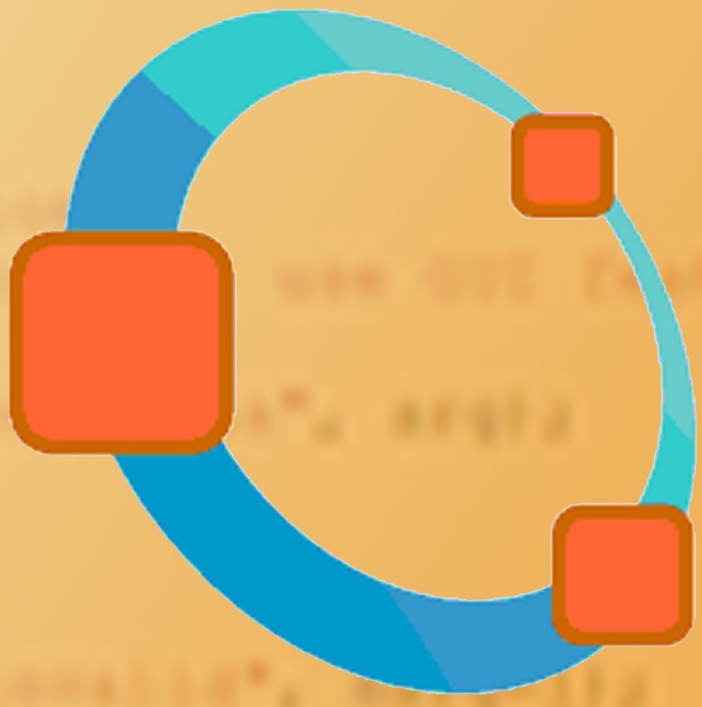
Компьютерные науки и технологии

Алексеев Е.Р., Чеснокова О.В.

GNU Octave

для студентов и преподавателей

Свободная система математических вычислений



ДонНТУ

Алексеев Евгений Ростиславович –
кандидат технических наук, доцент, профессор кафедры вычислительной математики и программирования Донецкого национального технического университета, доцент кафедры «Документоведение и информационная деятельность» Донецкого инженерно-экономического колледжа, автор 14 книг, изданных в Москве и Донецке, общий тираж которых превышает 70 тыс. экземпляров, а также более 80 научных и методических работ, специалист в области свободного программного обеспечения, программирования, вычислительной математики, Интернет-технологий.

Чеснокова Оксана Витальевна –
старший преподаватель кафедры «Вычислительная математика и программирование» Донецкого национального технического университета, автор 10 книг, изданных в Москве и Донецке, общий тираж которых превышает 55 тыс. экземпляров, а также более 60 научных и методических работ, специалист в области свободного программного обеспечения, программирования и вычислительной математики.

Алексеев Е.Р.
Чеснокова О.В.

GNU Octave для студентов и преподавателей

Министерство образования и науки, молодежи и спорта Украины
ГВУЗ «Донецкий национальный технический университет»

серия «Компьютерные науки и технологии»

90-летию ДонНТУ посвящается

Алексеев Е.Р., Чеснокова О.В.

GNU Octave
для студентов и преподавателей

УНИТЕХ

УДК 519.87 (075.8)

ISBN 978-966-8248-30-6

Рецензенты: *Аноприенко А.Я.* — кандидат технических наук, профессор, декан факультета компьютерных наук и технологий Донецкого национального технического университета.

А 47 **Алексеев Е.Р., Чеснокова О.В.** GNU Octave для студентов и преподавателей. - Донецк.: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 332с.

Рекомендовано Ученым советом Донецкого национального технического университета, как учебное пособие для студентов всех специальностей (протокол № 6 от 17 июня 2011г).

Ответственный за выпуск заведующий кафедрой вычислительной математики и программирования, д.т.н., проф. Павлыш В.Н.

Книга посвящена свободно распространяемому пакету Octave. Читатель держит в руках первое описание пакета на русском языке. Описан встроенный язык пакета, подробно рассмотрены графические возможности пакета.

Подробно рассмотрено решение различных инженерных и математических задач. Особое внимание уделено операциям с матрицами, решению нелинейных уравнений и систем, дифференцированию и интегрированию, решению дифференциальных уравнений, оптимизационным задачам и обработке экспериментальных данных (интерполяции и аппроксимации). Наряду со встроенным языком пакета описана среда qtOctave.

Книга адресована студентам, преподавателям инженерных и математических специальностей, а также научным работникам.

ISBN 978-966-8248-30-6 **ООО** «Технопарк ДонНТУ УНИТЕХ»

Содержание

Введение

Книга посвящена решению задач вычислительной математики в свободно распространяемом пакете Octave. На взгляд авторов GNU Octave является одним из самых мощных математических пакетов. По своим возможностям он уже сейчас превосходит такой широко известный проприетарный пакет, как MathCAD, и является конкурентом проприетарному пакету MATLAB при решении задач вычислительной математики.

Синтаксис и правила работы в Octave достаточно понятен и схож с MATLAB, пакет обладает широким спектром встроенных элементарных математических функций и поддерживает работу с комплексными числами. Кроме того, можно создавать свои собственные функции.

Octave обладает мощной графической базой, поддерживает двух-, трехмерную графику и анимацию.

В программе предусмотрено огромное количество специальных функций, предназначенных для работы с матрицами и векторами, решением задач линейной алгебры. Octave хорошо справляется с решением сложных нелинейных уравнений и систем. Существует множество функций, реализующих численное интегрирование и дифференцирование. Octave предоставляет достаточное количество функций для решения дифференциальных уравнений различного вида. Очень мощные алгоритмы решения многих оптимизационных задач реализованы во встроенных функциях Octave. Также очень удобен Octave и при решении задач обработки эксперимента. Этим возможности стандартного пакета GNU Octave не ограничиваются.

К особенностям программы можно отнести тот факт, что Octave не имеет мощной графической оболочки, и это вызывает определённые сложности при его освоении. Однако, это компенсируется мощным языком программирования, большим количеством специальных функций и огромным количеством пакетов расширений, которые позволяют решать задачи практически из любых отраслей знаний.

Именно поэтому авторы в своей книге решили сосредоточиться именно на языке Octave, особое внимание уделено функциям решения математических задач.

Это первое описание пакета на русском языке. На большом количестве практических примеров подробно рассмотрены основы работы в пакете, методы решения задач линейной алгебры, аналитической геометрии, обработки экспериментальных данных, математического анализа. Отдельные главы посвящены методам решения нелинейных уравнений и систем, численному решению дифференциальных уравнений и задач оптимизации. В книге приведены необходимые сведения по основам программирования в пакете Octave и описана среда qtOctave.

В первую очередь издание предназначено для читателей, осваивающих численные методы и решение вычислительных задач в математических пакетах. Книга может быть полезна студентам, аспирантам и преподавателям технических вузов, а также инженерам, научно-техническим работникам и всем, применяющим численные методы при решении прикладных задач.

Книга состоит из двенадцати глав.

В *первой главе* читатель знакомится с Octave и получает необходимую информацию о его установке на свой ПК.

Вторая глава дает читателю возможность изучить основы работы с пакетом и выполнить элементарные вычисления.

Третья глава посвящена основам программирования в Octave. Прочитав эту главу читатель сможет создавать свои программы и функции и тем самым значительно расширит свои возможности работы с пакетом.

В *четвертой главе* описаны графические возможности Octave. Изучив эту главу

читатель сможет самостоятельно создать график, поверхность или движущееся изображение.

Пятая глава познакомит читателя с инструментами Octave, предназначенными для работы с векторами и матрицами, а также с возможностями, которые предоставляет пакет при непосредственном решении задач линейной алгебры.

В *шестой главе* рассмотрено решении задач векторной алгебры и аналитической геометрии с использованием графических средств Octave.

В *седьмой главе* рассказывается о решении нелинейных уравнений и систем.

Восьмая глава посвящена методам численного интегрирования и дифференцирования.

В *девятой главе* представлены методы решения обыкновенных дифференциальных уравнений средствами пакета.

Десятая глава знакомит читателя с возможностями Octave при решении различных оптимизационных задач.

Последние главы посвящены вопросам обработки результатов эксперимента. В одиннадцатой *главе* описан метод наименьших квадратов и его реализация в Octave. Двенадцатая глава рассказывает читателю о различных методах интерполирования и возможностях пакета при решении подобных задач.

С рабочими материалами книги можно познакомиться на сайте http://gnu-octave.narod2.ru.

Авторы выражают благодарность своим родным за помощь и понимание.

Алексеев Е.Р., Чеснокова О.В.

Донецк, сентябрь 2011 г.

Сведения об авторах

Алексеев Евгений Ростиславович — кандидат технических наук, доцент, профессор кафедры «Вычислительная математика и программирование» Донецкого национального технического университета. Преподавательский стаж Алексеева Е.Р. — 19 лет. Евгений Ростиславович — автор тринадцати книг, вышедших в Москве и Донецке, общий тираж которых превышает 70 тыс. экземпляров. Е.Р. Алексеев — автор более 80 научных и методических работ. Область его научных интересов — программирование, вычислительная математика, Интернет-технологии, использование свободно распространяемого программного обеспечения.

Чеснокова Оксана Витальевна — старший преподаватель кафедры «Вычислительная математика и программирование» Донецкого национального технического университета. Преподавательский стаж Чесноковой О. В. более 15 лет. Оксана Витальевна — автор десяти книг, общий тираж которых превышает 55 тыс. экземпляров, а также — около 60 научных и методических работ. Область ее научных интересов — программирование, вычислительная математика.

1. Общие сведения об Octave. Установка Octave на компьютер

GNU Octave [11-13] — высокоуровневый интерпретируемый язык программирования, предназначенный для решения задач вычислительной математики. Свое имя он получил в честь профессора Octave Levenspiel и первоначально разрабатывался как вспомогательное программное обеспечение для курса по проектированию ядерных реакторов еще в 1988 году. Активная работа над Octave началась в 1992 году. Первая версия программы вышла 17 февраля 1994 года.

Octave представляет интерактивный командный интерфейс (интерпретатор Octave), реализованный в ОС Windows и Linux, для решения задач вычислительной математики. Интерпретатор Octave запускается из терминала ОС Linux или из его порта в Windows. После запуска Octave пользователь видит окно интерпретатора Octave (рис. 1.1).

```
Файл  Правка  Вид  Терминал  Справка
octave-3.2.3:2> a=[1 2 3;4 5 6;8 7 9]
a =

   1   2   3
   4   5   6
   8   7   9

octave-3.2.3:3> inverse(a)
ans =

  -0.33333  -0.33333   0.33333
  -1.33333   1.66667  -0.66667
   1.33333  -1.00000   0.33333

octave-3.2.3:4> a*ans
ans =

   1.0000e+00  -2.2204e-16   1.6653e-16
  -2.2204e-16   1.0000e+00   3.3307e-16
  -4.4409e-16  -4.4409e-16   1.0000e+00

octave-3.2.3:5>
```

Рис. 1.1 Окно интерпретатора Octave

В окне интерпретатора Octave пользователь может вводить, как отдельные команды языка Octave, так и группы команд, объединяемые в программы. Если строка заканчивается символом «;», то результаты на экран не выводятся. Если в конце строки символ «;» отсутствует, то результаты работы выводятся экран (рис. 1.2). Текст в стоке, который идет после символа % является строкой комментария и интерпретатором не обрабатывается[1] (рис. 1.2). Рассмотрим несколько несложных примеров.

ЗАДАЧА 1.1. Решить систему линейных алгебраических уравнений (СЛАУ)

1 Строки комментариев авторы книги будут использовать для пояснения функций и текстов программ.

$$\begin{cases} 3x_1+5x_2-7x_3=11 \\ 3x_1-4x_2+33x_3=25 \\ 22x_1-11x_3+17x_3=22 \end{cases}.$$

Возможны два варианта решения любой задачи в Octave:

1. *Терминальный режим*. В этом режиме последовательно вводятся отдельные команды в окне интерпретатора.
2. *Программный режим*. В программном режиме создается текстовый файл (с расширением **.m**), в котором хранятся последовательно выполняемые команды Octave. А потом этот текстовый файл (программа на языке Octave) запускается на выполнение в среде Octave.

```
Файл  Правка  Вид  Терминал  Справка
octave-3.2.3:5> a=[1 2 3; 4 5 6;7 8 9] % Это определение матрицы a
a =

   1   2   3
   4   5   6
   7   8   9

octave-3.2.3:6> b=[11 21 35; 41 25 16;17 83 93]; % Это определение матрицы
octave-3.2.3:7> c=a*b % Умножение матриц
c =

    144    320    346
    351    707    778
    558   1094   1210

octave-3.2.3:8>
```

Рис. 1.2 Использование символов «;» и «%» в Octave

Для решения СЛАУ в терминальном режиме интерпретатора Octave необходимо ввести следующие команды (листинг 1.1):

```
%Определение матрицы
%коэффициентов системы линейных уравнений.
A=[3 5 -7;3 -4 33;22 -11 17];
% Вектор правых частей СЛАУ.
b=[11; 25; 22];
%Решение системы методом обратной матрицы.
x=A^(-1)*b
x = 1.56361
    2.55742
    0.92542
octave-3.2.3:27> A*x %Проверка.
ans = 11.000
      25.000
      22.000
```

Листинг 1.1.

В переменной `ans` хранится результат последней операции, если команда не содержит знака присваивания. Следует помнить, что значение переменной `ans` изменяется после каждого вызова команды без операции присваивания.

Теперь рассмотрим, как решить эту же задачу в программном режиме. Вызовем любой

текстовый редактор[2], например `gedit`, в окне которого последовательно введем следующие команды:

```
A=[3 5 -7;3 -4 33;22 -11 17]
b=[11; 25; 22]
x=A^(-1)*b
A*x
```

Сохраним введенные команды в виде файла с расширением **.m**. Например, **/home/evgeniy/prim1_1.m** (рис. 1.3). Теперь эту программу необходимо запустить на выполнение из интерпретатора.

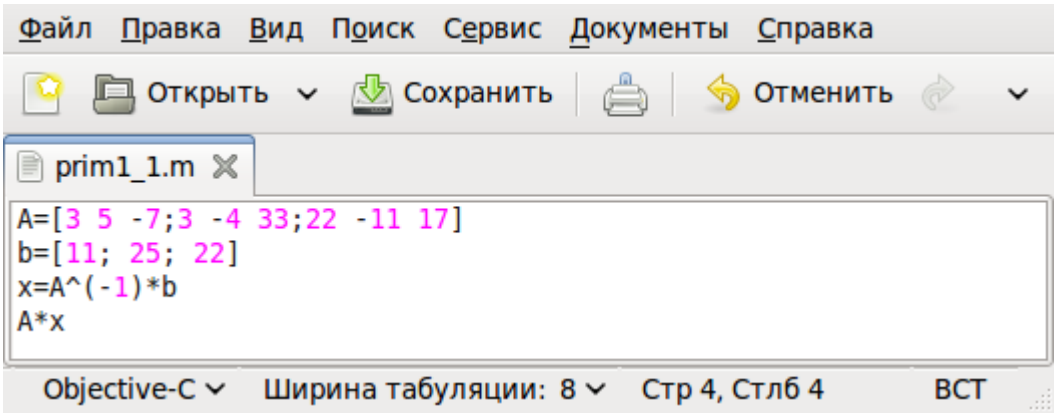


Рис. 1.3. Программа для решения примера 1.1

Для этого в окне интерпретатора введем команды (листинг 1.2):

```
cd '/home/evgeniy'% Переход в каталог, где хранится программа.
prim1_1 % Запуск программы.
```

Листинг 1.2

Окно интерпретатора примет вид, представленный на рис. 1.4. Просмотрев результаты работы программы, нажав символ `q`, вернемся в режим ввода команд терминала.

ЗАДАЧА 1.2. Решить квадратное уравнение $ax^2+bx+c=0$.

Напомним читателю, что корни квадратного уравнения определяют по формулам

$$x_{1,2}=\frac{-b\mp\sqrt{(D)}}{2a},$$

где дискриминант D вычисляется по формуле $D=b^2-4ac$. В Octave, как и в большинстве математических пакетов, все математические функции определены сразу как для действительных, так и для комплексных чисел, поэтому нет необходимости в тексте программы проверять знак D. Текст программы решения задачи 1.2 приведён в листинге 1.3.

```
a=input('a='); % Ввод значения переменной a.
b=input('b='); % Ввод значения переменной b.
c=input('c=');% Ввод значения переменной c.
d=b^2-4*a*c; % Вычисление значения дискриминанта.
x1=(-b+sqrt(d))/2/a % Вычисление значения x1.
x2=(-b-sqrt(d))/2/a % Вычисление значения x2.
```

Листинг 1.3

Для запуска программы на выполнение в окне интерпретатора введем текст:

```
cd '/home/evgeniy'
prim1_2
```

Здесь, `/home/evgeniy` — имя папки, где хранится программа, `prim1_2.m` — имя файла в папке `/home/evgeniy`, где хранится листинг 1.3.

2 Именно текстовый редактор! Не путайте с текстовыми процессорами типа Microsoft Word, OpenOffice.org Writer.

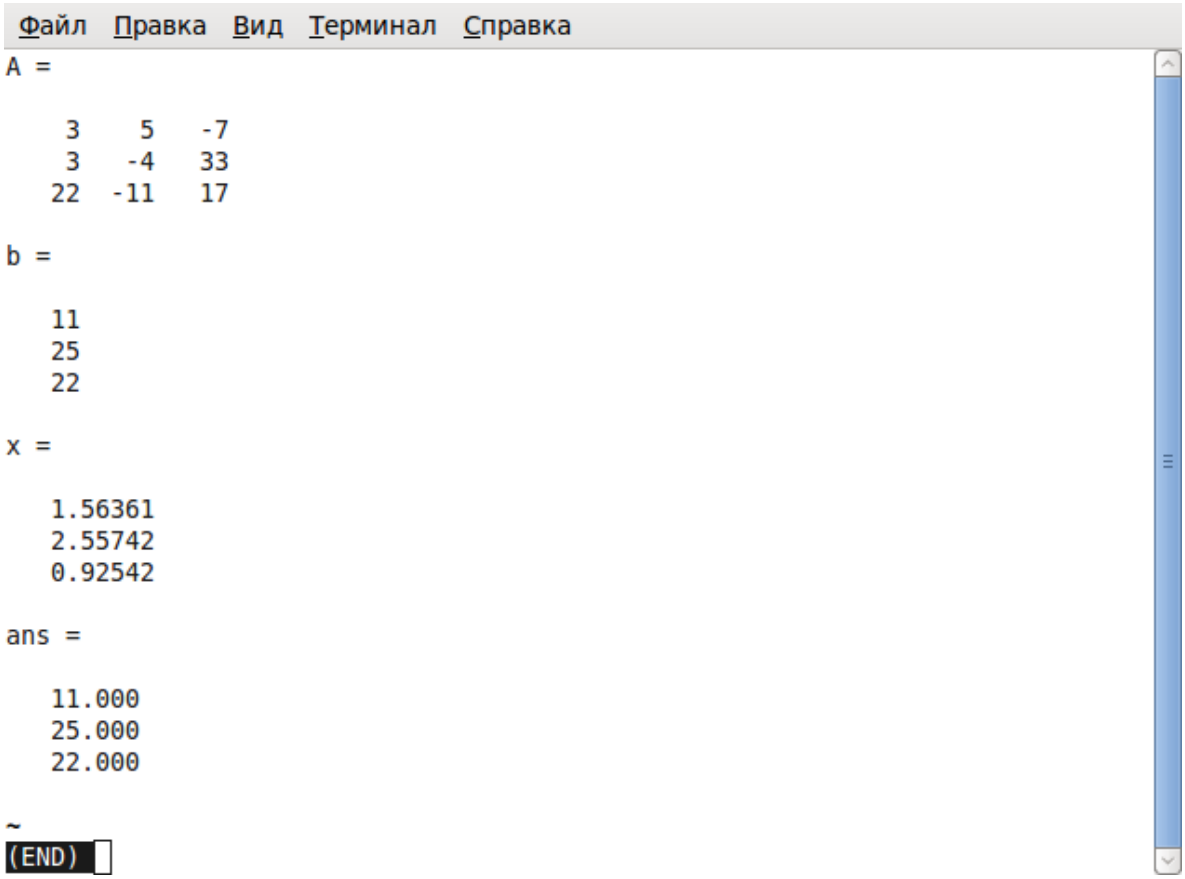


Рис. 1.4. Окно терминала

Далее пользователь должен ввести значение переменных *a*, *b* и *c*, после чего появятся результаты работы программы (листинг 1.4).

```
octave-3.2.3:5> prim1_2
a=2
b=1
c=1
x1 = -0.25000 + 0.66144i
x2 = -0.25000 – 0.66144i
```

Листинг 1.4

ЗАДАЧА 1.3. Построить графики функций *y=sin(x)* и *z=cos(x)* на интервале [*-2π*; *2π*].

Для вычисления значения *π* в Octave есть встроенная функция без параметров *pi()*. Для построения графика функций *y=sin(x)* и *z=cos(x)* в окне интерпретатора Octave надо ввести следующие команды:

```
x=-2*pi():0.02:2*pi(); y=sin(x); z=cos(x); plot(x,y,x,z)
```

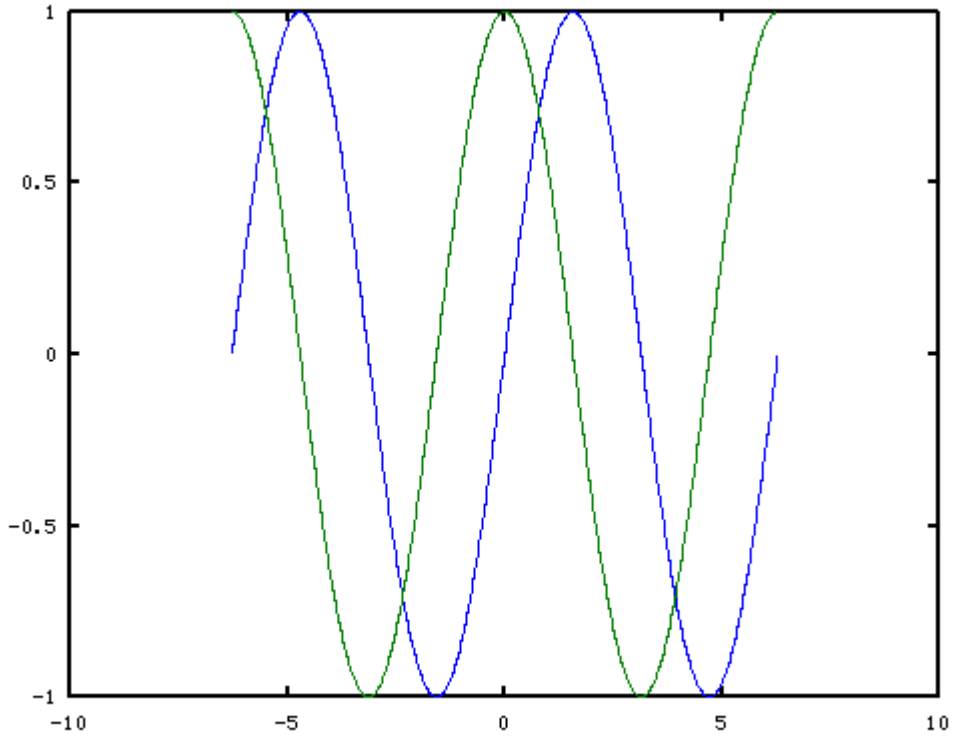
Результатом работы команд будет графическое окно с графиками двух функций *y=sin(x)* и *z=cos(x)*. (рис. 1.5).

Как видно из простейших примеров у Octave достаточно широкие возможности, по синтаксису он близок к Matlab.

Однако, для практического использования Octave интерпретатор не совсем удобен, поэтому были разработаны *профессиональные графические оболочки* для работы с Octave:

1. Octave workshop (рис. 1.6) — графическая оболочка для работы в ОС Windows.
2. QtOctave (рис. 1.7) — графическая оболочка для работы в ОС Linux (портирована в ОС Windows в виде portable версии).

Рассмотрим процесс установки Octave и графических оболочек на персональный компьютер.



-13.4152, 0.736290

Рис. 1.5. Графики функций y=sin(x), z=cos(x)

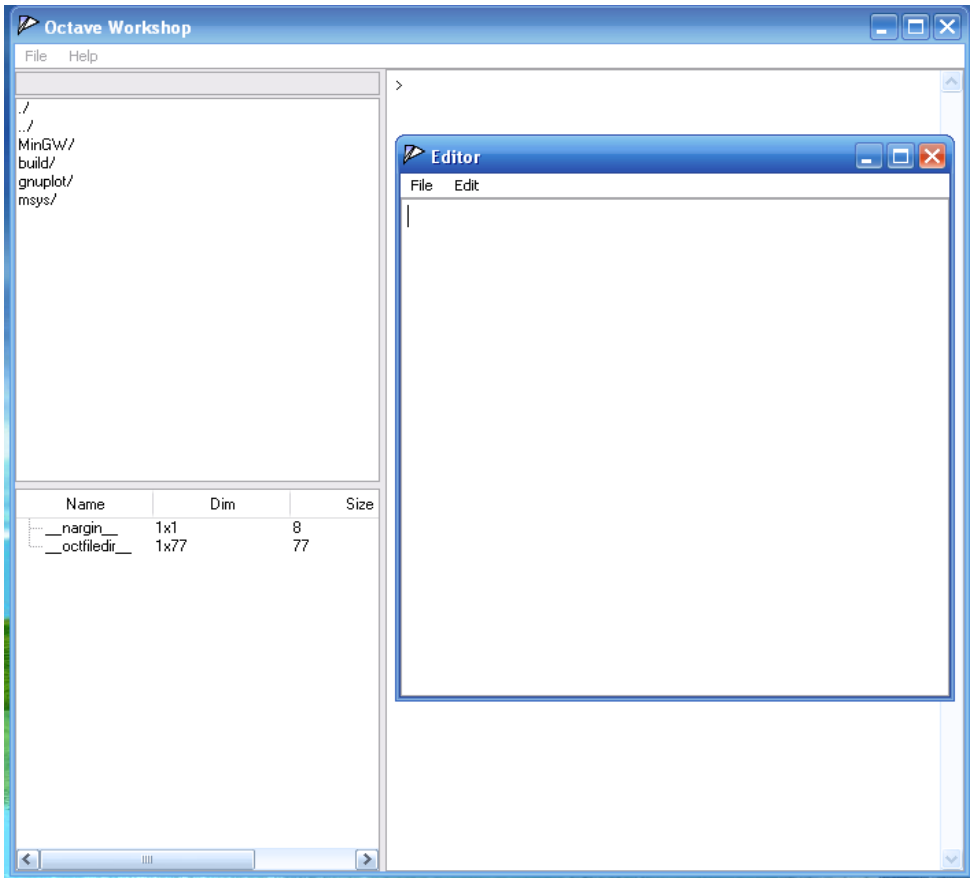


Рис. 1.6. Окно Octave workshop

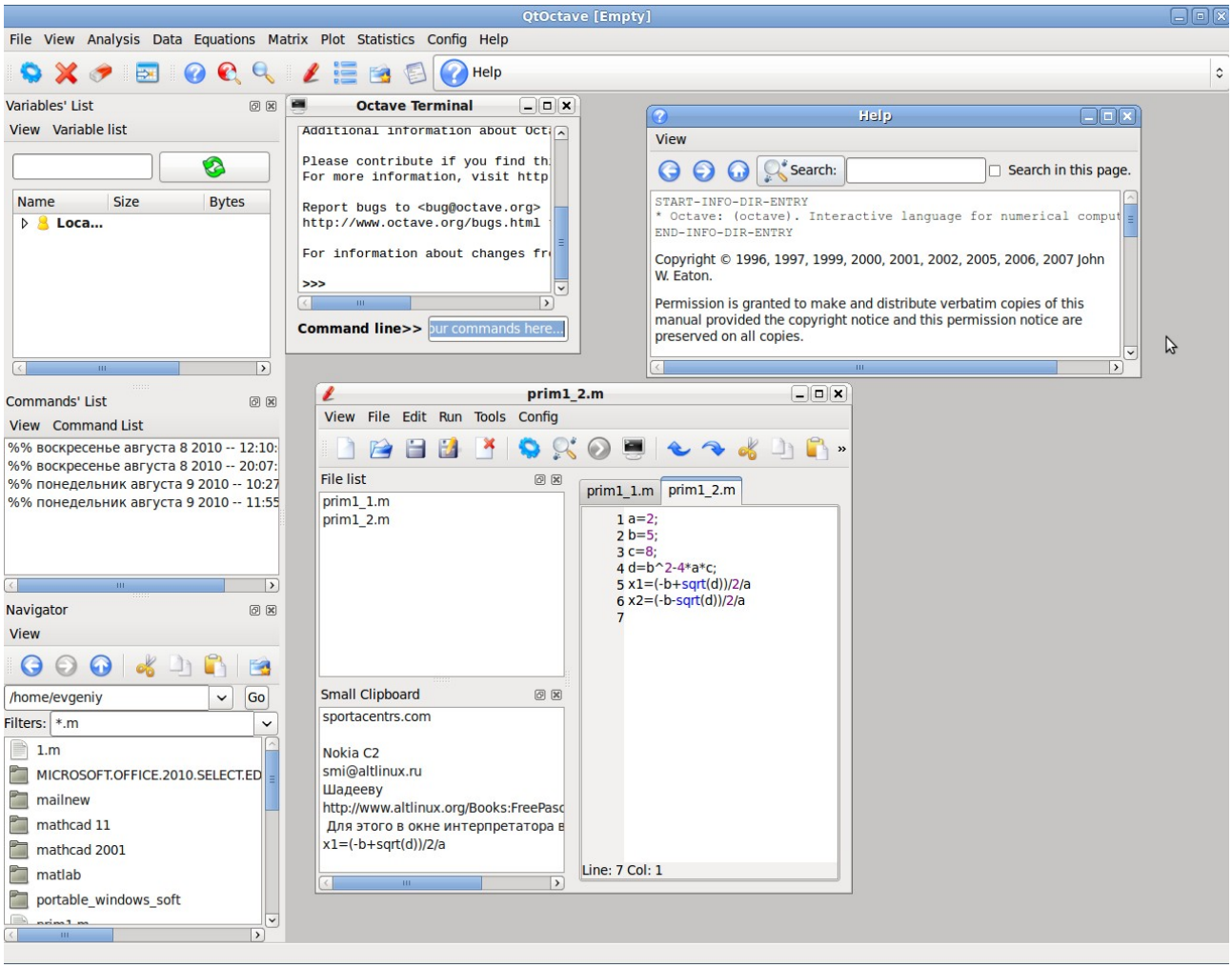


Рис. 1.7. Окно QtOctave

1.1 Установка Octave в Windows

Установка Octave в ОС Windows проходит стандартным образом. Необходимо с официального сайта [11] скачать Windows-версию программы (http://www.gnu.org/software/octave/download.html) и принятым в Windows способом установить ее.

На первом этапе нужно выбрать папку для установки программы(рис. 1.8). На следующем этапе — определить правильно ли выбрана платформа (параметр — ATLAS Libraries), под которую будет оптимизирована программа Octave (рис. 1.9), и на этом же этапе выбрать пакеты расширений (параметр Octave Forge), которые будут установлены вместе с программой (рис. 1.10). В результате будет установлен текстовый редактор Niotepad++, интерпретатор Octave, пакеты расширений, англоязычная документация по Octave. Принципы работы с интерпретатором описаны ранее.

Существует и графическая оболочка (среда) для работы с Octave – Octave Workshop (рис. 1.6). Программу можно скачать с официального сайта

http://www.unige.ch/math/folks/loisel/www.math.mcgill.ca/loisel/octave-workshop/Octave.Workshop.Installer.exe.

Ее установка проходит стандартным для Windows способом. В состав программы Octave Workshop входит графическая оболочка, сам интерпретатор Octave и некоторые пакеты расширений.

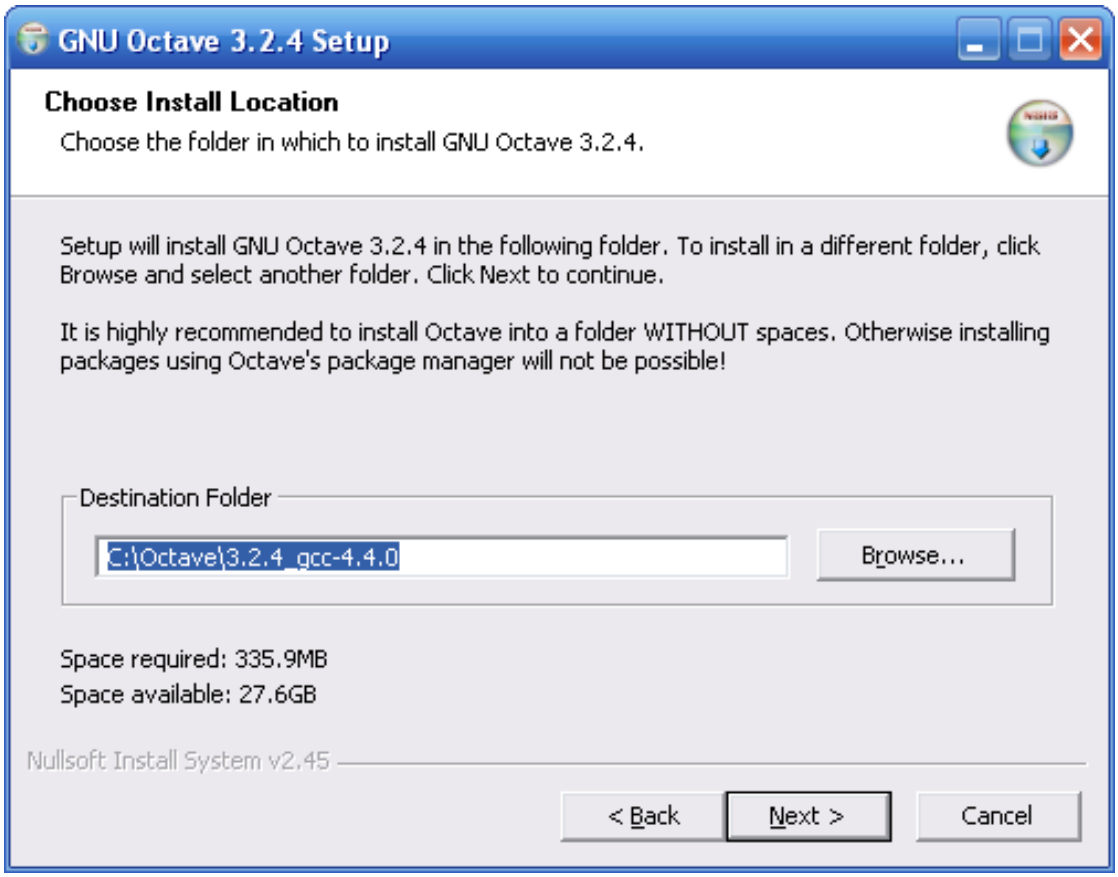


Рис. 1.8. Установка Octave. Выбор папки для установки.

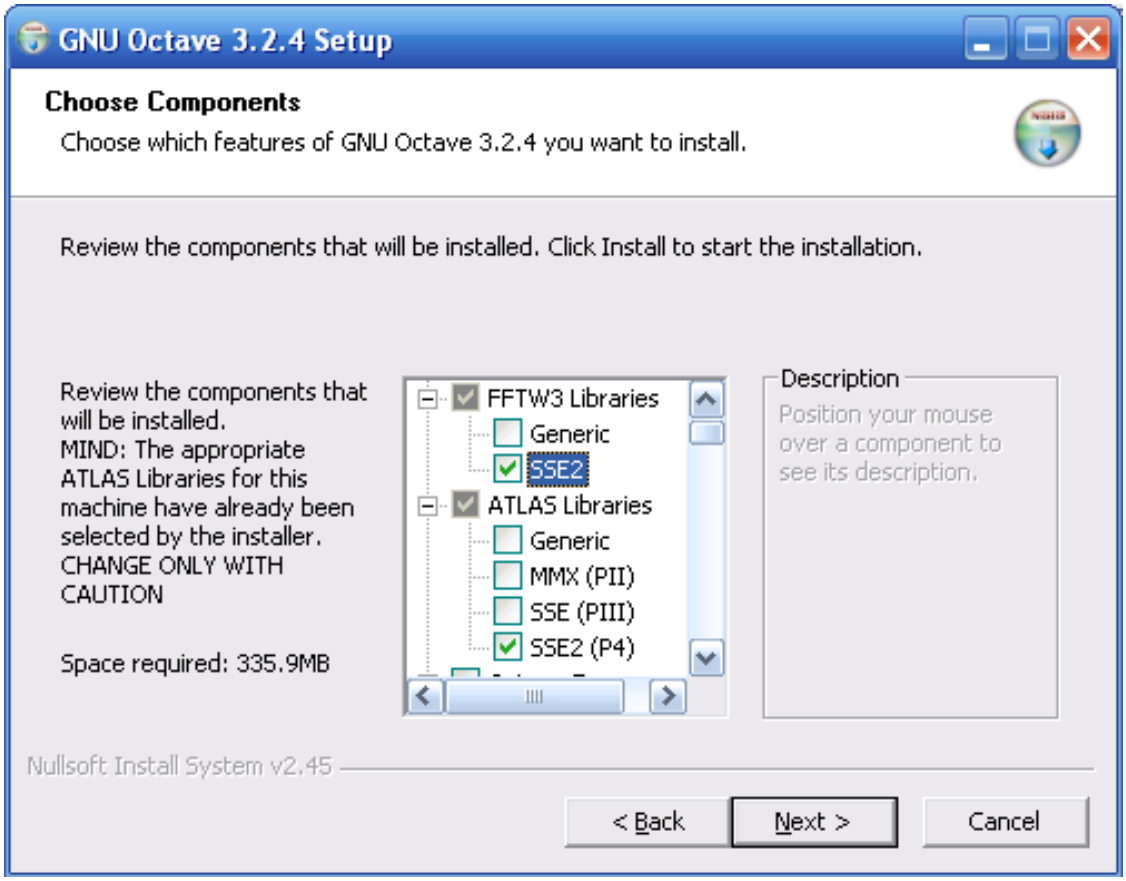


Рис. 1.9. Установка Octave. Выбор архитектуры процессора.

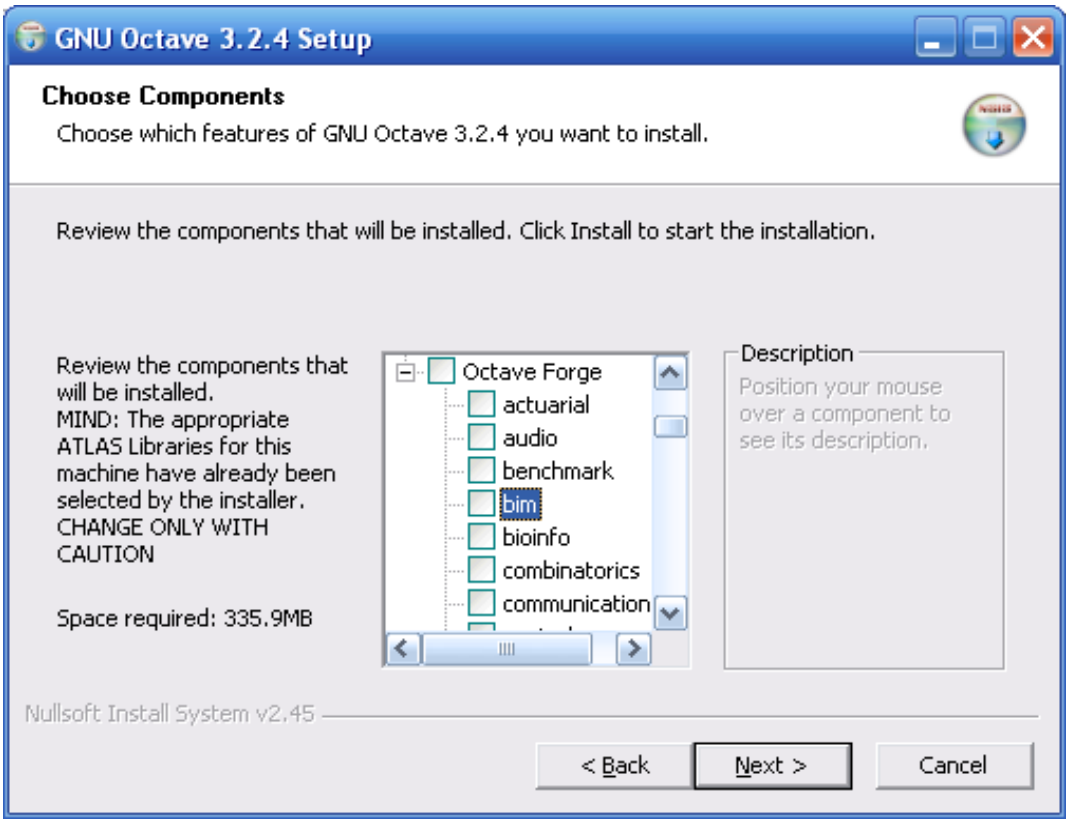


Рис. 1.10. Установка Octave. Выбор пакетов расширений.

Наиболее мощной графической оболочкой для работы с Octave является программа QtOctave, которая разработана для ОС Linux. Она портирована в ОС Linux в виде portable-версии, которую можно скачать по адресу http://qtoctave.wordpress.com/download (https://forja.rediris.es/frs/download.php/433/qtoctave0.6.8_octave2.9.15_Portable_win32.zip).

Программу нужно разархивировать и запустить файл qtoctave.exe. Окно QtOctave в ОС Windows представлено на рис.1.11.

1.2 Установка Octave в Linux

Математический пакет Octave разрабатывался для ОС Linux и поэтому именно в ОС Linux, пользователь получит возможность полноценно работать с Octave и использовать все возможности пакета.

Установка в современных дистрибутивах Linux осуществляется стандартным образом, например, через менеджер пакетов Synaptic (рис. 1.12). В менеджере пакетов Synaptic нужно щелкнуть по кнопке **Найти**, и в строке поиска ввести: *octave*. В результате поиска пользователю будет предложен список, в котором нужно выбрать *qtoctave* (графическая оболочка для работы с Octave), *octave3.2* (интерпретатор Octave), *octave3.0-doc* (документация по Octave на английском языке в формате pdf), *octave3.0-htmldoc* (документация по Octave на английском языке в формате html), а так же необходимые пользователю пакеты расширений (например, *octave-linear-algebra*, *octave-optim* и многие другие)[3]. Процесс установки начнется после щелчка по кнопке **Применить**. Время установки будет зависит от количества выбранных пакетов и скорости Интернет-соединения. После установки в группе программ **Программирование и Наука** появятся ярлыки программ **GNU Octave** (интерпретатор Octave) и **QtOctave** (графическая оболочка Octave).

3 Номера версий 0.6.8, 3.0, 3.2 в именах файлов или именах пакетов являлись текущими на момент написания книги. Когда книга выйдет номера могут быть другими.

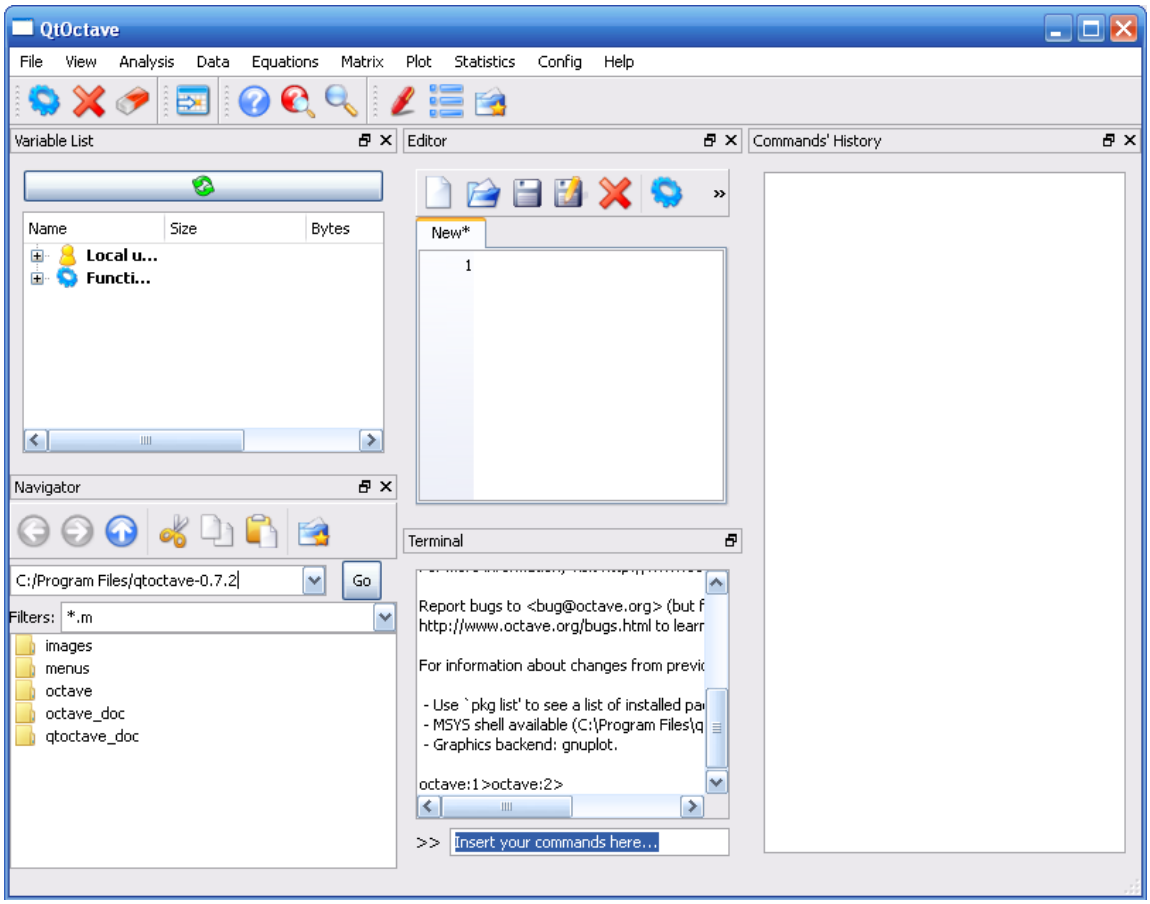


Рис. 1.11. Окно QtOctave под управлением ОС Windows

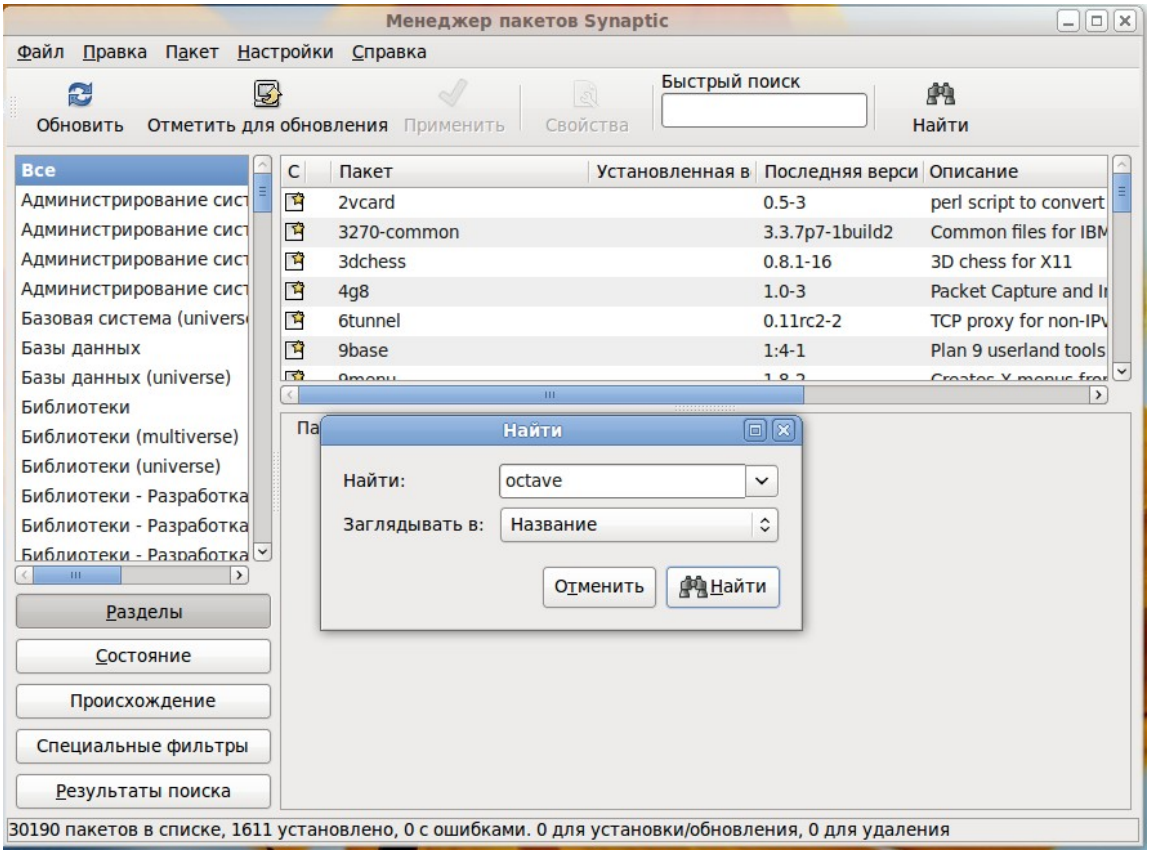


Рис. 1.12. Окно менеджера пакетов Synaptic

1.3 Графическая оболочка QtOctave

После запуска QtOctave на экране появляется основное окно приложения (рис. 1.13). Внешний вид окна зависит от настроек пользователя, но оно обязательно содержит *меню*, *панель инструментов, командную строку* и *рабочую область*[4] **Octave Terminal**.

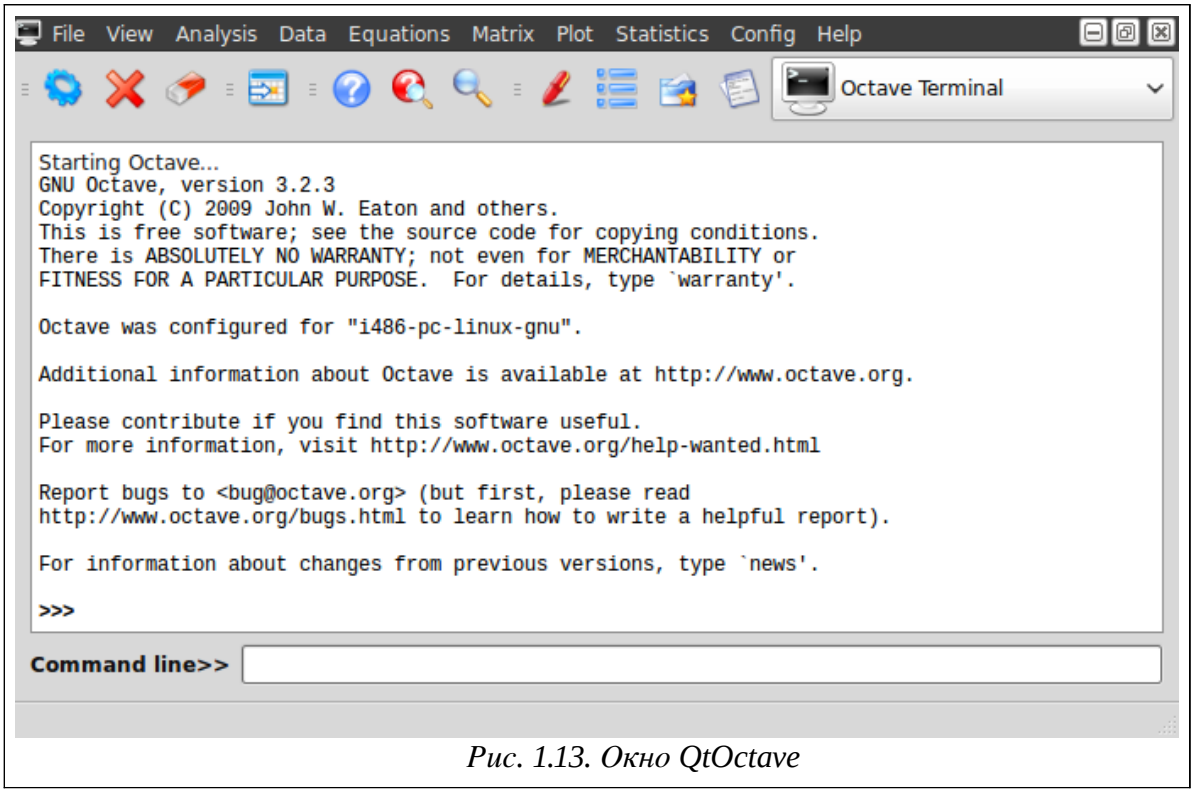


Рис. 1.13. Окно QtOctave

Признаком того, что система готова к работе, является наличие знака приглашения >>>. Ввод команд осуществляется с клавиатуры в командной строке **Command line**. Нажатие клавиши **Enter** заставляет систему выполнить команду и вывести результат, например, так как показано на рис. 1.14.

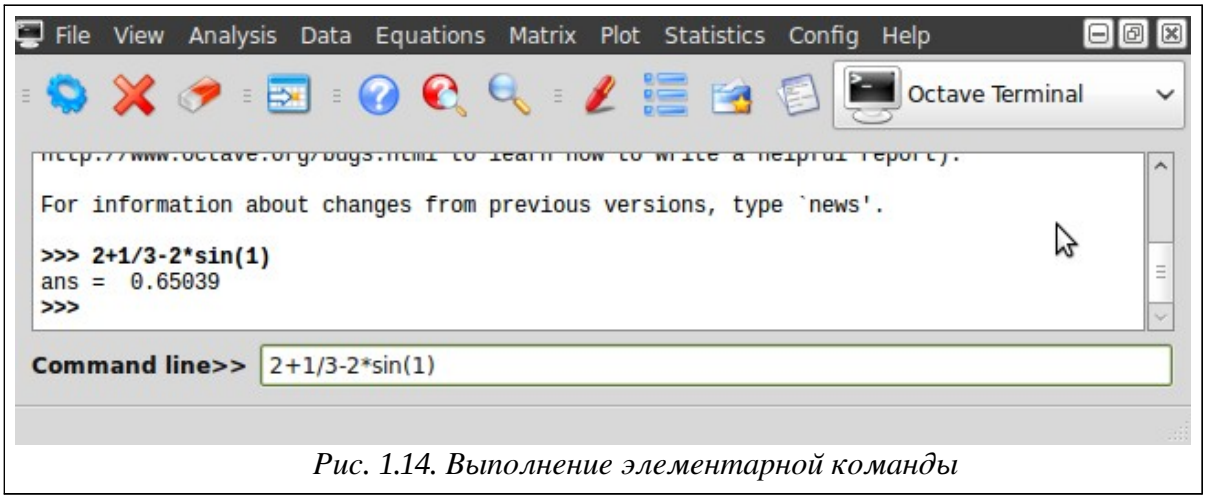


Рис. 1.14. Выполнение элементарной команды

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

4 Часть окна, в которой отображаются все выполняемые команды.

Page Down. В рабочей области нельзя ничего исправлять или вводить. Единственная допустимая операция это выделение информации с помощью мыши и копирование ее в буфер обмена, например, для дальнейшего помещения в командную строку.

В командной строке 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.

Клавиши ↑ и ↓ позволяют вернуть в командную строку ранее введенные команды или другую входную информацию, так как вся эта информация сохраняется в специальной области памяти. Так, если в пустой активной командной строке нажать клавишу ↑, то появится последняя вводимая команда, повторное нажатие вызовет предпоследнюю и так далее. Клавиша ↓ выводит команды в обратном порядке.

Кроме того, существуют особенности ввода команд. Если команда заканчивается точкой с запятой «`;`», то результат ее действия не отображается в рабочей области. В противном случае, при отсутствии знака «`;`», результат действия команды сразу же выводится в рабочую область (рис. 1.15).

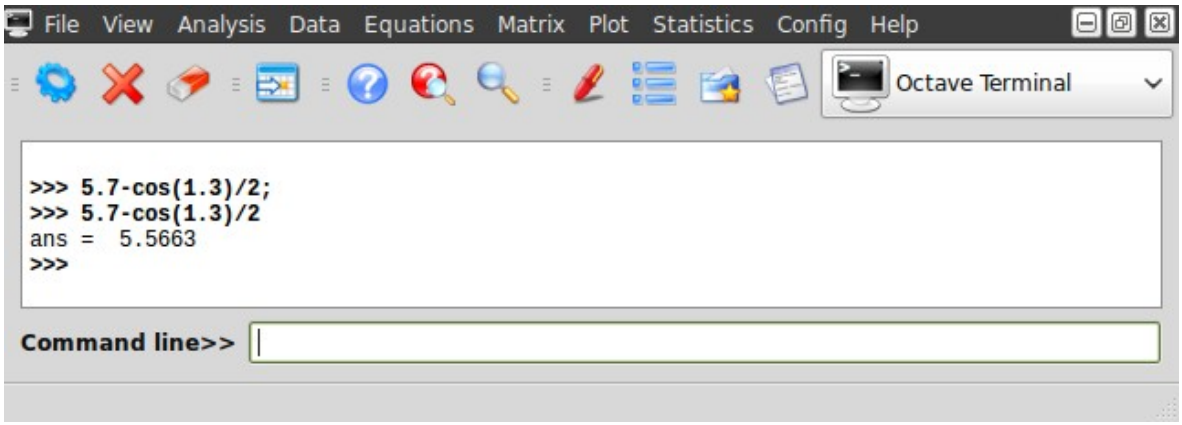


Рис. 1.15. Примеры вывода результатов вычислений

Работа в среде QtOctave может осуществляться в программном режиме. В этом случае в командной строке указывается имя программы, составленной из управляющих команд и функций Octave и имеющей расширение .m. Это достаточно удобный режим, так как он позволяет сохранить разработанный вычислительный алгоритм в виде файла и повторять его при других исходных данных и в других сеансах работы.

Выполнить команды Octave, хранящиеся в файле с расширением .**m** позволяет команда главного меню

File / Run an Octave Script.

Эта команда продублирована кнопкой в панели инструментов и открывает окно диалога, представленное на рис. 1.16.

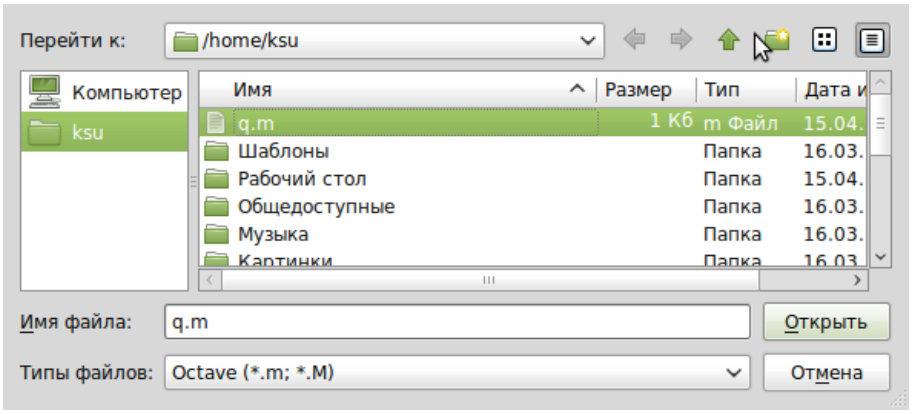


Рис. 1.16. Выбор файла для выполнения в Octave

Смена текущей директории осуществляется командой

File / Change Directory.

Команда также открывает диалоговое окно, предназначенное для выбора нового каталога.

Выход из программы выполняет команда

File / Quit.

Очистить рабочую область от введенных ранее команд можно, обратившись к пункту меню

View / Clear Nerminal.

Команда продублирована кнопкой в виде ластика на панели инструментов.

Окно со списком переменных, открывает команда

View / Dock Tools / Variable List.

Здесь (рис. 1.17) пользователю доступны значения всех переменных, вычисленные в течение текущей сессии. Они сохраняются в специально зарезервированной области памяти и при желании, определения всех переменных и функций, входящих в текущую сессию, можно сохранить на диске в виде файла.

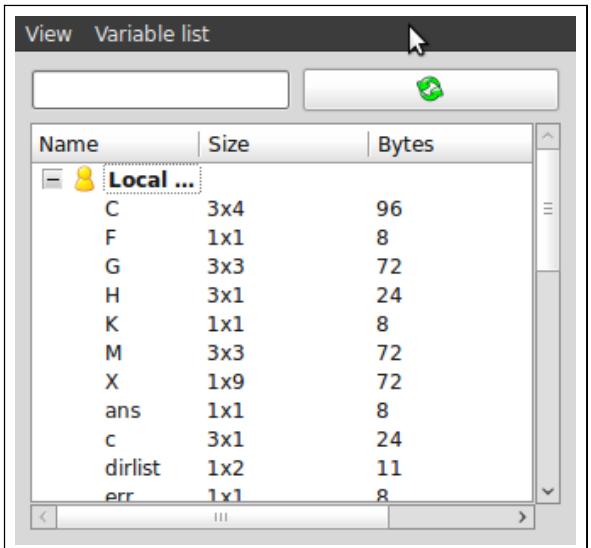


Рис. 1.17. Список переменных, определенных в процессе работы

Окно представленное на рис. 1.18 содержит список выполненных команд и открывается командой

View / Dock Tools / Command List.

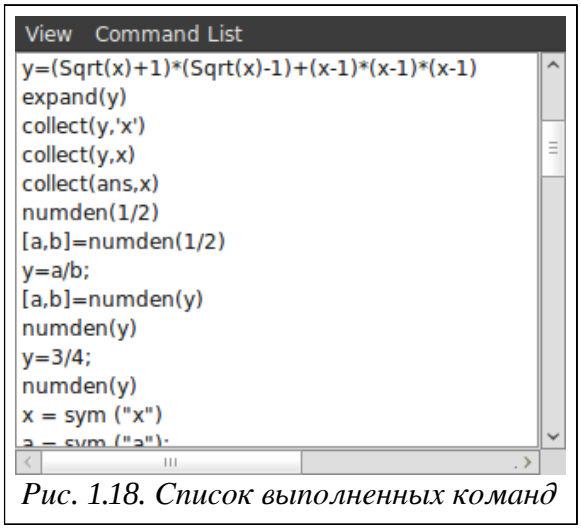


Рис. 1.18. Список выполненных команд

Выполнить поиск, просмотр, открытие файлов и каталогов, осуществить смену текущей директории, установить путь к файлу и так далее можно в окне показанном на рис.1.19. Это окно появится если выполнить команду

View / Dock Tools / Navigator.

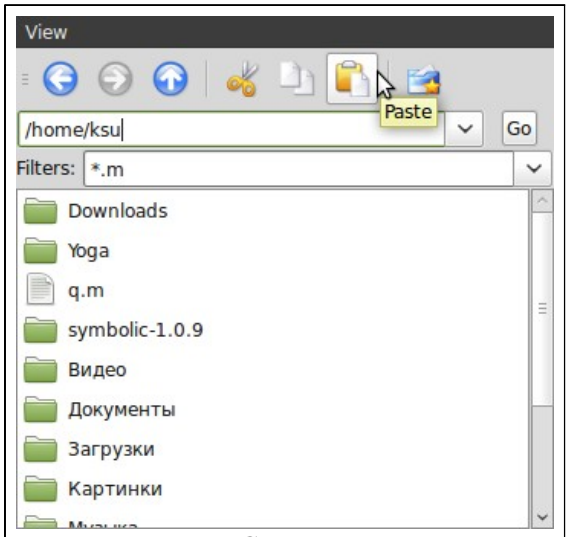


Рис. 1.19. Смена каталога

Текстовый редактор в QtOctave вызывает команда

View / Dock Tools / Editor.

Ввод текста в окно редактора осуществляется по правилам принятым для команд Octave. Рис. 1.20 содержит пример ввода команд для решения биквадратного уравнения $2x^4-9x^2+4=0$.

Для *сохранения введенной информации* необходимо выполнить команду

File / Save

из главного меню редактора. Если информация сохраняется впервые, то появится окно **Save file As…**.

Выполнить команды, набранные в текстовом редакторе, может команда меню редактора

Run / Run.

Кроме того, как было сказано выше, можно набрать имя созданного в текстовом редакторе файла в командной строке QtOctave и нажать ENTER.

Все эти действия приведут к появлению в рабочей области результатов вычислений, как видно на рис. 1.20.

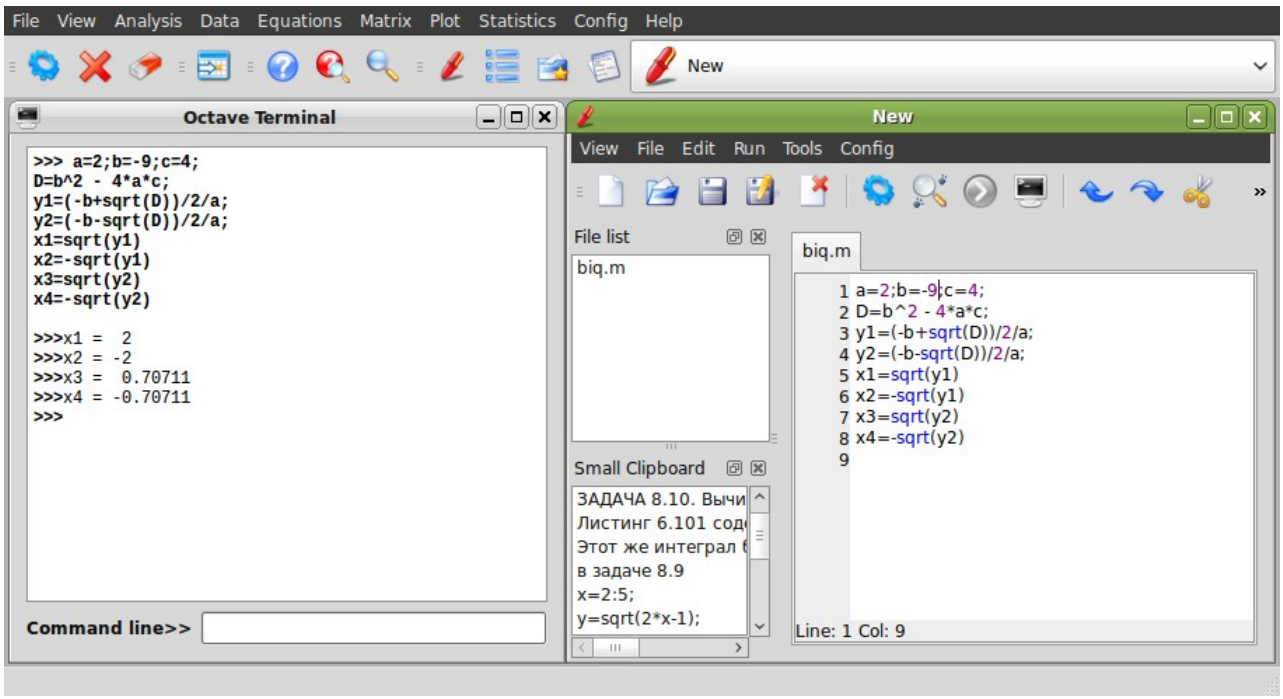


Рис. 1.20. Пример работы в текстовом редакторе

Текстовый редактор QtOctave имеет возможность работы с множеством окон и обладает принятыми для текстовых редакторов приемами редактирования, на которых подробно мы останавливаться не будем.

Выйти из режима редактирования, можно просто закрыв окно или командой **File / Close**. *Открывает ранее созданный файл* команда главного меню редактора **File / Open**.

Окна, представленные на рис. 1.17 - 1.20, обладают общим свойством. Команды **View / Show inside of main window** и **View / Show outside of main window** позволяют выводить окна внутри основного окна QtOctave (рис. 1.7) и за его пределами, соответственно.

Управлять положением окон в среде QtOctave можно командой **View / Windows Layout**. А команда **View / Show** позволяет отображать или удалять кнопки на панели инструментов.

Далее представлено краткое описание других пунктов главного меню QtOctave:

- **Analysis** – решение некоторых задач математического анализа (интегрирование и решение обыкновенных дифференциальных уравнений);
- **Data** – работа с матрицами (ввод, форматированный ввод, ввод из файла, запись в файл);
- **Equations** – решение линейных и нелинейных уравнений;
- **Matrix** – действия над матрицами (сложение, вычитание, умножение, транспонирование, инвертирование, вычисление определителя);
- **Plot** – работа с графикой (построение двумерных и трехмерных графиков, форматирование графической области, запись графического изображения в файл);
- **Statistics** – вычисление некоторых статистических функций;
- **Config** – настройка конфигурации системы, установка пакетов расширений;
- **Help** – справочная информация.

2. Основы работы в Octave

Умение выполнять элементарные математические операции, определять переменные, использовать встроенные функции системы и создавать собственные, работать с массивами данных — это азы работы в системе Octave.

2.1 Элементарные математические выражения в Octave

Простейшие арифметические операции в Octave выполняют с помощью следующих операторов:

\+ *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:

```
>>> 13.5/(0.2+4.2)^2-2.3
ans = -1.6027
```

Листинг 2.1

Обратите внимание, что при вводе вещественных чисел для отделения дробной части используется точка.

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

```
>>> 1+2*3-4....
>>> +5/6+7....
>>> -8+9
ans = 11.833
```

Листинг 2.2

В первой главе мы уже говорили о значении символа точки с запятой «`;`» в конце выражения. Напомним, что если он указан в конце выражения, то результат вычислений не выводится, а активизируется следующая командная строка:

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

Листинг 2.3

2.2 Текстовые комментарии

Правилом хорошего тона в программировании всегда считался «читаемый» программный код, снабженный большим количеством комментарием. Поскольку далее речь пойдет о достаточно сложных вычислениях, для наглядности мы так же будем применять текстовые комментарии.

Текстовый комментарий в Octave это строка, начинающаяся с символа `%`. Использовать текстовые комментарии можно как в командной строке, так и в тексте программы. Строка после символа `%` не воспринимается как команда и нажатие клавиши Enter приводит к активизации следующей командной строки.

Примеры использования текстовых комментариев присутствуют уже в следующем параграфе.

2.3 Представление вещественного числа в Octave

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

$6.42E+2 = 6.42.10^{2} = 642$

$-3.2E-6 = -3.2.10^{-6} = -0.0000032$

В листинге 2.4 приведен пример ввода вещественного числа.

```
>>> 0.987654321
ans =  0.98765
```

Листинг 2.4

Не трудно заметить, что число знаков в дробной части числа в строке ввода больше чем в строке вывода. Происходит это потому, что результат вычислений выводится в виде, который определяется предварительно установленным форматом представления чисел.

Команда, с помощью которой можно установить формат числа имеет вид:

```
format формат числа;
```

В Octave предусмотрены следующие форматы чисел:

- `Short` – краткая запись, применяется по умолчанию;
- `Long` – длинная запись;

```
>>> format short
>>> pi
ans =  3.1416
>>> format long
>>> pi
ans =  3.14159265358979
```

Листинг 2.5

- `Short E` (`Short e`) – краткая запись в формате с плавающей точкой;
- `Long E` (`Long e`) – длинная запись в формате с плавающей точкой;

```
>>> format short E
>>> pi
ans =  3.1416E+00
>>> format long E
>>> pi
ans =  3.14159265358979E+00
```

Листинг 2.6

- `Short G` (`Short g`) – вторая форма краткой записи в формате с плавающей точкой;
- `Long G` (`Long g`) – вторая форма длинной записи в формате с плавающей точкой;

```
>>> format short G
>>> pi
ans =     3.1416
>>> format long G
>>> pi
ans =     3.14159265358979
```

Листинг 2.7

- `Hex` – запись в виде шестнадцатеричного числа;
- `native-Hex` – запись в виде шестнадцатеричного числа, в таком виде, в каком оно хранится в памяти компьютера;
- `Bit` – запись в виде двоичного числа;
- `native-Bit` – запись в виде двоичного числа, в таком виде, в каком оно хранится в памяти компьютера;

```
>>> format native-hex
>>> pi
ans = 182d4454fb210940
>>> format hex
>>> pi
ans = 400921fb54442d18
>>> format bit
>>> pi
ans =
0100000000001001001000011111101101010100010001000010110100011000
>>> format native-bit
>>> pi
ans =
0001100010110100001000100010101011011111100001001001000000000010
```

Листинг 2.8

- `Bank` – запись до сотых долей;
- `Plus` – записывается только знак числа;

```
>>> format bank
>>> pi
ans = 3.14
>>> format +
>>> pi
ans = +
>>> -pi
ans =
```

Листинг 2.9

- `Free` – запись без форматирования, чаще всего этот формат применяют для представления комплексного числа (подробно о комплексных числах см. п. 2.5.2);

```
>>> format short
>>> 3.1234+2.9876*i
ans = 3.1234 + 2.9876i
>>> format free
>>> 3.1234+2.9876*i
ans = (3.123,2.988)
```

Листинг 2.10

- `Compact` – запись в формате не превышающем шесть позиций, включая десятичную точку, если целая часть числа превышает четыре знака, число будет записано в экспоненциальной форме.

```
>>> format compact
>>> 123.123456
```

```
ans =  123.12
>>> 1234.12345
ans =  1234.1
>>> 12345.123
ans =  1.2345e+04
```

Листинг 2.11

Обратите внимание, что формат `Short` установлен по умолчанию. Вызов команды `format` с другим числовым форматом означает, что теперь вывод чисел будет осуществляться в установленном формате.

2.4 Переменные в Octave

В Octave можно определять переменные и использовать их в выражениях. 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. То есть, если в общем виде оператор присваивания записать как

`имя_переменной = выражение`,

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

`Имя_переменной` не должно совпадать с именами встроенных процедур, функций и встроенных переменных системы. Система различает большие и малые буквы в именах переменных. То есть `ABC`, `abc`, `Abc`, `aBc` – это имена разных переменных.

Выражение в правой части оператора присваивания может быть числом, арифметическим выражением, строкой символов или символьным выражением. Если речь идет о символьной или строковой переменной, то выражение в правой части оператора присваивания следует брать в одинарные кавычки.

```
>>>%Определение переменной
>>> x=1.2
x = 1.2000
>>> y=3.14
y = 3.1400
>>>%---------------------------------
>>>%Использование переменных в арифметическом выражении
>>> z=x+y
z = 4.3400
>>>%---------------------------------
>>>%Определение символьной переменной
>>> s='a'
s = a
>>>%---------------------------------
>>>%Определение строковой переменной
>>> str='Посадил дед репку. Выросла репка большая, пребольшая'
str = Посадил дед репку. Выросла репка большая, пребольшая
```

Листинг 2.12

Несколько примеров присвоения значений переменным приведено в листинге 2.13 Обратите внимание, что если символ «`;`» в конце выражения отсутствует, то в качестве результата выводится имя переменной и ее значение. Наличие символа «`;`» передает управление следующей командной строке. Это позволяет использовать имена переменных для записи промежуточных результатов в память компьютера.

```
>>> a=1;b=2;c=a*b;d=c^2
d =  4
```

Листинг 2.13

Листинг 2.14 содержит пример использования в выражении ранее неопределенной переменной.

```
>>>%Сообщение об ошибке. Переменная не определена.
>>> t/(a+b) error: 't' undefined near line 37 column 1
```

Листинг 2.14

Если команда не содержит знака присваивания, то по умолчанию вычисленное значение присваивается специальной системной переменной `ans`. Причем полученное значение можно использовать в последующих вычислениях, но важно помнить, что значение `ans` изменяется после каждого вызова команды без оператора присваивания. Примеры использования системной переменной `ans` можно увидеть в листинге 2.15.

```
>> >25-3
ans = 22
>>> %Значение системной переменной равно 22
>>> 2*ans
ans = 44
>> >%Значение системной переменной равно 44
>> x=ans^0.5
x =    6.6332
>> %Значение системной переменной не изменилось и равно 44
>> ans
ans =   44
```

Листинг 2.15

Кроме переменной `ans` в Octave существуют и другие системные переменные:

`i, j` – мнимая единица ($\sqrt{-1}$);

`pi` – число π (3.141592653589793);

`e` – число *e* (экспонента 2.71828183)

`inf` – машинный символ бесконечности (∞);

`NaN` – неопределенный результат ($\frac{0}{0}$, $\frac{\infty}{\infty}$, 1^{∞} и т.п.);

`realmin` – наименьшее число с плавающей точкой (2.2251e-308);

`realmax` – наибольшее число с плавающей точкой (1.7977e+308);

Все перечисленные переменные можно использовать в математических выражениях.

Если речь идет об уничтожении определения одной или нескольких переменных, то можно применить команду:

```
clear имя_переменной
```

Пример применения команды `clear` приведен с листинге 2.16.

```
 >>> a=5.2;b=a^2;
 >>> a,b
 a = 5.2000
 b = 27.040
 >>> clear a
 >>> a,b
error: 'a' undefined near line 126 column 1
 >>> b
 b = 27.040
```

Листинг 2.16

2.5 Функции в Octave

Все функции, используемые в Octave, можно разделить на два класса *встроенные* и *определенные пользователем*.

В общем виде обращение к функции в Octave имеет вид:

```
имя_переменной = имя_функции(аргумент)
```

или

```
имя_функции(аргумент)
```

Если `имя_переменной` указано, то ей будет присвоен результат работы функции. Если же оно отсутствует, то значение вычисленного функцией результата присваивается системной переменной `ans`.

Например,

```
>>> x=pi/2; %Определение значения аргумента
>>> y=sin(x)%Вызов функции
y =  1
>>> cos(pi/3)%Вызов функции
ans =  0.50000
```

Листинг 2.17

Рассмотрим элементарные встроенные функции Octave. С остальными будем знакомиться по мере изучения материала.

2.5.1 Элементарные математические функции

В табл. 2.1 представлены тригонометрические функции Octave.

Таблица 2.1. Тригонометрические функции

Функция	Описание функции
`sin(x)`	синус числа x
`cos(x)`	косинус числа x
`tan(x)`	тангенс числа x
`cot(x)`	котангенс числа x
`sec(x)`	секанс числа x
`csc(x)`	косеканс числа x
`asin(x)`	арксинус числа x
`acos(x)`	арккосинус числа x
`atan(x)`	арктангенс числа x
`acot(x)`	арккотангенс числа x
`asec(x)`	арксеканс числа x
`acsc(x)`	арккосеканс числа x

Примеры работы с тригонометрическими функциями:

```
>>> x=pi/7;
>>> sin(x)
ans = 0.43388
>>> (1-cos(x)^2)^0.5
ans = 0.43388
>>> tan(x)/(1+tan(x)^2)^0.5
ans = 0.43388
>>> (sec(x)^2-1)^0.5/sec(x)
ans = 0.43388
>>> 1/csc(x)
```

```
ans = 0.43388
>>> asin(x)
ans = 0.46542
>>> acos((1-x^2)^0.5)
ans = 0.46542
>>> atan(x/((1-x^2)^0.5))
ans = 0.46542
```

Листинг 2.18

Экспоненциальные функции представлены в табл. 2.2.

Таблица 2.2. Экспоненциальные функции

Функция	Описание функции	Функция	Описание функции
`exp(x)`	Экспонента числа x	`log(x)`	Натуральный логарифм числа x

Применение экспоненциальных функций:

```
>>> x=1;
>>> exp(x)
ans =  2.7183
>>> log(x)
ans = 0
>>> log(e^2)
ans =  2
```

Листинг 2.19

Таблица 2.3 содержит гиперболические функции.

Таблица 2.3. Гиперболические функции

Функция	Описание функции
`sinh(x)`	гиперболический синус числа x
`cosh(x)`	гиперболический косинус числа x
`tanh(x)`	гиперболический тангенс числа x
`coth(x)`	гиперболический котангенс числа x
`sech(x)`	гиперболический секанс числа x
`csch(x)`	гиперболический косеканс числа x

Примеры работы с гиперболическими функциями.

```
>>> cosh(x)^2-sinh(x)^2
ans =  1
>>> tanh(x)*coth(x)
ans =  1
```

Листинг 2.20

В таблице 2.4 представлены целочисленные функции.

Таблица 2.4. Целочисленные функции

Функция	Описание функции
`fix(x)`	округление числа x до ближайшего целого в сторону нуля
`floor(x)`	округление числа x до ближайшего целого в сторону отрицательной бесконечности
`ceil(x)`	округление числа x до ближайшего целого в сторону положительной бесконечности
`round(x)`	обычное округление числа x до ближайшего целого
`rem(x, y)`	вычисление остатка от деления x на y
`sign(x)`	сигнум–функция числа x, выдает 0, если x=0 , –1 при x<0 и 1 при x>0

Примеры работы с тригонометрическими функциями:

```
>>> pi
ans =  3.1416
>>> fix(pi)
ans =  3
>>> floor(pi)
ans =  3
>>> floor(-pi)
ans = -4
>>> ceil(pi)
ans =  4
>>> ceil(-pi)
ans = -3
>>> round(pi)
ans =  3
>>> pi/2
ans =  1.5708
>>> round(pi/2)
ans =  2
>>> rem(5,2)
ans =  1
>>> sign(0)
ans = 0
>>> sign(pi)
ans =  1
>>> sign(-pi)
ans = -1
```

Листинг 2.21

Другие элементарные функции представлены в табл. 2.5.

Таблица 2.5. Другие элементарные функции

Функция	Описание функции
`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
`pow2(x)`	возведение двойки в степень x
`gcd(x, y)`	наибольший общий делитель чисел x и y
`lcm(x, y)`	наименьшее общее кратное чисел x и y
`rats(x)`	представление числа x в виде рациональной дроби

Далее приведены примеры работы с функциям из таблицы 2.5.

```
>>> x=9;
>>> sqrt(x)
ans =  3
>>> abs(-x)
ans =  9
>>> abs(x)
ans =  9
>>> x=10;
>>> log10(x)
```

```
ans =  1
>>> log10(10*x)
ans =  2
>>> x=4;
>>> log2(x)
ans =  2
>>> pow2(x)
ans =  16
>>> x=8;y=24;
>>> gcd(x,y)
ans =  8
>>> lcm(x,y)
ans =  24
>>> rats(pi)
ans =   355/113
>>> rats(e)
ans = 2721/1001
```

Листинг 2.22

2.5.2 Комплексные числа. Функции комплексного аргумента

Рассмотрим реализацию комплексной арифметики в Octave. Как было отмечено выше, для обозначения мнимой единицы зарезервировано два имени – `i`, `j`, поэтому ввод комплексного числа производится в формате:

```
действительная_часть + i * мнимая_часть
```

или

```
действительная_часть + j * мнимая_часть
```

Пример ввода и вывода комплексного числа:

```
>>> 3+i*5
ans =  3 + 5i
>>> -2+3*i
ans = -2 + 3i
>>> 7+2*j
ans =  7 + 2i
>>> 0+7i
ans =  0 + 7i
>>> 6+0*j
ans =  6
```

Листинг 2.23

Кроме того, к комплексным числам применимы элементарные арифметические операции: +, –, *, \, /, ^ (листинг 2.24).

```
>>> a=-5+2i;b=3-5*i;
>>> a+b
ans = -2 - 3i
>>> a-b
ans = -8 + 7i
>>> a*b
ans =  -5 + 31i
>>> a/b
ans = -0.73529 - 0.55882i
>>> a^2+b^2
```

```
ans =   5 - 50i
```

Листинг 2.24

Функции для работы с комплексными числами приведены в таблице 2.6.

Таблица 2.6. Функции работы с комплексными числами

Функция	Описание функции
`real(Z)`	выдает действительную часть комплексного аргумента Z
`imag(Z)`	выдает мнимую часть комплексного аргумента Z
`angle(Z)`	вычисляет значение аргумента комплексного числа Z в радианах от –π до π
`conj(Z)`	Выдает число комплексно сопряженное Z

Примеры использования функций из табл. 2.6:

```
>>> a=-3;b=4;Z=a+b*i
Z = -3 + 4i
>>> real(Z)
ans = -3
>>> imag(Z)
ans =  4
>>> angle(Z)
ans =  2.2143
>>> conj(Z)
ans = -3 - 4i
```

Листинг 2.25

Обратите внимание, что большая часть математических функций, описанных в п. 2.5.1 работают с комплексным аргументом:

```
>>> a=-3;b=4;Z=a+b*i
Z = -3 + 4i
>>> sin(Z)
ans =  -3.8537 - 27.0168i
>>> exp(Z)
ans = -0.032543 - 0.037679i
>>> sqrt(Z)
ans =  1 + 2i
>>> abs(Z)
ans =  5
```

Листинг 2.26

2.5.3 Операции отношения в Octave

Операции отношения выполняют сравнение двух операндов и определяют, истинно выражение или ложно (табл. 2.7). Результат операции отношения – логическое значение. В качестве *логических значений* в Octave используются 1 («истина») и 0 («ложь»).

Таблица 2.7. Операции отношения.

Операция	Описание операции
<	меньше
>	больше
==	равно
~=	не равно
<=	меньше или равно
>=	больше или равно

2.5.4 Логические выражения в Octave

Логическое выражение может быть составлено из операций отношения и логических операций (операторов). Логические выражения выполняются над логическими данными. В табл. 2.8 представлены основные логические выражения: «и», «или», «не».

Таблица 2.8. Логические операции

A	B	«не» A	A «и» B	A «или» B	A «исключающее или» B
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	0	1	0

В Octave существует возможность представления логических выражений в виде логических операторов и логических операций (табл. 2.9).

Таблица 2.9. Виды логических выражений в Octave

Тип выражения	Выражение	Логический оператор	Логическая операция
Логическое «и»	`A and B`	`and(A, B)`	`A & B`
Логическое «или»	`A or B`	`or(A, B)`	`A | B`
Исключающее «или»	`A xor B`	`xor(A,B)`	
Отрицание	`not A`	`not (A)`	`~ A`

В таблице 2.9 `A` и `B` – логические выражения (или целочисленные значения 1 и 0).

Логические операторы (выражения) определены и над массивами (матрицами) одинаковой размерности и выполняются поэлементно над элементами. В итоге формируется результирующий массив (матрица) логических значений, каждый элемент которого вычисляется путем выполнения логической операции над соответствующими элементами исходных массивов.

При одновременном использовании в выражении логических и арифметических операций возникает проблема последовательности их выполнения. В Octave принят следующий *приоритет операций*.

1. Логические операторы.
2. Логическая операция ~.
3. Операции возведения в степень, унарный + и −.
4. Умножение, деление
5. Сложение, вычитание.
6. Операции отношения.
7. Логическая операция «и» – `&`
8. Логическая операция «или» – `|`

В листинге 2.27 представлены примеры использования логических операций над скалярными и матричными значениями.

```
>>> A=3;B=4;C=2*pi;
>>> P=~(((A+B)*C)>A^B)|(A+B)==((B-A)+A^2)
P =  1
>>> X=[2 1 6];Y=[1 3 5];
>>> Z=~or((X>2*Y),(X>Y))|and((X>2*Y),(X<Y))
Z =
   0   1   0
```

Листинг 2.27

2.5.5 Функции, определенные пользователем

В первой главе мы уже создавали небольшую программу, которая решала конкретное квадратное уравнение. В этой программе отсутствовал заголовок (первая строка определенного вида) и в нее невозможно было передать входные параметры, то есть это был обычный список команд, воспринимаемый системой как единый оператор.

Функция, как и программа, предназначена для неоднократного использования, но она имеет входные параметры и не выполняется без их предварительного задания. Функция имеет заголовок вида

```
function [name1 [, name2, …]] = fun(var1 [, var2, …])
```

где `name1` [, `name2`, …] – список выходных параметров, то есть переменных, которым будет присвоен конечный результат вычислений, `fun` – имя функции, `var1` [, `var2`, …] – входные параметры. Таким образом, простейший заголовок функции выглядит так:

```
function name = fun(var)
```

Все имена переменных внутри функции, а так же имена из списка входных и выходных параметров, воспринимаются системой как локальные, то есть эти переменные считаются определенными только внутри функции.

Программы и функции в Octave могут быть созданы при помощи текстового редактора и сохранены в виде файла с расширением .m или .M. Но при создании и сохранении функции следует помнить, что ее имя должно совпадать с именем файла.

Вызов программ в Octave осуществляется из командной строки. Программу можно запустить на выполнение, указав имя файла, в котором она сохранена.

Обращение к функции осуществляется так же, как и к любой другой встроенной функции системы, то есть с указанием входных и выходных параметров. Вызвать функцию можно из командной строки или использовать ее как один из операторов программы.

ЗАДАЧА 2.1. Создать функцию для решения кубического уравнения.

Кубическое уравнение

$$ax^3+bx^2+cx+d=0 \qquad (2.1)$$

после деления на a принимает канонический вид:

$$x^3+rx^2+sx+t=0 \quad , \qquad (2.2)$$

где

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

В уравнении (2.2) сделаем замену

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

и получим следующее приведенное уравнение:

$$y^3+py+q=0 \quad , \qquad (2.3)$$

где

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

Число действительных корней приведенного уравнения (2.3) зависит от знака дискриминанта $D=\frac{p^3}{27}+\frac{q^3}{8}$ (табл. 2.10).

Таблица 2.10. Количество корней кубического уравнения

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

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

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

Здесь

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

Далее представлен текст функции, реализующей описанный выше способ решения кубического уравнения (листинг 2.28) и результат ее работы (листинг 2.29).

```
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
```

Листинг 2.28

```
>>> [x1,x2,x3]=cub(3,-2,-1,-4)
x1 =  1.4905
x2 = -0.41191 + 0.85141i
x3 = -0.41191 - 0.85141i
```

Листинг 2.29

Редактирование и отладка файлов описана в первой главе.

2.6 Массивы в Octave

Как правило массивы используют для хранения и обработки множества однотипных данных. Вместо создания переменной, для хранения каждого данного, создают один массив, где каждый элемент имеет порядковый номер. Таким образом, *массив* – множественный тип данных, состоящий из фиксированного числа элементов одного типа. Как и любой другой переменной, массиву должно быть присвоено *имя*.

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

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

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

Самый простой способ задать одномерный массив в Octave имеет вид

```
имя_массива = Xn:dX:Xk
```

где Xn – значение первого элемента массива, Xk – значение последнего элемента массива, dX – шаг, с помощью которого формируется каждый следующий элемент массива, то есть значение второго элемента составит Xn+ dX, третьего Xn+ dX+ dX и так далее до Xk.

Если параметр dX в конструкции отсутствует:

```
имя массива = Xn:Xk
```

это означает, что по умолчанию он принимает значение равное единице, то есть каждый следующий элемент массива равен значению предыдущего плюс один.

Примеры создания массивов приведены в листинге 2.30.

```
>>> A=1:5
A =
1 2 3 4 5
>>> B=2:2:10
B =
2 4 6 8 10
>>> xn=-3.5;xk=3.5;dx=0.5;
>>> X=xn:dx:xk
X =
Columns 1 through 8:
-3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0
Columns 9 through 15:
0.5 1.0 1.5 2.0 2.5 3.0 3.5
```

Листинг 2.30

Переменную заданную как массив можно использовать в арифметических выражениях и в качестве аргумента математических функций. Результатом работы таких операторов являются массивы (листинг 2.31).

```
>>> xn=-3.5;xk=3.5;dx=0.5;
>>> X=xn:dx:xk;
>>> Y=cos(X/2)
Y =
 Columns 1 through 7:
-0.1782 0.0707 0.3153 0.5403 0.7316 0.8775   0.9689
 Columns 8 through 14:
1.0 0.9689 0.8775 0.7316 0.5403 0.3153 0.0707 -0.1782
>>> B=2:2:10;C=sqrt(B)
C =
   1.4142   2.0000   2.4495   2.8284   3.1623
>>> -2:2
ans =
  -2  -1   0   1   2
>>> ans*2-pi/2
ans =    -5.5708   -3.5708   -1.5708    0.4292    2.4292
```

Листинг 2.31

Векторы и матрицы в Octave можно вводить поэлементно. Так для определения *вектора–строки* следует ввести имя массива, а затем после знака присваивания, в квадратных скобках через пробел или запятую перечислить элементы массива:

```
>>> x=[2 4 6 8 10]
x =     2    4    6    8   10
>>> y=[-1.2 3.4 -0.8 9.1 5.6 -7.3]
y = -1.20000   3.40000  -0.80000   9.10000   5.60000  -7.30000
```

Листинг 2.32

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

```
>>> X=[1;3;5;7;9]
X =
   1
   3
   5
   7
   9
```

Листинг 2.33

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

```
>>> x=[2 4 6 8 10];
>>> y=[-1.2 3.4 -0.8 9.1 5.6 -7.3];
>>> x(1)%значение первого элемента массива x
ans =  2
>>> y(5)%значение пятого элемента массива y
ans =  5.6000
>>> x(1)/2+y(3)^2-x(4)/y(5)
ans =  0.21143
```

Листинг 2.34

Ввод элементов матрицы так же осуществляется в квадратных скобках, при этом элементы строки отделяются друг от друга пробелом или запятой, а строки разделяются между собой точкой с запятой. *Обратиться к элементу матрицы* можно, указав после имени матрицы, в круглых скобках, через запятую, номер строки и номер столбца на пересечении которых элемент расположен. Примеры задания матриц и обращение к их элементам показаны в листинге 2.35.

```
>>> M=[2 4 6;1 3 5;7 8 9]
M =
   2   4   6
   1   3   5
   7   8   9
>>> M(1,2)
ans =  4
>>> M(3,1)
ans =  7
>>> M(2,2)/2+M(3,3)^0.5-M(1,1)*5
ans = -5.5000
```

Листинг 2.35

Подробно работа с векторами и матрицами описана в пятой главе.

2.7 Символьные вычисления в Octave

Символьные вычисления в Octave поддерживает специальный пакет расширений **octave-symbolic**. Процедура установки пакетов расширений описана в первой главе. Если пакет уже установлен, то перед началом работы его нужно загрузить командой[5]

```
pkg load symbolic
```

Теперь можно использовать любые функции из пакета **symbolic**.

Оператор `symbols` *инициализирует символические операции*, с этого оператора должны начинаться любые действия в символьных переменных.

Работа с символьными переменными в Octave требует их специального *объявления*:

5 Команда `pkg load` загружает и другие пакеты расширений.

```
sym('имя_переменной')
```

Например, команда `x = sym ("x")` объявляет символьную переменную `x`.

