin(2t) \\ z(t) = t/5 \end{cases} \quad \text{и} \quad \begin{cases} x(t) = \cos(t) \\ y(t) = \cos(2t) \\ z(t) = \sin(t) \end{cases}$$

с помощью функции `param3d`.

№	t	№	t	№	t
1	$[0; 7\pi]$	**6**	$\left[\frac{\pi}{2}; 7\pi\right]$	**11**	$[0; 4\pi]$
2	$[\pi; 4\pi]$	**7**	$[0; 5\pi]$	**12**	$\left[\frac{3\pi}{2}; 7\pi\right]$
3	$\left[\frac{\pi}{2}; 5\pi\right]$	**8**	$[2\pi; 9\pi]$	**13**	$[\pi; 8\pi]$
4	$[2\pi; 8\pi]$	**9**	$[0; 2\pi]$	**14**	$\left[\frac{\pi}{2}; 6\pi\right]$
5	$\left[\frac{3\pi}{2}; 9\pi\right]$	**10**	$[\pi; 7\pi]$	**15**	$[0; 9\pi]$

14.4 Задания по теме «Нелинейные уравнения и системы»

Задание 4.1. Найти корни полиномов.

1. $1,1x^4 - x - 0,9 = 0$
   $x^3 + x - 4 = 0$
2. $2x^4 - x - 1,5 = 0$
   $3x^3 - 5x^2 + 9x - 10 = 0$
3. $2x^4 - 9,25x^2 - 63x + 5 = 0$
   $3x^3 - 21x + 2 = 0$
4. $0,9x^4 + 4,2x^3 - 8,5x^2 - 13 = 0$
   $5x^3 + 13x - 11 = 0$
5. $3x^4 + 4x^3 - 12x^2 - 5 = 0$
   $x^3 + 2x^2 + 2 = 0$
6. $3,2x^4 + 7,75x^3 + 6,3x^2 - 10,5 = 0$
   $2x^3 + 0,48x^2 + 1,6x - 2,6 = 0$
7. $2x^4 - 3x^2 - 5 = 0$
   $2x^3 - 0,52x^2 + 5,4x - 7,4 = 0$
8. $1,05x^4 - 17x^2 + 6 = 0$
   $2x^3 - 0,35x^2 + 0,85x + 1 = 0$

9. $\begin{array}{l} 3,25x^4+7,67x^3+5x^2-11=0 \\ 2x^3+5x^2+11x+7=0 \end{array}$

10. $\begin{array}{l} 2,2x^4-1,2x^2-11=0 \\ 3x^3-0,42x^2+0,95x-2=0 \end{array}$

11. $\begin{array}{l} -x^4-18x^2+6=0 \\ 2x^3-0,08x^2+0,94x+1,3=0 \end{array}$

12. $\begin{array}{l} -1,21x^4+x^3+2x^2-3x-5=0 \\ 3x^3-13x^2+16x-15=0 \end{array}$

13. $\begin{array}{l} 0,89x^4+3,67x^3-7,92x^2-13=0 \\ 2x^3-0,35x^2+0,47x-1,43=0 \end{array}$

14. $\begin{array}{l} 6x^4+8x^3-23x^2+2,1=0 \\ 5x^3+20x^2+5x+8=0 \end{array}$

15. $\begin{array}{l} 2x^4-2x^3-4x^2+6x-7=0 \\ 1,9x^3+7x-11=0 \end{array}$

Задание 4.2. Решение решить систему уравнений.

