Библиотека ALT

Е. Р. Алексеев, К. В. Дога, О. В. Чеснокова

Scilab

Решение инженерных и математических задач

ДМК ПРЕСС
издательство
Москва, 2024

УДК 004.67
ББК 22.1
А47

Алексеев Е. Р., Дога К. В., Чеснокова О. В.

А47 Scilab: Решение инженерных и математических задач: учеб. издание / отв. ред. В. Л. Чёрный. – М.: Базальт СПО; ДМК Пресс, 2024. – 440 с.: ил. – (Библиотека ALT)

ISBN 978-5-93700-271-6

Книга посвящена свободно распространяемому математическому пакету Scilab. Описан язык программирования Scilab. Особое внимание уделено визуальным средствам программирования. Описаны графические возможности пакета. Подробно рассмотрено решение математических задач (нелинейные уравнения и системы, задачи линейной алгебры, задачи оптимизации, дифференцирование и интегрирование, задачи обработки экспериментальных данных, обыкновенные дифференциальные уравнения и системы, уравнения в частных производных).

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

УДК 004.67
ББК 22.1

ISBN 978-5-93700-271-6

Содержание

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Сведения об авторах

Алексеев Евгений Ростиславович, кандидат технических наук, доцент, доцент кафедры информационных образовательных технологий Кубанского государственного университета, автор 15 книг и более 120 научных и методических работ.

Чеснокова Оксана Витальевна, кандидат технических наук, доцент кафедры общематематических и естественно-научных дисциплин Московского финансово-юридического университета, автор 10 книг и более 80 научных и методических работ.

Дога Кристина Вячеславовна, студент факультета математики и компьютерных наук Кубанского государственного университета, педагог дополнительного образования центра детского творчества «Прикубанский».

Рецензенты

Д. А. Тархов – доктор технических наук, профессор кафедры высшей математики Санкт-Петербургского политехнического университета.

В. И. Родионов – кандидат физико-математических наук, доцент, заведующий кафедрой информатики и математики Удмуртского государственного университета.

Введение

Первая наша книга, посвящённая Scilab – системе компьютерной математики, вышла в свет в 2008 году [2]. Она была благожелательно принята читателями. Возможно, потому, что это была одна из первых книг по Scilab (наши коллеги утверждают, что это было первое издание на русском языке), а может быть, и потому, что получилась неплохая книга. Авторы многие годы использовали эту книгу при обучении студентов в университетах России и Украины (Донецкий национальный технический университет, Вятский государственный университет, Кубанский государственный университет). Также книгу использовали коллеги из Москвы, Санкт-Петербурга, Новосибирска, Львова, Ижевска и других городов. Авторы получили много отзывов из всей России. Книга познакомила нас со многими учёными и преподавателями (профессор С. П. Шарый из Новосибирска, профессора А. Н. Васильев, Д. А. Тархов из Санкт-Петербурга, доценты В. Р. Кристалинский, В. И. Мунерман из Смоленска и многие другие).

Однако авторы отдавали себе отчёт, что книга неидеальна, в ней есть моменты, которые хотелось бы поправить. Кроме того, за 17 лет, прошедших с момента написания книги, значительно изменился сам пакет Scilab, расширились его возможности. Scilab за эти годы стал достаточно популярен. Теперь нет необходимости сравнивать его с проприетарными математическими пакетами. Изменился и пользователь пакета. Свободное программное обеспечение шагнуло далеко за пределы узкого круга «хакеров-линуксоидов». Уровень информационных технологий сделал доступным знакомство с наукой в очень молодом возрасте.

За последнее десятилетие появилось достаточное количество литературы по Scilab, в частности русскоязычной [4, 7, 10, 11, 12, 14]. Особое внимание хотелось бы обратить на книги Б. И. Квасова [10] и А. Н. Титова, Р. Ф. Тазиевой [12]. В обеих работах авторы рассматривают Scilab как инструмент для решения математических задач. Много ссылок на литературу на разных языках можно найти на странице официального сайта Scilab https://www.scilab.org/about/community/books.

В предлагаемой читателю книге значительно расширены разделы, посвящённые графическому отображению информации, созданию визуальных приложений в Scilab и решению математических задач. В книге можно

прочесть об использовании функций Scilab для решения математических, инженерных и экономических задач, об алгоритмах решения подобных задач. Авторы сочли возможным написать свои функции, реализующие описанные алгоритмы вычислительной математики. Хотим сразу извиниться перед специалистами по численным методам, которым наше описание алгоритмов решения вычислительных задач покажется кратким и неполным. Были описаны только проверенные вычислительные схемы решения задач. Авторы отсылают читателя к классическим и современным учебникам по численным методам [5, 6, 10, 15].

Существуют версии Scilab для различных операционных систем: для ОС семейства Linux, ОС Windows и даже для macOS. При написании книги авторы использовали ОС «Альт Образование». На момент написания книги последней была версия пакета Scilab 2023.1.0. Последнюю версию пакета можно скачать на официальном сайте www.scilab.org.

В первых главах книги описан входной язык системы Scilab, что позволит читателю на самых первых этапах при решении математических и инженерных задач не только использовать встроенные команды пакета, но и разрабатывать собственные программы.

Книга состоит из тринадцати глав.

Первая глава является своеобразным введением в Scilab. В ней описаны основные возможности пакета, уделено внимание особенностям установки на компьютер.

Во *второй главе* описан язык программирования Scilab.

Третья глава посвящена работе с массивами и матрицами в Scilab, в ней рассматриваются возможности Scilab при решении задач линейной алгебры.

В *четвёртой главе* речь идет о графических возможностях Scilab. Подробно описаны возможности пакета для построения двумерных и трёхмерных графиков. Рассмотрены возможности форматирования графиков. Глава значительно расширена.

Пятая глава знакомит читателя с разработкой оконных приложений в Scilab.

Шестая глава знакомит читателя с различными способами решения нелинейных уравнений и систем в Scilab. Авторы добавили сюда описание методов решения нелинейных уравнений. Отдельный параграф посвящён алгоритмам нахождения корней полинома.

В *седьмой главе* внимание уделено вычислительным методам интегрирования и дифференцирования, а в восьмой главе рассмотрено решение обыкновенных дифференциальных уравнений в Scilab. Добавлено краткое описание основных численных методов решения обыкновенных дифференциальных уравнений и систем.

В *девятой главе* описаны задачи обработки результатов эксперимента и методы их решения.

В *десятой 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 *одиннадцатой главе* вниманию читателя представлены задачи оптимизации функций одной и многих переменных, а также задачи линейного прог-

раммирования. Добавлен раздел, посвящённый решению задач квадратичного программирования.

Двенадцатая глава посвящена использованию Scilab в качестве вычислительного ядра для создания интерактивных документов в формате Jupyter Notebook.

В *тринадцатой главе* приведено большое количество заданий для самостоятельного решения или проведения лабораторных (практических) занятий, что позволит использовать книгу в учебном процессе. Кроме того, читатель всегда сможет воспользоваться книгой как справочником по решению математических задач в среде Scilab.

Авторы благодарят компанию «Базальт СПО» за возможность издания книги, посвященной пакету Scilab.

Е. Р. Алексеев, К. В. Дога, О. В. Чеснокова
Краснодар, Москва, 2024

Глава 1

Пакет Scilab. Начало работы

Scilab – это свободная система компьютерной математики, которая предназначена для выполнения математических, инженерных и научных вычислений.

Основные характеристики пакета Scilab:

1) кросс-платформенность. Существуют версии пакета Scilab для операционных систем Linux и Windows. Интерфейсы этих версий несколько различаются, но все команды идентичны;
2) пакет оснащен интерфейсом и системой помощи;
3) содержит функции, реализующие основные алгоритмы базовой математики;
4) имеет достаточно мощный собственный язык программирования высокого уровня;
5) содержит большое количество встроенных функций для решения различных математических задач;
6) обладает широкими возможностями построения и редактирования графиков и поверхностей.

В Scilab есть большое количество встроенных функций, предназначенных для решения:

- нелинейных уравнений и систем;
- задач линейной алгебры;
- задач оптимизации.

Кроме того, можно выполнять операции дифференцирования и интегрирования, решать дифференциальные уравнения и задачи обработки экспериментальных данных и множество других задач.

Несмотря на то что система Scilab содержит достаточное количество встроенных команд, операторов и функций, отличительная её черта – это гибкость. Пользователь может создать любую новую команду или функцию, а затем использовать её 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 можно получить на сайте программы www.scilab.org. Онлайн-справка на русском языке доступна по адресу https://help.scilab.org/docs/2023.1.0/ru_ RU/index.html. Здесь 2023.1.0 – последняя на момент написания книги версия. Вместо неё можно указать любую интересующую версию. Доступна справка по всем версиям.

Кроме того, в любой момент времени пользователь может получить справку по любому объекту Scilab, набрав в командном окне Scilab

```
help name
```

Здесь `name` – служебное слово или встроенная функция Scilab.

Scilab может быть установлен на этапе инсталляции операционной системы «Альт Образование». Пакет находится в репозиториях большинства современных дистрибутивов: операционные системы семейства «Альт», ОС Debian, ОС семейства Ubuntu, ОС Linux Mint и др.

В дистрибутивах семейства «Альт» установка Scilab осуществляется командой `apt-get` от имени администратора.

```
#apt-get install scilab
```

Аналогично Scilab устанавливается и в дистрибутивах семейства Debian и Ubuntu.

```
#apt install scilab
```

Также можно скачать одну из последних стабильных двоичных версий Scilab по адресу https://www.scilab.org/download.

Имя файла архива имеет вид *scilab-m.n.k.bin.x86_64-linux-gnu.tar.xz*. Адрес для скачивания обычно такой: https://www.scilab.org/download/m. n.k/scilab-m.n.k.bin.x86_64-linux-gnu.tar.xz. Здесь *m.n.k* – номер версии. После этого архив надо развернуть. В процессе разархивирования будет создан каталог *scilab-m.n.k*, внутри которого существует каталог *bin* с исполняемыми файлами программ. Для запуска Scilab необходимо запустить файл *scilab*[1] из каталога *bin*[2].

1.2 Первое знакомство со Scilab

После запуска Scilab на экране появится *основное окно приложения*. Окно содержит *меню*, *панель инструментов*, *рабочую область*, *обозреватель файлов*, *обозреватель переменных*, *журнал команд* и *окно подачи новостей*. Признаком того, что система готова к выполнению команды, является наличие знака

[1] Убедитесь, что у вас есть право запуска исполняемого файла Scilab.

[2] Авторы сталкивались с тем, что после запуска Scilab иногда не отображаются графики. В этом случае имеет смысл попробовать запускать исполняемый файл так: LIBGL_ALWAYS_SOFTWARE=1 scilab-m.n.k/bin/scilab.

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. 1.1).

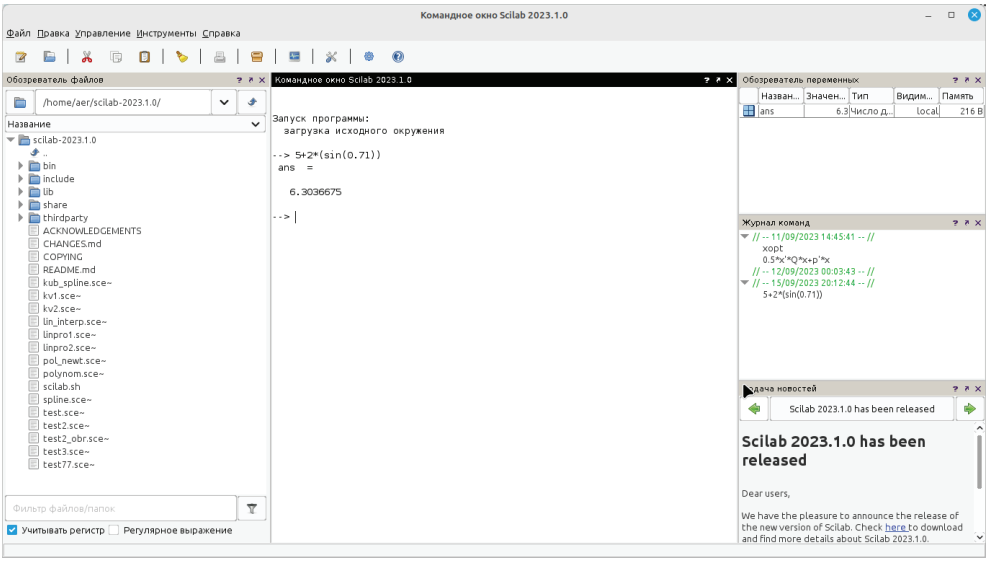

Рис. 1.1. Выполнение элементарной команды в Scilab

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 ↑ и ↓ также управляют курсором, однако в Scilab они имеют другое назначение. Эти клавиши позволяют вернуть в командную строку ранее введённые 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 т. д. 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 её 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 её 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.

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 приложениями пакета.

1.3 Редактирование и отладка файлов-сценариев

Файл-сценарий – это список команд Scilab, сохраненный на диске. Для подготовки, редактирования и отладки файлов-сценариев служит специальный *редактор* SciNotes, который можно вызвать, выполнив команду главного меню **Файл** ⇒ **Новый**. В результате работы этой команды будет создан новый файл-сценарий. По умолчанию он имеет имя *Безымянный документ 1.sce*.

Современный текстовый редактор SciNotes выглядит стандартно, имеет заголовок, меню, панели инструментов, строку состояния, но вместе с тем имеет достаточно много возможностей, рекомендуем с ним ознакомиться. Его интерфейс полностью переведён на русский язык.

Ввод текста в окно редактора файла-сценария осуществляется по правилам, принятым для команд Scilab. Рисунок 1.2 содержит пример ввода команд для решения квадратного уравнения $3x^2 - 4x - 5 = 0$. Точка с запятой «;» ставится после тех команд, которые не требуют вывода значений.

Для *сохранения* введённой информации необходимо выполнить команду **Файл** ⇒ **Сохранить**. Если информация сохраняется впервые, то появится

окно **Сохранить как...**. Ввод имени в поле **File Name** и щелчок по кнопке **OK** приведет к сохранению информации, находящейся в окне редактора. Файлы-сценарии сохраняют с расширением *.sce*. *Открывает* ранее созданный файл команда главного меню **Файл** ⇒ **Открыть**.

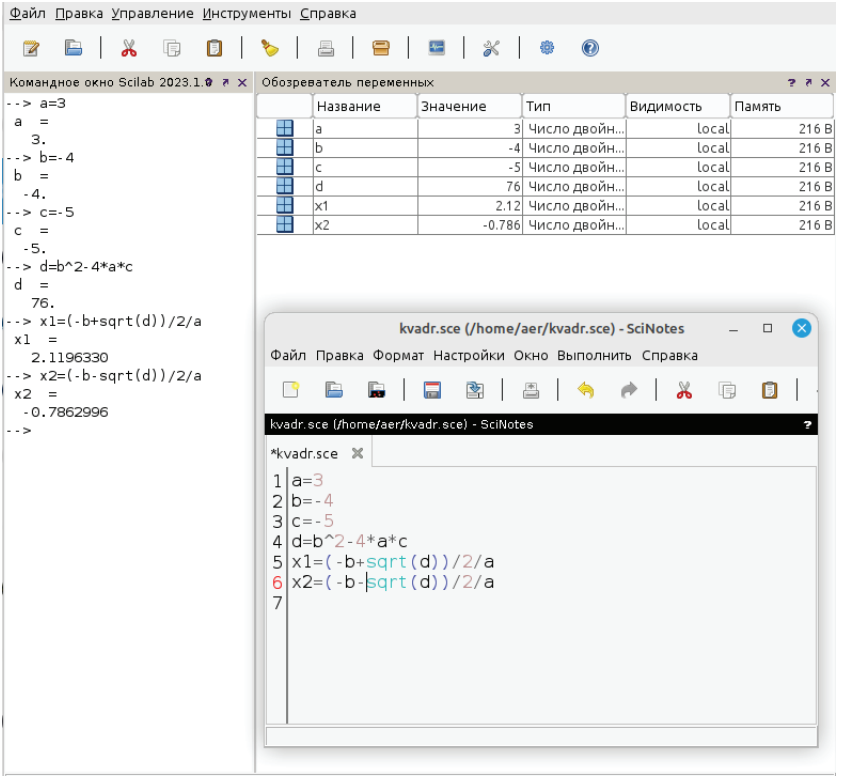

Рис. 1.2. Выполнение файла-сценария Scilab

На рис. 1.2 кроме фрагмента рабочей области и окна редактора показано окно обозревателя переменных. В этом окне пользователь может в любой момент посмотреть значение любой используемой в данной сессии переменной.

Запустить файл-сценарий можно из пункта меню **Выполнить**:

- ...файла без отображения команд (**Ctrl**+**Shift**+**E**), команда запускает файл-сценарий без отображения команд в командном окне **Scilab**;
- ...файла с отображением команд (**Ctrl**+**L**), команда запускает файл-сценарий с отображением команд, команды без «;» выводят результат в командном окне **Scilab**;
- ...до курсора с отображением команд (**Ctrl**+**E**), файл-сценарий выполняется до курсора, после приостановки программы в окне обозревателя переменных можно посмотреть значения любой переменной, этот режим, по существу, является отладочным;

- сохранить и выполнить (**F5**);
- сохранить и выполнить все файлы (**Ctrl**+**F5**).

Отметим, что редактор SciNotes имеет возможность работы со множеством окон (пункт меню **Окно**), обладает принятыми для текстовых редакторов приемами редактирования и поиска (пункт меню **Правка**). Кроме того, можно выполнить настройку среды редактора SciNotes (пункт меню **Настройки**), вызвать справочную информацию (пункт меню **Справка**).

Выйти из режима редактирования можно, просто закрыв окно SciNotes или выполнив команду **Файл** ⇒ **Выйти из SciNotes** (**Ctrl**+**Q**).

1.4 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 1.2. Пример использования комментария

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

При написании кода в редакторе SciNotes можно использовать следующий синтаксис:

```
/*Это комментарий, который
может занимать несколько строк*/
```

1.5 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 1.3. 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 1.4. Выражение, расположенное на нескольких строках

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

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

Листинг 1.5. Использование «;»

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

1.6 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. Таким образом, 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. Система различает большие и малые буквы в именах переменных. ABC, abc, Abc, aBc – это разные имена. 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 или двойные кавычки.

Если символ «;» в конце выражения отсутствует, то в качестве результата выводится имя переменной и её значение. Наличие символа «;» передает управление следующей командной строке. Это позволяет использовать имена переменных для записи промежуточных результатов в память компьютера.

Листинг 1.6. 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
Неопределённая переменная: 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;
```

которая отменяет определения всех переменных данной сессии. Для отмены определения конкретной переменной используют команду

```
clear имя_переменной;
```

Далее приведены примеры применения этой команды.

Листинг 1.7. Пример использования команды `clear`

```
-->//Определение переменных x и y
--> x=3; y=-1;
-->//Отмена определения переменной x
--> clear x
-->//Переменная x не определена
--> x
Неопределённая переменная: 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
Неопределённая  переменная:  a
-->b
Неопределённая  переменная:  b
```

1.7 Системные переменные Scilab

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

Листинг 1.8. Пример работы с переменной `ans`

```
--> 25.7-3.14
ans =
22.56
--> //Значение системной переменной равно 22.5600
--> 2*ans
ans =
45.12
--> //Значение системной переменной увеличено вдвое
--> x=ans^0.3
x =
3.1355283
--> ans
ans =   45.12
--> //После использования в выражении значение
--> //системной переменной не изменилось и равно 45.12
```

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` – неопределённый результат (0/0, ∞/∞ и т. п.);
- `%eps` – условный ноль `%eps=2.220E-16`.

Все перечисленные переменные можно использовать в математических выражениях.

Листинг 1.9. 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 1.10. Неправильное обращение к переменной `%pi`

```
-->sin(pi/2)
Неопределённая переменная: pi
```

1.8 Числовые типы данных и представление результатов вычислений в Scilab

1.8.1 Целые числа в Scilab

Современный Scilab поддерживает работу с одно-, двух-, четырёх- и восьмибайтными целыми числами.

Для преобразования данных числового типа в определённый целочисленный предусмотрены следующие функции:

- **y=int8(x)** – преобразование в однобайтное представление целого числа, диапазон чисел −128...127;
- **y=uint8(x)** – преобразование в однобайтное представление беззнакового целого числа, диапазон чисел 0...255;
- **y=int16(x)** – преобразование в двухбайтное представление целого числа, диапазон чисел −32 768 (-2^{15})...32 767 ($2^{15}-1$);
- **y=uint16(x)** – преобразование в двухбайтное представление беззнакового целого числа, диапазон чисел 0...65 535 ($2^{16}-1$);
- **y=int32(x)** – преобразование в четырёхбайтное представление целого числа, диапазон чисел −2 147 483 648 (-2^{31})...−2 147 483 647 ($2^{31}-1$);
- **y=uint32(x)** – преобразование в четырёхбайтное представление беззнакового целого числа, диапазон чисел 0...4 294 967 295 ($2^{32}-1$);
- **y=int64(x)** – преобразование в восьмибайтное представление целого числа, диапазон чисел

  −9 223 372 036 854 775 808 (-2^{63})...−9 223 372 036 854 775 807 ($2^{63}-1$);
- **y=uint64(x)** – преобразование в восьмибайтное представление беззнакового целого числа, диапазон чисел 0...18 446 744 073 709 551 615 ($2^{64}-1$).

Здесь *x* – матрица любого числового типа, *y* – матрица соответствующего целочисленного типа.

1.8.2 Представление вещественных чисел в Scilab

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). Чтобы привести 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 1.11. 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, определённой пользователем.

Листинг 1.12. 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 1.13. 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)
1123.456789
```

Для преобразования к вещественному типу используется функция

```
y=double(x)
```

Здесь `x` – матрица любого числового типа, `y` – матрица вещественного типа double[1].

1.8.3 Представление комплексных чисел в Scilab

Комплексные числа в Scilab представляются в виде пары вещественных чисел. Предопределённая константа `%i` представляет собой мнимую единицу.

Элементарные функции работают с комплексными числами. В этом случае возвращаемое значение также будет комплексным. Ниже приведены наиболее часто используемые функции:

[1] Совпадает с типом **double** стандартов языков C(C++).

- **complex** – создаёт комплексное число;
- **real** – возвращает действительную часть комплексного числа;
- **imag** – возвращает мнимую часть комплексного числа;
- **imult** – умножает число на мнимую единицу;
- **isreal** – проверяет отсутствие мнимой части.

Для примера присвоим переменной **x** значение **1+i** и выполним над этой переменной несколько простых операций.

```
// Создание комплексного числа с помощью %i
-->x = 1 + %i
x = 1. + i

//Вывод сопряженного числа
-->x'
ans = 1. - i

//Вывод вещественной части числа
-->real(x)
ans = 1.

//Вывод мнимой части сопряженного числа
-->imag(x')
ans = - 1.

// Проверка равенства (1 + i)(1 - i) = 1 - i^2 = 2
-->x * x'
ans = 2.

// Создание комплексного числа с помощью функции
--> complex(2,3)
ans  = 2. + 3.i

// Создание комплексного числа без мнимой части
--> m=complex(2)
m = 2.

//Проверка, является ли число комплексным
--> isreal(m)
ans = F
```

1.9 Функции в Scilab

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

- *встроенные*;
- *определённые 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.

1.9.1 Элементарные математические функции

Пакет Scilab снабжен достаточным количеством всевозможных встроенных функций, знакомство с которыми будет происходить в следующих разделах. Здесь приведём только элементарные математические функции, используемые чаще всего (табл. 1.1).

Таблица 1.1. 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*
`round(x)`	Округление числа *x*
`nthroot(x,n)`	Корень степени *n* числа *x*
`factorial(x)`	Факториал числа *x*

Задача 1.1

Вычислить значение выражения $z = \sqrt{\left|\sin\left(\frac{x}{y}\right)\right|} \cdot e^{x^y}$.

Листинг 1.14. Вычисление математического выражения

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

1.9.2 Функции, определённые пользователем

Функция, как правило, предназначена для неоднократного использования, она имеет входные параметры и не выполняется без их предварительного определения. 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, 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 вычислено в листинге 1.15).

Листинг 1.15. Пример определения функции 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 1.16. Функция вычисления площади треугольника

```
-->deff('S=G(a,b,c)','p=(a+b+c)/2;S=sqrt(p*(p-a)*(p-b)*(p-c))');
-->G(2,3,3)
ans  = 2.8284271;
```

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 $D = b^2 - 4ac$; $x_{1,2} = \dfrac{-b \pm \sqrt{D}}{2a}$.

Листинг 1.17. Функция решения квадратного уравнения

```
//Функция должна быть объявлена в одной строке
-->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
```

или

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

где `имя1,...,имяN` – список выходных 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 определёнными только внутри функции.

Вообще говоря, функции в Scilab играют роль подпрограмм. Поэтому целесообразно набирать их тексты в редакторе и сохранять в виде отдельных файлов. Причём 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...**.

Задача 1.2.
Решить кубическое уравнение.

Кубическое уравнение

$$ax^3 + bx^2 + cx + d = 0 \tag{1.1}$$

после деления на a принимает канонический вид:

$$x^3 + rx^2 + sx + t = 0, \tag{1.2}$$

где

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

В уравнении (1.2) сделаем замену

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

и получим следующее приведённое уравнение:

$$y^3 + py + q = 0, \tag{1.3}$$

где

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

Число действительных корней приведённого уравнения (1.3) зависит от знака дискриминанта $D = \left(\frac{p}{3}\right)^3 + \left(\frac{q}{2}\right)^3$ (табл. 1.2).

Таблица 1.2. Количество корней кубического уравнения

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

Корни приведённого уравнения могут быть рассчитаны по формулам Кардано:

$$y_1 = u + v, \quad y_2 = \frac{-(u+v)}{2} + \frac{(u-v)}{2}i\sqrt{3}, \quad y_3 = \frac{-(u+v)}{2} - \frac{(u-v)}{2}i\sqrt{3}. \tag{1.4}$$

Здесь

$$u = \sqrt[3]{\frac{-q}{2} + \sqrt{D}}, \quad u = \sqrt[3]{\frac{-q}{2} - \sqrt{D}}.$$

Далее представлен текст функции, реализующий описанный выше способ решения кубического уравнения.

Листинг 1.18. Решение кубического уравнения

```
//файл cub.sce
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 её работы:
-->exec('./cub.sce');
-->[x1,x2,x3]=cub(3,-20,-3,4)
x3  =
0.3880206
x2  =
- 0.5064407
x1  =
6.7850868
```

Глава 2

Программирование в Scilab

В Scilab встроен мощный язык программирования с поддержкой объектов. В этой главе будут описаны возможности структурного программирования.

Как уже отмечалось ранее, 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) выполнить команду меню **Инструменты** ⇒ **Текстовый редактор sciNotes**;
2) в окне редактора **sciNotes** набрать текст программы (см. рис. 2.1);
3) сохранить текст программы с помощью команды **Файл** ⇒ **Сохранить** в виде файла с расширением *sce*, например *file.sce*;

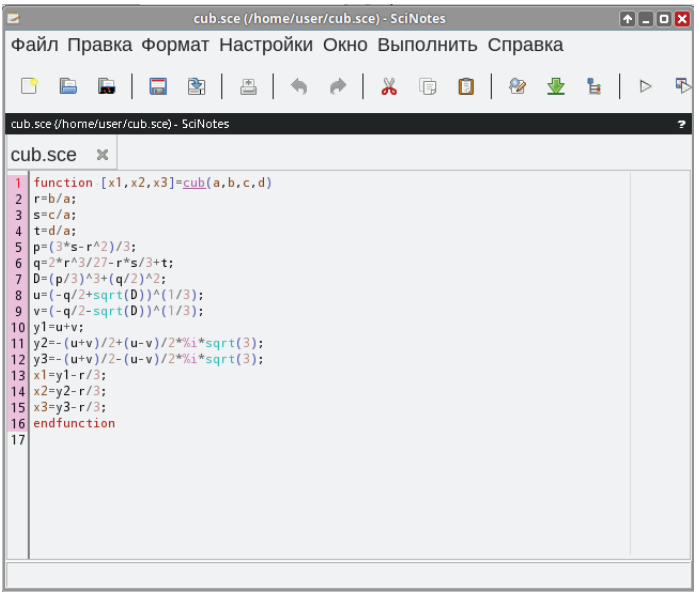

Рис. 2.1. Окно SciNotes с программой решения кубического уравнения

4) после этого программу можно будет вызвать, набрав в командной строке `exec`, например `exec("file.sce")`. Другие способы вызова – воспользоваться командой меню **Файл** ⇒ **Выполнить**... или, находясь в окне SciNotes, выполнить команду **Сохранить и выполнить** (**F5**).

Программный режим достаточно удобен, так как позволяет сохранить разработанный алгоритм в виде файла и повторять его при других исходных данных в иных сессиях. В программных файлах можно использовать операторы языка программирования Scilab (язык программирования Scilab будем называть sci-языком).

2.1 Основные операторы sci-языка

Изучение sci-языка начнем с функций ввода-вывода.

2.1.1 Функции ввода-вывода в Scilab

Даже при разработке простейших программ возникает необходимость ввода исходных данных и вывода результатов. Для ввода исходных данных следует использовать функцию

```
имя_переменной = input('подсказка');
```

Если в тексте программы встречается оператор `input`, то выполнение программы приостанавливается, Scilab выводит в командную строку текст подсказки и переходит в режим ожидания ввода. Пользователь вводит с клавиатуры значение и нажимает клавишу **Enter**. Введённое пользователем значение будет присвоено переменной, имя которой указано слева от знака присваивания.

Для вывода результатов можно использовать функцию следующей структуры: `disp('строка_символов')` или `disp(имя_переменной)`.

Задача 2.1

Создать программу для вычисления значения y по формуле $y = \sin(x)$ при заданном значении x.

Текст программы и результаты её работы приведены далее.

Листинг 2.1. Решение задачи 2.1

```
x=input('Введите значение x = ');
y=sin(x);
disp('Значение y = ');
disp(y);

//Результат работы программы
Значение y = -0.5440211
```

Для организации простейшего ввода в Scilab можно также воспользоваться функцией

```
x=x_dialog('title', 'stroka');
```

x_dialog выводит на экран диалоговое окно с именем title и значением по умолчанию stroka. После этого пользователь может нажать **OK**, и тогда stroka вернется в качестве результата в переменную x. Или можно ввести новое значение вместо stroka, которое и вернется в качестве результата в переменную x. На рис. 2.2 представлено диалоговое окно, которое формируется строкой x=x_dialog('Input X', '5').

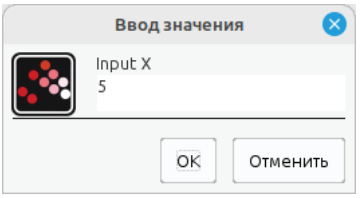

Рис. 2.2. Окно ввода

Функция x_dialog возвращает строковое значение. Для получения числового значения следует полученную строку преобразовать в число. Для этого используется функция evstr. Пример использования функции x_dialog для получения числовых значений:

```
x=evstr(x_dialog('title','stroka'));
```

2.1.2 Форматированный вывод

Форматированный ввод-вывод данных в Scilab осуществляется с помощью функций printf и scanf. Функция

```
printf(строка_форматов, список_выводимых_переменных);
```

выполняет форматированный вывод переменных, указанных в списке, в соответствии со строкой форматов.

Функция

```
переменная = scanf(строка_форматов);
```

выполняет ввод переменных, адреса которых указаны в списке, в соответствии со строкой форматов. Строка форматов содержит символы, которые будут выводиться на экран или запрашиваться с клавиатуры, и так называемые спецификации.

Спецификации – это строки, которые начинаются символом % и выполняют управление форматированием: % флаг ширина . точность модификатор тип.

Параметры флаг, ширина, точность и модификатор в спецификациях могут отсутствовать.

Значения параметров строки преобразования приведены в табл. 2.1.

В строке вывода могут использоваться некоторые специальные символы, приведённые в табл. 2.2.

Таблица 2.1. 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
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.dddde[+|-]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 2.2. Некоторые специальные символы

Символ	Назначение
\b	Сдвиг текущей позиции влево
\n	Перевод строки
\r	Перевод в начало строки без перехода на новую строку
\t	Горизонтальная табуляция
\'	Символ одинарной кавычки
\”	Символ двойной кавычки
\?	Символ «?»

Рассмотрим работу функций на примере следующей задачи.

Задача 2.2

Зная a, b, c – длины сторон треугольника, вычислить площадь S и периметр P этого треугольника.

Входные данные: a, b, c. Выходные данные: S, P. Для вычисления площади применим формулу Герона:

$$S = \sqrt{r \cdot (r-a) \cdot (r-b) \cdot (r-c)},$$

где $r = \frac{a+b+c}{2}$ – полупериметр.

Листинг 2.2. Решение задачи 2.2

```
//Вывод на экран символов 'Введите значение a:'.
mprintf("Введите значение a: ");
//Запись в переменную a значения, введённого с клавиатуры.
a = mscanf("%f");
//Вывод на экран символов 'Введите значение b:'.
mprintf("Введите значение b: ");
//Запись в переменную b значения, введённого с клавиатуры.
b = mscanf("%f");
//Вывод на экран символов 'Введите значение c:'.
mprintf("Введите значение c: ");
//Запись в переменную c значения, введённого с клавиатуры.
c = mscanf("%f");
//Вычисление полупериметра.
r=(a+b+c)/2;
//Вычисление площади треугольника.
S=sqrt(r*(r-a)*(r-b)*(r-c));
//Вывод символов 'S=', значения S и символа табуляции \t.
//Спецификация %5.2f означает, что будет выведено вещественное
//число из пяти знаков, два из которых после точки.
mprintf("S=%5.2f \t",S);
//Вывод символов 'p=', значения выражения 2*r
//и символа окончания строки.
mprintf("p=%5.2f \n",2*r);

//Результат работы:
Введите значение a:
--> 3
Введите значение b:
--> 4
Введите значение  c:
--> 5
S= 6.00         p=12.00
```

2.1.3 Оператор присваивания

Оператор присваивания служит как для определения новой переменной, так и для перезаписи значения уже существующей переменной. Для того чтобы определить новую переменную, достаточно присвоить ей значение:

```
имя_переменной = значение_выражения
```

Чтобы присвоить новое значение уже существующей переменной, используется тот же синтаксис:

```
имя_переменной = значение_выражения
```

Любую переменную Scilab воспринимает как матрицу. В простейшем случае матрица может состоять из одной строки и одного столбца. Работа с массивами и матрицами будет рассмотрена подробнее в следующих разделах. Примеры работы с переменной:

```
--> m=pi
m =  3.1416
--> m
m =  3.1416
--> m(1)
ans =  3.1416
--> m(1,1)
ans =  3.1416
--> m(1,2)
Недопустимый индекс.
--> m(3)
Недопустимый индекс.
--> M=e; M(3,3)=e/2;
--> M
M =
2.71828   0.00000   0.00000
0.00000   0.00000   0.00000
0.00000   0.00000   1.35914
```

2.1.4 Условный оператор

Одним из основных операторов, реализующих ветвление в большинстве языков программирования, является условный оператор. Существуют обычная, сокращённая и расширенная формы этого оператора в языке программирования Scilab.

Сокращённый условный оператор записывают так:

```
if условие
        операторы
end
```

Работает этот оператор следующим образом. Если условие истинно, то выполняются `операторы`, в противном случае управление передаётся оператору, следующему за оператором `if`.

Листинг 2.3. Пример работы условного оператора

```
x=input('x=');
y=input('y=');
z=0;
if (x<y)
        z=x+y;
end;
disp('Значение  Z=');
disp(z);

// Результаты работы программы
x= 3
y= 5
Значение Z= 8
x= 3
y= 3
Значение Z= 0
```

Обычный условный оператор имеет вид:

```
if условие
 операторы1
else
 операторы2
end
```

Здесь `условие` – логическое выражение, `операторы_1`, `операторы_2` – операторы языка или встроенные функции Scilab. Обычный оператор `if` работает по следующему алгоритму: если условие истинно, то выполняются `операторы_1`, если ложно – `операторы_2`.

Задача 2.3

Даны вещественные числа *x* и *y*. Определить, принадлежит ли точка с координатами (*x*; *y*) заштрихованной части плоскости (рис. 2.3).

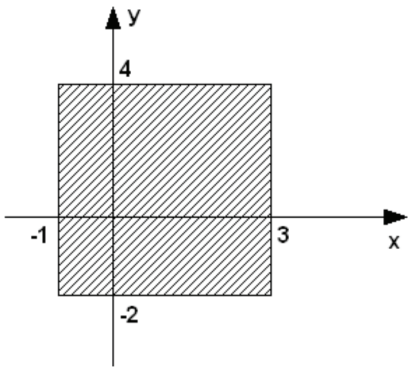

Рис. 2.3. Графическое представление задачи 2.3

Как показано на рис. 2.3, фигура на плоскости ограничена линиями $x = -1$, $x = 3$, $y = -2$ и $y = 4$. Значит, точка с координатами $(x; y)$ будет принадлежать этой фигуре, если будут выполняться следующие условия: $x \geq -1, x \leq 3, y \geq -2$ и $y \leq 4$. Иначе точка лежит за пределами фигуры.

Далее приведены текст программы и результаты её работы.

Листинг 2.4. Решение задачи 2.3

```
x=input('x=');  y=input('y=');
if (x>=-1) & (x<=3) & (y>=-2) & (y<=4)
disp('Точка принадлежит фигуре')
else
disp('Точка не принадлежит фигуре');
end

// Результаты работы программы
x= 3
y= 3
Точка принадлежит фигуре
//-----------------------------
x= 4
y= 4
Точка не принадлежит фигуре
```

Расширенный условный оператор применяют, когда одного условия для принятия решения недостаточно. Синтаксис оператора следующий:

```
if условие_1
 операторы1
elseif условие_2
 операторы2
elseif условие3
 операторы3
...
elseif условие_n
операторы_n
else
операторы
end
```

Расширенный оператор `if` работает так. Если `условие_1` истинно, то выполняются `операторы_1`, иначе проверяется `условие_2`, если оно истинно, то выполняются `операторы_2`, иначе проверяется `условие_3` и т. д. Если ни одно из условий по веткам `elseif` не выполняется, то выполняются операторы по ветке `else`.

Рассмотрим использование расширенного условного оператора на примере.

Задача 2.4

Дано вещественное число x. Для функции, график которой приведён на рис. 2.4, вычислить $y = f(x)$.

Аналитически функцию, представленную на рис. 2.4, можно записать так:

$$\begin{cases} 4, & x \le -2 \\ 1, & x \ge 1 \\ x^2, & -2 < x < 1 \end{cases}.$$

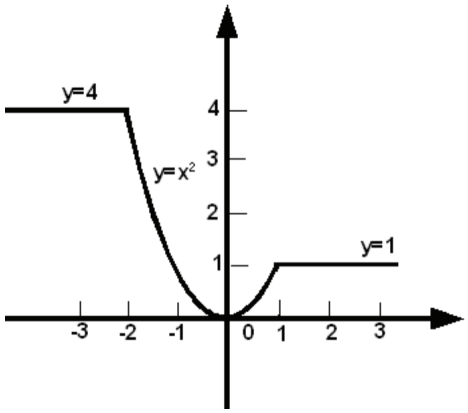

Рис. 2.4. Графическое представление задачи 2.4

Составим словесный алгоритм решения этой задачи.

1. Начало алгоритма.
2. Ввод числа x (аргумент функции).
3. Если значение x меньше либо равно -2, то переход к п. 4, иначе переход к п. 5.
4. Вычисление значения функции: $y = 4$, переход к п. 8.
5. Если значение x больше либо равно 1, то переход к п. 6, иначе переход к п. 7.
6. Вычисление значения функции: $y = 1$, переход к п. 8.
7. Вычисление значения функции: $y = x^2$.
8. Вывод значений аргумента x и функции y.
9. Конец алгоритма.

Текст программы будет иметь вид:

Листинг 2.5. Решение задачи 2.4

```
x=input('x=');
if x<=-2
y=4;
elseif x>=1
y=1;
else
y=x^2;
end;
```

```
disp('y=');
disp(y);

// Результаты работы программы
x= 2
y= 1
//-----------------------------
x= -3
y= 4
//-----------------------------
x= 0.5
y=  0.25000
```

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).

Задача 2.5

Создать программу для решения 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 $y_1 \geq 0$ и $y_2 \geq 0$, 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 в листинге 2.6, её вызов и результаты – в листинге 2.7.

Листинг 2.6. 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('Нет действительных корней');
else
        //иначе вычисление корней соответствующего
        //квадратного уравнения.
        y1=(-b+sqrt(d))/2/a;
        y2=(-b-sqrt(d))/2/a;
        //Если оба корня отрицательны,
        if (y1<0)&(y2<0)
                //вывод сообщения об отсутствии действительных корней.
                disp('Нет действительных корней');
        //иначе, если оба корня положительны,
        elseif (y1>=0)&(y2>=0)
                //вычисление четырёх корней.
                disp('Четыре действительных корня');
                x1=sqrt(y1);
                x2=-x1;
                x3=sqrt(y2);
                x4=-x2;
                disp(x1,x2,x3,x4);
        //Иначе, если оба условия (y1<0)&(y2<0) и (y1>=0)&(y2>=0)
        //не выполняются,
        else
                //то вывод сообщения
                disp('Два действительных корня');
                //Проверка знака y1.
                if y1>=0
                        //Если y1 положителен, то
                        //вычисление двух корней биквадратного
                        //уравнения извлечением корня из y1,
                        x1=sqrt(y1);
                        x2=-x1;
                        disp(x1);
                        disp(x2);
                //иначе (остался один вариант - y2 положителен)
                //вычисление двух корней биквадратного уравнения
                //извлечением корня из y2.
                else
                        x1=sqrt(y2);
                        x2=-x1;
                        disp(x1);
                        disp(x2);
                end
        end
end
```

Листинг 2.7. Решение биквадратного уравнения

```
-->exec("l1.sci");
a=-->-6
b=-->9
c=-->-1
Четыре действительных корня
0.3476307
1.1743734
- 0.3476307
0.3476307
```

Найти корни биквадратного уравнения можно и без оператора if, воспользовавшись тем, что в Scilab определены операции над комплексными числами (см. листинг 2.8).

Листинг 2.8. Решение биквадратного уравнения

```
a=input('a=');
b=input('b=');
c=input('c=');
d=b*b-4*a*c;
y1=(-b+sqrt(d))/2/a;
y2=(-b-sqrt(d))/2/a;
x1=sqrt(y1);
x2=-x1;
x3=sqrt(y2);
x4=-x3;
disp(x1,x2,x3,x4);
```

Листинг 2.9. Результат работы программы

```
-->a=3
-->b=8
-->c=-1
-1.6692213i
1.6692213i
-0.3458800
0.3458800
```

2.1.5 Оператор альтернативного выбора

Ещё одним способом организации разветвлений является оператор альтернативного выбора `select`:

```
select параметр
case значение1 then операторы1
case значение2 then операторы2
...
else операторы
end
```

Оператор `select` работает следующим образом: если значение параметра равно `значению1`, то выполняются `операторы1`, иначе, если параметр равен `значению2`, выполняются `операторы2`. В противном случае, если значение `параметра` совпадает со `значением3`, 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 оператора `select`, применяя только `if`, но использование оператора альтернативного выбора `select` делает программу более компактной.

Рассмотрим применение оператора `select` на примере решения следующей задачи.

Задача 2.6

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 т. д. Вычислить остаток от деления числа *x* на *k* можно с помощью функции `pmodulo`.

```
pmodulo(x,k);
```

Эта функция в качестве результата возвращает остаток от деления числа *x* на число *k*.

При построении алгоритма необходимо использовать семь условных операторов.

Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта (см. листинг 2.10). Вызов программы представлен в листинге 2.11.

Листинг 2.10. Решение задачи 2.6

```
D=input('Введите число от 1 до 31:  ');
//Вычисление остатка от деления D на 7, сравнение его с числами
//от 0 до 6.
select pmodulo(D,7)
case 1  then disp('Понедельник');
case 2 then disp('Вторник');
case 3 then disp('Среда');
case 4 then disp('Четверг');
case 5 then disp('Пятница');
case 6 then disp('Суббота');
else
disp('Воскресенье');
end
```

Листинг 2.11. Вызов функции для решения задачи

```
-->exec('l2.sci');
Введите число от 1 до 31
-->19
Пятница
```

Оператор альтернативного выбора также может иметь следующую структуру:

```
switch параметр
case значение1
 операторы1
case значение2
 операторы2
case значение3
 операторы3
...
otherwise
операторы
end
```

Оператор `switch` работает аналогично оператору `select`. Различие только в синтаксисе.

Листинг 2.12. Решение задачи 2.6 с помощью `switch`

```
D=input('Введите число от 1 до 31: ');
//Вычисление остатка от деления D на 7, сравнение его с числами
//от 0 до 6.
switch pmodulo(D,7)
case 1  then disp('Понедельник');
case 2 then disp('Вторник');
case 3 then disp('Среда');
case 4 then disp('Четверг');
case 5 then disp('Пятница');
case 6 then disp('Суббота');
otherwise
disp('Воскресенье');
end
```

2.1.6 Оператор цикла while

Оператор цикла с предусловием в языке программирования Scilab имеет вид:

```
while выражение
операторы
end
```

Работает цикл с предусловием следующим образом. Вычисляется значение условия выражение. Если оно истинно, выполняются операторы. В противном случае цикл заканчивается, и управление передаётся оператору, следующему за телом цикла. Выражение вычисляется перед каждой итерацией цикла. Если при первой проверке выражение ложно, цикл не выполнится ни разу. Выражение должно быть переменной или логическим выражением.

Задача 2.7

Дано натуральное число *N*. Определить количество цифр в числе.

Для того чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело. Например, пусть $N = 12345$, тогда количество цифр $kol = 5$. Результаты вычислений сведены в табл. 2.3.

Таблица 2.3. Определение количества цифр числа

kol	*N*
1	12345
2	12345/10=1234
3	1234/10=123
4	123/10=12
5	12/10=1
5	1/10=0

Текст программы, реализующей данную задачу, можно записать так.

Листинг 2.13. Решение задачи 2.7

```
N = input('N=');
// Сохранить значение переменной N.
M = N;
// Число содержит хотя бы одну цифру.
kol=1;
// Выполнять тело цикла, пока частное от деления
// M на 10, округлённое до целого, больше 0.
while round(M/10) > 0
  // Счётчик количества цифр
  kol = kol + 1;
  // Изменение числа.
  M = round(M/10);
end
disp('kol=');
disp(kol);

// Результат работы программы
N= 12345678
kol= 8
```

2.1.7 Оператор for

Цикл с известным числом повторений реализован с помощью оператора `for`:

```
for параметр = начальное_значение:шаг:конечное_значение
        операторы
end
```

Выполнение цикла начинается с присвоения `параметру` цикла `начального_значения`. Затем следует проверка, не превосходит ли параметр цикла `конечное_значение`. Если результат проверки утвердительный, цикл считается завершённым и управление передаётся следующему за телом цикла оператору. В противном случае выполняются `операторы` в цикле. Далее `параметр` увеличивает своё значение на значение `шага` и снова производится проверка – не превзошло ли значение `параметра` цикла `конечное_значение`. В случае положительного ответа алгоритм повторяется, в противном – цикл завершается.

Если шаг цикла равен 1, то оператор записывают так:

```
for параметр = начальное_значение:конечное_значение
операторы
end
```

Задача 2.8

Дано натуральное число N. Определить K – количество делителей этого числа, которые меньше его. Например, для $N = 12$ делители 1, 2, 3, 4, 6. Количество делителей $K = 5$.

Для решения поставленной задачи нужно реализовать следующий алгоритм: в переменную K, предназначенную для подсчёта количества делителей заданного числа, поместить значение, которое не влияло бы на результат, т. е. ноль. Далее организовать цикл, в котором изменяющийся параметр i выполняет роль возможных делителей числа N. Если заданное число делится нацело на параметр цикла, это означает, что i является делителем N и значение переменной K следует увеличить на единицу. Цикл необходимо повторить только $N/2$ раз[1].

Листинг 2.14. Количество делителей числа (к примеру 2.8)

```
N = input('N = ');
// Количество делителей числа
K = 0;
for i = 1 : N / 2
  // Если N делится нацело на i, то
  if pmodulo(N, i) == 0
     // увеличить счётчик на единицу.
```

[1] Если делитель числа меньше самого числа, значит, он не превосходит его половины. – *Прим. ред.*

```
    K=K+1;
  end
end
disp('K = ');
disp(K);

// Результат работы программы
N = 12
K = 5
```

2.1.8 Операторы передачи управления

Операторы передачи управления принудительно изменяют порядок выполнения команд. В языке программирования Scilab таких операторов два. Операторы `break` и `continue` используют только внутри циклов. Так, оператор `break` осуществляет немедленный выход из циклов `while`, `for`, и управление передаётся оператору, находящемуся непосредственно за циклом. Оператор `continue` начинает новую итерацию цикла, даже если предыдущая не была завершена.

Задача 2.9

Дано натуральное число *N*. Определить, является ли оно простым. Натуральное число *N* называется простым, если оно делится нацело без остатка только на единицу и *N*. Число 13 – простое, так как делится только на 1 и 13, 12 не является простым, так как делится на 1, 2, 3, 4, 6 и 12.

Алгоритм решения этой задачи заключается в том, что число *N* делится на параметр цикла *i*, изменяющийся в диапазоне от 2 до корня из числа *N*. Если среди значений параметра не найдётся ни одного числа, делящего заданное число нацело, то *N* – простое число, иначе оно таковым не является. Разумно предусмотреть в программе два выхода из цикла. Первый – естественный, при исчерпании всех значений параметра, а второй – досрочный, с помощью оператора `break`. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра.

Текст программы приведён в листинге 2.15.

Листинг 2.15. Является ли число простым (к задаче 2.9)

```
N=input('Введите число ');
// Предполагаем, что число N является простым (pr=1).
pr=1;
// Перебираем все возможные делители числа N от 2 до N/2.
for i=2:sqrt(N)
// Если N делится на i,
  if pmodulo(N,i)==0
    // то число N не является простым (pr=0)
    pr=0;
    // и прерывается выполнение цикла.
    break;
```

```
  end
end
// Если pr равно 1, то N --- простое число.
if pr==1
  disp('ПРОСТОЕ ЧИСЛО')
// Если pr равно 0, то N --- не является простым.
else
  disp('НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ')
end

// Результаты работы программы
// Вводим сначала число 12, затем 13
Введите число 12
НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ
//---------------------------
Введите число 13
ПРОСТОЕ ЧИСЛО
```

2.2 Обработка массивов и матриц в Scilab

Для работы со 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 определённом 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, на пересечении которых находится необходимый элемент.

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

Для работы с массивами в Scilab предусмотрено большое количество встроенных функций, подробно будем говорить о них позже. В этой главе рассмотрим основные алгоритмы поэлементной обработки массивов и матриц и их реализацию на sci-языке.

2.2.1 Ввод-вывод массивов и матриц

Ввод массивов и матриц следует организовывать поэлементно. В листингах 2.16 и 2.17 приведены программы ввода элементов массивов и матриц на sci-языке.

Листинг 2.16. Ввод элементов массива

```
N=input('N=');
disp('Ввод массива x');
for i=1:N
  x(i)=input('X=');
end
disp(x);
```

Листинг 2.17. Ввод элементов матрицы

```
N=input('N=');
M=input('M=');
disp('Ввод матрицы');
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].

2.2.2 Вычисление суммы и произведения элементов массива (матрицы)

В sci-языке существует функция

```
length(x)
```

Данная функция позволяет узнать размер массива *x*, т. е. количество элементов в нём.

Аналогично для матриц существует функция

```
size(A)
```

которая позволяет узнать размеры матрицы **A**.

Теперь рассмотрим алгоритм нахождения суммы элементов массива. Изначально сумма равна 0 ($s = 0$). Затем к *s* добавляем первый элемент массива и результат записываем в переменную *s*. Далее к переменной *s* добавляем второй элемент массива и результат записываем в *s* и потом аналогично добавляем к *s* остальные элементы массива.

Листинг 2.18. Вычисление суммы элементов массива

```
//Записываем в переменную s число 0.
s=0;
//Перебираем все элементы массива
```

[1] Напоминаем читателю, что если после имени массива (матрицы) не поставить точку с запятой, то массив (матрица) будет выведен в окне Scilab.

```
for i=1:length(x)
  //накопление суммы
  s=s+x(i);
end
```

При нахождении суммы элементов матрицы последовательно суммируем элементы всех строк.

Листинг 2.19. Вычисление 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);
```

Алгоритм нахождения произведения элементов следующий: на первом этапе начальное значение произведения равно 1 ($p = 1$). Затем p последовательно умножается на очередной элемент, результат записывается в p и т. д. В листингах 2.20–2.21 представлены элементы программ, реализующие эти алгоритмы.

Листинг 2.20. Вычисление произведения элементов массива

```
p=1;
for i=1:length(x)
p=p*x(i);
end
```

Листинг 2.21. Вычисление произведения элементов матрицы

```
//Начальное значение произведения 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
```

2.2.3 Поиск максимального (минимального) элемента массива (матрицы)

Алгоритм решения задачи поиска максимума и его номера в массиве следующий. Пусть в переменной с именем `Nmax` хранится номер максимального эле-

мента массива. Предположим, что первый элемент является максимальным, и запишем его номер в переменную Nmax. Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем текущее значение индекса i в переменную Nmax. В листинге 2.22 представлен фрагмент программы поиска максимума.

Листинг 2.22. Поиск максимума

```
//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 элемента с номером Nmax,
  if x(i)>x(Nmax)
    //то в Nmax записываем номер текущего элемента как нового максимального
    Nmax=i;
  end;
end;
```

Алгоритм поиска минимального элемента в массиве будет отличаться от приведённого выше лишь тем, что в операторе if знак поменяется с «>» на «<».

В листинге 2.23 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 цикла 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 минимальным (максимальным) элементом.

Листинг 2.23. Поиск минимального элемента матрицы и его индексов

```
//В Nmin и Lmin записываем число 1.
Nmin=1; Lmin=1;
for i=1:N
  for j=1:M
  //Если текущий элемент матрицы меньше min,
    if a(i,j)<a(Nmin,Lmin)
      //то текущий элемент объявляем минимальным,
      Nmin=i;
      Lmin=j;
    end;
  end;
end;
```

2.2.4 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 в порядке возрастания значений его элементов, если

$$X[1] \le X[2] \le \cdots \le X[n],$$

и в порядке убывания, если

$$X[1] \ge X[2] \ge \cdots \ge X[n].$$

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 повторять алгоритм до тех пор, пока не упорядочим массив.

Алгоритм сортировки по убыванию будет отличаться от приведённого заменой знака «>» на «<».

В листинге 2.24 представлен фрагмент программы на sci-языке, реализующий алгоритм сортировки по возрастанию.

Листинг 2.24. Сортировка 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);
```

2.2.5 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. В листинге 2.25 приведен фрагмент программы, реализующий описанный алгоритм.

Листинг 2.25. Удаление *m*-го элемента из массива

```
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 преобразованного массива.
disp(x);
```

2.2.6 Примеры задач

В Scilab есть встроенные функции вычисления суммы (`sum`), произведения (`prod`) элементов массива (матрицы), поиска максимума (`max`) и минимума (`min`), сортировки (`gsort` – быстрая сортировка), но лишь понимание алгоритмов работы функций позволит решать нестандартные задачи обработки массивов и матриц. Рассмотрим решение нескольких практических задач.

Задача 2.10

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 в листинге 2.26. Использование переменной `min1` в программе обусловлено тем, что в Scilab есть встроенная функция `min`. Применение переменной с именем `min` не позволит использовать встроенную функцию `min`.

Листинг 2.26. Решение задачи 2.10

```
//Ввод количества элементов в массиве y.
N=input('N=');
disp('Ввод массива 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. При удалении происходит
    //сдвиг элементов на 1 влево, 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);
```

Задача 2.11

Найти наименьшее простое число в массиве $x(n)$, если таких чисел несколько, определить их количество.

Листинг 2.27 содержит программу решения этой задачи с подробными комментариями.

Листинг 2.27. Поиск наименьшего простого числа (пример 2.11)

```
// Ввод размера массива.
N = input('N= ');
// Цикл для ввода элементов массива.
for i=1:N
x(i)=input('');
end
// pr=0 - простых чисел нет. pr=1 - простые числа есть.
pr = 0;
for i = 1:N
 // L используется при проверке, является ли данный элемент
 // массива простым числом, L=1, пока не встретились делители
 // числа, L станет равным 0, если встретятся делители числа.
 L=1;
 // Цикл для проверки, является ли число простым
 //(поиск возможных делителей числа).
 for j=2:x(i)/2
    // Если x(i) делится на j, то найден делитель числа x(i)
    if pmodulo(x(i),j)==0
      // x(i) не является простым, поэтому L=0 и
      L=0;
      // выходим из цикла по j с помощью оператора break.
      break;
    end;
end;

// Проверяем значение переменной L,
// если L=1, то число x(i) - простое.
  if L==1
    // Если при этом pr=0, то это означает, что встретилось
    // первое простое число
    if pr==0
      // в массиве найдены простые числа.
      pr=1;
      // записываем в переменную Min, т. е. предполагаем,
      // что x(i) и является минимальным простым
      Min=x(i);
      // количество минимумов равно 1,
      k=1;
    // Иначе, если pr=1, т. е. встретилось очередное
    // (не первое) простое число,
    else
      // сравниваем x(i) с Min, если x(i)<Min,
```

```
      if x(i)<Min
        // этот элемент записываем в переменную Min,
        Min=x(i);
        // количество минимумов равно 1.
        k=1;
      else
        // Если очередной элемент x(i) равен Min,
        if x(i)==Min
          // то количество минимумов увеличивается.
          k=k+1;
        end;
      end;
    end;
  end;
end;
// Если после перебора всех элементов массива переменная pr
// осталась равной 0 (простых чисел нет),
if pr == 0
  // то вывод соответствующего сообщения.
  disp('Простых чисел нет!!!!')
// Если были простые числа,
else
  // то вывод min (минимальное простое число)
  disp(Min);
  // и k (количество минимумов)
  disp(k);
end;
```

Задача 2.12

В квадратной матрице $A(N, N)$ обнулить столбцы, в которых элемент на побочной диагонали является максимальным.

Алгоритм решения этой задачи состоит в следующем: в каждом столбце находим максимальный элемент и проверяем; если наибольший элемент расположен на побочной диагонали, то обнуляем все элементы в том столбце. Элемент находится на побочной диагонали, если его номер строки i и номер столбца j связаны соотношением $i + j = n + 1$.

Листинг 2.28 содержит текст программы для решения поставленной задачи.

Листинг 2.28. Решение задачи 2.12

```
N=input('N='); // Ввод размера квадратной матрицы.
for i=1:N      // Ввод квадратной матрицы.
  for j=1:N
    A(i,j)=input('');
  end
end
// Цикл по всем столбцам матрицы, в каждом из которых ищем
// номер максимального элемента.
```

```
for j=1:N
  // Предполагаем, что первый элемент в столбце максимальный
  // В nmax будет храниться номер максимального элемента j-го столбца
  nmax=1;
  for i=2:N // В цикле по i перебираем все элементы j-го столбца.
    // Если очередной элемент больше элемента в строке nmax,
    if A(i,j)>A(nmax,j)
      nmax=i; // то номер его строки и становится nmax;
    end;
  end;
  // Если в текущем столбце максимум находится на побочной диагонали,
  if nmax==N+1-j
    for i=1:N // то обнуляем все элементы в этом столбце.
      A(i,j)=0;
    end;
  end;
end;
disp(A);
```

2.3 Работа с файлами в Scilab

Scilab предоставляет широкие возможности для работы с файлами. Текстовыми называют файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «конец строки». Конец самого файла обозначается символом «конец файла». При записи информации в текстовый файл все данные преобразуются к символьному типу и хранятся в символьном виде. Этот файл можно просмотреть с помощью любого текстового редактора.

2.3.1 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)
```

где `file` – строка, в которой хранится полное имя файла; `mode` – режим работы с файлом:

- `'r'` – текстовый файл открывается в режиме чтения;
- `'w'` – открывается пустой текстовый 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;
- `'r+'` – открывается текстовый файл, который будет использоваться в режиме чтения и записи;
- `'w+'` – создаваемый пустой текстовый 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.

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 ≠ 0, то это значит, что файл открыть не удалось.

2.3.2 Функция записи в текстовый файл 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 табл. 2.1. В строке вывода могут использоваться некоторые специальные символы, приведённые в табл. 2.2.

2.3.3 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 разделёнными, 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.

2.3.4 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 в листинге 2.29.

Листинг 2.29. Создание текстового файла

```
//В текстовый файл abc.txt запишем размеры матрицы N и M
//и саму матрицу 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, разделённые 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. 2.5.

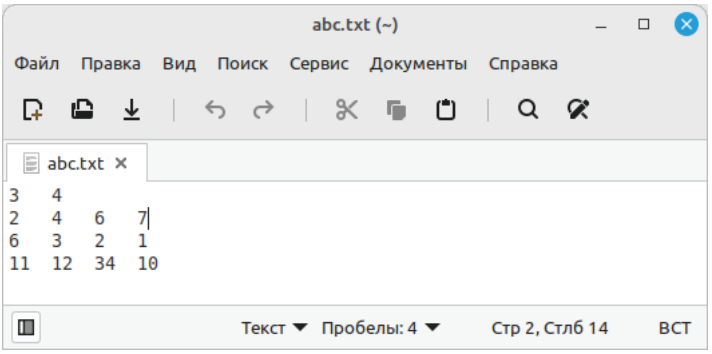

Рис. 2.5. Созданный текстовый файл

Программа чтения данных из этого текстового файла приведена в листинге 2.30.

Листинг 2.30. 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);
disp(A);

//Результат работы программы
2.    4.    6.    7.
6.    3.    2.    1.
11.   12.   34.   10.
```

2.3.5 Примеры решения задач

Рассмотрим использование рассмотренных выше функций на простых примерах.

Задача 2.13

Поменять местами элементы, расположенные на главной и побочной диагоналях квадратной матрицы $A(N, N)$. Исходную и преобразованную матрицы вывести в текстовый файл *prim.txt*.

Далее приведена программа решения задачи с комментариями.

Листинг 2.31. Решение к примеру 2.13

```
// Ввод размеров матрицы.
N=input('N=');
// Ввод элементов матрицы.
for i=1:N
  for j=1:N
    A(i,j)=input('A=');
  end
end
// Открыть файл для записи (создать новый пустой файл).
f=mopen('prim.txt','wt');
// Вывод в файл строки 'Исходная матрица A'
// и перевод курсора на новую строку (символ \n).
mfprintf(f,'Исходная матрица A\n');
for i=1:N // Цикл для построчной записи элементов матрицы в файл.
  for j=1:N // Цикл для записи в файл i-й строки матрицы.
```

```
    mfprintf(f,'%f\t',A(i,j)); // Запись очередного элемента
    // A(i,j) и символа табуляции в файл.
  end
  mfprintf(f,'\n'); // После записи очередной строки переход к
  // следующей строке файла.
end
for i=1:N // В каждой строке матрицы
  b=A(i,i); // поменять местами элементы, расположенные
  A(i,i)=A(i,N+1-i); // на главной и побочной диагоналях.
  A(i,N+1-i)=b;
end
// Вывод в файл строки 'Матрица A после преобразования'
// и перевод курсора на новую строку (символ \n).
mfprintf(f,'Матрица A после преобразования\n');

for i=1:N // Двойной цикл для вывода матрицы в файл.
  for j=1:N
    mfprintf(f,'%f\t',A(i,j));
  end
  mfprintf(f,'\n');
end
mclose(f); // Закрытие файла после записи в него необходимой информации.
```

В результате работы программы создан файл *prim.txt*, который можно открыть при помощи обычного текстового редактора (см. рис. 2.6).

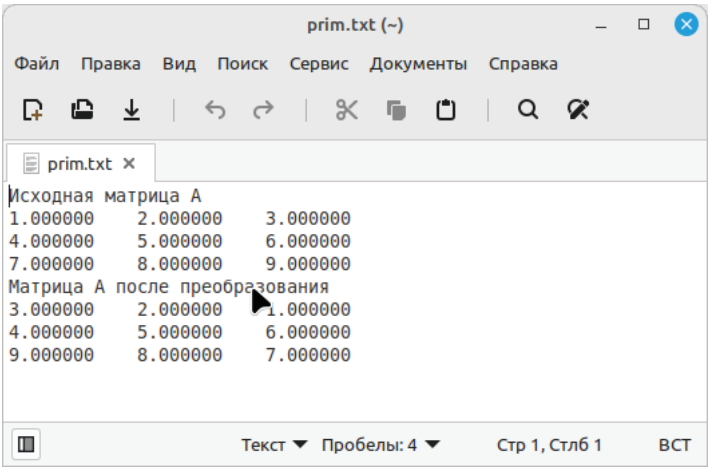

Рис. 2.6. Файл *prim.txt*

Обратите внимание, что после записи информации в файл этот файл обязательно надо закрывать с помощью функции `mclose`. Дело в том, что `mfprintf` не обращается непосредственно к диску – он пишет информацию

в специальный участок памяти, называемый буфером файла. После того как буфер заполнится, вся информация из него вносится в файл. При вызове функции `mclose` сначала происходит запись буфера файла на диск, и только потом файл закрывается. Если файл не закрыть, то он автоматически закрывается при завершении работы программы, но при этом пропадает информация, хранимая в буфере файла.

Задача 2.14

Записать матрицу $A(N, M)$ в файл следующим образом. Пусть в первой строке текстового файла хранятся числа N и M, а затем – построчно матрица A.

Листинг 2.32 содержит программу для создания подобного файла.

Листинг 2.32. Запись матрицы в файл (пример 2.14)

```
// Ввод размеров матрицы.
N=input('N=');
M=input('M=');
// Ввод элементов матрицы.
for i=1:N
  for j=1:M
    A(i,j)=input('A=');
  end
end
// Открыть файл для записи.
f=mopen('primer.txt','wt');
// Записать в файл N и M, разделив их
// символом табуляции, после чего перейти на новую строку в файле.
mfprintf(f,'%d\t%d\n',N,M);
// Цикл для построчной записи элементов матрицы в файл.
for i=1:N
  // Цикл для поэлементной записи i-й строки матрицы.
  for j=1:M
    // Запись очередного элемента A(i,j)
    // и символа табуляции в файл.
    mfprintf(f,'%g\t',A(i,j));
  end;
  // Строка записана, переходим к следующей.
  mfprintf(f,'\n');
end;
// Закрыть файл.
mclose(f);
```

После выполнения этой программы будет создан текстовый файл *primer.txt* (рис. 2.7).

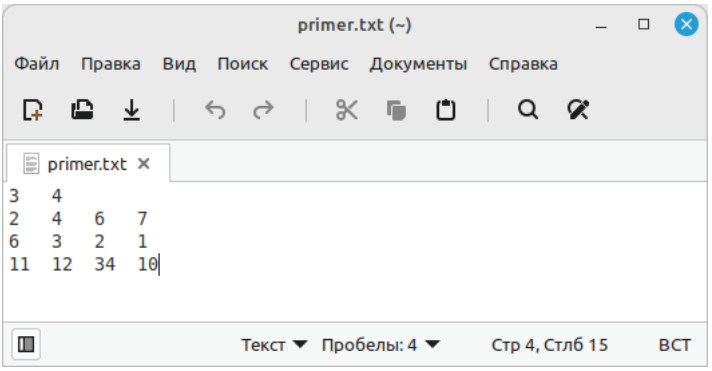

Рис. 2.7. Файл *primer.txt*

Задача 2.15
Считать в массив вещественные значения из текстового файла *one.txt*.

Листинг 2.33. Файл one.txt

```
1.22 3.45 5.6 7.8
9.1 8.2 9.3 7.41 10
```

Листинг 2.34. Поэлементное считывание данных

```
// Открываем файл для чтения.
f=mopen('one.txt');
// В переменной i хранится номер элемента массива, в который будет
// осуществляться считывание очередного значения из файла;
// в начале в i записываем 0, пока в массиве нет элементов.
i=0;
// Проверяем, если не достигнут конец файла,
while ~meof(f)
// то увеличиваем i на единицу
i=i+1;
// и считываем очередной i-й элемент
X(i)=mfscanf(f,'%f');
end
disp(X);

// Результат работы программы 1.22
3.45
5.5999999
7.8000002
9.1000004
8.1999998
9.3000002
7.4099998
10.
```

2.4 Пользовательские функции в Scilab

В Scilab файлы с расширением *.sce* могут содержать не только тексты программ, но и отдельные функции. В этом случае имя функции должно совпадать с именем файла, в котором она хранится (например, функция с именем `primer` должна храниться в файле *primer.sce*).

Функция в Scilab имеет следующую структуру. Первая строка функции – это заголовок:

```
function[y1, y2, . . . yn] = name_function(x1, x2, ..., xm)
```

Здесь `name_function` – имя функции, `x1, x2, ..., xm` – список входных параметров функции, `y1, y2, ..., yn` – список выходных параметров функции. Функция заканчивается служебным словом `end`. Таким образом, в простейшем случае структуру функции можно записать следующим образом:

```
function [y1,y2, ... ,yn]=name_function(x1,x2, ... ,xm)
оператор1;
оператор2;
 ...
операторk;
end
```

В файле с расширением *.sce*, кроме основной функции, имя которой совпадает с именем файла, могут находиться так называемые подфункции. Эти функции доступны только внутри файла.

Таким образом, общую структуру функции можно представить так:

```
// Здесь начинается основная функция sce-файла, имя которой должно
// совпадать с именем файла, в котором она хранится
function [y1,y2, ... ,yn]=name_function(x1,x2, ... ,xm)
        // Среди операторов основной функции могут быть операторы
        // вызова подфункций f1, f2, f3, ... ,fl оператор1;
        оператор2;
         ...
        операторk;
end;   // здесь заканчивается основная функция

function [y1,y2, ... ,yn]=f1(x1,x2, ... ,xm) // начало первой подфункции
        операторы
end // конец первой подфункции

function [y1,y2, ... ,yn]=f2(x1,x2, ... ,xm) // начало второй подфункции
        операторы
end // конец второй подфункции
 ...
function [y1,y2, ... ,yn]=fn(x1,x2, ... ,xm) // начало n-й подфункции
        операторы
end // конец n-й подфункции
```

В Scilab возможен синтаксис, в котором разрешено использование вложенных функций, поэтому структура основной функции может быть и такой:

```
// Здесь начинается основная функция sce-файла, имя которой должно
// совпадать с именем файла, в котором она хранится
function [y1,y2, ... ,yn]=name_function(x1,x2, ... ,xm)

  function [y1,y2, ... ,yn]=f1(x1,x2, ... ,xm)// начало первой подфункции
    операторы
  end // конец первой подфункции

  function [y1,y2, ... ,yn]=f2(x1,x2, ... ,xm)// начало второй подфункции
    операторы
  end // конец второй подфункции
   ...
  function [y1,y2, ... ,yn]=fn(x1,x2, ... ,xm) // начало n-й подфункции
    операторы
  end // конец n-й подфункции

  оператор1;
  оператор2;
   ...
   операторk;

end    // здесь заканчивается основная функция
```

Задача 2.16

Написать функцию, предназначенную для удаления из массива $x(N)$ простых чисел.

Функцию назовём `udal_prostoe`. Её входными данными являются: числовой массив x; N – количество элементов в массиве. Выходными данными функции `udal_prostoe` будут: массив x, из которого удалены простые числа; новый размер массива N после удаления из него простых чисел.

В функции `udal_prostoe` будут использоваться две вспомогательные функции: функция `prostoe`, которая проверяет, является ли число простым, и функция `udal` удаления элемента из массива.

Заголовок основной функции имеет вид:

```
function[x, N] = udal_prostoe(x, N)
```

Заголовок подфункции запишем так:

```
function pr = prostoe(P )
```

Функция `prostoe` проверяет, является ли число P простым. Она возвращает 1, если P – простое, 0 – в противном случае.

Заголовок подфункции `udal` имеет вид:

```
function [x, N] = udal(x, m, N)
```

Функция `udal` удаляет из массива $x(N)$ элемент с номером m, функция возвращает модифицированный массив x и изменённое значение N.

В листинге 2.35 приведено содержимое файла *udal_prostoe.sce* с комментариями.

Листинг 2.35. Файл udal_prostoe.sce

```
// Функция udal_prostoe удаляет из массива x(N) простые числа и возвращает
// модифицированный массив x и изменённое значение N в качестве результата.
function [x, N]=udal_prostoe(x, N)
   i=1;
   while i<=N
      L=prostoe(x(i)); // Проверка, является ли число x(i) простым.
      if L==1 // если число простое (L=1), то удаляем из массива
      // i-й элемент,
         [x N]=udal(x,i,N);
      else // иначе переходим к следующему элементу массива.
         i=i+1;
      end;
   end;
end // Окончание основной функции udal_prostoe.

// Функция prostoe проверяет, является ли число P простым,
// она возвращает 1, если P - простое,
//0 - если число P не является простым.
function pr=prostoe(P)
   pr=1
   for i=2:P/2
      if pmodulo(P,i)==0
         pr=0;
         break;
      end
   end
end // Окончание функции prostoe.

// Функция udal удаляет из массива x элемент с номером m.
function [x,N]=udal(x,m,N)
   // Удаление происходит путём смещения элементов,
   // начиная с m-го, на одну позицию влево. Выходными элементами
   // функции будут массив x, из которого удалён один элемент,
   // и уменьшенное на 1 количество (N) элементов в массиве.
   for i=m:N-1
      x(i)=x(i+1); end
   x(:,N)=[]; // После смещения элементов удаляем последний элемент
   N=N-1; // и уменьшаем количество элементов в массиве на 1.
end // Окончание функции udal.
```

Листинг 2.36 содержит текст программы, структура которой допускает вложенность функций.

Листинг 2.36. Решение примера 2.16 с использованием вложенных функций

```
function [x, N]=udal_prostoe1(x, N)
   function pr=prostoe(P)
      pr=1;
      for i=2:P/2
         if pmodulo(P,i)==0
            pr=0;
         break;
         end
      end
   end

   function [x, N]=udal(x,m,N)
      for i=m:N-1
         x(i)=x(i+1);
      end
      x(:,N)=[];
      N=N-1;
   end

   i=1;
   while i<=N
      L=prostoe(x(i));
      if L==1
         [x, N]=udal(x,i,N);
      else
         i=i+1;
      end;
   end;
end
```

Ниже представлено обращение к функции для удаления простых чисел из массива z(8).

```
--> z=[4 6 8 7 100 13 88 125];
--> [y, k]=udal_prostoe(z,8);
y = 4 6 8 100 88 125
k = 6
```

Аналогичным образом можно составлять и более сложные функции, в состав которых входит множество вспомогательных функций.

Под **рекурсией** в программировании понимается вызов функции из её тела. Классическими рекурсивными алгоритмами являются возведение числа в целую положительную степень, вычисление факториала и т. д. В рекурсивных алгоритмах функция вызывает саму себя до выполнения какого-либо условия. Рассмотрим пример.

Задача 2.17

Вычислить n-е число Фибоначчи.

Если нулевой элемент последовательности равен нулю, первый – единице, а каждый последующий представляет собой сумму двух предыдущих, то это последовательность Фибоначчи (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...).

Текст функции приведён в листинге 2.37.

Листинг 2.37. Вычисление n-го числа последовательности Фибоначчи

```
function F=fibonachi(N)
   if (N==0) | (N==1)
      F=N;
   else
      F=fibonachi(N-1)+fibonachi(N-2);
   end
end

// Вызов функции
--> fibonachi(2)
ans = 1
--> fibonachi(0)
ans = 0
--> fibonachi(6)
ans = 8
```

Задача 2.18

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. 2.8).

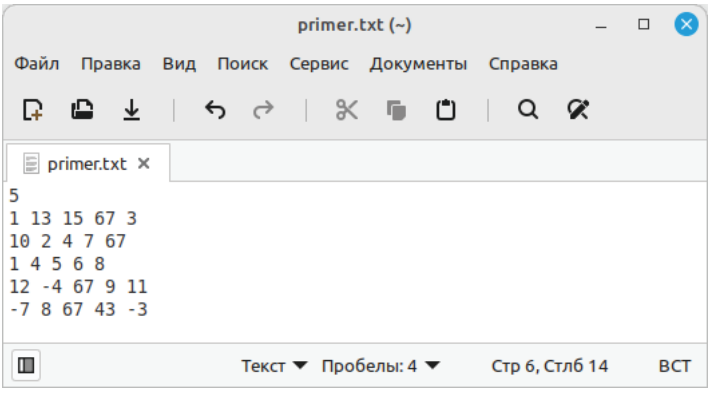

Рис. 2.8. Текстовый файл *primer.txt*

Для нахождения суммы (summa), максимума (maximum) и количества максимумов (kolichestvo) оформим функцию, заголовок её будет иметь вид:

```
function [summa, maximum, kolichestvo]=matrica_A(N,N)
```

Текст функции приведен в листинге 2.38.

Листинг 2.38. 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 нечётное, т. е. элемент,
  //который расположен одновременно на главной и побочной
  //диагоналях, мы его просуммировали дважды: и как элемент
  //главной, и как элемент побочной диагонали, поэтому его надо
  //вычесть из суммы.
  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
end
```

Текст функции, предназначенной для чтения матрицы из файла и вызова функции matrica_A, приведен в листинге 2.39, результаты – в листинге 2.40.

Листинг 2.39. Вызов функции matrica_A

```
f=mopen('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 2.40. Результаты работы программы

```
-->exec("matrica_2.sce");
-->s
 s = 21.
-->m
 m = 67.
-->k
k = 4.
```

Глава 3

Массивы и матрицы в Scilab. Решение задач линейной алгебры

Познакомимся с инструментами Scilab, предназначенными для работы с векторами и матрицами, а также с возможностями, которые предоставляет пакет при решении задач линейной алгебры.

3.1 Ввод и формирование векторов 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 т. д. 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 =
-3.5  -2.  -0.5   1.   2.5   4.
--> 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
```

Ещё 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 показан ниже.

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 =
1.1   2.3  -0.1   5.88
```

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

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

Пример ввода вектора-столбца приведён ниже.

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 = 0.9
```

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 обращения к её 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 =  3.5555556
```

Листинг 3.6. Форматированный вывод матрицы с помощью функции `mprintf`

```
//Вариант 1
A = [3.1416 1.0000 2.0000 3.0000; 4.0000 5.0000 6.0000 -3.1416];
mprintf('Матрица A:\n');
//Указывается формат вывода каждого столбца матрицы
mprintf("%4.2f\t%4.2f\t%4.2f\t%4.2f\n", A);

//Результат работы программы Матрица A:
3.14    1.00    2.00    3.00
4.00    5.00    6.00    -3.14
//Вариант 2
D=[5 2 -1;1 -3 2; 4 5 -3];
[N M]=size(D);
for i=1:N
        for j=1:M
                mprintf('%2.1f\t',D(i,j));
        end
        mprintf('\n');
end
```

```
//Результат работы программы
5.0      2.0      -1.0
1.0      -3.0      2.0
4.0      5.0      -3.0
```

Матрицы и векторы можно формировать, составляя их из ранее заданных матриц и векторов:

Листинг 3.7. 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 (листинг 3.8).

Листинг 3.8. 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 6 5
7 8 7
6 10 9
5 9 10
--> //Удалить из матрицы A третью строку
--> A(3,:)=[]
A =
5 6 5
7 8 7
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)=[];
b =
7
9
```

3.2 Действия над векторами

Рассмотрим действия над векторами, предусмотренные в Scilab.

Операция сложения определена только для векторов одного типа, т. е. суммировать можно либо векторы-столбцы, либо векторы-строки одинаковой длины. Элементы вектора, являющегося суммой двух векторов, представляют собой сумму соответствующих элементов слагаемых. Для записи операции сложения векторов используют знак «+».

Листинг 3.9. Пример сложения векторов

```
a=[2 4 6];
b=[1 3 5];
```

```
c=a+b

//Результат работы программы
c =
3.   7.   11.
```

Вычитание векторов определено аналогично сложению: из элементов вектора-уменьшаемого вычитаются соответствующие элементы второго вектора. Запись операции вычитания выполняется с помощью знака «-».

Листинг 3.10. Пример вычитания векторов

```
a=[2 4 6];
b=[1 3 5];
c=a-b

//Результат работы программы
c =
1.   1.   1.
```

Операция *транспонирования вектора* – это замена вектора-столбца вектором-строкой и наоборот. Знак апострофа «'» применяют для записи операции транспонирования вектора.

Листинг 3.11. Пример транспонирования векторов

```
a = [2 4 6];
b = [1; 3; 5];
mprintf('Вектор a:\n')
disp(a);
mprintf('Транспонированный вектор a:\n')
disp(a');
mprintf('Вектор b:\n')
disp(b);
mprintf('Транспонированный вектор b:\n')
disp(b');

c = (a+b');
mprintf('Вектор c:\n')
disp(c);
mprintf('Транспонированный вектор c:\n')
disp(c');

//Результат работы программы
Вектор a:
2.   4.   6.

Транспонированный вектор a:
2.
4.
6.

Вектор b:
1.
```

```
3.
5.

Транспонированный вектор b:
1.   3.   5.

Вектор c:
3.   7.   11.

Транспонированный вектор c:
3.
7.
11.
```

Умножение вектора на число есть умножение каждого элемента вектора на это число. Запись операции умножения вектора на число осуществляется с помощью знака «*».

Листинг 3.12. Пример умножения векторов

```
a=[2 4 6];
b=[1 3 5];
z=2*a+0.5*b

//Результат работы программы
z =
4.5   9.5   14.5
```

Деление вектора на число определяется аналогично умножению: как деление каждого элемента вектора на это число[1]. Знак деления «/» применяют для записи операции деления вектора на число.

Листинг 3.13. Пример деления вектора на число

```
z=2*a+b/2

//Результат работы программы
z =
4.5   9.5   14.5
```

Умножение векторов определено только для векторов одинакового размера, причём один из них должен быть вектором-столбцом, а второй – вектором-строкой. Если вектор-строку a умножать на вектор-столбец b, получится скалярное произведение векторов $c = \sum a_i b_i$, а если умножать вектор-столбец b на вектор-строку a, то получится матрица C, у которой каждая строка представляет собой исходную вектор-строку, умноженную на соответствующие элементы вектора-столбца. Операция умножения вектора на вектор записывается с помощью знака «*», так же как и операция умножения вектора на число. Примеры умножения векторов приведены ниже.

---

[1] Если делитель будет равен нулю, то в ответе будет `Inf` или `NaN` и предупреждение `warning: division by zero`. – *Прим. ред.*

Листинг 3.14. Пример умножения векторов

```
a=[2 4 6];
b=[1 3 5];
// В результате умножения вектора-строки на вектор-столбец получится число
c=a*b';
// Результат умножения вектора-столбца на вектор-строку - матрица
D=a'*b;
mprintf('число c:\n');
disp(c);
mprintf("матрица D:\n");
disp(D);

//Результат работы программы
число c:
44.

матрица D:
2.   6.    10.
4.   12.   20.
6.   18.   30.
```

Листинг 3.15. Некорректное умножение векторов

```
w1=a*b;
w2=a'*b';

//Результат работы программы
Несогласованные размеры по строкам/столбцам.
Несогласованные размеры по строкам/столбцам.
```

Все перечисленные действия над векторами определены в математике и относятся к так называемым векторным вычислениям. Но Scilab допускает и *поэлементное преобразование* векторов. Существуют операции, которые работают с вектором не как с математическим объектом, а как с обычным одномерным массивом. Например, если к некоторому заданному вектору применить математическую функцию, то результатом будет новый вектор того же размера и структуры, но элементы его будут преобразованы в соответствии с заданной функцией.

Листинг 3.16. Применение математической функции к вектору

```
x=[-%pi/2,-%pi/3,-%pi/4,0,%pi/4,%pi/3,%pi/2]
y=sin(2*x)+cos(2*x);
z=2*exp(x/5);
mprintf('вектор y:\n');
disp(y);
mprintf("вектор z:\n");
disp(z);

//Результат работы программы
вектор y:
-1.  -1.3660254  -1.   1.   1.   0.3660254  -1.

вектор z:
1.4608054   1.6220774   1.709272   2.   2.3401776   2.4659736   2.7382155
```

Рассмотрим ещё несколько операций поэлементного преобразования вектора. К каждому элементу вектора можно добавить (вычесть) число, используя арифметическую операцию «+» («-»)[1].

Листинг 3.17. Пример поэлементного сложения векторов

```
--> x=[-%pi/2,-%pi/3,-%pi/4,0,%pi/4,%pi/3,%pi/2];
--> y=x-1.2+%e/3;
--> mprintf("%1.2f\t",y')
-1.86   -1.34   -1.08   -0.29   0.49   0.75   1.28
```

Поэлементное умножение векторов выполняется при помощи оператора «.*», результатом такого умножения является вектор, каждый элемент которого равен произведению соответствующих элементов заданных векторов.

Листинг 3.18. Пример поэлементного умножения векторов

```
--> a=[2 4 6];
--> b=[1 3 5];
--> a.*b
ans = 2   12   30
--> b.*a
ans = 2   12   30
```

Поэлементное деление одного вектора на другой осуществляется при помощи оператора «./». В результате получается вектор, каждый элемент которого – частное от деления соответствующего элемента первого вектора на соответствующий элемент второго.

Совокупность знаков «.\» применяют для деления векторов в обратном направлении (поэлементное деление второго вектора на первый).

Листинг 3.19. Пример поэлементного деления векторов

```
a=[2 4 6];
b=[1 3 5];
c=a./b;
mprintf("%1.2f\t",c')

//Результат работы программы
2.00   1.33   1.20

//поэлементное деление второго вектора на первый
a=[2 4 6];
b=[1 3 5];
d=a.\b
mprintf("%1.2f\t",d')

//Результат работы программы
0.50    0.75    0.83
```

[1] Вдумчивый читатель сразу заметит, что это частный случай предыдущего примера – применение к вектору линейного преобразования. – *Прим. ред.*

Поэлементное возведение в степень. Осуществляется с помощью оператора «.^». Результатом является вектор, каждый элемент которого – это соответствующий элемент заданного вектора, возведённый в указанную степень.

Листинг 3.20. Пример поэлементного возведения в степень

```
--> a=[2 4 6]; b=[1 3 5];
--> a.^2// Каждый элемент вектора возвести в квадрат
ans = 4   16   36

--> b.^(1/2)// Извлечь корень квадратный из каждого элемента вектора
ans =  1.   1.7320508   2.236068

--> b.^a// Каждый элемент вектора b возвести в степень a
ans =  1      81   15625

--> a.^(1./b)// Извлечь корень b-й степени из каждого элемента вектора a
ans =  2.   1.5874011   1.4309691
```

3.3 Действия над матрицами

Напомним основные определения алгебры матриц. Если $m \times n$ выражений расставлены в прямоугольной таблице из m строк и n столбцов, то говорят о матрице размера $m \times n$.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \tag{3.1}$$

Выражения a_{ij} называют элементами матрицы. Элементы a_{ii} $(i = 1 \dots n)$, стоящие в таблице на линии, проходящей из левого верхнего угла в правый нижний угол квадрата $n \times n$, образуют главную диагональ матрицы.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ii} & \dots & a_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & \dots & \dots & a_{mn} \end{pmatrix}. \tag{3.2}$$

Матрица размером $m \times n$ (где $m \neq n$) называется прямоугольной. В случае если $m = n$, матрицу называют квадратной матрицей порядка n.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}. \qquad (3.3)$$

В частности, матрица типа $1\times n$ – это вектор-строка: $(a_{11}\ \ a_{12}\ \dots\ a_{1n})$. Матрица размером $m\times 1$ является вектором-столбцом: $\begin{pmatrix} a_{11} \\ a_{21} \\ \dots \\ a_{m1} \end{pmatrix}$.

Число (скаляр) можно рассматривать как матрицу типа 1×1 – a_{11}. Квадратная матрица $A = \{a_{ij}\}$ размером $n\times n$ называется:

- *нулевой*, если все её элементы равны нулю: $\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix}$;
- *верхней треугольной*, если все элементы, расположенные ниже главной диагонали, равны нулю: $\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$;
- *нижней треугольной*, если все элементы, расположенные выше главной диагонали, равны нулю: $\begin{pmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$;
- *диагональной*, если все элементы, кроме элементов главной диагонали, равны нулю: $\begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$;
- *единичной*, если элементы главной диагонали равны единице, а все остальные – нулю: $\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}$.

Определителем (детерминантом) матрицы A является число $\det A$ или Δ, вычисляемое по правилу: $\det A = \sum (-1)^{\tilde{\lambda}} a_{1i_1} a_{2i_2} \dots a_{ni_n}$, где сумма распределена

на всевозможные перестановки $(i_1, i_2, \ldots, i_n)$ элементов $1, 2, \ldots, n$ и, следовательно, содержит $n!$ слагаемых, причём $\tilde{\lambda} = 0$, если перестановка чётная, и $\tilde{\lambda} = 1$, если перестановка нечётная.

Квадратная матрица A называется *невырожденной*, если её определитель отличен от нуля $\det A \neq 0$. В противном случае $\det A = 0$ матрица называется *вырожденной*, или *сингулярной*.

С матрицами можно проводить операции *сравнения*, *сложения* и *умножения*. Две матрицы $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ считаются *равными*, если они одного типа, т. е. имеют одинаковое число строк и столбцов, и соответствующие элементы их равны $\{a_{ij}\} = \{b_{ij}\}$.

Суммой двух матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ одинакового типа называется матрица $C = \{c_{ij}\}$ того же типа, элементы которой равны сумме соответствующих элементов матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$: $\{c_{ij}\} = \{a_{ij}\} + \{b_{ij}\}$.

Разность матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ определяется аналогично: $\{c_{ij}\} = \{a_{ij}\} - \{b_{ij}\}$.

Произведением числа $\hbar$ и матрицы $A = \{a_{ij}\}$ (или умножением матрицы на число) называется матрица, элементы которой получены умножением всех элементов матрицы $A = \{a_{ij}\}$ на число $\hbar$: $\hbar \cdot A = \{\hbar \cdot a_{ij}\}$.

Произведением матриц $A = \{a_{ij}\}$ размерностью $m \times n$ и $B = \{b_{ij}\}$ размерностью $n \times s$ является матрица $C = \{c_{ij}\}$ размерностью $m \times s$, каждый элемент которой можно представить формулой $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$, где $i = 1 \ldots m$, $j = 1 \ldots s$.

Таким образом, произведение матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ имеет смысл тогда и только тогда, когда количество строк матрицы $A = \{a_{ij}\}$ совпадает с количеством столбцов матрицы $B = \{b_{ij}\}$. Кроме того, произведение двух матриц не обладает переместительным законом, т. е. $A \cdot B \neq B \cdot A$. В тех случаях, когда $A \cdot B = B \cdot A$, матрицы $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ называются *перестановочными*.

Если в матрице $A = \{a_{ij}\}$ размерностью $m \times n$ заменить строки соответствующими столбцами, то получится *транспонированная матрица*: $A^T = \{a_{ji}\}$.

В частности, для вектора-строки $a = \{a_1 a_2 \ldots a_n\}$ транспонированной матрицей является вектор-столбец:

$$a^T = \begin{pmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{m1} \end{pmatrix}.$$

Обратной матрицей по отношению к данной матрице $A = \{a_{ij}\}$ размерностью $n \times n$ называется матрица $A^{-1} = \{a_{ij}\}$ того же типа, которая, будучи умноженной как справа, так и слева на данную матрицу, в результате даёт единичную матрицу $E = \{\delta_{ij}\}$, где $\delta_{ii} = 1$, $\delta_{ij} = 0$, при $i \neq j$: $A \cdot A^{-1} = A^{-1} \cdot A = E$.

Нахождение обратной матрицы для данной называется *обращением данной матрицы*. Всякая неособенная матрица имеет обратную матрицу.

Начнём с операций, которые применимы к матрицам с точки зрения классической математики. Одним из базовых действий над матрицами является *сложение* «+» (*вычитание* «-»). Здесь важно помнить, что суммируемые (вы-

читаемые) матрицы должны быть одной размерности. Результатом такой операции является матрица.

Листинг 3.21. Пример сложения и вычитания матриц

```
A=[1 2 3;4 5 6;7 8 9]
B=[0 9 8;7 6 5;4 3 2]
C=A+B;
D=A-B;
mprintf('матрица C:\n')
disp(C);
mprintf('матрица D:\n')
disp(D);

//результат работы программы
матрица C:
1.    11.   11.
11.   11.   11.
11.   11.   11.

матрица D:
1.  -7.  -5.
-3.  -1.   1.
3.   5.   7.
```

Умножать на число (запись «*») можно любую матрицу, результатом также будет матрица, каждый элемент которой будет помножен на заданное число.