ЗАДАЧА 2.2. Выполнить арифметические операции с символьными переменными $z = x \cdot y, t = \frac{x^3}{z}$, где $x = a + b, y = a^2 - b^2$ (листинг 2.36).

```
>>>%Инициализация символьных операций
>>>symbols
>>>%Объявление символьных переменных
>>> x = sym ("x");
>>> y = sym ("y");
>>> z = sym ("z");
>>> t = sym ("t");
>>> a = sym ("a");
>>> b = sym ("b");
%Вычисление символьных выражений
>>> x=a+b
x =
a+b
>>> y=a^2-b^2
y =
-b^(2.0)+a^(2.0)
>>> z=x*y
z =
-(b^(2.0)-a^(2.0))*(a+b)
>>> t=x^3/z
t =
-(b^(2.0)-a^(2.0))^(-1)*(a+b)^(2.0)
```

Листинг 2.36

Символьные вычисления в Octave предусматривают работу с элементарными математическими функциями (табл. 2.11).

Таблица 2.11. Функции в символьных вычислениях

Функция	Описание функции
`Sin(x)`	синус числа x
`Cos(x)`	косинус числа x
`Tan(x)`	тангенс числа x
`aSin(x)`	арксинус числа x
`aCos(x)`	арккосинус числа x
`aTan(x)`	арктангенс числа x
`Log(x)`	натуральный логарифм числа x
`Exp(x)`	экспонента числа x (e^x)
`Sqrt(x)`	корень квадратный из числа x ($\sqrt{x}$)
`Pi`	число π

Вычислить *значение символьного выражения* при заданном значении переменной можно с помощью функции

```
subs(выражение,имя_переменной,значение_переменной)
```

ЗАДАЧА 2.3. Вычислить значение выражения $y = \sin(\alpha)^2 - \cos(\alpha)^2$, при $\alpha_1 = \frac{\pi}{3}, \alpha_2 = \frac{\pi}{6}$ (листинг 2.37).

```
>>> x = sym ("x") ;
>>> y = sym ("y") ;
>>> y=Sin(x)^2-Cos(x)^2
y =
-cos(x)^(2.0)+sin(x)^(2.0)
%Значение выражения при заданном значении переменной
>>> subs(y,x,Pi/3)
ans =
0.50000000000000001
>>> subs(y,x,Pi/6)
ans =
-0.49999999999999999
```

Листинг 2.37

Преобразовать символьное выражение, представить его в виде элементарных функций, возможно командой

```
expand(выражение)
```

ЗАДАЧА 2.4. Раскрыть скобки в выражении $y=(\sqrt{x}+1)(\sqrt{x}-1)+(x-1)^3$ (листинг 2.38).

```
>>> y=(Sqrt(x)+1)*(Sqrt(x)-1)+(x-1)*(x-1)*(x-1)
y =
(-1.0+x)^3+(-1.0+sqrt(x))*(1.0+sqrt(x))
>>> expand(y)
ans =
-2.0+(4.0)*x+x^3-(3.0)*x^2
```

Листинг 2.38

Далее, по ходу изложения материала, будут рассмотрены операции с матрицами символов, решение систем линейных уравнений в символьных переменных (п. 5.9), решение нелинейных уравнений и систем (п. 7.4), дифференцирование (п. 8.1).

3. Программирование в Octave

3.1 Основные операторы языка программирования Octave.

Рассмотренные в предыдущих главах группы команд, состоящие из операторов присваивания и обращения к встроенным функциям, представляют собой *простейшие программы* Octave. Если такая программа хранится в файле с расширением .m (.M), то для ее выполнения достаточно в командной строке Octave ввести имя этого файла (без расширения). В Octave встроен достаточно мощный язык программирования. Рассмотрим основные операторы этого языка и примеры их использования.

3.1.1 Оператор присваивания

Оператор присваивания служит для определения переменной (п. 2.2). Для того, чтобы определить переменную, достаточно присвоить ей значение:

```
имя_переменной = выражение
```

Любую переменную Octave воспринимает как матрицу. В простейшем случае матрица может состоять из одной строки и одного столбца (листинг 3.1).

```
>>> m=pi
m =  3.1416
>>> m
m =  3.1416
>>> m(1)
ans =  3.1416
>>> m(1,1)
ans =  3.1416
>>> m(1,2)
error: A(I): Index exceeds matrix dimension.
>>> m(3)
error: A(I): Index exceeds matrix dimension.
>>> 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
```

Листинг 3.1

3.1.2 Организация простейшего ввода и вывода в диалоговом режиме

Даже при разработке простейших программ возникает необходимость ввода исходных данных и вывода результатов. Если для вывода результатов на экран можно просто не ставить «`;`» после оператора, то для *ввода исходных данных* при разработке программ, работающих в диалоговом режиме, следует использовать функцию

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

Если в тексте программы встречается оператор `input`, то выполнение программы приостанавливается, Octave выводит на экран текст `подсказки` и переходит в режим ожидания ввода. Пользователь вводит с клавиатуры значение и нажимает клавишу Enter.

Введенное 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 3.1. Создать программу для вычисления значения *y* по формуле *y*=*sin*(*x*), при заданном значении *x*.

Текст программы и результаты ее работы показаны в листинге 3.2.

```
x=input('Введите значение x=');
y=sin(x);
disp('Значение y=');disp(y);
>>>%Результат работы программы
>>>Введите значение x= pi/4
Значение y=
0.70711
```

Листинг 3.2

3.1.3 Условный оператор

Одним из основных операторов, реализующим ветвление в большинстве языков программирования, является *условный оператор*. Существует обычная, сокращенная и расширенная формы этого оператора в языке программирования Octave 7.

Обычный условный оператор имеет вид:

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

Здесь `условие` – логическое выражение, `операторы1`, `операторы2` – операторы языка или встроенные функции Octave. Обычный оператор `if` работает по следующем алгоритму: если `условие` истинно, то выполняются `операторы1`, если ложно – `операторы2`.

ЗАДАЧА 3.2. Даны вещественные числа *x* и *y*. Определить принадлежит ли точка с координатами (*x*; *y*) заштрихованной части плоскости (рис. 3.1).

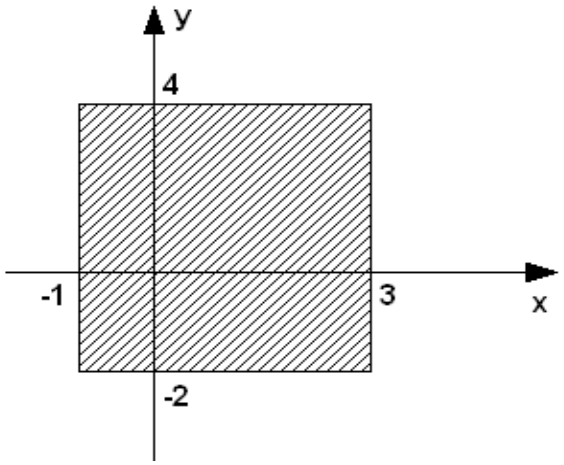


Рис. 3.1: Графическое представление задачи 3.2

Как показано на рис. 3.1, плоскость ограничена линиями *x*=-1, *x*=3, *y*=-2 и *y*=4. Значит

точка с координатами (x; y) будет принадлежать этой плоскости, если будут выполняться следующие условия: $x \geq -1$, $x \leq 3$, $y \geq -2$ и $y \leq 4$. Иначе точка лежит за пределами плоскости.

Далее приведен текст программы и результаты ее работы.

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

Листинг 3.3

Сокращенный условный оператор записывают так:

```
if условие
      операторы
end
```

Работает этот оператор следующим образом. Если `условие` истинно, то выполняются `операторы`, в противном случае управление передается оператору следующему за оператором `if` (листинг 3.4).

```
z=0;
x=input('x=');
y=input('y=');
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
```

Листинг 3.4

Расширенный условный оператор применяют когда одного условия для принятия решения недостаточно:

```
if условие1
      операторы1
elseif условие2
      операторы2
elseif условие 3
      операторы3
…
```

```
elseif условие n
      операторыn

else
      операторы
end
```

Расширеный оператор `if` работает так. Если `условие1` истинно, то выполняются `операторы1`, иначе проверяется `условие2`, если оно истинно, то выполняются `операторы2`, иначе проверяется `условие3` и т.д. Если ни одно из условий по веткам `elseif` не выполняется, то выполняются операторы по ветке `else`.

Рассмотрим использование расширенного условного оператора на примере.

ЗАДАЧА 3.3. Дано вещественное число *x*. Для функции, график которой приведен на рис. 3.2 вычислить *y=f(x)*.

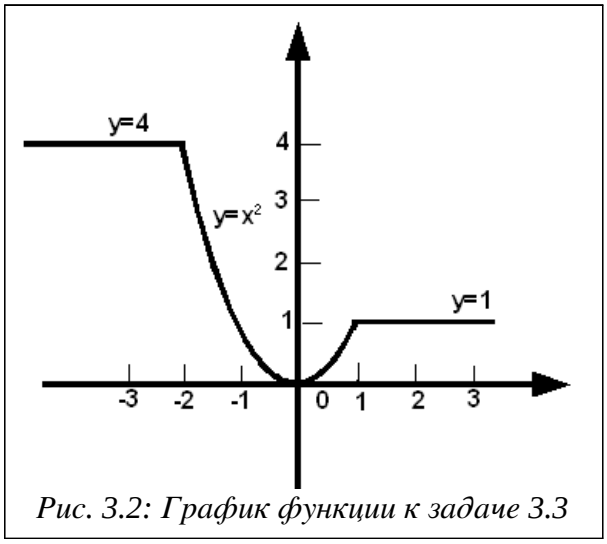


Рис. 3.2: График функции к задаче 3.3

Аналитически функцию представленную на рис. 3.2 можно записать так:

$$y(x)=\begin{cases} 4, & x\leqslant -2 \\ 1, & x\geqslant 1 \\ x^2, & -2<x<1 \end{cases}$$

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:

```
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
```

Листинг 3.5

3.1.4 Оператор альтернативного выбора

Еще одним способом организации разветвлений является оператор альтернативного выбора следующей структуры:

```
swith параметр
 case значение1
      операторы1
 case значение2
      операторы2
 case значение3
      операторы3
…
 otherwise
      операторы
end
```

Оператор `switch` работает следующим образом: если значение `параметра` равно `значениию1`, то выполняются `операторы1`, иначе если `параметр` равен `значениию2`, то выполняются `операторы2`; в противном случае, если значение `параметра` совпадает со `значением3`, то выполняются `операторы3` и т.д. Если значение `параметра` не совпадает ни с одним из значений в группах `case`, то выполняются `операторы`, которые идут после служебного слова `otherwise`.

Конечно, любой алгоритм можно запрограммировать без использования `switch`, используя только `if`, но использование оператора альтернативного выбора `switch` делает программу более компактной.

Рассмотрим использование оператора switch на следующем примере.

ЗАДАЧА 3.4. 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 функцией mod, позволяющей вычислить остаток от деления двух чисел, и условием, что 1-е число – понедельник. Если в результате остаток от деления заданного числа D на семь будет равен единице, то это понедельник, двойке – вторник, тройке – среда и так далее. Следовательно, при построении алгоритма необходимо использовать семь условных операторов. Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта (листинг 3.6).

```
D=input('Введите число от 1 до 31   ');
%Вычисление остатка от деления D на 7,
%сравнение его с числами от 0 до 6.
```

```
switch mod(D,7)
case 1
      disp('ПОНЕДЕЛЬНИК')
case 2
      disp('ВТОРНИК')
case 3
      disp('СРЕДА')
case 4
      disp('ЧЕТВЕРГ')
case 5
      disp('ПЯТНИЦА')
case 6
      disp('СУББОТА')
otherwise
      disp('ВОСКРЕСЕНЬЕ')
end
```

Листинг 3.6

3.1.5 Условный циклический оператор

Оператор *цикла с предусловием* в языке программирования Octave имеет вид:

```
while выражение
      операторы
end
```

Работает цикл с предусловием следующим образом. Вычисляется значение `выражения`. Если оно истинно, выполняются `операторы`. В противном случае цикл заканчивается, и управление передается 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 3.5. 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. 3.1.

Таблица 3.1. Определение количества цифр числа

kol	N
1	12345
2	12345 div10=1234
3	1234 div 10=123
4	123 div 10=12
5	12 div 10=1
	1 div 10=0

Текст программы, реализующей данную задачу, можно записать так:

```
N=input('N=');
M=N;        %Сохранить значение переменной N.
kol=1;      %Пусть число состоит из одной цифры.
while round(M/10) > 0
%Выполнять тело цикла, пока число делится нацело на 10.
      kol=kol+1;          %Счетчик количества цифр
      M=round(M/10);      %Изменение числа.
```

```
end
disp('kol=');disp(kol);
%Результат работы программы
>>>N= 12345678
kol=
8
```

Листинг 3.7

3.1.6 Оператор цикла с известным числом повторений

Для записи цикла с известным числом повторений применяют оператор

```
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, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняются операторы в цикле. Далее параметр меняет свое значение на значение шага . Снова производится проверка значения параметра цикла и алгоритм повторяется.

Если шаг цикла равен 1, то оператор записывают так

```
for параметр = начальное значение:конечное значение
      операторы
end
```

ЗАДАЧА 3.6. Дано натуральное число *N*. Определить *K* – количество делителей этого числа, не превышающих его (Например, для *N*=12 делители 1, 2, 3, 4, 6. Количество *K*=5).

Для решения поставленной задачи нужно реализовать следующий алгоритм: в переменную K, предназначенную для подсчета количества делителей заданного числа, поместить значение, которое не влияло бы на результат, т.е. нуль. 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 N/2 раз.

```
N=input('N=');
K=0;
for i=1:round(N / 2)
%Если N делится нацело на i, то
if mod(N, i) == 0
K=K+1;     %увеличить счетчик на единицу.
end
end
disp(' K=');disp(K);
%Результат работы программы
>>>N= 12
 K= 5
```

Листинг 3.8

3.1.7 Операторы передачи управления

Операторы передачи управления принудительно изменяют порядок выполнения команд. В языке программирования Octave таких операторов два. Операторы break и

continue используют только внутри циклов. Так оператор break осуществляет немедленный выход из циклов while, for и управление передается оператору, находящемуся непосредственно за циклом. Оператор continue начинает новую итерацию цикла, даже если предыдущая не была завершена.

ЗАДАЧА 3.7. 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, N=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 N/2. Если среди значений параметра не найдется 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 приведен в листинге 3.9.

```
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:N/2
% Если N делится на i,
      if mod(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
>>>Введите число  12
НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ
>>>%---------------------------
>>>Введите число  13
ПРОСТОЕ ЧИСЛО
```

Листинг 3.9

3.2 Обработка массивов и матриц

Octave содержит достаточное количество операций предназначенных для работы с векторами и матрицами (см. гл. 6). В этом параграфе мы остановимся на поэлементной обработке одномерных и двумерных массивов.

Ввод массивов и *матриц* следует организовывать поэлементно, например так:

```
N=input('N='); %Ввод элементов массива
```

```
for i=1:N
x(i)=input(strcat('x(',int2str(i),')='));
end
>>>% Результат работы программы
>>>N= 5
x(1)= 1
x(2)= 2
x(3)= 3
x(4)= 4
x(5)= 5
%Ввод элементов матрицы
N=input('N=');
M=input('M=');
for i=1:N
for j=1:M
a(i,j)=input(strcat('a(',int2str(i),',',int2str(j),')='));
end
end
>>>% Результат работы программы
>>>N= 3
M= 3
a(1,1)= 1
a(1,2)= 2
a(1,3)= 3
a(2,1)= 4
a(2,2)= 5
a(2,3)= 6
a(3,1)= 7
a(3,2)= 8
a(3,3)= 9
```

Листинг 3.10

Для вывода приглашений вида `x(i)=` и `a(i,j)=` в функции `input` использовались функции работы со строками: `strcat(s1, s2, …, sn)` и `int2str(d)` . Функция `strcat` предназначена для объединения строк `s1`, `s2`, …, `sn` в одну строку, которая и возвращается в качестве результата. Функция `num2str` преобразовывает число `d` в строку символов.

Алгоритм *вычисления суммы элементов массива* достаточно прост. В переменную предназначенную для накапливания суммы записывают ноль (`s=0`), затем добавляют к `s` первый элемент массива и результат записывают в переменную `s`, далее к переменной `s` добавляют второй элемент массива и результат записывают в `s` и далее аналогично добавляем к `s` остальные элементы массива (листинг 3.11).

```
s=0;
for i=1:N
      s=s+x(i);
end
```

Листинг 3.11

При нахождении *суммы элементов матрицы* последовательно суммируют элементы всех строк:

```
s=0;
for i=1:N
```

```
 for j=1:M
      s=s+a(i,j);
 end
end
```

Листинг 3.12

Алгоритм вычисления *произведения элементов массива* следующий: на первом шаге начальное значение произведения равно 1 (`p=1`), затем последовательно умножают `p` на очередной элемент, и результат записывают в `p)` (листинг 3.13).

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

Листинг 3.13

При вычислении *произведения элементов матрицы* последовательно перемножают элементы всех строк:

```
p=1;
for i=1:N
 for j=1:M
      p=p*a(i,j);
 end
end
```

Листинг 3.14

Алгоритм решения задачи *поиска максимума* и его номера в массиве следующий. Пусть в переменной с именем `Max` хранится значение максимального элемента массива, а в переменной с именем `Nmax` – его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную `Max`, а в `Nmax` – его номер (то есть 1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную `Max`, а в переменную `Nmax`– текущее значение индекса `i`.

На листинге 3.15 представлен фрагмент программы поиска максимума.

```
Max=a(1);
Nmax=1;
for i=1:N
 if x(i)>Max
      Max=x(i);
      Nmax=i;
 end;
end;
```

Листинг 3.15

Алгоритм *поиска минимального элемента* в массиве будет отличаться от приведенного выше лишь тем, что в конструкции `if` текста программы знак поменяется с > на <.

В листинге 3.16 приведен фрагмент программы, реализующий алгоритм *поиска минимального элемента матрицы* и *его индексов*.

```
Min=a(1,1);
Nmin=1;
Lmin=1;
for i=1:N
 for j=1:M
      if a(i,j)<Min
           Min=a(i,j);
```

```
            Nmin=i;
            Lmin=j;
      end;
 end;
end;
```

Листинг 3.16

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Рассмотрим наиболее известный алгоритм сортировки *методом пузырька*. Пусть необходимо упорядочить элементы в массива по возрастанию. 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. Так повторяем до тех пор, пока не упорядочим по возрастанию весь массив. Фрагмент программы *сортировки элементов массива по возрастанию* приведен ниже.

```
for i=1:N-1
 for j=1:N-i
      if x(j)>x(j+1)
            b=x(j);
            x(j)=x(j+1);
            x(j+1)=b;
      end;
 end;
end;
```

Листинг 3.17

Для *сортировки по убыванию* нужно в операторе `if` заменить знак > на <.

Рассмотрим *удаление элемента из массива*. Пусть необходимо удалить из массива `x`, состоящего из `n` элементов, `m`–й по номеру элемент. Для этого достаточно записать элемент (`m+1`) на место элемента `m`, (`m+2`) – на место (`m+1`) и т.д., `n` – на место (`n−1`) и при дальнейшей работе с этим массивом использовать `n−1` элемент. В листинге 3.18 приведен фрагмент программы, реализующий этот алгоритм.

```
for i=m:1:n-1
      x(i)=x(i+1);
end;
```

Листинг 3.18

В Octave есть встроенные функции вычисления суммы (`sum`), произведения (`prod`) элементов массива (матрицы), поиска максимума (`max`) и минимума (`min`), сортировки (`sort`), однако только понимание алгоритмов работы функций позволит решать нестандартные задачи обработки массивов и матриц. Рассмотрим решение нескольких практических задач.

ЗАДАЧА 3.8. Найти наименьшее простое число в массиве $x(n)$, если таких чисел несколько, определите их количество.

Листинг содержит программу решения этой задачи с подробными комментариями.

```
% Ввод размера массива.
N=input('N=');
% Цикл для ввода элементов массива.
for i=1:N
```

```
      x(i)=input(strcat('x(',int2str(i),')='));
end
% Переменная pr=0 (в массиве не обнаружено простых чисел).
% Если pr=0 –простых чисел нет, pr=1, простые числа есть.
pr=0;
for i=1:N
% Переменная L используется для проверки,
%является ли данный элемент массива x(i) простым числом,
%L=1, пока не встретились делители числа,
% L станет равным 0, если встретятся делители числа.
      L=1;
% Цикл по j от 2 до x(i)/2 для проверки является ли
%число простым (поиск возможных делителей числа).
      for j=2:x(i)/2
% Если x(i) делится на j, то встретился делитель числа,
%x(i) не является простым, в L=0 и выходим из цикла по j
%с помощью оператора break.
           if mod(x(i),j)==0
                L=0;
                break;
           end;
      end;
% Проверяем значение переменной L, если L=1,
%то число x(i) - простое.
      if L==1
% Если число x(i) - простое и при этом pr=0,
%то это означает, что встретилось первое простое число,
           if pr==0
% и его записываем в переменную min,
% т.е. предполагаем, что x(i) и является минимальным простым,
                min=x(i);
% количество минимумов равно 1,
                k=1;
% записываем в pr 1, т.к. в массиве есть простые числа.
                pr=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 x(i) 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)
%и k (количество минимумов).
        disp(min);
        disp(k);
end;
```

Листинг 3.19

ЗАДАЧА 3.9. 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`.

Листинге 3.20 содержит текст программы для решения поставленной задачи.

```
% Ввод размера квадратной матрицы.
N=input('N=');
% Ввод квадратной матрицы.
for i=1:N
 for j=1:N
 A(i,j)=input(strcat('a(',int2str(i),',',int2str(j),')='));
 end
end
% Цикл по всем столбцам матрицы, в каждом из которых ищем
%максимальный элемент и его номер.
for j=1:N
% Предполагаем, что первый элемент в столбце
% является максимальным,
        max=A(1,j);
% В переменную nmax, в которой будет
% храниться номер максимального
% элемента j-го столбца записываем 1.
        nmax=1;
% В цикле по i перебираем все элементы j-го столбца.
        for i=2:N
% Если очередной элемент больше max,
            if A(i,j)>max
% то в переменную max записываем этот элемент,
                max=A(i,j);
% а в переменную nmax – номер строки, где находится элемент.
                nmax=i;
            end;
        end;
```

```
% Если в текущем столбце максимальный элемент
%находится на побочной диагонали,
 if nmax==N+1-j
% то обнуляем все элементы в этом столбце.
      for i=1:N
           A(i,j)=0;
      end;
 end;
end;
disp(A);
```

Листинг 3.20

3.3 Обработка строк в Octave

В языке программирования Octave есть множество функций работы со строками. Рассмотрим некоторые из них.

Таблица 3.2. Функции для работы со строками.

Функция	**Описание функции**	**Пример использования**

`char(code)`	Возвращает символ по его коду `code`	`>>> char(100)`
		`ans = d`
		`>>> char(80:85)`
		`ans = PQRSTU`

`deblank(s)`	Формируется новая строка путем удаления пробелов в конце строки `s`	`>>> deblank('OCTAVE  ')` `ans = OCTAVE`

`int2str(x)`	Преобразование чисел, хранящихся в массиве (матрице) `x` к целому типу и запись результатов в массив символов	`>>> int2str(123.456)`
		`ans = 123`
		`>>> int2str([9.8 6.9])`
		`ans = 10 7`

`findstr(str,substr)`	Возвращает номер позиции, начиная с которой подстрока `substr` входит в строку `str`	`>>> Str='Visual C++';`
		`>>> S='C++';`
		`>> findstr(Str,S)`
		`ans =8`

`lower(s)`	Возвращает строку путем преобразования строки `s` к строчным буквам	`>>> S='QtOctave';`
		`>>> lower(S)`
		`ans = qtoctave`

`mat2str(x,n)`	Преобразовывает числовую матрицу `x` в строку; если присутствует необязательный параметр `n`, то перед преобразованием в строку все элементы матрицы округляются до `n` значащих цифр в числе	`>>> X=[7.895; -9.325]`
		`X =`
		`7.8950`
		`-9.3250`
		`>>> mat2str(X)`
		`ans =`
		`[7.8949999999999996;`
		`-9.3249999999999993]`
		`>>> mat2str(X,2)`
		`ans = [7.9;-9.3]`

`num2str(x,n)`	Преобразовывает число-	`>>>X=[7.89578; -9.32985]` `>>> num2str(X)`

Функция	Описание функции	Пример использования

	вую матрицу (массив) x в массив символов с точностью 4 цифры после десятичной точки, если присутствует необязательный параметр n, то перед преобразованием в строку все элементы матрицы округляются до n цифр в числе	ans =
		 7.8958
		-9.3299
		>>> num2str(X,2)
		ans =
		 7.9
		-9.3
		>>> num2str(X,1)
		ans =
		 8
		-9

sprintf(format, x)	Формирует строку из чисел, хранящихся в числовой переменной x в соответствии с форматом format.	>>> x=789.65432145;
		>>> sprintf('X=%4.2e',x)
		ans = X=7.90e+02
		>>> y=-654.12345678;
		>>> sprintf('Y=%7.3f',y)
		ans = Y=-654.123

sscanf(s, format)	Функция возвращает из строки s числовое значение или массив значений в соответствии с форматом	>>> s='1234.5'
		s = 1234.5
		>>> x=sscanf(s, '%f')
		x = 1234.5
		>>> x=sscanf(s, '%d')
		x = 1234

str2double(s)	Формирование числа из строки s, если это возможно	>>> s='1.456e-2';
		>>> str2double(s)
		ans = 0.014560

str2num(s)	Формирование массива чисел из строки (массива символов) s	>>> s='-pi 2 1.6';
		>>> str2num(s)
		ans =
		-3.1416 2.0000 1.6000

strcat(s1,s2,…sn)	Формируется строка путем объединения строк s1, s2, …, sn	>>> s1='Octave';
		>>> s2='Qt';
		>>> s=strcat(s2,s1)
		s = QtOctave ;

strcmp(s1,s2)	Возвращает 1, если строки s1 и s2 совпадают, 0 – в противном случае	>>> S1='Пример';
		>>> strcmp(S1,'Пример')
		ans = 1
		
		>>> S2='1-е Мая';
		>>> S3='1-е мая';
		>>> strcmp(S2,S3)
		ans = 0

strcmpi(s1,s2)	Сравнение строк s1 и s2, не различая строчные и прописные буквы	>>> S2='1-е Мая';
		>>> S3='1-е мая';
		
		>>> strcmpi(S2,S3)
		ans = 0

strjust(s,direction)	Выравнивание строки s в соответствии с	>>>S='Pascal 7.0 '; >>>strjust(S,'right')

Функция	Описание функции	Пример использования
	направлением direct: right – выравнивание по правому краю, left – выравнивание по левому краю, center – выравнивание по центру	ans = Pascal 7.0 >> S=' Pascal 7.0'; >>strjust(S,'left') ans = Pascal 7.0 >> S=' Pascal 7.0'; >>strjust(S,'center') ans = Pascal 7.0

strncmp(s1,s2,n)	Сравнение первых n символов строк s1 и s2, возвращает 1, если первые n символов строк s1 и s2 совпадают, 0 – в противном случае	>>> S1='Мама мыла раму';
		>>> S2='Мама мыла Петю';
		>>> strncmp(S1,S2,10)
		ans = 1
		>>> strncmp(S1,S2,11)
		ans = 0

strrep(s,subs,subsnew)	Формирует новую строку из строки s путем замены подстрок subs на подстроки subsnew	>>>S='07. 07. 2007'; >>> strrep(S,'7','8') Ans = 08. 08. 2008

strtok(s,delimiter)	Поиск первой подстроки в строке s, отделённой пробелом или символом табуляции (при отсутствии параметра delimiter) или первой подстроки, отделенной от s одним из символов, входящих в delimiter. Функция может возвращать 2 параметра: первый – найденная подстрока, второй – содержит остаток строки s после strtok	>>>S='Винни-Пух
		и Пятачок';
		>>> strtok(S)
		ans = Винни-Пух
		
		>>> S='Привет, Пух!';
		>>> strtok(S)
		ans = Привет,
		
		>>>[S1,S2]=strtok(S,',')
		S1 = Привет
		S2 = , Пух!

upper(s)	Возвращает строку s преобразованную к прописным буквам	>>> S='Octave';
		>>> upper(S)
		ans = OCTAVE

3.4 Работа с файлами в Octave

Octave предоставляет широкие возможности для работы с текстовыми и двоичными 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, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.

В *двоичных файлах* информация считывается и записывается в виде блоков определенного размера, в них могут храниться данные любого вида и структуры.

Операции с файлами в Octave имеют много общего с функциями обработки файлов в языке Си. Читатель, имеющий опыт программирования на Си, сможет убедиться, что

фрагменты Си-программ обработки файлов могут с минимальными изменениями быть перенесены в Octave.

3.4.1 Обработка текстовых файлов

Для начала работы с текстовым файлом его необходимо *открыть*, для чего в Octave используется функция следующей структуры

```
fopen(filename, mode)
```

Здесь `filename` – строка, в которой хранится полное имя открываемого файла, `mode` – строка, которая определяет режим работы с файлом. Параметр `mode` может принимать следующие значения:

'`rt`' – открываемый текстовый файл используется в режиме чтения;

'`rt+`' – открываемый текстовый файл используется в режиме чтения и записи;

'`wt`' – создаваемый пустой текстовый файл предназначен только для записи информации;

'`wt+`' – создаваемый пустой текстовый файл предназначен для чтения и записи информации;

'`at`' – открываемый текстовый файл будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;

'`at+` – открываемый текстовый файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан.

Функция `fopen` возвращает идентификатор файла (номер, присвоенный файлу). Существует три стандартных системных файла: стандартный ввод (`stdin`), стандартный вывод (`stdout`) и файл, отвечающий за вывод сообщений об ошибках (`stderr`), за которыми закреплены идентификаторы 0, 1 и 2 соответственно.

Для форматированного вывода информации в файл можно использовать функцию следующего вида:

```
fprintf(f, s1, s2).
```

Здесь `f` – идентификатор файла (значение идентификатора возвращается функцией `fopen`), `s1` – строка вывода, `s2` – список выводимых переменных.

В строке вывода вместо выводимых переменных указывается строка преобразования следующего вида:

```
%[флаг][ширина][.точность] тип.
```

Значения основных параметров строки преобразования (символы управления форматированием) приведены в табл. 3.3.

Табл. 3.3. Символы управления форматированием

Параметр	Назначение
	Флаги
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «−»
0	Заполнение незаполненные позиции дополняются нулями
	Ширина
n	Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции дополняются пробелами
	Точность
ничего	Точность по умолчанию

Параметр	Назначение
m	Для типов *e*, *E*, *f* выводить m знаков после десятичной точки
Тип	
c	При вводе символьный тип char, при выводе один байт.
d	Десятичное целое со знаком
o	Восьмеричное целое без знака
u	Десятичное целое без знака
x, X	Шестнадцатеричное целое без знака, при x используются символы a - f, при X – A-F.
f	Значение со знаком вида [-]dddd.dddd
e	Значение со знаком вида [-]d.dddd e[+|-]ddd
E	Значение со знаком вида [-]d.dddd E[+|-]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

В строке вывода могут использоваться некоторые специальные символы, приведенные в табл. 3.4.

Табл. 3.4. Специальные символы

Символ	Назначение
\b	Сдвиг текущей позиции влево
\n	Перевод строки
\r	Перевод в начало строки, не переходя на новую строку
\t	Горизонтальная табуляция
\'	Символ одинарной кавычки
\''	Символ двойной кавычки
\?	Символ ?

При *считывании данных из файла* можно воспользоваться функцией следующего вида:

```
A=fscanf(f, s1, n).
```

Здесь `f` – идентификатор файла, который возвращается функцией `fopen`, `s1` – строка форматов вида `%[ширина][.точность]тип`, `s2` – имя переменной, количество считываемых значений.

Функция `fscanf` работает следующим образом: из файла с идентификатором `f` считывается в переменную `A` `n` значений в соответствии с форматом `s1`. При чтении числовых значений из текстового файла следует помнить, что два числа считаются разделенными, если между ними есть хотя бы один пробел, символ табуляции или символ перехода на новую строку.

При считывании данных из текстового файла пользователь должен следить, достигнут ли конец файла с помощью функции `feof(f)` (`f` – идентификатор файла), которая возвращает единицу, если достигнут конец файла, и ноль в противном случае.

После выполнения всех операций с файлом он должен быть закрыт с помощью функции:

```
fclose(f).
```

Здесь `f` – идентификатор закрываемого файла. С помощью функции `fclose('all')` можно закрыть сразу все открытые файлы, кроме стандартных системных файлов.

Рассмотрим использование рассмотренных выше функций на простых примерах.

ЗАДАЧА 3.10. Поменять местами элементы, расположенные на главной и побочной диагонали квадратной матрицы *A*(*N*,*N*). Исходную и преобразованную матрицы вывести в текстовый файл *prim_3_10.txt*.

Далее приведена программа решения задачи с комментариями.

```
N=input('N=');% Ввод размеров матрицы.
% Ввод элементов матрицы.
for i=1:N
 for j=1:N
 A(i,j)=input(strcat('A(',int2str(i),',',int2str(j),')='));
 end
end
% Открыть файл  для записи (создать новый пустой файл).
f=fopen('prim_4_9.txt','wt');
% Вывод в файл строки ИСХОДНАЯ МАТРИЦА A
% и перевод курсора на новую  строку (символ \n).
fprintf(f,'ИСХОДНАЯ МАТРИЦА A\n');
% Цикл для построчной записи элементов матрицы в файл.
for i=1:N
% Цикл для поэлементной записи в файл i-й строки матрицы.
 for j=1:N
%Запись очередного элемента A(i,j) и символа табуляции в файл.
      fprintf(f,'%f\t',A(i,j));
 end
% После записи очередной строки переход
%к следующей строке файла.
 fprintf(f,'\n');
end
% В каждой строке матрице
for i=1:N
% поменять местами элементы,
%расположенные на главной и побочной диагоналях.
      b=A(i,i);
      A(i,i)=A(i,N+1-i);
      A(i,N+1-i)=b;
end
% Вывод в файл строки МАТРИЦА A после преобразования
% и перевод курсора на новую строку (символ \n).
fprintf(f,'МАТРИЦА A после преобразования\n');
% Двойной цикл для вывода матрицы в файл.
for i=1:N
 for j=1:N
      fprintf(f,'%f\t',A(i,j));
 end
 fprintf(f,'\n');
end
% Закрытие файла после записи в него необходимой информации.
fclose(f);
```

Листинг 3.21

В результате работы программы создан файл *prim_3_10.txt* (рис. 3.3), который можно открыть при помощи обычного текстового редактора.

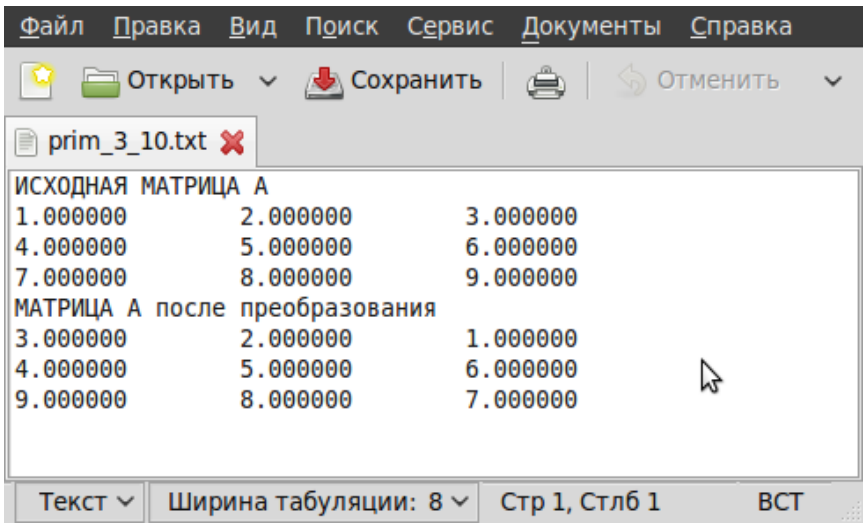


Рис. 3.3: Результат работы программы из задачи3.10

Обратите внимание, что после записи информации в файл, его обязательно надо закрывать с помощью функции `fclose`. Дело в том, что `fprintf` 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 `fclose` 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 3.11. 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 3.22 содержит фрагмент программы для создания подобного файла.

```
% Ввод размеров матрицы.
N=input('N=');
M=input('M=');
% Ввод элементов матрицы.
for i=1:N
 for j=1:M
 A(i,j)=input(strcat('A(',int2str(i),',',int2str(j),')='));
 end
end
% Открыть файл  для записи.
f=fopen('primer.txt','wt');
% Записать в файл N и M, разделив их символом табуляции,
% после чего перейти на новую строку в файле.
fprintf(f,'%d\t%d\n',N,M);
% Цикл для построчной записи элементов матрицы в файл.
for i=1:N
% Цикл для поэлементной записи в файл i-й строки матрицы.
 for j=1:M
%Запись очередного элемента A(i,j) и символа табуляции в файл.
      fprintf(f,'%g\t',A(i,j));
 end;
% После записи очередной строки переход к следующей строке.
 fprintf(f,'\n');
```

```
end;
% Закрыть файл.
fclose(f);
```

Листинг 3.22

После выполнения этой программы будет создан текстовый файл *prim_3_11.txt* (рис. 3.4).

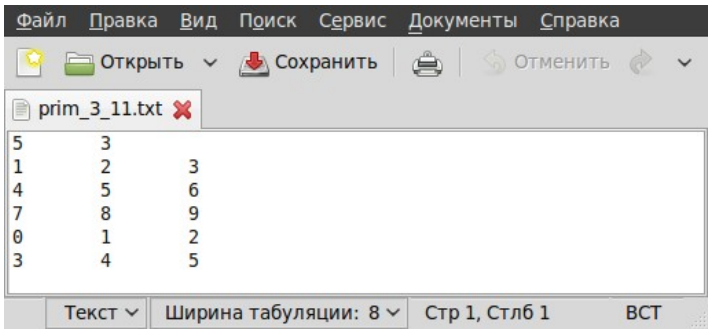


Рис. 3.4: Результат работы программы из задачи 3.11

ЗАДАЧА 3.12. Считать информацию из файла *prim_3_11.txt* в матрицу. Рассмотрим поэлементное (листинг 3.23) и построчное (листинг 3.24) чтение матрицы из файла. В обоих случаях чтение из текстового файла начинается с чтения значений `N` и `M`, которые хранятся в первой строке файла *prim_3_11.txt*. Затем при построчном чтении организован цикл, в котором считывается одна строка с помощью функции `fscanf`. При поэлементном чтении организован двойной цикл, в котором функция `fscanf` считывает значение одного элемента матрицы из файла.

```
f=fopen('primer.txt','rt');% Открываем файл для чтения.
% Считываем одно целое число
% в переменную N (количество строк матрицы).
N=fscanf(f,'%d',1);
% Считываем одно целое число
% в переменную M (количество столбцов матрицы).
M=fscanf(f,'%d',1);
% Двойной цикл по строкам и столбцам.
for i=1:N
 for j=1:M
% Считываем один элемент из файла в матрицу.
      A(i,j)=fscanf(f,'%g',1);
 end;
end;
fclose(f);% Закрываем файл.
A % Вывод матрицы на экран
% Результат работы программы
>>>A =
1 2 3
4 5 6
7 8 9
0 1 2
3 4 5
```

Листинг 3.23

```
% ------------------------------------------------
```

```
% Открываем файл для чтения.
f=fopen('primer.txt','rt');
% Считываем одно целое число
% в переменную N (количество строк матрицы).
N=fscanf(f,'%d',1);
% Считываем одно целое число
% в переменную M (количество столбцов матрицы).
M=fscanf(f,'%d',1);
% Открываем цикл по строкам.
for i=1:N
% Считываем в i-ю строку M элементов матрицы (всю строку).
      A(i,:)=fscanf(f,'%g',M);
end;
% Закрываем файл.
fclose(f);
A % Вывод матрицы на экран
% Результат работы программы
>>>A =
1 2 3
4 5 6
7 8 9
0 1 2
3 4 5
```

Листинг 3.24

ЗАДАЧА 3.13. В текстовом файле *one.txt* хранятся вещественные значения (рис. 3.5). Считать их в массив.

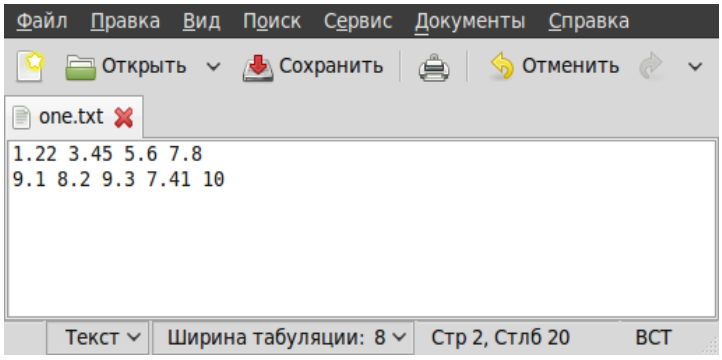


Рис. 3.5: Текстовый файл one.txt

Возможно считывание из файла всего массива целиком (листинг 3.25) или поэлементное считывание данных из файла (листинг 3.26).

```
f=fopen('one.txt');% Открываем файл для чтения.
% Считываем содержимое файла целиком в массив.
x=fscanf(f,'%f');
% Сформирован массив x.
% Результат работы программы
>>>x =
1.2200
3.4500
5.6000
7.8000
```

```
9.1000
8.2000
9.3000
7.4100
10.0000
```

Листинг 3.25

```
% ------------------------------------------------
f=fopen('one.txt');% Открываем файл для чтения.
% В переменной i будет храниться номер элемента массива,
% куда будет осуществляться считывание очередного значения
% из файла, в начале в i записываем 0,
% пока в массиве нет элементов.
i=0;
while ~ feof(f) % Проверяем, если не достигнут конец файла,
i=i+1;          % то увеличиваем i,
% и считываем очередной i-й элемент из файла в массив
X(i)=fscanf(f,'%f',1);
end
% В массиве x из i элементов будут храниться
% все числа из файла one.txt
X
% Результат работы программы
>>>X = 1.220 3.450 5.600 7.800 9.100 8.200 9.300 7.410  10.000
```

Листинг 3.26

Обратите внимание, если данные в файле располагаются в несколько строк, то программы, аналогичные приведенным на листингах 3.23, 3.24, считывают их как матрицу значений, а программы, приведенные на листингах 3.25, 3.26 считывают значения из файла в одномерный массив.

В языке программирования Octave есть функции для записи и чтения матриц в текстовый файл и из текстового файла.

Функция `dlmread` предназначена для *чтения числовых данных из текстового файла* в матрицу. Существуют четыре варианта использования функции.

1. `M=dlmread('filename')` – чтение чисел из текстового файла `filename` в матрицу `M`, числа в строке разделены пробелом. В листинге 3.27 представлен результат чтения данных из файла *one.txt* (рис. 3.5).

```
>>> H=dlmread('one.txt')
H =
    1.22000    3.45000    5.60000    7.80000    0.00000
    9.10000    8.20000    9.30000    7.41000   10.00000
```

Листинг 3.27

Если в каких-либо строках текстового файла пропущено значение, то недостающий элемент матрицы будет равен нулю.

2. `M=dlmread('filename', delimiter)` – чтение чисел из текстового файла `filename` в матрицу `M`, числа отделяются символом, хранящимся в `delimiter`. Например, `M=dlmread('ab.txt', '\t')` – чтение из файла *ab.txt* чисел в матрицу `M`, внутри строки числа отделяются табуляцией. В качестве примера рассмотрим файл *two.txt* (рис. 3.6), в котором числа внутри строки разделены двоеточием. В листинге 3.28 представлен результат чтения данных из файла *two.txt*.

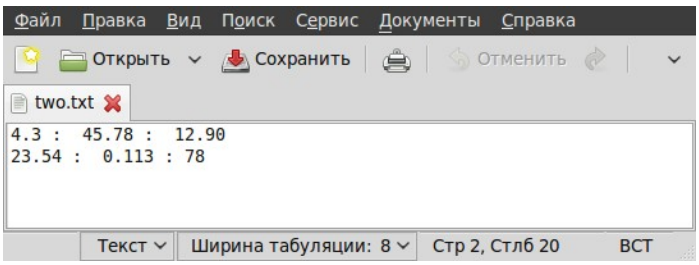


Рис. 3.6: Файл two.txt

```
>>> L=dlmread('two.txt',':')
L =
    4.30000   45.78000   12.90000
   23.54000    0.11300   78.00000
```

Листинг 3.28

3. `M=dlmread('filename', delimiter, R, C)` – чтение чисел из текстового файла `filename` в матрицу `M`, числа внутри строки отделяются символом, хранящимся в `delimiter`, начиная со строки `R` и столбца `C`. Строки и столбцы нумеруются с 0. Листинг 3.21 представлено чтение матрицы из файла *two.txt* (рис. 3.6) начиная со второй строки и третьего столбца.

```
>>> L=dlmread('two.txt',':',1,2)
L =  78
```

Листинг 3.29

4. `M = dlmread ('filename', delimiter, range)` – чтение чисел из текстового файла `filename` в матрицу `M`, числа внутри строки отделяются символом, хранящимся в `delimiter`, `range` определяет область [`row_start col_start row_end col_end`], `row_start`, `col_start` - левый верхний угол, а `row_end`, `col_end` - правый нижний угол области данных, которые считываются из файла. Область `range` может быть задана в стиле электронных таблиц, например – `'A1:C3'`. В листинге 3.30 представлено чтение матрицы из файла *one.txt* (рис. 3.5).

```
>>> P=dlmread('one.txt',' ',[0 0 1 2])
P =
1.2200 3.4500 5.6000
9.1000 8.2000 9.3000
>>> P=dlmread('one.txt',' ','A1:C2')
P =
1.2200 3.4500 5.6000
9.1000 8.2000 9.3000
```

Листинг 3.30.

Функция `dlmwrite` предназначена для записи матрицы в текстовый файл. Существуют три варианта использования функции.

1. `dlmwriter('filename',M)` – запись матрицы `M` в текстовый файл `filename`, числа внутри строки отделяются запятой.

2. `dlmwriter('filename', M, delimiter)` – запись матрицы `M` в текстовый файл, числа внутри строки отделяются символом, хранящимся в `delimiter`.

3. `dlmwriter('filename', M, delimiter, R, C)` – запись матрицы `M`, начиная со строки `R` и столбца `C` в текстовый файл `filename`, числа внутри строки отделяются символом, хранящимся в `dlmwriter`.

Вывести содержимое текстового файла с именем `filename` на экран можно с помощью функции:

```
type('filename');
```

Листинг 3.31 содержит пример вызова функций `dlmwrite` и `type`.

```
>>> M=[1 2 3;4 5 6; 7 8 9];
>>> dlmwrite('file.txt',M);
>>> type('file.txt')
file.txt is the user-defined function defined from: ./file.txt
1,2,3
4,5,6
7,8,9
```

Листинг 3.31

3.4.2 Обработка двоичных файлов в Octave

Двоичный файл, как и текстовый открывается с помощью функции `fopen`. Разница в том, что в параметре `mode` вместо буквы `t` должна использоваться буква `b` (от слова `binary` – двоичный): «`rb+`», «`wb+`» и т.д.

Чтение из двоичного файла осуществляется с помощью обращения к функции

```
[A, k]=fread (f, n, type);
```

где `f` – идентификатор файла, `n` – количество считываемых из файла элементов, `type` – тип считываемых из файла элементов, `A` – массив, `k` – количество считанных из файла элементов.

Возможные значения параметра `type` приведены в табл. 3.5.

Таблица 3.5. Возможные значения параметра type

Параметр type	Размер элемента в байтах	Описание
`uchar`	1	Целое число без знака (0÷255)
`schar`	1	Целое со знаком (-128÷127)
`int16`	2	Целое со знаком (-32768÷32767)
`int32`	4	Целое со знаком (-2147483648÷2147483647)
`int64`	8	Целое со знаком $-(2^{63}-1)\div(2^{63}-1)$
`uint16`	2	Целое без знака (0÷65535)
`uint32`	4	Целое без знака (0÷4294967295)
`uint64`	8	Целое без знака $(0\div 2^{64}-1)$
`float32`	4	Вещественное число (3.4E-38÷3.4E+38)
`float64`	8	Вещественное число (1.7E-308÷1.7E+308)

Функция `fread` считывает из предварительно открытого файла с идентификатором `f` `n` элементов типа `type` и записывает их в массив (матрицу) `A`, количество реально считанных элементов возвращается в переменной `k`. Если при обращении к функции `fread` отсутствует параметр `type`, то подразумевается что из двоичного файла будут считываться значения типа `uchar` (однобайтовое целое без знака). Если пропущен и параметр `n`, то в массив `A` будут считываться все значения до конца файла. Параметр `n` может быть представлен в виде `[m k]`, в этом случае данные считываются в матрицу размером `m` на `k`.

Файл – последовательная структура данных. После открытия файла доступен первый элемент, хранящийся в файле. После чтения очередной порции данных указатель файла смещается на следующую порцию данных.

Текущая позиция указателя файла (смещения от начала файла в байтах) возвращается функцией

```
ftell(f);
```

Здесь `f` – идентификатор уже открытого с помощью `fopen` файла.

Для *перемещения указателя в начало файла* служит функция

```
frewind(f);
```

Функция

```
fseek(f, n, origin);
```

обеспечивает все остальные *перемещения указателя файла*. Функция перемещает текущую позицию в файле с идентификатором `f` на `n` байт, относительно позиции `origin`.

Параметр `origin` может принимать одно из следующих значений:

строка '`bof`' или число -1 определяет смещение относительно начала файла, в этом случае значение n может быть только положительным;

строка '`eof`' или число 1 определяет смещение относительно конца файла на `n` байтов назад, в этом случае значение n также должно быть положительным;

строка '`cof`' или число 0 определяет смещение относительно текущей позиции на `n` байтов вперед (`n`>0) или назад (`n`<0).

Запись в двоичный файл осуществляется с помощью функции

```
n=fwrite(f, A, type),
```

где `f` – идентификатор файла, `A` – массив (матрица) значений, `type` – тип записываемых в файл элементов. Функция `fwrite` записывает в заранее открытый с идентификатором `f` массив `A`, и возвращает количество реально записанных в файл значений.

Рассмотрим несколько примеров работы с двоичными файлами.

ЗАДАЧА 3.14. Создать двоичный файл *abc.dat*, куда записать целое число `N`, а затем `N` вещественных чисел. Решить эту задачу можно двумя способами:

1. Записать в файл целое число `N`, а затем в цикле `N` вещественных чисел.

2. Записать в файл целое число N, а затем одним оператором `fwrite` записать в файл массив из `N` вещественных чисел.

Решение задачи с комментариями первым способом представлено на листинге 3.32, вторым – на листинге 3.33.

```
% Ввод значения переменной N.
N=input('N=');
% Открытие нового двоичного файла abc.dat в режиме записи.
f=fopen('abc.dat','wb');
% Запись числа N в двоичный файл abc.dat.
fwrite(f,N,'int16');
% Цикл для ввода N вещественных чисел и записи их
% в двоичный файл abc.dat
for i=1:N
% Ввод очередного вещественного числа x.
x=input('X=');
% Запись очередного числа x в двоичный файл abc.dat.
fwrite(f,x,'float32');
end;
% Закрытие файла.
fclose(f);
```

Листинг 3.32

```
% ----------------------------------------
% Ввод значения переменной N.
N=input('N=');
% Открытие нового двоичного файла abc.dat в режиме записи.
f=fopen('abc.dat','wb');
% Запись числа N в двоичный файл abc.dat.
fwrite(f,N,'int16');
```

```
% Цикл для ввода массива из N вещественных чисел.
for i=1:N
% Ввод очередного вещественного числа в массив x.
x(i)=input(strcat('x(',int2str(i),')='));
end;
% Запись массива вещественных чисел в двоичный файл abc.dat.
fwrite(f,x,'float32');
% Закрытие файла.
fclose(f);
```

Листинг 3.33

В результате будет сформирован двоичный файл размером N*4+2 байт. Запустим любую из этих программ на выполнение в командной строке, введем N=20 и сформируем файл abc.dat размером 82 байта.

ЗАДАЧА 3.15. Считать данные из файла *abc.dat*, сформированного в задаче 3.14 в массив вещественных чисел.

Программа решения этой задачи представлена на листинге 3.34.

```
% Открытие двоичного файла abc.dat в режиме чтения.
f=fopen('abc.dat','rb');
% Чтение числа N из двоичного файла abc.dat.
N=fread(f,1,'int16');
% Чтение массива из N вещественных
% чисел из двоичного файла abc.dat.
x=fread(f,N,'float32');
% Закрытие файла.
fclose(f);
```

Листинг 3.34

ЗАДАЧА 3.16. Считать данные из файла *abc.dat*, сформированного в задаче 3.14 в матрицу вещественных чисел. Зная, что в файле *abc.dat* хранится 20 чисел, в качестве примера запишем их матрицу размером 4x5. Программа решения этой задачи представлена в листинге 3.35. В результате работы этой программы будет сформирована матрица вещественных чисел G(4,5).

```
% Открытие двоичного файла abc.dat в режиме чтения.
f=fopen('abc.dat','rb');
% Чтение числа N из двоичного файла abc.dat.
N=fread(f,1,'int16');
% Чтение матрицы вещественных чисел G(4,5)
% из двоичного файла abc.dat.
G=fread(f,[4 5],'float32');
% Закрытие файла.
fclose(f);
>>> G
G =
1.20000 9.00000 7.40000 8.90000 5.40000
3.40000 0.10000 6.50000 0.90000 4.30000
5.60000 9.20000 5.60000 8.70000 3.20000
7.80000 8.30000 7.80000 6.50000 2.10000
```

Листинг 3.35

Отдельную задачу представляет чтение данных из двоичного файла, если заранее не известно количество элементов в файле. В листинге 3.36 представлена программа, с помощью которой можно создать файл вещественных чисел.

```
% Ввод значения переменной N.
```

```
N=input('N=');
% Открытие нового двоичного файла abc2.dat в режиме записи.
f=fopen('abc2.dat','wb');
% Цикл для ввода N вещественных чисел и записи
% их в двоичный файл abc2.dat
for i=1:N
% Ввод очередного вещественного числа x.
x=input('X=');
% Запись очередного числа x в двоичный файл abc.dat.
fwrite(f,x,'float32');
end;
fclose(f);
```

Листинг 3.36

Отличие этой программы от представленных на листингах 3.32, 3.33 состоит в том, что количество записанных в файл вещественных чисел *abc2.dat* в нем не хранится. Поэтому чтение данных из такого файла осуществляется несколько по-другому. Известно, что функция `ftell(f)` возвращает текущее положение указателя файла. Если с помощью функции `fseek(f,0,1)` передвинуть указатель в конец файла, а затем обратиться к функции `ftell(f)`, можно вычислить количество байт в файле. Разделив полученное число на 4 (размера вещественного числа типа '`float32`'), получим количество элементов в файле, после чего считаем нужное количество элементов в массив с помощью функции `fread`. Программа, реализующая описанные выше действия, представлена на листинге 3.37.

```
% Открытие двоичного файла abc2.dat в режиме чтения.
f=fopen('abc2.dat','rb');
% Перевод указателя в конец файла.
fseek(f,0,1);
% С помощью функции ftell вычисляем количество байт в файле,
% после чего делим на размер одного элемента,
% для 'float32' это число 4.
N=ftell(f)/4;
% Переводим указатель на начало файла
frewind(f);
% Чтение из двоичного файла abc2.dat в массива x
% N вещественных чисел.
x=fread(f,N,'float32');
% Закрытие файла.
fclose(f);
```

Листинг 3.37

Аналогичным образом можно будет считать данные любого типа из двоичного файла. Отличие будет состоять только в том, что в операторе `N=ftell(f)/4`; необходимо заменить число 4 на действительный размер элементов, хранящихся в файле и при обращении к функции `fread` указать в качестве третьего параметра реальный тип считываемых данных. Функция `fread` не считает больше элементов, чем находится в файле, независимо от того, что указано во втором параметре. Поэтому, если в листинге 3.37 оператор вычисления `N` записать следующим образом `N=ftell(f)`, то программа будет корректно считывать данные в массив любого типа. Будет происходить следующее: оператор `x=fread(f,N,type)` будет пытаться считать `N` элементов из двоичного файла, но не считает значений больше, чем их там есть, остановится на конце файла.

3.5 Функции в Octave

В Octave файлы с расширением .m могут содержать не только тексты программ (группа операторов и функций Octave), но и могут быть оформлены как 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.m*).

Функция в Octave 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 *.m*, 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 m-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
```

Такая структура, близка к структуре программ на языке Си. Она не допускает вложенности функций друг в друга. Однако в Octave возможен и другой синтаксис, в котором разрешено использование вложенных функций, поэтому структура основной функции может быть и такой:

```
function [y1,y2,…yn]=name_function(x1,x2,…,xm)
 function [y1,y2,…yn]=f1(x1,x2,…,xm) % начало первой подфукции
      операторы
 end % конец первой подфункции
```

```
 function [y1,y2,…yn]=f2(x1,x2,…,xm) % начало второй подфукции
      операторы
 end % конец второй подфункции
…
 function [y1,y2,…yn]=fn(x1,x2,…,xm) % начало n-й подфукции
      операторы
 end % конец n-й подфункции
 оператор1;
 оператор2;
 …
 операторk;
end   %  здесь заканчивается основная функция
```

Рассмотрим пример.

ЗАДАЧА 3.17. Написать функцию, предназначенную для удаления из массива *x*(*N*) простых чисел.

Функцию назовем `udal_prostoe`. Ее 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; 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 измененное значение `N`.

В листинге 3.38 приведено содержимое файла *udal_prostoe.m* с комментариями.

```
% Функция udal_prostoe удаляет из массива x(N) простые числа
% и возвращает модифицированный массив x и измененное значение
%  N в качестве результата.
function [x N]=udal_prostoe(x, N)
 i=1;
 while i<=N
% Обращение к функции prostoe для проверки является ли число
% x(i) простым.
      L=prostoe(x(i));
% Число простое (L=1),
      if L==1
% то обращение к функции udal для удаления из массива x(N)
% i-го элемента,
           [x N]=udal(x,i,N);
% иначе переход к следующему элементу массива.
      else
           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 mod(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 путем смещения элементов, начиная с m-го
% на одну позицию влево. Выходными элементами функции будут
% массив x, из которого удален один элемент, уменьшенное на 1
% количество(N) элементов в массиве.
for i=m:N-1
 x(i)=x(i+1);
end
% После смещения элементов удаляем последний элемент и
x(:,N)=[];
% уменьшаем количество элементов в массиве на 1.
N=N-1;
end % Окончание функции udal.
```

Листинг 3.38

В листинге 3.38 был использован синтаксис не допускающий вложенность функций друг в друга. Листинг 3.39 содержит текст программы, структура которой допускает вложенность функций.

```
function [x N]=udal_prostoe1(x, N)

function pr=prostoe(P)
pr=1;
for i=2:P/2
 if mod(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
```

Листинг 3.39

На листинге 3.40 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
y = 4 6    8    100  88   125
>> k
k= 6
```

Листинг 3.40

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

В Octave есть возможность *передавать имя функции как входной параметр*, что существенно расширяет возможности программирования. Вообще говоря, имя функции передается как строка, а ее вычисление осуществляется с помощью функции `feval`.

Функция `feval` предоставляет альтернативный способ вычисления значения функции.

Параметрами функции `feval` являются: строка с именем вызываемой функции, в качестве имени может быть встроенная функция или определенная пользователем функция; параметры этой функции, разделенные запятой.

В качестве параметра можно передать строку с именем функции, а затем с помощью функции `feval` обратиться к передаваемой функции. Рассмотрим передачу функции как параметра на примере решения следующей задачи.

ЗАДАЧА 3.18. Вычислить значение функций *sin*(x) и *cos*(x) в точке $x=\pi/12$.

Вычисление можно осуществить обычным способом или с использованием функции `feval`:

```
>>> x=pi/12
x = 0.26180
>>> sin(pi/13)
ans = 0.23932
>>> feval('sin',x)
ans = 0.25882
>>> cos(pi/13)
ans = 0.97094
>>> feval('cos',x)
ans = 0.96593
```

Листинг 3.41

Как известно, многие функции Octave допускают обращение к ним с различным числом параметров. При этом алгоритм функций анализирует количество входных параметров и осуществляет корректную работу при различном количестве входных параметров.

Рассмотрим, как создавать функции, в которых может использоваться *разное*

количество входных параметров. В качестве входного параметра в этом случае будет использоваться массив ячеек, который позволяет хранить разнородные данные. В этом случае все входные параметры хранятся в виде единственного параметра массива ячеек `varargin`. С помощью функции `length(varargin)` можно вычислить количество поступивших в функцию входных параметров, а с помощью конструкции `varargin{i}` – обратиться к i-му входному параметру.

Рассмотрим простой пример функции с переменным числом параметров.

ЗАДАЧА 3.19. Найти сумму всех входных параметров функции. Будем считать, что все входные параметры – скалярные величины.

Выходными параметрами функции будут найденная сумма `sum` и строка `s`, в которой будет храниться аварийное сообщение, если строка не найдена.

В листинге приведена программа решения задачи с комментариями.

```
% Функция вычисления суммы входных параметров.
% Входные параметры хранятся в массиве ячеек.
% Функция возвращает значение суммы в переменной sum,
% а в переменной s формируется сообщение об ошибке,
% если невозможно найти сумму (если среди входных параметров
%  были нечисловые значения).
function [sum s]=sum_var(varargin)
% Переменная pr=1, если все элементы массива ячеек являются
% числами.
pr=1;
% До начала суммирования в переменную sum запишем 0.
sum=0;
% length(varavgin) возвращает количество входных параметров
% в функции sum_var, затем открываем цикл по i для перебора
%  всех входных параметров от 1 до length(varargin).
for i=1:length(varargin)
% С помощью функции isnumeric проверяем, является ли очередной
%  входной параметр числом,
if isnumeric(varargin{i})==1
% если является, добавляем очередной входной
% параметр varargin{i} к sum.
sum=sum+varargin{i};
else
% если не является, записываем в pr=0,
pr=0;
% обнуляем сумму sum
sum=0;
% и прерываем цикл.
break;
end;
end
% Если после проверки всех входных параметров pr=1,
% что означает, что все входные параметры были числами и сумма
% их найдена, очищаем переменную s,где должно хранится
% аварийное сообщение,
if pr==1
s=[];
% иначе записываем в s аварийное сообщение.
```

```
else
s='Во входных параметрах были нечисловые данные'
end
end
>>>% Вызов функции
>>> Summa=sum_var(1,2,3,4,5)
Summa = 15
>>> Summa=sum_var(pi,1.23,e)
Summa = 7.0899
```

Листинг 3.42

Под *рекурсией* в программировании понимается вызов функции из ее 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 3.20. Вычислить *n*-е число Фибоначчи.

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

Текст функции:

```
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
```

Листинг 3.43

4. Построение графиков в Octave

В этой главе читатель научится строить графики в Octave. Первый параграф посвящен работе с двумерными графиками. Во втором параграфе рассмотрено создание различных трехмерных графиков. Затем описаны анимационные возможности Octave. Завершается глава описанием графических возможностей пакета.

4.1 Построение двумерных графиков

Двумерным будем считать такой график, в котором положение точки определяется двумя величинами. Двумерные графики наиболее часто строят в декартовой и полярной системах координат.

4.1.1 Построение графиков в декартовой системе координат

Декартова или прямоугольная система координат, задается 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, определяется как уравнение 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 определенных 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 задается одной или несколькими формулами (уравнениями), при этом, если зависимость между x и y выражена уравнением, разрешенным 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.

Далее будем рассматривать построение графиков в прямоугольной системе координат на конкретных примерах.

ЗАДАЧА 4.1. Построить график функции $y=\sin x+\frac{1}{3}\sin 3x+\frac{1}{5}\sin 5x$ на интервале [-10;10].

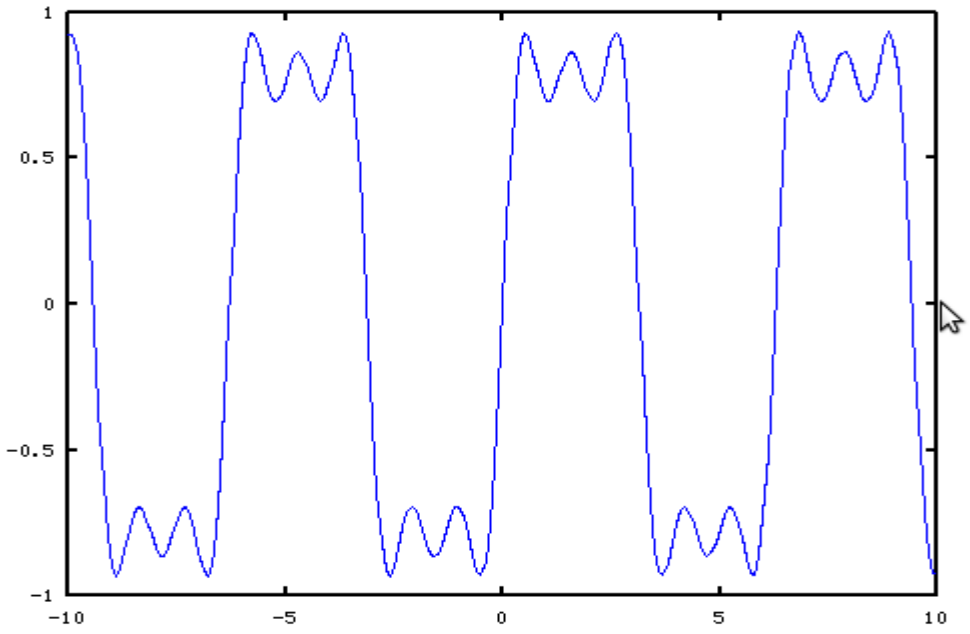
Для того, чтобы построить график функции $f(x)$ необходимо сформировать два массива x и y одинаковой размерности, а затем обратиться к функции `plot`.

Решение этой задачи представлено на листинге 4.1.

```
x=-10:0.1:10; %Формирование массива x.
y=cos(x/2)+cos(5*x)/5; %Формирование массива y.
plot(x,y) %Построение графика функции.
```

Листинг 4.1

В результате обращения к функции `plot(x,y)` будет создано окно с именем *Figure* 1, в котором будет построен график функции $y=\sin(x)+\frac{1}{3}\sin(3x)+\frac{1}{5}\sin(5x)$ (рис. 4.1).




```
10.0819,  0.000299670
```

Рис. 4.1. График функции $y=\sin x+\frac{1}{3}\sin 3x+\frac{1}{5}\sin 5x$ *на интервале [-10;10]*

График формируется путем соединения соседних точек прямыми линиями. Чем больше будет интервал между соседними точками (чем меньше будет точек), тем больше будет заметно, что график представляет из себя ломанную.

Если повторно обратиться к функции `plot`, то в этом же окне будет удален первый график и нарисован второй. Для построения нескольких графиков в одной системе координат можно поступить одним из следующих способов:

1. Обратиться к функции `plot` следующим образом `plot(x1,y1,x2,y2,…xn,yn)`, где *x1*, *y1* – массивы абсцисс и ординат первого графика, *x2*, *y2* – массивы абсцисс и ординат второго графика, …, *xn*, *yn* – массивы абсцисс и ординат n-ого графика.
2. Каждый график изображать с помощью функции `plot(x,y)` , но перед обращением к функциям `plot(x2,y2)`, `plot(x3,y3)`, …, `plot(xn,yn)` вызвать команду `hold on`[6], которая блокирует режим очистки окна.

Рассмотрим построение нескольких графиков этими способами на примере решения следующей задачи.

ЗАДАЧА 4.2. Построить графики функций

6 Команда hold работает в режиме переключателя: hold on — блокирует режим очистки экрана, hold off — включает режим очистки экрана.

$$v=\sin x,\ w=\cos x,\ r=\sin\frac{x}{2},\ p=\frac{3}{2}\cos x$$

на интервале [*-4π; 4π*].

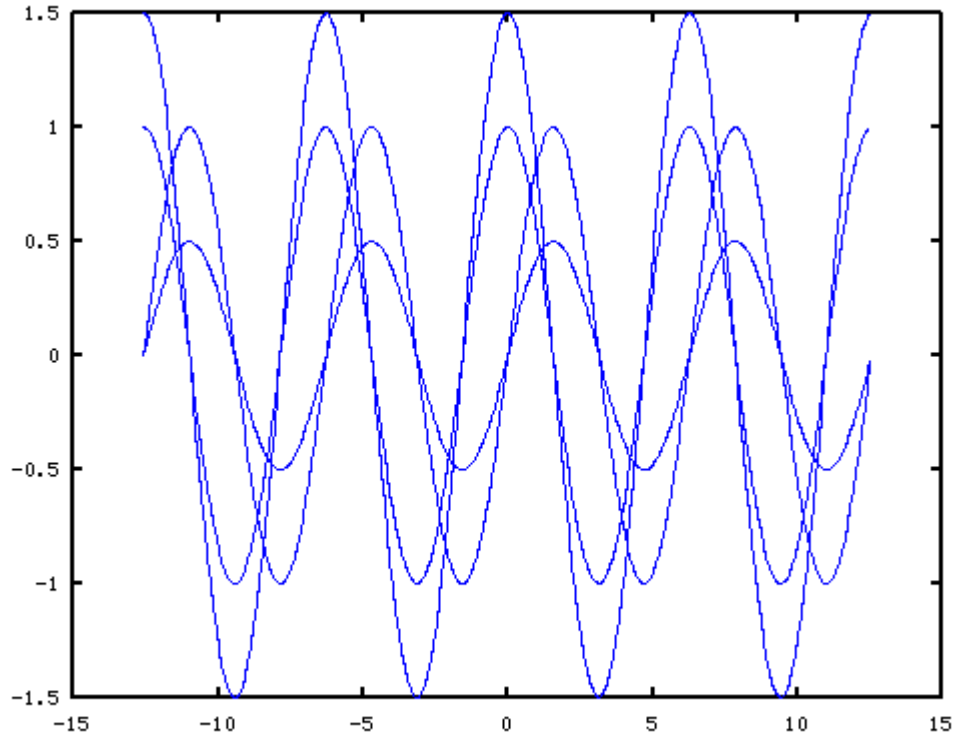
Построение графиков функций первый способом представлено на листинге 4.2, вторым – на листинге 4.3. Получившиеся графики функций представлены на рис. 4.2.

```
x=-4*pi:0.1:4*pi;
v=sin(x); w=cos(x); r=sin(x)/2; p=1.5*cos(x);
plot(x,v,x,w,x,r,x,p);
```

Листинг 4.2.

```
x=-4*pi:0.1:4*pi;
v=sin(x);plot(x,v);
hold on;
v=cos(x);plot(x,v);
v=sin(x)/2;plot(x,v);
v=1.5*cos(x);plot(x,v);
```

Листинг 4.3




```
18.0246,  1.26266
```

Рис. 4.2. Графики функций $v=\sin x,\ w=\cos x,\ r=\sin\frac{x}{2},\ p=\frac{3}{2}\cos x$

Обратите внимание, что при построении графиков первым способом Octave автоматически изменяет цвета изображаемых в одной системе координат графиков. Однако управлять цветом и видом каждого из изображаемых графиков может и пользователь, для чего необходимо воспользоваться полной формой 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 форматов, определяющая параметры линии и при необходимости, позволяющая вывести легенду.

В строке могут участвовать символы, отвечающие за тип линии, маркер, его размер, цвет линии и вывод легенды. Попробуем разобраться с этими символами. За сплошную линию отвечает символ «-». За маркеры отвечают следующие символы (табл. 4.1).

Цвет линии определяется буквой латинского алфавита (табл. 4.2), можно использовать и цифры, но на взгляд авторов использование букв более логично (их легче запомнить по английским названиям цветов)

Таблица 4.1: Символы маркеров

Символ маркера	Изображение маркера
.	точка
*	✳
x	×
+	+
o	⊙
s	■
d	◆
v	▼
^	▲
<	▽
>	△
p	⊡
h	◇

Таблица 4.2: Цвета линии

Символ	Цвет линии
y	желтый
m	розовый
c	голубой
r	красный
g	зеленый
b	синий
w	белый

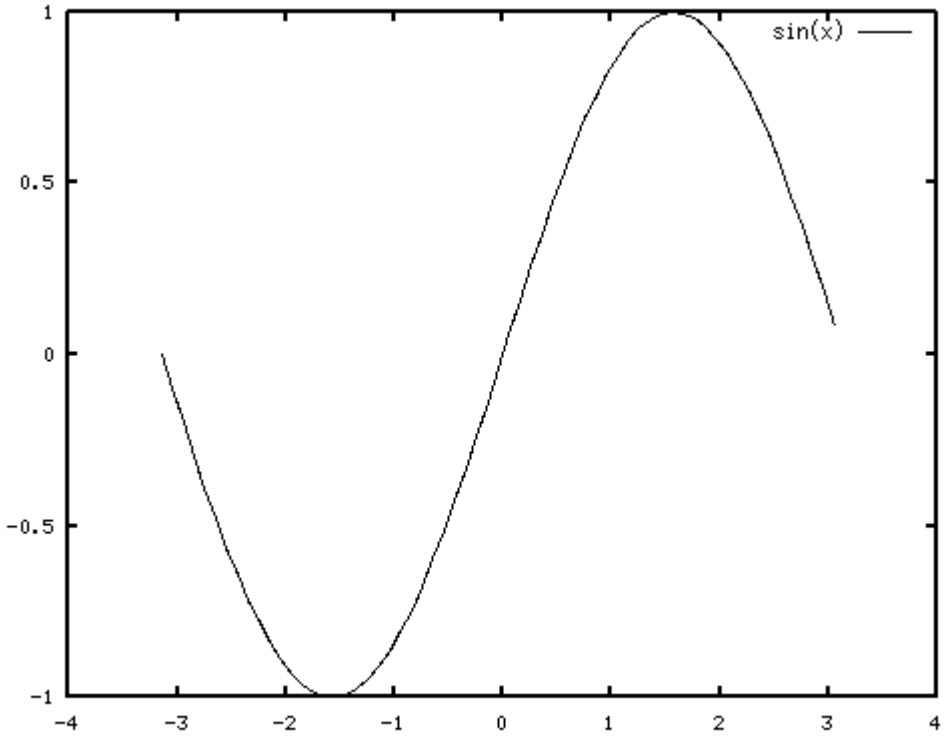
При определении строки, отвечающей за вывод линии, следует учитывать следующее:

- не важен порядок символа цвета и символа маркера;
- если присутствует символ «-», то линия всегда будет сплошная, при этом, если присутствует символ маркера, то все изображаемые точки еще будут помечаться маркером, если символа маркера — нет, то соседние точки просто будут соединяться линиями;
- если символ «-» отсутствует, то линия может быть, как сплошная, так и точечная; это зависит от наличия символа маркера, если символа маркера нет, то будет сплошная линия, иначе – точеная.

Если необходима легенда для графика, то ее следует включить в строку форматов, заключенную в символы «;». Например, команда

```
plot(x=-pi:0.1:pi,sin(x),"-k;sin(x);")
```

выведет на экран график функции *y=sin(x)* черного цвета на интервале [*-π;π*] с легендой «*sin(x)*» (рис. 4.3)



5.00819, -0.467785

Рис. 4.3. Результаты работы функции plot(x=-pi:0.1:pi,sin(x),"-k;sin(x);")

Пользователь может управлять и величиной маркера, для этого после строки форматов следует указать имя параметра "markersize" (размер маркера) и через запятую величину — целое число), определяющее размер маркера на графике.

Например, команда

```
plot(x=-pi:0.1:pi,sin(x),"-ok;sin(x);", "markersize",4);
```

выведет на экран график, представленный на рис. 4.4.

Для того, чтобы чтобы вывести график в новом окне, перед функцией `plot`, следует вызвать функцию `figure()`.

Внимание!!! При работе с графиками в Octave **необходимо понимать следующее:** щелчок по кнопке закрытия окна с графиками приводит не к закрытию окна, а к его скрытию. При повторном вызове команды рисования графиков происходит восстановление окна, в котором и изображаются графики. Корректно закрыть графическое окно можно только программно. Создается графическое окно функцией

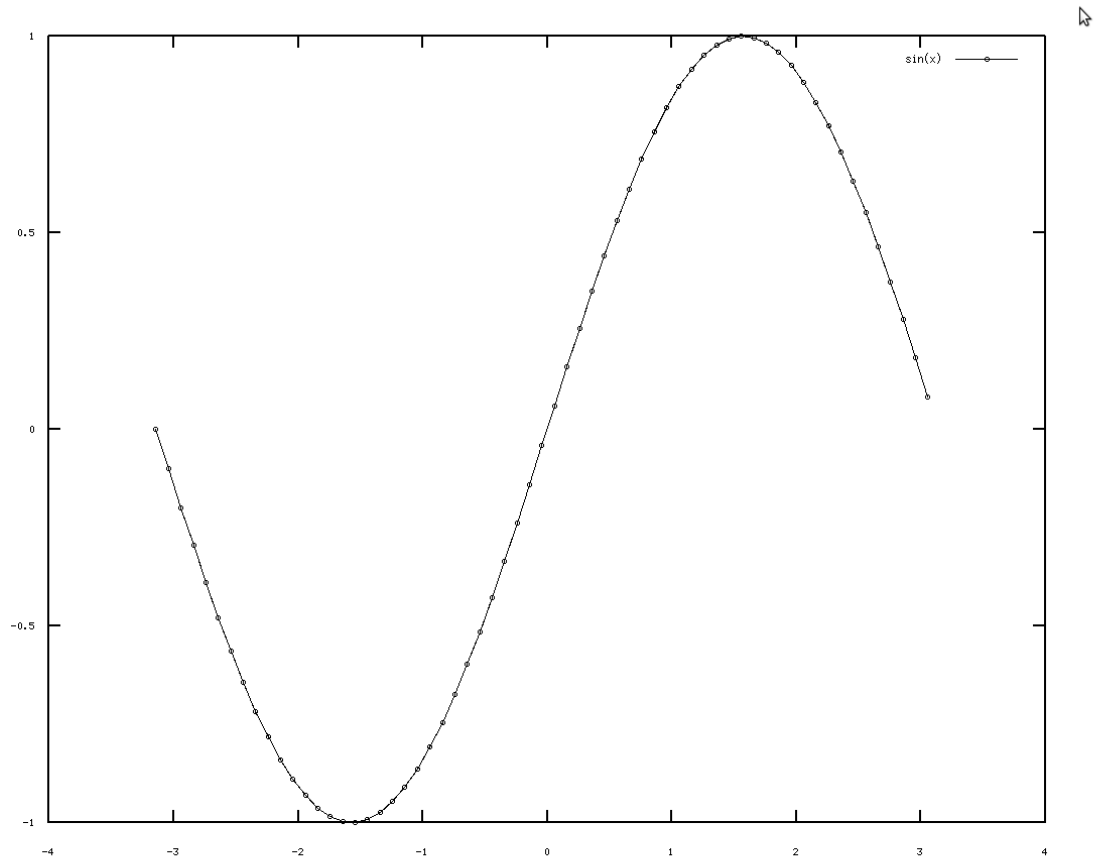
```
h = figure();
```

где, `h` — переменная, в которой будет храниться дескриптор[7] (номер) окна. Закрыть окно можно с помощью функции `delete(h)`, где `h` — имя дескриптора закрываемого окна.

Перед функцией закрытия окна можно применить функцию `pause(n)`, которая

7 Для дальнейших операций с окном надо будет использовать именно переменную, в которой храниться дескриптор.

приостанавливает выполнение программы на `n` секунд.



4.27473, 1.06952

Рис. 4.4. Работа функции plot(x=-pi:0.1:pi,sin(x),"-ok;sin(x);", "markersize",4);

Octave представляет дополнительные возможности для оформления графиков:

• команда `grid on` (`grid`) наносит сетку на график, `grid off` убирает сетку с графика;

• функция `axis[xmin, xmax, ymin, ymax]` выводит только часть графика, определяемую прямоугольной областью $x_{min} \le x \le x_{max}$, $y_{min} \le y < y_{max}$;

• функция `title('Заголовок')` предназначена для вывода заголовка графика;

• функции `xlabel('Подпись под осью x')`, `ylabel('Подпись под осью y')` служат для подписей осей *x* и *y* соответственно;

• функция `text(x,y,'текст')` выводит `текст` левее точки с координатами *(x,y)*;

• функция `legend('легенда1', 'легенда2', …, 'легендаn', m)` выводит легенды для каждого из графиков, параметр m определяет месторасположение легенды в графическом окне: 1 – в правом верхнем углу графика (значение по умолчанию); 2 – в левом верхнем углу графика; 3 – в левом нижнем углу графика; 4 – в правом нижнем углу графика.

При выводе текста с помощью функций `xlabel`, `ylabel`, `title`, `text` можно выводить греческие буквы[8] (табл. 4.3), использовать символы верхнего и нижнего индекса. Для вывода текста в верхнем индексе используется символ «^», в нижнем — символ «_». Например, для вывода $e^{\cos(x)}$ необходимо будет ввести текст `e^{cos(x)}`, а для вывода

8 Для вывода греческих букв и кириллицы в вашей операционной системе должны быть установлены соответствующие шрифты.

x_{min} — текст `x_{min}`. При работе с текстом можно также использовать синтаксис TEX.

Таблица 4.3: Греческие буквы

Команда	**Символ**	**Команда**	**Символ**
\alpha	α	\upsilon	υ
\beta	β	\phi	φ
\gamma	γ	\chi	χ
\delta	δ	\psi	ψ
\epsilon	ε	\omega	ω
\zeta	ζ	\Gamma	Γ
\eta	η	\Delta	Δ
\theta	θ	\Theta	Θ
\iota	ι	\Lambda	Λ
\kappa	κ	\Xi	Ξ
\lambda	λ	\Pi	Π
\mu	μ	\Sigma	Σ
\nu	ν	\Upsilon	Υ
\xi	ξ	\Phi	Φ
\pi	π	\Psi	Ψ
\rho	ρ	\Omega	Ω
\sigma	σ	\forall	∀
\varsigma	ς	\exists	∃
\tau	τ	\approx	≈
\int	∫	\in	Π
\wedge	^	\sim	~
\vee	v	\leq	≤
\pm	±	\leftrightarrow	↔
\geq	≥	\leftarrow	←
\inftyo	∞	\uparrow	↑
\partial	∂	\rightarrow	→
\neq	≠	\downarrow	↓
\nabla	Δ	\circ	º

После описания основных возможностей по оформлению графиков рассмотрим еще несколько примеров построения графиков.

ЗАДАЧА 4.3. Последовательно вывести в графическое окно графики функций

$$y=\sin x,\ y=\sin\frac{3x}{4},\ y=\cos x,\ y=\cos\frac{x}{3}$$

с задержкой 5 секунд. Текст программы решения задачи на Octave с комментариями приведен

на листинге 4.4.

```
% Создаём графическое окно с дескриптором okno1.
okno1=figure();
% Определяем аргумент (массив x) на интервале [-6π;6π].
x=-6*pi():pi()/50:6*pi();
%Вычисляем значение функции sin(x).
y=sin(x);
%Выводим график функции sin(x) чёрного цвета.
plot(x,y,'k');
%Выводим линии сетки.
grid on;
%Выводим заголовок графика.
title('Plot y=sin(x)');
%Приостанавливаем выполнение программы на 5 секунд.
pause(5);
y=sin(0.75*x);
%Выводим график функции sin(0.75x) голубого цвета.
plot(x,y,'b');
%Выводим линии сетки.
grid on;
%Выводим заголовок графика.
title('Plot y=sin(0.75x)');
%Приостанавливаем выполнение программы на 5 секунд.
pause(5);
y=cos(x);
%Выводим график функции cos(x) красного цвета.
plot(x,y,'r');
%Выводим линии сетки.
grid on;
%Выводим заголовок графика.
title('Plot y=cos(x)');
%Приостанавливаем выполнение программы на 5 секунд.
pause(5);
y=cos(x/3);
%Выводим график функции cos(x/3) зеленого цвета.
plot(x,y,'g');
%Выводим линии сетки.
grid on;
%Выводим заголовок графика.
title('Plot y=cos(x/3)');
%Приостанавливаем выполнение программы на 5 секунд.
pause(5);
%Закрываем окно с дескриптором okno1.
delete(okno);
```

Листинг 4.4

При запуске программы будет создано графическое окно, в котором будет выведен график функции *sin(x)* черного цвета с линиями сетки и заголовком, которое будет на экране в течении 5 секунд, после этого окно очиститься. Будет выведен график функции *sin(0.75x)* голубого цвета с линиями сетки и заголовком, которое будет на экране в течении 5 секунд. Далее аналогично будут с задержками 5 секунд выведены графики функций *y=cos(x)* и *y=cos(x/3)*. После этого окно автоматически закроется. Для понимания механизма работы с

графическими окнами в Octave, авторы рекомендуют читателю при выводе графика *sin(x)* попытаться закрыть окно с помощью мыши и посмотреть, что из этого получится.

ЗАДАЧА 4.4. Построить графики функций

$$y=e^{\sin x}, u=e^{\cos\frac{x}{3}}, v=e^{\sin\frac{x}{2}}$$

на интервале $[-3\pi;3\pi]$.

Рассмотрим два варианта построения графиков (листинги 4.5, 4.6).

```
x=-3*pi():pi()/20:3*pi();% Формируем массив x.
y=exp(sin(x)); % Формируем массив y.
u=exp(cos(x/3));% Формируем массив u.
v=exp(sin(x/2));% Формируем массив v.
%Строим график функции y(x), сплошная чёрная линия,
% без маркера, качестве легенды выводим  e^{sin(x)} .
plot(x, y, "k;e^{sin(x)};")
%Блокируем режим очистки окна.
hold on;
%Строим график функции u(x), сплошная чёрная линия,
%с маркером «треугольник», размер маркера — 4,
%в качестве легенды выводим  e^{cos(x/3)} .
plot( x, u, "->k; e^{cos(x/3)};", "markersize", 4)
%Строим график функции v(x), сплошная чёрная линия,
%с маркером «окружность», размер маркера — 4,
%в качестве легенды выводим  e^{sin(x/2)} .
plot( x,v,"-ok;e^{sin(x/2)};", "markersize",4);
```

Листинг 4.5

```
x=-3*pi():pi()/20:3*pi();
y=exp(sin(x));
u=exp(cos(x/3));
v=exp(sin(x/2));
%Отличие вывода трёх графиков состоит в том, вместо 3-х
%функций plot и двух hold on используется одна функция plot,
%в которой указаны те же параметры вывода графиков,
%что и на листинге 4.6
plot(x,  y,  "k;e^{sin(x)};",  x,  u,  "->k;  e^{cos(x/3)};",
"markersize", 4, x,v,"-ok;e^{sin(x/2)};", "markersize",4);
```

Листинг 4.6

Оба способа формируют один и тот же график (рис. 4.5).

ЗАДАЧА 4.5. Построить график функции $y(x)=1-\frac{0.4}{x}+\frac{0.05}{x^2}$ на интервале *.[-2;2]*. В связи с тем, что функция не определена в точке *x=0*, будем строить её, как графики двух функций *y(x)* на интервале *[-2;0)* и *y(x)* на интервале *(0;2]*. Текст программы на Octave с комментариями приведён на листинге 4.7, график функции — на рис. 4.6.

```
a=1;d=-0.4;c=0.05;
% Определяем аргумент (массив x) на интервале [-2;-0.1].
x1=-2:0.01:-0.1;
% Определяем аргумент (массив x) на интервале [0.1;2].
x2=0.1:0.01:2;
```

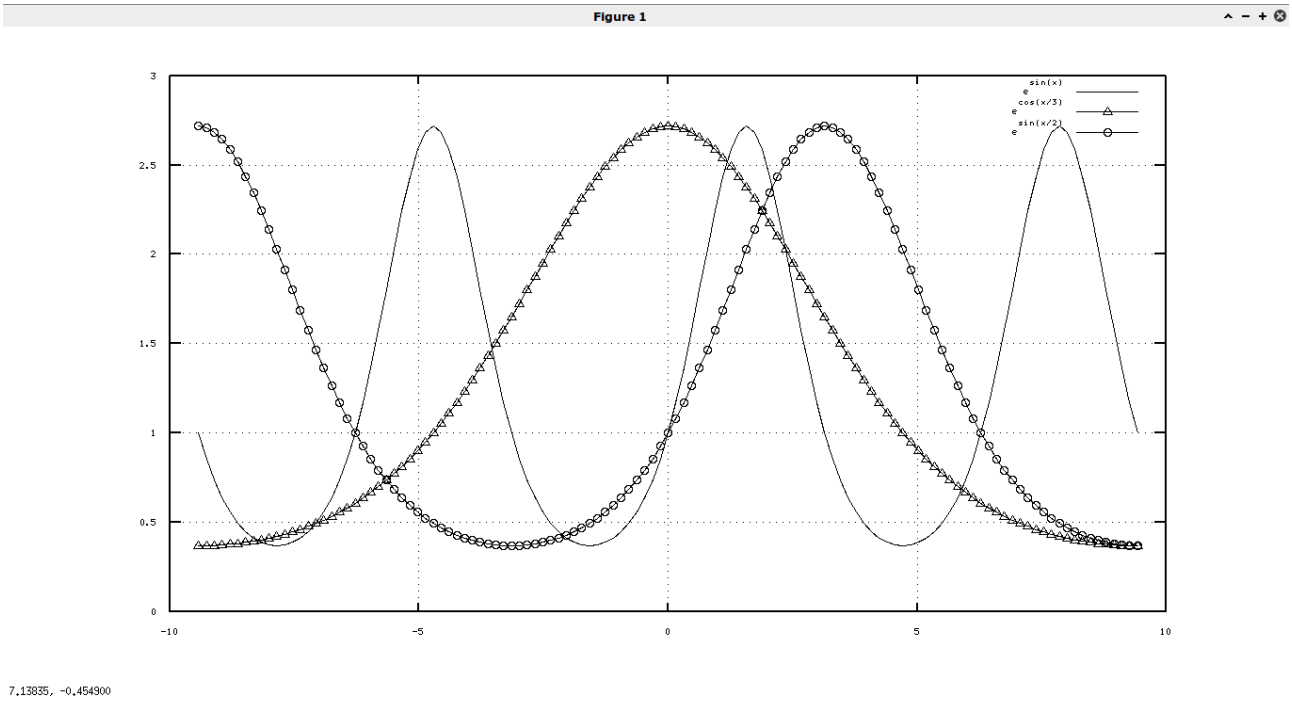


Рис. 4.5. Графики функций $y=e^{\sin x}, u=e^{\cos\frac{x}{3}}, v=e^{\sin\frac{x}{2}}$

```
%Вычисляем значение функции y(x) на интервале [-2;-0.1].
y1=a+b./x1+c./x1./x1;
%Вычисляем значение функции y(x) на интервале [0.1;2].
y2=a+b./x2+c./x2./x2;
% Строим график  y(x)=1-0.4/x+0.05/x^2  как график двух функций,
% на интервалах [-2;-0.1], [0.1;2], цвет графика чёрный,
% Легенда -  f(x)=a+b/x+c/x^2  .
plot(x1,y1,'k;f(x)=a+b/x+c/x^2;',x2,y2,'k');
title('y=f(x)');% Подпись над графиком.
xlabel('X'); % Подпись оси X.
ylabel('Y');% Подпись оси Y.
grid on; % Рисуем линии сетки.
```

Листинг 4.7

ЗАДАЧА 4.6. Построить график функции

$$y(x)=\frac{1}{x^2-2x-3}$$

на интервале [-5;7].

Уравнение $x^2-2x-3=0$ имеет корни *-1*, *3*. Поэтому наша функция *y(x)* будет иметь разрывы в этих точках *x=-1*, *x=3*. Будем строить её, как графики трёх функций, на трёх интервалах [*-5;-1.1*], [*0.9;2.9*], [*3.1;7*]. Листинг 4.8 демонстрирует решение примера 4.5. На рис. 4.7 изображен график функции $y(x)=\frac{1}{x^2-2x-3}$, который получился в результате работы программы, представленной на листинге 4.8.