1. $\begin{cases} \sin(x+1)-y=1,2; \\ 2x+\cos y=2; \end{cases}$

2. $\begin{cases} \cos(x-1)+y=0,5; \\ x-\cos y=3; \end{cases}$

3. $\begin{cases} \cos(x-1)+y=0,5; \\ x-\cos y=3; \end{cases}$

4. $\begin{cases} \sin x+2y=2; \\ \cos(y-1)+x=0,7; \end{cases}$

5. $\begin{cases} \sin x+2y=2; \\ \cos(y-1)+x=0,7; \end{cases}$

6. $\begin{cases} \sin(x+y)-1,2x=0,2; \\ x^2+y^2=1; \end{cases}$

7. $\begin{cases} \operatorname{tg}(xy+0,3)=x^2; \\ 0,9x^2+2y^2=1; \end{cases}$

8. $\begin{cases} \sin(y+1)-x=1,2; \\ 2y+\cos x=2; \end{cases}$

9. $\begin{cases} \sin(x+y)-1,2x=0,1; \\ x^2+y^2=1; \end{cases}$

10. $\begin{cases} 2y-\cos(x+1)=0; \\ x+\sin y=-0,4; \end{cases}$

11. $\begin{cases} \cos(x+0,5)-y=2; \\ \sin y-2x=1; \end{cases}$

12. $\begin{cases} \operatorname{tg} xy=x^2; \\ 0,7x^2+2y^2=1; \end{cases}$

13. $\begin{cases} \sin(x-1)=1,3-y; \\ x-\sin(y+1)=0; \end{cases}$

14. $\begin{cases} \sin(y-1)+x=1,3; \\ y-\sin(x+1)=0,8; \end{cases}$

15. $\begin{cases} \sin(y+1)=x+1; \\ 2y+\cos x=2; \end{cases}$

14.5 Задания по теме «Обработка экспериментальных данных»

Задание 5.1. В результате эксперимента была определена некоторая табличная зависимость. С помощью метода наименьших квадратов определить линию регрессии, рассчитать коэффициент корреляции, подобрать функциональную зависимость заданного вида, вычислить коэффициент регрессии. Определить суммарную ошибку.

1. $P(s) = As^3 + Bs^2 + D$

s	0	1	1.5	2	2.5	3	3.5	4	4.5	5
P	12	10.1	11.58	17.4	30.68	53.6	87.78	136.9	202.5	287

2. $G(s) = As^b$

s	0.5	1.5	2	2.5	3	3.5	4	4.5	5
G	3.99	5.65	6.41	6.71	7.215	7.611	7.83	8.19	8.3

3. $V(s) = As^b e^{Cs}$

s	0.2	0.7	1.2	1.7	2.2	2.7	3.2
V	2.3198	2.8569	3.5999	4.4357	5.5781	6.9459	8.6621

4. $W(s) = \dfrac{A}{Bs + C}$

s	1	2	3	4	5	6	7	8	9
W	0.529	0.298	0.267	0.171	0.156	0.124	0.1	0.078	0.075

5. $Q(s) = As^2 + Bs + C$

s	1	1.25	1.5	1.75	2	2.25	2.5	2.75	3
Q	5.21	4.196	3.759	3.672	4.592	4.621	5.758	7.173	9.269

6. $Y = \dfrac{x}{Ax - B}$

x	3	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9
Y	0.61	0.6	0.592	0.58	0.585	0.583	0.582	0.57	0.572	0.571

7. $V = \dfrac{1}{A + Be^{-U}}$

U	0	1	1.5	2	2.5	3	3.5	4	4.5	5
V	12	10.1	11.58	17.4	30.68	53.6	87.78	136.9	202.5	287

8. $Z = At^4 + Bt^3 + Ct^2 + Dt + K$

t	0.66	0.9	1.17	1.47	1.7	1.74	2.08	2.63	3.12
Z	38.9	68.8	64.4	66.5	64.95	59.36	82.6	90.63	113.5

9. $R = Ch^2 + Dh + K$

h	2	4	6	8	10	12	14	16
R	0.035	0.09	0.147	0.2	0.24	0.28	0.31	0.34

10. $Y = Ax^3 + Bx^2 + Cx + D$

x	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3
Y	1.5	2.7	3.9	5.5	7.1	9.1	11.1	12.9	15.5	17.9

11. $Y = Ax^3 + Cx + D$

x	0	0.4	0.8	1.2	1.6	2
Y	1.2	2.2	3.0	6.0	7.7	13.6

12. $R = Ch^2 + K$

h	0.29	0.57	0.86	0.14	1.43	1.71	1.82	2
R	3.33	6.67	7.5	13.33	16.67	23.33	27.8	33.35