Листинг 3.22. Пример умножения матрицы на число

```
--> A=[1 2 3;4 5 6;7 8 9];
--> B=0.2*A
B =
0.2   0.4   0.6
0.8   1.    1.2
1.4   1.6   1.8
```

Операция *транспонирования* (запись «'») меняет в заданной матрице строки на столбцы и также применима к матрицам любой размерности.

Листинг 3.23. Пример транспонирования матрицы

```
--> A
A =
1.   2.   3.
4.   5.   6.
7.   8.   9.

--> A'
ans =
1.   4.   7.
2.   5.   8.
3.   6.   9.
```

При *умножении матриц* («*») важно помнить, что число столбцов первой перемножаемой матрицы должно быть равно числу строк второй.

Листинг 3.24. Примеры умножения матриц

```
--> A=[1 2 3;4 5 6;7 8 9];
--> B=[0 9 8;7 6 5;4 3 2];
--> A*B
ans =
26.   30.    24.
59.   84.    69.
92.   138.   114.

--> A=[-3 2;0 1];
--> B=[0 -2;3 -1;0 1];
--> B*A
ans =
0.  -2.
-9.   5.
0.   1.

// Некорректное умножение матриц
--> A*B
Несогласованные размеры по строкам/столбцам.
```

Возведение матрицы в степень эквивалентно её умножению на себя указанное число раз. При этом целочисленный показатель степени может быть как положительным, так и отрицательным. Матрица в степени -1 называется обратной к данной. При возведении матрицы в положительную степень выполняется алгоритм умножения матрицы на себя указанное число раз. Возведение в отрицательную степень означает, что умножается на себя матрица, обратная к данной.

Листинг 3.25. Примеры возведения в степень

```
--> D=[3 2 1;1 0 2;4 1 3];
--> D^3
ans =
92.    40.   59.
65.    29.   40.
146.   65.   92.
--> D^(-1)
ans =
-0.4  -1.   0.8
1.    1.  -1.
0.2   1.  -0.4
--> D^(-3)
ans =
0.544   1.24  -0.888
-1.12   -1.2    1.24
-0.072  -1.12   0.544
```

Для *поэлементного преобразования матриц* (листинг 3.26) можно применять операции, описанные ранее, как операции поэлементного преобразования векторов: *добавление (вычитание) числа к каждому элементу матрицы* («+» или «-»), *поэлементное умножение матриц* («.*») одинакового размера, *поэлементное деление матриц* одинакового размера (прямое «./» и обратное «.\»), *поэлементное возведение в степень* («.^») и *применение к каждому элементу матрицы математических функций*.

Листинг 3.26. Примеры преобразования матриц

```
--> M=[3 2 1;1 1 2;4 1 3];
--> N=[4 -2 -1;9 6 -2;-3 -1 2];

--> 2*M
ans =
6.   4.   2.
2.   2.   4.
8.   2.   6.

--> N/3
ans =
1.3333333  -0.6666667  -0.3333333
3.          2.         -0.6666667
-1.         -0.3333333   0.6666667

--> M.*N
ans =
12.  -4.  -1.
9.    6.  -4.
-12.  -1.   6.

--> N.*M
ans =
12.  -4.  -1.
9.    6.  -4.
-12.  -1.   6.

--> M./N
ans =
0.75       -1.         -1.
0.1111111   0.1666667  -1.
-1.3333333  -1.          1.5

--> M.\N
ans =
1.3333333  -1.  -1.
9.          6.  -1.
-0.75       -1.   0.6666667

--> M.^0.2
ans =
1.2457309   1.1486984   1.
```

```
1.          1.          1.1486984
1.3195079   1.          1.2457309

--> N.^M
ans =
64.   4.  -1.
9.    6.   4.
81.  -1.   8.
```

Задача 3.1

Вычислить математическое выражение $(2A + \frac{1}{3}B^T)^2 - AB^{-1}$ для заданных значений A и B.

Решение задачи показано в листинге 3.27.

Листинг 3.27. Вычисление математического выражения (задача 3.1)

```
--> A=[-3 2 0;0 1 2;5 3 1];
-->B=[0 -2 1;3 -1 1;0 1 1];
--> (2*A+1/3*B')^2-A*B^(-1)
ans =
32.666667  -20.666667   20.666667
47.333333   26.888889   15.666667
-40.333333   75.333333   31.777778
```

Довольно необычное, с точки зрения математики, применение нашлось для операторов / и \. Символ «/» используется для операции, называемой делением матриц слева направо, соответственно, знак \ применяется для деления матриц справа налево. Операция B/A эквивалентна выражению $B \cdot A^{-1}$, её удобно использовать для решения матричных уравнений вида $X \cdot A = B$:

Листинг 3.28. Пример решения матричного уравнения вида $X \cdot A = B$

```
A=[2 -1 2;-1 2 -2;2 -2 5];
mprintf('Матрица A:\n')
mprintf('%1.2f\t%1.2f\t%1.2f\n',A)
B=[7 0 0;0 1 0;0 0 1];
mprintf('Матрица B:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',B)
X=B/A
mprintf('Матрица X:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',X)
X0=X*A-B // Проверка X*A-B=0
mprintf('Проверка:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',X0)

//Результат работы программы
Матрица A:
2.00    -1.00   2.00
-1.00   2.00    -2.00
2.00    -2.00   5.00
```

```
Матрица B:
7.00    0.00    0.00
0.00    1.00    0.00
0.00    0.00    1.00
Матрица X:
6.00    1.00    -2.00
0.14    0.86    0.29
-0.29   0.29    0.43
Проверка:
0.00    0.00    0.00
0.00    -0.00   0.00
0.00    0.00    0.00
```

Соответственно, $A\backslash B$ эквивалентно $A^{-1} \cdot B$ и применяется для решения уравнения $A \cdot X = B$.

Листинг 3.29. Пример решения матричного уравнения вида $A \cdot X = B$

```
A=[2 -1 2;-1 2 -2;2 -2 5];
mprintf('Матрица A:\n')
mprintf('%1.2f\t%1.2f\t%1.2f\n',A)
B=[7 0 0;0 1 0;0 0 1];
mprintf('Матрица B:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',B)
X=A\B
mprintf('Матрица X:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',X)
X0=A*X-B // Проверка A*X-B=0
mprintf('Проверка:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',X0)

//Результат работы программы
Матрица A:
2.00    -1.00   2.00
-1.00   2.00    -2.00
2.00    -2.00   5.00
Матрица B:
7.00    0.00    0.00
0.00    1.00    0.00
0.00    0.00    1.00
Матрица X:
6.00    0.14    -0.29
1.00    0.86    0.29
-2.00   0.29    0.43
Проверка:
0.00    0.00    0.00
0.00    -0.00   0.00
0.00    0.00    0.00
```

Если предположить, что x и b – это векторы, а A – матрица, то получим запись системы линейных алгебраических уравнений в матричной форме $Ax = b$. Это значит, что оператор \ можно применять для *решения линейных систем*.

Листинг 3.30. Пример решения линейной системы вида *Ax = b*

```
A=[1 2;1 1];
b=[7;6];
mprintf('Матрица A:\n')
mprintf('%1.2f\t%1.2f\n',A)
mprintf('Вектор b:\n');
mprintf('%1.2f\n',b)
x=A\b
mprintf('Вектор x:\n');
mprintf('%1.2f\n',x)
x0=A*x // Проверка Ax=b
mprintf('Проверка:\n');
mprintf('%1.2f\n',x0)

//Результат работы программы
Матрица A:
1.00    2.00
1.00    1.00
Вектор b:
7.00
6.00
Вектор x:
5.00
1.00
Проверка:
7.00
6.00
```

3.4 Символьные матрицы и операции над ними

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

Листинг 3.31. 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 3.32. Операции над символьными матрицами

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

3.5 Функции для работы с матрицами и векторами

В Scilab существуют специальные функции, предназначенные для работы с матрицами и векторами. Эти функции можно разделить на следующие группы:

1) функции для работы с векторами;
2) функции для работы с матрицами;
3) функции, реализующие численные алгоритмы решения задач линейной алгебры.

Рассмотрим наиболее часто используемые функции.

3.5.1 Функции для работы с векторами

`length(X)` определяет количество элементов массива `X`; если `X` – вектор, то его длину; если `X` – матрица, вычисляет общее число её элементов.

```
--> 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.

--> X=[1 2 3 4 5 6 7 8 9];
--> n=length(X)
n = 9

--> Y=[-2;-1;0;1;2]
Y =
-2
-1
0
1
2
--> m=length(Y)
m = 5
```

`prod(X)` вычисляет произведение элементов вектора `X`.

```
--> X=[1 2 3 4 5 6 7 8 9]; prod(X)
ans =  362880
```

`cumprod(X)` формирует вектор кумулятивного произведения – вектор того же типа и размера, что и `X` вида: $x_1, x_1 \cdot x_2, x_1 \cdot x_2 \cdot x_3, \ldots, x_1 \cdot x_2 \cdots x_n$, каждый элемент которого рассчитывается по формулам $x_i' = x_1 \cdot x_2 \cdots x_i$, т. е. i-й элемент вектора X умножается на произведение всех предыдущих элементов.

```
--> X=[1 2 3 4 5 ]; cumprod(X)
ans = 1     2     6    24   120
```

`sum(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].

```
-->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
```

`cumsum(X)` формирует вектор кумулятивной суммы – вектор того же типа и размера, что и `X` вида: $x_1, x_1 + x_2, x_1 + x_2 + x_3, \ldots, x_1 + x_2 + \cdots + x_n$, каждый элемент которого рассчитывается по формуле $x_i' = x_1 + x_2 + \cdots + x_i$, т. е. к i-му элементу вектора `X` прибавляется сумма всех предыдущих элементов.

```
--> X=[1 2 3 4 5 ]; cumsum(X)
ans = 1    3    6   10   15
```

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

`diff(X)` формирует вектор вида $x_2 - x_1, x_3 - x_2, \ldots, x_n - x_{n-1}$, размер которого на единицу меньше, чем у вектора `X`, а каждый элемент представляет собой разность между двумя соседними элементами массива `X`, т. е. $x_i' = x_i - x_{i-1}$.

```
--> X=[1 2 3 4 5 6 7 8 9]; diff(X)
ans = 1 1 1 1 1 1 1 1
```

`min(X)` находит минимальный элемент вектора `X`, вызов в формате `[minimum,nomer]=min(X)` даёт возможность определить минимальный элемент `nomX` и его номер `nom` в массиве `X`.

```
--> X=[-1 2 3 9 -8 7 5];
--> min(X)
ans = -8
--> [minimum,nomer]=min(X)
nomer   =  5.
minimum = -8.
```

`max(X)` находит максимальный элемент массива `X` или при вызове вида `[maximum,nomer]=max(X)` определяет максимум и его номер.

```
--> X=[-1 2 3 9 -8 7 5];
--> max(X)
ans = 9
--> [maximum,nomer]=max(X)
nomer   = 4.
maximum = 9.
```

`mean(X)` определяет среднее арифметическое массива `X`.

```
--> X=[-1 2 3 9 -8 7 5];
--> Sr=mean(X)
Sr =  2.4285714
--> sum(X)/length(X)
ans =  2.4285714
```

`cross(x1,x2)` вычисляет векторное произведение векторов x_1 и x_2.

```
--> x1=[2 -3 0];x2=[0 1 -2];
--> x=cross(x1,x2)
x = 6.   4.   2.
--> x1=[2;-3;0];x2=[0;1;-2];
--> x=cross(x1,x2)
x =
6.
4.
2.
```

`gsort(X)` выполняет сортировку массива `X`.

```
--> X=[-1 2 3 9 -8 7 5];
--> gsort(X)// Сортировка по возрастанию
ans = -8  -1   2   3   5   7   9
```

```
--> -gsort(-X)// Сортировка по убыванию
ans = 9   7   5   3   2  -1  -8
```

3.5.2 Функции для работы с матрицами

`prod(M[, k])` формирует вектор-строку или вектор-столбец в зависимости от значения `k`, каждый элемент которой(го) является произведением элементов соответствующего столбца или строки матрицы `M`. Если значение параметра `k` отсутствует, то вычисляется произведение всех элементов матрицы.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> prod(M)
ans = -345600.

--> prod(M,1)
ans =
24.  -25.   32.   18.

--> prod(M,2)
ans =
6.
-40.
-12.
-120.
```

`cumprod(M[, k])` отличается от функции `cumprod(X)` тем, что операции, описанные для неё, применяются к строкам или столбцам матрицы `M` в зависимости от значения параметра `k`, по умолчанию накапливание произведения выполняется по столбцам матрицы `M`.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> cumprod(M)
ans =
-1.    24.    1200.   -57600.
-4.    120.  -1200.   -115200.
-12.  -120.  -4800.   -115200.
24.  -600.  -19200.  -345600.

--> cumprod(M,1)
ans =
-1.    1.   -2.    3.
-4.    5.    2.    6.
-12.  -5.    8.    6.
24.  -25.   32.   18.

--> cumprod(M,2)
ans =
-1.  -1.    2.    6.
4.   20.  -20.  -40.
3.  -3.   -12.  -12.
-2.  -10.  -40.  -120.
```

`sum(M[, k])` формирует вектор-строку или вектор-столбец в зависимости от значения `k`, каждый элемент которой(го) является суммой элементов со-

ответствующего столбца или строки матрицы M, если значение параметра k отсутствует, то вычисляется сумма всех элементов матрицы.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> sum(M)
ans =
28.

--> sum(M,1)
ans =
4.   10.   5.   9.

--> sum(M,2)
ans =
1.
10.
7.
10.
```

cumsum(M, [k]) отличается от функции cumsum(X) тем, что операции, описанные для неё, применяются либо к строкам, либо к столбцам матрицы M в зависимости от значения параметра k, по умолчанию результатом работы функции является матрица кумулятивных сумм столбцов матрицы M.

```
--> M=[-1 1 -2 3 ;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> cumsum(M)
ans =
-1.   5.    12.   22.
3.   10.   11.   24.
6.   9.    15.   25.
4.   14.   19.   28.

--> cumsum(M,1)
ans =
-1.   1.   -2.   3.
3.   6.   -3.   5.
6.   5.    1.   6.
4.   10.   5.   9.

--> cumsum(M,2)
ans =
-1.   0.  -2.   1.
4.   9.   8.   10.
3.   2.   6.   7.
-2.   3.   7.   10.
```

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.

```
-->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`.

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

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

```
--> M=[5 0 3;2 7 1;0 4 9];
-->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`.

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

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

`gsort(M)` выдаёт матрицу того же размера, что и `M`, каждый столбец которой упорядочен по возрастанию. Синтаксис:

```
[B [,k]]=gsort(A)
[B [,k]]=gsort(A,option)
[B [,k]]=gsort(A,option,direction)
```

Здесь:

- `A` – матрица;
- `option` – символьная строка. Она задаёт тип требуемой сортировки:
  - `'r'` – сортируется каждый столбец A;
  - `'c'` – сортируется каждая строка A;
  - `'g'` – сортируются все элементы A. Это значение по умолчанию;
  - `'lr'` – лексикографическая сортировка строк A;
  - `'lc'` – лексикографическая сортировка столбцов A;
- `direction` – символьная строка. Она задаёт направление сортировки; `'i'` устанавливает порядок возрастания, а `'d'` – порядок убывания (по умолчанию);
- `B` – массив того же типа и размеров, что и `A`;
- `k` – вещественный массив целочисленных значений тех же размеров, что и `A`. Содержит исходные индексы.

Когда элементы являются комплексными значениями, элементы сортируются по их модулям, `'g'` в качестве второго аргумента работает с комплексными значениями.

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

//Сортировка матрицы A по столбцам в порядке убывания.
//Такой же результат будет выдан B=gsort(A,'g','d') и B=gsort(A,'g')
--> B=gsort(A)
B  =
5.   4.   2.  -1.
5.   3.   1.  -1.
```

```
4.   3.   1.  -2.
4.   3.  -1.  -2.

//Сортировка матрицы A по столбцам в порядке возрастания.
--> B=gsort(A,'g','i')
B  =
-2.  -1.   3.   4.
-2.   1.   3.   4.
-1.   1.   3.   5.
-1.   2.   4.   5.

//Сортировка строк
--> A=['а' 'г' 'е';'б' 'д' 'ю';'в' 'е' 'я']
A  =
!а  г  е  !
!        !
!б  д  ю  !
!        !
!в  е  я  !

//сортирует строки A в лексическом порядке убывания.
//B получается перестановкой строк матрицы A таким образом,
//чтобы строки B удовлетворяли B(i,:)>=B(j,:), если i<j.
--> B=gsort(A,'lr')
B  =

!в  е  я  !
!        !
!б  д  ю  !
!        !
!а  г  е  !

//сортирует строки A в лексическом порядке возрастания.
//B получается перестановкой строк матрицы A таким образом,
//чтобы строки B удовлетворяли B(i,:)<=B(j,:), если i<j.
--> B=gsort(A,'lr','i')
B  =

!а  г  е  !
!        !
!б  д  ю  !
!        !
!в  е  я  !

//сортирует столбцы A в лексическом порядке убывания.
//B получается перестановкой столбцов матрицы A таким образом,
//чтобы столбцы B удовлетворяли B(:,i)>=B(:,j), если i<j.
--> B=gsort(A,'lc')
B  =
!е  г  а  !
!        !
!ю  д  б  !
!        !
!я  е  в  !
```

```
//сортирует столбцы A в лексическом порядке возрастания.
//B получается перестановкой столбцов матрицы A таким образом,
//чтобы столбцы B удовлетворяли B(:,i)<=B(:,j), если i<j.
--> B=gsort(A,'lc','i')
B =

!а  г  е  !
!         !
!б  д  ю  !
!         !
!в  е  я  !
```

`sqrtm(M)` относится к так называемым матричным функциям и возвращает матрицу `X`, для которой $X * X = M$ (матрица `M` должна быть квадратной).

```
A=[1 0 -3;0 1 2;2 0 -1];
mprintf('Матрица A:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',A);
X=sqrtm(A);
mprintf('Матрица X:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',X);
P=X*X// Проверка
mprintf('Проверка. P должна быть равна A\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',P);

//Результат работы программы
Матрица A:
1.00    0.00    -3.00
0.00    1.00    2.00
2.00    0.00    -1.00
Матрица X:
1.53    0.00    -1.42
-0.35   1.00    0.95
0.95    0.00    0.58
Проверка. P должна быть равна A
1.00    0.00    -3.00
0.00    1.00    2.00
2.00    0.00    -1.00

//Следует помнить, что обычное и поэлементное умножение матриц -
//это разные операции, которые возвращают разный результат.
//sqrtm(A) и sqrt(A) дают различные результаты

Y=sqrt(A)
mprintf('Матрица Y:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',Y);

Матрица Y:
1.00    0.00    0.00
0.00    1.00    1.41
1.41    0.00    0.00
```

`expm(M)` и `logm(M)` – взаимообратные матричные функции, первая вычисляет матричную экспоненту e^M, а вторая выполняет логарифмирование по основанию e.

```
A=[1 0 -3;0 1 2;2 0 -1];
mprintf('Матрица A:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',A);
B=expm(A);
mprintf('Матрица B:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',B);
C=logm(B);
mprintf('Матрица C:\n');
mprintf('%1.2f\t%1.2f\t%1.2f\n',C);

//Результат работы программы
Матрица A:
1.00    0.00    -3.00
0.00    1.00    2.00
2.00    0.00    -1.00
Матрица B:
-0.27   0.00    -1.06
1.99    2.72    0.70
0.70    0.00    -0.97
Матрица C:
1.00    0.00    -3.00
0.00    1.00    2.00
2.00    0.00    -1.00
```

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

```
-->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.
```

`sparse([i1j1; i2j2; ...; injn], [n1, n2, ..., nn])` формирует разрежённую матрицу[1]. Для создания матрицы такого типа необходимо указать индексы её ненулевых элементов – `[i1j1, i2j2, ..., injn]` 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 её размерности, а также значения ненулевых элементов и их местоположение в матрице.

`full(M)` – вывод разрежённой матрицы `M` в виде таблицы.

```
-->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.
```

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

```
-->//Пример создания матрицы M,
-->//состоящей из трёх 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.
```

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

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

[1] Разрежённая матрица – матрица, большинство элементов которой – нули.

```
m  =
2.
n  =
4.
-->size(M,1)
ans  = 4.
-->size(M,2)
ans  =
2.
```

eye(n[, m]) возвращает единичную матрицу (вектор) соответствующей размерности.

```
--> eye(4,4)
ans =
1.   0.   0.   0.
0.   1.   0.   0.
0.   0.   1.   0.
0.   0.   0.   1.

--> eye(2,4)
ans =
1.   0.   0.   0.
0.   1.   0.   0.

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

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

ones(n[, m, p, …]) формирует матрицу (вектор), состоящую из единиц.

```
--> ones(2)
ans =
1.

--> ones(2,2)
ans =
1.   1.
1.   1.

--> ones(3,3)
ans =
1.   1.   1.
1.   1.   1.
1.   1.   1.

--> ones(1,4)
ans =
1.   1.   1.   1.
```

```
--> ones(2,1)
ans =
1.
1.

--> ones(4,2)
ans =
1.   1.
1.   1.
1.   1.
1.   1.

--> ones(2,3,4)
ans = (:,:,1)
1.   1.   1.
1.   1.   1.
(:,:,2)
1.   1.   1.
1.   1.   1.
(:,:,3)
1.   1.   1.
1.   1.   1.
(:,:,4)
1.   1.   1.
1.   1.   1.
```

`zeros(n[, m, p, …])` возвращает нулевую матрицу (вектор) соответствующей размерности.

```
--> zeros(3)
ans =
0.

--> zeros(3,3)
ans =
0.   0.   0.
0.   0.   0.
0.   0.   0.

--> zeros(1,2)
ans =
0.   0.

--> zeros(3,2)
ans =
0.   0.
0.   0.
0.   0.

--> zeros(4,1)
ans =
0.
0.
0.
0.
```

```
--> zeros(2,2,2)
ans =
(:,:,1)
0.   0.
0.   0.
(:,:,2)
0.   0.
0.   0.
```

`diag(X[, k])` возвращает квадратную матрицу с элементами `X` на главной диагонали или на *k*-й. Функция `diag(M[, k])`, где `M` – ранее определённая матрица, в качестве результата выдаст вектор-столбец, содержащий элементы главной или *k*-й диагонали матрицы `M`.

```
--> X=[-1 2 3 9 -8 7 5];
--> diag(X)
ans =
-1.   0.   0.   0.   0.   0.   0.
0.   2.   0.   0.   0.   0.   0.
0.   0.   3.   0.   0.   0.   0.
0.   0.   0.   9.   0.   0.   0.
0.   0.   0.   0.  -8.   0.   0.
0.   0.   0.   0.   0.   7.   0.
0.   0.   0.   0.   0.   0.   5.

--> diag(X,0)
ans =
-1.   0.   0.   0.   0.   0.   0.
0.   2.   0.   0.   0.   0.   0.
0.   0.   3.   0.   0.   0.   0.
0.   0.   0.   9.   0.   0.   0.
0.   0.   0.   0.  -8.   0.   0.
0.   0.   0.   0.   0.   7.   0.
0.   0.   0.   0.   0.   0.   5.

--> x=[2;-3; 0];
--> diag(x,1)
ans =
0.   2.   0.   0.
0.   0.  -3.   0.
0.   0.   0.   0.
0.   0.   0.   0.

--> diag(x,-1)
ans =
0.   0.   0.   0.
2.   0.   0.   0.
0.  -3.   0.   0.
0.   0.   0.   0.

--> diag(x,2)
ans =
0.   0.   2.   0.   0.
```

```
0.   0.   0.  -3.   0.
0.   0.   0.   0.   0.
0.   0.   0.   0.   0.
0.   0.   0.   0.   0.

--> diag(x,-2)
ans =
0.   0.   0.   0.   0.
0.   0.   0.   0.   0.
2.   0.   0.   0.   0.
0.  -3.   0.   0.   0.
0.   0.   0.   0.   0.

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

--> diag(M)
ans =
1.
5.
9.

--> diag(M,1)
ans =
2.
6.

--> diag(M,-2)
ans = 7.
```

`rand([n, m, p, …])` возвращает матрицу (вектор) с элементами, распределёнными по равномерному закону, `rand` без аргументов возвращает одно случайное число.

```
--> rand(2)
ans =
0.2113249

--> rand(3,1)
ans =
0.7560439
0.0002211
0.3303271

--> rand(1,4)
ans =
0.6653811   0.6283918   0.8497452   0.685731

--> rand(2,5)
ans =
0.8833888   0.3076091   0.2146008   0.3616361   0.5664249
0.6525135   0.9329616   0.312642    0.2922267   0.4826472
```

```
--> rand
ans = 0.3321719
--> rand
ans = 0.42848
```

`randn([n, m, p …])` возвращает матрицу (вектор), элементы которой являются числами, распределёнными по нормальному закону, `randn` без аргументов возвращает одно случайное число.

```
--> randn(2)
ans =
-1.04321  -1.81309
1.09223  -0.83071
--> randn(2,4)
ans =
-0.222773 -0.540185 0.026355 0.308437
1.510429  1.360071 0.298315 1.186672
--> randn(1,3)
ans =
0.38577  -2.33667  -1.35689
--> randn(2,1)
ans =
-0.66235
0.32907
--> randn
ans = -1.0607
--> randn
ans = -0.47825
```

`linspace(a, b[, n])` возвращает массив из 100 (если *n* не указано) или из *n* точек, равномерно распределённых между значениями *a* и *b*.

```
--> linspace(a,b,3)
ans = -2   0   2
--> a=-2;b=2;n=5;
--> linspace(a,b,n)
ans = -2  -1   0   1   2
--> linspace(0,50,5)
ans = 0.00000 12.50000 25.00000 37.50000 50.00000

//Работа с векторами.
-->a=[0;10;100];
-->b=[0;50;500];

--> linspace(a,b,1)
ans  =
0.
50.
500.

-->linspace(a,b,3)
ans  =
0.     0.     0.
```

```
10.    30.    50.
100.   300.   500.

--> linspace(a,b,5)
ans  =
0.     0.     0.     0.     0.
10.    20.    30.    40.    50.
100.   200.   300.   400.   500.
```

`logspace(a, b[, n])` формирует массив из 50 (если n не указано) или из n точек, равномерно распределённых в логарифмическом масштабе между значениями 10^a и 10^b; функция `logspace(a, pi)` даёт равномерное распределение из 50 точек в интервале от 10^a до pi[1].

```
--> logspace(1,2,5)
ans = 10.000    17.783    31.623    56.234    100.000

//Работа с векторами.
a=[1;2;3];
b=[2;3;4];

--> logspace(a,b,1)
ans  =
100.
1000.
10000.

--> logspace(a,b,3)
ans  =
10.     31.622777   100.
100.    316.22777   1000.
1000.   3162.2777   10000.

--> logspace(a,b,5)
ans  =
10.     17.782794   31.622777   56.234133   100.
100.    177.82794   316.22777   562.34133   1000.
1000.   1778.2794   3162.2777   5623.4133   10000.
```

`repmat(M, n[, m])` формирует матрицу, состоящую $n \times n$ или из $n \times m$ копий матрицы `M`, если `M` – скаляр, то формируется матрица, элементы которой равны значению `M`.

```
--> M=[1 2 3;4 5 6;7 8 9];
--> repmat(A,2)
ans =
1.   2.   3.   1.   2.   3.
4.   5.   6.   4.   5.   6.
7.   8.   9.   7.   8.   9.
```

[1] Обратите внимание, что это идёт вразрез с определением функции для произвольного $b \neq pi$, согласно которому интервал должен был бы быть от 10^a до 10^{pi}, сделано это для совместимости с соответствующей функцией matlab. – *Прим. ред.*

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

--> repmat(M,2)
ans =
1.   2.   3.   1.   2.   3.
4.   5.   6.   4.   5.   6.
7.   8.   9.   7.   8.   9.
1.   2.   3.   1.   2.   3.
4.   5.   6.   4.   5.   6.
7.   8.   9.   7.   8.   9.

--> repmat(M,2,3)
ans =
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.
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.

--> repmat(M,3,1)
ans =
1.   2.   3.
4.   5.   6.
7.   8.   9.
1.   2.   3.
4.   5.   6.
7.   8.   9.
1.   2.   3.
4.   5.   6.
7.   8.   9.

--> repmat(9,3)
ans =
9.   9.   9.
9.   9.   9.
9.   9.   9.
```

`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 $[A, B]$.

```
--> A=[0 1 2;3 4 5;6 7 8];
--> B=[11 12 13;14 15 16;17 18 19];
--> cat(2,A,B)
ans =
0.   1.   2.   11.   12.   13.
3.   4.   5.   14.   15.   16.
6.   7.   8.   17.   18.   19.

--> [A,B]
ans =
0.   1.   2.   11.   12.   13.
```

```
3.   4.   5.   14.   15.   16.
6.   7.   8.   17.   18.   19.

--> cat(1,A,B)
ans =
0.    1.    2.
3.    4.    5.
6.    7.    8.
11.   12.   13.
14.   15.   16.
17.   18.   19.

--> [A;B]
ans =
0.    1.    2.
3.    4.    5.
6.    7.    8.
11.   12.   13.
14.   15.   16.
17.   18.   19.

--> x1=[2;-3; 0];x2=[0; 1;-2];
--> cat(2,x1,x2)
ans =
2.   0.
-3.   1.
0.   -2.

--> cat(1,x1,x2)
ans =
2.
-3.
0.
0.
1.
-2.

--> x1=[2 -3 0];x2=[0 1 -2];
--> cat(2,x1,x2)
ans =
2.  -3.   0.   0.   1.  -2.

--> [x1 x2]
ans =
2.  -3.   0.   0.   1.  -2.

--> cat(1,x1,x2)
ans =
2.  -3.   0.
0.   1.  -2.

--> [x1;x2]
ans =
2.  -3.   0.
0.   1.  -2.
```

`tril(M[, k])` формирует из матрицы `M` нижнюю треугольную матрицу, начиная с главной или с *k*-й диагонали.

```
--> M=[0 1 2 3;4 5 6 7;8 9 0 1; 6 5 4 3];
--> tril(M)
ans =
0.   0.   0.   0.
4.   5.   0.   0.
8.   9.   0.   0.
6.   5.   4.   3.

--> tril(M,1)
ans =
0.   1.   0.   0.
4.   5.   6.   0.
8.   9.   0.   1.
6.   5.   4.   3.

--> tril(M,-1)
ans =
0.   0.   0.   0.
4.   0.   0.   0.
8.   9.   0.   0.
6.   5.   4.   0.

--> tril(M,2)
ans =
0.   1.   2.   0.
4.   5.   6.   7.
8.   9.   0.   1.
6.   5.   4.   3.

--> tril(M,-2)
ans =
0.   0.   0.   0.
0.   0.   0.   0.
8.   0.   0.   0.
6.   5.   0.   0.

--> X=[-1 2 3 9 -8 7 5];
--> tril(X)
ans =
-1.   0.   0.   0.   0.   0.   0.

--> tril(X')
ans =
-1.
2.
3.
9.
-8.
7.
5.
```

`triu(M[, k])` формирует из матрицы `M` верхнюю треугольную матрицу, начиная с главной или с *k*-й диагонали.

```
--> M=[0 1 2 3;4 5 6 7;8 9 0 1; 6 5 4 3]
M =
0.   1.   2.   3.
4.   5.   6.   7.
8.   9.   0.   1.
6.   5.   4.   3.

--> triu(M)
ans =
0.   1.   2.   3.
0.   5.   6.   7.
0.   0.   0.   1.
0.   0.   0.   3.

--> triu(M,1)
ans =
0.   1.   2.   3.
0.   0.   6.   7.
0.   0.   0.   1.
0.   0.   0.   0.

--> triu(M,-2)
ans =
0.   1.   2.   3.
4.   5.   6.   7.
8.   9.   0.   1.
0.   5.   4.   3.

X=[-1 2 3 9 -8   7 5];
--> triu(X)
ans =
-1.   2.   3.   9.  -8.   7.   5.

--> triu(X')
ans =
-1.
0.
0.
0.
0.
0.
0.
```

3.5.3 Функции, реализующие численные алгоритмы решения задач линейной алгебры

`det(M)` вычисляет определитель квадратной матрицы `M`.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> det(M)
```

```
ans = 682
--> A=[1 0 -3;0 1 2;2 0 -1];
--> det(A)
ans = 5
```

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

```
-->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.
```

`trace(M)` вычисляет след матрицы `M`, т. е. сумму элементов главной диагонали.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3]
M =
-1   1  -2   3
4   5  -1   2
3  -1   4   1
-2   5   4   3
--> trace(M)
ans = 11
--> sum(diag(M))
ans = 11
```

`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.

```
-->M=[1 0 2;3 2 1;0 3 1];
-->norm(M,1)
ans  = 5.
-->norm(M,2)
ans  = 4.5806705
-->norm(M,'inf')
```

---

1 *Ранг матрицы* – максимальное число линейно независимых строк.

2 *Вторая норма матрицы* – её 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.

```
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[, p])` возвращает число обусловленности матрицы `M`, основанное на норме *p*.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> cond(M)
ans =     3.3323623
--> cond(M,2)
ans =     3.3323623
--> cond(M,1)
ans =     6.4926686
--> cond(M,%inf)
ans =     6.7741935
--> cond(M,'fro')
ans =     5.7154131
```

`rcond(M)` – вычисляет величину, обратную значению числа обусловленности матрицы относительно первой нормы, если полученная величина близка к единице, то матрица хорошо обусловлена, если приближается к нулю, то плохо.

```
--> M=[-1 1 -2 3;4 5 -1 2;3 -1 4 1;-2 5 4 3];
--> rcond(M)
ans =     0.1738022
```

`inv(A)` вычисляет *матрицу, обратную*[1] к `A`[2].

```
-->//Пример вычисления обратной матрицы.
-->A=[1 2 3 5;0 1 3 2;4 2 1 1;2 3 0 1];
```

---

[1] 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. При умножении `A` на `inv(A)` слева должна получиться единичная матрица.

[2] Вычисление основано на методе LU-разложения.

```
-->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

//При умножении обратной матрицы на исходную
//получилась матрица, близкая к единичной.
-->inv(A)*A
ans =
1.   0.   0.   0.
0.   1.   0.   0.
0.   0.   1.   0.
0.   0.   0.   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)
inv: Задача вырождена.
```

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

```
-->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 $\mathbf{A} \cdot \vec{x} + \vec{b} = 0$.

```
//Решение системы линейных уравнений
//{x1+2*x2-7=0; x1+x2-6=0}.
//Свободные коэффициенты вводятся как вектор-столбец
//и с учётом знаков.
-->A=[1 2;1 1];b=[-7;-6];
-->x=linsolve(A,b)
x  =
5.
1.
//Результатом операции A*x+b является вектор, достаточно
//близкий к нулю, это значит, что система решена верно.
-->P=A*x+b;
```

[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 к нулю.

```
--> mprintf('%1.2f\n',P)
-0.00
0.00

//Решение системы {x1+x2-1=0; x1+x2-3=0},
//которая не имеет решений:
-->A=[1 1;1 1]; b=[-1;-3];
-->linsolve(A,b)
ВНИМАНИЕ: Конфликтующие линейные ограничения.
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
```

`spec(M)` вычисляет *собственные значения* и *собственные векторы* квадратной матрицы `M`.

```
-->M=[3 -2;-4 1]
M  =
3.      -2.
-4.     1.
-->spec(M) //Собственные числа матрицы
ans  =
5.
-1.

//X - собственные векторы, соответствующие
//собственным значениям из матрицы Y.
-->[X,Y]=spec(M)
Y  =
5.   0.
0.  -1.

X  =
0.7071068   0.4472136
-0.7071068   0.8944272
```

`rref(M)` осуществляет приведение матрицы `M` к треугольной форме, используя метод исключения Гаусса.

```
--> M=[3 -2 1 5;6 -4 2 7;9 -6 3 12]
M =
3  -2   1   5
6  -4   2   7
9  -6   3  12
--> rref(M)
```

```
ans =
1  -0.66667   0.33333   0
0   0         0         1
0   0         0         0
```

`chol(M)` возвращает разложение по Холецкому для положительно определённой симметрической матрицы `M`.

```
--> M=[10 1 1;2 10 1;2 2 10]
M =
10     1     1
2    10     1
2     2    10
--> chol(M)
ans =
3.1622777   0.3162278   0.3162278
0.          3.1464265   0.2860388
0.          0.          3.1333978
// Матрица не симметрическая
--> A=[1 2;1 1]; chol(A)
chol: Матрица не является положительно определённой.

--> M=[3 -2 1 5;6 -4 2 7;9 -6 3 12];
--> chol(M)
chol: Неверный тип аргумента 1: ожидалась квадратная матрица.
```

`lu(M)` выполняет треугольное разложение матрицы `M`[1].

```
-->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.
```

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

`qr(M)` выполняет QR-разложение, команда `[Q,R,P]=qr(M)` возвращает три матрицы: ортогональную матрицу Q, верхнюю треугольную матрицу R и матрицу перестановок P, причём $A \cdot P = Q \cdot P$.

```
--> M=[3 -2 1;5 6 -4;2 7 9];
--> [Q,R,P]=qr(M)
P  =
0.   0.   1.
0.   1.   0.
1.   0.   0.

R  =
-9.8994949  -3.7375644  -0.1010153
0.         -8.6620213  -4.3433799
0.          0.         -4.3731964

Q  =
-0.1010153   0.2744799  -0.9562723
0.404061   -0.8670267  -0.2915464
-0.9091373  -0.415843   -0.0233237

--> Z=M*P-Q*R
Z  =
9.992D-16   2.220D-15   8.882D-16
-4.441D-16  -8.882D-16  -1.776D-15
0.          0.         -4.441D-16

--> mprintf('%1.2f\t%1.2f\t%1.2f\n',Z);
0.00    0.00    0.00
-0.00   -0.00   -0.00
0.00    0.00    -0.00
```

`svd(M)` возвращает вектор сингулярных чисел матрицы, при использовании в формате `[U,S,V]=svd(M)` выполняет сингулярное разложение матрицы `M`, выдаёт три матрицы: `U` – сформирована из ортонормированных собственных векторов, отвечающих наибольшим собственным значениям матрицы $M \cdot M^T$, `V` состоит из ортонормированных собственных векторов матрицы $M \cdot M^T$, `S` – диагональная матрица из сингулярных чисел (неотрицательных значений квадратных корней из собственных значений матрицы $M \cdot M^T$), матрицы удовлетворяют условию $A = U \cdot S \cdot V^T$.

```
--> M=[3 -2 1; 5 6 -4; 2 7 9];
--> svd(M)
ans =
11.755348
8.5347066
3.7377236

--> [U,S,V]=svd(M)
V  =
0.2736345  -0.5001845   0.8215471
0.7042053  -0.4776138  -0.525338
```

```
0.6551482   0.7222884   0.2215409

S  =
11.755348   0.          0.
0.          8.5347066   0.
0.          0.          3.7377236

U =
0.0057541   0.0207345   0.9997685
0.2528901  -0.9673161   0.0186059
0.9674779   0.2527245  -0.0108095
```

`kernel([, tol[, fl]])` – определение *ядра матрицы*[1] `M`, параметры `tol` и `fl` являются необязательными. Первый задаёт точность вычислений, второй – используемый при вычислении алгоритм, принимающий значение `'qr'` или `'svd'`.

```
-->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
```

Рассмотрим некоторые задачи линейной алгебры, которые могут быть решены с помощью описанных выше функций.

3.6 Решение некоторых задач алгебры матриц

Задача 3.2

Для заданных матриц A, B и C проверить выполнение следующих тождеств:

$$(A \cdot \mathrm{B}) \cdot C = A \cdot (B \cdot C), \tag{3.4}$$

$$(A^T + B) \cdot C = A^T \cdot C + B \cdot C. \tag{3.5}$$

Из листинга 3.33 видно, что матрицы, получившиеся в результате вычисления левой и правой частей тождества (3.4), равны, следовательно, первое тождество истинно.

---

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

Листинг 3.33. Проверка матричных тождеств (пример 3.2)

```
--> A=[1 -2 0; -3 0 4];B=[3 1;2 0;-1 1];C=[1 2;-1 0];
--> (A*B)*C
ans =
-2 -2
-14 -26
--> A*(B*C)
ans =
-2 -2
-14 -26
```

Для исследования тождества (3.5) из левой части равенства вычитаем правую и получаем нулевую матрицу, что также приводит к выводу об истинности тождества. Результат представлен в листинге 3.34.

Листинг 3.34. Проверка матричных тождеств (пример 3.2)

```
--> (A'+B)*C-(A'*C+B*C)
ans =
0   0
0   0
0   0
```

Задача 3.3

Проверить, является ли матрица симметрической. Квадратная матрица называется *симметрической*, если $A^T = A$.

Из листинга 3.35 видно, что в результате вычитания из матрицы A транспонированной матрицы A^T получена нулевая матрица, т. е. тождество выполнено, заданная матрица – симметрическая.

Листинг 3.35. Проверка симметрической матрицы (пример 3.3)

```
--> A=[1 -0.5 1.5;-0.5 0 2.5;1.5 2.5 -2]; A-A’
ans =
0 0 0
0 0 0
0 0 0
```

Задача 3.4

Проверить, является ли матрица кососимметрической. Квадратная матрица называется *кососимметрической*, если $A^T = -A$.

Проверив равенство для заданной матрицы (листинг 3.36), убеждаемся в его истинности.

Листинг 3.36. Проверка кососимметрической матрицы (пример 3.4)

```
--> A=[0 -0.25 0.75;0.25 0 -1.25;-0.75 1.25 0]; A'+A
ans =
0   0   0
0   0   0
0   0   0
```

Задача 3.5

Проверить, является ли матрица ортогональной. Квадратная матрица называется *ортогональной*, если $|A| = \det A \neq 0$ и $A^T = A^{-1}$.

Для решения поставленной задачи необходимо вычислить определитель заданной матрицы и убедиться в том, что он не равен нулю. Затем следует транспонировать исходную матрицу и найти матрицу, обратную к ней. Если визуально сложно убедиться в том, что транспонированная матрица равна обратной, можно вычислить их разность. В результате должна получиться нулевая матрица (листинг 3.37).

Листинг 3.37. Проверка ортогональности матрицы (пример 3.5)

```
--> A=[0.5 0.7071 0.5;0.7071 0 -0.7071;0.5 -0.7071 0.5]
A =
0.5      0.7071   0.5
0.7071   0.      -0.7071
0.5     -0.7071   0.5

--> det(A)// Определитель матрицы A отличен от нуля
ans = -0.9999808

// При вычитании из транспонированной матрицы A обратной
// к ней матрицы получаем нулевую матрицу, значит, A - ортогональная.
--> R=A'-inv(A);
--> mprintf('%1.2f\t%1.2f\t%1.2f\n',R)
0.00    -0.00   0.00
-0.00   0.00    0.00
0.00    0.00    0.00
```

Задача 3.6

Задана матрица A. Показать, что матрица $B = 2A - E$, где E – единичная матрица, инволютивна. Квадратная матрица называется *инволютивной*, если $B^2 = E$, где E – единичная матрица.

Листинг 3.38. Проверка матрицы на инволютивность (пример 3.6)

```
--> A=[6 -15;2 -5]; B=2*A-eye(2,2); B^2
ans =
1   0
0   1
```

Задача 3.7

Решить матричные уравнения $A \cdot X = B$ и $X \cdot A = B$, выполнить проверку.

Матричное уравнение – это уравнение вида $A \cdot X = B$ или $X \cdot A = B$, где X – это неизвестная матрица. Если умножить матричное уравнение на матрицу, обратную к A, то оно примет вид: $A^{-1} \cdot A \cdot X = B \cdot A^{-1}$ или $X \cdot A \cdot A^{-1} = B \cdot A^{-1}$. Так как $A^{-1} \cdot A = A \cdot A^{-1} = E$, а $E \cdot X = X \cdot E = X$, то неизвестную матрицу X можно вычислить так: $X = A^{-1} \cdot B$ или $X = B \cdot A^{-1}$. Понятно, что матричное уравнение имеет единственное решение, если A и B – квадратные матрицы n-го порядка и определитель матрицы A не равен нулю. Как решить матричное уравнение в Scilab, показано в листинге 3.39.

Листинг 3.39. Решение матричного уравнения (пример 3.7)

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

// Решение уравнения A*X=B. Первый способ
--> X=A\B
X =
0.5555556   1.3888889
0.2962963   0.7407407

// Второй способ
--> X=inv(A)*B
X =
0.5555556   1.3888889
0.2962963   0.7407407

// Проверка A*X-B=0
--> P=A*X-B;
--> mprintf('%1.2f\t%1.2f\n',P)
0.00    0.00
-0.00   -0.00

// Решение уравнения X*A=B. Первый способ
--> X=B/A
X =
1.2222222   0.2222222
0.4074074   0.0740741

// Второй способ
--> X=B*inv(A)
X =
1.2222222   0.2222222
0.4074074   0.0740741

// Проверка X*A-B=0
--> P=X*A-B;
--> mprintf('%1.2f\t%1.2f\n',P)
0.00    0.00
0.00    0.00
```

3.7 Решение систем линейных уравнений

Система m уравнений с n неизвестными вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

называется *системой линейных алгебраических уравнений* (*СЛАУ*), причём

- x_j – *неизвестные*;
- a_{ij} – *коэффициенты* при неизвестных;
- b_i – *свободные коэффициенты* ($i = 1\ldots m, j = 1\ldots n$).

Кроме того, система из m линейных уравнений с n неизвестными может быть описана при помощи матриц: $A \cdot x = b$, где $x = \{x_j\}$ – *вектор неизвестных*; $A = \{a_{ij}\}$ – *матрица коэффициентов* при неизвестных или *матрица системы*; $b = \{b_i\}$ – *вектор свободных членов* системы или *вектор правых частей* ($i = 1\ldots m, j = 1\ldots n$).

Матрица $(A|b)$, которая формируется путём приписывания к матрице коэффициентов A столбца свободных членов b, называется *расширенной матрицей* системы.

Если все $b_i = 0$, то речь идёт об *однородной системе линейных уравнений*, иначе говорят о *неоднородной системе*.

Совокупность всех решений системы $(x_1, x_2\ldots x_n)$ называется *множеством решений*, или просто *решением системы*. Две системы уравнений называются *эквивалентными*, если они имеют одинаковое множество решений.

Однородные системы линейных уравнений $Ax = 0$ всегда разрешимы, так как последовательность $(x_1 = 0, x_2 = 0, \ldots, x_n = 0)$ удовлетворяет всем уравнениям системы. Такое решение называют *тривиальным*. Вопрос о решении однородных систем сводится к вопросу о том, существуют ли, кроме тривиального, другие, нетривиальные решения.

Система линейных уравнений может не иметь ни одного решения, и тогда она называется *несовместной*. Например, в системе

$$\begin{cases} x_1 + x_2 = 1 \\ x_1 + x_2 = 3 \end{cases}$$

левые части уравнений совпадают, а правые различны, поэтому никакие значения x_1 и x_2 не могут удовлетворить обоим уравнениям сразу.

Если же система линейных уравнений обладает решением, то она называется *совместной*. Совместная система называется *определённой*, если она обладает одним-единственным решением, и *неопределённой*, если решений больше, чем одно. Так, система

$$\begin{cases} x_1 + 2x_2 = 7 \\ x_1 + x_2 = 6 \end{cases}$$

определена и имеет единственное решение $x_1 = 5, x_2 = 1$, а система уравнений

$$\begin{cases} 3x_1 - x_2 = 1 \\ 6x_1 - 2x_2 = 2 \end{cases}$$

не определена, так как имеет бесконечное множество решений вида $x_1 = k$, $x_2 = 3k - 1$, где число k произвольно.

Совокупность всех решений неопределённой системы уравнений называется её *общим решением*, а какое-то одно конкретное решение – *частным*. Частное решение, полученное из общего при нулевых значениях свободных переменных, называется *базисным*.

При определении совместности систем уравнений важную роль играет понятие *ранга* матрицы. Пусть дана матрица A размером $n \times m$. Вычёркиванием некоторых строк или столбцов из неё можно получить квадратные матрицы k-го порядка, определители которых называются *минорами* порядка k матрицы A. Наивысший порядок не равных нулю миноров матрицы A называют рангом матрицы и обозначают $r(A)$. Из определения вытекает, что $r(A) \leq \min(n, m)$, а $r(A) = 0$, только если матрица нулевая и $r(A) = n$ для невырожденной матрицы n-го порядка. При элементарных преобразованиях (перестановка строк матрицы, умножение строк на число, отличное от нуля, и сложение строк) ранг матрицы не изменяется. Итак, если речь идёт об исследовании системы на совместность, следует помнить, что система n линейных уравнений с m неизвестными:

- *несовместна*, если $r(A|b) > r(A)$;
- *совместна*, если $r(A|b) = r(A)$, причём при $r(A|b) = r(A) = m$ имеет *единственное решение*, а при $r(A|b) = r(A) < m$ имеет *бесконечно много решений*.

Существует немало методов для практического отыскания решений систем линейных уравнений. Эти методы разделяют на *точные* и *приближённые*. Метод относится к классу точных, если с его помощью можно найти решение в результате конечного числа арифметических и логических операций.

Задача 3.8

Решить заданную систему линейных уравнений при помощи правила Крамера.

$$\begin{cases} 2x_1 - x_2 + 5x_3 = 1 \\ 3x_1 + 2x_2 - 5x_3 = 1. \\ x_1 + x_2 - 2x_3 = 4 \end{cases} \tag{3.6}$$

Правило Крамера заключается в следующем. Если определитель $\det A$ матрицы системы $Ax = b$ из n уравнений с n неизвестными отличен от нуля, то система имеет единственное решение $(x_1, x_2, \ldots, x_n)$, определяемое по формулам Крамера $x_i = \frac{\det_i}{\det}$, где $\det_i$ – определитель матрицы, полученной из мат-

рицы системы A заменой i-го столбца столбцом свободных членов b. Если определитель матрицы системы равен нулю, это не означает, что система не имеет решений: возможно, её нельзя решить по формулам Крамера.

Итак, для решения поставленной задачи необходимо выполнить следующие действия:

1) представить систему в матричном виде, т. е. сформировать матрицу системы A и вектор правых частей b;
2) вычислить главный определитель $\det A$;
3) сформировать вспомогательные матрицы для вычисления определителей $\det_i$;
4) вычислить определители $\det_i$;
5) найти решение системы по формуле $x_i = \frac{\det_i}{\det}$.

Листинг 3.40 содержит решение поставленной задачи.

Листинг 3.40. Решение СЛАУ методом Крамера (пример 3.8)

```
disp('Решение СЛАУ методом Крамера');
disp('Матрица системы:'); A=[2 -1 5;3 2 -5;1 1 -2]
disp('Вектор свободных коэффициентов:'); b=[0;1;4]
disp('Главный определитель:'); D=det(A)
disp('Вспомогательные матрицы:');
A1=A; A1(:,1)=b
A2=A; A2(:,2)=b
A3=A; A3(:,3)=b
disp('Вспомогательные определители:');
d(1)=det(A1);
d(2)=det(A2);
d(3)=det(A3);
d
disp('Вектор решений СЛАУ Ax=b'); x=d/D
disp('Проверка Ax-b=0');
P=A*x'-b;
mprintf('%1.2f\n',P)

//Результат работы программы

 Решение СЛАУ методом Крамера
Матрица системы:
A  =
2.  -1.   5.
3.   2.  -5.
1.   1.  -2.
Вектор свободных коэффициентов:
b  =
        0.
        1.
        4.
Главный определитель:
D  = 6.
```

```
Вспомогательные матрицы:
A1  =
0.  -1.   5.
1.   2.  -5.
4.   1.  -2.
A2  =
2.   0.   5.
3.   1.  -5.
1.   4.  -2.
A3  =
2.  -1.   0.
3.   2.   1.
1.   1.   4.
Вспомогательные определители:
d  =
-17.   91.   25.
Вектор решений СЛАУ Ax=b
x  =
-2.8333333   15.166667   4.1666667

Проверка Ax-b=0
-0.00
0.00
0.00
```

Решение СЛАУ по формулам Крамера выглядит достаточно громоздко, поэтому на практике его используют довольно редко.

Задача 3.9

Решить систему линейных уравнений 3.6 из примера 3.8 методом обратной матрицы.

Метод обратной матрицы: для системы из n линейных уравнений с n неизвестными $Ax = b$, при условии, что определитель матрицы A не равен нулю, единственное решение можно представить в виде $x = A^{-1}b$ (вывод формулы см. в примере 3.7).

Итак, для того чтобы решить систему линейных уравнений методом обратной матрицы, необходимо выполнить следующие действия:

1) сформировать матрицу коэффициентов и вектор свободных членов заданной системы;
2) решить систему, представив вектор неизвестных как произведение матрицы, обратной к матрице системы, и вектора свободных членов (листинг 3.41).

Листинг 3.41. Решение СЛАУ примера 3.8 методом обратной матрицы (пример 3.9)

```
disp('Решение СЛАУ методом обратной матрицы');
disp('Матрица системы:');
A=[2 -1 5;3 2 -5;1 1 -2]
disp('Вектор свободных коэффициентов:');
```

```
 b=[0;1;4]
 disp('Вектор решений СЛАУ Ax=b');
 x=A^(-1)*b
 disp('Вектор решений СЛАУ Ax=b с помощью функции inv(A)');
 x=inv(A)*b
 disp('Проверка Ax=b');
 P=A*x;
 mprintf('%1.2f\n',P)

//Результат работы программы

 Решение СЛАУ методом обратной матрицы
Матрица системы:
A  =
2.  -1.   5.
3.   2.  -5.
1.   1.  -2.
Вектор свободных коэффициентов:
b  =
        0.
        1.
        4.
Вектор решений СЛАУ Ax=b
x  =
        -2.8333333
        15.166667
        4.1666667

Вектор решений СЛАУ Ax=b с помощью функции inv(A)
x =
        -2.8333333
        15.166667
        4.1666667

Проверка Ax=b
        0.00
        1.00
        4.00
```

Задача 3.10

Решить данную систему линейных уравнений методом Гаусса.

$$\begin{cases} 2x_1 + x_2 - 5x_3 + x_4 = 8 \\ x_1 - 3x_2 - 6x_4 = 9 \\ 2x_2 - x_3 + 2x_4 = -5 \\ x_1 + 4x_2 - 7x_3 + 6x_4 = 0 \end{cases}. \tag{3.7}$$

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 путём элементарных преобразований (перестановка уравнений системы, умножение уравнений на число, отличное от нуля, и сложение уравнений) приводится к ступенчатому виду:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right) \Rightarrow \left(\begin{array}{cccc|c} 1 & c_{12} & \dots & c_{1n} & d_1 \\ 0 & 1 & \dots & c_{2n} & d_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & d_n \end{array}\right).$$

На втором этапе (обратный ход) ступенчатую матрицу преобразовывают так, чтобы в первых n столбцах получилась единичная матрица:

$$\left(\begin{array}{cccc|c} 1 & 0 & \dots & 0 & x_1 \\ 0 & 1 & \dots & 0 & x_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & x_n \end{array}\right).$$

Последний, $n + 1$ столбец этой матрицы содержит *решение системы линейных уравнений*.

Исходя из вышеизложенного, порядок решения задачи в Scilab (листинг 3.42) следующий:

1) сформировать матрицу коэффициентов A и вектор свободных членов b заданной системы;
2) сформировать расширенную матрицу системы, объединив A и b;
3) используя функцию `rref`, привести расширенную матрицу к ступенчатому виду;
4) найти решение системы, выделив последний столбец матрицы, полученной в предыдущем пункте;
5) выполнить вычисление $Ax - b$; если в результате получился нулевой вектор, задача решена верно.

Листинг 3.42. Решение СЛАУ методом Гаусса (пример 3.10)

```
disp('Решение СЛАУ методом Гаусса');
disp('Матрица системы:'); A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6]
disp('Вектор свободных коэффициентов:'); b=[8;9;-5;0]
disp('Расширенная матрица системы:'); C=rref([A b])
disp('Размерность матрицы C:'); n=size(C)
disp('Вектор решений СЛАУ Ax=b'); x=C(:,n(2))
disp('Проверка Ax-b'); P=A*x-b;
mprintf('%1.2f\n',P)

//Результат работы программы
```

```
Решение СЛАУ методом Гаусса
Матрица системы:
A  =
2.   1.  -5.   1.
1.  -3.   0.  -6.
0.   2.  -1.   2.
1.   4.  -7.   6.
Вектор свободных коэффициентов:
b =
        8.
        9.
        -5.
        0.
Расширенная матрица системы:
C  =
1.   0.   0.   0.   3.
0.   1.   0.   0.  -4.
0.   0.   1.   0.  -1.
0.   0.   0.   1.   1.
Размерность матрицы C:
n  =
4.   5.
Вектор решений СЛАУ Ax=b
x  =
        3.
        -4.
        -1.
        1.
Проверка Ax-b
0.00
0.00
0.00
-0.00
```

Задача 3.11

Решить систему линейных уравнений из примера 3.10 с помощью *LU*-разложения.

Дадим определение разложения матрицы на множители. Если все определители квадратной матрицы A отличны от нуля, то существуют такие нижняя L и верхняя U треугольные матрицы, что $A = LU$:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}.$$

Если диагональные элементы одной из матриц ненулевые, то такое разложение единственно.

Метод решения системы линейных уравнений с использованием разложения матрицы коэффициентов на множители называют *LU-разложением*, или *LU-факторизацией*.

Если матрица A исходной системы $Ax = b$ разложена в произведение треугольных матриц L и U, то можно записать уравнение: $LUx = b$.

Введя вектор вспомогательных переменных $y = (y_1, y_2, ..., y_n)^T$, уравнение $LUx = b$ можно переписать в виде системы: $\begin{cases} Ly = b \\ Ux = y \end{cases}$.

Таким образом решение системы $Ax = b$ с квадратной матрицей коэффициентов свелось к последовательному решению двух систем с треугольными матрицами коэффициентов.

Обратим внимание на тот факт, что выполнение приведённых расчётов можно интерпретировать как преобразование данной системы к треугольной. Иными словами, *LU-разложение* – это другая схема реализации метода Гаусса.

Исходя из средств, которыми располагает Scilab, решение поставленной задачи будет выглядеть так (листинг 3.43):

1) сформируем матрицу коэффициентов A и вектор свободных членов b заданной системы;
2) воспользовавшись функцией lu(A), получим матрицы L (нижняя треугольная матрица), U (верхняя треугольная матрица) и P (матрица перестановок, или, иначе, матрица, которая демонстрирует, каким образом были переставлены строки исходной матрицы при разложении на множители L и U);
3) поскольку в задаче речь идёт о решении системы, то элементы вектора b должны занять места, соответствующие строкам матрицы A, для чего необходимо выполнить действие Pb;
4) решим системы уравнений $Ly = b$ относительно y;
5) зная U и y, найдём решение системы $Ux = y$.

Листинг 3.43. Решение СЛАУ методом LU-разложения (пример 3.11)

```
disp('Решение СЛАУ методом LU-разложения');
disp('Матрица системы:'); A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6]
disp('Вектор свободных коэффициентов:'); b=[8;9;-5;0]
disp('LU-разложение:'); [L,U,P]=lu(A)
Y=rref([L P*b])
n=size(Y)
y=Y(:,n(2))
X=rref([U y])
n=size(X)
x=X(:,n(2))
disp('Проверка Ax-b'); P=A*x-b;
mprintf('%1.2f\n',P)

//Результат работы программы
```

```
Решение СЛАУ методом LU-разложения
Матрица системы:
A  =
2.   1.  -5.   1.
1.  -3.   0.  -6.
0.   2.  -1.   2.
1.   4.  -7.   6.

Вектор свободных коэффициентов:
b =
        8.
        9.
        -5.
        0.

LU-разложение:
P  =
1.   0.   0.   0.
0.   1.   0.   0.
0.   0.   0.   1.
0.   0.   1.   0.

U  =
2.   1.  -5.    1.
0.  -3.5  2.5  -6.5
0.   0.  -2.   -1.
0.   0.   0.   -1.9285714

L  =
1.    0.          0.          0.
0.5   1.          0.          0.
0.5  -1.          1.          0.
0.   -0.5714286  -0.2142857   1.

Y  =
1.   0.   0.   0.   8.
0.   1.   0.   0.   5.
0.   0.   1.   0.   1.
0.   0.   0.   1.  -1.9285714
n  =
4.   5.
y  =
        8.
        5.
        1.
-1.9285714

X  =
1.   0.   0.   0.   3.
0.   1.   0.   0.  -4.
0.   0.   1.   0.  -1.
0.   0.   0.   1.   1.
```

```
n  =
4.   5.

x  =
        3.
        -4.
        -1.
        1.

Проверка Ax-b
0.00
0.00
-0.00
0.00
```

Задача 3.12

Решить систему линейных уравнений с помощью QR-разложения.

$$\begin{cases} 3x_1 + x_2 - x_3 + 2x_4 = 6 \\ -5x_1 + x_2 + 3x_3 - 4x_4 = -12 \\ 2x_1 + x_3 - x_4 = 1 \\ x_1 - 5x_2 + 3x_3 + 3x_4 = 3 \end{cases}. \quad (3.8)$$

Квадратную матрицу A можно представить в виде произведения ортогональной матрицы Q и верхней треугольной матрицы R. Использование этого свойства матриц при решении системы линейных уравнений называют методом *QR-разложения*.

Идея решения системы этим методом аналогична той, что была описана в предыдущей задаче:

$$Ax = b \Rightarrow QRx = b \Rightarrow (Qy = b, Rx = y).$$

Таким образом, решение системы уравнений с квадратной матрицей коэффициентов сводится к решению двух систем, матрица коэффициентов первой *ортогональная*, второй – *верхняя треугольная*.

Как решить эту задачу средствами Scilab, показано в листинге 3.44.

Листинг 3.44. Решение СЛАУ методом QR-разложения (пример 3.12)

```
disp('Решение линейной системы с помощью QR-разложения');
A=[3,1,-1,2;-5,1,3,-4;2,0,1,-1;1,-5,3,-3]
b=[6;-12;1;3]
[Q,R]=qr(A)
y=Q'*b
X=rref([R y])
x=X(1:4,5:5)

//Результат работы программы
```

```
Решение линейной системы с помощью QR-разложения
A  =
3.   1.  -1.   2.
-5.   1.   3.  -4.
2.   0.   1.  -1.
1.  -5.   3.  -3.

b =
        6.
        -12.
        1.
        3.

R  =
-6.244998   1.1208971   2.081666   -3.3626912
0.        -5.0738141   3.0220526  -3.3050542
0.         0.         -2.5561425   2.7431773
0.         0.          0.          0.4938648

Q  =
-0.4803845  -0.303216   -0.3584834   0.7407972
0.8006408  -0.0202144  -0.5455182   0.2469324
-0.3202563  -0.0707504  -0.7356703  -0.5926378
-0.1601282   0.9500767  -0.1808003   0.1975459

y  =
        -13.290637
        1.2027567
        3.117247
        1.4815944

X  =
1.   0.   0.   0.   1.
0.   1.   0.   0.  -1.
0.   0.   1.   0.   2.
0.   0.   0.   1.   3.

x  =
        1.
        -1.
        2.
        3.
```

Задача 3.13

Исследовать систему на совместность и, если возможно, решить её.

$$\text{а)} \begin{cases} x_1 + 2x_2 + 2x_3 = -9 \\ x_1 - x_2 + x_3 = 2. \\ 3x_1 - 6x_2 - x_3 = 25 \end{cases}$$

$$\text{б)}\begin{cases} x_1 - 5x_2 - 8x_3 + x_4 = 3 \\ 3x_1 + x_2 - 3x_3 - 5x_4 = 1 \\ x_1 - 7x_2 + 2x_4 = -5 \\ 11x_2 + 20x_3 - 9x_4 = 2 \end{cases}.$$

$$\text{в)}\begin{cases} 4x_1 + x_2 - 3x_3 - x_4 = 0 \\ 2x_1 + 3x_2 + x_3 - 5x_4 = 0 \\ x_1 - 2x_2 - 2x_3 + 3x_4 = 0 \end{cases}.$$

Для решения задачи (листинг 3.45) введём исходные данные, т. е. матрицу коэффициентов системы и вектор правых частей. Затем выполним вычисление рангов матрицы коэффициентов и расширенной матрицы системы. Возможны три случая:

1) ранги матриц равны и совпадают с количеством неизвестных $r(A|b) = r(A) = 3$, значит, система совместна и имеет единственное решение;
2) ранг расширенной матрицы больше ранга матрицы системы $r(A|b) > r(A)$, что означает несовместность системы;
3) ранг расширенной матрицы равен рангу матрицы системы $r(A|b) = r(A) = 3$, но меньше, чем количество неизвестных системы $r(A|b) = r(A) < 4$. Значит, система совместна, но имеет бесконечное множество решений.

Листинг 3.45. Исследование системы на совместность (пример 3.13)

```
disp('Исследование системы на совместность');
disp('Введите матрицу системы:'); A=input('A=');
disp('Введите вектор свободных коэффициентов:'); b=input('b=');

[n,m]=size(A);
mprintf('Размерность системы: %d %d\n',n,m);
r=rank(A)
mprintf('Ранг матрицы системы: %d\n',r);
R=rank([A b])
mprintf('Ранг расширенной матрицы: %d\n', R);
if r==R
        disp('Система совместна.');
        if r==m
                disp('Система имеет единственное решение.');
                disp('Решение системы методом обратной матрицы:');
                x=inv(A)*b;
                disp(x);

                disp('Проверка Ax-b=0:');
                y=A*x-b;
                disp(y);
        else
                disp('Система имеет бесконечно много решений.');
        end;
else
        disp('Система не совместна');
```

```
end;

//Результат работы программы

Исследование системы а) на совместность

Введите матрицу системы:
A=[1 2 5;1 -1 3; 3 -6 -1];
Введите вектор свободных коэффициентов:
b=[-9;2;25];
Размерность системы: 3 3
Ранг матрицы системы: 3
Ранг расширенной матрицы: 3
Система совместна.
Система имеет единственное решение.
Решение системы методом обратной матрицы:
2.
-3.
-1.
Проверка Ax-b=0:
0.
0.
-3.553D-15

Исследование системы б) на совместность

 Введите матрицу системы:
A=[1 -5 -8 1;3 1 -3 -5;1 0 -7 2;0 11 20 -9]
Введите вектор свободных коэффициентов:
b=[3;1;-5;2]
Размерность системы: 4 4
Ранг матрицы системы: 3
Ранг расширенной матрицы: 4
Система не совместна

Исследование системы в) на совместность

Введите матрицу системы:
A=[4 1 -3 -1;2 3 1 -5;1 -2 -2 4]
Введите вектор свободных коэффициентов:
b=[0;0;0]
Размерность системы: 3 4
Ранг матрицы системы: 3
Ранг расширенной матрицы: 3
Система совместна.
Система имеет бесконечно много решений.
```

3.8 Собственные значения и собственные векторы

Пусть A – матрица размерностью $n \times n$. Любой ненулевой вектор x, принадлежащий некоторому векторному пространству, для которого $Ax = \lambda x$, где λ – некоторое число, называется *собственным вектором матрицы*, а λ – принадлежащим ему или соответствующим ему *собственным значением матрицы* A.

Уравнение $Ax = \lambda x$ эквивалентно уравнению $(A - \lambda E)x = 0$. Это однородная система линейных уравнений, нетривиальные решения которой являются искомыми *собственными векторами*. Она имеет нетривиальные решения только тогда, когда $r(A - \lambda E) < n$, т. е. если $\det(A - \lambda E) = 0$. Многочлен $\det(A - \lambda E)$ называется *характеристическим многочленом матрицы A*, а уравнение $\det(A - \lambda E) = 0$ – *характеристическим уравнением матрицы A*. Если λ_i – собственные значения A, то нетривиальные решения однородной системы линейных уравнений $\det(A - \lambda E) = 0$ есть *собственные векторы A*, принадлежащие собственному значению λ_i. Множество решений этой системы уравнений называют *собственным подпространством матрицы A*, принадлежащим собственному значению λ_i, каждый ненулевой вектор собственного подпространства является *собственным вектором матрицы A*.

Иногда требуется найти собственные векторы y и собственные значения $\hbar$, определяемые соотношением $Ay = \hbar By$, $(y \neq 0)$, где B – невырожденная матрица. Векторы y и числа $\hbar$ обязательно являются собственными векторами и собственными значениями матрицы $B^{-1}A$. Пусть $A = a_{ij}$ и $B = b_{ij}$, причём матрица B является положительно определённой, тогда собственные значения $\hbar$ совпадают с корнями уравнения n-й степени $\det(A - \hbar B) = \det(a_{ij} - \hbar b_{ij}) = 0$. Это уравнение называют *характеристическим уравнением* для обобщённой задачи о собственных значениях. Для каждого корня $\hbar$ кратности m существует ровно m линейно независимых собственных векторов y.

Задача 3.14

Найти собственные значения и собственные векторы матрицы A.

В листинге 3.46 показано решение поставленной задачи.

Листинг 3.46. Нахождение собственных значений (пример 3.14)

```
disp('Введите матрицу:'); A=input('A=');
[n,m]=size(A);
d=spec(A)
disp('Вектор собственных значений матрицы A:');disp(d);
[L, D]=spec(A);
disp('L - Матрица собственных векторов:'); disp(L);
disp('D - Диагональная матрица собственных значений:');
[N M]=size(D);
for i=1:N
        for j=1:M
                mprintf('%2.1f\t',D(i,j));
        end
        mprintf('\n');
end
disp('Проверка:');
for i=1:n
p=(A-D(i,i)*eye(n,n))*L(:,i);
mprintf('%2.2f\t',p);
mprintf('\n');
end;
```

```
//Результат работы программы

Введите матрицу:
A=[5 2 -1;1 -3 2; 4 5 -3]

Вектор собственных значений матрицы A:
4.9083269
-1.180D-16
-5.9083269

L - Матрица собственных векторов:
-0.796113   -0.0493264   0.1813029
-0.2410436   0.54259    -0.5988027
-0.5550694   0.8385482   0.7801055

D - Диагональная матрица собственных значений:
4.9             0.0             0.0
0.0             -0.0            0.0
0.0             0.0             -5.9

Проверка:
-0.00    0.00    -0.00
0.00     0.00    0.00
0.00     -0.00   -0.00
```

Задача 3.15

Привести заданную матрицу к диагональному виду.

Задача состоит в том, чтобы для квадратной матрицы A подобрать такую матрицу C, чтобы матрица $B = C^{-1}AC$ имела диагональный вид. Эта задача связана с теорией собственных значений, так как разрешима только в том случае, если матрица C состоит из собственных векторов матрицы A.

Листинг 3.47. Приведение к диагональному виду (пример 3.15)

```
disp('Введите матрицу:'); A=input('A=');
[C,D]=spec(A);
disp('Диагональная матрица к матрице A:');
[N M]=size(D);
for i=1:N
        for j=1:M
                mprintf('%2.1f\t',D(i,j));
        end
        mprintf('\n');
end

disp('Проверка B=D');
B=inv(C)*A*C
[N M]=size(B);
for i=1:N
        for j=1:M
                mprintf('%2.1f\t',B(i,j));
```

```
        end
        mprintf('\n');
end

//результат работы программы

Введите матрицу:
A=[2 1 3;1 -2 1;3 2 2]

Диагональная матрица к матрице A:
5.4             0.0             0.0
0.0             -1.0            0.0
0.0             0.0             -2.4

Проверка B=D
5.4             -0.0            -0.0
-0.0            -1.0            -0.0
0.0             0.0             -2.4
```

3.9 Норма и число обусловленности матрицы

Матричная норма – это некоторая скалярная числовая характеристика, которую ставят в соответствие матрице. В задачах линейной алгебры используются различные матричные нормы:

- первая норма $||A||_1$ квадратной матрицы $A = \{a_{ij}\}$:

  $$\|A\|_1 = \max \sum_{i=1}^{n} |a_{ij}|;$$

- вторая норма $||A||_2$ квадратной матрицы $A = \{a_{ij}\}$:

  $$\|A\|_2 = \sqrt{\lambda_{\max}(AA^T)},$$

  где $\sqrt{\lambda_{\max}(AA^T)}$ – максимальное собственное значение матрицы $A = \{a_{ij}\}$;

- евклидова норма $||A||_e$ квадратной матрицы $A = \{a_{ij}\}$:

  $$\|A\|_e = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2};$$

- бесконечная норма $||A||_i$ квадратной матрицы $A = \{a_{ij}\}$:

  $$\|A\|_i = \max \sum_{j=1}^{n} |a_{ij}|.$$

Число обусловленности матрицы A используется для определения меры чувствительности системы линейных уравнений $Ax = b$ к погрешностям задания вектора b. Чем больше число обусловленности, тем более неустойчив процесс нахождения решения системы. Существует несколько вариантов вычисления числа обусловленности, но все они связаны с нормой матрицы и равны произведению нормы исходной матрицы на норму обратной:

- число обусловленности матрицы, вычисленное в норме $||A||_1$: $N = ||A||_1 \cdot ||A^{-1}||_1$;
- число обусловленности матрицы, вычисленное в норме $||A||_2$: $M = ||A||_2 \cdot ||A^{-1}||_2$;
- число обусловленности матрицы, вычисленное в норме $||A||_i$: $P = ||A||_i \cdot ||A^{-1}||_i$;
- число обусловленности матрицы, вычисленное в норме $||A||_e$: $H = ||A||_e \cdot ||A^{-1}||_e$.

Задача 3.16

Вычислить нормы и числа обусловленности матрицы A.

В листинге 3.48 приведён фрагмент документа, в котором происходит вычисление норм матрицы A с помощью функции norm и по соответствующим формулам. Вычисление чисел обусловленности проведено при помощи функции cond(A) и по формулам, отражающим зависимость числа обусловленности от соответствующей нормы матрицы.

Листинг 3.48. Вычисление матричных норм (пример 3.16)

```
disp('Введите матрицу:'); A=input('A='); [n,m]=size(A);

n_1=norm(A,1); N_1=max(sum(abs(A),'c'));
mprintf('Первая норма: %2.2f\n', N_1);

n_2=norm(A,2); N_2= sqrt(max(spec(A*A')));
mprintf('Вторая норма: %2.2f\n',N_2);

n_i= norm(A,'inf'); N_i=max(sum(abs(A'),'c'));
mprintf('Бесконечная норма: %2.2f\n',N_i);

n_e= norm(A,'fro'); N_e= sqrt(sum(diag(A*A')));
mprintf('Евклидова норма: %2.2f\n', N_e);

c_1= cond(A,1); C_1= norm(A,1)*norm(inv(A),1);
mprintf('Число обусловленности в первой норме: %2.2f\n',C_1);

c_2=cond(A,2); C_2=norm(A,2)*norm(inv(A),2);
mprintf('Число обусловленности во второй норме: %2.2f\n',C_2);

c_i= cond(A,'inf'); C_i= norm(A,'inf')*norm(inv(A),'inf');
mprintf('Число обусловленности в бесконечной норме: %2.2f\n',C_i);

c_e= cond(A,'fro'); C_e= norm(A,'fro')*norm(inv(A),'fro');
mprintf('Число обусловленности в евклидовой норме: %2.2f\n',C_e);

//Результат работы программы
```

```
 Введите матрицу:
A=[5 7 6 5;7 10 8 7;6 8 10 9;5 7 9 10]

Первая норма: 33.00
Вторая норма: 30.29
Бесконечная норма: 33.00
Евклидова норма: 30.55
Число обусловленности в первой норме: 4488.00
Число обусловленности во второй норме: 2984.09
Число обусловленности в бесконечной норме: 4488.00
Число обусловленности в евклидовой норме: 3009.58
```

Глава 4

Построение графиков в Scilab

В этой главе читатель может познакомиться с графическим аппаратом Scilab для построения двумерных и трёхмерных графиков.

4.1 Построение графиков в декартовой системе координат

Декартова (прямоугольная) система координат задаётся двумя перпендикулярными прямыми, называемыми осями координат. Горизонтальная прямая X – ось абсцисс, а вертикальная Y – ось ординат. Точку пересечения осей называют началом координат. Четыре угла, образованные осями координат, носят название координатных углов. Положение точки в прямоугольной системе координат определяется значением двух величин, называемых координатами точки. Если точка имеет координаты x и y, то x – абсцисса точки, y – ордината. Уравнение, связывающее координаты x и y, является уравнением линии, если координаты любой точки этой линии удовлетворяют ему.

Величина y называется *функцией* переменной величины x, если каждому из тех значений, которые может принимать x, соответствует одно или несколько определённых значений y. При этом переменная величина x называется аргументом функции $y = f(x)$. Говорят также, что величина y зависит от величины x. Функция считается заданной, если для каждого значения аргумента существует соответствующее значение функции. Чаще всего используют следующие способы задания функций:

- табличный – числовые значения функции уже заданы и занесены в таблицу; недостаток заключается в том, что таблица может не содержать все нужные значения функции;
- графический – значения функции заданы при помощи линии (графика), у которой абсциссы изображают значения аргумента, а ординаты – соответствующие значения функции;
- аналитический – функция задаётся одной или несколькими формулами (уравнениями); при этом если зависимость между x и y выражена уравнением, разрешённым относительно y, то говорят о явно заданной функции, в противном случае функция считается неявной.

Совокупность всех значений, которые может принимать в условиях поставленной задачи аргумент x функции $y = f(x)$, называется областью определения этой функции. Совокупность значений y, которые принимает функция $f(x)$, называется множеством значений функции.

Для того чтобы построить график функции $f(x)$, необходимо сформировать два массива x и y одинаковой размерности, а затем обратиться к функции `plot`:

```
plot(x,y)
```

Здесь x – массив абсцисс; y – массив ординат.

В простейшем случае обращение к функции имеет вид plot(y). В качестве массива x выступает массив номеров точек массива y. В листинге 4.1 и на рис. 4.1 представлен пример построения графика функции $y = f(i)$, где i – номер точки в массиве y.

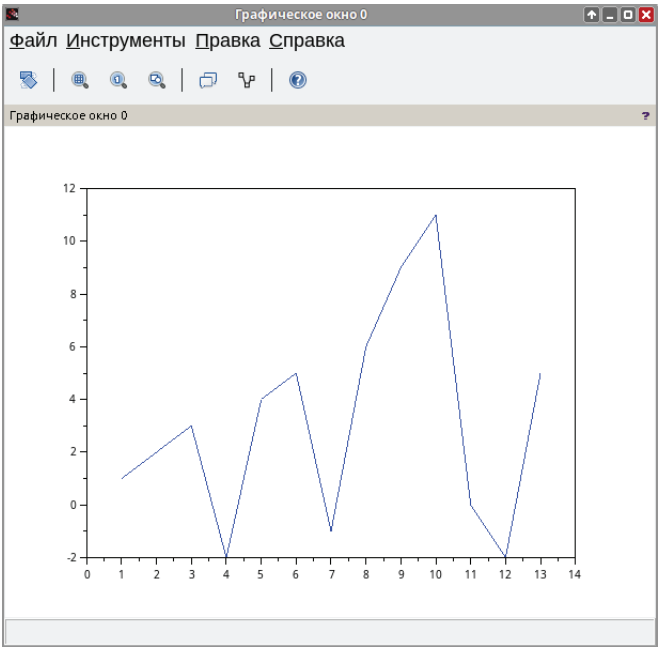

Рис. 4.1. График функции $y = f(i)$

В результате обращения к функции plot(y) будет создано окно с именем *Графическое окно 0*, в котором будет построен график функции $y = f(i)$.

График формируется путём соединения соседних точек прямыми линиями. Чем больше интервал между соседними точками (т. е. чем меньше точек), тем больше заметно, что график представляет из себя ломаную.

Листинг 4.1. Построение графика функции вида $y = f(i)$

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

4.2 Особенности работы функции plot

В Scilab любой объект является матрицей. В функции plot матрица *x* отвечает за построение оси абсцисс. Матрица *y* отвечает за построение графика функции по заданным значениям в заданных точках. Для нормального построения графика должны совпадать размерности матриц *x* и *y*.

Функция считывает матрицу следующим образом: количество строк матрицы – это количество точек для построения графика, а количество столбцов матрицы – это количество функций.

Пример: пусть матрица *X* выглядит так:

```
0.
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.
```

Матрица `y=sin(x)`:

```
0.
0.0998334
0.1986693
0.2955202
0.3894183
0.4794255
0.5646425
0.6442177
0.7173561
0.7833269
0.841471
```

В таком случае функция `plot(x,y)` построит один график по одиннадцати точкам (см. рис. 4.2).

Добавим ещё один вектор-столбец (т. е. матрицу размером `Nx1`) `z=cos(x)`:

```
1.
0.9950042
0.9800666
0.9553365
0.921061
0.8775826
0.8253356
0.7648422
0.6967067
0.62161
0.5403023
```

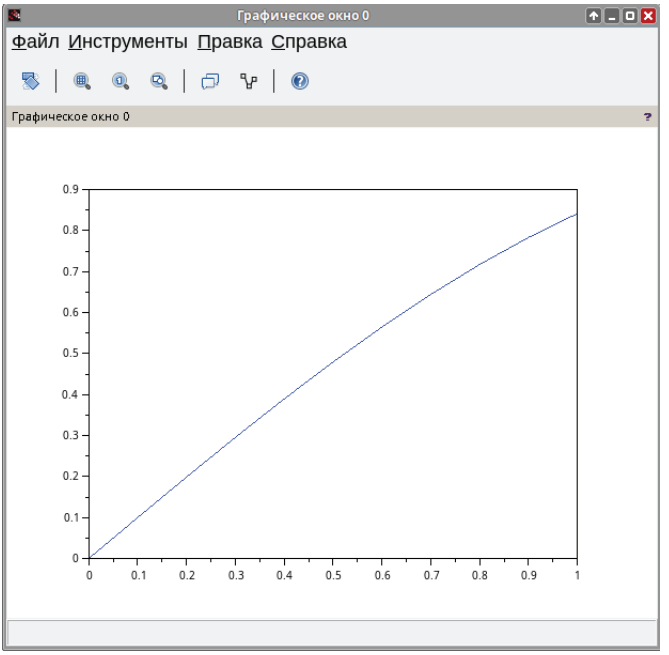

Рис. 4.2. Результат работы функции `plot(x,y)`

Чтобы сохранить количество входных параметров, объединим y и z в одну матрицу.

```
0.          1.
0.0998334   0.9950042
0.1986693   0.9800666
0.2955202   0.9553365
0.3894183   0.921061
0.4794255   0.8775826
0.5646425   0.8253356
0.6442177   0.7648422
0.7173561   0.6967067
0.7833269   0.62161
0.841471    0.5403023
```

Теперь функция `plot(x,[y;z])` построит два графика по одиннадцати точкам (см. рис. 4.3).

Однако при написании кода в Scilab:

```
x=0:0.1:1;
y=sin(x);
z=cos(x);
plot(x,[y;z]);
```

кроме окна с графиком выводится сообщение:

```
ВНИМАНИЕ: Транспонирование вектора-строки X для совместимости размеров
ВНИМАНИЕ: Транспонирование матрицы данных Y для получения совместимости
размерности
```

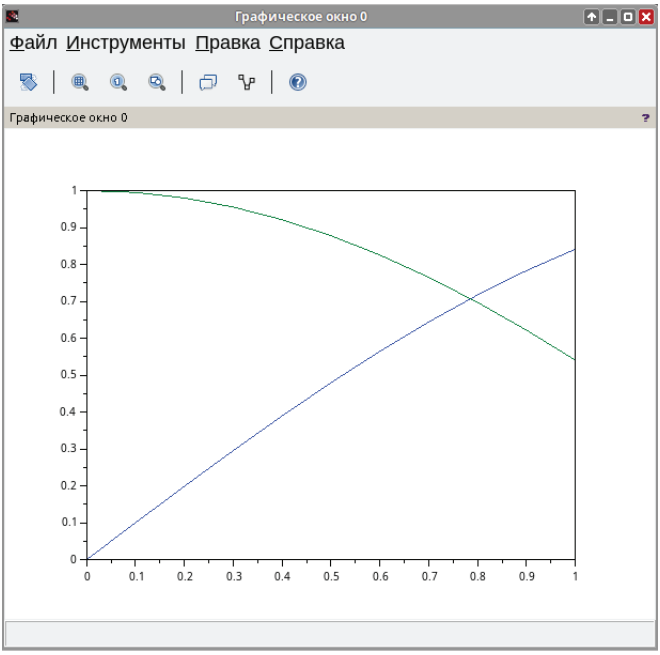

Рис. 4.3. Результат работы функции `plot(x,[y;z])`

Дело в том, что при создании массива стандартным способом формируется матрица с размерами `1xN` (1 строка на *N* столбцов).

```
--> x=0:0.1:1
x = 0.00 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 1.00

--> y=sin(x)
y = 0.000 0.100 0.199 0.296 0.389 0.479 0.565 0.644 0.717 0.783 0.841

--> z=cos(x)
z = 1.000 0.995 0.980 0.955 0.921 0.878 0.825 0.765 0.697
0.622 0.540
```

И матрица `[y;z]` выглядит так:

```
0.000 0.100 0.199 0.296 0.389 0.479 0.565 0.644 0.717 0.783 0.841
1.000 0.995 0.980 0.955 0.921 0.878 0.825 0.765 0.69  0.622 0.540
```

Если количество строк в матрице – это количество точек для построения графика, то массив *X* задаёт всего одну. Построить график в единственной точке нельзя, следовательно, массив *X* не выполняет свою функцию.

Scilab предотвращает такое поведение при помощи автоматического транспонирования. Именно об этом и выводится сообщение. Если транспонировать матрицу `[y;z]`, т. е. вызвать функцию

```
--> plot(x,[y;z]')
```

будет выведено одно сообщение:

```
ВНИМАНИЕ: Транспонирование вектора-строки X для совместимости размеров
```

так как матрица `[y;z]` введена корректно и транспонировать нужно только вектор `X`.

Однако автоматическое транспонирование происходит не всегда.

Например, нужно построить графики функций `y=sin(x)` на интервале `[0;1]` и `z=cos(x)` на интервале `[1;2]`. Оба графика должны быть расположены в одной системе координат.

Первое, что нужно сделать для решения задачи, – разделить общий интервал `[0;2]` на два: `[0;1]` и `[1;2]`. Сделаем это с помощью массивов `x1` и `x2`.

```
//Шаг 0.1
--> x1=0:0.1:1
x1 = 0.   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.
--> x2=1:0.1:2
x2  = 1.   1.1   1.2   1.3   1.4   1.5   1.6   1.7   1.8   1.9   2.
```

Теперь зададим функции `y=sin(x)` и `z=cos(x)` с учётом того, что каждая функция должна быть отображена на своём интервале.

```
--> y=sin(x1)
y  = 0.000  0.100  0.199  0.296  0.389  0.479  0.565  0.644  0.717
0.783 0.841
--> z=cos(x2)
z  = 0.540  0.454  0.362  0.267  0.170  0.071  -0.029  -0.129
 -0.227  -0.323  -0.416
```

Для сохранения количества входных параметров объединим массивы `x1` и `x2` в одну матрицу, а `y` и `z` – в другую. Матрица `[x1;x2]`:

```
0.   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.
1.   1.1   1.2   1.3   1.4   1.5   1.6   1.7   1.8   1.9   2.
```

Матрица `[y;z]`:

```
0.000 0.100 0.199 0.296 0.389 0.479 0.565 0.644 0.717   0.783 0.841
0.540 0.454 0.362 0.267 0.170 0.071 -0.029 -0.129 -0.227 -0.323 -0.416
```

Вызываем функцию

```
-->plot([x1;x2],[y;z])
```

Сообщения о транспонировании нет, а в окне Scilab построено одиннадцать графиков по двум точкам (см. рис. 4.4).

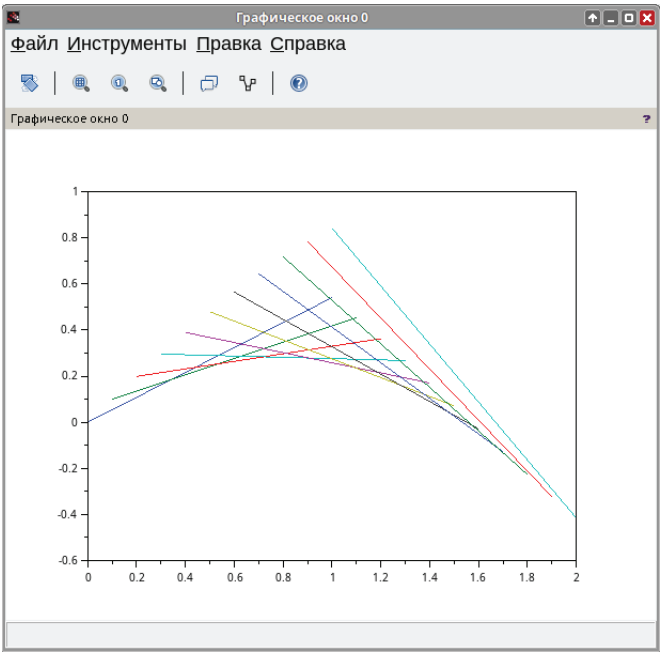

Рис. 4.4. Результат работы функции plot([x1;x2],[y;z])

Если посмотреть на вид вышеприведённых матриц, можно увидеть, что количество строк равно двум, а количество столбцов – одиннадцати. Иначе говоря, количество точек для построения графика равно двум, а количество функций – одиннадцати. Здесь нет никаких несоответствий, а значит, не нужно ничего транспонировать.

Автоматическое транспонирование применяется только к векторам.

А вот для вывода двух графиков функций по одиннадцати точкам следует при вызове функции `plot` явно указать, что матрицы должны быть транспонированы. Тогда

```
--> plot([x1;x2]',[y;z]')
```

выводит ожидаемые графики функций(см. рис. 4.5).

Также обратите внимание, что и матрица `[x1;x2]`, и матрица `[y;z]` имеют одинаковые размеры `2x11` (или `11x2` в случае транспонирования). Совпадение размеров матриц крайне важно, так как первым параметром в функцию передаются точки для построения графика, а вторым – значения в этих точках. Если размеры матриц будут разные, то функция `plot` не сможет построить график. Будет выведено сообщение:

```
plot: Неверный размер входных аргументов №2 и №3: размеры несовместимы.
```

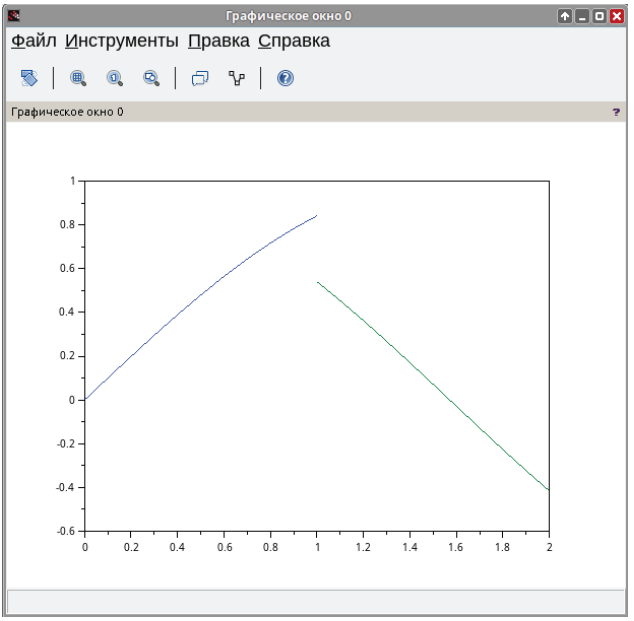

Рис. 4.5. Результат работы функции `plot([x1;x2]',[y;z]')`

Задача 4.1

Построить график функции $y = \sin(\cos(x))$ на интервале $[-2\pi; 2\pi]$ с шагом $\pi/10$ с помощью функции `plot`.

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 (см. листинг 4.2, рис. 4.6).

Листинг 4.2. Построение графика функции $y = \sin(\cos(x))$

```
x=-2*%pi:%pi/10:2*%pi;// Формирование массива x.
y=sin(cos(x));// Формирование массива y.
plot(x,y)// Построение графика функции.
```

Задача 4.2

Построить график функции $y = \sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x)$ на интервале $[-10; 10]$ с шагом 0.1.

Решение этой задачи представлено в листинге 4.3.