```
%Определяем аргументы на интервалах [-5;-1.1], [0.9;2.9],
% [3.1;7].
x1=-5:0.01:-1.1;
x2=-0.9:0.01:2.9;
x3=3.1:0.01:7;
```

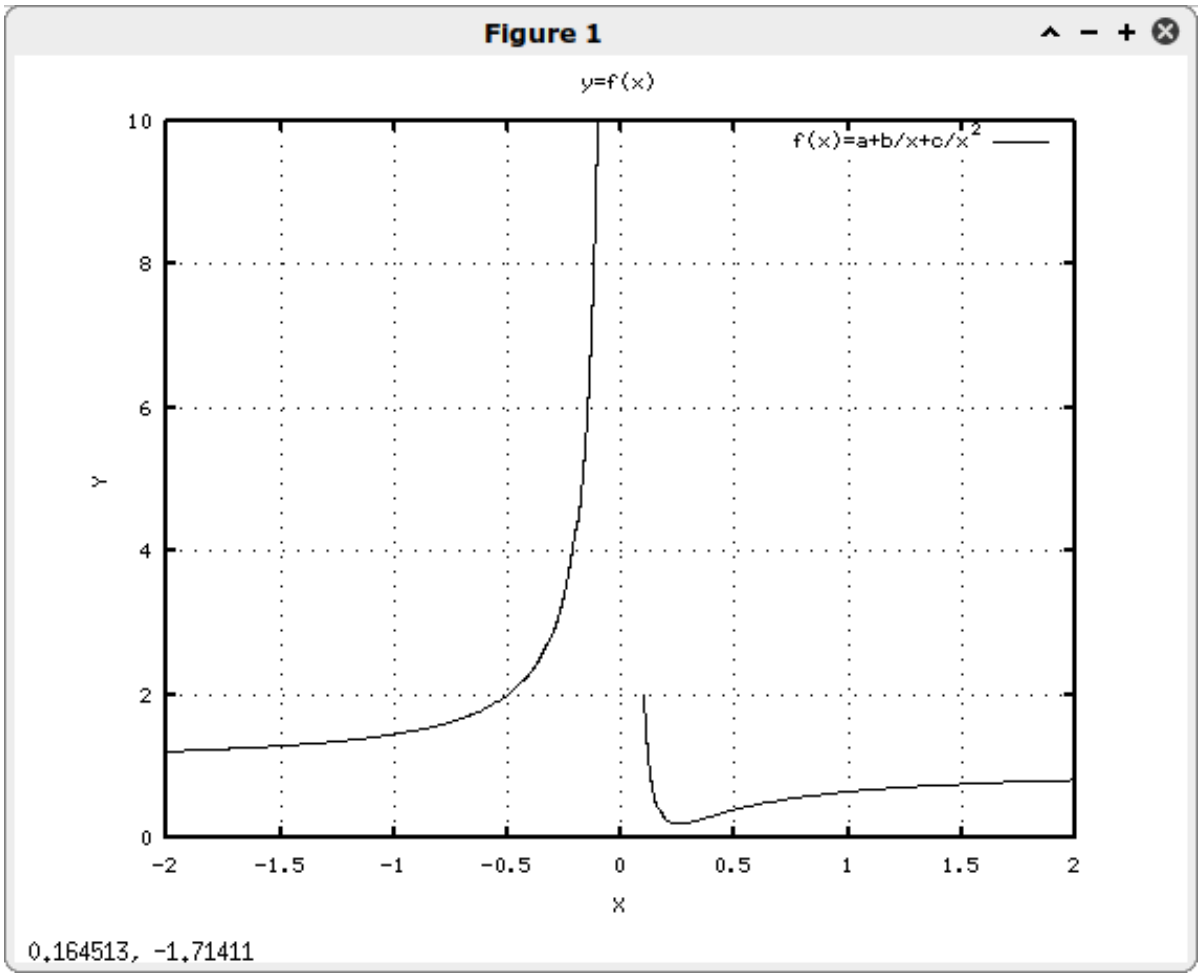


Рис. 4.6. График функции $y(x)=1-\frac{0.4}{x}+\frac{0.05}{x^2}$

```
%Вычисляем значение y(x) на соответствующих интервалах.
y1=1./(x1.*x1-2*x1-3);
y2=1./(x2.*x2-2*x2-3);
y3=1./(x3.*x3-2*x3-3);
%Строим график  y(x)=1/(x^2-2x-3)  черного цвета,
%как график 3-х функций.
plot(x1,y1,'k',x2,y2,'k', x3,y3,'k');
title('y=f(x)'); % Подпись над графиком.
xlabel('X'); % Подпись оси X.
ylabel('Y');.% Подпись оси Y.
legend('f(x)=1/(x^2-2x-3)',4);% Вывод легенды.
grid on; % Рисуем линии сетки.
```

Листинг 4.8

Еще одну интересную возможность построения двух графиков предоставляет функция `plotyy` , которая позволяет изображать на графике две оси ординат, что очень удобно при построении графиков разных порядков. К сожалению эта функция не позволяет явно задавать типы изображаемых линий. На каждой из осей ординат подписи значений выводятся тем же цветом, что и график функции. Использование функции `plotyy` рассмотрено ниже.

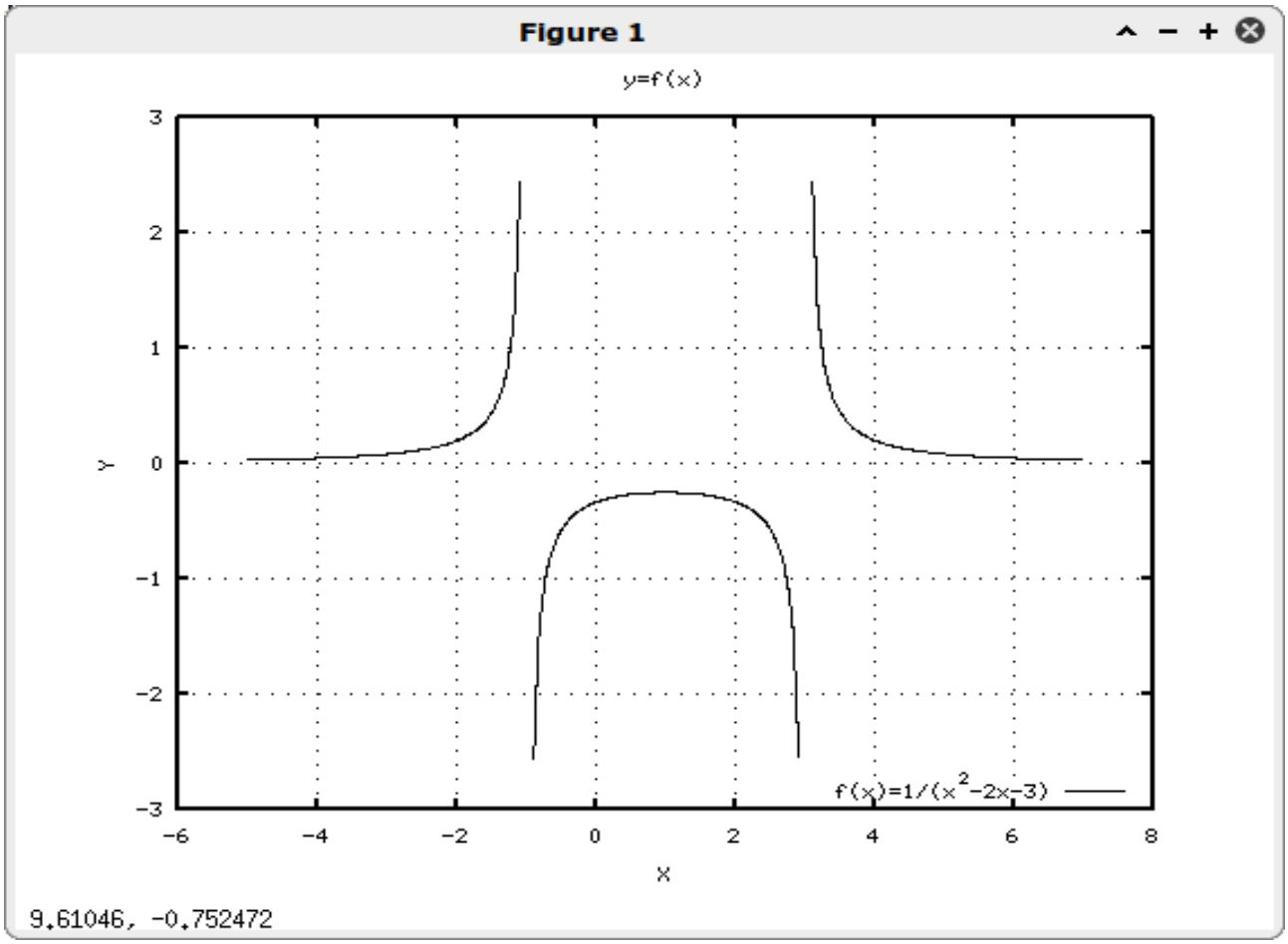


Рис. 4.7. График функции $y(x)=\dfrac{1}{x^2-2x-3}$

ЗАДАЧА 4.7. Построить графики функции $y=x^3$, $y=\cos 2x$ на интервале *[-5;5]*.

Решение этой задачи представлено на листинге 4.9, полученный график – на рис. 4.8.

```
x=-5:0.1:5;
y=x.^3;
z=cos(2*x);
plotyy(x,y,x,z);
grid on;
title('Plot x^3, cos(x/2)');
xlabel('X');
ylabel('Y');
```

Листинг 4.9.

Octave предоставляет возможность построить несколько осей в графическом окне и вывести на каждую из них свои графики. Для этого следует использовать функцию:

```
subplot(row, col, cur);
```

Параметры `row` и `col` определяют количество графиков по вертикали и горизонтали соответственно, `cur` определяет номер текущего графика. Повторное обращение к функции `subplot` с теми же значениями `row` и `col` позволяет просто изменять номер текущего графика и может использоваться для переключения между графиками. Рассмотрим использование функции `subplot` при решении следующей задачи.

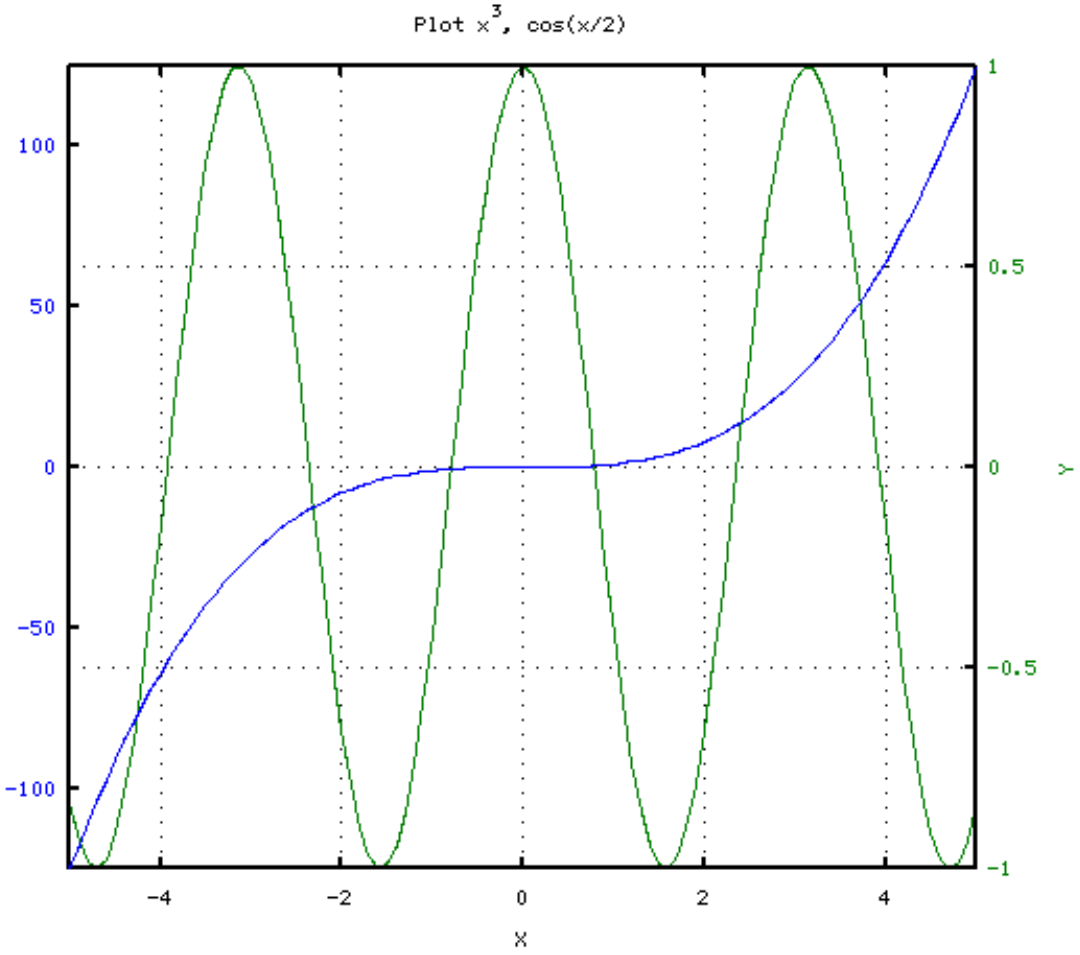


Рис. 4.8: Графики функции $y=x^3, y=\cos 2x$

ЗАДАЧА 4.8. Построить графики функции

$$y=2e^{-0.15x^2}, z=e^{0.7x-0.25x^2}, u=0.5e^{-0.33x}\sin\left(2x+\frac{\pi}{3}\right), k=3\sin(x-0.22x^2), v=\cos x, w=e^{\cos x}$$

на интервале [-6;6]. Решение задачи представлено на листинге 4.10, полученное графическое окно – на рис. 4.9.

```
%Формируем массивы x,y,z,u,k,v,w.
x=-6:0.2:6;
y=2*exp(-0.15*x.^2); z=exp (0.7*x-0.25*x.^2)
u=0.5*exp (-0.33*x).*sin (2*x+pi ()/3);
k=3*sin (x-0.22.*x.^2); v=cos(x); w=exp (cos (x));
%Делим окно на 6 частей и объявляем 1-й график текущим.
subplot(3,2,1);
% Строим график y(x) с линиями сетки и подписями.
plot(x,y); grid on;
title('Plot y=2e^{-0.15x^2}');
xlabel('x'); ylabel('y');
% Второй график объявляем текущим.
Subplot(3,2,2);
% Строим график z(x) с линиями сетки и подписями.
plot(x,z); grid on;
```

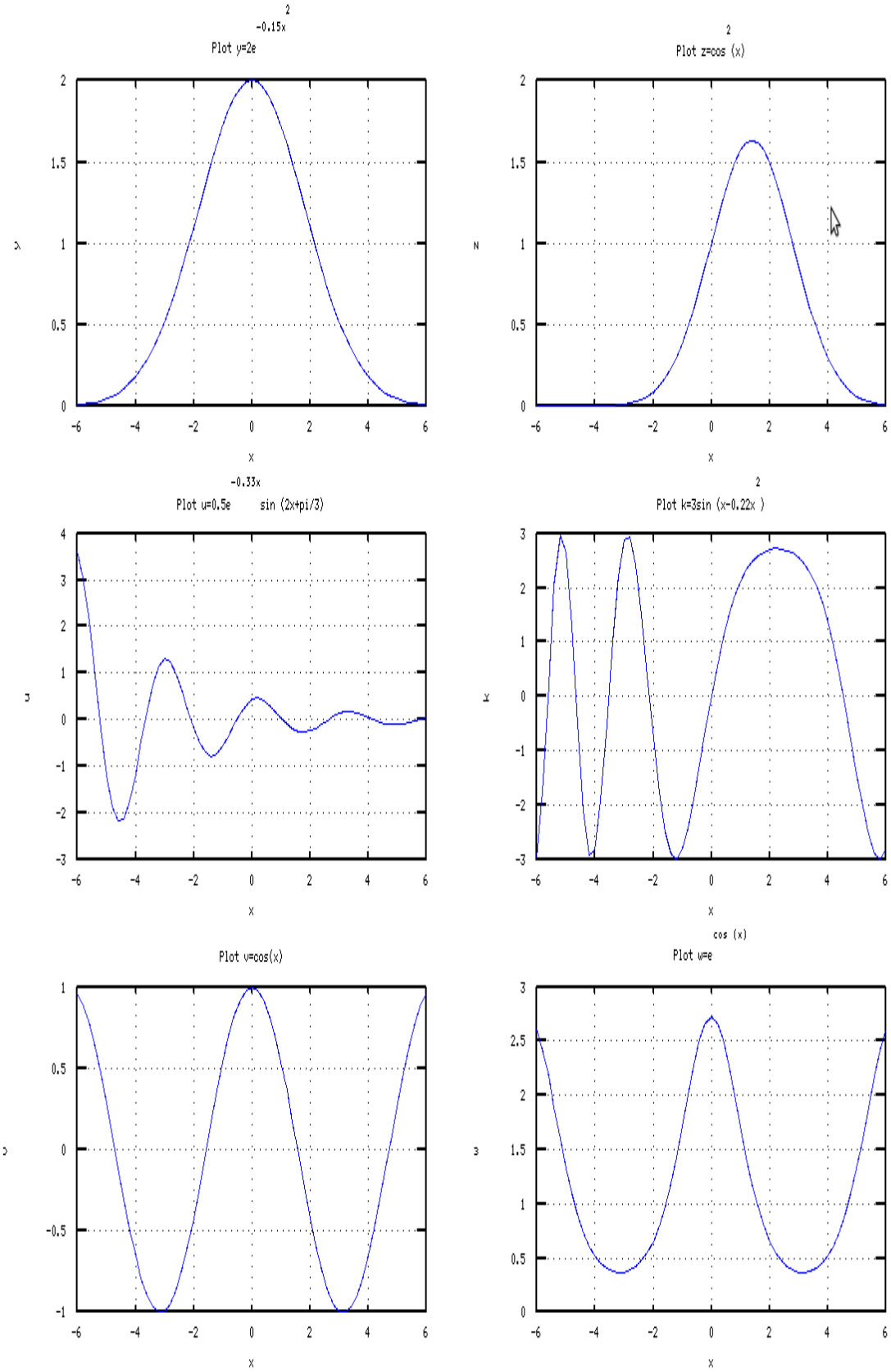


Рис. 4.9. Графики функций y(x), z(x), u(x), k(x), v(x), w(x)

```
title('Plot z=cos^2(x)');
xlabel('x'); ylabel('z');
```

```
% Третий график объявляем текущим.
subplot(3,2,3);
% Строим график u(x) с линиями сетки и подписями.
plot(x,u); grid on;
title('Plot u=0.5e^{-0.33x}sin (2x+pi/3)');
xlabel('x'); ylabel('u');
% Четвертый график объявляем текущим.
subplot(3,2,4);
% Строим график k(x) с линиями сетки и подписями.
plot(x,k); grid on;
title('Plot k=3sin (x-0.22x^2)');
xlabel('x'); ylabel('k');
% Пятый график объявляем текущим.
subplot(3,2,5);
% Строим график v(x) с линиями сетки и подписями.
plot(x,v); grid on;
title('Plot v=cos(x)');
xlabel('x'); ylabel('v');
% Шестой график объявляем текущим.
subplot(3,2,6);
% Строим график w(x) с линиями сетки и подписями.
plot(x,w); grid on;
title('Plot w=e^{cos (x)}');
xlabel('x'); ylabel('w');
```

Листинг 4.10

Для построения графика функции можно использовать функцию

fplot(@f,[xmin, xmax],s).

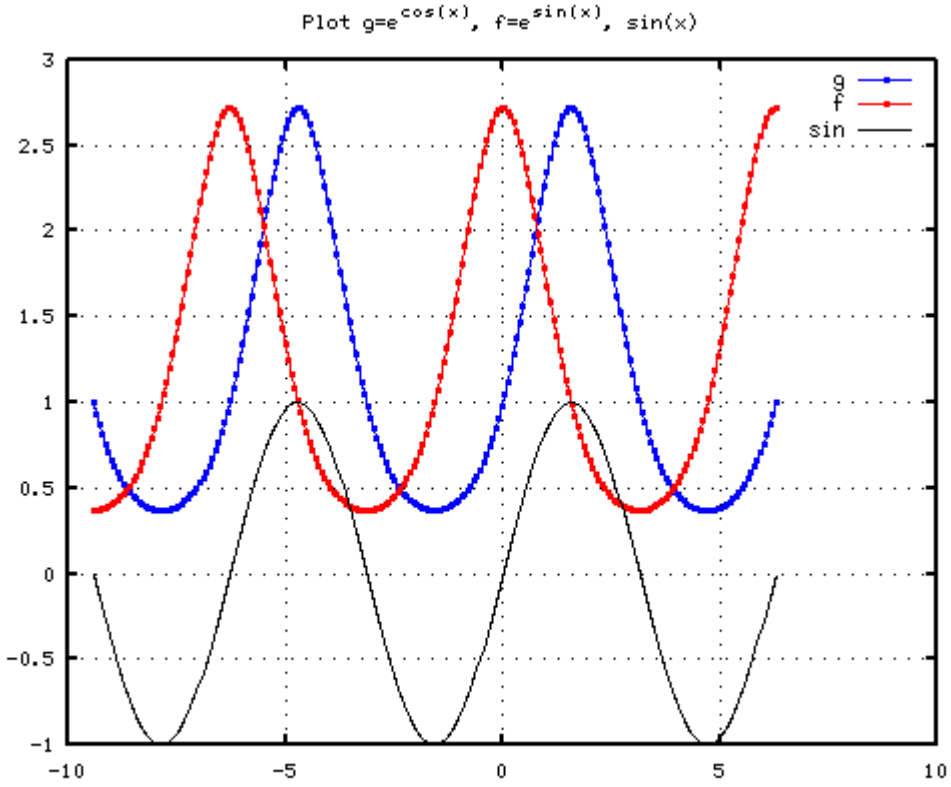
Здесь `f` — имя функции (стандартной функции Octave или функции, определенной пользователем), `[xmin, xmax]` — интервал, на котором будет строиться график, `s` — строка формата, определяющая только параметры линии (но не легенду). Легенда графика формируется автоматически функцией `fplot` без использования функции `legend`. Не позволяет формировать легенду и третий параметр функции `fplot`, в отличии от функции `plot`.

ЗАДАЧА 4.9. Используя функцию `fplot`, построить графики функций $e^{\sin x}, e^{\cos x}, \sin x$ на интервале [*-3π;2π*]. Текст программы на языке Octave с комментариями приведен на листинге 4.11. Полученные графики можно увидеть на рис. 4.10

```
% Определяем функцию f=exp(cos(x))
function y=f(x)
y=exp(cos(x));
end
% Определяем функцию g=exp(cos(x))
function z=g(x)
z=exp(sin(x));
end
%Строим график g=exp(cos(x)) на интервале [-3п;2п] голубого
% цвета с маркером ⊡.
fplot (@g, [-3*pi(), 2*pi()],'-pb');
hold on;
%Строим график f=exp(sin(x)) на интервале [-3п;2п],
% красного цвета с маркером ⊙.
```

```
fplot (@f, [-3*pi(), 2*pi()],'-or');
%Строим график sin(x) на интервале [-3π;2π] чёрного цвета
fplot (@sin, [-3*pi(), 2*pi()],'k');
%Подписываем график.
title('Plot y=e^{cos(x)}, z=e^{cos(x)}, sin(x)');
% Проводим линии сетки
grid on;
```

Листинг 4.11



11.7580, 3.37519

Рис. 4.10. Графики функций $e^{\sin x}, e^{\cos x}, \sin x$

4.1.2 Построение графиков в полярной системе координат

Полярная система координат состоит из заданной фиксированной точки O, называемой полюсом, концентрических окружностей с центром в полюсе и лучей, выходящих из точки O, один из которых, OX, называют полярной осью. Положение любой точки M в полярных координатах можно задать положительным числом ρ=|OM| (полярный радиус) и числом φ, равным величине угла XOM (полярный угол). Числа ρ и φ называют полярными координатами точки M и обозначают M(ρ, φ).

Для формирования графика в полярной системе координат необходимо сформировать массивы значений полярного угла и полярного радиуса и обратиться к функции

```
polar(phi, ro, s)
```

где `phi` – массив полярного угла; `ro` – массив полярного радиуса; `s` – строка, состоящая из трех символов, которые определяют цвет линии, тип маркера и тип линии (табл. 4.1, 4.2).

В качестве примера рассмотрим решение следующей задачи.

ЗАДАЧА 4.10. Построить график лемнискаты.

Уравнение лемнискаты в полярных координатах имеет вид:

$$\rho = a\sqrt{2\cos 2\varphi} \quad .$$

Функция ρ определена при $-\frac{\pi}{4} \le \varphi \le \frac{\pi}{4}$ ($\cos 2\varphi \ge 0$), поэтому листинг для изображения графика лемнискаты будет таким:

```
% Определяем массив полярного угла fi.
fi=-pi/4:pi/200:pi/4;
%Определяем массив положительных значений
%полярного радиуса ro лемнискаты.
ro=3*sqrt(2*cos(2*fi));
% Рисуем правую часть лемнискаты.
polar(fi,ro,'r');
%Блокируем режим очистки окна.
hold on;
% Рисуем левую часть лемнискаты.
polar(fi,-ro,'r');
grid on;
```

Листинг 4.12

Полученный график изображен на рис. 4.11.

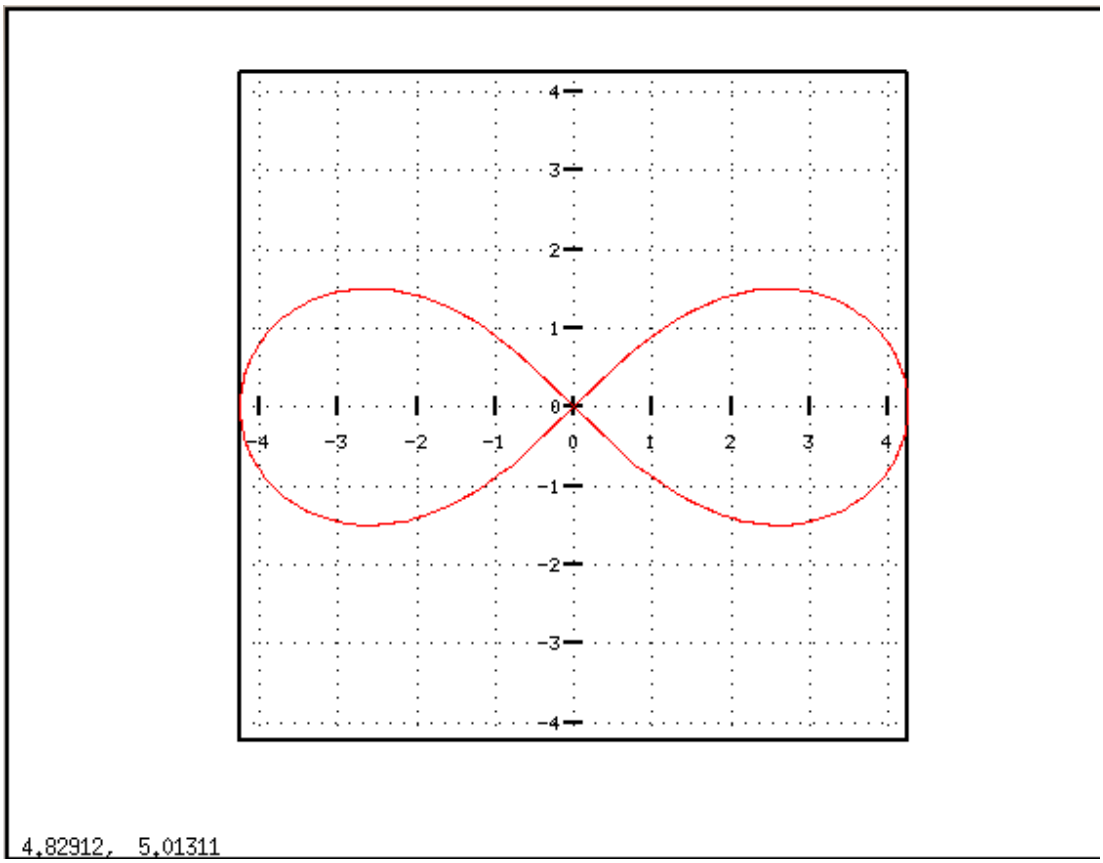


Рис. 4.11. График лемнискаты в полярных координатах

ЗАДАЧА 4.11. Построить графики архимедовой, гиперболической и логарифмической спиралей в полярных координатах.

Уравнение архимедовой спирали в полярных координатах – $\rho = a\varphi$, гиперболической – $\rho = \frac{a}{\varphi}$.

Соотношение $\rho = a e^{k\varphi}, k = ctg\,\alpha$ является уравнением логарифмической спирали в полярных координатах. Частным случаем логарифмической спирали ($\alpha = \frac{\pi}{2}, k = 0$) является уравнение окружности ($\rho = a$).

На листинге 4.13 приведён текст программы, позволяющей изобразить 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. 4.12.

```
h=figure()
clear all;
%Формируем массивы fi1,fi2,fi3,fi4,ro1,ro2,ro3,ro4.
fi1=0:pi/20:6*pi;
fi2=pi/3:pi/200:6*pi;
fi3=0:pi/20:4*pi;
fi4=-pi:pi/20:pi
ro1=4*fi1;
r2=0.5./fi2;
```

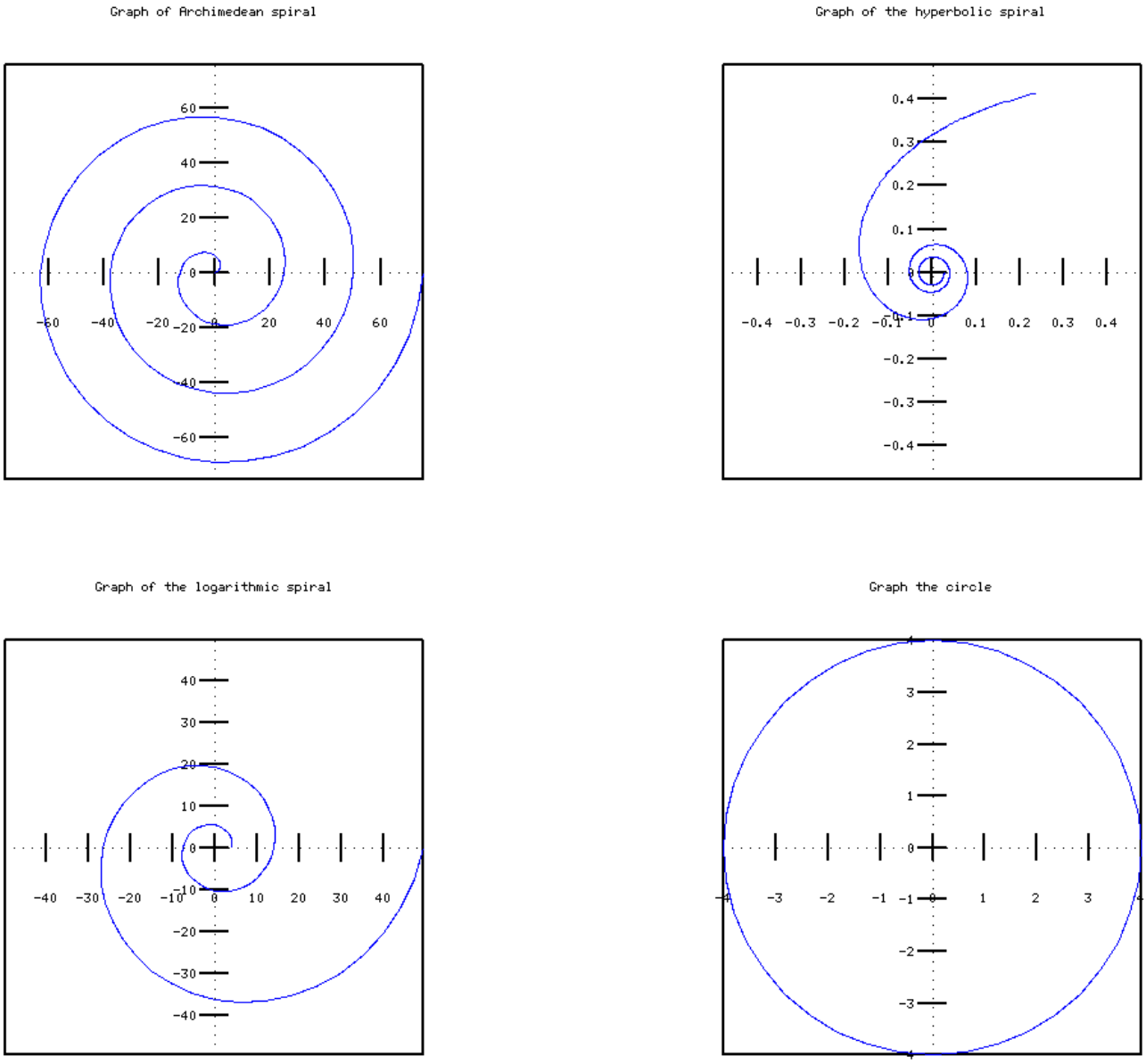


Рис. 4.12. Графики архимедовой , гиперболической и логарифмической спирали, окружности в полярных координатах

```
ro3=4*exp(0.2*fi3);
for i = 1:41
ro4(i)=4;
endfor
% Делим графическое окно на 4 части и объявляем первый график
% текущим.
subplot(2,2,1);
% Строим график архимедовой спирали.
polar(fi1,ro1);
% Подписываем график.
title('Graph of Archimedean spiral');
% Второй график объявляем текущим.
subplot(2,2,2);
% Строим график гиперболической спирали.
polar(fi2,r2);
% Подписываем график.
title('Graph of the hyperbolic spiral');
% Третий график объявляем текущим.
subplot(2,2,3);
% Строим график логарифмической спирали.
polar(fi3,ro3);
% Подписываем график.
title('Graph of the logarithmic spiral');
% Четвертый график объявляем текущим.
subplot(2,2,4);
% Строим график окружности.
polar(fi4,ro4);
% Подписываем график.
title('Graph the circle');
```

Листинг 4.13

4.1.3 Построение графиков, 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 функции *plot(x,y)*.

В качестве примера рассмотрим построение графика эпициклоиды и астроиды.

ЗАДАЧА 4.12. Построить график эпициклоиды. Уравнение эпициклоиды в параметрической форме имеет вид $x=4\cos t-\cos 4t,\ y=4\sin t-\sin 4t,\ t\in[0;2\pi]$. На листинге 4.14 представлен текст программы для изображения графика эпициклоиды, а на рис. 4.13 – сам график.

```
t=0:pi/50:2*pi;
x=4*cos(t)-cos(4*t);
y=4*sin(t)-sin(4*t);
plot(x,y);
grid on;
```

Листинг 4.14

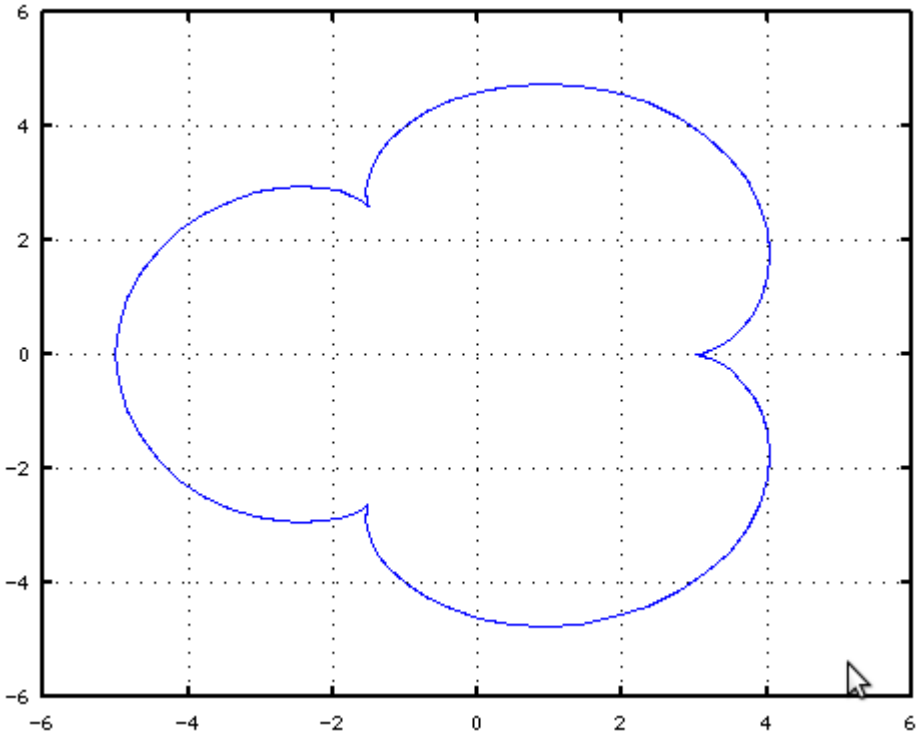


Рис. 4.13. График эпициклоиды

ЗАДАЧА 4.13. Построить график астроиды. Уравнение астроиды в параметрической форме имеет вид $x=3\cos^3 t, y=3\sin^3 t, t\in[0;2\pi]$. На листинге 4.15 представлен текст программы для изображения графика астроиды, а на рис. 4.14 – сам график.

```
t=0:pi/50:2*pi; x=3*cos(t).^3; y=3*sin(t).^3;
plot(x,y); grid on;
```

Листинг 4.15

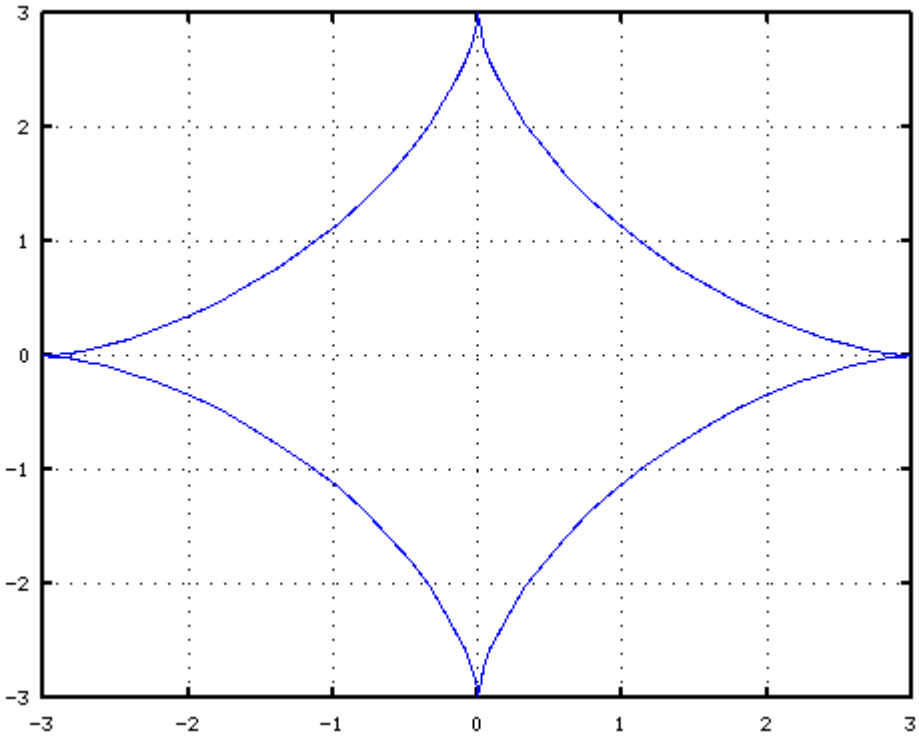


Рис. 4.14. График астроиды

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 которого должны быть упорядочены в порядке возрастания.

Рассмотрим несколько примеров.

Фрагмент

```
y=[5; 6;7; 8; 9; 8 ;7;6;4;3]; bar(y);
```

строит гистограмму, представленную на рис. 4.15.

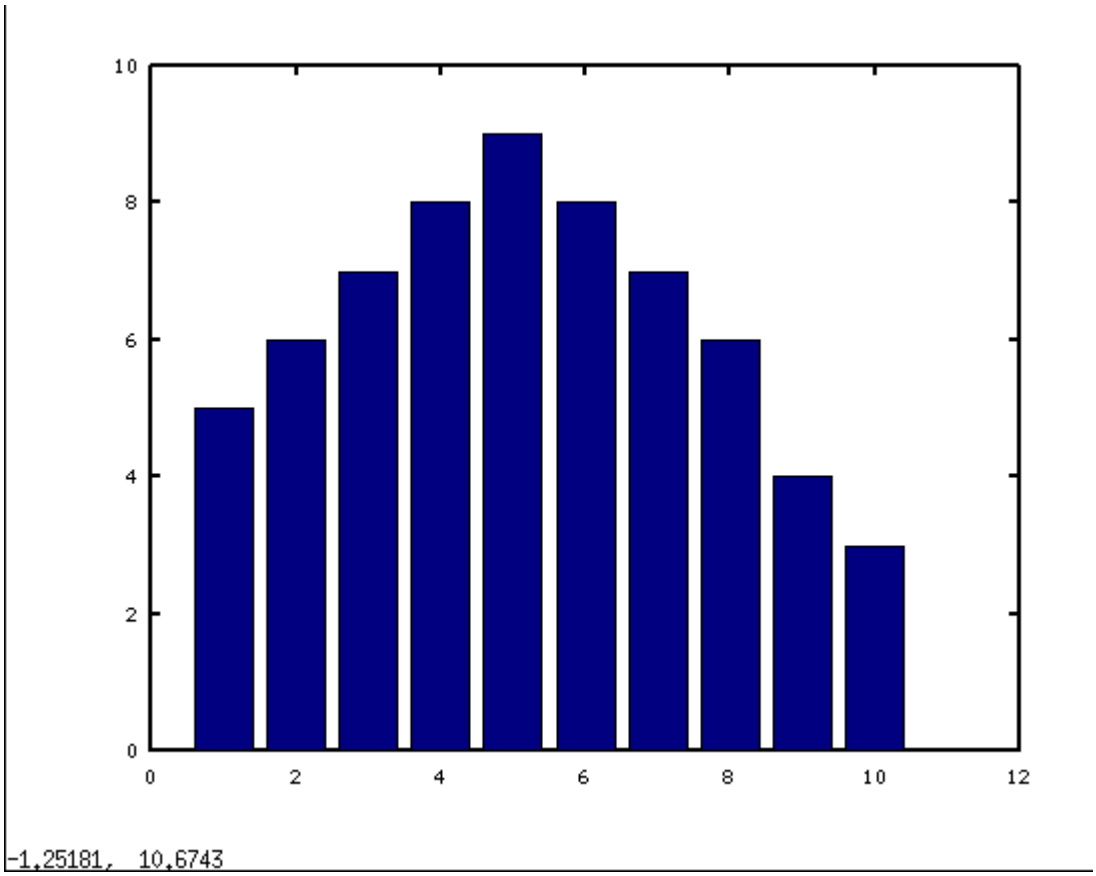


Рис. 4.15. Пример использования функции bar(y)

Фрагмент

```
x1=[-2,-1,0,1,2,3,4];y1=exp(sin(x1));bar(x1,y1);
```

строит гистограмму, представленную на рис. 4.16.

4.2 Построение трёхмерных графиков

График поверхности (трехмерный или 3D-график) - это график, положение точки в котором определяется значениями трех координат.

4.2.1 Построение графиков поверхностей

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 трех 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 трех координат и обозначается *M(x,y,z)*. Три плоскости, содержащие пары координатных осей, называются *координатными плоскостями* **XY**, **XZ** и **YZ**.

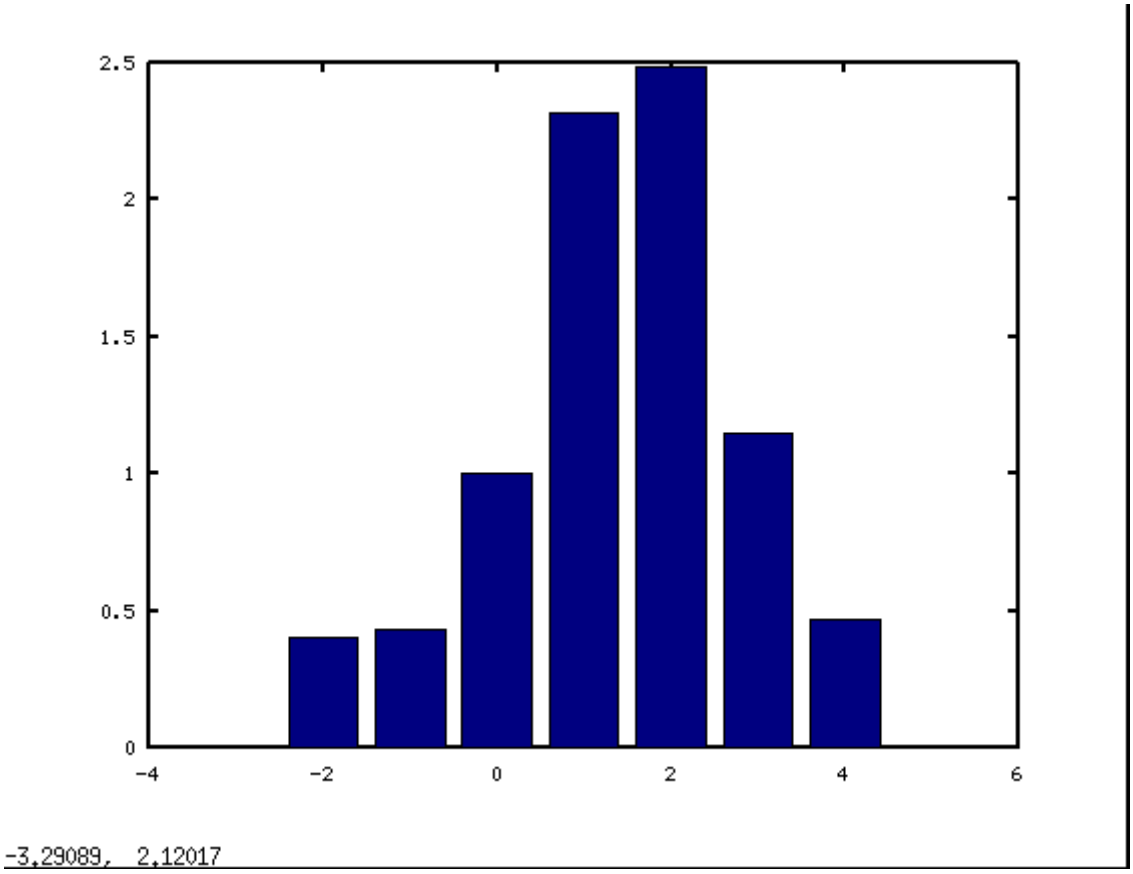


Рис. 4.16. Пример использования функции bar(x,y)

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 определенных 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

$$x_i = x_0 + ih,\ h = \frac{x_n - x_0}{n}\ ,\qquad i = 0,1,2,\ldots,n\ ,$$

$$y_j = y_0 + j\Delta,\ \Delta = \frac{y_k - y_0}{k}\ ,\quad j = 0,1,2,\ldots,k\ .$$

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.

Для формирования прямоугольной сетки в Octave есть функция `meshgrid`. Рассмотрим построение 3-х мерного графика на следующем примере.

ЗАДАЧА 4.14. 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)
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
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
```

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(x,y,z);
```

После это будет создано графическое окно с трехмерным графиком (рис. 4.17).

Как видно, график получился грубым. Для построения более точного графика сетку можно сделать более плотной (листинг 4.16 и рис. 4.18).

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

Листинг 4.16

Любой трёхмерный график можно вращать, используя мышку.

Для построения поверхностей, кроме функции `mesh`, которая строит каркасный график, есть функция `surf`, которая строит каркасную поверхность, заливая ее каждую клетку цветом. Цвет зависит от значения функции в узлах сетки.

ЗАДАЧА 4.15. С использованием функции `surf` построить график функции $z(x,y)=\sqrt{\sin^2 x+\cos^2 y}$.

На листинге 4.17 представлено решение задачи, а на рис. 4.19 изображён получившийся график.

```
[x y]=meshgrid(-2:0.2:2,0:0.2:4);
z=sqrt(sin(x).^2+cos(y).^2);
surf(x,y,z);
```

Листинг 4.17

В Octave можно построить графики двух поверхностей в одной системе координат, для этого, как и для плоских графиков следует использовать команду `hold on`, которая блокирует создание второго нового окна при выполнении команд `surf` или `mesh`.

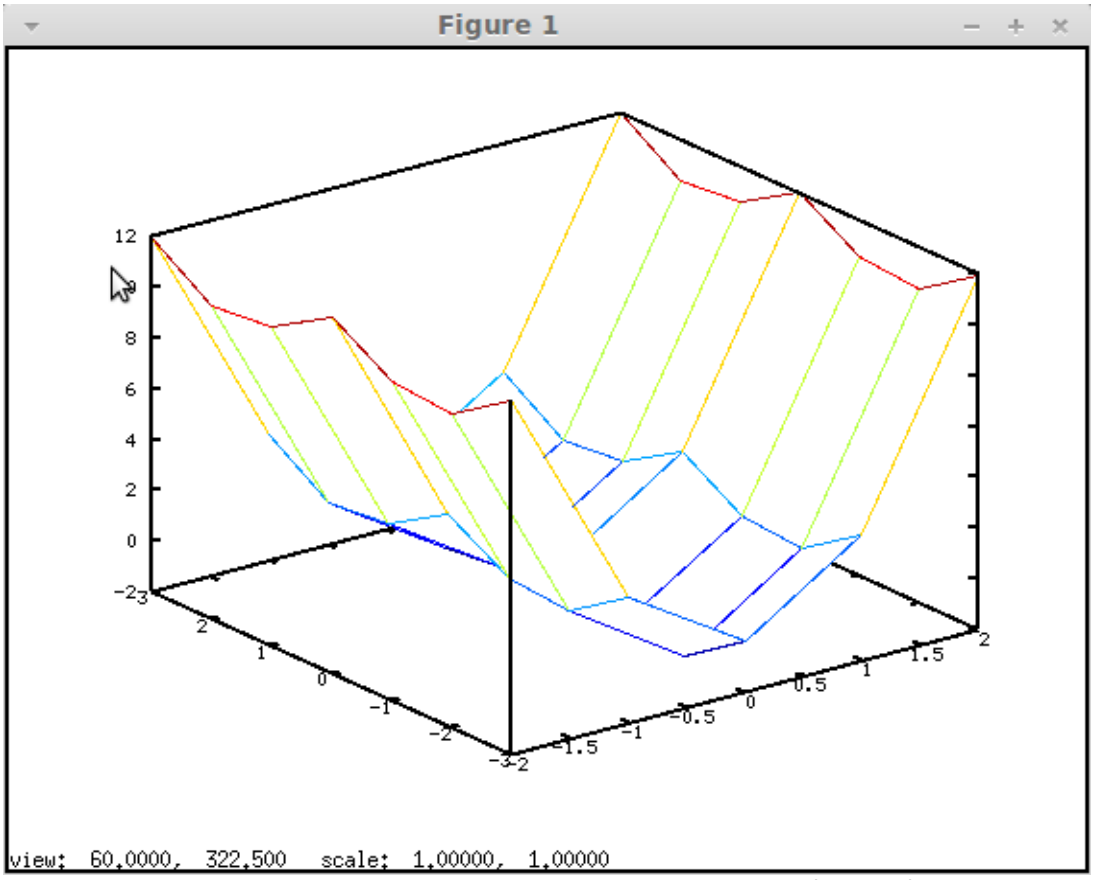


Рис. 4.17. График функции $z(x, y)=3x^2-2\sin^2 y$

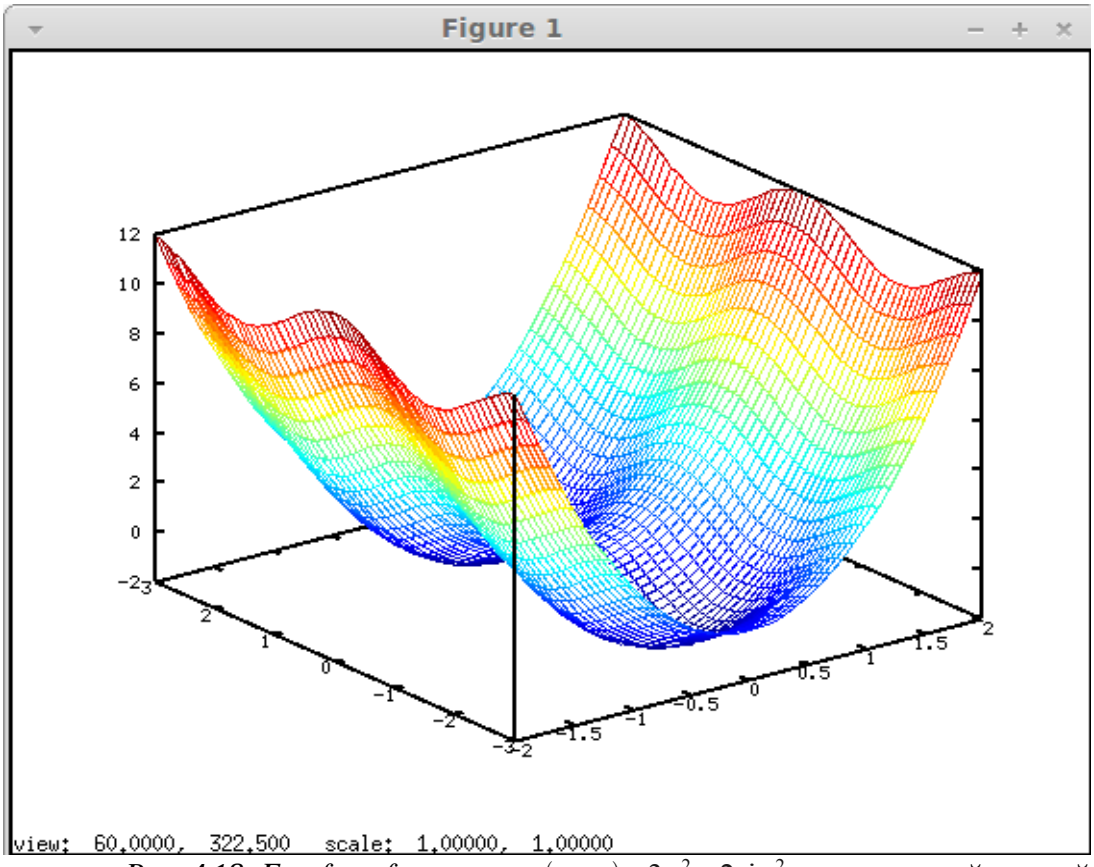


Рис. 4.18. График функции $z(x, y)=3x^2-2\sin^2 y$ *с плотной сеткой*

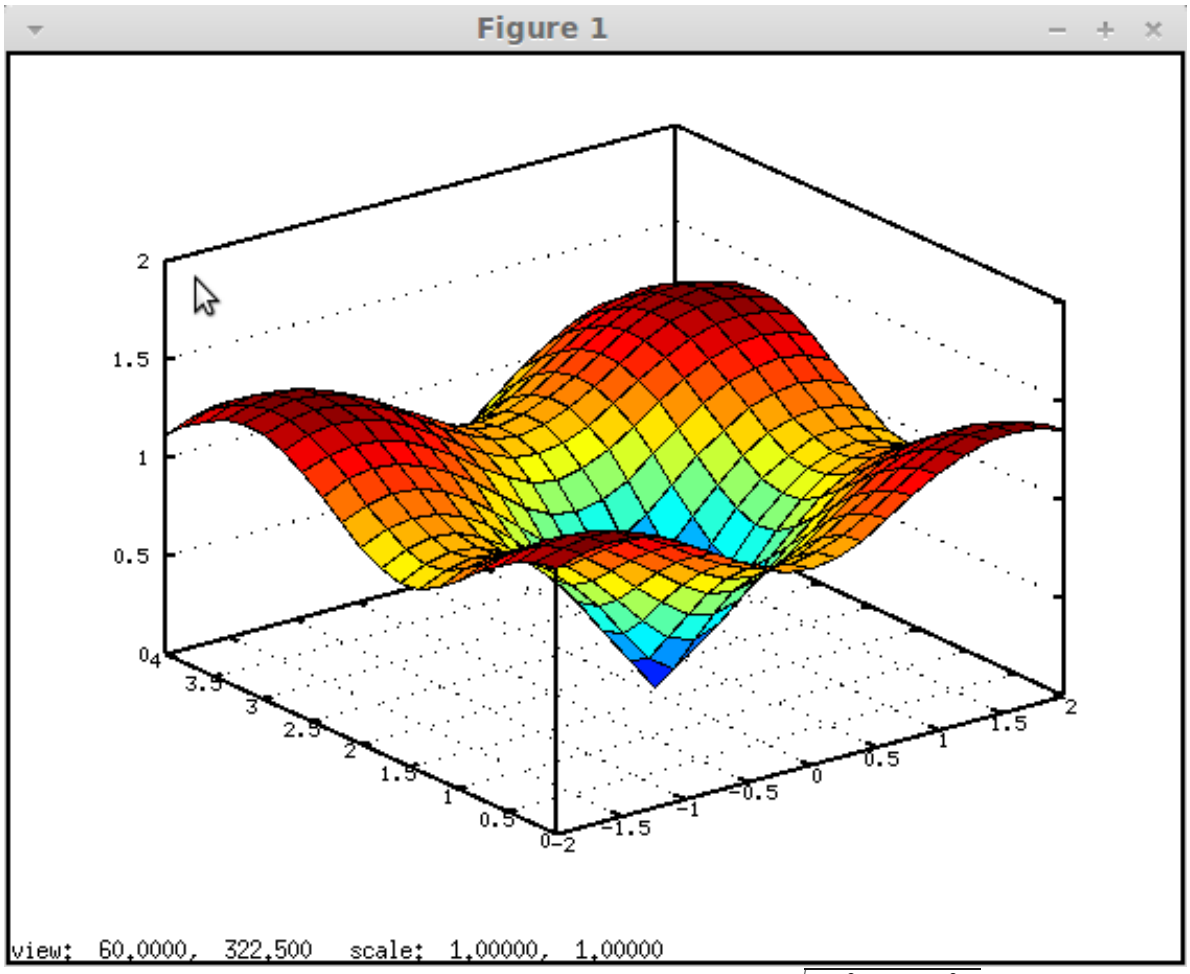


Рис. 4.19. График функции $z(x, y)=\sqrt{\sin^2 x+\cos^2 y}$

ЗАДАЧА 4.16. Построить график функций $z(x, y)=\pm(2x^2+ 3y^4)-1$.

Решение задачи с использованием функции `surf` представлено на листинге 4.18, полученный график изображен на рис. 4.20.

```
h=figure();
[x y]=meshgrid(-2:0.1:2,-3:0.1:3);
z=2*x.^2+3*y.^4-1;
z1=-2*x.^2-3*y.^4-1;
surf(x,y,z);
hold on
surf(x,y,z1);
```

Листинг 4.18

Построение поверхности с помощью функции `mesh` можно осуществить аналогично (листинг 4.19), графики функций – можно увидеть на рис.4.21.

```
h=figure();
[x y]=meshgrid(-2:0.1:2,-3:0.1:3);
z=2*x.^2+3*y.^4-1;
z1=-2*x.^2-3*y.^4-1;
mesh(x,y,z);
hold on
mesh(x,y,z1);
```

Листинг 4.19.

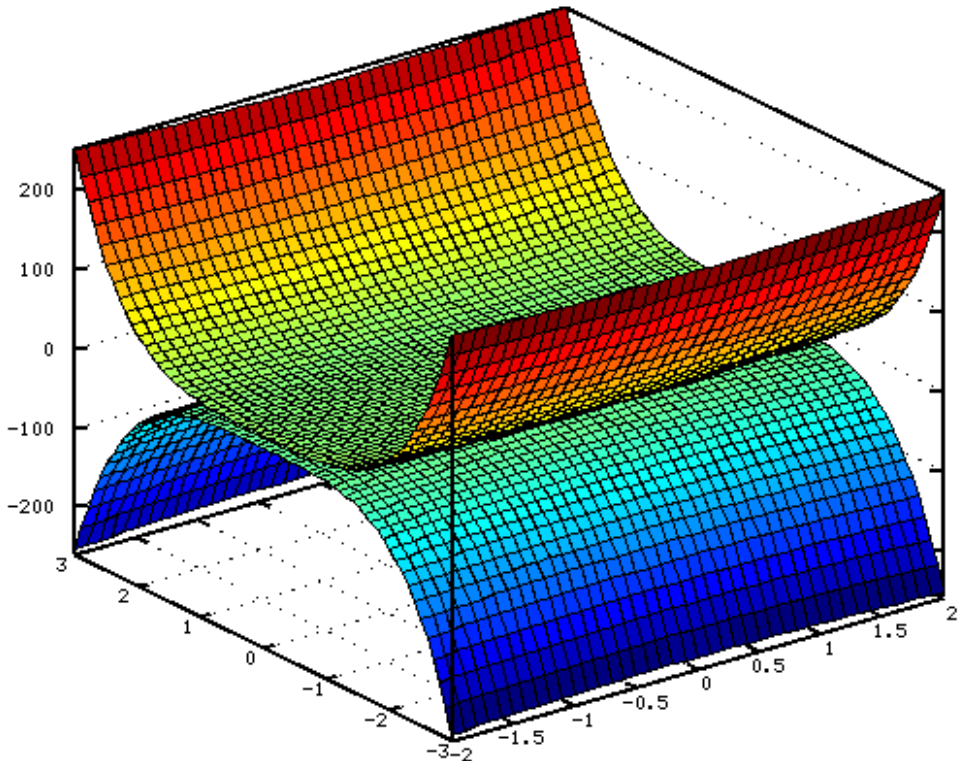


Рис. 4.20. Изображение двух поверхностей в одной системе координат с использованием функции `surf`

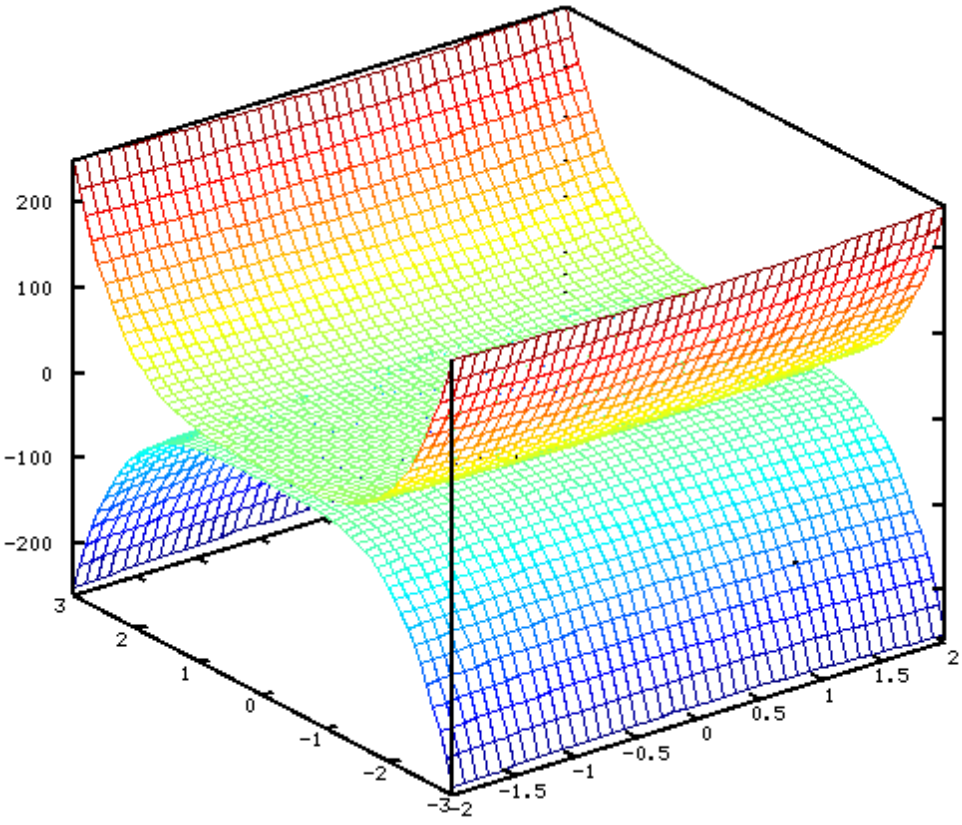


Рис. 4.21. Изображение двух поверхностей в одной системе координат с использованием функции `mesh`

4.2.2 Построение графиков поверхностей, заданных параметрически

При построении графиков поверхностей, заданных параметрически $x(u,v)$, $y(u,v)$ и $z(u,v)$ необходимо построить матрицы *X*, *Y* и *Z* одинакового размера. Для этого массивы u и v должны быть одинакового размера. Можно выделить два основных вида представления *x*, *y* и *z* в случае параметрического задания поверхностей:

1. Если *x*, *y* и *z* представимы в виде *f(u)·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* представимы в виде *f(u)* или *g(v)*, то в этом случае матрицы *X*, *Y* и *Z* следует записывать в виде *f(u)·ones(size(v))* или *g(v)·ones(size(u))* соответственно.

Рассмотрим несколько задач построения графиков поверхностей заданных параметрически.

ЗАДАЧА 4.17. Построить поверхность однополостного гиперболоида, уравнение которого задано в параметрическом виде

x(u,v)=ch(u)cos(v), y(u,v)= ch(u)sin(v), z(u,v)=sh(u), $u \in [0,\pi], v \in [0,2\pi]$.

На листинге 4.20 представлено решение этой задачи, согласно описанному выше алгоритму. График однополостного гиперболоида представлен на рис.4.22.

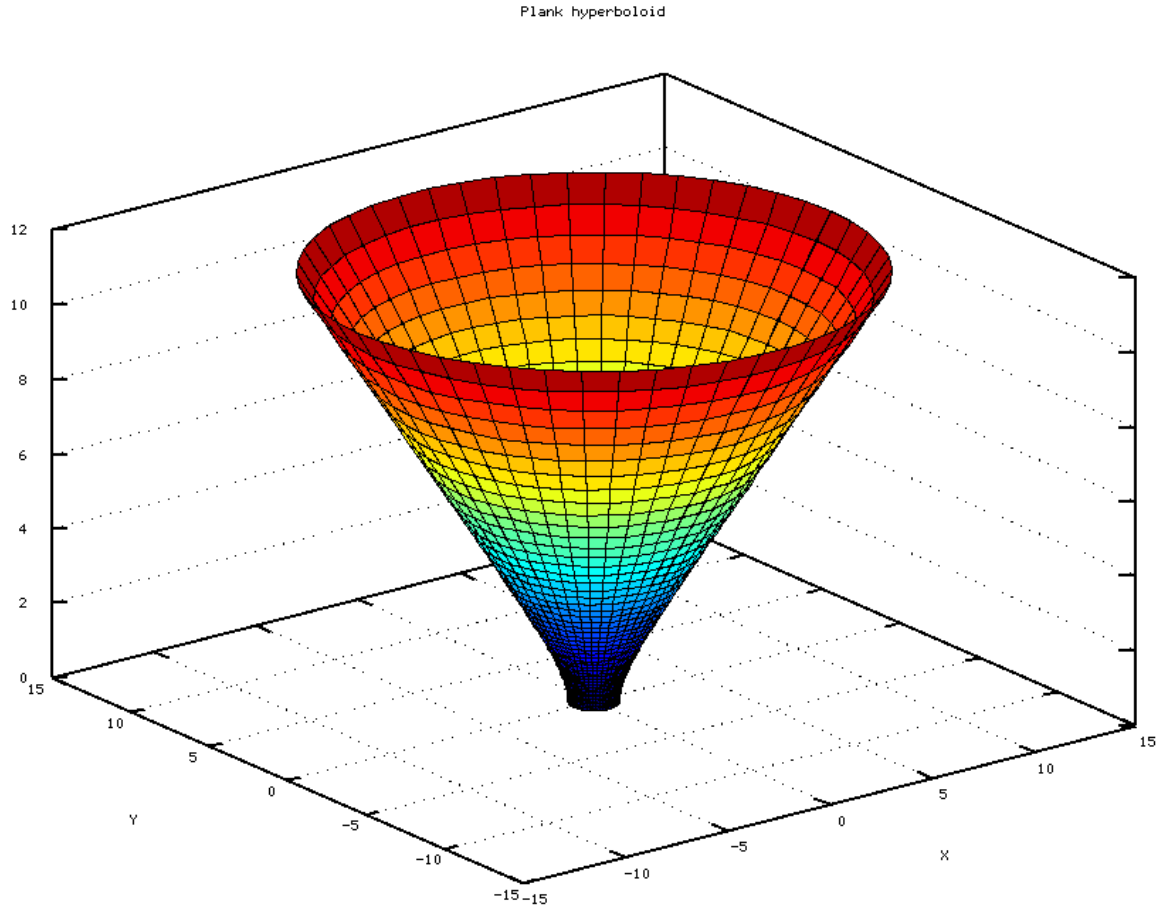


Рис. 4.22. График однополостного гиперболоида

```
clear all;
h=3.14/50;
% Формируем вектор-столбец u.
```

```
u=[0:h:3.14]';
% 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(size(v)).
Z=sinh(u)*ones(size(v));
% Формируем график поверхности.
surf(X,Y,Z);
grid on;
% Подписываем график и оси.
title('Plank hyperboloid');
xlabel('X'); ylabel('Y'); zlabel('Z');
```

Листинг 4.20.

Рассмотрим несколько способов построения сферы в Octave.

ЗАДАЧА 4.18. Построить поверхность сферы с центром $(x_{0,}y_{0,}z_{0})$ и радиусом R. $\left((x-x_0)^2+(y-y_0)^2+(z-z_0)^2=R^2\right)$. Уравнение сферы можно записать в параметрическом виде $x(u,v)=x_0+R\sin(u)\cos(v),\ y(u,v)=y_0+R\sin(u)\sin(v),\ z(u,v)=z_0+R\cos(u)$, $u\in[0,2\pi]$, $v\in[0,\pi]$.

Методика построения сферы подобна построению однополостного гиперболоида, описанному в задаче 4.4. На листинге 4.21 представлен текст программы построения сферы с центром в точке (1, 1, 1) и радиусом *R=4*, а на рис. 4.23 изображена сфера.

```
clear all;
h=pi/30;
% Формируем вектор-столбец u.
u=[-0:h:pi]';
% Формируем вектор-строку v.
v=[0:2*h:2*pi];
% 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(size(v)).
z=1+4*cos(u)*ones(size(v));
% Формируем график поверхности.
surf(x,y,z); grid on;
% Подписываем график и оси.
title('SPHERE');
xlabel('X');ylabel('Y');zlabel('Z');
```

Листинг 4.21.

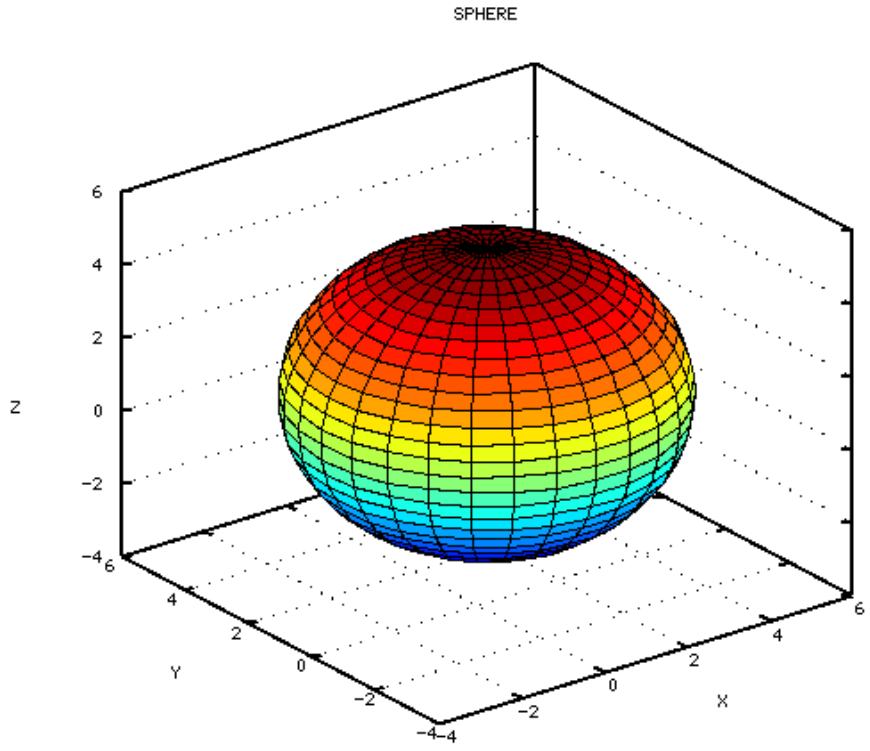


Рис. 4.23 График сферы, построенный с использованием функции `surf`

Octave содержит встроенную функцию `[X,Y,Z] = sphere(n)`, позволяющую формировать матрицы (`X`, `Y`, `Z`) размерности `n+1` для построения сферы единичного радиуса (пример 4.3) с центром в начале координат.

Для построения сферы единичного радиуса с центром в начале координат достаточно двух команд

`[X,Y,Z]=sphere(n); surf(X,Y,Z).`

Чем n больше, тем более «округлой» будет сфера. На рис. 4.24 изображена сфера единичного радиуса в центре в начале координат при `n=25`.

Встроенную функцию `sphere(n)` можно использовать и для построения сферы с центром (x_0, y_0, z_0) и радиусом R. На листинге 4.22 приведено решение примера 4.3 с помощью функции `sphere(n)`.

```
clear all;
% Определяем центр и радиус сферы.
x0=2;y0=-2;z0=5;R=10
% Формируем матрицы X, Y, Z для построения сферы единичного
%радиуса с центром в начале координат, используя функцию
% sphere(n).
[X,Y,Z]=sphere(25);
% Пересчитываем матрицы X,Y,Z для сферы с центром x0, y0, z0 и
% радиусом R.
X=x0+R*X; Y=y0+R*Y; Z=z0+R*Z;
% Изображаем сферу
surf(X,Y,Z)
```

Листинг 4.22.

В результате работы программы будет построена сфера, представленная на рис. 4.25.

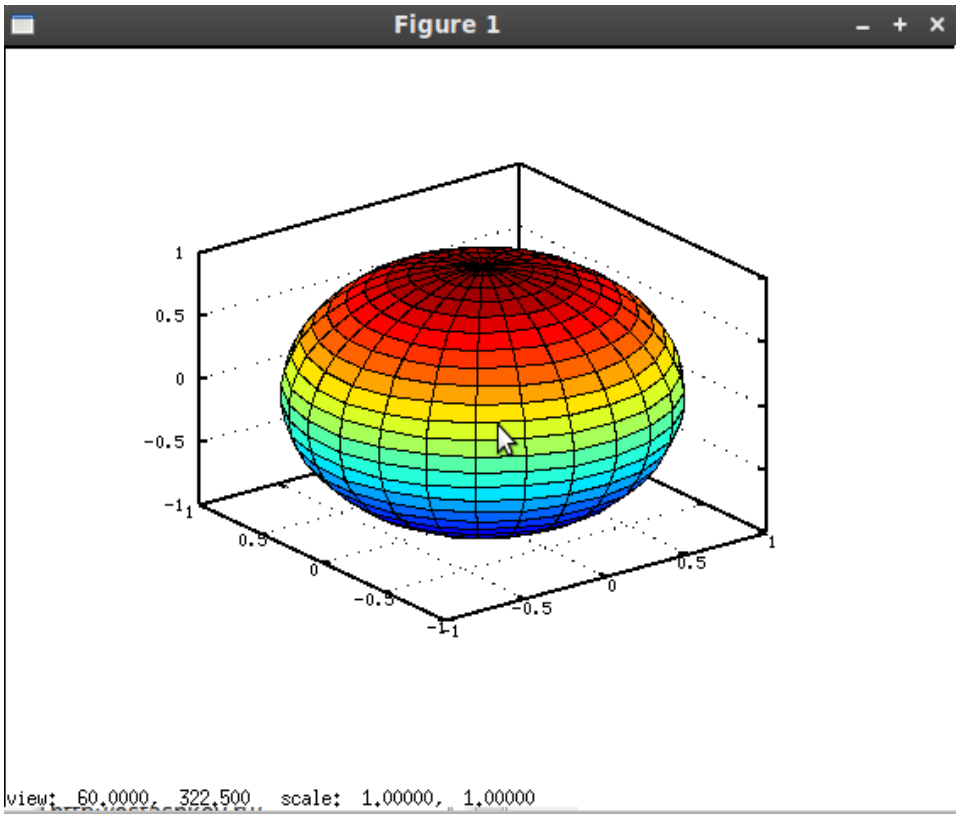


Рис. 4.24. График сферы единичного радиуса, построенный с использованием функции `sphere(25)`

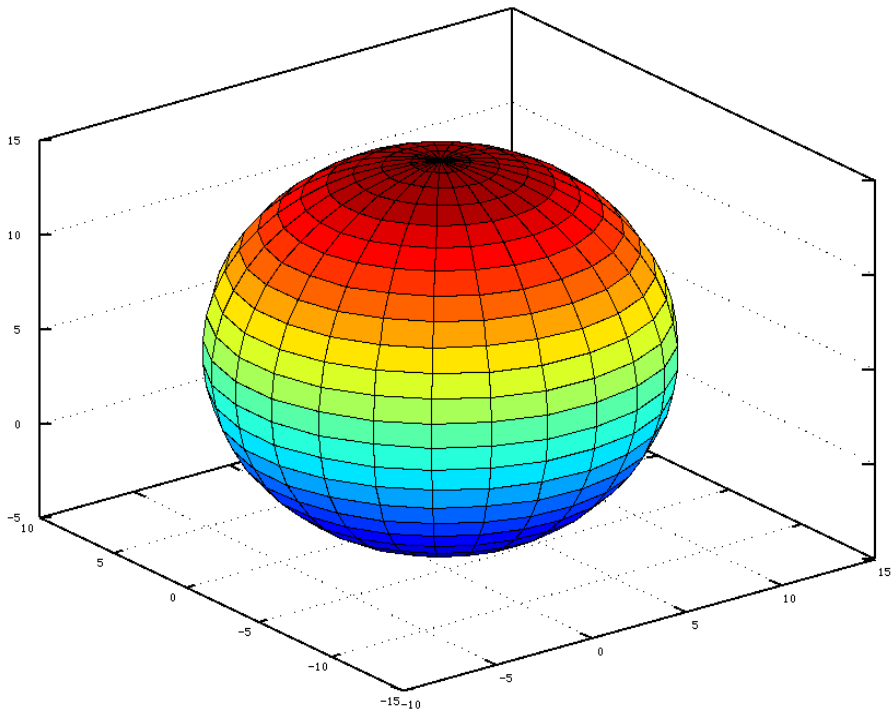


Рис. 4.25. Сфера, построенная с помощью программы, представленной на листинге 4.22

Сфера является частным случаем более общей фигуры эллипсоида. Рассмотрим два способа построения эллипсоида.

ЗАДАЧА 4.19. Построить поверхность эллипсоида. Уравнение задано в параметрическом виде:

$$x(u,v)=x_0+a\sin(u)\cos(v),$$
$$y(u,v)=y_0+b\sin(u)\sin(v),$$
$$z(u,v)=z_0+c\cos(u).$$

Здесь *a*, *b*, *c* – полуоси эллипсоида, x_0, y_0, z_0 – центр эллипсоида, $u\in[0,2\pi]$, $v\in[0,\pi]$.

Методика построения эллипсоида подобна тому, как ранее были построены однополостный гиперболоид (задача 4.17) и сфера (задача 4.18). Для этого необходимо сформировать матрицы *X*, *Y* и *Z*, после чего вызвать функцию `surf`. Как это сделать показано на листинге 4.23.

```
clear all;
h=pi/30;
% Формируем вектор-столбец u.
u=[-0:h:pi]';
% Формируем вектор-строку v.
v=[0:2*h:2*pi];
% Формируем матрицу X, используя матричное произведение
% sin(u)*cos(v).
a=3;b=7;c=1; x0=y0=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(size(v)).
z=z0+c*cos(u)*ones(size(v));
% Формируем эллипсоид.
surf(x,y,z); grid on;
% Подписываем график и оси.
title('ELLIPSOID'); xlabel('X'); ylabel('Y'); zlabel('Z');
```

Листинг 4.23.

Эллипсоид с центром в точке (10, 10,10) и полуосями `a=3, b=7,c=1` представлен на рис. 4.26.

Однако, для построения эллипсоида в Octave существует функция

```
[X, Y, Z] = ellipsoid(xc,yc, zc, xr, yr, zr, n),
```

которая позволяет автоматически сформировать матрицы `X`, `Y`, `Z`.

Здесь `X`, `Y`, `Z` – формируемые для построения поверхности матрицы размерности n+1, `xc`, `yc`, `zc` – координаты центра эллипсоида, `xr`, `yr`, `zr` – полуоси эллипсоида.

Для построения эллипсоида, представленного на рис. 4.26 достаточно ввести команды

```
a=3;b=7;c=1; x0=y0=z0=10;
[X, Y, Z]=ellipsoid (x0,y0, z0, a, b, c, 30);
surf(x,y,z); grid on;
title('ELLIPSOID');
xlabel('X');ylabel('Y');zlabel('Z');
```

Листинг 4.24

Функцию `ellipsoid` можно использовать и для построения сферы с центром в заданной точке.

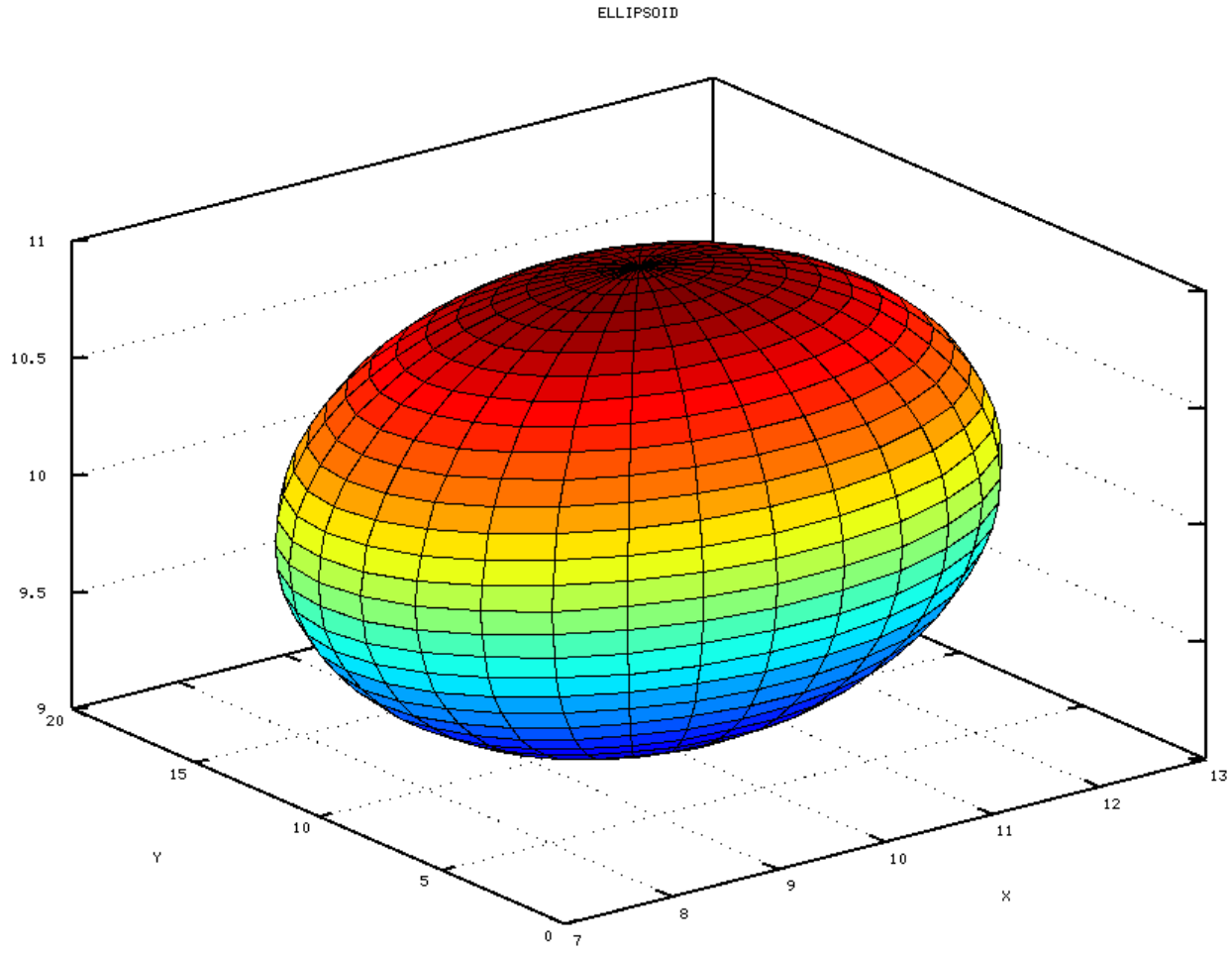


Рис. 4.26. Эллипсоид с центром в точке (10, 10,10) и полуосями a=3, b=7,c=1

Для построения цилиндров и круговых конусов можно использовать функции `cylinder` и `surf`. Функция `cylinder` формирует матрицы `X`, `Y`, `Z`. Сама поверхность (цилиндр, конус) строится с помощью функции `surf`.

Обращение к функции `cylinder` имеет вид.

```
[X, Y, Z] = cylinder(r,n);
```

Здесь

`r` – массив радиусов; если мы строим цилиндр, `r` – массив, состоящий из двух одинаковых значений, функция требует, как минимум два значения и для построения цилиндра это будут радиус верхнего и нижнего основания; при построении конуса `r` является массивом радиусов горизонтальных сечений кругового конуса;

`X`, `Y`, `Z` – формируемые для построения поверхности (конуса, цилиндра) матрицы размерности `n+1`.

ЗАДАЧА 4.20. Построить цилиндр радиуса *R=4* и высотой *h=1*.

Текст программы приведен на листинге 4.25, график – на рисунке 4.27.

```
clear all
%Формирование матриц x, y, z.
[x, y, z] = cylinder ([4,4],25);
grid on;
% Построение цилиндра.
surf(x, y, z); title ("Cylinder")
```

Листинг 4.25.

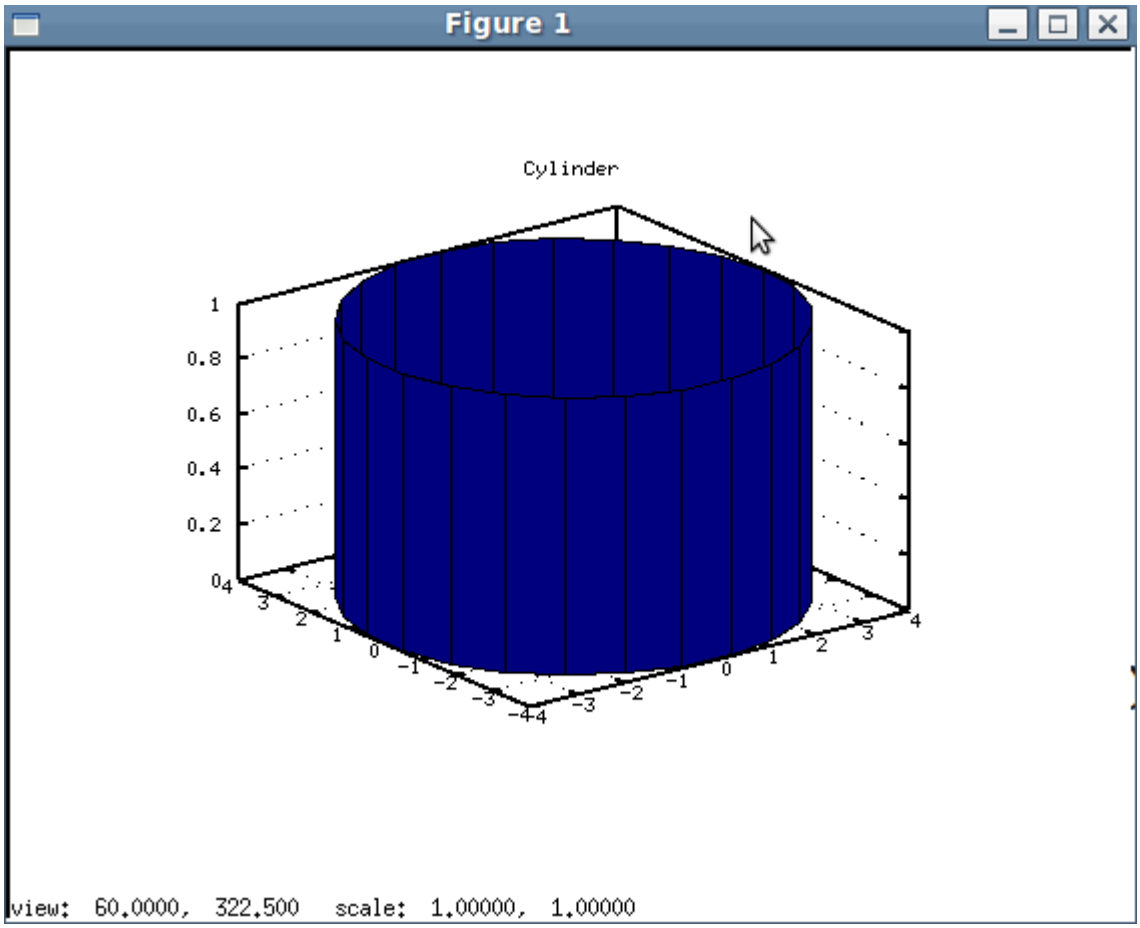


Рис. 4.27. Цилиндр радиуса R=4 и высотой h=1

ЗАДАЧА 4.21. Построить цилиндр радиуса 4 и высотой 20.

Текст программы приведен на листинге 4.26, график – на рисунке 4.28.

```
clear all
%Формирование матриц x, y, z.
[x, y, z] = cylinder ([4,4],25); grid on;
% Построение цилиндра с учётом высоты h=20.
surf (x, y, 20*z); title ("Cylinder")
```

Листинг 4.26.

ЗАДАЧА 4.22. Построить различные круговые конусы.

Усеченный круговой конус, представленный на рис. 4.29, генерируется программой на листинге 4.27.

```
clear all
[x, y, z] = cylinder (2:1:10,25); grid on;
surf (x, y, z); title ("Cone")
xlabel('X'); ylabel('Y'); zlabel('Z');
```

Листинг 4.27.

Круговой конус, представленный на рис. 4.30, генерируется программой на листинге 4.28.

```
clear all
[x, y, z] = cylinder ([5,4,3,2,1,0,1,2,3,4,5],25); grid on;
mesh(x, y, z); title ("Cone")
xlabel('X'); ylabel('Y'); zlabel('Z');
```

Листинг 4.28.

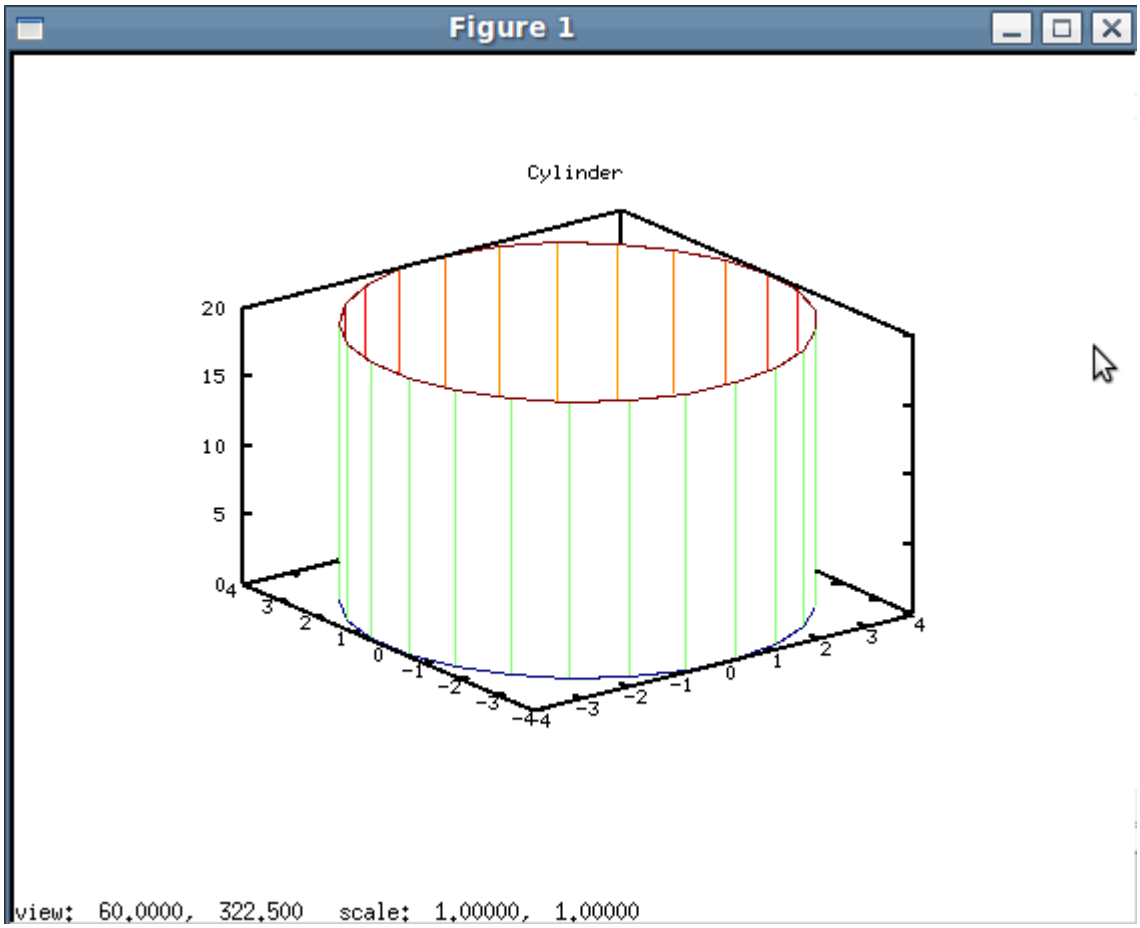


Рис. 4.28. Цилиндр радиуса R=4 и высотой h=20

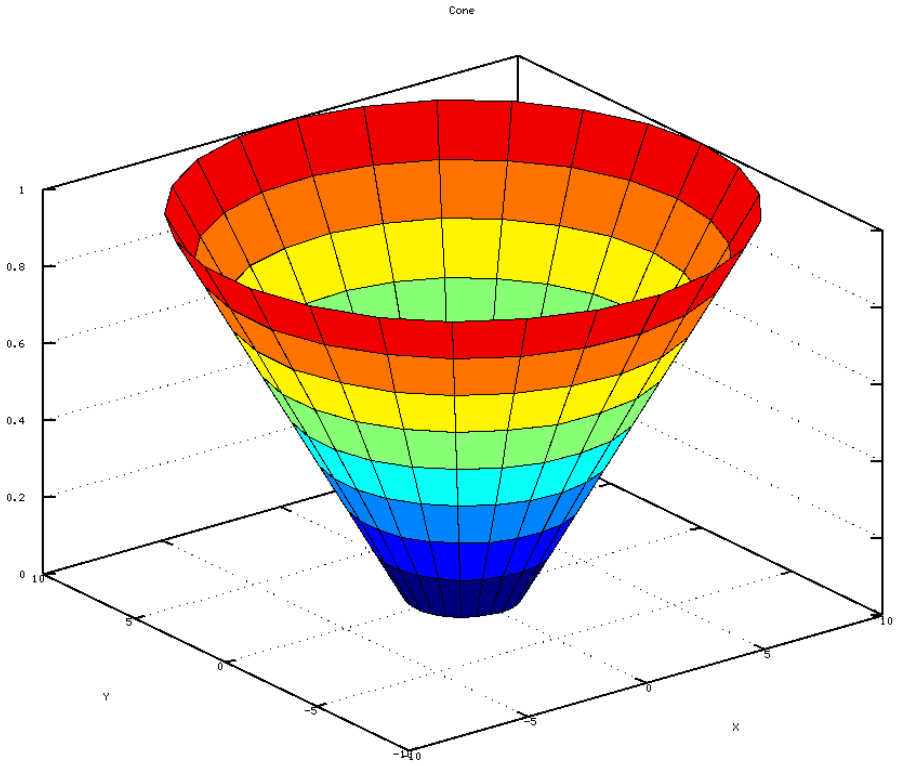


Рис. 4.29 Усеченный круговой конус к листингу 4.27

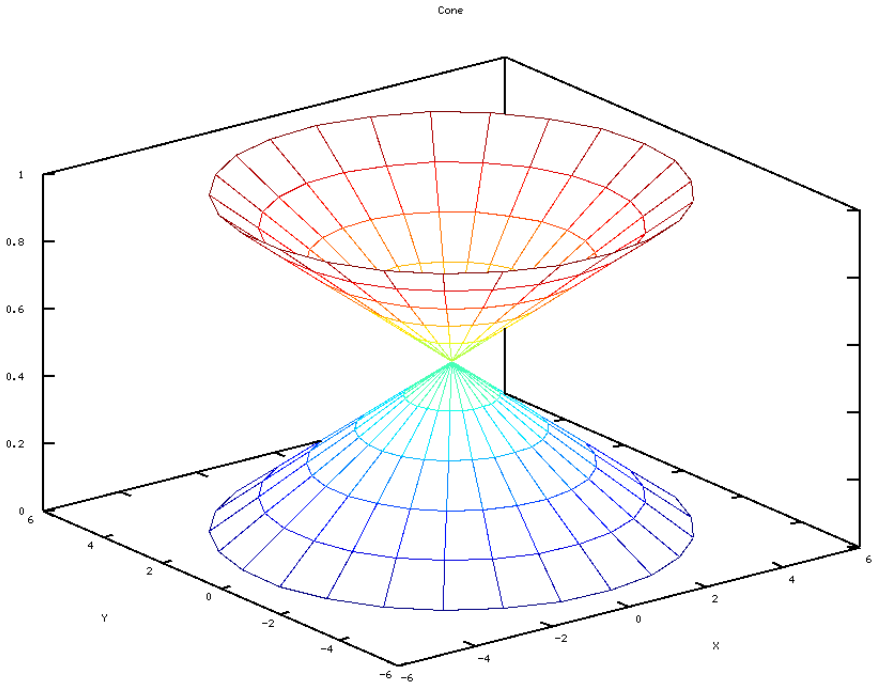


Рис. 4.30. Круговой конус к листингу 4.27

В завершении приведен листинг 4.29, который генерирует поверхность, представленную на рис. 4.31.