13. $Z = At^4 + Ct^2 + K$

t	1	1.14	1.29	1.43	1.57	1.71	1.86	1.92	2
Z	6.2	7.2	9.6	12.5	17.1	22.2	28.3	35.3	36.5

14. $Z = At^4 + Bt^3 + Dt + K$

t	2	2.13	2.25	2.38	2.5	2.63	2.75	2.88	3
Z	12.57	16.43	19	22.86	26.71	31.86	37.0	43.43	49.86

15. $Z = At^4 + Dt + K$

t	0.88	0.9	0.91	0.93	0.94	0.96	0.97	0.99	1
Z	0.029	0.086	0.17	0.31	0.43	0.57	0.71	0.86	0.97

Задание 5.2. Для *вариантов 1-7* найти приближенное значение функции при заданном значении аргумента с помощью функции линейной интерполяции. Функция задана таблично.

1. $x_1 = 0.702, \quad x_2 = 0.512, \quad x_3 = 608$

x	0.43	0.48	0.55	0.62	0.7	0.75
y	1.63597	1.73234	1.87686	2.03345	2.22846	2.35973

2. $x_1 = 0.102, \quad x_2 = 0.203, \quad x_3 = 0.154$

x	0.02	0.08	0.12	0.17	0.23	0.30
y	1.02316	1.09509	1.14725	1.21423	1.30120	1.40907

3. $x_1 = 0.526, \quad x_2 = 0.453, \quad x_3 = 0.436$

x	0.35	0.41	0.47	0.51	0.56	0.64
y	2.73951	2.30080	1.96864	1.78776	1.59502	1.34310

4. $x_1 = 0.616, \quad x_2 = 0.478, \quad x_3 = 0.537$

x	0.41	0.46	0.52	0.6	0.65	0.72
y	2.57418	2.32513	2.09336	1.86203	1.74926	1.62098

5. $x_1 = 0.896, \quad x_2 = 0.774, \quad x_3 = 0.955$

x	0.68	0.73	0.80	0.88	0.93	0.99
y	0.80866	0.89492	1.02964	1.20966	1.34087	1.52368

6. $x_1 = 0.314, \quad x_2 = 0.235, \quad x_3 = 0.186$

x	0.11	0.15	0.21	0.29	0.35	0.40
y	9.05421	6.61659	4.69170	3.35106	2.73951	2.36522

7. $x_1 = 1.3832, \quad x_2 = 1.3926, \quad x_3 = 1.3866$

x	1.375	1.380	1.385	1.390	1.395	1.400
y	5.04192	5.17744	5.32016	5.47069	5.62968	5.79788

Для *вариантов 8-15* найти приближенное значение функции при заданном значении аргумента с помощью сплайн-интерполяции. Функция задана таблично.

8. $x_1 = 0.308, \quad x_2 = 0.325, \quad x_3 = 0.312$

x	0.298	0.303	0.310	0.317	0.323	0.330
y	3.25578	3.17639	3.12180	3.04819	2.98755	2.91950

9. $x_1 = 0.608, \quad x_2 = 0.594, \quad x_3 = 0.631$

x	0.593	0.598	0.605	0.613	0.619	0.627
y	0.53205	0.53562	0.54059	0.54623	0.55043	0.55598

10. $x_1 = 0.115, \quad x_2 = 0.130, \quad x_3 = 0.164$

x	0.100	0.108	0.119	0.127	0.135	0.146
y	1.12128	1.13160	1.14594	1.15648	1.16712	1.18191

11. $x_1 = 0.720, \quad x_2 = 0.777, \quad x_3 = 0.700$

x	0	1	1.5	2	2.5	3	3.5	4	4.5	5
y	12	10.1	11.58	17.4	30.68	53.6	87.78	136.9	202.5	287

12. $x_1 = 0.238, \quad x_2 = 0.261, \quad x_3 = 0.275$

x	0.235	0.240	0.250	0.255	0.265	0.280
y	1.20800	1.21256	1.22169	1.22628	1.23547	1.24933