Листинг 4.3. Построение графика (пример 4.2)

```
x=-10:0.1:10;
y=sin(x)+sin(3*x)/3+sin(5*x)/5;
plot(x,y)
```

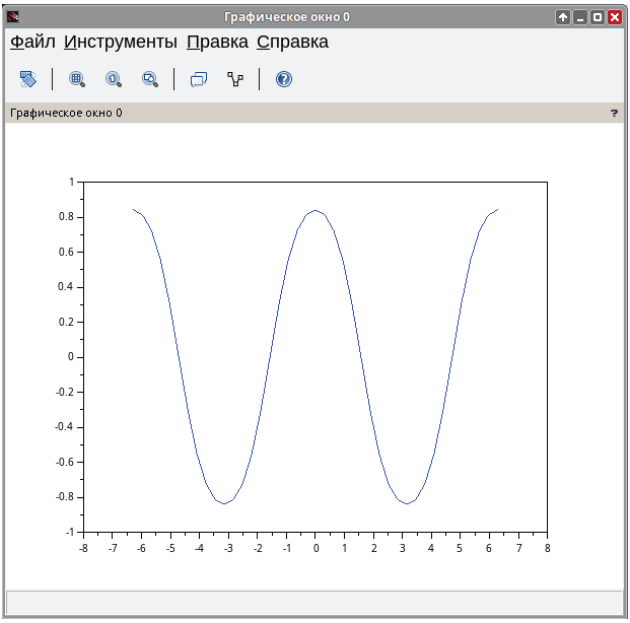

Рис. 4.6. График функции $y = \sin(\cos(x))$

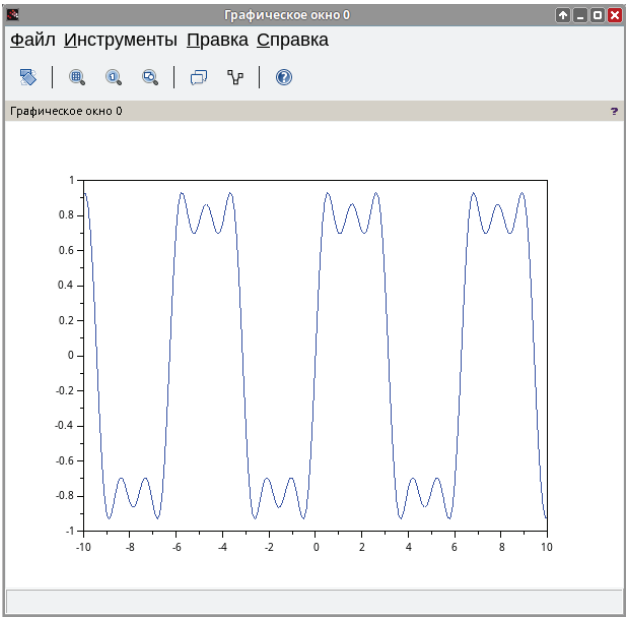

Рис. 4.7. График функции $y = \sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x)$

4.3 Построение нескольких графиков в одной системе координат

Построить несколько графиков в одной системе координат можно несколькими способами:

1) в функции `plot(x,y)` вторым параметром указать матрицу, в которой количество столбцов равно количеству функций;
2) использовать расширенный синтаксис функции `plot`:

   `plot(x1,y1,x2,y2,...,xn,yn);`
3) повторно вызвать команду `plot` с другими параметрами.

Задача 4.3

В одной системе координат построить графики функций $y = \sin(\cos(x))$, $z = \cos(\sin(x))$, $v = e^{\sin(x)}$, $t = e^{\cos(x)}$ на интервале $[-2\pi; 2\pi]$ с шагом 0,1.

Листинг 4.4. Построение графиков нескольких функций. Способ 1

```
x=-2*%pi:%pi/10:2*%pi;
y=sin(cos(x));
z=cos(sin(x));
v=exp(sin(x));
t=exp(cos(x));
plot(x',[y;z;v;t]');
```

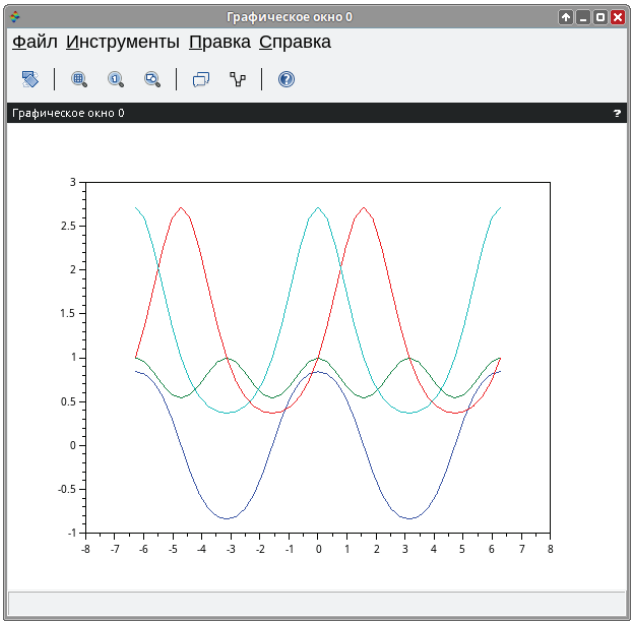

Рис. 4.8. Построение графиков нескольких функций

Вообще говоря, не обязательно формировать для каждой функции свой массив значений. Достаточно указать в квадратных скобках через точку с запятой их математические выражения, и массивы автоматически будут созданы как промежуточный этап построения функций.

```
x=-2*%pi:%pi/10:2*%pi;
plot(x',[sin(cos(x));cos(sin(x));exp(sin(x));exp(cos(x));]');
```

Задача 4.4
В одной системе координат построить графики функций $y = \sin\left(\frac{x}{2}\right)$, $z = \cos(x)$ и $v = e^{\cos(x)}$. Интервал изменения x: [−6,28; 6,28], шаг: 0.02.

Для построения нескольких графиков в одной системе координат также 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-го графика.

В нашем случае обращение к функции `plot` будет иметь вид `plot(x,y,x,z,x,v)` (листинг 4.5 и рис. 4.9).

Листинг 4.5. Построение графиков нескольких функций. Способ 2

```
//Вариант 1
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);

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

Обратите внимание, что при построении графиков этим способом Scilab автоматически изменяет цвета кривых, изображаемых в одной системе координат.

Задача 4.5
Построить в одной системе координат графики функций $y = \sin\left(\frac{x}{2}\right)$, $z = \cos(x)$ и $v = e^{\cos(x)}$. Интервал изменения x: [−6,28; 6,28], шаг: 0.02.

Если повторно обратиться к функции `plot`, новый график функции будет дорисован поверх предыдущего в этом же окне.

Следовательно, несколько графиков в одной системе координат можно вывести повторным вызовом функции `plot` с другими параметрами.

Чтобы стереть график и построить вместо него новый, достаточно перед вызовом `plot` вызвать функцию

```
mtlb_hold('off')
```

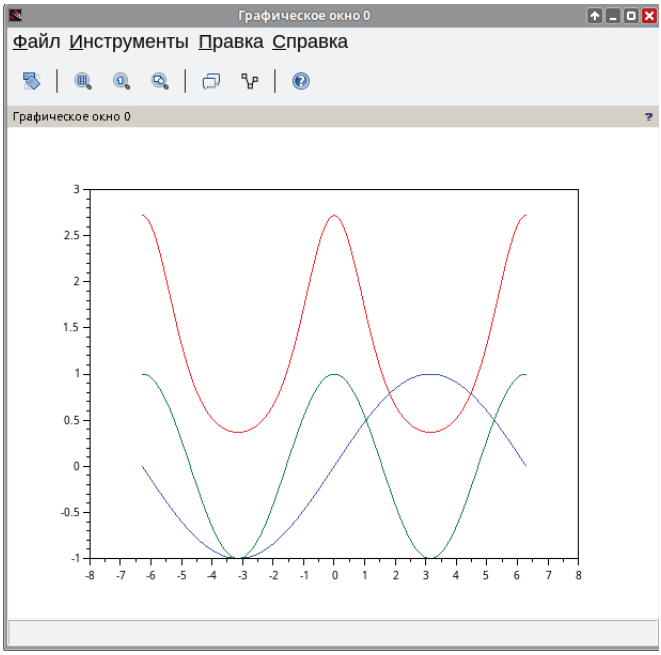

Рис. 4.9. Графики нескольких функций

Листинг 4.6. Построение графиков нескольких функций. Способ 3

```
//Вывод нескольких графиков в одной системе координат
x=-6.28:0.02:6.28;
y=sin(x/2);
z=cos(x);
v=exp(cos(x));
plot(x,y);
plot(x,z);
plot(x,v);

//Перерисовка графиков
x=-6.28:0.02:6.28;
y=sin(x/2);
z=cos(x);
v=exp(cos(x));
mtlb_hold('off');
plot(x,y);
plot(x,z);
plot(x,v);
//В результате в графическом окне остаётся
//только последний график
```

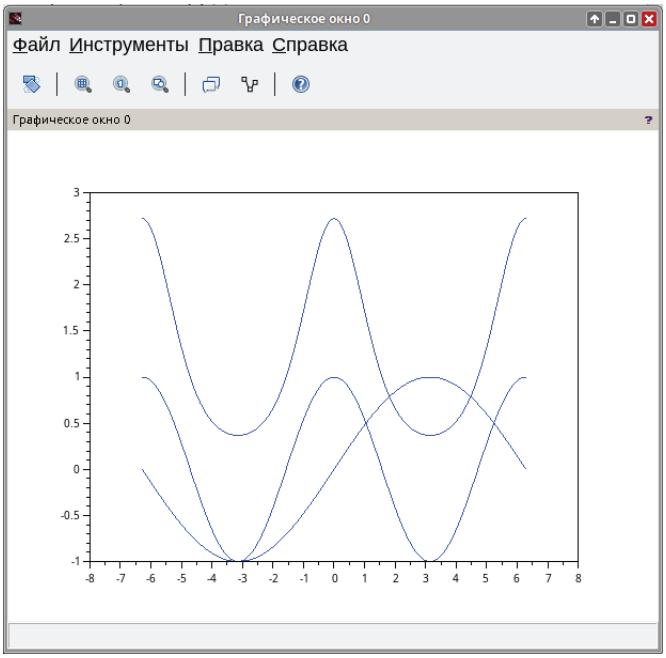

Рис. 4.10. Графики нескольких функций

4.4 Построение нескольких графиков в одном графическом окне

В Scilab можно выводить несколько графиков в одном окне, не совмещая их в одной системе координат.

Вывести несколько графиков в одном окне можно с помощью функции `subplot`. Она разделяет графическое окно на несколько отдельных областей, в каждой из которых строится свой график с собственной системой координат. Обращение к функции 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.

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. Пусть в каждом столбце по вертикали будет три, а по горизонтали две 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. 4.11).

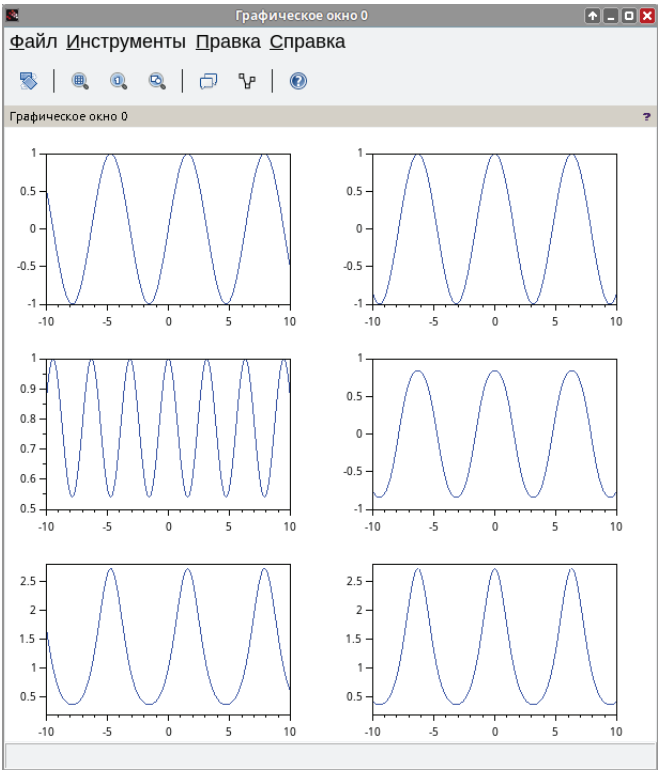

Рис. 4.11. Графики шести функций в одном графическом окне

Листинг 4.7. Несколько графиков в одном окне, но в разных системах координат

```
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);
```

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 трёх 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.

Задача 4.7

Построить графики функций $y = \sin\left(\frac{x}{2}\right)$, $z = \cos(x)$ и $v = e^{\cos(x)}$ в одних координатных осях и независимо определить внешний вид каждого графика.

Пусть график функции $y = \sin\left(\frac{x}{2}\right)$ будет штриховым, $z = \cos(x)$ – чёрного цвета, с маркером в виде звёздочки, $v = \exp^{(\cos(x))}$ – штриховым, красного цвета, с маркером в виде точки (см. листинг 4.8 и рис. 4.12).

Листинг 4.8. Изменение внешнего вида графика с помощью `plot`

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

Таблица 4.1. Символы, определяющие цвет линии графика

Символ	Описание
y	Жёлтый
m	Розовый
c	Голубой
r	Красный
g	Зелёный
b	Синий
w	Белый
k	Чёрный

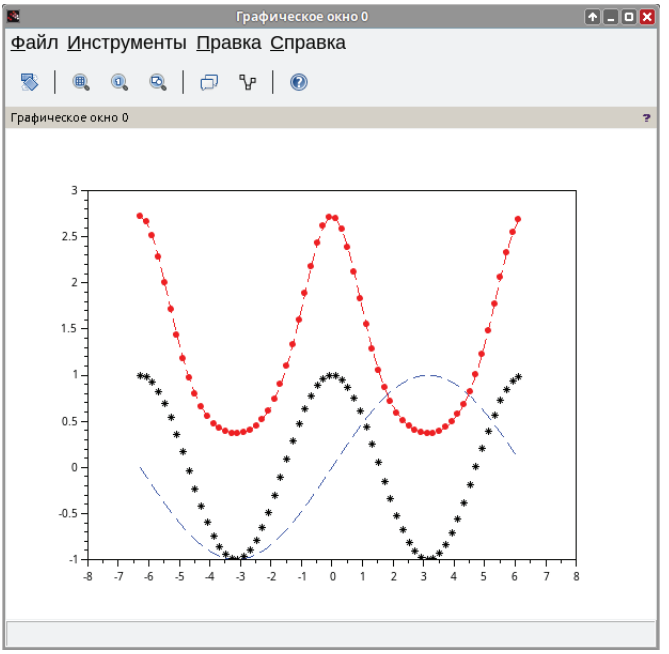

Рис. 4.12. Управление свойствами линии графика при помощи функции `plot`

Таблица 4.2. Символы, определяющие тип маркера

Символ	Описание
`.`	Точка
`o`	Кружок
`x`	Крестик
`+`	Знак «плюс»
`*`	Звёздочка
`s`	Квадрат
`d`	Ромб
`v`	Треугольник вершиной вниз
`^`	Треугольник вершиной вверх
`<`	Треугольник вершиной влево
`>`	Треугольник вершиной вправо
`p`	Пятиконечная звезда

Чтобы график было проще анализировать, удобно выводить сетку – дополнительные оси для *X* и *Y*. В Scilab это можно сделать с помощью команды

```
xgrid(цвет, толщина линии, стиль)
```

где `цвет` определяет цвет линии сетки; `толщина линии` определяет толщину линии сетки; `стиль` определяет стиль линии сетки.

Таблица 4.3. Символы, определяющие тип линии графика

Символ	Описание
-	Сплошная (по умолчанию)
:	Штрих, чередующийся с двумя точками
-.	Штрих, чередующийся с одной точкой
--	Штриховая

Каждый параметр принимает либо число, либо вектор чисел из двух элементов (первый отвечает за внешний вид вертикальных линий сетки, второй – за внешний вид горизонтальных).

Некоторые из возможных цветов:

- 1 – чёрный;
- 2 – синий;
- 3 – зелёный;
- 4 – голубой;
- 5 – красный;
- 6 – розовый;
- 7 – жёлтый;
- 8 – белый.

Толщина задаётся числом от 1 до 10. Некоторые из возможных стилей:

- 1 – сплошная линия;
- 2 – штрих;
- 3 – мелкий штрих;
- 4 – точка и штрих;
- 5 – две точки и штрих;
- 6 – штрих и тире;
- 7 – точки;
- 8 – тире.

Если оставить скобки пустыми, по умолчанию будет прорисована сетка чёрного 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 4.9 и рис. 4.13).

Листинг 4.9. Вывод сетки графика с помощью команды `xgrid`

```
x=-10:0.1:10;
y=sin(cos(x));
z=cos(sin(x));
plot(x,y,'m*',x,z,'gd'); //построение y - розовый график из "*",
// z - зелёный график из ромбов.
//Сетка по умолчанию
xgrid()
```

```
//Вариант 1
//сетка чёрная, толщины 1, штрих.
//Для обеих осей внешний вид одинаковый
--> xgrid (1, 1, 2)
\\
//Вариант 2
// вертикальные линии сетки красные, а горизонтальные - чёрные;
//толщина 2; сплошная линия
--> xgrid ([5;1],2 ,1)

//Вариант 3
//вертикальные линии сетки синие, толщина 4, сплошная линия
//горизонтальные линии сетки чёрные, толщина 2, штрих с двумя точками
--> xgrid ([9;1],[4;2],[1;5])
```

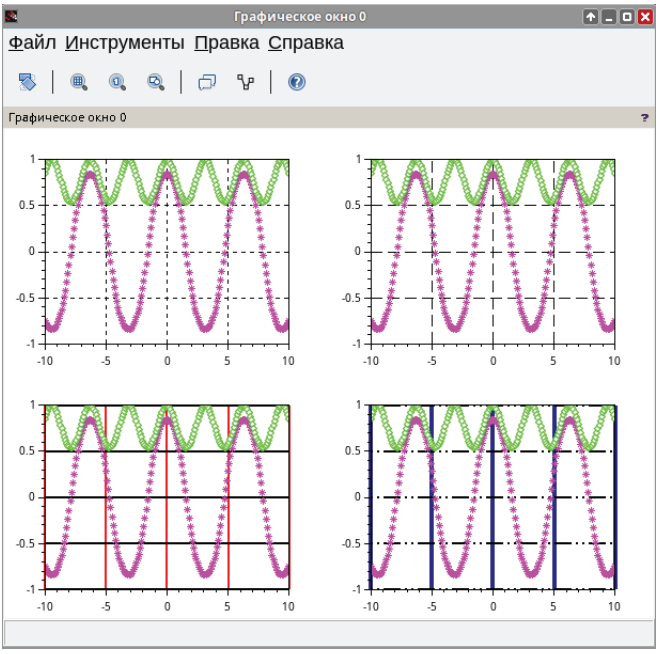

Рис. 4.13. Команда xgrid

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*.

Воспользуемся предыдущим примером и добавим к графику заголовок `'График y=cos(x)'` и подписи к координатным осям *X* и *Y* (см. листинг 4.10 и рис. 4.14).

Листинг 4.10. Вывод заголовка графика и подписей координатных осей

```
x=-10:0.1:10;
y=cos(x);
plot(x,y,'m*');
xgrid();
xtitle('График y=cos(x)','Ось X','Ось Y');
```

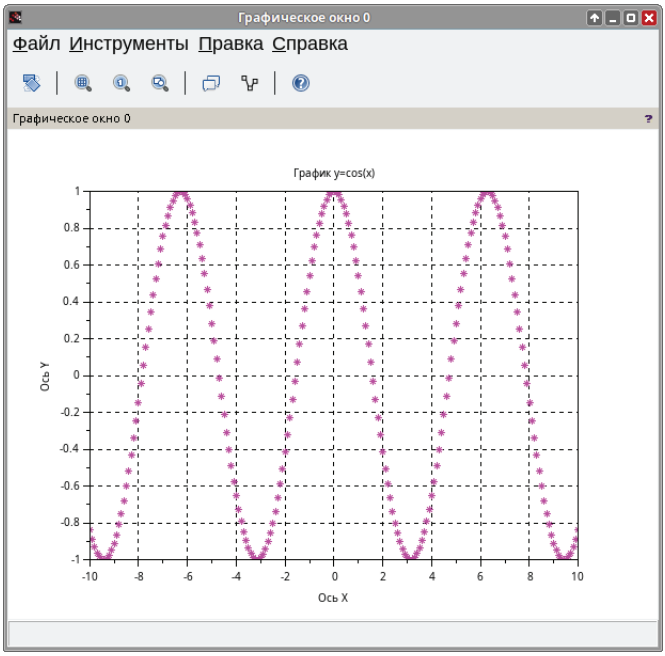

Рис. 4.14. Команда `xtitle`

В случаях когда в одной координатной плоскости изображаются графики нескольких функций, как в нашем примере, возникает необходимость в «легенде». Её 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; `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 вывода графика;

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

Выведем на графике одного из предыдущих примеров легенду и сами определим её местоположение (после вывода графика под курсором мыши будет расположена легенда. Чтобы прикрепить её к графику, нужно кликнуть левой кнопкой мыши в нужном месте). Также отменим построение рамки вокруг легенды (см. листинг 4.11 и рис. 4.15).

Листинг 4.11. Вывод легенды графика и определение её свойств

```
x=-10:0.1:10;
y=sin(cos(x));
z=cos(sin(x));
plot(x,y,'m*',x,z,'gd');
xgrid();
xtitle('График sin(cos(x)) и cos(sin(x))','X','Y');
legend('sin(cos(x))','cos(sin(x))',5,%f);
```

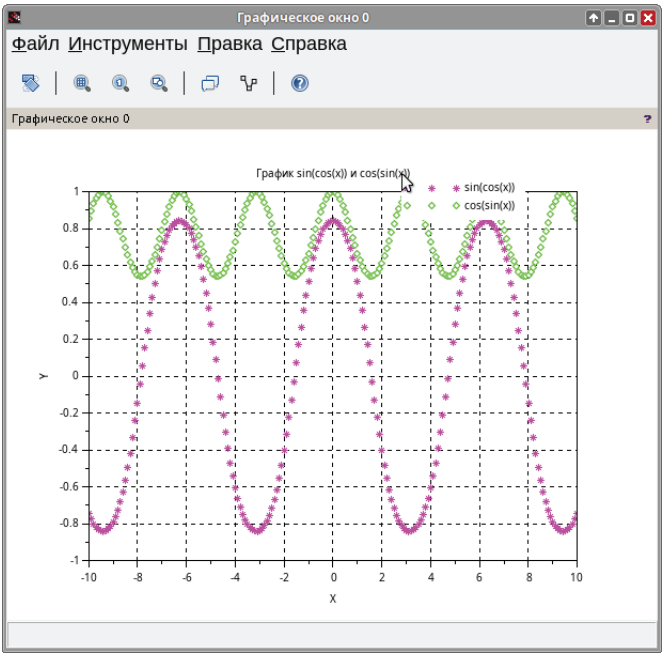

Рис. 4.15. Команда `legend`

4.6 Функция plot2d

Для построения графиков также можно использовать функцию `plot2d`. Обращение к функции имеет вид:

```
plot2d([logflag],x,y)
```

где `logflag` (необязательный параметр) – 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.

При построении графиков принцип работы и синтаксис `plot2d` ничем не отличаются от `plot`. Использовать можно любую из функций.

Задача 4.8

Построить график функции $y = \sin(x)$ на интервале $[-2\pi; 2\pi]$ с шагом 0,1.

В данной задаче использование функции `plot2d` аналогично функции `plot` (см. листинг 4.12 и рис. 4.16).

Листинг 4.12. Пример простейшего использования функции `plot2d`

```
x=[-2*%pi:%pi/10:2*%pi];
plot2d(x,sin(x));
```

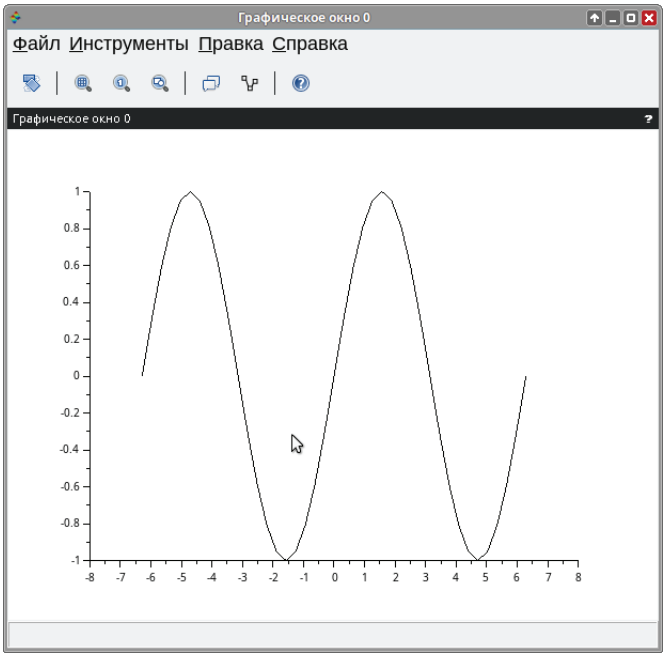

Рис. 4.16. График функции $y = \sin(x)$

Используя функцию `plot2d`, можно также построить несколько графиков в одной системе координат.

Задача 4.9

Построить графики функций $y = \sin(x)$, $y_1 = \sin(2x)$, $y_2 = \sin(3x)$ на интервале $[0; 2\pi]$ с шагом 0,1 в одной системе координат.

В качестве массива *Y* в квадратных скобках поочередно укажем математические выражения заданных функций, разделяя их пробелами (см. листинг 4.13 и рис. 4.17).

Листинг 4.13. Несколько графиков в одних координатных осях. Команда `plot2d`

```
x=[0:%pi/20:2*%pi]';//транспонирование вектора x
plot2d(x,[sin(x) sin(2*x) sin(3*x)]);
```

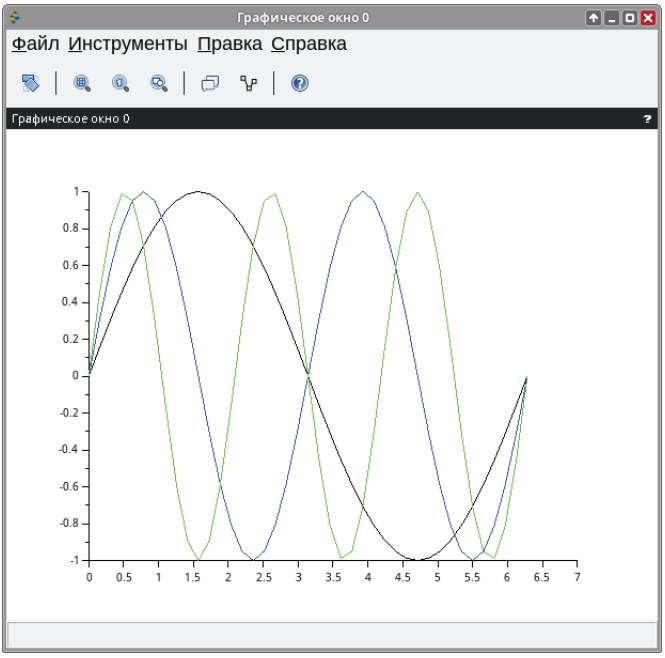

Рис. 4.17. Графики функций $y = \sin(x)$, $y = \sin(2x)$, $y = \sin(3x)$

4.7 Оформление графиков при помощи функции plot2d

В общем виде обращение к функции имеет вид:

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

где `logflag` (необязательный параметр) – 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, определяющих его внешний вид:

```
key1=value1, key2=value2,..., keyn=valuen
```

Полная форма обращения к функции `plot2d` дает возможность самостоятельно определять внешний вид графика – за это в ней отвечает параметр `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.
  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* зелёного цвета (0,176,0) (см. листинг 4.14).
  Полученные графики представлены на рис. 4.18.

Листинг 4.14. Изменение цвета линии графика

```
x=[-2*%pi:%pi/100:2*%pi];
y=[sin(x);cos(x)];
plot2d(x,y',style=[color("red"),color(0,176,0)]);
```

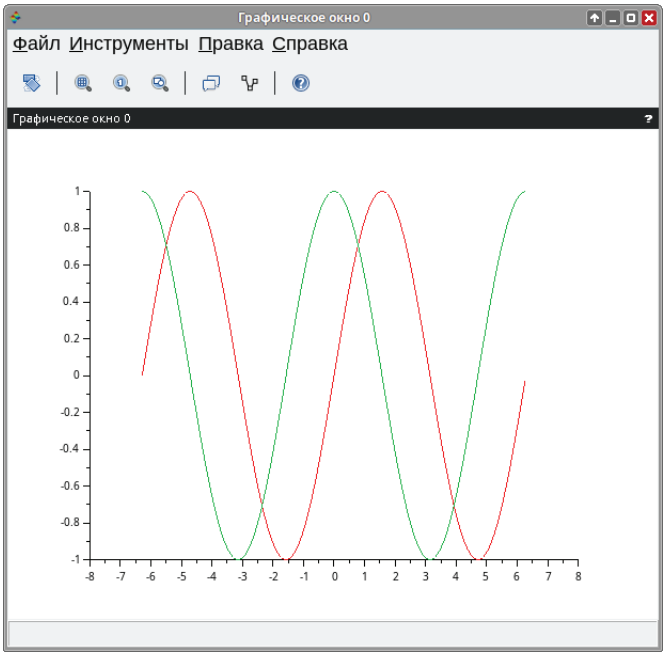

Рис. 4.18. Параметр `style` функции `plot2d`

- `rect` – значение параметра `keyn=valuen` функции `plot2d` – это вектор `[xmin, ymin, xmax, ymax]`, определяющий размер окна вокруг графика:
  – `xmin`, `xmax` – минимальное и максимальное значения по оси *X*;
  – `ymin`, `ymax` – минимальное и максимальное значения по оси *Y*.
  Применим параметр `rect` к предыдущему примеру, установив для него следующие значения [–8,–2,8,2] (см. листинг 4.15 и рис. 4.19).

Листинг 4.15. Определение размеров окна вокруг графика

```
x=[-2*%pi:%pi/100:2*%pi];
y=[sin(x);cos(x)];
subplot(1,2,1);

//Построение графика [-8;8] по X и [-2;2] по Y
plot2d(x,y',style=[color("red"),color(0,176,0)],rect=[-8,-2,8,2]);
subplot(1,2,2);

//Построение графика [-2;2] по X и [-2;2] по Y
plot2d(x,y',style=[color("red"),color(0,176,0)],rect=[-2,-2,2,2]);
```

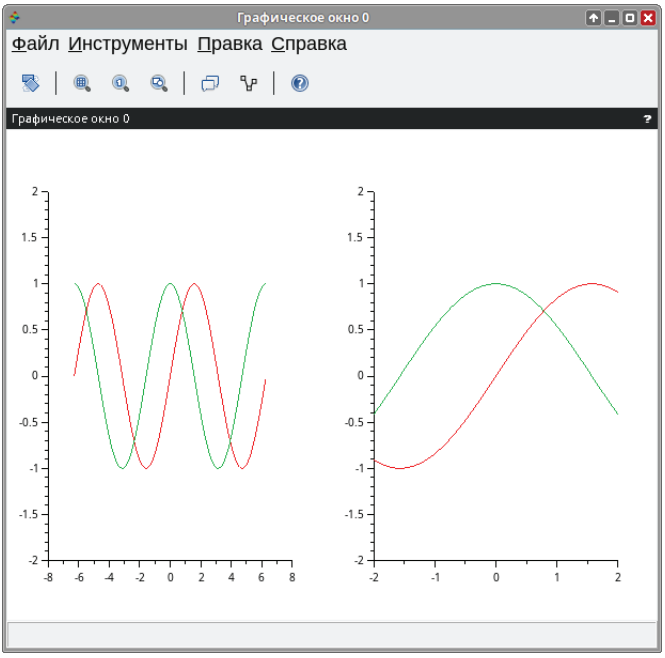

Рис. 4.19. Параметр rect функции `plot2d`

- `axesflag` – значение параметра `keyn=valuen` функции `plot2d` – определяет наличие рамки вокруг графика. Необходимо выделить следующие базисные значения этого параметра:
  0 – нет рамки; нет осей;
  1 – есть рамка; ось *y* слева (по умолчанию);

2 – есть рамка; нет осей;
3 – есть рамка; ось *y* справа;
4 – нет рамки; оси проходят через точку (0, 0);
5 – есть рамка; оси проходят через точку (0, 0).

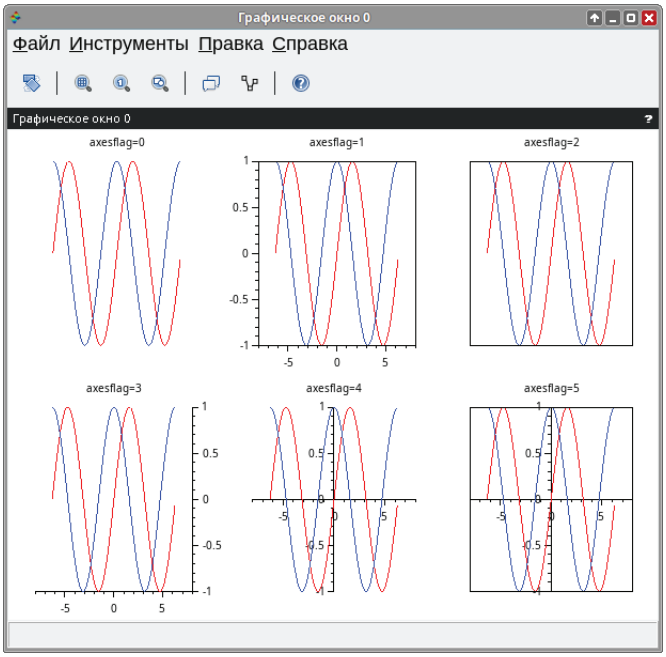

Рис. 4.20. Параметр axesflag функции `plot2d`

Задача 4.10

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

Листинг 4.16. Вывод рамки и определение размера координатных осей

```
x=[-2*%pi:%pi/50:2*%pi];
y=[sin(x); cos(x)];
subplot(2,3,1)
xtitle('axesflag=0')
plot2d(x,y',style=[color("red"), color("blue")], axesflag=0);
subplot(2,3,2)
xtitle('axesflag=1')
plot2d(x,y',style=[color("red"), color("blue")], axesflag=1);
subplot(2,3,3)
xtitle('axesflag=2')
plot2d(x,y',style=[color("red"), color("blue")], axesflag=2);
subplot(2,3,4)
```

```
xtitle('axesflag=3')
plot2d(x,y',style=[color("red"), color("blue")], axesflag=3);
subplot(2,3,5)
xtitle('axesflag=4')
plot2d(x,y',style=[color("red"), color("blue")], axesflag=4);
subplot(2,3,6)
xtitle('axesflag=5')
plot2d(x,y',style=[color("red"), color("blue")], axesflag=5);
```

- Для того чтобы определить число основных и промежуточных делений координатных осей, в Scilab существует параметр nax. Параметр `axesflag=1` (по умолчанию) – это массив из четырёх значений: `[nx, Nx, ny, Ny]`.
  Здесь `Nx (Ny)` – число основных делений с подписями под осью X (Y); `nx (ny)` – число промежуточных делений.

Задача 4.11

Построить графики функций sin(x) и 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 девятью основными делениями (засечками), каждое основное – четырьмя промежуточными, а ось Y – соответственно шестью и тремя (см. листинг 4.17 и рис. 4.21).

Листинг 4.17. Нанесение на график основных и промежуточных делений

```
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@\ldots@legn
  ```

  где `leg1` – легенда первого графика; `legn` – легенда n-го графика.

Построим графики функций sin(x) и 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. 4.22).

Листинг 4.18. Вывод легенды графика

```
x=[-2*%pi:%pi/100:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',axesflag=5, leg="sin(x)@cos(x)");
```

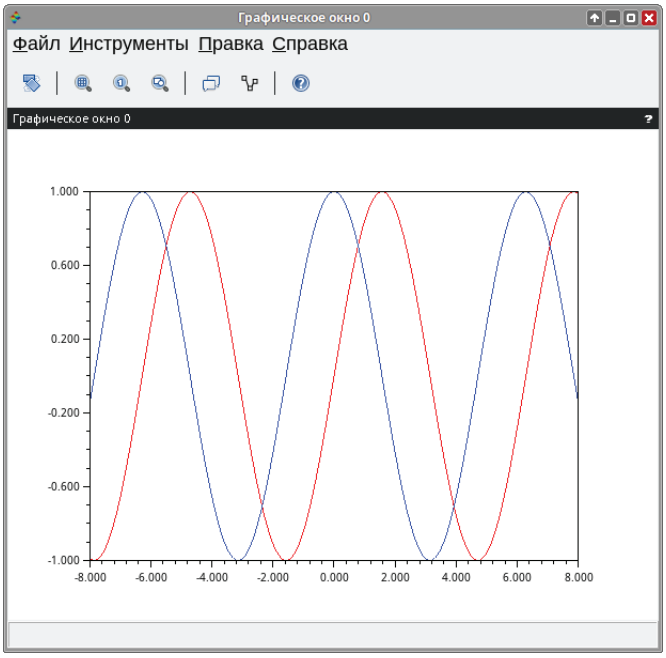

Рис. 4.21. Параметр `nax` функции `plot2d`

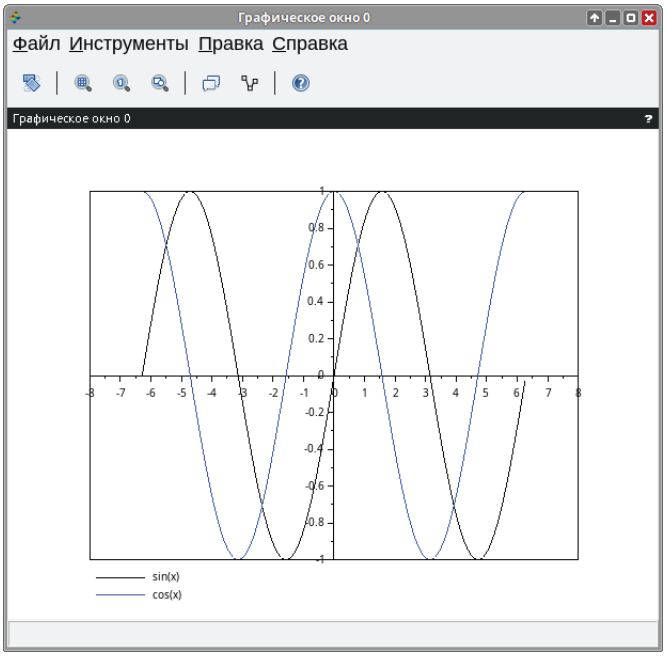

Рис. 4.22. Параметр `leg` функции `plot2d`

4.7.1 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).

Задача 4.12

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

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	Звёздочка
–11	Квадрат
–12	Треугольник вершиной вправо
–13	Треугольник вершиной влево
–14	Пятиконечная звезда

При построении кривой с помощью функции `plot2d` укажем аргумент –8, определяющий тип маркера «плюс, вписанный в ромб» (см. листинг 4.19 и рис. 4.23).

Листинг 4.19. Построение точечных графиков

```
x=[-2*%pi:%pi/50:2*%pi];
y=sin(x);
plot2d(x,y,-8);
```

4.7.2 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 4.13

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.

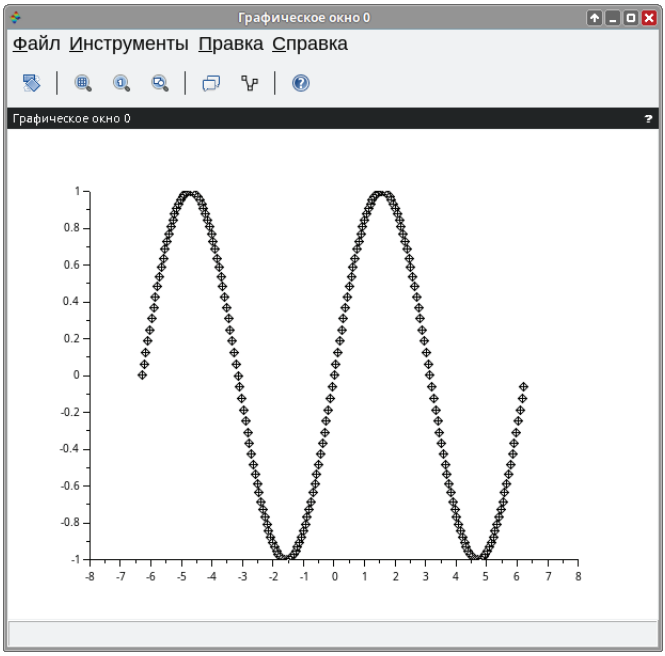

Рис. 4.23. Точечный график функции $y = \sin(x)$

Поэлементно введём массивы X и Y и воспользуемся функцией `plot2d2(x,y)` (см. листинг 4.20 и рис. 4.24).

Листинг 4.20. Построение графиков в виде ступенчатой линии

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

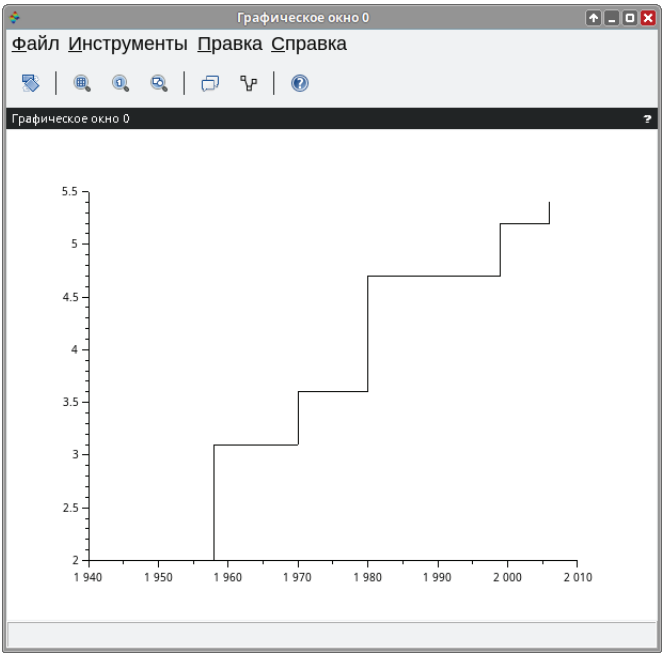

Рис. 4.24. Построение ступенчатого графика при помощи `plot2d2`

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

где `fi` – полярный угол; `ro` – полярный радиус; `keyn=valuen` – последовательность значений свойств графика.

Задача 4.14

Построить полярные графики функций $3\cos 5\varphi$ и $3\cos 3\varphi$.

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. 4.25).

Листинг 4.21. График функций в полярной системе координат

```
fi=0:%pi/100:2*%pi;
ro=3*cos(5*fi);ro1=3*cos(3*fi);
polarplot(fi,ro,style=color("red"));
polarplot(fi,ro1,style=color("blue"));
```

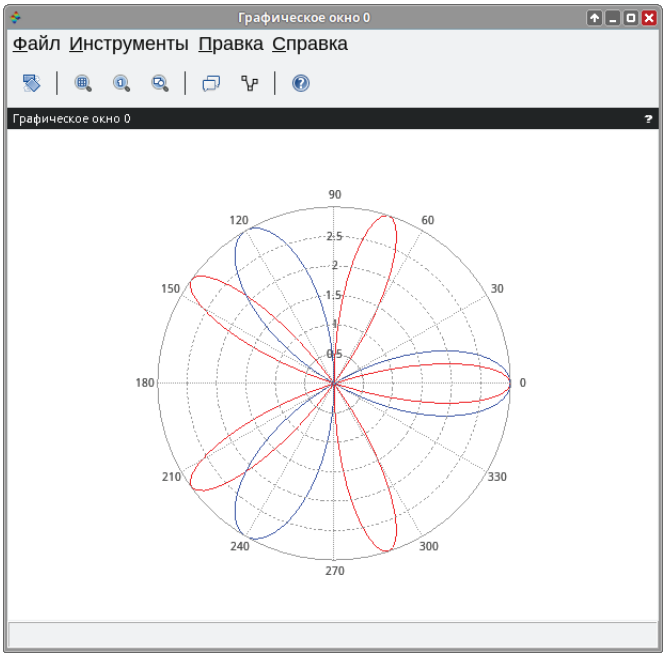

Рис. 4.25. Полярные графики функций $\rho = 3\cos(5\varphi)$, $\rho_1 = 3\cos(3\varphi)$

Задача 4.15

Построить графики архимедовой спирали, гиперболической спирали и логарифмической спирали в полярных координатах.

Уравнение архимедовой спирали в полярных координатах имеет вид: $\rho = a\varphi$, гиперболической – $\rho = \frac{a}{\varphi}$. Соотношение $\rho = ae^{k\varphi}$, $k = \operatorname{ctg} a$ является уравнением логарифмической спирали в полярных координатах. Частным случаем логарифмической спирали ($a = \frac{\pi}{2}$, $k = 0$) является уравнение окружности ($\rho = a$).

В листинге 4.22 приведён текст программы, позволяющей построить в одном графическом окне четыре оси координат, в каждом из которых построить свой график – архимедову, гиперболическую и логарифмическую спирали, а также окружность.

График представлен на рис. 4.26.

Листинг 4.22. Построение графиков спиралей (пример 4.15)

```
fi1=0:%pi/15:6*%pi;
fi2=%pi/3:%pi/15:6*%pi;
fi3=0:%pi/15:4*%pi;
fi4=-%pi:%pi/15:%pi;
ro1=4*fi1;
ro2=0.5./fi2;
ro3=4*exp(0.2*fi3);
for i = 1:length(fi4)
        ro4(i)=4;
end

subplot(2,2,1);
polarplot(fi1,ro1,-3);
title('график архимедовой спирали');
subplot(2,2,2);
polarplot(fi2,ro2,-4);
title('график гиперболической спирали');
subplot(2,2,3);
polarplot(fi3,ro3,-10);
title('график логарифмической спирали ');
subplot(2,2,4);
polarplot(fi4,ro4,-14); title('график окружности');
```

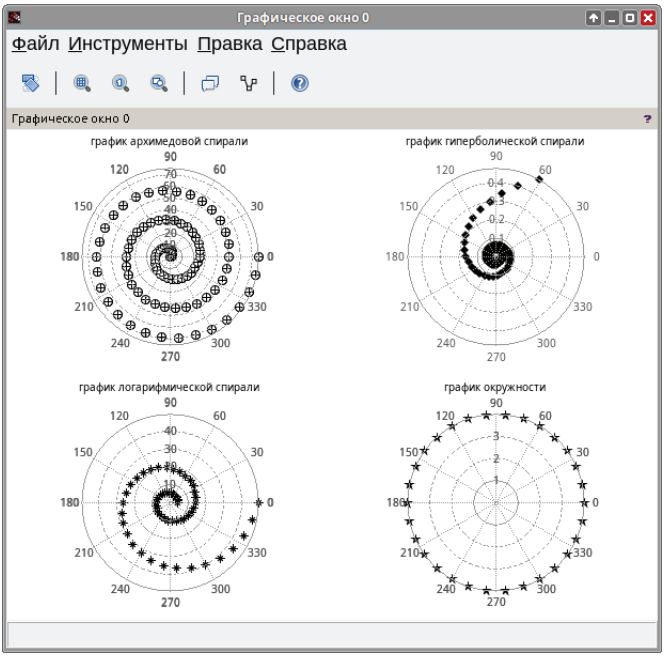

Рис. 4.26. Графики архимедовой, гиперболической и логарифмической спиралей, окружности в полярных координатах

4.9 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*.

Построение графика функции, заданной параметрически, можно осуществлять следующим образом:

1) определить массив t;
2) определить массивы $x = f(t)$ и $y = g(t)$;
3) построить график функции $y(x)$ с помощью `plot(x,y)` либо `plot2d(x,y)`.

Задача 4.16
Построить график строфоиды.

Напомним, что строфоида представляет собой алгебраическую кривую третьего порядка, которая в общем виде задаётся 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 4.23, рис. 4.27).

Листинг 4.23. Построение строфоиды с помощью функции `plot`

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

Задача 4.17
Построить график полукубической параболы.

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

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

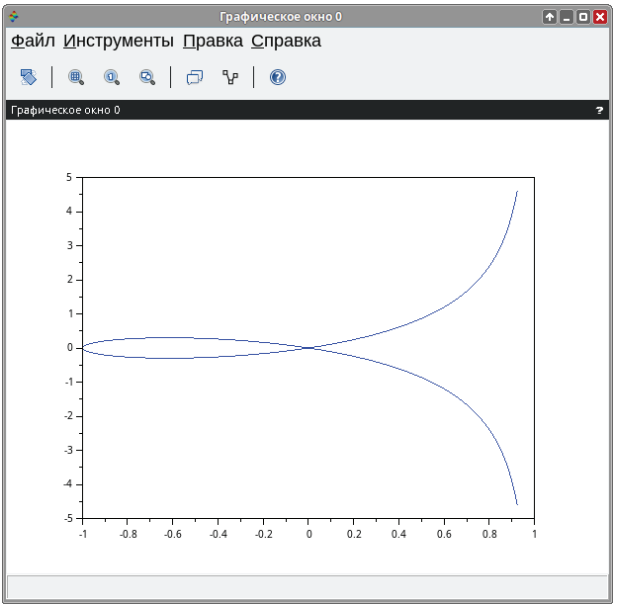

Рис. 4.27. График строфоиды

Приведём это уравнение к параметрической форме:

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

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 4.24, рис. 4.28).

Листинг 4.24. Построение полукубической параболы

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

Задача 4.18

Построить график эпициклоиды. Уравнение эпициклоиды в параметрической форме имеет вид: $x = 4\cos t - \cos 4t$, $y = 4\sin t - \sin 4t$, $t \in [0; 2\pi]$.

В листинге 4.25 представлен текст программы для изображения графика эпициклоиды, а на рис. 4.29 – сам график.

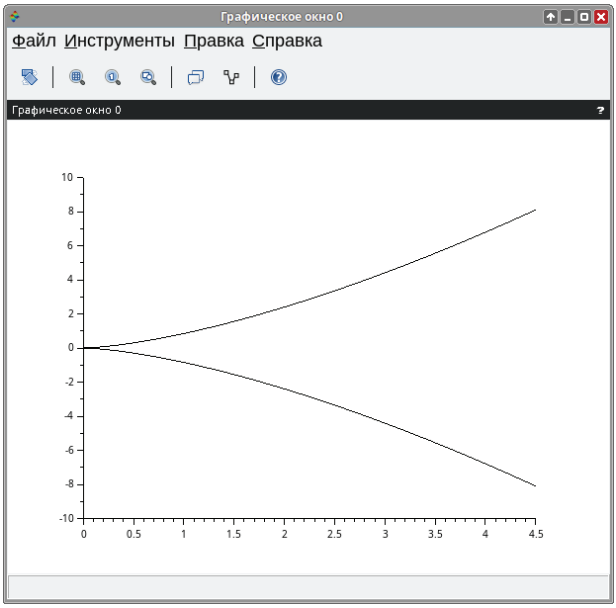

Рис. 4.28. График полукубической параболы

Листинг 4.25. Построение графика эпициклоиды (пример 4.18)

```
t=0:%pi/50:2*%pi;x=4*cos(t)-cos(4*t);y=4*sin(t)-sin(4*t);
plot(x,y);
xgrid();
```

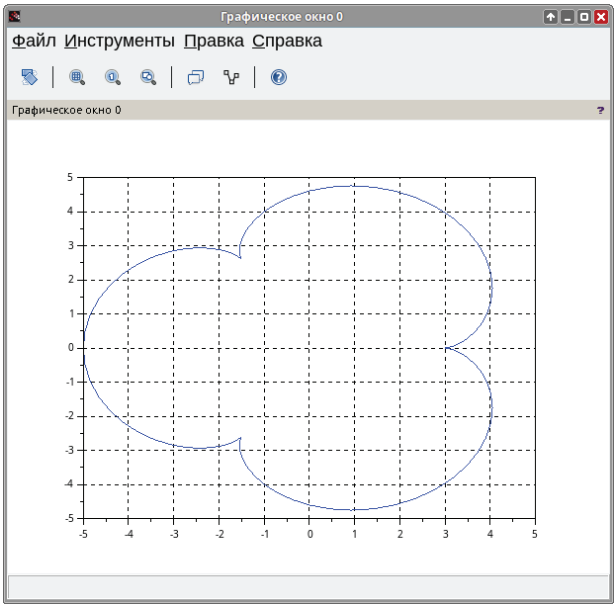

Рис. 4.29. График эпициклоиды

Задача 4.19
Построить график астроиды.

Уравнение астроиды в параметрической форме имеет вид $x = 3\cos^3 t$, $y = 3\sin^3 t$, $t \in [0; 2\pi]$. В листинге 4.26 представлен текст программы для изображения графика астроиды, а на рис. 4.30 – сам график.

Листинг 4.26. Построение графика астроиды (пример 4.19)

```
t=0:%pi/50:2*%pi;
x=3*cos(t).^3;
y=3*sin(t).^3;
plot(x,y);
xgrid();
```

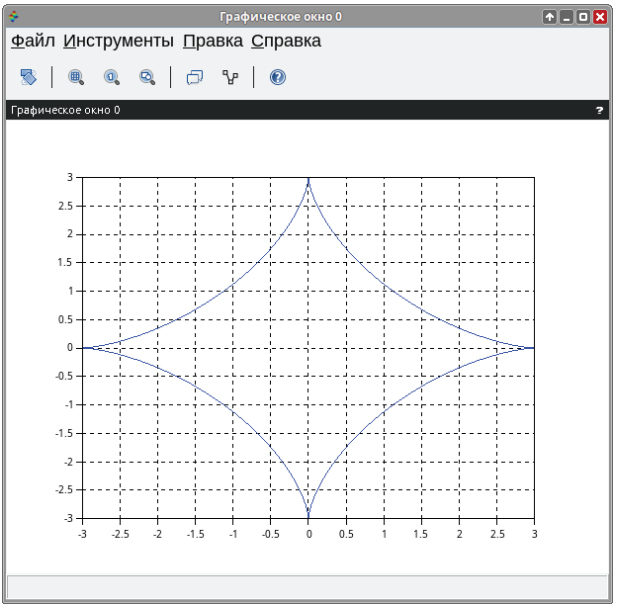

Рис. 4.30. График астроиды

Функция `bar` предназначена для построения гистограммы.

Функция bar(*y*) выводит элементы массива *y* в виде гистограммы, в качестве массива *x* выступает массив номеров элементов массива *y*. Функция bar(*x*, *y*) выводит гистограмму элементов массива *y* в виде столбцов в позициях, определяемых массивом *x*. Элементы должны быть упорядочены в порядке возрастания.

Рассмотрим несколько примеров построения гистограмм.

```
y=[5; 6; 7; 8; 9; 8; 7; 6; 4; 3];
bar(y);
```

строит гистограмму, представленную на рис. 4.31.

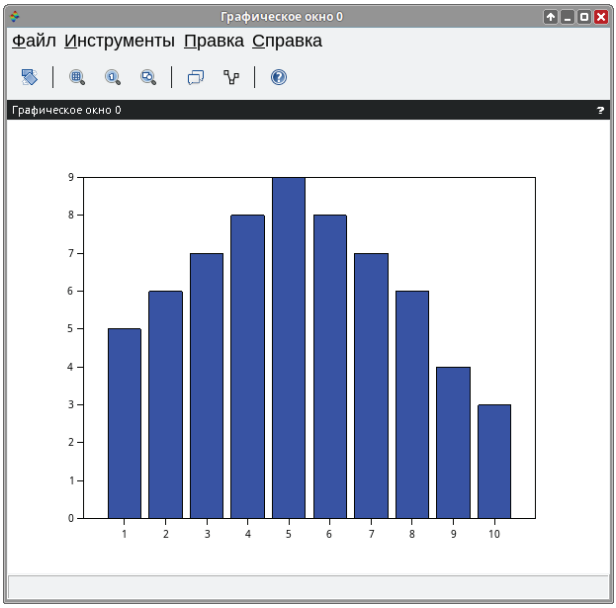

Рис. 4.31. Пример использования функции bar(*y*)

```
x1=[-2,-1,0,1,2,3,4];
y1=exp(sin(x1));
bar(x1,y1);
```

строит гистограмму, представленную на рис. 4.32.

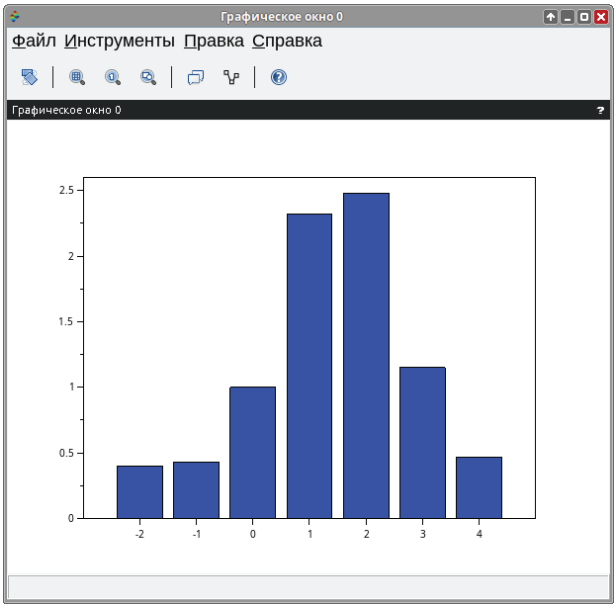

Рис. 4.32. Пример использования функции bar(*x*, *y*)

4.10 Примеры решения некоторых задач

Задача 4.20
Построить график функции $y = \sin(\cos(x))$ на интервале -2π до 2π.

Решение задачи представлено в листинге 4.27 и на рис. 4.33.

Листинг 4.27. Построение графика функции $y = \sin(\cos(x))$

```
x=-2*%pi : %pi/100 : 2*%pi;
y=sin(cos(x));
plot(x,y);
```

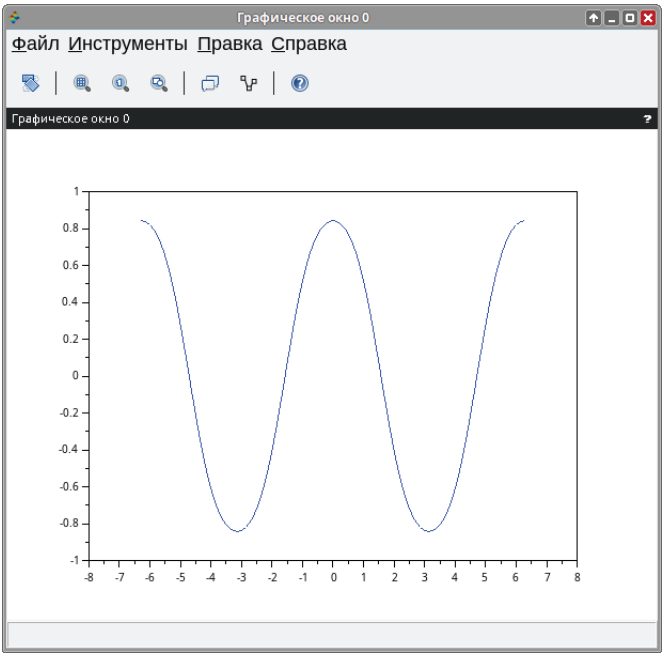

Рис. 4.33. График функции $y = \sin(\cos(x))$

Задача 4.21
Изобразить график функции $y = x^2/2$ на интервале [0.6;2.5] и точечный график функции, заданный таблично.

x	1	1.3	1.5	1.4	1.6	2	2.1	2.3
y	1	1.5	1.7	1.8	1.75	1.6	1.53	1.4

Листинг 4.28. Построение графика нескольких функций

```
x1=0.6:0.05:2.5;
y1=x1.*x1/2;
x=[1 1.3 1.5 1.4 1.6 2 2.1 2.3]
y=[1 1.5 1.7 1.8 1.75 1.6 1.53 1.4]
plot(x,y,'ob',x1,y1,'r');
```

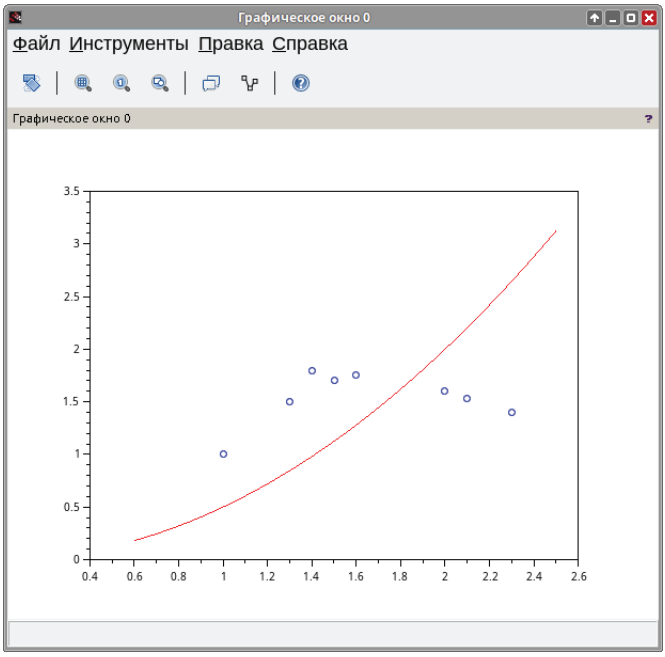

Рис. 4.34. График нескольких функций

Задача 4.22

Построить график разрывной функции $y = \frac{4x^2+5}{4x+8}$.

Область определения функции:

$$4x + 8 \neq 0 \Rightarrow 4x \neq -8 \Rightarrow x \neq -2.$$

Попробуем построить график в Scilab, не обращая внимания на имеющуюся точку разрыва (листинг 4.29, рис. 4.35).

Листинг 4.29. Построение графика разрывной функции

```
x=-5:0.1:5;
y=(4*x.^2+5)./(4*x+8);
plot(x,y);
```

Далеко не каждый график можно построить, не обращая внимания на разрывы. Рассмотрим ещё два примера.

Задача 4.23

Построить график функции $y = \frac{4x^3-3x}{4x^4-1}$.

Функция имеет разрыв в точках $x = \pm\frac{1}{\sqrt{2}}$.

Листинг 4.30. Построение графика функции $y = \frac{4x^3 - 3x}{4x^4 - 1}$

```
x=-2:0.1:2;
y=(4*x.^3-3*x)./(4*x.^4-1);
plot(x,y);
```

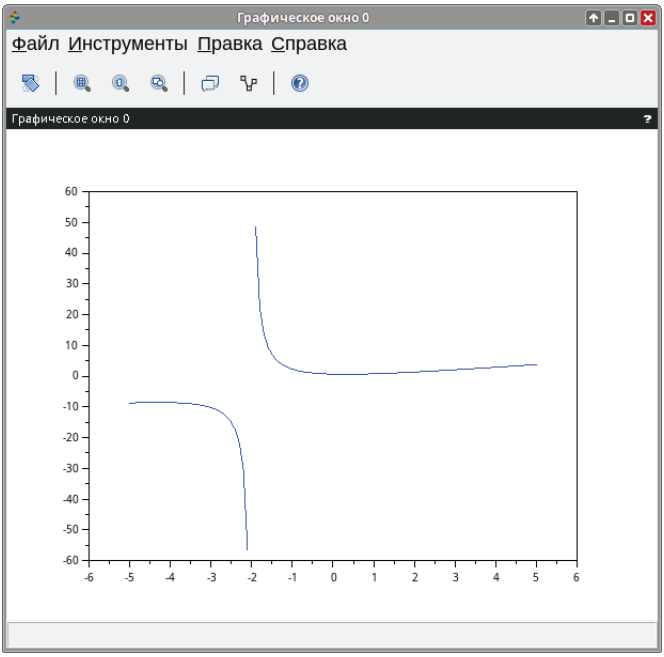

Рис. 4.35. График разрывной функции

Полученный график представлен на рис. 4.36.

А теперь попробуем «обойти» точки разрыва (рис. 4.37).

```
clear x y
deff('y=F(x)','y=(4*x.^3-3*x)./(4*x.^4-1)')
h=0.05
x1=-2:0.02:-1/sqrt(2)-h;
x2=-1/sqrt(2)+h:0.02:1/sqrt(2)-h;
x3=1/sqrt(2)+h:0.02:2;
plot(x1,F(x1),'r',x2,F(x2),'r',x3,F(x3),'r');
```

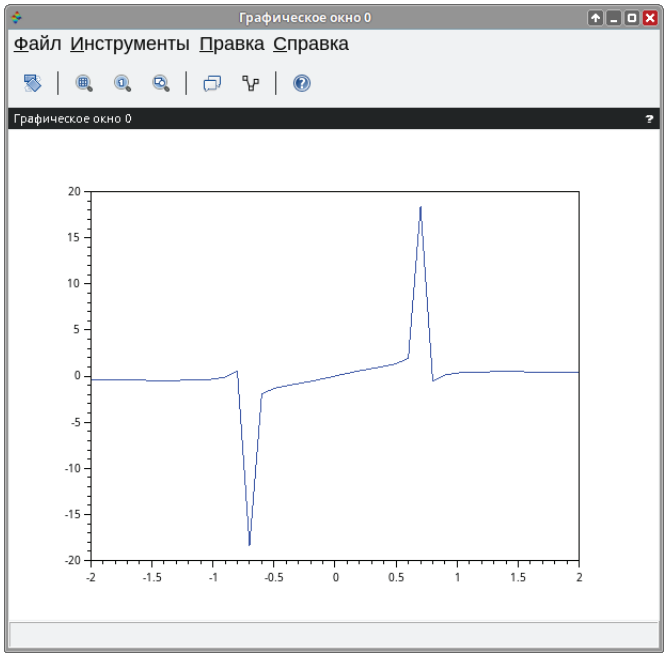

Рис. 4.36. Неверное построение графика функции с точками разрыва

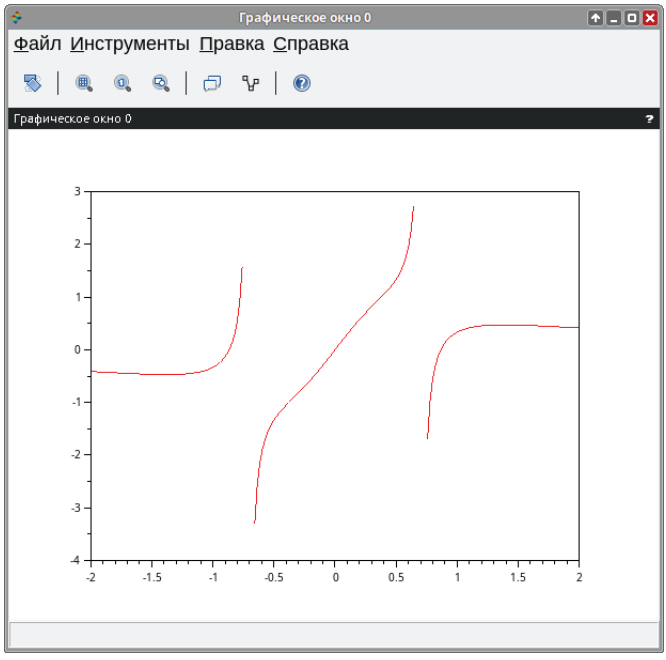

Рис. 4.37. Верное построение графика функции с точками разрыва

Задача 4.24

Построить график функции $f(x) = \frac{7x^2-3}{\sqrt{x^2-1}}$.

Область определения: $x^2 - 1 > 0 \Rightarrow x^2 > 1 \Rightarrow x \in (-\infty; -1) \cup (1; +\infty)$.

Листинг 4.31. Построение графика функции $f(x) = \frac{7x^2 - 3}{\sqrt{x^2 - 1}}$

```
clf
deff('y=F(x)','y=(7*x.^2-3)./(x.^2-1).^0.5')
x1=-3:0.01:-1.01;
x2=1.01:0.01:3;
plot(x1,F(x1),'b',x2,F(x2),'b');
xgrid();
```

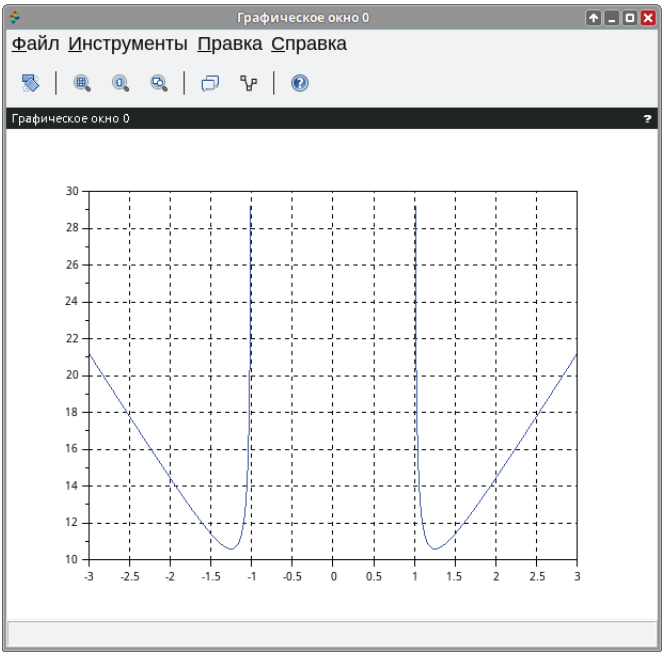

Рис. 4.38. График функции $f(x) = \frac{7x^2 - 3}{\sqrt{x^2 - 1}}$

Задача 4.25

Изобразить линию, заданную неявно, уравнением: $4y^2 + 5x^2 - 20 = 0$.

Разрешим заданное уравнение относительно переменной y. График $f(x, y)$ представлен линиями $f_1(x) = \frac{\sqrt{20-5x^2}}{2}$ и $f_2(x) = -\frac{\sqrt{20-5x^2}}{2}$. Область определения: $20 - 5x^2 \geq 0 \Rightarrow x^2 \leq 4 \Rightarrow x \in [-2; 2]$.

Листинг 4.32. Решение задачи 4.25

```
clear;clf;
deff('y=F(x)','y=sqrt(20-5*x^2)/2')
x=-2:0.01:2;
plot(x,F(x),'r',x,-F(x),'r')
xgrid();
```

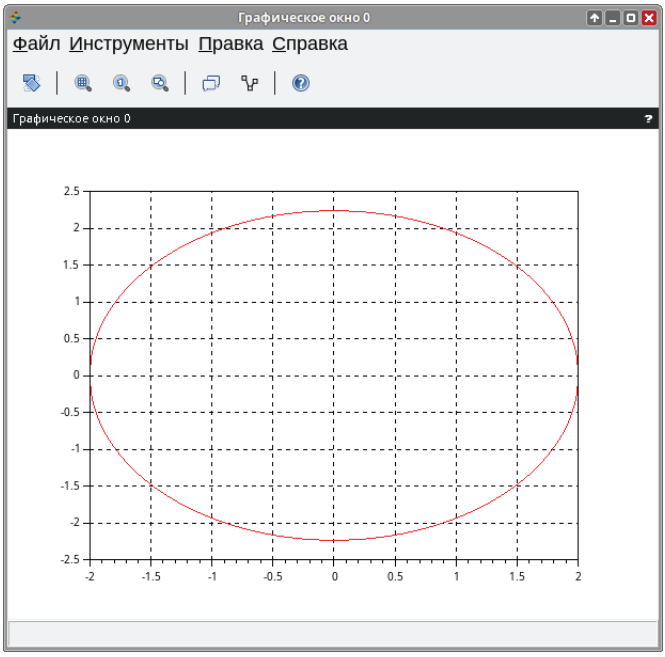

Рис. 4.39. График функции $4y^2 + 5x^2 - 20 = 0$

Задача 4.26

Изобразить линию, заданную неявно: $\frac{x^2}{4} - \frac{y^2}{9} = 1$.

Решим уравнение относительно переменной y: $f_1(x) = \frac{3}{2}\sqrt{x^2 - 4}$ и $f_2(x) = -\frac{3}{2}\sqrt{x^2 - 4}$. Область определения: $x^2 - 4 \geq 0 \Rightarrow x \in (-\infty, -2] \cup [2, +\infty)$.

Листинг 4.33. Решение задачи 4.26

```
deff('y=f(x)' ,'y=(3/2)*(x^2-4)^0.5');
x1=-6:0.01:-2; x2=2:0.01:6;
plot(x1,f(x1),'k',x1,-f(x1),'k',x2,f(x2),'k',x2,-f(x2),'k')
xgrid();
xtitle('График функции','Ось X','Ось Y')
```

График функции представлен на рис. 4.40.

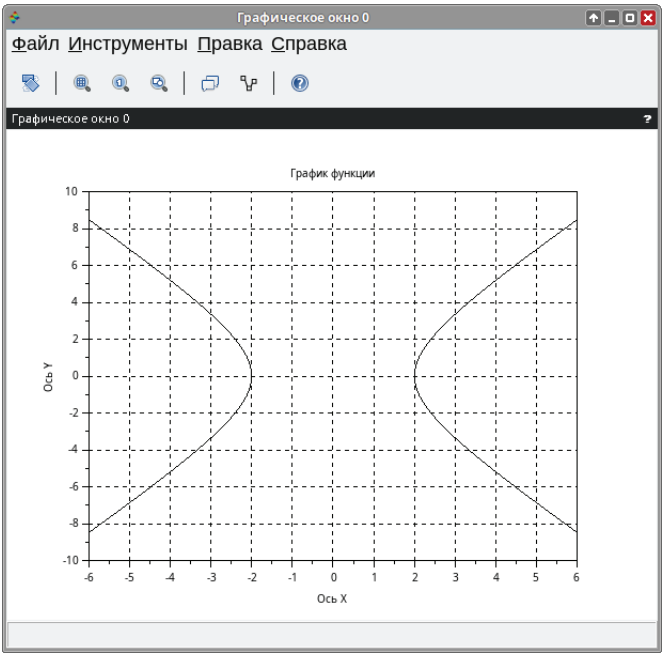

Рис. 4.40. График функции $\frac{x^2}{4} - \frac{y^2}{9} = 1$

Задача 4.27

Изобразить графики функций $\sin(x)$, $\cos(x)$, $\sin(\cos(x))$, $\cos(\sin(x))$.

Листинг 4.34. Построение графиков функций $\sin(x)$, $\cos(x)$, $\sin(\cos(x))$, $\cos(\sin(x))$

```
x=-10:0.01:10;
y=sin(x);
z=cos(x);
u=cos(sin(x));
v=sin(cos(x));
subplot(2,2,1);
plot(x,y);
xgrid();
xtitle('График функции SIN(X)','Ось X','Ось Y');
subplot(2,2,2);
plot(x,z);
xgrid();
xtitle('График функции COS(X)','Ось X','Ось Y');
subplot(2,2,3);
plot(x,u);
xgrid();
```

```
xtitle('График функции COS(SIN(X))','Ось X','Ось Y');
subplot(2,2,4);
plot(x,v);
xgrid();
xtitle('График функции SIN(COS(X))','Ось X','Ось Y');
```

Графики представлены на рис. 4.41.

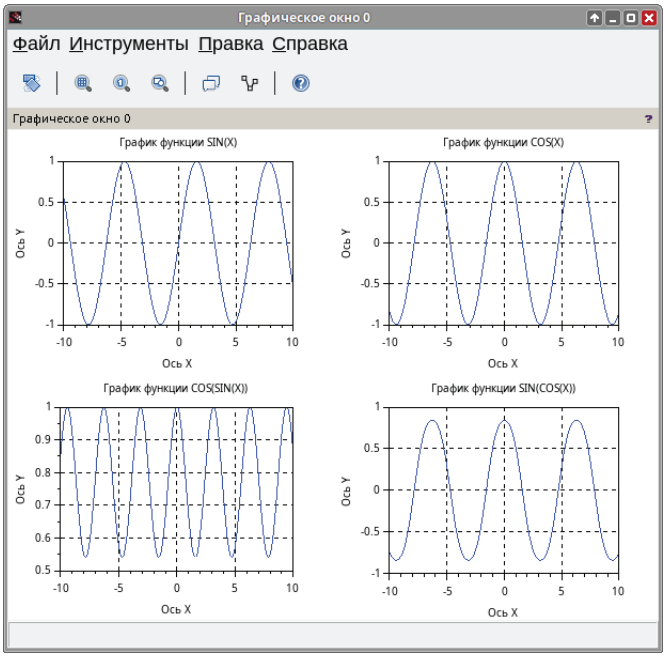

Рис. 4.41. Графики функций sin(*x*), cos(*x*), sin(cos(*x*)), cos(sin(*x*))

Задача 4.28

Построить графики нескольких функций в одной системе координат. Вывести сетку, заголовок графика и осей координат, легенду.

Листинг 4.35. Построение нескольких графиков

```
x=-2*%pi:%pi/100:2*%pi;
plot(x,sin(cos(x)),'r',x,cos(sin(x)),'b',x,exp(sin(x)),...
'm:',x,exp(cos(x)),'k');
xgrid();
xtitle('Графики нескольких функций с легендами','Ось X','Ось Y');
legend('sin(cos(x))','cos(sin(x))','exp(sin(x))','exp(cos(x))');
```

Графики представлены на рис. 4.42.

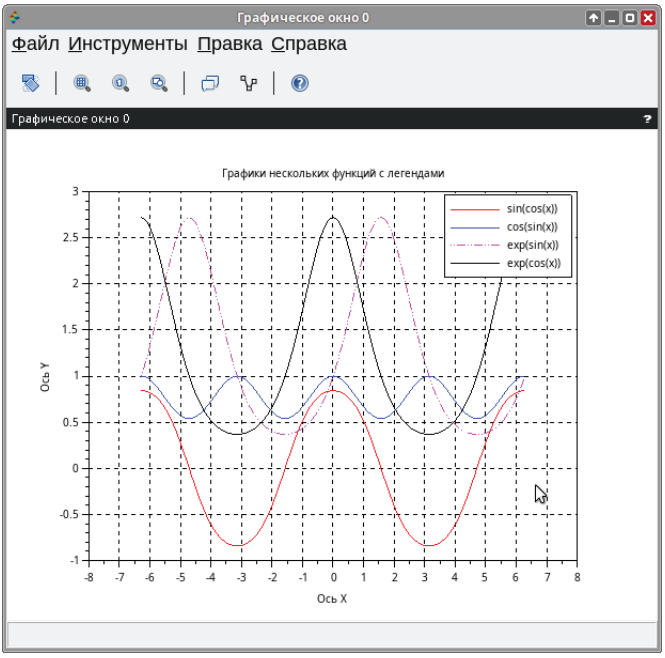

Рис. 4.42. Несколько графиков в одной системе координат

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 **Правка** ⇒ **Свойства графического окна** в 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 4.36, рис. 4.43).

Листинг 4.36. Построение графиков функций $y_1 = \sin(2x)$ и $y_2 = \sin(3x)$

```
x=[0:%pi/50:2*%pi]';
plot2d(x,[sin(2*x) sin(3*x)]);
```

Командой меню графического окна **Правка** ⇒ **Свойства графического окна** вызываем окно форматирования полученного графика **Figure Editor** (см. рис. 4.44).

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.

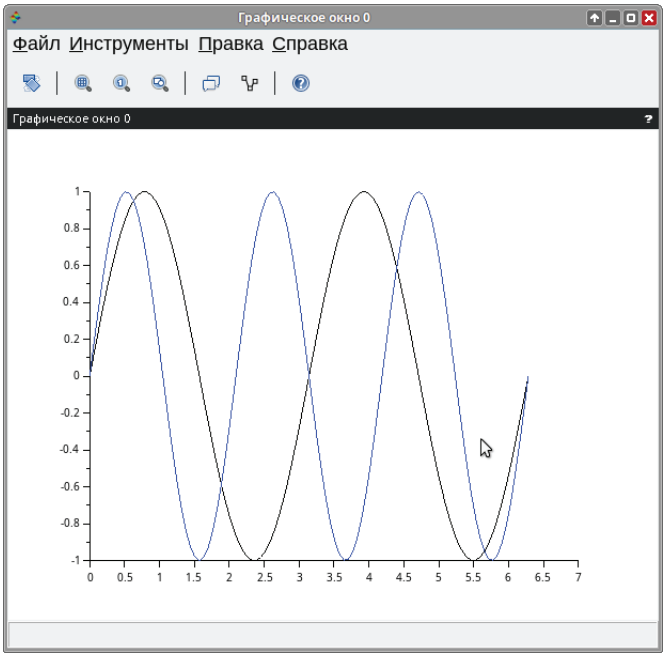

Рис. 4.43. Графики функций $y_1 = \sin(2x)$ и $y_2 = \sin(3x)$

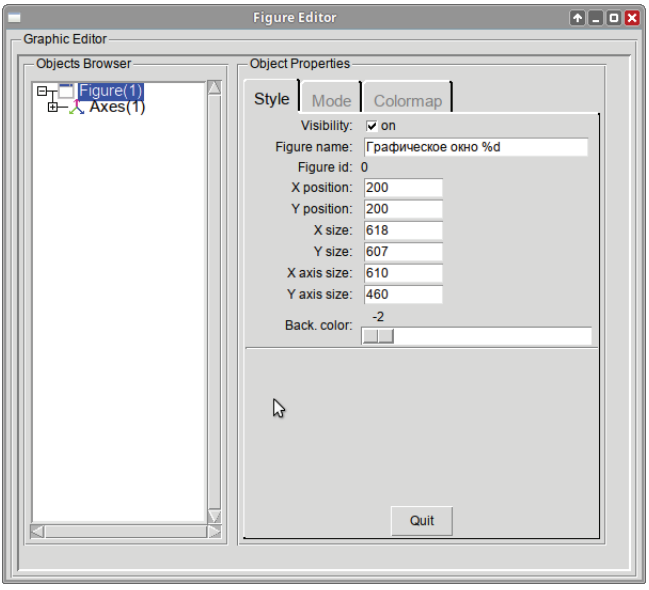

Рис. 4.44. Окно форматирования графика **Figure Editor**

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.

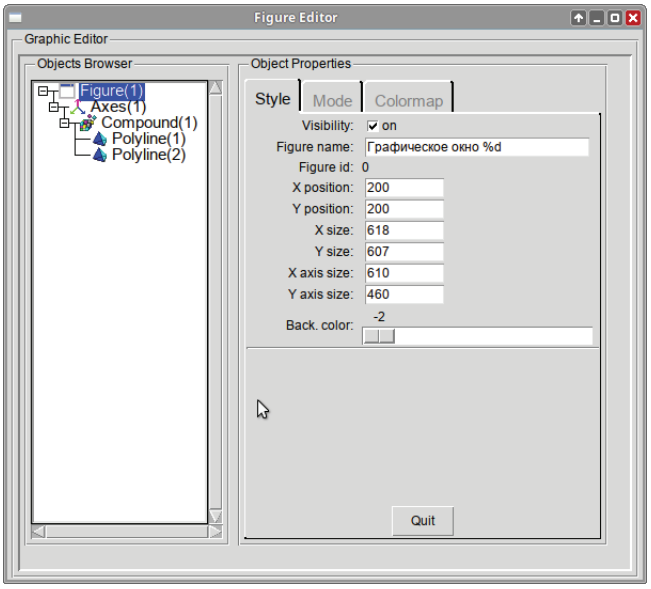

Рис. 4.45. Иерархия объектов в поле **Object Browser**

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. 4.45).

4.11.1 Форматирование объекта Figure

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. 4.45 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 **Графическое окно (%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 **График y=f(x)** и нажать клавишу **Enter**. Заголовок окна будет изменён (см. рис. 4.46);

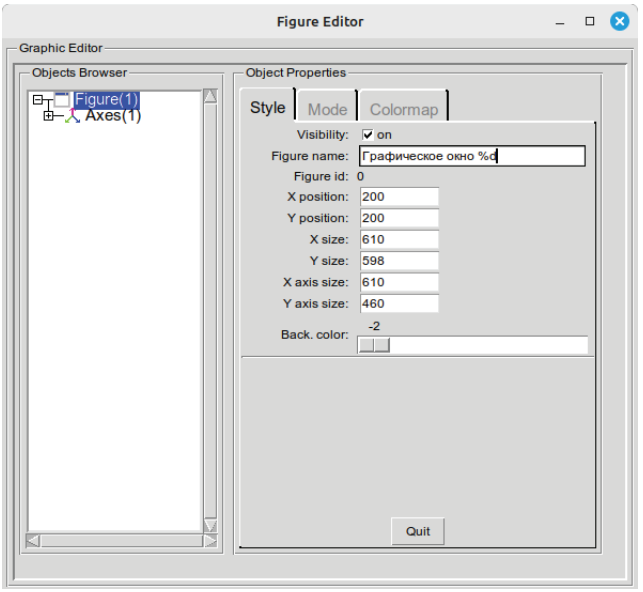

Рис. 4.46. Изменение заголовка графического окна в окне форматирования

- **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 – жёлто-горячий).

Установим ползунок в положение 15. Цвет фона станет зелёным (см. рис. 4.47).

Цветовая палитра может быть изменена пользователем на закладке **Colormap** (см. рис. 4.48).

Устанавливая значения для красного (*RED*), зелёного (*GREEN*) и синего (*BLUE*) цветов, можно изменять каждый оттенок независимо, не вызывая изменений других цветов в палитре. Например, вектор [0 0 0] задаёт чёрный 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.».

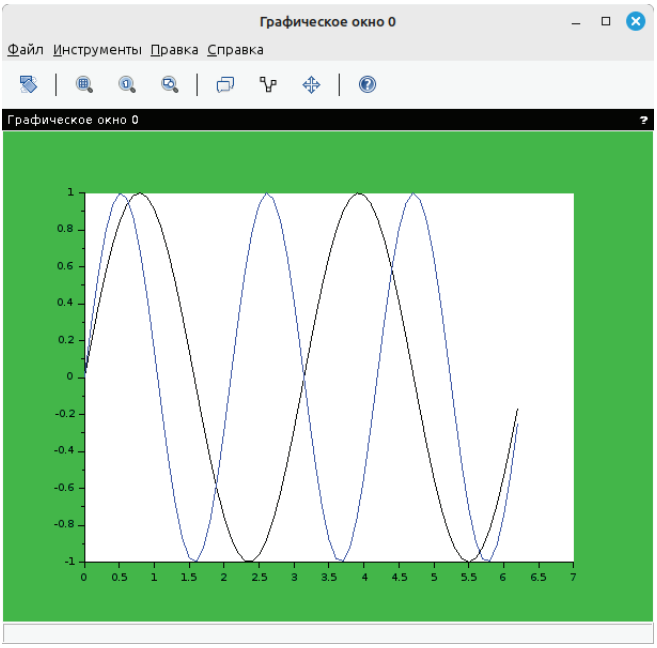

Рис. 4.47. Изменение фона графика в окне форматирования **Figure Editor**

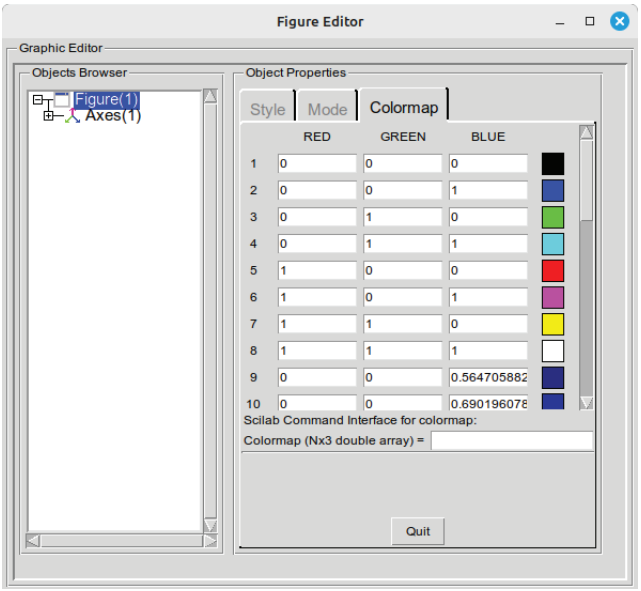

Рис. 4.48. Изменение палитры цветов на закладке **Colormap** окна форматирования **Figure Editor**

На закладке **Mode** в области **Object Properties** для объекта **Figure** (см. рис. 4.49) можно установить следующие свойства:

- **Auto resize** – свойство, которое позволяет изменять размер графика. Когда этот режим включён (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. 4.50);

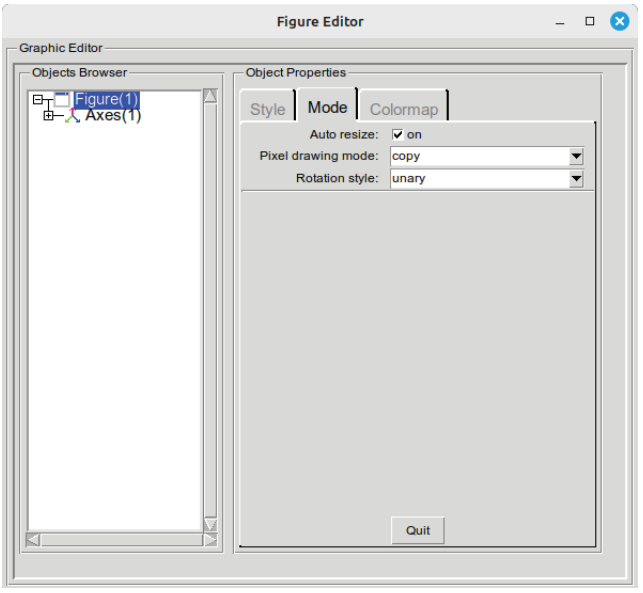

Рис. 4.49. Закладка **Mode** окна форматирования **Figure Editor**

- **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. 4.51.
- **Rotation style** – это свойство применимо лишь к трёхмерным графикам. Режим по умолчанию «*unary*» предназначен для вращения выделенных графиков, при включённом режиме «*multiple*» вращаются все трёхмерные графики.

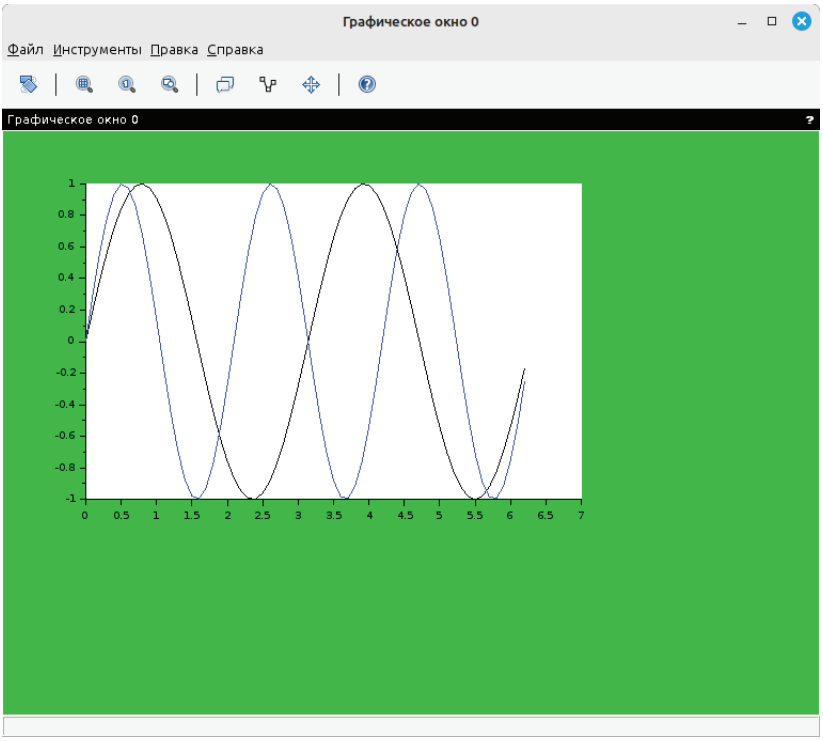

Рис. 4.50. Использование режима **Auto resize**

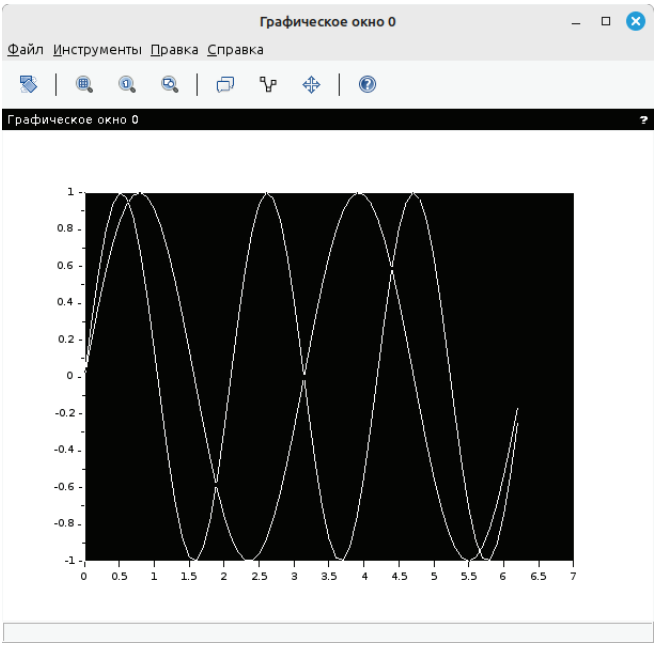

Рис. 4.51. Режим «*invert*» свойства **Pixel drawing mode** на вкладке **Mode**

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. 4.52).

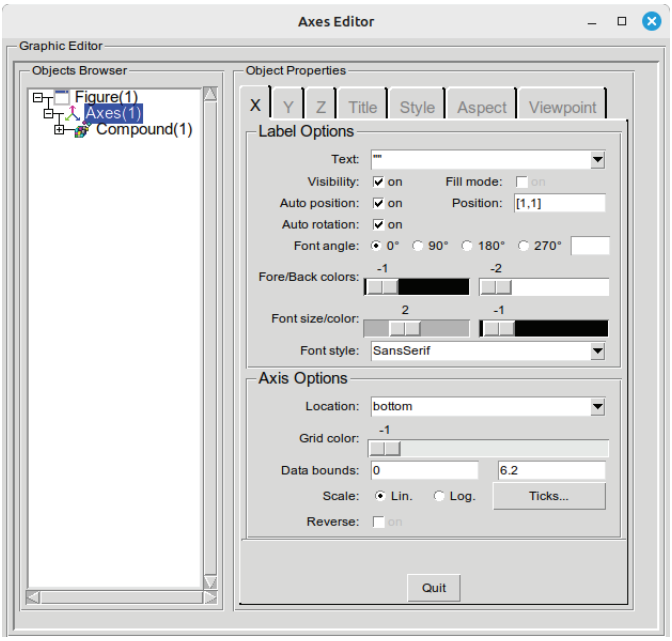

Рис. 4.52. Закладка *X* окна форматирования свойств объекта **Axes**

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:

- **Text** – 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 отключён (состояние переключателя «off»);
- **Font angle** – угол поворота подписи оси. Можно установить одно из предлагаемых значений: 0°, 90°, 180° и 270°, а также любой произвольный угол поворота надписи в последнем поле (см. рис. 4.52);
- **Fore/Back colors** – цвет символов и цвет фона подписи оси соответственно, устанавливаются при помощи ползунка, каждому положению которого отвечает определённый цвет. Всего доступно 35 цветов;
- **Font size/color** – размер символов подписи оси (возможны значения от 0 до 6) и их цвет (всего доступно 35 цветов). По умолчанию для шрифта установлен размер 1;
- **Font style** – стиль начертания символов подписи оси. По умолчанию установлен стиль **Serif**;

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 чёрные линии, кроме того, доступны ещё 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. 4.53.

  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 включён (значение переключателя «on»). Однако существует возможность само-

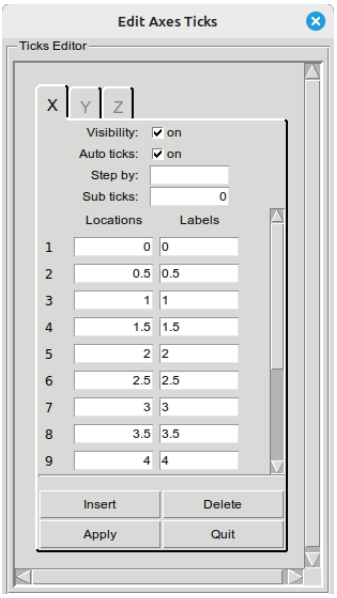

Рис. 4.53. Окно **Edit Axes Ticks**

стоятельно 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»;

– **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** задаёт 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 её 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**.

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. 4.54).

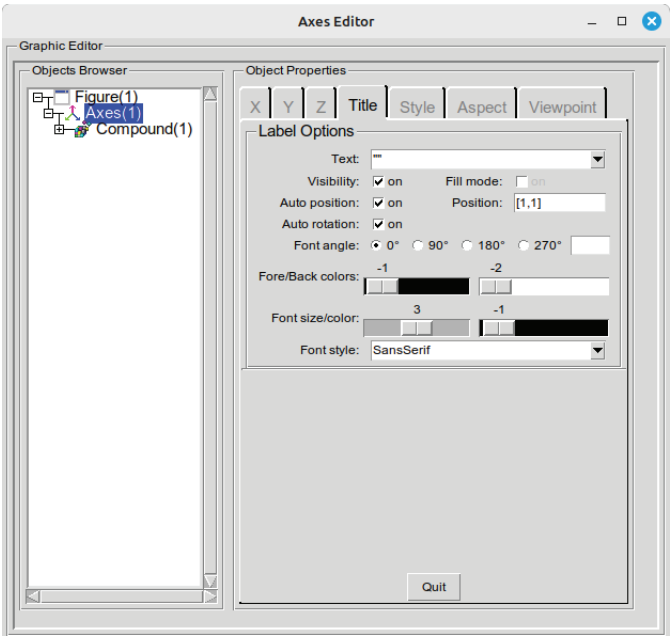

Рис. 4.54. Закладка **Title** окна форматирования **Axes Editor**

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* «Ось абсцисс» и для оси *Y* «Ось ординат», установим для подписей стиль шрифта *Serif 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 «График y=f(x)», определив стиль шрифта *Serif Bold*, размер символов 4. Включив режим заливки, установим её цвет – голубой (положение ползунка 12) и цвет границы – синий (положение ползунка 11).

В итоге сформируется графическое окно, представленное на рис. 4.55.

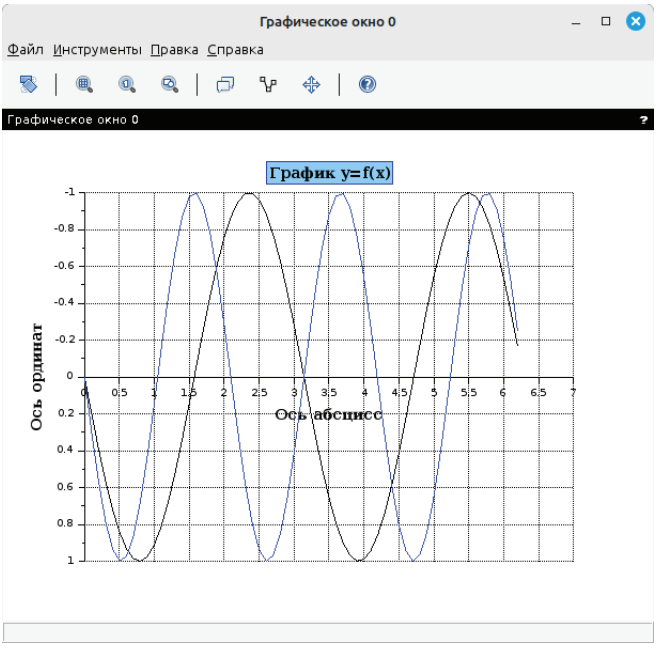

Рис. 4.55. Форматирование осей графика средствами закладок **X**, **Y**, **Title**

Закладка **Style** окна форматирования осей графика **Axes Editor** (см. рис. 4.56) 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 *Serif*;

- **Font color** – ползунок, каждое положение которого определяет цвет символов подписей засечек. По умолчанию установлен в положении –1 – чёрный цвет;
- **Font size** – размер символов подписей засечек на оси, возможны значения от 0 до 6. По умолчанию для шрифта установлен размер 1;
- **Fore. color** – ползунок, каждое положение которого определяет цвет собственно координатной оси. По умолчанию установлен в положении –1 – чёрный 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.

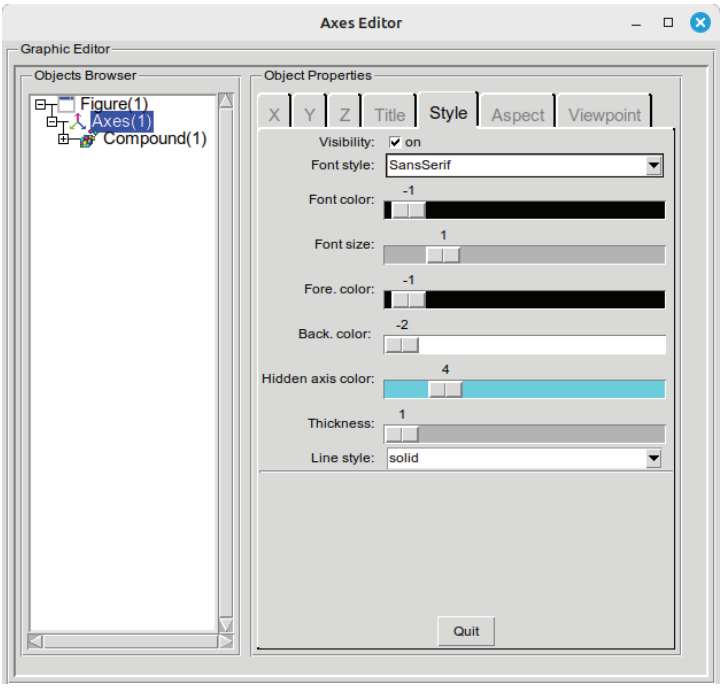

Рис. 4.56. Закладка **Style** окна форматирования осей **Axes Editor**

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

На экран будет выведено графическое окно, представленное на рис. 4.57.

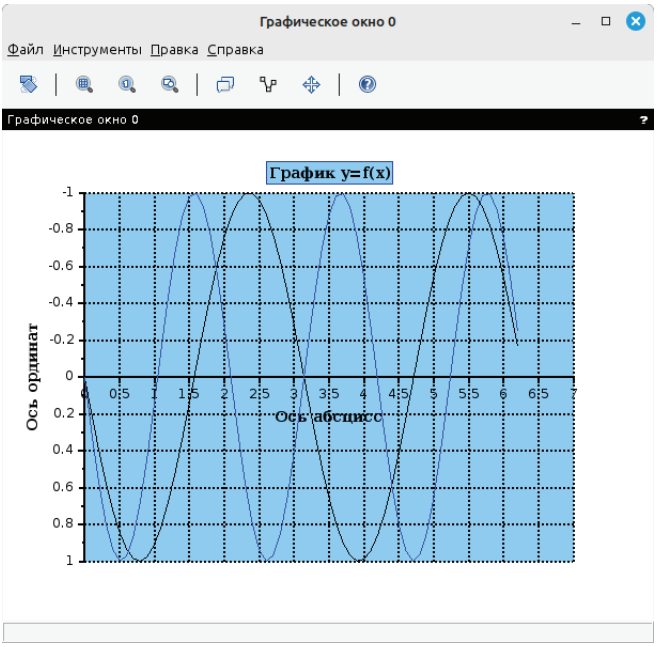

Рис. 4.57. Форматирование осей графика средствами закладки **Style**

Закладка **Aspect** окна форматирования **Axes Editor** (см. рис. 4.58) 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 отключён (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 всём заданном интервале, но в том же масштабе, что и предыдущий график. При отключённом режиме **Auto scale** новый график будет построен в пределах осей предыдущего графика и, возможно, будет отражать лишь часть заданного интервала.

  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** включён, оси будут изменены так, что оба графика сформируются полностью на заданных интервалах (см. листинг 4.37, рис. 4.59).

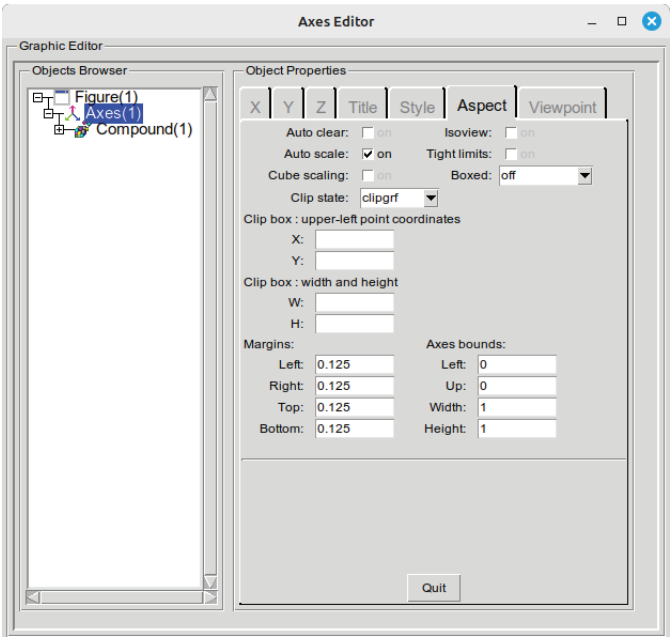

Рис. 4.58. Закладка **Aspect** окна форматирования осей **Axes Editor**

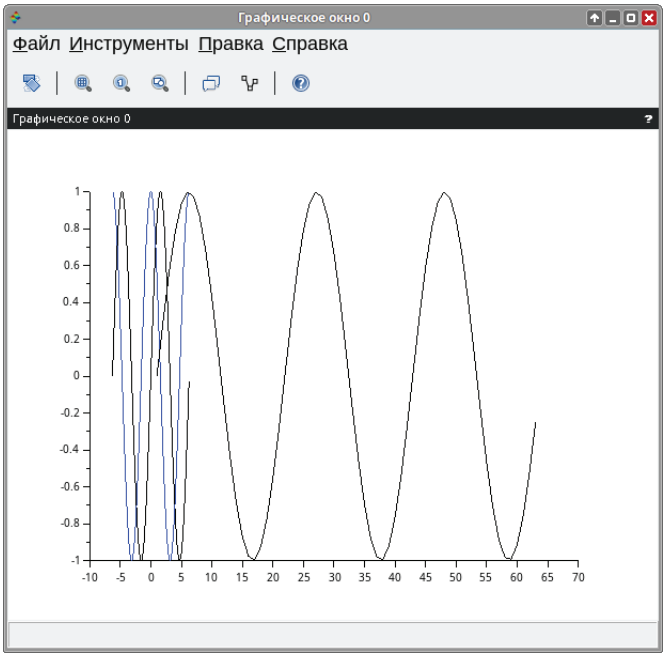

Рис. 4.59. Построение графиков с включённым режимом **Auto scale**

Листинг 4.37. Пример использования режима автоматического масштабирования координатных осей **Auto scale**