```
clear all
[x, y, z] = cylinder([1,3,5,7,6,4],25); surf(x, y, z);
title ("Surface"); xlabel('X'); ylabel('Y'); zlabel('Z');
```

Листинг 4.29.

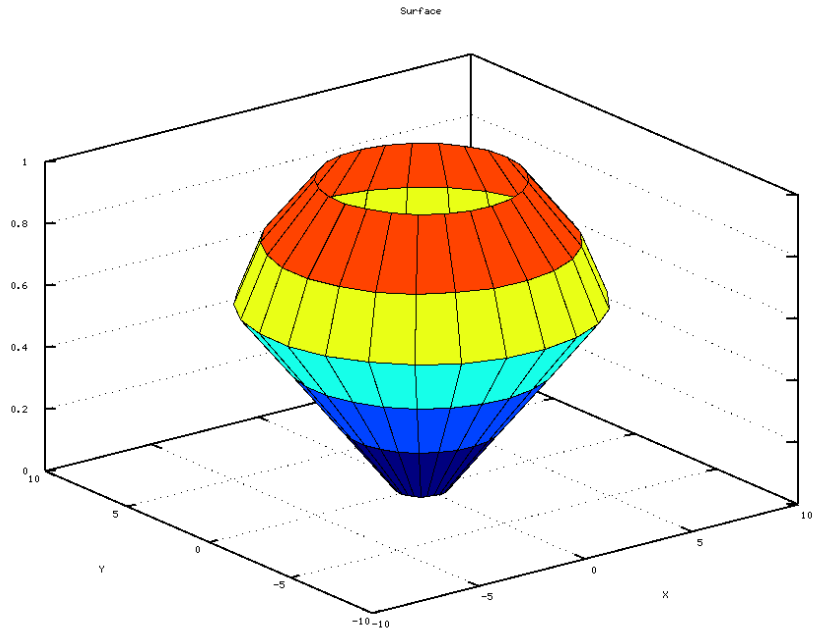


Рис. 4.31. Поверхность к листингу 4.29

4.3 Анимация

При изучении движения точки на плоскости Octave 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);
```

Листинг 4.30

Процесс движения точки вдоль синусоиды представлен на рис. 4.32, окончательный вид траектории движения точки вдоль синусоиды показан на рис. 4.33.

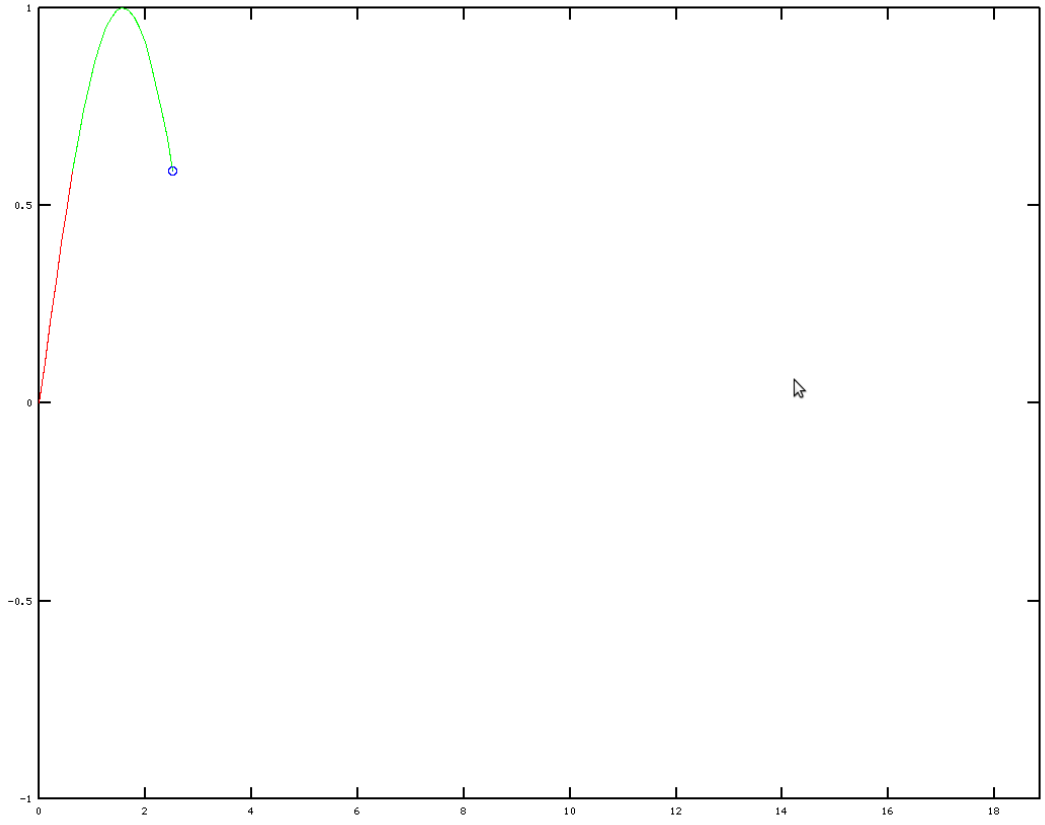


Рис. 4.32. Движение точки вдоль синусоиды

4.4 Построение линий и фигур

Построить линию как на плоскости, так и в пространстве можно с помощью функции

`line(x, y [, z, свойство, значение, …])`,

где `x, y, z,` - массивы. Функция соединяет прямыми линиями точки с координатами (`x1, y1 [, z1])`, `(x2, y2 [, z2])`, `...` Если обращение к функции происходит в формате `line(x, y)`, то открывается графическое окно с декартовой системой координат. Функция `line(x, y, z)` открывает графическое окно с трехмерной системой координат.

На рис. 4.34 приведен пример использования описанной функции. С помощью команд из листинга 4.31 выполняется построение отрезков, заданных координатами начала [0, 0] и конца $[x_i, y_i]$.

В листинге 4.32 представлены команды, с помощью которых в графическом окне (рис. 4.34) изображены случайным образом подобранные точки.

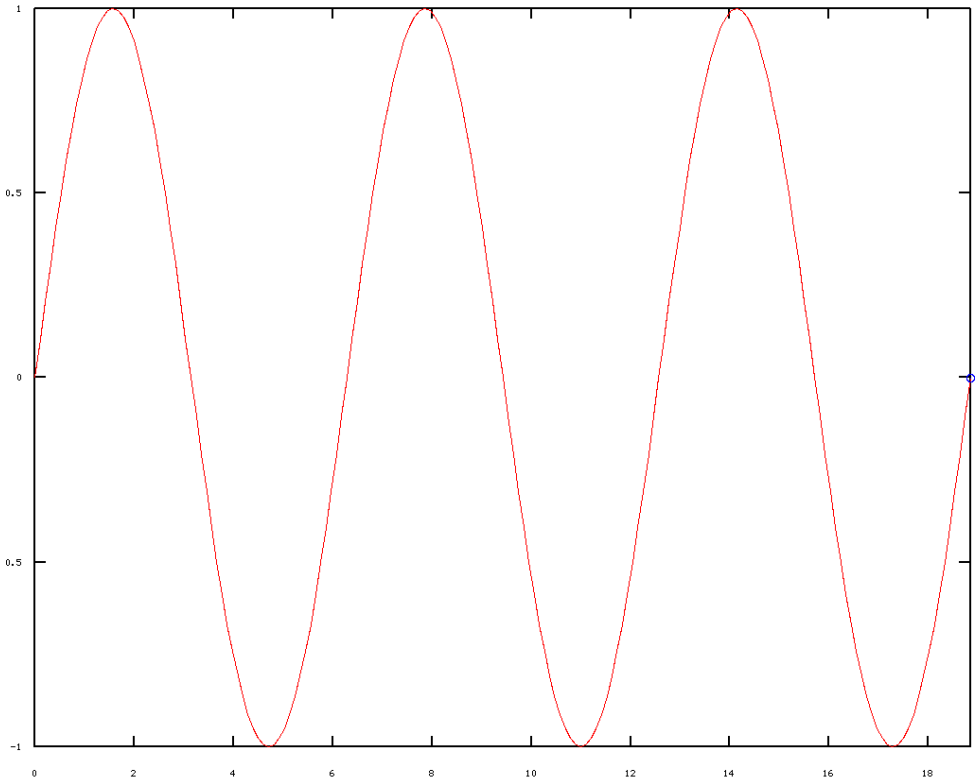


Рис. 4.33. Окончательный вид траектории движения точки

```
clear all; clf; cla;
for i=1:10
x=rand; y=rand;
line([0,x],[0,y],'linewidth',3);
end;
```

Листинг 4.31

```
for i=1:100
x=rand; y=rand; line(x,y,'marker','*','markersize',16);
end;
```

Листинг 4.32

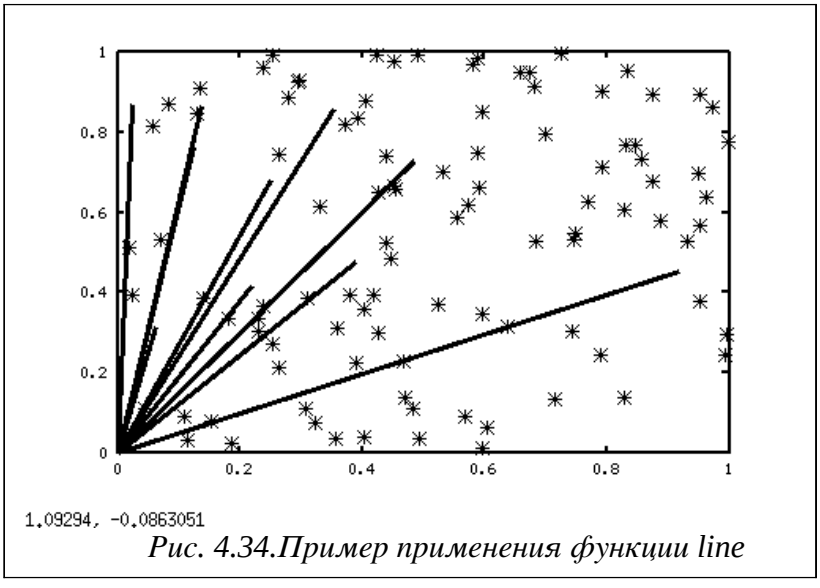


Рис. 4.34.Пример применения функции line

Выполнить построение многоугольника позволяет функция

```
patch(x, y [, z, свойство, значение, …]),
```

где `x, y, z,` - массивы. Функция строит многоугольник с вершинами в точках `(x1, y1 [, z1])`, `(x2, y2 [, z2])`, `...` Функции `patch(x, y)` и `patch(x, y, z)` выполняют построение в декартовой и трехмерной системах координат, соответственно. На рис. 4.35 показан треугольник, координаты которого были получены случайным образом (листинг 4.33).

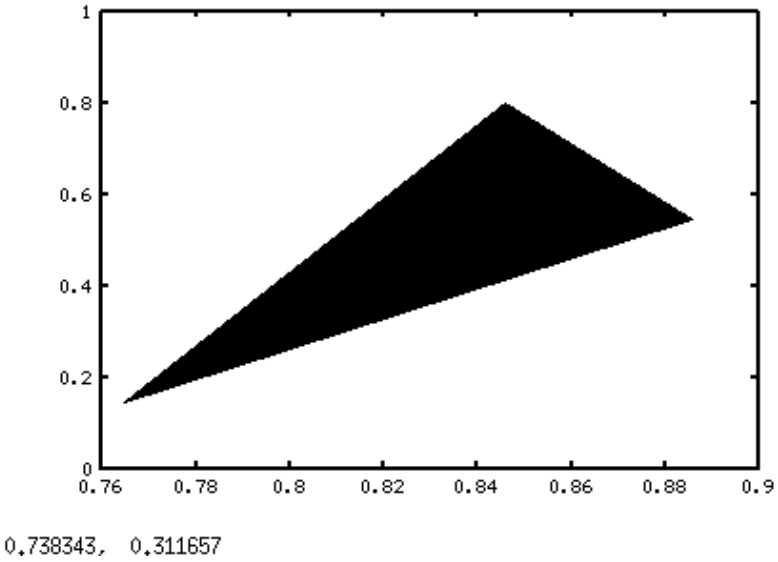


Рис. 4.35. Пример применения функции patch

```
clear all;
clf;
cla;
x=rand(1,3);
y=rand(1,3);
patch(x,y,'k');
```

Листинг 4.33

Большое количество примеров с использованием описанных функций приведено в шестой главе.

4.5 Графические объекты Octave

Встроенный язык Octave – объектно-ориентированный язык программирования. Все объекты находятся в определенной иерархии по отношению друг к другу. Рассмотрим основные графические объекты для работы с графикой и общие принципы работы с объектами на примере построения графика функции $x(t)=\sin(t)$ на интервале $[-3\pi;3\pi]$.

Построим график функции x(t) (листинг. 4.34).

Окно с графиком синуса на интервале $[-3\pi;3\pi]$ представлено на рис. 4.36.

```
t=-3*pi:pi/100:3*pi;
x=sin(t);
plot(t,x);
```

Листинг 4.34.

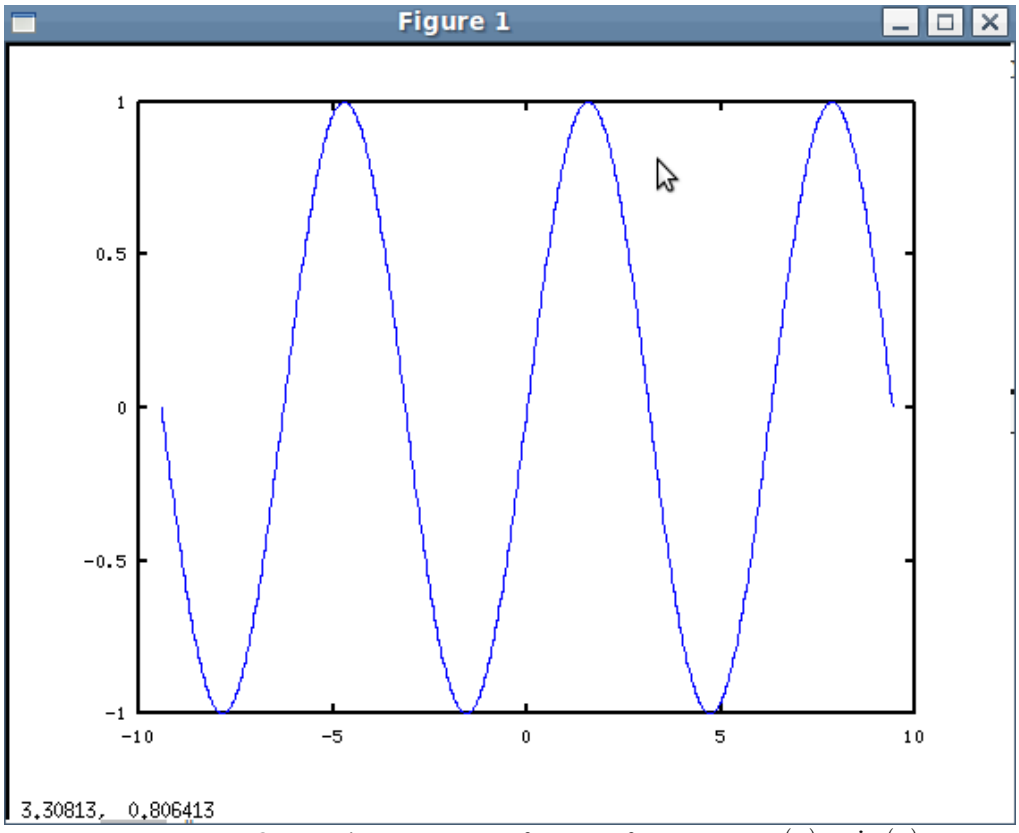


Рис. 4.36. Окно Figure 1 с графиком функции $x(t)=\sin(t)$ *на интервале* $[-3\pi;3\pi]$

В результате работы функции `plot` были созданы три графических объекта:

- графическое окно **Figure 1**;
- линия графика `sin(t)`;
- оси.

Для работы с графическими объектами используются переменные, в которых хранятся указатели на эти объекты. При работе с переменными, в которых хранятся объекты, пользователь оперирует ими, как обычными переменными. Однако, реально эти указатели – адреса в памяти, в которых хранятся объекты, в Octave в качестве указателя используется номер объекта.

В языке Octave есть три функции:

- `gcf()`[9] возвращает указатель на текущее графическое окно;
- `gca()` возвращает указатель на текущие оси;
- `gco()` возвращает указатель на текущий графический объект.

4.5.1 Свойства графических объекта

Для установки свойств объектов служит функция

```
set(h, 'Свойство1', Значение1, 'Свойство2', Значение2, 'Свойство3', Значение3, …)
```

Здесь

- `h` – указатель на объект, свойства которого будут устанавливаться (изменяться);
- `'Свойство1', 'Свойство2', 'Свойство3', …` – имена свойств, которые будут изменяться;

9 Синтаксис Octave допускает обращение и без скобок (gcf — это верно).

- Значение1, Значение2, Значение3, … – новые значения свойств.

В простейшем виде функция set имеет вид:

```
set(h, 'Свойство', Значение)
```

Для получения свойства объекта служит функция

```
get(h, 'Свойство');
```

Функция возвращает значения Свойства объекта с указателем h.

Если к функции get обратиться с одним параметром h, то функция вернет значения всех свойств объекта в виде Свойство = Значение.

4.5.2 Работа с графическим окном

Как уже рассматривалось ранее, для создания графического окна служит функция figure(), которая создает пустое графическое окно (см. рис. 4.37) и возвращает указатель на него.

Например:

```
>>> g=figure()
g = 12
```

Если есть несколько окон, то окно с указателем g выдвигается на передний план и становится текущим.

Как при создании графиков с определенными свойствами с помощью функции plot, при создании графических окон, осей и других объектов можно сразу определять некоторые свойства создаваемых объектов. Обращение к функции создания окна с определёнными свойствами имеет вид

```
figure('Свойство1', Значение1, 'Свойство2', Значение2, 'Свойство3', Значение3, …);
```

Для удаления (закрытия) окна с указателем h служит функция delete(h) .

Доступ к имени окна осуществляется с помощью свойства name, которое определяет строку, которая будет дописана к имени окна после стандартного имени окна Figure 1, Figure 2, … ;

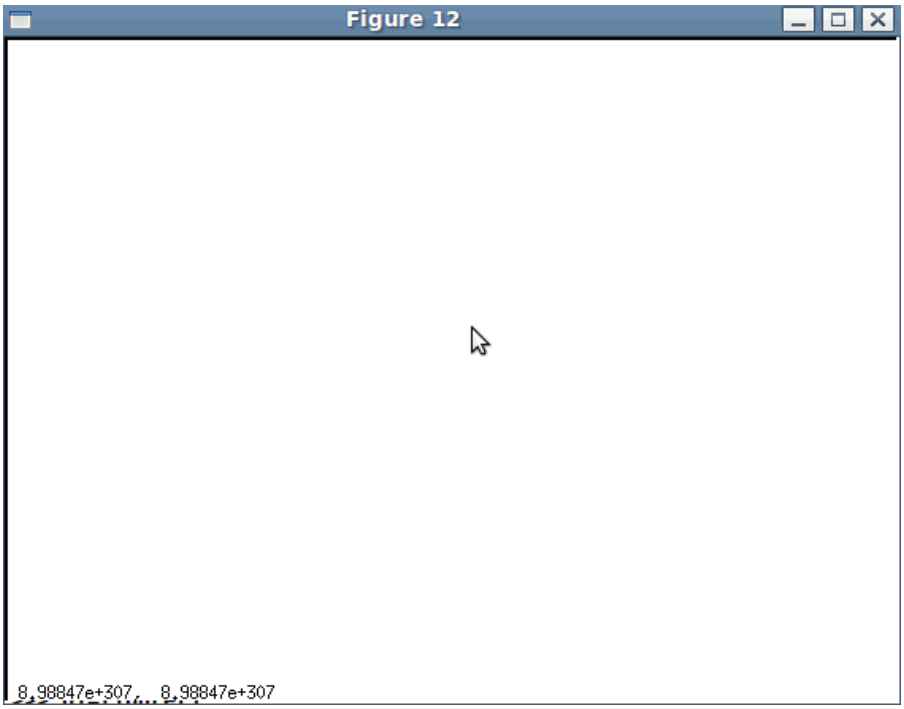


Рис. 4.37. Окно, созданное с помощью функции figure()

Например,

```
h=figure(); set(h,'name','New Window')
```

В результате появится окно, представленное на рис. 4.38.

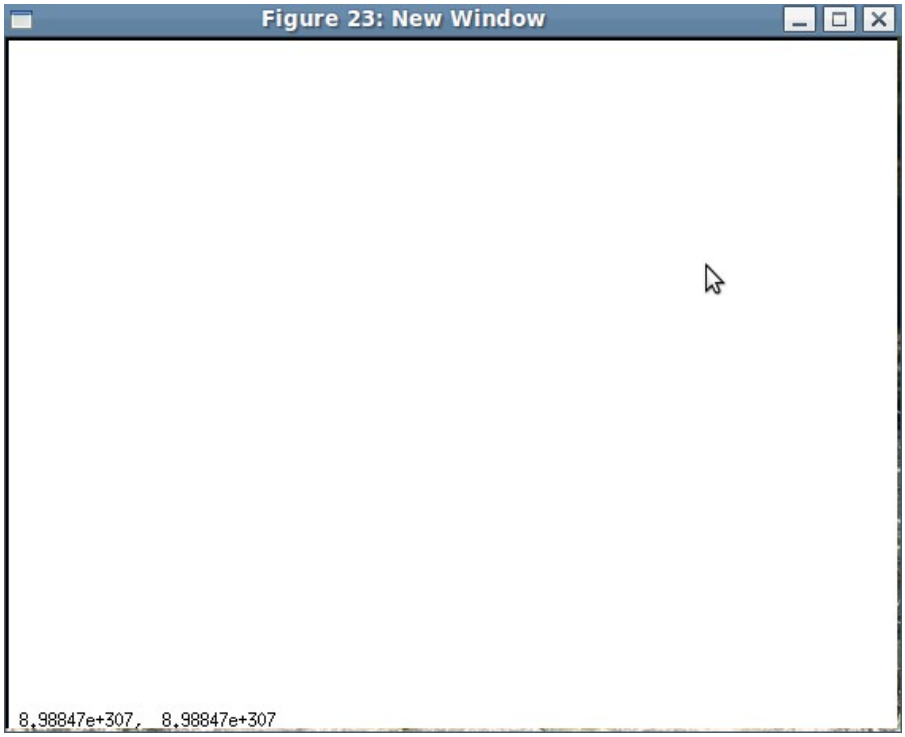


Рис. 4.38. Окно с изменённым заголовком

Если свойству `numbertitle` присвоить значение `'off'`, то это позволит отказаться от текущей нумерации окон `Figure 1, Figure 2, …` (листинг 4.35 и рис. 4.39)

```
h=figure(); set(h,'numbertitle','off');set(h,'name','New Window')
```

Листинг 4.35

Рис. 4.39. Окно после выключения стандартной нумерации окон

Как и у многих рассматриваемых графических объектов у окна есть свойство Position, определяющее расположение объекта. Position – массив из четырех элементов [xleft ybottom width height], xleft, ybottom определяют координаты левого нижнего угла экрана, относительно левого нижнего угла монитора width – ширина, height – высота графического окна в пикселях.

Создадим окно с именем «Our Window» шириной и высотой 400 пикселей, с левым нижним углом с координатами (75, 90) (листинг 4.36)[10].

```
h=figure('position',[75 90 400 400])
set(h,'numbertitle','off')
set(h,'name','New Window')
```

Листинг 4.36

Для создания осей в текущем окне служит функция axes, которая возвращает указатель на созданные оси.

4.5.3 Свойства осей графика

Для получения всех свойств осей построенного с помощью листинга 4.34 графика пользователь может ввести команду get(gca).

На листинге 4.37 приведены свойства осей графика, представленного на рис. 4.36.

```
>>>ans =
{
beingdeleted = off
busyaction = queue
buttondownfcn = [](0x0)
children = -6.1976
clipping = on
createfcn = [](0x0)
deletefcn = [](0x0)
handlevisibility = on
hittest = on
interruptible = on
parent = 1
selected = off
selectionhighlight = on
tag =
type = axes
userdata = [](0x0)
visible = on
__modified__ = on
uicontextmenu = [](0x0)
position =
0.13000 0.11000 0.77500 0.81500
box = on
key = off
keybox = off
keyreverse = off
keypos = 1
colororder =
```

10 У авторов при использовании GNU Octave, version 3.2.3 после изменения свойства position у окна с помощью функции set не происходило перерисовывания окна на новом месте; будем надеяться, что в последующих версиях эта проблема будет устранена.

```
0.00000 0.00000 1.00000
0.00000 0.50000 0.00000
1.00000 0.00000 0.00000
0.00000 0.75000 0.75000
0.75000 0.00000 0.75000
0.75000 0.75000 0.00000
0.25000 0.25000 0.25000
dataaspectratio =
20 2 1
dataaspectratiomode = auto
layer = bottom
xlim =
-10 10
ylim =
-1 1
zlim =
0 1
clim =
0 1
alim =
0 1
xlimmode = auto
ylimmode = auto
zlimmode = auto
climmode = auto
alimmode = auto
xlabel = -5.3352
ylabel = -4.7682
zlabel = -3.2778
title = -2.5540
xgrid = off
ygrid = off
zgrid = off
xminorgrid = off
yminorgrid = off
zminorgrid = off
xtick =
-10 -5 0 5 10
ytick =
-1.00000 -0.50000 0.00000 0.50000 1.00000
ztick = [](0x0)
xtickmode = auto
ytickmode = auto
ztickmode = auto
xminortick = off
yminortick = off
zminortick = off
xticklabel =
yticklabel =
zticklabel =
xticklabelmode = auto
```

```
yticklabelmode = auto
zticklabelmode = auto
interpreter = none
color =
1 1 1
xcolor =
0 0 0
ycolor =
0 0 0
zcolor =
0 0 0
xscale = linear
yscale = linear
zscale = linear
xdir = normal
ydir = normal
zdir = normal
yaxislocation = left
xaxislocation = bottom
view =
0 90
nextplot = replace
outerposition =
0 0 1 1
activepositionproperty = outerposition
ambientlightcolor =
1 1 1
cameraposition =
0.00000 0.00000 9.16025
cameratarget =
0.00000 0.00000 0.50000
cameraupvector =
-0 2 0
cameraviewangle = 6.6086
camerapositionmode = auto
cameratargetmode = auto
cameraupvectormode = auto
cameraviewanglemode = auto
currentpoint =
0 0 0
0 0 0
drawmode = normal
fontangle = normal
fontname = *
fontsize = 12
fontunits = points
fontweight = normal
gridlinestyle = :
linestyleorder = -
linewidth = 0.50000
minorgridlinestyle = :
```

```
plotboxaspectratio =
1 1 1
plotboxaspectratiomode = auto
projection = orthographic
tickdir = in
tickdirmode = auto
ticklength =
0.010000 0.025000
tightinset =
0 0 0 0
units = normalized
```

Листинг 4.37.

Рассмотрим наиболее часто используемые свойства осей:

• `box` определяет, заключать оси в прямоугольную рамку '`on`' (значение по умолчанию) или нет '`off`';

• `color` должен определять цвет фона графика, цвет задается в формате **RGB** `[r g b]`, где `r`, `g`, `b` - яркость красного, зеленого и синего цветов соответственно, которая меняется от 0 до 1 (табл. 4.4) или один из предопределенных цветов.

Таблица 4.4: Наиболее распространенные цвета

Цвет	**Цвет в формате RGB**
Черный	[0 0 0]
Синий	[0 0 1]
Темно-синий	[0 0 128/255]
Зеленый	[0 1 0]
Темно-зеленый	[0 128/255 0]
Голубой	[0 1 1]
Темно-голубой	[0 128/255 128/255]
Красный	[1 0 0]
Темно-красный	[128/255 0 0]
Пурпурный	[1 0 1]
Темно-пурпурный	[128/255 0 128/255]
Желтый	[1 1 0]
Темно-желтый	[128/255 128/255 0]
Темно-серый	[128/255 128/255 128/255]
Светло-серый	[192/255 192/255 192/255]
Белый	[1 1 1]

• `fontangle` позволит установить наклон шрифта разметки осей '`italic`' или не использовать наклон '`normal`' (значение по умолчанию);

• `fontname` определяет название шрифта, используемого при подписи осей (например, '`Arial`');

• `fontsize` определяет размер шрифта в пунктах;

• `fontweight` определяет толщину шрифта, наиболее часто используемые значения

'normal' (по умолчанию) и 'bold';

• gridlinestyle позволяет изменять стиль линий сетки, значения стиля линий подробно рассмотрены при описании функции plot;

• linewidth определяет толщину линий осей, значение по умолчанию 0.5;

• visible – видимость осей: 'on' (значение по умолчанию) – оси видимы, 'off' – оси невидимы;

• xcolor, ycolor, zcolor определяет цвет соответствующей оси в формате **RGB**;

• xdir, ydir, zdir определяет направление соответствующей оси: нормальное 'normal' (значение по умолчанию) или обратное 'reverse';

• xgrid, ygrid, zgrid определяет наличие 'on' или отсутствие 'off' (значение по умолчанию) сетки, перпендикулярной оси;

• xaxislocation определяет расположение оси X: сверху – 'top' или снизу – 'bottom' (значение по умолчанию);

• yaxislocation определяет расположение оси Y: справа – 'right' или слева – 'left' (значение по умолчанию);

• xlim, ylim, zlim задают пределы изменения переменных *x*, *y* и *z* в виде массива из двух значений;

• xscale, yscale и zscale определяют масштаб соответствующих осей: линейный 'linear' (значение по умолчанию) или логарифмический 'log';

• xtick, ytick, ztick – вектора, определяющие координаты разметки соответствующих осей.

На листинге 4.38 представлены команды, изменяющие внешний вид осей графика. График функции $x=\sin(t)$ на интервале $[-6\pi;6\pi]$ после их применения представлен на рис. 4.40.

```
h=figure();
t=-3*pi:pi/100:3*pi;
x=sin(t);
plot(t,x);
% Убираем прямоугольную сетку вокруг оси.
set(gca,'box','off');
% Определяем шрифт.
set(gca,'fontname','Arial');
% Определяем размер шрифта 20.
set(gca,'fontsize',20);
% Включаем линии сетки, перпендикулярные OX и OY.
set(gca,'xgrid','on');
set(gca,'ygrid','on');
% Устанавливаем координаты линий сетки, перпендикулярной OX.
set(gca,'xtick',[-3 -1 0 1 2] );
```

Листинг 4.38.

Обращение к функции создания осей с определёнными свойствами имеет вид:

```
axes('Свойство1', Значение1, 'Свойство2', Значение2, 'Свойство3', Значение3, …);
```

С помощью функции set можно также изменять свойства линий, которые формируется с помощью подробно рассмотренной ранее функции plot.

Рассмотрим наиболее часто используемые свойства линий:

• color определяет цвет текущей линии в формате RGB или с помощью предопределенного цвета;

• linestyle устанавливает стиль линий;

• `linewidth` определяет толщину линии в пунктах;
• `marker` устанавливает тип маркера для изображения точек на графике.
• `markersize` определяет размер маркера в пунктах.

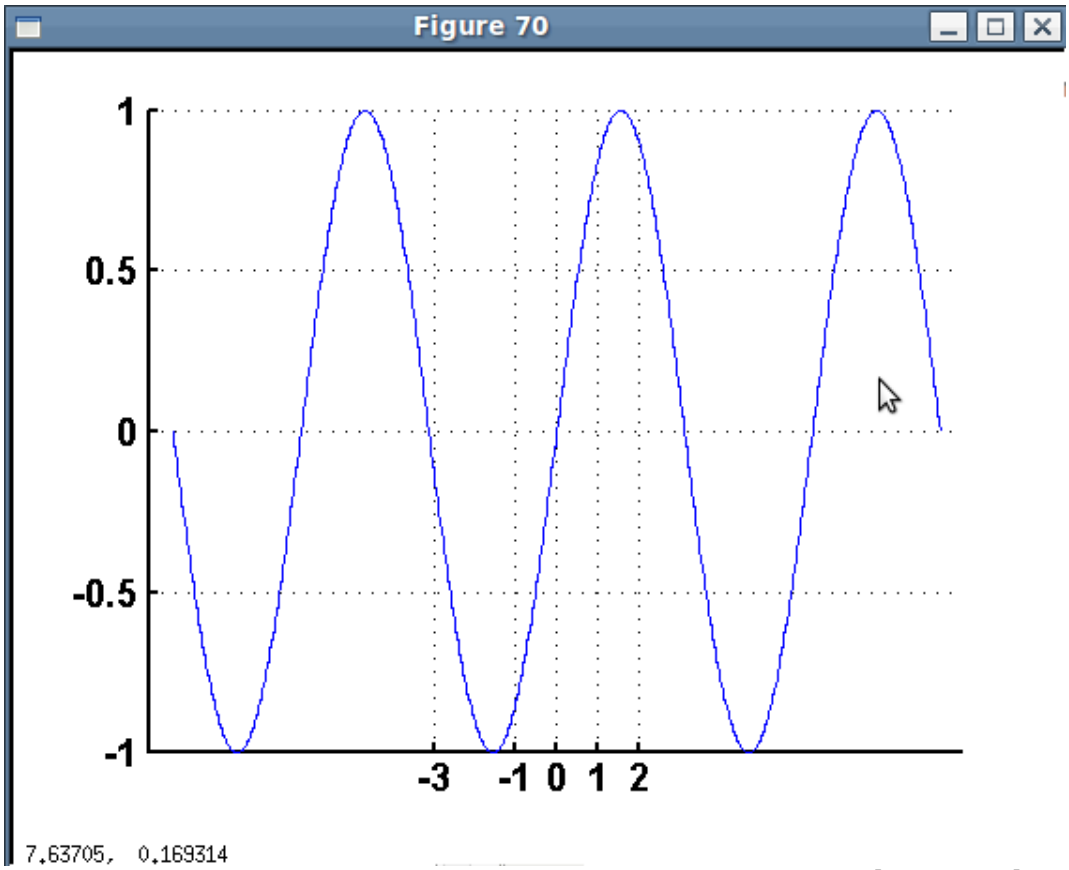


Рис. 4.40. График функции $x=\sin(t)$ *на интервале* $[-6\pi;6\pi]$ *после выполнения команд из листинга 4.38*

4.5.4 Удаление и очистка объектов

Для того, чтобы удалить объект в графическом окне, необходимо вызвать функцию

```
delete(h)
```

где `h` – указатель на удаляемый объект (указатель на линию, оси и т.д.). Следует понимать, что удаление осей приведет к исчезновению всех объектов, которые располагались на осях.

Очистка текущих осей осуществляется командой `cla`, очистка текущего окна – командой `clf`.

Рассмотрим описанные возможности работы с окнами на нескольких примерах. Авторы рекомендуют читателю внимательно изучить задачи 4.23, 4.24, в которых собраны стандартные приемы работы с окнами, линиями графиков и осями и их свойствами.

4.5.5 Примеры

ЗАДАЧА 4.23. Написать программу создания графиков функций $x(t)=e^{\cos(t)}$, $y=e^{\sin(t)}$, $z=\cos(t^2)$ на интервале [-5;5]. Графики функций $x(t)=e^{\cos(t)}$, $y=e^{\sin(t)}$ изобразить в графическом окне с именем **WINDOW1** красным и синим цветом, а график функции $z=\cos(t^2)$ – в окне с именем **WINDOW2** зеленым. В

обоих окнах вывести линии сетки.

На листинге 4.39 приведено решение этой задачи с подробными комментариями.

```
t=-5:0.1:5;
x=exp(cos(t));
y=exp(sin(t));
z=cos(t.^2);
% Создаем первое графическое окно, указатель, на которое будет
%храниться в переменной hfig1.
hfig1=figure;
% Создаем второе графическое окно, указатель, на которое будет
% храниться в переменной hfig2.
hfig2=figure;
% Объявляем первое графическое окно текущим.
figure(hfig1);
% Выводим в нем оси, указатель на на которые будет храниться в
% переменной hAxes1.
hAxes1=axes;
% Выводим в этом окне график функций x(t) и y(t), указатель на
% который записываем в переменную h_gr1. В h_gr(1) будет
% храниться первая линия - x(t), в h_gr(2) будет храниться
% вторая линия - y(t).
h_gr1=plot(t,x,t,y);
% Объявляем второе графическое окно текущим.
figure(hfig2);
% Выводим в нем оси, указатель на которые будет храниться в
% переменной hAxes2.
hAxes2=axes;
% Выводим в этом окне график функции z(t), уазатель на который
% записываем в переменную h_gr2.
h_gr2=plot(t,z);
% Объявляем первое графическое окно текущим
figure(hfig1);
% Далее устанавливаем свойства осей и графиков в первом окне.
% Отказываемся от стандартной нумерации окон для первого окна.
set(hfig1,'numbertitle','off');
% Устанавливаем новое имя первого графического окна.
set(hfig1,'name','WINDOW1');
% Включаем отображение линий сетки, перпендикулярной оси OX
% для осей hAxes1.
set(hAxes1,'xgrid','on');
% Включаем отображение линий сетки, перпендикулярной оси OY
% для осе. hAxes1.
set(hAxes1,'ygrid','on');
% Устанавливаем красный цвет первой линии в первом графическом
% окне.
set(h_gr1(1),'color','r');
% Устанавливаем синий цвет второй линии в первом графическом
% окне.
set(h_gr1(2),'color','b');
% Объявляем второе графическое окно текущим
figure(hfig2);
```

```
% Далее устанавливаем свойства осей и графиков во втором окне.
% Отказываемся от стандартной нумерации окон для второго окна.
set(hfig2,'numbertitle','off');
% Устанавливаем новое имя второго графического окна.
set(hfig2,'name','WINDOW2');
% Включаем отображение линий сетки, перпендикулярной оси OX
% для осей hAxes2.
set(hAxes2,'xgrid','on');
% Включаем отображение линий сетки, перпендикулярной оси OY
% для осей hAxes2.
set(hAxes2,'ygrid','on');
% Устанавливаем зеленый цвет первой линии во втором
% графическом окне.
set(h_gr2,'color','g');
% Эта функция может быть и такой.
% set(h_gr2(1),'Color','g');
```

Листинг 4.39.

На рис. 4.41 и 4.42 представлены созданные с помощью программы, приведенной в листинге 4.39, окна с графиками функций $x(t)=e^{\cos(t)}$, $y=e^{\sin(t)}$ и $z=\cos(t^2)$ соответственно.

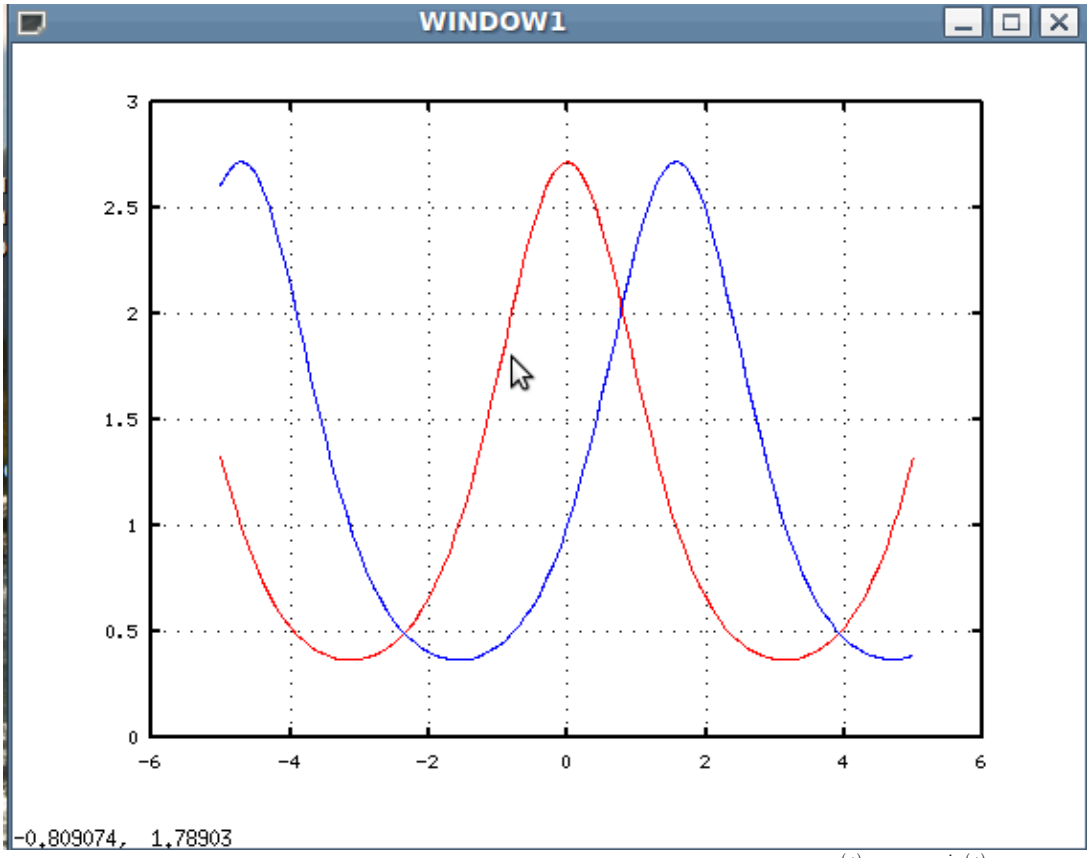


Рис. 4.41. Окно с графиками функций $x(t)=e^{\cos(t)}$, $y=e^{\sin(t)}$

При программировании работы с графическими окнами, вызов функции `plot` может осуществляться со всеми параметрами, рассмотренными в этой главе. Для того, что бы добавить новый график в текущие оси необходимо перед вызовом функции plot выполнить команду `hold on`. При добавлении графика в текущие оси с помощью функции plot необходимо самостоятельно устанавливать цвет и тип графика.

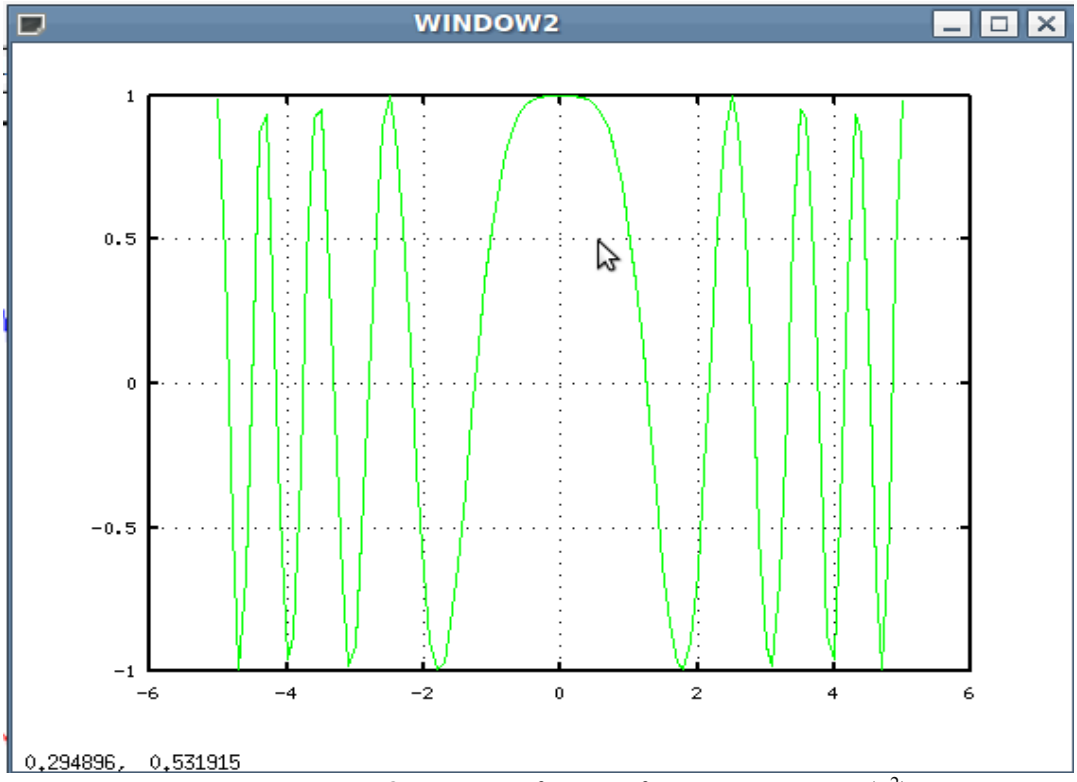


Рис. 4.42. Окно с графиком функции $z=\cos(t^2)$

ЗАДАЧА 4.24. Изобразить функции

$$x_i=\alpha_i e^{\sin(t)},\ y_i=\sin(\alpha_i t),\ z_i=\cos(\alpha_i t),\ v_i=\sin(2\alpha_i t)+\cos(3\alpha_i t)$$

на интервале [-5; 5], если коэффициенты $\alpha_i=0.5, 0.6, 0.73, 0.79$ хранятся в текстовом файле **gr.txt** (рис. 4.43).

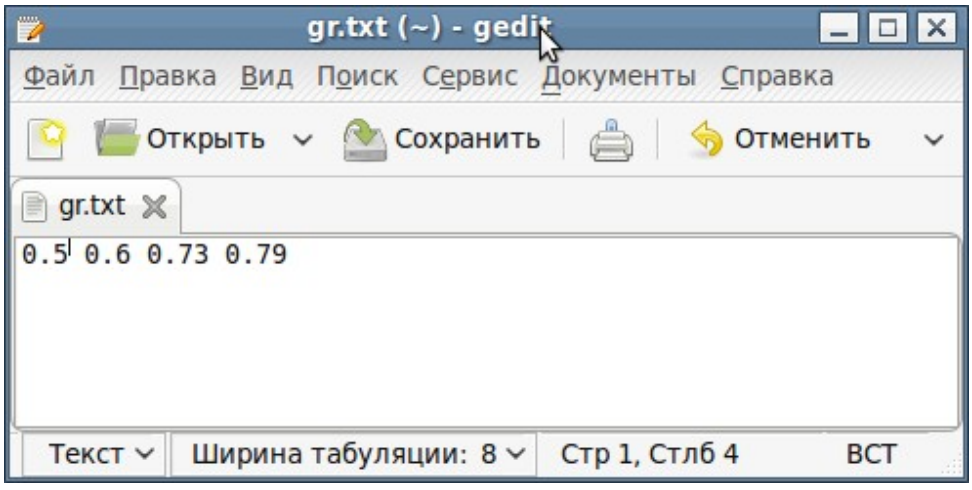


Рис. 4.43. Содержимое файла ***gr.txt***

При решении этой задачи необходимо будет построить четыре множества графиков x_i, y_i, z_i и v_i. Каждое множество будем изображать в своих осях. На листинге 4.40 приведена программа решения задачи 4.24, а на рис. 4.44 представлено полученное в результате графическое окно.

```
% Открываем текстовый файл gr.txt в режиме чтения.
f=fopen('gr.txt','rt');
% Считываем из него данные в массив alf.
alf=fscanf(f,'%f',4);
% Формируем массивы t, x, y, z, v
```

```
t=-5:0.1:5;
x=alf*exp(sin(t));
y=sin(alf*t);
z=cos(alf*t);
v=sin(2*alf*t)+cos(3*alf*t);
% Создаем графическое окно
hfig1=figure;
% Устанавливаем новое имя первого графического окна.
set(hfig1,'numbertitle','off'); set(hfig1,'name','Plots');
% Выводим в нем оси, указатель на которые будет храниться в
% переменной haxes1. Оси будут располагаться в левом нижнем
% углу графического окна.
haxes1=axes('position',[0.05 0.05 0.4 0.4]);
% Выводим множество графиков xi(t)
plot(t,x);
% Выводим линии сетки на осях.
set(haxes1,'xgrid','on','ygrid','on');
% Выводим в графическом окне оси, указатель на которые будет
% храниться в переменной haxes2. Оси будут располагаться в
% правом нижнем углу графического окна.
haxes2=axes('position',[0.5 0.05 0.4 0.4]);
```

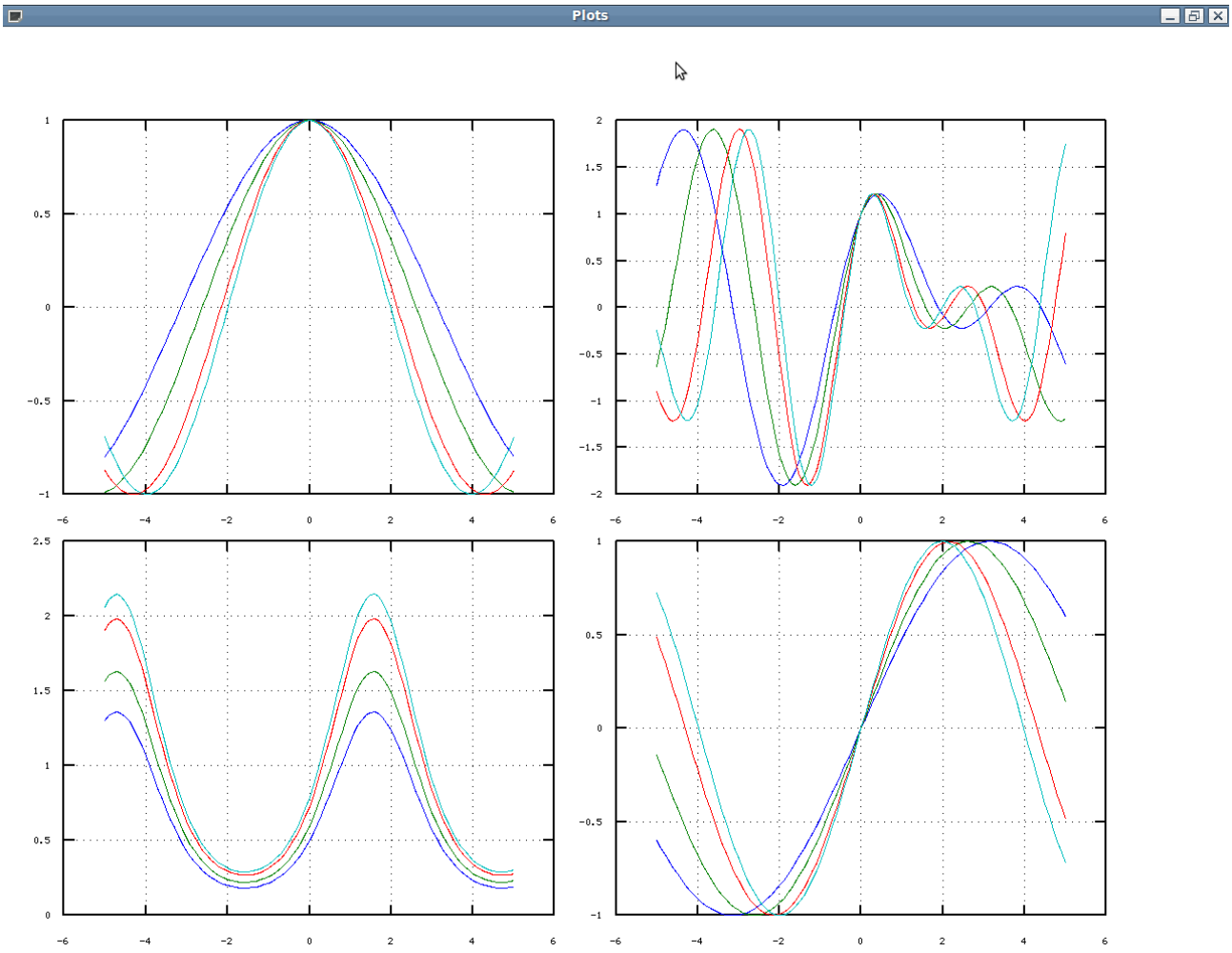


Рис. 4.44. Окно с графиками функций

$x_i=\alpha_i e^{\sin(t)}$, $y_i=\sin(\alpha_i t)$, $z_i=\cos(\alpha_i t)$, $v_i=\sin(2\alpha_i t)+\cos(3\alpha_i t)$

```
% Выводим множество графиков yi(t).
plot(t,y);
% Выводим линии сетки на осях.
set(haxes2,'xgrid','on','ygrid','on');
% Выводим в графическом окне оси, указатель на которые будет
% храниться в переменной haxes3. Оси будут располагаться в
% левом верхнем углу графического окна.
haxes3=axes('position',[0.05 0.5 0.4 0.4]);
% Выводим множество графиков zi(t)
plot(t,z);
% Выводим линии сетки на осях.
set(haxes3,'xgrid','on','ygrid','on');
% Выводим в графическом окне оси, указатель на которые будет
% храниться в переменной haxes3. Оси будут располагаться в
% правом верхнем углу графического окна.
haxes4=axes('position',[0.5 0.5 0.4 0.4]);
% Выводим множество графиков zi(t).
plot(t,v);
% Выводим линии сетки на осях.
set(haxes4,'xgrid','on','ygrid','on');
```

Листинг 4.40.

На графиках не хватает текстовой информации, которая бы поясняла выведенные графики.

Для вывода текста можно использовать следующие функции:

- `title('Заголовок')` предназначена для вывода заголовка графика;
- `xlabel('Подпись')` служит для вывода текста под осью **OX**;
- `ylabel('Подпись')` предназначена для вывода названия оси **OY**;
- `text(x,y,'Text')` служит для вывода текста в точке с координатами (`x,y`); координаты (`x,y`) задаются в системе координат графика, нет необходимости пересчитывать их в "экранную" систему координат.

В листинге 4.41 представлена программа построения графика $x=\sin(t)$ на интервале $[-2\pi;2\pi]$ вместе с заголовками, подписями осей и примером использования функции text. Полученный в результате работы программы график представлен на рис. 4.45.

```
t=-2*pi:pi/50:2*pi;
x=sin(t);
plot(t,x);
xlabel('t');
ylabel('x');
title('Plot function x=sin(t)');
text(-1,-0.8,'<- Point (-1,-0.8)');
```

Листинг 4.41.

Все рассмотренные функции вывода текста формирует указатель на созданный текстовый объект, у которого с помощью функции `set` можно установить соответствующие свойства, наиболее часто встречающиеся из которых приведены ниже:

- `color` определяет цвет шрифта;
- `backroundcolor` позволяет определить цвет фона;
- `fontangle` позволяет установить наклон шрифта;
- `fontname` определяет название шрифта;
- `fontsize` определяет размер шрифта в пунктах;
- `fontweight` определяет толщину шрифта;

• `linestyle` позволяет изменять стиль прямоугольной рамки;
• `linewidth` определяет толщину линий прямоугольной рамки.

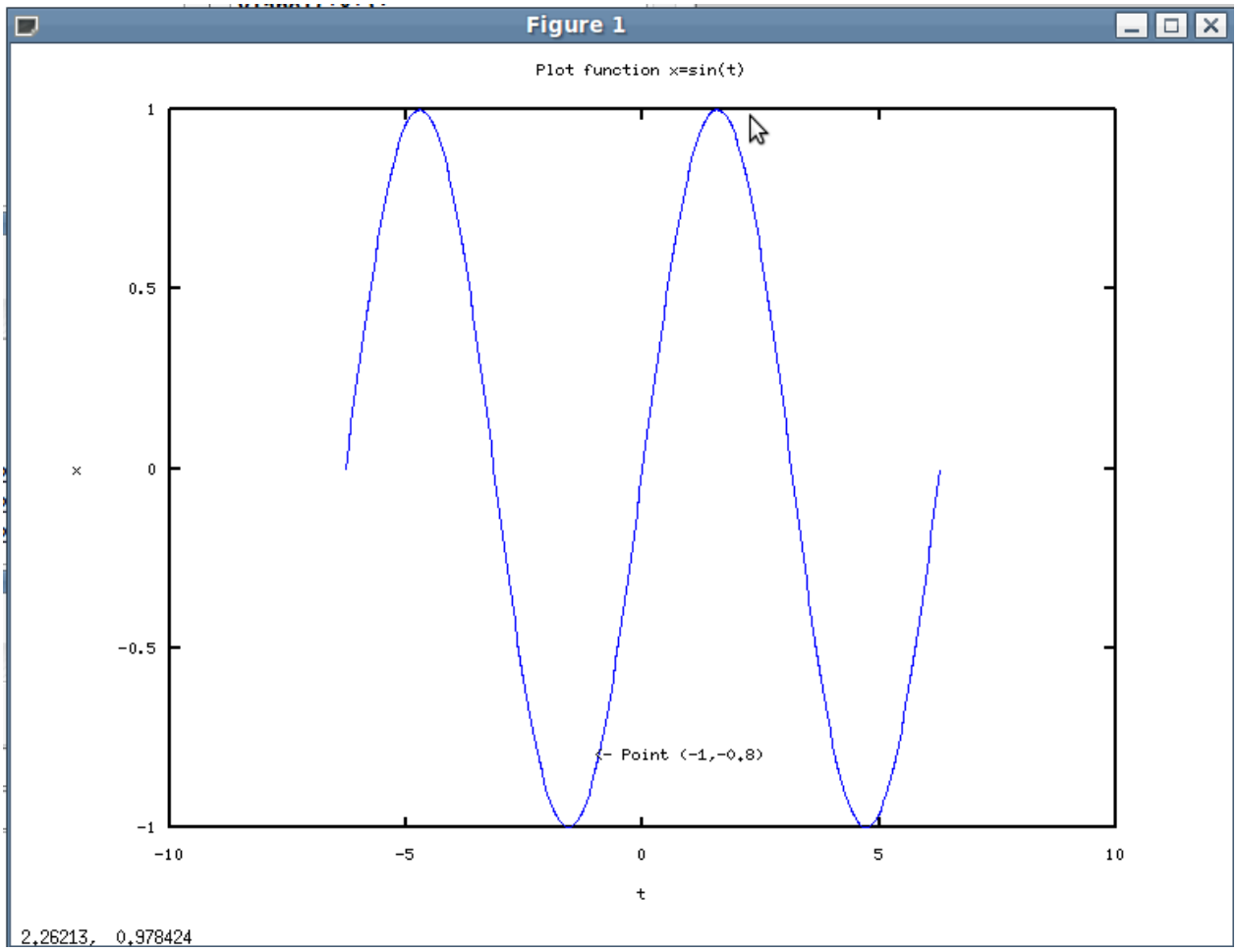


Рис. 4.45. График функции x=sin(t) с подписями

На этом мы заканчивает краткое знакомство с графическими объектами языка Octave и предлагаем читателю самостоятельно поэкспериментировать с описанными свойствами графических объектов при написании собственных программ.

В четвертой главе мы рассмотрели основные возможности Octave для построения двух- и трёхмерных графиков, а также средства языка программирования для работы с графиками.

5. Задачи линейной алгебры

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

5.1 Ввод и формирование векторов и матриц в Octave

Векторы и матрицы в Octave задаются путем ввода их элементов.

Элементы *вектора–строки* отделяют пробелами или запятыми, а всю конструкцию заключают в квадратные скобки:

```
>>> a=[2 -3 5 6 -1 0 7 -9]
a =
2 -3 5 6 -1 0 7 -9
>>> b=[-1,0,1]
b =
-1 0 1
```

Листинг 5.1

Вектор–столбец можно задать, если элементы отделять друг от друга точкой с запятой:

```
>>> c=[-pi;-pi/2;0;pi/2;pi]
c =
-3.14159
-1.57080
0.00000
1.57080
3.14159
```

Листинг 5.2

Обратится к элементу вектора можно, указав имя вектора, а в круглых скобках номер элемента под которым он хранится в этом векторе:

```
>>> a(1)
ans = 2
>>> b(3)
ans = 1
>>> c(5)
ans = 3.1416
```

Листинг 5.3

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

```
>>> Matr=[0 1 2 3;4 5 6 7]
Matr =
0 1 2 3
4 5 6 7
```

Листинг 5.4

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

```
>>> Matr(2,3)
ans = 6
>>> Matr(1,1)
```

```
ans = 0
>>> Matr(1,1)=pi;Matr(2,4)=-pi;
>>> Matr
Matr =
3.1416 1.0000 2.0000 3.0000
4.0000 5.0000 6.0000 -3.1416
```

Листинг 5.5

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

```
>>> a=[-3 0 2];b=[3 2 -1];c=[5 -2 0];
>>> %Горизонтальная конкатенация векторов-строк,
>>> %результат вектор-строка
>>> M=[a b c]
M = -3 0 2 3 2 -1 5 -2 0
>>> %Вертикальная конкатенация векторов-строк,
>>> %результат матрица
>>> N=[a;b;c]
N =
  -3   0   2
   3   2  -1
   5  -2   0
>>> %Горизонтальная конкатенация матриц
>>> Matrica=[N N N]
Matrica =
  -3   0   2  -3   0   2  -3   0   2
   3   2  -1   3   2  -1   3   2  -1
   5  -2   0   5  -2   0   5  -2   0
>>> %Вертикальная конкатенация матриц
>>> Tablica=[M;M;M]
Tablica =
  -3   0   2   3   2  -1   5  -2   0
  -3   0   2   3   2  -1   5  -2   0
  -3   0   2   3   2  -1   5  -2   0
```

Листинг 5.6

Важную роль при работе с матрицами играет знак двоеточия «:». Примеры с подробными комментариями приведены в листинге 5.7.

```
>>>Tabl=[-1.2 3.4 0.8;0.9 -0.1 1.1;7.6 -4.5 5.6;9.0 1.3 -8.5]
    Tabl =
      -1.20000   3.40000   0.80000
       0.90000  -0.10000   1.10000
       7.60000  -4.50000   5.60000
       9.00000   1.30000  -8.50000
    >>> %Выделить из матрицы 3-й столбец
    >>> Tabl(:,3)
    ans =
       0.80000
       1.10000
       5.60000
      -8.50000
    >>> %Выделить из матрицы 1-ю строку
    >>> Tabl(1,:)octave_all.odt
```

```
ans =   -1.20000   3.40000   0.80000
>>> %Выделить из матрицы подматрицу
>>> Matr=Tabl(2:3,1:2)
Matr =
   0.90000  -0.10000
   7.60000  -4.50000
>>> %Вставить подматрицу в проctave_all.odtавый нижний угол
>>> %исходной матрицы
>>> Tabl(3:4,2:3)=Matr
Tabl =
  -1.20000   3.40000   0.80000
   0.90000  -0.10000   1.10000
   7.60000   0.90000  -0.10000
   9.00000   7.60000  -4.50000
>>> %Удалить из матрицы 2-й столбец
>>> Tabl(:,2)=[]
Tabl =
  -1.20000   0.80000
   0.90000   1.10000
   7.60000  -0.10000
   9.00000  -4.50000
>>> %Удалить из матрицы 2-ю строку
>>> Tabl(2,:)=[]
Tabl =
  -1.20000   0.80000
   7.60000  -0.10000
   9.00000  -4.50000
>>>%Представить матрицу в виде вектора-столбца
>>> Matr
Matr =
   0.90000  -0.10000
   7.60000  -4.50000
>>> Vector=Matr(:)
Vector =
   0.90000
   7.60000octave_all.odt
  -0.10000
  -4.50000
>>> %Выделить из вектора элементы со 1-го по 3-й
>>> V=Vector(1:3)
V =
   0.90000
   7.60000
  -0.10000
>>> %Удалить из массива 2-й элемент
>>> V(2)=[]
V =
   0.90000
  -0.10000
```

Листинг 5.7

5.2 Действия над векторами в Octave

Рассмотрим *действия над векторами* предусмотренные в Octave.

Для *сложения векторов* используют знак «+» (листинг 5.8). Операция сложения определена только для векторов одного типа, то есть суммировать можно либо векторы–столбцы, либо векторы–строки одинаковой длины.

```
>>> a=[2 4 6];b=[1 3 5];
>>> c=a+b
c =
    3    7   11
```

Листинг 5.8

Вычитание векторов выполняется с помощью знака «–»:

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

Листинг 5.9

Знак апострофа «'» применяют для *транспонирования вектора:*

```
>>> a'
ans =
   2
   4
   6
>>> b'
ans =
   1
   3
   5
>>> t=(a+b)'
t =
    3
    7
   11
```

Листинг 5.10

Умножение вектора на число осуществляется с помощью знака «*».

```
>>> a=[2 4 6];b=[1 3 5];
>>> z=2*a+0.5*b
z =
    4.5000    9.5000   14.5000
```

Листинг 5.11

Знак деления «/» применяют для того, чтобы *разделить вектор на число*:

```
>>> z=2*a+b/2
z =
    4.5000    9.5000   14.5000
```

Листинг 5.12

Умножение вектора на вектор выполняется так же с помощью знака «*». Перемножать можно только векторы одинакового размера, причем один из них должен быть вектором–столбцом, а второй вектором–строкой. Примеры умножения векторов показаны в листинге 5.13.

```
>>> a=[2 4 6];b=[1 3 5];
>>> %В результате умножения вектора-строки
```

```
>>> % на вектор-столбец получится число
>>> a*b'
ans =  44
>>> %В результате умножения вектора-столбца
>>> %на вектор-строку получится матрица
>>> a'*b
ans =
    2    6   10
    4   12   20
    6   18   30
>>> % Некорректное умножение векторов
>>> a*b
error: operator *: nonconformant arguments
                                (op1 is 1x3, op2 is 1x3)
>>> a'*b'
error: operator *: nonconformant arguments
                                (op1 is 3x1, op2 is 3x1)
```

Листинг 5.13

Все перечисленные действия над векторами определены в математике и относятся к так называемым *векторным вычислениям*. Но Octave 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 5.14).

```
>>> x=[-pi/2,-pi/3,-pi/4,0,pi/4,pi/3,pi/2]
x =
-1.5708 -1.0472 -0.7854 0.0000 0.7854 1.0472 1.5708
>>> y=sin(2*x)+cos(2*x)
y =
-1.0000 -1.36603 -1.0000 1.0000 1.0000 0.36603 -1.0000
>>> y=2*exp(x/5)
y = 1.4608 1.6221 1.7093 2.0000 2.3402 2.4660 2.7382
```

Листинг 5.14

Рассмотрим еще несколько операций поэлементного преобразования вектора. К каждому элементу вектора можно *добавить* (*вычесть*) *число*, используя арифметическую операцию «+» («−»):

```
>>> x=[-pi/2,-pi/3,-pi/4,0,pi/4,pi/3,pi/2];
>>> x-1.2+e/3
ans =
-1.86470 -1.34110 -1.07930 -0.29391 0.49149 0.75329 1.27689
```

Листинг 5.15

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

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

Листинг 5.16

Поэлементное деление одного вектора на другой осуществляется при помощи следующей конструкции «./». 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). Примеры деления векторов показаны в листинге 5.17.

```
>>> a=[2 4 6];b=[1 3 5];
>>> a./b
ans =
   2.0000   1.3333   1.2000
>>> a.\b
ans =
   0.50000   0.75000   0.83333
```

Листинг 5.17

Поэлементное возведение в степень выполняет оператор «.^», результатом является вектор, каждый элемент которого это соответствующий элемент заданного вектора, возведенный в указанную степень (листинг 5.18).

```
>>> a=[2 4 6];b=[1 3 5];
>>>% Каждый элемент вектора возвести в квадрат
>>> a.^2
ans =     4   16   36
>>>% Извлечь корень квадратный из каждого элемента вектора
>>> b.^(1/2)
ans =    1.0000   1.7321   2.2361
>>>% Каждый элемент вектора b возвести в степень a
>>> b.^a
ans =        1      81   15625
>>>% Извлечь корень b-й степени
>>>%из каждого элемента вектора a
>>> a.^(1./b)
ans =
   2.0000   1.5874   1.4310
```

Листинг 5.18

5.3 Действия над матрицами в Octave

Начнем 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 (вычитаемые) матрицы должны быть одной размерности. Результатом такой операции является матрица:

```
>>> Matr_1=[1 2 3;4 5 6;7 8 9]
Matr_1 =
   1   2   3
   4   5   6
 7   8   9
>>> Matr_2=[0 9 8;7 6 5;4 3 2]
Matr_2 =
   0   9   8
   7   6   5
   4   3   2
>>> Matr_3=Matr_1+Matr_2
```

```
Matr_3 =
    1   11   11
   11   11   11
   11   11   11
>>> Matr_4=Matr_2-Matr_1
Matr_4 =
  -1   7   5
   3   1  -1
  -3  -5  -7
```

Листинг 5.19

Умножать на число «*» можно любую матрицу, результатом так же будет матрица, каждый элемент которой будет помножен на заданное число.

```
>>> Matr_1=[1 2 3;4 5 6;7 8 9];
>>> Matr_5=0.2*Matr_1
Matr_5 =
   0.20000   0.40000   0.60000
   0.80000   1.00000   1.20000
   1.40000   1.60000   1.80000
```

Листинг 5.20

Операция транспонирования «'» меняет в заданной матрице строки на столбцы и так же применима к матрицам любой размерности.

```
>>> Matr_5'
ans =
   0.20000   0.80000   1.40000
   0.40000   1.00000   1.60000
   0.60000   1.20000   1.80000
```

Листинг 5.21

При *умножении матриц* «*» важно помнить, что число столбцов первой перемножаемой матрицы должно быть равно числу строк второй. Примеры умножения матриц показаны в листинге 5.22.

```
>>> Matr_1=[1 2 3;4 5 6;7 8 9];
>>> Matr_2=[0 9 8;7 6 5;4 3 2];
>>> Matr_1*Matr_2
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
error: operator *: nonconformant arguments
                                (op1 is 2x2, op2 is 3x2)
```

Листинг 5.22

Возведение матрицы в степень «^» эквивалентно ее умножению на себя указанное число раз. При этом целочисленный показатель степени может быть как положительным, так

и отрицательным. Матрица в степени −1 называется *обратная 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. Примеры возведения в степень можно увидеть в листинге 5.23.

```
>>> Matr_6=[3 2 1;1 0 2;4 1 3];
>>> Matr_6^3
ans =
    92    40    59
    65    29    40
   146    65    92
>>> Matr_6^(-1)
ans =
  -0.40000  -1.00000   0.80000
   1.00000   1.00000  -1.00000
   0.20000   1.00000  -0.40000
>>> Matr_6^(-3)
ans =
   0.544000   1.240000  -0.888000
  -1.120000  -1.200000   1.240000
  -0.072000  -1.120000   0.544000
```

Листинг 5.23

Для *поэлементного преобразования матриц* (листинг 5.24) можно применять операции, описанные ранее, как операции поэлементного преобразования векторов: *добавление* (*вычитание*) *числа к каждому элементу матрицы* «+» («−»), *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*.

```
>>> 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 =
   1.33333  -0.66667  -0.33333
   3.00000   2.00000  -0.66667
  -1.00000  -0.33333   0.66667
>>> 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 =
   0.75000  -1.00000  -1.00000
   0.11111   0.16667  -1.00000
  -1.33333  -1.00000   1.50000
>>> M.\N
ans =
   1.33333  -1.00000  -1.00000
   9.00000   6.00000  -1.00000
  -0.75000  -1.00000   0.66667
>>> M.^0.2
ans =
   1.2457   1.1487   1.0000
   1.0000   1.0000   1.1487
   1.3195   1.0000   1.2457
>>> N.^M
ans =
   64    4   -1
    9    6    4
   81   -1    8
```

Листинг 5.24

Рассмотрим работу с матрицами на следующем примере.

ЗАДАЧА 5.1. Вычислить математическое выражение $(2\mathrm{A}+\frac{1}{3}B^T)^2-AB^{-1}$ для заданных значений *A* и *B*.

Решение задачи показано в листинге 5.23.

```
>>> 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 =
   32.667  -20.667   20.667
   47.333   26.889   15.667
  -40.333   75.333   31.778
```

Листинг 5.25

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 $B\cdot A^{-1}$, ее удобно использовать для решения матричных уравнений вида $X\cdot A=B$:

```
>>> A=[2 -1 2;-1 2 -2;2 -2 5]
A =
   2  -1   2
  -1   2  -2
   2  -2   5
>>> B=[7 0 0;0 1 0;0 0 1]
B =
   7   0   0
   0   1   0
   0   0   1
>>> X=B/A

X =
```

```
   6.00000   1.00000  -2.00000
   0.14286   0.85714   0.28571
  -0.28571   0.28571   0.42857
>>>% Проверка XA-B=0
>>> X*A-B
ans =
  -8.8818e-16   4.4409e-16   6.6613e-16
   0.0000e+00   2.2204e-16  -2.2204e-16
   0.0000e+00   5.5511e-17  -2.2204e-16
```

Листинг 5.26

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

```
>>> A=[2 -1 2;-1 2 -2;2 -2 5];
>>> B=[7 0 0;0 1 0;0 0 1];
>>> X=A\B
X =
   6.00000   0.14286  -0.28571
   1.00000   0.85714   0.28571
  -2.00000   0.28571   0.42857
>>>% Проверка AX-B=0
>>> A*X-B
ans =
  -8.8818e-16   0.0000e+00   0.0000e+00
   4.4409e-16   2.2204e-16   5.5511e-17
   6.6613e-16  -2.2204e-16  -2.2204e-16
```

Листинг 5.27

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 5.28).

```
>>> A=[1 2;1 1];
>>> b=[7;6];
>>> x=A\b
x =
   5
   1
>>>% Проверка Ax=b
>>> A*x
ans =
   7
   6
```

Листинг 5.28

5.4 Функции для работы с матрицами и векторами

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

- функции для работы с векторами;
- функции для работы с матрицами;
- функции, реализующие численные алгоритмы решения задач линейной алгебры.

Рассмотрим наиболее часто используемые функции.

5.4.1 Функции для работы с векторами

- `length(X)` – определяет длину вектора `X`;

```
>>> 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
```

Листинг 5.29

- `prod(X)` – вычисляет произведение элементов вектора `X`;

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

Листинг 5.30

- `cumprod(X)` – формирует вектор того же типа и размера, что и `X`, каждый элемент которого рассчитывается по формулам $x_1, x_1 \cdot x_2, x_1 \cdot x_2 \cdot x_3, \ldots, x_1 \cdot x_2 \cdot \ldots \cdot x_n$, то есть *i*–й элементу вектора `X` умножается на произведение всех предыдущих элементов;

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

Листинг 5.31

- `sum(X)` – вычисляет сумму элементов вектора `X`;

```
>>> X=[1 2 3 4 5 6 7 8 9];
>>> sum(X)
ans = 45
```

Листинг 5.32

- `cumsum(X)` – формирует вектор кумулятивной суммы, это вектор того же типа и размера, что и `X`, каждый элемент которого рассчитывается следующим образом $x_1, x_1+x_2, x_1+x_2+x_3, \ldots, x_1+x_2+\ldots+x_n$, то есть 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
```

Листинг 5.33

- `diff(X)` – формирует вектор, размер которого на единицу меньше чем у вектора `X`, а каждый элемент представляет собой разность между двумя соседними элементами

массива X, то есть $x_2-x_1, x_3-x_4, ..., x_n-x_{n-1}$;

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

Листинг 5.34

- `min(X)` – находит минимальный элемент вектора `X`, вызов в формате `[nomX, nom]=min(X)` дает возможность определить минимальный элемент `nomX` и его номер `nom` в массиве `X`;

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

Листинг 5.35

- `max(X)` – находит максимальный элемент массива `X` или при `[nomX,nom]=max(X)` определяет максимум и его номер;

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

Листинг 5.36

- `mean(X)` – определяет среднее арифметическое массива `X`;

```
>> X=[-1 2 3 9 -8 7 5];
>>> Sr=mean(X)
Sr =  2.4286
>>> sum(X)/length(X)
ans =  2.4286
```

Листинг 5.37

- `dot(x1,x2)` – вычисляет скалярное произведение векторов `x1` и `x2`;

```
>>> x1=[2 -3 0 5 1];x2=[0 1 -2 3 -4];
>>> dot(x1,x2)
ans =  8
>>> sum(x1.*x2)
ans =  8
>>> x1=[2;-3;0];x2=[0;1;-2];
>>> dot(x1,x2)
ans = -3
>>> sum(x1.*x2)
ans = -3
```

Листинг 5.38

- `cross(x1,x2)` – определяет векторное произведение векторов `x1` и `x2`;

```
>>> 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
```

Листинг 5.39

- `sort(X)` – выполняет сортировку массива `X`;

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

Листинг 5.40

5.4.2 Функции для работы с матрицами

- `eye(n [, m])` – возвращает единичную матрицу (вектор) соответствующей размерности;

```
>>> eye(4)
ans =
Diagonal Matrix
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
>>> eye(2,4)
ans =
Diagonal Matrix
   1   0   0   0
   0   1   0   0
>>> eye(3,1)
ans =
Diagonal Matrix
   1
   0
   0
>>> eye(1,5)
ans =
Diagonal Matrix
   1   0   0   0   0
```

Листинг 5.41

- `ones((n [, m, p, …])` – формирует матрицу (вектор), состоящую из единиц;

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

Листинг 5.42

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

```
>>> zeros(3)
ans =
   0   0   0
   0   0   0
   0   0   0
>>> zeros(1,1)
ans = 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 =
ans(:,:,1) =
    0    0
    0    0
ans(:,:,2) =
    0    0
    0    0
```

Листинг 5.43

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

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

```
    0    0    0    0    0    0    7    0
    0    0    0    0    0    0    0    5
    0    0    0    0    0    0    0    0
>>> diag(x,1)
ans =
    0    2    0    0
    0    0   -3    0
    0    0    0    0
    0    0    0    0
>>> 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
>>> x=[2;-3; 1];
>>> diag(x,1)
ans =
    0    2    0    0
    0    0   -3    0
    0    0    0    1
    0    0    0    0
>>> diag(x,-1)
ans =
    0    0    0    0
    2    0    0    0
    0   -3    0    0
    0    0    1    0
>>> diag(x,2)
ans =
    0    0    2    0    0
    0    0    0   -3    0
    0    0    0    0    1
    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    1    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,-1)
ans =
   4
   8
>>> diag(M,2)
ans =  3
>>> diag(M,-2)
ans =  7
```

Листинг 5.44

- `rand([n, m, p, …])` – возвращает матрицу (вектор), элементы которой — случайные числа, распределенные по равномерному закону, `rand` без аргументов возвращает одно случайно число;

```
>>> rand(2)
ans =
   0.15907   0.80147
   0.90460   0.40293
>>> rand(3,1)
ans =
   0.279005
   0.031504
   0.529279
>>> rand(1,4)
ans =
   0.85038   0.13899   0.50764   0.82887
>>> rand(2,5)
ans =
   0.782173   0.286649   0.563683   0.969862   0.708655
   0.300415   0.545783   0.011614   0.143827   0.644821
>>> rand
ans =  0.99252
>>> rand
ans =  0.42848
```

Листинг 5.45

- `randn([n, m, p…])` – возвращает матрицу (вектор), элементы которой — случайные числа распределенные по нормальному закону; `randn` без аргументов – возвращает одно случайно 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
>>> randn
ans = -0.75891
```

Листинг 5.46

- `linspace(a, b [, n])` – возвращает массив из 100 или из `n` точек равномерно распределенных между значениями `a` и `b`;

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

Листинг 5.47

- `logspace(a, b [, n])` – формирует массив из 50 или из n точек равномерно распределенных в логарифмическом масштабе между значениями `10a` и `10b`; функция `logspace(a, pi)` дает равномерное распределение из 50 точек в интервале от `10a` до `pi`;

```
>>> logspace(1,2,5)
ans =
    10.000    17.783    31.623    56.234   100.000
>>> logspace(2,pi)
ans =
 Columns 1 through 7:
100.0000 93.1815 86.8279 80.9075 75.3908 70.2503 65.4602
 Columns 8 through 14:
```

```
60.9968 56.8377 52.9622 49.3510 45.9860 42.8504 39.9287
 Columns 15 through 21:
37.2061 34.6692 32.3053 30.1025 28.0500 26.1374 24.3552
 Columns 22 through 28:
22.6945 21.1471 19.7052 18.3616 17.1096 15.9430 14.8559
 Columns 29 through 35:
13.8429 12.8991 12.0195 11.2000 10.4363 9.7247 9.0616
 Columns 36 through 42:
8.4438 7.8680 7.3315 6.8316 6.3658 5.9318 5.5273
 Columns 43 through 49:
5.1504 4.7992 4.4720 4.1671 3.8829 3.6182 3.3715
 Column 50:     3.1416
```

Листинг 5.48

- `repmat(M, n [, m])` – формирует матрицу состоящую `n` на `n` или из `n` на `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(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
```

Листинг 5.49

- `reshape(M, m, n)` – возвращает матрицу размерностью `m` на `n` сформированную из `M` путем последовательной выборки по столбцам, если `M` не имеет `m` на `n` элементов, то выдается сообщение об ошибке;

```
>>> M=[0 1 2 3;4 5 6 7;8 9 0 1]
M =
   0   1   2   3
   4   5   6   7
   8   9   0   1
>>> reshape(M,3,2)
>>>error: reshape: can't reshape 3x4 array to 3x2 array
>>> reshape(M,3,4)
ans =
   0   1   2   3
   4   5   6   7
   8   9   0   1
>>> reshape(M,4,3)
ans =
   0   5   0
   4   9   3
   8   2   7
   1   6   1
>>> reshape(M,2,6)
ans =
   0   8   5   2   0   7
   4   1   9   6   3   1
>>> reshape(M,6,2)
ans =
   0   2
   4   6
   8   0
   1   3
   5   7
   9   1
>>> reshape(M,1,12)
ans =    0   4   8   1   5   9   2   6   0   3   7   1
>>> reshape(M,12,1)
ans =
   0
   4
   8
   1
   5
   9
   2
   6
   0
   3
   7
   1
```

Листинг 5.50

- `cat(n, A, B, [C, …])` – объединяет матрицы `A` и `B` или все входящие матрицы, если `n=1`, слияние матриц происходит по столбцам, если `n=2` – по строкам;

```
>>> 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
```

Листинг 5.51

- `rot90(M [, k])` – осуществляет поворот матрицы `M` на 90 градусов или на величину `90k`, где `k` – целое число;

```
>>> M=[0 1 2 3;4 5 6 7;8 9 0 1]
M =
   0   1   2   3
   4   5   6   7
   8   9   0   1
>>> rot90(M)
ans =
   3   7   1
   2   6   0
   1   5   9
   0   4   8
>>> rot90(M,2)
ans =
   1   0   9   8
   7   6   5   4
   3   2   1   0
>>> rot90(M,3)
ans =
   8   4   0
   9   5   1
   0   6   2
   1   7   3
```

Листинг 5.52

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

Листинг 5.53

- `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
>>> triu(X)
ans =
  -1   2   3   9  -8   7   5
>>> triu(X')
ans =
  -1
   0
   0
   0
   0
   0
   0
```

Листинг 5.54

- `size(M)` – определяет число строк и столбцов матрицы `A`, результатом ее работы является вектор [`n`, `m`];

```
>>> M=[0 1 2 3 ;4 5 6 7;8 9 0 1;6 5 4 3];
>>> size(M)
ans =    4   4
>>> X=[-1 2 3 9 -8 7 5];
>>> size(X)
ans =    1   7
>>> size(X')
ans =    7   1
>>> eye(size(M))
ans =
Diagonal Matrix
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
>>> zeros(size(X))
ans =    0   0   0   0   0   0   0
```

Листинг 5.55

- `prod(M [,k])` – формирует вектор–строку или вектор–столбец, в зависимости от значения `k`, каждый элемент которого является произведением элементов соответствующего столбца (`k=1`) или строки (`k=2`) матрицы `M`, если значение параметра `k` в конструкции отсутствует, то по умолчанию вычисляются произведения столбцов матрицы; понятно, что результатом работы функции `prod(prod(A))` будет произведение всех элементов матрицы;

```
>>> M=[-1 1 -2 3 ;4 5 -1 2;3 -1 4 1;-2 5 4 3];
>>> prod(M)
ans =
   24  -25   32   18
>>> prod(M,1)
ans =
   24  -25   32   18
>>> prod(M,2)
ans =
     6
   -40
   -12
  -120
>>> prod(prod(M))
ans = -345600
```

Листинг 5.56

- `cumprod(M [,k])` – отличается от функции `cumprod(X)` тем, что операции описанные для нее применяются либо к строкам либо к 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    1   -2    3
   -4    5    2    6
  -12   -5    8    6
   24  -25   32   18
>>> 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
```

Листинг 5.57

- `sum(M [,k])` – формирует вектор–строку или вектор–столбец, в зависимости от значения `k`, каждый элемент которого является суммой элементов соответствующего столбца (`k=1`) или строки (`k=2`) матрицы `M`, если значение параметра `k` в конструкции отсутствует, то по умолчанию вычисляются суммы столбцов матрицы; произведение всех элементов матрицы вычисляет функция `sum(sum(M))`;

```
>>> M=[-1 1 -2 3 ;4 5 -1 2;3 -1 4 1;-2 5 4 3];
>>> sum(M)
ans =
    4   10    5    9
```

```
>>> sum(M,1)
ans =    4   10    5    9
>>> sum(M,2)
ans =
    1
   10
    7
   10
>>> sum(sum(M))
ans =  28
```

Листинг 5.58

- `cumsum(M, [k])` – отличается от функции `cumsum(X)` тем, что операции описанные для нее 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    1   -2    3
    3    6   -3    5
    6    5    1    6
    4   10    5    9
>>> 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
```

Листинг 5.59

- `diff(M)` – из матрицы `M` размерностью `n` на `m` формирует матрицу размером `n-1` на `m` элементы которой представляют собой разность между элементами соседних строк `M`;

```
>>> 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
>>> diff(M)
ans =
   5   4   1  -1
  -1  -6   5  -1
  -5   6   0   2
```

Листинг 5.60

- `min(M)` – формирует вектор–строку каждый элемент которого является наименьшим элементом соответствующего столбца матрицы `M`, определить положение этих элементов в матрице можно, если вызвать функцию в формате `[n, m]=min(M)`, где `n` – это вектор минимальных элементов столбцов матрицы `M`, а `m` – вектор номеров строк матрицы `M`, в которых находятся эти элементы, конструкция `min(min(M))` позволит отыскать минимум среди всех элементов матрицы (листинг 5.61); вызов функции в виде `min(M, [], k)` или `[n, m]=min(M, [], k)` позволяет управлять направлением поиска, в частности можно отыскать минимальные элементы и их положение в строках матрицы `M` (листинг 5.62); и наконец функция `min(A, B)` сформирует матрицу (листинг 5.63) из строк `min(A)` и `min(B)`;

```
>>> 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
>>> min(M)
ans =
  -2  -1  -2   1
>>> [n,m]=min(M)
n =
  -2  -1  -2   1
m =
   4   3   1   3
>>> min(M')
ans =
  -2  -1  -1  -2
>>> [n,m]=min(M')
n =
  -2  -1  -1  -2
m =
   3   3   2   1
>>> min(min(M))
ans = -2
>>> [n,m]=min(min(M))
n = -2
m =  1
```

Листинг 5.61

```
>>> min(M,[],1)
ans =   -2  -1  -2   1
>>> %То же что и min(M), то есть Формирует вектор-строку,
>>> %каждый элемент которого равен минимальному элементу
>>> %в соответствующем столбце матрицы M
>>> min(M,[],2)
ans =
  -2
  -1
  -1
  -2
>>>%Формирует вектор-столбец, каждый элемент которого,
```

```
>>>%равен минимальному элементу в соответствующей
>>>%строке матрицы M
>>> [n,m]=min(M,[],2)
n =
  -2
  -1
  -1
  -2
m =
   3
   3
   2
   1
>>> %и их положение в матрице, то есть номера столбцов
>>> %в которых они находятся
```

Листинг 5.62

```
>>> A=[0 1 2;3 4 5;6 7 8];B=[11 12 13;14 15 16;17 18 19];
>>> min(A,B)
ans =
   0   1   2
   3   4   5
   6   7   8
>>> %Первая строка результирующей матрицы равна минимумам
>>> %столбцов матрицы A, а вторая матрицы B
```

Листинг 5.63

- `max(M)` – формирует вектор–строку каждый элемент которого является наибольшим элементом соответствующего столбца матрицы `M`, действия функций `[n, m]= max(M),max(max(M)),max(M, [], k),[n, m]= max(A, [], k),max(A,B)` понятно из примеров листинга 5.64;

```
>>> M=[-1 1 -2 3 ;4 5 -1 2;3 -1 4 1;-2 5 4 3];
>>> max(M)
ans =
   4   5   4   3
>>> [n,m]=max(M)
n =
   4   5   4   3
m =
   2   2   3   1
>>> max(M')
ans =    3   5   4   5
>>> [n,m]=max(M')
n =
   3   5   4   5
m =
   4   2   3   2
>>> max(max(M))
ans =  5
>>> [n,m]=max(max(M))
n =  5
m =  2
```

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

Листинг 5.64

- `mean(M, [k])` – формирует вектор–строку или вектор–столбец, в зависимости от значения `k`, каждый элемент которого является средним значением элементов соответствующего столбца или строки матрицы `M`, если значение параметра `k` в конструкции отсутствует, то по умолчанию вычисляются средние значения столбцов матрицы; среднее всех элементов матрицы вычисляет функция `mean(mean(M))`;

```
>>> M=[-1 1 -2 3 ;4 5 -1 2;3 -1 4 1;-2 5 4 3];
>>> mean(M)
ans =    1.0000   2.5000   1.2500   2.2500
>>> mean(M,1)
ans =    1.0000   2.5000   1.2500   2.2500
>>> mean(M,2)
ans =
   0.25000
   2.50000
   1.75000
   2.50000
>>> mean(mean(M))
ans =  1.7500
```

Листинг 5.65

- `sort(M)` – выдает матрицу того же размера, что и `M`, каждый столбец которой упорядочен по возрастанию;

```
>>> 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
>>> sort(M)
ans =
  -2  -1  -2   1
  -1   1  -1   2
   3   5   4   3
   4   5   4   3
>>> sort(M')
ans =
  -2  -1  -1  -2
  -1   2   1   3
   1   4   3   4
   3   5   4   5
>>> -sort(-M)
ans =
   4   5   4   3
   3   5   4   3
  -1   1  -1   2
  -2  -1  -2   1
>>> -sort(-M')
ans =
   3   5   4   5
   1   4   3   4
  -1   2   1   3
  -2  -1  -1  -2
```

Листинг 5.66

- `sqrtm(A)` – относится к так называемым матричным функциям и возвращает матрицу `X`, для которой `X*X=A`;

```
>>> A=[1 0 -3;0 1 2;2 0 -1]
A =
   1   0  -3
   0   1   2
   2   0  -1
>>> X=sqrtm(A)
X =
   1.53024   0.00000  -1.41861
  -0.35349   1.00000   0.94574
   0.94574   0.00000   0.58450
>>> X*X %Проверка
ans =
   1.00000   0.00000  -3.00000
   0.00000   1.00000   2.00000
   2.00000   0.00000  -1.00000
>>>%Извлечение квадратного корня из каждого элемента матрицы A
>>> Y=sqrt(A)
Y =
1.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 1.73205i
```

```
0.00000 + 0.00000i 1.00000 + 0.00000i 1.41421 + 0.00000i
1.41421 + 0.00000i 0.00000 + 0.00000i 0.00000 + 1.00000i
>>> %sqrtm(A) и sqrt(A) дают различные результаты
>>> Y*Y%Матричное умножение
ans =
1.00000 + 2.44949i 0.00000 + 0.00000i -1.73205 + 1.73205i
2.00000 + 0.00000i 1.00000 + 0.00000i 1.41421 + 1.41421i
1.41421 + 1.41421i 0.00000 + 0.00000i -1.00000 + 2.44949i
>>> Y.*Y %Поэлементное умножение
ans =
   1.00000   0.00000  -3.00000
   0.00000   1.00000   2.00000
   2.00000   0.00000  -1.00000
```

Листинг 5.67

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

```
>>> A=[1 0 -3;0 1 2;2 0 -1];
>>> B=expm(A)
B =
  -0.26543   0.00000  -1.05553
   1.98914   2.71828   0.70369
   0.70369   0.00000  -0.96912
>>> logm(B)
ans =
 1.00000 + 0.00000i 0.00000 + 0.00000i -3.00000 - 0.00000i
-0.00000 - 0.00000i 1.00000 + 0.00000i  2.00000 + 0.00000i
 2.00000 + 0.00000i 0.00000 + 0.00000i -1.00000 - 0.00000i
```

Листинг 5.68

5.4.3 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
```

Листинг 5.69

- `trace(M)` – вычисляет след матрицы `M`, то есть 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
```

Листинг 5.70

- `norm(M [, p])` – возвращает различные виды норм матрицы `M` в зависимости от `p`, если аргумент `p = 1, 2, inf, fro` не задан, то вычисляется вторая норма матрицы `M`;

```
>>% Вторая норма
>>> norm(M)
ans =     8.5506
>>> norm(M,2)
ans =     8.5506
>>>% Первая норма
>>> norm(M,1)
ans =         12
>>>% Бесконечная норма
>>> norm(M,inf)
ans =         14
>>>% Евклидова норма
>>> norm(M,'fro')
ans =     11.916
```

Листинг 5.71

- `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 =     3.3324
>>> cond(M,2)
ans =     3.3324
>>> cond(M,1)
ans =     6.4927
>>> cond(M,inf)
ans =     6.7742
>>> cond(M,'fro')
ans =     5.7154
```

Листинг 5.72

- `rcond(M)` – вычисляет величину обратную значению числа обусловленности матрицы относительно первой нормы, если полученная величина близка к единице, то матрица хорошо обусловлена, если к 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 =     0.1738
```

Листинг 5.73

- `inv(M)` – возвращает матрицу обратную к `M`;

```
>>> A=[1 0 -3;0 1 2;2 0 -1];
>>> invA=inv(A)
invA =
        -0.2           0         0.6
         0.8           1        -0.4
        -0.4           0         0.2
>>>%Проверка
```

```
>>> A*invA
ans =
           1           0  -5.5511e-17
           0           1           0
           0           0           1
```

Листинг 5.74

- `eig(M)` – возвращает вектор собственных значений матрицы `M`, вызов функции в формате `[Matr, D]= eig(M)` даст матрицу `Matr`, столбцы которой собственные векторы матрицы `M` и диагональную матрицу `D` , содержащую собственные значения матрицы `M`; функция `eig(A, B)`, где `A` и `B` квадратные матрицы, выдает вектор обобщенных собственных значений.

```
>>> M=[3 -2;-4 1]
M =
           3          -2
          -4           1
>>> eig(M)
ans =
           5
          -1
>>> [Matr,D]=eig(M)
Matr =
     0.70711     0.44721
    -0.70711     0.89443
D =
Diagonal Matrix
           5           0
           0          -1
>>> %Проверка A*M=M*D
>>> M*Matr
ans =
      3.5355    -0.44721
     -3.5355    -0.89443
>>> Matr*D
ans =
      3.5355    -0.44721
     -3.5355    -0.89443
```

Листинг 5.75

- `poly(M)` – возвращает вектор–строку коэффициентов характеристического полинома матрицы `M`;

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

Листинг 5.76

- `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
```

Листинг 5.77

- `chol(M)` – возвращает разложение по Холецкому для положительно определенной симметричной матрицы `M`; функция `L=chol(M)` возвращает верхнюю треугольную матрицу `L`, для которой $M = L^T \cdot L$, функция `L=chol(M,'lower')` возвращает нижнюю треугольную матрицу `L`, для которой $M = L \cdot L^T$; матрица `L` в обоих случаях со строго положительными элементами на главной диагонали;

```
>>> M=[10 2 1;2 10 3;1 3 10]
M =
      10   2    1
      2    10   3
      1    3    10
>>> [L]=chol(M)
L =
      3.16228   0.63246   0.31623
      0.00000   3.09839   0.90370
      0.00000   0.00000   3.01386
>>> L'*L
ans =
      10.0000   2.0000    1.0000
      2.0000    10.0000   3.0000
      1.0000    3.0000    10.0000
>>> [L]=chol(M,'lower')
L =
      3.16228   0.00000   0.00000
      0.63246   3.09839   0.00000
      0.31623   0.90370   3.01386
>>> L*L'
ans =
      10.0000   2.0000    1.0000
      2.0000    10.0000   3.0000
      1.0000    3.0000    10.0000
>>> A=[1 2;1 1];%Матрица не симметрическая
>>> chol(A)
error: chol: matrix not positive definite
>>%Матрица содержит отрицательные элементы
>>> M=[3 -2 1 5;6 -4 2 7;9 -6 3 12];
>>> chol(M)
error: CHOL requires square matrix
```

Листинг 5.78

- `lu(M)` – выполняет LU–разложение, функция `[L, U, P]=lu(M)` возвращает три матрицы: `L` – нижняя треугольная, `U` – верхняя треугольная и `P` – матрица перестановок, причем $P \cdot M = L \cdot U$; функция `lu(M)` без параметров возвращает одну матрицу, которая в свою очередь, является комбинацией матриц `L` и `U`;

```
>>> M=[3 -2 1; 5 6 -4; 2 7 9];
>>> lu(M)
ans =
           5           6          -4
         0.6        -5.6         3.4
         0.4    -0.82143      13.393
>>> [L,U,P]=lu(M)
L =
           1           0           0
         0.6           1           0
         0.4    -0.82143           1
U =
           5           6          -4
           0        -5.6         3.4
           0           0      13.393
P =
Permutation Matrix
   0   1   0
   1   0   0
   0   0   1
>>> L*U
ans =
           5           6          -4
           3          -2           1
           2           7           9
>>> P*M
ans =
           5           6          -4
           3          -2           1
           2           7           9
>>> lu(M)
ans =
           5           6          -4
         0.6        -5.6         3.4
         0.4    -0.82143      13.393
>>> triu(lu(M))
ans =
           5           6          -4
           0        -5.6         3.4
           0           0      13.393
>>> tril(lu(M))
ans =
           5           0           0
         0.6        -5.6           0
         0.4    -0.82143      13.393
```

Листинг 5.79

- `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`, причем $M \cdot P = Q \cdot P$;

```
>>> M=[3 -2 1; 5 6 -4; 2 7 9];
>>> [Q,R,P]=qr(M)
Q =
     0.10102    -0.27448     0.95627
    -0.40406     0.86703     0.29155
     0.90914     0.41584    0.023324
R =
      9.8995      3.7376     0.10102
           0       8.662      4.3434
           0           0      4.3732
P =
Permutation Matrix
   0   0   1
   0   1   0
   1   0   0
>>> M*P-Q*R
ans =
  -1.1102e-15  4.4409e-16  -4.4409e-16
  -4.4409e-16  -1.7764e-15           0
            0           0  -4.4409e-16
```

Листинг 5.80

- `svd(M)` – возвращает вектор сингулярных чисел матрицы, при использовании в формате `[U, S, V]= svd(M)` выполняет сингулярное разложение матрицы `M`, выдает 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 $M = U \cdot S \cdot V^T$;

```
>>> M=[3 -2 1; 5 6 -4; 2 7 9];
>>> svd(M)
ans =
11.7553
8.5347
3.7377
>>> [U,S,V]=svd(M)
U =
0.0057541 0.0207345 0.9997685
0.2528901 -0.9673161 0.0186059
0.9674779 0.2527245 -0.0108095
S =
Diagonal Matrix
11.7553 0 0
0 8.5347 0
0 0 3.7377
V =
```

```
0.27363 -0.50018 0.82155
0.70421 -0.47761 -0.52534
0.65515 0.72229 0.22154
```

Листинг 5.81

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

5.5 Решение некоторых задач алгебры матриц

Напомним основные определения алгебры матриц [7]. Если $m \times n$ выражений расставлены в прямоугольной таблице из m строк и n столбцов, то говорят о *матрице* размера $m \times n$:

$$\begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix}$$

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

$$\begin{matrix} a_{11} & a_{12} & \dots & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ii} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & \dots & \dots & a_{mn} \end{matrix}$$

Матрица размером $m \times n (m \neq n)$ называется *прямоугольной*.

$$\begin{matrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{matrix} \qquad \begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix}$$

В случае если $m = n$, то матрицу называют *квадратной матрицей* порядка *n*.

$$\begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{matrix}$$

В частности матрица типа $1 \times n$ это *вектор–строка:* $a_{11} \quad a_{12} \quad \dots \quad a_{1n}$

Матрица размером $m \times 1$ является в*ектором–столбцом:*

$$\begin{matrix} a_{11} \\ a_{21} \\ \dots \\ a_{m1} \end{matrix}$$

Число (скаляр) можно рассматривать как матрицу типа 1×1 - a_{11} .

Квадратная матрица $A = \{a_{ij}\}$ размером $n \times n$ называется

- *нулевой*, если все ее элементы равны нулю:

$$\begin{matrix} 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 0 \end{matrix}$$

- *верхней треугольной*, если все элементы, расположенные ниже главной диагонали, равны нулю:

$$\begin{matrix} a_{11} & a_{12} & ... & a_{1n} \\ 0 & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{matrix}$$

- *нижней треугольной*, если все элементы, расположенные выше главной диагонали, равны нулю:

$$\begin{matrix} a_{11} & 0 & ... & 0 \\ a_{21} & a_{22} & ... & 0 \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{matrix}$$

- *диагональной*, если все элементы, кроме элементов главной диагонали, равны нулю:

$$\begin{matrix} a_{11} & 0 & ... & 0 \\ 0 & a_{22} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{matrix}$$

- *единичной*, если элементы главной диагонали равны единице, а все остальные нулю:

$$\begin{matrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 1 \end{matrix}$$

Определителем (детерминантом) матрицы A является число $detA$ или Δ, вычисляемое по правилу:

$$det\, A = \sum (-1)^{\lambda} a_{1i_1} a_{2i_2} ... a_{ni_n} \quad ,$$

где сумма распределена на всевозможные перестановки ($i_1, i_2, ..., i_n$) элементов 1, 2, … , n и, следовательно, содержит $n!$ слагаемых, причем $\lambda = 0$, если перестановка четная, и $\lambda = 1$, если перестановка нечетная.

Квадратная матрица называется *невырожденной*, если ее 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, то есть 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 размера называется матрица $C = \{c_{ij}\}$ того же размера, элементы которой равны сумме соответствующих элементов матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$:

$$\{c_{ij}\} = \{a_{ij} + b_{ij}\} \quad .$$

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

$$\{c_{ij}\}=\{a_{ij}-b_{ij}\} \quad .$$

Произведением числа h на матрицу $A=\{a_{ij}\}$ (или *произведением матрицы на число*) называется матрица, элементы которой получены умножением всех элементов матрицы $A=\{a_{ij}\}$ на число h :

$$h\,A=\{h\cdot a_{ij}\} \quad .$$

Произведением матриц $A=\{a_{ij}\}$ размерностью $m\times n$ и $B=\{b_{ij}\}$ размерностью $p\times s$ является матрица $C=\{c_{ij}\}$ размерностью $m\times s$, каждый элемент которой можно представить формулой

$$C=\{c_{ij}\}=\{a_{i1}b_{1j}+a_{i2}b_{2j}+...+a_{in}b_{nj}\} \quad ,\ (\ i=1..\mathrm{m}\,,\ j=1..\mathrm{s}\ ).$$

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, то есть $A\cdot B\neq B\cdot A$. 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*

$$A^T=\{a_{ji}\} \quad .$$

В частности, для вектора–строки $a=\{a_1\,a_2\,a_3\,...\,a_n\}$ транспонированной матрицей является вектор–столбец:

$$a^T=\begin{Vmatrix} a_1 \\ a_2 \\ a_3 \\ ... \\ a_n \end{Vmatrix}$$

Обратной матрицей по отношению к данной матрице $A=\{a_{ij}\}$ размерностью $n\times n$, называется матрица $A^{-1}=\{A_{ij}\}$ того же размера, которая, будучи умноженной как справа, так и слева на данную матрицу, в результате дает единичную матрицу $E=\{\partial_{ij}\}$:

$$A\,A^{-1}=A^{-1}\,A=E \quad .$$

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

Перейдем к конкретным задачам.

ЗАДАЧА 5.2. Для матриц *A*, *B* и *C* проверить выполнение следующих тождеств:

- $(A\cdot B)\cdot C=A\cdot(B\cdot C)$;
- $(A^T+B)\cdot C=A^T\cdot C+B\cdot C$.

В листинге 5.82 видно, что матрицы получившиеся в результате вычисления левой и правой частей первого тождества равны, следовательно, первое предположение истинно. Для исследования второго тождества из левой части равенства вычитаем правую и получаем нулевую матрицу, что так же приводит к выводу об истинности предположения.