13. $x_1 = 0.105, \quad x_2 = 0.109 \quad , x_3 = 0.111$

x	0.095	0.102	0.104	0.107	0.110	0.112
y	1.09131	1.23490	1.27994	1.35142	1.42815	1.48256

14. $x_1 = 0.1817, \quad x_2 = 0.2275, \quad x_3 = 0.175$

x	0.180	0.185	0.190	0.195	0.200	0.205
y	5.61543	5.46693	5.32634	5.19304	5.06642	4.94619

15. $x_1 = 3.522, \quad x_2 = 4.176, \quad x_3 = 3.475$

x	3.50	3.55	3.60	3.65	3.70	3.75
y	33.1154	34.8133	36.5982	38.4747	40.4473	42.5211

14.6 Задания по теме «Решение задач оптимизации»

Задание 6. Решить задачу целочисленного программирования.

1. $W = 2x_1 - x_2 + x_4 \to \min$
$$\begin{cases} x_1 + x_2 + x_3 - x_4 \leqslant 1 \\ x_1 - x_2 + x_3 - x_4 \leqslant 0 \\ 2x_1 + x_2 + x_3 - x_4 \geqslant 3 \end{cases}$$

2. $W = x_1 + x_3 \to \max$
$$\begin{cases} 2x_1 - 7x_2 + 22x_3 \leqslant 22 \\ 2x_1 - x_2 + 6x_3 \leqslant 6 \\ 2x_1 - 5x_2 + 2x_3 \leqslant 2 \\ -4x_1 + x_2 + x_3 \leqslant 1 \end{cases}$$

3. $W = 3 + 2x_2 + x_3 \to \max$
$$\begin{cases} x_1 - x_2 + 2x_3 + x_4 \geqslant 1 \\ 2x_1 - x_2 + x_3 - x_4 \geqslant 1 \\ x_1 - 2x_2 + x_3 - x_4 \geqslant -1 \\ x_1 + x_2 + x_3 + 2x_4 \leqslant 5 \end{cases}$$

4. $W = x_3 + 3x_4 \to \min$
$$\begin{cases} x_1 + x_2 - x_3 - x_4 \leqslant 2 \\ x_1 - x_2 - x_3 + x_4 \geqslant 0 \\ -x_1 - x_2 + 2x_3 - x_4 \geqslant -3 \\ x_1 \geqslant 1 \end{cases}$$

5. $W = -x_1 + x_2 \to \max$
$$\begin{cases} x_1 - 2x_2 \geqslant 2 \\ 2x_1 - x_2 \geqslant 2 \\ x_1 + x_2 \geqslant 5 \end{cases}$$

6. $W = x_1 - x_2 - 2x_4 \to \max$
$$\begin{cases} 2x_1 - x_2 + 2x_3 - x_4 \leqslant 4 \\ x_1 - 2x_2 + x_3 - 2x_4 \geqslant 2 \\ x_1 - x_4 \geqslant 1 \\ x_2 + x_3 \leqslant 1 \end{cases}$$

7. $W = x_1 - x_2 + 3x_3 + x_4 \to \max$
$$\begin{cases} x_1 - x_2 + x_4 \leqslant 1 \\ x_2 - x_3 + x_4 \leqslant 1 \\ x_1 + x_3 + 2x_4 \leqslant 2 \\ -2x_2 + x_4 \leqslant 0 \end{cases}$$

8. $W = -x_2 - 2x_3 + x_4 \to \min$
$$\begin{cases} 3x_1 - x_2 \leqslant 2 \\ x_2 - 2x_3 \leqslant -1 \\ 4x_3 - x_4 \leqslant 3 \\ 5x_1 + x_4 \geqslant 6 \end{cases}$$

9. $W = x_1 + x_2 + 3x_3 - x_4 \to \max$
$$\begin{cases} x_1 - 5x_2 + 4x_3 \leqslant 5 \\ x_1 - 2x_2 - 3x_3 \leqslant 4 \\ x_1 + 6x_2 + 5x_3 \leqslant 4 \\ x_2 + x_3 \leqslant 1 \end{cases}$$