```
x=[-2*%pi:%pi/100:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y');
x=[0:0.1:2*%pi];
plot2d(sin(3*x));
```

Теперь построим первые два графика, отключим режим **Auto scale**, а затем построим третий график. Он будет сформирован лишь частично (см. листинг 4.38, рис. 4.60).

Листинг 4.38. Построение графиков нескольких функций в одном графическом окне при выключенном режиме **Auto scale**

```
x=[-2*%pi:%pi/100:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y');
//отключаем режим Auto scale
x=[0:%pi/100:2*%pi];
plot2d(sin(3*x));
```

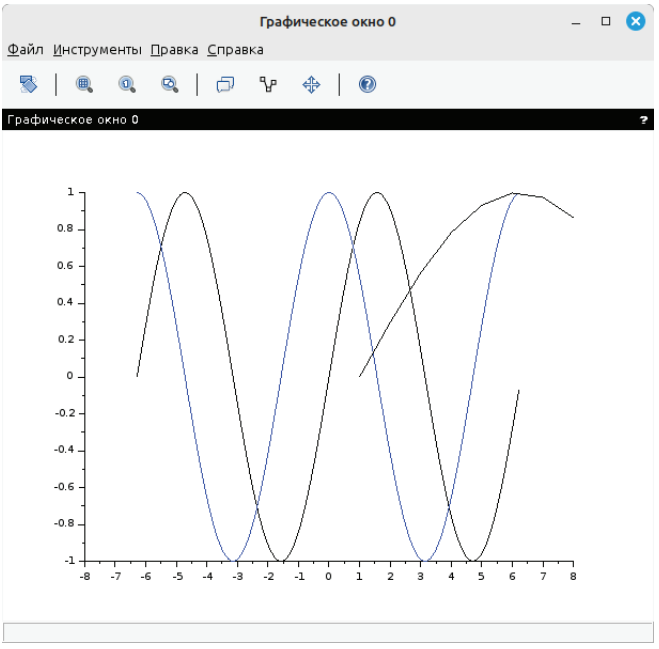

Рис. 4.60. Построение графиков
с отключённым режимом **Auto scale**

- 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 включён, 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 неё засечки.
- **Cube scaling** – эта опция применима лишь к трёхмерным 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 задаёт 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 задаётся матрицей [0 0 1 1].

Закладка **View Point** окна форматирования осей графика **Axes Editor** (см. рис. 4.61) позволяет установить лишь одно свойство – угол, под которым наблюдатель видит график. По умолчанию установлены значения углов поворота наблюдателя (**Rotation angles**) 0 и 270.

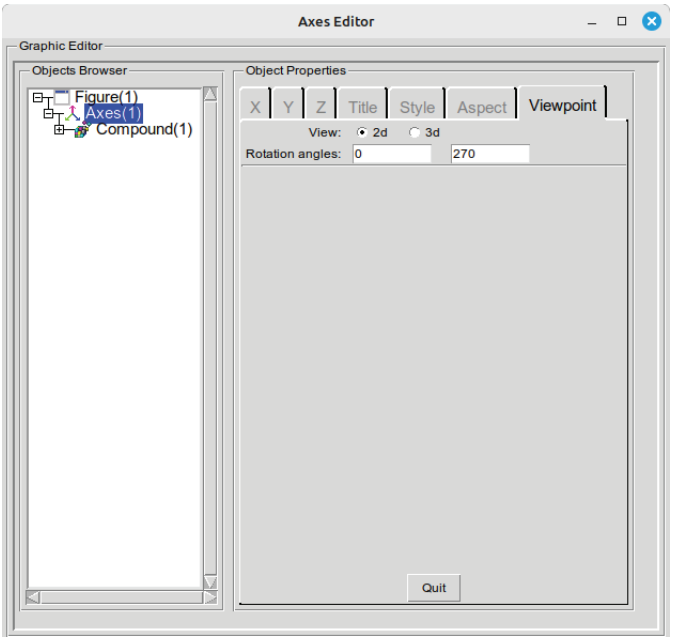

Рис. 4.61. Закладка **View Point** окна форматирования **Axes Editor**

4.11.2 Форматирование объекта Polyline

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 трёх закладках: **Style**, **Data**, **Clipping**.

Закладка **Style** окна форматирования **Polyline Editor** (см. рис. 4.62) позволяет установить значения следующих свойств:

- **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. 4.63);
- **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;

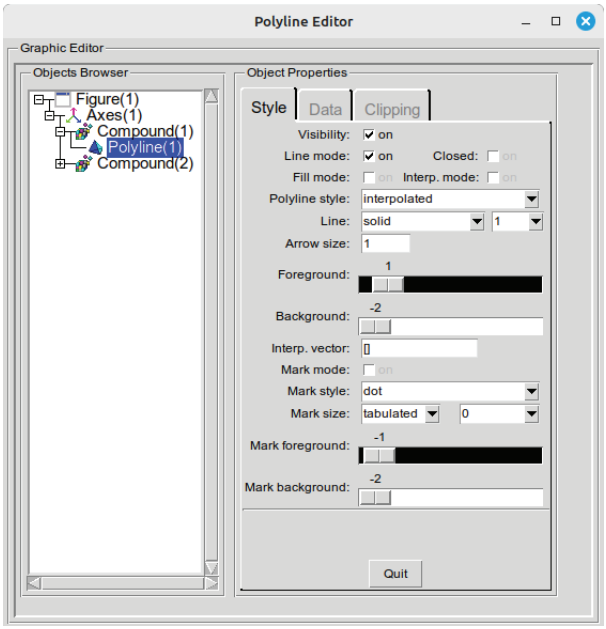

Рис. 4.62. Закладка **Style** окна форматирования **Polyline Editor**

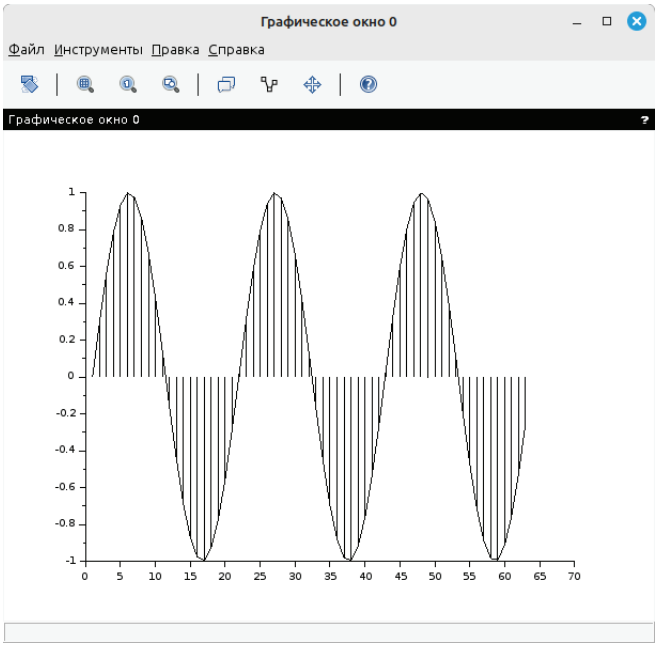

Рис. 4.63. График функции $y = \sin(3x)$.
Стиль отображения линии графика – *bar*

- **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»;
- **Interp color vector** – вектор, определяющий заливку каждого сегмента графика;
- **Mark mode** – режим, позволяющий строить точечные графики (положение переключателя «on»). По умолчанию это свойство отключено;
- **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* – звёздочка; *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. 4.64 представлен точечный график функции $y = \sin(3x)$ с типом маркера *filled diamond*, размером маркера 1, значением цвета заливки маркера *Mark foreground* 5.

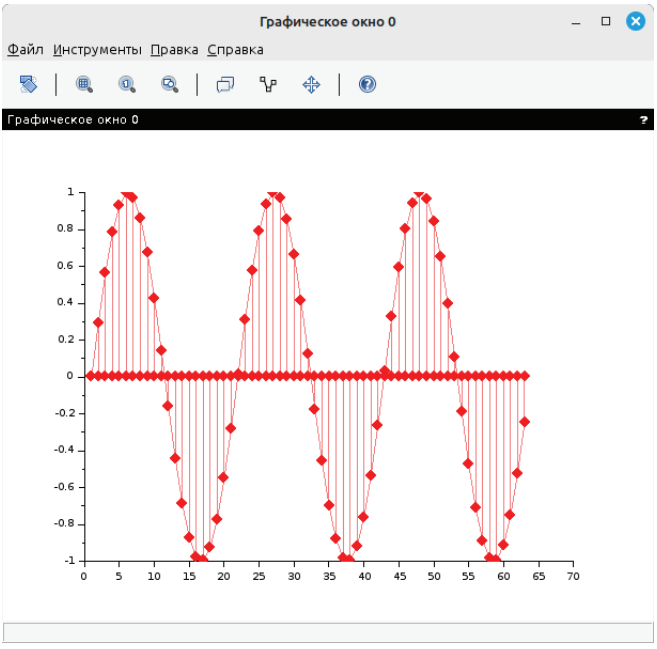

Рис. 4.64. Форматирование точечного графика функции $y = \sin(3x)$

Закладка **Data** окна форматирования **Polyline Editor** позволяет уточнить область данных, по которым строится график. 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. 4.65). Однако в этом списке можно выбрать строку **Edit data** и отредактировать таблицу исходных данных. Закладка **Clipping** (Обрезка) окна форматирования **Polyline Editor** позволяет установить границы прямоугольной области – **Clip box** (Кадр), которая останется видимой после обрезки изображения (см. рис. 4.66).

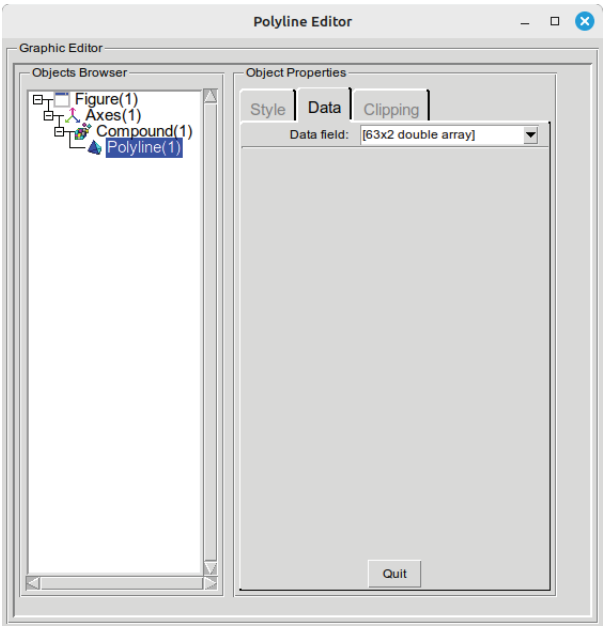

Рис. 4.65. Закладка **Data** окна форматирования **Polyline Editor**

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**.

Дадим определение прямоугольной (или декартовой) системы координат в пространстве.

Прямоугольная система координат в пространстве состоит из заданной фиксированной точки *O* пространства, называемой *началом координат*, и трёх перпендикулярных прямых пространства *OX*, *OY* и *OZ*, не лежащих в одной плоскости и пересекающихся в начале координат, – их называют координатными осями (*OX* – ось абсцисс, *OY* – ось ординат, *OZ* – ось аппликат). Положение точки *M* в пространственной системе координат определяется значением трёх координат и обозначается $M(x, y, z)$. Три плоскости,

содержащие пары координатных осей, называются *координатными плоскостями XY*, *XZ* и *YZ*.

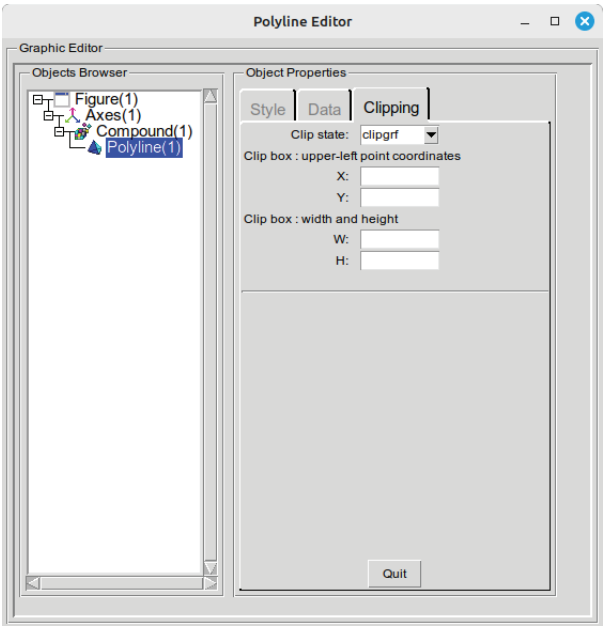

Рис. 4.66. Закладка **Clipping** окна форматирования **Polyline Editor**

Величина z называется функцией двух величин x и y, если каждой паре чисел, которые могут быть значениями переменных x и y, соответствует одно или несколько определённых значений величины z. При этом переменные x и y называют аргументами функции $z(x, y)$. Пары тех чисел, которые могут быть значениями аргументов x, y функции $z(x, y)$, в совокупности составляют область определения этой функции.

Для построения графика двух переменных $z = f(x, y)$ необходимо выполнить следующие действия:

1) сформировать в области построения графика прямоугольную сетку, проводя прямые, параллельные осям $y = y_j$ и $x = x_i$, где $x_i = x_0 + ih$, $h = \frac{x_n - x_0}{n}$, $i = 0, 1, 2, \ldots, n$; $y_j = y_0 + j\delta$, $\delta = \frac{y_k - y_0}{k}$, $j = 0, 1, 2, \ldots, k$;
2) вычислить значения $z_{i,j} = f(x_i, y_j)$ во всех узлах сетки;
3) обратиться к функции построения поверхности, передавая ей в качестве параметров сетку и матрицу $Z = \{z_{i,j}\}$ значений в узлах сетки.

4.12 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 её одним цветом, а `plot3d1` строит поверхность, каждая ячейка которой

имеет цвет, зависящий от значения функции в каждом соответствующем узле сетки (см. рис. 4.67).

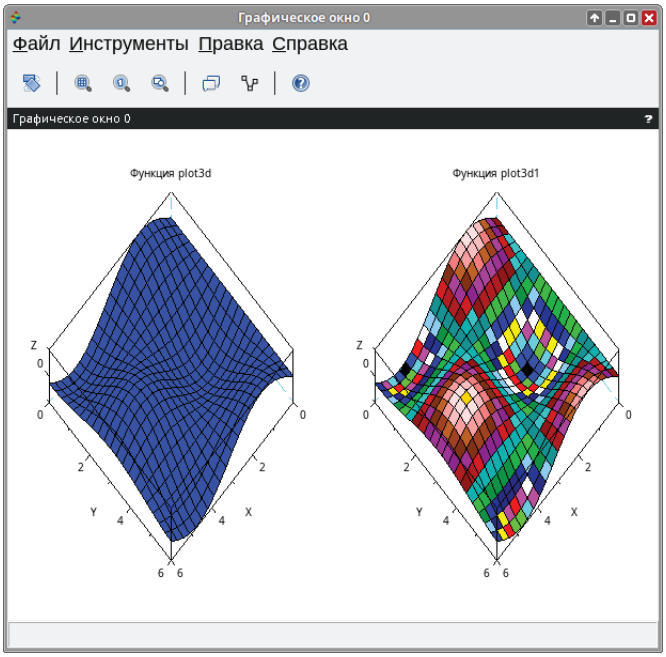

Рис. 4.67. Отличие `plot3d` от `plot3d1`

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 трёх целочисленных параметров: `[mode,type,box]`.
  Здесь:
  – `mode` – устанавливает цвет поверхности (см. табл. 4.5). По умолчанию равен 2 – цвет заливки синий, прямоугольная сетка выводится;

Таблица 4.5. 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. 4.6), по умолчанию имеет значение 2;

Таблица 4.6. 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. 4.7), по умолчанию равен 4;

Таблица 4.7. 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, её цвет, цвет заливки фона графического окна, наличие маркера и др.

Задача 4.29

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 4.39, рис. 4.68).

Листинг 4.39. 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);
```

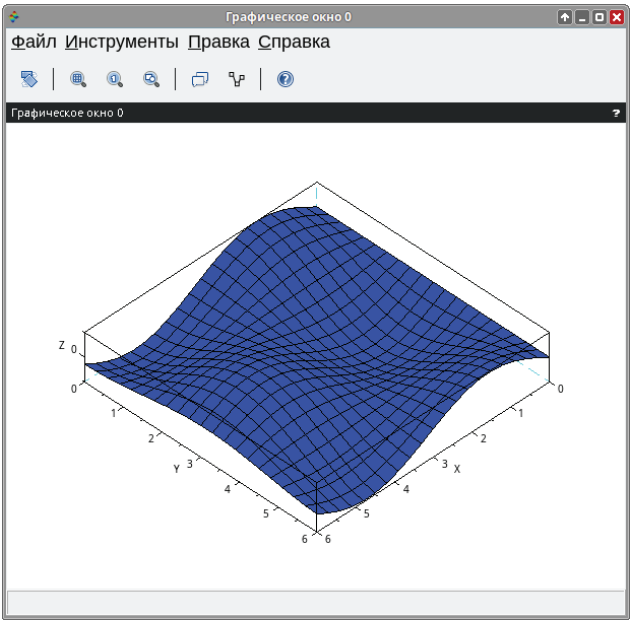

Рис. 4.68. График функции $Z = \sin(t) \cdot \cos(t)$

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

Задача 4.30

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

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.

Обратите внимание, в этой задаче мы создали шкалу цвета. Её использование часто облегчает чтение графика. Для её вывода в Scilab существует

команда `colorbar(n, m)`, здесь `n` – минимальное значение диапазона, `m` – максимальное значение.

Для построения поверхности обратимся к функции `plot3d1` (см. листинг 4.40, рис. 4.69).

Листинг 4.40. Матрица значений функции $Z = 5y^2 - x^2$

```
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
plot3d(x',y',z,-125,51);
colorbar(-3,3) //функция, которая выводит шкалу цвета
```

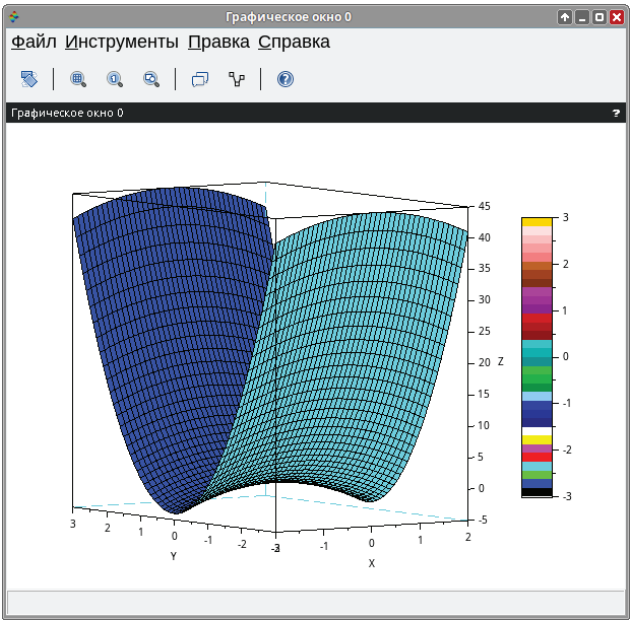

Рис. 4.69. График функции $Z = 5y^2 - x^2$

Также для построения трёхмерных графиков используется 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 4.31
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`. Для формирования графика обратимся к функции `plot3d` (см. листинг 4.41, рис. 4.70).

Листинг 4.41. Создание прямоугольной сетки графика командой `genfac3d` и построение её 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);
```

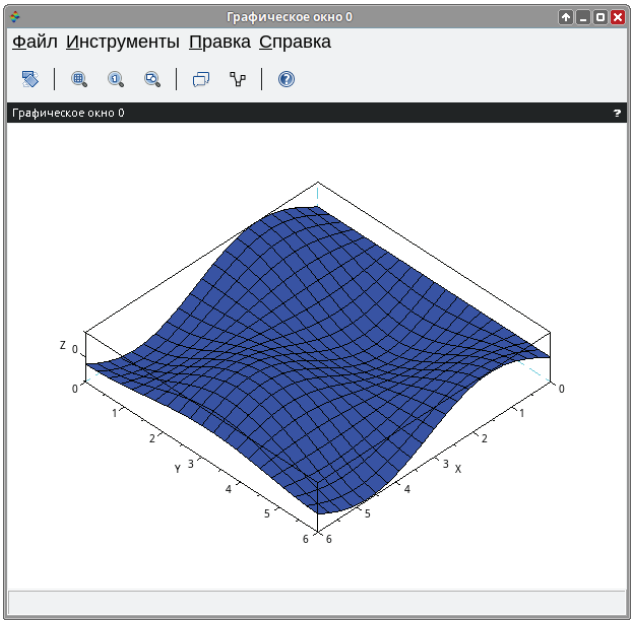

Рис. 4.70. График функции $Z = \sin(t) \cdot \cos(t)$, построенный при помощи команды `genfac3d`

Для построения графиков можно использовать команду `eval3dp`:

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

Здесь:

- `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, определённая пользователем, которая задаёт трёхмерный график;
- `p1` – вектор размера m;
- `p2` – вектор размера n.

Задача 4.32

Построить график, заданный следующими уравнениями:

$x = p_1 \cdot \sin(p_1) \cdot \cos(p_2), \quad y = p_1 \cdot \cos(p_1) \cdot \cos(p_2), \quad z = p_1 \cdot \sin(p_2).$

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

Листинг 4.42. Создание прямоугольной сетки графика командой `eval3dp` и построение её графика с помощью функции `plot3d`

```
p1=linspace(0,2*%pi,10);
p2=linspace(0,2*%pi,10);
function [x,y,z]=scp(p1,p2)
x=p1.*sin(p1).*cos(p2);
y=p1.*cos(p1).*cos(p2);
z=p1.*sin(p2)
end;
[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. 4.71).

4.13 Функции meshgrid, surf и mesh

Для формирования прямоугольной сетки существует функция `meshgrid`. Обращение к ней имеет вид:

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

Здесь:

- `(x[, y, z])` – векторы, указываемые через запятую;
- `[X[, Y, Z]]` – матрицы, если два входных аргумента. Если входных аргументов три, то трёхмерные массивы.

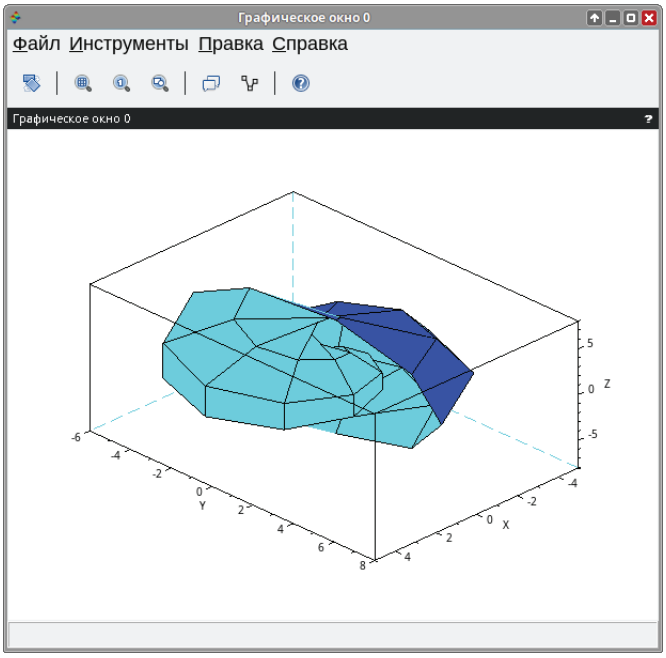

Рис. 4.71. График функции,
построенный при помощи команды `eval3dp`

После формирования сетки вывести в неё график можно с помощью функций `surf` либо `mesh`.

`surf` строит каркасную поверхность, заливая каждую ячейку своим цветом. Цвет конкретной ячейки зависит от значения функции в конкретном узле сетки (аналогично `plot3d1`).

`mesh` строит каркасную поверхность и заливает её одним цветом (аналогично `plot3d`). По умолчанию цвет заливки – белый.

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 сетки;
- `keyn=valuen` – последовательность значений свойств графика `key1=value1, key2=value2, ..., keyn=valuen`, определяющих его внешний вид.

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

Задача 4.33
Построить график функции $z(x, y) = 3x^2 - 2\sin^2 y$, $x \in [-2, 2]$, $y \in [-3, 3]$.

Для формирования сетки воспользуемся функцией `meshgrid`.

```
-->[x y]=meshgrid(-2:2,-3:3)
y  =
-3.  -3.  -3.  -3.  -3.
-2.  -2.  -2.  -2.  -2.
-1.  -1.  -1.  -1.  -1.
0.   0.   0.   0.   0.
1.   1.   1.   1.   1.
2.   2.   2.   2.   2.
3.   3.   3.   3.   3.

x  =
-2.  -1.   0.   1.   2.
-2.  -1.   0.   1.   2.
-2.  -1.   0.   1.   2.
-2.  -1.   0.   1.   2.
-2.  -1.   0.   1.   2.
-2.  -1.   0.   1.   2.
-2.  -1.   0.   1.   2.
```

После формирования сетки вычислим значение функции во всех узловых точках:

```
--> z=3*x.*x-2*sin(y).^2
z =
11.96017    2.96017   -0.03983    2.96017    11.96017
10.34636    1.34636   -1.65364    1.34636    10.34636
10.58385    1.58385   -1.41615    1.58385    10.58385
12.00000    3.00000    0.00000    3.00000    12.00000
10.58385    1.58385   -1.41615    1.58385    10.58385
10.34636    1.34636   -1.65364    1.34636    10.34636
11.96017    2.96017   -0.03983    2.96017    11.96017
```

Для построения каркасного графика следует обратиться к функции `mesh`:

```
mesh(x, y, z);
```

После это будет создано графическое окно с трёхмерным графиком (см. рис. 4.72).

Как видно, построенный график получился грубым. Для создания более точного и плавного графика следует сделать сетку более плотной, т. е. самостоятельно указать шаг (см. листинг 4.43 и рис. 4.73).

Листинг 4.43. Построение графика поверхности с заданным шагом

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

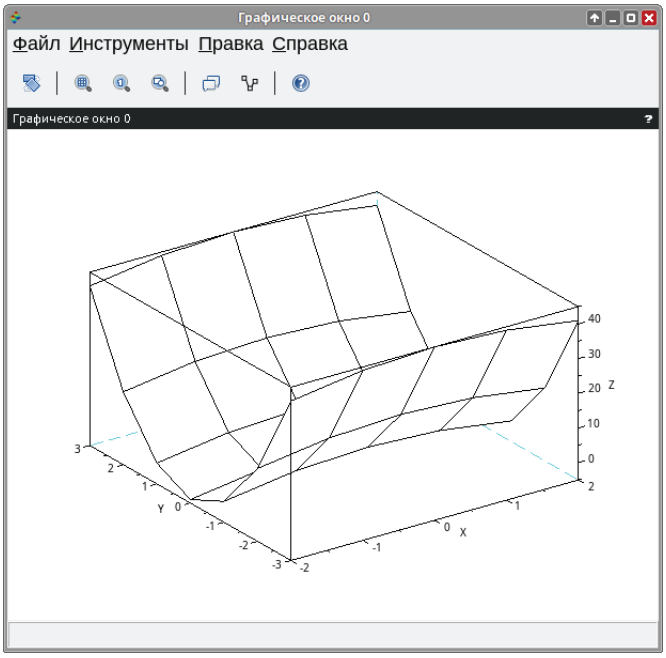

Рис. 4.72. График функции $z(x, y) = 3x^2 - 2\sin^2 y$

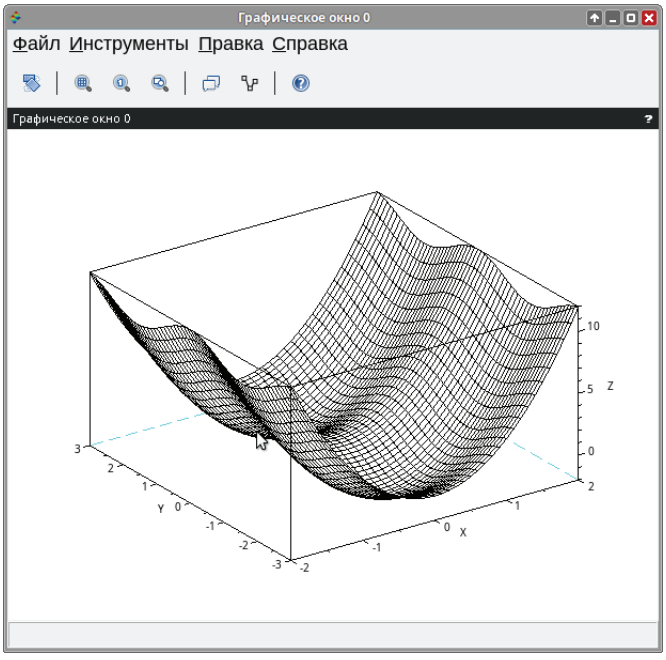

Рис. 4.73. График функции $z(x, y) = 3x^2 - 2\sin^2 y$
с более плотной сеткой

Любой трёхмерный график можно вращать. Для этого следует зажать правую клавишу мыши и потянуть в нужном направлении. Кроме того, можно изменить масштаб графика, используя колесо мыши.

Задача 4.34

С использованием функции `surf` построить график функции $z(x, y) = \sqrt{\sin^2 x + \cos^2 y}$.

В листинге 4.44 представлено решение задачи, а на рис. 4.74 изображён получившийся график.

Листинг 4.44. Построение графика поверхности

```
[x y]=meshgrid(-2:0.2:2,0:0.2:4);
z=sqrt(sin(x).^2+cos(y).^2);
surf(x,y,z);
```

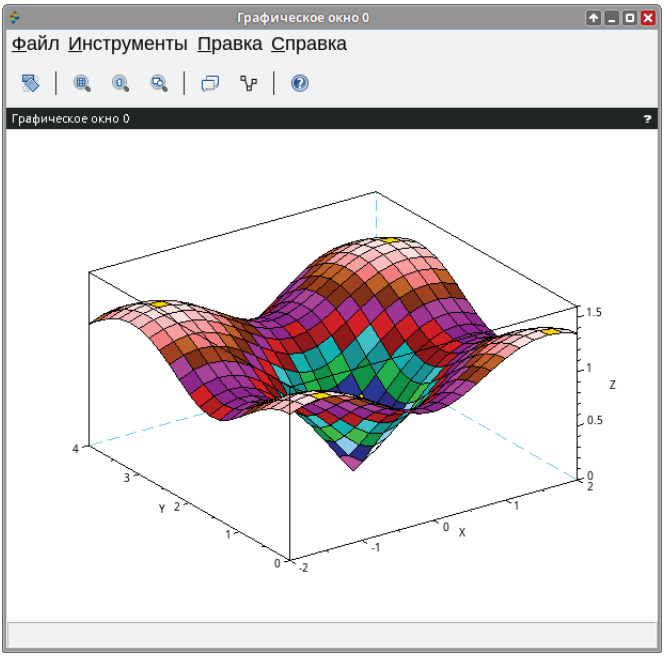

Рис. 4.74. График функции $z(x, y) = \sqrt{\sin^2 x + \cos^2 y}$

В Scilab можно построить графики двух поверхностей в одной системе координат. По умолчанию, как и для двумерных графиков, команда `mtlb_hold` имеет значение `'on'` и блокирует создание нового графического окна при выполнении команд `surf` или `mesh`. Для перерисовки графиков в окне следует использовать `mtlb_hold('off')`.

Задача 4.35

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`. Вычислим значения функций во всех узлах сетки.

Поверхность $z = +(3x^2 + 4y^2) - 1$ построим с помощью команды `surf`, каждая её ячейка будет залита цветом, зависящим от значения функции в узле сетки. А с помощью команды `mesh` построим поверхность $z_1 = -(3x^2 + 4y^2) - 1$ в одних координатных осях с $z = +(3x^2 + 4y^2) - 1$, при этом будет выведена прямоугольная сетка, а ячейки залиты белым цветом (см. листинг 4.45, рис. 4.75).

Листинг 4.45. Построение графиков нескольких функций в одном графическом окне

```
[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);
mesh(x,y,z1);
```

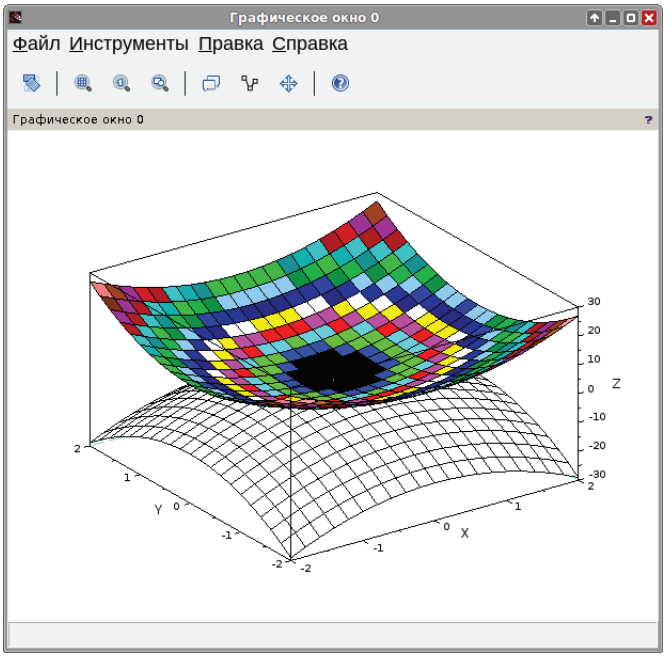

Рис. 4.75. График функции $z(x, y) = \pm(3x^2 + 4y^2) - 1$

4.13.1 Построение графиков поверхностей, заданных параметрически

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. 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 представить в виде $f(u)$ или $g(v)$, то в этом случае матрицы X, Y и Z следует записывать в виде $f(u) \cdot \text{ones}(v)$ или $g(v) \cdot \text{ones}(u)$ соответственно.

4.14 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 задаётся набором граней. То есть 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 4.36

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`.

Листинг 4.46. 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);
```

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 4.46). Построенная функцией `plot3d2` сфера представлена на рис. 4.76.

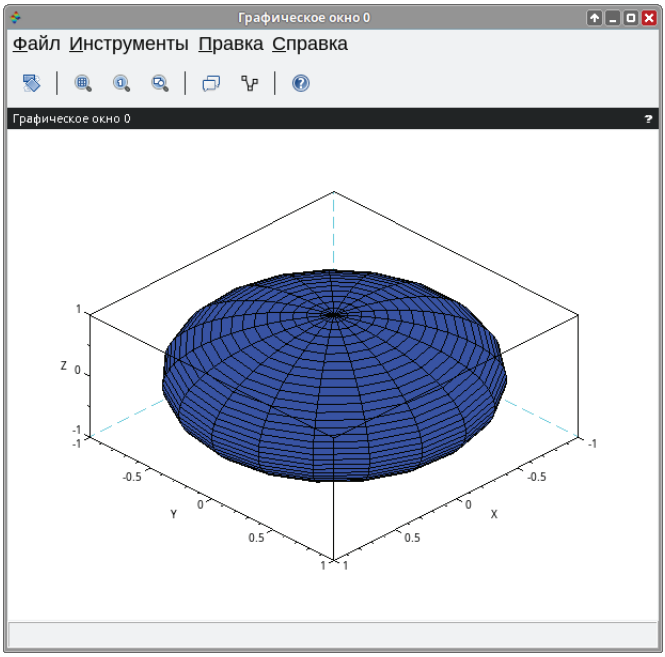

Рис. 4.76. График сферы, построенный функцией `plot3d2`

Теперь посмотрим, как эту же задачу выполнят функции `plot3d` и `plot3d3`. Расчёты матриц и векторов аналогичны, отличается лишь функция построения графика.

Листинг 4.47. 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);
```

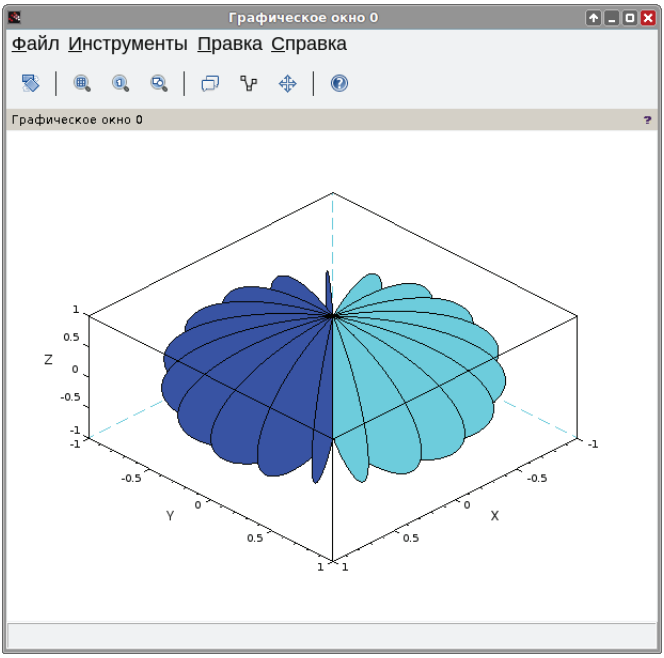

Рис. 4.77. График сферы, построенный функцией `plot3d`

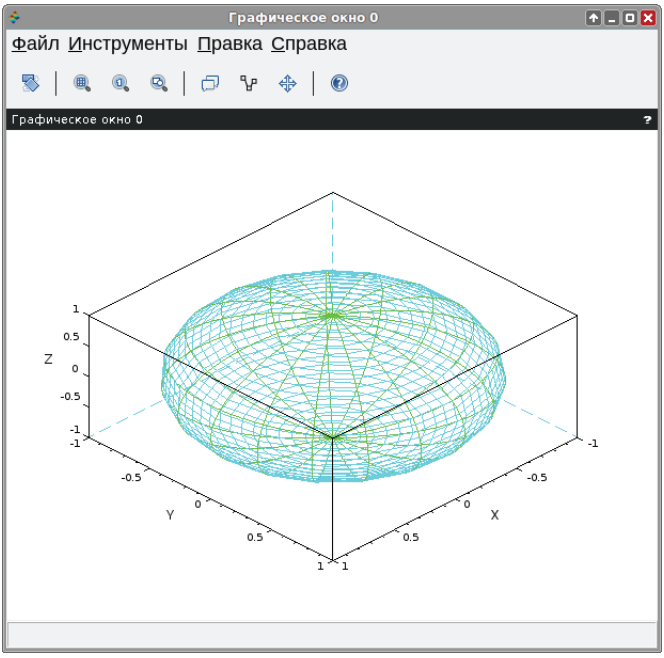

Рис. 4.78. График сферы, построенный функцией `plot3d3`

Листинг 4.48. Построение сферы с помощью функции plot3d3

```
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);
plot3d3(X,Y,Z);
```

4.15 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 4.37

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 4.49, рис. 4.79).

Листинг 4.49. Построение линии, заданной параметрически, с помощью функции param3d

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

Задача 4.38

Построить линию, заданную параметрически:

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

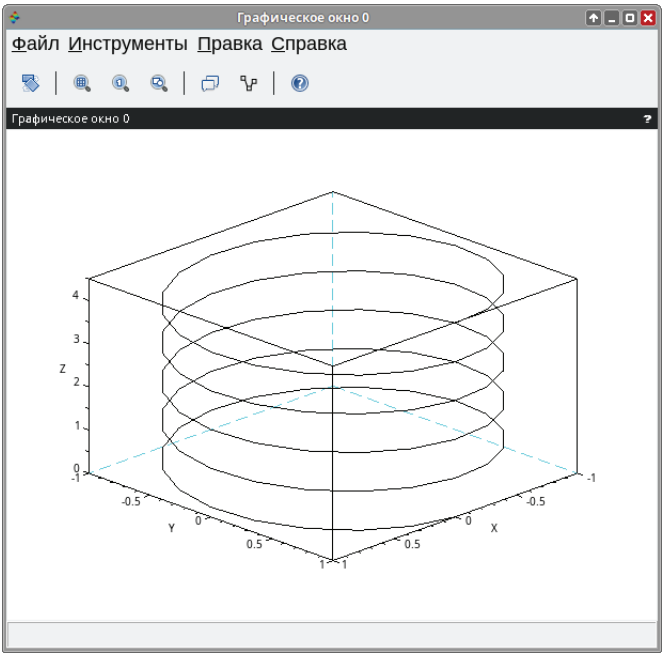

Рис. 4.79. График параметрической линии, построенный функцией `param3d`

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° (см. листинг 4.50, рис. 4.80).

Листинг 4.50. Построение линии, заданной параметрически, с помощью функции `param3d`

```
t=-50*%pi:%pi/10:50*%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.

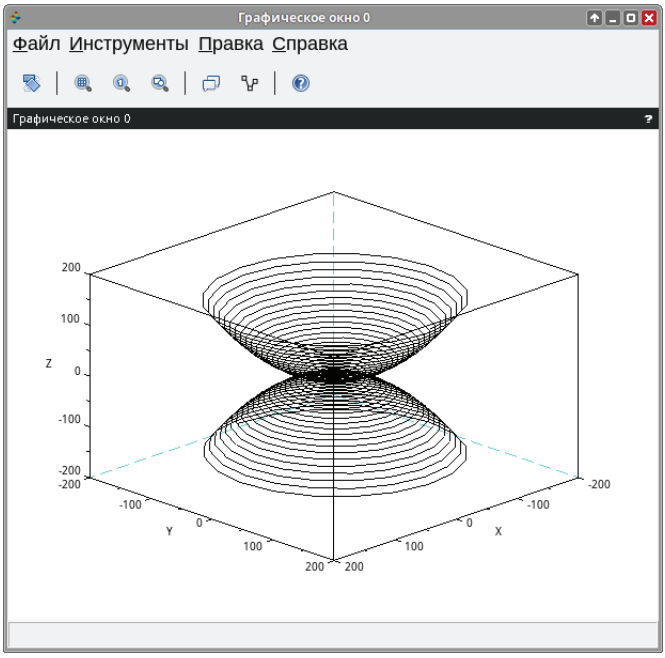

Рис. 4.80. График параметрической линии, построенный функцией `param3d`

Задача 4.39

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` определим 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 4.51, рис. 4.81).

Листинг 4.51. Построение нескольких параметрически заданных линий с помощью команды `list` функции `param3d`

```
t=[0:%pi/50:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],...
list([t/10,sin(t)],[9,5]),35,45,"X@Y@Z");
```

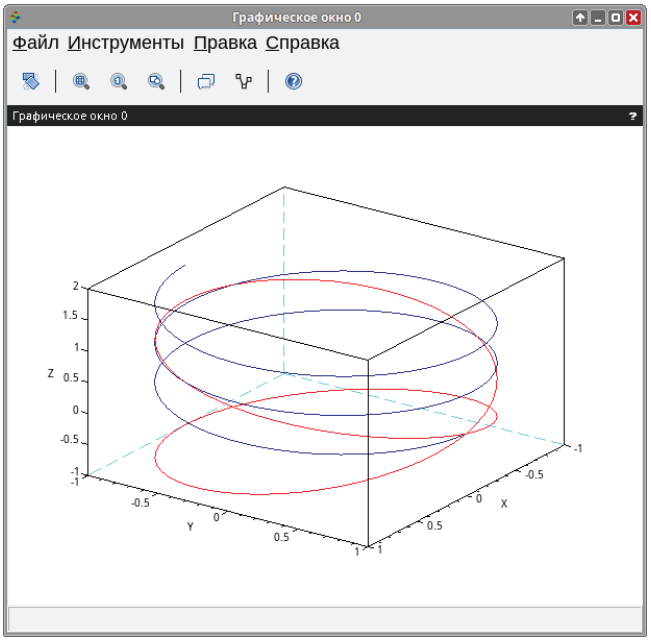

Рис. 4.81. Графики параметрических кривых, построенные функцией `param3d1`

Задача 4.40

Построить поверхность однополостного гиперболоида, уравнение которого задано в параметрическом виде: $x(u, v) = \text{ch}(u)\cos(v)$, $y(u, v) = \text{ch}(u)\sin(v)$, $z(u, v) = \text{sh}(u)$, $u \in [0, \pi]$, $v \in [0, 2\pi]$.

В листинге 4.52 представлено решение этой задачи. График однополостного гиперболоида представлен на рис. 4.82.

Листинг 4.52. Построение поверхности гиперболоида (пример 4.40)

```
clear; h=%pi/50;
u=[0:h:%pi]'; // Формируем вектор-столбец u.
// Формируем вектор-строку v. Обратите внимание, u - столбец,
// v - строка с одинаковым количеством элементов.
v=[0:2*h:6.28];
// Формируем матрицу X как матричное произведение ch(u)*cos(v).
X=cosh(u)*cos(v);
// Формируем матрицу Y как матричное произведение ch(u)*sin(v).
Y=cosh(u)*sin(v);
// Формируем матрицу Z как матричное произведение столбца sh(u)
// на строку ones(v).
Z=sinh(u)*ones(v);
surf(X,Y,Z);
xgrid();
title('Однополостный гиперболоид');xlabel('X');ylabel('Y');zlabel('Z')
```

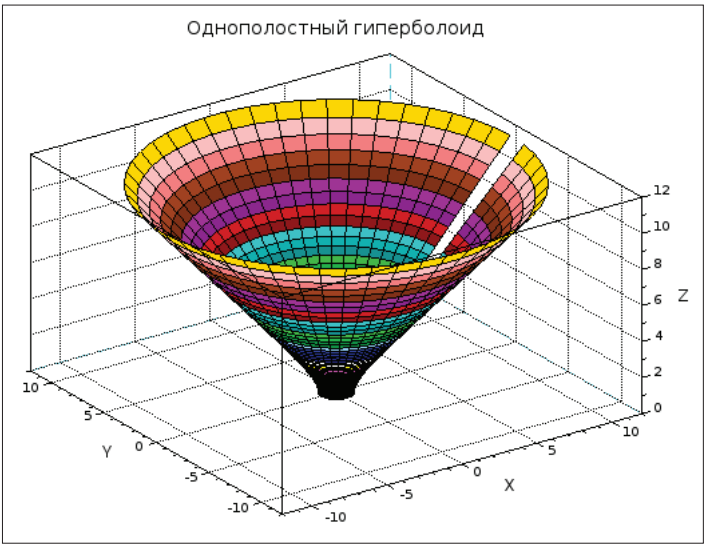

Рис. 4.82. График однополостного гиперболоида

Построение сферы в Scilab:

Задача 4.41

Построить поверхность сферы с центром (x_0, y_0, z_0) и радиусом R.

В декартовой системе координат уравнение сферы имеет вид: $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2$. Его можно записать в параметрическом виде:

$$\begin{cases} x(u,v) = x_0 + R\sin(u)\cos(v) \\ y(u,v) = y_0 + R\sin(u)\sin(v), \\ z(u,v) = z_0 + R\cos(u) \end{cases}$$

где $u \in [0, 2\pi)$, $v \in [0, \pi]$.

Методика построения сферы подобна методике построения однополостного гиперболоида, описанной в примере 4.40. В листинге 4.53 представлен текст программы построения сферы с центром в точке (1, 1, 1) и радиусом $R = 4$, а на рис. 4.83 изображена сфера.

Листинг 4.53. Построение сферы (пример 4.41)

```
clear;
h=%pi/30;
u=[-0:h:%pi]'; // Формируем вектор-столбец u.
v=[0:2*h:2*%pi]; // Формируем вектор-строку v.
// Формируем матрицу X, используя матричное произведение sin(u)*cos(v).
x=1+4*sin(u)*cos(v);
// Формируем матрицу Y, используя произведение sin(u)*sin(v).
```

```
y=1+4*sin(u)*sin(v);
// Формируем матрицу Z, используя произведение столбца
// cos(u) на строку ones(v).
z=1+4*cos(u)*ones(v);
// Формируем график поверхности.
surf(x,y,z,45,60);
xgrid();
// Подписываем график и оси.
title('Сфера');xlabel('X');ylabel('Y');zlabel('Z');
```

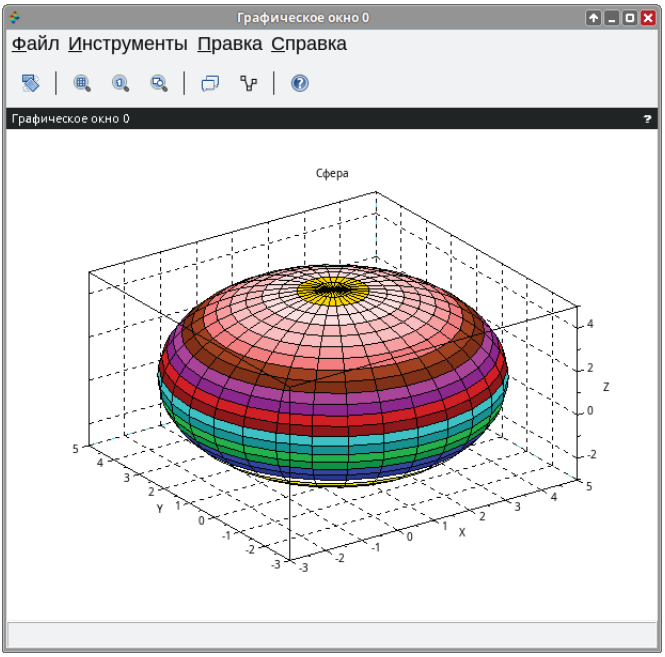

Рис. 4.83. График сферы,
построенный с использованием функции surf

Сфера является частным случаем более общей фигуры – эллипсоида. Рассмотрим способ построения эллипсоида.

Задача 4.42

Построить поверхность эллипсоида, уравнение которой задано в параметрическом виде:

$$\begin{cases} x(u,v) = x_0 + a\sin(u)\cos(v) \\ y(u,v) = y_0 + b\sin(u)\sin(v) \\ z(u,v) = z_0 + c\cos(u) \end{cases}.$$

Здесь a, b, c – полуоси эллипсоида, (x_0, y_0, z_0) – центр эллипсоида.

Методика построения эллипсоида подобна тому, как ранее были построены однополостный гиперболоид (пример 4.40) и сфера (пример 4.41). Для этого необходимо сформировать матрицы X, Y и Z, после чего вызвать функцию `surf`. Решение показано в листинге 4.54.

Листинг 4.54. Построение поверхности эллипсоида (пример 4.42)

```
clear;
h=%pi/30;
u=[-0:h:%pi]'; // Формируем вектор-столбец u.
v=[0:2*h:2*%pi]; // Формируем вектор-строку v.
// Формируем матрицу X, используя матричное произведение sin(u)*cos(v).
a=3;b=7;c=1;
x0=10;y0=10;z0=10;
x=x0+a*sin(u)*cos(v);
// Формируем матрицу Y, используя произведение sin(u)*sin(v).
y=y0+b*sin(u)*sin(v);
// Формируем матрицу Z, используя произведение столбца
// cos(u) на строку ones(v).
z=z0+c*cos(u)*ones(v);
surf(x,y,z);
xgrid(); // Формируем эллипсоид.
// Подписываем график и оси.
title('Элипсоид');xlabel('X');ylabel('Y');zlabel('Z');
```

Эллипсоид с центром в точке (10, 10, 10) и полуосями $a = 3$, $b = 7$, $c = 1$ представлен на рис. 4.84.

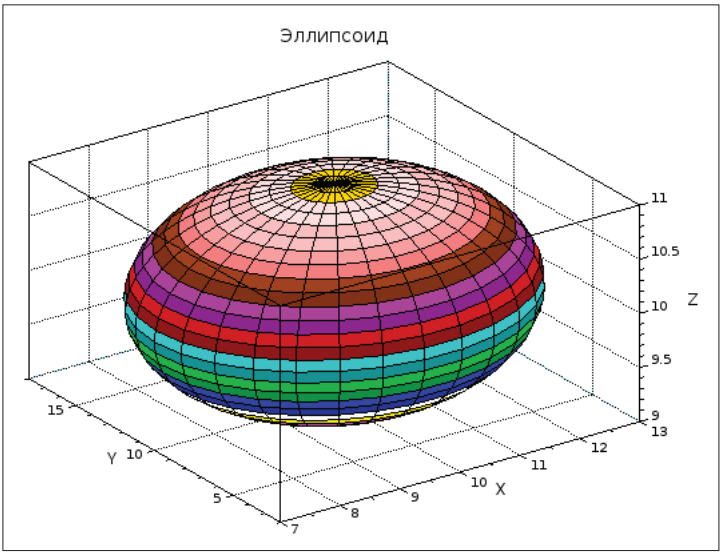

Рис. 4.84. Эллипсоид с центром в точке (10, 10, 10) и полуосями $a = 3$, $b = 7$, $c = 1$

4.16 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 будет проведено `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 трёх целочисленных параметров: `[mode,type,box]`.
  Здесь:
  - `mode` – устанавливает способ и место нанесения линий уровня (см. табл. 4.8);

Таблица 4.8. Значения параметра `mode`

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

  - `type` – позволяет управлять масштабом графика (см. табл. 4.6), по умолчанию имеет значение 2;

  - box – определяет наличие рамки вокруг отображаемого графика (см. табл. 4.7). 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 задаёт 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, определённую 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 её вызова; e1,e2,... – входные параметры.

Второй способ создания функции – это примение конструкции вида:

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

где lhs_arguments – список выходных параметров; function_name – имя создаваемой функции; rhs_arguments – входные параметры.

Задача 4.43

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

Введём 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 4.55, рис. 4.85).

Листинг 4.55. 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)
end
contour(t,t,my_surface,10);
```

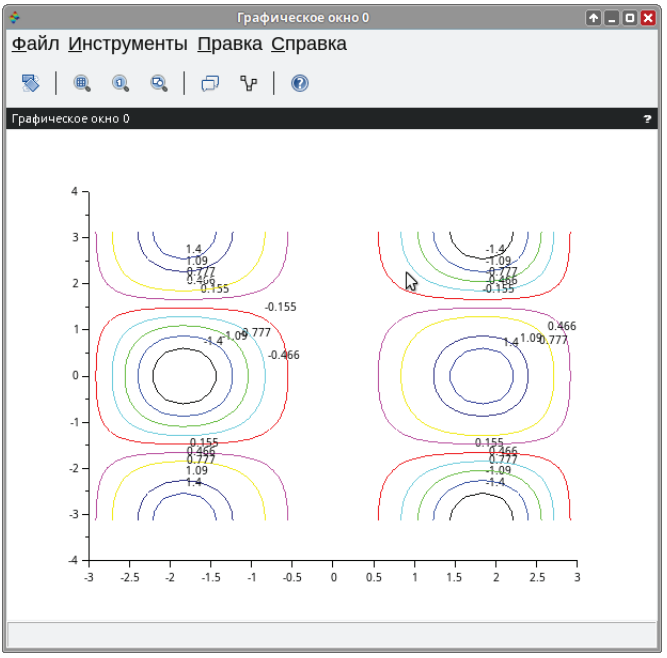

Рис. 4.85. Изолинии поверхности $Z = x \cdot \sin(x)^2 \cdot \cos(y)$

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

Задача 4.44

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

Прежде всего введём параметр t и сформируем массив его значений. Создадим функцию `Surf`. С помощью команды `rect` установим границы области построения графика в графическом окне, чтобы совместить поверхность с изолиниями.

Напомним, что при построении графика функции вида $Z(x, y)$ с помощью `plot3d` необходимо использовать команду `feval` для формирования матрицы значений функции $z_{ij} = f(x_i, y_j)$.

Далее с помощью `plot3d` строим график поверхности $Z = \sin(x) \cdot \cos(y)$, устанавливая углы обзора наблюдателя и подписи для координатных осей. Определяем 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 задаётся тем же уравнением, что и поверхность ($Z = \sin(x)\cdot\cos(y)$), 1 – границы области построения графика 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 4.56, рис. 4.86).

Листинг 4.56. Построение поверхности $Z = \sin(x)\cdot\cos(y)$ (функция `plot3d`) и вывод её изолиний (функция `contour`) в одном графическом окне командой `rect`

```
t=%pi*(-10:10)/10;
function [z]=Surf(x,y)
z=sin(x)*cos(y);
end;
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 и contour');
```

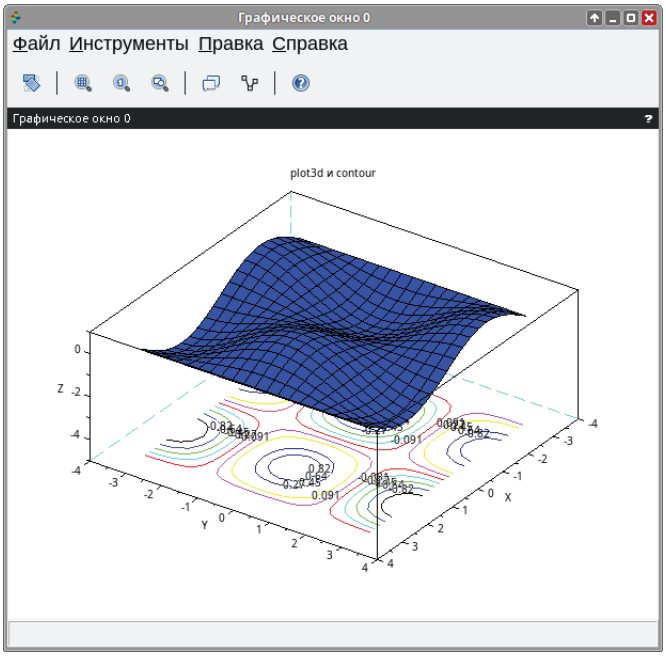

Рис. 4.86. График поверхности $Z = \sin(x)\cdot\cos(y)$ и её изолинии в одном окне

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

Задача 4.45

Совместить график поверхности $Z = \sin(x) \cdot \cos(y)$ с её 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 её 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 4.57, рис. 4.87).

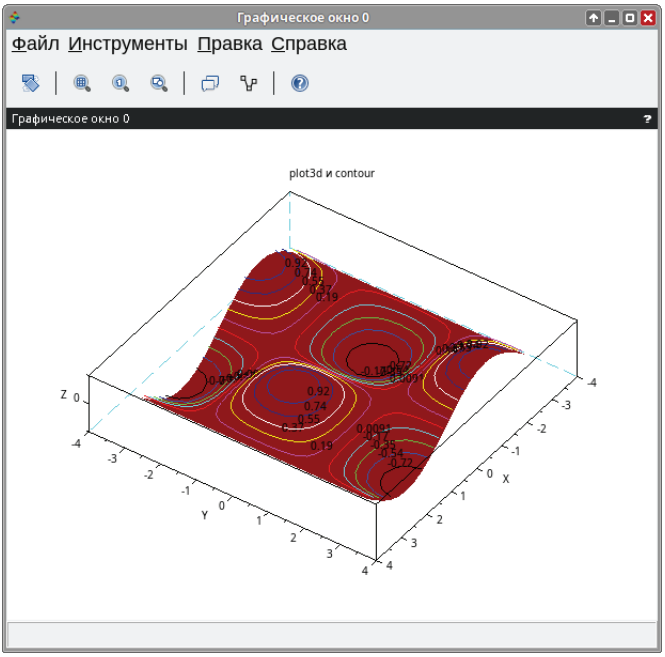

Рис. 4.87. Совмещение поверхности и её изолиний

Листинг 4.57. Пример совмещения графика поверхности с её изолиниями

```
t=%pi*(-10:10)/10;
function [z]=Surf(x,y)
z=sin(x)*cos(y)
end;
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 и contour');
```

4.17 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 будет проведено `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 трёх чисел – «`csa`». Здесь `c` (Captions) устанавливает режим отображения подписей графика (см. табл. 4.9); `s` (Scaling) – режим масштабирования (см. табл. 4.10); `a` (Axes) определяет положение осей графика (см. табл. 4.11);

Таблица 4.9. Значение параметра `c` (Captions) строки `strf`

Значение	Описание
0	Нет подписей
1	Отображаются подписи, заданные параметром leg

Таблица 4.10. Значение параметра s (Scaling) строки strf

Значение	Описание
0	Масштабирование по умолчанию
1	Устанавливается параметром rect
2	Масштаб зависит от минимального и максимального значений 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 в зависимости от значений параметра rect
6	Расширение осей для наилучшего вида в зависимости от входных данных

Таблица 4.11. 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)

- 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;
- nax – это массив из четырёх 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 4.46

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

Введём параметр *t* и создадим массив его значений, определим функцию surf. Для наглядности приведём график поверхности $Z = \sin(x) \cdot \cos(y)$, построенный функцией plot3d1, и её изображение на горизонтальной плоскости, сформированное функцией contourf, в одном графическом окне. С этой целью обратимся 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 её 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 передаём 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 значений входных данных; 1 – режим отображения координатных осей, ось *Y* находится слева).

Выведем и для этого графика подписи осей и графика в целом – «contourf» – при помощи команды `xtitle` (см. листинг 4.58, рис. 4.88).

Листинг 4.58. Построение поверхности $Z = \sin(x) \cdot \cos(y)$ (функция `plot3d`) и её изображения на горизонтальной поверхности (функция `contourf`) в одном графическом окне

```
t=-%pi:0.2:%pi;
function [z]=Surf(x,y)
z=sin(x)*cos(y)
end;
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');
```

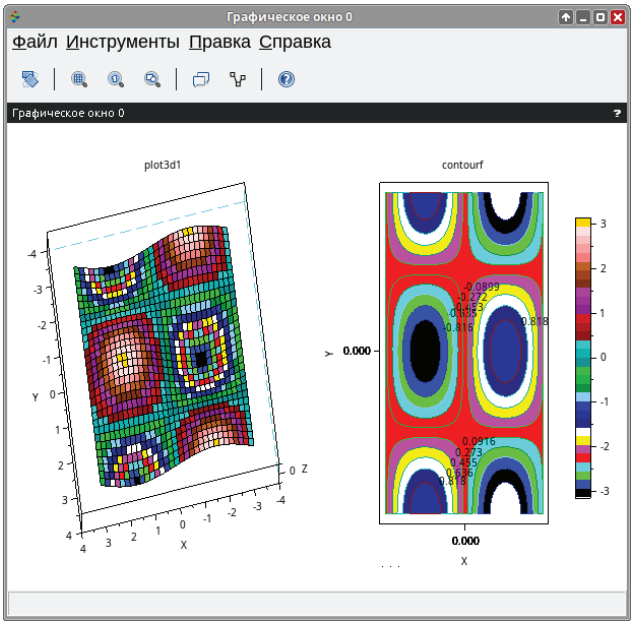

Рис. 4.88. График функции $Z = \sin(x) \cdot \cos(y)$, построенный функцией `plot3d1` и `contourf`

4.18 Функция hist3d

Для построения трёхмерных 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 4.47

Построить трёхмерную гистограмму.

Для формирования матрицы входных данных воспользуемся командой `rand`. Напомним, чтобы создать матрицу случайных значений размером `(m,n)`, необходимо использовать конструкцию `rand(m,n)` (см. листинг 4.59).

Листинг 4.59. Построение трёхмерной гистограммы при помощи функции `hist3d`

```
hist3d(9.7*rand(10,10),20,35);
```

Полученная гистограмма изображена на рис. 4.89.

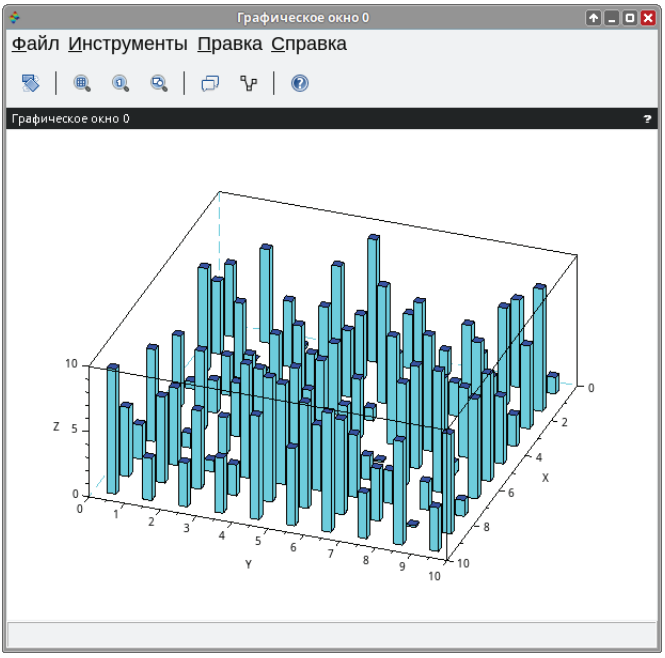

Рис. 4.89. Трёхмерная гистограмма, построенная функцией `hist3d`

4.19 Примеры построения некоторых трёхмерных графиков в Scilab

В этом параграфе мы рассмотрим приёмы построения некоторых нестандартных трёхмерных графиков средствами Scilab.

Прежде всего научимся вырезать из графиков «ненужные» части.

Задача 4.48

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.

`find` имеет синтаксис:

```
x1 = find(x)
```

где `x` – логическое выражение, с помощью которого задаётся условие поиска элементов в массиве; `x1` – результирующий массив индексов тех элементов, которые удовлетворяют заданному условию.

Далее мы будем использовать `%inf` – машинный символ бесконечности. Команда `z(z1)=%inf*z1` сделает равными бесконечности те элементы массива `z`, чьи индексы содержатся в `z1`.

После формирования прямоугольной сетки `plot3d1` зальёт ячейки элементов со значением «бесконечность» белым цветом. Таким образом нам удастся создать эффект вырезания целых областей поверхности $Z = \sin(t) \cdot \cos(t)$ (см. листинг 4.60, рис. 4.90).

Листинг 4.60. 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);
```

Теперь поставим перед собой другую задачу: построить геометрическое тело, полое внутри.

Задача 4.49

Построить полую сферу.

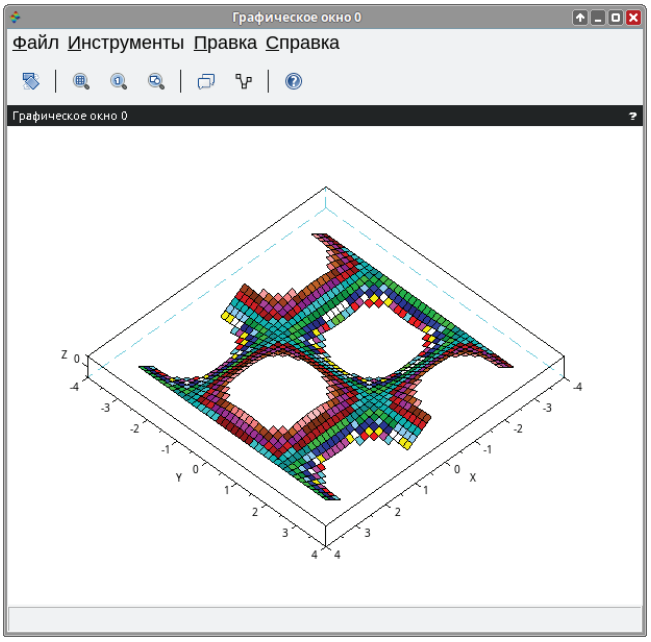

Рис. 4.90. Пример вырезания областей из трёхмерного графика

Создадим функцию `sph`, которая задаёт сферу тремя параметрическими уравнениями X, Y, Z:

```
function [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);
end;
```

Далее задаём 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 добьёмся того, что в графическое окно будет выведена сфера, имеющая две «шляпки», сверху и снизу (см. рис. 4.91). 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 4.61. Построение сферы с полостью внутри и «срезанными» полюсами

```
function [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);
end;
```

```
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);
```

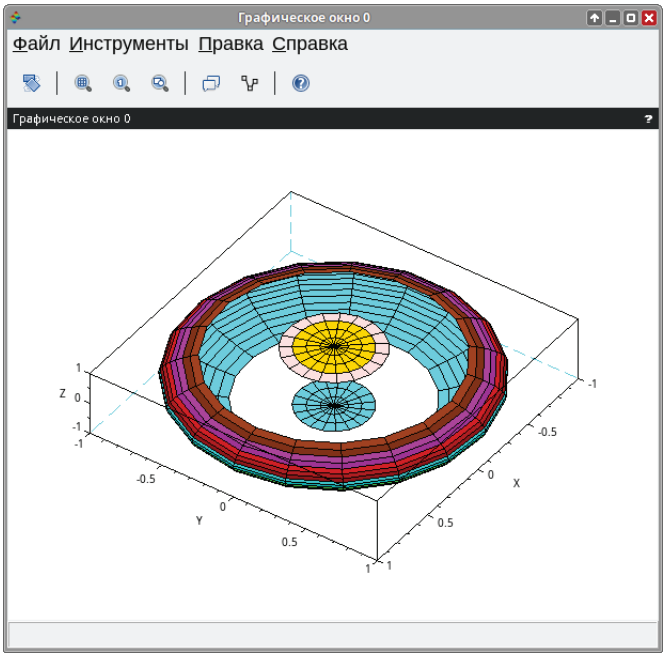

Рис. 4.91. График полой сферы

Ранее мы уже познакомились с возможностями функции `plot3d2`. Теперь рассмотрим ещё несколько примеров её применения.

Задача 4.50

Построить ракушкообразный график.

Такой график можно задать следующей системой уравнений:

$$\begin{cases} x = \cos(u)\cdot u\cdot\left(1+\dfrac{\cos(v)}{2}\right) \\ y = \dfrac{u}{2}\cdot\sin(v) \\ z = \sin(u)\cdot u\cdot\left(1+\dfrac{\cos(v)}{2}\right) \end{cases}.$$

Зададим массивы значений параметров u, v. Вычислим значения функций x, y, z (см. листинг 4.62). Обратившись к функции `plot3d2`, получим график, представленный на рис. 4.92.

Листинг 4.62. 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);
```

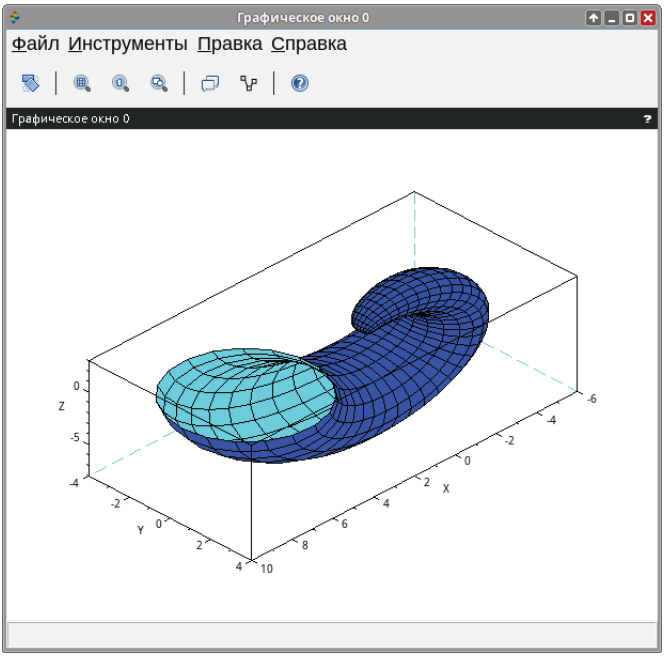

Рис. 4.92. Ракушкообразный график, построенный функцией `plot3d2`

Задача 4.51

Построить ленту Мёбиуса при помощи функции `plot3d2`.

Лента Мёбиуса – простейшая односторонняя поверхность с краем. Попасть из одной точки этой поверхности в любую другую можно, не пересекая края.

В общем параметрическом виде лента Мёбиуса может быть представлена системой уравнений:

$$\begin{cases} x(u,v) = \left(1 + \frac{v}{2} \cdot \cos\left(\frac{u}{2}\right)\right) \cdot \cos(u) \\ y(u,v) = \left(1 + \frac{v}{2} \cdot \cos\left(\frac{u}{2}\right)\right) \cdot \sin(u). \\ z(u,v) = \frac{v}{2} \cdot \sin\left(\frac{u}{2}\right) \end{cases}$$

Здесь u принадлежит интервалу $[0; 2\pi]$, а v – $[-1; 1]$. Эти формулы задают ленту Мёбиуса шириной 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 задаёт расстояние от края.

В листинге 4.63 предложен один из способов построения ленты Мёбиуса, а её график представлен на рис. 4.93.

Листинг 4.63. Построение ленты Мёбиуса 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);
```

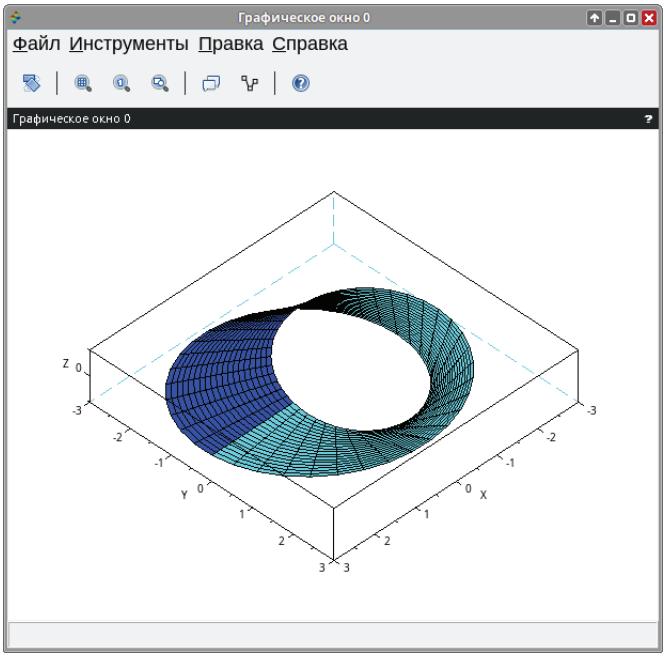

Рис. 4.93. Лента Мёбиуса

Задача 4.52
Построить тор с узкой и широкой сторонами при помощи функции `plot3d2`.

Тор – поверхность вращения в форме бублика, получаемая вращением окружности вокруг оси, лежащей в плоскости окружности и не пересекающей её. 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 $[0; 2\pi]$, R – расстояние от центра окружности до оси вращения, r – радиус окружности.

В листинге 4.64 приведен способ построения тора с узкой и широкой сторонами подобно ленте Мёбиуса, график тора изображен на рис. 4.94.

Листинг 4.64. Построение тора с узкой и широкой сторонами 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,);
```

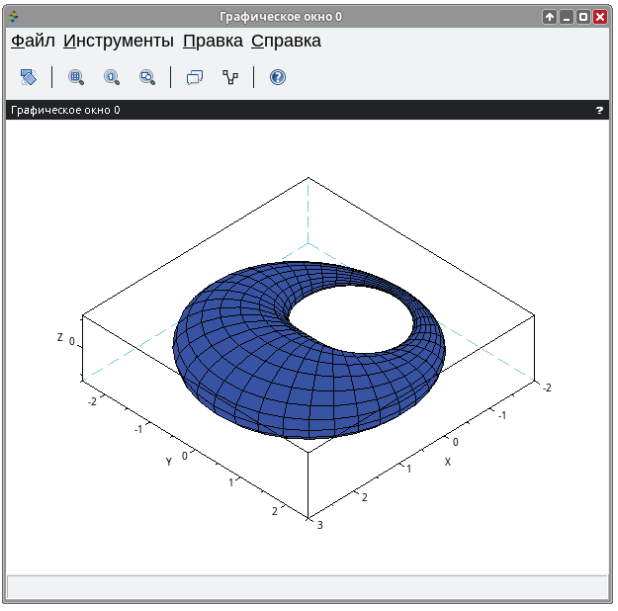

Рис. 4.94. Тор с узкой и широкой сторонами

Задача 4.53
Построить деформированный тор при помощи функции `plot3d2`.

Листинг 4.65 демонстрирует один из возможных способов построения вогнутого тора при помощи функции `plot3d2`, график тора представлен на рис. 4.95.