```
>>>%Исследование первого тождества
>>> A=[1 -2 0; -3 0 4]
A =
1 -2 0
-3 0 4
>>> B=[3 1;2 0;-1 1]
B =
```

```
3 1
2 0
-1 1
>>> C=[1 2;-1 0]
C =
1 2
-1 0
>>> (A*B)*C
ans =
-2 -2
-14 -26
>>> A*(B*C)
ans =
-2 -2
-14 -26
>>>%Исследование второго тождества
>>> (A'+B)*C-(A'*C+B*C)
ans =
   0   0
   0   0
   0   0
```

Листинг 5.82

ЗАДАЧА 5.3. Проверить является ли матрица симметрической. Квадратная матрица называется *симметрической*, если $A^T = A$.

В листинге 5.83 видно, что в результате вычитания из матрицы А транспонированной матрицы A^T получена нулевая матрица, то есть тождество $A^T = A \Rightarrow A^T - A = 0$ выполнено и заданная матрица симметрическая.

```
>>> A=[1 -0.5 1.5;-0.5 0 2.5;1.5 2.5 -2]
A =
 1.00000 -0.50000  1.50000
-0.50000  0.00000  2.50000
 1.50000  2.50000 -2.00000
>>> A-A'
ans =
0 0 0
0 0 0
0 0 0
```

Листинг 5.83

ЗАДАЧА 5.4. Проверить является ли матрица кососимметрической. Квадратная матрица называется *кососимметрической*, если $A^T = -A$.

Проверив равенство $A^T = -A \Rightarrow A^T + A = 0$ для заданной матрицы (листинг 5.84) убеждаемся в его истинности.

```
>>> A=[0 -0.25 0.75;0.25 0 -1.25;-0.75 1.25 0]
A =
   0.00000  -0.25000   0.75000
   0.25000   0.00000  -1.25000
  -0.75000   1.25000   0.00000
>>> A'+A
ans =
   0   0   0
```

```
   0   0   0
   0   0   0
```

Листинг 5.84

ЗАДАЧА 5.5. 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 транспонировать исходную матрицу и найти 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 5.85).

```
>>> A=[0.5 0.7071 0.5;0.7071 0 -0.7071;0.5 -0.7071 0.5]
A =
   0.50000   0.70710   0.50000
   0.70710   0.00000  -0.70710
   0.50000  -0.70710   0.50000
>>> %Определитель матрицы A отличен от нуля
>>> det(A)
ans = -0.99998
>>> %В результате вычитания из транспонированной матрицы A
>>> обратной к ней матрицы получаем нулевую матрицу.
>> Это значит что A – ортогональная.
an>>> A'-inv(A)
ans =
   0.0000e+00  -1.3562e-05   0.0000e+00
  -1.3562e-05   0.0000e+00   1.3562e-05
   0.0000e+00   1.3562e-05   0.0000e+00
```

Листинг 5.85

ЗАДАЧА 5.6. 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.

Решение задачи:

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

Листинг 5.86

ЗАДАЧА 5.7. 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:

$$A^{-1}\cdot A\cdot X=A^{-1}\cdot B \quad \text{или} \quad X\cdot A\cdot A^{-1}=B\cdot A^{-1} \ .$$

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 Octave, показано в листинге 5.87.

```
>>> A=[ 2 3;-2  6]
A =
   2   3
  -2   6
>>> B=[2 5;2/3 5/3]
B =
   2.00000   5.00000
   0.66667   1.66667
>>> %Решение матричного уравнения A·X=B
>>> %Первый способ
>>> X=A\B
X =
   0.55556   1.38889
   0.29630   0.74074
>>> %Второй способ
>>> X=inv(A)*B
X =
   0.55556   1.38889
   0.29630   0.74074
>>> %Проверка A·X-B=0
>>> A*X-B
ans =
   0.0000e+00   0.0000e+00
  -3.3307e-16  -6.6613e-16
>>> %Решение матричного уравнения X A=B
>>> %Первый способ
>> X=B/A
X =
   1.222222   0.222222
   0.407407   0.074074
>>> %Второй способ
>>> X=B*inv(A)
X =
   1.222222   0.222222
   0.407407   0.074074
>>> %Проверка X A-B=0
>>> X*A-B
ans =
   0.0000e+00   0.0000e+00
   1.1102e-16   0.0000e+00
```

Листинг 5.87

5.6 Решение систем линейных уравнений

Система *m* уравнений с *n* неизвестными вида [7]

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

называется *системой линейных уравнений*, причем x_j – *неизвестные*, a_{ij} – *коэффициенты при неизвестных*, b_i – *свободные коэффициенты* (i=1..m, j=1..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..*m*, *j*=1..*n*).

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

Если все $b_i = 0$, то речь идет 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 $A \cdot x = 0$ всегда разрешимы, так как последовательность $(x_1 = 0, x_2 = 0, \ldots, x_n = 0)$ удовлетворяет всем уравнениям системы. Решение $(x_1 = 0, x_2 = 0, \ldots, x_n = 0)$ 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:

$$x_1 + x_2 = 1$$
$$x_1 + x_2 = 3$$

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 *определенной*, если она обладает одним единственным решением, и *неопределенной*, если решений больше чем одно. Так, система

$$x_1 + 2x_2 = 7$$
$$x_1 + x_2 = 6$$

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

$$3x_1 - x_2 = 1$$
$$6x_1 - 2x_2 = 2$$

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

Совокупность всех решений неопределенной системы уравнений называется ее *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 $n \times m$. Вычеркиванием некоторых строк или столбцов из нее 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 $r(A) \leqslant min(n, m), r(A) = 0$, 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 идет об исследовании системы на совместность, следует помнить, что система n линейных уравнений с m неизвестными:

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

Существует не мало методов для практического отыскания решений систем линейных уравнений. Эти методы разделяют на *точные* и *приближенные*. Метод относится к классу точных, если с его помощью можно найти решение в результате конечного числа арифметических и логических операций. В этом разделе на конкретных примерах будут рассмотрены только точные методы решения систем.

ЗАДАЧА 5.8. Решить систему линейных уравнений

$$2x_1 - x_2 + 5x_3 = 0$$
$$3x_1 + 2x_2 - 5x_3 = 1$$
$$x_1 + x_2 - 2x_3 = 4$$

при помощи правила Крамера.

Правило Крамера заключается в следующем [7]. Если определитель $det\,A$ матрицы системы $A \cdot x = b$ 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 матрицы системы A заменой *i*–го столбца столбцом свободных членов b.

Итак, для решения поставленной задачи необходимо выполнить следующие действия:

- представить систему в матричном виде, то есть сформировать матрицу системы A и вектор правых частей b;
- вычислить главный определитель $det\,A$;
- сформировать вспомогательные матрицы для вычисления определителей det_i;
- вычислить определители det_i;
- найти решение системы по формуле $x_i = \frac{det_i}{det}$.

Листинге 5.88 содержит решение поставленной задачи.

```
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');
A*x'-b
%-------------------------------------------------
>>>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 = 6.0000
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 =
-17.000 91.000 25.000
Вектор решений СЛАУ Ax=b
x =
-2.8333 15.1667 4.1667
Проверка Ax-b=0
ans =
-4.4409e-15
3.5527e-15
8.8818e-16
```

Листинг 5.88

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

ЗАДАЧА 5.9. Решить систему линейных уравнений из задачи 5.8 методом обратной матрицы [7].

Метод обратной матрицы: для системы из n линейных уравнений с n неизвестными $A \cdot x = b$, при условии, что определитель матрицы A не равен нулю, единственное решение можно представить в виде $x = A^{-1} \cdot b$ (вывод формулы см. в задаче 5.7). Итак, для того, чтобы решить систему линейных уравнений методом обратной матрицы, необходимо выполнить следующие действия:

- сформировать матрицу коэффициентов и вектор свободных членов заданной системы;
- решить систему, представив вектор неизвестных как произведение матрицы обратной к матрице системы и вектора свободных членов (листинг 5.89).

```
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');
A*x
%--------------------------------------
>>>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 =
   -2.8333
   15.1667
    4.1667
Вектор решений СЛАУ Ax=b с помощью функции inv(A)
x =
   -2.8333
   15.1667
    4.1667
Проверка Ax=b
ans =
  -5.3291e-15
   1.0000e+00
   4.0000e+00
```

Листинг 5.89

ЗАДАЧА 5.10. Решить систему линейных уравнений

$$2x_1+x_2-5x_3+x_4=8;$$
$$x_1-3x_2-6x_4=9;$$
$$2x_2-x_3+2x_4=-5;$$
$$x_1+4x_2-7x_3+6x_4=0$$

методом Гаусса [6].

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

Метод Гаусса состоит из двух этапов. *Первый этап* это прямой ход, в результате которого расширенная матрица системы путем элементарных преобразований (перестановка уравнений системы, умножение уравнений на число отличное от нуля и сложение уравнений) приводится к ступенчатому виду:

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

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 выше изложенного, порядок решения задачи в Octave (листинг 5.90) следующий:

• сформировать матрицу коэффициентов A и вектор свободных членов b заданной системы;

• сформировать расширенную матрицу системы, объединив A и b ;

• используя функцию `rref` привести расширенную матрицу к ступенчатому виду;

• найти решение системы, выделив последний столбец матрицы, полученной в предыдущем пункте;

• выполнить вычисление $A \cdot x - b$, если в результате получился нулевой вектор, задача решена верно.

```
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');
A*x-b
%---------------------------------------
>>>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 =
   1.00000   0.00000   0.00000   0.00000   3.00000
   0.00000   1.00000   0.00000   0.00000  -4.00000
   0.00000   0.00000   1.00000   0.00000  -1.00000
   0.00000   0.00000   0.00000   1.00000   1.00000
Размерность матрицы C:
n =
   4   5
Вектор решений СЛАУ Ax=b
x =
   3.00000
  -4.00000
  -1.00000
   1.00000
Проверка Ax-b
ans =
   0.0000e+00
   1.7764e-15
  -8.8818e-16
  -1.6653e-15
```

Листинг 5.90

ЗАДАЧА 5.11. Решить систему линейных уравнений из задачи 5.10 с помощью LU–разложения [6].

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 $A=L\cdot U$:

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

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 $A\cdot x=b$ разложена в произведение треугольных матриц L и U , то можно записать уравнение: $L\cdot U\cdot x=b$.

Введя вектор вспомогательных переменных $y=(y_{1,}y_{2,}\dots,y_n)^T$, уравнение $L\cdot U\cdot x=b$ можно переписать в виде системы:

$$L\cdot y=b$$
$$U\cdot x=y$$

Таким образом, решение системы $A\cdot x=b$ с квадратной матрицей коэффициентов свелось к последовательному решению двух систем с треугольными матрицами коэффициентов.

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

Исходя из средств, которыми располагает Octave, решение поставленной задачи будет выглядеть так (листинг 5.91):

- сформируем матрицу коэффициентов A и вектор свободных членов b заданной системы;
- 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);
- по скольку в задаче речь идет о решении системы, то элементы вектора b должны занять места соответствующие строкам матрицы A, для чего необходимо выполнить действие $P \cdot b$;
- решим системы уравнений $L \cdot y = b$ относительно y;
- зная U и y найдем решение x системы $U \cdot x = y$.

```
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');
A*x-b
%---------------------------------------
>>>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:
L =
   1.00000   0.00000   0.00000   0.00000
   0.50000   1.00000   0.00000   0.00000
   0.50000  -1.00000   1.00000   0.00000
   0.00000  -0.57143  -0.21429   1.00000
U =
   2.00000   1.00000  -5.00000   1.00000
   0.00000  -3.50000   2.50000  -6.50000
   0.00000   0.00000  -2.00000  -1.00000
   0.00000   0.00000   0.00000  -1.92857
```

```
P =
Permutation Matrix
   1   0   0   0
   0   1   0   0
   0   0   0   1
   0   0   1   0
Y =
   1.00000   0.00000   0.00000   0.00000   8.00000
   0.00000   1.00000   0.00000   0.00000   5.00000
   0.00000   0.00000   1.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000  -1.92857
n =    4   5
y =
   8.0000
   5.0000
   1.0000
  -1.9286
X =
   1.00000   0.00000   0.00000   0.00000   3.00000
   0.00000   1.00000   0.00000   0.00000  -4.00000
   0.00000   0.00000   1.00000   0.00000  -1.00000
   0.00000   0.00000   0.00000   1.00000   1.00000
n =    4   5
x =
   3.0000
  -4.0000
  -1.0000
   1.0000
Проверка Ax-b
ans =
   0.0000e+00
   0.0000e+00
  -8.8818e-16
   6.6613e-16
```

Листинг 5.91

ЗАДАЧА 5.12. Решить систему линейных уравнений

$$\begin{aligned} 3x_1+x_2-x_3+2x_4&=6;\\ -5x_1+x_2+3x_3-4x_4&=-12;\\ 2x_1+x_3-x_4&=1;\\ x_1-5x_2+3x_3+3x_4&=3 \end{aligned}$$

с помощью QR–разложения.

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:

$$A\cdot x=b\Rightarrow Q\cdot R\cdot x=b\Rightarrow (Q\cdot y=b,\ R\cdot x=y)\ .$$

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

Как решить эту задачу средствами Octave, показано в листинге 5.92.

```
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)
%--------------------------------------
>>>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 =
   0.480384   0.303216   0.358483   0.740797
  -0.800641   0.020214   0.545518   0.246932
   0.320256   0.070750   0.735670  -0.592638
   0.160128  -0.950077   0.180800   0.197546
R =
   6.24500  -1.12090  -2.08167   3.36269
   0.00000   5.07381  -3.02205   3.30505
   0.00000   0.00000   2.55614  -2.74318
   0.00000   0.00000   0.00000   0.49386
y =
   13.2906
   -1.2028
   -3.1172
    1.4816
X =
   1.00000   0.00000   0.00000   0.00000   1.00000
   0.00000   1.00000   0.00000   0.00000  -1.00000
   0.00000   0.00000   1.00000   0.00000   2.00000
   0.00000   0.00000   0.00000   1.00000   3.00000
x =
   1.00000
  -1.00000
   2.00000
   3.00000
```

Листинг 5.92

ЗАДАЧА 5.13. Исследовать систему на совместность и если возможно решить ее:

а) $$\begin{cases} x_1+2x_2+5x_3=-9 \\ x_1-x_2+3x_3=2 \\ 3x_1-6x_2-x_3=25 \end{cases}$$

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

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 5.93.) введем исходные данные , то есть матрицу коэффициентов системы и вектор правых частей. Затем выполним вычисление рангов матрицы коэффициентов и расширенной матрицы системы.

В случае а) ранги матриц равны и совпадают с количеством неизвестных $r(A|b)=r(A)=3$, значит, система совместна и имеет единственное решение.

Вычисление рангов матрицы системы и расширенной матрицы системы б) показывает, что ранг расширенной матрицы больше ранга матрицы системы $r(A|b)>r(A)$, что означает несовместность системы.

В процессе вычислений для системы в) выясняется, что 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.

```
disp('Исследование системы на совместность');
disp('Введите матрицу системы:');
A=input('A=');
disp('Введите вектор свободных коэффициентов:');
b=input('b=');
disp('Размерность системы:');
[n,m]=size(A)
disp('Ранг матрицы системы:');
r=rank(A)
disp('Ранг расширенной матрицы:');
R=rank([A b])
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('Проверка Ax-b=0:');
           A*x-b
      else
           disp('Система имеет бесконечно много решений.');
      end;
else
       disp('Система не совместна');
end;
%----------------------------------------
%Исследование системы а)
>>>Исследование системы 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:
n =  3
m =  3
Ранг матрицы системы:
r =  3
Ранг расширенной матрицы:
R =  3
Система совместна.
Система имеет единственное решение.
Решение системы методом обратной матрицы:
x =
   2.0000
  -3.0000
  -1.0000
Проверка Ax-b=0:
ans =
   0.0000e+00
   8.8818e-16
   3.5527e-15
%----------------------------------------
%Исследование системы б)
>>>Исследование системы 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:
n =  4
m =  4
Ранг матрицы системы:
r =  3
Ранг расширенной матрицы:
R =  4
Система не совместна
%----------------------------------------
%Исследование системы в)
>>>Исследование системы 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:
n =  3
m =  4
Ранг матрицы системы:
r =  3
Ранг расширенной матрицы:
R =  3
```

```
Система совместна.
Система имеет бесконечное множество решений.
```

Листинг 5.93

5.7 Собственные значения и собственные векторы

Пусть A – матрица размерностью $n \times n$. Любой ненулевой вектор x, принадлежащий некоторому векторному пространству, для которого $A \cdot x = \lambda \cdot x$, где λ некоторое число, называется *собственным вектором матрицы* A, а λ – принадлежащим ему или соответствующим ему *собственным значением матрицы* A [7].

Уравнение $A \cdot x = \lambda \cdot x$ эквивалентно уравнению $(A - \lambda \cdot E) \cdot x = 0$. Это однородная система линейных уравнений, нетривиальные решения которой являются искомыми *собственными векторами*. Она имеет нетривиальные решения только тогда, когда $r(A - \lambda \cdot E) < n$, то есть, если $det(A - \lambda \cdot E) = 0$.

Многочлен $det(A - \lambda \cdot E)$ называется *характеристическим многочленом матрицы* A, а уравнение $det(A - \lambda \cdot E) = 0$ – *характеристическим уравнением матрицы* A. Если λ_i – собственные значения A, то нетривиальные решения однородной системы линейных уравнений $(A - \lambda \cdot E) = 0$ 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 h, определяемые соотношением $A \cdot y = h \cdot B \cdot y, (y \neq 0)$, где B – невырожденная матрица. Векторы y и числа h обязательно являются собственными векторами и собственными значениями матрицы $B^{-1} \cdot A$. Пусть $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$, причем матрица B является положительно определенной, тогда собственные значения h совпадают с корнями уравнения *n*–й степени $det(A - h \cdot B) = det(a_{ij} - h \cdot b_{ij}) = 0$. Это уравнение называют *характеристическим уравнением* для обобщенной задачи о собственных значениях. Для каждого корня h кратности m существует ровно m линейно независимых собственных векторов y.

ЗАДАЧА 5.14. Найти собственные значения и собственные векторы матрицы *A*.

На листинге 5.94 показано решение поставленной задачи.

```
disp('Введите матрицу:');
A=input('A=');
[n,m]=size(A);
disp('Вектор собственных значений матрицы A:');
d=eig(A)
[L, D]=eig(A);
disp('L- Матрица собственных векторов:');
L
disp('D - Диагональная матрица собственных значений:');
D
disp('Проверка:');
for i=1:n
  (A-D(i,i)*eye(n))*L(:,i)
end;
%----------------------------------------
Введите матрицу:
A= [5 2 -1;1 -3 2; 4 5 -3]
```

```
Вектор собственных значений матрицы A:
d =
   4.9083e+00
  -2.1495e-16
  -5.9083e+00
L- Матрица собственных векторов:
L =
  -0.796113  -0.049326   0.181303
  -0.241044   0.542590  -0.598803
  -0.555069   0.838548   0.780106
D - Диагональная матрица собственных значений:
D =
Diagonal Matrix
   4.9083e+00            0            0
            0  -2.1495e-16            0
            0            0  -5.9083e+00
Проверка:
ans =
  -2.3657e-16
  -4.3585e-16
   7.4420e-16
ans =
   2.7756e-17
  -4.1633e-16
   4.4409e-16
ans =
   2.0632e-16
   1.5772e-15
   2.6606e-16
```

Листинг 5.94

ЗАДАЧА 5.15. Привести заданную матрицу к диагональному виду.

Задача состоит в том, чтобы для квадратной матрицы A подобрать такую матрицу C, чтобы матрица $B=C^{-1}\cdot A\cdot C$ 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.

На листинге 5.95 показано, как можно решить поставленную задачу.

```
disp('Введите матрицу:'); A=input('A=');
format bank; [C,D]=eig(A);
disp('Диагональная матрица к матрице A:');
D
disp('Проверка B=D'); B=inv(C)*A*C
%---------------------------------------
Введите матрицу:
A= [2 1 3;1 -2 1;3 2 2]
Диагональная матрица к матрице A:
D =
Diagonal Matrix
  5.41     0     0
     0  -1.00     0
     0     0  -2.41
Проверка B=D
```

```
B =
  5.41  -0.00  -0.00
  0.00  -1.00  0.00
  -0.00  -0.00  -2.41
```

Листинг 5.95

ЗАДАЧА 5.16. Найти решение обобщенной задачи о собственных значениях для матриц *A* и *B*.

Обобщенную задачу о собственных значениях (листинг 5.96) решают при помощи функции `eig(A,B)`, которая в качестве результата выдает матрицу обобщенных собственные векторов и диагональную матрицу, содержащую обобщенные собственные значения.

```
disp('Введите матрицу A:');
A=input('A=');
disp('Введите матрицу B:');
B=input('B=');
[X,V]=eig(A,B);
disp('Матрица обобщенных собственных векторов:');
X
disp('Матрица обобщенных собственных значений:');
V
disp('Обобщенные собственные значения:');
v=diag(V)
%---------------------------------------
Введите матрицу A:
A= [1 -3;-3 4]
Введите матрицу B:
B= [1 2;-3 1]
Матрица обобщенных собственных векторов:
X =
  1.00  0.92
  0.00  1.00
Матрица, содержащая обобщенные собственные значения:
V =
Diagonal Matrix
  1.00     0
     0  -0.71
Обобщенные собственные значения:
v =
  1.00
  -0.71
```

Листинг 5.96

5.8 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 [7]:

- *первая норма* $\|A\|_1$ квадратной матрицы :

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

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

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

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

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

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

- *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 $A\cdot x=b$ 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 5.17. Вычислить нормы и числа обусловленности матрицы A.

В листинге 5.97 приведен 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.

```
disp('Введите матрицу:'); A=input('A=');
[n,m]=size(A);
disp('Первая норма:');
n_1=norm(A,1)
N_1=max(sum(abs(A)))
disp('Вторая норма:');
n_2=norm(A,2)
N_2= sqrt(max(eig(A*A')))
disp('Бесконечная норма:');
n_i= norm(A,inf)
N_i=max(sum(abs(A')))
disp('Евклидова норма:');
n_e= norm(A,'fro')
N_e= sqrt(sum(diag(A*A')))
disp('Число обусловленности в первой норме:');
c_1= cond(A,1)
C_1= norm(A,1)*norm(inv(A),1)
disp('Число обусловленности во второй норме:');
c_2=cond(A,2)
C_2=norm(A,2)*norm(inv(A),2)
disp('Число обусловленности в бесконечной норме:');
```

```
c_i= cond(A,inf)
C_i= norm(A,inf)*norm(inv(A),inf)
disp('Число обусловленности в евклидовой норме:');
c_e= cond(A,'fro')
C_e= norm(A,'fro')*norm(inv(A),'fro')
%---------------------------------------
Введите матрицу:
A= [5 7 6 5;7 10 8 7;6 8 10 9;5 7 9 10]
Первая норма:
n_1 = 33.00
N_1 = 33.00
Вторая норма:
n_2 = 30.29
N_2 = 30.29
Бесконечная норма:
n_i = 33.00
N_i = 33.00
Евклидова норма:
n_e = 30.55
N_e = 30.55
Число обусловленности в первой норме:
c_1 = 4488.00
C_1 = 4488.00
Число обусловленности во второй норме:
c_2 = 2984.09
C_2 = 2984.09
Число обусловленности в бесконечной норме:
c_i = 4488.00
C_i = 4488.00
Число обусловленности в евклидовой норме:
c_e = 3009.58
C_e = 3009.58
```

Листинг 5.97

5.9 Задачи линейной алгебры в символьных вычислениях

Основы работы с символьными переменными в Octave описаны в п. 2.7. Рассмотрим работу с матрицами, заданными в символьных переменных и выражениях.

Для *определения символической матрицы* служит функция

```
ex_matrix(число строк, число столбцов, элементы матрицы)
```

Например,

```
>>> symbols
%Определение символьных переменных
>>> a= sym ("a"); b = sym ("b"); c = sym ("c"); d = sym ("d");
%Матрица строка
>>> Matr=ex_matrix(1,3,a,b,c)
Matr = [[a,b,c]]
%Матрица столбец
>>> Matr=ex_matrix(4,1,a,b,c,d)
Matr = [[a],[b],[c],[d]]
%Матрица 2 на 2
```

```
>>> Matr=ex_matrix(2,2,a,b,c,d)
Matr = [[a,b],[c,d]]
%Матрица 3 на 3
>>> Matr=ex_matrix(3,3,a,0,b,c,1,1,d,0,2)
Matr = [[a,0.0,b],[c,1.0,1.0],[d,0.0,2.0]]
```

Листинг 5.98

Над символьными матрицами определены операции *сложения*, *вычитания*, *умножения*.

ЗАДАЧА 5.18. Выполнить действия над матрицами $(A+B)(A-B)$ (листинг 5.99).

```
>>> symbols
%Определение символьных переменных
>>> a= sym ("a"); b = sym ("b"); c = sym ("c"); d = sym ("d");
%Определение матриц
>>> A=ex_matrix(2,2,a,b,c,d)
A = [[a,b],[c,d]]
>>> B=ex_matrix(2,2,d,b,c,a)
B = [[d,b],[c,a]]
>>> C=A+B
C = [[d+a,2*b],[2*c,d+a]]
>>> D=A-B
D = [[-d+a,0],[0,d-a]]
>>> C*D
ans = [[-(d-a)*(d+a),2*(d-a)*b],[-2*(d-a)*c,(d-a)*(d+a)]]
```

Листинг 5.99

К сожалению над символьными матрицами не определены операции вычисления определителя и обратной матрицы.

Для *решения системы линейных алгебраических уравнений* можно воспользоваться функцией `symlsolve`.

ЗАДАЧА 5.19. Решить систему линейных алгебраических уравнений $\begin{cases} ax+by=c \\ x+y=d \end{cases}$ относительно переменных *x* и *y* (листинг 5.100).

```
>>> x = sym ("x"); y = sym ("y"); a = sym ("a");
>>> b = sym ("b"); c = sym ("c"); d = sym ("d");
>>> sols = symlsolve({a*x+b*y==c,x+y==d},{x,y})
sols =
(  [1] = -(a-b)^(-1)*(d*b-c)
   [2] = -(a-b)^(-1)*(c-d*a)
)
```

Листинг 5.100

ЗАДАЧА 5.20. Решить систему линейных алгебраических уравнений (листинг 5.101)

$$\begin{cases} x_1+2x_2+5x_3=-9 \\ x_1-x_2+3x_3=2 \\ 3x_1-6x_2-x_3=25 \end{cases}.$$

```
>>>sols = symlsolve({x1+2*x2+5*x3==-9,x1-x2+3*x3==2,
                    3*x1-6*x2-x3==25},{x1,x2,x3})
sols =
(  [1] =2.0
   [2] =-3.0
   [3] =-1.0
)
```

Листинг 5.101

6. Векторная алгебра и аналитическая геометрия

Рассмотрим возможности Octave при решении задач векторной алгебры и аналитической геометрии. Благодаря мощным графическим средствам пакета эти задачи становятся более понятными и наглядными.

6.1 Векторная алгебра

В геометрии *вектором* называется всякий направленный отрезок. Учение о действиях над векторами называется *векторной алгеброй* [7].

Вектор, началом которого служит точка **A**, а концом точка **B**, обозначается $\overrightarrow{AB}$ или $\vec{a}$. Если начало и конец вектора совпадают, то отрезок превращается в точку и теряет направление, такой отрезок называют *нуль-вектором*. Если вектор задан точками $A(x_1, y_1)$ и $B(x_2, y_2)$, то его *координаты*: $|\overrightarrow{AB}|=\{(x_2-x_1),(y_2-y_1)\}=\{X,Y\}$. Длина вектора называется также его *модулем* , обозначается $|\overrightarrow{AB}|$ или $|\vec{a}|$ и вычисляется по формуле: $\vec{a}=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$. ормулы $M_x=\frac{x_1+x_2}{2}, M_y=\frac{y_1+y_2}{2}$ служат для вычисления координат *середины отрезка* $\overrightarrow{AB}$. *Разделить отрезок* $\overrightarrow{AB}$ в заданном отношении λ можно так: $L_x=\frac{x_1+\lambda x_2}{1+\lambda}, L_y=\frac{y_1+\lambda y_2}{1+\lambda}$, здесь L_x и L_y - координаты точки **L**, делящей отрезок в отношении $AL:LB=l_1:l_2=\lambda$.

Напомним, что векторы в Octave задаются путем поэлементного ввода:

```
>>>%Вектор-строка
>>> a=[1 0 3]
a =
1 0 3
>>>%Вектор-столбец
>>> b=[0;1;4]
b =
0
1
4
```

Листинг 6.1

ЗАДАЧА 6.1. Построить вектор $|\vec{a}|=\{5,7\}$.

Решение задачи показано на рис. 6.1. Листинге 6.2. содержит команды Octave, с помощью которых был выполнен рисунок.

```
clear all;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
set(gcf,'name','Vector')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');ylabel('y');
a=[5 7];
```

```
L1=line([0, a(1)],[0, a(2)]);
set(L1,'LineWidth',3,'Color','k');
L1_=line([a(1), a(1)],[a(2), a(2)]);
set(L1_,'LineWidth',5,'Color','k');
set(L1_,'marker','<','markersize',16);
```

Листинг 6.2

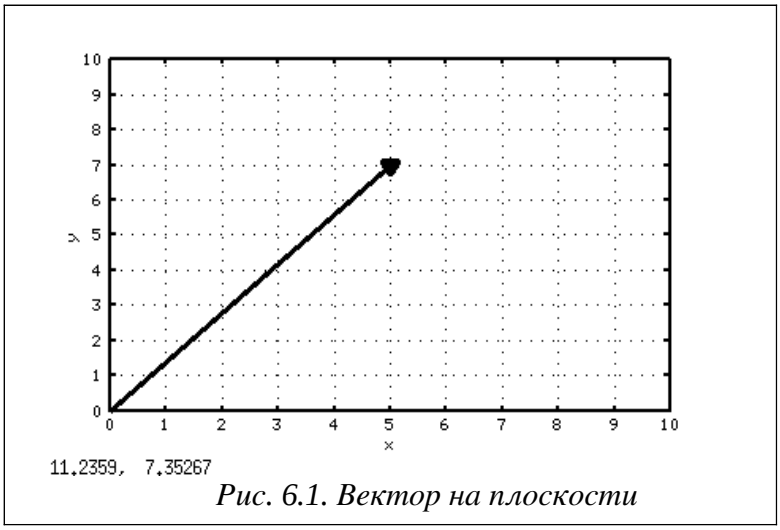


Рис. 6.1. Вектор на плоскости

ЗАДАЧА 6.2. Построить векторы, заданные координатами начала и конца:

$\vec{a}=\{(2,3),(4,6)\}, \vec{b}=\{(9,7),(6,5)\},$

$\vec{c}=\{(1,8),(4,8)\}, \vec{d}=\{(6,7),(6,9)\},$

$\vec{k}=\{(8,4),(8,1)\}, \vec{p}=\{(7,3),(5,3)\}.$

Решение задачи показано в листинге 6.3 и на рис. 6.2. Обратите внимание, что для изображения вектора была создана специальная функция `vector(A,B)` . Эта функция изображает направленный отрезок $|\overrightarrow{AB}|$ в декартовой системе координат и возвращает координаты его середины. В данном случае координаты середины отрезка нужны для нанесения соответствующей надписи, обозначающей вектор на рисунке.

```
clear all;
%Функция рисует направленный отрезок AB,
%в качестве результата выдает координаты
%середины отрезка AB.
function [M]=vector(A,B)
x1=A(1);x2=B(1); y1=A(2);y2=B(2);
%Угол в вершине стрелки в радианах
alf=30*pi/180;
%Деление отрезка в заданном отношении
L=15;
xm=(x1+L*x2)/(1+L); ym=(y1+L*y2)/(1+L);
%Угол наклона прямой AB
k1=(y2-y1)/(x2-x1);
if (k1==Inf)|(k1==-inf)
%Отрезок перпендикулярен оси Ox
%Координаты основания треугольника, образующего стрелку
      x4=xm-0.2; y4=ym; x3=xm+0.2; y3=ym;
elseif k1==0
%Отрезок перпендикулярен  оси Oy
      x4=xm; y4=ym-0.2; x3=xm; y3=ym+0.2;
```

```
else
%Уравнение прямой AB
      k1=(y2-y1)/(x2-x1); m1=y1-x1*(y2-y1)/(x2-x1);
%Уравнение прямой перпендикулярной AB
      k3=-1/k1; m3=1/k1*xm+ym;
%Уравнение прямой, проходящей через точку B
%под углом alf к прямой AB
      k2=(-k1-tan(alf))/(tan(alf)*k1-1); m2=y2-k2*x2;
%Уравнение прямой, проходящей через точку B
%под углом -alf к прямой AB
      k4=(-k1-tan(-alf))/(tan(-alf)*k1-1); m4=y2-k4*x2;
%Координаты основания треугольника, образующего стрелку
      x4=(m3-m2)/(k2-k3); y4=k2*x4+m2;
      x3=(m3-m4)/(k4-k3); y3=k3*x3+m3;
end;
```

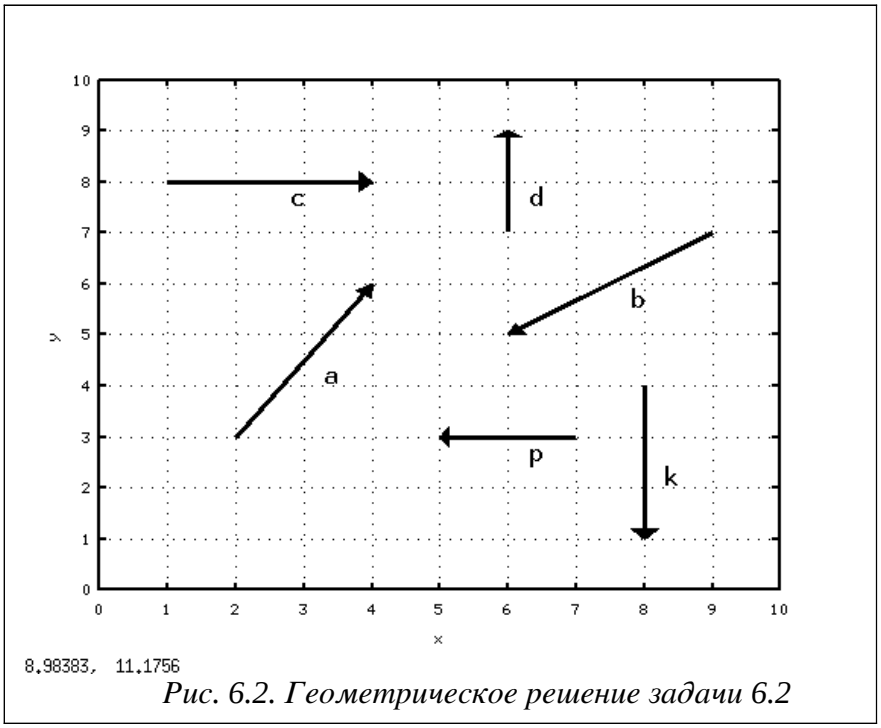


Рис. 6.2. Геометрическое решение задачи 6.2

```
%Изображение прямой AB
line([A(1),B(1)],[A(2),B(2)],'LineWidth',3,'Color','k');
%Изображение стрелки в точке B
patch([x2,x3,x4],[y2,y3,y4],'k');
%Координаты середины отрезка AB
M(1)=(x1+x2)/2; M(2)=(y1+y2)/2;
end;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off');
set(gcf,'name','Vector');
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]); set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]); set(gca,'ytick',[0:10]);
grid on;
```

```
xlabel('x');
ylabel('y');
%Построение вектора  a
ma=vector([2,3],[4,6]);
T=text(ma(1)+0.3,ma(2)-0.3,'a');
set(T,'FontSize',20)
%Построение вектора  b
mb=vector([9,7],[6,5]);
T=text(mb(1)+0.3,mb(2)-0.3,'b');
set(T,'FontSize',20)
%Построение вектора  c
mc=vector([1,8],[4,8]);
T=text(mc(1)+0.3,mc(2)-0.3,'c');
set(T,'FontSize',20)
%Построение вектора  d
md=vector([6,7],[6,9]);
T=text(md(1)+0.3,md(2)-0.3,'d');
set(T,'FontSize',20)
%Построение вектора  k
mk=vector([8,4],[8,1]);
T=text(mk(1)+0.3,mk(2)-0.3,'k');
set(T,'FontSize',20)
%Построение вектора  p
mp=vector([7,3],[5,3]);
T=text(mp(1)+0.3,mp(2)-0.3,'p');
set(T,'FontSize',20)
```

Листинг 6.3

Два ненулевых вектора $\vec{a}$ и $\vec{b}$ *равны*, если они равнонаправлены и имеют один и тот же модуль. Все нулевые векторы равны. Во всех остальных случаях векторы *не равны*. Два вектора имеющие равные модули и противоположные направления,называются *противоположными*. Векторы лежащие на параллельных прямых называются *коллинеарными* [7].

Задача 6.3. Сравнить векторы $\overrightarrow{AB}$ и $\overrightarrow{CD}$, $\overrightarrow{ON}$, $\overrightarrow{OM}$ и $\overrightarrow{KL}$, $\overrightarrow{PR}$ и $\overrightarrow{UV}$ заданные координатами начала и конца:

$A(1,2), B(3,5), C(3,2), D(5,5),$ $O(7,9), M(6,6), N(8,6), K(4,9), L(3,6),$
$P(9,2), R(6,2), U(6,3), V(9,3).$

Текст файла-сценария представлен в листинге 6.4. При решении задачи была создана функция `dlina(X,Y)`, которая вычисляет длину отрезка **XY**, заданного координатами точек $X(x_1, x_2)$ и $Y(y_1, y_2)$. Для изображения векторов использовалась функция `vector(A,B)`, описанная в задаче 6.2.

Решение задачи показано в листинге 6.5 и на рис. 6.3. По полученным числовым результатам и геометрической интерпретации задачи можно сделать вывод, что векторы $\overrightarrow{AB}$ и $\overrightarrow{CD}$ равны. Векторы $\overrightarrow{ON}$ и $\overrightarrow{OM}$ не равны, хотя у них и одинаковые длины, но направления различны. Векторы $\overrightarrow{ON}$ и $\overrightarrow{KL}$ неравны по той же причине, а векторы $\overrightarrow{OM}$ и $\overrightarrow{KL}$ равны. Векторы $\overrightarrow{PR}$ и $\overrightarrow{UV}$ - противоположные, так как имеют одинаковый модуль и противоположные направления.

```
%Функция возвращает длину отрезка XY
function d=dlina(X,Y)
d=sqrt((Y(1)-X(1))^2+(Y(2)-X(2))^2);
end;
```

```
clf;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
set(gcf,'name','Vector')
cla;
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');
ylabel('y');
%Исходные данные
A=[1,2];B=[3,5];
C=[3,2];D=[5,5];
O=[7,9];M=[6,6];N=[8,6];
K=[4,9];L=[3,6];
P=[9,2];R=[6,2];
U=[6,3];V=[9,3];
%Длины отрезков
dAB=dlina(A,B)
dCD=dlina(C,D)
dON=dlina(O,N)
dOM=dlina(O,M)
dKL=dlina(K,L)
dPR=dlina(P,R)
dUV=dlina(U,V)
%Построение вектора  AB
vector(A,B);
A_=text(A(1)+0.3,A(2)-0.3,'A');
B_=text(B(1)+0.3,B(2)-0.3,'B');
set(A_,'FontSize',20)
set(B_,'FontSize',20)
%Построение вектора  CD
vector(C,D);
C_=text(C(1)+0.3,C(2)-0.3,'C');
D_=text(D(1)+0.3,D(2)-0.3,'D');
set(C_,'FontSize',20)
set(D_,'FontSize',20)
%Построение вектора  OM
vector(O,M);
O_=text(O(1)+0.3,O(2)-0.3,'O');
M_=text(M(1)+0.3,M(2)-0.3,'M');
set(O_,'FontSize',20)
set(M_,'FontSize',20)
%Построение вектора  ON
vector(O,N);
%O_=text(O(1)+0.3,O(2)-0.3,'O');
N_=text(N(1)+0.3,N(2)-0.3,'N');
%set(O_,'FontSize',20)
```

```
set(N_,'FontSize',20)
%Построение вектора  KL
vector(K,L);
K_=text(K(1)+0.3,K(2)-0.3,'K');
L_=text(L(1)+0.3,L(2)-0.3,'L');
set(K_,'FontSize',20)
set(L_,'FontSize',20)
%Построение вектора  PR
vector(P,R);
P_=text(P(1)+0.3,P(2)-0.3,'P');
R_=text(R(1)+0.3,R(2)-0.3,'R');
set(P_,'FontSize',20)
set(R_,'FontSize',20)
%Построение вектора  UV
vector(U,V);
U_=text(U(1)+0.3,U(2)-0.3,'U');
V_=text(V(1)+0.3,V(2)-0.3,'V');
set(U_,'FontSize',20)
set(V_,'FontSize',20)
```

Листинг 6.4

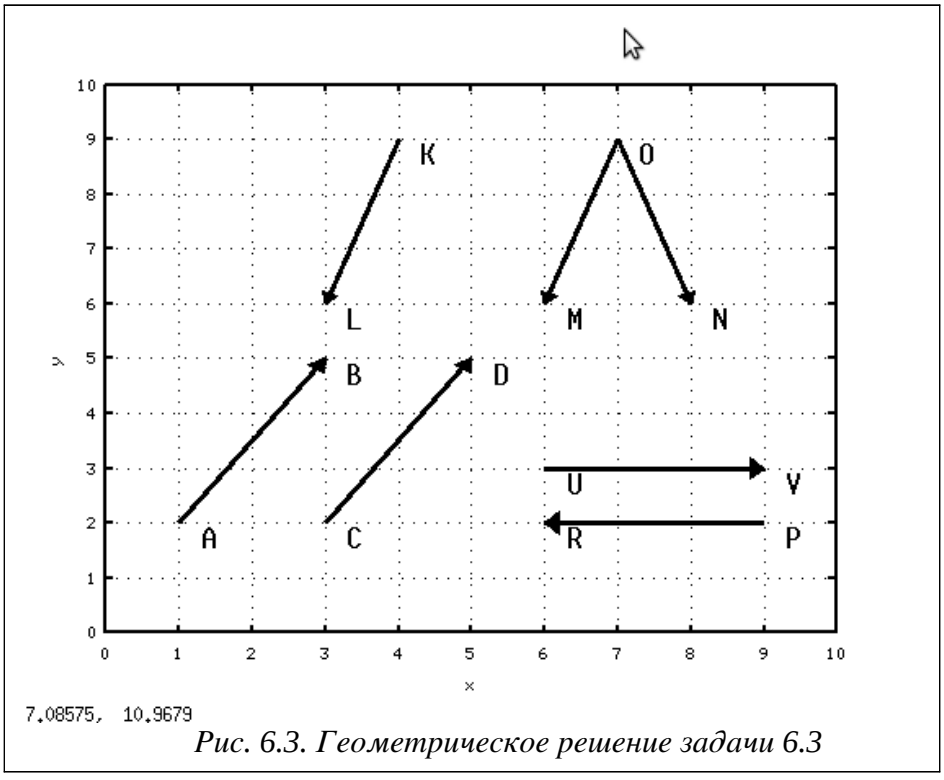


Рис. 6.3. Геометрическое решение задачи 6.3

```
>>>%Длины отрезков:
>>>AB =  3.6056
>>>CD =  3.6056
>>>ON =  3.1623
>>>OM =  3.1623
>>>KL =  3.1623
>>>PR =  3
>>>UV =  3
```

Листинг 6.5

ЗАДАЧА 6.4. Проверить коллинеарны ли векторы $\overrightarrow{AB}$ и $\overrightarrow{CD}$, $\overrightarrow{NM}$ и $\overrightarrow{KL}$, $\overrightarrow{PR}$ и $\overrightarrow{UV}$. Координаты точек:

$A(4,2), B(2,3), C(3,2), D(7,0), \quad M(2,1), N(6,1), K(1,2), L(7,2),$
$P(3,3), R(5,3), U(1,4), V(5,4).$

Текст файла-сценария представлен в листинге . При решении задачи была создана функция `kollin(a,b)`, которая определяет коллинеарны ли векторы $\vec{a}=\{x_1, y_1\}$ и $\vec{b}=\{x_2, y_2\}$. Результатом работы функции является *коэффициент пропорциональности векторов*. Если векторы коллинеарны, то их соответствующие координаты пропорциональны: $\pm\lambda=\frac{x_2}{x_1}=\frac{y_2}{y_1}$. Знак коэффициента пропорциональности говорит о направлении векторов: «+» - в одну сторону, «-» - в разные. Для изображения векторов использовалась функция `vector(A,B)`, описанная в задаче 6.2.

Решение задачи показано в листинге 6.6 и на рис. 6.4. По полученным числовым результатам и геометрической интерпретации задачи можно сделать вывод, что векторы $\overrightarrow{AB}$ и $\overrightarrow{CD}$ коллинеарны, но направлены в разные стороны. Векторы $\overrightarrow{NM}$ и $\overrightarrow{KL}$ не являются коллинеарными. Векторы $\overrightarrow{PR}$ и $\overrightarrow{UV}$ - коллинеарны и имеют одно направление.

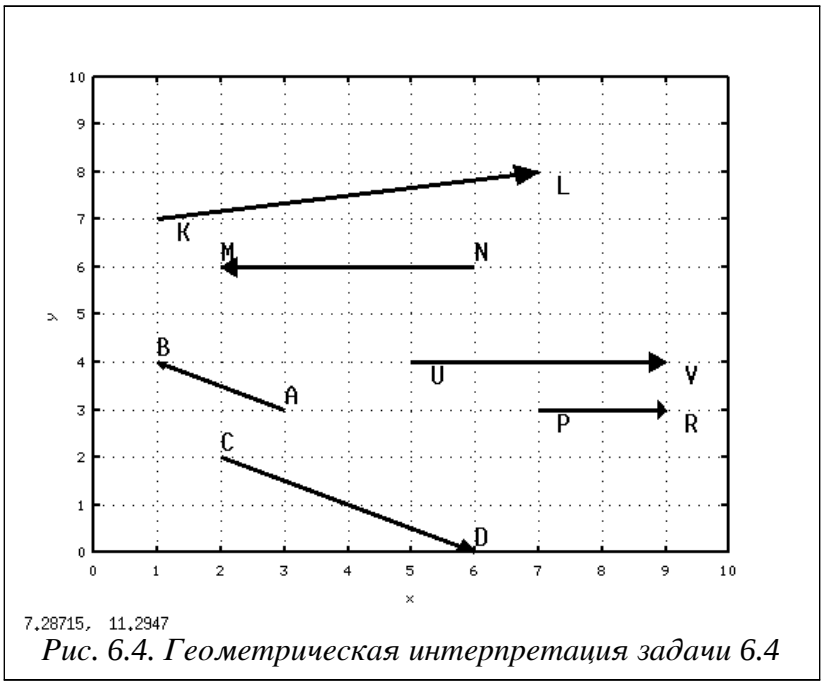


Рис. 6.4. Геометрическая интерпретация задачи 6.4

```
function [lam]=kollin(a,b)
if (a(2)==0) & (b(2)==0)
      lam=a(1)/b(1);
elseif (a(1)==0) & (b(1)==0)
      lam=a(2)/b(2);
elseif a(1)/b(1)==a(2)/b(2)
      lam=a(1)/b(1);
else
      lam=Inf;
end;
end;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
```

```
set(gcf,'name','Vector')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');
ylabel('y');
A=[3,3];B=[1,4];
C=[2,2];D=[6,0];
M=[2,6];N=[6,6];
K=[1,7];L=[7,8];
P=[7,3];R=[9,3];
U=[5,4];V=[9,4];
AB_CD=kollin(B-A,D-C)
MN_KL=kollin(N-M,L-K)
PR_UV=kollin(R-P,V-U)
%Построение вектора AB
vector(A,B);
A_=text(A(1),A(2)+0.3,'A');
B_=text(B(1),B(2)+0.3,'B');
set(A_,'FontSize',20)
set(B_,'FontSize',20)
%Построение вектора  CD
vector(C,D);
C_=text(C(1),C(2)+0.3,'C');
D_=text(D(1),D(2)+0.3,'D');
set(C_,'FontSize',20)
set(D_,'FontSize',20)
%Построение вектора  NM
vector(N,M);
N_=text(N(1),N(2)+0.3,'N');
M_=text(M(1),M(2)+0.3,'M');
set(N_,'FontSize',20)
set(M_,'FontSize',20)
%Построение вектора  KL
vector(K,L);
K_=text(K(1)+0.3,K(2)-0.3,'K');
L_=text(L(1)+0.3,L(2)-0.3,'L');
set(K_,'FontSize',20)
set(L_,'FontSize',20)
%Построение вектора  PR
vector(P,R);
P_=text(P(1)+0.3,P(2)-0.3,'P');
R_=text(R(1)+0.3,R(2)-0.3,'R');
set(P_,'FontSize',20)
set(R_,'FontSize',20)
%Построение вектора  UV
vector(U,V);
U_=text(U(1)+0.3,U(2)-0.3,'U');
```

```
V_=text(V(1)+0.3,V(2)-0.3,'V');
set(U_,'FontSize',20)
set(V_,'FontSize',20)
%Результаты работы программы
%Коэффициенты пропорциональности векторов
>>>AB_CD = -0.50000
>>>MN_KL = Inf
>>>PR_UV =  0.50000
```

Листинг 6.6

Геометрической проекций вектора $\overrightarrow{AB}$ на ось **OX** называется вектор $\overrightarrow{A'B'}$, начало которого A' есть проекция точки A на ось **OX**, а конец B' - проекция точки B на ту же ось. Обозначается: $\overrightarrow{A'B'} = Пр_{OX}\overrightarrow{AB}$. *Алгебраической проекций вектора* $\overrightarrow{AB}$ на ось **OX** называется длина вектора $\overrightarrow{A'B'}$, взятая со знаком «+», если направление вектора $\overrightarrow{AB}$ совпадает с направлением оси **OX**, или со знаком «-» в противном случае [7].

Задача 6.5. Найти проекции векторов $\overrightarrow{AB}$ и $\overrightarrow{PR}$ на ось **OX.** Координаты точек, задающих векторы: $A(1,2), B(3,5), \quad P(9,2), R(6,2)$

Текст файла-сценария представлен в листинге 6.7. При решении задачи была создана функция `pr_OX(X)`, которая вычисляет длину проекции вектора на ось **OX.** Аргументом функции является массив абсцисс $X(x_1, x_2)$ заданного вектора. Для изображения векторов и их проекций использовалась функция `vector(A,B)`, описанная в задаче 6.2. Значения алгебраических проекции векторов представлены в листинге 6.8. Геометрические проекции показаны на рис. 6.5.

```
%Длина проекции вектора на ось OX
function pr=pr_OX(X)
pr=X(2)-X(1);
end;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
set(gcf,'name','Vector')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');ylabel('y');
A=[1,2];B=[3,5];
P=[9,2];R=[6,2];
%Длины проекций
prAB=pr_OX([A(1),B(1)])
prPR=pr_OX([P(1),R(1)])
%Построение вектора  AB
vector(A,B);
A_=text(A(1)+0.3,A(2)-0.3,'A');
B_=text(B(1)+0.3,B(2)-0.3,'B');
set(A_,'FontSize',20)
set(B_,'FontSize',20)
%Построение проекции вектора AB на ось OX
```

```
mAB=vector([A(1),0],[B(1),0]);
text(mAB(1),mAB(2)+0.5,'prAB','FontSize',18);
%Построение вектора PR
vector(P,R);
P_=text(P(1)+0.3,P(2)-0.3,'P');
R_=text(R(1)+0.3,R(2)-0.3,'R');
set(P_,'FontSize',20)
set(R_,'FontSize',20)
%Построение проекции вектора PR на ось OX
mPR=vector([P(1),0],[R(1),0]);
text(mPR(1),mPR(2)+0.3,'prPR','FontSize',18);
```

Листинг 6.7

```
>>>prAB =  2
>>>prPR = -3
```

Листинг 6.8

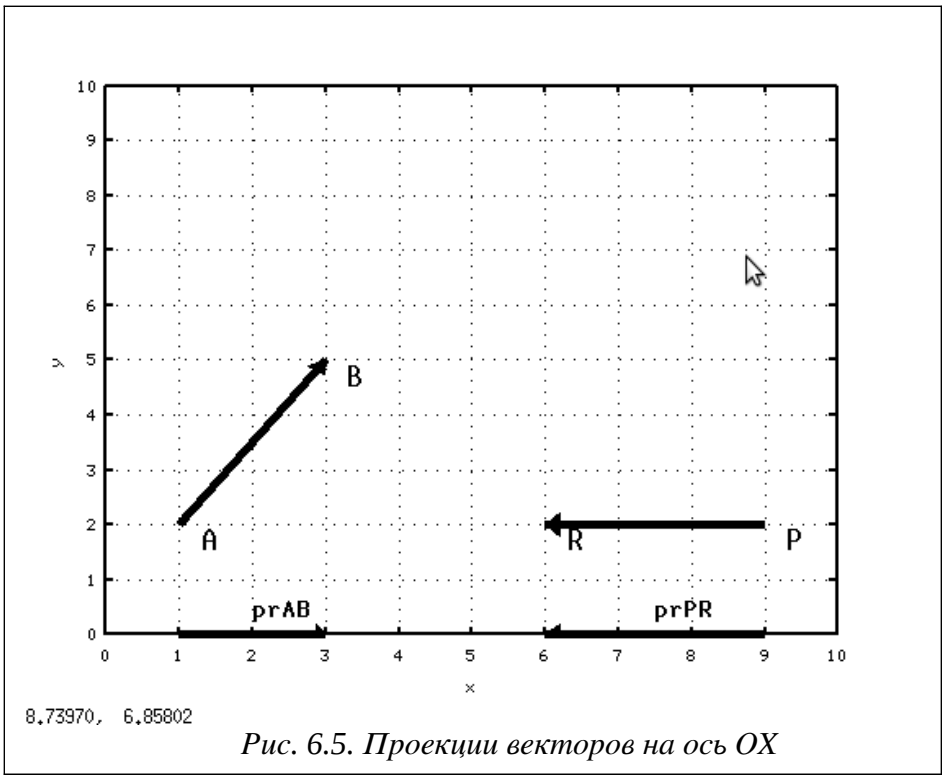


Рис. 6.5. Проекции векторов на ось OX

Всякие векторы можно *привести к общему началу*, то есть построить векторы равные данным и имеющие общее начало в некоторой точке **O**. Над векторами производят различные *действия:* сложение, вычитание, умножение. При *сложении* (*вычитании*) векторов их координаты складываются (вычитаются): $\vec{a} \pm \vec{b} = \{(a_1 \pm b_1), (a_2 \pm b_2)\}$. При *умножении* (*делении*) вектора на число все координаты умножаются (делятся) на это число [7]: $\lambda \vec{a} = \{\lambda a_1, \lambda a_2\}$, $\frac{\vec{a}}{\lambda} = \{\frac{a_1}{\lambda}, \frac{a_2}{\lambda}\}$.

ЗАДАЧА 6.6. Найти сумму векторов $|\vec{a}| = \{1,4\}$ и $|\vec{b}| = \{5,3\}$.

Если векторы $\vec{a}$ и $\vec{b}$ не коллинеарны, то геометрически вектор $\vec{c} = \vec{a} + \vec{b}$ является диагональю параллелограмма построенного на векторах $\vec{a}$ и $\vec{b}$ (*правило параллелограмма*).

Листинг 6.9. содержит команды Octave, с помощью которых была решена задача и результаты ее работы. Функция `vector(A,B)`, описана в задаче 6.2. Геометрическое

решение задачи показано на рис. 6.6.

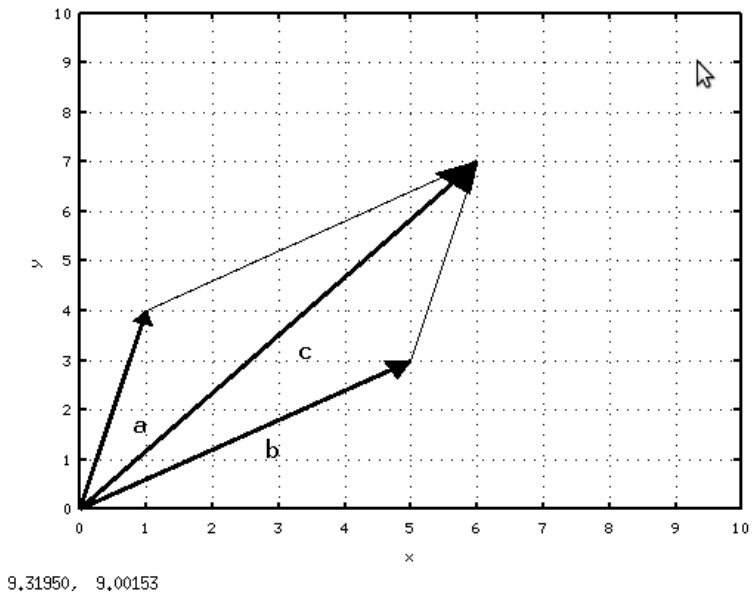


Рис. 6.6. Геометрическое представление суммы векторов

```
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
set(gcf,'name','Vector c=a+b')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');ylabel('y');
a=[1,4];
b=[5,3];
%Сумма векторов
c=[a(1)+b(1),a(2)+b(2)]
%Построение вектора  a
ma=vector([0,0],a);
text(ma(1)+0.3,ma(2)-0.3,'a','FontSize',20);
%Построение вектора  b
mb=vector([0,0],b);
text(mb(1)+0.3,mb(2)-0.3,'b','FontSize',20);
%Построение вектора  c=a+b
mc=vector([0,0],c);
text(mc(1)+0.3,mc(2)-0.3,'c','FontSize',20);
line([a(1),c(1)],[a(2),c(2)],'LineWidth',1,'Color','k');
line([b(1),c(1)],[b(2),c(2)],'LineWidth',1,'Color','k');
%Результаты работы программы. Координаты вектора c=a+b:
>>c =   6   7
```

Листинг 6.9

ЗАДАЧА 6.7. Выполнить действия над векторами $\vec{c}=3\vec{a}-2\vec{b}$, где $|\vec{a}|=\{1,2\}$ и $|\vec{b}|=\{3,1\}$. Геометрически вычесть из вектора $\vec{a}$ вектор $\vec{b}$, значит найти такой вектор $\vec{x}$ для которого $\vec{x}+\vec{b}=\vec{a}$. Иначе говоря, если на векторах $\vec{x}$ и $\vec{b}$ построить треугольник, то $\vec{a}$ - его третья сторона (*правило треугольника*) [7].

Листинг 6.10. содержит команды Octave и результаты работы файла-сценария. Функция `vector(A,B),` описана в задаче 6.2. Решение задачи показано на рис. 6.7.

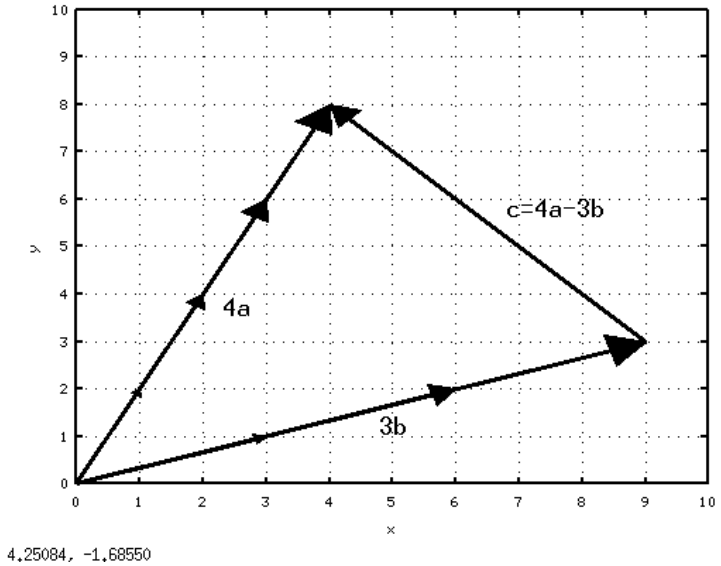


Рис. 6.7. Геометрическое представление действий над векторами

```
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
set(gcf,'name','Vector c=4a-3b')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');ylabel('y');
a=[1,2];b=[3,1];
%Действия над векторами
c=[4*a(1)-3*b(1),4*a(2)-3*b(2)]
%Построение вектора 4a
for i=1:4
ma=vector([0,0],i*a);
end;
text(ma(1)+0.3,ma(2)-0.3,'4a','FontSize',20);
%Построение вектора 3 b
for i=1:3
mb=vector([0,0],i*b);
end;
text(mb(1)+0.3,mb(2)-0.3,'3b','FontSize',20);
```

```
%Построение вектора  c=4a-3b
mc=vector([3*b(1),3*b(2)],[4*a(1),4*a(2)]);
text(mc(1)+0.3,mc(2)+0.3,'c=4a-3b','FontSize',20);
%Результаты работы программы
%Координаты отрезка c=4a-3b
>>>c =  -5   5
```

Листинг 6.10

ЗАДАЧА 6.8. Найти углы образуемые осями координат с вектором $|\vec{a}|=\{2,-2,-1\}$. Углы, образуемые положительными направлениями осей с вектором $|\vec{a}|$, можно рассчитать по формулам: $\cos(\alpha)=\frac{a_1}{|a|}$, $\cos(\beta)=\frac{a_2}{|a|}$, $\cos(\gamma)=\frac{a_3}{|a|}$ [7].

Листинг 6.11 содержит команды Octave и результаты работы файла-сценария.

```
%Углы, образуемые осями координат с вектором X
function [U]=ugol(X)
m=sqrt(X(1)^2+X(2)^2+X(3)^2);
U=acos(X/m);
end;
%Перевод радиан в градусы и минуты
function gr=rad_gr(rad)
gr(1)=round(rad*180/pi); %Градусы
gr(2)=round((rad*180/pi-gr(1))*60); %Минуты
end;
%Вычисление углов в радианах
u=ugol([2,-2,-1])
%Углы в градусах и минутах
alf=rad_gr(u(1))
bet=rad_gr(u(2))
gam=rad_gr(u(3))
%Результаты работы
%Углы в радианах
>>>u =
   0.84107   2.30052   1.91063
%Углы в градусах и минутах
>>>alf =
   48   11
>>>bet =
   132   -11
>>>gam =
   109    28
```

Листинг 6.11

Скалярным произведением вектора $\vec{a}$ на вектор $\vec{b}$ называется произведение их модулей на косинус угла между ними: $\vec{ab}=|\vec{a}|\cdot|\vec{b}|\cos(\widehat{a,b})$. Если $\vec{a}=\{a_1,a_2,a_3\}$ и $\vec{b}=\{b_1,b_2,b_3\}$, то $\vec{ab}=a_1b_1+a_2b_2+a_3b_3$ [7].

ЗАДАЧА 6.9. Найти *угол между векторами* $\vec{a}=\{-2,1,2\}$ и $\vec{b}=\{-2,-2,1\}$:

$$\cos(\widehat{a,b})=\frac{\vec{ab}}{|\vec{a}|\cdot|\vec{b}|}.$$

Текст файла-сценария и результат его работы представлены в листинге 6.12.

```
%Перевод радиан в градусы и минуты
function gr=rad_gr(rad)
gr(1)=round(rad*180/pi); %Градусы
```

```
gr(2)=round((rad*180/pi-gr(1))*60); %Минуты
end;
a=[-2,1,2];b=[-2,-2,1];
da=sqrt(a(1)^2+a(2)^2+a(3)^2);
db=sqrt(b(1)^2+b(2)^2+b(3)^2);
ab=sum(a.*b);
alf=acos(ab/(da*db))
rad_gr(alf)
>>>%Результат
>>>%Угол в радианах
>>>alf =  1.1102
>>>%Угол в градусах
>>>ans =
   64  -23
```

Листинг 6.12

ЗАДАЧА 6.10. Проверить являются ли векторы $\overrightarrow{NM}$ и $\overrightarrow{KL}$, $\overrightarrow{PR}$ и $\overrightarrow{UV}$ взаимно перпендикулярными. Координаты точек:

$M(2,1), N(6,1), K(1,2), L(7,2), \quad P(3,3), R(5,3), U(1,4), V(5,4).$

Если $\vec{a}=\{a_1, a_2\}$ и $\vec{b}=\{b_1, b_2\}$ перпендикулярны, то их скалярное произведение равно нулю: $\vec{ab}=a_1b_1+a_2b_2=0$. На рис. 6.8 Видно, что векторы $\overrightarrow{NM}$ и $\overrightarrow{KL}$ перпендикулярны, а $\overrightarrow{PR}$ и $\overrightarrow{UV}$ нет. Аналитические вычисления подтверждают это (листинг 6.13).

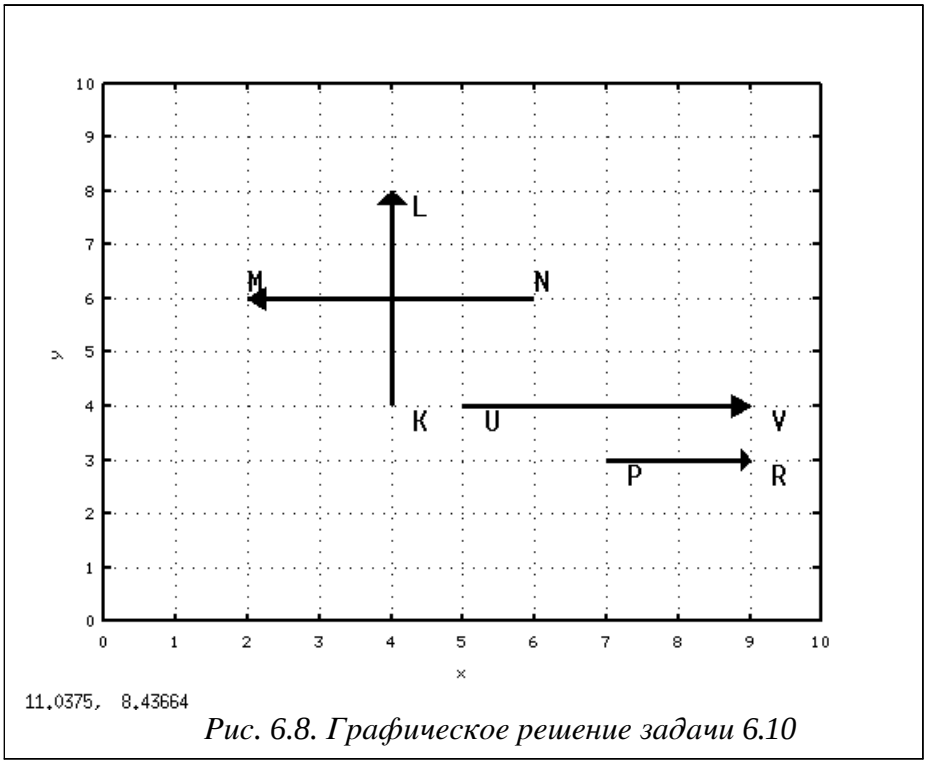


Рис. 6.8. Графическое решение задачи 6.10

```
%Вычисление скалярного произведения векторов a и b
function [ab]=scal(a,b)
ab=a(1)*b(1)+a(2)*b(2);
end;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
```

```
set(gcf,'name','Vector')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
grid on;
xlabel('x');ylabel('y');
M=[2,6];N=[6,6];
K=[4,4]; L=[4,8];
P=[7,3];R=[9,3];
U=[5,4];V=[9,4];
MN_KL=scal(N-M,L-K)
PR_UV=scal(R-P,V-U)
%Построение вектора NM
vector(N,M);
N_=text(N(1),N(2)+0.3,'N');
M_=text(M(1),M(2)+0.3,'M');
set(N_,'FontSize',20)
set(M_,'FontSize',20)
%Построение вектора KL
vector(K,L);
K_=text(K(1)+0.3,K(2)-0.3,'K');
L_=text(L(1)+0.3,L(2)-0.3,'L');
set(K_,'FontSize',20)
set(L_,'FontSize',20)
%Построение вектора PR
vector(P,R);
P_=text(P(1)+0.3,P(2)-0.3,'P');
R_=text(R(1)+0.3,R(2)-0.3,'R');
set(P_,'FontSize',20)
set(R_,'FontSize',20)
%Построение вектора UV
vector(U,V);
U_=text(U(1)+0.3,U(2)-0.3,'U');
V_=text(V(1)+0.3,V(2)-0.3,'V');
set(U_,'FontSize',20)
set(V_,'FontSize',20)
%Результат работы программы
>>>MN_KL = 0
>>>PR_UV =  8
```

Листинг 6.13

Векторным произведением вектора $\vec{a}$ на не коллинеарный с ним вектор $\vec{b}$ называется вектор $\vec{c}$, модуль которого численно равен площади параллелограмма построенного на векторах $\vec{a}$ и $\vec{b}$: $\vec{a}\times\vec{b}=|\vec{a}|\cdot|\vec{b}|\sin(\widehat{a,b})$ [7]. Направление вектора $\vec{c}$ перпендикулярно плоскости параллелограмма. Если векторы $\vec{a}=\{x_1, y_1, z_1\}$ и $\vec{b}=\{x_1, y_2, z_2\}$, то

$$\vec{a}\times\vec{b}=\left\{\begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix}, \begin{vmatrix} z_1 & x_1 \\ z_2 & x_2 \end{vmatrix}, \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}\right\} .$$

ЗАДАЧА 6.11. Найти векторное произведение векторов $|\vec{a}|=\{1,4\}$ и $|\vec{b}|=\{5,3\}$. Вычислить угол между векторами.

Графическое решение показано на рис. 6.9. Листинг содержит текст программы и результаты ее работы.

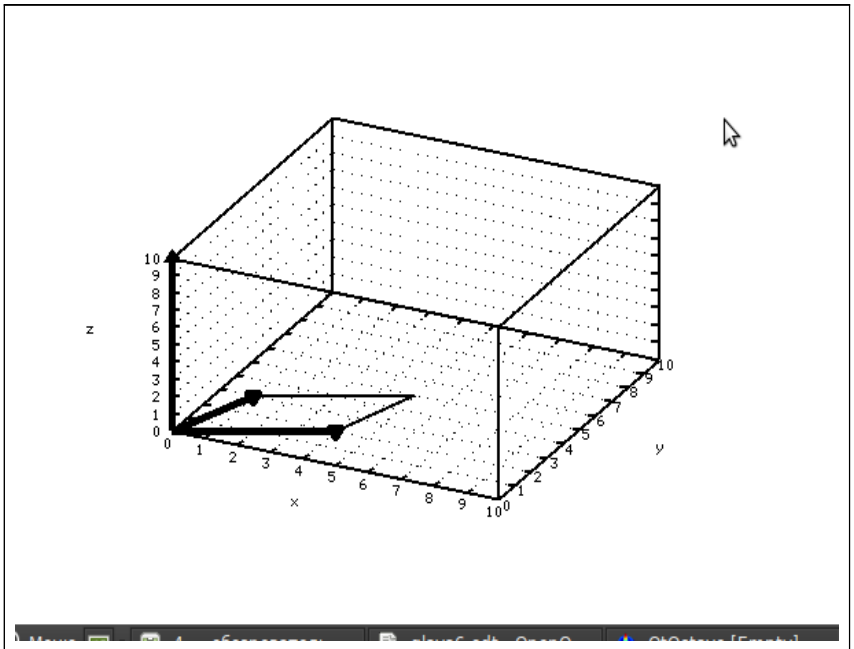


Рис. 6.9. Геометрическая интерпретация векторного произведения

```
clear all;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gcf,'numbertitle','off')
set(gcf,'name','Vector')
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,10]);
set(gca,'ylim',[0,10]);
set(gca,'zlim',[0,10]);
set(gca,'xtick',[0:10]);
set(gca,'ytick',[0:10]);
set(gca,'ztick',[0:10]);
set(gca,'View',[30 30],'box','on');
xlabel('x');ylabel('y');zlabel('z');
axis([0,10,0,10,0,10])
grid on;
%Длина вектора
function D=dlin(x)
D=(x(1)^2+x(2)^2+x(3)^2)^(1/2);
end;
%Перевод радиан в градусы и минуты
function gr=rad_gr(rad)
gr(1)=round(rad*180/pi); %Градусы
gr(2)=round((rad*180/pi-gr(1))*60); %Минуты
end;
%Исходные данные
```

```
x1=4;y1=2;z1=0;x2=1;y2=3;z2=0;
a=[x1,y1,z1];b=[x2,y2,z2];
%Расчет координат векторного произведения
M=[a;b];
M1=M(1:2,2:3);M2=[M(:,1),M(:,3)];M3=M(1:2,1:2);
c(1)=det(M1);
c(2)=-det(M2);
c(3)=(det(M3));
%Расчет угла между векторами a  и b
da=dlin(a);
db=dlin(b);
dc=dlin(c);
alf=asin(dc/(da*db));
%Изображение векторов a,b,c
line([0,a(1)],[0,a(2)],[0,a(3)],
                            'LineWidth',5,'Color','k');
line([0,b(1)],[0,b(2)],[0,b(3)],
                            'LineWidth',5,'Color','k');
line([0,c(1)],[0,c(2)],[0,c(3)],
                            'LineWidth',5,'Color','k');
%Изображение стрелок на векторах
line([c(1),c(1)],[c(2),c(2)],[c(3),c(3)],
                     'LineWidth',5,'Color','k',
                          'marker','^','markersize',16);
line([b(1),b(1)],[b(2),b(2)],[b(3),b(3)],
                     'LineWidth',5,'Color','k',
                          'marker','<','markersize',10);
line([a(1),a(1)],[a(2),a(2)],[a(3),a(3)],
                     'LineWidth',5,'Color','k',
                          'marker','<','markersize',10);
%Расчет координат вершины параллелограмма
k1=y1/x1;k2=y2/x2;
d(1)=(y1-y2+k1*x2-k2*x1)/(k1-k2);
d(2)=k1*d(1)+y2-k1*x2;
d(3)=0;
%Стороны параллелограмма
line([a(1),d(1)],[a(2),d(2)],[a(3),d(3)],
                               'LineWidth',2,'Color','k');
line([b(1),d(1)],[b(2),d(2)],[b(3),d(3)],
                               'LineWidth',2,'Color','k');
%Координаты векторного произведения
c
%Угол между векторами a  и b
alfa=rad_gr(alf)
%Результаты работы программы
%Координаты векторного произведения
>>>c =
    0   -0   10
%Угол между векторами
>>>alfa =   45   -0
```

Листинг 6.14

Три вектора называют *компланарными*, если они, будучи приведены к общему началу, лежат в одной плоскости. *Смешанным* или *векторно-скалярным произведением* трех векторов $\vec{a}$, $\vec{b}$ и $\vec{c}$ называется скалярное произведение вектора $\vec{a}$ на векторное произведение $\vec{b}\times\vec{c}$, то есть число $\overrightarrow{abc}=\vec{a}\cdot(\vec{b}\times\vec{c})=(\vec{b}\times\vec{c})\cdot\vec{a}$. Если векторы $\vec{a}=\{x_1, y_1, z_1\}$, $\vec{b}=\{x_1, y_2, z_2\}$ и $\vec{c}=\{x_3, y_3, z_3\}$ даны своими координатами, то смешанное произведение вычисляют по формуле [7]:

$$\overrightarrow{abc}=\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}.$$

Необходимым и достаточным *условием компланарности векторов* $\vec{a}$, $\vec{b}$ и $\vec{c}$ является равенство нулю их смешанного произведение $\overrightarrow{abc}=0$.

ЗАДАЧА 6.12. Поверить компланарность векторов (листинг 6.15):

а) $\vec{a}=\{-2,-1,-3\}$, $\vec{b}=\{-1,4,6\}$ и $\vec{c}=\{1,5,9\}$;

б) $\vec{a}=\{1,2,3\}$, $\vec{b}=\{-1,3,4\}$ и $\vec{c}=\{2,5,2\}$

```
function d=komp(A,B,C)
M=[A;B;C];
d=det(M);
if d==0
 disp('Векторы компланарны');
else
 disp('Векторы не компланарны');
end;
end;
a=[-2,-1,-3];b=[-1,4,6];c=[1,5,9];
d1=komp(a,b,c)
a=[1,2,3];b=[-1,3,4];c=[2,5,2];
d2=komp(a,b,c)
%Результат работы программы
>>>Векторы компланарны
d1 = 0
>>>Векторы не компланарны
d2 = -27
```

Листинг 6.15

6.2 Аналитическая геометрия

Плоскость, проходящая через точку $M_0(x_0, y_0, z_0)$ и перпендикулярная к вектору $\vec{N}\{A, B, C\}$ представляется *уравнением*

$$A(x-x_0)+B(y-y_0)+C(z-z_0)=0 \quad \text{или} \quad Ax+By+Cz+D=0.$$

Вектор $\vec{N}\{A, B, C\}$ называется *нормальным вектором* плоскости [7].

ЗАДАЧА 6.13. Записать уравнение и построить плоскость, проходящую через точку $M_0(2,0,4)$ и перпендикулярную вектору $\vec{N}\{3,0,-3\}$.

Исходя из условия задачи уравнение плоскости имеет вид

$$3(x-2)+0(y-1)-3(z-4)=0\Rightarrow 3x-6-3z+12=0\Rightarrow 3x-3z+6=0.$$

Для построения плоскости средствами Octave преобразуем уравнение плоскости к виду функции двух переменных:

$$Ax+By+Cz+D=0\Rightarrow z(x, y)=-\frac{A}{C}x-\frac{B}{C}y-\frac{D}{C},$$

$$z(x,y)=b_0+b_1x+b_2y, b_0=-\frac{A}{C}, b_1=\frac{-B}{C}, b_2=\frac{-D}{C}.$$

Решение задачи представлено в листинге 6.16 и на рис. 6.10

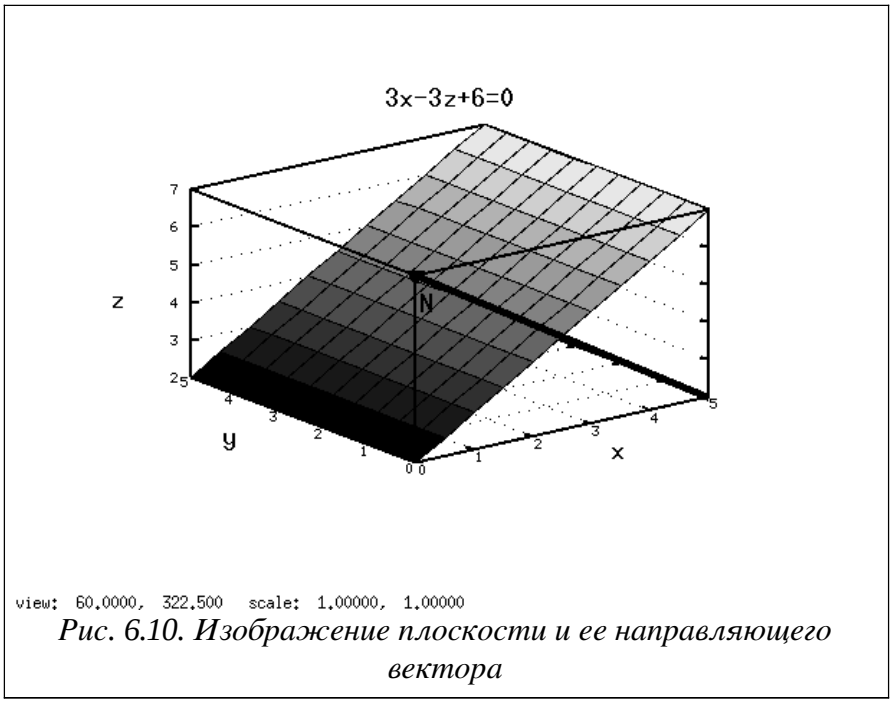


Рис. 6.10. Изображение плоскости и ее направляющего вектора

```
clf;cla;
set(gcf,'Position',[20,20,400,400]);
axis([0,10,0,10,0,10])
%Параметры плоскости
A=3;B=0;C=-3;D=6;N=[A,B,C];
%Параметры уравнения плоскости,
%преобразованного к функции двух переменных
b0=-D/C;b1=-A/C;b2=-B/C;
%Построение плоскости
xk=5;yk=5;
X=0:0.5:xk;Y=0:0.5:yk;
[x,y]=meshgrid(X,Y);
z=b0+b1*x+b2*y;
surf(x,y,z), colormap gray
grid on;
xlabel('x','FontSize',20);
ylabel('y','FontSize',20);
zlabel('z','FontSize',20);
set(gca,'FontSize',12);
set(gca,'box','on');
%Построение направляющего вектора
line([xk,0],[0,0],[b0,yk+b0],'LineWidth',5,'Color','k');
line([0,0],[0,0],[yk+b0,yk+b0],
  'LineWidth',5,'Color','k','marker','v','markersize',16);
text(0+0.3,0+0.3,yk+b0-1,'N','FontSize',20);
%Заголовок
title('3x-3z+6=0','FontSize',20)
```

Листинг 6.16

Если плоскость проходит через заданную точку $M_1(x_{1,}y_{1,}z_1)$ и параллельна плоскости $Ax+By+Cz+D=0$, то ее уравнение записывают так [7]

$$A(x-x_1)+B(y-y_1)+C(z-z_1)=0 \quad .$$

ЗАДАЧА 6.14. Записать уравнение и построить плоскость, проходящую через точку $M_0(1,3,5)$ параллельно плоскости $3x-3z+6=0$.

Уравнение плоскости имеет вид:

$$3(x-1)-3(z-5)=0\Rightarrow 3x-3-3z+15=0\Rightarrow 3x-3z+12=0$$

Решение задачи показано на рис. 6.11и в листинге 6.17.

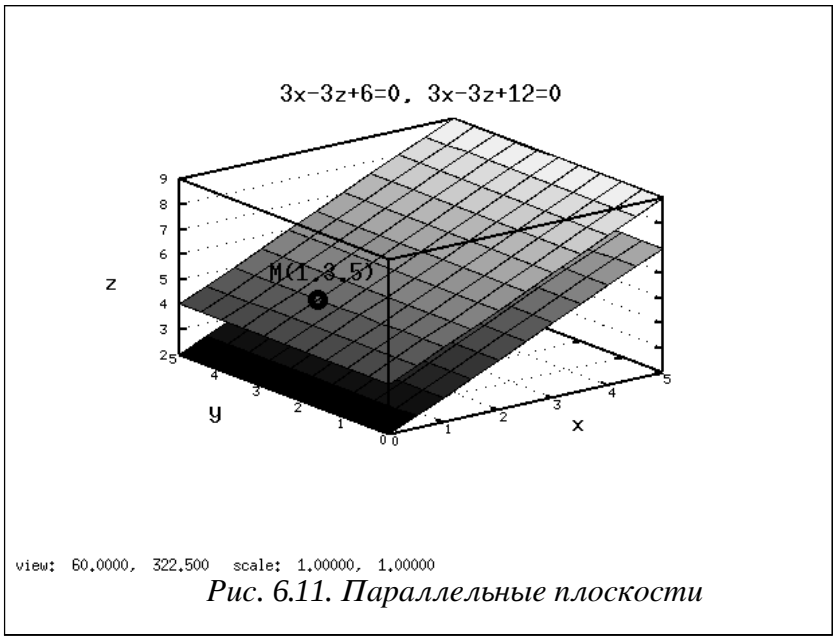


Рис. 6.11. Параллельные плоскости

```
function p=plos(A,B,C,D)
%Параметры уравнения плоскости,
%преобразованного к функции двух переменных
b0=-D/C;b1=-A/C;b2=-B/C;
%Построение плоскости
xk=5;yk=5;
X=0:0.5:xk;Y=0:0.5:yk;
[x,y]=meshgrid(X,Y);
z=b0+b1*x+b2*y;
surf(x,y,z), colormap gray
grid on;
p=0;
end;
clf;cla;
set(gcf,'Position',[20,20,400,400]);
axis([0,10,0,10,0,10])
%Параметры плоскости
A1=3;B1=0;C1=-3;D1=6;
%Построение плоскостей
plos(A1,B1,C1,D1)
hold on
A2=3;B2=0;C2=-3;D2=12;
p=plos(A2,B2,C2,D2)
xlabel('x','FontSize',20);
```

```
ylabel('y','FontSize',20);
zlabel('z','FontSize',20);
set(gca,'FontSize',12);
set(gca,'box','on');
%Изображение точки
M=[1,3,5];
line([M(1),M(1)],[M(2),M(2)],[M(3),M(3)],
  'LineWidth',5,'Color','k','marker','o','markersize',16);
text(M(1)-0.5,M(2)+0.5,M(3)+1,'M(1,3,5)','FontSize',20);
title('3x-3z+6=0, 3x-3z+12=0','FontSize',20)
```

Листинг 6.17

Если три точки $M_0(x_{0,}y_{0,}z_0)$, $M_1(x_{1,}y_{1,}z_1)$ и $M_2(x_{2,}y_{2,}z_2)$ не лежат на одной прямой, то уравнение плоскости, проходящей через них, представляется уравнением [7]:

$$\begin{vmatrix} x-x_0 & y-y_0 & z-z_0 \\ x_1-x_0 & y_1-y_0 & z_1-z_0 \\ x_2-x_0 & y_2-y_0 & z_2-z_0 \end{vmatrix}=0 \quad .$$

ЗАДАЧА 6.15. Записать уравнение и построить плоскость, проходящую через точки $M_0(1,2,3)$, $M_1(2,1,2)$ и $M_2(3,3,1)$.

Заданные точки не лежат на одной прямой, так как векторы $\overrightarrow{M_0M_1}$ и $\overrightarrow{M_0M_2}$ не коллинеарны. Рис. 6.12 и рис. 6.13 доказывают это утверждение. Изображение на рис. 6.12 получено в результате работы программы показанной в листинге 6.18. Рис. 6.13 получен из рис. 6.12 путем поворота в графическом окне.

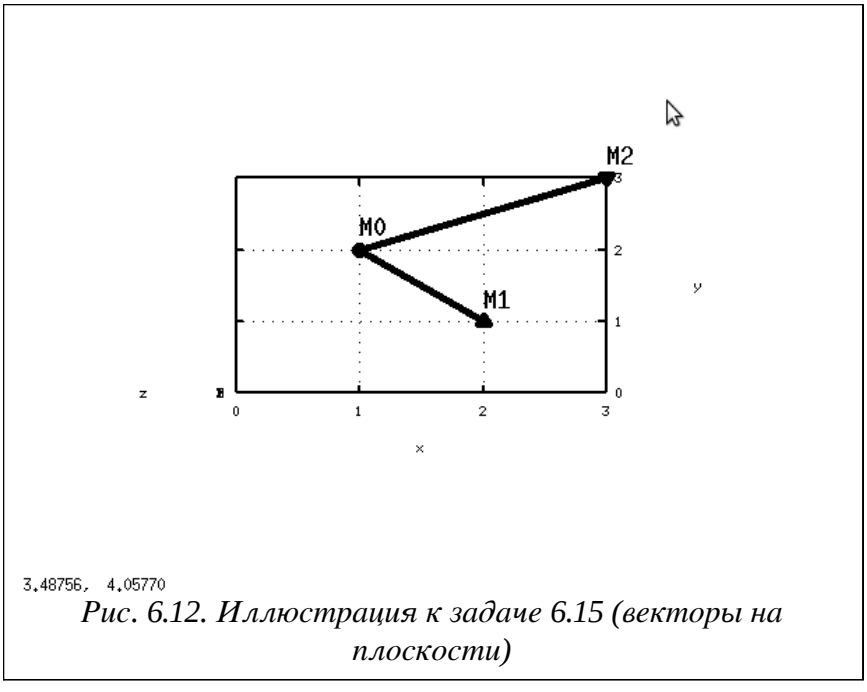


Рис. 6.12. Иллюстрация к задаче 6.15 (векторы на плоскости)

```
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'xlim',[0,3]);
set(gca,'ylim',[0,3]);
set(gca,'zlim',[0,3]);
set(gca,'xtick',[0:3]);
set(gca,'ytick',[0:3]);
```

```
set(gca,'ztick',[0:3]);
set(gca,'box','on');
xlabel('x');ylabel('y');zlabel('z');
axis([0,3,0,3,0,3])
grid on;
%Исходные данные
M0=[1,2,3];M1=[2,1,2];M2=[3,3,1];
%Изображение векторов M0M1 и M0M2
line([M1(1),M0(1)],[M1(2),M0(2)],[M1(3),M0(3)],
                                 'LineWidth',5,'Color','k');
line([M2(1),M0(1)],[M2(2),M0(2)],[M2(3),M0(3)],
                                 'LineWidth',5,'Color','k');
%Изображение стрелок на векторах
line([M1(1),M1(1)],[M1(2),M1(2)],[M1(3),M1(3)],
  'LineWidth',5,'Color','k','marker','>','markersize',10);
line([M2(1),M2(1)],[M2(2),M2(2)],[M2(3),M2(3)],
  'LineWidth',5,'Color','k','marker','<','markersize',10);
line([M0(1),M0(1)],[M0(2),M0(2)],[M0(3),M0(3)],
  'LineWidth',5,'Color','k','marker','o','markersize',10);
%Подписи
text(M0(1),M0(2)+0.3,M0(3),'M0','FontSize',20);
text(M1(1),M1(2)+0.3,M1(3),'M1','FontSize',20);
text(M2(1),M2(2)+0.3,M2(3),'M2','FontSize',20);
```

Листинг 6.18

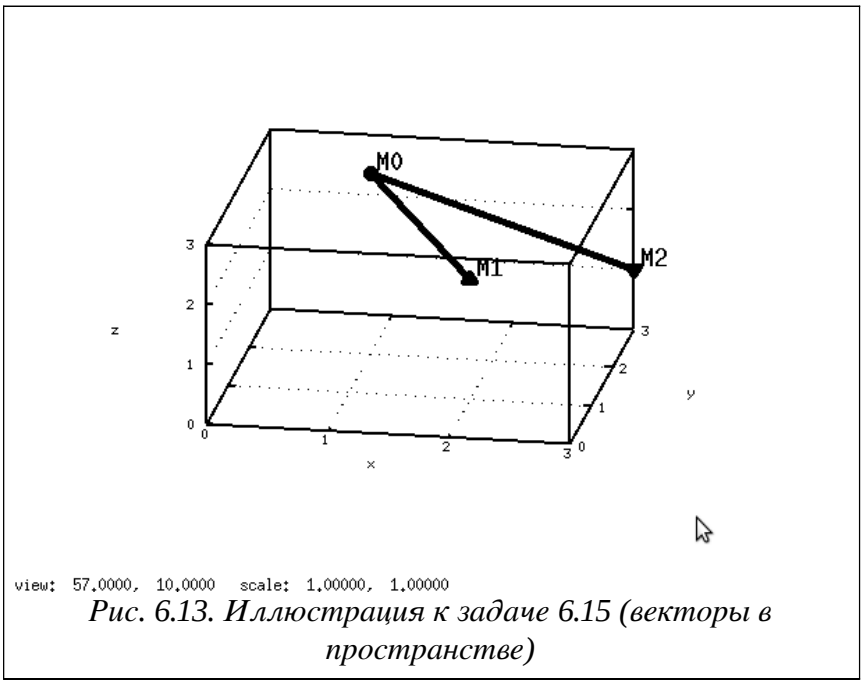


Рис. 6.13. Иллюстрация к задаче 6.15 (векторы в пространстве)

Плоскость представлена уравнением:

$$\begin{vmatrix} x-1 & y-2 & z-3 \\ 1 & -1 & -1 \\ 2 & 1 & -2 \end{vmatrix} = 0 \Rightarrow x+z-4=0 \quad .$$

Графическое решение задачи показано на рис. 6.14. Текст программы в листинге 6.19.

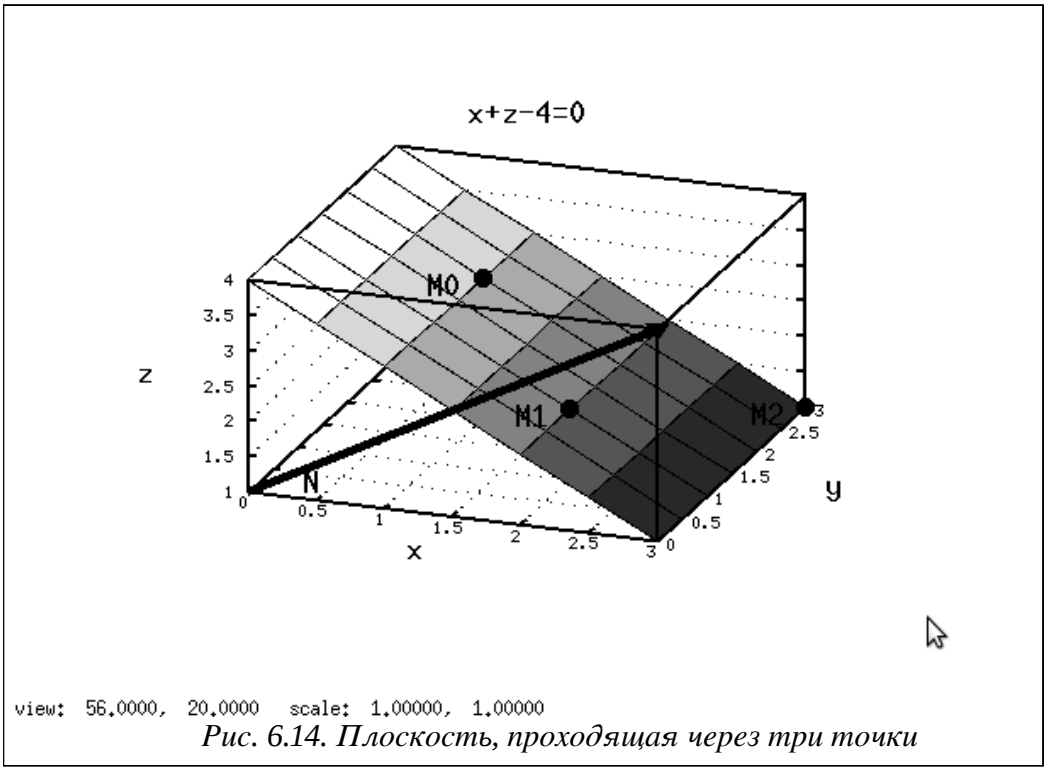


Рис. 6.14. Плоскость, проходящая через три точки

```
clf;cla;
set(gcf,'Position',[20,20,400,400]);
axis([0,10,0,10,0,10])
%Параметры плоскости
A=1;B=0;C=1;D=-4;N=[A,B,C];
%Параметры уравнения плоскости,
%преобразованного к функции двух переменных
b0=-D/C;b1=-A/C;b2=-B/C;
%Построение плоскости
xk=3;yk=3;
X=0:0.5:xk;Y=0:0.5:yk;
[x,y]=meshgrid(X,Y);
z=b0+b1*x+b2*y;
surf(x,y,z), colormap gray
grid on;
xlabel('x','FontSize',20);
ylabel('y','FontSize',20);
zlabel('z','FontSize',20);
set(gca,'FontSize',12);
set(gca,'box','on');
%Построение направляющего вектора
line([xk,0],[0,0],[b0,b0-yk],'LineWidth',5,'Color','k');
line([xk,xk],[0,0],[b0,b0],
  'LineWidth',5,'Color','k','marker','v','markersize',16);
text(0+0.3,0+0.3,b0-yk,'N','FontSize',20);
%Исходные данные
M0=[1,2,3];M1=[2,1,2];M2=[3,3,1];
%Нанесение точек на график
line([M1(1),M1(1)],[M1(2),M1(2)],[M1(3),M1(3)],
```

```
  'LineWidth',5,'Color','k','marker','o','markersize',10);
line([M2(1),M2(1)],[M2(2),M2(2)],[M2(3),M2(3)],
  'LineWidth',5,'Color','k','marker','o','markersize',10);
line([M0(1),M0(1)],[M0(2),M0(2)],[M0(3),M0(3)],
  'LineWidth',5,'Color','k','marker','o','markersize',10);
%Подписи
text(M0(1)-0.3,M0(2)-0.3,M0(3),'M0','FontSize',20);
text(M1(1)-0.3,M1(2)-0.3,M1(3),'M1','FontSize',20);
text(M2(1)-0.3,M2(2)-0.3,M2(3),'M2','FontSize',20);
%Заголовок
title('x+z-4=0','FontSize',20)
```

Листинг 6.19

Если плоскость $Ax+By+Cz+D=0$ не параллельна оси **OX** ($A\neq 0$), то она отсекает на этой оси отрезок $a=-\frac{D}{A}$. Аналогично отрезки на осях **OY, OZ** будут $b=-\frac{D}{B},(B\neq 0),$ и $c=-\frac{D}{C},(C\neq 0)$. Таким образом, плоскость отсекающую на осях отрезки *a*, *b* и *c* можно представить уравнением $\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=1$, которое называется *уравнением плоскости в отрезках* [7].

ЗАДАЧА 6.16. Написать уравнение плоскости $3x-6y+2z-12=0$ в отрезках и построить эту плоскость.

Найдем длины отрезков:

$a=-\frac{D}{A}=-\frac{-12}{3}=4$, $b=-\frac{D}{B}=-\frac{-12}{-6}=-2$, $c=-\frac{D}{C}=-\frac{-12}{3}=6$.

Запишем уравнение плоскости в отрезках:

$$\frac{x}{4}+\frac{y}{-2}+\frac{z}{6}=1$$

Решение задачи показано на рис. 6.15 и в листинге 6.20.