10. $W = -4 - 2x_1 - x_2 - x_3 \to \min$
$$\begin{cases} x_1 - 2x_2 + 3x_3 - 4x_4 \geqslant -10 \\ x_1 + x_2 - x_3 - x_4 \leqslant -4 \\ x_1 - x_2 + x_3 - x_4 \geqslant -6 \\ x_1 + x_2 + x_3 + x_4 \leqslant 10 \end{cases}$$

11. $W = x_1 + x_2 + x_3 + 1 \to \min$
$$\begin{cases} x_1 + x_2 \geqslant 0 \\ x_1 + x_3 \geqslant 1 \\ x_2 - x_3 \geqslant 1 \\ x_1 + 2x_2 + 3x_3 \geqslant 0 \end{cases}$$

12. $W = 2 + 2x_2 - x_3 + 3x_4 \to \max$
$$\begin{cases} -x_1 + x_2 - 2x_4 \geqslant -1 \\ x_1 + x_3 + x_4 \geqslant 1 \\ x_2 + x_3 - x_4 \geqslant 1 \\ x_3 \leqslant 4; \qquad x_2 \leqslant 10 \end{cases}$$

13. $W = x_1 + x_2 + 3 \to \max$
$$\begin{cases} x_1 - x_2 \leqslant 1 \\ x_1 - 2x_2 \geqslant -2 \\ -x_1 + x_2 \geqslant -1 \\ 2x_1 + x_2 \geqslant -2 \end{cases}$$

14. $W = x_1 - 10x_2 + 100x_3 \to \max$
$$\begin{cases} x_1 + x_2 + x_3 \leqslant 1 \\ x_1 - x_2 - x_3 \leqslant 2 \\ -x_1 + 2x_3 \leqslant 0 \\ x_1 + 2x_3 \leqslant 5 \end{cases}$$

15. $W = -3 + x_1 + 3x_2 + 5x_3 \to \max$
$$\begin{cases} x_1 - x_2 + x_3 \leqslant 1 \\ 2x_1 + x_2 + x_3 \leqslant 1 \\ x_1 + 2x_2 + x_3 \leqslant 1 \\ x_1 + x_2 + 2x_3 \leqslant 1 \end{cases}$$

Предметный указатель

Литература

[1] Вержбицкий В. М. Основы численных методов. — М.: Высшая школа, 2002. — 840 с.

[2] Голосков Д. П. Уравнения математической физики. Решение задач в системе Maple. — СПб.: Питер, 2004. — 539 с.

[3] Тихонов А. Н., Самарский А. А. Уравнения математической физики. — М.: Наука, 1966. — 724 с.

[4] `www.freefem.org`

Сведения об авторах

Алексеев Евгений Ростиславович, кандидат технических наук, доцент кафедры «Вычислительная математика и программирование» Донецкого национального технического университета, автор 8 книг и более 40 научных и методических работ.

Чеснокова Оксана Витальевна, старший преподаватель кафедры «Вычислительная математика и программирование» Донецкого национального технического университета, автор 6 книг и более 20 научных и методических работ.

Рудченко Екатерина Александровна, бакалавр по специальности «Экологическая геология» Донецкого национального технического университета.

Учебное издание

Алексеев Евгений Ростиславович
Чеснокова Оксана Витальевна
Рудченко Екатерина Александровна

Scilab: Решение инженерных и математических задач

Редактор серии: К. А. Маслинский
Редактор: В. М. Жуков
Оформление обложки: В. Меламед
Вёрстка: К. И. Михайленко

Подписано в печать 16.05.08. Формат 70x100/16.
Гарнитура Computer Modern. Печать офсетная. Бумага офсетная.
Усл. печ. л. 22,1. Тираж 2000 экз. Заказ

ООО «Альт Линукс Технолоджи»
Адрес для переписки: 119334, Москва, 5-й Донской проезд, д. 21Б, стр. 21
Телефон: (495) 662-38-83. E-mail: sales@altlinux.ru
http://altlinux.ru

Издательство «БИНОМ. Лаборатория знаний»
Адрес для переписки: 125167, Москва, проезд Аэропорта, 3
Телефон (499) 157-52-72. E-mail: binom@Lbz.ru
http://www.Lbz.ru