Листинг 4.65. 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);
```

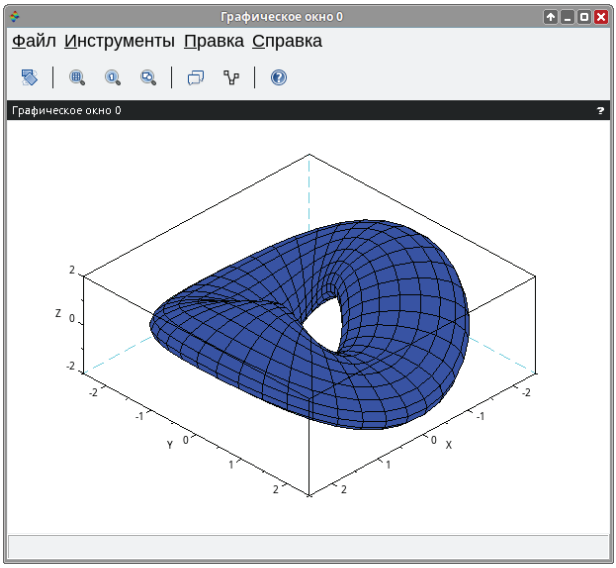

Рис. 4.95. Деформированный тор

4.20 Анимация

При изучении движения точки на плоскости Scilab позволит построить график движения и проследить за движением. Построить анимационный ролик можно с помощью функции comet(x, y), которая позволит увидеть движение точки вдоль кривой $y(x)$ на плоскости.

Для движения точки на плоскости вдоль синусоиды достаточно ввести команды:

```
x=0:%pi/30:6*%pi;
y=sin(x);
comet(x,y);
```

Процесс движения точки вдоль синусоиды представлен на рис. 4.96, окончательный вид траектории движения точки вдоль синусоиды показан на рис. 4.97.

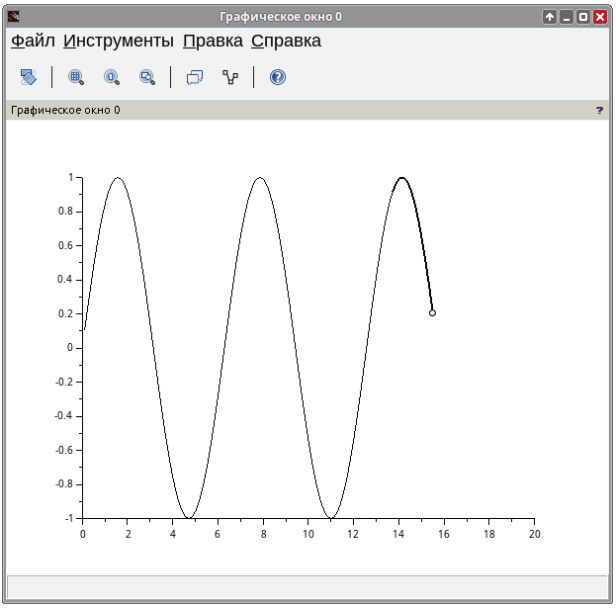

Рис. 4.96. Движение точки вдоль синусоиды

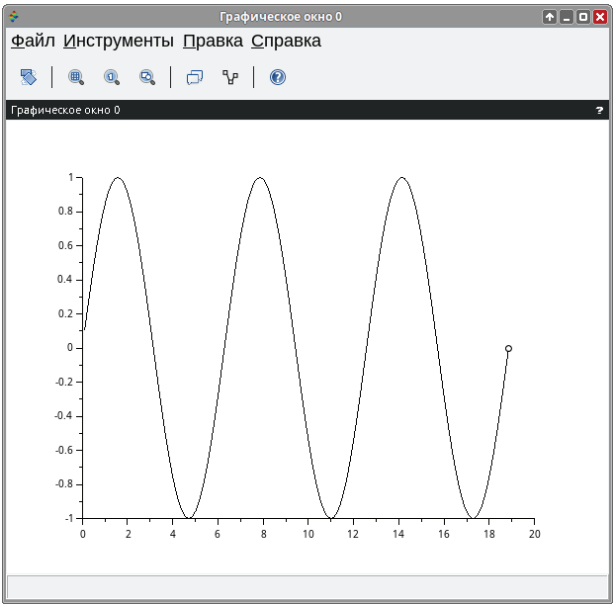

Рис. 4.97. Окончательный вид траектории движения точки

Глава 5

Создание графических приложений в среде Scilab

Scilab позволяет создавать не только обычные программы для автоматизации расчётов, но и визуальные приложения, которые будут запускаться в среде Scilab. Основным объектом в среде Scilab является графическое окно.

5.1 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 *Графическое окно n*, где *n* – это номер окна. По умолчанию первое окно получает имя *Графическое окно 0*, второе – *Графическое окно 1* и т. д. Объект созданного графического окна со всеми его свойствами записывается в переменную `F`.

Пример: выведем на экран содержимое переменной `F` после создания графического окна.

Листинг 5.1. Свойства объекта

```
-->F=figure();
-->F

//Результат работы программы

F  =

Handle of type "Figure" with properties:
========================================
children: "Axes"
figure_position = [200,200]
figure_size = [616,577]
axes_size = [610,460]
```

```
auto_resize = «on»
viewport = [0,0]
figure_name = "Графическое окно %d"
figure_id = 0
info_message = ""
color_map = matrix 33x3
pixel_drawing_mode = "copy"
anti_aliasing = «off»
immediate_drawing = «on»
background =  33
visible = «on»
rotation_style = "unary"
event_handler = ""
event_handler_enable = «off»
user_data = []
resizefcn = ""
closerequestfcn = ""
resize = «on»
toolbar = "figure"
toolbar_visible = «on»
menubar = "figure"
menubar_visible = «on»
infobar_visible = «on»
dockable = «on»
layout = "none"
layout_options = "OptNoLayout"
default_axes = «on»
icon = ""
tag = ""
```

Описание свойств объекта `F` представлено ниже.

- `children`: вектор дочерних элементов фигуры. Все дочерние элементы относятся к типу **Axes** (Оси). При создании объекта `figure` одновременно создается и объект `Axes`;
- `figure_position`: положение графического окна на экране в пикселях. Это вектор `[x,y]`, определяющий положение верхнего левого угла окна. Позиция `[0,0]` находится в верхнем левом углу экрана;
- `figure_size`: вектор [ширина, высота], задающий размер графического окна в пикселях. Операционная система определяет некоторые минимальные размеры графического окна. Если указать значения меньше, то фактически будут установлены минимальные;
- `axes_size`: вектор [ширина, высота] размеров всей графической области в окне в пикселях;
- `auto_resize`: автоматическое изменение размера дочерних осей при изменении размера окна. Принимает значения «on» и «off». Если при изменении размеров графического окна значение `auto_resize = on`, то `axes_size` обновляется и содержимое осей масштабируется так, чтобы полностью вмещаться в окно нового размера. Если при изменении размеров графического окна значение `auto_resize = off`, то `axes_size`

остаётся неизменным. Масштаб отображения дочерних осей не изменяется. Если размер окна становится меньше размеров осей, отображаются полосы прокрутки;

- `viewport`: положение видимой части графика;
- `figure_name`: название окна. Это имя представляет собой символьную строку, отображаемую в верхней части окна. Имя по умолчанию содержит выражение `%d`, которое заменяется на `figure_id`;
- `figure_id`: идентификатор рисунка. Это целое число, которое задаётся при создании объекта;
- `info_message`: текст, отображаемый в информационной строке графического окна;
- `color_map`: свойство, определяющее цветовую схему, используемую в этом окне. Цветовая карта представляет собой матрицу размером *m* на 3; *m* – это количество цветов. Номер цвета *i* задаётся в виде 3-кратных значений R, G, B, задающих соответственно интенсивности красного, зелёного и синего цветов в диапазоне от 0 до 1;
- `pixel_drawing_mode`: это поле определяет побитовую операцию, используемую для рендеринга пикселей. Режим по умолчанию – копирование;
- `anti_aliasing`: это свойство управляет уровнем сглаживания, используемым для улучшения качества графики. Если для свойства задано значение «off», сглаживание отключено. Чтобы включить сглаживание, для свойства должно быть установлено значение 2×, 4×, 8× или 16×. В данном случае это означает уровень сглаживания. Например, 16× – это более высокий уровень качества, чем 2×. Указание более высокого уровня сглаживания улучшает качество изображения, но снижает производительность графики;
- `immediate_drawing`: это свойство управляет отображением окна. Принимаются значения «on» (режим по умолчанию) и «off». Используется для задержки длинной последовательности графических команд (подразумевающих несколько рисунков или перерисовок);
- `background`: цвет фона окна. Принимает в качестве значения индекс относительно текущей цветовой карты;
- `visible`: свойство отвечает за то, будет ли видно окно на экране компьютера. Значение «off» скрывает окно, но оно всё равно существует, а значение «on» выводит окно на экран;
- `rotation_style`: свойство связано с поворотом трёхмерных графиков. Принимает значения «unary» – поворот одного графика и «multiple» – поворот всех графиков одновременно;
- `event_handler`: функция Scilab, которая предназначена для обработки событий. Установка пустой строки приводит к отключению обработчика событий;
- `event_handler_enable`: используется для включения или отключения обработчика событий. Принимает значения «on» и «off»;

- `user_data`: это свойство можно использовать для хранения и извлечения любой переменной Scilab в структуре данных `figure`;
- `resizefcn`: это поле может использоваться для хранения имени функции Scilab в виде символьной строки. Эта символьная строка будет вычисляться при создании и изменении размеров графического окна;
- `closerequestfcn`: это поле может использоваться для хранения имени функции Scilab в виде символьной строки. Эта символьная строка будет вычисляться всякий раз, когда пользователь попытается закрыть фигуру, используя крестик в правом верхнем углу;
- `resize`: «on» (значение по умолчанию) позволяет пользователю изменять размер рисунка с помощью мыши, «off» запрещает это делать. Но изменить размер по-прежнему можно с помощью свойства `figure_size`;
- `toolbar`: это свойство определяет тип панели инструментов в созданном окне. Если его значение «none», то в окне не будет никакой панели меню. Если его значение «figure», то будет создано окно с панелью инструментов для рисования по умолчанию. Это значение может быть установлено только при создании окна;
- `toolbar_visible`: это свойство управляет видимостью панели инструментов. Принимает значения «on» (значение по умолчанию) или «off»;
- `menubar`: это свойство определяет тип строки меню в окне. Если его значение равно «none», то в окне не будет никакой панели меню. Если его значение равно «figure», то будет создано окно со стандартной строкой меню. Это значение может быть установлено только при создании.
- `menubar_visible`: это свойство управляет видимостью панели меню рисунков. Принимает значения «on» (значение по умолчанию) или «off»;
- `infobar_visible`: это свойство управляет видимостью строки состояния рисунка. Принимает значения «on» (значение по умолчанию) или «off»;
- `dockable`: это свойство определяет, может ли созданное окно быть закреплено внутри среды Scilab. Если его значение равно «on», то в окне будет панель, позволяющая пользователю закрепить/отстыковать его. В противном случае окно будет выглядеть как стандартное окно операционной системы. Это значение может быть установлено только при создании;
- `layout`: это свойство задаёт макет, используемый для размещения дочерних элементов окна;
- `layout_options`: это свойство задаёт параметры макета, используемого для размещения дочерних элементов окна;
- `default_axes`: «on» (значение по умолчанию) означает, что окно по умолчанию всегда имеет оси: они автоматически устанавливаются при создании окна или автоматически сбрасываются, когда оси окна намеренно удаляются. «off» означает, что вновь созданная фигура не имеет осей по умолчанию и для существующей фигуры можно фактически удалить все её существующие оси;

- `icon`: это поле можно использовать для настройки значка окна Scilab. Его значение представляет собой символьную строку, содержащую (абсолютный или относительный) путь к файлу изображения, содержащему значок;
- `tag`: это свойство для хранения символьной строки, обычно используемой для идентификации элемента управления. В основном применяется в сочетании с `findobj()`.

Установить или изменить свойство объекта можно несколькими способами.

1. Непосредственно при создании графического окна. 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('figure_size',[300,300],'figure_position',[0,0]);
   ```

   в левом верхнем углу экрана будет создано окно шириной и высотой по 300 пикселей.
2. После создания графического окна с помощью функции

   ```
   set(f,'Свойство1', 'Значение1', 'Свойство2', 'Значение2', ...,
   'Свойствоn', 'Значениеn')
   ```

   устанавливается значение параметров. Здесь `f` – графическое окно, `'Свойство'` – имя параметра, `'Значение'` – его значение.
   Например, с помощью команд

   ```
   set(f,'figure_size',[400,500],'info_message','Это сообщение в графическом
   окне');
   ```

   можно задать размер окна и информационное сообщение.
3. Так как `f` – это объект, то к его свойствам можно обратиться напрямую, не используя функцию `set`.
   Например, вышеприведённые команды эквивалентны следующим:

   ```
   f.figure_size = [400,500];
   f.info_message = 'Это сообщение в графическом окне';
   ```

Получить значение свойства объекта также можно несколькими способами.

---

[1] Значение `i` будет использоваться в кавычках, если значением параметра является строка, если же значением параметра является число, то кавычки использовать не надо.

1. После создания графического окна с помощью функции

   ```
   get(f,'Свойство')
   ```

   Здесь `f` – графическое окно, `'Свойство'` – имя параметра.
   Например, с помощью команд

   ```
   get(f,'figure_size'); get(f,'info_message');
   ```

   можно получить данные о размерах окна и информационное сообщение.
2. Так как `f` – это объект, то к его свойствам можно обратиться напрямую, не используя функцию `get`.
   Например, вышеприведённые команды эквивалентны следующим:

   ```
   f.figure_size f.info_message
   ```

3. Результат работы функции `get`, как и результат обращения к свойству объекта, можно записать в переменную:

   ```
   fs=get(f,'figure_size') im=f.info_message
   ```

Задача 5.1

Создать окно с заголовком **Первое окно**, задать этому окну зелёный цвет фона. Записать заголовок в переменную `title`, а цвет фона – в `color`.

Решение приведено в листинге 5.2.

Листинг 5.2. Решение задачи 5.1

```
Вариант 1
f=figure('background',3);
f.figure_name='Первое окно';
title=f.figure_name
color=f.background

Вариант 2
f=figure('figure_name','Первое окно','background',3);
title=f.figure_name
color=f.background

Вариант 3 f=figure('background',3);
set(f,'figure_name','Первое окно');
title=get(f,'figure_name')
color=get(f,'background')
```

Графическое окно можно закрыть с помощью функции `close(f)`, где `f` – графическое окно.

Удаляется окно с помощью функции `delete(f)`, где `f` – графическое окно.

5.2 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.), а их свойствам – присваиваться соответствующие значения.

Для создания любого интерфейсного компонента с заданными свойствами используется функция uicontrol. Обращение к ней имеет вид:

```
C=uicontrol(F, 'Style', 'тип_компонента', 'Свойство_1', Значение_1,
'Свойство_2', Значение_2,..., 'Свойство_k',Значение_k);
```

Здесь:

- `C` – объект создаваемого компонента;
- `F` – объект, внутри которого будет создаваться компонент (чаще всего этим компонентом будет окно);
- `'Style'` – служебная строка `Style` указывает на стиль создаваемого компонента;
- `'тип_компонента'` – определяет, к какому классу принадлежит создаваемый компонент. Возможные значения:
  - `Checkbox`: кнопка с двумя состояниями (используется для нескольких независимых вариантов выбора);
  - `Edit`: редактируемая строковая зона;
  - `Frame`: контейнер для других элементов `uicontrols` или осей;
  - `Image`: компонент, в котором отображается указанное изображение;
  - `Layer`: контейнер для элементов управления объектами типа `frame`, позволяющий переключаться между ними программно с помощью свойства `value`;
  - `Listbox`: список элементов, которые можно прокручивать. Элементы можно выбирать с помощью мыши;
  - `Popupmenu`: кнопка, при нажатии на которую появляется меню;
  - `Pushbutton` (стиль по умолчанию): прямоугольная кнопка, обычно используемая для выполнения обратного вызова;
  - `Radiobutton`: кнопка с двумя состояниями. В отличие от `Checkbox` предполагает выбор только одного варианта из списка;
  - `Slider`: ползунок, используемый для установки значения в заданном интервале с помощью мыши;
  - `Spinner`: компонент, который позволяет пользователю выбирать/редактировать значение между границами с фиксированным шагом;
  - `Tab`: контейнер для элементов управления объектами типа `frame`, позволяющий переключаться между ними, щелкая по вкладке с заданным названием и/или значком;

- `Table`: редактируемая таблица;
- `Text`: текстовый элемент управления (обычно статический);

- `'Свойство_k'`, `Значение_k` – определяют свойства и значения отдельных компонентов. Список свойств объектов `uicontrol`:
  - `BackgroundColor`: цвет фона объекта. Задаётся в виде массива из трёх значений: уровень красного в итоговом цвете, зелёного и синего. Это вещественные числа в диапазоне от 0 до 1. Цвет может быть задан как в виде вектора [R, G, B], так и в виде строки, где каждое значение разделяется символом «|», т. е. «R|G|B». Установка этого свойства в [−1 −1 −1] позволит внешнему виду задавать цвет фона по умолчанию;
  - `Border`: объект границы. Он используется только для `frame`. При установке этого значения на [] граница рамки удаляется;
  - `Callback`: строка с инструкциями Scilab, которая выполняется при активации объекта (например, при нажатии на кнопку). Это свойство не используется для `layer`, `frame` и `text`;
  - `Callback_Type`: тип обратного вызова, передаваемого в `uicontrol`:

    –1 – нет (обратный вызов деактивирован);

    0 (по умолчанию) – пользовательские инструкции Scilab, записанные в виде строки в свойстве Callback;

    1 – функции Fortran;
  - `Constraints`: объект создается с помощью `createConstraints` и описывает ограничения на положение и размер объекта относительно его родительского элемента. Установка этого значения на [] снимает ограничения;
  - `Enable`: если для этого свойства установлено значение «on» (по умолчанию), объект будет реагировать на действия мыши, если установлено «off» – нет (объект будет выделен серым цветом);
  - `FontAngle`: применяется для элементов, содержащих какой-либо текст. Это свойство задаёт наклон шрифта. Есть возможность сделать шрифт обычным, курсивным или наклонным. Значения «normal», «italic» и «oblique» соответственно;
  - `FontSize`: свойство задаёт размер шрифта текста. Принимает скалярное значение;
  - `FontUnits`: в этом свойстве можно указать единицы измерения, с помощью которых задаётся размер шрифта: «points», «pixels», «normalized»;
  - `FontWeight`: свойство задаёт насыщенность шрифта. Можно указать значения: «light», «normal», «demi», «bold»;
  - `FontName`: строка, в которой указывается название шрифта;
  - `ForegroundColor`: цвет текста объекта. Задаётся в виде массива из трёх значений: уровень красного в итоговом цвете, зелёного и синего. Это вещественные числа в диапазоне от 0 до 1. Цвет может быть

задан как в виде вектора [R, G, B], так и в виде строки, где каждое значение разделяется символом «|», т. е. «R|G|B»;

- `Groupname`: это свойство, используемое для типов `radiobutton` и `checkbox`. Позволяет управлять всеми объектами одной и той же группы. Например, для `checkbox` будет осуществляться автоматическая проверка уникальности выбора;
- `HorizontalAlignment`: устанавливает выравнивание текста по горизонтали. Возможные значения: «left», «center», «right»;
- `Icon`: это свойство представляет собой строку, содержащую относительный (к рабочему каталогу Scilab) или абсолютный путь к файлу изображения, который будет использоваться в качестве значка для `pushbutton` или `text`;
- `Layout`: это свойство задаёт шаблон, используемый для удаления дочерних элементов `frame`;
- `Layout_options`: это свойство задаёт параметры шаблона, используемого для удаления дочерних элементов `frame`;
- `ListboxTop`: это свойство используется только для `listbox`. Оно указывает, какой элемент списка отображается в первой строке видимой области списка. В качестве значения принимает номер элемента списка;
- `Margins`: вектор [top, left, bottom, right]. Свойство задаёт пустое пространство вокруг элемента в пикселях;
- `Max`: указывает наибольшее значение, которое может принимать свойство «value». Однако оно имеет разное значение для каждого объекта:
  - **checkbox** и **radiobutton:** значение `max` – это значение, которое принимает свойство «value» при установке флажка;
  - **slider** и **spinner:** максимальное значение, которое можно выбрать;
  - **listbox:** если `(max-min)>1`, список допускает множественный выбор (**Ctrl** + клик левой кнопкой мыши по пункту списка);
- `Min`: указывает наименьшее значение, на которое может быть установлено свойство «value». Однако оно имеет разное значение для каждого объекта:
  - **checkbox** и **radiobutton:** `min` – это значение, которое принимает свойство «value», когда контроль снят;
  - **slider** и **spinner:** минимальное значение компонента;
  - **listbox:** если `(max-min)>1`, список допускает множественный выбор;
- `Parent`: дескриптор родительского элемента `uicontrol`. Изменение этого свойства позволяет перемещать элемент с одного графического окна на другое;
- `Position`: [x y w h] числовой вектор или строка «x|y|w|h». Задаёт или получает положение и размер элемента:
  - `x` – горизонтальное положение левой части компонента по отношению к левому краю ссылочного элемента;

- y – вертикальное положение нижней стороны компонента по отношению к нижнему краю ссылочного элемента;
- w и h – это ширина и высота объекта.

x, y, w, h также могут быть заданы с помощью отдельной строки «x|y|w|h». В качестве разделителя используется «|». Единица измерения устанавливается с помощью свойства Units.

Для ползунков: w > h устанавливает ползунок по горизонтали, в противном случае – по вертикали;

– Relief: внешний вид границы объекта. Принимает значения «default», «flat», «groove», «raised», «ridge», «solid», «sunken»;
– Scrollable: свойство, используемое для типов frame и edit, указывает, должен элемент иметь возможность прокрутки («on») или нет («off»). Для frame это значение необходимо задать при создании элемента;
– SliderStep: [1,2] вещественный вектор для работы с масштабом:
  - 1 – размер маленького шага. Срабатывает при нажатии на ползунок или при нажатии на клавиши со стрелками (когда ползунок находится в фокусе);
  - 2 – размер большого шага. Срабатывает при нажатии **Ctrl** + нажатие стрелки клавиатуры. Если большой шаг опущен, по умолчанию он равен 1/10 от масштаба;
– String: текст, отображаемый в объекте (за исключением Frame, Slider и Spinner):
  - для таблиц значением является строковая матрица;
  - для списков и всплывающего меню – вектор строк или одна строка, элементы которой разделены символом «|»;
  - для текстовых элементов эта строка может содержать HTML-код для форматирования текста;
  - для pushbutton или text, если текст заключен между двумя $ (знак доллара), он будет рассматриваться как выражение LaTeX, а если текст заключен между < и >, то он будет рассматриваться как выражение MathML;
  - для layer и tab значение указывает тег выбранного дочернего элемента;
  - для image значение указывает путь к файлу изображения;
  - для table значение определяет данные таблицы:

```
[IGNORED COL1-HEADER...COLN-HEADER;
ROW1-HEADER, ROW1COL1-DATA, ... , ROW1COLN-DATA;
...
ROWM-HEADER, ROWMCOL1-DATA, ... , ROWMCOLN-DATA].
```

– Tag: это свойство обычно используется для идентификации элемента управления. Оно позволяет присвоить ему «имя». В основном используется в сочетании с findobj();

- `Title_position`: расположение вкладок `tab`. Возможные значения: «top», «left», «bottom», «right»;
- `Title_scroll`: это свойство указывает, должны вкладки `tab` иметь прокрутку («on») или нет («off»);
- `TooltipString`: это свойство представляет текст всплывающей подсказки около элемента, появляющийся при наведении курсора мыши на него;
- `Units`: это свойство задаёт единицы измерения, используемые для свойства «position». Нормализованные позиции и размеры указаны в пределах от 0 до 1. Возможные значения: «points», «pixels», «normalized»;
- `Userdata`: привязка некоторых объектов Scilab (string, string matrix, matrix mxn) к `uicontrol`;
- `Value`: скаляр или вектор. Значение объекта. Точное значение зависит от типа объекта:
  - **checkbox** и **radiobutton**: значение устанавливается равным `max` (см. выше) при включении и `min` при выключении флага;
  - **listbox** и **popupmenu**: значение представляет собой вектор индексов, соответствующих индексам выбранных записей в списке. 1 – первый элемент списка;
  - **slider** и **spinner**: числовое значение, указываемое элементом;
  - **layer** и **tab**: индекс отображаемого элемента;
  - **image**: установка некоторых свойств изображения [X-Scale Y-Scale X-Shear Y-Shear RotationAngle];
- `Verticalalignment`: выравнивание текста по вертикали. Это свойство действует только с объектами типа Text и CheckBox. Возможные значения: «top», «middle», «bottom»;
- `Visible`: видимость объекта. Если для этого свойства установлено значение «on» (по умолчанию), объект виден, но если значение «off», объект не будет отображаться в родительском окне.

5.2.1 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.

Листинг 5.3. Создание окна с кнопкой

```
//Создаем окно
d=figure();
//Создаем кнопку, устанавливая свойство Style.
dbt=uicontrol(d,'Style','pushbutton');
```

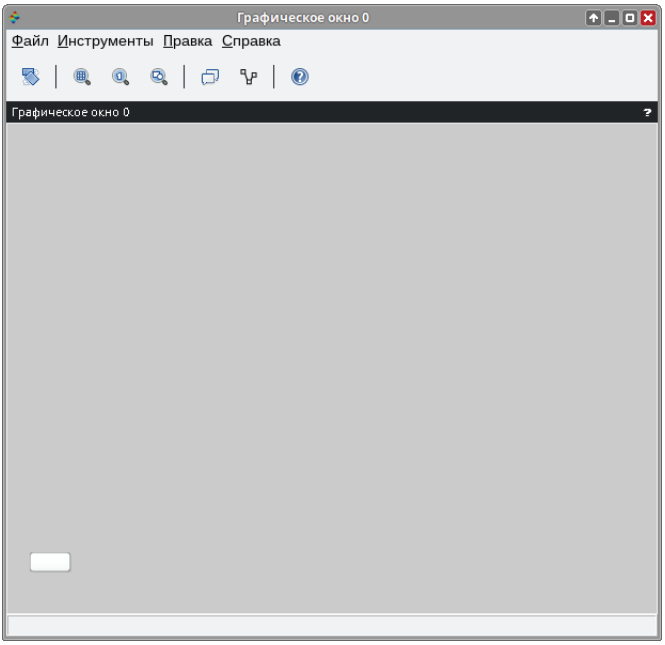

Рис. 5.1. Кнопка в окне

При щелчке по кнопке вокруг её надписи появляется синяя рамка, свидетельствующая о том, что кнопка «находится в фокусе». Модифицируем программу создания кнопки, задав дополнительно значения некоторых свойств:

- месторасположение и заголовок окна;
- надпись на кнопке;
- месторасположение кнопки.

Текст программы приведен в листинге 5.4, а на рис. 5.2 можно увидеть окно, которое получилось в результате работы этой программы.

Листинг 5.4. 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]);
```

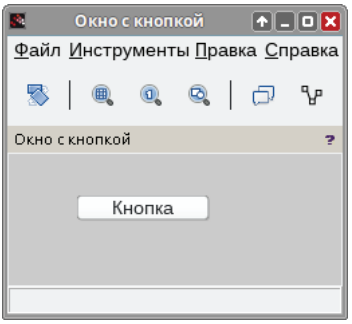

Рис. 5.2. Окно с кнопкой

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 5.5). После запуска этой программы появится окно, представленное на рис. 5.3, при щелчке по кнопке **Button** вызывается обработчик события – функция `gr_sin`, в результате появляется окно, изображенное на рис. 5.4.

Листинг 5.5. Пример кнопки с обработчиком события `CallBack`

```
f=figure();
set(f,'position',[0,0,400,400])
set(f,'figure_name','График');

//Создаем кнопку, которая при щелчке по ней мышкой вызывает
//функцию gr_sin.
Button=uicontrol('style','pushbutton','string','Button',...
'position',[0,0,100,20],'CallBack','gr_sin');
function y=gr_sin()
x=-5:0.2:5;
y=sin(x);
plot(x,y);
xgrid();
endfunction
```

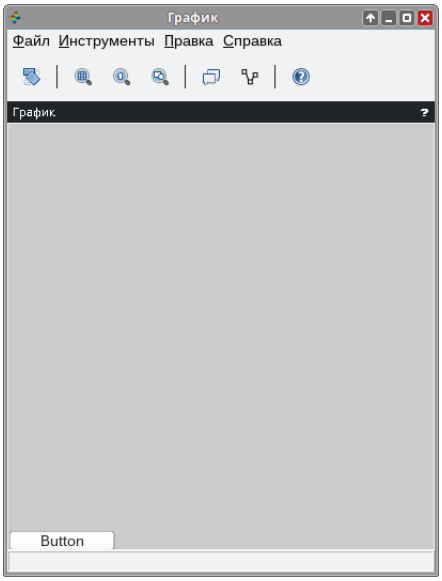

Рис. 5.3. Окно программы

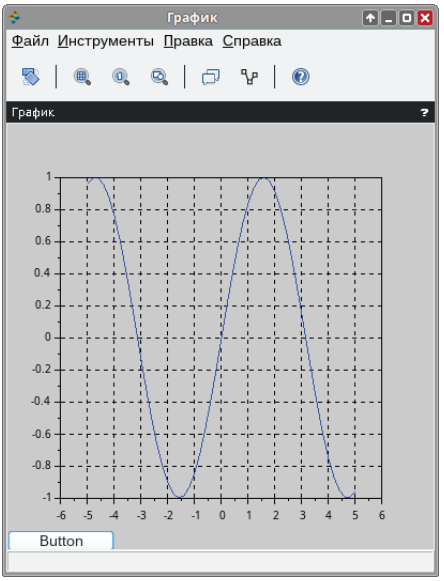

Рис. 5.4. График функции $y = \sin(x)$

5.2.2 Метка

Следующим наиболее часто используемым компонентом является метка – текстовое поле для отображения символьной информации. Для определения метки значения параметра `'Style'` в функции `uicontrol` должно быть значение `'text'`. Компонент предназначен для вывода символьной строки (или нескольких строк). Выводимый на метку текст – значение параметра `'String'` – может быть изменён только программно.

Рассмотрим пример создания текстового поля (метки) с помощью функции `uicontrol` (см. листинг 5.6 и рис. 5.5).

Листинг 5.6. Создание метки

```
f=figure();
uicontrol('Style','text','Position',[10,130,150,20],...
'String', 'Метка');
```

Одним из основных свойств метки является горизонтальное выравнивание текста, которое определяется свойством `HorizontalAlignment`. В качестве примера рассмотрим окно, содержащее 4 текстовых поля с разными значениями свойства `HorizontalAlignment`. Текст программы представлен в листинге 5.7, а окно с четырьмя метками – на рис. 5.6.

Листинг 5.7. Создание нескольких меток

```
hFig=figure('figure_size',[300 300]);
hSt1=uicontrol('Style','text','Position',[30,30,150,20], ...
'String', 'Метка 1');
hSt1.BackgroundColor=[1 1 1]);
```

```
hSt1.HorizontalAlignment='left';
hSt2=uicontrol('Style', 'text', 'Position', [30,60,150,20],...
'HorizontalAlignment', 'center', 'BackgroundColor', [1 1 1],...
'String', 'Метка 2');
hSt3=uicontrol('Style','text','Position',[30,90,150,20],...
'HorizontalAlignment','right','BackgroundColor',[1 1 1],...
'String','Метка 3');
hSt4=uicontrol('Style','text','Position',[30,120,150,20],...
'BackgroundColor',[1 1 1],'String','Метка 4');
```

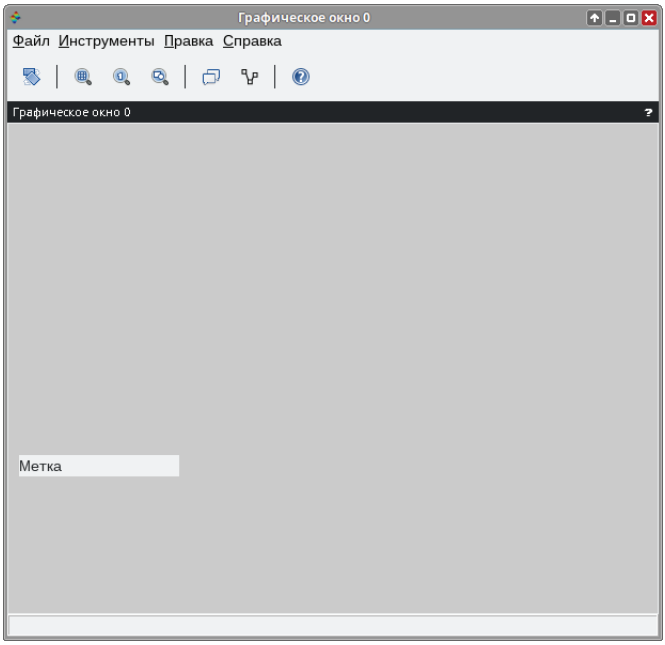

Рис. 5.5. Окно с меткой

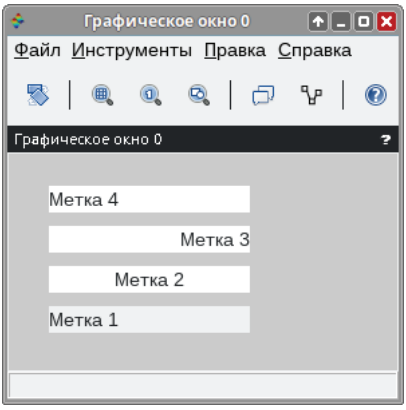

Рис. 5.6. Окно с несколькими метками

5.2.3 Переключатель и флажок

Рассмотрим ещё два компонента – переключатель и флажок, которые позволяют переключаться между состояниями кнопок или выбирать значения из списка.

У флажка свойство `'Style'` принимает значение `'checkbox'`, у переключателя – `'radiobutton'`.

Оба объекта создаются с помощью функции `uicontrol`. Пример создания переключателя представлен в листинге 5.8 и на рис. 5.7.

Листинг 5.8. Создание переключателя

```
hFig=figure('figure_size',[300 300]);
R=uicontrol('Style','radiobutton','String','имя',...
'value',1,'position', [25,50,70,30]);
```

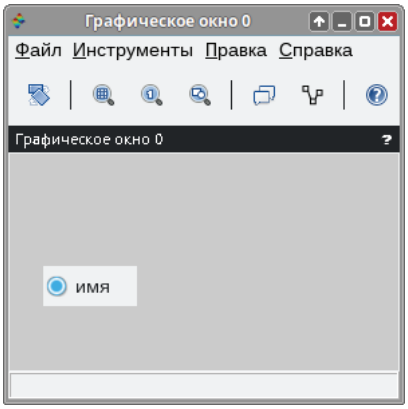

Рис. 5.7. Окно с переключателем

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`:

```
set(R,'value',0)
```

или напрямую обратиться к свойству объекта:

```
R.value=0
```

Получить значение свойства `'value'` можно с помощью функции `get`.

Переключатель может реагировать на событие `'CallBack'` и вызывать на выполнение определённую функцию. В этом случае создать кнопку можно с помощью вызова следующей функции `uicontrol`:

```
r1=uicontrol('Style','radiobutton','String','sin(x)','value',
0,'CallBack','F1');
```

Здесь F1 – имя функции, которая будет вызываться при щелчке по переключателю (при этом автоматически происходит смена его состояния).

Пример: написать программу, в которой с помощью переключателя можно выбрать функцию. Её график будет нарисован при щелчке по кнопке **Plot**. Следует помнить, что переключатели обычно используются для выбора одного варианта из предложенных. Для этого присвоим свойству 'groupname' обоих элементов одинаковые значения, объединив переключатели в одну группу (см. листинг 5.9 и рис. 5.8).

Листинг 5.9. Пример работы с переключателями

```
//Создаем графическое окно.
hFig=figure('Position',[50,50,800,800]);

//Создаем переключатели
hRb1=uicontrol('Style','radiobutton','String','sin(x)',...
'value',1, 'Position',[120,40,80,20],'groupname','plotFunction');
hRb2=uicontrol('Style','radiobutton','String','cos(x)',...
'value',1, 'Position',[25,40,80,20],'groupname','plotFunction');

//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',[25,10,80,20],'CallBack','Radio');

//Создаем кнопку с именем Close, которая с помощью обработчика
//Final закрывает окно.
Button1=uicontrol('style','pushbutton','string','Close',...
'position',[120,10,80,20],'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
```

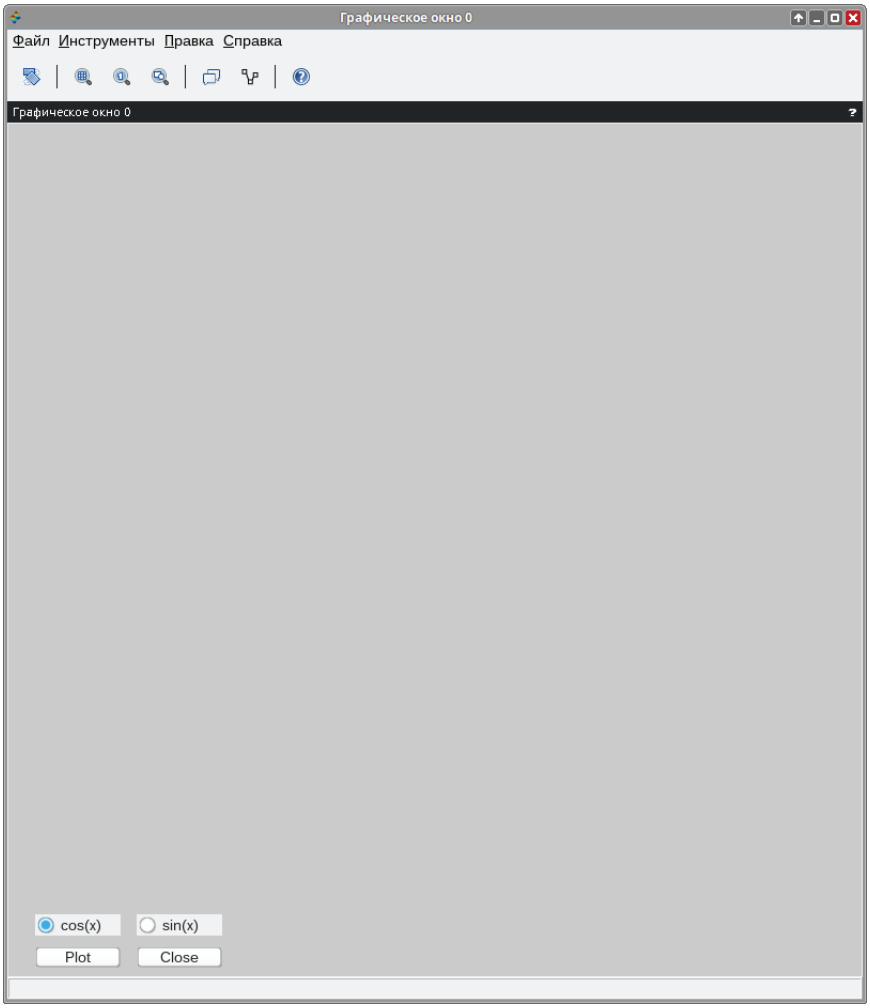

Рис. 5.8. Окно приложения

На рис. 5.8 представлено окно приложения. Кнопка **Close** закрывает его. Изменение состояния переключателей происходит автоматически при щелчке по ним.

Элемент флажок используется для множественного выбора. Генерация события `'CallBack'` и автоматическое выделение кнопки происходят при щелчке по квадратику или сопровождающей его надписи. Если флажок включён, то значение свойства `'value'` равно 1. Щелчок по флажку автоматически изменяет состояние на противоположное. Использование флажка аналогично переключателю.

Листинг 5.10. Пример работы с флажками

```
//Создание флажков вместо переключателей
//Остальной код не отличается от предыдущего
```

```
hRb1=uicontrol('Style','checkbox','String','sin(x)',...
'value',1, 'Position',[120,40,80,20]);
hRb2=uicontrol('Style','checkbox','String','cos(x)',...
'value',1, 'Position',[25,40,80,20]);
```

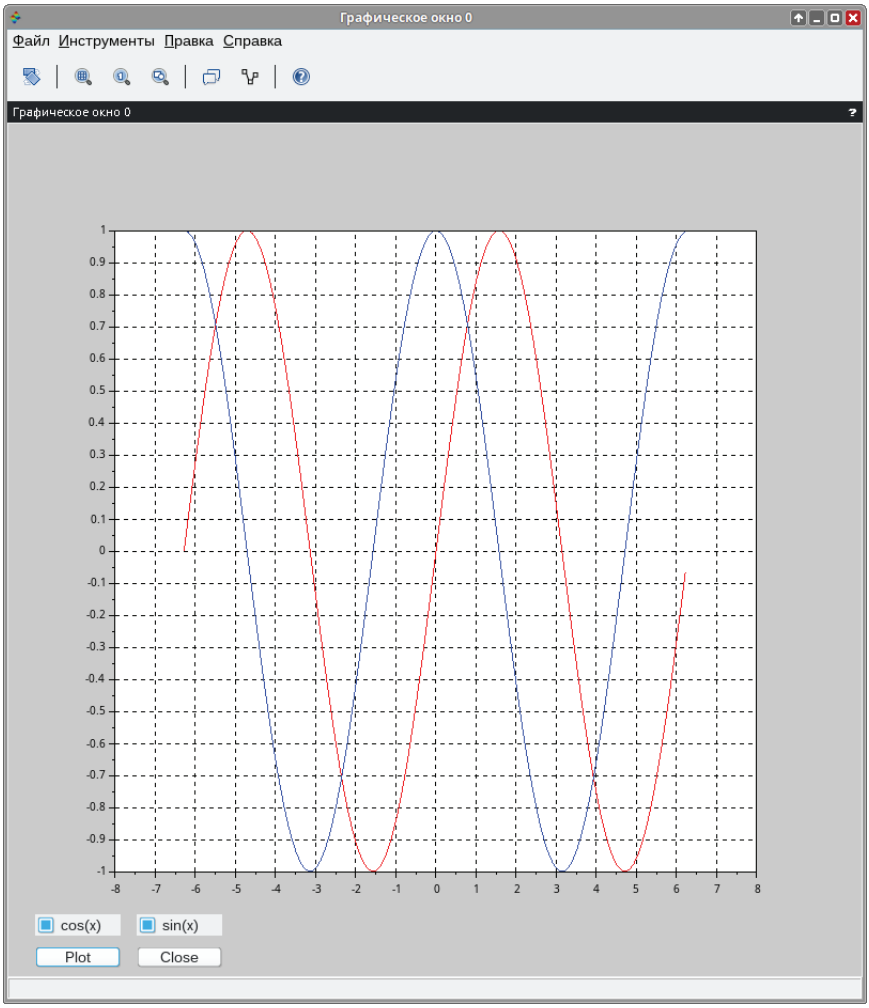

Рис. 5.9. Построение графиков обеих функций

5.2.4 Окно редактирования

Интерфейсный элемент «окно редактирования» (у этого 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. Изменить значение этого свойства можно с помощью функции `set`, а посчитать 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 5.2

Написать программу решения квадратного или биквадратного уравнения. Выбор типа уравнения будем проводить с помощью компонента «Переключатель».

Программа с комментариями представлена в листинге 5.11.

Листинг 5.11. Решение квадратного или биквадратного уравнения

```
f=figure(); //Создание графического объекта.

//Устанавливаем размер окна.
set(f,'position',[500,300,500,300])

//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',... [0, 250, 100, 20]);
//Подпись B=
lab_b=uicontrol(f,'style','text','string','B=','position',... [100, 250, 100, 20]);
//Подпись C=
lab_c=uicontrol(f,'style','text','string','C=','position',... [200, 250, 100, 20]);

//Поле редактирования для ввода коэффициента a.
edit_a=uicontrol(f,'style','edit','string','1','position',... [0, 230, 100, 20]);
//Поле редактирования для ввода коэффициента b.
edit_b=uicontrol(f,'style','edit','string','2','position',... [100, 230, 100, 20]);
//Поле редактирования для ввода коэффициента c.
edit_c=uicontrol(f,'style','edit','string','1','position',... [200, 230, 100, 20]);

//Флажок, отвечающий за выбор типа уравнения.
radio_bikv=uicontrol('style','radiobutton','string',...
'Это биквадратное уравнение?', 'value',1,'position',... [0,200,300,20]);
```

```
//Кнопки для решения уравнения и закрытия окна
BtSolve=uicontrol('style','pushbutton','string','Решить',...
'CallBack', 'Solve','position',[0,170,120,20]);
BtClose=uicontrol('style','pushbutton','string','Закрыть',...
'CallBack', '_Close','position',[150,170,120,20]);

//Текстовое поле, определяющее вывод результатов.
textresult=uicontrol(f,'style','text','string','','position',...[0, 100, 500, 20]);

//Функция решения уравнения.
function Solve()
  //Считываем значения переменных из текстовых полей и
  //преобразовываем их к числовому типу.
  a=evstr(get(edit_a,'string'));
  b=evstr(get(edit_b,'string'));
  c=evstr(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;
      //sprintf - функция для вывода информации в текстовое поле
      set(textresult,'string',...
sprintf("2 корня квадратного уравнения: x1=%1.2f x2=%1.2f",x1,x2));
    end;
  //если флажок включён,
  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: x1=%1.2f x2=%1.2f x3=%1.2f
x4=%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("2 корня биквадратного уравнения: x1=%1.2f x2=%1.2f",x1,x2));
```

```
      end;
    end;
  end
endfunction

// Функция закрытия окна.
function _Close()
  close(f)
endfunction
```

На рис. 5.10 представлено окно программы.

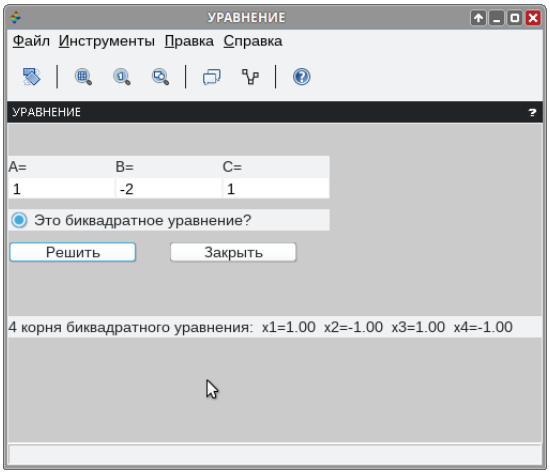

Рис. 5.10. Окно программы решения уравнения

5.2.5 Списки

Интерфейсный компонент «список» 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 5.12 и рис. 5.11). По умолчанию можно выбрать только один пункт списка. Чтобы разрешить выбор нескольких пунктов, свойствам `min` и `max` нужно присвоить числовые значения, различающиеся больше чем на 1. Например, `min=1` и `max=4`.

Листинг 5.12. Создание списка

```
//Создание графического окна.
f=figure();
//Создание listbox
h=uicontrol(f,'style','listbox','position', [10 10 150 160]);
//Разрешение на выбор нескольких
```

```
//Заполнение списка.
set(h, 'string', "строка 1|строка 2|строка 3");
h.min=1;
h.max=4;
```

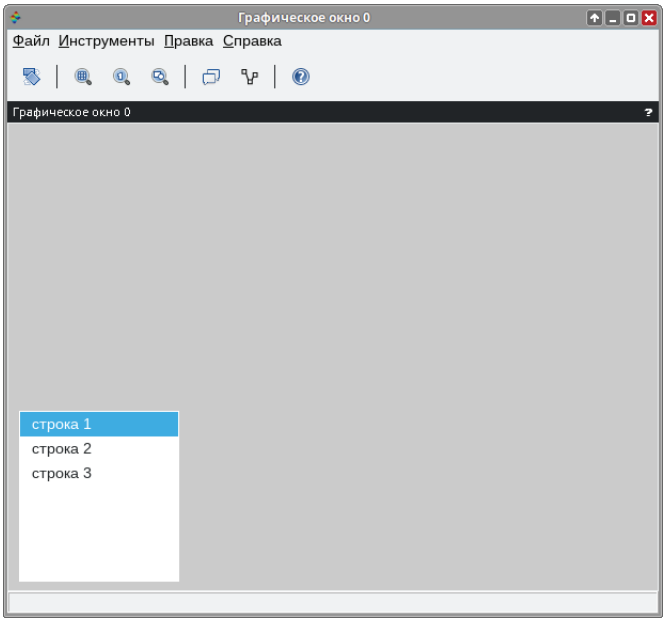

Рис. 5.11. Список с выделенными элементами

Список позволяет пользователю выбрать строку(и) из списка 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 её номер заносится в свойство `'value'` и генерируется событие `'CallBack'`. Строки в списке нумеруются от 1. Для выбора нескольких строк нужно зажать клавишу **Ctrl** и щелкать мышью по нужным строкам. При этом каждая выделяемая строка подсвечивается, а её номер запоминается в векторе `'value'`. Для выбора группы подряд идущих строк можно нажать и удерживать клавишу **Shift**, а затем щелкнуть по первой и последней строкам группы. Все промежуточные строки тоже будут выделены, и все их номера запомнятся в векторе `'value'`.

5.2.6 Таблицы

Интерфейсный компонент «таблица» можно рассматривать как окно с матрицей строк в нем. Если высота таблицы больше, чем высота окна, то для перемещения по таблице может использоваться вертикальная полоса прокрутки, которая генерируется автоматически.

Создание таблицы производится с помощью функции `uicontrol` при задании параметра `'Style'` – `'table'`. Также основным свойством здесь явля-

ется string, значение которого – массив данных таблицы. Рассмотрим это на простом примере.

Задача 5.3

Написать калькулятор для выполнения базовых математических операций над матрицей и числом. Поэлементное сложение, вычитание, умножение и деление элементов матрицы на заданное число.

(См. листинг 5.13 и рис. 5.12.)

Листинг 5.13. Создание списка

```
//Функция для выполнения мат. операций
function calc()
  //Считывание матрицы из введённых данных
  data=table.string;

  //Напоминание о корректности ввода данных
  set(f,'info\_message','Значениями могут быть ...
   только вещественные числа с разделителем точка');

  //Если выбран пункт "сложение",
  if radioSum.value==1 then
    //считываем коэффициент
    a=evstr(editSum.string);
    //Цикл по строкам
    for i=2:size(data,'r')
      //Цикл по столбцам
      for j=2:size(data,'c')
      //Перезапись значения на текущее + коэффициент
      data(i,j)=string(evstr(data(i,j))+a);
    end;
  end;
end;

//Если выбран пункт "вычитание"
//Действия аналогичны первому блоку
if radioSub.value==1 then
  a=evstr(editSub.string);
  for i=2:size(data,'r')
    for j=2:size(data,'c')
    //Перезапись значения на текущее - коэффициент
      data(i,j)=string(evstr(data(i,j))-a);
    end;
  end;
end;

//Если выбран пункт "умножение"
//Действия аналогичны первому блоку
if radioProd.value==1 then
  a=evstr(editProd.string);
```

```
  for i=2:size(data,'r')
    for j=2:size(data,'c')
    //Перезапись значения на текущее * на коэффициент
      data(i,j)=string(evstr(data(i,j))*a);
    end;
  end;
end;

//Если выбран пункт "деление"
//Действия аналогичны первому блоку
if radioDiv.value==1 then
  a=evstr(editDiv.string);
    //Проверка деления на 0
    if a==0 then
      set(f,'info_message','Деление на 0!');
    end
    for i=2:size(data,'r')
      for j=2:size(data,'c')
      //Перезапись значения на текущее / на коэффициент
        data(i,j)=string(evstr(data(i,j))/a);
      end;
    end;
end;

//Вывод результата в ячейки таблицы
set(table,'string',data);
endfunction

function del()
//Переприсвоение ячейкам таблицы исходных значений
set(table,'string',['','c1','c2','c3','c4'; 'r1','0','0','0','0'; ...
 'r2','0','0','0','0'; 'r3','0','0','0','0'; 'r4','0','0','0','0';]);
endfunction

//Начальное присвоение ячейкам таблицы исходных значений
dataTable=['','c1','c2','c3','c4'; 'r1','0','0','0','0'; ...
'r2','0','0','0','0'; 'r3','0','0','0','0'; 'r4','0','0','0','0';]

//Создание окна
f = figure('figure_size', [670 500],'figure_position', ...
 [1000 0],'background',8,'resize','off');

//Создание таблицы с начальными данными
table = uicontrol("style","table","string",dataTable,...
"position",[5 185 330 105]);

//Создание кнопок очистки и расчёта
btnCalc=uicontrol("style","pushbutton","string",'Рассчитать',...
"position",[5 20 150 30],'callback','calc');
btnCler=uicontrol("style","pushbutton","string",'Очистить',...
"position",[170 20 150 30],'callback','del');

//Создание переключателей для выбора операции
radioSum=uicontrol('style','radiobutton','string','Прибавить ',...
'position',[350 260 150 30],'groupname','math','value',1);
```

```
radioSub=uicontrol('style','radiobutton','string','Вычесть ',...
'position',[350 230 150 30],'groupname','math');
radioProd=uicontrol('style','radiobutton','string','Умножить на ',...
'position',[350 200 150 30],'groupname','math');
radioDiv=uicontrol('style','radiobutton','string','Разделить на ',...
'position',[350 170 150 30],'groupname','math');


//Создание редактируемых полей
//Для ввода коэффициентов
editSum=uicontrol('style','edit','string','2 ',...
'position',[500 260 50 30],'groupname','math');
editSub=uicontrol('style','edit','string','2',...
'position',[500 230 50 30],'groupname','math');
editProd=uicontrol('style','edit','string','2',...
'position',[500 200 50 30],'groupname','math');
editDiv=uicontrol('style','edit','string','2',...
'position',[500 170 50 30],'groupname','math');

//Тексты
titletext=uicontrol('style','text','string',...
'Калькулятор базовых математических операций','position',[0 310 500 30],...
'ForegroundColor',[0,0,1],'BackgroundColor',[1,1,1],'fontsize',20);
subTitleText=uicontrol('style','text','string',...
'Все операции выполняются поэлементно.','position',[0 100 550 30],...
'ForegroundColor',[0,0,1],'BackgroundColor',[0.9,0.9,1],'fontsize',14);
```

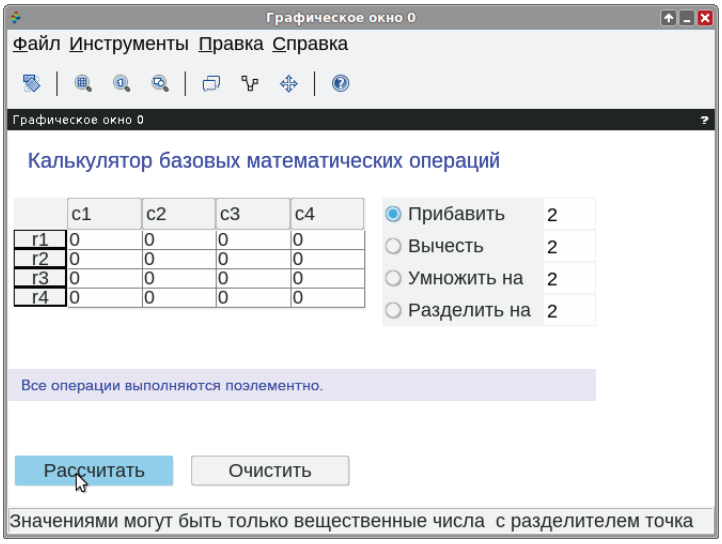

Рис. 5.12. Калькулятор

С помощью рассмотренных в этой главе компонентов и встроенных функций Scilab можно создавать визуальные программы для решения инженерных и математических задач любой сложности.

Глава 6

Нелинейные уравнения и системы в Scilab

Решение нелинейных уравнений – задача достаточно сложная. В этой главе читатель познакомится с методами решения нелинейных уравнений и их реализацией в Scilab. Отдельный параграф посвящён использованию встроенных функций Scilab для решения нелинейных уравнений. Завершает главу параграф, из которого читатель узнает, как решать системы нелинейных уравнений в Scilab.

6.1 Методы решения нелинейных уравнений

Уравнение $f(x) = 0$, в котором неизвестное входит в аргумент трансцендентных функций, называется *трансцендентным уравнением*, если функция $f(x)$ является аналитической функцией, но не является алгебраической. 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* [1, 5, 6, 10].

6.1.1 Решение нелинейных и трансцендентных уравнений

Численное решение нелинейного уравнения проводят в два этапа [1, 5, 6, 10]. Вначале 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. Методы определения интервала изоляции корня основаны на следующем свойстве: если непрерывная функция $f(x)$ на интервале $[a, b]$ поменяла знак, т. е. $f(a) \cdot f(b) < 0$, то она имеет на этом интервале хотя бы один корень. На втором этапе проводят уточнение отделённых корней, или, иначе говоря, находят корни с заданной точностью.

В качестве примера на рис. 6.1 показано графическое решение уравнения $f(x) = x^2 - \cos(5 \cdot x) = 0$. Функция дважды пересекает ось абсцисс, следова-

тельно, уравнение $x^2 - \cos(5 \cdot x) = 0$ имеет два корня. Первый находится на интервале $[-0.4; -0.2]$, второй принадлежит отрезку $[0.2; 0.4]$.

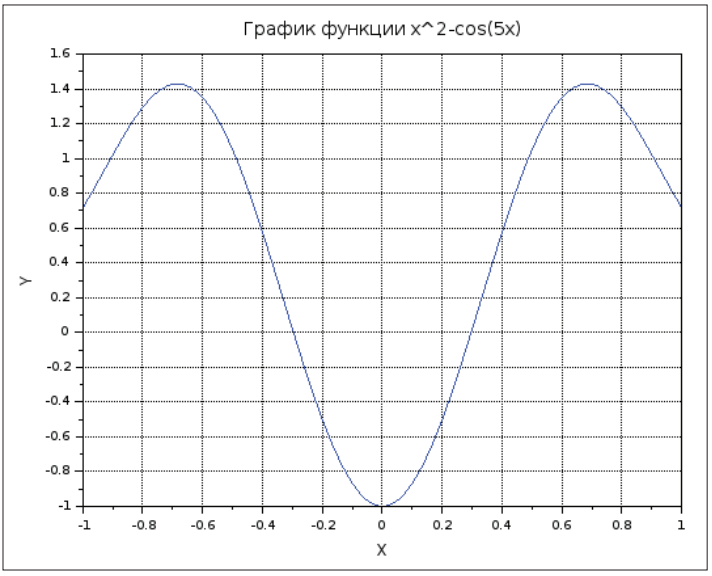

Рис. 6.1. Интервалы изоляции корня уравнения $x^2 - \cos(5 \cdot x) = 0$

Перед тем как начать знакомство с методами уточнения корней, давайте формально определим понятие *интервал изоляции*. На интервале изоляции корня функция $f(x)$ должна удовлетворять следующим условиям [1, 5, 6, 10]:

1) функция $f(x)$ непрерывна вместе со своими производными первого и второго порядка;
2) функция $f(x)$ на концах интервала $[a, b]$ имеет разные знаки $f(a) \cdot f(b) < 0$;
3) первая и вторая производные $f'(x)$ и $f''(x)$ сохраняют определённый знак на всём интервале $[a, b]$.

6.1.1.1 Метод половинного деления

Метод половинного деления (метод дихотомии) [1, 10] является простейшим методом уточнения корня. Пусть был выбран интервал изоляции $[a, b]$. Примем за первое приближение корня точку c, которая является серединой отрезка $[a, b]$. Далее будем действовать по следующему алгоритму:

1) находим точку $c = \frac{a+b}{2}$ (см. рис. 6.2);
2) находим значение $f(c)$;
3) если $f(a) \cdot f(c) < 0$, то корень лежит на интервале $[a, c]$, иначе корень лежит на интервале $[c, b]$;
4) если величина интервала меньше либо равна ε, то найден корень с точностью ε, иначе возвращаемся к п. 1.

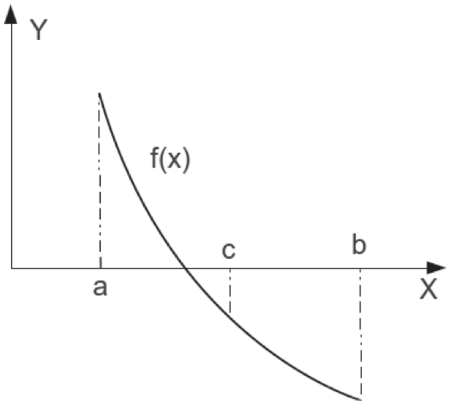

Рис. 6.2. Графическая интерпретация метода половинного деления

Однако, несмотря на простоту, такое последовательное сужение интервала не всегда рационально, так как требует слишком большого количества вычислений. Кроме того, этот способ не всегда позволяет найти решение с заданной точностью. Рассмотрим другие способы уточнения корня.

6.1.1.2 Метод хорд

Этот метод отличается от метода половинного деления тем, что очередное приближение берём не в середине отрезка, а в точке пересечения с осью X (рис. 6.3) прямой, соединяющей точки $(a, f(a))$ и $(b, f(b))$ [1, 10].

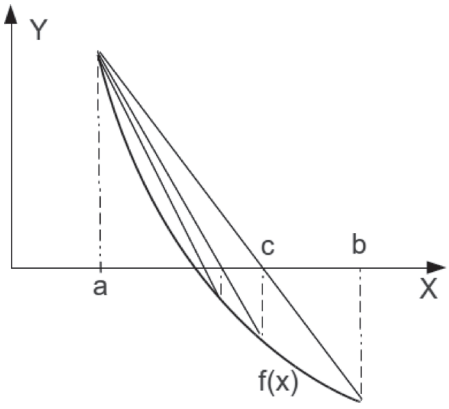

Рис. 6.3. Графическая интерпретация метода хорд

Запишем уравнение прямой, проходящей через точки с координатами $(a, f(a))$ и $(b, f(b))$:

$$\frac{y - f(a)}{f(b) - f(a)} = \frac{x - a}{b - a}, \quad y = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a). \tag{6.1}$$

Прямая, заданная уравнением (6.1), пересекает ось X при условии $y = 0$. Найдём точку пересечения хорды с осью X:

$$y = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a), \quad x = a - \frac{f(a) \cdot (b - a)}{f(b) - f(a)}.$$

Значит, $c = a - \frac{f(a)}{f(b) - f(a)}(b - a)$.

Алгоритм, реализующий метод хорд, можно представить так:

1) находим точку c по формуле $c = a - \frac{f(a)}{f(b) - f(a)}(b - a)$;
2) если $f(a) \cdot f(c) < 0$, то корень лежит на интервале $[a, c]$, иначе – на интервале $[c, b]$;
3) если абсолютное значение $f()$ не превышает некоторое достаточно малое число ϵ, то найден корень с точностью ϵ, иначе возвращаемся к п. 1.

Заметим, что описанный способ позволяет достаточно быстро (за меньшее количество шагов, чем в методе дихотомии) вычислить значение корня уравнения с заданной точностью.

6.1.1.3 Метод касательных

Метод касательных [1, 10] имеет второе название – метод Ньютона. В одной из точек интервала $[a; b]$, пусть это будет точка c, проведём касательную (рис. 6.4). Запишем уравнение этой прямой:

$$y = k \cdot x + m. \tag{6.2}$$

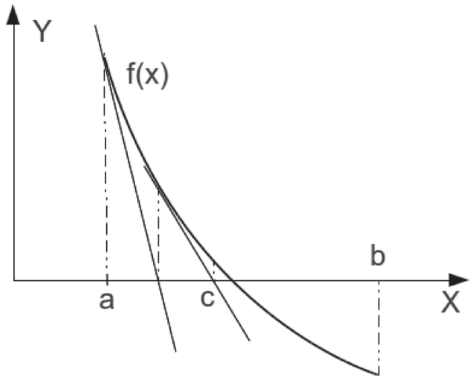

Рис. 6.4. Графическая интерпретация метода касательных

Так как эта прямая является касательной и проходит через точку $(c, f(c))$, то $k = f'(c)$.

Следовательно,

$$y = f'(x) \cdot x + m, f(c) = f'(c) \cdot c + m, m = f(c) - c \cdot f'(c),$$

$$y = f'(c) \cdot x + f(c) - c \cdot f'(c), y = f'(c) \cdot (x - c) + f(c).$$

Найдём точку пересечения касательной с осью X: $f'(c)\cdot(x-c)+f(c)=0$,

$$x = c - \frac{f(c)}{f'(c)}. \tag{6.3}$$

Если $|f(x)|<\varepsilon$, то точность достигнута и точка x – решение; иначе необходимо переменной c присвоить значение x и провести касательную через новую точку c; так продолжать до тех пор, пока $|f(x)|$ не станет меньше ε. Осталось решить вопрос, что выбрать в качестве точки начального приближения c.

В этой точке должны совпадать знаки функции и её второй производной. А так как нами было сделано допущение, что вторая и первая производные не меняют знак, то можно проверить условие $f(x)\cdot f''(x)>0$ на обоих концах интервала и в качестве начального приближения взять ту точку, где это условие выполняется. Для реализации данного алгоритма нужно найти первую и вторую производные функции $f(x)$.

6.1.1.4 Метод простой итерации

Для решения уравнения этим методом необходимо записать уравнение $f(x)=0$ в виде $x=\phi(x)$, задать начальное приближение $x_0\in[a;b]$ и организовать следующий итерационный вычислительный процесс [1, 3, 10]:

$$x_{k+1}=\phi(x_k),\quad k=0,1,2,\ldots.$$

Вычисление прекратить, если $|x_{k+1}-x_k|<\varepsilon$ (ε – точность).

Если неравенство $|\phi'(x)|<1$ выполняется на всём интервале $[a;b]$, то последовательность $x_0, x_1, x_2, \ldots, x_n, \ldots$ сходится к решению x^* (т. е. $\lim\limits_{k\to\infty} x_k = x^*$).

Значение функции $\phi(x)$ должно удовлетворять условию $|\phi'(x)|<1$, для того чтобы можно было применить метод простых итераций. Условие $|\phi'(x)|<1$ является *достаточным условием сходимости* метода простой итерации.

Уравнение $f(x)=0$ можно привести к виду $x=\phi(x)$ следующим образом. Умножить обе части уравнения $f(x)=0$ на число λ. К обеим частям уравнения $\lambda\cdot f(x)=0$ добавить число x. Получим $x=x+\lambda\cdot f(x)$. Это и есть уравнение вида $x=\phi(x)$, где

$$\phi(x)=x+\lambda\cdot f(x).$$

Необходимо, чтобы неравенство $|\phi'(x)|<1$ выполнялось на интервале $[a;b]$, следовательно, $|\phi'(x)|=|1+\lambda\cdot f'(x)|$ и $|1+\lambda\cdot f'(x)|<1$ ($|1+\lambda\cdot f'(a)|<1$, $|1+\lambda\cdot f'(b)|<1$), а значит, с помощью *подбора параметра* λ можно добиться выполнения *условия сходимости*.

Рассмотрим процесс нахождения λ на примере уже рассмотренного выше уравнения $x^2-\cos(5\cdot x)=0$, один из корней находится на интервале изоляции $a=0.2$; $b=0.4$. Подберём значение λ, решив неравенство $|1+\lambda\cdot f'(x)|<1$:

$$|1+\lambda\cdot f'(a)|<1 \text{ и } |1+\lambda\cdot f'(b)|<1,$$

$$f(x)=x^2-\cos(5\cdot x),\ f'(x)=2\cdot x+5\cdot\sin(5\cdot x),$$

$$f'(a)=2\cdot 0.2+5\cdot\sin(5\cdot 0.2)\approx 4.6\ ,\ f'(b)=2\cdot 0.4+5\cdot\sin(5\cdot 0.4)\approx 5.35,$$

$$|1+\lambda\cdot 4.6|<1 \text{ и } |1+\lambda\cdot 5.35|<1.$$

$$\begin{cases}\begin{cases}1+4.6\cdot\lambda<1\\1+4.6\cdot\lambda>-1\end{cases}\\\begin{cases}\lambda<0\\\lambda>-0.37\end{cases}\end{cases}\Rightarrow\begin{cases}\begin{cases}\lambda<0\\\lambda>-0.4\end{cases}\\\begin{cases}\lambda<0\\\lambda>-0.37\end{cases}\end{cases}\Rightarrow\begin{cases}\lambda\in(-0.4;0)\\\lambda\in(-0.37;0)\end{cases}$$

и, следовательно, $\lambda\in(-0.37;0)$.

6.1.1.5 Метод секущих

Суть метода секущих [10] в следующем. Если нам известны значения в двух точках $(x_k, f(x_k))$, $(x_{k+1}, f(x_{k+1}))$ внутри интервала изоляции $[a; b]$, то через эти точки можно провести прямую до точки (x_{k+2}) пересечения её с осью абсцисс (см рис. 6.5 [10]). Выпишем формулу для вычисления x_{k+2}.

$$x_{k+2}=x_{k+1}-\frac{f(x_{k+1})\cdot(x_{k+1}-x_k)}{f(x_{k+1})-f(x_k)},\quad k=0,1,2,\ldots.$$

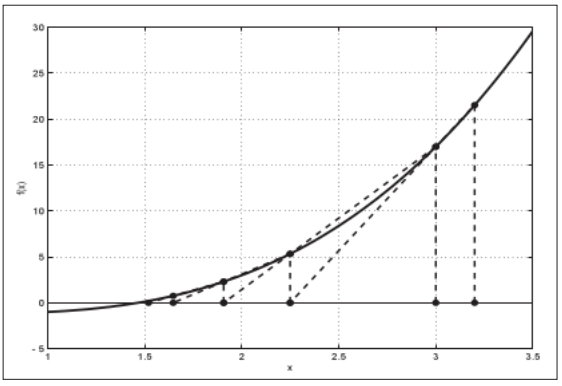

Рис. 6.5. Графическая интерпретация метода секущих

Вычисления будем продолжать до тех пор, пока очередное значение $f(x_{k+2})$ не станет меньше точности ε.

Рассмотрим реализацию описанных выше методов в Scilab (см. листинг 6.2) на примере уравнения $x^2-\cos(5\cdot x)=0$.

Листинг 6.1. Функции решения нелинейных уравнений

```
//Левая часть уравнения
function y=f(x)
  y=x.*x-cos(5*x)
endfunction

//Правая часть уравнения x=fi(x)
function y=fi(lyam,x)
```

```
  y=x+lyam*f(x)
endfunction

//Производная функции f(x)
function y=f1(x)
  y=2*x+5*sin(5*x)
endfunction

//Вторая производная функции f(x)
function y=f2(x)
  y=2+25*cos(5*x)
endfunction

//Функция уточнения корня методом половинного деления
//a,b - интервал изоляции
//eps - точность решения уравнения
//c - найденный корень уравнения
//k - количество итераций для нахождения корня
//с точностью eps
function [k,c]=dihot(a,b,eps)
  k=0;
  while abs(b-a)>eps
    k=k+1;
    c=(a+b)/2;
    if f(a)*f(c)>0 then
      a=c;
    else
      b=c;
    end;
    if abs(f(c))<eps then
      break;
    end
  end
endfunction

//Функция уточнения корня методом хорд
//a,b - интервал изоляции
//eps - точность решения уравнения
//c - найденный корень уравнения
//k - количество итераций для нахождения корня
//с точностью eps
function [k,c]=hord(a,b,eps)
  c=a-f(a)/(f(b)-f(a))*(b-a);
  k=0;
  while abs(f(c))>eps
    k=k+1;
    c=a-f(a)/(f(b)-f(a))*(b-a);
    if f(a)*f(c)>0 then
      a=c;
    else
      b=c;
    end;
  end;
endfunction
```

```
//Функция уточнения корня методом секущих
//xo,x1 - две точки внутри интервала изоляции
//eps - точность решения уравнения
//c - найденный корень уравнения
//k - количество итераций для нахождения корня
//с точностью eps
function [k,x2]=sek(x0,x1,eps)
  k=0;
  while abs(f(x1))>eps
    k=k+1;
    x2=x1-f(x1)*(x1-x0)/(f(x1)-f(x0));
    x0=x1;
    x1=x2;
  end
endfunction

//Функция уточнения корня методом касательных
//a,b - интервал изоляции
//eps - точность решения уравнения
//c - найденный корень уравнения
//k - количество итераций для нахождения корня
//с точностью eps
function [k,c]=kasat(a,b,eps)
  k=0
  if f(a)*f2(a)>0 then
    c=a;
  else
    c=b
  end
  while abs(f(c))>eps
    k=k+1;
    c=c-f(c)/f1(c)
  end
endfunction

//Функция уточнения корня методом половинного деления
//a,b - интервал изоляции
//layam - подобранный коэффициент для записи правой части
//уравнения в виде x=fi(x)=x+lyam*f(x))
//eps - точность решения уравнения
//c - найденный корень уравнения
//k - количество итераций для нахождения корня
//с точностью eps
function [k,x]=iter(a,b,lyam,eps)
  x=(a+b)/2;
  k=0
  while abs(f(x))>eps
    k=k+1;
    x=fi(lyam,x)
  end
endfunction

//Интервал изоляции корня уравнения x=f(x)=x^2-cos(5*x)
a=0.2;
```

```
b=0.4;
//Точность решения уравнения
eps=%eps;
//Построение графика функции y=f(x)=x^2-cos(5x)
xx=-1:0.02:1;
plot(xx,f(xx))
xgrid;
xtitle("График функции x^2-cos(5x)","X","Y");
[k,c]=dihot(a,b,eps);
mprintf("Метод половинного деления\n");
mprintf("x=%e\tf(x)=%e\tk=%d\n",c,f(c),k);
[k,c]=hord(a,b,eps);
mprintf("Метод хорд\n");
mprintf("x=%e\tf(x)=%e\tk=%d\n",c,f(c),k);
[k,c]=sek(0.2,0.21,eps);
mprintf("Метод секущих\n");
mprintf("x=%e\tf(x)=%e\tk=%d\n",c,f(c),k);
mprintf("Метод касательных\n");
[k,c]=kasat(0.2,0.4,eps);
mprintf("x=%e\tf(x)=%e\tk=%d\n",c,f(c),k);
mprintf("Метод простой итерации\n");
[k,c]=iter(0.2,0.4,-0.2,eps);
mprintf("x=%e\tf(x)=%e\tk=%d\n",c,f(c),k);
```

График функции приводился ранее (см. рис. 6.1). Решение уравнения $x^2 - \cos(5 \cdot x) = 0$ с помощью написанных авторами функций решения нелинейных уравнений приведено ниже.

Листинг 6.2. Решение уравнения $x^2 - \cos(5x) = 0$

```
Метод половинного деления
x=2.965483e-01 f(x)=3.330669e-16 k=50
Метод хорд
x=2.965483e-01 f(x)=8.326673e-17 k=5
Метод секущих
x=2.965483e-01 f(x)=8.326673e-17 k=6
Метод касательных
x=2.965483e-01 f(x)=8.326673e-17 k=4
Метод простой итерации
x=2.965483e-01 f(x)=-1.804112e-16 k=15
```

6.1.2 Особенности решения алгебраических уравнений

Любое уравнение $f(x) = 0$, где $f(x)$ – это многочлен, отличный от нулевого, называется *алгебраическим уравнением*, или *полиномом*. Всякое алгебраическое уравнение относительно x можно записать в виде:

$$P_1 x^n + P_2 x^{n-1} + \cdots + P_n x + P_{n+1} = 0,$$

где $P_1 \neq 0$, $n \geq 1$ и P_i – коэффициенты алгебраического уравнения n-й степени. Например, линейное уравнение – это алгебраическое уравнение первой степени, квадратное – второй, кубическое – третьей и т. д.

Рассмотрим **алгоритм** *нахождения всех корней полинома n-й степени* [6, 10]. В общем случае многочлен n-й степени имеет n корней. Однако некоторые из них могут быть комплексными. Алгоритм состоит в многократном повторении (количество итераций определяется степенью многочлена) следующих действий:

1) нахождение очередного корня α *методом парабол*. Выбор метода парабол обусловлен тем, что процесс сходится при любом начальном приближении. Формально это не доказано, но ни разу никем не опровергнуто. Метод парабол будет представлен ниже;
2) понижение порядка полинома путём деления многочлена на $x - \alpha$. Деление полинома на $x - \alpha$ будет осуществляться методом Горнера, который будет описан ниже.

6.1.2.1 Метод парабол

На практике используется метод параболы и обратной параболы. Метод параболы (или метод Мюллера) позволяет найти **все** (*действительные* и *комплексные*) корни полинома, он сходится при любом начальном приближении. Однако следует учитывать, что вычисления следует вести в комплексных числах. Возможна даже парадоксальная ситуация нахождения действительных корней с использованием комплексных вычислений. Основная идея алгоритма состоит в следующем. Задаются три точки начального приближения $(x_0, y_0 = f(x_0))$, $(x_1, y_1 = f(x_1))$ и $(x_2, y_2 = f(x_2))$. Для практического использования можно задать одну точку x^*, а ещё две вычислить по формулам $0.85 \cdot x^*$, $1.15 \cdot x^*$. Через эти точки проводим параболу вида $y = a \cdot x^2 + b \cdot x + c$. Точка пересечения параболы с осью абсцисс и будет точкой следующего приближения. Подробно о методе парабол можно прочесть в [1, 10]. Расчётная формула для вычисления следующего приближения корня имеет вид [6, 10]:

$$x = x_0 - \frac{2 \cdot f(x_0)}{\omega \pm \sqrt{\omega^2 - 4 \cdot f(x_0) \cdot y_{012}}}. \tag{6.4}$$

Здесь ω вычисляется по формуле $\omega = y_{01} + y_{02} - y_{12}$,

$$y_{01} = \frac{y_0 - y_1}{x_0 - x_1}, \quad y_{02} = \frac{y_0 - y_2}{x_0 - x_2}, \quad y_{12} = \frac{y_1 - y_2}{x_1 - x_2}$$

– разделённые разности первого порядка;

$$y_{012} = \frac{y_{12} - y_{01}}{x_0 - x_2}$$

– разделённая разность второго порядка.

Знак «+» или «–» в формуле (6.4) выбирается таким образом, чтобы знаменатель в (6.4) был больше. Дальше отбрасываем x_0 и имеем три точки x, x_1, x_2. Процесс повторяем до тех пор, пока $|f(x)|$ не станет меньше точности ε.

Если же необходимо найти только действительные корни полинома, то для нахождения очередного корня можно использовать метод обратной параболы.

6.1.2.2 Метод обратной параболы

В этом методе [6, 10] через точки $(x_0, f(x_0))$, $(x_1, f(x_1))$ и $(x_2, f(x_2))$ проводим параболу вида $x = a \cdot y^2 + b \cdot y + c$ [1, 10]. Коэффициент параболы c и будет следующим приближением:

$$a = \frac{(x_2 - x_1)(f(x_1) - f(x_0)) - (x_1 - x_0)(f(x_2) - f(x_1))}{(f(x_2) - f(x_1))(f(x_1) - f(x_0))(f(x_2) - f(x_0))}, \tag{6.5}$$

$$b = -a(f(x_2) + f(x_1)) + \frac{x_2 - x_1}{f(x_2) - f(x_1)}, \tag{6.6}$$

$$c = x_{n+1} = x_2 - a \cdot f(x_2)^2 - b \cdot f(x_2). \tag{6.7}$$

Далее $x_0 = x_1$, $x_1 = x_2$, $x_2 = c$, пересчитываем $f(x_0)$, $f(x_1)$, $f(x_2)$ и повторяем вычислительный процесс до тех пор, пока $|f(x_2)|$ не станет меньше точности ε. К значительным минусам метода обратной параболы следует отнести чувствительность к выбору точки начального приближения [6, 10].

6.1.2.3 Понижение порядка полинома

Рассмотрим алгоритм понижения порядка полинома

$$P_1 x^n + P_2 x^{n-1} + \cdots + P_n x + P_{n+1} = 0$$

путём деления многочлена на $x - \alpha$, где α – корень многочлена.

$$P_1 x^n + P_2 x^{n-1} + \cdots + P_n x + P_{n+1} = (b_1 x^{n-1} + b_2 x^{n-2} + \cdots + b_n)(x - \alpha) + b_{n+1},$$

b_{n+1} – остаток от деления, $b_{n+1} = 0$, если α – корень полинома. Алгоритм нахождения коэффициентов b можно записать так:

```
b(1)=P(1)
for i=2:n+1
b(i)=b(i-1)*alpha+P(i)
end
```

Этот алгоритм носит название процедуры синтетического деления.

6.1.2.4 Вычислительная схема Горнера

Для понимания алгоритма поиска корней полинома напомним схему Горнера, с помощью которой можно рассчитать значение полинома

$$f(x) = P_1 x^n + P_2 x^{n-1} + \cdots + P_n x + P_{n+1}$$

в заданной точке.

Более подробно запишем вычислительную схему Горнера для полинома 4-й степени.

$$f(x) = P_1 x^4 + P_2 x^3 + P_3 x^2 + P_4 x + P_5 = P_5 + x(P_4 + x(P_3 + x(P_2 + xP_1))).$$

Вычислительная схема Горнера для полинома 4-го порядка:

```
b=P(1)
b=P(2) + x*b
b=P(3) + x*b
b=P(4) + x*b
b=P(5) + x*b
```

Обобщим её на случай полинома n-й степени:

```
b=P(1)
for i=2:n
b=P(i) + x*b
end
```

Можно записать алгоритм даже так:

```
b=0
for i=1:n
b=P(i) + x*b
end
```

Здесь `n` – количество коэффициентов в полиноме.

Теперь рассмотрим программы нахождения корней полинома.

Задача 6.1

Написать программу нахождения всех корней полинома

$$P_1 x^n + P_2 x^{n-1} + \cdots + P_n x + P_{n+1} = 0.$$

В качестве тестовой будем использовать задачу нахождения корней полинома [6]:

$$x^4 - 35x^3 + 280x^2 - 1350x + 1000 = 0.$$

Ниже приведен код программы с комментариями[1].

Листинг 6.3. Программа вычисления всех корней полинома

```
clear;
//Функция вычисления разделённой разности первого порядка
function y=razd(x1,x2,f1,f2)
y=(f2-f1)/(x2-x1);
endfunction
//Функция вычисления разделённой разности второго порядка
function y=razd2(x0,x1,x2,f0,f1,f2)
```

[1] Авторы книги выражают благодарность студенткам факультета математики и компьютерных наук Кубанского государственного университета Горбуновой Ирине, Маклаковой Анастасии, Марченко Алине за помощь в разработке алгоритма и написании кода программы нахождения корней полинома.

```
y=(razd(x1,x2,f1,f2)-razd(x0,x1,f0,f1))/(x0-x2);
endfunction
//Функция вычисления значения полинома в точке
//Если pr=1, вычисление по схеме Горнера
//Если pr=2, вычисление по классической формуле полинома
function y0=gr(P,N,x0,pr)
if pr==1 then
y0=0;
for i=1:1:N y0=P(i)+x0*y0;
end
end
if pr==2 then
y0=0
for i=1:N
y0=y0+x0^(N-i-1)*P(i);
end
end
endfunction
//Функция понижения порядка полинома с помощью
//процедуры синтетического деления.
function B=delenie_gorner(P,N,alfa)
B(1)=P(1);
for i=2:1:N
B(i)=B(i-1)*alfa+P(i);
end
endfunction
//Функция нахождения очередного корня полинома
//методом Мюллера (парабол)
function x=Muller(P,N,x1)
x2=1.15*x1;
x0=0.85*x1;
f0=gr(P,N,x0,1);
f1=gr(P,N,x1,1);
f2=gr(P,N,x2,1);
x=x2;
while abs(gr(P,N,x,1))>abs(%eps)
u=razd(x0,x1,f0,f1)+razd(x0,x2,f0,f2)-razd(x1,x2,f1,f2);
t1=u-sqrt(u*u-4.*f0*razd2(x0, x1, x2,f0,f1,f2));
t2=u+sqrt(u*u-4.*f0*razd2(x0, x1, x2,f0,f1,f2));
if abs(t1)>abs(t2) then
t=t1;
else
t=t2;
end
x=x0-2*gr(P,N,x0,1)/t;
x2=x1;
x1=x0;
x0=x;
f0=gr(P,N,x0,1);
f1=gr(P,N,x1,1);
f2=gr(P,N,x2,1);
end
```

```
endfunction
//Функция нахождения всех корней полинома
function r=korni_polynom(P,N)
t=1;
mprintf("Корни полинома:\n");
while(N>1)
x=1;
x=Muller(P,N, x);
mprintf("x=%10.7f\n",x);
r(t)=x;
P=delenie_gorner(P,N,x)
N=N-1;
t=t+1;
end
endfunction
//Задание коэффициентов тестового полинома
H=[1 -35 280 -1350 1000];
//Вызов функции нахождения корней полинома
Y=korni_polynom(H,5);
//построение графика полинома
L=0:0.1:18;
for i=1:length(L)
M(i)=gr(H,5,L(i),1);
end
plot(L',M);
xgrid;
xtitle("Грaфик функции x^4 -35x^3+280x^2-1350x+1000","OX","OY");
```

Результат работы программы представлен в листинге 6.4. На рис. 6.6 показан график полинома.

Листинг 6.4. Результат работы программы

```
Корни полинома:
x= 0.9942249
x= 5.0000000
x=11.4743548
x=17.5314203
```

Задача 6.2

Написать программу нахождения действительных корней многочлена

$$P_1 x^n + P_2 x^{n-1} + \cdots + P_n x + P_{n+1} = 0.$$

В качестве тестового будем использовать тот же многочлен.

$$x^4 - 35x^3 + 280x^2 - 1350x + 1000 = 0$$

В этой программе будем использовать метод обратной параболы. Код программы приведён ниже. Результаты работы этой программы и программы из задачи 6.1 совпадают.

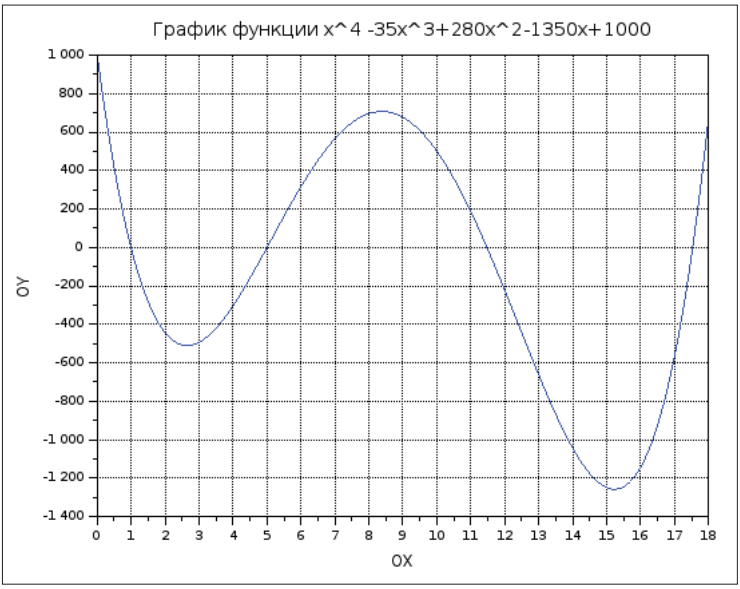

Рис. 6.6. График полинома $x^4 - 35x^3 + 280x^2 - 1350x + 1000 = 0$

Листинг 6.5. Программа вычисления корней полинома

```
//Функция вычисления разделённой разности первого порядка
function y=razd(x1,x2,f1,f2)
  y=(f2-f1)/(x2-x1);
endfunction

//Функция вычисления разделённой разности первого порядка
function y=razd2(x0,x1,x2,f0,f1,f2)
  y=(razd(x1,x2,f1,f2)-razd(x0,x1,f0,f1))/(x0-x2);
endfunction

//Функция вычисления значения полинома в точке по схеме Горнера
function y0=gorner(P,N,x0)
  y0=0;
  for i=1:1:N
    y0=P(i)+x0*y0;
  end
endfunction

//Функция понижения порядка полинома с помощью
//процедуры синтетического деления.
function B=delenie_gorner(P,N,alfa)
  B(1)=P(1);
  for i=2:1:N
    B(i)=B(i-1)*alfa+P(i);
  end
endfunction

//Функция нахождения очередного корня полинома
//методом обратной параболы
```

```
function [pr,x]=obr_parabol(P,N,x1,K)
  x2=1.15*x1;
  x0=0.85*x1;
  pr=%t;
  f0=gorner(P,N,x0);
  f1=gorner(P,N,x1);
  f2=gorner(P,N,x2);
  i=0;
  while abs(f2)>%eps
    i=i+1;
    a=((x2-x1)*(f1-f0)-(x1-x0)*(f2-f1))/((f2-f1)*(f1-f0)*(f2-f0));
    b=-a*(f2+f1)+(x2-x1)/(f2-f1);
    c=x2-a*f2*f2-b*f2;
    x0=x1;
    x1=x2;
    x2=c;
    f0=gorner(P,N,x0);
    f1=gorner(P,N,x1);
    f2=gorner(P,N,x2);
    if i>K then
      pr=%f;
      break;
    end
  end
  x=x2;
endfunction

//Функция нахождения всех корней полинома
function r=korni_obr_parabola(P,N)
  t=1;
  while(N>1)
    x=1;
    [fl,x]=obr_parabol(P,N,x,500);
    if fl then
      mprintf("x2=%e\n",x);
      r(t)=x;
      P=delenie_gorner(P,N,x)
      t=t+1;
    else
      break;
      end;
      N=N-1;
  end
endfunction

//Задание коэффициентов тестового полинома
H=[1 -35 280 -1350 1000];
//Вызов функции нахождения корней полинома
Yobr=korni_obr_parabola(H,5);
//Вывод корней
disp(Yobr);
```

На взгляд авторов, следует пользоваться ***именно*** методом парабол, а если вам нужны только действительные корни, то следует избавиться от комплексных уже после нахождения всех корней.

Задачи решения нелинейных уравнений и систем не являются тривиальными. Написание программ решения нестандартных уравнений требует значительного опыта программирования. Рассмотрим возможности Scilab [2] для решения подобных задач.

6.2 Встроенные функции Scilab для решения нелинейных уравнений

6.2.1 Решение алгебраических уравнений

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 6.6. 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 6.7. 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 6.8. 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 6.3
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 6.9. 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 6.10. Использование функции `roots`

```
-->X=roots(p)
X  =
!   0.4588039 !
! - 0.3065630 !
!   1.5411961 !
!   2.306563  !
```

Графическое решение задачи[1], показанное на рис. 6.7, позволяет убедиться, что корни найдены верно.

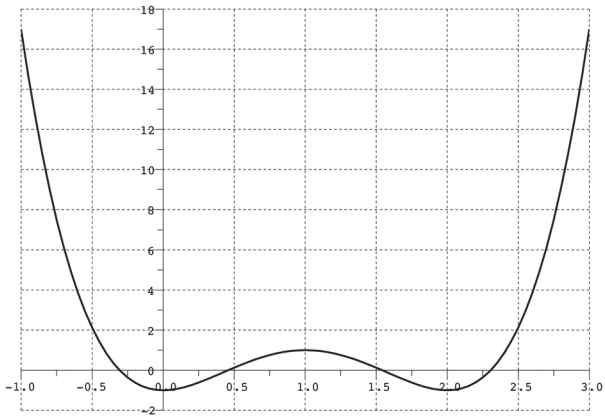

Рис. 6.7. Графическое решение задачи 6.3

[1] Графическим решением уравнения $f(x) = 0$ является точка пересечения линии $f(x)$ с осью абсцисс.

Задача 6.4
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.

Листинг 6.11. Решение задачи 6.4

```
-->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. 6.8) и два комплексных корня.

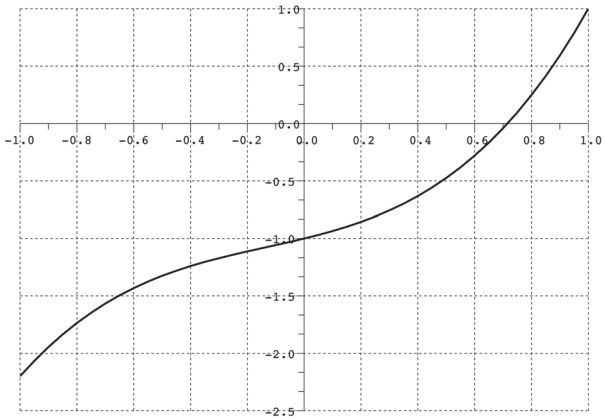

Рис. 6.8. Графическое решение задачи 6.4

Задача 6.5
Найти решение уравнения $y(x) = 0$, если $y(x) = x^4 - 18x^2 + 6$.

Решение этой задачи представлено в листинге 6.12 и отличается от предыдущих лишь способом определения полинома. Графическое решение представлено на рис. 6.9.

Листинг 6.12. Решение задачи 6.5

```
-->x=poly(0,'x');
-->y=x^4-18*x^2+.6;
-->roots(y)
ans  =
!   0.1827438 !
```

```
! - 0.1827438 !
! - 4.2387032 !
!   4.2387032 !
```

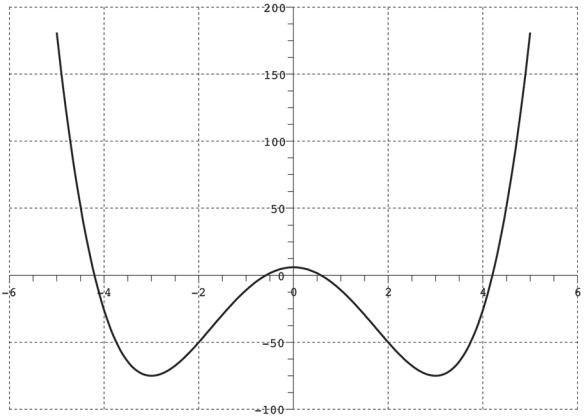

Рис. 6.9. Графическое решение задачи 6.5

6.2.2 Решение трансцендентных уравнений

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 6.6

Найти решение уравнения $\sqrt[3]{(x-1)^2} - \sqrt[3]{x^2} = 0$.

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. 6.10 видно, что корень данного уравнения лежит в интервале [0; 1].

Выберем ноль в качестве начального приближения, зададим функцию, описывающую уравнение, и решим его.

Листинг 6.13. Решение задачи 6.6

```
-->deff('[y]=f1(x)','y1=((x-1)^2)^(1/3),y2=(x^2)^(1/3),y=y1-y2')
-->fsolve(0,f1)
ans  =     0.5
```

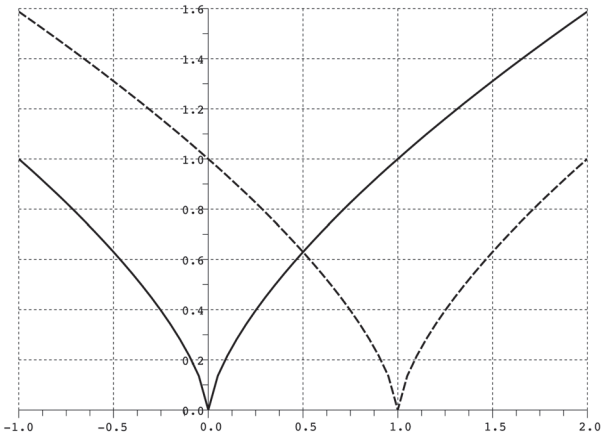

Рис. 6.10. Графическое решение задачи 6.6

Задача 6.7
Найти все корни уравнения $f(x) = e^x/5 - 2(x-1)^2$.

На рис. 6.11 видно, что график функции $f(x)$ трижды пересекает ось абсцисс, т. е. уравнение имеет три корня.

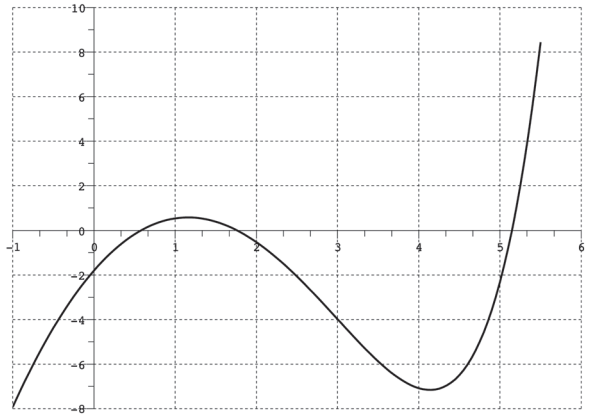

Рис. 6.11. Графическое решение задачи 6.7

Вызовем функцию `fsolve`, передавая в качестве первого параметра массив значений начальных приближений корней уравнений, получим все решения заданного уравнения.

Листинг 6.14. Решение задачи 6.7

```
function y=f(x)
y=exp(x)/5-2*(x-1)^2
```

```
end
x=fsolve([0 2 5],f)
for i=1:3
mprintf("x=%7.4f\n",x(i))
end
x= 0.5778
x= 1.7639
x= 5.1477
```

Задача 6.8

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. 6.12).