```
clf;cla; a=4;b=-2;c=6;
set(gcf,'Position',[50,50,400,400]);
axis([0,a,0,b,0,c]);
xlabel('x');ylabel('y');zlabel('z');
X=0:0.2:a;Y=b:0.2:0;
[x y]=meshgrid(X,Y);
z=c-c/a*x-c/b*y;
hfig=surf(x,y,z);
set(hfig,'FaceColor','none','EdgeColor','k')
%Изображение векторов a, b и c
line([0,a],[0,0],[0,0],'LineWidth',5,'Color','k');
line([0,0],[0,b],[0,0],'LineWidth',5,'Color','k');
line([0,0],[0,0],[0,c],'LineWidth',5,'Color','k');
%Подписи
text(a/2,0,1,'a','FontSize',20);
text(0,b/2,1,'b','FontSize',20);
text(0.2,0,c/2,'c','FontSize',20);
%Заголовок
title('a=4, b=-2, c=6','FontSize',14)
set(gca,'Position',[.1,.1,.8,.8]);set(gca,'View',[25 22])
```

Листинг 6.20

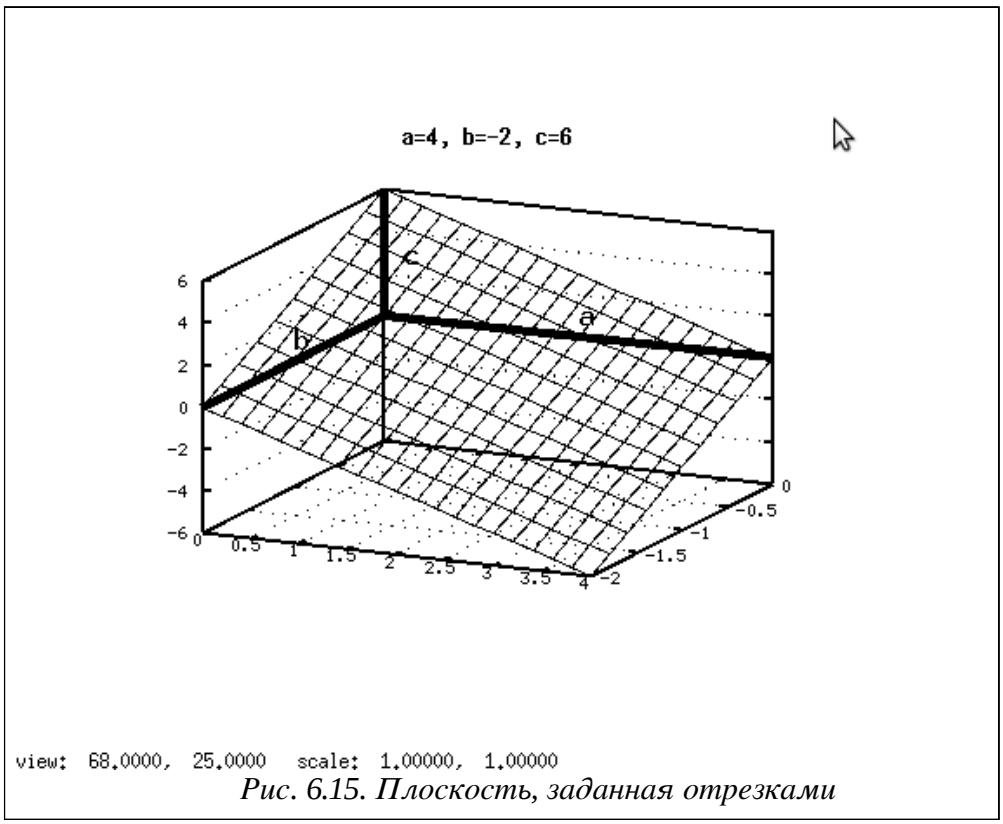


Рис. 6.15. Плоскость, заданная отрезками

Рассмотрим *особые случаи положения плоскости относительно системы координат:*

- Уравнение $Ax+By+Cz=0,(D=0)$ представляет плоскость, проходящую через начало координат.
- Уравнение $Ax+By+D=0,(C=0)$ представляет плоскость параллельную оси **OZ**, уравнение $Ax+CZ+D=0,(B=0)$ - плоскость, параллельную оси **OY**, уравнение $By+CZ+D=0,(A=0)$ - плоскость, параллельную оси **OX**.
- Уравнение $Ax+D=0,(B=0,C=0)$ представляет плоскость параллельную координатной плоскости **YOZ**, уравнение $By+D=0,(A=0,C=0)$ - плоскость, параллельную плоскости **XOZ**, уравнение $CZ+D=0,(A=0,B=0)$ - плоскость, параллельную оси **XOY**.
- Уравнения $X=0,Y=0,Z=0$ представляют собой плоскости **YOZ**, **XOZ** и **XOY**.

ЗАДАЧА 6.17. Построить плоскости $x+y-1=0$, $x-z+1=0$, $y+z+2=0$, $x-y+2=0$, $2x+3=0$, $3y-2=0$.

Ход решения задачи описан в листинге 6.21. Графическое решение показано на рис. 6.16.

```
function p=plos(A,B,C,D)
%Параметры плоскости
M=[A,B,C];
d=[0.1,0.1,0.1];
if A==0
      M(1)=1;
end;
if B==0
      M(2)=1;
end;
```

```
if C==0
      M(3)=1;
end;
if A<0
      d(1)=-0.1;
end;
if B<0
      d(2)=-0.1;
end;
if C<0
      d(3)=-0.1;
end;
X=0:d(1):M(1);
Y=0:d(2):M(2);
Z=0:d(3):M(3);
%Построение плоскости
if C!=0
%Параметры уравнения плоскости,
%преобразованного к функции двух переменных z(x,y)
      [x,y]=meshgrid(X,Y);
      b0=-D/C;b1=-A/C;b2=-B/C;
      z=b0+b1*x+b2*y; f1=surf(x,y,z); colormap gray
else
if B!=0
%Параметры уравнения плоскости,
%преобразованного к функции двух переменных y(x,z)
      [x,z]=meshgrid(X,Z);
      b0=-D/B;b1=-A/B;b2=-C/B;
      y=b0+b1*x+b2*z; f1=surf(x,y,z); colormap gray
else
%Параметры уравнения плоскости,
%преобразованного к функции двух переменных x(y,z)
      [y,z]=meshgrid(Y,Z);
      b0=-D/A; b1=-B/A; b2=-C/A;
      x=b0+b1*y+b2*z; f1=surf(x,y,z); colormap gray
end;
end;
grid on;
xlabel('x');
ylabel('y');
zlabel('z');
set(gca,'xtick',[0:M(1)]);
set(gca,'ytick',[0:M(1)]);
set(gca,'ztick',[0:M(3)]);
set(gca,'box','on');
p=f1;
end;
% конец функции
```

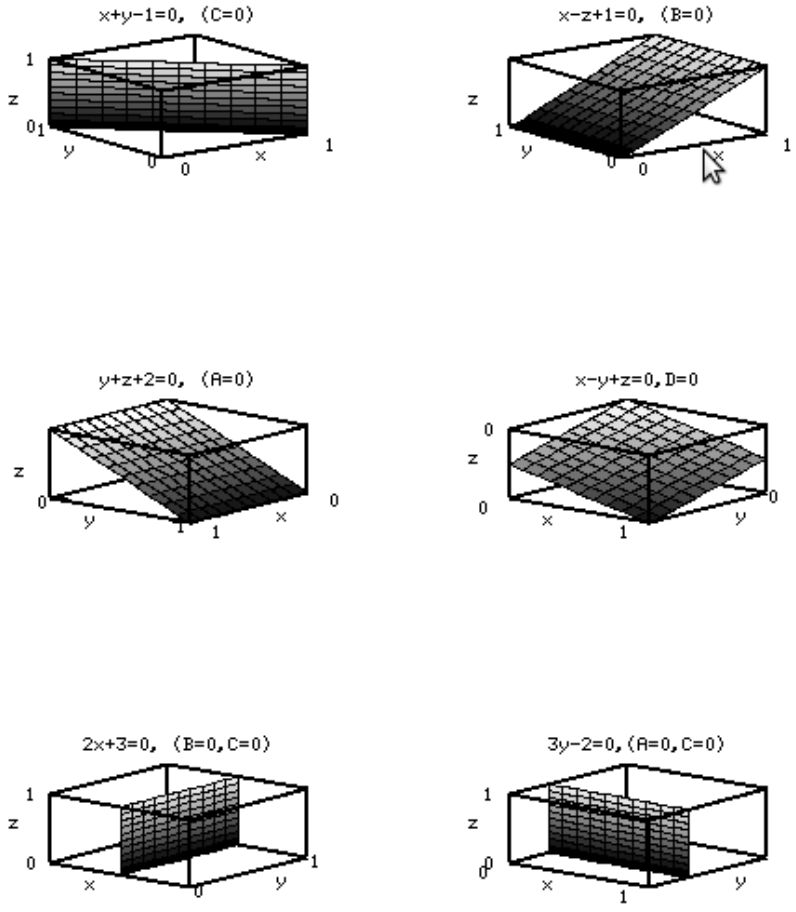


Рис. 6.16. Особые случаи положения плоскости относительно системы координат

```
%Изображение плоскостей
clf;cla;
subplot(3,2,1);
%Плоскость x+y-1=0
A1=1;B1=1;C1=0;D1=-1;
plos(A1,B1,C1,D1)
title('x+y-1=0, (C=0)')
subplot(3,2,2);
%Плоскость x-z+1=0
A2=1;B2=0;C2=-1;D2=1;
plos(A2,B2,C2,D2)
title('x-z+1=0, (B=0)')
subplot(3,2,3);
%Плоскость y+z+2=0
```

```
A3=0;B3=1;C3=1;D3=2;
p=plos(A3,B3,C3,D3)
title('y+z+2=0, (A=0)')
set(gca,'View',[130 30])
%Плоскость x-y+z-2=0
subplot(3,2,4);
A4=1;B4=-1;C4=1;D4=0;
plos(A4,B4,C4,D4)
set(gca,'View',[40 30])
title('x-y+z=0,D=0')
%Плоскость 2x+3=0
subplot(3,2,5);
A5=2;B5=0;C5=0;D5=3;
plos(A5,B5,C5,D5)
set(gca,'View',[40 30])
title('2x+3=0, (B=0,C=0)')
%Плоскость 3y-2=0
subplot(3,2,6);
A6=0;B6=3;C6=0;D6=-2;
plos(A6,B6,C6,D6)
set(gca,'View',[40 30])
title('3y-2=0,(A=0,C=0)')
```

Листинг 6.21

Расстояние от точки $M_1(x_1, y_1, z_1)$ *до плоскости* $Ax+By+Cz+D=0$ равно абсолютному значению величины $d=\dfrac{|Ax_1+Bx_2+Cx_3+D|}{\sqrt{A^2+B^2+C^2}}$.

ЗАДАЧА 6.18. Найти расстояние от точки $M_1(3,9,1)$ до плоскости $x-2y+2z+3=0$.

Решение показано в листинге 6.22.

```
%Исходные данные
A=1;B=-2;C=2;D=-3;
M=[3,9,1];
N=[A;B;C];
%Расстояние от точки M(3,9,1) до плоскости x-2y+2z-3=0
d=abs(M*N+D)/norm(N)
>>>d =  5.3333
```

Листинг 6.22

Две плоскости $A_1x+B_1y+C_1z+D_1=0$ и $A_2x+B_2y+C_2z+D_2=0$ образуют четыре *двугранных угла*, равных попарно. Один из них всегда равен углу между нормальными векторами $\vec{N}_1\{A_1, B_1, C_1\}$ и $\vec{N}_2\{A_2, B_2, C_2\}$. Вычисляют любой из двугранных углов по формуле [7]

$$\cos(\phi)=\pm\frac{A_1A_2+B_1B_2+C_1C_2}{\sqrt{A_1^2+B_1^2+C_1^2}\sqrt{A_2^2+B_2^2+C_2^2}} ,$$

причем выбирая «+» получаем $\cos(\widehat{N_1N_2})$, выбирая «-» получаем $\cos(180-\widehat{N_1N_2})$.

ЗАДАЧА 6.19. Найти угол между плоскостями $x-y+\sqrt{2}z+2=0$ и $x+y+\sqrt{2}z-3=0$.

Решение показано в листинге 6.23.

```
%Исходные данные
N1=[1,-1,sqrt(2)];
```

```
N2=[1,1,sqrt(2)];
%Угол между плоскостями
fi=acos(dot(N1,N2)/norm(N1)/norm(N2));
fi_1=round(fi*180/pi)
fi_2=180-fi_1
%Решение
>>>fi_1 =  60
>>>fi_2 =  120
```

Листинг 6.23

Два уравнения $A_1x+B_1y+C_1z+D_1=0$ и $A_2x+B_2y+C_2z+D_2=0$ представляют *прямую линию*, если коэффициенты A_1, B_1, C_1 не пропорциональны коэффициентам A_2, B_2, C_2 (то есть плоскости не параллельны). Если коэффициенты A_1, B_1, C_1 пропорциональны коэффициентам A_2, B_2, C_2, но свободные члены не подчинены той же пропорции $\frac{A_2}{A_1}=\frac{B_2}{B_1}=\frac{C_2}{C_1}\neq\frac{D_2}{D_1}$, то заданные уравнения не представляют никакого геометрического образа. Если все четыре величины пропорциональны $\frac{A_2}{A_1}=\frac{B_2}{B_1}=\frac{C_2}{C_1}=\frac{D_2}{D_1}$, то заданные уравнения представляют одну и ту же плоскость [7].

ЗАДАЧА 6.20. Построить прямые линии, заданные уравнениями $2x-y=0$ и $x+y-1=0$, $x-y+z-1=0$ и $2x-2y+2z-2=0$, $2x-7y+12z-4=0$ и $4x-14y+24z-12=0$.

Решение показано в листинге 6.24. Для построения плоскости применялась функция `plos(A,B,C,D)`, описанная в задаче 6.16.

```
function flag=line_(N1,N2)
if N1(1)==0
      k1=0;
else
      k1=N2(1)/N1(1);
end;
if N1(2)==0
      k2=0;
else
      k2=N2(2)/N1(2);
end;
if N1(3)==0
      k3=0;
else
      k3=N2(3)/N1(3);
end;
if N1(4)==0
      k4=0;
else
      k4=N2(4)/N1(4);
end;
if  ( k1!= k2) | (k2 != k3)
      flag=0
      clf;cla;
      plos(N1(1),N1(2),N1(3),N1(4));
      hold on
      plos(N2(1),N2(2),N2(3),N2(4));
```

```
elseif  (k1 == k2) & (k2 == k3) &  (k3 == k4)
      flag=1;
      clf;cla;
      plos(N1(1),N1(2),N1(3),N1(4));
elseif  (k1 == k2) & (k2 == k3) &  (k3 != k4)
      flag=2;
      disp('Геометрическая фигура не определена!')
end;
end;
%Пример 1
A1=2;B1=-1;C1=0;D1=0;
A2=1;B2=1;C2=0;D2=-1;
n1=[A1,B1,C1,D1];
n2=[A2,B2,C2,D2];
line_(n1,n2)
title('2x-y=0, x+y-1=0');
set(gca,'View',[110 30]);
```

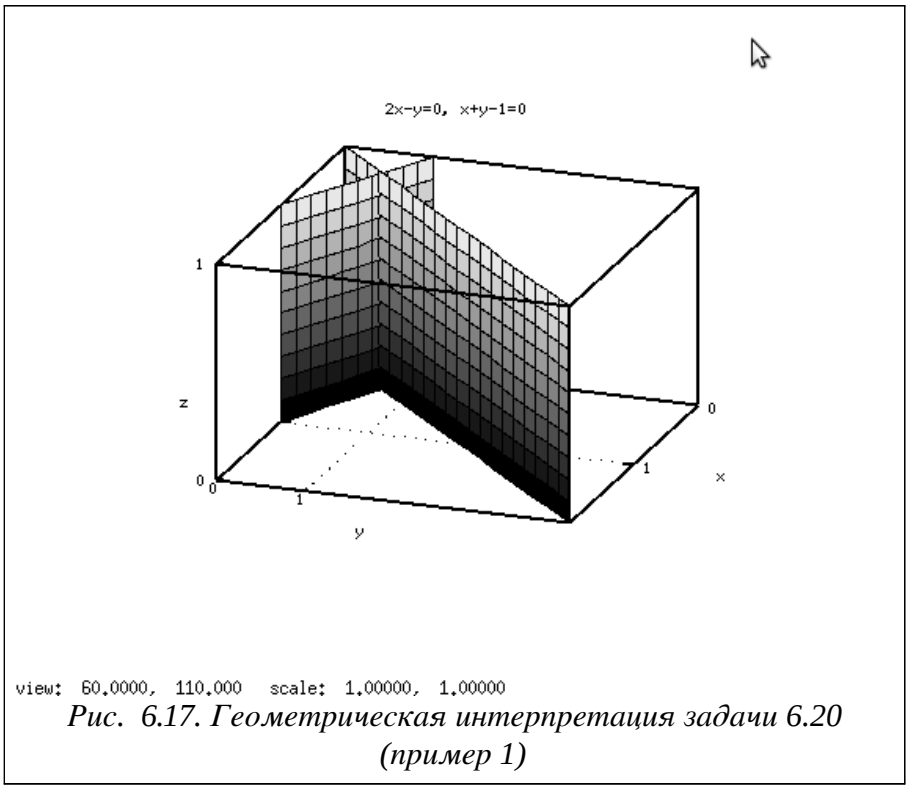


Рис. 6.17. Геометрическая интерпретация задачи 6.20 (пример 1)

```
%Пример 2
A1=1;B1=-1;C1=1;D1=-1;
A2=2;B2=-2;C2=2;D2=-2;
n1=[A1,B1,C1,D1];
n2=[A2,B2,C2,D2];
line_(n1,n2)
title('x-y+z-1=0, 2x-2y+2z-2=0');
set(gca,'View',[60 30]);
```

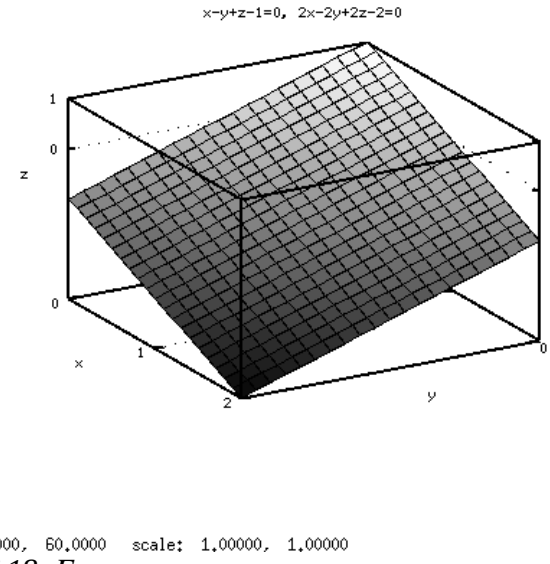


Рис. 6.18. Геометрическая интерпретация задачи 6.20 (пример 2)

```
%Пример 3
A1=2;B1=-7;C1=12;D1=-4;
A2=4;B2=-14;C2=24;D2=-12;
n1=[A1,B1,C1,D1];
n2=[A2,B2,C2,D2];
line_(n1,n2)
%Результат работы пример 3
>>>Геометрическая фигура не определена!
```

Листинг 6.24

Всякий вектор $\vec{a}\{l,m,n\}$, лежащий на прямой (или параллельный ей), называется *направляющим вектором* этой прямой. Координаты $\{l,m,n\}$ называются *направляющими коэффициентами* прямой. За направляющий вектор прямой $A_1x+B_1y+C_1z+D_1=0$, $A_2x+B_2y+C_2z+D_2=0$ можно принять векторное произведение $\vec{N}_1\times\vec{N}_2$, где $\vec{N}_1=\{A_1,B_1,C_1\}$, $\vec{N}_2=\{A_2,B_2,C_2\}$ - нормальные векторы плоскостей, образующих прямую [7].

ЗАДАЧА 6.21. Найти направляющие коэффициенты прямой $2x-2y-z+8=0$ и $x+2y-2z+1=0$ (листинг 6.25).

```
N1=[2,-2,-1];N2=[1,2,-2];
%Расчет координат векторного произведения
M=[N1;N2];
M1=M(1:2,2:3);M2=[M(:,1),M(:,3)];M3=M(1:2,1:2);
a(1)=det(M1);
a(2)=-det(M2);
a(3)=(det(M3));
a
%Векторное произведение
>>>a =    6    3    6
```

Листинг 6.25

Прямая L, проходящая через точку $M_0(x_0,y_0,z_0)$ и имеющая направляющий вектор

$\vec{a}\{l,m,n\}$ представляется уравнениями $\frac{x-x_0}{l}=\frac{y-y_0}{m}=\frac{z-z_0}{n}$. Эти уравнения выражают коллинеарность векторов $\overrightarrow{M_0M_1}\{x-x_0, y-y_0, z-z_0\}$, $\vec{a}\{l,m,n\}$ и называются *каноническими уравнениями прямой*.

Уравнения $x=x_0+lt, y=y_0+mt, z=z_0+nt$ называют *параметрическими уравнениями прямой*. Здесь величина t является *параметром* и принимает различные значения [7].

ЗАДАЧА 6.22. Записать параметрическое уравнение прямой, проходящей через две точки $A(5,-3,2)$ и $B(3,1,-2)$.

Если в качестве направляющего вектора выбрать вектор $\overrightarrow{AB}=\{3-5,1-(-3),-2-2\}=\{-2,4,-4\}$, то каноническое уравнение будет иметь вид $\frac{x-5}{-2}=\frac{y+3}{4}=\frac{z-2}{-4}$, следовательно параметрическое уравнение запишем так: $x=5+2t, y=-3+4t, z=2-4t$.

Команды, которые применялись для графического решения задачи (рис. 6.19) показаны в листинге 6.26.

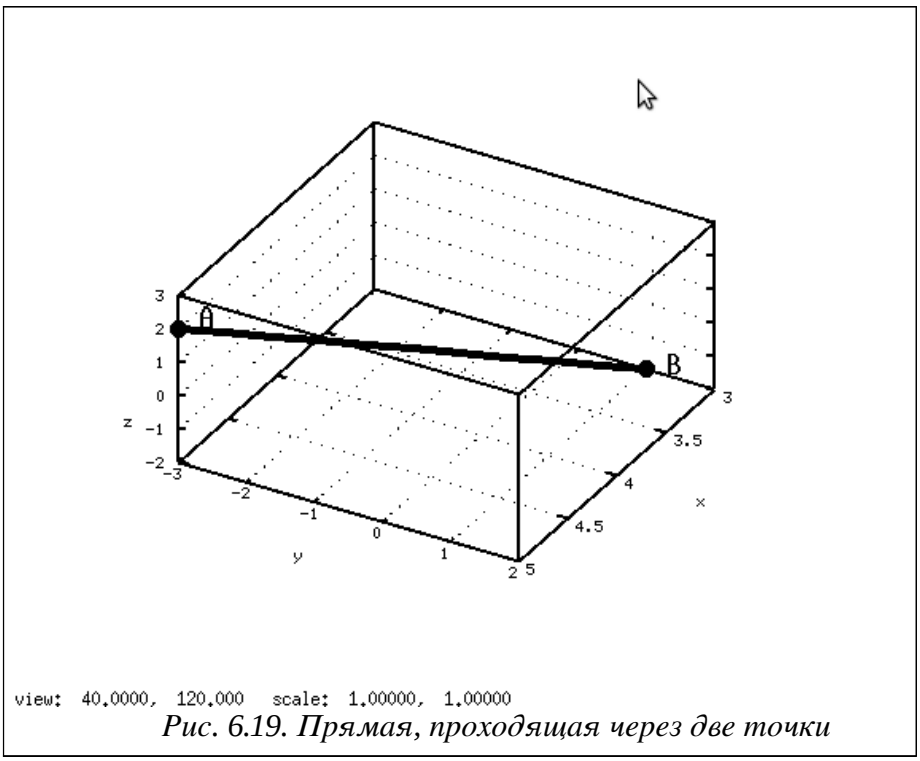


Рис. 6.19. Прямая, проходящая через две точки

```
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gca,'Position',[.1,.1,.8,.8]);
xlabel('x');ylabel('y');zlabel('z');
%axis([0,3,0,3,0,3])
grid on;
%Исходные данные
A=[5,-3,2];B=[3,1,-2];
t=0:0.1:1;
x=5-2*t; y=-3+4*t; z=2-4*t;
%Изображение прямой
line(x,y,z,'LineWidth',5,'Color','k');
%Изображение точек
```

```
line([A(1),A(1)],[A(2),A(2)],[A(3),A(3)],
  'LineWidth',5,'Color','k','marker','o','markersize',10);
line([B(1),B(1)],[B(2),B(2)],[B(3),B(3)],
  'LineWidth',5,'Color','k','marker','o','markersize',10);
%Подписи
text(A(1),A(2)+0.3,A(3)+0.5,'A','FontSize',20);
text(B(1),B(2)+0.3,B(3)+0.3,'B','FontSize',20);
set(gca,'View',[120 50]);
```

Листинг 6.26

Если известны направляющие векторы двух прямых $\vec{a}\{l,m,n\}$ и $\vec{b}\{l',m',n'\}$, то угол между этими прямыми можно вычислить по формуле

$$\cos(\phi)=\pm\frac{ll'+mm'+nn'}{\sqrt{l^2+m^2+n^2}\sqrt{l'^2+m'^2+'^2}} .$$

ЗАДАЧА 6.23. Найти угол между прямыми $x=t, y=2t, z=3t$ и $x=-1+2t, y=1+t, z=-1+4t$ (листинг 6.27, рис. 6.20).

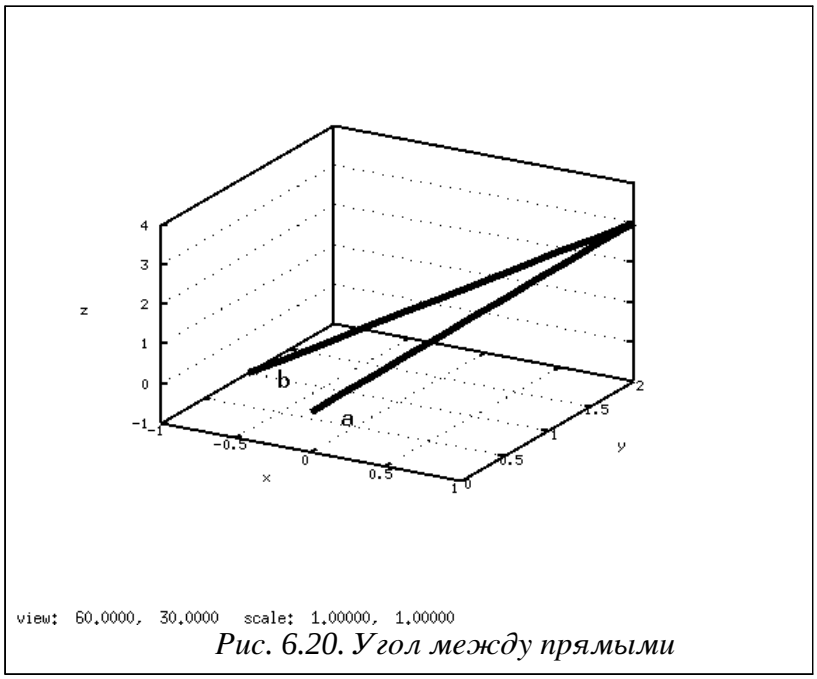


Рис. 6.20. Угол между прямыми

```
%Исходные данные
a=[1,2,3];b=[2,1,4];
fi=acos(dot(a,b)/norm(a)/norm(b));
fi_1=round(fi*180/pi)
clf;cla;
set(gcf,'Position',[20,20,400,400]);
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'box','off');
xlabel('x');ylabel('y');zlabel('z');
%axis([0,3,0,3,0,3])
grid on;
t=0:0.1:1;
x=-1+2*t; y=1+t; z=-1+4*t;
%Изображение прямой
line(x,y,z,'LineWidth',5,'Color','k');
%Подписи
```

```
text(-0.8,1,-1,'b','FontSize',20);
x=t; y=2*t; z=3*t;
%Изображение прямой
line(x,y,z,'LineWidth',5,'Color','k');
%Подписи
text(0.2,0,0,'a','FontSize',20);
set(gca,'View',[60 320])
%Результат
>>>fi_1 =  21
```

Листинг 6.27

Прямая $x=x_0+l\,t\,,y=y_0+mt\,,z=z_0+nt$ и плоскость $Ax+By+Cz+D=0$ могут иметь одну общую точку, могут не иметь общих точек (прямая параллельна плоскости) и иметь бесконечное множество общих точек (прямая лежит на плоскости). *Общую точку* (если такая существует) плоскости и прямой можно вычислить, если подставить уравнение прямой в уравнение плоскости и найти значение параметра *t* [7].

ЗАДАЧА 6.24. Найти точку пересечения прямой $x=-5+3t\,,y=3-t\,,z=-3+2t$ с плоскостью $2x+3y+3z-8=0$.

Выполним расчеты в технике символьных вычислений (листинг 6.28).

```
symbols
%Определение символьных переменных
x = sym ("x") ;
y = sym ("y");
z = sym ("z");
t = sym ("t");
%Параметрическое уравнение прямой
x=-5+3*t;
y=3-t;
z=-3+2*t;
%Уравнение плоскости
f=2*x+3*y+3*z-8
%Вычисление значения параметра t
t = symfsolve(f,0)
%Определение точки пересечения прямой и плоскости
x=-5+3*t
y=3-t
z=-3+2*t
%Результаты вычислений
%Уравнение плоскости, выраженное через параметр t
>>>f =
-18.0+(9.0)*t
%Значение параметра t
>>>t =  2.0000
%Точка пересечения прямой и плоскости
>>>x =  1.00000
>>>y =  1.0000
>>>z =  1.00000
```

Листинг 6.28

7. Нелинейные уравнения и системы

В общем случае аналитическое решение уравнения $f(x)=0$ можно найти только для узкого класса функций. Чаще всего приходится решать это уравнение численными методами. *Численное решение уравнения* проводят в два этапа. На *первом этапе* отделяют корни уравнения, т.е. находят достаточно тесные промежутки, в которых содержится только один корень. Эти промежутки называют *интервалами изоляции корня*. Определить интервалы изоляции корня можно, например, изобразив график функции. Идея *графического метода* основана на том, что непрерывная функции *f(x)* имеет на интервале [*a*, *b*] хотя бы один корень, если она поменяла знак $f(a)\cdot f(b)<0$. Границы интервала *a* и *b* называют *пределами интервала изоляции*. На *втором этапе* проводят *уточнение* отделенных корней, т.е. находят корни с заданной точностью [9].

7.1 Многочлены и действия над ними

Рассмотрим возможности пакета при работе с многочленами. В общем виде *многочлен* (*полиномом*) может быть записан так:

$$a_0+a_1x+a_2x^2+...+a_nx^n \ ,$$

где x – переменная, a_i – коэффициенты полинома, n – его степень [7].

В Octave *определить многочлен* можно в виде вектора его коэффициентов $p=\{a_n, a_{n-1}, ..., a_1, a_0\}$. Например, полином $2x^5+3x^3-1$ задается вектором

```
>>> p=[2,0,3,0,-1]
p = 2 0 3 0 -1
```

Листинг 7.1

Рассмотрим функции, предназначенные для *действий над многочленами*.

Произведение двух многочленов вычисляет функция

```
q =conv(p1, p2).
```

где `p1` — многочлен степени *n*, `p2` — многочлен степени *m*. Функция формирует вектор `q`, соответствующий коэффициентам многочлена степени *n*+*m*, полученного в результате умножения `p1` на `p2`;

ЗАДАЧА 7.1. Определить многочлен, который получиться в результате умножения выражений $3x^4-7x^2+5$ и x^3+2x-1. Как видно из листинга 7.2 в результате имеем: $(3x^4-7x^2+5)(x^3+2x-1)=3x^7-x^5-3x^4-9x^3+7x^2+10x-5$.

```
>>> p1=[3 0 -7 0 5]; p2=[1 0 2 -1];
p=conv(p2,p1)
>>>p =     3     0    -1    -3    -9     7    10    -5
```

Листинг 7.2

Частное и остаток от деления двух многочленов находит функция

```
[q, r] =deconv((p1, p2)
```

здесь, `p1` — многочлен степени *n*, `p2` — многочлен степени *m*. Функция формирует вектор `q` – коэффициенты многочлена, который получается в результате деления `p1` на `p2` и вектор `r` – коэффициенты многочлена, который является остатком от деления `p1` на `p2`;

ЗАДАЧА 7.2. Найти частное и остаток от деления многочлена $x^6-x^5+3x^4-8x^2+x-10$ на многочлен x^3+x-1.

В результате имеем (листинг 7.3)

$$\frac{x^6-x^5+3x^4-8x^2+x-10}{x^3+x-1}=x^3-x^2+2x+2+\frac{1}{-11x^2+x-8} \ .$$

```
>>> p1=[1 -1 3 0 -8 1 -10]; p2=[1 0 1 -1];
```

```
[q,r]=deconv(p1,p2)
>>>q =  1  -1   2   2
r =     0    0    0    0  -11    1   -8
```

Листинг 7.3

Выполнить *разложение частного двух многочленов*, представляющих собой правильную дробь на простейшие рациональные дроби вида [7]

$$\frac{P_1(x)}{P_2(x)}=\sum_{j=1}^{M}\frac{a_j}{(x-b_j)^{k_j}}+\sum_{i=1}^{N}c_i x^{N-i}$$

можно с помощью функции

```
[a, b, c, k] = residue(p1, p2),
```

где `p1` — многочлен степени n (числитель), `p2` — многочлен степени m (знаменатель), причем $n<m$. В результате работы функция формирует четыре вектора: `a` — вектор коэффициентов, расположенных в числителях простейших дробей, `b` — вектор коэффициентов, расположенных в знаменателях простейших дробей, `k` — вектор степеней знаменателей простейших дробей (кратность), `c` — вектор коэффициентов остаточного члена.

ЗАДАЧА 7.3. Разложить выражение $\frac{x^3+1}{x^4-3x^3+3x^2-x}$ на простейшие дроби. Проанализировав листинг 7.4 запишем решение:

$$\frac{x^3+1}{x^4-3x^3+3x^2-x}=\frac{2}{x-1}+\frac{1}{(x-1)^2}+\frac{2}{(x-1)^3}+\frac{-1}{x}\quad .$$

Значение вектора `c = [](0x0)` говорит об отсутствии остаточного члена.

```
>>> p1=[1 0 0 1]; p2=[1 -3 3 -1 0];
[a,b,c,k]=residue(p1,p2)
>>>a =
   2.00000
   1.00000
   2.00000
  -1.00000
b =
   1.00000
   1.00000
   1.00000
   0.00000
c = [](0x0)
k =
   1
   2
   3
   1
```

Листинг 7.4

ЗАДАЧА 7.4. Разложить выражение $\frac{7x^2+26x-9}{x^4+4x^3+4x^2-9}$ на простейшие дроби. Из листинга 7.5 видно, что значения векторов `a` и `b` — комплексные числа, то есть, на первый взгляд кажется, что выражение невозможно разложить на простейшие рациональные дроби. Однако задача имеет решение:

$$\frac{7x^2+26x-9}{x^4+4x^3+4x^2-9}=\frac{1}{x-1}+\frac{1}{x+3}+\frac{-2x+5}{x^2+2x+3}\ .$$

Выражение $\frac{-2x+5}{x^2+2x+3}$ — простейшая дробь вида $\frac{Mx+N}{x^2+px+q}$. Здесь знаменатель невозможно разложить на простые рациональные множители первой степени. Таким образом, функция `residue(p1, p2)` выполняет разложение только на простейшие дроби вида $\frac{a}{(x-b)^k}$.

```
>>>a =
  -1.3738 + 1.2229i
  -1.3738 - 1.2229i
   1.3738 + 0.7771i
   1.3738 - 0.7771i
b =
  -2.4426 + 1.0398i
  -2.4426 - 1.0398i
   0.4426 - 1.0398i
   0.4426 + 1.0398i
c = [](0x0)
k =
   1
   1
   1
   1
```

Листинг 7.5

Вычислить *значение многочлена в заданной точке* можно с помощью функции

`polyval(p, x)`

где `p` - многочлен степени *n*, `x` — значение, которое нужно подставить в многочлен.

ЗАДАЧА 7.5. Вычислить значение многочлена $x^6-x^5+3x^4-8x^2+x-10$ в точках $x_1=-1, x_2=1$ (листинг 7.6).

```
>>> p=[1 -1 3 0 -8 1 -10];
x=[-1,1];
polyval(p,x)
>>>ans =
  -14  -14
```

Листинг 7.6

Вычислить *производную от многочлена* позволяет функция

`polyder(p)`

где `p` - многочлен степени *n*. Функция формирует вектор коэффициентов многочлена, являющегося производной от `p`.

Производную произведения двух векторов вычисляет функция

`polyder(p1, p2)`

где `p1` и `p2` — многочлены.

Вызов функции в общем виде

`[q, r]= polyder(p1, p2)`

приведет к вычислению *производной от частного* `p1` на `p2` и выдаст результат в виде отношения полиномов `q` и `r`.

ЗАДАЧА 7.6. Вычислить производную от многочлена $x^6-x^5+3x^4-8x^2+x-10$. Листинг 7.7 показал, что решением задачи является многочлен

$$6x^5-5x^4+12x^3-16x+1 \quad .$$

```
>>> p=[1 -1 3 0 -8 1 -10];
polyder(p)
>>>ans =
    6   -5   12    0  -16    1
```

Листинг 7.7

ЗАДАЧА 7.7. Вычислить производную от многочлена $(3x^4-7x^2+5)(x^3+2x-1)$.

Листинг 7.8 показал, что решением задачи является многочлен

$$21x^6-5x^4-12x^3-27x^2+14x+10 \quad .$$

```
>>> p1=[3 0 -7 0 5];
p2=[1 0 2 -1];
polyder(p1,p2)
>>>ans =
   21    0   -5  -12  -27   14   10
```

Листинг 7.8

ЗАДАЧА 7.8. Вычислить производную от выражения $\dfrac{x^3+1}{x^4-3x^3+3x^2-x}$.

Из листинга 7.9 видно, что решение задачи имеет вид

$$\frac{-x^4-2x^3-4x+1}{x^6-4x^5+6x^4-4x^3+x^2} \quad .$$

```
>>> p1=[1 0 0 1]; p2=[1 -3 3 -1 0];
[q,r]=polyder(p1,p2)
>>>q =
  -1  -2   0  -4   1
r =
  1  -4   6  -4   1  0  0
```

Листинг 7.9

Взять *интеграл от многочлена* позволяет функция

```
polyint(p [,K]),
```

где `p` - многочлен степени *n*, `K` — постоянная интегрирования, значение `K` по умолчанию равно нулю. Функция формирует вектор коэффициентов многочлена, являющегося интегралом от `p`.

ЗАДАЧА 7.9. Найти $\int (x^2+2x+3)dx$.

Согласно листингу 7.10 решение имеет вид:

$$\int (x^2+2x+3)dx=0.333x^3+x^2+3x \quad .$$

Если определить значение постоянной интегрирования (листинг 7.11), то решение будет таким:

$$\int (x^2+2x+3)dx=0.333x^3+x^2+3x+5 \quad .$$

```
>>> p=[1 2 3]; polyint(p)
>>> ans
0.33333 1.00000 3.00000 0.00000
```

Листинг 7.10

```
>>> p=[1 2 3];
polyint(p,5)
>>>ans =
   0.33333   1.00000   3.00000   5.00000
```

Листинг 7.11

7.2 Решение алгебраических уравнений

Любое уравнение $P(x)=0$, где $P(x)$ это многочлен (полиномом), отличный от нулевого, называется *алгебраическим уравнением* относительно переменных x. Всякое алгебраическое уравнение относительно x можно записать в виде

$$a_0+a_1x+a_2x^2+...+a_nx^n=0 \quad ,$$

где, a_i – коэффициенты алгебраического уравнения n–й степени [9]. Например, линейное уравнение это алгебраическое уравнение первой степени, квадратное – второй, кубическое – третьей и так далее.

Построить многочлен по заданному вектору его корней позволяет функция

`poly(x),`

где `x` — вектор корней искомого полинома.

ЗАДАЧА 7.10. Записать алгебраическое уравнение, если известно, что его корни $x_1=-2, x_2=3$. Согласно листингу 7.12 решение задачи имеет вид: $x^2-x-6=0$

```
>>> x=[-2 3];
poly(x)
>>>ans = 1  -1  -6
```

Листинг 7.12

Решить *алгебраическое уравнение* $P(x)=0$ можно при помощи встроенной функции

`roots(p),`

где `p` - многочлен степени n. Функция формирует вектор, элементы которого являются корнями заданного полинома.

ЗАДАЧА 7.11. Решить алгебраическое уравнение $x^2-x-6=0$.

Из листинга 7.13 видно, что значения $x_1=-2, x_2=3$ являются решением уравнения.

```
p=[1 -1 -6];
roots(p)
>>>ans =
    3
   -2
```

Листинг 7.13

ЗАДАЧА 7.12. Найти корни полинома $2x^3-3x^2-12x-5=0$.

Найдем корни полинома (листинг 7.14).

```
>>> p=[2 -3 -12 -5];
x=roots(p)
>>>x =
    3.44949
   -1.44949
   -0.50000
```

Листинг 7.14

Графическое решение заданного уравнения показано в листинге 7.15 и на рис. 7.1. Точки пересечения графика с осью абсцисс и есть корнями уравнения. Не трудно заметить, что графическое решение совпадает с найденным в листинге 7.14.

```
cla;
okno1=figure();
x=-2:0.1:5.5;
y=2*x.^3-3*x.^2-12*x-5;
pol=plot(x,y);
set(pol,'LineWidth',3,'Color','k')
set(gca,'xlim',[-2,4]);
set(gca,'ylim',[-30,30]);
```

```
set(gca,'xtick',[-2:0.5:4]);
set(gca,'ytick',[-30:5:30]);
grid on;
xlabel('x');
ylabel('y');
title('Plot y=2*x^3-3*x^2-12*x-5');
```

Листинг 7.15

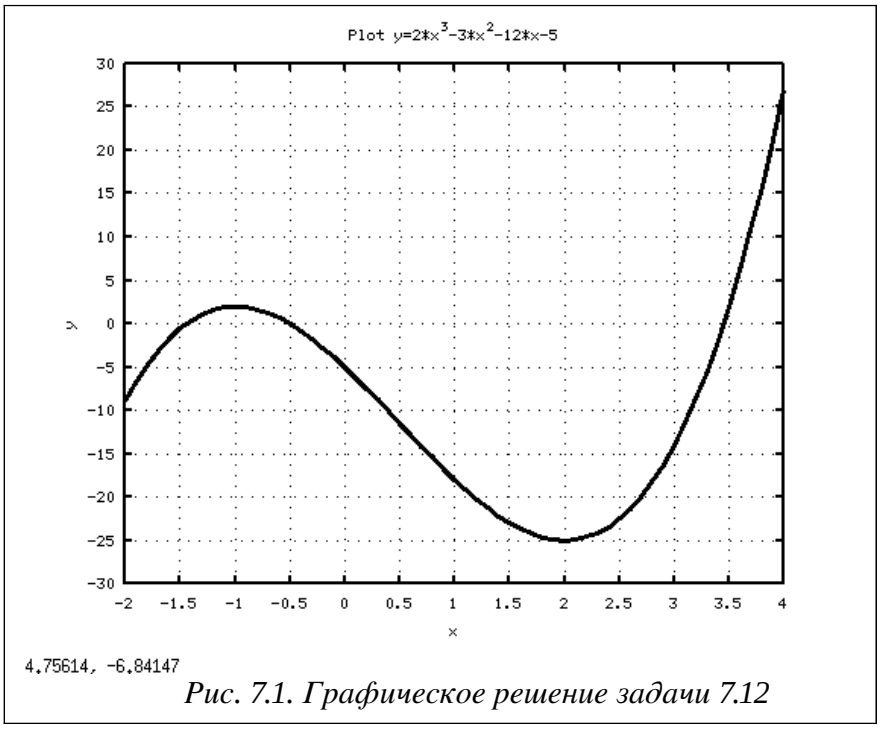


Рис. 7.1. Графическое решение задачи 7.12

ЗАДАЧА 7.13. Найти решение уравнения $x^4+4x^3+4x^2-9=0$.

Графическое решение задачи было получено при помощи последовательности команд приведенных в листинге 7.16.

```
okno1=figure();
x=-4:0.1:2;
y=x.^4+4*x.^3+4*x.^2-9;
cla;
pol=plot(x,y);
set(pol,'LineWidth',3,'Color','k')
set(gca,'xlim',[-4,2]);
set(gca,'ylim',[-10,5]);
set(gca,'xtick',[-4:0.5:2]);
set(gca,'ytick',[-10:1:5]);
grid on;
xlabel('x');
ylabel('y');
title('Plot y=x^4+4*x^3+4*x^2-9');
```

Листинг 7.16

На рис. 7.2 видно, что заданное алгебраическое уравнение имеет два действительных корня. Решение задачи, представленное в листинге 7.17 показывает не только действительные, но и комплексные корни.

```
p=[1 4 4 0 -9];
x=roots(p)
```

```
>>>x =
  -3.00000 + 0.00000i
  -1.00000 + 1.41421i
  -1.00000 - 1.41421i
   1.00000 + 0.00000i
```

Листинг 7.17

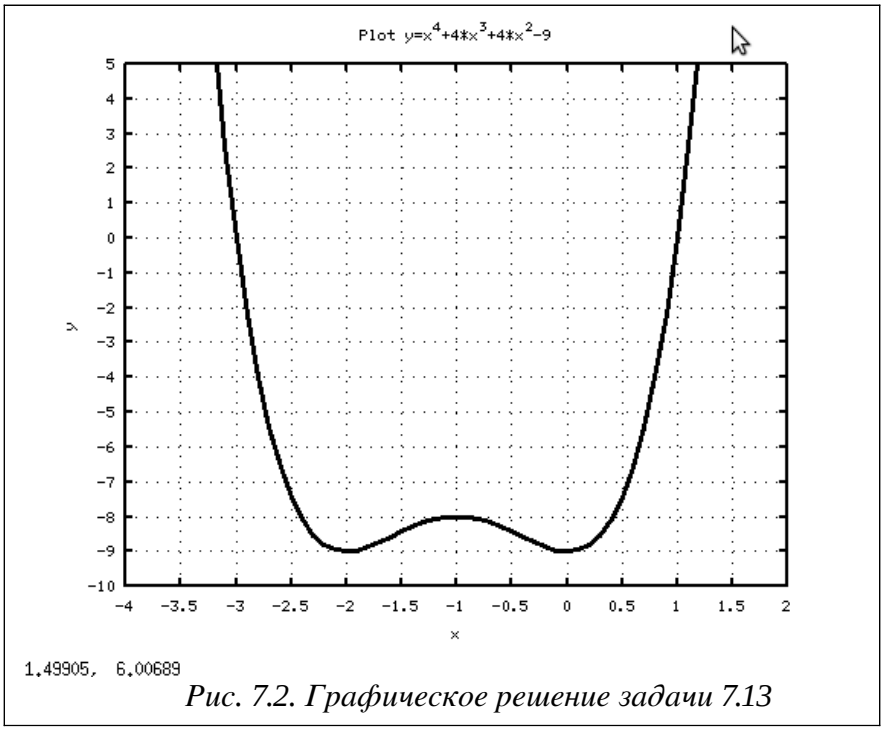


Рис. 7.2. Графическое решение задачи 7.13

7.3 Решение трансцендентных уравнений

Уравнение, в котором неизвестное входит в аргумент трансцендентных функций, называется *трансцендентным уравнением*. К трансцендентным уравнениям принадлежат показательные, логарифмические, тригонометрические [9].

Для *решения трансцендентных уравнений* вида $f(x)=0$ в Octave существует функция

`fzero(name, x0)` или `fzero(name, [a, b])`,

где `name` – имя функции, вычисляющей левую часть уравнения, `x0` – начальное приближение к корню или интервал изоляции корня `[a, b]`.

Если функция вызывается в формате:

`[x, y]= fzero(name, x0)`,

то здесь `x` – корень уравнения, `y` – значение функции в точке `x`.

ЗАДАЧА 7.14. Найти решение уравнения:

$$\sqrt[3]{(2x-3)^2}-\sqrt[3]{(x-1)^2}=0 \quad .$$

Начнем решение данного трансцендентного уравнения с определения интервала изоляции корня. Воспользуемся для этого графическим методом. Построим график функции, указанной в левой части уравнения (листинга 7.18), создав предварительно функцию для ее определения.

```
%Функция для вычисления левой части уравнения f(x)=0
function y=f1(x)
      y=((2*x-3).^2).^(1/3)-((x-1).^2).^(1/3);
end;
```

```
%Построение графика функции f(x)
okno1=figure();
x=-1:0.1:3;
y=f1(x);
cla;
pol=plot(x,y);
set(pol,'LineWidth',3,'Color','k')
set(gca,'xlim',[-1,3]);
set(gca,'ylim',[-1,1.5]);
set(gca,'xtick',[-1:0.5:3]);
set(gca,'ytick',[-1:0.5:1.5]);
grid on;
xlabel('x');
ylabel('y');
title('Plot y=(2x-3)^(2/3)-(x-1)^(2/3)');
```

Листинг 7.18

На графике (рис. 7.3) видно, что функция *f*(*x*) дважды пересекает ось Ox . Первый раз на интервале $[1,1.5]$, второй - $[1.5,2.5]$.

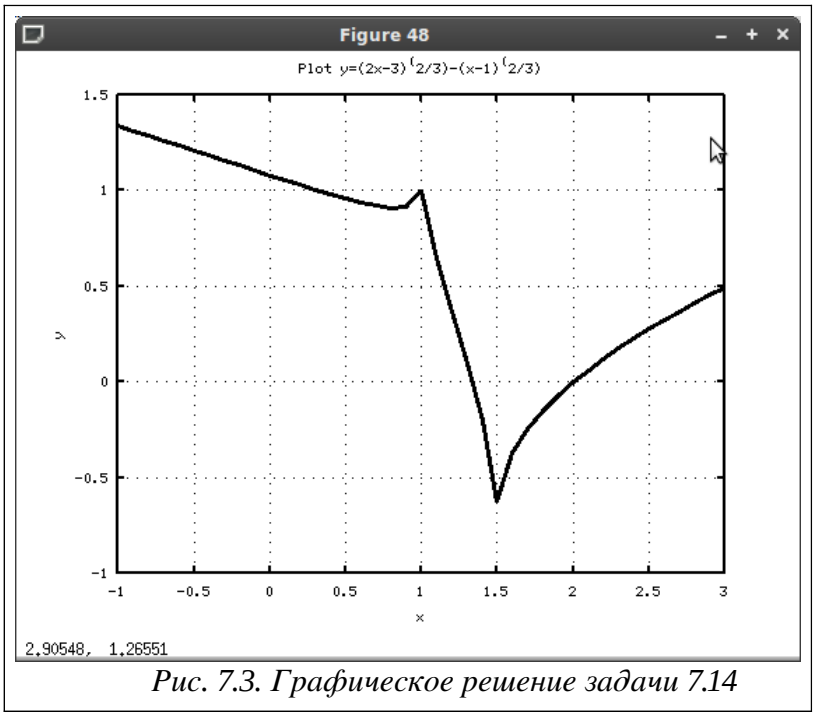


Рис. 7.3. Графическое решение задачи 7.14

Уточним корни, полученные графическим методом. Создадим функцию, вычисляющую левую часть заданного уравнения (листинг 7.18) и обратимся к функции `fzero`, указав в качестве параметров имя созданной функции и число близкое к первому корню:

```
x1=fzero('f1', 1)
>>x1=1.3333
```

Листинг 7.19

Теперь применим функцию `fzero`, указав в качестве параметров имя функции, вычисляющей левую часть тождества и интервал изоляции второго корня:

```
x2=fzero('f1', [1.5 2.5])
>>x1= 2
```

Листинг 7.20

Не трудно заметить, что и в первом и во втором случае функция `fzero` правильно нашла корни заданного уравнения.

Листинг 7.21 содержит пример некорректного обращения к функции `fzero`, здесь интервал изоляции корня задан неверно. На графике видно, что на концах этого интервала функция знак не меняет. Или, другими словами, выбранный интервал содержит сразу два корня.

```
fzero('f1', [1 3])
>>>error: fzero: not a valid initial bracketing
error: called from:
error:       /usr/share/octave/3.2.3/m/optimization/fzero.m  at
line 137, column 5
```

Листинг 7.21

В листинге 7.22 приведен пример обращения к функции `fzero` в полном формате:

```
[X(1),Y(1)]=fzero('f1', [1 1.5]);
[X(2),Y(2)]=fzero('f1', [1.5 2.5]);
X
Y
%Решение уравнения
>>>X =
   1.3333   2.0000
%Значения функции в точке X
>>>Y =
  -2.3870e-15   0.0000e+00
```

Листинг 7.22

ЗАДАЧА 7.15. Найти решение уравнения $x^4+4x^3+4x^2-9=0$.

Как видим, левая часть уравнения представляет собой полином. В задаче 7.13 было показано, что данное уравнение имеет четыре корня: два действительных и два мнимых (листинг 7.17).

Листинге 7.23 демонстрирует решение алгебраического уравнения при помощи функции `fzero`. Не трудно заметить, что результатом работы функции являются только действительные корни. Графическое решение (рис. 7.2) подтверждает это: функция дважды пересекает ось абсцисс.

```
function y=f2(x)
      y=x.^4+4*x.^3+4*x.^2-9;
end;
[X(1),Y(1)]=fzero('f2', [-4 -2]);
[X(2),Y(2)]=fzero('f2', [0 2]);
X
Y
>>>X =
  -3   1
>>>Y =
   0   0
```

Листинг 7.23

7.4 Решение систем нелинейных уравнений

Напомним, что если заданы *m* уравнений с *n* неизвестными и требуется найти последовательность из *n* чисел, которые одновременно удовлетворяют каждому из *m* уравнений, то говорят о *системе уравнений* [9].

Несложную систему элементарной подстановкой можно привести к нелинейному

уравнению. Рассмотрим несколько примеров, в которых описан такой прием решения нелинейной системы.

ЗАДАЧА 7.16. Решить систему уравнений:

$$\begin{cases} x^2+2y^2=1 \\ x-y=1 \end{cases}$$

Найдем графическое решение с помощью команд листинга 7.24. На рис.7.4 видно, что система имеет два решения.

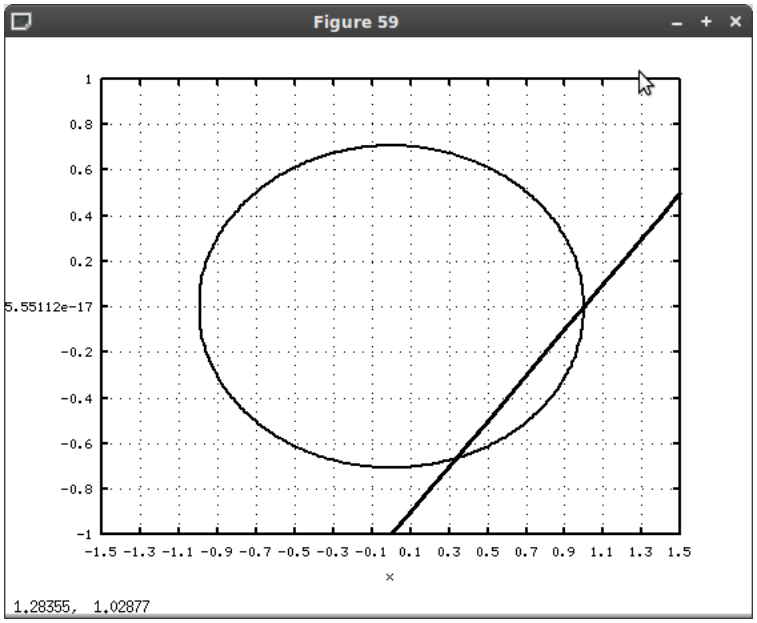


Рис. 7.4. Решение системы нелинейных уравнений из задачи 7.16

```
%Верхняя часть эллипса
function y=f1(x)
      y=sqrt((1-x.^2)/2);
end;
%Нижняя часть эллипса
function y=f2(x)
      y=-sqrt((1-x.^2)/2);
end;
%Прямая
function y=f3(x)
      y=x-1;
end;
%Построение графика
okno1=figure();
x1=-1:0.01:1;
x2=-1.5:0.1:1.5;
cla;
L1=plot(x1,f1(x1),x1,f2(x1));
set(L1,'LineWidth',2,'Color','k')
hold on
L2=plot(x2,f3(x2));
set(L2,'LineWidth',3,'Color','k')
set(gca,'xlim',[-1.5,1.5]);
```

```
set(gca,'ylim',[-1,1]);
set(gca,'xtick',[-1.5:0.2:1.5]);
set(gca,'ytick',[-1:0.2:1]);
grid on;
xlabel('x');
ylabel('y');
```

Листинг 7.24

Не сложно убедиться, что данная система легко сводится к одному уравнению:

$$\{y=x-1, x^2+2y^2=1\} \Rightarrow x^2+2(x-1)^2-1=0 \Rightarrow 3x^2-4x+1=0 \quad .$$

Решив это уравнение с помощью функции `roots`, найдем значения x. Затем подставим их в одно из уравнений системы, например во второе, и тем самым вычислим значения y. Листинг 7.25 содержит решение данной задачи. Понятно, что система имеет два решения $x_1=1, y_1=0$ и $x_2=0.333, y_1=-0.666$ (рис. 7.4). Графическое решение уравнения (листинг 7.26), к которому сводится система, показано на рис. 7.5.

```
>>> p=[3 -4 1];
x=roots(p)
y=x-1
>>>x =
   1.00000
   0.33333
>>>y =
  -1.1102e-16
  -6.6667e-01
```

Листинг 7.25

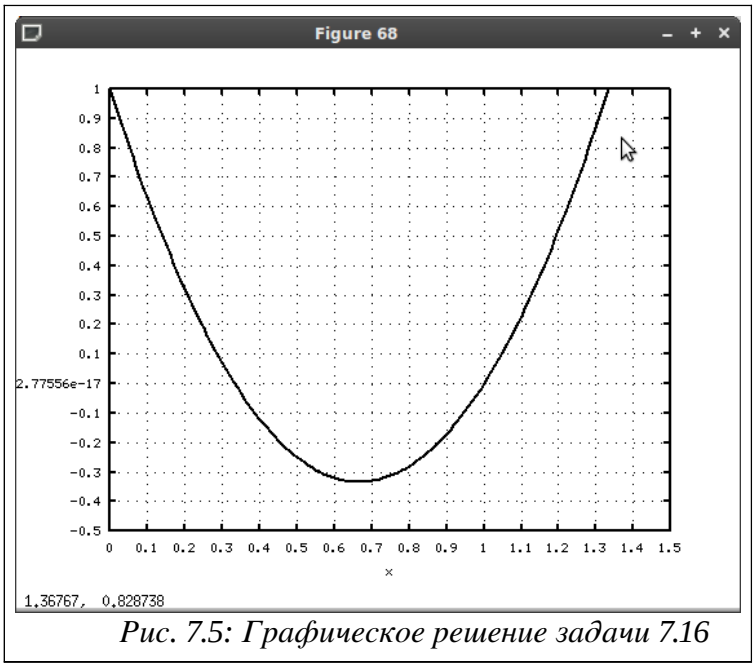


Рис. 7.5: Графическое решение задачи 7.16

```
function y=f(x)
      y=3*x.^2-4*x+1;
end;
okno1=figure();
x=0:0.01:1.5;
cla;
L=plot(x,f(x));
set(L,'LineWidth',2,'Color','k')
```

```
set(gca,'xlim',[0,1.5]);
set(gca,'ylim',[-0.5,1]);
set(gca,'xtick',[0:0.1:1.5]);
set(gca,'ytick',[-0.5:0.1:1]);
grid on;
xlabel('x');
ylabel('y');
```

Листинг 7.26

ЗАДАЧА 7.17. Решить систему уравнений:

$$\begin{cases} \sin(x+1)-y=1.2 \\ 2x+\cos(y)=2 \end{cases}.$$

Проведем элементарные алгебраические преобразования и представим систему в виде одного уравнения:

$$\{y=\sin(x+1)-1.2,\ 2x+\cos(y)-2=0\} \Rightarrow 2x+\cos(\sin(x+1)-1.2)-2=0$$

Рис. 7.6 содержит графическое решение уравнения (листинг 7.27), к которому сводится система, его удобно использовать для выбора начального приближения функции `fzero`.

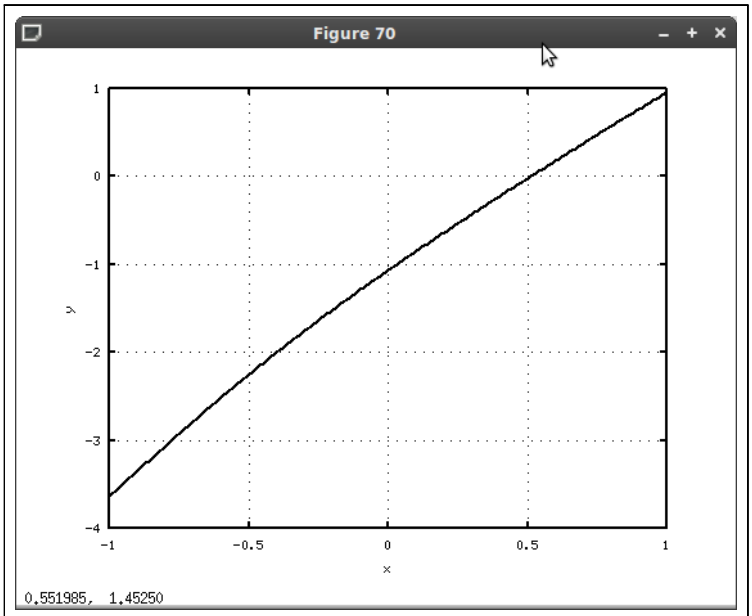


Рис. 7.6. Графическое решение задачи 7.17

```
function y=fun(x)
z=sin(x+1)-1.2;
y=2*x+cos(z)-2;
end;
okno1=figure();
x=-1:0.01:1;
cla;
L=plot(x,fun(x));
set(L,'LineWidth',2,'Color','k');
grid on;
xlabel('x');
ylabel('y');
```

Листинг 7.27

Листинг 7.28 содержит решение заданной системы. Здесь значение x вычисляется при помощи функции `fzero`, а значение y определяется из первого уравнения системы.

```
>>>%Решение системы
X=fzero('fun',0)
Y=sin(X+1)-1.2
>>>X =  0.51015
>>>Y = -0.20184
```

Листинг 7.28

*Решить систему нелинейных уравнени*й, или одно нелинейное уравнение в Octave можно с помощью функции

`fsolve(fun, x0),`

где `fun` – имя функции, которая определяет левую часть уравнения $f(x)=0$ или системы уравнений $\mathbf{F(x)=0}$ (она должна принимать на входе вектор аргументов и возвращать вектор значений), `x0` – вектор приближений, относительно которого будет осуществляться поиск решения.

ЗАДАЧА 7.18. Найти решение системы нелинейных уравнений:

$$\begin{cases} \cos(x)+2y=2 \\ \dfrac{x^2}{3}-\dfrac{y^2}{3}=1 \end{cases}.$$

Решим систему графически, для чего выполним перечень команд указанных в листинге 7.29. Результат работы этих команд показан на рис. 7.7. Понятно, что система имеет два корня.

```
%Уравнения, описывающие линии гиперболы
function y=f1(x)
 y=sqrt(x.^2-3);
end;
function y=f2(x)
 y=-sqrt(x.^2-3);
end;
%Уравнение косинусоиды
function y=f3(x)
 y=1-cos(x)/2;
end;
%Построение графика
okno1=figure();
x1=-5:0.001:-sqrt(3);
x2=sqrt(3):0.001:5;
x3=-5:0.1:5;
cla;
%Гипербола
L1=plot(x1,f1(x1),x1,f2(x1),x2,f1(x2),x2,f2(x2));
set(L1,'LineWidth',3,'Color','k')
hold on
%Косинусоида
L2=plot(x3,f3(x3));
set(L2,'LineWidth',3,'Color','k')
grid on;
xlabel('x');
ylabel('y');
```

Листинг 7.29

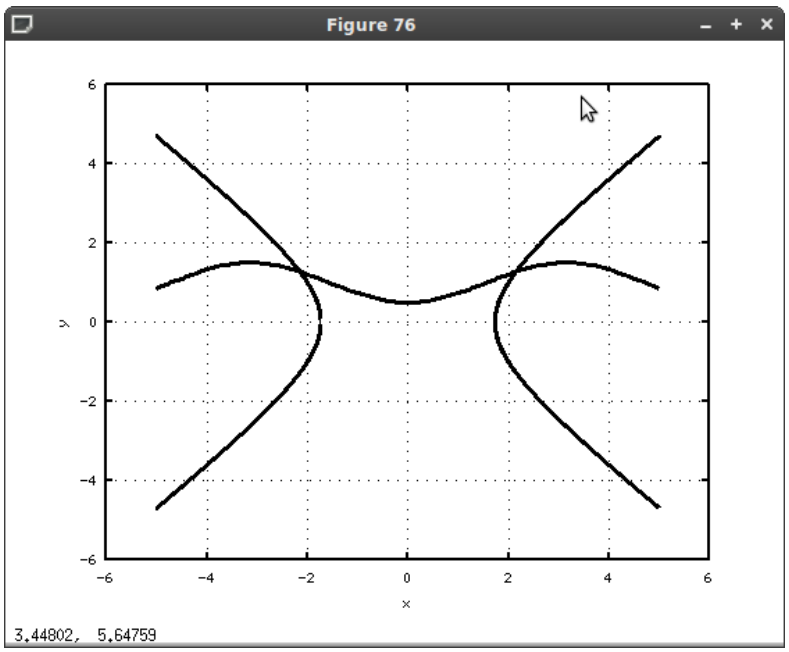


Рис. 7.7. Графическое решение задачи 7.18

Составим функцию, соответствующую левой части системы (листинг 7.30). Здесь важно помнить, что все уравнения должны иметь вид $F(x)=0$. Кроме того, обратите внимание, что `x` и `y` в этой функции — векторы (`x` — вектор неизвестных, `y` — вектор решений).

```
function [y]=fun(x)
      y(1)=cos(x(1))+2*x(2)-2;
      y(2)=x(1)^2/3-x(2)^2/3-1;
end;
```

Листинг 7.30

Теперь решим систему (листинг 7.31), указав в качестве начального приближения в начале вектор $[-3,-1]$, затем $[1,3]$.

```
[X1_Y1]=fsolve('fun', [-3 -1])
[X2_Y2]=fsolve('fun', [1 3])
>>>X1_Y1 =
  -2.1499   1.2736
>>>X2_Y2 =
   2.1499   1.2736
```

Листинг 7.31

Понятно, что решением задачи являются пары $x_1=-2.15$, $y_1=1.27$ и $x_2=2.15$, $y_2=1.27$, что соответствует графическому решению (рис. 7.7).

Если функция решения нелинейных уравнений и систем имеет вид

```
[x, f, ex] = fsolve(fun, x0)
```

то здесь `x` – вектор решений системы, `f` – вектор значений уравнений системы для найденного значения `x`, `ex` – признак завершения алгоритма решения нелинейной системы, отрицательное значение параметра `ex` означает, что решение не найдено, ноль – досрочное прерывание вычислительного процесса при достижении максимально допустимого числа итераций, положительное значение подтверждает, что решение найдено с заданной точностью.

ЗАДАЧА 7.19. Решить систему нелинейных уравнений:

$$\begin{cases} x_1^2+x_2^2+x_3^2=1 \\ 2x_1^2+x_2^2-4x_3=0 \\ 3x_1^2-4x_2^2+x_3^2=0 \end{cases}.$$

Листинг 7.32 содержит функцию заданной системы и ее решение. Обратите внимание на выходные параметры функции `fsolve`. В нашем случае значения функции `f` для найденного решения `x` близки к нулю и признак завершения `ex` положительный, значит, найдено верное решение.

```
>>> function f=Y(x)
f(1)=x(1)^2+x(2)^2+x(3)^2-1;
f(2)=2*x(1)^2+x(2)^2-4*x(3);
f(3)=3*x(1)^2-4*x(2)+x(3)^2;
end
[x,f,ex]=fsolve('Y',[0.5 0.5 0.5])
>>>x =
   0.78520   0.49661   0.36992
f =
   1.7571e-08   3.5199e-08   5.2791e-08
ex =  1
```

Листинг 7.32

ЗАДАЧА 7.20. Решить систему:

$$\begin{cases} x^2+y^2=1 \\ 2\sin(x-1)+y=1 \end{cases}.$$

Графическое решение системы (рис. 7.8) показало, что она корней не имеет. Рисунок был получен с помощью команд листинга 7.33.

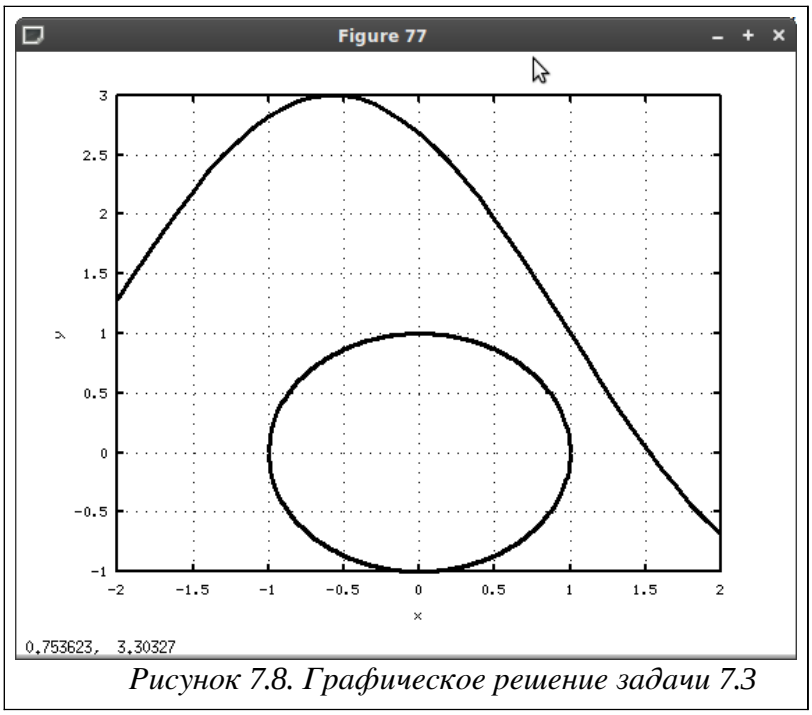


Рисунок 7.8. Графическое решение задачи 7.3

```
%Уравнения линий окружности
function y=f1(x)
 y=sqrt(1-x.^2);
end;
```

```
function y=f2(x)
 y=-sqrt(1-x.^2);
end;
%Уравнение синусоиды
function y=f3(x)
 y=1-2*sin(x-1);
end;
okno1=figure();
x1=-1:0.01:1;
x3=-2:0.1:2;
cla;
%Окружность
L1=plot(x1,f1(x1),x1,f2(x1));
set(L1,'LineWidth',3,'Color','k')
hold on
%Синусоида
L2=plot(x3,f3(x3));
set(L2,'LineWidth',3,'Color','k')
grid on;
xlabel('x');
ylabel('y');
```

Листинг 7.33

Однако применение к системе функции `fsolve` дает положительный ответ, что видно из листинга 7.34. Происходит это потому, что алгоритм, реализованный в этой функции, основан на минимизации суммы квадратов компонент вектор–функции. Следовательно, функция `fsolve` в этом случае нашла точку минимума, а наличие точки минимума не гарантирует существование корней системы в ее окрестности.

```
function [y]=fun(x)
 y(1)=x(1)^2+x(2)^2-1;
 y(2)=2*sin(x(1)-1)+x(2)-1;
end;
[X1_Y1]=fsolve('fun', [1 1])
>>>X1_Y1 =
   1.04584   0.52342
```

Листинг 7.34

7.5 Решение нелинейных уравнений и систем в символьных переменных

Напомним, что для работы с символьными переменными в Octave должен быть подключен специальный пакет расширений **octave-symbolic**. Процедура установки пакетов расширений описана в первой главе. Техника работы с символьными переменными описана в п. 2.7 второй главы.

Для *решения системы нелинейных уравнений* или одного *нелинейного уравнения* можно воспользоваться функцией `symsolve`.

ЗАДАЧА 7.21. Решить уравнение $\frac{e^x}{5}-3(2x-1)=0$.

Команды, с помощью которых выполнено графическое (рис. 7.9) и аналитическое решение представлены в листинге 7.35.

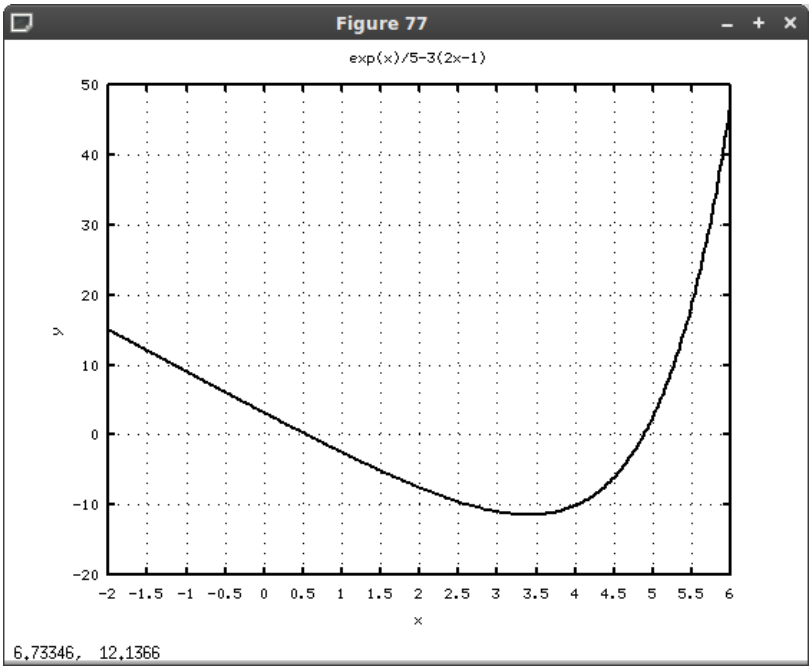


Рис. 7.9. Графическое решение задачи 7.21

```
clear all;
clf;
cla;
symbols
x=sym("x");
y=Exp(x)/5-3*(2*x-1);
L=ezplot('exp(x)/5-3*(2*x-1)');
set(L,'LineWidth',2,'Color','k')
set(gca,'xlim',[-2,6]);
set(gca,'ylim',[-20,50]);
set(gca,'xtick',[-2:0.5:6]);
set(gca,'ytick',[-20:10:50]);
grid on;
xlabel('x');
ylabel('y');
q1 = symfsolve(y,0)
q2 = symfsolve(y,4)
>>>q1 =  0.55825
>>>q2 =  4.8777
```

Листинг 7.35

ЗАДАЧА 7.22. Решить систему:

$$\begin{cases} x^2+y^2+3x-2y=4 \\ x+2y=5 \end{cases}.$$

Решение системы (листинг 7.36) показало, что она имеет два корня $x_1=1, y_1=2$ и $x_2=-2.2, y_2=3.6$, что соответствует графическому решению (рис. 7.10).

```
clear all;
clf;
cla;
symbols
x=sym("x");
```

```
y=sym("y");
L1=ezplot('x^2+y^2+3*x-2*y-4');
set(L1,'LineWidth',2,'Color','k')
hold on
L2=ezplot('x+2*y-5');
set(L2,'LineWidth',2,'Color','k')
set(gca,'xlim',[-5,4]);
set(gca,'ylim',[-2,5]);
set(gca,'xtick',[-5:0.5:4]);
set(gca,'ytick',[-2:0.5:5]);
grid on;
xlabel('x');
ylabel('y');
title('x^2+y^2+3x-2y=4, x+2y=5')
f1=x^2+y^2+3*x-2*y-4;
f2=x+2*y-5;
q1 = symfsolve(f1,f2,{x==0,y==1})
q2 = symfsolve(f1,f2,{x==-1,y==3})
>>>q1 =    1.0000   2.0000
>>>q2 =   -2.2000   3.6000
```

Листинг 7.36

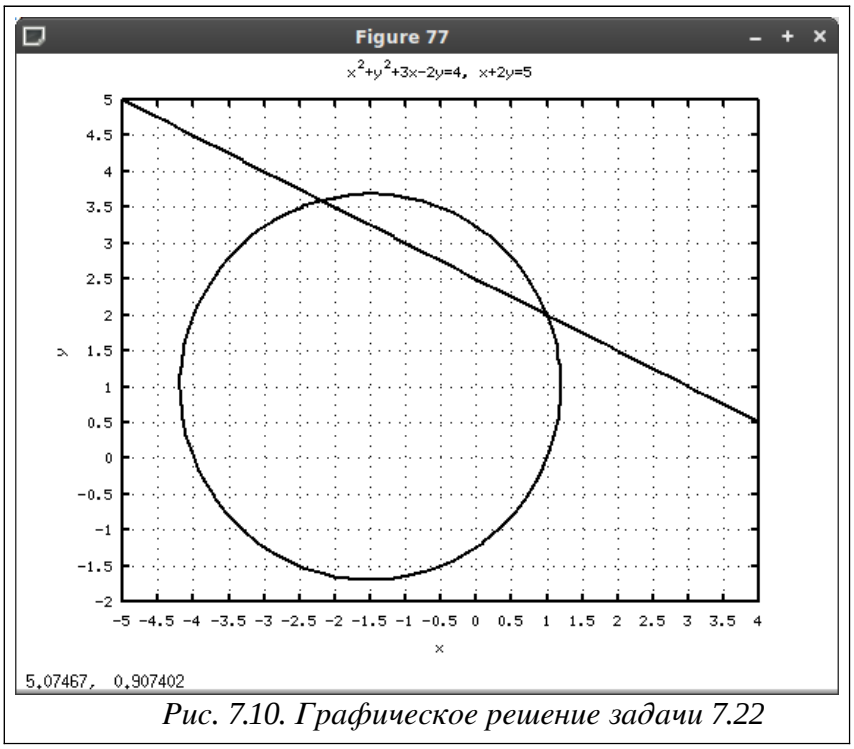


Рис. 7.10. Графическое решение задачи 7.22

8. Интегрирование и дифференцирование

Дифференцирование в Octave осуществляется в технике символьных переменных. В функциях интегрирования реализованы различные численные алгоритмы.

8.1 Вычисление производной

Дифференцирование в Octave осуществляется с помощью функции

```
differentiate(f(a,x [, n]),
```

где `a` — символьное выражение, `x` — переменная дифференцирования, `n` — порядок дифференцирования (при `n=1` параметр можно опустить). Иными словами функция вычисляет `n`-ю производную выражения `a` по переменной `x`.

Напомним, что для работы с символьными переменными в Octave должен быть подключен специальный пакет расширений **octave-symbolic**. Процедура установки пакетов расширений описана в первой главе. Техника работы с символьными переменными описана в п. 2.7 второй главы.

Производной функции $f(x)$ в точке x_0 называется предел, к которому стремится отношение бесконечно малого приращения функции к соответствующему бесконечно малому приращению аргумента. Геометрический смысл этого понятия заключается в том, что если к графику функции $f(x)$ провести касательную в точке x_0, то ее угловой коэффициент, будет равен значению производной в этой точке $k=f'(x)$. Следовательно, уравнение касательной к линии в заданной точке имеет вид [7]:

$$y(x)=f'(x)(x-x_0)+f(x_0) \ .$$

ЗАДАЧА 8.1. Записать уравнение касательной к функции $f(x)=\frac{3x^2-7}{2x+1}$ в точке $x_0=-1$.

Из листинга 8.1 видим, что уравнение касательной к функции в заданной точке имеет вид $y(x)=14x+18$.