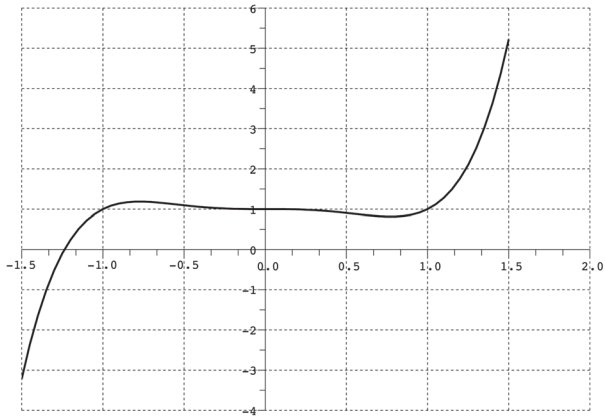

Рис. 6.12. Графическое решение задачи 6.8

Решим эту задачу при помощи функции fsolve.

Листинг 6.15. Решение задачи 6.8

```
-->deff('[f]=y(x)','f=x^5-x^3+1')
-->X=fsolve(-2,y)
X  = 1.2365057
```

Теперь применим функцию `roots`.

Листинг 6.16. Решение задачи 6.8 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 6.15), имеет и мнимые (листинг 6.16). Поэтому для отыскания всех корней полинома лучше использовать функцию `roots`.

6.3 Решение систем нелинейных уравнений в Scilab

Если задано 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 6.9

Решить систему уравнений: $\begin{cases} x^2 + y^2 = 1 \\ x^3 - y = 0 \end{cases}$.

Графическое решение системы (рис. 6.13) показывает, что она имеет две пары корней.

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 6.17. Решение задачи 6.9

```
function [y]=fun(x)
y(1)=x(1)^2+x(2)^2-1;
y(2)=x(1)^3-x(2);
endfunction
fsolve([0.5 0.5],fun)
ans  = 0.8260314    0.5636242
fsolve([-0.5 -0.5],fun)
ans  =  - 0.8260314  - 0.5636242
```

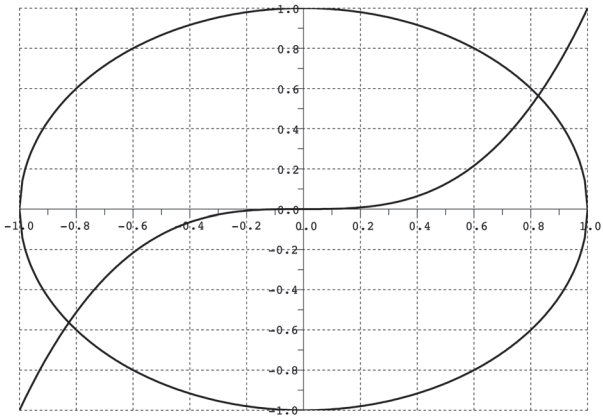

Рис. 6.13. Графическое решение системы уравнений

Задача 6.10

Исследовать систему из трёх нелинейных уравнений с тремя неизвестными:

$$\begin{cases} x^2 + y^2 + z^2 = 1 \\ 2x^2 + y^2 - 4z = 0. \\ 3x^2 - 4y + z^2 = 0 \end{cases}$$

Решение системы представлено ниже (листинг 6.18).

Листинг 6.18. Решение задачи 6.10

```
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
fsolve([0.5 0.5 0.5],fun)//решение системы
ans  = !   0.7851969    0.4966114    0.3699228 !
```

Как видно из приведённых примеров решения систем нелинейных уравнений, вызов функции `fsolve` довольно прост. Самой большой сложностью при решении систем нелинейных уравнений является выбор начального приближения. Эта проблема выходит далеко за рамки книги, и авторы советуют интересующимся читателям подробно изучить данный вопрос в классической и современной литературе по численным методам [5, 6, 10].

Глава 7

Численное интегрирование и дифференцирование

В данной главе рассмотрены численные алгоритмы интегрирования и дифференцирования, которые реализованы в функциях в Scilab. Также описаны основные встроенные функции Scilab численного интегрирования и дифференцирования.

7.1 Основные методы численного интегрирования

Пусть дана функция $f(x)$, известно, что она непрерывна на интервале $[a, b]$ и уже определена её первообразная $F(x)$, тогда *определённый интеграл* от этой функции можно вычислить в пределах от a до b по *формуле Ньютона–Лейбница*:

$$\int_a^b f(x)dx = F(b) - F(a), \quad \text{где } F'(x) = f(x). \tag{7.1}$$

На практике часто встречаются интегралы с первообразной, которая не может быть выражена через элементарные функции или является слишком сложной, что затрудняет или делает невозможным вычисления по формуле Ньютона–Лейбница. Кроме того, нередко подынтегральная функция задаётся таблицей или графиком, и тогда понятие первообразной вообще теряет смысл. В этом случае большое значение имеют *численные методы интегрирования*, основная задача которых заключается в вычислении значения определённого интеграла на основании значений подынтегральной функции.

Численное вычисление определённого интеграла называют *механической квадратурой*. Формулы, соответствующие тому или иному численному методу приближённого интегрирования, называют *квадратурными*. Подобное название связано с *геометрическим смыслом определённого интеграла*: значение определённого интеграла

$$y = \int_a^b f(x)dx, \quad f(x) \neq 0$$

равно площади криволинейной трапеции с основаниями $[a, b]$ и $f(x)$.

7.1.1 Интегрирование по методу трапеций

Изложим геометрическую интерпретацию *интегрирования по методу трапеций* [1, 3, 5, 8]. Для этого участок интегрирования $[a, b]$ разобьём точками на n равных частей (рис. 7.1), причём $x_0 = a$, $x_n = b$.

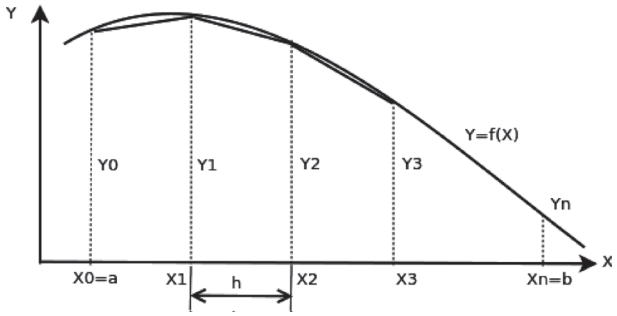

Рис. 7.1. Геометрическая интерпретация метода трапеций

Тогда длина каждой части будет равна $h = \frac{b-a}{n}$, а значение абсциссы каждой из точек разбиения можно вычислить по формуле: $x_i = x_0 + ih$, $i = 1, 2, \ldots, n-1$. Теперь из каждой точки x_i проведём перпендикуляр до пересечения с кривой $f(x)$, а затем заменим каждую из полученных криволинейных трапеций прямоугольной. Приближённое значение интеграла будем рассматривать как сумму площадей прямолинейных трапеций, причём площадь отдельной трапеции составляет

$$S_i = \frac{y_{i-1} + y_i}{2} h,$$

следовательно, площадь искомой фигуры вычисляют по формуле:

$$S = \int_a^b f(x)dx = \sum_{i=1}^{n} S_i = \frac{h}{2} \sum_{i=1}^{n} (y_{i-1} + y_i) = h\left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i \right).$$

Таким образом, получена *квадратурная формула трапеций* для численного интегрирования:

$$I = \int_a^b f(x)dx = h\left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i) \right). \tag{7.2}$$

7.1.2 Интегрирование по методу Симпсона

Идея *интегрирования по методу Симпсона* [1, 3, 5, 8] заключается в следующем. Пусть $n = 2m$ – чётное число, а $y_i = f(x_i)$ $(i = 0, 1, \ldots, n)$ – значения функции $y = f(x)$ для равноотстоящих точек $a = x_0, x_1, x_2, \ldots, x_n = b$ с шагом

$$h = \frac{b-a}{n} = \frac{b-a}{2m}.$$

На паре соседних участков (рис. 7.2) кривая $y = f(x)$ заменяется параболой $y = L(x)$, коэффициенты которой подобраны так, что она проходит через точки Y_0, Y_1, Y_2.

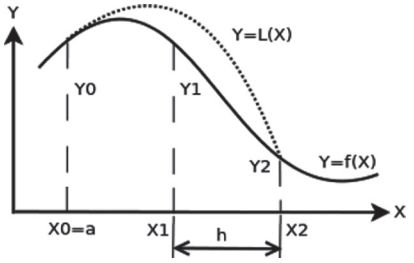

Рис. 7.2. Геометрическая интерпретация интегрирования по методу Симпсона

Площадь криволинейной трапеции, ограниченной сверху параболой, составит:

$$S_i = \frac{h}{3}(y_{i-1} + 4y_i + y_{i+1}).$$

Суммируя площади всех криволинейных трапеций, получим:

$$S = \int_a^b f(x)dx \approx \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \cdots + 2y_{2m-2} + 3y_{2m-1} + y_{2m}) =$$
$$= \frac{h}{3}\left(y_0 + y_{2m} + \sum_{i=1}^{2m-1} py_i\right),$$

где $p = 2$ при чётном i и $p = 4$ при нечётном i.

Следовательно, *формула Симпсона для численного интегрирования* имеет вид:

$$I = \int_a^b f(x)dx = \frac{h}{3}\left(f(a) + f(b) + \sum_{i=1}^{2m-1} py_i\right). \tag{7.3}$$

Методы трапеций и Симпсона являются частными случаями *квадратурных формул Ньютона–Котеса* [1, 3, 5, 8], которые, вообще говоря, имеют вид:

$$\int_a^b ydx = (b-a)\sum_{i=0}^{n} H_i y_i,$$

где H_i – это некоторые константы, называемые *постоянными Ньютона–Котеса*. Если для квадратурных формул Ньютона–Котеса принять $n = 1$, то получим метод трапеций, а при $n = 2$ – метод Симпсона. Поэтому эти методы называют *квадратурными методами низших порядков*. Для $n > 2$ получают

квадратурные формулы Ньютона–Котеса высших порядков. При $n = 3$ квадратурная формула *Ньютона–Котеса* имеет вид [1]:

$$\int_{x_0}^{x_3} y dx = \frac{3h}{8}(y_0 + 3y_1 + 3y_2 + y_3).$$

Эта формула носит название *квадратурной формулы Ньютона*, или *правила трёх восьмых*. Более подробно о квадратурных формулах Ньютона–Котеса можно прочитать в [1].

При вычислении интеграла любым из вышеперечисленных методов возникает вопрос: насколько точно был вычислен интеграл? Для ответа на этот вопрос можно воспользоваться правилом Рунге [5].

7.1.3 Правило Рунге оценки точности интегрирования

Необходимо вычислить определённый интеграл от функции $f(x)$ на интервале $[a; b]$. Для этого участок интегрирования $[a, b]$ разобьём точками на n равных частей, причём $x_0 = a$; $x_i = x_0 + ih$, $i = 1, 2, \ldots, n - 1$; $x_n = b$.

Вычислим интеграл на интервале $[x_i; x_{i+1}]$ с шагом h и обозначим [10]

$$I_i = \int_{x_i}^{x_{i+1}} f(x)dx = I_{h,i} + R_i,$$

где $I_{h,i}$ – любая из рассмотренных ранее квадратурных формул; $R_i = C_i h^k + O(k^{k+1})$ – ошибка квадратурной формулы на интервале $[x_i; x_{i+1}]$; k – порядок точности формулы численного интегрирования ($k = 1$ – для метода прямоугольников, $k = 2$ – в случае метода трапеций, $k = 4$ – при использовании метода Симпсона).

Вычислим этот же интеграл на интервале $[x_i; x_{i+1}]$ с шагом $\frac{h}{2}$ и обозначим $I_{h/2,i}$. Оценку точности интегрирования на интервале $[x_i; x_{i+1}]$ (с помощью любой квадратурной формулы) можно провести с помощью *правила Рунге* [5, 8, 10].

Если

$$\frac{|I_{h,i} - I_{h/2,i}|}{2^k - 1} < \epsilon, \tag{7.4}$$

то вычисления на этом интервале прекращаем. Иначе уменьшаем шаг вдвое и повторяем вычисления до тех пор, пока неравенство не будет выполнено.

7.1.4 Квадратурные формулы Гаусса и Чебышёва

На практике высокую точность интегрирования обеспечивают квадратурные формулы Гаусса и Чебышёва.

Квадратурная формула *Гаусса* численного интегрирования имеет вид [1]:

$$\int_a^b f(t)dt = \frac{b-a}{2}\sum_{i=0}^{n-1} A_i \cdot f\left(\frac{b+a}{2} + \frac{b-a}{2}t_i\right). \tag{7.5}$$

На практике эта формула используется при $n = 2, 3, 4, 5, 6, 7, 8$, значения коэффициентов представлены в табл. 7.1.

Таблица 7.1. Значения t_i и A_i для квадратурной формулы Гаусса

n	**Массив** t	**Массив** A
2	−0.57735027, 0.57735027	1, 1
3	−0.77459667, 0, 0.77459667	5/9, 8/9, 5/9
4	−0.86113631, −0.33998104, 0.33998104, 0.86113631	0.34785484, 0.65214516, 0.65214516, 0.34785484
5	−0.90617985, −0.53846931, 0, 0.53846931, 0.90617985	0.23692688, 0.47862868, 0.568888889, 0.47862868, 0.23692688
6	−0.93246951, −0.66120939, −0.23861919, 0.23861919, 0.66120939, 0.93246951	0.17132450, 0.36076158, 0.46791394, 0.46791394, 0.36076158, 0.17132450
7	−0.94910791, −0.74153119, −0.40584515, 0, 0.40584515, 0.74153119, 0.94910791	0.12948496, 0.27970540, 0.38183006, 0.41795918, 0.38183006, 0.27970540, 0.12948496
8	−0.96028986, −0.79666648, −0.52553242, −0.18343464, 0.18343464, 0.52553242, 0.79666648, 0.96028986	0.10122854, 0.22238104, 0.31370664, 0.36268378, 0.36268378, 0.31370664, 0.22238104, 0.10122854

Квадратурная формула *Чебышёва* численного интегрирования имеет вид [1]:

$$\int_a^b f(t)dt = \frac{b-a}{n}\sum_{i=0}^{n-1} f\left(\frac{b+a}{2} + \frac{b-a}{2}t_i\right). \tag{7.6}$$

На практике формула Чебышёва используется при $n = 2, 3, 4, 5, 6, 7, 9$, значения коэффициентов представлены в табл. 7.2.

Таблица 7.2. Значения t_i для квадратурной формулы Чебышёва

n	**Массив** t
2	−0.577350, 0.577350
3	−0.707107, 0, −0.707107
4	−0.794654, −0.187592, 0.187592, 0.794654
5	0.832498, −0.374541, 0, 0.374541, 0.832498
6	−0.866247, −0.422519, −0.266635, 0.266635, 0.422519, 0.866247
7	−0.883862, −0.529657, −0.323912, 0, 0.323912, 0.529657, 0.883862
9	−0.911589, −0.601019, −0.528762, −0.167906, 0, 0.167906, 0.528762, 0.601019, 0.911589

Рассмотрим реализацию описанных выше квадратурных формул (методы трапеций и Симпсона, формулы Гаусса и Чебышёва) в Scilab (см. листинг 7.1) на примере вычисления определённого интеграла $\int_0^1 \frac{t^2}{\sqrt{3+\sin t}}dt$.

Листинг 7.1. Функции вычисления определённого интеграла

```
function y=f(x)
  // подынтегральная функция
  y=x.^2/sqrt(3+sin(x))
endfunction

//Вычисление интеграла методом трапеций см. (7.2)
function integral=trap(a,b,n)
  //[a;b] - интервал интегрирования,
  //n - количество участков.
  h=(b-a)/n;
  integral=(f(a)+f(b))/2;
  for i=1:n-1
    integral=integral+f(a+i*h)
  end
  integral=integral*h;
endfunction

//вычисление интеграла методом трапеций с заданной точностью eps
//Оценка точности проводится по правилу Рунге (7.4) для
//метода трапеций
function [k,integral]=trap2(a,b,eps)
  //[a;b] - интервал интегрирования,
  //eps - точность интегрирования.
  //Функция возвращает
  //k - количество итераций для достижения точности
  //integral - значение интеграла с точностью eps
  n=5;
  Ih=trap(a,b,n);
  tochnost=100*eps;
  k=1;
  while tochnost>eps
    Ih2=trap(a,b,2*n);
    tochnost=abs(Ih-Ih2)/3;
    k=k+1;
    n=2*n;
    Ih=Ih2;
  end
  integral=Ih2;
endfunction

//Вычисление интеграла методом Симпсона
function integral=simpson(a,b,m)
  //[a;b] - интервал интегрирования,
  //2*n - количество участков, число интервалов должно быть чётным.
  n=2*m
  h=(b-a)/n;
  integral=f(a)+f(b);
  p=4;
```

```
  for i=1:n-1
    integral=integral+p*f(a+i*h)
    //p - меняющийся коэффициент в формуле 4,2,4,2,....
    p=6-p;
  end
  integral=integral*h/3;
endfunction

//вычисление интеграла методом трапеций с заданной точностью eps
//Оценка точности проводится по правилу Рунге (7.4) для метода
//Симпсона

function [k,integral]=simpson2(a,b,eps)
  //[a;b] - интервал интегрирования,
  //eps - точность интегрирования.
  //Функция возвращает
  //k - количество итераций для достижения точности
  //integral - значение интеграла с точностью eps
  n=3;
  Ih=simpson(a,b,n);
  tochnost=100*eps;
  k=1;
  while tochnost>eps
    Ih2=simpson(a,b,2*n);
    tochnost=abs(Ih-Ih2)/15;
    k=k+1;
    Ih=Ih2;
    n=2*n;
  end
  integral=Ih2;
endfunction

//Интегрирование по формуле Гаусса (7.5) при n=8
function integral=gauss(a,b)
  t=[-0.96028986, -0.79666648, -0.52553242, -0.18343464,..
  0.18343464, 0.52553242, 0.79666648, 0.96028986];
  A=[0.10122854, 0.22238104, 0.31370664, 0.36268378, 0.36268378,..
  0.31370664, 0.22238104, 0.10122854];
  integral=0;
  for i=1:length(t)
    integral=integral+A(i)*f((a+b)/2+(a-b)/2*t(i))
  end
  integral=integral*(b-a)/2;
endfunction

//Интегрирование по формуле Чебышёва (7.6) при n=9
function integral=chebishev(a,b)
  t=[-0.911589, -0.601019, -0.528762, -0.167906, 0, 0.167906,..
  0.528762, 0.601019, 0.911589]
  integral=0;
  for i=1:length(t)
    integral=integral+f((a+b)/2+(a-b)/2*t(i))
  end
```

```
  integral=integral*(b-a)/length(t);
endfunction

//Интервал интегрирования
a=0;
b=1;
//Точность интегрирования
eps=1D-9
[m,integra]=trap2(a,b,eps)
mprintf("Точность интегрирования =%14.12le\n",eps);
mprintf("Метод трапеций, интеграл=%14.12le\t сделано %d шагов\n",... integra,m);
mprintf("Точность интегрирования =%14.12le\n",eps); [m,integra]=simpson2(a,b,eps)
mprintf("Метод Симпсона, интеграл=%14.12le\t сделано %d шагов\n",... integra,m);
integra=gauss(a,b)
mprintf("Метод Гаусса, интеграл=%14.12le\n",integra); integra=chebishev(a,b)
mprintf("Метод Чебышёва, интеграл=%14.12lf\n",integra);
```

Результат работы программы представлен ниже.

Листинг 7.2. Результат работы программы вычисления определённого интеграла

```
Точность интегрирования =1.000000000000e-09
Метод трапеций, интеграл=1.741192413368e-01  сделано 12 шагов
Точность интегрирования =1.000000000000e-09
Метод Симпсона, интеграл=1.741192413296e-01  сделано 4 шага
Метод Гаусса, интеграл=1.741192417391e-01
Метод Чебышёва, интеграл=0.174119240869
```

7.2 Встроенные функции интегрирования Scilab

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 определённый 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 не составит труда:

Листинг 7.3. 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 определённого интеграла *метод трапеций*. Рассмотрим несколько вариантов решения данной задачи, используя функцию `inttrap`. В первом случае интервал интегрирования делится на отрезки с шагом 1, во втором – 0.5 и в третьем – 0.1. Нетрудно заметить, что чем больше точек разбиения, тем точнее значение искомого интеграла.

Листинг 7.4. Приближённое 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
```

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, также задаётся в виде символа; `a`, `b` – пределы интегрирования, действительные числа; `er1` и `er2` – параметры, отражающие абсолютную и относительную точности вычислений (действительные числа). Рассмотрим решение задачи 7.1 с использованием функции `integrate`.

Листинг 7.5. Использование функции `integrate`

```
-->integrate('(2*x-1)^0.5','x',5,13)
 ans  =     32.666667
```

Наиболее универсальной командой интегрирования в Scilab является

```
[I,err]=intg(a, b, name [,er1 [,er2]])
```

где `I` – результат интегрирования; `err` – погрешность вычислений; `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 точности вычислений (необязательные параметры).

Рассмотрим решение задачи 7.1 с использованием функции `intg` (листинг 7.6).

Листинг 7.6. Использование функции `intg`

```
-->deff('y=G(x)','y=sqrt(2*x-1)'); intg(5,13,G)
 ans  =     32.666667
```

Задача 7.2

Вычислить интеграл $\int_0^1 \frac{t^2}{\sqrt{3+\sin t}} dt$.

Численное решение интеграла показано в листинге 7.7.

Листинг 7.7. Решение задачи 7.2

```
-->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
```

7.3 Численное дифференцирование в Scilab

Идея численного дифференцирования заключается в том, что функцию $y(x)$, заданную в равноотстоящих точках x_i (i = 0, 1, ..., n) отрезка $[a, b]$ с помощью значений $y_i = f(x_i)$, приближённо заменяют интерполяционным полиномом Ньютона, построенным для системы узлов x_0, x_1, ..., x_k ($k \le n$), и вычисляют производные $y' = f'(x)$, $y'' = f''$(x) и т. д. [2].

$$y'(x_0) = \frac{1}{h}\left(\Delta y_0 - \frac{\Delta^2 y_0}{2} + \frac{\Delta^3 y_0}{3} - \frac{\Delta^4 y_0}{4} + \frac{\Delta^5 y_0}{5} - \cdots\right). \tag{7.7}$$

На практике приближённое 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 Δy, $\Delta_2 y$, ..., $\Delta_k y$. Рассмотрим работу функции на примере.

Задача 7.3

Найти $y'(50)$, $y'(52)$, $y'(54)$ функции $y = \lg(x)$, заданной в виде таблицы.

Программа решения данной задачи с комментариями представлена в листинге 7.8.

Листинг 7.8. Использование функции diff

```
h=2;x=50:2:60;
y=log10(x)
mprintf("Массив значений y\n");
for i=1:length(y)
        mprintf("%lf, ",y(i))
end
mprintf("\n");
//Вычисление массива разделённых разностей первого порядка
dy=diff(y)
mprintf("Массив разделённых разностей первого порядка\n");
for i=1:length(dy)
        mprintf("%lf, ",dy(i))
end
mprintf("\n");
//Вычисление массива разделённых разностей второго порядка
dy2=diff(y,2)
mprintf("Массив разделённых разностей второго порядка\n");
for i=1:length(dy2)
        mprintf("%lf, ",dy2(i))
end
mprintf("\n");
//Вычисление массива разделённых разностей третьего порядка
dy3=diff(y,3)
mprintf("Массив разделённых разностей третьего порядка\n");
for i=1:length(dy3)
        mprintf("%lf, ",dy3(i))
end
mprintf("\n");
//Приближённое значение y'(50),y'(52),y'(54) по формуле (7.7)
for i=1:3
//Учёт одного слагаемого
Y1(i)=dy(i)/h
//Учёт двух слагаемых
Y2(i)=(dy(i)-dy2(i)/2)/h
//Учёт трёх слагаемых
Y3(i)=(dy(i)-dy2(i)/2+dy3(i)/3)/h
//Точное значение производной
//Значение y'(50),y'(52), y'(54) для lg'(x)=1/ln(10)/x
Y(i)=1/log(10)/x(i)
mprintf("Y1=%lf, Y2=%lf, Y3=%lf, Y=%lf\n",Y1(i), Y2(i), Y3(i),.. Y(i));
end
```

Результаты работы программы представлены в листинге 7.9. Как видно из результатов, для вычисления производной в данном случае вполне достаточно трёх слагаемых формулы (7.7).

Листинг 7.9. Вычисление производной функции lg*x* в точках 50, 52, 54 с помощью функции diff

```
1.698970, 1.716003, 1.732394, 1.748188, 1.763428, 1.778151,
Массив разделённых разностей первого порядка
0.017033, 0.016390, 0.015794, 0.015240, 0.014723,
```

```
Массив разделённых разностей второго порядка
-0.000643, -0.000596, -0.000554, -0.000517,
Массив разделённых разностей третьего порядка
0.000047, 0.000042, 0.000038,
Y1=0.008517, Y2=0.008677, Y3=0.008685, Y=0.008686
Y1=0.008195, Y2=0.008344, Y3=0.008351, Y=0.008352
Y1=0.007897, Y2=0.008036, Y3=0.008042, Y=0.008042
```

Более универсальной командой дифференцирования является команда

```
g=numderivative(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 проводиться дифференцирование.

Если параметры `p1, p2, ..., pn` присутствуют в описании функции, то они должны быть обязательно определены при вызове, например так:

```
g=numderivative(list(fun,p1,p2,...pn),x)
```

Результат работы функции – матрица

$$g_{ij} = \frac{df_i}{dx_j}.$$

Рассмотрим решение задачи 7.3 с помощью функции `numderivative`. Листинг 7.10 содержит программу решения данной задачи, а листинг 7.11 – результаты её работы.

Листинг 7.10. Программа решения задачи 7.3 с помощью функции `numderivative`

```
function y=f(x)
        y=log10(x)
endfunction
X=[50 52 54]
G=numderivative(f,X);
disp(G);
for i=1:length(X)
g(i)=G(i,i)
end
mprintf("Вычисленные значения производной в точках\n");
for i=1:length(X)
        mprintf("df/dx(%lf)=%lf\t",X(i),g(i))
end
```

Листинг 7.11. Решение задачи 7.3 с помощью функции `numderivative`

```
0.0086859   0.          0.
0.          0.0083518   0.
0.          0.          0.0080425
Вычисленные значения производной в точках
df/dx(50.0)=0.008686 df/dx(52.0)=0.008352 df/dx(54.0)=0.008042
```

В завершение этой главы решим следующую задачу с использованием функции numderivative.

Задача 7.4

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).

Программа решения задачи представлена в листинге 7.12. Результаты работы программы – в листинге 7.13.

Листинг 7.12. Решение задачи 7.4

```
//Исходная функция
function Y=f(X)
        Y=X(1)*X(2)^X(3)+X(1)^2*X(3)
endfunction
//Функция частных производных
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
X=[1 2 3];
//Вычисление частных производных функции f
//в точке (1,2,3) с помощью функции
//numderivative
G=numderivative(f,X)
mprintf("Частные производные функции f в точке (1,2,3),\n");
mprintf("найденные с помощью функции numderivative\n");
for i=1:length(X)
mprintf("%lf\t",G(i))
end
mprintf("\n");
//Вычисление точного значения частных производных
//функции f в точке (1,2,3) g=f1(X)
mprintf("Точные значения частных производных функции f\n");
mprintf("в точке (1,2,3)\n");
for i=1:length(X)
        mprintf("%lf\t",g(i))
end
mprintf("\n");
```

Листинг 7.13. Результаты решения задачи 7.4

```
Частные производные функции f в точке (1,2,3),
найденные с помощью функции numderivative
14.000000 12.000000 6.545177
Точные значения частных производных функции f
в точке (1,2,3)
14.000000 12.000000 6.545177
```

7.4 Примеры решения некоторых задач

Задача 7.5
Найти производную от функции вида $f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$.

Чтобы найти производную от функции вида $f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$, представляющую собой полином (многочлен), применяют команду `derivat(p)`, где `p` – полином, определённый ранее.

```
//Найти первую и вторую производные
//f(x)=x^4-3x^2+1
p=poly([1 0 -3 0 1],'x','c');
p1=derivat(p)
p2=derivat(p1)
disp(p,p1,p2)
//Найти первую и вторую производные
//f(x)=(3x-2)/(x^2+1)
u=poly([-2 3],'x','c');
v=poly([1 0 1],'x','c');
P=u/v
P1=derivat(P)
P2=derivat(P1)
disp(P,P1,P2)

//Результат работы программы
       2
-6 +12x

       3
-6x +4x

     2  4
1 -3x +x

            2     3     4    5
4 - 18x - 8x - 12x - 12x + 6x
------------------------------
      2    4    6   8
1 + 4x + 6x + 4x + x

           2
3 + 4x - 3x
------------
      2   4
1 + 2x + x

-2 + 3x
--------
     2
1 + x
```

Задача 7.6

Построить касательную к графику функции $f(x) = \frac{e^x}{5} - 2(x-1)^2$ в точке $x_0 = -1$.

Решение задачи представлено в листинге 7.14 и на рис. 7.3.

Листинг 7.14. Результаты решения задачи 7.6

```
clf
clear
function y=f(x)
        y=exp(x)/5-2*(x-1)^2
endfunction
function y=f1(x,X0)
        y=f(X0)+numderivative(f,X0)*(x-X0)
endfunction
x0=-1;
t=-3:0.1:3
plot(t,f(t),t,f1(t,x0),x0,f(x0),'o')
xgrid
```

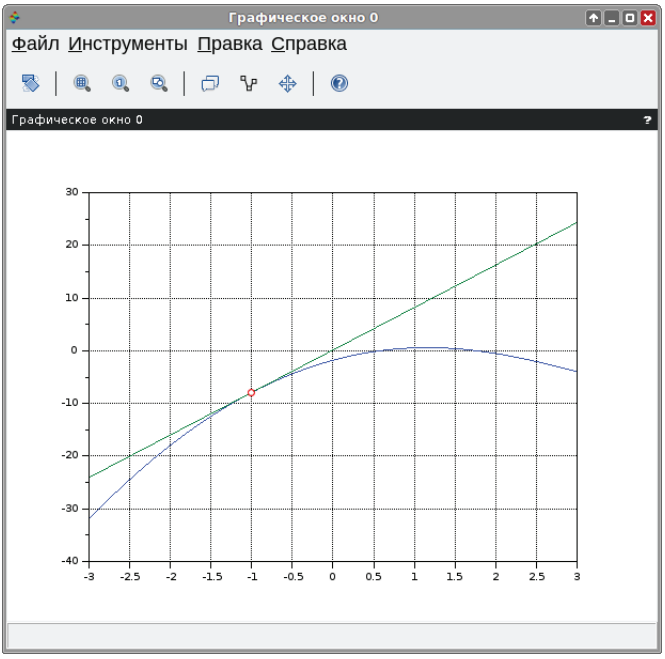

Рис. 7.3. Решение задачи 7.6

Задача 7.7

Оценить ошибку вычисления функции $F = \frac{a^2+b^5}{\cos(t)}$ при $a = 28.3 \pm 0.02$, $b = 7.45 \pm 0.01$, $t = 0.7854 \pm 0.0001$.

Абсолютная погрешность функции y для малых $\Delta(x_i)$ вычисляется по формуле:

$$|\Delta F| = \sum_{i=1}^{n} \left| \frac{\partial f(x_i)}{\partial x_i} \right| \cdot \Delta(x_i).$$

В нашем случае формула имеет вид:

$$|\Delta F| = \frac{\partial F}{\partial a} \cdot \Delta a + \frac{\partial F}{\partial b} \cdot \Delta b + \frac{\partial F}{\partial t} \cdot \Delta t.$$

Относительную погрешность вычислим по формуле:

$$|\delta F| = \frac{\Delta F}{F}.$$

Листинг 7.15. Решение задачи 7.7

```
clear
a=28.3;Da=0.02;b=7.45;Db=0.01;t=0.7854;Dt=0.0001;
X=[a b t];D=[Da Db Dt]
function [Y]=F(x)
Y=(x(1)^2+x(2)^3)/cos(x(3))
end
U=numderivative(F,X)
DF=sum(U.*D)
dF=DF/F(X)
disp('Относительная погрешность',dF)
disp('Абсолютная погрешность',DF)

//Результат работы программы

"Относительная погрешность"

0.0024033

"Абсолютная погрешность"

4.1274094
```

Задача 7.8

Найти площадь фигуры, ограниченной линиями $y = f(x)$, $x = \pi/7$, $x = \pi/3$.

Решение задачи представлено в листинге 7.16 и на рис. 7.4.

Листинг 7.16. Решение задачи 7.8

```
clear clf;
function y=f(x)
        y=1./((1+sin(x)-cos(x)).^2)
endfunction
a=%pi/7;b=%pi/3;h=0.01
t=a:h:b;
plot2d3(t,f(t))
plot2d(t,f(t))
//Площадь фигуры, ограниченной
//кривой f(x) на отрезке [a;b]
S=intg(a,b,f)
//Преобразование вещественного
//числа в строку символов
S1=string(S)
xset("font",0,3)
xstring(0.5,3,["Площадь фигуры S = ", S1],0,1)
```

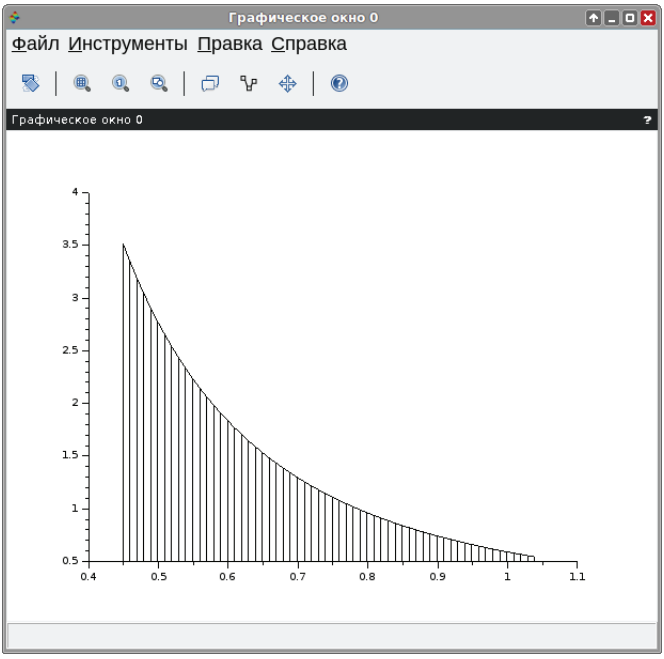

Рис. 7.4. Результаты решения задачи 7.8

Глава 8

Решение обыкновенных дифференциальных уравнений и систем

Дифференциальные уравнения и системы описывают многие динамические процессы и возникают при решении различных задач физики, электротехники, химии и других наук. Данная глава посвящена численному решению дифференциальных уравнений и систем средствами *Scilab*.

8.1 Общие сведения о дифференциальных уравнениях

Дифференциальным уравнением n-го порядка называется соотношение вида

$$H(t, x, x', x'', \ldots, x^{(n)}) = 0. \tag{8.1}$$

Решением дифференциального уравнения называется функция $x(t)$, которая обращает уравнение в тождество.

Системой дифференциальных уравнений n-го порядка называется система вида

$$\begin{cases} x_1' = f_1(t, x_1, x_2, \ldots, x_n) \\ x_2' = f_2(t, x_1, x_2, \ldots, x_n) \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ x_n' = f_n(t, x_1, x_2, \ldots, x_n) \end{cases}. \tag{8.2}$$

Системой линейных дифференциальных уравнений называется система вида

$$\begin{cases} x_1' = \sum_{j=1}^{n} a_{1j} x_j + b_1 \\ x_2' = \sum_{j=1}^{n} a_{2j} x_j + b_2 \\ \dots\dots\dots\dots\dots \\ x_n' = \sum_{j=1}^{n} a_{nj} x_j + b_n \end{cases}. \tag{8.3}$$

Решением системы называется вектор $x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{pmatrix}$, который обращает уравнения систем (8.2), (8.3) в тождества.

Каждое дифференциальное уравнение, так же как и система, имеет бесконечное множество решений, которые отличаются друг от друга константами. Для однозначного определения решения необходимо определить 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*, в случае если все дополнительные условия заданы в одной (чаще начальной) точке интервала;
- *краевую задачу*, в случае когда дополнительные условия заданы на границах интервала.

Различают *точные* (аналитические) и *приближённые* (численные) методы решения дифференциальных уравнений. Большое количество уравнений может быть решено точно. Однако есть уравнения, а особенно системы уравнений, для которых нельзя записать точное решение. Но даже для уравнений с известным аналитическим решением очень часто необходимо вычислить числовое значение при определённых исходных данных. Поэтому широкое распространение получили численные методы решения обыкновенных дифференциальных уравнений.

8.2 Численные методы решения дифференциальных уравнений

Численные методы решения дифференциального уравнения первого порядка будем рассматривать для следующей задачи Коши. Найти решение дифференциального уравнения:

$$x' = f(x, t), \tag{8.4}$$

удовлетворяющего начальному условию

$$x(t_0) = x_0. \tag{8.5}$$

Иными словами, требуется найти интегральную кривую $x = x(t)$, проходящую через заданную точку $M_0(t_0, x_0)$ (рис. 8.1).

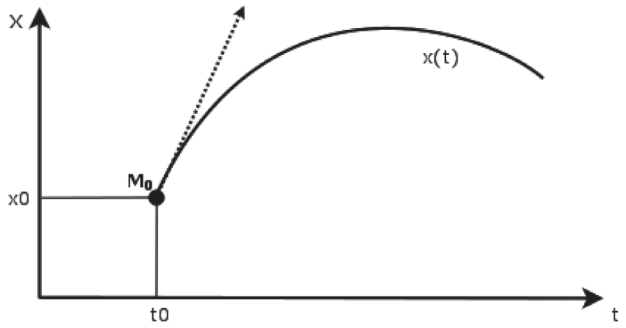

Рис. 8.1. Интегральная кривая, проходящая через точку $M_0(t_0, x_0)$

Для дифференциального уравнения n-го порядка

$$x^{(n)} = f(t, x, x', x'', \ldots, x^{(n-1)}) \tag{8.6}$$

задача Коши состоит в нахождении решения $x = x(t)$, удовлетворяющего уравнению (8.6) и начальным условиям

$$x(t_0) = x_0, x'(t_0) = x'_0, \ldots, x^{(n-1)}(t_0) = x_0^{(n-1)}. \tag{8.7}$$

Рассмотрим основные численные методы решения задачи Коши [8, 3].

8.2.1 Решение дифференциальных уравнений методом Эйлера

При решении задачи Коши (8.3), (8.4) на интервале $[t_0, t_n]$, выбрав достаточно малый шаг h, построим систему точек

$$t_i = t_0 + ih, \quad i = 0, 1, \ldots, n, \quad h = \frac{t_n - t_0}{n}. \tag{8.8}$$

Для вычисления значения функции в точке t_1 разложим функцию $x = x(t)$ в окрестности точки t_0 в ряд Тейлора [8]:

$$x(t_1) = x(t_0 + h) = x(t_0) + x'(t_0)h + x''(t_0)\frac{h^2}{2} + \ldots. \tag{8.9}$$

При достаточно малом значении h членами выше второго порядка можно пренебречь, и с учётом $x'(t_0) = f(x_0, t_0)$ получим следующую формулу для вычисления приближённого значения функции $x(t)$ в точке t_1:

$$x_1 = x_0 + hf(x_0, t_0). \tag{8.10}$$

Рассматривая найденную точку (x_1, t_1) как начальное условие задачи Коши, запишем аналогичную формулу для нахождения значения функции $x(t)$ в точке t_2:

$$x_2 = x_1 + hf(x_1, t_1).$$

Повторяя этот процесс, сформируем последовательность значений x_i в точках t_i по формуле [8, 3]:

$$x_{i+1} = x_i + hf(x_i, t_i), \quad i = 0, 1, \ldots, n-1. \tag{8.11}$$

Процесс нахождения значений функции x_i в узловых точках t_i по формуле (8.11) называется *методом Эйлера*. Геометрическая интерпретация метода Эйлера состоит в замене интегральной кривой $x(t)$ ломаной $M_0, M_1, M_2, \ldots, M_n$ с вершинами $M_i(x_i, y_i)$. Звенья ломаной Эйлера M_iM_{i+1} в каждой вершине M_i имеют направление $y_i = f(t_i, x_i)$, совпадающее с направлением интегральной кривой $x(t)$, проходящей через точку M_i (рис. 8.2). Последовательность ломаных Эйлера при $h \to 0$ на достаточно малом отрезке $[x_i, x_i + h]$ стремится к искомой интегральной кривой.

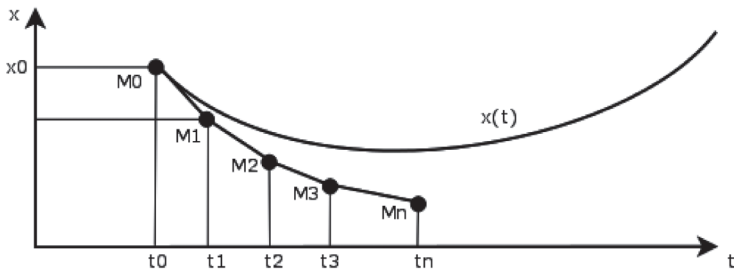

Рис. 8.2. Геометрическая интерпретация метода Эйлера

На каждом шаге решение $x(t)$ определяется с ошибкой за счёт отбрасывания членов ряда Тейлора выше первой степени, что в случае быстро меняющейся функции $f(t, x)$ может привести к быстрому накапливанию ошибки. В методе Эйлера следует выбирать достаточно малый шаг h.

8.2.2 Решение дифференциальных уравнений при помощи модифицированного метода Эйлера

Более точным методом решения задачи (8.3)–(8.4) является *модифицированный метод Эйлера*, при котором сначала вычисляют промежуточные значения [8]:

$$t_p = t_y + \frac{h}{2}, \quad x_p = x_i + \frac{h}{2} f(x_i, t_i), \tag{8.12}$$

после чего находят значение x_{i+1} по формуле

$$x_{i+1} = x_i + hf(x_p, t_p), \quad i = 0, 1, \ldots, n-1. \tag{8.13}$$

8.2.3 Решение дифференциальных уравнений методами Рунге–Кутта

Рассмотренные выше методы Эйлера (как обычный, так и модифицированный) являются частными случаями явного *метода Рунге–Кутта* k-го порядка. В общем случае формула вычисления очередного приближения методом Рунге–Кутта имеет вид [8]:

$$x_{i+1} = x_i + h\varphi(t_i, x_i, h), \quad i = 0, 1, \ldots, n-1. \tag{8.14}$$

Функция $\varphi(t, x, h)$ приближает отрезок ряда Тейлора до k-го порядка и не содержит частных производных $f(t, x)$ [8].

Метод Эйлера является *методом Рунге–Кутта первого порядка* ($k = 1$) и получается при $\varphi(t, x, h) = f(t, x)$.

Семейство *методов Рунге–Кутта второго порядка* имеет вид [8]:

$$x_{i+1} = x_i + h\left((1-\alpha)f(t_i, x_i) + \alpha f\left(t_i + \frac{h}{2\alpha}, x_i + \frac{h}{2\alpha}f(t_i, x_i)\right)\right), \tag{8.15}$$
$$i = 0, 1, \ldots, n-1.$$

Два наиболее известных среди методов Рунге–Кутта второго порядка [8] – это метод Хойна ($\alpha = \frac{1}{2}$) и модифицированный метод Эйлера ($\alpha = 1$).

Подставив $\alpha = \frac{1}{2}$ в формулу (8.15), получаем расчётную формулу *метода Хойна* [8]:

$$x_{i+1} = x_i + \frac{h}{2}(f(t_i, x_i) + f(t_i + h, x_i + hf(t_i, x_i))), \quad i = 0, 1, \ldots, n-1.$$

Подставив $\alpha = 1$ в формулу (8.15), получаем расчётную формулу уже рассмотренного выше модифицированного метода Эйлера:

$$x_{i+1} = x_i + h\left(f\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}f(t_i, x_i)\right)\right), \quad i = 0, 1, \ldots, n-1.$$

Наиболее известным является *метод Рунге–Кутта четвёртого порядка*, расчётные формулы которого можно записать в виде [8]:

$$\begin{cases} x_{i+1} = x_i + \Delta x_i, \quad i = 0, 1, \ldots, n-1 \\ \Delta x_i = \frac{h}{6}(K_1^i + 2K_2^i + 2K_3^i + K_4^i) \\ K_1^i = f(t_i, x_i) \\ K_2^i = f\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_1^i\right) \\ K_3^i = f\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_2^i\right) \\ K_4^i = f\left(t_i + h, x_i + hK_3^i\right) \end{cases}.$$

Одной из модификаций метода Рунге–Кутта является *метод Кутта–Мерсона* (или *пятиэтапный метод Рунге–Кутта четвёртого порядка*), который состоит в следующем [8].

1. На i-м шаге рассчитываются коэффициенты

$$\begin{aligned} K_1^i &= f(t_i, x_i), \\ K_2^i &= f\left(t_i + \frac{h}{3}, x_i + \frac{3}{2}K_1^i\right), \\ K_3^i &= f\left(t_i + \frac{h}{3}, x_i + \frac{h}{6}K_1^i + \frac{h}{6}K_2^i\right), \\ K_4^i &= f\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_1^i - \frac{3h}{2}K_3^i + 2hK_4^i\right). \end{aligned} \tag{8.16}$$

2. Вычисляем приближённое значение $x(t_{i+1})$ по формуле

$$\tilde{x}_{i+1} = x_i + \frac{h}{2}(K_1^i - 3K_3^i + 4K_4^i). \tag{8.17}$$

3. Вычисляем приближённое значение $x(t_{i+1})$ по формуле

$$x_{i+1} = x_i + \frac{h}{6}(K_1^i + 4K_3^i + K_5^i). \tag{8.18}$$

4. Вычисляем оценочный коэффициент по формуле

$$R = 0.2|x_{i+1} - \tilde{x}_{i+1}|. \tag{8.19}$$

5. Сравниваем R с точностью вычислений ε. Если $R \geq \varepsilon$, то уменьшаем шаг вдвое и возвращаемся к п. 1. Если $R < \varepsilon$, то значение, вычисленное по формуле (8.18), и будет вычисленным значением $x(t_{i+1})$(с точностью ε).
6. Перед переходом к вычислению следующего значения x сравниваем R с $\frac{\varepsilon}{64}$. Если $R < \frac{\varepsilon}{64}$, то дальнейшие вычисления можно проводить с удвоенным шагом $h = 2h$.

Рассмотренные методы Рунге–Кутта относятся к классу одношаговых методов, в которых для вычисления значения в очередной точке x_{k+1} нужно знать значение в предыдущей точке x_k.

Ещё один класс методов решения задачи Коши – *многошаговые методы*, в которых используются точки x_{k-3}, x_{k-2}, x_{k-1}, x_k для вычисления x_{k+1}. В многошаговых методах первые четыре начальные точки (t_0, x_0), (t_1, x_1), (t_2, x_2), (t_3, x_3) должны быть получены заранее любым из одношаговых методов (метод Эйлера, Рунге–Кутта и т. д.). Наиболее известными *многошаговыми методами* являются методы *прогноза-коррекции Адамса* и *Милна*.

8.2.4 Решение дифференциальных уравнений методом прогноза-коррекции Адамса

Рассмотрим решение уравнения (8.3), (8.4) на интервале $[t_i, t_{i+1}]$. Будем считать, что решение в точках $t_0, t_1, t_2, \ldots, t_i$ уже найдено, и значения в этих точках будем использовать для нахождения значения $x(t_{i+1})$.

Проинтегрируем уравнение (8.4) на интервале $[t_i, t_{i+1}]$ и получим соотношение [8]

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(t, x(t))dt. \tag{8.20}$$

При вычислении интеграла, входящего в (8.20), вместо функции $f(t, x(t))$ будем использовать интерполяционный полином Лагранжа, построенный по точкам (t_{i-3}, x_{i-3}), (t_{i-2}, x_{i-2}), (t_{i-1}, x_{i-1}), (t_i, x_i). Подставив полином Лагранжа в (8.20), получаем первое приближение (прогноз) $\tilde{x}_{i+1}$ для значения функции в точке t_{i+1}

$$\begin{aligned}\tilde{x}_{i+1} = x_i + \frac{h}{24}(&-9f(t_{i-3}, x_{i-3}) + 37f(t_{i-2}, x_{i-2}) \\ &- 59f(t_{i-1}, x_{i-1}) + 55f(t_i, x_i)).\end{aligned} \tag{8.21}$$

Как только $\tilde{x}_{i+1}$ вычислено, его можно использовать. Следующий полином Лагранжа для функции $f(t, x(t))$ построим по точкам (t_{i-2}, x_{i-2}), (t_{i-1}, x_{i-1}), (t_i, x_i) и новой точке $(t_{i+1}, \tilde{x}_{i+1})$, после чего подставляем его в (9.20) и получаем второе приближение (корректор):

$$\begin{aligned}x_{i+1} = x_i + \frac{h}{24}(&f(t_{i-2}, x_{i-2}) - 5f(t_{i-1}, x_{i-1}) \\ &+ 19f(t_i, x_i) + 9f(t_{i+1}, \tilde{x}_{i+1})).\end{aligned} \tag{8.22}$$

Таким образом, для вычисления значения $x(t_{i+1})$ методом Адамса необходимо последовательно применять формулы (8.21), (8.22), а первые четыре точки можно получить методом Рунге–Кутта.

8.2.5 Решение дифференциальных уравнений методом Милна

Отличие *метода Милна* от метода Адамса состоит в использовании в качестве интерполяционного полинома Ньютона.

Подставив в (8.20) вместо функции $f(t, x(t))$ интерполяционный полином Ньютона, построенный по точкам (t_{k-3}, x_{k-3}), (t_{k-2}, x_{k-2}), (t_{k-1}, x_{k-1}), (t_k, x_k), получаем первое приближение – прогноз Милна $\tilde{x}_{k+1}$ для значения функции в точке t_{k+1} [8]:

$$\tilde{x}_{k+1} = x_{k-3} + \frac{4h}{3}(2f(t_{k-2}, x_{k-2}) - f(t_{k-1}, x_{k-1}) + 2f(t_k, x_k)). \tag{8.23}$$

Следующий полином Ньютона для функции $f(t, x(t))$ построим по точкам (t_{k-2}, x_{k-2}), (t_{k-1}, x_{k-1}), (t_k, x_k) и новой точке $(t_{k+1}, \tilde{x}_{k+1})$, после чего подставляем его в (8.22) и получаем второе приближение – корректор Милна [8]:

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, \tilde{x}_{k+1})). \tag{8.24}$$

В методе Милна для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (8.23), (8.24), а первые четыре точки можно получить методом Рунге–Кутта.

Существует *модифицированный метод Милна*. В нём сначала вычисляется первое приближение по формуле (8.23), затем вычисляется управляющий параметр [8]

$$m_{k+1} = \tilde{x}_{k+1} + \frac{28}{29}(x_k - \tilde{x}_k), \tag{8.25}$$

после чего вычисляется значение второго приближения – корректор Милна по формуле

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, m_{k+1})). \tag{8.26}$$

В модифицированном методе Милна первые четыре точки можно получить методом Рунге–Кутта, а для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (8.23), (8.25), (8.26).

Рассмотрим использование приведённых выше методов на примере решения следующей задачи Коши.

Задача 8.1

Решить задачу Коши

$$\begin{cases} x'(t) = 6x - 13t^3 - 22t^2 + 17t - 11 + \sin t \\ x(0) = 2 \end{cases}. \tag{8.27}$$

Известно точное решение задачи:

$$x(t) = \frac{119}{296}e^{6t} + \frac{1}{24}(52t^3 + 114t^2 - 30t + 39) - \frac{1}{37}(6\sin t + \cos t).$$

В листинге 8.1 представлено решение задачи 8.1 методами:

- модифицированным методом Эйлера;
- Рунге–Кутта;
- Кутта–Мерсона;
- Адамса;
- Милна.

Графическое решение задачи представлено на рис. 8.3.

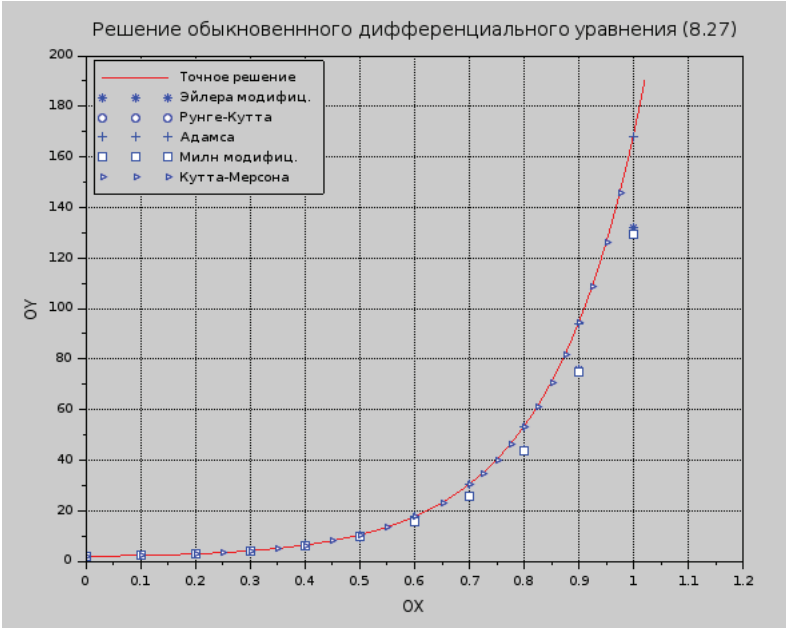

Рис. 8.3. Графическое решение задачи 8.1

Листинг 8.1. Решение задачи 8.1

```
//Точное решение
function q=fi(x)
  q=119/296*exp(6*x)+1/24*(52*x.^3+114*x.^2-30*x+39)-
  6*sin(x)/37-cos(x)/37;
end

//Правая часть дифференциального уравнения.
function y=g(t,x)
  y=6*x-13*t^3-22*t^2+17*t-11+sin(t);
end

//Функция решения задачи Коши модифицированным методом Эйлера.
function [x,t]=eiler_m(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;
  //Формула (8.8)
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end
  //Формулы (8.12)-(8.13)
  for i=2:n+1
    tp=t(i-1)+h/2;
    xp=x(i-1)+h/2*g(t(i-1),x(i-1));
    x(i)=x(i-1)+h*g(tp,xp);
  end
end
```

```
//Функция решения задачи Коши методом Рунге-Кутта, п. 8.2.3
function [x,t]=runge_kut(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end
  for i=2:n+1
    K1=g(t(i-1),x(i-1));
    K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
    K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);
    K4=g(t(i-1)+h,x(i-1)+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
  end
end

//Функция решения задачи Коши методом Кутта-Мерсона.
function [x,t,j]=kut_merson(a,b,n,eps,x0)
  h=(b-a)/n;
  x(1)=x0;t(1)=a;i=2;
  while (t(i-1)+h)<=b
    R=3*eps;
    while R>eps
      //Формулы (8.16)
      K1=g(t(i-1),x(i-1));
      K2=g(t(i-1)+h/3,x(i-1)+h/3*K1);
      K3=g(t(i-1)+h/3,x(i-1)+h/6*K1+h/6*K2);
      K4=g(t(i-1)+h/2,x(i-1)+h/8*K1+3*h/8*K2);
      K5=g(t(i-1)+h,x(i-1)+h/2*K1-3*h/2*K3+2*h*K4);
      //Формула (8.17)
      X1=x(i-1)+h/2*(K1-3*K3+4*K4);
      //Формула (8.18)
      X2=x(i-1)+h/6*(K1+4*K4+K5);
      //Оценочный коэффициент R (8.19)
      R=0.2*abs(X1-X2);
      if R>eps
        h=h/2;
      else
        t(i)=t(i-1)+h;
        x(i)=X2;
        i=i+1;
        if R<=eps/64
          if (t(i-1)+2*h)<=b
            h=2*h;
          end
        end
      end
    end
  end
j=i-1
end
```

```
//Функция решения задачи Коши модифицированным
//методом Милна.
function [x,t]=miln(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;xp(1)=x(1);
  for i=1:n+1
    t(i)=a+(i-1)*h;
end

//Расчёт первых 4 точек методом Рунге-Кутта
for i=2:4
  K1=g(t(i-1),x(i-1));
  K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
  K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);
  K4=g(t(i-1)+h,x(i-1)+h*K3);
  delt=h/6*(K1+2*K2+2*K3+K4);
  x(i)=x(i-1)+delt;
  xp(i)=x(i); end
  //Формулы (8.23), (8.25), (8.26).
  for i=4:n
    xp(i+1)=x(i-3)+4*h/3*(2*g(t(i-2),x(i-2))-g(t(i-1),x(i-1))...
    +g(t(i),x(i)));
    m=xp(i+1)+28/29*(x(i)-xp(i));
    x(i+1)=x(i-1)+h/3*(g(t(i-1),x(i-1))+4*g(t(i),x(i))+...
    g(t(i+1),m));
  end
end

//Функция решения задачи Коши методом Адамса.
function [x,t]=adams(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end
  //Расчёт первых 4 точек методом Рунге-Кутта
  for i=2:4
    K1=g(t(i-1),x(i-1));
    K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
    K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);
    K4=g(t(i-1)+h,x(i-1)+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
  end
  //Формулы (8.21), (8.22).
  for i=4:n
    xp=x(i)+h/24*(-9*g(t(i-3),x(i-3))+37*g(t(i-2),x(i-2))-...
    59*g(t(i-1),x(i-1))+55*g(t(i),x(i)));
    x(i+1)=x(i)+h/24*(g(t(i-2),x(i-2))-5*g(t(i-1),x(i-1))+...
    19*g(t(i),x(i))+9*g(t(i+1),xp));
  end
end
```

```
//Решение дифференциального уравнения модифицированным методом
// Эйлера.
[YE_M,XE_M]=eiler_m(0,1,10,2);
//Решение дифференциального уравнения методом Рунге-Кутта.
[YR,XR]=runge_kut(0,1,10,2);
//Решение дифференциального уравнения методом Кутта-Мерсона.
[YKM,XKM,KM]=kut_merson(0,1,5,0.001,2);
//Решение дифференциального уравнения методом Адамса.
[YA,XA]=adams(0,1,10,2);
//Решение дифференциального уравнения методом Милна.
[YM,XM]=miln(0,1,10,2);
//Точное решение. x1=0:0.05:1;y1=fi(x1);
//Построение графиков.
figure();
plot(x1,y1,"r-",XE_M,YE_M,"b*",XR,YR,"bo",XA,YA,"b^",...
XM,YM,"b>",XKM,YKM,"b<");
xtitle("Решение обыкновеннного дифференциального уравнения (8.27)",...
"OX","OY");
legend("Точное решение","Эйлера модифиц.","Рунге-Кутта",...
"Адамса","Милна модифиц.","Кутта-Мерсона" ,2);
xgrid;
```

В листинге 8.2 приведен код визуального приложения для решения задачи 8.1 с комментариями.

Листинг 8.2. Текст визуального приложения решения задачи 8.1

```
clear
//Решение задачи
//Точное решение
function q=fi(x)
  q=119/296*exp(6*x)+1/24*(52*x.^3+114*x.^2-30*x+39)-6*sin(x)/37...
  -cos(x)/37;
end

//Правая часть дифференциального уравнения.
function y=g(t,x)
  y=6*x-13*t^3-22*t^2+17*t-11+sin(t);
end

//Функция решения задачи Коши модифицированным методом Эйлера.
function [x,t]=eiler_m(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;
  //Формула (8.8)
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end
  //Формулы (8.12-8.13)
  for i=2:n+1
    tp=t(i-1)+h/2;
    xp=x(i-1)+h/2*g(t(i-1),x(i-1));
    x(i)=x(i-1)+h*g(tp,xp);
  end
end
```

```
//Функция решения задачи Коши методом Рунге-Кутта, п. 8.2.3
function [x,t]=runge_kut(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end
  for i=2:n+1
    K1=g(t(i-1),x(i-1));
    K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
    K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);
    K4=g(t(i-1)+h,x(i-1)+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
  end
end

//Функция решения задачи Коши методом Кутта-Мерсона.
function [x,t,j]=kut_merson(a,b,n,eps,x0)
  h=(b-a)/n;
  x(1)=x0;t(1)=a;i=2;
  while (t(i-1)+h)<=b
    R=3*eps;
    while R>eps
      //Формулы (8.16)
      K1=g(t(i-1),x(i-1));
      K2=g(t(i-1)+h/3,x(i-1)+h/3*K1);
      K3=g(t(i-1)+h/3,x(i-1)+h/6*K1+h/6*K2);
      K4=g(t(i-1)+h/2,x(i-1)+h/8*K1+3*h/8*K2);
      K5=g(t(i-1)+h,x(i-1)+h/2*K1-3*h/2*K3+2*h*K4);
      //Формула (8.17)
      X1=x(i-1)+h/2*(K1-3*K3+4*K4);
      //Формула (8.18)
      X2=x(i-1)+h/6*(K1+4*K4+K5);
      //Оценочный коэффициент R (8.19)
      R=0.2*abs(X1-X2);
      if R>eps
        h=h/2;
      else
        t(i)=t(i-1)+h;
        x(i)=X2;
        i=i+1;
        if R<=eps/64
          if (t(i-1)+2*h)<=b
            h=2*h;
          end
        end
      end
    end
  end
j=i-1
end
```

```
//Функция решения задачи Коши модифицированным
//методом Милна.
function [x,t]=miln(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;xp(1)=x(1);
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end

//Расчёт первых 4 точек методом Рунге-Кутта
  for i=2:4
    K1=g(t(i-1),x(i-1));
    K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
    K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);
    K4=g(t(i-1)+h,x(i-1)+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
    xp(i)=x(i);
  end
//Формулы (8.23), (8.25), (8.26).
  for i=4:n
    xp(i+1)=x(i-3)+4*h/3*(2*g(t(i-2),x(i-2))-g(t(i-1),x(i-1))...
    +g(t(i),x(i)));
    m=xp(i+1)+28/29*(x(i)-xp(i));
    x(i+1)=x(i-1)+h/3*(g(t(i-1),x(i-1))+4*g(t(i),x(i))+...
    g(t(i+1),m));
  end
end

//Функция решения задачи Коши методом Адамса.
  function [x,t]=adams(a,b,n,x0)
  h=(b-a)/n;
  x(1)=x0;
  for i=1:n+1
    t(i)=a+(i-1)*h;
  end
//Расчёт первых 4 точек методом Рунге-Кутта
  for i=2:4
    K1=g(t(i-1),x(i-1));
    K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
    K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);
    K4=g(t(i-1)+h,x(i-1)+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
  end
//Формулы (8.21), (8.22).
  for i=4:n
    xp=x(i)+h/24*(-9*g(t(i-3),x(i-3))+37*g(t(i-2),x(i-2))-...
    59*g(t(i-1),x(i-1))+55*g(t(i),x(i)));
    x(i+1)=x(i)+h/24*(g(t(i-2),x(i-2))-5*g(t(i-1),x(i-1))+...
    19*g(t(i),x(i))+9*g(t(i+1),xp));
  end
end
```

```
//Построение графиков.

//Создание окна
f=figure('figure_position',[300 300],'figure_size',[1000 600],...
'resize','off','background',8);
//Создание подписей L00=uicontrol(f,'Style','text','string',...
'   Методы:','position',[0,340,180,30],'BackgroundColor',...[1,1,1]);
L1=uicontrol(f,'Style','text','string',...
'   Точное решение','position',[0,300,180,30],'BackgroundColor',...[1,1,1]);
L2=uicontrol(f,'Style','text','string','   Модиф. Эйлера',...
'position',[0,260,180,30],'BackgroundColor',[1,1,1]);
L3=uicontrol(f,'Style','text','string','   Рунге-Кутта',...
'position',[0,220,180,30],'BackgroundColor',[1,1,1]);
L4=uicontrol(f,'Style','text','string','   Кутта-Мерсона',...
'position',[0,180,180,30],'BackgroundColor',[1,1,1]);
L5=uicontrol(f,'Style','text','string','   Адамса',...
'position',[0,140,180,30],'BackgroundColor',[1,1,1]);
L6=uicontrol(f,'Style','text','string','   Милна',...
'position',[0,100,180,30],'BackgroundColor',[1,1,1]);
Li=uicontrol(f,'Style','text','string','   Конец интервала:',...
'position',[0,60,180,30],'BackgroundColor',[1,1,1]);

L01=uicontrol(f,'Style','text','string','   График:',...
'position',[200,340,100,30],'BackgroundColor',[1,1,1]);
//Создание флажков
ChB1=uicontrol(f,'Style','checkbox','position',[235,300,20,20],...
'BackgroundColor',[1,1,1]);
ChB2=uicontrol(f,'Style','checkbox','position',[235,260,20,20],...
'BackgroundColor',[1,1,1]);
ChB3=uicontrol(f,'Style','checkbox','position',[235,220,20,20],...
'BackgroundColor',[1,1,1]);
ChB4=uicontrol(f,'Style','checkbox','position',[235,180,20,20],...
'BackgroundColor',[1,1,1]);
ChB5=uicontrol(f,'Style','checkbox','position',[235,140,20,20],...
'BackgroundColor',[1,1,1]);
ChB6=uicontrol(f,'Style','checkbox','position',[235,100,20,20],...
'BackgroundColor',[1,1,1]);

//Запись флажков в единый массив
ChB=[ChB1 ChB2 ChB3 ChB4 ChB5 ChB6];
//Создание кнопки построения графика
BtnPlot=uicontrol(f,'Style','pushbutton','callback','visual',...
'String','Построить','position',[20,20,130,30]);
Ei=uicontrol(f,'Style','edit','string','1','position',...[200,60,100,30]);
//Массив с названиями графика
titl=["Точное решение","модиф. Эйлера",...
"Рунге-Кутта","Кутта-Мерсона","Адамса","Милна"];
titleLeg=[];
j=1;
function visual()
//Создание фрейма и осей в нём
fr=uicontrol(f,'Style','frame','position',[300,-20,700,500]);
a = newaxes();
a.parent=fr;
```

```
//Считывание конца интервала из edit
[Nk,err]=evstr(Ei.String);
if err<>0 then set(f,'info_message',...
'Введите число больше 0. Для вещественных чисел разделитель - точка');
else set(f,'info_message','');
end
//Создание массива x
x1=0:0.05:Nk;
//Проходимся по массиву из флажков
for i=1:length(ChB)
  //Если какой-то выбран
  if ChB(i).value == 1 then
    //Название графика внести в массив для вывода на легенде
    titleLeg(j)=titl(i);
    j=j+1;
    //Построить сам график
    select i
      case 1 then
      //Точное решение.
      y1=fi(x1);
      plot(x1,y1,"r-");
      case 2 then
      //Решение дифференциального уравнения модифицированным методом Эйлера.
      [YE_M,XE_M]=eiler_m(0,Nk,10,2);
      plot(XE_M,YE_M,"b*");
      case 3 then
      //Решение дифференциального уравнения методом Рунге-Кутта.
      [YR,XR]=runge_kut(0,Nk,10,2);
      plot(XR,YR,"go");
      case 4 then
      //Решение дифференциального уравнения методом Кутта-Мерсона.
      [YKM,XKM,KM]=kut_merson(0,Nk,5,0.001,2);
      plot(XKM,YKM,"m<");
      case 5 then
      //Решение дифференциального уравнения методом Адамса.
      [YA,XA]=adams(0,Nk,10,2);
      plot(XA,YA,"k^");
      case 6 then
      //Решение дифференциального уравнения методом Милна.
      [YM,XM]=miln(0,Nk,10,2);
      plot(XM,YM,"b>");
    end
  end
end
//Вывод легенды
legend(titleLeg,2)
//Вывод сетки
xgrid;
//Вывод заголовка
xtitle("Решение обыкновенного дифференциального уравнения",..."OX","OY");
endfunction
```

На рис. 8.4 представлено окно визуального приложения с решением задачи 8.1.

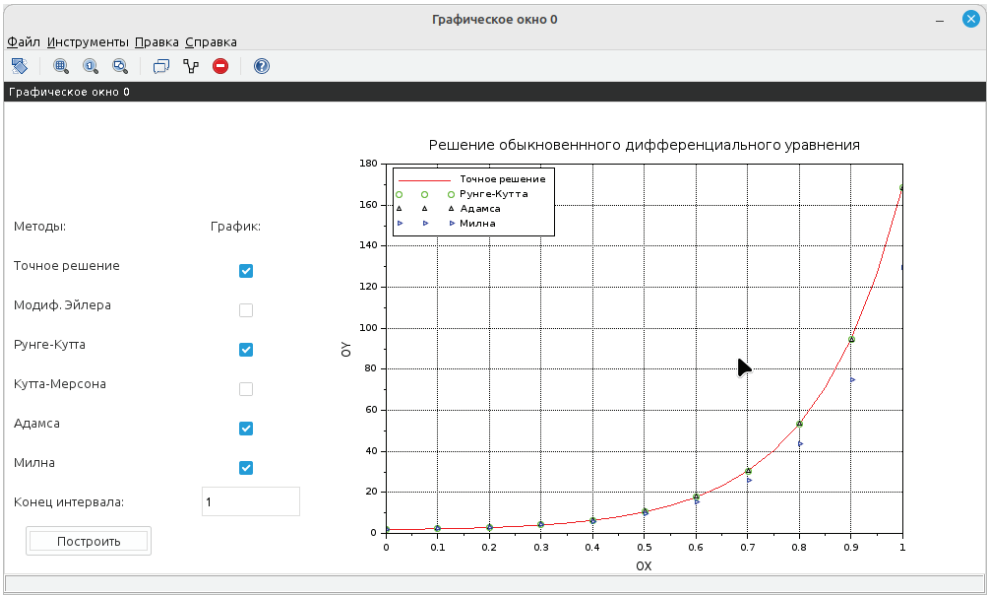

Рис. 8.4. Окно визуального приложения с решением задачи 8.1

8.3 Решение систем дифференциальных уравнений

Все рассмотренные методы решения дифференциальных уравнений применимы и для систем дифференциальных уравнений. Рассмотрим на примере метода Рунге–Кутта, все остальные методы можно обобщить для систем аналогично.

Пусть дана система дифференциальных уравнений в матричном виде:

$$\begin{cases} \dfrac{d\bar{x}}{dt} = \bar{f}(t,\bar{x}) \\ \bar{x}(t_0) = \bar{x}_0 \text{ (начальное условие)} \end{cases}, \tag{8.28}$$

где $\bar{x} = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{pmatrix}$, $\bar{f}(t,\bar{x}) = \begin{pmatrix} f_1(t,x_1,x_2,\dots,X_n) \\ f_2(t,x_1,x_2,\dots,X_n) \\ \dots\dots\dots\dots\dots \\ f_n(t,x_1,x_2,\dots,X_n) \end{pmatrix}$, $\bar{x}_0 = \begin{pmatrix} x_1^0 \\ x_2^0 \\ \dots \\ x_n^0 \end{pmatrix}$.

Задавшись некоторым шагом h и введя стандартные обозначения $t_i = t_0 + ih$, $x_i = x(t_i)$, $\Delta x_i = x_{i+1} - x_i$, $i = 1, 2, \dots, n$, получим формулы метода Рунге–Кутта для системы:

$$\begin{cases} \overline{x}_{i+1} = \overline{x}_i + \Delta\overline{x}_i, \quad i = 0, 1, \ldots, n-1 \\ \Delta\overline{x}_i = \frac{h}{6}(\overline{K}_1^i + 2\overline{K}_2^i + 2\overline{K}_3^i + \overline{K}_4^i) \\ \overline{K}_1^i = \overline{f}(t_i, \overline{x}_i) \\ \overline{K}_2^i = \overline{f}\left(t_i + \frac{h}{2}, \overline{x}_i + \frac{h}{2}\overline{K}_1^i\right) \\ \overline{K}_3^i = \overline{f}\left(t_i + \frac{h}{2}, \overline{x}_i + \frac{h}{2}\overline{K}_2^i\right) \\ \overline{K}_4^i = \overline{f}\left(t_i + h, \overline{x}_i + h\overline{K}_3^i\right) \end{cases}. \tag{8.29}$$

8.4 Возможности Scilab для решения дифференциальных уравнений и систем

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 (8.3); `y` – вектор решений задачи (8.3)–(8.4).

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 жёстких 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 жёсткой 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.

Для более подробного описания функции `ode` авторы рекомендуют читателю обратиться к справке по Scilab. Описание функции `ode` переведено на русский язык и проиллюстрировано примерами.

Рассмотрим использование функции на примере следующих задач.

Задача 8.2

Решить задачу Коши $\frac{dx}{dt} + x = \sin(xt), x(0) = 1.5$.

Перепишем уравнение следующим образом: $\frac{dx}{dt} = -x + \sin(xt), x(0) = 1.5$.

График, моделирующий процесс, описанный заданным уравнением, представлен на рис. 8.5.

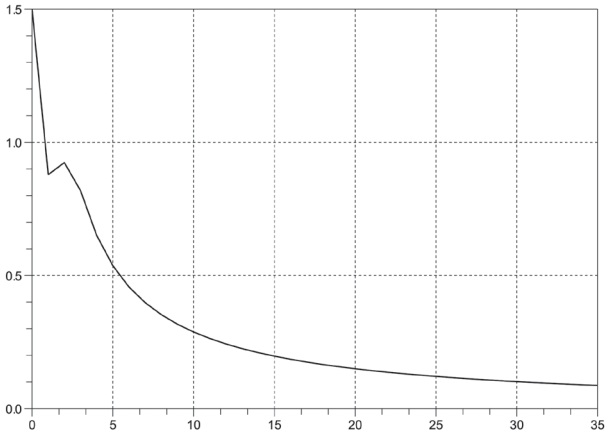

Рис. 8.5. Решение дифференциального уравнения из задачи 8.2

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 8.3. Решение задачи 8.2

```
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);
xgrid;
```

Рассмотрим использование функции ode на примере решения ещё нескольких задач.

Задача 8.3

Решить задачу Коши

$$\begin{cases} x' = \cos(xy) \\ y' = \sin(x + ty) \end{cases},$$
$$x(0) = 0, \quad y(0) = 0$$

на интервале [0; 10].

Далее приведена функция, описывающая заданную систему обыкновенных дифференциальных уравнений, и команды Scilab, необходимые для её численного и графического решений (рис. 8.6).

Листинг 8.4. Решение задачи 8.3

```
//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
figure();
plot(t,y);
xgrid;
```

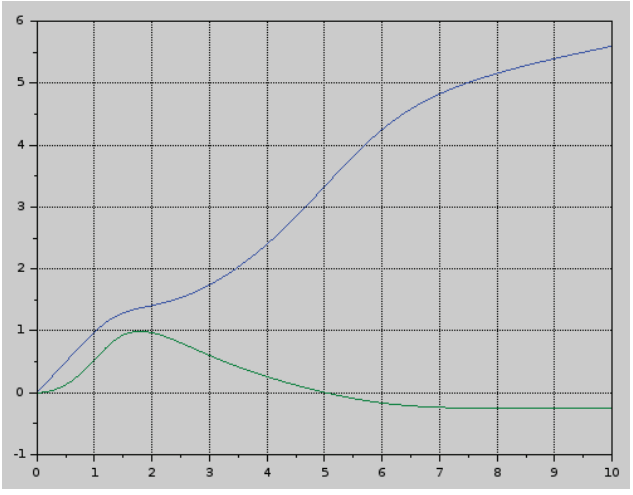

Рис. 8.6. Решение задачи 8.3

Задача 8.4

Найти решение задачи Коши для следующей жёсткой системы:

$$\frac{dX}{dt} = \begin{pmatrix} 119.46 & 185.38 & 126.88 & 121.03 \\ -10.395 & -10.136 & -3.636 & 8.577 \\ -53.302 & -85.932 & -63.182 & 54.211 \\ -115.58 & -181.75 & -112.8 & -199 \end{pmatrix} X; \quad X(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Решение задачи приведено в листинге 8.5 и на рис. 8.7.

Листинг 8.5. Решение задачи 8.4

```
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
//Начальные условия задачи Коши
x0=[1;1;1;1]
t0=0;
//Решение системы ОДУ
t=0:0.01:5;
y=ode("stiff",x0,t0,t,syst1,Jac);
//Построение графика решения
figure();
plot(t,y);
xgrid();
```

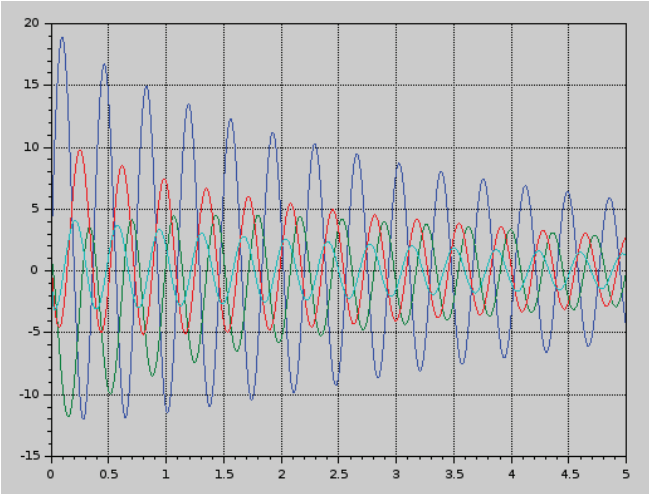

Рис. 8.7. Графическое решение задачи 8.4

Задача 8.5

Решить нелинейную жёсткую систему дифференциальных уравнений:

$$\begin{cases} \dfrac{dx_1}{dt} = -7x_1 + 7x_2 \\ \dfrac{dx_2}{dt} = 157x_1 - 1.15x_2x_3 \\ \dfrac{dx_3}{dt} = 0.96x_1x_2 - 8.36x_3 \end{cases}, \quad X(0) = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}.$$

Программа на языке Scilab, решающая задачу, приведена ниже.

Листинг 8.6. Решение задачи 8.5

```
//Функция, задающая систему ОДУ
function dx=syst2(t,x) 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
//начальное условие
x0=[-1;0;1];
t0=0;
t=0:0.01:2;
//Решение ОДУ
y=ode("stiff",x0,t0,t,syst2);
//Построение графика функции
figure();
plot(t,y);
xgrid();
```

На рис. 8.8 показано решение системы на интервале [0; 2].

Задача 8.6

Решить следующую краевую задачу на интервале [0.25; 2]:

$$\frac{d^2x}{dt^2} + 4\frac{dx}{dt} + 13 = e^{\sin(t)}, \quad x(0.25) = -1, \quad x'(0.25) = 1.$$

Преобразуем уравнение в систему, сделав замену $y = \dfrac{dx}{dt}$:

$$\frac{dy}{dt} = -4y - 13x + e^{\sin(t)}, \quad \frac{dx}{dt} = y, \quad y(0.25) = 1, \quad x(0.25) = -1.$$

Составим функцию вычисления системы и решим её.

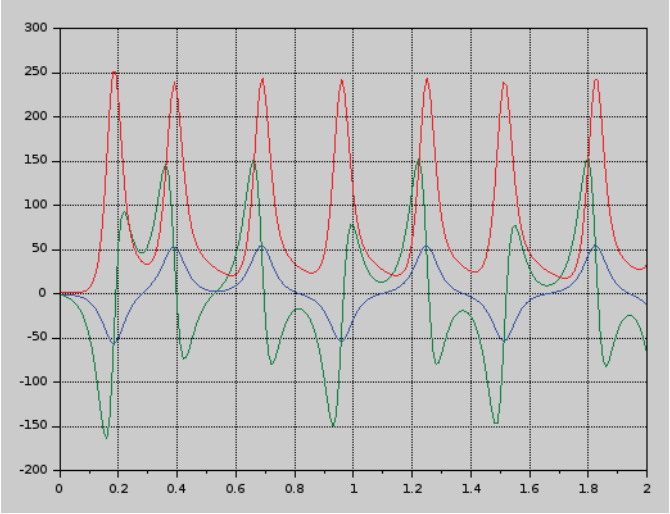

Рис. 8.8. Графическое решение задачи 8.5

График решения приведен на рис. 8.9.

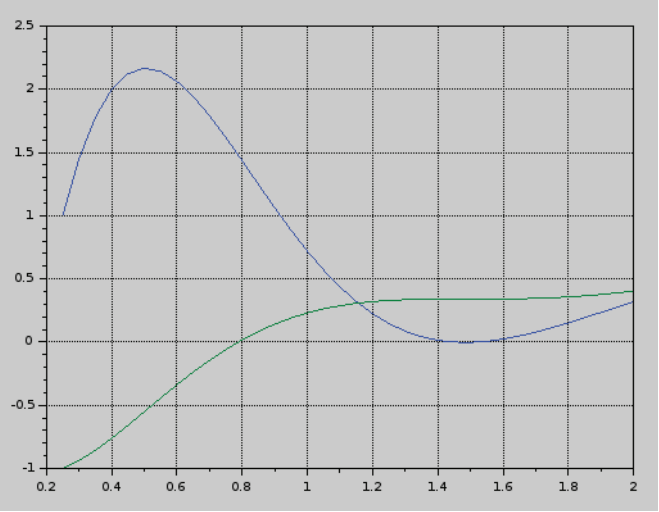

Рис. 8.9. Решение задачи 8.6

Листинг 8.7. Решение задачи 8.6

```
//Определение правой части системы ОДУ
function F=FF(t,x)
F=[-4*x(1)-13*x(2)+exp(t);x(1)];
endfunction
//Начальные условия задачи Коши.
X0=[1;-1];
t0=0.25;
t=0.25:0.05:2;
//Решение системы дифференциальных уравнений
y=ode("stiff",X0,t0,t,FF);
//Вывод графика решения
figure();
plot(t,y);
xgrid();
```

Глава 9

Обработка экспериментальных данных

Данная глава посвящена решению часто встречающихся на практике задач по обработке реальных количественных экспериментальных данных, полученных в результате всевозможных научных опытов, технических испытаний. Будут описаны метод наименьших квадратов и интерполяция, их реализация в Scilab.

9.1 Метод наименьших квадратов

Метод наименьших квадратов (МНК) позволяет по экспериментальным данным подобрать такую аналитическую функцию, которая проходит настолько близко к экспериментальным точкам, насколько это возможно.

9.1.1 Постановка задачи

В общем случае задачу можно сформулировать следующим образом. Пусть в результате эксперимента были получены некая экспериментальная зависимость $y(x)$, представленная в табл. 9.1. Необходимо построить аналитическую зависимость $f(x, a_1, a_2, \ldots, a_k)$, наиболее точно описывающую результаты эксперимента. Для построения параметров функции $f(x, a_1, a_2, \ldots, a_k)$ будем использовать *метод наименьших квадратов*. Идея метода наименьших квадратов заключается в том, что функцию $f(x, a_1, a_2, \ldots, a_k)$ необходимо подобрать таким образом, чтобы сумма квадратов отклонений измеренных значений y_i от расчётных $Y_i = f(x_i, a_1, a_2, \ldots, a_k)$ была бы наименьшей (см. рис. 9.1) [1, 3]:

$$S(a_1,a_2,\ldots,a_k) = \sum_{i=1}^{n}(y_i - Y_i)^2 = \sum_{i=1}^{n}(y_i - f(x_1,a_1,a_2,\ldots,a_k))^2 \to \min. \qquad (9.1)$$

Таблица 9.1. Данные эксперимента

x	x_1	x_2	x_3	…	x_{n-1}	x_n
y	y_1	y_2	y_3	…	y_{n-1}	y_n

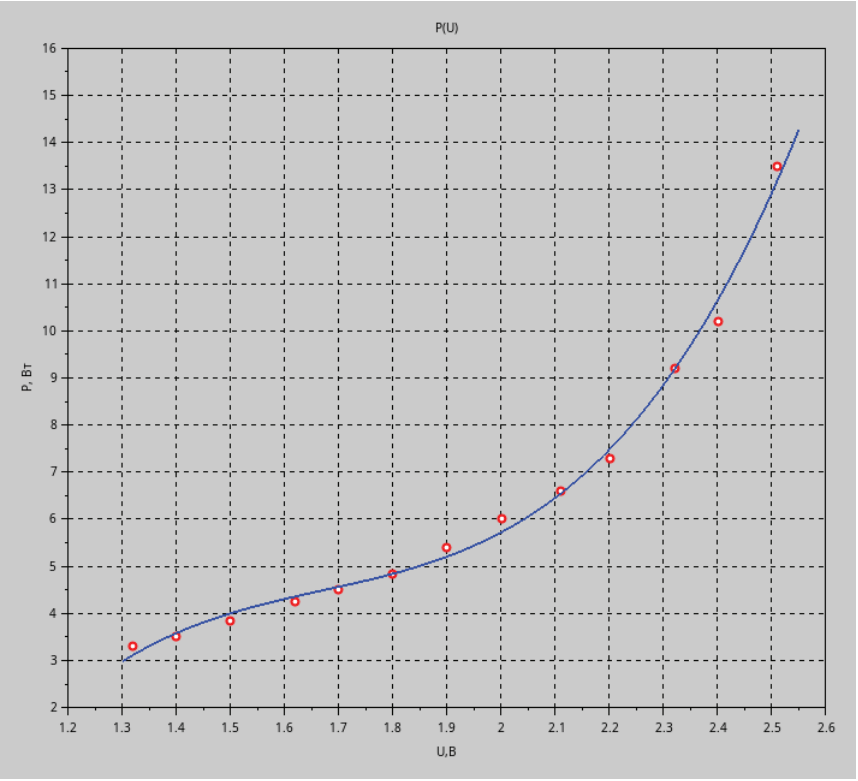

Рис. 9.1. Геометрическая интерпретация МНК

Построение подобной кривой называют *аппроксимацией экспериментальных значений*.

Задача состоит из двух этапов:

1) по результатам эксперимента выбрать функциональную зависимость $Y = f(x, a_1, a_2, \ldots, a_k)$;
2) подобрать коэффициенты $a_1, a_2, \ldots, a_k$ выбранной зависимости.

Математически задача подбора коэффициентов зависимости сводится к определению коэффициентов a_i из условия (9.1).

9.1.2 Подбор параметров экспериментальной зависимости методом наименьших квадратов

Вспомним некоторые сведения из высшей математики, необходимые для решения задачи подбора зависимости методом наименьших квадратов [3].

Достаточным условием минимума функции $S(a_1, a_2, \ldots, a_k)$ (9.1) является равенство нулю всех её частных производных. Поэтому задача поиска минимума функции (9.1) эквивалентна решению системы алгебраических уравнений [3]:

$$\begin{cases} \frac{\partial S}{\partial a_1} = 0 \\ \frac{\partial S}{\partial a_2} = 0 \\ \ldots\ldots\ldots \\ \frac{\partial S}{\partial a_k} = 0 \end{cases}. \tag{9.2}$$

Если параметры a_i входят в зависимость $Y = f(x, a_1, a_2, \ldots, a_k)$ линейно, то получим систему (9.3) из k линейных уравнений с k неизвестными [3].

$$\begin{cases} \sum_{i=1}^{n} 2(y_i - f(x_i, a_1, a_2, \ldots, a_k)) \frac{\partial f}{\partial a_1} = 0 \\ \sum_{i=1}^{n} 2(y_i - f(x_i, a_1, a_2, \ldots, a_k)) \frac{\partial f}{\partial a_2} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum_{i=1}^{n} 2(y_i - f(x_i, a_1, a_2, \ldots, a_k)) \frac{\partial f}{\partial a_k} = 0 \end{cases}. \tag{9.3}$$

Составим систему (9.3) для наиболее часто используемых функций.

9.1.2.1 Подбор коэффициентов линейной зависимости

Запишем функцию (9.1) для подбора параметров линейной функции $Y = a_1 + a_2 x$:

$$S(a_1, a_2) = \sum_{i=1}^{n} (y_i - a_1 - a_2 x_i)^2 \to \min. \tag{9.4}$$

Продифференцировав функцию S по a_1 и a_2, получим систему уравнений:

$$\begin{cases} 2\sum_{i=1}^{n} (y_i - a_1 - a_2 x_i)(-1) = 0 \\ 2\sum_{i=1}^{n} (y_i - a_1 - a_2 x_i)(-x_i) = 0 \end{cases} \Rightarrow \begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i x_i \end{cases}, \tag{9.5}$$

решив которую, определим коэффициенты функции $Y = a_1 + a_2 x$ [3]:

$$\begin{cases} a_1 = \frac{1}{n}\left(\sum_{i=1}^{n} y_i - a_2 \sum_{i=1}^{n} x_i\right) \\ a_2 = \dfrac{n\sum_{i=1}^{n} y_i x_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}. \end{cases} \tag{9.6}$$

9.1.2.2 Подбор коэффициентов полинома k-й степени

Для определения параметров зависимости $Y = a_1 + a_2 x + a_3 x^2$ составим функцию $S(a_1, a_2, a_3)$ (9.1):

$$S(a_1, a_2, a_3) = \sum_{i=1}^{n} (y_i - a_1 - a_2 x_i - a_3 x_i^2)^2 \to \min. \tag{9.7}$$

После дифференцирования S по a_1, a_2 и a_3 получим систему линейных алгебраических уравнений:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} y_i x_i \\ a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + a_3 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} y_i x_i^2 \end{cases}. \tag{9.8}$$

Решив систему (9.8), найдём значения параметров a_1, a_2 и a_3.

Аналогично определим параметры многочлена третьей степени: $Y = a_1 + a_2 x + a_3 x^2 + a_4 x^3$. Составим функцию $S(a_1, a_2, a_3, a_4)$:

$$S(a_1, a_2, a_3, a_4) = \sum_{i=1}^{n} (y_i - a_1 - a_2 x_i - a_3 x_i^2 - a_4 x_i^3)^2 \to \min. \tag{9.9}$$

После дифференцирования S по a_1, a_2, a_3 и a_4 система линейных алгебраических уравнений для вычисления параметров a_1, a_2, a_3, a_4 примет вид:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 + a_4 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 + a_4 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} y_i x_i \\ a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + a_3 \sum_{i=1}^{n} x_i^4 + a_4 \sum_{i=1}^{n} x_i^5 = \sum_{i=1}^{n} y_i x_i^2 \\ a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 + a_3 \sum_{i=1}^{n} x_i^5 + a_4 \sum_{i=1}^{n} x_i^6 = \sum_{i=1}^{n} y_i x_i^3 \end{cases}. \tag{9.10}$$

Решив систему (9.10), найдём коэффициенты a_1, a_2, a_3 и a_4.

В общем случае система уравнений для вычисления параметров a_i многочлена k-й степени $Y = \sum_{i=1}^{k=1} a_i x^{i-1}$ имеет вид [3]:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 + \cdots + a_{k+1} \sum_{i=1}^{n} x_i^k = \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 + \cdots + a_{k+1} \sum_{i=1}^{n} x_i^{k+1} = \sum_{i=1}^{n} y_i x_i \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_1 \sum_{i=1}^{n} x_i^{k+1} + a_2 \sum_{i=1}^{n} x_i^{k+2} + a_3 \sum_{i=1}^{n} x_i^{k+3} + \cdots + a_{k+1} \sum_{i=1}^{n} x_i^{2k} = \sum_{i=1}^{n} y_i x_i^k \end{cases}. \tag{9.11}$$

В матричном виде систему (9.11) можно записать так:

$$Ca = g. \tag{9.12}$$

Элементы матрицы C и вектора g рассчитываются по формулам

$$C_{i,j} = \sum_{k=1}^{n} x_k^{i+j-2}, \quad i = 1, \ldots, k+1, \quad j = 1, \ldots, k+1, \tag{9.13}$$

$$g_i = \sum_{k=1}^{n} y_k x_k^{i-1}, \quad i = 1, \ldots, k+1. \tag{9.14}$$

Решив систему (9.12), определим параметры зависимости $Y = a_1 + a_2x + a_3x^2 + \cdots + a_{k+1}x^k$.

9.1.2.3 Подбор коэффициентов функции $Y = ax^be^{cx}$

Параметры b и c входят в зависимость $Y = ax^be^{cx}$ нелинейным образом. Чтобы избавиться от нелинейности, предварительно прологарифмируем[1] выражение $Y = ax^be^{cx}$: $\ln Y = \ln a + b\ln x + cx$. Сделаем замену $Y1 = \ln Y$, $A = \ln a$, после этого функция примет вид: $Y1 = A + b\ln x + cx$.