```
clear all;
x0=-1;
symbols
x = sym ("x");
f=(3*x^2-7)/(2*x+1);
%Первая производная от заданной функции
f1=differentiate(f,x)
%Уравнение касательной:
y=subs(f1,x,x0)*(x-x0)+subs(f,x,x0)
>>>f1 =
(6.0)*x*(1.0+(2.0)*x)^(-1)-
           (2.0)*(-7.0+(3.0)*x^(2.0))*(1.0+(2.0)*x)^(-2)
>>>y =
18.0-9.029803704631804845E-19*I+
                      (14.0-6.019869136421203297E-19*I)*x
```

Листинг 8.1

На рис. 8.1 представлены графики заданной функции и ее касательной. Рисунок построен с помощью команд из листинга 8.2.

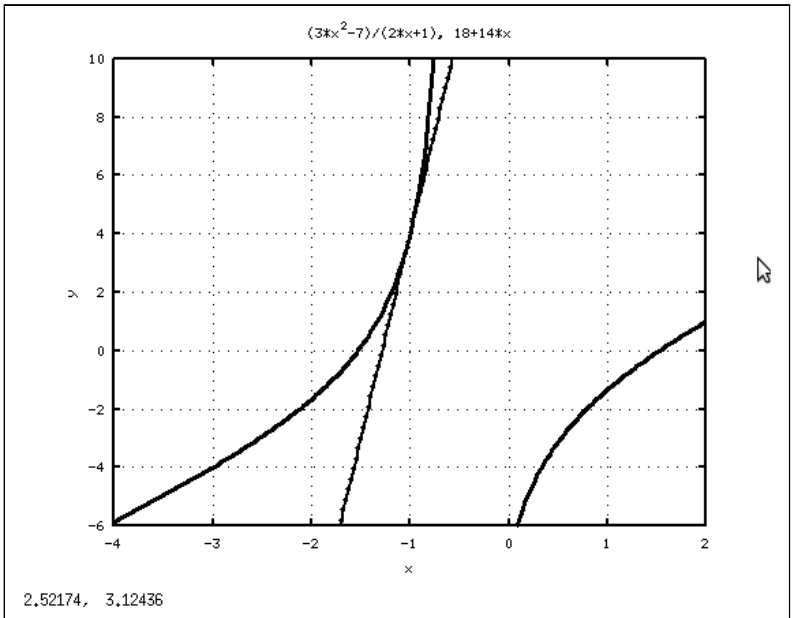


Рис. 8.1. График функции и ее касательной

```
clear all;
clf;cla;
symbols
x=sym("x");
L1=ezplot('(3*x^2-7)/(2*x+1)');
set(L1,'LineWidth',3,'Color','k')
hold on
L2=ezplot('18+14*x');
set(L2,'LineWidth',2,'Color','k','Marker','o')
set(gca,'xlim',[-4,2]);
set(gca,'ylim',[-6,10]);
set(gca,'xtick',[-4:2]);
set(gca,'ytick',[-6:2:10]);
grid on;
xlabel('x');
ylabel('y');
title('(3*x^2-7)/(2*x+1), 18+14*x');
```

Листинг 8.2

ЗАДАЧА 8.2. Найти $f'(x)=\frac{5\sin(2x)}{\sqrt{\cos(2x)}}$ и $f'(x)=tg(\sqrt[3]{\ln(x)})$.

Решение задачи показано в листингах 8.3 и 8.4 соответственно.

```
clear all;
symbols
x = sym ("x") ;
f=(5*Sin(2*x))/Sqrt(Cos(2*x));
f1=differentiate(f,x)
>>>f1 =
(5.0)*sin((2.0)*x)^2*cos((2.0)*x)^(-3/2)+
                                (10.0)*sqrt(cos((2.0)*x))
```

Листинг 8.3

```
clear all;
symbols
x = sym ("x") ;
f=Tan(Log(x)^(1/3));
f1=differentiate(f,x)
>>>f1 =
(0.333)*(1+tan(log(x)^(0.333))^2)*x^(-1)*log(x)^(-0.666)
```

Листинг 8.4

Если функция $y(x)$ задана параметрическими уравнениями $x=\phi(t), y=\psi(t)$, то производная вычисляется по формуле $y'(x)=\frac{\phi'(t)}{\psi'(t)}=\frac{y_t}{x_t}$ [7].

ЗАДАЧА 8.3. Найти производную функции, заданной параметрически:

$$\begin{cases} x(t)=3\cos^3(t) \\ y(t)=3\sin^3(t) \end{cases}.$$

Листинг 8.5 содержит решение задачи.

```
clear all;
symbols
t = sym ("t") ;
x=3*Cos(t)^3;
y=3*Sin(t)^3;
xt=differentiate(x,t);
yt=differentiate(y,t);
f=yt/xt
>>>f =
-sin(t)*cos(t)^(-1.0)
```

Листинг 8.5

ЗАДАЧА 8.4. Найти производные $f''(x)=\ln(\cos(x))$ (листинг 8.6), $f^{IV}(x)=\tan(x)$ (листинг 8.7).

```
>>> clear all;
symbols
x = sym ("x") ;
f=Log(Cos(x));
differentiate(f,x,2)
>>>ans =
-1-cos(x)^(-2)*sin(x)^2
```

Листинг 8.6

```
>>> clear all;
symbols
x = sym ("x") ;
f=Tan(x);
differentiate(f,x,4)
>>>ans =
16*(1+tan(x)^2)^2*tan(x)+8*(1+tan(x)^2)*tan(x)^3
```

Листинг 8.7

ЗАДАЧА 8.5. Найти производную $y''(x)$ функции, заданной параметрически

$$\begin{cases} x(t)=t-\sin(t) \\ y(t)=1-\cos(t) \end{cases}.$$

Выражение для вычисления второй производной параметрической функции:

$$y''(x)=\left(\frac{\phi'(t)}{\psi'(t)}\right)'=\frac{\phi'(t)\psi''(t)-\psi'(t)\phi''(t)}{(\phi'(t))^2}\ .$$

В листинге 8.8 представлено решение задачи

```
>>> clear all;
symbols
t = sym ("t") ;
x=t-Sin(t);
y=1-Cos(t);
xt=differentiate(x,t);
yt=differentiate(y,t);
xt2=differentiate(x,t,2);
yt2=differentiate(y,t,2);
f=(xt*yt2-yt*xt2)/xt^2
>>>f =
-(1-cos(t))^(-2.0)*(sin(t)^2+(-1+cos(t))*cos(t))
```

Листинг 8.8

8.2 Применение производной

Понятие производной тесно связано с *задачей исследования функции*. Дадим несколько определений.

Если производная функции $f(x)$ положительна на всем интервале $[a,b]$, то функция на нем *возрастает*, если всюду отрицательна, то $f(x)$ *убывает*.

ЗАДАЧА 8.6. Построить график функции $f(x)=1-2x-x^2$ и ее производной. Исследовать функцию на возрастание и убывание.

Вычислим производную заданной функции:

```
symbols
x=sym("x");
f=1-2*x-x^2;
differentiate(f,x)
>>>ans =
-2.0-(2.0)*x
```

Листинг 8.9

С помощью команд листинга 8.10 построим график заданной функции и ее производной.

```
clf;
cla;
L1=ezplot('1-2*x-x^2');
set(L1,'LineWidth',3,'Color','k')
hold on
L2=ezplot('-2-2*x');
set(L2,'LineWidth',2,'Color','k')
grid on;
xlabel('x');
ylabel('y');
title('1-2x-x^2, -2-2x')
```

Листинг 8.10

На рис. 8.2 видим, что там, где $y=f'(x)$ принимает положительные значения, $f(x)$ возрастает, соответственно при отрицательных значениях $y=f'(x)$ функция $f(x)$ убывает.

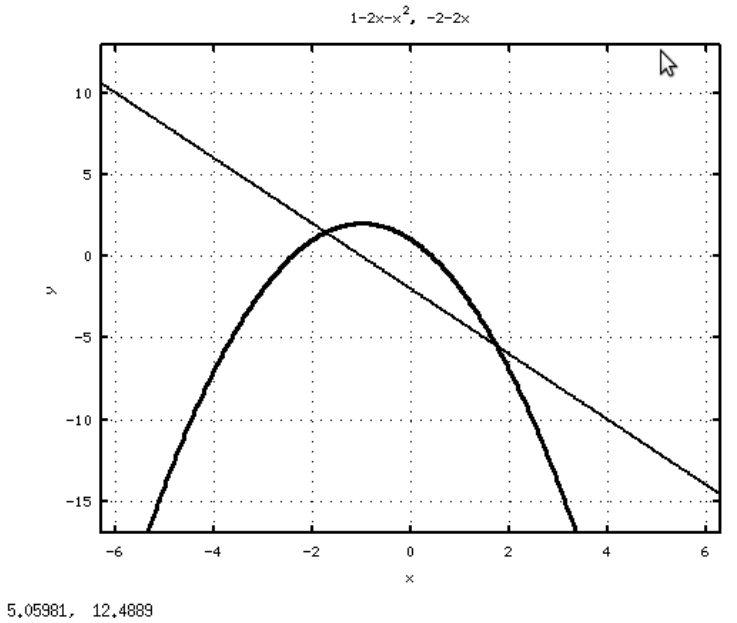


Рис. 8.2. Исследование функции на возрастание и убывание

Говорят, что непрерывная функция $f(x)$ имеет *максимум в точке* $x=a$, если в достаточной близости от этой точки производная $f'(x)$ положительна слева от *a* и отрицательна справа от *a*. Если наоборот, то $f(x)$ имеет *минимум в точке* $x=a$. Максимум и минимум объединяют названием *экстремум*. Если первая производная в этой точке $f'(a)$ либо равна нулю, либо не существует, то в этой точке может быть экстремум [7].

ЗАДАЧА 8.7. Исследовать функцию $f(x)=\frac{x^3}{3}-2x^2+3x+1$ на экстремум.

Найдем производную функции:

```
clear all;
symbols
x=sym("x");
f=x^3/3-2*x^2+3*x+1;
%Производная от функции f(x)
y=differentiate(f,x)
>>>y = 3.0+x^(2.0)-(4.0)*x
```

Листинг 8.11

Изобразим функцию и ее производную на графике и найдем корни уравнения $f'(x)=0$ с помощью команд листинга 8.12.

```
clf;cla;
L1=ezplot('x^3/3-2*x^2+3*x+1');
set(L1,'LineWidth',3,'Color','k')
hold on
L2=ezplot('3.0+x^(2.0)-(4.0)*x');
set(L2,'LineWidth',2,'Color','k')
set(gca,'xlim',[-2,5]);
set(gca,'ylim',[-5,5]);
grid on;
xlabel('x');ylabel('y');
title('x^3/3-2x^2+3x+1, 3+x^2-4x')
```

```
%Корни уравнения  f'(x)=0
x1 = symfsolve(y,1)
x2 = symfsolve(y,3)
>>>x1 =  1
>>>x2 =  3
```

Листинг 8.12

На рис. 8.3 и в листинге 8.12 видно, что первая производная обращается в ноль в точках 1 и 3. При переходе через точку 1 $f'(x)$ меняет знак «+» на «−», следовательно, это точка максимума функции $f(x)$, а в точке 3 знак первой производной меняется с «−» на «+», то есть это точка минимума.

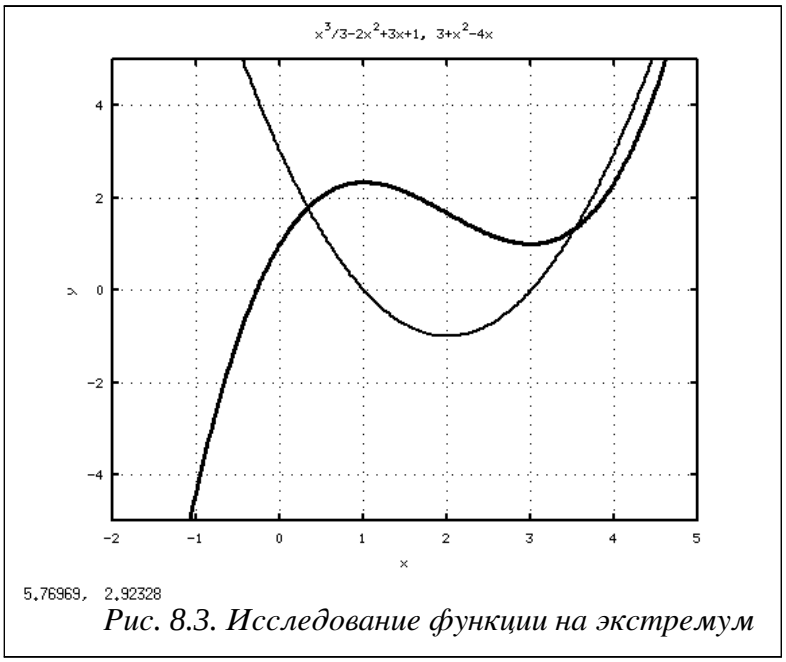


Рис. 8.3. Исследование функции на экстремум

График функции называется *выпуклым на промежутке* $[a,b]$, если он расположен выше касательной, проведенной в любой точке этого интервала. Если же график функции лежит ниже касательной, то он называется *вогнутым*. Функция будет *выпуклой на интервале* $[a,b]$, если вторая производная $f''(x)$ на нем положительна. И наоборот, если вторая производная отрицательна, то *функция вогнута*. Если же вторая производная равна нулю в некоторой точке *a*, а слева и справа от нее имеет значения разных знаков, то точка *a – точка перегиба* [7].

ЗАДАЧА 8.8. Определить точки перегиба функции $f(x)=\dfrac{3x-2}{x^2+1}$.

Найдем вторую производную заданной функции. Построим графики функции и ее второй производной. Определим точки в которых вторая производная обращается в ноль (листинг 8.13).

```
>>> clear all;
symbols
x=sym("x");
f=(3*x-2)/(x^2+1);
y=differentiate(f,x,2)
>>>y =
-(2.0)*(1.0+x^(2.0))^(-2)*(-2.0+(3.0)*x)-
           (12.0)*(1.0+x^(2.0))^(-2)*x+
           (8.0)*(1.0+x^(2.0))^(-3)*x^2*(-2.0+(3.0)*x)
clf;cla;
```

```
L1=ezplot('(3*x-2)/(x^2+1)');
set(L1,'LineWidth',4,'Color','k')
hold on
L2=ezplot('-(2.0)*(1.0+x^(2.0))^(-2)*(-2.0+(3.0)*x)-
           (12.0)*(1.0+x^(2.0))^(-2)*x+
           (8.0)*(1.0+x^(2.0))^(-3)*x^2*(-2.0+(3.0)*x)');
set(L2,'LineWidth',2,'Color','k')
set(gca,'xlim',[-5,5]);
set(gca,'ylim',[-5,7]);
grid on;
xlabel('x');ylabel('y');
title('  ')
x1 = symfsolve(y,-1)
x2 = symfsolve(y,0)
x3 = symfsolve(y,2)
>>>x1 = -1.1411
>>>x2 =  0.19855
>>>x3 =  2.9425
```

Листинг 8.13

Иллюстрации приведены на рис. 8.4. Исследование второй производной функции $f''(x)$ показывает, что она определена на всей числовой оси и обращается в нуль в трех точках $x_1=-1.1411, x_2=0.19855, x_3=2.9425$, причем при переходе через них она меняет знак. Следовательно, на интервале $(-\infty, x_1)$ функция $f(x)$ вогнутая, так как $f''(x)<0$, на (x_1, x_2) – выпуклая ($f''(x)>0$), на (x_2, x_3) – вогнутая ($f''(x)<0$) и на $(x_3, +\infty)$ опять выпуклая, потому что $f''(x)>0$.

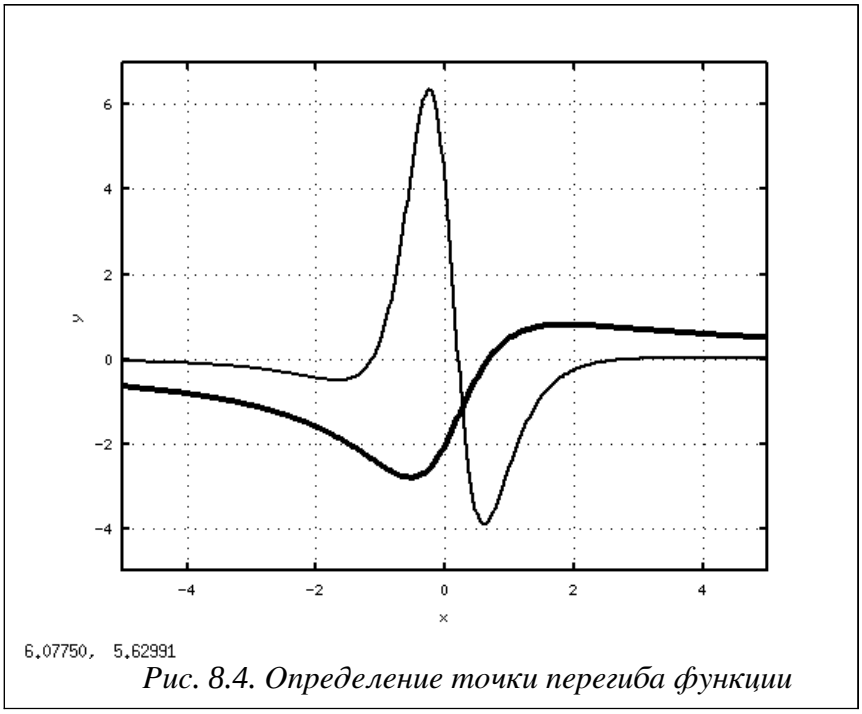


Рис. 8.4. Определение точки перегиба функции

8.3 Численное интегрирование

Пусть дана функция $f(x)$, известно, что она непрерывна на интервале $[a, b]$ и уже определена ее первообразная $F(x)$, тогда *определенный интеграл* от этой функции можно вычислить в пределах от *a* до *b* по *формуле Ньютона–Лейбница* [9]:

$$\int_a^b f(x)\,dx = F(b) - F(a) \quad ,$$

где $F'(x) = f(x)$.

ЗАДАЧА 8.9. Вычислить определенный интеграл $I = \int_2^5 \sqrt{2x-1}$.

К сожалению в Octave не предусмотрены средства символьного интегрирования, поэтому обратимся к таблице интегралов и найдем, что

$$I = \int_2^5 \sqrt{2x-1} = \frac{1}{3}\sqrt[3]{(2x-1)^2} + C \quad .$$

Теперь вычислим интеграл по формуле Ньютона–Лейбница:

```
>>> clear all;
%Функция, определяющая подынтегральное выражение
%x – переменная интегрирования,
%C – постоянная интегрирования.
function y=F(x,C)
y=1/3*(2*x-1)^(3/2)+C;
end;
a=2; b=5;
%Вычисление интеграла по формуле Ньютона-Лейбница
I=F(b,0)-F(a,0)
>>>I =  7.2679
```

Листинг 8.14

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 задается 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 определенного интеграла на основании значений подынтегральной функции.

Численное вычисление определенного интеграла называют *механической квадратурой*. Формулы, соответствующие тому или иному численному методу приближенного интегрирования называют *квадратурными*. Подобное название связано с *геометрическим смыслом определенного интеграла*: значение определенного интеграла

$$y = \int_a^b f(x)\,dx,\ f(x) \neq 0 \quad ,$$

равно площади квадрата, которая в свою очередь, совпадает с площадью криволинейной трапеции с основаниями $[a,b]$ и $f(x)$ [9].

Вообще говоря, классические учебники по численной математике предлагают не мало методов интегрирования, но здесь мы рассмотрим только те методы, которые имеют непосредственное отношение к функциям Octave.

8.3.1 Интегрирование по методу трапеций

Изложим геометрическую интерпретацию *интегрирования по методу трапеций*. Для этого участок интегрирования $[a,b]$ разобьем точками на *n* равных частей (рис. 8.5), причем $x_0 = a, x_n = b$.

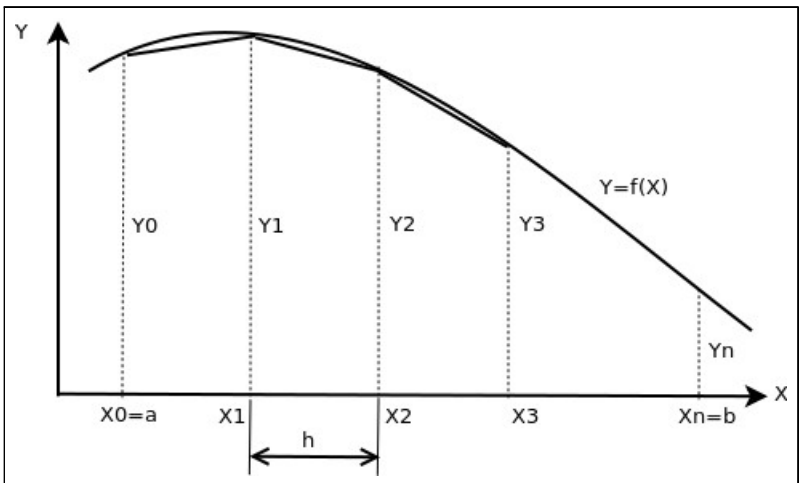


Рис. 8.5. Геометрическая интерпретация метода трапеций

Тогда длина каждой части будет равна $h=\frac{b-a}{n}$, а значение абсциссы каждой из точек разбиения можно вычислить по формуле $x_i=x_0+i\,h, i=1,n-1$. Теперь из каждой точки x_i проведем перпендикуляр до пересечения с кривой $f(x)$, а затем заменим каждую из полученных криволинейных трапеций прямолинейной. Приближенное значение интеграла будем рассматривать как сумму площадей прямолинейных трапеций, причем площадь отдельной трапеции составляет $S=\frac{y_{i-1}+y_i}{2}h$, следовательно, площадь искомой фигуры вычисляют по формуле:

$$S=\int_a^b f(x)\,dx=\sum_{i=1}^{n} S_i=\frac{h}{2}\sum_{i=1}^{n}(y_{i-1}-y_i)=h\frac{y_0+y_n}{2}+\sum_{i=1}^{n-1} y_i \quad .$$

Таким образом, получена *квадратурная формула трапеций* для численного интегрирования [9]:

$$I=\int_a^b f(x)\,dx=h\frac{f(a)+f(b)}{2}+\int_{i=1}^{n-1} f(x_i) \quad .$$

Функций `trapz` и `cumtrapz` реализуют численное интегрирование по методу трапеций в Octave.

Площадь фигуры под графиком функции $y(x)$, в котором все точки заданы векторами `x` и `y`, вычисляет команда `trapz(x, y)`.

Если вызвать функцию `trapz(y)` с одним аргументом, то будет вычислена площадь фигуры под графиком функции $y(x)$, в котором все точки заданы векторами `x` и `y`, причем по умолчанию элементы вектора `x` принимают значения номеров элементов вектора `y`.

ЗАДАЧА 8.10. Вычислить интеграл от функции $y(x)=\cos(x)$. Значения функции представлены в табл. 8.1.

Таблица 8.1. Значения функции $y(x)=\cos(x)$

x	-1.5708	-1.0708	-0.5708	-0.0708	0.4292	0.9292	01.01.92
y	0	0.47943	0.84147	0.99749	0.90930	0.59847	0.14112

Решение задачи представлено в листинге 8.15.

```
>>> I=trapz(x, y)
I =  1.9484
```

```
>>> clear all;
x=[-1.5708 -1.0708 -0.5708 -0.0708 0.4292 0.9292 1.4292];
y=[0 0.47943 0.84147 0.99749 0.90930 0.59847 0.14112];
I=trapz(x, y)
>>>I =  1.9484
```

Листинг 8.15

ЗАДАЧА 8.11. Вычислить интеграл $I=\int_2^5 \sqrt{2x-1}$.

Листинг 8.16 содержит несколько вариантов решения данной задачи. В первом случае интервал интегрирования делится на отрезки с шагом 1, во втором 0.5, в третьем 0.1 и в четвертом 0.05. Не трудно заметить, что чем больше точек разбиения, тем точнее значение искомого интеграла. Решение можно сравнить с результатом полученным в задаче 8.1, где этот же интеграл был найден по формулам Ньютона-Лейбница (листинг 8.14).

```
>>> clear all;
%1. h=1
x=2:5;
y=sqrt(2*x-1);
I1=trapz(x, y)
%2. h=0.5
x=2:0.5:5;
y=sqrt(2*x-1);
I2=trapz(x, y)
%3. h=0.1
x=2:0.1:5;
y=sqrt(2*x-1);
I3=trapz(x, y)
%4. h=0.05
x=2:0.05:5;
y=sqrt(2*x-1);
I4=trapz(x, y)
%Результаты интегрирования
>>>I1 =  7.2478
>>>I2 =  7.2629
>>>I3 =  7.2677
>>>I4 =  7.2679
```

Листинг 8.16

В листинге 8.17 приведен пример использования функции `trapz` с одним аргументом. Как видим, в первом случае значение интеграла вычисленного при помощи этой функции не точно и совпадает со значением, полученным функцией `trapz(x, y)` на интервале $[2,5]$ с шагом 1 (листинг 8.16, первый пример). То есть мы нашли сумму площадей трех прямолинейных трапеций с основанием h=1 и боковыми сторонами, заданными вектором y. Во втором случае, при попытке увеличить точность интегрирования, значение интеграла существенно увеличивается. Дело в том, что уменьшив шаг разбиения интервала интегрирования до 0.05, мы увеличили количество элементов векторов x и y и применение функции `trapz(y)` приведет к вычислению суммы площадей шестидесяти трапеций с основанием h=1 и боковыми сторонами, заданными вектором y. Таким образом, в первом и втором примерах листинга 8.17 вычисляются площади совершенно разных фигур.

```
%1.
x=2:5;
y=sqrt(2*x-1);
```

```
I=trapz(y)
>>>I =  7.2478
%2.
x=2:0.05:5;
y=sqrt(2*x-1);
I=trapz(y)
>>>I =  145.36
```

Листинг 8.17

Функция `cumtrapz` выполняет так называемое «интегрирование с накоплением» по методу трапеций . Это означает, что она, так же как и `trapz`, вычисляет площадь фигуры под графиком функции $y(x)$, но результатом ее работы является вектор, состоящий из промежуточных вычислений. То есть, если общая площадь S криволинейной трапеции сформирована из суммы площадей $S_i(i=1,n)$ прямолинейных трапеций, то элементы вектора представляют собой следующую последовательность $S_1=0, S_2=S_1+S_2, S_3=S_1+S_2+S_3, \ldots, S_n=S_1+S_2+S_3+\ldots+S_n$. Таким образом, последний элемент вектора будет равен искомой площади фигуры S . Функцию интегрирования с накоплением можно вызывать в форматах

`cumtrapz(x, y)` и `cumtrapz(y)`,

где `x` и `y` векторы, определяющие функцию $y(x)$.

ЗАДАЧА 8.12. Вычислить интеграл $I=\int_0^{\frac{\pi}{2}} \frac{1}{5+\sin(x)}$.

Листинг 8.18 демонстрирует применение функции интегрирования с накоплением `cumtrapz` к поставленной задаче. Там же приведена интерпретация работы этой функции с помощью команды `trapz`.

```
>>> x=0:0.1:pi/2;
y=(5+sin(x)).^(-1);
%1. Интегрирование с накоплением
I1=cumtrapz(x,y)
>>>I1 =
 Columns 1 through 8:
0.0 0.0198 0.03923 0.05829 0.07701 0.09541 0.11352 0.13136
 Columns 9 through 16:
0.1489 0.1663 0.18356 0.2006 0.2175 0.23434 0.25108 0.2677
 %2. Обычное интегрирование
 I2=trapz(x,y)
%Значение I2 совпадает с последним значением вектора I1
>>>I2 =  0.26777
 %3.
 x=0:0.1:1;
 y=(5+sin(x)).^(-1);
  I3=trapz(x,y)
%Значение I3 совпадает с 11-м значением вектора I1
>>>I3 =  0.18356
```

Листинг 8.18

8.3.2 Интегрирование по методу Симпсона

Изложим идею *интегрирования по методу Симпсона*. Пусть $n=2m$ – четное число, а $y_i=f(x_i)(i=0,n)$ – 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.) кривая $y=f(x)$ заменяется параболой $y=L(x)$, коэффициенты которой подобраны так, что она проходит через точки y_0, y_1, y_2 .

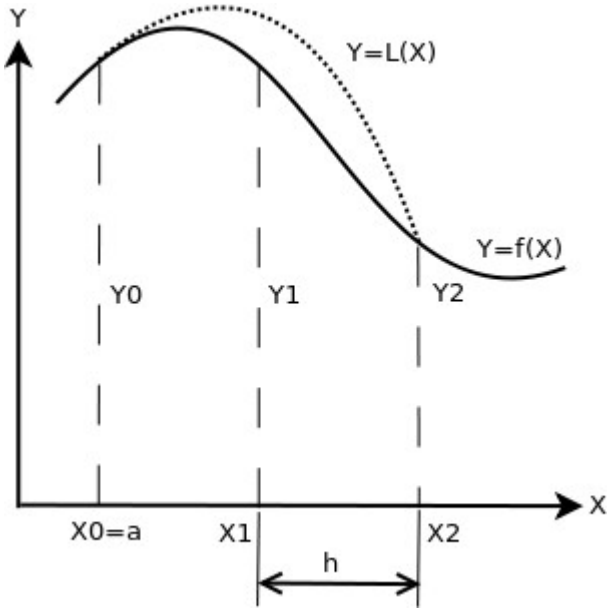


Рис. 8.6. Геометрическая интерпретация интегрирования по методу Симпсона

Площадь криволинейной трапеции, ограниченной сверху параболой, составит:

$$S_i=\frac{h}{3}(y_{i-1}+4y_i+y_{i+1}) \quad .$$

Суммируя площади всех криволинейных трапеций, получим:

$$S=\int_a^b f(x)\,dx\approx\frac{h}{3}(y_0+4y_1+2y_2+4y_3+\ldots+2y_{2m-2}+4y_{2m-1}+y_{2m})=\frac{h}{3}\left(y_0+y_{2m}+\sum_{i=1}^{2m-1}p\,y_i\right)$$

где $p=4, p=6-p$. Следовательно, *формула Симпсона для численного интегрирования* имеет вид [9]:

$$I=\int_a^b f(x)\,dx=\frac{h}{3}\left(f(a)+f(b)+\sum_{i=1}^{2m-1}p\,y_i\right) \quad .$$

Методы трапеций и Симпсона являются частными случаями *квадратурных формул Ньютона–Котеса*, которые, вообще говоря, имеют вид

$$\int_a^b y\,dx=(b-a)\sum_{i=0}^{n}H_i y_i \quad ,$$

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* [9].

В Octave реализован вычислительный *алгоритм метода Симпсона с автоматическим выбором шага*. Автоматический выбор шага интегрирования заключается в том, что интервал интегрирования разбивают на *n* отрезков и вычисляют значение интеграла, если полученное значение не удовлетворяет заданной точности вычислений, то *n* увеличивают вдвое и вновь вычисляют значение интеграла, так повторяют до тех пор пока не будет достигнута заданная

точность. Итак, вычисление интеграла по методу Симпсона обеспечивает функция

```
[F, K]=quadv(name, a, b [, tol, trace]),
```

где:

`name` – имя функции, задающей подынтегральное выражение;

`a`, `b` – пределы интегрирования; `tol` – точность вычислений;

`trace` – параметр позволяющий получить информацию о ходе вычислений в виде таблицы, в столбцах которой представлены значение количества вычислений, начальная точка текущего промежутка интегрирования, его длина и значение интеграла;

`F` – значение интеграла; `K` – количество итераций.

ЗАДАЧА 8.13. Вычислить интеграл $\int_0^1 \sqrt{4-x^2}\,dx$.

Решение задачи с применением функции `quadv` приведено в листинге 8.19.

```
%Подынтегральная функция
function y=G(x)
y=(4-x^2).^(1/2);
end;
%Вычисление интеграла по методу Симпсона
format long
%Точность установлена по умолчанию 1.0e-06
[F1,K1]=quadv('G',0,1)
%Результат - значение интеграла и количество итераций
>>>F1 =  1.91322288999134
K1 =  17
%Точность установлена пользователем 1.0e-07
[F2,K2]=quadv('G',0,1,1.0e-07)
%Результат - значение интеграла и количество итераций
>>>F2 =  1.91322295090669
K2 =  33
format short
%Вызов функций с заданной степенью точности
%Вывод дополнительной информации о вычислениях
quadv('G',0,1,1.0e-05,5)
>>>5.00000   0.00000   1.00000   1.91321
   7.00000   0.00000   0.50000   0.98948
   9.00000   0.50000   0.50000   0.92374
   11.0000   0.50000   0.25000   0.47458
   13.0000   0.75000   0.25000   0.44916
ans =  1.9132
```

Листинг 8.19

8.3.3 Интегрирование по квадратурным формулам Гаусса

Запишем в общем виде квадратурную формулу для функции заданной на промежутке [–1; 1]

$$\int_{-1}^{1} f(t)\,dt = \sum_{i=0}^{n} A_i \cdot f(t_i) \quad .$$

Попытаемся найти коэффициенты A_i и узловые точки t_i, таким образом, чтобы квадратурная формула была точной для всех полиномов $f(t)=1, t, t^2, \ldots, t^{2n-1}$.

В этом случае построение квадратурной формулы приводит к определению A_i и t_i

из нелинейной системы $2n$ уравнений:

$$\begin{cases} \sum_{i=1}^{n} A_i = 2, \\ \sum_{i=1}^{n} A_i \cdot t_i = 0, \\ \dots \\ \sum_{i=1}^{n} A_i \cdot t_i^{2n-2} = \frac{2}{2n-1}, \\ \sum_{i=1}^{n} A_i \cdot t_i^{2n-1} = 0. \end{cases}$$

Решение нелинейной системы задача не тривиальная, но ее можно обойти, если знать, что значениями t_i квадратурной формулы служат корни многочлена Лежандра

$$P_n(t) = \frac{1}{2^n \cdot n!} \cdot \frac{d^n}{dt^n}((t^2-1)^2), (n = 0,1,2,\dots).$$

Как известно, корни полинома Лежандра существуют при любом *n*, различны и принадлежат интервалу [–1; 1].

Итак, *квадратурной формулой Гаусса* называют выражение [9]

$$\int_a^b f(x)\,dx = \frac{b-a}{2} \sum_{i=1}^{n} A_i \cdot f\left(\frac{a+b}{2} + \frac{b-a}{2} \cdot t_i\right),$$

где t_i – корни полинома Лежандра, а A_i определяется интегрированием базисных многочленов Лежандра $P_i(t)$ степени $n-1$:

$$A_i = \int_{-1}^{1} \frac{(t-t_1)\dots(t-t_{i-1})(t-t_{i+1})\dots(t-t_n)}{(t_i-t_1)\dots(t_i-t_{i-1})(t_i-t_{i+1})\dots(t_i-t_n)}\,dt .$$

В Octave *интегрирование по квадратуре Гаусса* выполняет функция

```
[F, kod, K, err] = quad(name, a, b, tol, sing),
```

где

`name` – имя функции, задающей подынтегральное выражение;

`a`, `b` – пределы интегрирования;

`tol` – точность вычислений;

`sing` – вектор значений, близких к тем, в которых подынтегральная функция терпит разрыв;

`F` – значение интеграла;

`kod` – код ошибки в решении (0 — решение завершено успешно);

`K` – количество итераций; `err` – погрешность вычислений.

ЗАДАЧА 8.14. Вычислить интеграл $\int_0^1 t^2 \cdot \sqrt{(3+\sin(\frac{1}{t}))}\,dt$.

Обратите внимание, что в нижней границе интегрирования подынтегральная функция терпит разрыв. Решение задачи с применением функции `quad` приведено в листинге 8.11.

```
clear all;
function y=f(x)
y=(x.^2).*sqrt(3+sin(1./x));
end;
format long
[F,kod , K, err]=quad('f',0,1)
>>>F =  0.654343719149802
```

```
kod = 0
K =  1323
err =  1.37557012147481e-08
[F,kod , K, err]=quad('f',0,1,1.0e-05)
>>>F =  0.654343738854992
kod = 0
K =  315
err =  3.82733563379833e-06
[F,kod , K, err]=quad('f',0,1,1.0e-20)
>>>F =  0.654343718970708
kod = 0
K =  1491
err =  9.39557628735834e-09
[F,kod , K, err]=quad('f',0,1,1.0e-20,0.1)
>>>F =  0.654343710193938
kod = 0
K =  840
err =  5.80259740257105e-09
[F,kod , K, err]=quad('f',0,1,1.0e-20,0.001)
>>>F =  0.654343718720156
kod = 0
K =  1596
err =  8.35248716562893e-09
```

Листинг 8.20

Функции

```
F = quadl(f, a, b [, tol, trace]),
[F,err] = quadgk(f, a, b [, tol, trace]),
```

где

`f` – имя функции, задающей подынтегральное выражение;

`a`, `b` – пределы интегрирования; `tol` – точность вычислений;

`trace` – таблица промежуточных вычислений;

`F` – значение интеграла; `err` – погрешность вычислений;

также выполняют *интегрирование по квадратуре Гаусса.* В этих функциях специальным образом подбирается шаг. В первом случае по методу Гаусса-Лобатто, во втором Гауса-Конрада.

ЗАДАЧА 8.15. Вычислить интеграл $\int_{\frac{-\pi}{3}}^{\frac{\pi}{3}} tg^4(x)\,dx$.

Решение задачи с применением функции `quadl` и `quadgk` приведено в листинге 8.9.

```
function y=f(x)
y = tan(x).^4;
end;
format long
[F]=quadl('f',-pi/3,pi/3,1.0e-05)
>>>F =  2.09439512983937
[F,err]=quadgk('f',-pi/3,pi/3,1.0e-05)
>>>F =  2.09439510239319
err =  1.02555919485880e-12
```

Листинг 8.21

9. Решение обыкновенных дифференциальных уравнений и систем

Дифференциальные уравнения и системы описывают очень 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 Octave.

9.1 Общие сведения о дифференциальных уравнениях

Дифференциальным уравнением n-го порядка называется соотношение вида [9]

$$H(t, x, x', x'', \ldots, x^{(n)}) = 0. \tag{9.1}$$

Решением дифференциального уравнения называется функция $x(t)$, которая обращает уравнение в тождество.

Системой дифференциальных уравнений n-го порядка называется система вида: [9]

$$\begin{cases} x_1' = f_1(t, x_1, x_2, \ldots, x_n), \\ x_2' = f_2(t, x_1, x_2, \ldots, x_n), \\ \quad \ldots\ldots\ldots\ldots \\ x_n' = f_n(t, x_1, x_2, \ldots, x_n). \end{cases} \tag{9.2}$$

Системой линейных дифференциальных уравнений называется система вида:

$$\begin{cases} x_1' = \sum_{j=1}^{n} a_{1j} x_j + b_1, \\ x_2' = \sum_{j=1}^{n} a_{2j} x_j + b_2, \\ \quad \ldots\ldots\ldots\ldots \\ x_n' = \sum_{j=1}^{n} a_{nj} x_j + b_n. \end{cases} \tag{9.3}$$

Решением системы называется вектор $x(t) = \begin{cases} x_1(t) \\ x_2(t) \\ \ldots \\ x_n(t) \end{cases}$, который обращает уравнения систем (9.2), (9.3) 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 *приближенные* (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 определенных исходных данных. Поэтому широкое распространение получили численные методы решения обыкновенных дифференциальных уравнений [9].

9.2 Численные методы решения дифференциальных уравнений и их реализация в Octave

Численные методы решения дифференциального уравнения первого порядка будем рассматривать для следующей задачи Коши. Найти решение дифференциального уравнения

$$x' = f(x, t) \tag{9.4}$$

удовлетворяющее начальному условию

$$x(t_0) = x_0 \tag{9.5}$$

иными словами, требуется найти интегральную кривую $x = x(t)$, проходящую через заданную точку $M_0(t_{0,} x_0)$ (рис. 9.1).

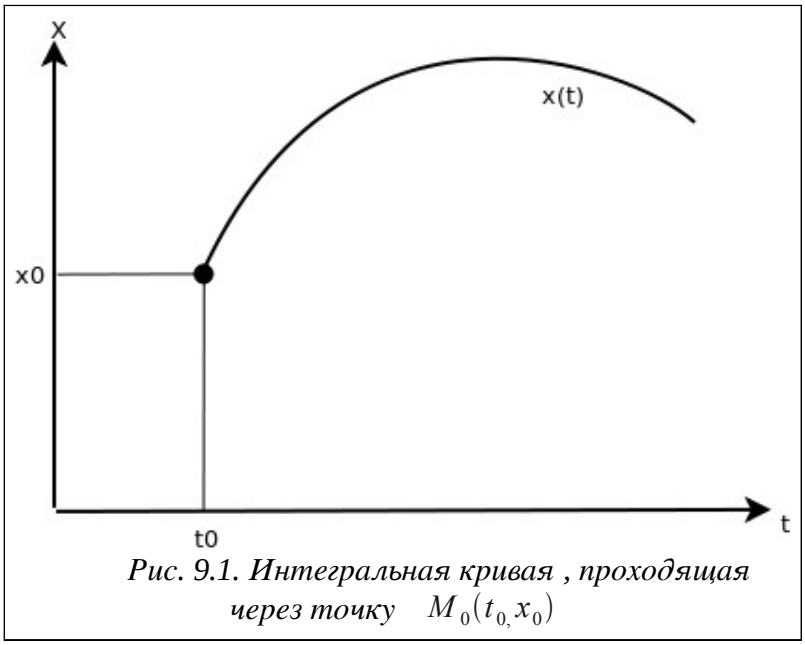


Рис. 9.1. Интегральная кривая , проходящая через точку $M_0(t_{0,} x_0)$

Для дифференциального уравнения n-го порядка

$$x^{(n)} = f(t, x, x', x'', \ldots, x^{(n-1)}) \tag{9.6}$$

задача Коши состоит в нахождении решения $x = x(t)$, удовлетворяющего уравнению (9.6) и начальным условиям

$$x(t_0) = x_0, x'(t_0) = x_0', \ldots, x^{(n-1)}(t_0) = x_0^{(n-1)}. \tag{9.7}$$

Рассмотрим основные численные методы решения задачи Коши.

9.2.1 Решение дифференциальных уравнений методом Эйлера

При решении задачи Коши (9.4), (9.5) на интервале $[t_0, t_n]$, выбрав достаточно малый шаг *h*, построим систему равноотстоящих точек

$$h=\frac{t_n-t_0}{n}, t_i=t_0+ih, i=0,n \quad . \tag{9.8}$$

Для вычисления значения функции в точке t_1 разложим функцию $x=x(t)$ в окрестности точки t_0 в ряд Тейлора [6]

$$x(t_1)=x(t_0+h)=x(t_0)+x'(t_0)h+x''(t_0)\frac{h^2}{2}+... \quad . \tag{9.9}$$

При достаточном малом значении *h* членами выше второго порядка можно пренебречь и с учетом $x'(t_0)=f(x_0, t_0)$ получим следующую формулу для вычисления приближенного значения функции $x(t)$ в точке t_1

$$. \quad x_1=x_0+h\,f(x_0, t_0). \tag{9.10}$$

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

$$x_2=x_1+h\,f(x_1, t_1).$$

Повторяя этот процесс, сформируем последовательность значений x_i в точках t_i по формуле

$$x_{i+1}=x_i+h\,f(x_i, t_i), i=1,n \quad . \tag{9.11}$$

Процесс нахождения значений функции x_i в узловых точках t_i по формуле (9.11) называется *методом Эйлера*. Геометрическая интерпретация метода Эйлера состоит в замене интегральной кривой $x(t)$ ломаной $M_0, M_1, M_2, ..., M_n$ с вершинами $M_i(x_i, y_i)$. Звенья ломанной Эйлера $M_i M_{i+1}$ в каждой вершине M_i имеют направление $y_i=f(t_i, x_i)$, совпадающее с направлением интегральной кривой $x(t)$ уравнения (9.4), проходящей через точку M_i (рис. 9.2). Последовательность ломанных Эйлера при $h \to 0$ на достаточно малом отрезке $[x_i, x_i+h]$ стремится к искомой интегральной кривой.

На каждом шаге решение $x(t)$ определяется с ошибкой за счет отбрасывания членов ряда Тейлора выше первой степени, что в случае быстроменяющейся функции $f(t, x)$ может привести к быстрому накапливанию ошибки. В методе Эйлера следует выбирать достаточной малый шаг *h*.

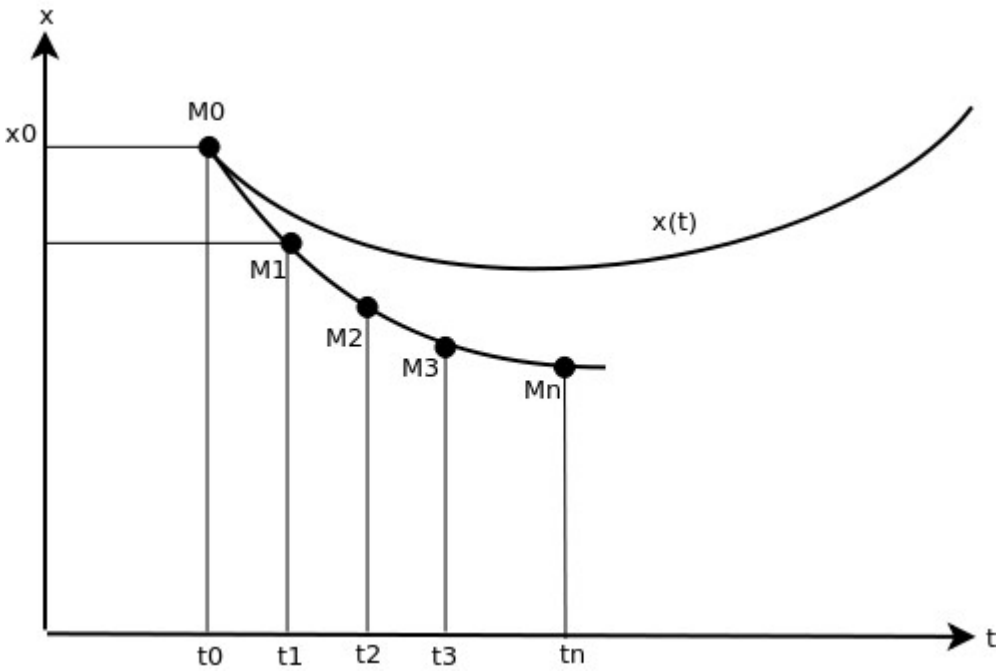


Рис. 9.2. Геометрическая интерпретация метода Эйлера

9.2.2 Решение дифференциальных уравнений при помощи модифицированного метода Эйлера

Более точным методом решения задачи (9.4)-(9.5) является *модифицированный метод Эйлера*, при котором сначала вычисляют промежуточные значения [2]

$$t_p = t_i + \frac{h}{2},\ x_p = x_i + \frac{h}{2} f(x_i, t_i) \quad . \tag{9.12}$$

после чего находят значение x_{i+1} по формуле

$$x_{i+1} = x_i + h\, f(x_p, t_p),\ i = 1, n \quad . \tag{9.13}$$

9.2.3 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 [6]:

$$x_{i+1} = x_i + h\phi(t_i, x_i, h),\ i = 1, n \quad . \tag{9.14}$$

Функция $\phi(t, x, h)$ приближает отрезок ряда Тейлора до k-го порядка и не содержит частных производных $f(t, x)$ [6].

Метод Эйлера является *методом Рунге-Кутта первого порядка* (k=1) и получается при $\phi(t, x, h) = f(t, x)$.

1. Семейство *методов Рунге-Кутта второго порядка* имеет вид [6]

$$x_{i+1}=x_i+((1-\alpha)f(t_i,x_i)+\alpha\, f(t_i+\frac{h}{2\alpha},x_i+\frac{h}{2\alpha}f(t_i,x_i))),i=1,n \quad . \tag{9.15}$$

Два наиболее известных среди методов Рунге-Кутта второго порядка [6] – это *метод Хойна* ($\alpha=\frac{1}{2}$) и модифицированный метод Эйлера ($\alpha=1$).

Подставив $\alpha=\frac{1}{2}$ в формулу (9.15), получаем расчетную формулу *метода Хойна* [6]:

$$x_{i+1}=x_i+\frac{h}{2}f(t_i,x_i)+f(t_i+h,x_i+h\,f(t_i,x_i)),i=1,n \quad . \tag{9.16}$$

Подставив $\alpha=1$ в формулу (9.15), получаем расчетную формулу уже рассмотренного выше модифицированного метода Эйлера

$$x_{i+1}=x_i+h(f(t_i+\frac{h}{2},x_i+\frac{h}{2}f(t_i,x_i))),i=1,n \quad . \tag{9.17}$$

Наиболее известным является *метод Рунге-Кутта четвертого порядка*, расчетные формулы которого можно записать в виде [6]:

$$\begin{cases} x_{i+1}=x_i+\Delta x_i, i=1,n, \\ \Delta x_i=\frac{h}{6}(K_1^i+2\mathrm{K}_2^i+2\mathrm{K}_3^i+K_4^i), \\ K_1^i=f(t_i,x_i), \\ K_2^i=f(t_i+\frac{h}{2},x_i+\frac{h}{2}K_1^i), \\ K_3^i=f(t_i+\frac{h}{2},x_i+\frac{h}{2}K_2^i), \\ K_4^i=f(t_i+h,x_i+hK_3^i). \end{cases} \tag{9.18}$$

Одной из модификаций метода Рунге-Кутта является *метод Кутта-Мерсона* (или *пятиэтапный метод Рунге-Кутта четвертого порядка*), который состоит в следующем [6].

1. На *i*-м шаге рассчитываются коэффициенты

$$\begin{cases} K_1^i=f(t_i,x_i), \\ K_2^i=f(t_i+\frac{h}{3},x_i+\frac{3}{2}K_1^i), \\ K_3^i=f(t_i+\frac{h}{3},x_i+\frac{h}{6}K_1^i+\frac{h}{6}K_2^i), \\ K_4^i=f(t_i+\frac{h}{2},x_i+\frac{h}{8}K_1^i+\frac{3h}{2}K_2^i), \\ K_5^i=f(t_i+h,x_i+\frac{h}{2}K_1^i-\frac{3h}{2}K_3^i+2hK_4^i). \end{cases} \tag{9.19}$$

2. Вычисляем приближенное значение $x(t_{i+1})$ по формуле

$$\tilde{x}_{i+1} = x_i + \frac{h}{2}(K_1^i - 3K_3^i + 4K_4^i) \quad . \tag{9.20}$$

3. Вычисляем приближенное значение $x(t_{i+1})$ по формуле

$$x_{i+1} = x_i + \frac{h}{6}(K_1^i + 4K_3^i + K_5^i) \quad . \tag{9.21}$$

4. Вычисляем оценочный коэффициент по формуле

$$R = 0.2|x_{i+1} - \tilde{x}_{i+1}| \quad . \tag{9.22}$$

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 (9.21) 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.

Еще 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*.

9.2.4 Решение дифференциальных уравнений методом прогноза-коррекции Адамса

Рассмотрим решение уравнения (9.1)-(9.2) 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 (9.1) на интервале $[t_i, t_{i+1}]$ и получим соотношение [6]

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(t, x(t))\,dt \quad . \tag{9.23}$$

При вычислении интеграла, входящего в (9.23), 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 (9.23), получаем первое приближение (прогноз) $\tilde{x}_{i+1}$ для значения функции в точке t_{i+1}

$$\tilde{x}_{i+1} = x_i + \frac{h}{24}(-9f(t_{i-3}, x_{i-3}) + 37f(t_{i-2}, x_{i-2}) - 59f(t_{i-1}, x_{i-1}) + 55f(t_i, x_i)) \quad . \tag{9.24}$$

Как только $x_{i+1}^{\sim}$ 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 $(t_{i+1},x_{i+1}^{\sim})$, после чего подставляем его в (9.23) и получаем второе приближение (корректор)

$$x_{i+1}=x_i+\frac{h}{24}(f(t_{i-2},x_{i-2})-5f(t_{i-1},x_{i-1})+19f(t_i,x_i)+9f(t_{i+1},x_{i+1}^{\sim})) \quad . \qquad (9.25)$$

Таким образом, для вычисления значения $x(t_{i+1})$ *методом Адамса* необходимо последовательно применять формулы (9.24), (9.25) [2], а первые четыре точки можно получить методом Рунге-Кутта.

9.2.5 Решение дифференциальных уравнений методом Милна

Отличие *метода Милна* от метода Адамса состоит в использовании интерполяционного полинома Ньютона.

Подставив в (9.23) 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 $x_{k+1}^{\sim}$ для значения функции в точке t_{k+1} [6]

$$x_{k+1}^{\sim}=x_{k-3}+\frac{4h}{3}(2f(t_{k-2},x_{k-2})-f(t_{k-1},x_{k-1})+2f(t_k,x_k)) \quad . \qquad (9.26)$$

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 $(t_{k+1},x_{k+1}^{\sim})$, после чего подставляем его в (9.23) и получаем второе приближение – корректор Милна [6]

$$x_{k+1}=x_{k-1}+\frac{h}{3}(f(t_{k-1},x_{k-1})+4f(t_k,x_k)+f(t_{k+1},x_{k+1}^{\sim})) \quad . \qquad (9.27)$$

В методе Милна для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (9.26), (9.27), а первые четыре точки можно получить методом Рунге-Кутта.

Существует *модифицированный метод Милна*. В нем сначала вычисляется первое приближение по формуле (9.26), затем вычисляется управляющий параметр [6]

$$m_{k+1}=x_{k+1}^{\sim}+\frac{28}{29}(x_k-\tilde{x}_k) \quad , \qquad (9.28)$$

после чего вычисляется значение второго приближения – корректор Милна по формуле

$$x_{k+1}=x_{k-1}+\frac{h}{3}(f(t_{k-1},x_{k-1})+4f(t_k,x_k)+f(t_{k+1},m_{k+1})) \quad . \qquad (9.29)$$

В модифицированном методе Милна первые четыре точки можно получить методом Рунге-Кутта, а для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (9.26), (9.28), (9.29).

9.3 Реализация численных методов в Octave

Ниже приведены тексты функций, реализующие рассмотренные в п. 9.2 численные методы решения дифференциальных уравнений.

Листинг 9.1 представляет функцию решения задачи Коши методом Эйлера.

```
function [x,t]=eiler(a,b,n,x0)
%Функция решения задачи Коши x'(t)=g(t,x) x(a)=x0 методом
%Эйлера на интервале интегрирования [a,b], n — количество
%отрезков, на которые разбивается интервал [a,b].
%Вычисление шага h.
h=(b-a)/n;
x(1)=x0;
% Формирование системы равноотстоящих узлов ti
for i=1:n+1
t(i)=a+(i-1)*h;
end
%Вычисление значение функции в узловых точках методом
%Эйлера по формуле (9.11)
for i=2:n+1
x(i)=x(i-1)+h*g(t(i-1),x(i-1));
end
end
```

Листинг 9.1.

Функция решения задачи Коши модифицированным методом Эйлера (листинг 9.2):

```
function [x,t]=eiler_m(a,b,n,x0)
%Функция решения задачи Коши x'(t)=g(t,x) x(a)=x0
%модифицированным методом Эйлера на интервале
% интегрирования [a,b], n - количество отрезков, на
% которые разбивается интервал [a,b].
% Вычисление шага h.
h=(b-a)/n;
x(1)=x0;
% Формирование системы равноотстоящих узлов ti
for i=1:n+1
t(i)=a+(i-1)*h;
end
%Вычисление значение функции в узловых точках
%модифицированным методом Эйлера по формулам
%(9.13)-(9.14).
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
```

Листинг 9.2.

Далее (листинг 9.3) представлена реализация задачи Коши методом Рунге-Кутта.

```
function [x,t]=runge_kut(a,b,n,x0)
%Функция решения задачи Коши x'(t)=g(t,x) x(a)=x0 методом
%Рунге-Кутта на интервале интегрирования [a,b],
%n - количество отрезков, на которые разбивается интервал
```

```
%[a,b].
%Вычисление шага h.
h=(b-a)/n;
x(1)=x0;
%Формирование системы равноотстоящих узлов ti
for i=1:n+1
t(i)=a+(i-1)*h;
end
%Вычисление значения функции в узловых точках методом
%Рунге-Кутта по формуле (9.19).
for i=2:n+1
% Расчет коэффициентов K1, K2, K3, K4
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);
%Расчет приращения delt
delt=h/6*(K1+2*K2+2*K3+K4);
x(i)=x(i-1)+delt;
end
end
```

Листинг 9.3.

Листинг 9.4 содержит функцию решения задачи Коши методом Кутта-Мерсона.

```
function [x,t,j]=kut_merson(a,b,n,eps,x0)
%Функция решения задачи Коши x'(t)=g(t,x) x(a)=x0 методом
%Кутта-Мерсона на интервале интегрирования [a,b]
%с точноктью eps, n - количество отрезков, на которые
% вначале разбивается интервал [a,b].
% Вычисление шага h.
h=(b-a)/n;
x(1)=x0;t(1)=a;i=2;
while (t(i-1)+h)<=b
R=3*eps;
while R>eps
%Расчет коэффициентов K1, K2, K3, K4, K5.
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);
%Вычисление сравниваемых значений x(i+1)
X1=x(i-1)+h/2*(K1-3*K3+4*K4);
X2=x(i-1)+h/6*(K1+4*K4+K5);
%Вычисление оценочного коэффициента R.
R=0.2*abs(X1-X2);
%Сравнение оценочного коэффициента R с точностью eps.
if R>eps
h=h/2;
else
%Если оценочный коэффициент R меньше точности eps,
%то происходит формирование очередной найденной точки и
```

```
%переход к следующему этапу по i.
t(i)=t(i-1)+h;
x(i)=X2;
i=i+1;
%Если оценочный коэффициент R меньше eps/64,
%то можно попробовать увеличить шаг.
if R<=eps/64
if (t(i-1)+2*h)<=b
h=2*h;
end
end
end
end
end
%В переменной j возвращается количество элементов в
%массивах x и t
j=i-1
end
```

Листинг 9.4.

Функция решения задачи Коши методом Милна[11] представлена в листинге 9.5.

```
function [x,t]=adams(a,b,n,x0)
% Функция решения задачи Коши x'(t)=g(t,x) x(a)=x0 методом
%Адамса на интервале интегрирования [a,b],
%n - количество отрезков, на которые разбивается интервал
%[a,b].
% Вычисление шага h
h=(b-a)/n;
x(1)=x0;
%Формирование системы равноотстоящих узлов ti.
for i=1:n+1
t(i)=a+(i-1)*h;
end
%Вычисление значения функции в трех узловых точках методом
%Рунге-Кутта по формуле(9.19)
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);
%Расчет приращения delt
delt=h/6*(K1+2*K2+2*K3+K4);
x(i)=x(i-1)+delt;
end
%Вычисление значение в остальных точках методом Адамса
for i=4:n
%Вычисление прогноза
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)));
% Вычисление корректора
x(i+1)=x(i)+h/24*(g(t(i-2),x(i-2))-5*g(t(i-1),x(i-1))...
```

11 Написать функцию модифицированного метода Милна авторы предоставляют читателю.

```
+19*g(t(i),x(i))+9*g(t(i+1),xp));
end
end
```

Листинг 9.5. Функция решения задачи Коши методом Адамса.

```
function [x,t]=miln(a,b,n,x0)
%Функция решения задачи Коши x'(t)=g(t,x) x(a)=x0 методом
%Милна на интервал интегрирования [a,b],n — количество
%отрезков, на которые разбивается интервал [a,b].
%Вычисление шага h
h=(b-a)/n;x(1)=x0;xp(1)=x(1);
%Формирование системы равноотстоящих узлов ti
for i=1:n+1
t(i)=a+(i-1)*h;
end
%Вычисление значение функции в трех узловых точках методом
%Рунге-Кутта по формуле(9.19)
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);
%Расчет приращения delt
delt=h/6*(K1+2*K2+2*K3+K4);
x(i)=x(i-1)+delt;
xp(i)=x(i);
end
%Вычисление значение в остальных точках методом Адамса
for i=4:n
%Вычисление прогноза
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)));
%Вычисление управляющего параметра
m=xp(i+1)+28/29*(x(i)-xp(i));
%Вычисление корректора.
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
```

Листинг 9.6.

Рассмотрим использование приведенных выше функций на примере решения следующей задачи Коши.

ЗАДАЧА 9.1. Решить задачу Коши $\begin{matrix} y'(x)=6y-13x^3-22x^2+17x-11+\sin(x) \\ y(0)=2 \end{matrix}$.

Известно точное решение задачи 9.1

$$y(x)=\frac{119}{296}e^{6x}+\frac{1}{24}(52x^3+114x^2-30x+39)-\frac{6\sin(x)}{37}-\frac{\cos(x)}{37}.$$

На листинге 9.2 представлено решение уравнения методами:

- модифицированным методом Эйлера;
- Рунге-Кутта;
- Кутта-Мерсона;

- Адамса;
- Милна.

```
%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;
for i=1:n+1
t(i)=a+(i-1)*h;
end
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.
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
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);
X1=x(i-1)+h/2*(K1-3*K3+4*K4);
X2=x(i-1)+h/6*(K1+4*K4+K5);
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 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
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
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;
```

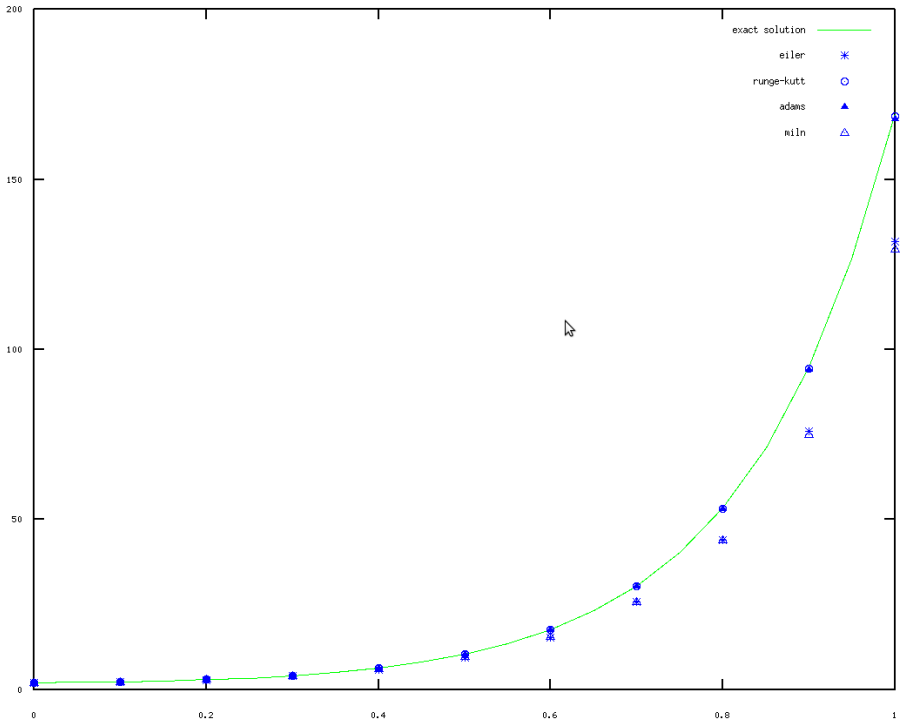
```
end
for i=2:4
K1=g(t(i-1),x(i-1));K2=g(t(i-1)+h/2,x(i-1)+h/2*K1);
K3=g(t(i-1)+h/2,x(i-1)+h/2*K2);K4=g(t(i-1)+h,x(i-1)+h*K3);
delt=h/6*(K1+2*K2+2*K3+K4);
x(i)=x(i-1)+delt;
end
for i=4:n
xp=x(i)+h/24*(-9*g(t(i-3),x(i-3))+37*g(t(i-2),x(i-2))...
-59*g(t(i-1),x(i-1))+55*g(t(i),x(i)));
x(i+1)=x(i)+h/24*(g(t(i-2),x(i-2))-5*g(t(i-1),x(i-1))...
+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.
plot(x1,y1,'-g;exact solution;',XE_M,YE_M,'*b;eiler;',...
XR,YR,'ob;runge-kutt;',XA,YA,'^b;adams;',...
XM,YM,'>b;miln;');
figure();
plot(x1,y1,'-g;exact solution;',XKM,YKM,'<b;kut-merson;');
grid on;
```

Листинг 9.7.

На рис. 9.3-9.4 приведены графики решения задачи модифицированным методом Эйлера, методами Рунге-Кутта, Кутта-Мерсона, Адамса, Милна и точного решения. При обращении к функции kut_merson в качестве *n* передавалось число 5.

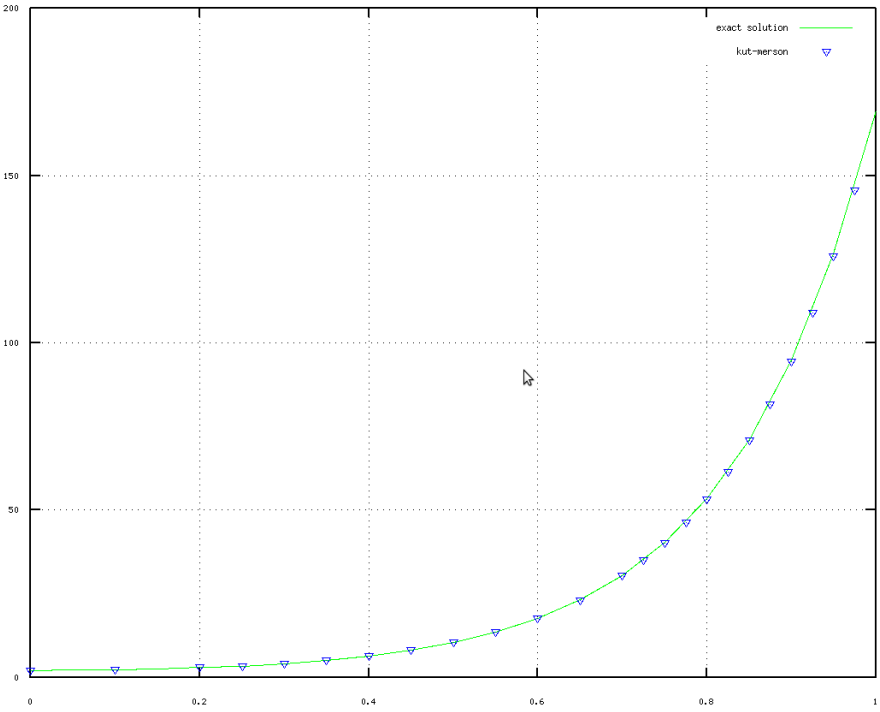
Функция выбирала оптимальный шаг на каждом из отрезков. Как видно из рисунка, шаг то увеличивается, то уменьшается. При решении уравнения методом Кутта-Мерсона невозможно гарантировать вычисление значения в заданных точках интервала. Однако после получения решения методом Кутта-Мерсона значение в любой точки можно вычислить, интерполируя полученную зависимость.

При решении данной задачи наиболее точными оказались методы Адамса, Рунге-Кутта и Кутта-Мерсона.



0.616572, 108.181

Рис. 9.3 Графики решения модифицированным методом Эйлера, методами Рунге-Кутта, Адамса, Милна и точного решения



0.583176, 91.5193

Рис. 9.4 Графики решения методом Кутта-Мерсона и точного решения

9.4 Решение систем дифференциальных уравнений

Все рассмотренные методы решения дифференциальных уравнений применимы и для систем дифференциальных уравнений. Рассмотрим на примере метода Рунге-Кутта, как, рассмотренные методы, можно обобщить для систем.

Пусть дана система дифференциальных уравнений в матричном виде [6]

$$\frac{d\bar{x}}{dt}=\bar{f}(t,\bar{x}) \tag{9.30}$$

с начальным условием $\bar{x}(t_0)=\bar{x_0}$,

где $\bar{x}=\begin{pmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{pmatrix}$, $\bar{f}(t,\bar{x})=\begin{pmatrix} f_1(t,x_1,x_{2,}\dots,x_n) \\ f_2(t,x_1,x_{2,}\dots,x_n) \\ \dots \\ f_n(t,x_1,x_{2,}\dots,x_n) \end{pmatrix}$, $\bar{x^0}=\begin{pmatrix} x_1^0 \\ x_2^0 \\ \dots \\ x_n^0 \end{pmatrix}$.

Задавшись некоторым шагом h и введя стандартные обозначения $t_i=t_0+i\,h$, $x_i=x(t_i)$, $\Delta x_i=x_{i+1}-x_i$, $i=1,n$ получим формулы метода Рунге-Кутта для системы:

$$\begin{cases} \bar{x_{i+1}}=\bar{x}_i+\Delta\bar{x}_i,\ i=1,n, \\ \Delta\bar{x}_i=\frac{h}{6}(\bar{K}_1^i+2\bar{K}_2^i+2\bar{K}_3^i+\bar{K}_4^i), \\ \bar{K}_1^i=\bar{f}(t_i,\bar{x}_i), \\ \bar{K}_2^i=\bar{f}(t_i+\frac{h}{2},\bar{x}_i+\frac{h}{2}\bar{K}_1^i), \\ \bar{K}_3^i=\bar{f}(t_i+\frac{h}{2},\bar{x}_i+\frac{h}{2}\bar{K}_2^i), \\ \bar{K}_4^i=\bar{f}(t_i+h,\bar{x}_i+h\bar{K}_3^i). \end{cases} \tag{9.31}$$

9.5 Встроенные функции Octave для решения дифференциальных уравнений

Наиболее часто используемыми в Octave функциями для решения дифференциальных уравнений являются:

- `ode23(@f, interval, X0, options)`, `ode45(@f, interval, X0, options)` – функции решений обыкновенных нежёстких дифференциальных уравнений (или систем) методом Рунге-Кутта 2-3-го и 4-5-го порядка точности соответсвенно;
- `ode5r(f, interval, X0, options)`, `ode2r(f, interval, X0, options)` – – функции решений обыкновенных жёстких дифференциальных уравнений (или систем).

Функции решают дифференциальные уравнения (системы), автоматически подбирая шаг для достижения необходимой точности.

Входными параметрами этих функций являются:

- `f` – вектор-функция для вычисления правой части дифференциального уравнения или системы;
- `interval` – массив из двух чисел, определяющий интервал интегрирования

дифференциального уравнения или системы;
- `X0` – вектор начальных условий системы дифференциальных систем;
- `options` – параметры управления ходом решения дифференциального уравнения или системы.

Для определения параметров управления ходом решения дифференциальных уравнений используется функция `odeset` следующей структуры

`options = odeset ("namepar1", val1, "namepar2", val2, …, "nameparn", valn).`

Здесь
- `"namepari"` – имя i-го параметра;
- `vali` – значение i-го параметра.

При решении дифференциальных уравнений необходимо определить следующие параметры:
- `RelTol` – относительная точность решения, значение по умолчанию 10^{-3} ;
- `AbsTol` – абсолютная точность решения, значение по умолчанию 10^{-3} ;
- `InitialStep` – начальное значение шага изменения независимой переменной, значение по умолчанию 0.025;
- `MaxStep` – максимальное значение шага изменения независимой переменной, значение по умолчанию 0.025.

Все функции возвращают:
- массив `T` – координат узлов сетки, в которых ищется решение;
- матрицу `X`, *i*-й столбец которой является значением вектор-функции решения в узле T_i .

Напомним читателю определение жёсткой системы дифференциальных уравнений. Система дифференциальных уравнений n-го порядка

$$\frac{d\,\boldsymbol{x}}{dt} = \boldsymbol{B}\,x \qquad (9.32)$$

называется жесткой [2], если выполнены следующие условия:
- действительные части всех собственных чисел матрицы $\boldsymbol{B}(n)$ отрицательны $\left|\mathrm{Re}(\lambda_k)\right| < 0, k = 1,2,\ldots,n$;
- величина $s = \dfrac{\max\limits_{1 \le k \le n} \left|\mathrm{Re}(\lambda_k)\right|}{\min\limits_{1 \le k \le n} \left|\mathrm{Re}(\lambda_k)\right|}$, называемая числом жесткости системы, велика.

При исследовании на жёсткость нелинейной системы дифференциальных уравнений (9.30) в роли матрицы **B** будет выступать матрица частных производных.

Решим задачу 9.1 с использованием функций `ode23`, `ode45`. Текст программы с комментариями представлен на листинге 9.1.

```
%Точное решение системы.
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
```

```
%Определение параметров управления ходом решения
% дифференциального уравнения.
%RelTol – относительная точность решения 1E-5,
%AbsTol – абсолютная точность решения 1E-5,
%InitialStep – начальное значение шага изменения
%независимой переменной 0.025,
%MaxStep – максимальное значение шага изменения
%независимой переменной 0.1.
par=odeset ("RelTol", 1e-5, "AbsTol", 1e-5,...
'InitialStep',0.025,'MaxStep',0.1);
%Решение дифференциального уравнения методом Рунге-Кутта
%2-3 порядка.
[X23,Y23]=ode23(@g,[0 0.25],2,par);
%Определение параметров управления ходом решения
% дифференциального уравнения.
%RelTol – относительная точность решения 1E-4,
%AbsTol – абсолютная точность решения 1E-4,
%InitialStep – начальное значение шага изменения
%независимой переменной 0.005,
%MaxStep – максимальное значение шага изменения
%независимой переменной 0.2.
par=odeset ("RelTol", 1e-4, "AbsTol",...
1e-4,'InitialStep',0.05,'MaxStep',0.2);
%Решение дифференциального уравнения методом Рунге-Кутта
%4-5 порядка.
[X45,Y45]=ode45(@g,[0 0.25],2,par);
%Точное решение
x1=0:0.05:0.25;
y1=fi(x1)
%График решения функцией ode23 и точного решения.
plot(x1,y1,'-g;exact solution;',X23,Y23,'*b;ode23;');
grid on;
figure();
%График решения функцией ode45 и точного решения.
plot(x1,y1,'-g;exact solution;',X45,Y45,'*b;ode45;');
grid on;
```

Листинг 9.8.

На рис. 9.5 представлено решение, найденное с помощью функции `ode23` с точностью 1E-5 и точное решение. На рис. 9.6 представлено решение, найденное с помощью функции `ode45` с точностью 1E-4 и точное решение.

Функции `ode23` и `ode45` позволяют найти решение с заданной точностью, однако, как и следовало ожидать, при использовании метода Рунге-Кутта более высокой точности шаг изменения переменной x намного меньше.

Рассмотрим решение жёсткой системы дифференциальных уравнений на примере.

ЗАДАЧА 9.2. Решить задачу Коши для жёсткой системы дифференциальных уравнений

$$\frac{d\boldsymbol{x}}{dt}=\begin{pmatrix}119.46 & 185.38 & 126.88 & 121.03\\ -10.395 & -10.136 & -3.636 & 8.577\\ -53.302 & -85.932 & -63.182 & -54.211\\ -115.58 & -181.75 & -112.8 & -199\end{pmatrix}\boldsymbol{x},\qquad \boldsymbol{x}(0)=\begin{pmatrix}1\\1\\1\\1\end{pmatrix}.$$

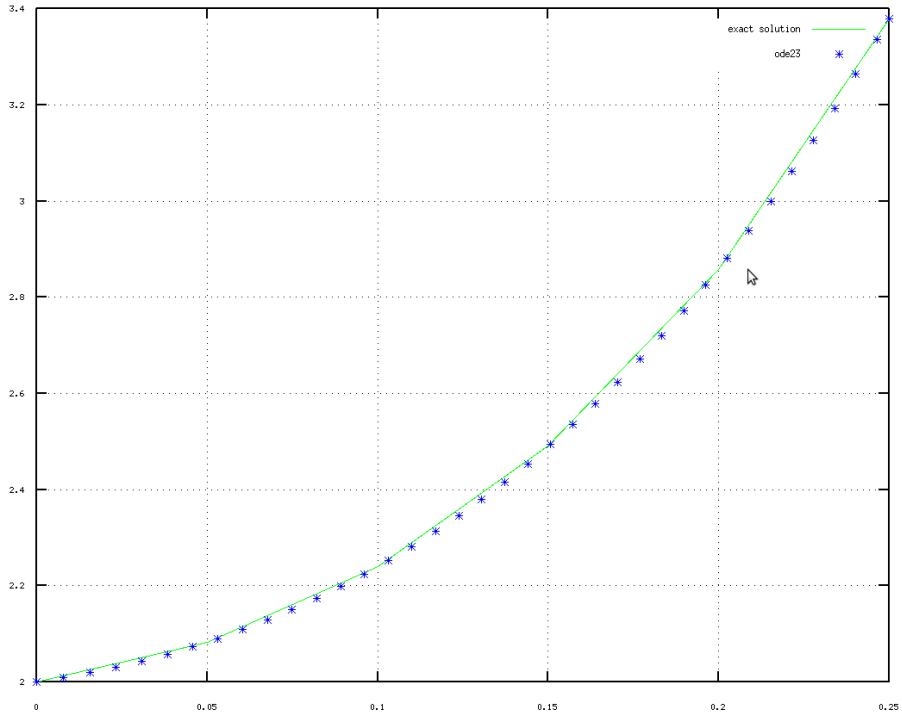


Рис. 9.5. Графики точного решения задачи 9.1 и решения, найденного с помощью функции ode23

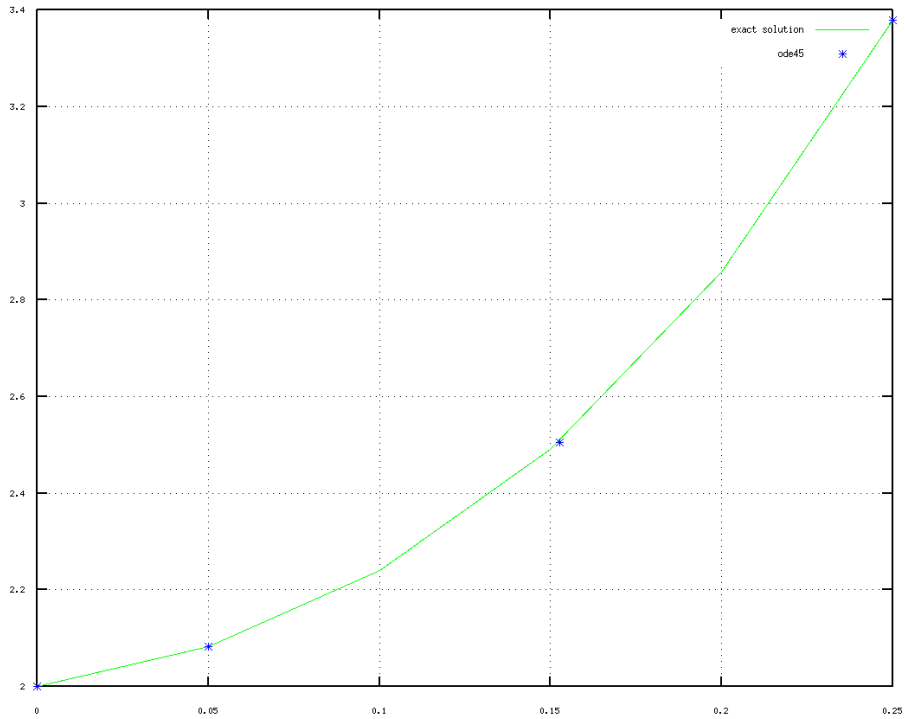


Рис. 9.6 Графики точного решения задачи 9.1 и решения, найденного с помощью функции ode45

Решение задачи с комментариями представлено в листинге 9.9. На рис. 9.7 можно увидеть график решения.

```
%Функция правой части жёсткой системы диф. уравнений.
function dx=syst1(t,x)
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];
dx=B*x;
end
```

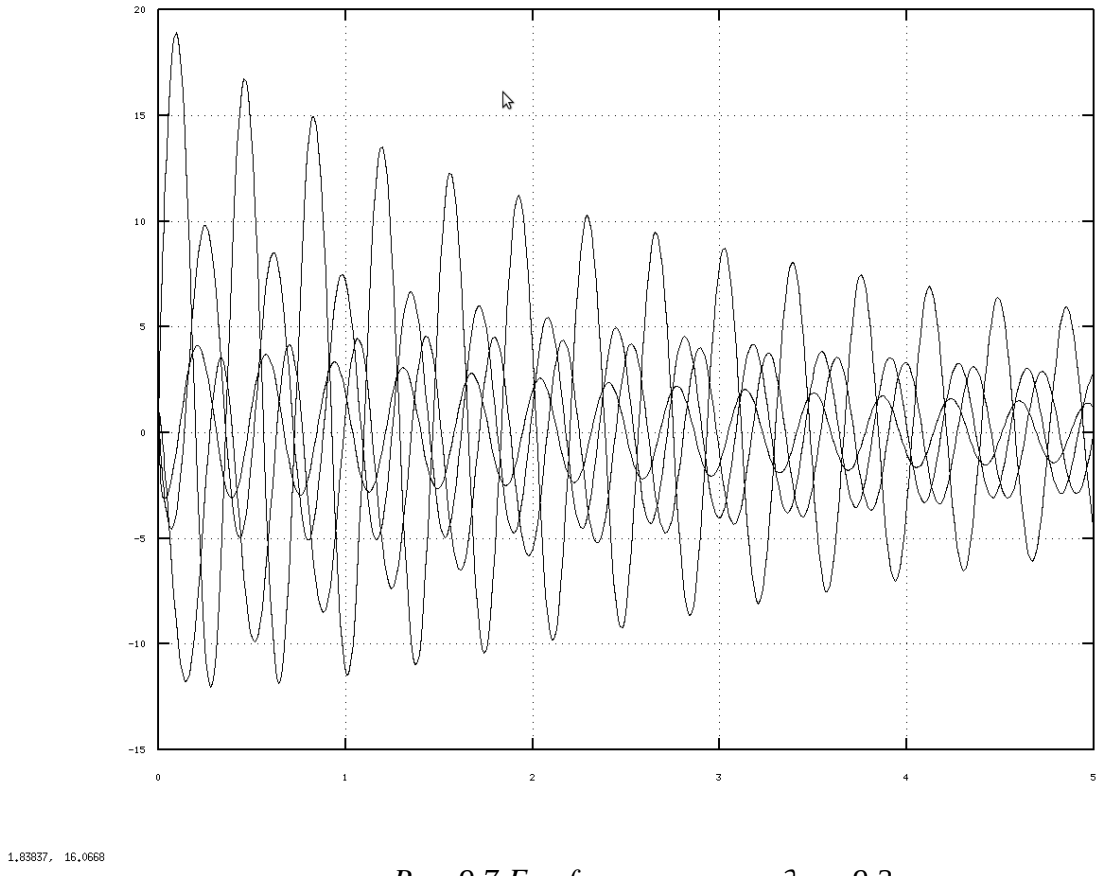


Рис. 9.7 График решения задачи 9.2

```
%Определение параметров управления ходом решения жёсткой
% системы дифференциальных уравнений.
%RelTol – относительная точность решения 1E-8,
%AbsTol – абсолютная точность решения 1E-8,
%InitialStep – начальное значение шага изменения
%независимой переменной 0.02,
%MaxStep – максимальное значение шага изменения
%независимой переменной 0.1.
par=odeset ("RelTol", 1e-8, "AbsTol", 1e-8,...
                    'InitialStep',0.02,'MaxStep',0.1);
%Решение жёсткой системы дифференциальных уравнений.
[A,B]=ode2r(@syst1,[0 5],[1;1;1;1]);
%Построение графика решения.
plot(A,B,'-k'); grid on;
```

Листинг 9.9.

Этим примером мы заканчиваем краткое описание возможностей Octave для решения дифференциальных уравнений. Однако, следует помнить о следующем: решение реального дифференциального уравнения (а тем более системы) достаточно сложная математическая задача. Для её решения недостаточно знания синтаксиса функций Octave, необходимо достаточно глубоко знать математические методы решения подобных задач. При решении дифференциальных уравнений необходимо определить метод решения и только потом пытаться использовать встроенные функции или писать свои. Авторы не случайно достаточно подробно напомнили читателю основные численные методы решения дифференциальных уравнений и систем. На наш взгляд, без знания численных и аналитических методов решения дифференциальных уравнений достаточно проблематично решить реальную задачу.

Кроме того, следует помнить, что функциями `ode23`, `ode45`, `ode2r`, `ode5r` возможности пакета не ограничиваются. Octave предоставляет достаточное количество функций для решения дифференциальных уравнений различного вида. Они подробно описаны в справке консольной версии приложения[12].

Множество функций решени дифференциальных уравнений находится в пакете расширений *odepkg*. Краткое описание функций этого пакета на английском языке с некоторыми примерами приведено на странице

http://octave.sourceforge.net/odepkg/overview.html.

12 Ещё раз напоминаем читателю, что справка по Octave, доступная из оболочки qtoctave недостаточно полная.

10. Решение оптимизационных задач в Octave

В данной главе рассматриваются решение задач поиска минимума (максимума) в Octave. В первой главе на примерах решения практических задач рассматривается функция sqp, предназначенная для поиска минимума функции одной или нескольких переменных с ограничениями. Вторая глава целиком посвящена задачам линейного программирования.

Изучение оптимизационных задач начнём с обычных задач поиска минимума (максимума) функции одной или нескольких переменных.

10.1 Поиск экстремума функции

Для решения классических оптимизационных задач с ограничениями в Octave можно воспользоваться следующей функцией

`[x,obj,info,iter]=spq(x0,phi,g,h,lb,ub,maxiter,tolerance)`,

которая предназначена для решения следующей оптимизационной задачи.

Найти минимум функции $\varphi(x)$ при следующих ограничениях:

$$\begin{cases} g(x)=0, \\ h(x)\geq 0, \\ lb\leq x\leq ub. \end{cases}$$

Функция `sqp` при решении задачи оптимизации использует метод квадратичного программирования.

Аргументами функции `sqp` являются

`x0` – начальное приближение значения `x`,

`phi` – оптимизируемая функция $\varphi(x)$,

`g` и `h` – функции ограничений $g(x)=0$ и $h(x)\geq 0$ соответственно,

`lb` и `ub` – верхняя и нижняя границы ограничения $lb\leq x\leq ub$,

`maxiter` – максимальное количество итераций, используемое при решении оптимизационной задачи, по умолчанию эта величина равна 100,

`tolerance` – точность ε , определяющая окончание вычислений, вычисления прекращаются при достижении точности $\sqrt{(\varepsilon)}$.

Функция `sqp` возвращает следующие значения

`x` – точка, в которой функция, достигает своего минимального значения,

`obj` – минимальное значение функции,

`info` – параметр, характеризующий корректность решения оптимизационной задачи, (если функция sqp возвращает значение $info=101$, то задача решена правильно),

`iter` – реальное количество итераций при решении задачи.

Рассмотрим несколько примеров использования функции `sqp` при решении задач поиска экстремума функции одной переменной без ограничений.

ЗАДАЧА 10.1. Найти минимум функции $\varphi(x)=x^4+3x^3-13x^2-6x+26$

При решении задачи оптимизации с помощью функции `sqp` необходимо иметь точку начального приближения. Построим график функции $\varphi(x)$ (см. рис. 10.1). Из графика видно, что функция имеет минимум в окрестности точки *x=-4*. В качестве точки начального приближения выберем x0=-3. Решение задачи представлено на листинге 10.1.

```
function obj = phi (x)
obj = x^4+3*x^3-13*x^2-6*x+26;
endfunction
[x, obj, info, iter]=sqp (-3, @phi)
>>>%Результаты решения:
```

```
>>>[x, obj, info, iter]=sqp (-3, @phi)
x = -3.8407
obj = -95.089
info = 101
iter = 5
```

Листинг 10.1

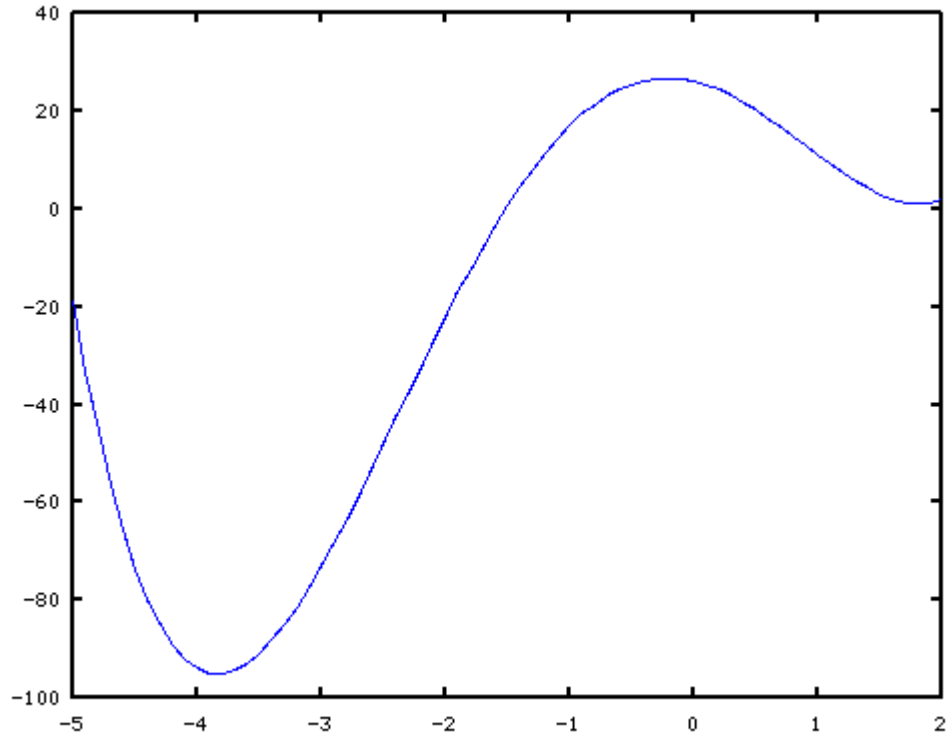


Рис. 10.1 График функции $\varphi(x)=x^4+3x^3-13x^2-6x+26$

Минимум функции $ivarphi(x)=-95.089$ достигается в точке $x=-3.8407$, количество итераций равно 5, параметр $info=101$ свидетельствует о корректном решении задачи поиска минимума $ivarphi(x)=x^4+3x^3-13x^2-6x+26$.

Рассмотрим пример поиска минимума функции нескольких переменных.

ЗАДАЧА 10.2. Найти минимум функции Розенброка $f(x,y)=N(y-x^2)^2+(1-x^2)^2$

Построим график функции Розенброка (листинг 10.2).

```
[x y]=meshgrid(-2:0.1:2,2:-0.1:-2);
z=20*(y-x.^2).^2+(1-x).^2;
surf(x,y,z);
```

Листинг 10.2

График полученной поверхности при N=20 приведён на рис. 10.2.

Как известно, функция Розенброка имеет минимум в точке (1,1) равный 0. В виду своей специфики функция Розенброка является тестовой для алгоритмов минимизации. Найдём минимум этой функции с помощью функции `sqp` (см. листинг 10.3). При решении задач на экстремум функций многих переменных следует следующие особенности синтаксиса при определении оптимизируемой функции. Аргументом функции многих переменных (в нашем

случае – её имя `r`) является массив *x*, первая переменная имеет имя *x(1)*, вторая *x(2)* и т. д. Если имя аргумента функции многих переменных будет другим – допустим *m*, то изменятся и имена переменных: *m(1)*, *m(2)*, *m(3)* и т. д.

```
function y=r(x)
y=20*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
x0=[0;0];
[x, obj, info, iter]=sqp(x0,@r)
>>>%Решение задачи:
>>>x =
 1.00000
 1.00000
obj = 7.1675e-13
info = 101
iter = 14
```

Листинг 10.3

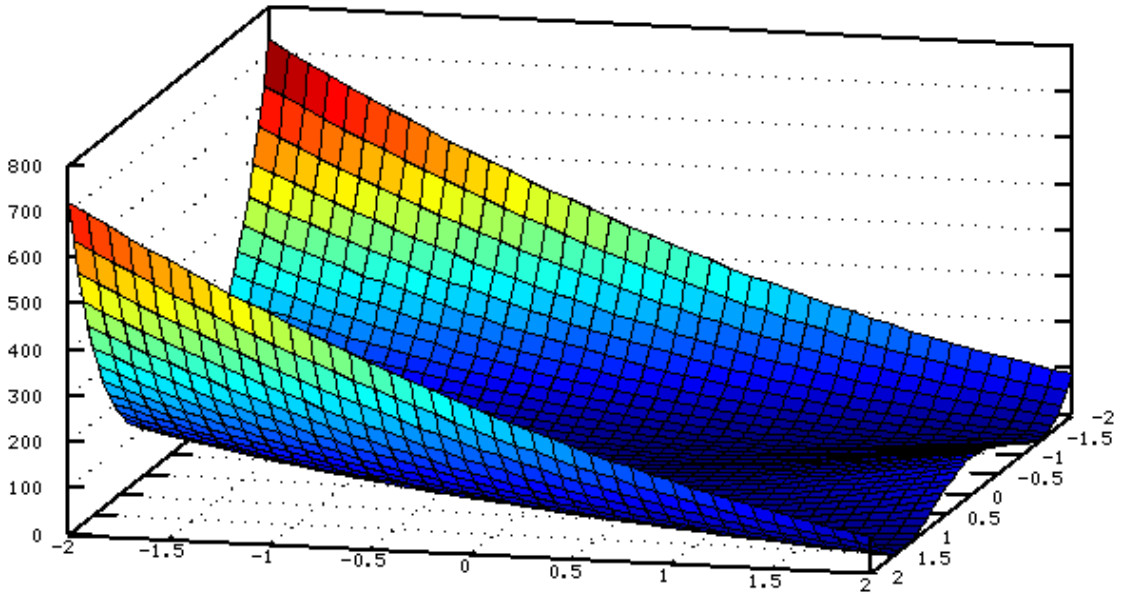


Рис. 10.2: График функции Розенброка

Как и следовало ожидать, функция `sqp` нашла минимум в точке (1,1), само значение 0 найдено достаточно точно ($7.2 \cdot 10^{-13}$). Значение `info=101` говорит о корректном решении задачи, для нахождения минимального значения функции Розенброка потребовалось `14` итераций.

Таким образом, функция `sqp` предназначена для поиска минимума функций (как одной, так и нескольких переменных) с различными ограничениями.

Рассмотрим несколько задач поиска экстремума с ограничениями.

ЗАДАЧА 10.3. Найти максимум и минимум функции $F=(x-3)^2-(y-4)^2$ при

ограничениях $\begin{cases} 3x+2y\geq 7, \\ 10x-y\leq 8, \\ -18x+4y\leq 12, \\ x\geq 0, \\ y\geq 0. \end{cases}$ [1].

В функции `sqp` все ограничения должны быть вида ≥0. Поэтому умножим второе и третье ограничение на *-1*, и перенесём всё в левую часть неравенств. В результате этих несложных преобразований система ограничений примет вид:

$$\boldsymbol{g}(x)=\begin{cases} 3x+2y-7\geq 0, \\ -10x+y+8\geq 0, \\ 18x-4y+12\geq 0, \\ x\geq 0, \\ y\geq 0. \end{cases} .$$

Последовательно рассмотрим задачу на минимум и максимум.

В задаче на минимум функция **F(x)** будет записана так

```
function y=f(x)
 y=(-x(1)-3)^2+(-x(2)-4)^2;
endfunction
```

Вектор-функцию ограничений **g(x)** можно записать так

```
function r = g (x)
 r=[3*x(1)+3*x(2)-7;
 -10*x(1)+x(2)+8;
 18*x(1)-4*x(2)+12;
 x(1);
 x(2)];
endfunction
```

Решение задачи минимум представлено на листинге 10.4.

```
function y=f(x)
 y=(x(1)-3)^2+(x(2)-4)^2;
endfunction
function r = g (x)
 r=[3*x(1)+3*x(2)-7;
 -10*x(1)+x(2)+8;
 18*x(1)-4*x(2)+12;
 x(1);
 x(2)];
endfunction
x0=[0;0]; [x, obj, info, iter]=sqp(x0,@f,[],@g)
```

Листинг 10.4

Минимум 3.2079 достигается в точке (1.2178, 4.1782), значение `info=101` говорит о корректном решении задачи. Для нахождения минимального значения потребовалось всего `5` итерации.

Теперь рассмотрим решение задачи на максимум. Функция `sqp` ищет только минимум. Поэтому вспомним, как задача на максимум сводится к задаче на минимум:

$$max\, f(x)=-min\, f(-x) \quad .$$

Введём дополнительную функцию

```
function y=f1(x)
y=-f(-x);
endfunction
```

Полный текст решения задачи на максимум представлен на листинге 10.5.