Составим функцию $S(A, b, c)$ по формуле (9.1):

$$S(A,b,c) = \sum_{i=1}^{n}(Y1_i - A - b\ln x - cx_i)^2 \to \min. \tag{9.15}$$

После дифференцирования получим систему трёх линейных алгебраических уравнений для определения коэффициентов A, b, c:

$$\begin{cases} nA + b\sum_{i=1}^{n}\ln x_i + c\sum_{i=1}^{n}x_i = \sum_{i=1}^{n}Y1_i \\ A\sum_{i=1}^{n}\ln x_i + b\sum_{i=1}^{n}(\ln x_i)^2 + c\sum_{i=1}^{n}x_i\ln x_i = \sum_{i=1}^{n}Y1_i\ln x_i. \\ A\sum_{i=1}^{n}x_i + b\sum_{i=1}^{n}x_i\ln x_i + c\sum_{i=1}^{n}x_i^2 = \sum_{i=1}^{n}Y1_ix_i \end{cases} \tag{9.16}$$

После решения системы (9.16) необходимо вычислить значение коэффициента a по формуле $a = e^A$.

9.1.2.4 Функции, приводимые к линейной

Для вычисления параметров функции $Y = ax$ необходимо предварительно её прологарифмировать: $\ln Y = \ln ax^b = \ln a + b\ln x$. После чего замена $Z = \ln Y$, $X = \ln x$, $A = \ln a$ приводит заданную функцию к линейному виду $Z = bX + A$, где коэффициенты A и b вычисляются по формулам (9.6), и, соответственно, $a = e^A$.

Аналогично можно подобрать параметры функции вида $Y = ae^{bx}$. Прологарифмируем заданную функцию $\ln y = \ln a + bx\ln e$, $\ln y = \ln a + bx$. Проведём замену $Y = \ln y$, $A = \ln a$ и получим линейную зависимость $Y = bx + A$. По формулам (9.6) найдём A и b, а затем вычислим $a = e^A$.

Рассмотрим ещё ряд зависимостей, которые сводятся к линейной.

Для подбора параметров функции $Y = \frac{1}{ax+b}$ сделаем замену $Z = \frac{1}{Y}$. В результате получим линейную зависимость $Z = ax + b$. Функция $Y = \frac{x}{ax+b}$ заменами $Z = \frac{1}{Y}$, $X = \frac{1}{x}$ сводится к линейной $Z = a + bX$. Для определения коэффициентов функциональной зависимости $Y = \frac{1}{ae^{-x}+b}$ необходимо сделать следующие замены: $Z = \frac{1}{Y}$, $X = e^{-x}$. В результате также получим линейную функцию $Z = aX + b$.

---

[1] Можно и не проводить предварительное логарифмирование выражения $Y = ax^be^{cx}$, однако в этом случае полученная система уравнений будет нелинейной, а такую систему решать сложнее.

Аналогичными преобразованиями (логарифмированием, заменами и т. п.) многие зависимости $f(x, a_1, a_2, \ldots, a_k)$ могут быть сведены к такому виду, что параметры $a_1, a_2, \ldots, a_k$ входят в них линейно. Это означает, что при формировании функции (9.1) и последующем её дифференцировании будет получена система линейных алгебраических уравнений (9.2), которую решить значительно легче, чем нелинейную.

9.1.3 Точность подбора параметров

После нахождения параметров зависимости $f(x, a_1, a_2, \ldots, a_k)$ возникает вопрос, насколько адекватно описывает подобранная зависимость экспериментальные данные. Чем ближе величина

$$S = \sum_{i=1}^{n} (y_i - f(x_i, a_1, a_2, \ldots, a_k))^2, \tag{9.17}$$

называемая *суммарной квадратичной ошибкой*, к нулю, тем точнее подобранная кривая описывает экспериментальные данные. Однако по суммарной квадратичной ошибке сложно судить о том, насколько точно подобранная кривая описывает физический процесс. Давайте введём ещё некоторые величины *ошибок*.

Средняя ошибка в точке (S_t):

$$S_t = \frac{\sqrt{\sum_{i=1}^{n} (y_i - f(x_i, a_1, a_2, \ldots, a_k))^2}}{n}. \tag{9.18}$$

По этой величине можно судить, насколько в среднем отклоняется расчётное значение (значение на подобранной кривой) от реального (результата эксперимента). Но этого значения тоже недостаточно для оценки точности подбора кривой. Допустим, экспериментальные значения находились в пределах 1000–2000, в этом случае $S_t = 20$ – очень хороший результат (1–2 %). Если же экспериментальные значения были в пределах 50–100, то говорить о хорошем подборе зависимости при $S_t = 20$ не приходится. Среднюю ошибку в точке необходимо соотносить с ординатами экспериментальных значений. Поэтому имеет смысл определить *относительную ошибку*:

$$S_{to} = \frac{S_t}{M_y} = \frac{\frac{\sqrt{\sum_{i=1}^{n} (y_i - f(x_i, a_1, a_2, \ldots, a_k))^2}}{n}}{\frac{\sum_{i=1}^{n} y_i}{n}}. \tag{9.19}$$

Относительную ошибку в точке можно выражать в процентах, если умножить на 100. С инженерной точки зрения достаточно хорошей является относительная ошибка в точке порядка 5–10 %. Если ошибка менее 2 %, это, конечно, хороший математический результат, но стоит задать вопросы, с какой точностью были получены экспериментальные данные и насколько точными были приборы.

9.1.4 Уравнение регрессии и коэффициент корреляции

Линия, описываемая уравнением вида $y = a_1 + a_2x$, называется *линией регрессии y* на *x*, параметры a_1 и a_2 называются *коэффициентами регрессии* и определяются формулами (9.6).

Чем меньше величина $S = \sum_{i=1}^{n}(y_i - a_1 - a_2x_i)^2$, тем более обоснованно предположение, что экспериментальные данные описываются линейной функцией.

Существует показатель, характеризующий тесноту линейной связи между *x* и *y*, который называется *коэффициентом корреляции* и рассчитывается по формуле

$$r = \frac{\sum_{i=1}^{n}(x_i - M_x)(y_i - M_y)}{\sqrt{\sum_{i=1}^{n}(x_i - M_x)^2 \sum_{i=1}^{n}(y_i - M_y)^2}}, \quad M_x = \frac{\sum_{i=1}^{n} x_i}{n}, \quad M_y = \frac{\sum_{i=1}^{n} y_i}{n}. \quad (9.20)$$

Значение коэффициента корреляции удовлетворяет соотношению $-1 \le r \le 1$. Чем меньше отличается абсолютная величина *r* от единицы, тем ближе к линии регрессии располагаются экспериментальные точки. Если $|r| = 1$, то все экспериментальные точки находятся на линии регрессии. Если коэффициент корреляции близок к нулю, то это означает, что между *x* и *y* не существует линейной связи, но между ними может существовать зависимость, отличная от линейной.

Для того чтобы проверить, значимо ли отличается от нуля коэффициент корреляции, можно использовать *критерий Стьюдента*. Вычисленное значение критерия определяется по формуле

$$t = r\sqrt{\frac{n-2}{1-r^2}}. \quad (9.21)$$

Рассчитанное по формуле (9.21) значение *t* сравнивается со значением, взятым из *таблицы распределения Стьюдента* (см. табл. 9.2), в соответствии с уровнем значимости *p* (стандартное значение $p = 0.95$) и числом степеней свободы $k = n - 2$. Если полученная по формуле (9.21) величина *t* больше табличного значения, то коэффициент корреляции значимо отличен от нуля.

9.1.5 Нелинейная корреляция

Коэффициент корреляции *r* применяется только в тех случаях, когда между данными существует прямолинейная связь. Если же связь нелинейная, то для выявления тесноты связи между переменными *y* и *x* в случае нелинейной зависимости используют *индекс корреляции*. Он показывает тесноту связи между фактором *x* и зависимой переменной *y* и рассчитывается по формуле [3]

$$R = \sqrt{1 - \frac{\sum_{i=1}^{n}(y_i - Y_i)^2}{\sum_{i=1}^{n}(y_i - M_y)^2}}, \quad (9.22)$$

где y – экспериментальные значения; Y – теоретические значения (рассчитанные по подобранной методом наименьших квадратов формуле); M_y – среднее значение y.

Таблица 9.2. Таблица распределения Стьюдента

k \ p	0.99	0.98	0.95	0.90	0.80	0.70	0.60
1	63.657	31.821	12.706	6.314	3.078	1.963	1.376
2	9.925	6.965	4.303	2.920	1.886	1.386	1.061
3	5.841	4.541	3.182	2.353	1.638	1.250	0.978
4	4.604	3.747	2.776	2.132	1.533	1.190	0.941
5	4.032	3.365	2.571	2.05	1.476	1.156	0.920
6	3.707	3.141	2.447	1.943	1.440	1.134	0.906
7	3.499	2.998	2.365	1.895	1.415	1.119	0.896
8	3.355	2.896	2.306	1.860	1.387	1.108	0.889
9	3.250	2.821	2.261	1.833	1.383	1.100	0.883
10	3.169	2.764	2.228	1.812	1.372	1.093	0.879
11	3.106	2.718	2.201	1.796	1.363	1.088	0.876
12	3.055	2.681	2.179	1.782	1.356	1.083	0.873
13	3.012	2.650	2.160	1.771	1.350	1.079	0.870
14	2.977	2.624	2.145	1.761	1.345	1.076	0.868
15	2.947	2.602	2.131	1.753	1.341	1.074	0.866
16	2.921	2.583	2.120	1.746	1.337	1.071	0.865
17	2.898	2.567	2.110	1.740	1.333	1.069	0.863
18	2.878	2.552	2.101	1.734	1.330	1.067	0.862
19	2.861	2.539	2.093	1.729	1.328	1.066	0.861
20	2.845	2.528	2.086	1.725	1.325	1.064	0.860
21	2.831	2.518	2.080	1.721	1.323	1.063	0.859
22	2.819	2.508	2.074	1.717	1.321	1.061	0.858
23	2.807	2.500	2.069	1.714	1.319	1.060	0.858
24	2.797	2.492	2.064	1.711	1.318	1.059	0.857
25	2.779	2.485	2.060	1.708	1.316	1.058	0.856
26	2.771	2.479	2.056	1.706	1.315	1.058	0.856
27	2.763	2.473	2.052	1.703	1.314	1.057	0.855
28	2.756	2.467	2.048	1.701	1.313	1.056	0.855
29	2.750	2.462	2.045	1.699	1.311	1.055	0.854
30	2.704	2.457	2.042	1.697	1.310	1.055	0.854
40	2.660	2.423	2.021	1.684	1.303	1.050	0.851
60	2.612	2.390	2.000	1.671	1.296	1.046	0.848
120	2.617	2.358	1.980	1.658	1.289	1.041	0.845
∞	2.576	2.326	1.960	1.645	1.282	1.036	0.842

Индекс корреляции лежит в пределах от 0 до 1. При наличии функциональной зависимости индекс корреляции близок к 1. При отсутствии связи R практически равен нулю. Если коэффициент корреляции r является мерой тесноты связи только для линейной формы связи, то индекс корреляции R – как для линейной, так и для нелинейной. При прямолинейной связи коэффициент корреляции по своей абсолютной величине равен индексу корреляции: $|r| = R$.

9.2 Решение задач аппроксимации в Scilab

Решать задачу подбора параметров кривой (задачу аппроксимации) можно двумя способами:

- выводить систему уравнений, как описано в предыдущем параграфе, и решать её;
- использовать функцию `datafit`.

Синтаксис функции `datafit`:

```
[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 расчётных.

Рассмотрим решение задачи 9.1 двумя способами.

Задача 9.1

В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P, Вт) от входного напряжения (U, В) для асинхронного двигателя (табл. 9.3).

Таблица 9.3. 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 $P = a_1 + a_2U + a_3U^2 + a_4U^3$.

Способ 1. Для вычисления коэффициентов полинома третьей степени необходимо найти минимум функции S (9.9), что равносильно решению системы (9.10).

Код программы приведён ниже.

Листинг 9.1. Решение задачи 9.1

```
//Результаты эксперимента
U=[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];
P=[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];
```

```
//Количество экспериментальных данных
  N=length(U);
//Формируем систему (3.10), (3.12) для нахождения параметров
//подбираемой зависимости

  for i=1:4
    for j=1:4
      sum=0
      for k=1:N
        sum=sum+U(k)^(i+j-2)
      end
      A(i,j)=sum
    end
  end
  for i=1:4
    sum=0;
    for k=1:N
      sum=sum+P(k)*U(k)^(i-1)
    end
    b(i)=sum
  end
//Решаем систему (3.10), (3.12)
//Находим параметры подбираемой зависимости
  a=linsolve(A,-b);
  mprintf("Подобранные коэффициенты\n");
  for i=1:4
    mprintf("%8.4f\t",a(i))
  end
//Вычисляем суммарную квадратичную ошибку
  oshibka=0
  for i=1:length(U)
    oshibka=oshibka+(P(i)-a(1)-a(2)*U(i)-a(3)*U(i)^2-a(4)*U(i)^3)^2
  end
  mprintf("\nСуммарная квадратичная ошибка = %8.4f\n",oshibka);
  s=0;
  for i=1:length(U);
    s=s+P(i)
  end
//Вычисляем среднюю ошибку в точке
  st=sqrt(oshibka)/length(U);
  mprintf("Средняя ошибка в точке = %8.4f\n",st);
  sto=st/(s/length(U));
//Вычисляем относительную ошибку
  mprintf("Относительная ошибка = %8.4f\n",sto);
//Строим график
  okno1=figure();
  U2=1.3:0.01:2.55;
  for i=1:length(U2)
  P2(i)=a(1)+a(2)*U2(i)+a(3)*U2(i).^2+a(4).*U2(i).^3
  end
  plot(U,P,"ro",U2,P2,"b-");
  xgrid;
  xtitle("P(U)","U,B","P, Вт");
```

Результаты вычислений приведены ниже. Графическое решение задачи представлено в листинге 9.2 и на рис. 9.2.

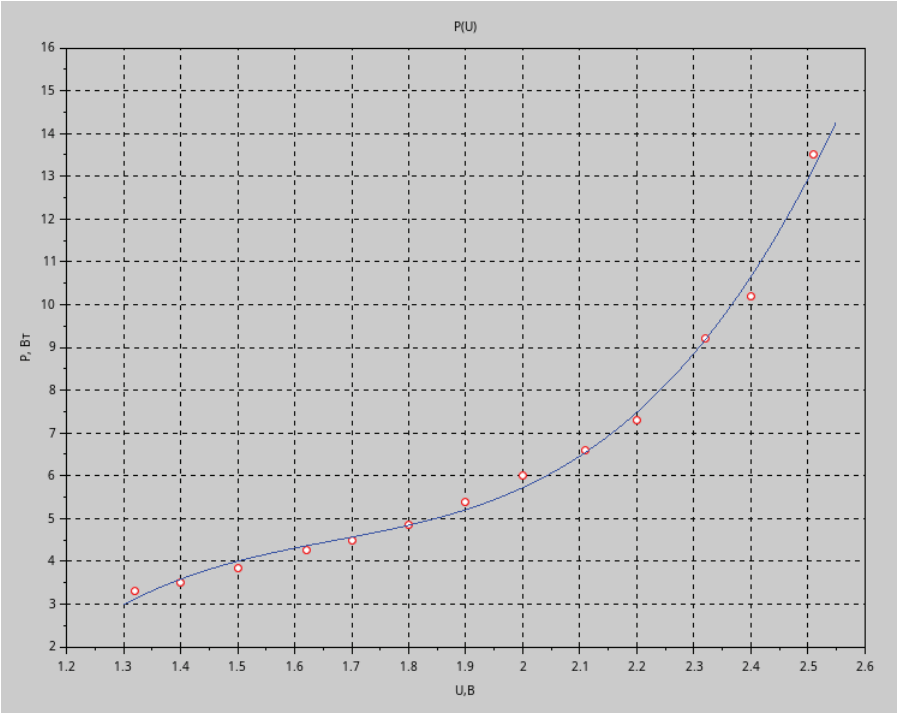

Рис. 9.2. Решение задачи 9.1

Листинг 9.2. Графическое решение задачи 9.1

```
Подобранные коэффициенты
-51.5767  95.5947 -55.6953  11.1115
Суммарная квадратичная ошибка =   0.5288
Средняя ошибка  в точке =   0.0559
Относительная ошибка  в точке =   0.0088
```

Напишем визуальное приложение для решения этой задачи. Его код с комментариями представлен в листинге 9.3.

Листинг 9.3. Решение задачи 9.1

```
clear
//Создание окна
f=figure('figure_position',[300 300],'figure_size',[1000 800],...
'resize','off','background',8);
//Создание фрейма и осей в нём
fr=uicontrol(f,'Style','frame','position',[500,-10,500,500]);
ax = newaxes();
ax.parent=fr;
```

```
//Создание подписей
L0=uicontrol(f,'Style','text','string','   Коэффициенты: ',...
'position',[0,380,150,30],'BackgroundColor',[1,1,1]);
L1=uicontrol(f,'Style','text','string',...
'   Суммарная квадратичная ошибка:',...
'position',[0,340,380,30],'BackgroundColor',[1,1,1]);
L2=uicontrol(f,'Style','text','string','   Средняя ошибка в точке:',...
'position',[0,300,320,30],'BackgroundColor',[1,1,1]);
L3=uicontrol(f,'Style','text','string','   Относительная ошибка:',...
'position',[0,260,320,30],'BackgroundColor',[1,1,1]);
D0=uicontrol(f,'Style','text','position',[170,380,340,30],...
'BackgroundColor',[1,1,1]);
//Результаты эксперимента
LU=uicontrol(f,'Style','text','string','   U,B ',...
'position',[0,550,50,30],'BackgroundColor',[1,1,1]);
EU=uicontrol(f,'Style','edit','string',...
'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',...
'position',[70,550,800,30]);
LP=uicontrol(f,'Style','text','string','   P, Вт',...
'position',[0,510,60,30],'BackgroundColor',[1,1,1]);
EP=uicontrol(f,'Style','edit','string',...
'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',...
'position',[70,510,800,30]);

function visual()
  fr=uicontrol(f,'Style','frame','position',[500,-10,500,500]);
  ax = newaxes();
  ax.parent=fr;
  //Считывание введённых данных и преобразование их к числам
  U=strsplit(EU.string,[' ',',']);
  U=strtod(U);
  P=strsplit(EP.string,[' ',',']);
  P=strtod(P);
  //Если длины массивов не совпадают, вывести сообщение
  if length(P)<>length(U) then
    set(f,'info_message','Размеры массивов не совпадают');
  else
    set(f,'info_message',''); end
  //Количество экспериментальных данных
  N=length(U);
  //Формируем систему (3.10), (3.12) для нахождения параметров
  //подбираемой зависимости
  for i=1:4
    for j=1:4
      sum=0
      for k=1:N
        sum=sum+U(k)^(i+j-2)
```

```
      end
      A(i,j)=sum
    end
  end
  for i=1:4
    sum=0;
    for k=1:N
      sum=sum+P(k)*U(k)^(i-1)
    end
    b(i)=sum
  end
  //Решаем систему (3.10), (3.12)
  //Находим параметры подбираемой зависимости
  a=linsolve(A,-b);
  //Вычисляем суммарную квадратичную ошибку
  oshibka=0
  for i=1:length(U)
    oshibka=oshibka+(P(i)-a(1)-a(2)*U(i)-a(3)*U(i)^2-a(4)*U(i)^3)^2
  end
  s=0;
  for i=1:length(U);
    s=s+P(i)
  end
  //Вычисляем среднюю ошибку в точке
  st=sqrt(oshibka)/length(U);
  //Вычисляем относительную ошибку
  sto=st/(s/length(U));
  U2=U(1):0.01:U(N);
  for i=1:length(U2)
    P2(i)=a(1)+a(2)*U2(i)+a(3)*U2(i).^2+a(4).*U2(i).^3;
  end
  //Вывод результатов расчётов
  set(L1,'string',sprintf('   Суммарная квадратичная ошибка: %8.4f',...oshibka));
  set(L2,'string',sprintf('   Средняя ошибка в точке: %8.4f',st))
  set(L3,'string',sprintf('   Относительная ошибка: %8.4f',sto))
  set(D0,'string',sprintf("%8.4f ",a));
  //Построение графиков
  plot(U,P,"ro",U2,P2,"b-");
  xgrid;
  xtitle("P(U)","U,B","P, Вт");
endfunction
//Создание кнопки для расчётов
BtnPlot=uicontrol(f,'Style','pushbutton','callback','visual','String',...
'Построить','position',[20,20,130,30]);
```

Окно визуального приложения для решения задачи 9.1 представлено на рис. 9.3.

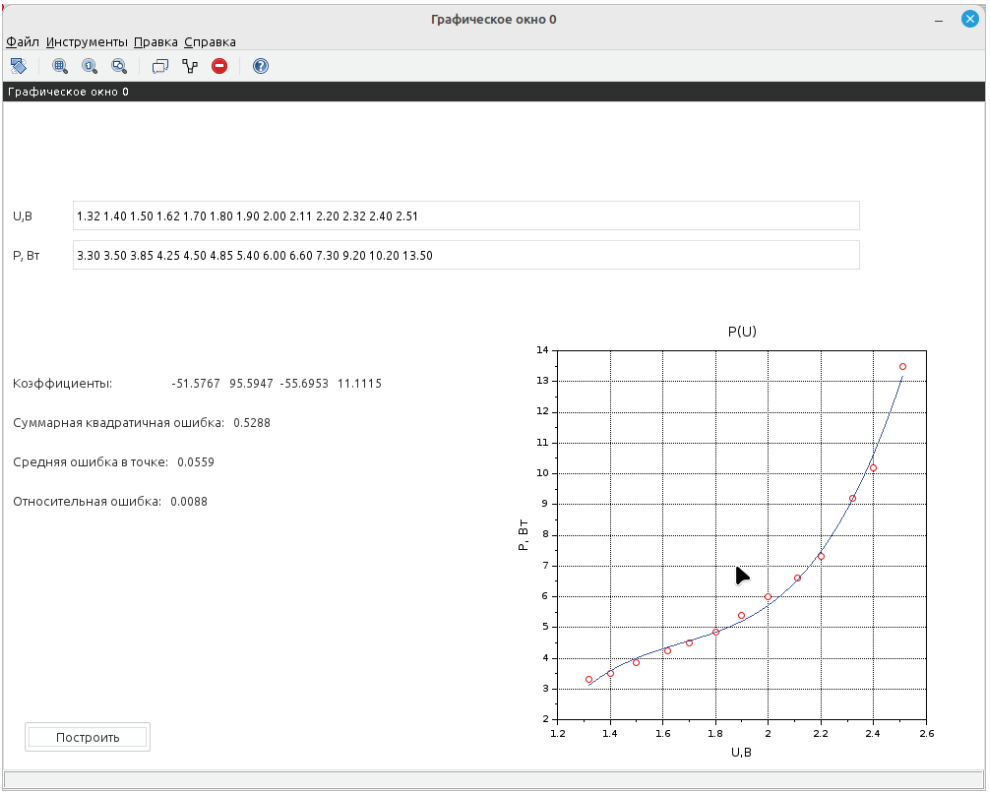

Рис. 9.3. Визуальное приложение для решения задачи 9.1

Способ 2. Решение задачи 9.1 с помощью встроенной функции Scilab `datafit`.

Листинг 9.4. Решение задачи 9.1 с помощью функции `datafit`

```
//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
```

В результате работы функции `datafit` была подобрана такая же аналитическая зависимость $P = -51.577 + 95.595U - 55.695U^2 + 11.111U^3$, суммарная квадратичная ошибка такая же, как и в первом решении.

Возникает закономерный вопрос: какой же метод подбора кривой использовать? Попробуем ответить на этот вопрос при решении следующей задачи.

Задача 9.2

Результаты некоторого эксперимента заданы в таблице. Подобрать зависимость вида $Y = a \cdot x^b$.

Рассмотрим те же два способа решения, что и в задаче 9.1.

Первый способ. Составим систему уравнений. Для вычисления параметров функции $Y = ax^b$ необходимо предварительно её прологарифмировать: $\ln Y = \ln ax^b = \ln a + b \ln x$. После чего замена $Z = \ln Y, X = \ln x, A = \ln a$ приводит заданную функцию к линейному виду $Z = bX + A$, где коэффициенты A и b являются решением системы двух линейных алгебраических уравнений (9.5), (9.8). Не забываем, что $a = e^A$.

Второй способ. Использование функции `datafit`.

Программа с комментариями, которая решает задачу двумя способами, представлена ниже.

Листинг 9.5. Решение задачи 9.2

```
clear
//Функция, которая возвращает массив ошибок.
//Необходима для функции datafit
function [zr]=F(c,z)
zr=z(2)-c(1).*z(1).^c(2);
endfunction
//Результаты эксперимента.
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];
//Решение задачи с помощью функции datafit
z=[x;y];
c=[0;0];
[a,S]=datafit(F,z,c);
```

```
mprintf("Коэффициенты кривой, найденные с помощью datafit\n");
for i=1:length(a)
mprintf("%8.4f\t",a(i));
end
//Решение задачи с помощью системы уравнений
//Формирование системы уравнений для решения задачи
C(1,1)=length(x);
C(2,1)=sum(log(x));
C(1,2)=sum(log(x));
C(2,2)=sum(log(x).*log(x));
b(1)=sum(log(y));
b(2)=sum(log(y).*log(x));
//Решение системы и нахождение коэффициентов.
q=inv(C)*b;
mprintf("\nКоэффициенты кривой, найденные путём решения СЛАУ\n");
mprintf("a=%8.4f,\tb=%8.4f\n",exp(q(1)),q(2));
//Суммарная ошибка при решении задачи путём решения СЛАУ S2=0;
for i=1:length(x)
S2=S2+(y(i)-exp(q(1)).*x(i).^q(2));
end
//Вычисление относительных ошибок
s=sqrt(S)/length(x)/(sum(y)/length(y));
s2=sqrt(S2)/length(x)/(sum(y)/length(y));
mprintf("Ошибки при подборе кривой с помощью datafit\n");
mprintf("\nСуммарная квадратичная ошибка %8.4f\n",S)
mprintf("Относительная ошибка %8.4f\n",s)
mprintf("Ошибки при подборе кривой при решении СЛАУ\n");
mprintf("\nСуммарная квадратичная ошибка %8.4f\n",S2)
mprintf("Относительная ошибка %8.4f\n",s2)
//Графическая интерпретация решения
t=10:0.01:19;
for i=1:length(t)
Yt(i)=a(1)*t(i)^a(2);
Ytt(i)=exp(q(1)).*t(i).^q(2);
end
figure();
plot(x,y,"bo",t,Yt,"k-",t,Ytt,"r-");
xgrid;
xtitle("y(x)","OX","OY");
legend("Эксперимент","datafit","СЛАУ")
```

Ниже представлены результаты работы программы, а на рис. 9.4 – графики подобранных кривых.

Листинг 9.6. Результат решения задачи 9.2

```
Коэффициенты кривой, найденные с помощью datafit
1.1232   1.5814
Коэффициенты кривой, найденные путём решения СЛАУ
a=  0.2477,b=  2.1465
Ошибки при подборе кривой с помощью datafit
Суммарная квадратичная ошибка 3041.4276
```

```
Относительная ошибка   0.0400
Ошибки при подборе кривой при решении СЛАУ
Суммарная квадратичная ошибка   9.8014
Относительная ошибка   0.0023
```

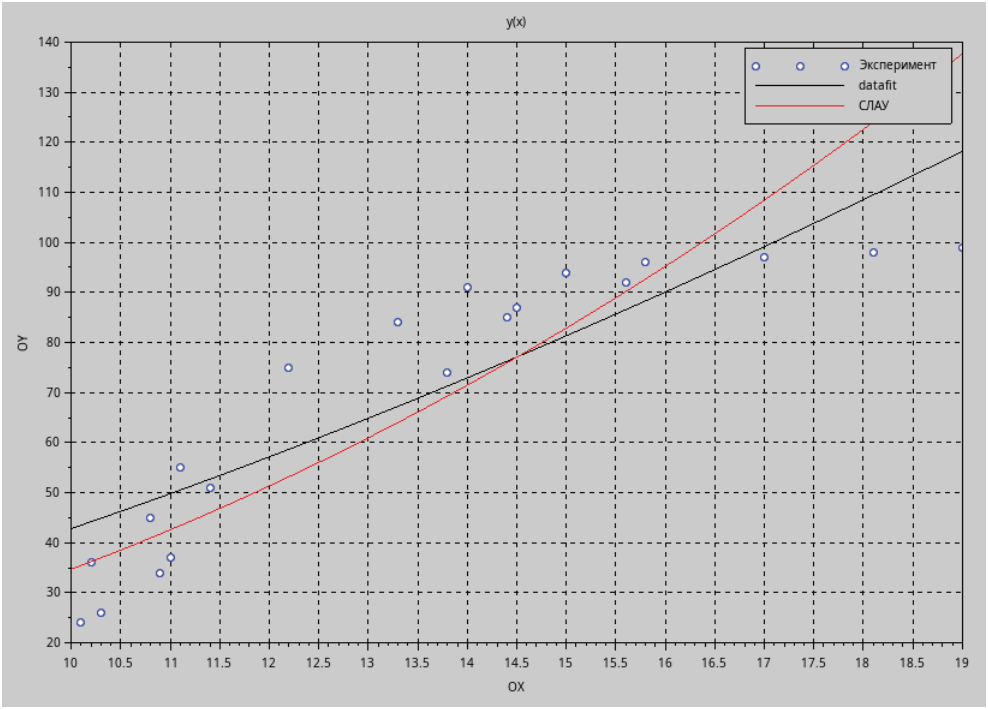

Рис. 9.4. Графическое решение задачи 9.2

Анализируя полученные результаты, можно сделать следующий вывод. Результаты, полученные двумя способами, различны. Ближе к экспериментальным точкам проходит кривая, параметры которой были найдены как решение системы (9.5). Попробуем понять, почему так происходит. В предыдущей задаче результаты подбора параметров кривой двумя способами были идентичны.

Функция `datafit` находит решение задачи подбора параметров как задачу поиска экстремума функции (9.1). Если параметры a_i входят в функцию f линейно, то это решение задачи поиска экстремума функции S сводится к системе линейных уравнений (9.2)–(9.3). Методы решения систем линейных алгебраических уравнений хорошо разработаны. Если же параметры a_i входят в функцию (9.1) нелинейно, то задача экстремума сводится к системе нелинейных уравнений, алгоритмы которых нетривиальны и очень чувствительны к начальному приближению.

Поэтому, на взгляд авторов, если параметры a_i входят в функцию нелинейно, то в первую очередь надо постараться избавиться от нелинейности, если это возможно. Об этом речь шла выше. Потом особой разницы нет, ис-

пользовать функцию `datafit` или самостоятельно решать систему линейных алгебраических уравнений.

Если от нелинейности параметров a_i избавиться не удалось, то используем функцию `datafit`, но необходимо учитывать, что решение чувствительно к начальному приближению, подбор которого становится непростой задачей.

В завершение рассмотрим одну хорошо известную задачу обработки эксперимента.

Задача 9.3

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. 9.4. Требуется определить растворимость азотнокислого натрия при температуре 32 °C в случае линейной зависимости и найти коэффициент и индекс корреляции.

Таблица 9.4. 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

Программа решения задачи 9.3 с комментариями представлена ниже.

Листинг 9.7. Решение задачи 9.3

```
//Экспериментальные данные
  function [zr]=F(c,z)
    zr=z(2)-c(1)*z(1)- c(2);
  endfunction
  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];
//Расчёт коэффициентов регрессии
  z=[x;y];
  c=[0;0];
  [a,S]=datafit(F,z,c);
//Растворимость азотнокислого натрия при температуре 32 градуса
  T=32;
//Коэффициент корреляции (11.20)
  r=sum((x-mean(x)).*(y-mean(y)))/sqrt(sum((x-mean(x))^2)
  *sum((y-mean(y))^2))
//Индекс корреляции (11.22)
  R=sqrt(1-sum((y-(a(1)*x+a(2)))^2)/sum((y-mean(y))^2))
  t=0:70;
  Yt=a(1)*t+a(2); mprintf("a1=%8.4f\ta2=%8.4f\n",a(1),a(2));
  mprintf("f(%6.2f)=%6.2f\n",T,a(1)*T+a(2));
  mprintf("r=%8.4f\tR=%8.4f\n",r,R);
  figure();
//Построение графика
  plot(x,y,"ro",T,a(1)*T+a(2),"bo",t,Yt,"m-");
```

```
xgrid;
xtitle("Решение задачи Менделеева","Растворимость","Температура");
legend("эксперимент","t=32","линия регрессии")
```

Результаты работы программы представлены ниже.

Листинг 9.8. Решение задачи 9.3

```
a1=  0.8706 a2= 67.5078
f( 32.00)= 95.37
r=  0.9990 R=  0.9990
```

Графическое решение задачи можно увидеть на рис. 9.5.

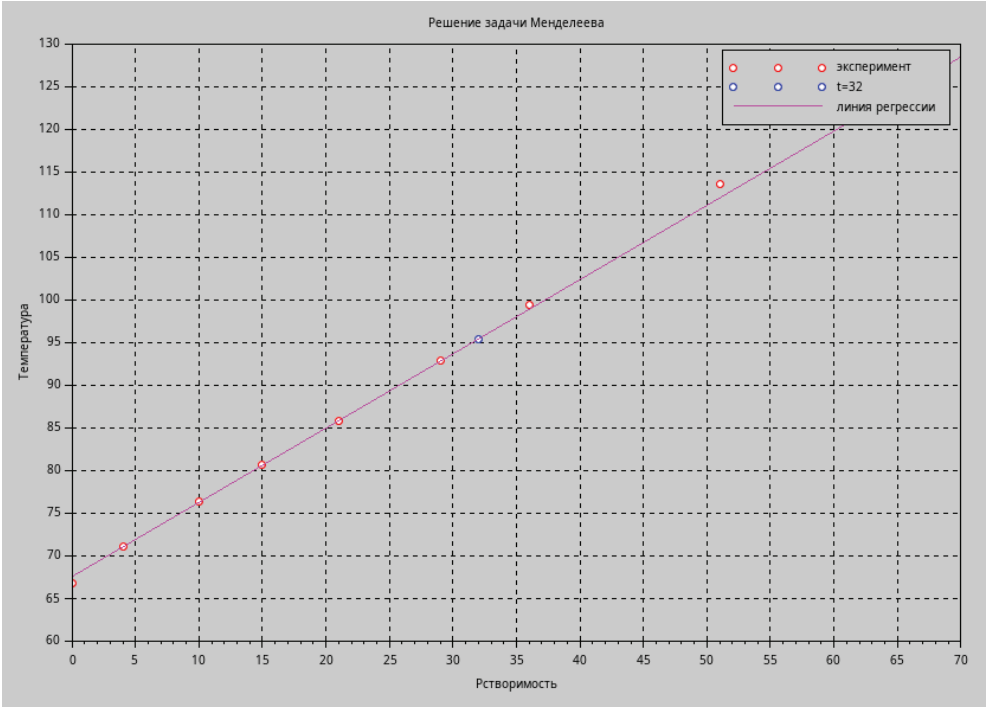

Рис. 9.5. Графическое решение задачи 9.3

9.3 Интерполяция функций

В отличие от задачи аппроксимации, при решении задач интерполяции будет строиться функция, проходящая через экспериментальные точки.

Простейшая *задача интерполирования* заключается в следующем. На отрезке $[a, b]$ заданы $n + 1$ точек $x_1, x_2, x_3, \ldots, x_n, x_{n+1}$ $(a = x_1, b = x_{n+1})$, называемых узлами интерполяции, и значения неизвестной функции $f(x)$ в этих точках:

$$f(x_1) = y_1, f(x_2) = y_2, f(x_3) = y_3, \ldots, f(x_{n+1}) = y_{n+1}. \quad (9.23)$$

Требуется построить интерполирующую функцию $F(x)$, которая в узлах интерполяции принимает те же значения, что и $f(x)$ [8, 3]:

$$F(x_1) = y_1, F(x_2) = y_2, F(x_3) = y_3, \ldots, F(x_{n+1}) = y_{n+1}. \tag{9.24}$$

В общей постановке задача может не иметь однозначного решения или совсем не иметь решений. Задача становится однозначной, если функцию $F(x)$ нужно искать в виде полинома $F(x) = P_n(x)$ степени n, удовлетворяющего условиям (9.24).

Полученную интерполяционную формулу $y = F(x)$ зачастую используют для нахождения приближённых значений данной функции $f(x)$ в точках x, отличных от узлов интерполирования. Такая операция называется интерполированием функции $f(x)$. При этом различают *интерполирование* в узком смысле, когда $x \in [x_1, x_{n+1}]$, и *экстраполирование*, когда $x \notin [x_1, x_{n+1}]$.

Рассмотрим некоторые наиболее часто используемые методы построения интерполяционных полиномов.

9.3.1 Канонический полином

Будем искать интерполирующую функцию $F(x)$ в виде канонического полинома степени n [8, 3].

$$F(x) = P_n(x) = a_1 + a_2 \cdot x + a_3 \cdot x^3 + \cdots + a_{n+1} \cdot x^n. \tag{9.25}$$

Выбор многочлена степени n основан на том факте, что через $n + 1$ точек проходит единственная кривая степени n. Подставив (9.25) в (9.24), получим систему линейных алгебраических уравнений (9.26):

$$\begin{cases} a_1 + a_2 x_1 + a_2 x_1^2 + \cdots + a_{n+1} x_1^n = y_1 \\ a_1 + a_2 x_2 + a_2 x_2^2 + \cdots + a_{n+1} x_2^n = y_2 \\ a_1 + a_2 x_3 + a_2 x_3^2 + \cdots + a_{n+1} x_3^n = y_3. \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_1 + a_2 x_{n+1} + \cdots + a_{n+1} x_{n+1}^n = y_{n+1} \end{cases} \tag{9.26}$$

В матричном виде систему можно записать так:

$$C \cdot a = y. \tag{9.27}$$

Матрица C ($C_{i,j} = x_i^{j-1}$), известная в математике как матрица Вандермонда, имеет ряд особенностей.

1. Система (9.26) имеет решение, если все x_i различны.
2. Коэффициенты матрицы C могут иметь сильно различающиеся значения[1]. Это может привести к вычислительным сложностям при решении системы (9.26), (9.27).

[1] Это может происходить при достаточно большом n ($n > 10$), если часть x_i больше 1, а часть меньше.

Решив эту систему линейных алгебраических уравнений, найдём коэффициенты интерполяционного полинома $a_1, a_1, \ldots, a_{n+1}$. Однако следует учитывать, что при решении системы (9.26) могут возникнуть значительные вычислительные сложности. Поэтому рассмотрим методы построения интерполяционных полиномов, не связанные с решением систем линейных алгебраических уравнений.

9.3.2 Полином Ньютона

И. Ньютон предложил интерполирующую функцию записать в виде следующего полинома n-й степени [8, 3]:

$$F(t) = A_1 + A_2(t - x_1) + A_3(t - x_1)(t - x_2) + \cdots + A_{n+1}(t - x_1)(t - x_2)\ldots(t - x_n). \tag{9.28}$$

Подставим $F(x_1) = y_1$ в (9.28) и вычислим значение коэффициента A_1:

$$A_1 = y_1.$$

Подставим $F(x_2) = y_2$ в (9.28), после чего получим соотношение для вычисления A_2: $F(x_2) = A_1 + A_2(x_1 - x_0) = y_1$.

Отсюда коэффициент A_2 рассчитывается по формуле

$$A_2 = \frac{y_1 - y_2}{x_1 - x_2} = y_{12},$$

где y_{12} – разделённая разность первого порядка, которая стремится к первой производной функции при $x_2 \to x_1$. По аналогии вводятся и другие разделённые разности первого порядка:

$$y_{13} = \frac{y_1 - y_3}{x_1 - x_3}, y_{14} = \frac{y_1 - y_4}{x_1 - x_4}, \ldots, y_{1n+1} = \frac{y_1 - y_{n+1}}{x_1 - x_{n+1}}.$$

Отметим, что количество разделённых разностей первого порядка равно n. Подставим соотношение $F(x_3) = y_3$ в (9.28), в результате чего получим:

$$A_1 + A_2(x_3 - x_1) + A_3(x_3 - x_1)(x_3 - x_2) = y_3,$$

$$y_1 + y_{12}(x_3 - x_1) + A_3(x_3 - x_1)(x_3 - x_2) = y_3.$$

Отсюда A_3 вычисляется по формуле

$$A_3 = y_{123} = \frac{y_{12} - y_{13}}{x_2 - x_3},$$

здесь y_{123} – разделённая разность второго порядка, эта величина стремится ко второй производной при $x_2 \to x_3$.

Аналогично вводятся

$$y_{124} = \frac{y_{12} - y_{14}}{x_1 - x_4},$$

$$y_{125} = \frac{y_{12} - y_{15}}{x_2 - x_5},$$

$$\ldots,$$

$$y_{12n+1} = \frac{y_{12} - y_{1n+1}}{x_2 - x_{n+1}}.$$

Всего имеем $n - 1$ разделённых разностей второго порядка.
Подставим $F(x_4) = y_4$ в (9.28), после чего получим

$$A_4 = y_{1234} = \frac{y_{123} - y_{124}}{x_3 - x_4}.$$

Аналогично можно ввести коэффициенты

$$y_{1235} = \frac{y_{123} - y_{125}}{x_3 - x_5}, \ldots, y_{123n+1} = \frac{y_{123} - y_{12n+1}}{x_3 - x_{n+1}}.$$

Этот процесс будем продолжать до тех пор, пока не вычислим

$$A_n = y_{123\ldots n+1} = \frac{y_{123\ldots n} - y_{123\ldots n+1}}{x_n - x_{n+1}}.$$

Полученные результаты запишем в табл. 9.5 [8, 3].

Таблица 9.5. Таблица разделённых разностей полинома Ньютона

x	$f(x)$	1	2	3	4	…	n
x_1	y_1						
x_2	y_2	y_{12}					
x_3	y_3	y_{13}	y_{123}				
x_4	y_4	y_{14}	y_{124}	y_{1234}			
x_5	y_5	y_{15}	y_{125}	y_{1235}	y_{12345}		
…	…	…	…	…	…	…	…
x_{n+1}	y_{n+1}	y_{1n+1}	y_{12n+1}	y_{123n+1}	$y_{1234n+1}$	…	$y_{123\ldots n+1}$

В вычислении по формуле (9.28) будут участвовать только диагональные элементы таблицы (т. е. коэффициенты A_i), а все остальные элементы таблицы являются промежуточными и нужны для вычисления диагональных элементов. Рассмотрим более подробно алгоритм построения полинома.

Этап 1. Формирование коэффициентов полинома Ньютона A.

1. Переписываем в массив A значения y.

2. Определяем `j=2`, формируем массив разделённых разностей `j-1` порядка следующим образом:

```
for i=j:N
A(i)=(A(i)-A(j-1))/(x(i)-x(j-1));
end
```

3. Повторяем пункт 2 при `j=3:N`.

Пример реализации подобного алгоритма приведён ниже.

Листинг 9.9. Вычисление коэффицентов полинома Ньютона

```
function A=newton(x,y)
  N=length(y);
  for i=1:N
    A(i)=y(i)
  end
  for j=2:N
    for i=j:N
      A(i)=(A(i)-A(j-1))/(x(i)-x(j-1));
    end
  end
endfunction
```

Этап 2. Вычисление значения полинома Ньютона в точке t. Используем вычислительную схему, аналогичную схеме Горнера.

Листинг 9.10. Вычисление полинома Ньютона в точке t

```
function b=calc_newton(x,A,t)
  N=length(x)
  b=A(N)
  for i=N-1:-1:1
    b=A(i)+(t-x(i))*b
  end
endfunction
```

9.3.3 Полином Лагранжа

Ещё одно представление интерполяционного полинома степени n предложил Лагранж [8, 3]:

$$F(t) = \sum_{i=1}^{n+1} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{t - x_j}{x_i - x_j}. \tag{9.29}$$

Напомним читателю, что рассмотренные три способа построения полинома – это три различные формы записи одной и той же функции.

Совет. Полином Лагранжа лучше использовать, если необходимо вычислить значение в небольшом количестве точек. Для расчёта во многих точках рационально использовать полином Ньютона, в котором можно один раз вычислить значения коэффициентов A_i, после чего можно рассчитать ожидае-

мое значение в точках по формуле (9.28). При использовании канонического полинома приходится решать систему линейных алгебраических уравнений (9.26), поэтому он используется значительно реже.

Полиномиальная интерполяция не всегда даёт удовлетворительные результаты при аппроксимации зависимостей. При представлении полиномами возможна большая погрешность на концах этих кривых. Несмотря на выполнение условий в узлах, интерполяционная функция может иметь значительное отклонение от аппроксимируемой кривой между узлами. При этом повышение степени интерполяционного полинома приводит не к уменьшению, а к увеличению погрешности. Решение этой проблемы предложено теорией сплайн-интерполяции (от анг. *spline* – рейка, линейка).

9.3.4 Интерполяция сплайнами

Рассмотрим один из наиболее распространённых вариантов интерполяции – интерполяцию кубическими сплайнами. Было установлено [8], что недеформируемая линейка между соседними углами проходит по линии, удовлетворяющей уравнению [8, 3]:

$$\varphi^{IV}(x) = 0. \tag{9.30}$$

Функцию $\varphi(x)$ будем использовать для интерполяции зависимости $y(x)$, заданной на интервале (a, b) в узлах $a = x_1, x_2, \ldots, x_n = b$ значениями $y_1, y_2, \ldots, y_n$.

Кубическим сплайном, интерполирующим на отрезке $[a, b]$ данную функцию $y(x)$, называется функция [8]

$$\begin{gathered} g_k(s) = a_k + b_k(s - x_k) + c_k(s - x_k)^2 + d_k(s - x_k)^3, \\ s \in [x_{k-1}, x_k],\ k = 2, 3, \ldots, n, \end{gathered} \tag{9.31}$$

удовлетворяющая следующим условиям [8, 3]:

- $g_k(x_k) = y_k$; $g_k(x_{k-1}) = y_{k-1}$ (условие интерполяции в узлах сплайна);
- функция $g(x)$ дважды непрерывно дифференцируема на интервале $[a, b]$;
- на концах интервала функция g должна удовлетворять следующим соотношениям: $g_1''(a) = g_n''(b) = 0$.

Для построения интерполяционного сплайна необходимо найти $4(n - 1)$ коэффициентов a_k, b_k, c_k, d_k $(k = 2, 3, \ldots, n)$.

Из определения сплайна получаем n соотношений (9.33) [8, 3]:

$$g_2(x_1) = y_1, g_k(x_k) = y_k,\ k = 2, 3, \ldots, n. \tag{9.32}$$

Из условий гладкой стыковки звеньев сплайна (во внутренних узловых точках совпадают значения двух соседних звеньев сплайна[1], их первые и вторые производные) получаем ещё ряд соотношений (9.33)–(9.34) [8]:

$$g_{k-1}(x_{k-1}) = g_k(x_k),$$

[1] Звеном сплайна называется функция $g_i(x)$ на интервале $[x_{i-1}, x_i]$.

$$\begin{gathered} g'_{k-1}(x_{k-1}) = g'_k(x_k), \\ g''_{k-1}(x_{k-1}) = g''_k(x_k), \\ k = 2, 3, \ldots, n; \end{gathered} \tag{9.33}$$

$$g''_1(x_0) = 0, g_n(x_n) = 0. \tag{9.34}$$

Соотношения (9.32)–(9.34) образуют $4(n-1)$ соотношений для нахождения коэффициентов сплайна [8, 3]. Подставляя выражения функций (9.31) и их производных (9.35)

$$\begin{gathered} g'_k(s) = b_k + 2c_k(s - x_k) + 3d_k(s - x_k)^2, \\ g''_k(s) = 2c_k + 6d_k(s - x_k) \end{gathered} \tag{9.35}$$

в соотношения (9.32)–(9.34) и принимая во внимание соотношение

$$h_k = x_k - x_{k-1}, k = 2, 3, \ldots, n, \tag{9.36}$$

получим следующую систему уравнений (9.37)–(9.43):

$$a_2 - b_2h_2 + c_2h_2^2 - d_2h_2^3 = y_1, \tag{9.37}$$

$$a_k = y_k, k = 2, 3, \ldots, n, \tag{9.38}$$

$$a_{k-1} = a_k - b_kh_k + c_kh_k^2 - d_kh_k^3, k = 3, 4, \ldots, n, \tag{9.39}$$

$$b_{k-1} = b_k - 2c_kh_k + 3d_kh_k^2, k = 3, 4, \ldots, n, \tag{9.40}$$

$$c_{k-1} = c_k - 3d_kh_k, k = 3, 4, \ldots, n, \tag{9.41}$$

$$c_2 - 3d_2h_2 = 0, \tag{9.42}$$

$$c_n = 0. \tag{9.43}$$

Задача интерполяции свелась к решению системы (9.37)–(9.43) [8, 3]. Из соотношения (9.38) следует, что все коэффициенты $a_k = y_k, k = 2, 3, \ldots, n$. Подставив соотношения (9.37), (9.38) в (9.39) и используя фиктивный коэффициент $c_1 = 0$, получим соотношение между b_k, c_k и d_k : $b_kh_k - c_kh_k^2 + d_kh_k^3 = y_k - y_{k-1}$.

Отсюда коэффициенты b_k вычисляются по формуле [8, 3]

$$b_k = \frac{y_k - y_{k-1}}{h_k} + c_kh_k - d_kh_k^2, \quad k = 2, 3, \ldots, n. \tag{9.44}$$

Из (9.41) и (9.42) выразим d_k через c_k (с учётом коэффициента $c_1 = 0$) [8, 3]:

$$d_k = \frac{c_k - c_{k-1}}{3h_k}, \quad k = 2, 3, \ldots, n. \tag{9.45}$$

Подставим (9.45) в (9.44):

$$b_k = \frac{y_k - y_{k-1}}{h_k} + \frac{2}{3}c_kh_k + \frac{1}{3}h_kc_{k-1}, \quad k = 2, 3, \ldots, n. \tag{9.46}$$

Введём обозначение

$$l_k = \frac{y_k - y_{k-1}}{h_k}, \quad k = 2, 3, \ldots, n, \tag{9.47}$$

после чего соотношение (9.44) примет вид:

$$b_k = l_k + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1}, \quad k = 2, 3, \ldots, n. \tag{9.48}$$

Подставим (9.48) и (9.45) в соотношение (9.40), получим систему относительно c_k [8, 3]:

$$h_{k-1} c_{k-2} + 2(h_{k-1} + h_k) c_k = 3(l_k - l_{k-1}), k = 3, 4, \ldots, n, \tag{9.49}$$

$$c_0 = 0, c_n = 0. \tag{9.50}$$

Систему (9.49) можно решить, используя метод прогонки (http://ru.wikipedia.org/wiki/Метод_прогонки). Этот метод сводится к нахождению прогоночных коэффициентов по формулам прямой прогонки [8, 3]:

$$\delta_2 = -\frac{1}{2} \frac{h_3}{h_2 + h_3}, \quad \lambda_2 = \frac{3}{2} \frac{l_3 - l_2}{h_2 + h_3}, \tag{9.51}$$

$$\begin{aligned} \delta_{k-1} &= -\frac{h_k}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}}, \\ \lambda_{k-1} &= \frac{3l_k - 3l_{k-1} - h_{k-1}\lambda_{k-2}}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}}, \end{aligned} \quad k = 4, 5 \ldots, n, \tag{9.52}$$

а затем к нахождению искомых коэффициентов c_k по формулам обратной прогонки [8, 3]:

$$c_{k-1} = \delta_{k-1} c_k + \lambda_{k-1}, k = n, n-1, \ldots, 3. \tag{9.53}$$

После нахождения коэффициентов c по формуле (9.53) находим b и d по формулам (9.48), (9.45) [8, 3].

Таким образом, алгоритм расчёта коэффициентов интерполяционного сплайна можно свести к следующим шагам [8, 3].

Шаг 1. Ввод значений табличной зависимости $y(x)$, массивов x и y.

Шаг 2. Расчёт элементов массивов h и l по формулам (9.36) и (9.47).

Шаг 3. Расчёт массивов прогоночных коэффициентов δ и λ по формулам (9.51), (9.52).

Шаг 4. Расчёт массивов коэффициентов c по формуле (9.53).

Шаг 5. Расчёт массивов коэффициентов b по формуле (9.48).

Шаг 6. Расчёт массивов коэффициентов d по формуле (9.45).

После этого в формулу (9.31) можно подставлять любую точку s и вычислять ожидаемое значение.

Задача 9.4

Найти приближённое значение функции при заданном значении аргумента с помощью интерполяции кубическими сплайнами в точках $x_1 = 150$, $x_2 = 160$, $x_3 = 165$. Функция задана таблично (табл. 9.6).

Таблица 9.6. Исходные данные к задаче 9.4

132	140	162	170	175	180
330	350	385	425	150	485

Код программы с комментариями приведён ниже.

```
clear;
//Функция вычисления коэффициентов сплайна b,c,d
function [b, c,d]=koef_spline(x,y)
  n=length(x);
  //Вычисление элементов массива h (11.36)
  for k=2:n
    h(k)=x(k)-x(k-1);
  end
  //Вычисление элементов массива l (11.47)
  for k=2:n
    l(k)=(y(k)-y(k-1))/h(k);
  end
  //Вычисление прогоночных коэффициентов (11.52)-(11.53)
  delt(2)=-h(3)/(2*(h(3)+h(2)));
  lyam(2)=1.5*(l(3)-l(2))/(h(3)+h(2));
  for k=4:n
    delt(k-1)=-h(k)/(2*(h(k-1)+h(k))+h(k-1)*delt(k-2));
    lyam(k-1)=(3*(l(k)-l(k-1))-h(k-1)*lyam(k-2))/...
    (2*(h(k-1)+h(k))+h(k-1)*delt(k-2));
  end
  //Вычисление коэффициентов сплайна c (11.49)-(11-50)
  c(n)=0;
  for k=n:-1:3
    c(k-1)=delt(k-1)*c(k)+lyam(k-1);
  end
  //Расчёт коэффициентов сплайна b,d (11.48), (11.45)
  for k=2:n
    d(k)=(c(k)-c(k-1))/3/h(k);
    b(k)=l(k)+(2*c(k)*h(k)+h(k)*c(k-1))/3;
  end
endfunction
//Расчёт ожидаемого значения в точке t
function z=my_spline(x,y,t)
  //Вычисление коэффициентов сплайна
  //Рекомендуем читателю подумать, имеет ли смысл перенести вызов этой
  //функции в главную программу. В каких случаях?
  [b,c,d]=koef_spline(x,y);
  n=length(x);
  a=y;
```

```
  //Определяем номер интервала j, в котором находится точка t
  if t>x(n-1)
    j=n;
  else
    for i=2:n-1
      if t<=x(i)
        j=i;
        break
      end
    end
  end
  //Вычисление ожидаемого значения в точке t
  z=a(j)+b(j)*(t-x(j))+c(j)*(t-x(j))^2+d(j)*(t-x(j))^3;
endfunction
//Исходные данные, зависимость P(U)
U=[132 140 162 170 180];
P=[330 350 385 425 485];
//Точки, в которых требуется вычислить ожидаемое значение.
UP=[150 160 165];
//Вычисление ожидаемого значения.
for i=1:length(UP)
  PP(i)=my_spline(U,P,UP(i))
end
//Формирование интерполяционной кривой на интервале [131;190]
U2=131:0.05:190;
for i=1:length(U2)
  P2(i)=my_spline(U,P,U2(i))
end
//Графическая интерпретация решения.
figure();
plot(U',P',"ro",UP,PP,"rx",U2',P2',"b-");xgrid;
xtitle("Точечный график P(U)","U,В","P,Вт");
legend("Эксперимент","Расчётные точки","Сплайн")
```

На рис. 9.6 представлено решение задачи 9.4 в графическом виде.

Расчёт коэффициентов кубического сплайна довольно сложен, и зачастую в инженерной практике вместо кубического сплайна используется *линейная интерполяция* (*линейный сплайн*). Использование линейного сплайна оправдано в случае, если необходимо просто вычислить значение в определённых точках и нет требования непрерывности производных интерполяционной функции.

В случае линейной интерполяции в качестве сплайна выступает линейная функция [8, 3]

$$f_k(s) = a_k + b_k s,\ s \in [x_{k-1}, x_k],\ k = 2, 3, \ldots, n+1, \tag{9.54}$$

удовлетворяющая условию интерполяции в узлах сплайна $f_k(x_k) = y_k$; $f_k(x_{k-1}) = y_{k-1}$. Коэффициенты a и b в этом случае рассчитываются по формулам (9.55), которые получаются из уравнения прямой, проходящей через две точки с координатами (x_{k-1}, y_{k-1}), (x_k, y_k) [8, 3].

$$a_k = y_{k-1} - \frac{y_k - y_{k-1}}{x_k - x_{k-1}} x_{k-1},\quad b_k = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}. \tag{9.55}$$

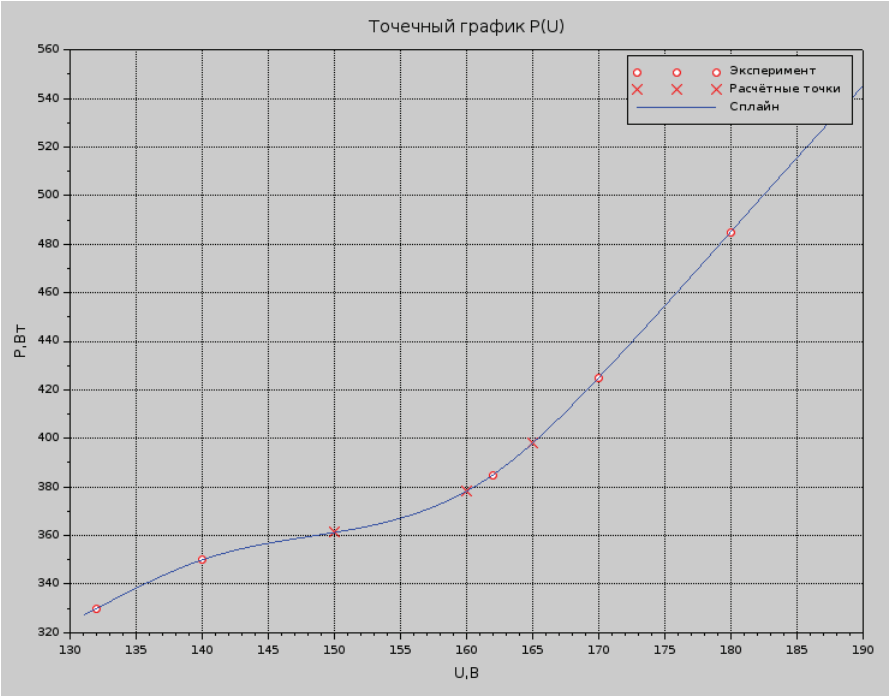

Рис. 9.6. Графическое решение задачи 9.4

Если коэффициенты линейного сплайна известны, можно рассчитать значения в любой точке интервала $[x_1, x_n]$. Линейная интерполяция даёт достаточно хорошие результаты при практическом счёте внутри интервала $[x_1, x_n]$, когда от получаемой функции не требуют дополнительных свойств (дифференцируемости и т. д.).

Как видите, построение линейного сплайна намного проще. Рекомендуем читателю самостоятельно решить уже известную задачу 9.4, используя линейный сплайн.

9.4 Встроенные функции Scilab для решения задачи интерполяции

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 точках `x`.

Рассмотрим использование функции `interpln` на примере следующей задачи.

Задача 9.5

В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P, Вт) от входного напряжения (U, В) для асинхронного двигателя (табл. 9.3). Построить зависимость вида $P(U)$ с помощью функции `interpln`.

Листинг 9.11. Решение задачи 9.5 с помощью 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];
z=[x;y];
t=132:1:252;
figure();
ptd=interpln(z,t);
plot(x,y,"ro",t,ptd,"b-");
xtitle("Линейный сплайн","X","Y");
xgrid;
```

Графическое решение задачи показано на рис. 9.7.

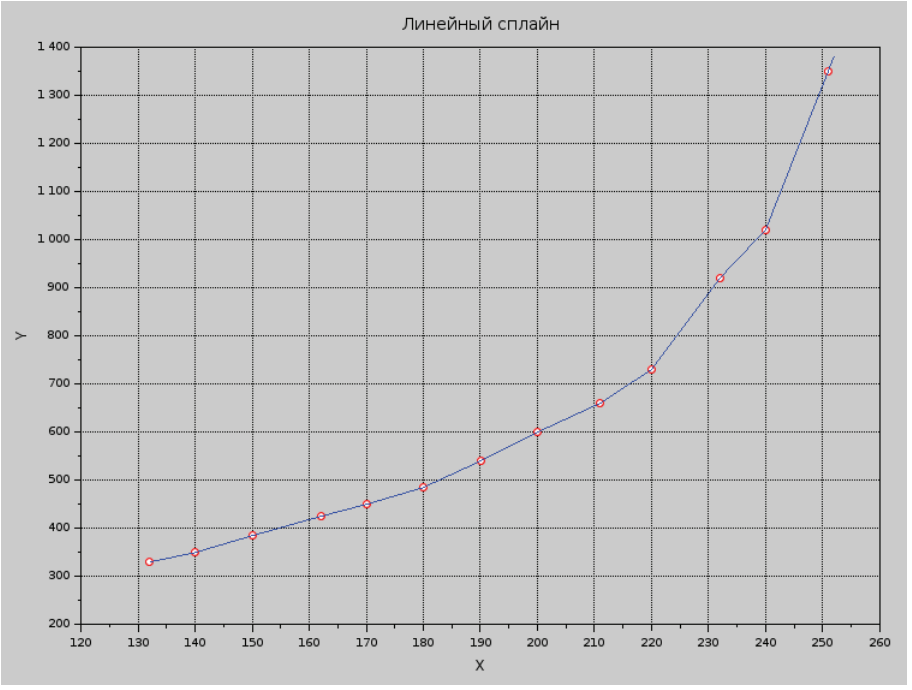

Рис. 9.7. Графическое решение задачи 9.5

Построение *кубического сплайна* в Scilab состоит из двух этапов: вначале вычисляются коэффициенты сплайна с помощью функции `d=splin(x,y)`,

а затем рассчитываются 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`.

Для практического знакомства с функциями `splin(x,y)` и `interp(t,x,y,d)` решим следующую задачу.

Задача 9.6

Найти приближённое значение функции при заданном значении аргумента с помощью интерполяции кубическими сплайнами в точках $x_1 = 0{,}702$, $x_2 = 0{,}512$, $x_3 = 0{,}608$. Функция задана таблично (табл. 9.7).

Таблица 9.7. Исходные данные к задаче 9.6

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 9.12. Решение задачи 9.6

```
//Экспериментальные данные
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];
//Расчёт коэффициентов сплайна
koeff=splin(x,y);
//Расчётные значения
X=[0.702 0.512 0.608];
//Значение функции в заданных точках
Y=interp(X,x,y,koeff)
//Построение кубического сплайна
t=0.43:0.01:0.75;
ptd=interp(t,x,y,koeff);
//Построение графика
figure();
plot(x,y,"bo",X,Y,"b*",t,ptd,"r-");
xgrid;
legend("эксперимент","расчётные значения","сплайн",2);
```

Графическое решение задачи представлено на рис. 9.8.

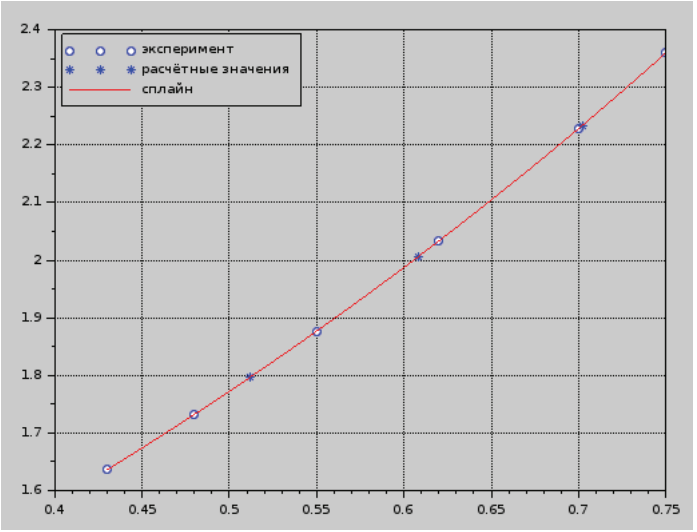

Рис. 9.8. Графическое решение задачи 9.6

Глава 10

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 описана реализация метода сеток для решения параболических, гиперболических и эллиптических уравнений в Scilab.

10.1 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 её частными производными вида [8]:

$$\begin{aligned} L(u) = A(x,y)\frac{\partial^2 u}{\partial x^2} + 2B(x,y)\frac{\partial^2 u}{\partial x \partial y} + C(x,y)\frac{\partial^2 u}{\partial y^2} \\ + D(x,y)\frac{\partial u}{\partial x} + E(x,y)\frac{\partial u}{\partial y} + G(x,y)u(x,y) = F(x,y). \end{aligned} \tag{10.1}$$

[1] В литературе эти уравнения часто называют уравнениями математической физики.

Если переменная функция u зависит от x и t, то уравнение (10.1) может быть записано следующим образом:

$$L(u) = A(x,t)\frac{\partial^2 u}{\partial x^2} + 2B(x,t)\frac{\partial^2 u}{\partial x \partial t} + C(x,t)\frac{\partial^2 u}{\partial t^2} + D(x,t)\frac{\partial u}{\partial x} + E(x,t)\frac{\partial u}{\partial t} + G(x,t)u(x,t) = F(x,t). \quad (10.2)$$

Если в правой части уравнения $F = 0$, то уравнения (10.1)–(10.2) называются однородными, иначе – неоднородными [9].

Если $B^2 - 4AC < 0$, то уравнение (10.1) относится к классу *эллиптических уравнений*, если $B^2 - 4AC > 0$, то (10.2) – это *гиперболическое уравнение*, если $B^2 - 4AC = 0$ – *параболическое уравнение*. В случае когда $B^2 - 4AC$ не имеет постоянного знака, 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 (10.2) – *эллиптическое уравнение*, если разные, то *гиперболическое*, а если A или C равно 0, то уравнение относится к *параболическим* [9].

К классическим эллиптическим уравнениям относятся [8, 9]:

- 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 [13]:

- волновые уравнения: одномерное *волновое* уравнение $\frac{\partial^2 u}{\partial t^2} = a^2\frac{\partial^2 u}{\partial x^2} + f(x,t)$, которое описывает вынужденные колебания струны; двумерное волновое уравнение $\frac{\partial^2 u}{\partial t^2} = a^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x,y,t)$, описывает колебания мембраны[2];
- *телеграфное* уравнение $\frac{\partial^2 u}{\partial t^2} + \frac{RC + LG}{LC}\frac{\partial u}{\partial t} + \frac{RG}{LC}u - \frac{1}{LC}\frac{\partial^2 u}{\partial x^2} = 0$, описывает изменение потенциала u в линиях электропередачи; L, C, R, G – коэффициент самоиндукции, ёмкость, сопротивление, характеристика потерь на единицу длины линии.

---

[1] В двумерном случае оператор Лапласа имеет вид $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, в трёхмерном – $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$.

[2] При $f = 0$ уравнение описывает свободные колебания струны или мембраны.

К классическим *параболическим* уравнениям относится уравнение *теплопроводности* $\frac{\partial u}{\partial t} = a^2 \Delta u + f$.

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 [13]:

- условие *Дирихле* $u_{(x,y,z)\in\Gamma} = \varphi(x, y, z)$, в этом случае на границе области Г, в которой ищут 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* $\left.\frac{\partial u}{\partial n}\right|_{(x,y,z)\in\Gamma} = \varphi(x, y, z)$, в этом случае на границе области задана производная по направлению n внешней нормали;
- смешанное условие $\left(\alpha u + \beta \frac{\partial u}{\partial n}\right)_{(x,y,z)\in\Gamma} = \varphi(x, y, z)$.

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 $\frac{\partial u(x,t_0)}{\partial t} = \psi_2(x)$.

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.

10.2 Использование метода сеток для решения параболических уравнений в частных производных

Одним из наиболее распространенных численных методов решения уравнений является *метод сеток*[1] [8]. В методе сеток область Ω, в которой необходимо найти решение уравнения, прямыми, параллельными осям $t = t_j$ и $x = x_i$, разобьем на прямоугольные области (см. рис. 10.1), где $x_i = x_0 + ih$, $h = \frac{x_n - x_0}{h}$,

[1] Метод сеток в литературе также называют методом конечных разностей.

$i = 0, 1, 2, \ldots, n$; $t_j = t_0 + j\Delta$, $\Delta = \dfrac{t_k - t_0}{k}$, $j = 0, 1, \ldots, k$. Точки, которые лежат на границе Г 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 $w_j = w(x_i, t_j)$, 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 данной главы будет рассмотрен метод сеток для каждого из трёх 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 (10.3):

$$\begin{gathered}\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \quad 0 \le x \le L, \quad 0 \le t \le T,\\ u(0,t) = \mu(t), \quad u(L,t) = \eta(t), \quad 0 \le t \le T,\\ u(x,0) = \varphi(x), \quad 0 \le x \le L.\end{gathered} \tag{10.3}$$

Построим сетку Ω_h^{Δ} (см. рис. 10.1). Для получения сеточного уравнения заменим производную $\dfrac{\partial^2 u}{\partial x^2}$ приближённой разностной формулой [8]:

$$\frac{\partial^2 u(x_i,t_j)}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}. \tag{10.4}$$

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 приближённых разностных формул [8]:

$$\frac{\partial u(x_i,t_j)}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta}, \tag{10.5}$$

$$\frac{\partial u(x_i,t_j)}{\partial t} = \frac{u_{i,j} - u_{i,j-1}}{\Delta}. \tag{10.6}$$

Кроме того, заменим начальные и граничные условия их разностной аппроксимацией:

$$u_{i,0} = \varphi(x_i) = \varphi_i, \quad i = 0, 1, \ldots, n, \tag{10.7}$$

$$u_{0,j} = \mu(t_j) = \mu_j, \quad u_{n,j} = \nu(t_j) = \nu_j, \quad j = 0, 1, \ldots, k. \tag{10.8}$$

Заменив частные производные в задаче (10.3) соотношениями (10.4) и (10.5) и учитывая условия (10.7)–(10.8), получим следующую вычислительную схему для расчёта значений функции u в узлах сетки Ω_h^{Δ}:

$$u_{i,j+1} = \gamma u_{i,j-1} + (1 - 2\gamma)u_{i,j} + \gamma u_{i,j+1} + \Delta f_{i,j}, \tag{10.9}$$

$$u_{0,j} = \mu_j, \quad u_{n,j} = \nu_j, \quad u_{i,0} = \varphi_i, \quad \gamma = \frac{a^2\Delta}{h^2}. \tag{10.10}$$

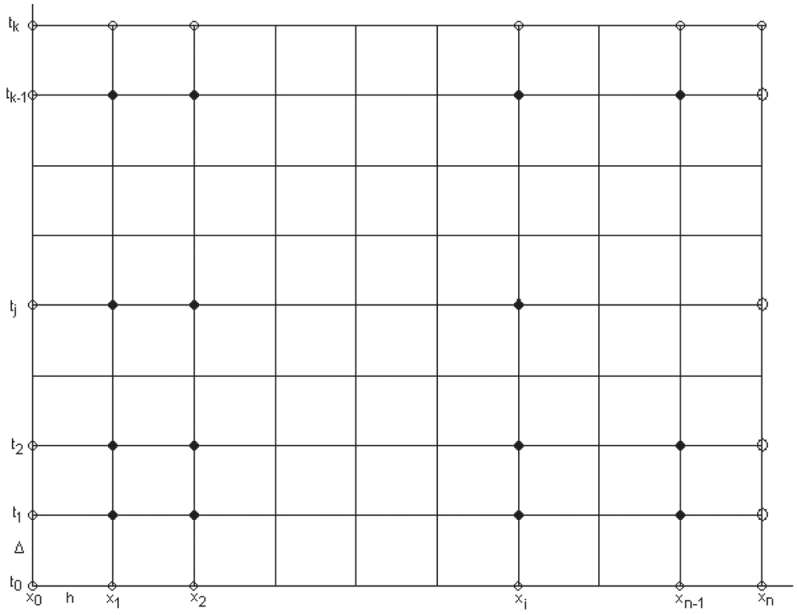

Рис. 10.1. Сетка Ω_h^{Δ} для области Ω с границей Γ

Это *явная двухслойная* разностная схема (см. рис. 10.2).

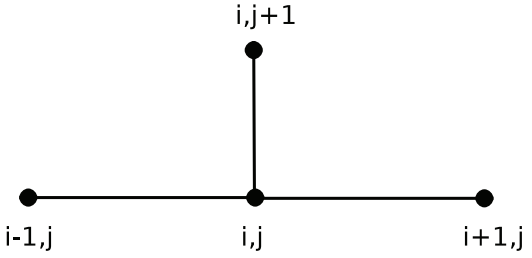

Рис. 10.2. Шаблон явной двухслойной разностной схемы

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 (10.8) можно сначала явно рассчитать значения $u_{i,1}$, затем $u_{i,2}$ и так до $u_{i,k}$. Для устойчивости разностной схемы (10.8) значения шагов по t и x должны удовлетворять следующему условию:

$$\Delta \le \frac{h^2}{2a^2}. \tag{10.11}$$

Рассмотрим решение параболического уравнения на примере следующей задачи.

Задача 10.1

Решить параболическое уравнение, описывающее распределение температуры в стержне длиной L, начальная температура стержня задаётся произвольной функцией $\varphi(x)$. Температуры концов стержня равны $u(0, t) = U_1 = \text{const}$, $u(L, t) = U_2 = \text{const}$.

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \quad a^2 = \frac{\lambda}{c\rho}, \quad 0 \le x \le L, \quad 0 \le t < \infty,$$

$$u(0, t) = U_1 = \text{const}, \quad u(L, t) = U_2 = \text{const}, \quad 0 \le t < \infty, \tag{10.12}$$

$$u(x, 0) = \varphi(x), \quad 0 \le x \le L.$$

Здесь a – коэффициент температуропроводности, λ – коэффициент теплопроводности материала стержня, c – удельная теплоёмкость, ρ – плотность.

Подпрограмма решения задачи 10.1 с помощью явной разностной схемы (10.9)–(10.10) в Scilab представлена в листинге 10.1[1].

Листинг 10.1. Подпрограмма решения задачи 10.1 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
endfunction
//Условие на правой границе
function y=nyu(x)
  y=2.117
```

[1] В листинге 10.1 и во всех последующих уже учтен тот факт, что массивы в Scilab нумеруются с 1.

```
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 строки
  //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 10.1 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, определённая на сетке Ω_h^{Δ}, массивы x и t.

В листинге 10.2 представлено обращение к функции `parabol` для решения задачи 10.1 и построение графика решения, который изображен на рис. 10.3.

Листинг 10.2. Вызов функции `parabol`

```
[U,X,T]=parabol(50,200,5,3,0.4);
surf(X,T,U');
title('Явная схема');
xlabel('X');
ylabel('T');
```

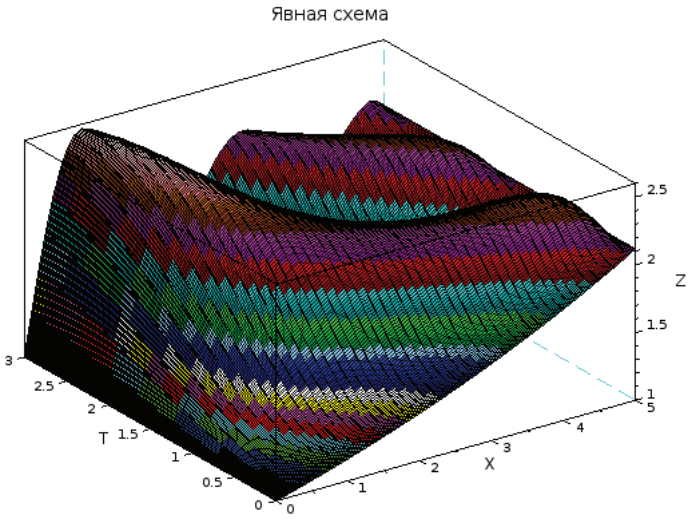

Рис. 10.3. График решения задачи 10.1 при $f(x, t) = \sin(xt)$

При решении параболических уравнений с помощью явной разностной схемы основной проблемой является устойчивость решения и правильный выбор шага по t, удовлетворяющего соотношению (10.11).

Для решения этой проблемы были предложены неявные разностные схемы [8]. Эти схемы абсолютно устойчивы, но алгоритм решения получаемого сеточного уравнения несколько сложнее, чем простой пересчёт по формуле (10.9). Рассмотрим неявную разностную схему для параболического уравнения. Для построения неявной разностной схемы заменим частные производные в задаче (10.1) соотношениями (10.4), (10.6[1]) и с учётом условий (10.7)–(10.8) получим следующую вычислительную схему для расчёта значений функции u в узлах сетки Ω_h^{Δ}.

$$\gamma u_{i-1,j} - (1 + 2\gamma)u_{i,j} + \gamma u_{i+1,j} = -u_{i,j-1} - \Delta f_{i,j},$$
$$i = 1, 2, \ldots, n-1, \quad j = 1, 2, \ldots, k. \tag{10.13}$$

Соотношения (10.13) вместе с равенствами (10.10) – *неявная двухслойная* разностная схема (см. рис. 10.4). Схема (10.10), (10.13) не позволяет явно выписать решение, и для нахождения $u_{i,j}$ при каждом значении j необходимо решить трёхдиагональную систему линейных алгебраических уравнений, для чего можно воспользоваться одним из итерационных методов (например, методом Зейделя [8] или прогонки [8]). Преобразуем систему (10.13) к следующему виду:

$$u_{i,j} = \frac{\gamma}{1 + 2\gamma}(u_{i-1,j} + u_{i+1,j}) + \frac{u_{i,j-1}}{1 + 2\gamma} + \frac{\Delta}{1 + 2\gamma} f(x_i, t_j). \tag{10.14}$$

[1] Вместо (10.5) для явной схемы.

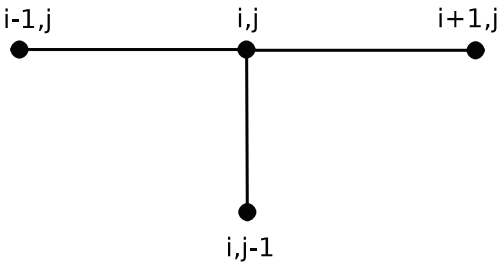

Рис. 10.4. Шаблон неявной двухслойной разностной схемы

Формула (10.14) позволит запрограммировать решение системы, получаемой с помощью неявной разностной схемы, одним из численных методов, например методом Зейделя.

Решение параболического уравнения с помощью неявной разностной схемы (с помощью функции `neyavn`) представлено в листинге 10.3.

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 (10.14) методом Зейделя[1]. Функция `neyavn` возвращает матрицу решений u задачи 10.1; массивы x и t; `r` – точность решения системы (10.14) методом Зейделя; количество итераций `k`.

Листинг 10.3. Решение задачи 10.1 с помощью неявной разностной схемы

```
//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
```

[1] Выбор метода Зейделя при программировании трёхдиагональной системы линейных алгебраических уравнений вида (10.13)–(10.14) – субъективный выбор авторов. Читатель может использовать и другой итерационный метод или специализированный метод решения подобных систем типа метода прогонки.

```
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 строки 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 её нулями
  R(N+1,K+1)=0;
  //Вычисляем коэффициент gamma
  gam=a^2*delta/h^2;
  r=1;
  k=0;
  //Цикл while для организации итерационного процесса
  //при решении системы уравнений методом Зейделя
  //с точностью eps
  while r>eps
    //Вычисление матрицы ошибок R во внутренних точках
    //и пересчёт значений u во внутренних точках при решении СЛАУ
    // 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('Неявная схема');
xlabel('X');
ylabel('T');
```

На рис. 10.5 представлен график решения задачи, которое было получено с помощью неявной разностной схемы.

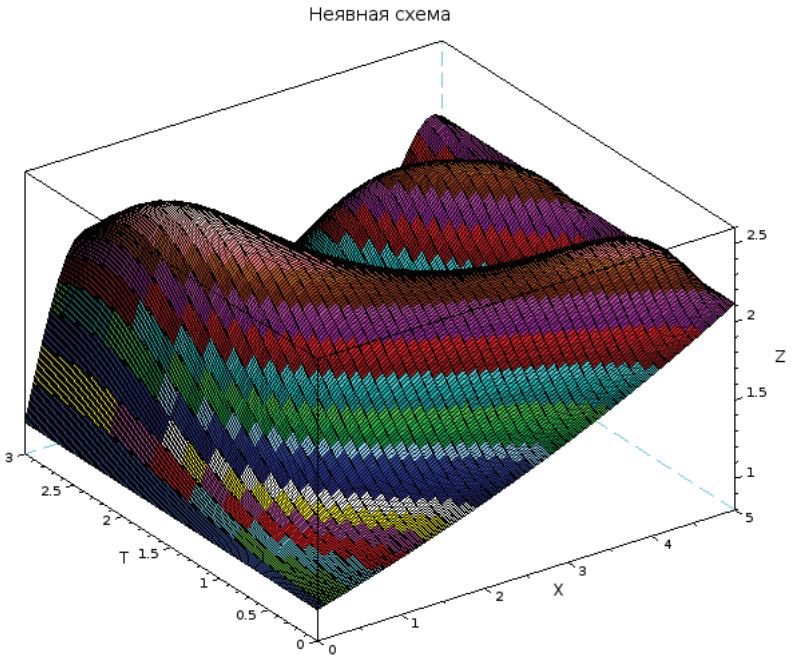

Рис. 10.5. График решения задачи 10.1 при $f(x, t) = \sin(xt)$

Как и ожидалось, разница между решениями, полученными с помощью явной и неявной разностных схем, начинает расти при увеличении x и t (см. рис. 10.6).

Обратите внимание, что функции, приведённые на листингах 10.1 и 10.3, позволяют решать уравнения вида (10.3) 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 ещё одной задачи.

Задача 10.2

Найти распределение температуры в стержне длиной L, начальная температура стержня задаётся 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.

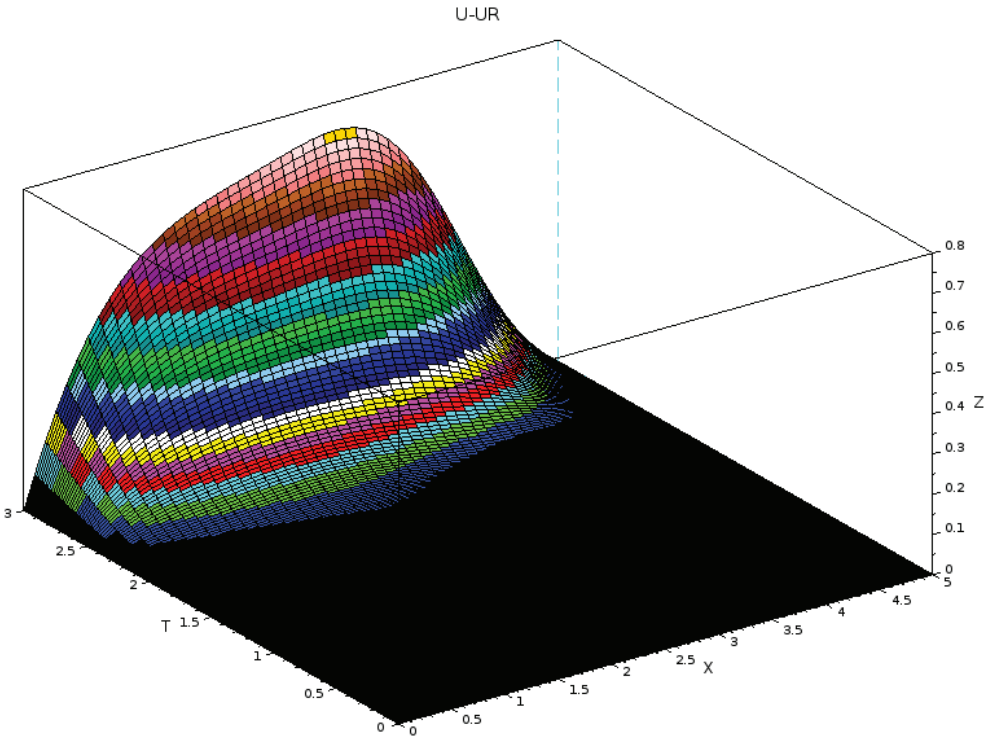

Рис. 10.6. Разность между решениями,
найденными с помощью явной и неявной схем

Начально-граничная задача, описывающая распределение температуры стержня, имеет вид:

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} - h(u - u_0),$$

$$a^2 = \frac{\lambda}{c\rho}, \quad h = \frac{\alpha p}{c\rho\sigma}, \quad 0 \le x \le L, \quad 0 \le t < \infty, \tag{10.15}$$

$$u(0, t) = U_1 = \text{const}, \quad u(L, t) = U_2 = \text{const}, \quad 0 \le t < \infty,$$

$$u(x, 0) = \varphi(x), \quad 0 \le x \le L.$$

Здесь α – коэффициент теплообмена, σ – площадь поперечного сечения стержня, p – периметр поперечного сечения стержня.

Построим сетку Ω_h^{Δ} $(x_i = ih, h = \frac{L}{h}, i = 0, 1, 2, \ldots, n)$, $(t_j = j\Delta, \Delta = \frac{T}{k}, j = 0, 1, \ldots, k)$. Для получения сеточного уравнения заменим производные $\frac{\partial^2 u}{\partial x^2}$ и $\frac{\partial u}{\partial t}$ приближёнными разностными формулами (10.4) и (10.6). Получится следующая неявная разностная схема (10.16)–(10.18):

$$u_{i,0} = \varphi(x_i), \quad i = 0, 1, ..., N,$$
$$u_{0,j} = U_1, \quad U_{N,j} = U_2, \quad j = 0, 1, ..., K, \tag{10.16}$$

$$u_{i,j} = \frac{1}{1+2\gamma+\Delta h} u_{i,j-1} + \frac{\gamma}{1+2\gamma+\Delta h}(u_{i-1,j} + u_{i+1,j}) + \frac{\Delta h}{1+2\gamma+\Delta h} u_0, \quad i = 1, 2, ..., N-1; \quad j = 1, 2, ..., k, \tag{10.17}$$

$$\gamma = a^2 \frac{\Delta}{hx^2}. \tag{10.18}$$

Применение неявной разностной схемы для решения задачи 10.2 представлено в листинге 10.4.

Листинг 10.4. Функция neiav решения задачи 10.2 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 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 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 строки 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 её 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 методом Зейделя с точностью eps
  while r>eps
    //Вычисление матрицы ошибок R во внутренних точках
    //и пересчёт значений u во внутренних точках при решениии СЛАУ
    // 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
[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');
xlabel('X');
ylabel('T');
```

Входные и выходные данные функции `neiav` решения задачи 10.2 описаны в комментариях листинга 10.4. На рис. 10.7 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.

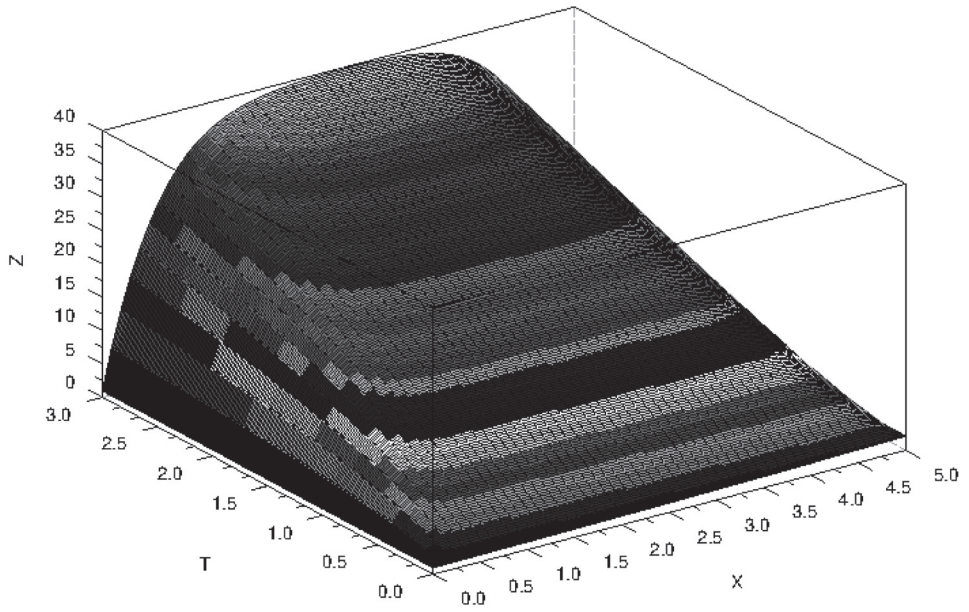

Рис. 10.7. График решения задачи 10.2

10.3 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:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \quad 0 \le x \le L, \quad 0 \le t \le T,$$
$$u(0,t) = \mu(t), \quad u(L,t) = \eta(t), \quad 0 \le t \le T, \tag{10.19}$$
$$u(x,0) = \varphi(x), \quad \frac{\partial u(x,0)}{\partial t} = \psi(x), \quad 0 \le x \le L.$$

Построим сетку Ω_h^{Δ} (см. рис. 10.1), в которой будем искать решение уравнения (10.19). Частную производную $\dfrac{\partial^2 u}{\partial x^2}$ заменим разностным соотношением (10.4), а производную $\dfrac{\partial^2 u}{\partial t^2}$ – соотношением (10.20) [8].

$$\frac{\partial^2 u(x_i,t_j)}{\partial t^2} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta^2}. \tag{10.20}$$

Подставляя (10.20), (10.4), (10.5) в гранично-начальную задачу (10.19), получим следующую явную разностную схему решения уравнения:

$$u_{i,j+1} = -u_{i,j-1} + \gamma(u_{i-1,j} + u_{i+1,j}) = (2 - 2\gamma)u_{i,j} + \Delta^2 f_{i,j},$$
$$i = 1, 2, ..., N-1, \quad j = 1, 2, ..., K-1,$$
$$u_{i,0} = \varphi(x_i), \quad \frac{u_{i,1} - u_{i,0}}{\Delta} = \Psi_i, \quad i = 0, 1, ..., N, \tag{10.21}$$
$$u_{0,j} = \mu_j, u_{N,j} = \nu_j, \quad j = 0, 1, ..., K,$$
$$\gamma = \frac{a^2\Delta^2}{h^2},$$

которая устойчива при $\gamma < 1$ и по аналогии с разностной схемой (10.9)–(10.10) может быть легко запрограммирована в Scilab.

В качестве примера рассмотрим следующую начально-граничную задачу.

Задача 10.3

Решить начально-граничную задачу

$$\frac{\partial^2\omega}{\partial t^2} = a^2\frac{\partial^2\omega}{\partial t^2} + \sin(xt), \quad 0 \le x \le L, \quad t \ge 0,$$
$$\omega(0, t) = \varphi(0), \quad \omega(L, t) = \varphi(L), \tag{10.22}$$
$$\omega(x, 0) = \varphi(x), \quad \omega_t(x, 0) = \psi(x).$$

В листинге 10.5 представлена функция `ggg` решения уравнения (10.22). Параметры функции `ggg` аналогичны рассмотренным ранее подпрограммам решения параболических уравнений.

Листинг 10.5. Функция `ggg` решения задачи 10.3 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 строки матрицы решений
  //U из граничных условий
  for j=1:K+1
    t(j)=(j-1)*delta;
  end
  //Формируем первую и последнюю строки 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
  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).

10.4 Использование метода сеток для решения эллиптических уравнений

Рассмотрим разностную схему для эллиптического уравнения в прямоугольной области $\Omega(R-b \le x \le R+b, -a \le y \le a)$ с граничными условиями Дирихле на границе Г.

Задача 10.4

$$\Delta u = \frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} - \frac{5}{x}\frac{\partial \Psi}{\partial x} = -2, \qquad (10.23)$$

$$\Psi_{(x,y)\in\Gamma} = 0.$$

Построим сетку Ω_{hx}^{hy}, для чего проведем в области Ω прямые, параллельные осям $y = y_j$ и $x = x_i$, где $x_i = R - b + i \cdot hx$, $hx = \frac{2b}{n}$, $i = 0, 1, 2, \ldots, Nx$; $y_j = -a + j \cdot hy$,

$hy = \frac{2a}{k}$, $j = 0, 1, ..., Ny$. Для построения разностного уравнения заменим частные производные и граничные условия следующими соотношениями:

$$\frac{\partial^2 \Psi(x_i, y_j)}{\partial x^2} = \frac{\Psi_{i-1,j} - 2\Psi_{i,j} + \Psi_{i+1,j}}{hx^2},$$
$$\frac{\partial^2 \Psi(x_i, y_j)}{\partial y^2} = \frac{\Psi_{i,j-1} - 2\Psi_{i,j} + \Psi_{i,j+1}}{hy^2}, \tag{10.24}$$

$$\Psi_{i,0} = \Psi_{i,Ny} = 0, \quad i = 0, 1, ..., Nx,$$
$$\Psi_{0,j} = \Psi_{Nx,j} = 0, \quad j = 0, 1, ..., Ny. \tag{10.25}$$

С помощью соотношений (10.24)–(10.25) преобразуем эллиптическую краевую задачу к следующей системе разностных уравнений:

$$\Psi_{i,j} = \frac{1}{A}(B_i \Psi_{i+1,j} + C_i \Psi_{i-1,j} + D(\Psi_{i,j-1} + \Psi_{i,j+1}) + 2),$$
$$A = \frac{2}{hx^2} + \frac{2}{hy^2}, \quad B_i = \frac{1}{hx^2} + \frac{5}{2hx\, x_i}, \quad C_i = \frac{1}{hx^2} - \frac{5}{2hx\, x_i}, \quad D = \frac{1}{hy^2},$$
$$i = 1, 2, ..., Nx - 1; \quad j = 1, 2, ..., Ny - 1, \tag{10.26}$$
$$\Psi_{i,0} = \Psi_{i,Ny} = 0, \quad i = 0, 1, ..., Nx,$$
$$\Psi_{0,j} = \Psi_{Nx,j} = 0, \quad i = 0, 1, ..., Ny.$$

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). В листинге 10.6 представлено решение уравнения (10.23) сеточным методом, а на рис. 10.8 – график найденного решения.

Листинг 10.6. Решение задачи 10.4

```
function [psi,x,y,k]=ellip(R,a,b,Nx,Ny,eps)
  //Функция ellip.
  //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 методом Зейделя.
  //Выходные данные:
  //psi - матрица решений в узлах сетки, массив x, массив y,
  //k - количество итераций при решении разностного уравнения
  // 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
  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 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');
```

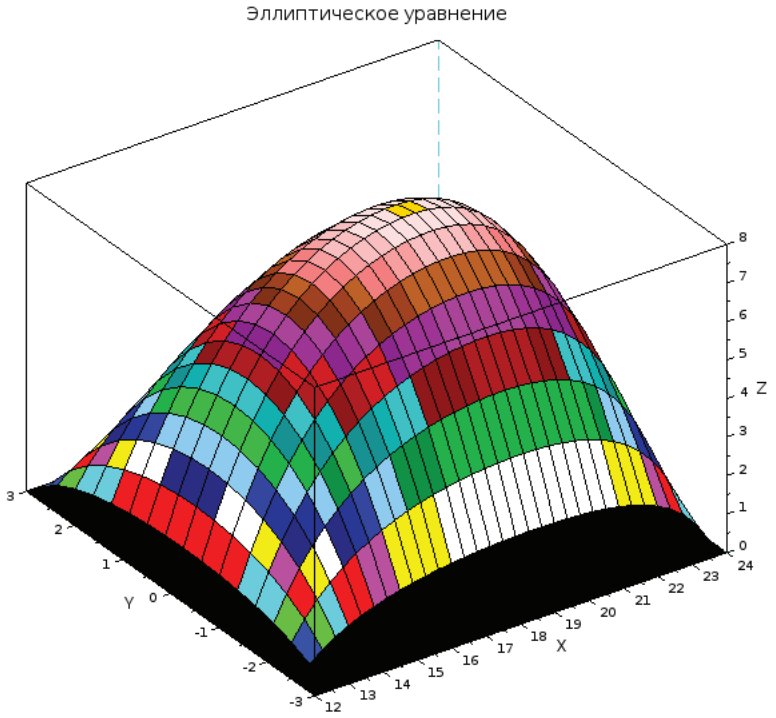

Рис. 10.8. График решения уравнения (10.23) сеточным методом

Авторы надеются, что читатель, разобравшийся с решением уравнения (10.23), 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 подобных случаях можно использовать метод конечных элементов, который реализован в кросс-платформенном свободно распространяемом пакете *freefem* [16].

Глава 11

Решение задач оптимизации

Эта глава посвящена решению оптимизационных задач средствами пакета Scilab. Будут рассмотрены:

- задачи поиска минимума функции;
- задачи линейного программирования;
- задачи квадратичного программирования.

11.1 Поиск минимума функции

В качестве простейшей оптимизационной задачи рассмотрим поиск локального минимума функции одной переменной.

Для нахождения значения минимума любой функции в Scilab служит функция [12]:

```
[fopt, xopt, [gopt, work, iters, evals, err]]
= optim(costf [,contr] ,x0 [,algo] [,df0 [,mem]] [,work] [,stop]
[,params] [,iprint=iflag]),
```

Входные параметры:

- `x0` – вектор-столбец начальных приближений длиной n;
- функция `costf` определяет функцию, минимум которой нужно найти (подробнее об этом ниже);
- `contr` – необязательный параметр, который определяется как `"b"`, `binf`, `bsup` и содержит ограничения на x: $\text{binf}_i \le x_i \le \text{bsup}_i$; если эти границы задаются, то `binf` и `bsup` являются векторами той же размерности, что и `x0`;
- `algo` – строка, с помощью которой задаётся алгоритм оптимизации (`"qn"` – квазиньютоновский, `"gc"` – сопряженных градиентов или `"nd"` – без дифференцирования) (по умолчанию `algo="qn"`);
- `df0` – скаляр, предположение об убывании целевой функции на первой итерации (по умолчанию `df0=1`);

- mem – целое число, которое определяет количество переменных, используемых для аппроксимации гессиана (Hessian); по умолчанию установлено значение, равное 10; опция доступна для безусловной оптимизации при значениях параметра algo "gc" или "nd";
- stop – совокупность переменных, контролирующих сходимость алгоритма, можно задать значения параметров nap, iter, epsg, epsf, epsx:
  - nap – максимальное число вызовов функции costf (по умолчанию 100);
  - iter – максимальное количество итераций (по умолчанию iter=100);
  - epsg – ограничение на норму градиента (по умолчанию epsg= %eps=2.220D-16);
  - epsf – пороговое значение, контролирующее убывание f (по умолчанию epsf=0);
- epsx – пороговое значение, контролирующее разброс x. Это вектор (возможно, матрица) той же размерности, что и x0. Может быть использован для масштабирования;
- params – в случае когда целевая функция является подпрограммой на C или Fortran, последовательность аргументов задаёт порядок обращения к целевой функции; параметр не имеет смысла, когда целевая функция задана в Scilab;
- "iprint=iflag" – аргумент задаёт режим трассировки, по умолчанию принимает значение 0: не печатается никаких дополнительных сообщений. В случае если значение аргумента не меньше единицы, на экран выводится больше информации; вид и количество информации определяются типом используемого алгоритма ("qn", "gc" или "nd").

Выходные результаты:

- fopt – минимальное значение функции;
- xopt – точка, в которой функция достигает этого значения xopt;
- gopt – значение градиента целевой функции в точке xopt;
- work – рабочий массив для работы квазиньютоновского метода оптимизации, который автоматически создается, когда происходит вызов функции optim; может быть задан как входной параметр функции для ускорения расчётов;
- iters – скаляр, определяющий число итераций, выводимых на печать, при значении iprint=2;
- evals – скаляр, который определяет число вызовов функции, выводимых на печать, при значении iprint=2;
- err – целое число, индикатор успешности решения задачи, может принимать значения от 1 до 10: err=1 – норма вычисленного градиента ниже допустимого; err=2 – на последней итерации f уменьшается; err=3 – оптимизация останавливается из-за слишком малых изменений для x; err=4 – остановка из-за превышения количества обращений к f; err=5 – остановка из-за превышения количества итераций; err=6 – остановка: слишком маленькие изменения в направлении градиента;

err=7 – остановка во время расчёта направления спуска; err=8 – остановка во время расчёта оценки матрицы Гессе; err=9 – окончание оптимизации, **задача решена**, успешное завершение; err=10 – успешное окончание оптимизации (линейный поиск не выполняется).

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, её градиент g.
//f - функция от вектора неизвестных x, минимум которой надо найти
f=gg(x);
//g - градиент функции f (вектор частной производной f по x),
g=numderivative(gg,x);
endfunction
```

Для *функции одной переменной* в качестве f возвращается *функция*, минимум которой надо найти, в качестве функции g – её *производная*.

Обратите внимание, что 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 определён в функции costf.

Таким образом, при использовании функции optim необходимо сформировать функцию costf, которая должна возвращать минимизируемую функцию f и её градиент (производную).

11.1.1 Поиск минимума функции одной переменной

Рассмотрим пример нахождения локального и глобального минимумов функции одной переменной.

Задача 11.1

Найти минимум функции $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$.

Решение задачи начнем с построения графика функции (см. листинг 11.1 и рис. 11.1).

Листинг 11.1. Построение графика функции

```
figure();
x=-5:0.1:3;
y=x.^4+3*x.^3-13*x.^2-6*x+26;
plot(x,y);
xgrid();
title('График функции f(x)=x^4+3*x^3-13*x^2-6*x+26','X','Y');
```

Как видно из графика (см. рис. 11.1), минимум функции достигается в районе $x_{min} \approx -4$. В листинге 11.2 представлено использование `optim` для поиска минимума функции одной переменной на примере $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$.

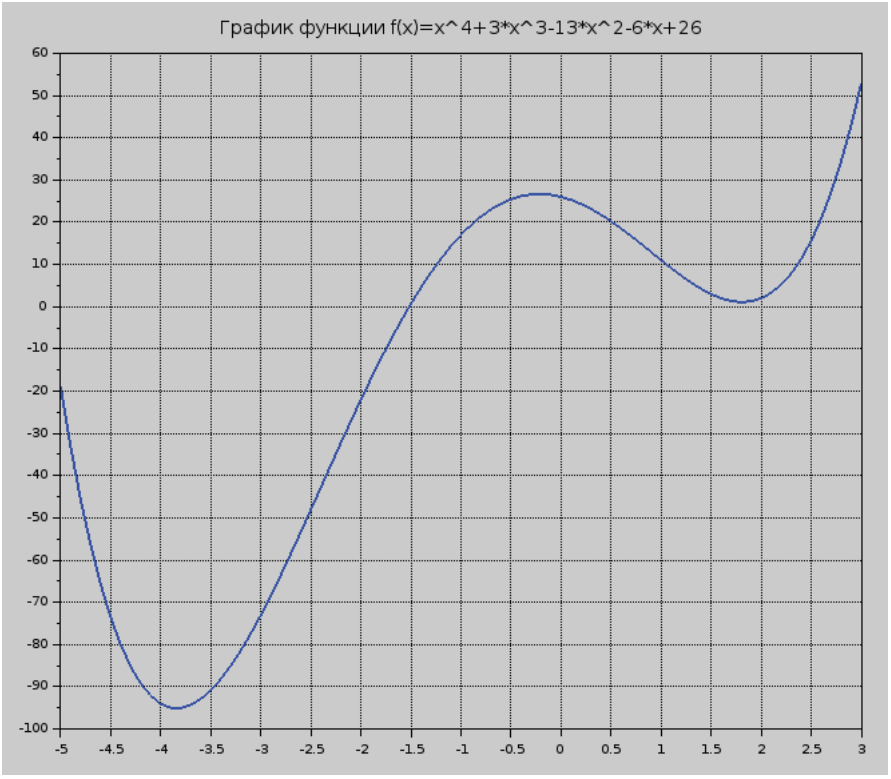

Рис. 11.1. График функции $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$

Листинг 11.2. Поиск минимума функции одной переменной

```
clear;
//Исходная функция
function y=ff(x)
y=x^4+3*x^3-13*x^2-6*x+26
endfunction
////Функция fi, в которой будет формироваться функция f и её
//производная g.
function [f,g,ind]=fi(x,ind)
//Функция f, минимум которой надо найти.
f=ff(x)
//Функция g - производная от функции f.
g=numderivative(ff,x)
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);
//Вывод результатов
mprintf("Глобальный минимум f(%10.6f)=%10.6f\n",xmin,fmin);
```

Ниже представлен результат поиска минимума функции одной переменной:

```
Глобальный минимум f( -3.840708)=-95.089413
```

Как видно по рис. 11.1, у функции $y = x^4 + 3x^3 - 13x^2 - 6x + 26$ есть локальный минимум на интервале [1; 3]. Программа решения этой задачи представлена в листинге 11.3.

Листинг 11.3. Поиск минимума функции одной переменной на заданном интервале

```
clear;
//Исходная функция
function y=ff(x)
y=x^4+3*x^3-13*x^2-6*x+26
endfunction
////Функция fi, в которой будут формироваться функция f и её
//производная g.
function [f,g,ind]=fi(x,ind)
//Функция f, минимум которой надо найти.
f=ff(x)
//Функция g - производная от функции f.
g=numderivative(ff,x)
endfunction
//Начальное приближение точки минимума.
y0=1.5;
//Для поиска точки локального минимума (xmin) и значения
// функции (fmin) на интервале [1;3] - вызов функции optim.
[fminlocal,xminlocal]=optim(fi,"b",1,3,y0);
//Вывод результатов
mprintf("Локальный минимум f(%10.6f)=%10.6f\n", xminlocal,fminlocal);
```

Результат представлен ниже.

```
Локальный минимум f(  1.806859)=  1.072528
```

Аналогично можно найти минимум любой другой функции одной переменной. Главной проблемой является правильный выбор точки начального приближения. Но это проблема не пакета Scilab, а отдельная математическая задача.

11.1.2 Поиск минимума функции многих переменных

При нахождении минимума функции многих переменных функцию `costf` необходимо построить таким образом, чтобы входными данными в неё 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 надо найти
f=gg(x);
//g - градиент функции f (вектор частной производной)
g=numderivative(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. 11.2.

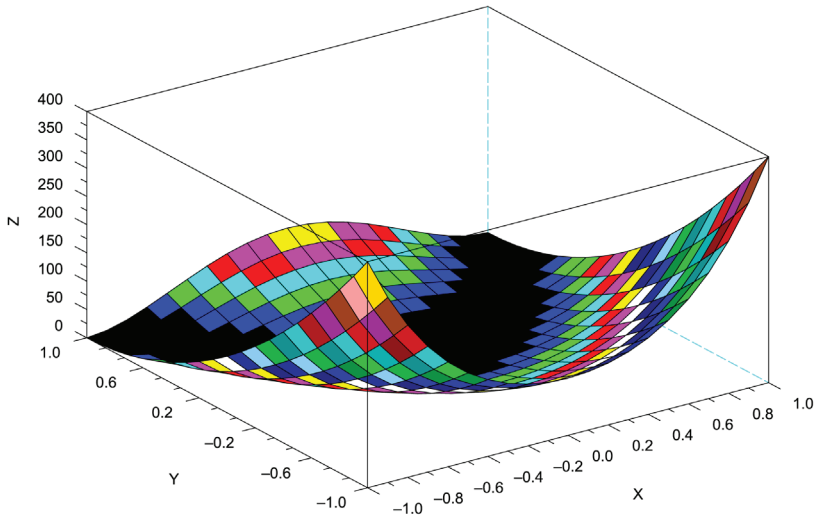

Рис. 11.2. График функции Розенброка

Как известно, функция Розенброка имеет минимум в точке (1, 1), равный 0. Ввиду своей специфики функция Розенброка является хорошим тестом для алгоритмов минимизации. Найдем минимум этой функции с помощью функции optim (листинг 11.4).

Листинг 11.4. Поиск минимума функции Розенброка

```
//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 её градиент.
function [f,g,ind]=cst(x,ind)
f=gg(x);
g=numderivative(gg,x);
endfunction
//Обратите внимание, итерационный процесс
//сходится и при достаточно плохом начальном приближении
x0=[4,-5];
//Вызов функции optim
[fopt,xopt]=optim(cst,x0)
//Вывод результатов
mprintf("f(%f,%f)=%f",xopt(1),xopt(2),fopt)
```

Ниже представлен результат поиска минимума функции Розенброка с помощью функции `optim`.

```
f(1.000000,1.000000)=0.000000
```

11.2 Решение задач линейного программирования

Ещё одной часто встречающейся в практике оптимизационной задачей является задача линейного программирования. Знакомство с задачами линейного программирования начнем на примере задачи об оптимальном рационе [2].

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. Причём 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. табл. 11.1).

Таблица 11.1. 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}

Требуется составить пищевой рацион, чтобы обеспечить заданные условия при минимальной стоимости.

Пусть x_1, x_2, x_3, x_4 – количество продуктов $П_1$, $П_2$, $П_3$, $П_4$. Общая стоимость рациона равна

$$L = c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 = \sum_{i=1}^{4} c_i x_i. \qquad (11.1)$$

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 белков и т. д. Следовательно, общее количество белков во всех четырёх типах продукта равно $\sum_{j=1}^{4} a_{j1}x_j$ и должно быть не меньше b_1. Получаем первое ограничение:

$$a_{11}x_1 + a_{21}x_2 + a_{31}x_3 + a_{41}x_4 \geq b_1. \tag{11.2}$$

Аналогичные ограничения для жиров и углеводов имеют вид:

$$a_{12}x_1 + a_{22}x_2 + a_{32}x_3 + a_{42}x_4 \geq b_2, \tag{11.3}$$

$$a_{13}x_1 + a_{23}x_2 + a_{33}x_3 + a_{43}x_4 \geq b_3. \tag{11.4}$$

Приняв во внимание, что x_1, x_2, x_3, x_4 – положительные значения, получим ещё четыре ограничения:

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0. \tag{11.5}$$

Таким образом, задачу об оптимальном рационе можно сформулировать следующим образом: найти значения переменных x_1, x_2, x_3, x_4, удовлетворяющие системе ограничений (11.2)–(11.5), при которых линейная функция (11.1) принимала бы минимальное значение.

Задача об оптимальном рационе является задачей линейного программирования, функция (11.1) называется функцией цели, а ограничения (11.2)–(11.5) – 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, ..., x_n, удовлетворяющие системе ограничений (11.6), при которых функция цели L (11.7) достигает своего минимального (максимального) значения:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i = 1, ..., m, \quad x_i \geq 0, \tag{11.6}$$

$$L = c_1x_1 + c_2x_2 + ... + c_nx_n = \sum_{i=1}^{n} c_ix_i. \tag{11.7}$$

Для решения задачи линейного программирования в Scilab сформулируем её в матричном виде следующим образом.

Найти вектор x, удовлетворяющий системе ограничений (11.9)–(11.11), при которых функция цели L (11.8) достигает своего минимального значения.

$$L = c^Tx, \tag{11.8}$$

$$Ax \le b, \tag{11.9}$$

$$A_{eq}x = b_{eq}, \tag{11.10}$$

$$l_b \le x \le u_b. \tag{11.11}$$

Решение задач линейного программирования в Scilab (начиная с версии 5, в версии 6 эта функция стала основной) осуществляется с помощью функции `karmarkar` следующей структуры [12]:

```
[xopt, fopt] = karmarkar (Aeq, beq, c, x0, rtolf, gam, maxiter, outfun,
A, b, lb, ub),
```

Здесь:

- `Aeq`, `beq` – матрица и вектор-столбец определяют ограничение типа равенства (11.10);
- `c` – массив (вектор-столбец) коэффициентов при неизвестных функции целях `L` (11.8), длина вектора `n` совпадает с количеством неизвестных `x`;
- `rtolf` – точность вычисления экстремума (по умолчанию 10^{-5});
- `x0` – вектор-столбец начальных приближений; если вы будете задавать реальное значение столбца начальных приближений, то оно должно удовлетворять системе ограничений решаемой задачи линейного программирования;
- `gam` – масштабный коэффициент в итерационном алгоритме оптимизации (по умолчанию `gam=0.5`);
- `maxiter` – максимальное количество итераций (по умолчанию `maxiter=200`);
- `outfun` – пользовательская функция вывода результатов, пример её использования приведён на странице справки https://help.scilab.org/docs/2023.1.0/ru_RU/karmarkar.html;
- `A` – матрица при неизвестных из левой части системы ограничений (11.9);
- `b` – массив (вектор-столбец), содержит свободные члены системы ограничений (11.9);
- `lb` – массив (вектор-столбец), содержит нижнюю (левую) границу векторной переменной `x` (11.11);
- `ub` – массив (вектор-столбец), содержит верхнюю (правую) границу переменных `x` (11.11);
- функция `karmarkar` возвращает массив неизвестных xopt, минимальное значение функции `fopt`.

Если любой (за исключением последнего) параметр отсутствует, то на его месте указывают [ ].

Рассмотрим использование функции `karmarkar` на примере решения следующей задачи линейного программирования [2].

Задача 11.2

Найти такие значения переменных x_1, x_2, x_3, x_4, при которых функция цели L достигает своего минимального значения и удовлетворяются ограничения:

$$L = -x_2 - 2x_3 + x_4,$$

$$3x_1 - x_2 \le 2,$$

$$x_2 - 2x_3 \le -1,$$

$$4x_3 - x_4 \le 3,$$

$$5x_1 + x_4 \ge 6,$$

$$x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0.$$

Обратите внимание, что в четвертом ограничении присутствует знак «$\ge$». Для приведения системы ограничений к виду (11.9) необходимо четвертое уравнение умножить на -1. Решение задачи 11.2 представлено в листинге 11.5.

Листинг 11.5. Решение задачи 11.2

```
//Коэффициенты с функции цели L.
c=[0;-1;-2;1]
//Ограничение вида Ax<=b
A=[3 -1 0 0;0 1 -2 0; 0 0 4 -1; -5 0 0 -1];
b=[2;-1;3;-6];
//Вызов функции karmarkar
//Обратите внимание на использование отсутствующих параметров
//и параметров по умолчанию в функции karmarkar!!!
[xopt,fopt] =karmarkar([],[],c,[],0.0001,[],[],[],A,b)
for i=1:length(c)
mprintf("x(%d)=%6.2f\n",i,xopt(i));
end
mprintf("Функция цели f=%6.2f\n",fopt);
```

Полученные значения представлены в листинге 11.6.

Листинг 11.6. Результаты решения задачи 11.2

```
x(1)=  0.98
x(2)=  1.32
x(3)=  1.16
x(4)=  1.64
Функция цели f= -2.00
```

Обратите внимание, что рассмотренная нами задача имеет не один экстремум.

Рассмотрим решение ещё одной задачи линейного программирования [2, 3].

Задача 11.3

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 как можно 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 указана в табл. 11.2.

Таблица 11.2

Турбаза	Дельфинарий	Ботанический сад	Поход в горы
1	5	6	20
2	10	12	5

Для решения задачи введём 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:

$$x_1 + x_4 \le 70\ ;$$
$$x_2 + x_5 \le 180;$$
$$x_3 + x_6 \le 110;$$
$$x_1 + x_2 + x_3 = 200;$$
$$x_4 + x_5 + x_6 = 150\ .$$

Кроме того, количество туристов, участвующих в мероприятиях, не может быть отрицательным: $x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0, x_5 \ge 0, x_6 \ge 0$.

Решение задачи представлено в листинге 11.7.

Листинг 11.7. Решение задачи 11.3

```
clear;
//Коэффициенты функции цели, см. табл.
```

```
c=[5;6;20;10;12;5]
//Ограничения вида Aeq x= beq,
//связанные с количеством туристов на турбазах.
Aeq=[1 1 1 0 0 0; 0 0 0 1 1 1]
beq=[200;150]
//Ограничения вида A x <= b,
//связанные с пропускной способностью дельфинария,
//ботанического сада и количеством туристов, идущих в горы.
A=[1 0 0 1 0 0;0 1 0 0 1 0; 0 0 1  0 0 1]
b=[70;180;110];
[xopt,fopt] =karmarkar(Aeq, beq, c, [],  0.000001, [], [], [],...
 A, b,[0;0;0;0;0;0])
for i=1:length(c)
mprintf("x(%d)=%6.2f\n",i,xopt(i));
end
mprintf("Функция цели f=%6.2f\n",fopt);
```

Полученное в Scilab решение задачи представлено в листинге 11.8.

Листинг 11.8. Результаты решения задачи 11.3

```
x(1)= 30.00
x(2)=170.00
x(3)=  0.00
x(4)= 40.00
x(5)=  0.00
x(6)=110.00
Функция цели f=2120.00
```

11.3 Решение задач квадратичного программирования

В задачах квадратичного программирования функция цели представляет из себя квадратичную функцию нескольких переменных, а ограничения по-прежнему линейны.

В общем случае в задаче квадратичного программирования функция цели имеет вид:

$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij} \cdot x_i \cdot x_j + \sum_{i=1}^{n} p_i \cdot x_i. \tag{11.12}$$

Ограничения:

$$Ax \le b, \tag{11.13}$$

$$Dx = c, \tag{11.14}$$

$$e_i \le x_i \le g_i. \tag{11.15}$$

Перепишем функцию цели в виде квадратичной формы:

$$f(x) = \frac{1}{2} \cdot x^T \cdot Q \cdot x + p^T \cdot x. \tag{11.16}$$

Здесь Q – положительно определённая симметричная матрица

$$Q = \begin{pmatrix} 2q_{11} & q_{12} & \dots & q_{1n} \\ q_{21} & 2q_{22} & \dots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \dots & 2q_{nn} \end{pmatrix}.$$

Задачу можно сформулировать так: найти вектор x, который доставляет минимум функционалу (11.16), при этом выполняются ограничения (11.13)–(11.15).

Для решения задачи квадратичного программирования в Scilab предназначена функция `qld` следующей структуры:

```
x  = qld(Q, p, E, z, ei, gi, m [,tol])
```

Аргументами функции `qld` являются [12]:

- `Q` – положительно определённая симметричная матрица размерности $n \times n$;
- `p` – вектор-столбец размерности n;
- `E` – матрица размерности $(m+k) \times n$ (`E=[D;A]`);
- `z` – вектор-столбец размерности $m + k$ (`z=[c;b]`);
- `ei` – вектор-столбец размерности n. Ограничение снизу на переменную `xi` (`ei≤xi`);
- `gi` – вектор-столбец верхних границ на переменные `xi` (`xi≤gi`);
- `m` – число ограничений типа равенств;
- `tol` – требуемая точность вычислений;
- `x` – найденное оптимальное решение.

Более подробное описание `qld` можно прочитать в справке. Рассмотрим пример решения задачи квадратичного программирования.

Задача 11.4

Решить задачу квадратичного программирования

$$f(x) = x_1^2 + 2x_2^2 - 2x_1x_2 - 2x_1 - 6x_2 \to \min,$$
$$x_1 + x_2 \le 2,$$
$$-x_1 + 2x_2 \le 2,$$
$$x_1 \ge 0, \quad x_2 \ge 0.$$

Необходимые для формулировки задачи квадратичного программирования (11.16), (11.13)–(11.15) матрицы Q, p, A, b имеют вид:

$$Q = \begin{pmatrix} 2 & -2 \\ -2 & 4 \end{pmatrix}, \quad p = \begin{pmatrix} -2 \\ -6 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 1 \\ -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}.$$

Матрица D и вектор c имеют нулевую размерность, значение параметра m равно 0.

Решение задачи 11.4 представлено в листинге 11.9.

Листинг 11.9. Решение задачи 11.4

```
//функция цели f=1/2 x' Q x +p'x
Q=[2 -2;-2 4];
p=[-2 ;-6];
//Ограничение Dx=c
D=[];
c=[];
//Ограничение Ax<=b
A=[1 1; -1 2];
b=[2;2];
//Ограничение x>=ei
ei=[0;0];
//Ограничение x<=gi
gi=[];
//Формируем матрицы E и z, см. описание функции qld
E=[D;A];
z=[c;b]
//Количество (m) ограничений типа равенство Dx=c
m=0;
//Решение задачи квадратичного программирования
x=qld(Q,p,E,z,ei,gi,m)
//Вычисление значения функции цели
f=0.5*x'*Q*x+p'*x
//Вывод результатов
for i=1:length(x)
mprintf("x(%d)=%6.2f\n",i,x(i));
end
mprintf("Значение функционала f=%6.2f\n",f);
```

Полученное в Scilab решение задачи представлено в листинге 11.10.

Листинг 11.10. Результаты решения задачи 11.4

```
x(1)=  0.80
x(2)=  1.20
Значение функционала f= -7.20
```

Рассмотрим ещё одну задачу квадратичного программирования.

Задача 11.5

Решить задачу квадратичного программирования

$$f(x) = x_1^2 + x_2^2 + x_3^2 \to \min,$$
$$2x_1 - x_2 + x_3 \le 5,$$
$$x_1 + x_2 + x_3 = 3.$$

Необходимые для формулировки задачи квадратичного программирования (11.16), (11.13)–(11.15) матрицы Q, p, D, c, A, b имеют вид:

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \quad p = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},$$

$$D = (1 \quad 1 \quad 1), c = 3,$$

$$A = (2 \quad -1 \quad 1), b = 5.$$

Значение параметра m равно 1.

Решение задачи 11.5 представлено в листинге 11.11.

Листинг 11.11. Решение задачи 11.5

```
//функция цели f=1/2 x' Q x +p'x
Q=[2 0 0;0 2 0; 0 0 2];
p=[0 ; 0; 0];
//Ограничение Dx=c
D=[1 1 1]; c=[3];
//Ограничение Ax<=b
A=[2 -1 1]; b=[5];
//Ограничение x>=ei
ei=[];
//Ограничение x<=gi
gi=[];
//Формируем матрицы E и z, см. описание функции qld
E=[D;A]; z=[c;b]
//Количество (m) ограничений типа равенство Dx=c
m=1;
//Решение задачи квадратичного программирования
x=qld(Q,p,E,z,ei,gi,m)
//Вычисление значения функции цели
f=0.5*x'*Q*x+p'*x
//Вывод результатов
for i=1:length(x)
mprintf("x(%d)=%6.2f\n",i,x(i));
end
mprintf("Значение функционала f=%6.2f\n",f);
```

Полученное в Scilab решение задачи представлено в листинге 11.12.

Листинг 11.12. Результаты решения задачи 11.5

```
x(1)=  1.00
x(2)=  1.00
x(3)=  1.00
Значение функционала f=  3.00
```

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 12

Использование Scilab для создания интерактивных документов

Эта глава будет посвящена возможностям Scilab для создания интерактивных документов, среди которых можно выделить электронные учебные пособия и интерактивные научные статьи.

12.1 Инструментальные средства разработки интерактивных документов

Долгое время одним из стандартов для формирования интерактивных математических документов являлся формат MathCAD. К его минусам можно отнести закрытый формат, смену структуры файла от версии к версии, высокую стоимость программы, работу только в одной проприетарной операционной системе, отсутствие выбора модуля интерактивной расчётной части. Выход версий MathCAD Prime решил проблему оформления документов, однако остались проблемы закрытого формата, высокой стоимости и поддержки работы только под управлением MS Windows.

Альтернативным инструментом является Jupyter Notebook, который позволяет создать документ, в котором могут быть блоки структурированного текста в формате `markdown` с формулами, таблицами, рисунками, гиперссылками; блоки текста в формате `TeX`, а также интерактивные вычислительные зоны. Jupyter Notebook традиционно использовался как интерактивная среда разработки языка программирования Python. Однако возможности Jupyter Notebook намного шире. В интерактивных вычислительных зонах документа можно использовать Python3, Julia, Scilab, Octave, Sage, Maxima, Wolfram Language и др.

Jupyter Notebook является полноценной кросс-платформенной IDE, позволяющей осуществлять вывод графиков, таблиц, формул как результата

работы интерактивного вычислительного блока. Jupyter Notebook имеет веб-интерфейс и открывается в браузере по умолчанию.

Здесь мы рассмотрим возможности совместного использования Jupyter Notebook и Scilab для создания интерактивных документов и научных статей.

12.2 Установка Jupyter Notebook

Для установки среды разработки Jupyter Notebook необходимо установить пакет *pip*. В AltLinux пакет называется *python3-module-pip*. Его установка может быть осуществлена командой

```
# apt-get install spyder python3-module-notebook
```

В Debian и его клонах пакет называется *python3-pip*, для его установки необходимо использовать команду

```
# apt install python3-pip
```

Установка Jupyter Notebook осуществляется с помощью Python'овской утилиты *pip*.

```
pip3 install notebook
```

Для организации взаимодействия между Jupyter Notebook и вычислительным ядром Scilab необходимо с помощью *pip3* установить модуль *scilab_kernel*:

```
pip3 install scilab_kernel
```

Теперь всё готово к использованию Jupyter Notebook с вычислительным ядром Scilab для создания электронных документов.

Для запуска программы Jupyter Notebook можно в терминале ввести команду `jupyter-notebook`. На рис. 12.1 можно увидеть внешний вид современного окна Jupyter Notebook после старта. Для создания интерактивного документа с вычислительным ядром Scilab выполним следующее:

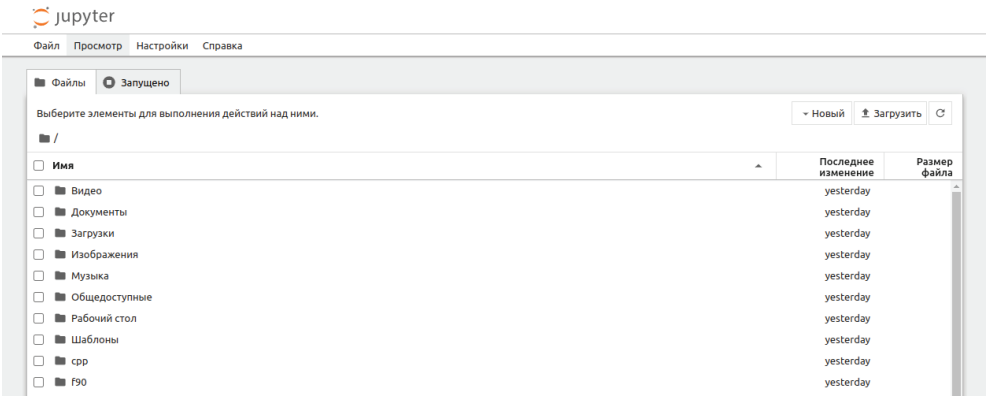

Рис. 12.1. Фрагмент окна Jupyter Notebook

- щёлкнем по кнопке **Новый**;
- выберем **Блокнот**;
- в открывшемся окне (рис. 12.2) выберем в качестве вычислительного ядра Scilab.

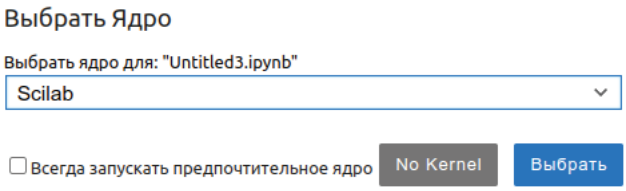

Рис. 12.2. Выбор вычислительного ядра блокнота

12.3 Создание документов с помощью Jupyter Notebook и Scilab

После всех этих манипуляций в браузере откроется Jupyter Notebook с пустым блокнотом, где в качестве вычислительной среды будет использоваться Scilab.

Сохраним его с помощью команды **Файл** ⇒ **Сохранить Notebook как...**. Сохраняемый файл блокнота имеет расширение *.ipunb*. По умолчанию файл сохраняется в домашнем каталоге пользователя. Если вы хотите сохранить его в другой каталог, то при сохранении необходимо указать не только имя файла, но и путь, начиная от домашнего каталога. Например, *scilab_ipunb/Scilab1.ipunb*. Окно пустого блокнота с вычислительной средой Scilab представлено на рис. 12.3[1].

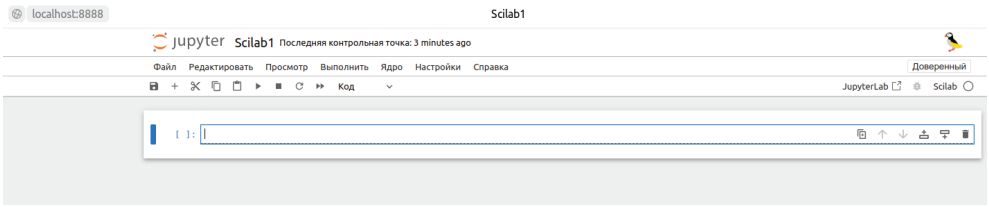

Рис. 12.3. Jupyter Notebook с пустым блокнотом

Электронный документ в формате Jupyter Notebook представляет собой набор входных и выходных ячеек. В качестве входных ячеек могут быть текстовые ячейки в различных форматах (markdown, LaTeX, html и др.) и вычислительные зоны. Выходная ячейка формируется после нажатия комбинации

[1] Для русификации интерфейса необходимо скачать файл локализации командой `pip3 install jupyterlab-language-pack-ru-RU`. После этого в меню **Setting** ⇒ **Language** выбрать **Русский**.

клавиш **Shift**+**Enter** по окончании ввода данных во входную ячейку. После этого в текствой ячейке формируется отформатированный текст с формулами, таблицами, рисунками и т. д. После нажатия **Shift**+**Enter** в вычислительной зоне происходят расчёты и формируется выходная ячейка с результатами.

В текстовых ячейках можно вводить описание математической модели, текст статьи, электронного конспекта, учебного пособия с формулами, рисунками, таблицами, гиперссылками и т. д. Текстовые ячейки выполняют роль текстового процессора. В этот же документ можно включать вычислительные зоны – это по существу фрагменты программ, которые придают документу интерактивность. Читатель (пользователь) такого документа может не только познакомиться с теоретическими положениями, но и самостоятельно провести расчёты, при необходимости поменять исходные данные и даже код программы[1].

Рассмотрим более подробно особенности работы с ячейками. Пользователь вводит данные в ячейку и нажимает **Shift**+**Enter**. После этого можно в любой момент вернуться к ячейке и отредактировать её[2]. В режиме редактирования ячейки, кроме изменения её содержимого, справа доступно 6 кнопок управления (рис. 12.4):

- кнопка дублирования ячейки;
- кнопка переноса ячейки вверх по документу;
- кнопка переноса ячейки вниз по документу;
- кнопка вставки новой ячейки выше;
- кнопка вставки новой ячейки ниже;
- кнопка удаления ячейки.

Рис. 12.4. Ячейка с кнопками управления

Тип информации, которая хранится в ячейке ввода, определяется с помощью кнопки панели инструментов (см. рис. 12.3, 12.5).

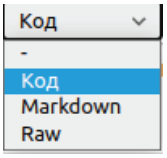

Рис. 12.5. Выбор типа входной ячейки

[1] Здесь и далее речь будет идти только о документах в формате Jupyter Notebook с вычислительными ячейками в формате Scilab.

[2] После редактирования ячейки не забывайте нажимать **Shift**+**Enter**.

Управление ячейками и их перезапуском осуществляется с помощью пункта меню **Выполнить**. Рассмотренной в этом параграфе информации достаточно, чтобы создавать интерактивные документы в формате Jupyter Notebook с вычислительным ядром Scilab. Переходим к решению практических задач.

12.4 Решение практических задач с помощью Jupyter Notebook и Scilab

Давайте создадим интерактивный документ, который предназначен для подбора параметров кривой методом наименьших квадратов, в качестве базовой выберем задачу 12.1.

Задача 12.1

В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P, Вт) от входного напряжения (U, В) для асинхронного двигателя (табл. 12.1). Методом наименьших квадратов подобрать аналитическую зависимость $P = f(U)$.

Таблица 12.1. Зависимость 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

На первом этапе построим зависимость вида $P = a_1 + a_2U + a_3U^2 + a_4U^3$. Затем попробуем обобщить алгоритм построения зависимости любого вида.

Запустим Jupyter Notebbok, создадим новый документ, выбрав в качестве вычислительного ядра Scilab.

Тип первой ячейки определим как `markdown`. Введём в неё следующий текст.

```
ЗАДАЧА. В результате опыта холостого хода определена зависимость потребляемой
из сети мощности (P,Вт) от входного напряжения (U,B) для асинхронного двигателя
(см.табл).

|U,B|132|140|150|162|170|180|190|200|211|220|232|240|251|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|P,кВТ|330|350|385|425|450|485|540|600|660|730|920|1020|1350|

Методом наименьших квадратов подобрать полином

$$ P=a_1 +a_2 U + a_3 U^2 + a_4 U^3 $$.
```

После нажатия клавиш **Shift**+**Enter** окно Jupyter Notebook примет следующий вид (см. рис. 12.6). Тем самым была создана ячейка, в которую ввели постановку задачи в формате `markdown`. Jupyter NoteBook отформатировал текст.

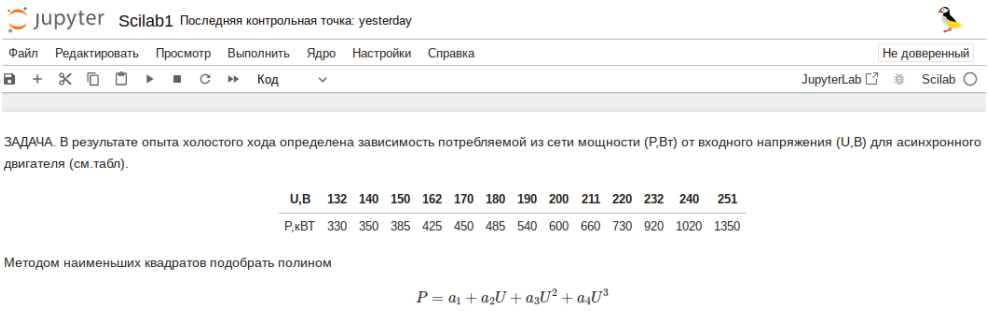

Рис. 12.6. Документ Jupyter Notebook с постановкой задачи

Подобным образом можно описывать фрагменты математических моделей, писать электронные конспекты и книги различной тематики. Добавим во вторую ячейку код решения задачи.

Листинг 12.1. Решение задачи 12.1

```
//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)\n
        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]
n=size(x)(2);
//Формирование матрицы исходных данных
z=[x;y];
//Вектор начальных приближений
c=[0;0;0;0];
//Решение задачи
[a,err]=datafit(G,z,c);
//Вывод результатов
mprintf(\"Суммарная квадратичная ошибка=%6.2f\\n\",err)
mprintf(\"Подобранные коэффициенты полинома \")
for i=1:4
    mprintf(\"%6.4f \",a(i))
end\n
mprintf(\"\\n\")
//Построение графика экспериментальных данных
//Построение графика подобранной функции
hx=(x(n)-x(1))/100;
t=x(1):hx:x(n);
Ptc=a(1)+a(2)*t+a(3)*t^2+a(4)*t^3;
```

```
plot(x,y,'ro',t,Ptc,'b-');
xgrid();
xtitle('Зависимость P(U)', 'U, В','P, Вт')
```

Результаты работы программы из листинга 12.1 представлены на рис. 12.7.

Таким образом был создан простейший интерактивный документ, который состоит из трёх ячеек:

1) входная ячейка – текст в формате `markdown`;
2) входная ячейка – код на языке Scilab;
3) выходная ячейка – результат работы программы на языке Scilab.

```
Суммарная квадратичная ошибка=  0.53

Подобранные коэффициенты полинома
-51.5766 95.5946 -55.6953 11.1114
```

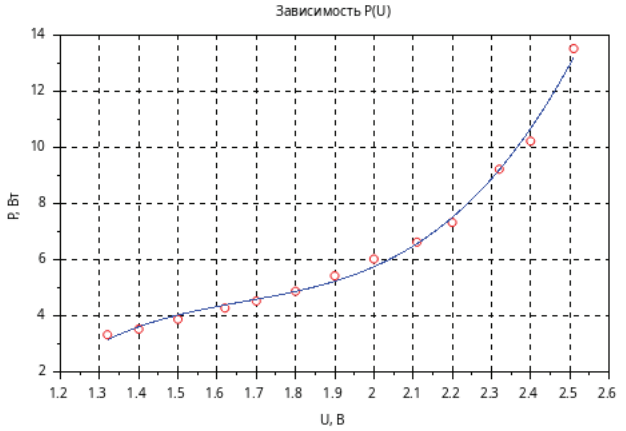

Рис. 12.7. Подбор параметров кривой методом наименьших квадратов

Пользователь может проверить работоспособность кода на других исходных данных, подобрать другую кривую методом наименьших квадратов, изменить постановку задачи в текстовой ячейке.

Очевидно, решение рассмотренной задачи не составляет особой сложности и было показано только с целью демонстрации возможностей Jupyter Notebook и Scilab для создания интерактивного документа. Более сложной и интересной авторам показалась следующая задача: создать интерактивный документ, в нем описать алгоритм решения и решить уравнение в частных производных.

Но тут перед авторами встала задача представления интерактивного документа большого объёма в бумажной книге. Довольно долго мы размышляли

и решили оставить ссылку на интерактивный документ, в котором приведено решение задачи.

Задача 12.2

Решить параболическое уравнение, описывающее распределение температуры в стержне длиной L, начальная температура стержня задаётся произвольной функцией $\varphi(x)$. Температуры концов стержня равны $u(0, t) = U_1 = \text{const}$, $u(L, t) = U_2 = \text{const}$.

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \quad a^2 = \frac{\lambda}{c\rho}, \quad 0 \le x \le L, \quad 0 \le t < \infty,$$

$$u(0, t) = U_1 = \text{const}, \quad u(L, t) = U_2 = \text{const}, \quad 0 \le t < \infty, \tag{12.1}$$

$$u(x, 0) = \varphi(x), \quad 0 \le x \le L.$$

Здесь a – коэффициент температуропроводности, λ – коэффициент теплопроводности материала стержня, c – удельная теплоёмкость, ρ – плотность.

Интерактивный документ, в котором описан алгоритм решения и приведено решение задачи 12.2, можно скачать по адресу https://disk.yandex.ru/d/NlNQsuhfkL1kNA.

Обращаем внимание читателя, что вычисления в среде Jupyter Notebook с вычислительным ядром Scilab проходят значительно медленнее, чем в Scilab. Поэтому авторы не рекомендуют использовать интерактивные блокноты с ядром Scilab для решения задач с большим количеством вычислений.

Глава 13

Задания для самостоятельной работы в Scilab

13.1 Программирование в Scilab

13.1.1 Программирование циклических вычислительных процессов в Scilab

Написать программы на языке Scilab. Программы из задания номер 3 должны работать и при достаточно большом значении n, полученный результат проверить с помощью встроенных функций Scilab. Проверьте работоспособность разработанной программы из задания № 5 (варианты 1–3, 6, 8, 10, 12–15) при $|x| > 1$. Полученные в задании № 5 результаты также проверьте с помощью встроенных функций Scilab.

Вариант 1

Задание 1. Вывести на экран первые 100 чисел-палиндромов > 12.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода 13 простых чисел. Для каждого введённого числа вывести на экран число, которое получится после записи цифр исходного числа в обратном порядке.

Задание 3. Найти сумму первых n ($100 \leq n \leq 1000$) натуральных чисел, кратных 5.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = x - \frac{x^3}{3!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

и функции $Y(x) = \sin(x)$ в диапазоне от 0 до 1 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$

и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 2

Задание 1. Вывести на экран первые 125 чисел-палиндромов > 100.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Найти наименьшее из введённых чисел, у которого сумма делителей нечётна.

Задание 3. Найти произведение первых n простых чисел. Программа должна работать при $n = 15$.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 + \frac{x^2}{2!} + \ldots + \frac{x^{2n}}{(2n)!}$$

и функции

$$Y(x) = \frac{e^x + e^{-x}}{2}$$

в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 + \frac{x \ln a}{1!} + \frac{(x \ln a)^2}{2!} + \ldots + \frac{(x \ln a)^n}{n!}.$$

Натуральное значение n введите с клавиатуры. Значения x ($|x| < 1$) и a также вводятся с клавиатуры.

Вариант 3

Задание 1. Вывести на экран последние 10 простых чисел в диапазоне от 1000 до 10 000.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Найти два наименьших числа из введённых, у которых сумма делителей нечётна.

Задание 3. Найти сумму первых n ($100 \leq n \leq 1000$) палиндромов.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 + \frac{\cos\left(\frac{\pi}{4}\right)}{1!}x + \ldots + \frac{\cos\left(n\frac{\pi}{4}\right)}{n!}x^n$$

и функции

$$Y(x) = e^{x\cos\left(\frac{\pi}{4}\right)} \cdot \cos\left(x \cdot \sin\frac{\pi}{4}\right)$$

в диапазоне от 0 до 1 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n+1}\frac{x^n}{n}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 4

Задание 1. Вывести на экран первые 100 простых чисел > 25.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Найти два наибольших числа из введённых, у которых сумма делителей чётна.

Задание 3. Найти среднее арифметическое первых n ($100 \leq n \leq 1000$) натуральных чисел, кратных 13, не кратных 2, 3, 5, 7.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 - \frac{x^2}{2!} + \ldots + (-1)^n\frac{x^{2n}}{(2n)!}$$

и функции $Y(x) = \cos(x)$ в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = \cfrac{1}{2 + \cfrac{1}{4 + \cfrac{1}{6 + \ldots + \cfrac{1}{2n}}}}.$$

Натуральное значение n введите с клавиатуры.

Вариант 5

Задание 1. Вывести на экран последние n простых чисел в диапазоне от a до b.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Введённые числа вывести в двоичной системе счисления.

Задание 3. Найти квадрат суммы первых n ($100 \leq n \leq 1000$) простых чисел.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 + 3x^2 + \ldots + \frac{2n+1}{n!}x^{2n}$$

и функции $Y(x) = (1 + 2x^2)e^{x^2}$ в диапазоне от 0 до 1 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 + \frac{1}{3 + \frac{1}{5 + \frac{1}{7 + \ldots + \frac{1}{2n+1}}}}.$$

Натуральное значение n введите с клавиатуры.

Вариант 6

Задание 1. Вывести на экран первые n простых чисел в диапазоне от a до b.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Введённые числа вывести в восьмеричной системе счисления.

Задание 3. Найти корень квадратный из суммы первых n ($100 \leq n \leq 1000$) чисел-палиндромов.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 + \frac{x^3}{3!} + \ldots + \frac{x^{2n-1}}{(2n-1)!}$$

и функции

$$Y(x) = \frac{e^x - e^{-x}}{2}$$

в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = -\frac{x^2}{1 \cdot 2} + \frac{x^3}{2 \cdot 4} - \frac{x^4}{3 \cdot 8} + \ldots + (-1)^n \frac{x^{n+1}}{n2^n}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 7

Задание 1. Вывести на экран последние n чисел-палиндромов в диапазоне от a до b.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Вывести номер наибольшей цифры в каждом введённом числе.

Задание 3. Найти сумму первых n ($100 \leq n \leq 1000$) натуральных чисел, кратных 13 и 22.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 - \frac{1}{2} \cdot x + \frac{1 \cdot 3}{2 \cdot 4} \cdot x^2 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \cdot x^3 + \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8} \cdot x^4 - \ldots$$

и функции

$$Y(x) = \frac{1}{\sqrt{x+1}}$$

в диапазоне от 0 до 0.95 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = \sqrt{1 + \sqrt{3 + \sqrt{5 + \ldots + \sqrt{2n+1}}}}.$$

Натуральное значение n введите с клавиатуры.

Вариант 8

Задание 1. Вывести на экран первые n чисел-палиндромов в диапазоне от a до b.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Вывести номер наименьшей цифры в каждом введённом числе.

Задание 3. Найти квадратный корень из суммы первых n ($100 \leq n \leq 1000$) составных чисел.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 + \frac{2x}{1!} + \ldots + \frac{(2x)^n}{n!}$$

и функции $Y(x) = e^{2x}$ в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 - \frac{3}{2} \cdot x + \frac{3 \cdot 5}{2 \cdot 4} \cdot x^2 - \frac{3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6} \cdot x^3 + \ldots + (-1)^n \frac{3 \cdot 5 \cdot \ldots \cdot (2n+1)}{2 \cdot 4 \cdot \ldots \cdot 2n} \cdot x^n.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x|<1$) также вводится с клавиатуры.

Вариант 9

Задание 1. Вывести на экран первые n чисел-палиндромов $> a$.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Найти количество чисел, состоящих из одинаковых цифр.

Задание 3. Найти синус суммы первых n ($100 \le n \le 1000$) составных чисел.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 + 2\frac{x}{2} + \ldots + \frac{n^2+1}{n!}\left(\frac{x}{2}\right)^n$$

и функции

$$Y(x) = \left(\frac{x^2}{4} + \frac{x}{2} + 1\right)e^{\frac{x}{2}}$$

в диапазоне от 0 до 1 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = \sqrt{2 + \sqrt{4 + \sqrt{6 + \ldots + \sqrt{2n}}}}.$$

Натуральное значение n введите с клавиатуры.

Вариант 10

Задание 1. Вывести на экран числа-палиндромы в диапазоне от a до b, их сумму и количество.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n чисел-палиндромов. Найти количество чисел, состоящих из нечётных цифр.

Задание 3. Возвести произведение первых n простых чисел в степень m. Программа должна работать при $n \leq 15$.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = x - \frac{x^3}{3} + \ldots + (-1)^n \frac{x^{2n+1}}{2n+1}$$

и функции $Y(x) = \text{arctg}(x)$ в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 - \frac{5}{2}x + \frac{5 \cdot 7}{2 \cdot 4}x^2 - \frac{5 \cdot 7 \cdot 9}{2 \cdot 4 \cdot 6}x^3 + \ldots + (-1)^n \frac{5 \cdot 7 \cdot \ldots \cdot (2n+3)}{2 \cdot 4 \cdot \ldots \cdot 2n}x^n.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 11

Задание 1. Вывести на экран простые числа в диапазоне от a до b, их сумму и количество.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n простых чисел. Для каждого введённого числа вывести все его делители.

Задание 3. Найти корень кубический из суммы первых n ($100 \leq n \leq 1000$) простых чисел.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 - \frac{3}{2}x^2 + \ldots + (-1)^n \frac{2n^2+1}{(2n)!}x^{2n}$$

и функции

$$Y(x) = \left(1 - \frac{x^2}{2}\right)\cos(x) - \frac{x}{2}\sin(x)$$

в диапазоне от 0 до 1 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = \sqrt{2n + \sqrt{2(n-1) + \ldots + \sqrt{4 + \sqrt{2}}}}.$$

Натуральное значение n введите с клавиатуры.

Вариант 12

Задание 1. Среди чисел, больших 100, найти первые 50 чисел-палиндромов и первые 70 простых чисел.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n простых чисел. Для каждого введённого числа вывести сумму его делителей.

Задание 3. Найти куб суммы первых n ($100 \leq n \leq 1000$) чисел-палиндромов.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = -4\frac{x^2}{2!} + 16\frac{x^4}{4!} \ldots + (-4)^n \frac{x^{2n}}{(2n)!}$$

и функции $Y(x) = \cos(2x) - 1$ в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots + \frac{x^{2n}}{(2n)!}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 13

Задание 1. Среди чисел, больших a, найти первые n чисел-палиндромов и первые m простых чисел.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n простых чисел. Для каждого введённого числа вывести его наибольший делитель, меньший самого числа.

Задание 3. Найти сумму квадратов первых n ($100 \leq n \leq 1000$) чисел, кратных 7.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 2x - \frac{(2x)^2}{2} + \frac{(2x)^2}{3} + \ldots + (-1)^{n-1} \cdot \frac{(2x)^n}{n}$$

и функции $Y(x) = \ln(2 \cdot x + 1)$ в диапазоне от 0 до 0.5 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = x + \frac{x^2}{1 \cdot 2} + \frac{x^3}{2 \cdot 4} + \frac{x^4}{3 \cdot 8} + \ldots + \frac{x^{n+1}}{n \cdot 2^n}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 14

Задание 1. Вводится последовательность целых чисел, которая заканчивается после ввода 20-го простого числа. Найти два наибольших значения из введённых чисел.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n простых чисел. Для каждого введённого числа вывести его наименьший делитель > 1.

Задание 3. Найти корень пятнадцатой степени из произведения первых n чётных чисел, но не кратных 8. Программа должна работать при $n = 13$.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = \frac{x}{3} - \frac{x^3}{162} + \ldots + (-1)^k \cdot \frac{\left(\frac{x}{3}\right)^{2n+1}}{(2n+1)!}$$

и функции

$$Y(x) = \sin\left(\frac{x}{3}\right)$$

в диапазоне от 0 до 1 с произвольным шагом h. Значение n для вычисления суммы вводится с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = 1 + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots + \frac{x^{2n+3}}{(2n+3)!}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

Вариант 15

Задание 1. Вводится последовательность целых чисел, которая заканчивается после ввода 17-го числа-палиндрома. Найти наибольшее и наименьшее значения из введённых чисел.

Задание 2. Вводится последовательность целых чисел, которая заканчивается после ввода n простых чисел. Для каждого введённого числа проверить, является ли сумма делителей числа простым числом.

Задание 3. Найти косинус суммы первых n ($100 \leq n \leq 1000$) простых чисел.

Задание 4. Составить программу вычисления значения суммы

$$S(x) = 1 - \frac{x^2}{18} + \frac{x^4}{1944} \ldots + \left(-\frac{1}{9}\right)^n \cdot \frac{x^{2n}}{(2n)!}$$

и функции

$$Y(x) = \cos\left(\frac{x}{3}\right)$$

в диапазоне от 0 до 1 с произвольным шагом h. $S(x)$ накапливать до тех пор, пока модуль очередного слагаемого не станет меньше ϵ, вводимого с клавиатуры. Вывести на экран таблицу значений функции $Y(x)$ и её разложение в ряд $S(x)$. Близость значений $Y(x)$ и $S(x)$ во всём диапазоне значений x указывает на правильность их вычисления.

Задание 5. Написать программу для вычисления y по формуле:

$$y = -1 + \frac{1+x}{2} - \frac{(1+x)^2}{4} + \ldots + (-1)^{n+1}\frac{(1+x)^n}{2^n}.$$

Натуральное значение n введите с клавиатуры. Значение x ($|x| < 1$) также вводится с клавиатуры.

13.1.2 Программирование задач обработки массивов в Scilab

Задание 1

1. Упорядочить по убыванию элементы целочисленного массива, расположенные между двумя наибольшими чётными значениями.
2. Упорядочить в порядке возрастания модулей элементы массива, расположенные между наибольшим и наименьшим значениями.
3. Упорядочить в порядке убывания модулей элементы, расположенные между первым и последним отрицательными значениями массива.
4. Упорядочить в порядке убывания элементы, расположенные между вторым положительным и предпоследним отрицательным значениями массива.

5. Упорядочить по возрастанию элементы целочисленного массива, расположенные между первым числом-палиндромом и последним отрицательным значением.
6. Упорядочить в порядке возрастания суммы цифр элементы целочисленного массива, расположенные между последним числом-палиндромом и первым простым числом.
7. Упорядочить по возрастанию модулей элементы целочисленного массива, расположенные между третьим и пятым простыми числами.
8. Упорядочить по убыванию элементы целочисленного массива, расположенные после минимального числа-палиндрома.
9. Удалить из целочисленного массива простые числа. В полученном массиве упорядочить по возрастанию модулей элементы, расположенные после наибольшего числа.
10. Удалить из целочисленного массива числа-палиндромы. В полученном массиве упорядочить по возрастанию модулей элементы, расположенные до наименьшего простого числа.
11. Удалить из целочисленного массива все составные числа. Упорядочить элементы массива в порядке возрастания суммы цифр чисел.
12. Удалить из целочисленного массива все числа, состоящие из одинаковых цифр. Упорядочить элементы массива в порядке убывания суммы их цифр.
13. Задан массив целых положительных чисел. Сформировать новый массив, куда записать элементы исходного массива, состоящие из одинаковых цифр. Упорядочить элементы полученного массива в порядке возрастания суммы цифр чисел.
14. Упорядочить по возрастанию модулей элементы, расположенные между двумя наименьшими значениями массива.
15. Упорядочить в порядке возрастания элементы, расположенные между четвёртым и девятым отрицательными числами массива.

Задание 2

1. Задан массив целых чисел $X(n)$. Все простые числа переписать в массив Y. Из массива Y удалить 5 наибольших элементов массива.
2. Заданы массивы целых чисел $X(n)$ и $Y(k)$. Все совершенные числа из этих массивов переписать в массив Z. В массиве Z найти четыре наименьших элемента. Удалить из массива Z все нулевые элементы.
3. Заданы массивы целых чисел $X(n)$ и $Y(k)$. Два наибольших элемента из массива X и пять последних простых чисел из массива Y переписать в массив Z. Проверить, содержит ли массив Z числа, в которых есть цифра 7.
4. Заданы массивы целых чисел $X(n)$ и $Y(k)$. Три наименьших простых числа из массива Y и числа из массива X, в которых есть цифры 1 и 9, переписать в массив Z. Из массива Z удалить все нечётные числа.
5. Задан массив целых чисел $X(n)$. Шесть наибольших чисел этого массива переписать в массив Z. Удалить из массива Z все чётные числа.

6. Заданы массивы целых чисел $X(n)$ и $Y(k)$. Числа из массива X, в которых нет нулей, и составные числа из массива Y, переписать в массив Z. Найти в массиве Z пять наибольших нечётных чисел. Выполнить сортировку массивов X, Y и Z в порядке возрастания их элементов.
7. В целочисленном массиве Z найти пять наибольших простых чисел. Удалить из массива Z все составные числа.
8. Задан массив целых положительных чисел $X(n)$. Все простые числа длиной не более пяти цифр переписать в массив Y. Удалить из массива два наибольших и три наименьших числа.
9. Найти сумму трёх наименьших и четырёх наибольших чисел целочисленного массива Z.
10. Заданы массивы целых положительных чисел $X(n)$, $Y(k)$, $Z(m)$. Сформировать массив U из таких элементов массивов X, Y, Z, которые представляют собой возрастающую последовательность цифр. Найти пять наибольших чисел в массиве U.
11. Задан массив целых положительных чисел $X(n)$. Все числа, в которых нет цифр 1, 2 и 3, переписать в массив Y. Найти сумму двух наибольших и трёх наименьших простых чисел в массиве Y.
12. Заданы массивы целых положительных чисел $X(n)$, $Y(k)$, $Z(m)$. Сформировать массив U из таких элементов массивов X, Y, Z, которые состоят из одинаковых цифр. Удалить из массива U наибольшее и наименьшее числа. Выполнить сортировку массивов $X(n)$, $Y(k)$, $Z(m)$ в порядке возрастания их элементов.
13. Задан массив целых положительных чисел $X(n)$. Все числа, в которых нет цифры ноль, а их длина не менее трёх цифр, переписать в массив Z. Поменять местами наибольшее составное число и наименьшее простое число в массиве Z.
14. Задан массив целых чисел $X(n)$. Все положительные числа, состоящие из одинаковых цифр, переписать в массив Z. Удалить из массива Z числа с чётной суммой цифр.
15. Заданы массивы целых чисел $X(n)$ и $Y(k)$. Все числа с нечётной суммой цифр переписать в массив Z. Найти три наибольших простых числа в массиве Z.

13.1.3 Программирование задач обработки матриц в Scilab

Задание

1. Задана матрица целых чисел $A(n \times m)$. Сформировать массив $B(m)$, в который записать среднее арифметическое элементов каждого столбца заданной матрицы. Вывести номера строк матрицы, в которых находится более двух простых чисел.
2. Задана матрица вещественных чисел $B(n \times m)$. Сформировать массив $A(n)$, в который записать среднее геометрическое положительных элементов каждой строки заданной матрицы. Определить количество столбцов, упорядоченных по возрастанию.

3. Задана матрица целых чисел $A(n\times n)$. Все простые числа, расположенные на побочной диагонали, заменить суммой цифр максимального элемента соответствующей строки матрицы. Сформировать массив $B(n)$, в который записать произведения элементов нечётных строк заданной матрицы.
4. В матрице целых чисел $X(n\times n)$ поменять местами диагональные элементы, упорядоченные по убыванию строк. Сформировать массив $Y(n)$, в который записать суммы элементов чётных столбцов заданной матрицы.
5. Задана матрица целых чисел $A(n\times n)$. Максимальный элемент каждого столбца заменить суммой цифр максимального элемента матрицы. Сформировать массив $B(n)$, в который записать количество чётных элементов в каждой строке заданной матрицы.
6. Задана матрица целых чисел $B(n\times m)$. Максимальный элемент каждого столбца заменить суммой цифр модуля минимального элемента матрицы. Сформировать массив $A(n)$, в который записать количество нечётных элементов в каждой строке заданной матрицы.
7. Задана матрица целых чисел $A(n\times n)$. Сформировать массив $B(n)$ из максимальных элементов столбцов заданной матрицы. Вывести номера строк, в которых числа-палиндромы находятся на диагоналях матрицы.
8. Задана матрица вещественных чисел $P(n\times m)$. Сформировать массив $R(k)$ из номеров столбцов матрицы, в которых есть хотя бы один ноль. Найти строку с максимальной суммой элементов и поменять её с первой строкой.
9. Задана матрица вещественных чисел $C(k\times m)$. Сформировать вектор $D(k)$ из средних арифметических положительных значений строк матрицы и вектор $G(n)$ из номеров столбцов, которые представляют собой знакочередующийся ряд.
10. В каждом столбце матрицы вещественных чисел $P(k\times m)$ заменить минимальный элемент суммой положительных элементов этого же столбца. Сформировать вектор $D(n)$ из номеров строк, представляющих собой знакочередующийся ряд.
11. В матрице целых чисел $A(n\times m)$ обнулить строки, в которых более двух простых чисел. Сформировать вектор $D(m)$ из минимальных значений столбцов матрицы.
12. В матрице вещественных чисел $P(n\times m)$ найти и вывести номера столбцов, упорядоченных по убыванию элементов. Сформировать вектор $R(n)$ из максимальных значений строк матрицы.
13. В матрице вещественных чисел $D(n\times m)$ найти и вывести номера строк, упорядоченных по возрастанию элементов. Сформировать вектор $C(m\times 2)$ из номеров минимальных и максимальных значений столбцов матрицы.
14. В матрице вещественных чисел $P(n\times m)$ найти и вывести номера столбцов, упорядоченных по возрастанию. Сформировать вектор $R(n\times 2)$ из номеров минимальных и максимальных значений строк матрицы.
15. В матрице вещественных чисел $D(n\times m)$ найти и вывести номера строк, упорядоченных по убыванию. Сформировать вектор $C(m\times 2)$ из максимальных и минимальных значений столбцов матрицы.

13.2 Задания по теме «Решение задач линейной алгебры»

Задание 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. $\begin{cases} 2x_1 + x_2 + 5x_3 + x_4 = 8 \\ 0.333x_1 - x_2 - 2x_4 = 3 \\ 2x_2 + x_3 + 2x_4 = -5 \\ x_1 + 4x_2 + 7x_3 + 6x_4 = 0 \end{cases}$.

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. $\begin{cases} 5x_1 - 3x_2 - 7x_3 + 3x_4 = 1 \\ -x_2 - 3x_3 + 4x_4 = -5 \\ x_1 - 2x_3 - 3x_4 = -4 \\ 1.3333x_1 - x_2 - 1.6667x_3 = 13 \end{cases}$.

8. $\begin{cases} -x_1 + x_2 + x_3 + x_4 = 12 \\ 2x_1 + x_2 + 2x_3 + 3x_4 = 13 \\ 1.5x_1 + x_2 + 0.5x_3 + x_4 = 7 \\ 4x_1 + 3x_2 + 2x_3 + x_4 = -15 \end{cases}$.

Задание 2. Если возможно, вычислить матрицу, обратную к матрице D.

1. $D = 2(A^2 + B)(2B - A)$, где

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

2. $D = 3A - (A + 2B)B^2$, где

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

3. $D = 3A^2 - (A + 2B)B$, где

$$A = \begin{pmatrix} 2 & 4 & 5 \\ 6 & 7 & 13 \\ -7.3 & 11 & 14 \end{pmatrix}, \quad B = \begin{pmatrix} 11 & 6.3 & -7 \\ 17 & 7.6 & -4 \\ -3 & 2 & 1 \end{pmatrix}.$$

4. $D = (A - B^2)(2A + B^3)$, где

$$A = \begin{pmatrix} 5 & 2 & 0 \\ 10 & 4 & 1 \\ 7 & 3 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 6 & -1 \\ -1 & -2 & 0 \\ 2 & 1 & 3 \end{pmatrix}.$$

5. $D = 2(3A - 5B)(A^2 + B^2)$, где

$$A = \begin{pmatrix} 2 & 4 & -5 \\ 16 & -7 & 3 \\ -0.3 & 1.1 & 1.4 \end{pmatrix}, \quad B = \begin{pmatrix} 1.1 & 6 & 0.7 \\ 1.7 & 6 & -4.1 \\ 3 & -2 & 3.3 \end{pmatrix}.$$

6. $D = (A - B)2A + 2B$, где

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

7. $D = (A^2 - B^2)(A + B^2)$, где

$$A = \begin{pmatrix} 7 & 2 & 0 \\ -7 & -2 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 & 3 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{pmatrix}.$$

8. $D = 2(A - B)(A^2 + B)$, где

$$A = \begin{pmatrix} 2 & 4 & -5 \\ 16 & -7 & 3 \\ -0.3 & 1.1 & 1.4 \end{pmatrix}, \quad B = \begin{pmatrix} 1.1 & 6 & 0.7 \\ 1.7 & 6 & -4.1 \\ 3 & -2 & 3.3 \end{pmatrix}.$$

9. $D = 2A - (A^2 + B)B$, где

$$A = \begin{pmatrix} 1 & 4 & 2 \\ 2 & 1 & -2 \\ 0 & 1 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 6 & -2 \\ 4 & 10 & 1 \\ 2 & 4 & -5 \end{pmatrix}.$$

10. $D = 2(A - 0{,}5B) + A^3B$, где

$$A = \begin{pmatrix} 5 & 3 & -1 \\ 2 & 0 & 4 \\ 3 & 5 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 4 & 16 \\ -3 & -2 & 0 \\ 5 & 7 & 2 \end{pmatrix}.$$

11. $D = (A - B)A^2 + 3B$, где

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

12. $D = 3(A^2 + B^2) - 2AB$, где

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

13. $D = 2A^3 + 3B(AB - 2A)$, где

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 3 & 1 \\ -1 & 2 & 0 \\ -3 & 0 & 0 \end{pmatrix}.$$

14. $D = A(A^2 - B) - 2(B + A)B$, где

$$A = \begin{pmatrix} 2 & 3 & 1 \\ -1 & 2 & 4 \\ 5 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 7 & 13 \\ -1 & 0 & 5 \\ 5 & 13 & 21 \end{pmatrix}.$$

15. $D = (2A - B)(3A + B) - 2A^2B$, где

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

13.3 Задания по теме «Построение двумерных графиков»

Задание 1. Изобразить 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. Изобразить график функции в полярных координатах.

1. $\rho(\varphi) = -2\sin(6\varphi)$.
2. $\rho(\varphi) = 2\cos6(\varphi)$.
9. $\rho(\varphi) = 2\varphi^2$.
10. $\rho(\varphi) = \cos(\varphi) - \sin(\varphi)$.

3. $\rho(\varphi) = 2\varphi + 1$.
4. $\rho(\varphi) = 2\cos^2 3\varphi$
5. $\rho(\varphi) = 3\varphi + 2$.
6. $\rho(\varphi) = 3\varphi^2 + \varphi$.
7. $\rho(\varphi) = 3\sin 4\varphi$.
8. $\rho(\varphi) = 1 + \cos(\varphi)$.
11. $\rho(\varphi) = \cos(\varphi) + \sin(\varphi)$.
12. $\rho(\varphi) = 10\cos(\varphi) + 5\sin(\varphi)$.
13. $\rho(\varphi) = 2\cos(\varphi) - 6\sin(\varphi)$.
14. $\rho(\varphi) = \dfrac{16}{5 - 3\sin(\varphi)}$.
15. $\rho(\varphi) = \dfrac{16}{5 + 3\cos(\varphi)}$.

Задание 3. Изобразить график функции, заданный параметрически [7, 11].

1. $\begin{cases} x(t) = a\cos t \\ y(t) = b\sin t, \ t \in [0; 2\pi] \end{cases}$.
2. $\begin{cases} x(t) = \cos^3 t \\ y(t) = \sin^3 t, \ t \in [0; 2\pi] \end{cases}$.
3. $\begin{cases} x(t) = 4\cos t + 3\sin t \\ y(t) = 3\cos t - 4\sin t, \ t \in [0; 2\pi] \end{cases}$.
4. $\begin{cases} x(t) = a\sqrt{2}\dfrac{t + t^3}{1 + t^4} \\ y(t) = a\sqrt{2}\dfrac{t - t^3}{1 + t^4} \end{cases}$.
5. $\begin{cases} x(t) = 2a\cos^2 t + b\cos t \\ y(t) = a\sin 2t + b\sin t, \ t \in [0; 2\pi] \end{cases}$.
6. $\begin{cases} x(t) = a(2\cos t + \cos 2t) \\ y(t) = a(2\sin t + \sin 2t), \ t \in [0; 2\pi] \end{cases}$.
7. $\begin{cases} x(t) = 2a(\cos^2 t + \cos t) \\ y(t) = a(\sin 2t + 2\sin t), \ t \in [0; 2\pi] \end{cases}$.
8. $\begin{cases} x(t) = a(t + \sin t) \\ y(t) = a(1 - \cos t) \end{cases}$.
9. $\begin{cases} x(t) = a(\cos t + t\sin t) \\ y(t) = a(\sin t - t\cos t), \ t \geq 0 \end{cases}$.
10. $\begin{cases} x(t) = 3\cos t - \cos 3t \\ y(t) = 3\sin t - \sin 3t, \ t \in [0; 2\pi] \end{cases}$.
11. $\begin{cases} x(t) = 6\cos t - \cos 6t \\ y(t) = 6\sin t - \sin 6t, \ t \in [0; 2\pi] \end{cases}$.
12. $\begin{cases} x(t) = 5\cos t + \cos 5t \\ y(t) = 5\sin t + \sin 5t, \ t \in [0; 2\pi] \end{cases}$.
13. $\begin{cases} x(t) = 9\cos t + \cos 9t \\ y(t) = 9\sin t + \sin 9t, \ t \in [0; 2\pi] \end{cases}$.
14. $\begin{cases} x(t) = t(2 - t) \\ y(t) = t^2(2 - t) \end{cases}$.
15. $\begin{cases} x(t) = t^2 \\ y(t) = \dfrac{t(t^2 - a)}{\sqrt{a}} \end{cases}$.

13.4 Задания по теме «Построение трёхмерных графиков»

Задание 1. Построить трёхмерный график, заданный параметрически:

$$\begin{cases} x = \cos(u)\cdot u\cdot\left(1+\cos\dfrac{(v)}{2}\right) \\ y = \dfrac{u}{2}\cdot\sin(v) \\ z = (\sin(u)\cdot u)\cdot\left(1+\cos\dfrac{(v)}{2}\right) \end{cases},$$

при помощи функции `plot3d2`.

1. $0 \le u \le 2\pi,\ \ 0 \le v \le 2\pi.$
2. $0 \le u \le 2\pi,\ \ 0 \le v \le 8\pi.$
3. $0 \le u \le 2\pi,\ \ 0 \le v \le 4\pi.$
4. $0 \le u \le 8\pi,\ \ 0 \le v \le 2\pi.$
5. $0 \le u \le 4\pi,\ \ 0 \le v \le 42\pi.$
6. $0 \le u \le 8\pi,\ \ 0 \le v \le 4\pi.$
7. $0 \le u \le 2\pi,\ \ 0 \le v \le 36\pi.$
8. $0 \le u \le 8\pi,\ \ 0 \le v \le 8\pi.$
9. $0 \le u \le 4\pi,\ \ 0 \le v \le 6\pi.$
10. $0 \le u \le 72\pi,\ \ 0 \le v \le 72\pi.$
11. $0 \le u \le 2\pi,\ \ 0 \le v \le 5\pi.$
12. $0 \le u \le 4\pi,\ \ 0 \le v \le 78\pi.$
13. $0 \le u \le 3\pi,\ \ 0 \le v \le 8\pi.$
14. $0 \le u \le 2\pi,\ \ 0 \le v \le 32\pi.$
15. $0 \le u \le 2\pi,\ \ 0 \le v \le 96\pi.$

Задание 2. Изобразить линии, заданные параметрически:

$$\begin{cases} x(t) = \sin(t) \\ y(t) = \sin(2t) \\ z(t) = t/5 \end{cases} \text{ и } \begin{cases} x(t) = \cos(t) \\ y(t) = \cos(2t), \\ z(t) = \sin(t) \end{cases}$$

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**	$[0; 9\pi]$

13.5 Задания по теме «Нелинейные уравнения и системы»

Задание 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. $3.25x^4 + 7.67x^3 + 5x^2 - 11 = 0.$
   $2x^3 + 5x^2 + 11x + 7 = 0.$
10. $2.2x^4 - 1.2x^2 - 11 = 0.$
    $3x^3 - 0.42x^2 + 0.95x - 2 = 0.$
11. $-x^4 - 18x^2 + 6 = 0.$
    $2x^3 - 0.08x^2 + 0.94x + 1.3 = 0.$
12. $-1.21x^4 + x^3 + 2x^2 - 3x - 5 = 0.$
    $3x^3 - 13x^2 + 16x - 15 = 0.$
13. $0.89x^4 + 3.67x^3 - 7.92x^2 - 13 = 0.$
    $2x^3 - 0.35x^2 + 0.47x - 1.43 = 0.$
14. $6x^4 + 8x^3 - 23x^2 + 2.1 = 0.$
    $5x^3 + 20x^2 + 5x + 8 = 0.$
15. $2x^4 - 2x^3 - 4x^2 + 6x - 7 = 0.$
    $1.9x^3 + 7x - 11 = 0.$

Задание 2. Решить нелинейное уравнение.

1. $x - \cos x = 0.$
2. $e^{-x} - x^2 = 0.$
3. $x - 0.2\sin(x + 0.5) = 0.$
4. $x^2 - \lg(x + 2) = 0.$
5. $x^2 - \cos x^2 = 6.$
6. $x^2 - 20\sin x = 0{,}14.$
7. $(x - 1)^2 - 0.5e^x = 0.$
8. $\sqrt{x} - 2\cos x = 0.$
9. $2 - x = \ln x.$
10. $e^x + x^2 = 2.3.$
11. $3x - \cos x = 1.$
12. $1.8x^2 - \sin 10x = 0.$
13. $2\ln x - \frac{1}{x} = 0.$
14. $\frac{1}{x^2} - \lg x = 0.$
15. $x\ln x - 100 = 0.$

Задание 3. 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. $\begin{cases}\cos(x-1)+y=0.7\\ 2x+\cos(y)=-1\end{cases}$.
4. $\begin{cases}\sin(x)+2y=-0.32\\ \cos(y)+x=0.25\end{cases}$.
5. $\begin{cases}2(y-5)+\sin(3x)=0.93\\ 3x+4\cos(y)=-0.212\end{cases}$.
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. $\begin{cases}\sin(y+1)=x+1\\ 2y+\cos x=2\end{cases}$.

13.6 Задания по теме «Численное интегрирование»

Задание. Вычислить определённый интеграл.

1. $\int_{1.6}^{2.2}\frac{dx}{\sqrt{x^2+2.5}}$.
2. $\int_{0.5}^{1.2}\frac{\operatorname{tg}x^2}{x+1}dx$.
3. $\int_{1.4}^{2}\frac{dx}{\sqrt{2x^2+0.7}}$.
4. $\int_{2.1}^{3.6}\frac{dx}{\sqrt{x^2-3}}$.
5. $\int_{0.5}^{1.2}\frac{\sin(x^2-0.4)}{x+2}dx$.
6. $\int_{1.3}^{2.5}\frac{dx}{\sqrt{0.2x^2+1}}$.
7. $\int_{0.32}^{0.66}\frac{dx}{\sqrt{x^2+2.3}}$.
8. $\int_{0.8}^{1.6}\frac{\lg(x^2+1)}{x+1}dx$.
9. $\int_{0.6}^{1.4}\frac{dx}{\sqrt{12x^2+0.5}}$.
10. $\int_{0.6}^{1.4}x^2\cos x dx$.
11. $\int_{0.8}^{1.2}\frac{\cos x}{x^2+1}dx$.
12. $\int_{1.2}^{5.8}\frac{x}{12}\sin^2\frac{x}{2}dx$.
13. $\int_{0.6}^{1.6}\frac{dx}{\sqrt{x^2+0.8}}$.
14. $\int_{0.6}^{0.72}(\sqrt{x}+1)\operatorname{tg}2x dx$.
15. $\int_{0.2}^{1}(x+1)\cos x^2 dx$.

13.7 Задания по теме «Обработка экспериментальных данных»

Задание 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. Построить график экспериментальной зависимости, линию регрессии и график подобранной зависимости. Определить суммарную квадратичную ошибку, среднюю и относительные ошибки для линии регрессии и подобранной функциональной зависимости. Написать программу на языке Scilab для решения задачи. Исходные данные для программы хранятся в файле. Структуру файла разработать самостоятельно. Решение проверить с помощью встроенных функций Scilab.

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. $G(s) = As^3 + Bs + D$.

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. $W(s) = As^3 + Bs^2 + Cs + D$.

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. $Y(x) = Ax^4 + Bx^2 + Cx + D$.

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. $V(U) = \dfrac{1}{A + Be^{-U}}$.

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. $Z(t) = At^4 + Bt^3 + Ct^2 + Dt + K$.

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. $R(h) = Ch^2 + Dh + K$.

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. $Y(x) = Ax^3 + Bx^2 + Cx + D$.

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. $Y(x) = Ax^3 + Cx + D$.

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. $R(h) = Ch^2 + K$.

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. $Z(t) = At^4 + Ct^2 + K$.

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. $Z(t) = At^4 + Bt^3 + Dt + K$.

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. $Z(t) = At^4 + Dt + K$.

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 2. В результате эксперимента была определена некоторая табличная зависимость. Вычислить ожидаемое значение функции в указанных точках. Построить график, на котором изобразить экспериментальные точки, график интерполяционной зависимости, ожидаемое значение в указанных точках. Реализовать следующие методы интерполяции:

- интерполяционный полином Лагранжа;
- интерполяционный полином Ньютона;
- канонический полином;
- функцию линейной интерполяции;
- функцию сплайн-интерполяции.

Написать программу на языке Scilab для решения задачи. Исходные данные для программы хранятся в файле. Структуру файла разработать самостоятельно. Решение проверить с помощью встроенных функций Scilab.

1. $x_1 = 0.702$, $x_2 = 0.512$, $x_3 = 608$.

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. $x_1 = 0.102,\ x_2 = 0.203,\ x_3 = 0.154.$

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. $x_1 = 0.526,\ x_2 = 0.453,\ x_3 = 0.436.$

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. $x_1 = 0.616,\ x_2 = 0.478,\ x_3 = 0.537.$

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. $x_1 = 0.896,\ x_2 = 0.774,\ x_3 = 0.955.$

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. $x_1 = 0.314,\ x_2 = 0.235,\ x_3 = 0.186.$

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. $x_1 = 1.3832,\ x_2 = 1.3926,\ x_3 = 1.3866.$

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

8. $x_1 = 0.308,\ x_2 = 0.325,\ x_3 = 0.312.$

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. $x_1 = 0.608,\ x_2 = 0.594,\ x_3 = 0.631.$

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. $x_1 = 0.115,\ x_2 = 0.130,\ x_3 = 0.164.$

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. $x_1 = 0.720,\ x_2 = 0.777,\ x_3 = 0.700.$

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. $x_1 = 0.238,\ x_2 = 0.261,\ x_3 = 0.275.$

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. $x_1 = 0.105,\ x_2 = 0.109,\ x_3 = 0.111.$

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. $x_1 = 0.1817,\ \ x_2 = 0.2275,\ \ x_3 = 0.175.$

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. $x_1 = 3.522,\ \ x_2 = 4.176,\ \ x_3 = 3.475.$

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

13.8 Задания по теме «Решение задач оптимизации»

Задание 1. Найти минимум функции одной переменной. Построить график функции.

1. $f(x) = \ln(1 + x^2) - \sin x, x \in [0; \pi/4]$.
2. $f(x) = \frac{1}{4} \cdot x^4 + x^2 - 8 \cdot x + 12, x \in [0; 2]$.
3. $f(x) = x^3 - x + 1x, x_0 \in [-2; 0]$.
4. $f(x) = 2x^2 - \frac{16}{x}, x \in [1; 5]$.
5. $f(x) = \frac{x^3}{3} - 2x + 3x^2 - e^{x/2}, x \in [-2; 2]$.
6. $f(x) = \frac{2}{3}t^3 - 12t^2, t \in [0; 20]$.
7. $f(x) = x^4 - 12x^3 + 47x^2 - 60x, x \in [0; 3]$.
8. $f(x) = x^3 - 3x^2 - x + 5, x \in [0; 3]$.
9. $f(x) = x^4 + 2x^2 + 4x + 1, x \in [-1; 0]$.
10. $f(x) = x^2 + e^{-x}, x \in [-1; 2]$.
11. $f(x) = 2x + e^{-x}, x \in [-3; 3]$.
12. $f(x) = x^2 + x + \sin x, x \in [-2; 1]$.
13. $f(x) = x^2 - x + e^{-x}, x \in [0; 2]$.
14. $f(x) = 2x^2 - 5 - 2^x, x \in [-2; 2]$.
15. $f(x) = 2.3x^3 - 5x^2 - 2, x \in [0.5; 3]$.

Задание 2. Решить задачу линейного программирования.

1. $W = 2x_1 - x_2 + x_4 \to \min.$

$$\begin{cases} x_1 + x_2 + x_3 - x_4 \le 1 \\ x_1 - x_2 + x_3 - x_4 \le 0 \\ 2x_1 + x_2 + x_3 - x_4 \ge 3 \end{cases}.$$

2. $W = x_1 + x_3 \to \max.$

$$\begin{cases} 2x_1 - 7x_2 + 22x_3 \le 22 \\ 2x_1 - x_2 + 6x_3 \le 6 \\ 2x_1 - 5x_2 + 2x_3 \le 2 \\ -4x_1 + x_2 + x_3 \le 1 \end{cases}.$$

3. $W = 3 + 2x_2 + x_3 \to \max.$
$$\begin{cases} x_1 - x_2 + 2x_3 + x_4 \geq 1 \\ 2x_1 - x_2 + x_3 - x_4 \geq 1 \\ x_1 - 2x_2 + x_3 - x_4 \geq -1 \\ x_1 + x_2 + x_3 + 2x_4 \leq 5 \end{cases}.$$

4. $W = x_3 + 3x_4 \to \min.$
$$\begin{cases} x_1 + x_2 - x_3 - x_4 \leq 2 \\ x_1 - x_2 - x_3 + x_4 \geq 0 \\ -x_1 - x_2 + 2x_3 - x_4 \geq -3 \\ x_1 \geq 1 \end{cases}.$$

5. $W = -x_1 + x_2 \to \max.$
$$\begin{cases} x_1 - 2x_2 \geq 2 \\ 2x_1 - x_2 \geq 2. \\ x_1 + x_2 \geq 5 \end{cases}$$

6. $W = x_1 - x_2 - 2x_4 \to \max.$
$$\begin{cases} 2x_1 - x_2 + 2x_3 - x_4 \leq 4 \\ x_1 - 2x_2 + x_3 - 2x_4 \geq 2 \\ x_1 - x_4 \geq 1 \\ x_2 + x_3 \leq 1 \end{cases}.$$

7. $W = x_1 - x_2 + 3x_3 + x_4 \to \max.$
$$\begin{cases} x_1 - x_2 + x_4 \leq 1 \\ x_2 - x_3 + x_4 \leq 1 \\ x_1 + x_3 + 2x_4 \leq 2 \\ -2x_2 + x_4 \leq 0 \end{cases}.$$

8. $W = -x_2 - 2x_3 + x_4 \to \min.$
$$\begin{cases} 3x_1 - x_2 \leq 2 \\ x_2 - 2x_3 \leq -1 \\ 4x_3 - x_4 \leq 3 \\ 5x_1 + x_4 \geq 6 \end{cases}.$$

9. $W = x_1 + x_2 + 3x_3 - x_4 \to \max.$
$$\begin{cases} x_1 - 5x_2 + 4x_3 \leq 5 \\ x_2 - 2x_3 - 3x_3 \leq 4 \\ x_1 + 6x_2 + 5x_3 \leq 4 \\ x_2 + x_3 \leq 1 \end{cases}.$$

10. $W = -4 - 2x_1 - x_2 - x_3 \to \min.$
$$\begin{cases} x_1 - 2x_2 + 3x_3 - 4x_4 \geq -10 \\ x_1 + x_2 - x_3 - x_4 \leq -4 \\ x_1 - x_2 + x_3 - x_4 \geq -6 \\ x_1 + x_2 + x_3 + x_4 \leq 10 \end{cases}.$$

11. $W = x_1 + x_2 + x_3 + 1 \to \min.$
$$\begin{cases} x_1 + x_2 \geq 0 \\ x_1 + x_3 \geq 1 \\ x_2 - x_3 \geq 1 \\ x_1 + 2x_2 + 3x_3 \geq 0 \end{cases}.$$

12. $W = 2 + 2x_2 - x_3 + 3x_4 \to \max.$
$$\begin{cases} -x_1 + x_2 - 2x_4 \geq -1 \\ x_1 + x_3 + x_4 \geq 1 \\ x_2 + x_3 - x_4 \geq 1 \\ x_3 \leq 4; \quad x_2 \leq 10 \end{cases}.$$

13. $W = x_1 + x_2 + 3 \to \max.$
$$\begin{cases} x_1 - x_2 \leq 1 \\ x_1 - 2x_2 \geq -2 \\ -x_1 + x_2 \geq -1 \\ 2x_1 + x_2 \geq -2 \end{cases}.$$

14. $W = x_1 - 10x_2 + 100x_3 \to \max.$
$$\begin{cases} x_1 + x_2 + x_3 \leq 1 \\ x_1 - x_2 - x_3 \leq 2 \\ -x_1 + 2x_3 \leq 0 \\ x_1 + 2x_3 \leq 5 \end{cases}.$$

15. $W = -3 + x_1 + 3x_2 + 5x_3 \to \max.$
$$\begin{cases} x_1 - x_2 + x_3 \leq 1 \\ 2x_1 + x_2 + x_3 \leq 1 \\ x_1 + 2x_2 + x_3 \leq 1 \\ x_1 + x_2 + 2x_3 \leq 1 \end{cases}.$$

Литература

[1] *Алексеев Е. Р., Чеснокова О. В.* Решение задач вычислительной математики в пакетах Mathcad 12, MATLAB 7, Maple 9. – М.: НТ-Пресс, 2006. – 496с.

[2] *Алексеев Е. Р., Чеснокова О. В., Рудченко Е. А.* Scilab: Решение инженерных и математических задач. – М.: ALT Linux; Бином, 2008. – 260с.

[3] *Алексеев Е. Р., Чеснокова О. В.* Введение в Octave для инженеров и математиков. – М.: ALT Linux, 2011. – 368с.

[4] *Андриевский А. Б., Андриевский Б. Р., Капитонов А. А., Фрадков А. Л.* Решение инженерных задач в Scilab. – СПб.: Литмо, 2013. – 98с.

[5] *Березин И. С., Жидков Н. П.* Методы вычислений. Т. 1. – М.: Физматгиз, 1962. – 464с.

[6] *Березин И. С., Жидков Н. П.* Методы вычислений. Т. 2. – М.: Физматгиз, 1962. – 640с.

[7] *Блинова И.В ., Попов И. Ю.* Кривые, заданные параметрически и в полярных координатах: учеб. пособие. – СПб.: Университет ИТМО, 2017. – 56 с.

[8] *Вержбицкий В. М.* Основы численных методов. – М.: Высшая школа, 2002. – 840 с.

[9] *Голосков Д. П.* Уравнения математической физики. Решение задач в системе Maple. – СПб.: Питер, 2004. – 539 с.

[10] *Квасов Б. И.* Численные методы анализа и линейной алгебры. – СПб.: Лань, 2016. – 328с.

[11] *Сильванович О. В.* Лабораторный практикум по высшей математике. Специальные кривые. – СПб.: Университет ИТМО, 2018. – 62 с.

[12] *Титов А. Н., Тазиева Р. Ф.* Решение задач линейной алгебры и прикладной математики в среде Scilab / Казан. нац. исслед. технол. ун-т. – Казань: Изд-во КНИТУ, 2020. – 98 с.

[13] *Тихонов А. Н., Самарский А. А.* Уравнения математической физики. – М.: Наука, 1966. – 724 с.

[14] *Тропин И. С., Михайлова О. И., Михайлов А. В.* Численные и технические расчёты в среде Scilab. – М.: 2008. – 65 с.

[15] *Шарый С. П.* Курс вычислительных методов. – Новосибирск, 2022. – 691с.

[16] www.freefem.org.

Предметный указатель

У

Ф

Э

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38, оф. 10;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине:
http://www.galaktika-dmk.com/.

Учебное издание

Серия «Библиотека ALT»

Евгений Ростиславович Алексеев,
Кристина Вячеславовна Дога,
Оксана Витальевна Чеснокова

Scilab

Решение инженерных и математических задач

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Редактор серии *Чёрный В. Л.*
Редактор *Губина Т. Н.*
Корректор *Синяева Г. И.*
Вёрстка *Чаннова А. А.*
Дизайн обложки *Осмоловская А. С.*

Издание доступно в РИНЦ по адресу **https://elibrary.ru**.

ООО «Базальт СПО»
Адрес для переписки: 127015, Москва, Бутырская ул, д. 75, оф. 301
Телефон: (495) 123-47-99. E-mail: **sales@basealt.ru**.
https://www.basealt.ru

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 35,75. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**