```
function y=f(x)
y=(x(1)-3)^2+(x(2)-4)^2;
endfunction
function y=f1(x)
y=-f(-x);
endfunction
function r = g (x)
r=[3*x(1)+2*x(2)-7;
-10*x(1)+x(2)+8;
18*x(1)-4*x(2)+12;
x(1);
x(2)];
endfunction
x0=[0;0];
[x, obj, info, iter]=sqp(x0,@f1,[],@g)
maximum=f(x)
>>>%Результаты работы программы:
>>>x =
2.0000
12.0000
obj = -281.00
info = 101
iter = 3
>>>maximum = 65.000
```

Листинг 10.5

Максимум достигается в точке (2,12), его величина равна 65.

ЗАДАЧА 10.4. План производства изделий трёх типов составляет 120 деталей(x_1 – количество изделий первого вида, x_2 – количество изделий второго вида, x_3 – количество изделий третьего вида). Изделия можно изготовить тремя способами. При первом технологическом способе производят изделия первого типа и затраты составляют $4x_1+ x_1^2$. Второй технологический способ предназначен для производства изделий второго типа и затраты составляют $8x_2+ x_2^2$. Третий способ позволяет производить изделия третьего типа и затраты в нём можно рассчитать по формуле x_3^2 . Определить, сколько изделий каждого типа надо изготовить, чтобы затраты были минимальными [1].

Сформулируем эту задачу, как задачу оптимизации. Найти минимум функции $f(x_1, x_2)=4x_1+x_1^2+8x_2+x_2^2+x_3^2$ при следующих ограничениях $x_1+ x_2+ x_3=120, x_1\ge 0, x_2\ge 0, x_3\ge 0$. Функция цели будет представлена следующим образом

```
function y=f(x)
y=4*x(1)+x(1)*x(1)+8*x(2)+x(2)*x(2) +x(3)*x(3);
endfunction
```

Две функции ограничений $g(x)=0$ и $\varphi(x)\ge 0$ можно будет записать так.

```
function z=g(x)
z=x(1)+x(2)+x(3)-120;
endfunction
function u=fi(x)
fi=[x(1);x(2);x(3)];
endfunction
```

На листинге 10.6 представлена программа решения задачи 10.2.

```
function y=f(x)
y=4*x(1)+x(1)*x(1)+8*x(2)+x(2)*x(2) +x(3)*x(3);
endfunction
function z=g(x)
z=x(1)+x(2)+x(3)-120;
endfunction
function u=fi(x)
fi=[x(1);x(2);x(3)];
endfunction
x0=[0;0;0]; [x, obj, info, iter]=sqp(x0,@f,@g)
>>>%Результаты решения:
>>>x =
40.000
38.000
42.000
obj = 5272.0
info = 101
iter = 8
```

Листинг 10.6

Минимальные затраты составят 5272 денежных единицы, при этом будет произведено 40 изделий первого вида, 38 – второго и 42 – третьего. Для решения задачи было проведено 8 итераций.

ЗАДАЧА 10.5. Найти максимум функции $f=-x_1^2-x_2^2$ при ограничениях $(x_1-7)^2+(x_2-7)^2\le 18$, $x_1\ge 0$, $x_2\ge 0$ [1].

Здесь необходимо свести задачу на максимум к задаче на минимум, а также путём, умножения на -1, заменить знак в неравенстве.

Текст программы решения задачи в Octave:

```
function y=f(x)
y=-x(1)*x(1)-x(2)*x(2) ;
endfunction
function y=f1(x)
y=-f(-x) ;
endfunction
function u=fi(x)
u=[-(x(1)-7)^2-(x(2)-7)^2+18;
x(1);
x(2)];
endfunction
x0=[0;0];
[xopt, obj, info, iter]=sqp(x0,@f1,[],@fi)
f(xopt)
>>>%Результаты решения:
>>>xopt =
4.0000
4.0000
obj=32.000
info=101
iter=8
>>>ans=-32.000
```

Листинг 10.7.

Функция достигает своего максимального значения $max\, f(x_1, x_2) = -32$ в точке $(x_1 = 4, x_2 = 4)$.

Следующим классом оптимизационных задач, рассматриваемых в этой главе, будут задачи линейного программирования (ЗЛП).

10.2 Решение задач линейного программирования

Эти задачи встречаются во многих отраслях знаний. Алгоритмы их решения хорошо известны. Эти алгоритмы реализованы во многих, как проприетарных, так и свободных, математических пакетах. Не является исключением и Octave. Но перед тем, как рассмотреть решение задач линейного программирования в Octave, давайте вспомним, что такое задача линейного программирования.

10.2.1 Задача линейного программирования

Знакомство с задачами линейного программирования начнем на примере задачи об оптимальном рационе.

Задача об оптимальном рационе. Имеется четыре вида продуктов питания: *П1*, *П2*, *П3*, *П4*. Известна стоимость единицы каждого продукта c_1, c_2, c_3, c_4 . Из этих продуктов необходимо составить пищевой рацион, который должен содержать не менее b_1 единиц белков, не менее b_2 единиц углеводов, не менее b_3 единиц жиров. Причем известно, в единице продукта *П1* содержится a_{11} единиц белков, a_{12} единиц углеводов и a_{13} единиц жиров и т.д. (табл. 10.1).

Таблица 10.1. Содержимое белков, углеводов и жиров в продуктах

Элемент	белки	углеводы	жиры
П1	a_{11}	a_{12}	a_{11}
П2	a_{21}	a_{22}	a_{23}
П3	a_{31}	a_{32}	a_{33}
П4	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 – количества продуктов *П1*, *П2*, *П3*, *П4*. Общая стоимость рациона равна

$$L = c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 = \sum_{i=1}^{4} c_i x_i \quad . \tag{10.1}$$

Сформулируем ограничение на количество белков, углеводов и жиров в виде неравенств. В одной единице продукта *П1* содержится a_{11} единиц белков, в x_1 единицах - $a_{11} x_1$, в x_2 единицах продукта *П2* содержится $a_{21} x_2$ единиц белка и т.д. Следовательно общее количество белков во всех четырех типах продуктов равно $\sum_{j=1}^{4} a_{j1} x_j$ и должно быть не больше b_1 . Получаем первое ограничение

$$a_{11} x_1 + a_{21} x_2 + a_{31} x_3 + a_{41} x_4 \le b_1 \tag{10.2}$$

Аналогичные ограничения для жиров и углеводов имеют вид:

$$a_{12}x_1 + a_{22}x_2 + a_{32}x_3 + a_{42}x_4 \le b_2 \qquad (10.3)$$

$$a_{13}x_1 + a_{23}x_2 + a_{33}x_3 + a_{43}x_4 \le b_3 \qquad (10.4)$$

Принимаем во внимание, что x_1, x_2, x_3, x_4 положительные значения, получим еще четыре ограничения

$$x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0 \qquad (10.5)$$

Таким образом, задачу о рациональном рационе можно сформулировать следующим образом: найти значения переменных x_1, x_2, x_3, x_4, удовлетворяющие системе ограничений (10.2) – (10.5), при которых линейная функция (10.1) принимала бы минимальное значение.

Задача об оптимальном рационе является задачей линейного программирования, функция (10.1) называется функцией цели, а ограничения (10.2) – (10.5) 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 положительные значения $x_1, x_2, \ldots, x_n$, при которых *функция цели L* (10.6) достигает своего минимального значения и удовлетворяет системе линейных ограничений (10.7):

$$L = c_1x_1 + c_2x_2 + \ldots + c_nx_n = \sum_{i=1}^{n} c_i x_i \ , \qquad (10.6)$$

$$\sum_{j=1}^{n} a_{ij}x_j \le b_i, i = 1, \ldots m \ . \qquad (10.7)$$

Если в задачу линейного программирования добавляется ограничение целочисленности значений *x*, то мы получаем задачу *целочисленного программирования*.

Octave позволяет решать задачи линейной оптимизации с ограничениями в более общей формулировке.

Найти такие положительные значения $x_1, x_2, \ldots, x_n$, при которых *функция цели L* (10.6) достигает своего минимального (максимального) значения и удовлетворяет системе линейных ограничений. Система ограничений может быть представлена неравенствами (10.8) или (10.9). При этом значения *x* могут быть, как вещественными, так и целочисленными, как положительными, так и отрицательными.

$$\sum_{j=1}^{n} a_{ij}x_j \le b_i, \qquad (10.8)$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m. \tag{10.9}$$

Рассмотрим решение задач линейного программирования в Octave.

10.2.2 Решение задач линейного программирования в Octave

Для решения задач линейного программирования в Octave существует функция

```
[xopt, fmin, status, extra] =
          qlpk(c, a, b, lb, ub, ctype, vartype, sense, param)
```

Здесь

`c` – вектор-столбец, включающий в себя коэффициенты при неизвестных функции цели, размерность вектора `c` равна количеству неизвестных `n` в задаче линейного программирования;

`a` – матрица при неизвестных из левой части системы ограничений, количество строк матрицы равно количеству ограничений `m`, а количество столбцов совпадает с количеством неизвестных `n`;

`b` – вектор-столбец, содержащий свободные члены системы ограничений, размерность вектора равна количеству ограничений *m*;

`lb` – вектор-столбец размерности `n`, содержащий верхнюю систему ограничений (**x**>**lb**), по умолчанию `lb` – вектор-столбец, состоящий из нулей;

`ub` – вектор-столбец размерности `n`, содержащий нижнюю систему ограничений (**x<ub**), по умолчанию верхняя система ограничений отсутствует, подразумевается, что все знаения вектора `ub` равны +∞;

`ctype` – массив символов размерности `n`, определяющий тип ограничения (например, (10.7) или (10.9)), элементы этого вектора могут принимать одно из следующих значений:

`"F"` – ограничение будет проигнорировано,

`"U"` – ограничение с верхней границей (`(A(i,:)*x <= b(i))`,

`"S"` – ограничение в виде равенства (`(A(i,:)*x = b(i))`,

`"L"` – ограничение с верхней границей (`(A(i,:)*x >= b(i))`,

`"D"` – двойное ограничение (`(A(i,:)*x <= b(i)` и `(A(i,:)*x >= b(i))`;

`vartype` – массив символов размерности `n`, который определяет тип переменной x_i,

"C" – вещественная переменная,

"I" – целочисленная переменная;

`sense` – значение, определяющее тип задачи оптимизации:

- 1 – задача минимизации,
- -1 – задача максимизации;

`param` – структура, определяющая параметры оптимизационных алгоритмов, при обращении к функции `glpk`.

Во многих случаях достаточно значений структуры `param` по умолчанию, в этом случае последний параметр в функции `glpk` можно не указывать. Подробное описание структуры `param` выходит за рамки книги, в случае необходимости его использования авторы рекомендуют обратиться к встроенной справке Octave.

Функция `glpk` возвращает следующие значения:

`xopt` – массив значений x, при котором функция цели `L` принимает оптимальное значение;

`fmin` – оптимальное значение функции цели;

`status` – переменная, определяющая как решена задача оптимизации; при `status=180`, решение найдено и задача оптимизации решена полностью[13];

`extra` – структура, включающая следующие поля:

`lambda` – множители Лагранжа;

`time` – время в секундах, затраченное на решение задачи;

`mem` – память в байтах, которая была использована при решении задачи (значение недоступно, если была использована библиотека линейного программирования GLPK 4.15[14] и выше).

Рассмотрим несколько примеров решения задач линейного программирования.

ЗАДАЧА 10.6. Найти такие значения переменных x_1, x_2, x_3, x_4 при которых функция цели $L=-x_2-2x_3+4x_4$ достигает своего минимального значения и удовлетворяются ограничения:

$$\begin{aligned} &3x_1-x_2\le 2\\ &x_2-2x_3\le -1\\ &4x_3-x_4\le 3\\ &5x_1+x_4\ge 6\\ &x_1\ge 0, x_2\ge 0, x_3\ge 0, x_4\ge 0 \end{aligned} \quad .$$

Сформируем параметры функции `gplk`

$c=\begin{bmatrix} 0\\ -1\\ -2\\ 4 \end{bmatrix}$ – коэффициенты при неизвестных функции цели,

$a=\begin{bmatrix} 3 & -1 & 0 & 0\\ 0 & 1 & -2 & 0\\ 0 & 0 & 4 & -1\\ 5 & 0 & 0 & 1 \end{bmatrix}$ – матрица системы ограничений,

$b=\begin{bmatrix} 2\\ -1\\ 3\\ 6 \end{bmatrix}$ – свободные члены системы ограничений,

ctype="UUUL" – массив символов, определяющий тип ограничения[15],

vartype="CCCC" – массив, определяющий тип переменной, в данном случае все переменные вещественные,

sense=1 – задача на минимум.

Текст программы решения задачи приведен в листинге 10.8.

```
c=[0;-1;-2;4];
a=[3 -1 0 0; 0 1 -2 0;0 0 4 -1;5 0 0 1];
b=[2; -1; 3; 6];
ctype="UUUL";
vartype="CCCC";
sense=1;
[xmin, fmin, status] =glpk (c, a, b, [0 ;0 ;0; 0], [], ctype, vartype, sense)
```

13 Если $status \neq 180$, то полученному решению доверять нельзя, подробнее о значениях переменной status в этом случае можно прочитать в справке.

14 В последней, на момент написания книги, версии octave использовалась библиотека GLPK версии 4.38.

15 Первые три ограничения типа «меньше», четвертое – типа «больше»

```
>>>%Результаты решения задачи:
>>>xmin =
1.00000
1.00000
1.00000
1.00000
fmin = 1.00000
status = 180
```

Листинг 10.8

Минимальное значение *L=1* достигается при $x=\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$. Значение переменной *status* равно 180, что свидетельствует о корректном решении задачи линейного программирования.

ЗАДАЧА 10.7. Найти такие значения переменных x_1, x_2, x_3, при которых функция цели $L=-5+x_1-x_2-3x_3$ достигает своего минимального значения и удовлетворяются ограничения:

$$\begin{aligned} &x_1+x_2\geq 2\\ &x_1-x_2\leq 0\\ &x_1+x_3\geq 2\\ &x_1+x_2-x_3\leq 3\\ &x_1\geq 0, x_2\geq 0, x_3\geq 0 \end{aligned} \quad .$$

Сформируем параметры функции `gplk`

$c=\begin{bmatrix}1\\-1\\-3\end{bmatrix}$ – коэффициенты при неизвестных функции цели,

$a=\begin{bmatrix}1 & 1 & 0\\1 & -1 & 0\\1 & 0 & 1\\1 & 1 & -1\end{bmatrix}$ – матрица системы ограничений (три переменных и четыре ограничения),

$b=\begin{bmatrix}2\\0\\2\\3\end{bmatrix}$ – свободные члены системы ограничений,

ctype="LULU" – массив символов, определяющий тип ограничения[16],

vartype="CCC" – массив, определяющий тип переменной, в данном случае все переменные вещественные,

sense=-1 – задача на максимум.

Текст программы решения задачи приведен на листинге .

```
c=[1;-1;-3];
a=[1 1 0; 1 -1 0;1 0 1;1 1 -1]; b=[2; 0;2; 3];
ctype="LULU";
vartype="CCC";
sense=-1;
[xmax,fmax,status]=glpk(c, a, b, [],[], ctype, vartype, sense)
```

16 Первое и третье ограничения типа «больше», второе четвертое – типа «меньше».

```
>>>%Результаты решения:
>>>xmax =
1.66667
1.66667
0.33333
fmax = -1.0000
status = 180
```

Листинг 10.9

Минимальное значение[17] *L=-6* достигается при $x=\begin{bmatrix}1.66667\\1.66667\\0.33333\end{bmatrix}$. Значение переменной *status* равно 180, что свидетельствует о корректном решении задачи линейного программирования.

Решим задачу 10.9, как задачу *целочисленного программирования* (листинг 10.10)

```
c=[1;-1;-3];
a=[1 1 0; 1 -1 0;1 0 1;1 1 -1];
b=[2; 0;2; 3];
ctype="LULU"; vartype="III"; sense=-1;
[xmax,fmax,status]=glpk(c, a, b, [],[], ctype, vartype, sense)
>>>%Результаты решения:
>>>xmin =
2
2
1
fmin = -3
status = 171
```

Листинг 10.10

Значение *status=171* свидетельствует о корректном решении задачи целочисленного программирования, значение *L=-8* достигается при $x=\begin{bmatrix}2\\2\\1\end{bmatrix}$.

ЗАДАЧА 10.8. Туристическая фирма заключила контракт с двумя турбазами на одном из черноморских курортов, рассчитанных, соответственно, на 195 и 165 человек. Туристам для посещения предлагается дельфинарий в городе, ботанический сад и походы в горы.

Составить маршрут движения туристов так, чтобы это обошлось возможно дешевле, если дельфинарий принимает в день 90 организованных туристов, ботанический сад – 170, а в горы в один день могут пойти 105 человек.

Стоимость одного посещения выражается табл. 10.2.

Таблица 10.2. Исходные данные к задаче 10.8

Турбаза	Дельфинарий	Ботанический сад	Поход в горы
1	5	9	20
2	10	12	24

Для решения задачи введем следующие обозначения:

x1 – число туристов первой турбазы, посещающих дельфинарий;

x2 – число туристов первой турбазы, посещающих ботанический сад;

x3 – число туристов первой турбазы, отправляющихся в поход;

17 Обратите внимание, что fmin было равно -1, а потом из него необходимо было вычесть -5

x4 – число туристов второй турбазы, посещающих дельфинарий;

x5 – число туристов второй турбазы, посещающих ботанический сад;

x6 – число туристов второй турбазы, отправляющихся в поход.

Составим функцию цели, заключающуюся минимизации стоимости мероприятий фирмы:

$Z=5x1+9x2+20x3+10x4+12x5+24x6$.

Руководствуясь условием задачи, определим ограничения:

$x1 + x4 \le 90$;

$x2 + x5 \le 170$;

$x3 + x6 \le 105$;

$x1 + x2 + x3 = 195$;

$x4 + x5 + x6 = 165$.

Кроме того, количество туристов, участвующих в мероприятиях, не может быть отрицательным: $x1 \ge 0$, $x2 \ge 0$, $x3 \ge 0$, $x4 \ge 0$, $x5 \ge 0$, $x6 \ge 0$.

Кроме того, необходимо помнить, что это задача целочисленного программирования (количество туристов — число целое).

В массиве *x* будут хранится значения *x1*, *x2*, *x3*, *x4*, *x5* и *x6*.

Сформируем параметры функции `gplk`:

$c=\begin{bmatrix}15\\9\\20\\10\\12\\24\end{bmatrix}$ – коэффициенты при неизвестных функции цели,

$a=\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\\1&1&1&0&0&0\\0&0&0&1&1&1\end{bmatrix}$ – матрица системы ограничений (шесть переменных и пять ограничений),

$b=\begin{bmatrix}90\\170\\105\\195\\165\end{bmatrix}$ – свободные члены системы ограничений,

ctype="UUUSS" – массив символов, определяющий тип ограничения[18],

vartype="IIIII" – массив, определяющий тип переменной, в данном случае все переменные целые (задача целочисленного программирования),

sense=1 – задача на минимум.

Решение задачи представлено в листинге 10.11.

```
c=[5;9;20;10;12;24];b=[90; 170; 105; 195; 165];
a=[1 0 0 1 0 0; 0 1 0 0 1 0;0 0 1 0 0 1;1 1 1 0 0 0 ;0 0 0 1 1 1];
    ctype="UUUSS";
    vartype="IIIIII";
    sense=1;
    [xmin, fmin, status]=glpk(c,a,b,[], [], ctype, vartype, sense)
    >>>%Результаты решения:
```

18 Первые три ограничения типа «меньше», четвертое и пятое – типа «равно».

```
>>>xmin =
90
5
100
0
165
0
fmin = 4475
status = 171
```

Листинг 10.11

Значение переменной *status=171* свидетельствует о корректном решении задачи целочисленного программирования.

В результате получилось следующее решение:

90 туристов первой турбазы посетят дельфинарий, 5 туристов первой турбазы и все 165 второй турбазы поедут в ботанический сад, *100* туристов первой турбазы отправятся в поход. Стоимость мероприятия составит 4475.

ЗАДАЧА 10.9. Для изготовления трех видов изделий (А, Б, В) используется токарное, фрезерное, шлифовальное и сварочное оборудование. Затраты времени на обработку одного изделия каждого типа представлены в табл. 10.3. Общий фонд рабочего времени каждого вида оборудования и прибыль от реализации изделий каждого типа представлены в этой же таблице. Составить план выпуска изделий для достижения максимальной прибыли [1].

Таблица 10.3. Затраты времени на обработку одного изделия

Тип оборудования	Затраты времени на обработку одного изделия вида (станко-ч)			Общий фонд времени работы оборудования (ч)
	А	Б	В	
Фрезерное	2	4	5	120
Токарное	1	8	6	280
Сварочное	7	4	5	240
Шлифовальное	4	6	7	360
Прибыль (тыс. грн)	10	14	12	

Пусть x_1 – количество изделий вида А, x_2 – количество изделий вида Б, x_3 – количество изделий вида В.

Прибыль от реализации всех изделий составляет

$$L = 10x_1 + 14x_2 + 12x_3 \quad . \tag{10.10}$$

Общий фонд рабочего времени фрезерного оборудования составляет $2x_1 + 4x_2 + 5x_3$. Эта величина не должна превышать 120 часов, следовательно имеем неравенство

$$2x_1 + 4x_2 + 5x_3 \le 120 \tag{10.11}$$

Запишем аналогичные ограничения для фонда рабочего времени токарного, сварочного, шлифовального оборудования:

$$\begin{cases} x_1+8x_2+6x_3\le 280, \\ 7x_1+4x_2+5x_3<240, \\ 4x_1+6x_2+7x_3\le 360. \end{cases} \qquad (10.12)$$

Таким образом, имеем следующую задачу линейного программирования.

Найти такие положительные значения x_1 , x_2 , x_3 при которых функция цели L (10.10) достигает максимального значения и выполняются ограничения (10.11)-(10.12).

Теперь решим эту задачу в Octave с помощью функции `gplk`.

Сформируем параметры функции `gplk`

$c=\begin{bmatrix}10\\14\\12\end{bmatrix}$ – коэффициенты при неизвестных функции цели, $a=\begin{bmatrix}2&4&5\\1&8&6\\7&4&5\\4&6&7\end{bmatrix}$ – матрица системы ограничений (три переменных и четыре ограничений), $b=\begin{bmatrix}120\\280\\240\\360\end{bmatrix}$ – свободные члены системы ограничений, *ctype="UUUГ"* – массив символов, определяющий тип ограничения[19], *vartype="III"* – массив, определяющий тип переменной, в данном случае все переменные целые, *sense=-1* – задача на максимум.

Решение задачи в Octave представлено на листинге

```
c=[10;14;12];
a=[2 4 5; 1 8 6;7 4 5;4 6 7];
b=[120; 280; 240; 360];
ctype="UUUU";
vartype="III";
sense=-1;
[xmax, fmax, status] =glpk (c, a, b, [0 ;0 ;0], [], ctype, vartype, sense)
>>>%Результаты решения:
>>>xmax =
24
18
0
fmax = 492
status = 171
```

Листинг 10.12

Таким образом для получения максимальной прибыли (*fmax=492*) необходимо произвести 24 единицы изделия типа А и 18 единиц изделия типа Б. Значение параметра *status=171* говорит о корректности решения задачи линейного программирования.

ЗАДАЧА 10.10. Для изготовления четырёх видов изделий используется токарное, фрезерное, сверлильное, расточное, шлифовальное оборудование, а также комплектующие изделия. Сборка изделий требует сборочно-наладочных работ. В таблице 10.4 представлены: нормы затрат ресурсов на изготовление различных изделий, наличие каждого из ресурсов, прибыль от реализации одного изделия, ограничения на выпуск изделий второго и третьего типа [1]. Сформировать план выпуска продукции для достижения максимальной прибыли.

19 Все четыре ограничения типа «меньше».

Таблица 10.4. Данные к задаче 10.10

Ресурсы	Нормы затрат на одно изделие				Общий объём ресурсов
	1	2	3	4	
Производительность оборудования (человеко-ч)					
токарного	550		620		64270
фрезерного	40	30	20	20	4800
сверлильного	86	110	150	52	22360
расточного	160	92	158	128	26240
шлифовального		158	30	50	7900
Комплектующие изделия (шт.)	3	4	3	3	520
Сборочно-наладочные работы (человеко-ч)	4.5	4.5	4.5	4.5	720
Прибыль от реализации одного изделия (тыс. руб.)	315	278	573	370	
Выпуск					
минимальный		40			
максимальный			120		

Пусть x_1 – количество изделий первого вида, x_2 – количество изделий второго вида, x_3 и x_4 – количество изделий третьего и четвёртого вида соответственно.

Тогда прибыль от реализации всех изделий вычисляется по формуле

$$L = 315x_1 + 278x_2 + 573x_3 + 370x_4 \quad . \tag{10.13}$$

Ограничения на фонд рабочего времени формируют следующие ограничения

$$\begin{cases} 550x_1 + 620x_3 \le 64270, \\ 30x_1 + 30x_2 + 20x_3 + 20x_4 \le 4800, \\ 86x_1 + 110x_2 + 150x_3 + 52x_4 \le 22360, \\ 160x_1 + 92x_2 + 158x_3 + 128x_4 \le 26240, \\ 158x_2 + 30x_3 + 50x_4 \le 7900. \end{cases} \quad . \tag{10.14}$$

Ограничение на возможное использование комплектующих изделий

$$3x_1 + 4x_2 + 3x_3 + 3x_4 \le 520 \quad . \tag{10.15}$$

Ограничение на выполнение сборочно-наладочных работ

$$4.5x_1 + 4.5x_2 + 4.5x_3 + 4.5x_4 \le 720 \quad . \tag{10.16}$$

Ограничения на возможный выпуск изделий каждого вида

$$x2 \ge 40, x3 \le 120, x1 \ge 0, x3 \ge 0, x4 \ge 0 \quad . \tag{10.17}$$

Сформулируем задачу линейного программирования.

Найти значения x_1 , x_2 , x_3 и x_4 при которых функция цели L (10.13) достигает своего максимального значения и выполняются ограничения (10.4)-(10.3).

Рассматриваемая задача из широко известной книги [1] была интересна авторам в связи с тем, что еще 25 лет назад для решения задач подобной сложности использовали большие ЭВМ и специализированные пакеты решения оптимизационных задач. На подготовку данные и решение её затрачивался не один час. Мы же попробуем решить эту задачу в Octave и посмотрим сколько времени на это уйдёт.

Сформируем параметры функции `gplk`.

$c = \begin{bmatrix} 315 \\ 278 \\ 573 \\ 370 \end{bmatrix}$ – коэффициенты при неизвестных функции цели,

$a = \begin{pmatrix} 550 & 0 & 620 & 0 \\ 40 & 30 & 20 & 20 \\ 86 & 110 & 150 & 52 \\ 160 & 92 & 158 & 128 \\ 0 & 158 & 30 & 50 \\ 3 & 4 & 3 & 3 \\ 4.5 & 4.5 & 4.5 & 4.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ – матрица системы ограничений (четыре переменных и двенадцать ограничений), $b = \begin{pmatrix} 64270 \\ 4800 \\ 22360 \\ 26240 \\ 7900 \\ 520 \\ 720 \\ 40 \\ 120 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ – свободные члены системы ограничений,

ctype="UUUUUUULULLL" – массив символов, определяющий тип ограничения[20],

vartype="IIII" – массив, определяющий тип переменной, в данном случае все

20 Первые три ограничения типа «меньше», четвертое и пятое – типа «равно».

переменные целые (задача целочисленного программирования),

sense=-1 – задача на максимум.

Программа решения задачи в Octave представлена в листинге 10.7.

```
c=[315; 278 ;573 ;370];
a=[550 0 620 0; 40 30 20 20; 86 110 150 52; 160 92 158 128; 0  158
30 50; 3 4 3 3; 4.5 4.5 4.5 4.5; 0 1 0 0; 0 0 1 0; 1 0 0 0; 0 0 1
0; 0 0 0 1];
b=[64270; 480 0; 22360; 26240; 7900; 520; 720; 40; 120; 0; 0; 0];
    ctype="UUUUUUULULLL"; vartype="IIII"; sense=-1;
    [xmax, fmax, status]=glpk(c,a,b,[], [], ctype, vartype, sense)
    >>>%Результаты решения:
    >>> [xmax, fmax, status]=
                    glpk(c,a,b,[], [], ctype, vartype, sense)
    xmax =
    65
    40
    46
    4
    fmax = 59433
    status = 171
```

Листинг 10.13

Для получения максимальной прибыли (*fmax=492*) необходимо произвести 65 единиц изделий первого типа, 40 – второго, 46 – третьего и 4 – четвёртого. Значение параметра *status=171* говорит о корректности решения задачи линейного программирования.

Кроме Octave, для решения задач линейного программирования авторы использовали электронные таблицы OpenOffice.org Calc, MS Office Excel, математические программы MathCad, Matlab, Maple, Mathematica, Scilab. На наш взгляд, именно Octave обладает самой мощной и гибкой функцией `gplk` для решения задач линейного программирования из всех свободных и проприетарных программ.

Рассмотренных функций (`gplk` и `sqp`) достаточно для решения очень многих оптимизационных задач. Если читателю встретятся оптимизационные задачи, которые невозможно решить с помощью `gplk` и `sqp`, то авторы рекомендуют обратиться к пакету расширений *Minimization* для GNU Octave. Краткое описание функций этого пакета на английском языке приведено на странице http://octave.sourceforge.net/optim/overview.html.

11. Обработка результатов эксперимента. Метод наименьших квадратов

Данная глава посвящена решению часто встречающихся на практике задач по обработке реальных количественных экспериментальных данных, полученных в результате всевозможных научных опытов, технических испытаний методом наименьших квадратов. В первых четырех параграфах читатель познакомится с математическими основами метода наименьших квадратов. Последний пятый параграф посвящён решению задач обработки экспериментальных данных методом наименьших квадратов с использованием пакета Octave.

11.1 Постановка задачи

Метод наименьших квадратов (МНК) позволяет по экспериментальным данным подобрать такую аналитическую функцию, которая проходит настолько близко к экспериментальным точкам, насколько это возможно.

В общем случае задачу можно сформулировать следующим образом.

Пусть в результате эксперимента была получена некая экспериментальная зависимость $y(x)$, представленная в табл. 11.1.

Таблица 11.1. Экспериментальная зависимость

x	x_1	x_2	x_3	…	x_{n-1}	x_n
y	y_1	y_2	y_3	...	y_{n-1}	y_n

Необходимо построить аналитическую зависимость $f(x, a_1, a_2, \dots a_k)$, наиболее точно описывающую результаты эксперимента. Для построения параметров функции $f(x, a_1, a_2, \dots a_k)$ будем использовать *метод наименьших квадратов*. Идея метода наименьших квадратов заключается в том, что функцию $f(x, a_1, a_2, \dots a_k)$ необходимо подобрать таким образом, чтобы сумма квадратов отклонений измеренных значений y_i от расчётных $Y_i = f(x_i, a_1, a_2, \dots, a_k)$ была бы наименьшей (рис. 11.1):

$$S(a_1, a_2, \dots, a_k) = \sum_{i=1}^{n} \left[y_i - Y_i \right]^2 = \sum_{i=1}^{n} \left[y_i - f(x_i, a_1, a_2, \dots, a_k) \right]^2 \to min \qquad (11.1)$$

Задача состоит из двух этапов:

1. По результатам эксперимента определить внешний вид подбираемой зависимости.
2. Подобрать коэффициенты зависимости $Y = f(x, a_1, a_2, \dots, a_k)$.

Математически задача подбора коэффициентов зависимости сводится к определению коэффициентов a_i из условия (11.1). В Octave её можно решать несколькими способами:

1. Решать как задачу поиска минимума функции многих переменных без ограничений с использованием функции `sqp`.
2. Использовать специализированную функцию `polyfit(x,y,n)`.
3. Используя аппарат высшей математики составить и решить систему алгебраических уравнений для определения коэффициентов a_i

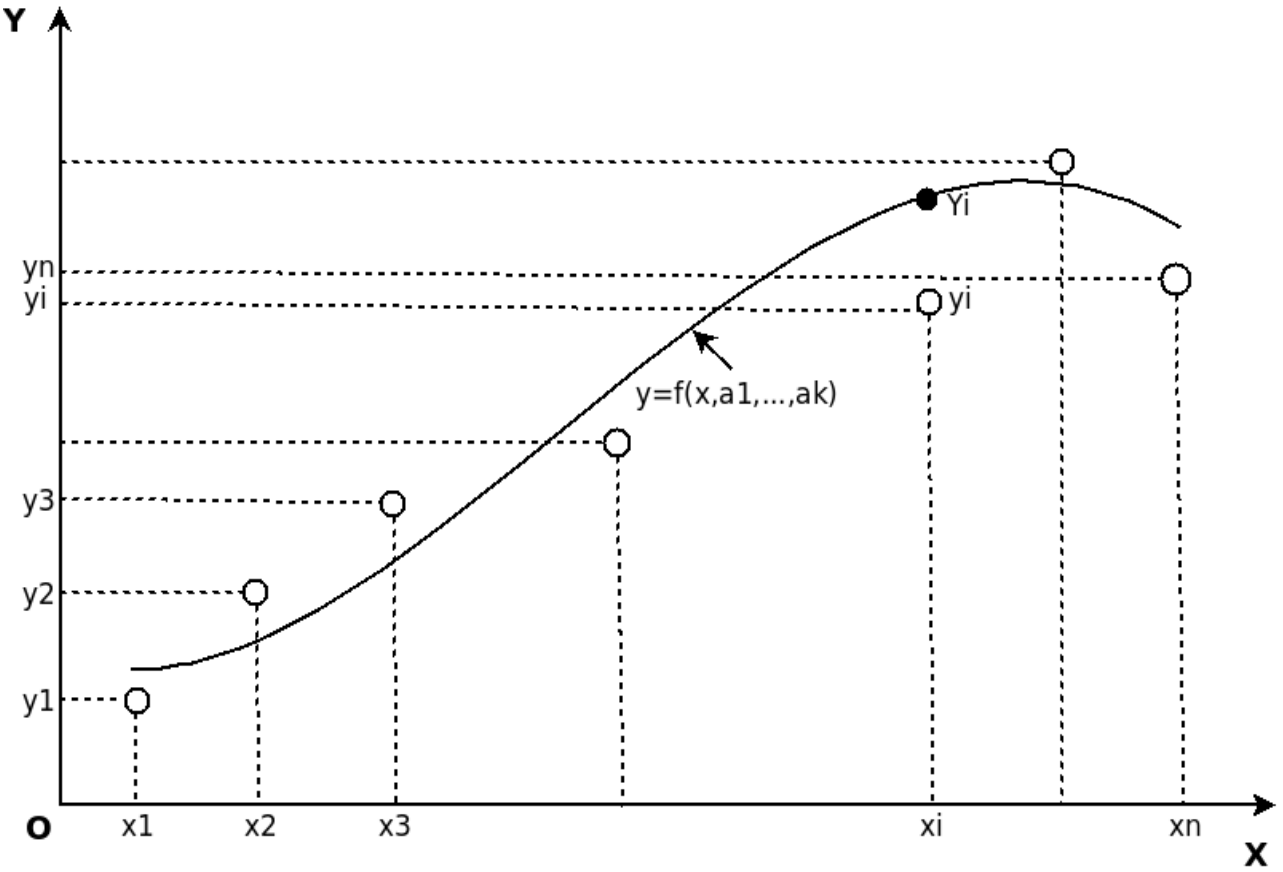


Рис. 11.1 Геометрическая интерпретация метода наименьших квадратов

11.2 Подбор параметров экспериментальной зависимости методом наименьших квадратов

Вспомним некоторые сведения из высшей математики, необходимые для решения задачи подбора зависимости методом наименьших квадратов.

Достаточным условием минимума функции $S(a_1, a_2, \dots, a_k)$ (11.1) является равенство нулю всех её частных производных. Поэтому задача поиска минимума функции (11.1) эквивалентна решению системы алгебраических уравнений:

$$\begin{cases} \dfrac{\partial S}{\partial a_1} = 0, \\ \dfrac{\partial S}{\partial a_2} = 0, \\ \dots \\ \dfrac{\partial S}{\partial a_k} = 0. \end{cases} . \qquad (11.2)$$

Если параметры a_i входят в зависимость $Y = f(x, a_1, a_2, \dots, a_k)$ линейно, то получим систему (11.3) из k линейных уравнений с k неизвестными.

$$\begin{cases} \sum_{i=1}^{n}\left[y_i - f(x_i, a_1, a_2, \dots, a_k)\right]\frac{\partial f}{\partial a_1} = 0, \\ \sum_{i=1}^{n}\left[y_i - f(x_i, a_1, a_2, \dots, a_k)\right]\frac{\partial f}{\partial a_2} = 0, \\ \dots \\ \sum_{i=1}^{n}\left[y_i - f(x_i, a_1, a_2, \dots, a_k)\right]\frac{\partial f}{\partial a_k} = 0. \end{cases} . \qquad (11.3)$$

Составим систему (11.3) для наиболее часто используемых функций.

11.2.1 Подбор коэффициентов линейной зависимости

Для подбора параметров линейной функции $Y = a_1 + a_2 x$ воспользуемся формулой (11.1):

$$S(a_1, a_2) = \sum_{i=1}^{n}\left[y_i - a_1 - a_2 x_i\right]^2 \to min \quad . \qquad (11.4)$$

Продифференцировав функцию S по a_1 и a_2, получим систему уравнений:

$$\begin{cases} 2\sum_{i=1}^{n}\left[y_i - a_1 - a_2 x_i\right](-1) = 0 \\ 2\sum_{i=1}^{n}\left[y_i - a_1 - a_2 x_i\right](-x_i) = 0 \end{cases} \Rightarrow \begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i x_i \end{cases} , \qquad (11.5)$$

решив которую, определим коэффициенты функции $Y = a_1 + a_2 x$:

$$\begin{cases} a_1 = \dfrac{\sum_{i=1}^{n} y_i}{n} - a_2 \dfrac{\sum_{i=1}^{n} x_i}{n} \\ a_2 = \dfrac{n\sum_{i=1}^{n} y_i x_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \end{cases} . \qquad (11.6)$$

11.2.2 Подбор коэффициентов полинома k–й степени

Для определения параметров зависимости $Y = a_1 + a_2 x + a_3 x^2$ составим функцию $S(a_1, a_2, a_3)$ по формуле (11.1):

$$S(a_1, a_2, a_3) = \sum_{i=1}^{n}\left[y_i - a_1 - a_2 x_i - a_3 x_i^2\right]^2 \to min \quad . \qquad (11.7)$$

После дифференцирования S по a_1, a_2 и a_3 получим систему линейных алгебраических уравнений:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i, \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} y_i x_i, \\ a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + a_3 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} y_i x_i^2. \end{cases} \quad . \qquad (11.8)$$

Решив систему (11.8), 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 $S(a_1, a_2, a_3, a_4)$:

$$S(a_1, a_2, a_3, a_4) = \sum_{i=1}^{n} \left[y_i - a_1 - a_2 x_i - a_3 x_i^2 - a_4 x_i^3 \right]^2 \to min \quad . \qquad (11.9)$$

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:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 + a_4 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} y_i, \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 + a_4 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} y_i x_i, \\ a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + a_3 \sum_{i=1}^{n} x_i^4 + a_4 \sum_{i=1}^{n} x_i^5 = \sum_{i=1}^{n} y_i x_i^2, \\ a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 + a_3 \sum_{i=1}^{n} x_i^5 + a_4 \sum_{i=1}^{n} x_i^6 = \sum_{i=1}^{n} y_i x_i^3. \end{cases} \quad . \qquad (11.10)$$

Решив систему (11.10), 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:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 + \ldots + a_{k+1} \sum_{i=1}^{n} x_i^k = \sum_{i=1}^{n} y_i, \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 + \ldots + a_{k+1} \sum_{i=1}^{n} x_i^{k+1} = \sum_{i=1}^{n} y_i x_i, \\ \ldots \\ a_1 \sum_{i=1}^{n} x_i^{k+1} + a_2 \sum_{i=1}^{n} x_i^{k+2} + a_3 \sum_{i=1}^{n} x_i^{k+2} \ldots + a_{k+1} \sum_{i=1}^{n} x_i^{2k} = \sum_{i=1}^{n} y_i x_i^k. \end{cases} \quad . \qquad (11.11)$$

В матричном виде систему (11.11) можно записать

$$Ca = g \quad , \qquad (11.12)$$

Элементы матрицы C и вектора g рассчитываются по формулам

$$C_{i,j} = \sum_{p=1}^{n} x_p^{i+j-2}, i = 1, \ldots, k+1, j = 1, \ldots, k+1 \quad , \qquad (11.13)$$

$$g_i = \sum_{p=1}^{n} y_p x_p^{i-1}, i = 1, \ldots, k+1 \quad . \qquad (11.14)$$

Решив систему (11.12), определим параметры зависимости $Y = a_1 + a_2 x + a_3 x^2 + \ldots + a_{k+1} x^k$.

11.2.3 Подбор коэффициентов функции $Y = ax^b e^{cx}$

Параметры b и c входят в зависимость $Y = ax^b e^{cx}$ нелинейным образом. Чтобы избавиться от нелинейности предварительно прологарифмируем[21] выражение $Y = ax^b e^{cx}$.

$$\ln Y = \ln a + b \ln x + cx$$

Сделаем замену $Y1 = \ln Y$, $A = \ln a$:

$$Y1 = A + b \ln x + cx .$$

Составим функцию *S(A, b, c)* по формуле (11.1):

$$S(A, b, c) = \sum_{i=1}^{n} \left[Y1_i - A - b \ln x_i - c x_i \right]^2 \to min \qquad (11.15)$$

После дифференцирования получим систему трёх линейных алгебраических уравнений для определения коэффициентов A, b и c.

$$\begin{cases} n A + b \sum_{i=1}^{n} \ln x_i + c \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} Y1_i \\ A \sum_{i=1}^{n} \ln x_i + b \sum_{i=1}^{n} (\ln x_i)^2 + b \sum_{i=1}^{n} x_i \ln x_i = \sum_{i=1}^{n} Y1_i \ln x_i \\ A \sum_{i=1}^{n} x_i + b \sum_{i=1}^{n} x_i \ln x_i + c \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} Y1_i x_i \end{cases} . \qquad (11.16)$$

После решения системы (11.6) необходимо вычислить значение коэффициента a по формуле $a = e^A$.

11.2.4 Функции, приводимые к линейной

Для вычисления параметров функции $Y = ax^b$ необходимо предварительно ее 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 (11.6) 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 (11.6) найдем 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 $Y = \frac{x}{ax + b}$ с помощью замены $Z = \frac{1}{Y}, X = \frac{1}{x}$ сводится к линейной $Z = a + bX$.

Для определения коэффициентов функциональной зависимости $Y = \frac{1}{ae^{-x} + b}$

---

21 Можно и не проводить предварительное логарифмирование выражения $Y = ax^b e^{cx}$, однако в этом случаем получаемая система уравнений будет нелинейной, которую решать сложнее.

необходимо сделать следующие замены $Z=\frac{1}{Y}, X=e^{-x}$. В результате также получим линейную функцию $Z=aX+b$.

Аналогичными приемами (логарифмированием, заменами и т. п.) можно многие подбираемые зависимости преобразовать к такому виду, что получаемая при решении задачи оптимизации система (11.2) была системой линейных алгебраических уравнений. При использовании Octave можно напрямую решать задачу подбора параметров, как задачу оптимизации (11.1) с использованием функции `sqp`.

После нахождения параметров зависимости $f(x, a_1, a_2, \dots a_k)$ возникает вопрос насколько адекватно описывает подобранная зависимость экспериментальные данные. Чем ближе величина

$$S=\sum_{i=1}^{n}\left[y_i-f(x_i, a_1, a_2, \dots, a_k)\right]^2 \quad , \tag{11.17}$$

называемая с*уммарной квадратичной ошибкой*, к нулю, тем точнее подобранная кривая описывает экспериментальные данные.

11.3 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 к*оэффициентами регрессии* и определяются формулами (11.6).

Чем меньше величина $S=\sum_{i=1}^{n}\left[y_i-a_1-a_2x_i\right]^2$, тем более обосновано 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:

$$r=\frac{\sum_{i=1}^{n}\left(x_i-M_x\right)\left(y_i-M_y\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i-M_x\right)^2\sum_{i=1}^{n}\left(y_i-M_y\right)^2}}, M_x=\frac{\sum_{i=1}^{n}x_i}{n}, M_y=\frac{\sum_{i=1}^{n}y_i}{n} \tag{11.18}$$

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 Стьдента*. Вычисленное значение критерия определяется по формуле:

$$t=r\sqrt{\frac{n-2}{1-r^2}} \quad . \tag{11.19}$$

Рассчитанное по формуле (11.19) значение t сравнивается со значением, взятым из *таблицы распределения Стьюдента* (табл. 11.2) в соответствии с уровнем значимости p (стандартное значение $p=0.95$) и числом степеней свободы $k=n-2$. Если полученная по формуле (11.19) величина t больше табличного значения, то коэффициент корреляции значимо отличен от нуля.

Таблица 11.2. 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

11.4 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, пользуются *индексом корреляции*. Он показывает тесноту связи между фактором x и зависимой переменной y и рассчитывается по формуле:

$$R=\sqrt{1-\frac{\sum_{i=1}^{n}\left(y_i-Y_i\right)^2}{\sum_{i=1}^{n}\left(y_i-M_y\right)^2}}\ , \qquad (11.20)$$

где y – экспериментальные значения, Y – теоретические значения (рассчитанные по подобранной методом наименьших квадратов формуле), M_y – среднее значение y.

Индекс корреляции лежит в пределах от 0 до 1. При наличии функциональной зависимости индекс корреляции близок к 1. При отсутствии связи индекс 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 связи. При прямолинейной связи коэффициент корреляции по своей абсолютной величине равен индексу корреляции: $|r|=R$.

11.5 Использование Octave для подбора зависимостей методом наименьших квадратов

11.5.1 Функции Octave, используемые для подбора зависимости МНК

Для решения задач подбора аналитических зависимостей по экспериментальным данным можно использовать следующие функции Octave:

`polyfit(x,y,k)` – функция подбора коэффициентов полинома *k*-й степени методом наименьших квадратов (`x` – массив абсцисс экспериментальных точек, `y` – массив ординат экспериментальных точек, `k` – степень полинома), функция возвращает массив коэффициентов полинома;

`sqp(x0,phi,g,h,lb,ub,maxiter,tolerance)` – функция поиска минимума (функция подробно описана в десятой главе);

`cor(x,y)` – функция вычисления коэффициента корреляции (`x` – массив абсцисс экспериментальных точек, `y` – массив ординат экспериментальных точек);

`mean(x)` – функция вычисления среднего арифметического.

11.5.2 Примеры решения задач

ЗАДАЧА 11.1. В «Основах химии» Д.И. Менделеева приводятся данные о растворимости азотнокислого натрия $NaNO_3$ в зависимости от температуры воды. В 100 частях воды (табл. 11.3) растворяется следующее число условных частей $NaNO_3$ при соответствующих температурах. Требуется определить растворимость азотнокислого натрия при температуре $t=32°C$ в случае линейной зависимости и найти коэффициент корреляции.

Таблица 11.3. Таблица растворимости

t	0°	4°	10°	15°	21°	29°	36°	51°	68°
P	66,7	71,0	76,3	80,6	85,7	92,9	99,4	113,6	125,1

Решение задачи 11.1 с комментариями приведено на листинге 11.1.

```
%Ввод экспериментальных данных
X=[0 4 10 15 21 29 36 51 68];
Y=[66.7 71.0 76.3 80.6 85.7 92.9 99.4 113.6 125.1];
%Вычисление вектора коэффициентов полинома y=a1*x+a2
[a]=polyfit(X,Y,1)
%Вычисление значения полинома y=a1*x+a2 в точке t=32
t=32;
yt=a(1)*t+a(2)
%Построение графика полинома y=a1*x+a2,
%экспериментальных точек и значения в заданной точке
%в одной графической области
x=0:68; y=a(1)*x+a(2);
plot(X,Y,'ok',x,y,'-k',t,yt,'xk')
grid
% Вычисление коэффициента корреляции
k =cor(x,y)
```

Листинг 11.1

Результаты работы программы приведены ниже

```
>>>a =   0.87064   67.50779
>>>yt =  95.368
>>>k =  1
```

На рис. 11.2 приведено графическое решение этой задачи, изображены экспериментальные точки и линия регрессии $y=a_1 x+ a_2$, на которой отмечена точка t=32.

ЗАДАЧА 11.2. В результате эксперимента получена табличная зависимость y(x) (табл. 11.4). Подобрать аналитическую зависимость $Y=ax^b e^{cx}$ методом наименьших квадратов. Вычислить ожидаемое значение в точках 2, 3, 4. Вычислить индекс корреляции.

Таблица 11.4. Результаты эксперимента

x	-2	-1,3	-0,6	0,1	0,8	1,5	2,2	2,9	3,6	4,3	5	5,7	6,4
y	-10	-5	0	0,7	0,8	2	3	5	8	30	60	100	238

Решение задачи подбора параметров функции $f(x)=ax^b e^{cx}$ в Octave возможно двумя способами:

1. Найти минимум функции (11.15)[22] и пересчитать значение коэффициента a по формуле $a=e^A$.
2. Сформировать систему линейных алгебраических уравнений (11.16)[23] и найти её решение.

Рассмотрим последовательно оба варианта решения задачи.

---

22 Команда `sqp` (п.10.4) может не справиться с задачей оптимизации функции $S(a,b,c)=\sum_{i=1}^{n}\left[y_i-ax_i^b e^{cx_i}\right]^2$. Дело в том, что при решении задачи оптимизации итерационными методами может возникнуть проблема возведения отрицательного числа в дробную степень (см. главу 2). Да и с точки зрения математики, если есть возможность решать линейную задачу вместо нелинейной, то лучше сделать это.

23 Следует помнить, что при отрицательных значениях y необходимо будет решать проблему замены $Y=\ln y$.

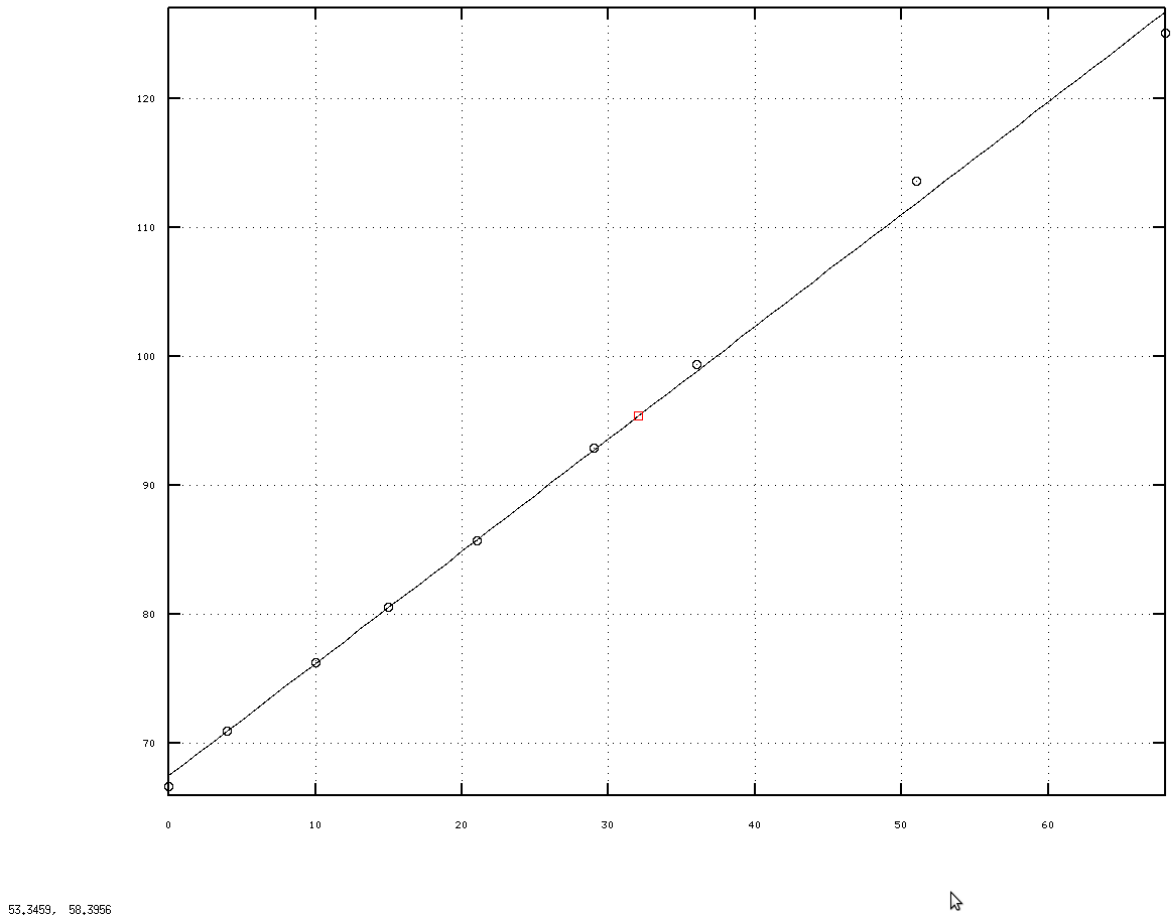


Рис. 11.2 Графическое решение задачи 11.1

Способ 1.

Функция (11.15) реализована в Octave с помощью функции `f_mnk`. Полный текст программы решения задачи первым способом с комментариями приведён в листинге 11.2. Вместо коэффициентов A , b , c из формул (11.9)-(11.7) в программе на Octave используется массив `c`.

```
function s=f_mnk(c)
%Переменные x,y являются глобальными,
% используются в нескольких функциях
global x;
global y;
s=0;
for i=1:length(x)
s=s+(log(y(i))-c(1)-c(2)*log(x(i)) -c(3)*x(i))^2;
end
end
%-------------------------------------------------
global x;
global y;
%Задание начального значения вектора c, при неправильном его
% определении, экстремум может быть найден неправильно.
c=[2;1;3];
```

```
    %Определение координат экспериментальных точек
    x=[1 1.4 1.8 2.2 2.6 3 3.4 3.8 4.2 4.6 5 5.4 5.8];
y=[0.7 0.75 0.67 0.62 0.51 0.45 0.4 0.32 0.28 0.25 0.22 0.16 0.1];
    Решение задачи оптимизации функции 11.9 с помощью sqp.
    c=sqp(c,@f_mnk)
    % Вычисление суммарной квадратичной ошибки для подобранной
    %зависимости и вывод её на экран.
    sum1=f_mnk(c)
    %Формирование точек для построения графика подобранной кривой.
    x1=1:0.1:6;
    y1=exp(c(1)).*x1.^c(2).*exp(c(3).*x1);
    %Вычисление значений на подобранной кривой в заданных точках.
    yr=exp(c(1)).*x.^c(2).*exp(c(3).*x);
    %Вычисление ожидаемого значения подобранной функции в точках
    %x=[2,3,4];
    x2=[2 3 4]
    y2=exp(c(1)).*x2.^c(2).*exp(c(3).*x2)
    %Построение графика: подобранная кривая, f(x2)
    % и экспериментальные точки.
    plot(x1,y1,'-r',x,y,'*b',x2,y2,'pk');
    %Вычисление индекса корреляции.
    R=sqrt(1-sum((y-yr).^2)/sum((y-mean(y)).^2))
```

Листинг 11.2

Результаты программы приведены ниже.

```
    >>>c =
       0.33503
       0.90183
      -0.69337
    >>>sum1=0.090533
    >>>x2=
       2   3   4
    >>>y2=
       0.65272   0.47033   0.30475
    >>>R=0.99533
```

Таким образом, подобрана зависимость $Y=0.33503\mathrm{x}^{0.90183}e^{-0.9337\mathrm{x}}$. Вычислено ожидаемое значение в точках 2, 3, 4: $Y(2)=0.65272, Y(3)=0.47033, Y(4)=0.30475$. График подобранной зависимости вместе с экспериментальными точками и расчётными значениями изображён на рис. 11.3. Индекс корреляции равен 0.99533.

Способ 2.

Теперь рассмотрим решение задачи 11.2 путём решения системы 11.7. Решение с комментариями приведено на листинге 11.3. Результаты и графики при решении обоими способами полностью совпадают.

```
    function s=f_mnk(c)
    %Переменные x,y являются глобальными,
    % используются в нескольких функциях
    global x;
    global y;
    s=0;
    for i=1:length(x)
    s=s+(log(y(i))-c(1)-c(2)*log(x(i)) -c(3)*x(i))^2;
    end
```

```
end
global x;
global y;
```

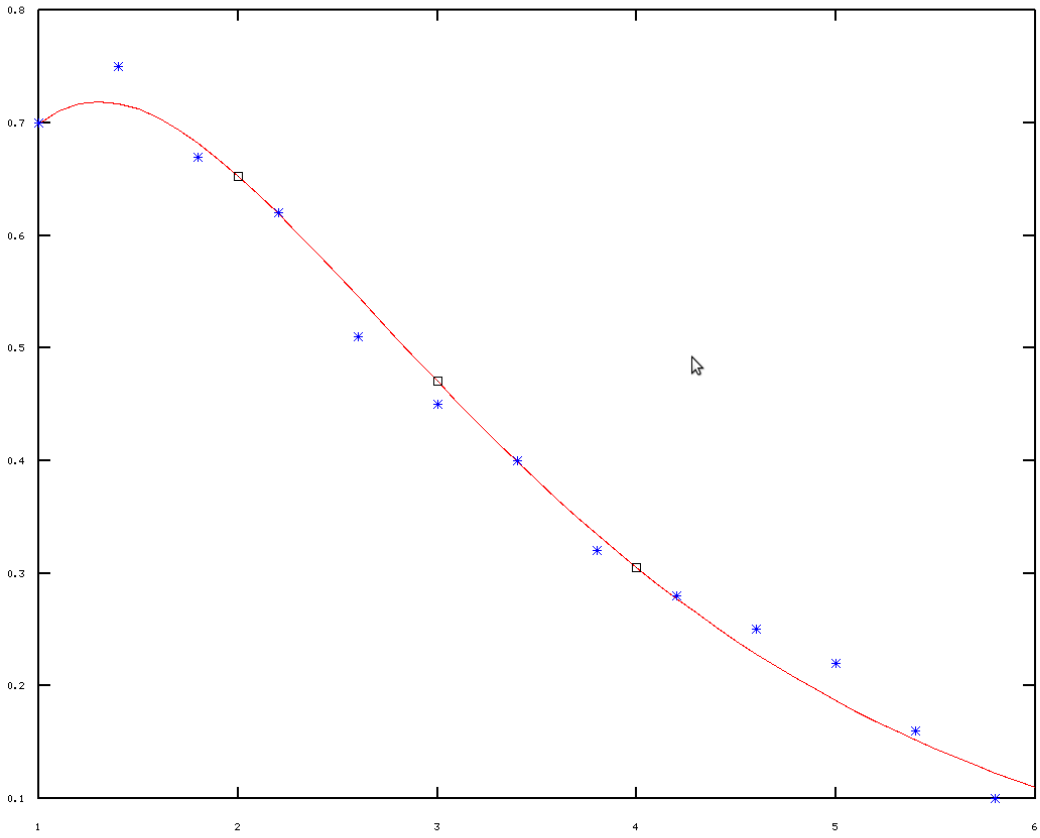


Рис. 11.3 График к задаче 11.2. Экспериментальные точки и, подобранная методом наименьших квадратов, зависимость

```
%Определение координат экспериментальных точек
x=[1 1.4 1.8 2.2 2.6 3 3.4 3.8 4.2 4.6 5 5.4 5.8];
y=[0.7 0.75 0.67 0.62 0.51 0.45 0.4 0.32 0.28 0.25 0.22 0.16 0.1];
%Формирование СЛАУ (11.7)
G=[length(x) sum(log(x)) sum(x);...
sum(log(x)) sum(log(x).*log(x)) sum(x.*log(x));...
sum(x) sum(x.*log(x)) sum(x.*x)];
H=[sum(log(y)); sum(log(y).*log(x)); sum(log(y).*x)];
%Решение СЛАУ методом Гаусса с помощью функции rref.
C=rref([G H]);
n=size(C);
c=C(:,n(2))
% Вычисление суммарной квадратичной ошибки для подобранной
%зависимости и вывод её на экран.
sum1=f_mnk(c)
%Формирование точек для построения графика подобранной кривой.
x1=1:0.1:6;
y1=exp(c(1)).*x1.^c(2).*exp(c(3).*x1);
%Вычисление значений на подобранной кривой в заданных точках.
```

```
yr=exp(c(1)).*x.^c(2).*exp(c(3).*x);
%Вычисление ожидаемого значения подобранной функции в точках
%x=[2,3,4];
x2=[2 3 4]
y2=exp(c(1)).*x2.^c(2).*exp(c(3).*x2)
%Построение графика: подобранная кривая, f(x2)
% и экспериментальные точки.
plot(x1,y1,'-r',x,y,'*b',x2,y2,'pk');
%Вычисление индекса корреляции.
R=sqrt(1-sum((y-yr).^2)/sum((y-mean(y)).^2))
```

Листинг 11.3

ЗАДАЧА 11.3. В результате эксперимента получена табличная зависимость y(x) (табл. 11.2). Подобрать аналитические зависимости вида $f(x)=b_1+b_2x+b_3x^2+b_4x^3+b_5x^4+b_6x^5$, $g(x)=a_1+a_2x+a_3x^2+a_4x^3+a_5x^5$ и $\varphi(x)=c_1+c_2x+c_4x^3+c_5x^5$ методом наименьших квадратов. Пользуясь значением индекса корреляции выбрать наилучшую зависимость, с помощью которой вычислить ожидаемое значение в точках 1, 2.5, 4.8. Построить графики экспериментальных точек подобранных зависимостей. На графиках отобразить рассчитанные значения в точках 1, 2.5, 4.8.

Таблица 11.5:

x	-2	-1,3	-0,6	0,1	0,8	1,5	2,2	2,9	3,6	4,3	5	5,7	6,4
y	-10	-5	0	0,7	0,8	2	3	5	8	30	60	100	238

Как рассматривалось ранее решать задачу подбора параметров полинома методом наименьших квадратов, в Octave можно тремя способами.

1. Сформировать и решить систему уравнений (11.3).
2. Решить задачу оптимизации (11.1). В случае полинома $f(x)=\sum_{i=1}^{k+1} a_i x^{i-1}$ подбираемые коэффициенты a_i будут входить в функцию (11.1) линейным образом и не должно возникнуть проблем при решении задачи оптимизации с помощью функции `sqp`.
3. Использовать функцию `polyfit`.

Чтобы продемонстрировать использование всех трех методов для подбора $f(x)=\sum_{i=1}^{6} b_i x^{i-1}$ воспользуемся функцией `polyfit`, для формирования коэффициентов функции $g(x)$ сформируем и решим систему уравнений (11.3), а функцию $\varphi(x)$ будем искать с помощью функции `sqp`.

Для формирования подбора коэффициентов функции $g(x)=a_1+a_2x+a_3x^2+a_4x^3+a_5x^5$ сформируем систему уравнений. Составим функцию $S(a_1,a_2,a_3,a_4,a_5)=\sum_{i=1}^{n}\left[y_i-a_1-a_2x_i-a_3x_i^2-a_4x_i^3-a_5x_i^5\right]^2$.

После дифференцирования S по a_1, a_2, a_3, a_4 и a_5 система линейных алгебраических уравнений для вычисления параметров a_1, a_2, a_3, a_4, a_5 примет вид:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i + a_3 \sum_{i=1}^{n} x_i^2 + a_4 \sum_{i=1}^{n} x_i^3 + a_5 \sum_{i=1}^{n} x_i^5 = \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + a_3 \sum_{i=1}^{n} x_i^3 + a_4 \sum_{i=1}^{n} x_i^4 + a_5 \sum_{i=1}^{n} x_i^6 = \sum_{i=1}^{n} y_i x_i \\ a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + a_3 \sum_{i=1}^{n} x_i^4 + a_4 \sum_{i=1}^{n} x_i^5 + a_5 \sum_{i=1}^{n} x_i^7 = \sum_{i=1}^{n} y_i x_i^2 \\ a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 + a_3 \sum_{i=1}^{n} x_i^5 + a_4 \sum_{i=1}^{n} x_i^6 + a_5 \sum_{i=1}^{n} x_i^8 = \sum_{i=1}^{n} y_i x_i^3 \\ a_1 \sum_{i=1}^{n} x_i^5 + a_2 \sum_{i=1}^{n} x_i^6 + a_3 \sum_{i=1}^{n} x_i^7 + a_4 \sum_{i=1}^{n} x_i^8 + a_5 \sum_{i=1}^{n} x_i^{10} = \sum_{i=1}^{n} y_i x_i^5 \end{cases} . \quad (11.21)$$

Решив систему (11.5), найдём коэффициенты a_1 , a_2 , a_3 , a_4 и a_5 функции $g(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + a_5 x^5$.

Для поиска функциональной зависимости вида $\varphi(x) = c_1 + c_2 x + c_4 x^3 + c_5 x^5$ необходимо будет найти такие значения c_1, c_2, c_3, c_4 , при которых функция

$$S(c_1, c_2, c_3, c_4) = \sum_{i=1}^{n} \left[y_i - c_1 - c_2 x_i - c_3 x_i^3 - c_4 x_i^5 \right]^2 \quad (11.22)$$

принимала наименьшее значение.

После вывода необходимых формул приступим к реализации в Octave. Текст программы в Octave с очень подробными комментариями приведён на листинге 11.4.

```
% Функция для подбора зависимости fi(x) методом наименьших
% квадратов.
function s=f_mnk(c)
%Переменные x,y являются глобальными, используются в
% функции f_mnk и главной функции.
global x; global y;
%Формирование суммы квадратов отклонений.
s=0;
for i=1:length(x)
s=s+(y(i)-c(1)-c(2)*x(i) -c(3)*x(i)^3 - c(4)*x(i)^5 )^2;
end
end
%-Главная функция-------------------------------------
%Переменные x,y являются глобальными, используются в
% функции f_mnk и главной функции.
global x;
global y;
%Определение координат экспериментальных точек
x=[-2 -1.3 -0.6 0.1 0.8 1.5 2.2 2.9 3.6 4.3 5 5.7 6.4];
y=[-10 -5 0 0.7 0.8 2 3 5 8 30 60 100 238];
z=[1 2.5 4.8]
%Подбор коэффициентов зависимости f(x) (полинома пятой
%степени) методом наименьших квадратов, используя функцию
%polyfit. Коэффициенты полинома будут хранится в переменной B.
B=polyfit(x,y,5)
%Формирование точек для построения графиков подобранных
% функций.
```

```
X1=-2:0.1:6.5;
%Вычисление ординат точек графика первой функции f(x).
Y1=polyval(B,X1);
%Формирование системы (11.5) для подбора функции g(x).
%Здесь GGL — матрица коэффициентов, H – вектор правых частей
% системы (11.5). матрица G — первые 4 строки и 4 столбца
%матрицы коэффициентов, G1 – пятый столбец матрицы
%коэффициентов, G2 — пятая строка матрицы коэффициентов.
for i = 1:4
 for j=1:4
      G(i,j)=sum(x.^(i+j-2));
 endfor
endfor
for i = 1:4
      G1(i)=sum(x.^(i+5));
      H(i)=sum(y.*x.^(i-1));
endfor
for i=1:4
      G2(i)=sum(x.^(i+4));
endfor
G2(5)=sum(x.^10);
%Формирование матрицы коэффициентов системы (11.5) из матриц
% G, G1 и G2.
GGL=[G G1'; G2]
H(5)=sum(y.*x.^5);
%Решение системы (11.5) методом обратной матрицы и
% формирование коэффициентов A функции g(x).
A=inv(GGL)*H'
%Подбор коэффициентов зависимости fi(x) методом наименьших
%квадратов, используя функцию %sqp. Коэффициенты функции будут
%хранится в переменной C.
%Задание начального значения вектора C, при неправильном его
%определении, экстремум функции может быть найден неправильно.
C=[2;1;3;1];
%Поиск вектора C, при котором функция (11.4) достигает своего
%минимального значения с помощью функции sqp,
%вектор C — коэффициенты функции fi.
C=sqp(C,@f_mnk)
%Вычисление ординат точек графика второй функции g(x).
Y2=A(1)+A(2)*X1+A(3)*X1.^2+A(4)*X1.^3+A(5)*X1.^5;
%Вычисление ординат точек графика третьей функции fi(x).
Y3=C(1)+C(2)*X1+C(3)*X1.^3 + C(4)*X1.^5;
%Вычисление значений на подобранной на первой функции f(x)
%в заданных точках.
yr1=polyval(B,x);
%Вычисление значений на подобранной на второй функции g(x)
%в заданных точках.
yr2=A(1)+A(2)*x+A(3)*x.^2+A(4)*x.^3+A(5)*x.^5;
%Вычисление значений на подобранной на второй функции fi(x)
%в заданных точках.
yr3=C(1)+C(2)*x+C(3)*x.^3 + C(4)*x.^5;
```

```
    %Вычисление индекса корреляции для первой функции f(x).
    R1=sqrt(1-sum((y-yr1).^2)/sum((y-mean(y)).^2))
    %Вычисление индекса корреляции для второй функции g(x).
    R2=sqrt(1-sum((y-yr2).^2)/sum((y-mean(y)).^2))
    %Вычисление индекса корреляции для третьей функции fi(x).
    R3=sqrt(1-sum((y-yr3).^2)/sum((y-mean(y)).^2))
    %Сравнивая значения трёх индексов корреляции, выбираем
    %наилучшую функцию и с её помощью вычисляем ожидаемое значение
    %в точнах  1, 2.5, 4.8.
    if R1>R2 & R1>R3
          yz=polyval(B,z)
          "R1="
          R1
    endif
    if R2>R1 & R2>R3
          yz=C2(1)+C2(2)*z+C2(3)*z.^2+C2(4)*z.^3+C2(5)*z.^5
          "R2="
          R2
    endif
    if R3>R1 & R3>R2
          yz=C(1)+C(2)*z+C(3)*z.^3 + C(4)*z.^5
          "R3="
          R3
    endif
    %Построение графика.
plot(x,y,"*r;esperiment;",X1,Y1,'-b;f(x);',...
X1,Y2,'dr;g(x);',X1,Y3,'ok;fi(x);',z,yz,'sb;f(z);'); grid();
```

Листинг 11.4

Ниже приведены результаты работы программы.

```
    >>>z =
     1.0000         2.5000         4.8000
    >>>B =
    0.083039   -0.567892 0.906779  1.609432  -1.115925 -1.355075
    GGL =
    1.3000e+01 2.8600e+01     1.5210e+02     7.2701e+02
     1.2793e+05
    2.8600e+01 1.5210e+02     7.2701e+02     3.9868e+03
     7.5030e+05
    1.5210e+02 7.2701e+02     3.9868e+03     2.2183e+04
     4.4706e+06
    7.2701e+02 3.9868e+03     2.2183e+04     1.2793e+05
     2.6938e+07
    2.2183e+04 1.2793e+05     7.5030e+05     4.4706e+06
     1.6383e+08
    >>>A =
    9.4262e+00
    -3.6516e+00
    -5.7767e+00
    1.7888e+00
    -5.8179e-05
    >>>C =
```

```
-1.030345
5.080391
-0.609721
0.033534
>>>R1=0.99690
>>>R2 =0.98136
>>>R3 =0.99573
>>>yz =
-0.43964   6.00854   40.77972
ans=R1=
R1=0.99690
```

На рисунке 11.4 представлено графическое решение задачи.

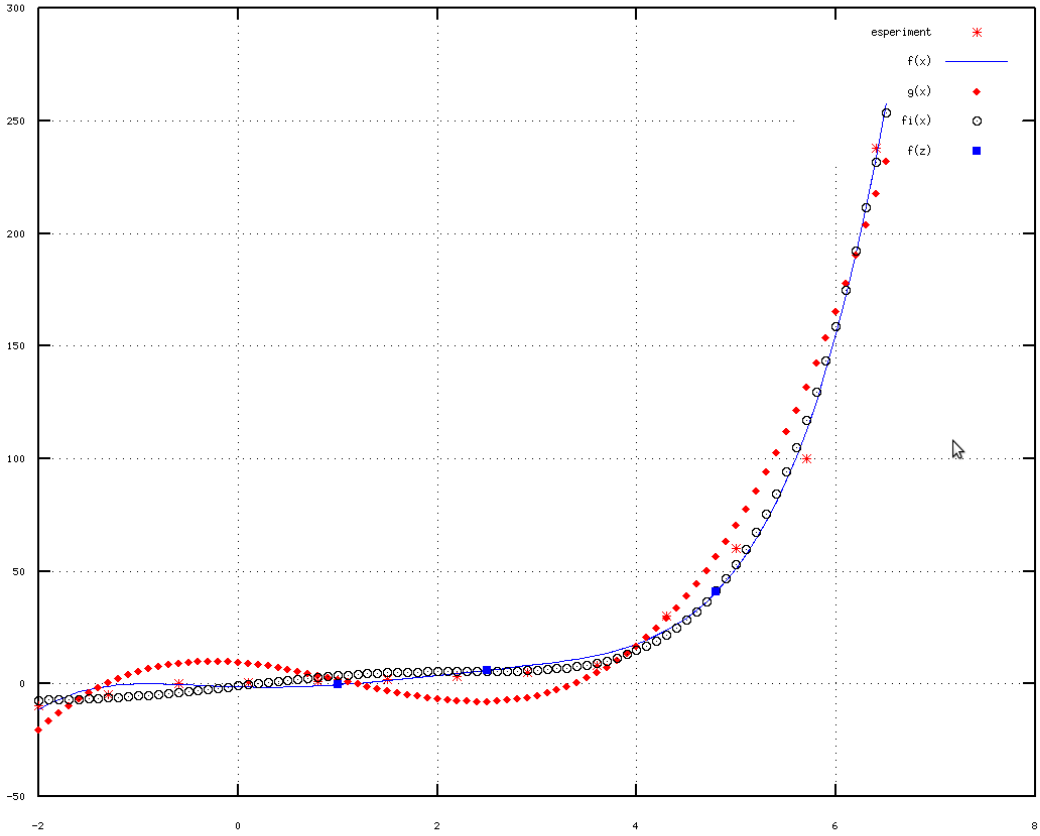


Рис. 11.4 Графическое решение задачи 11.3

Рассмотренная задача демонстрирует основные приёмы подбора зависимости методом наименьших квадратов в Octave. Авторы рекомендует внимательно рассмотреть её для понимания методов решения подобных задач в Octave.

В заключении авторы позволят несколько советов по решению задачи аппроксимации.

1. Подбор каждой зависимости по экспериментальным данным – довольно сложная математическая задача, поэтому следует аккуратно выбирать вид зависимости, наиболее точно описывающей экспериментальные точки.
2. Необходимо сформировать реальную систему уравнений исходя из соотношений

(11.1) – (11.3). Следует помнить, что проще и точнее решать систему линейных алгебраических уравнений, чем систему нелинейных уравнений. Поэтому, может быть, следует преобразовать исходную функцию (прологарифмировать, сделать замену и т. д.) и только после этого составлять систему уравнений.

3. При том, что функция `sqp` – довольно мощная, лучше использовать методы и функции решения систем линейных алгебраических уравнений, функцию `polyfit`, чем функцию `sqp`. Этот совет связан с тем, что в функции `sqp` используются приближённые итерационные алгоритмы, поэтому получаемый результат иногда может быть менее точен, чем при точных методах решения систем линейных алгебраических уравнений. Но, иногда, именно функция `sqp` – единственный метод решения задачи.
4. Для оценки корректности подобранной зависимости следует использовать коэффициент корреляции, критерий Стьюдента (для линейной зависимости) и индекс корреляции и суммарную квадратичную ошибку (для нелинейных зависимостей).

12. Обработка результатов эксперимента. Интерполяция функций

Данная глава посвящена решению часто встречающихся на практике задач по обработке реальных количественных экспериментальных данных, полученных в результате всевозможных научных опытов, технических испытаний методом интерполяции. Описаны численные методы интерполирования, а также рассмотрено решение задач интерполирования в Octave.

12.1 Постановка задачи

Напомним читателю задачу интерполирования. На отрезке [a,b] заданы n+1 точки x_0 , x_1 , x_2 , …, x_n $(a=x_0, b=x_n)$, называемые узлами интерполяции, и значения неизвестной функции $f(x)$ в этих точках

$$f(x_0)=y_0, f(x_1)=y_1, f(x_2)=y_2, \ldots, f(x_n)=y_n \quad . \tag{12.1}$$

Требуется построить интерполирующую функцию $F(x)$, которая в узлах интерполяции принимает те же значения, что и f(x):

$$F(x_0)=y_0, F(x_1)=y_1, F(x_2)=y_2, \ldots, F(x_n)=y_n \quad . \tag{12.2}$$

В общей постановке задача может не иметь однозначного решений или совсем не иметь решений. Задача становится однозначной, если функцию $F(x)$ будет искать в виде полинома $F(x)=P_n(x)$ степени n, удовлетворяющий условиям (12.2).

Полученную интерполяционную формулу $y=F(x)$ зачастую используют для приближенного нахождения 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 $x \in [x0, xn]$, и *экстраполирование*, когда x $x \notin [x0, xn]$.

Рассмотрим некоторые наиболее часто используемые интерполяционные полиномы.

12.1.1 Канонический полином

Будем искать интерполирующую функцию $F(x)$ в виде канонического полинома степени *n*.

$$F(x)=P_n(x)=a_0+ a_1 x+ a_2 x^2+ \ldots a_n x^n \quad . \tag{12.3}$$

Выбор многочлена степени *n* основан на том факте, что через *n+1* точку проходит единственная кривая степени *n*. Подставив (12.3) в (12.1), получим систему линейных алгебраических уравнений (12.4).

$$\begin{cases} a_0+a_1x_0+a_2x_0^2+\dots a_nx_0^n=y_0, \\ a_0+a_1x_1+a_2x_1^2+\dots a_nx_1^n=y_1, \\ a_0+a_1x_2+a_2x_2^2+\dots a_nx_2^n=y_2, \\ \dots \\ a_0+a_1x_n+a_2x_n^2+\dots a_nx_n^n=y_n. \end{cases} \qquad (12.4)$$

Решая эту систему линейных алгебраических уравнений, найдём коэффициенты интерполяционного полинома a_0 , a_1 , ..., a_n .

12.1.2 Полином Ньютона

Ньютон предложил интерполирующую функцию записать в виде следующего полинома n-й степени:

$$F(t)=A_0+A_1(t-x_0)+A_2(t-x_0)(t-x_1)+\dots+A_n(t-x_0)(t-x_1)\dots(t-x_{n-1}) \quad . \qquad (12.5)$$

Подставим $F(x_0)=y_0$ в (12.5) и вычислим значение коэффициента A_0

$A_0=y_0$.

Подставим $F(x_1)=y_1$ в (12.5), после чего получим соотношение для вычисления A_1 :

$F(x_1)=A_0+A_1(x_1-x_0)=y_1$.

Отсюда коэффициент A_1 рассчитывается по формуле:

$$A_1=\frac{y_0-y_1}{x_0-x_1}=y_{01} \quad ,$$

где y_{01} – разделённая разность первого порядка, которая стремится к первой производной функции при $x_1 \to x_0$. По аналогии вводятся и другие разделённые разности первого порядка:

$$y_{02}=\frac{y_0-y_2}{x_0-x_2}, y_{03}=\frac{y_0-y_3}{x_0-x_3}, \dots, y_{0n}=\frac{y_0-y_n}{x_0-x_n} \quad .$$

Подставим соотношение $F(x_2)=y_2$ в (12.5), в результате чего получим

$A_0+A_1(x_2-x_0)+A_2(x_2-x_0)(x_2-x_1)=y_2$,

$y_0+y_{01}(x_2-x_0)+A_2(x_2-x_0)(x_2-x_1)=y_2$.

Отсюда A_2 вычисляется по формуле

$$A_2=y_{012}=\frac{y_{01}-y_{02}}{x_1-x_2} \quad ,$$

здесь y_{012} – разделённая разность второго порядка, эта величина стремится ко второй производной при $x_1 \to x_2$. Аналогично вводятся

$$y_{013}=\frac{y_{01}-y_{03}}{x_0-x_3}, y_{014}=\frac{y_{01}-y_{04}}{x_1-x_4}, \dots, y_{01n}=\frac{y_{01}-y_{0n}}{x_1-x_n} \quad .$$

Подставим $F(x_3)=y_3$ в (12.5), после чего получим $A_3=y_{0123}=\frac{y_{012}-y_{013}}{x_2-x_3}$.

Аналогично можно ввести коэффициенты $y_{0124}=\frac{y_{012}-y_{014}}{x_2-x_4}, \dots, y_{012n}=\frac{y_{012}-y_{01n}}{x_2-x_n}$.

Этот процесс будем продолжать до тех пор, пока не вычислим

$$A_n = y_{012\ldots n} = \frac{y_{012\ldots n-1} - y_{01\ldots n-2n}}{x_{n-1} - x_n} \quad .$$

Полученные результаты запишем в табл. 12.1.

Таблица 12.1. Таблица разделенных разностей полинома Ньютона

x	f(x)	1	2	3	4	…	n
x_0	y_0						
x_1	y_1	y_{01}					
x_2	y_2	y_{02}	y_{012}				
x_3	y_3	y_{03}	y_{013}	y_{0123}			
x_4	y_4	y_{04}	y_{014}	y_{0124}	y_{01234}		
…	…	…	…	…	…	…	…
x_n	y_n	y_{0n}	y_{01n}	y_{012n}	y_{0123n}	…	$y_{012\ldots n}$

В вычислении по формуле (12.5) будут участвовать только диагональные элементы таблицы (т.е. коэффициенты A_i), а все остальные элементы таблицы являются промежуточными и нужны для вычисления диагональных элементов.

12.1.3 Полином Лагранжа

Еще одно представление интерполяционного полинома степени n предложил Лагранж:

$$F(t) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - x_j}{x_i - x_j} \quad . \qquad (12.6)$$

Напомним читателю, что рассмотренные три способа построения полинома – это три различных 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 ожидаемое значение в точках по формуле (12.5). При использовании канонического полинома приходится решать систему линейных алгебраических уравнений (12.4), поэтому он используется значительно реже.

12.1.4 Реализация интерполяционного полинома n-й степени в Octave

Построить интерполяционный полином n-й степени в Octave можно одним из следующих способов.

1. Средствами языка программирования реализовать один из рассмотренных алгоритмов построения полинома: канонический (см. листинг 12.1), полином Ньютона (см. листинг 12.2), полином Лагранжа (см. листинг 12.3), после чего посчитать значения в нужных точках.
2. Воспользоваться функцией `polyfit(x,y,k)` (в этом случае `k=length(x)-1`) для вычисления коэффициентов полинома, после чего с помощью функции `polyval(A,t)` вычислить значение полинома в необходимых точках.

Листинг 12.1 содержит пример применения функции `kanon` для вычисления канонического полинома в точке *t* по экспериментальным значениям (x_i, y_i)

```
%x — массив абсцисс экспериментальных точек, y — массив
```

```
% ординат экспериментальных точек, t — точка в которой
%требуется найти значение.
function s=kanon(x,y,t)
% Вычисление количества точек в массивах x и y
n=length(x);
%Формирование коэффициентов системы уравнений (12.4)
for i=1:n
for j=1:n
A(i,j)=x(i).^(j-1);
end
end
%Решение системы уравнений (12.4)
a=A^(-1)*y';
%Вычисление значения полинома в точке t по формуле (12.3)
s=0;
for i=1:n
s=s+a(i)*t^(i-1);
end
end
```

Листинг 12.1.

Применение функции `newton` для вычисления полинома Ньютона в точке t по экспериментальным значениям (x_i, y_i) представлено в листинге 12.2.

```
%x — массив абсцисс экспериментальных точек, y — массив
% ординат экспериментальных точек, t — точка в которой
%требуется найти значение.
function s=newton(x,y,t)
% Вычисление количества точек в массивах x и y
n=length(x);
% Запись в первый столбец матрицы разделенных разностей
% вектора y
for i=1:n
C(i,1)=y(i);
end
% Формирование матрицы разделенных разностей
for i=2:n
for j=2:n
if (i<j)
C(i,j)=0;
else
C(i,j)=(C(i,j-1)-C(j-1,j-1))/(x(i)-x(j-1));
end
end
end
% Формирование массива коэффициентов полинома Ньютона
for i=1:n
A(i)=C(i,i);
end
% Расчет значения полинома в точке t по формуле (12.5)
s=0;
for i=1:n
p=1;
```

```
for j=1:i-1
p=p*(t-x(j));
end
s=s+A(i)*p;
end
end
```

Листинг 12.2.

Пример использования функции `lagrang` для вычисления полинома Лагранжа в точке t по экспериментальным значениям (x_i, y_i) представлен в листинге 12.3.

```
%x — массив абсцисс экспериментальных точек, y — массив
% ординат экспериментальных точек, t — точка в которой
%требуется найти значение.
function s=lagrang(x,y,t)
% Вычисление количества точек в массивах x и y
n=length(x);
s=0;
% Расчет суммы произведений по формуле (12.6)
% для вычисления значения полинома Лагранжа в точке t
for i=1:n
p=1;
for j=1:n
if (j~=i)
p=p*(t-x(j))/(x(i)-x(j));
end
end
s=s+y(i)*p;
end
end
```

Листинг 12.3.

ЗАДАЧА 12.1. В результате эксперимента получена табличная зависимость $y(x)$ (табл. 12.2). Построить интерполяционный полином. Вычислить ожидаемое значение в точках 0.5, 0.6 и 0.7, построить график зависимости.

Таблица 12.2. Результаты эксперимента

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

Решение с подробными комментариями представлено на листинге 12.4.

```
function s=kanon(x,y,t)
% Вычисление количества точек в массивах x и y
n=length(x);
%Формирование коэффициентов системы уравнений (12.4)
for i=1:n
for j=1:n
A(i,j)=x(i).^(j-1);
end
end
%Решение системы уравнений (12.4)
a=A^(-1)*y';
%Вычисление значения полинома в точке t по формуле (12.3)
s=0;
```

```
for i=1:n
s=s+a(i)*t^(i-1);
end
end
function s=newton(x,y,t)
% Вычисление количества точек в массивах x и y
n=length(x);
% Запись в первый столбец матрицы разделенных разностей
% вектора y
for i=1:n
C(i,1)=y(i);
end
% Формирование матрицы разделенных разностей
for i=2:n
for j=2:n
if (i<j)
C(i,j)=0;
else
C(i,j)=(C(i,j-1)-C(j-1,j-1))/(x(i)-x(j-1));
end
end
end
% Формирование массива коэффициентов полинома Ньютона
for i=1:n
A(i)=C(i,i);
end
% Расчет значения полинома в точке t по формуле (12.5)
s=0;
for i=1:n
p=1;
for j=1:i-1
p=p*(t-x(j));
end
s=s+A(i)*p;
end
end
function s=lagrang(x,y,t)
% Вычисление количества точек в массивах x и y
n=length(x);
s=0;
% Расчет суммы произведений по формуле (12.6)
% для вычисления значения полинома Лагранжа в точке t
for i=1:n
p=1;
for j=1:n
if (j~=i)
p=p*(t-x(j))/(x(i)-x(j));
end
end
s=s+y(i)*p;
end
```

```
end
%x — массив абсцисс экспериментальных точек, y — массив
%ординат экспериментальных точек задачи 12.1.
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];
r=[0.5 0.6 0.7]
for i=1:3
%Вычисление ожидаемого значения интерполяционного полинома
% Ньютона в точках r=[0.5 0.6 0.7]
rsn(i)=newton(x,y,r(i));
%Вычисление ожидаемого значения канонического
%интерполяционного полинома в точках r=[0.5 0.6 0.7]
rsk(i)=kanon(x,y,r(i));
%Вычисление ожидаемого значения интерполяционного полинома
% Лагранжа в точках r=[0.5 0.6 0.7]
rsl(i)=lagrang(x,y,r(i));
end
rsn
rsk
rsl
%Вычисление ожидаемого значения интерполяционного полинома в
% точках r=[0.5 0.6 0.7] с помощью функции polyfit
A=polyfit(x,y,length(x)-1)
rsp=polyval(A,r)
%Вычисление точек для построения графика интерполяционного
%полинома.
x1=0.43:0.01:0.75;
y1=polyval(A,x1);
%Построение графика.
plot(x,y,'*b;experment;',x1,y1,'-r;interpolation;',...
r,rsp,'pb;f(r);');
grid();
%Результаты:
>>>r =
0.50000 0.60000 0.70000
>>>rsn =
1.7725 1.9874 2.2285
>>>rsk =
1.7725 1.9874 2.2285
>>>rsl =
1.7725 1.9874 2.2285
>>>A =
0.44180 -1.17180 1.70415 -0.18866 1.38721 0.97243
>>>rsp =
1.7725 1.9874 2.2285
```

Листинг 12.4.

Как и следовало ожидать, все четыре используемых метода построения интерполяционного полинома пятой степени дали одни и те же значения.

На рис. 12.1 представлено графическое решение задачи 12.1. Подобным образом можно подбирать коэффициенты интерполяционного полинома и для других задач.

Полиномиальная интерполяция не всегда дает 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 английского слова spline - рейка, линейка).

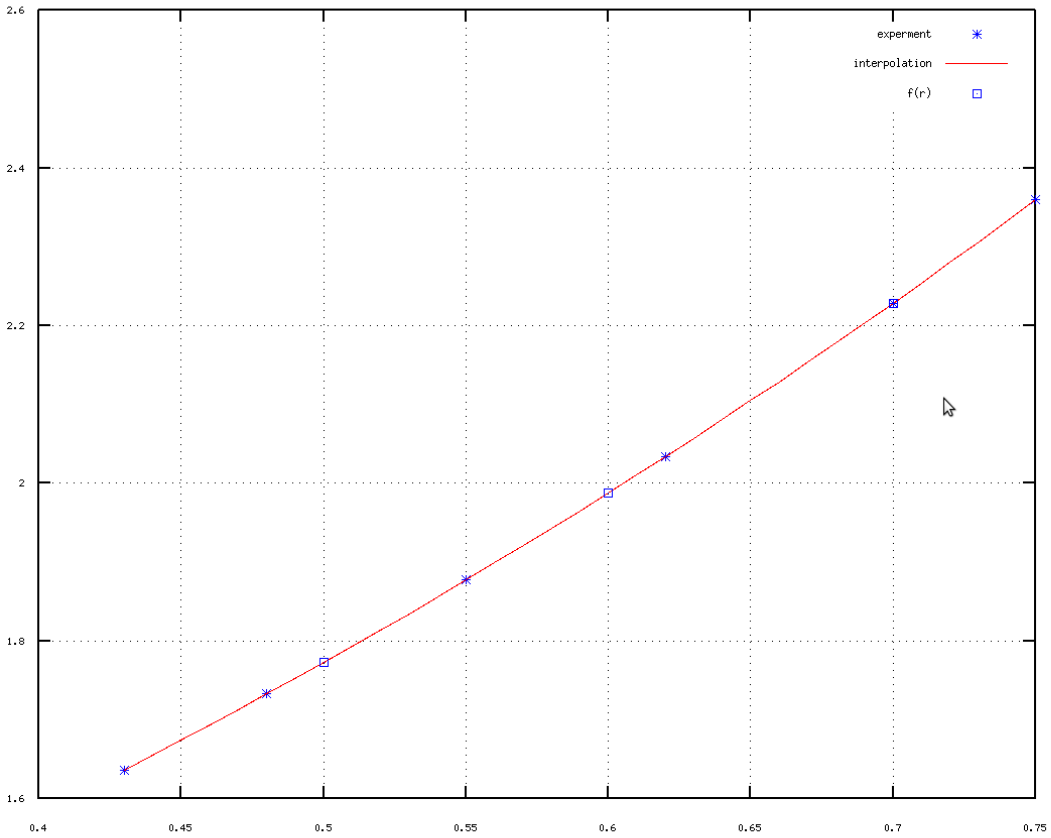


Рис. 12.1 Графическое решение задачи 12.1

12.2 Интерполяция сплайнами

Рассмотрим один из наиболее распространенных вариантов интерполяции кубическими сплайнами. Было установлено [8], что недеформируемая линейка между соседними углами проходит по линии, удовлетворяющей уравнению

$$\varphi^{IV}(x)=0 \quad . \qquad (12.7)$$

Функцию $\varphi(x)$ будем использовать для интерполяции зависимости $y(x)$, заданной на интервале (a,b) в узлах $a=x_0, x_1 \ldots, b=x_n$ значениями $y_0, y_1, \ldots y_n$.

Кубическим сплайном, интерполирующим на отрезке [a,b] данную функцию y(x), называется функция [8]

$$g_k(s)=a_k+b_k(s-x_k)+c_k(s-x_k)^2+d_k(s-x_k)^3, s\in[x_{k-1},x_k], k=1,2,\ldots,n \quad . \qquad (12.8)$$

удовлетворяющая следующим условиям:

• $g_k(x_k)=y_k; g_k(x_{k-1})=y_{k-1}$ (условие интерполяции в узлах сплайна);

• функция $g(x)$, дважды непрерывно дифференцируема на интервале $[a,b]$;

• на концах интервала функция g должна удовлетворять следующим соотношениям $g_1''(a)=g_k''(b)=0$.

Для построения интерполяционного сплайна необходимо найти 4n коэффициентов a_k, b_k, c_k, d_k (k=1,2,…,n).

Из определения сплайна получаем n+1 соотношение (12.9)

$$g_1(x_0)=y_0, g_k(x_k)=y_k, k=1,2,...,n \quad . \tag{12.9}$$

Из условий гладкой стыковки звеньев сплайна (во внутренних узловых точках совпадают значения двух соседних звеньев сплайна[24], их первые и вторые производные) получаем еще ряд соотношений (12.10 – 12.11) [8]:

$$\begin{array}{l} g_{k-1}(x_{k-1})=g_k(x_k), \\ g_{k-1}'(x_{k-1})=g_k'(x_k), \\ g_{k-1}''(x_{k-1})=g_k''(x_k), \\ k=2,3,...,n. \end{array} \quad , \tag{12.10}$$

$$g_1''(x_0)=0, g_n''(x_n)=0 \quad . \tag{12.11}$$

Соотношения (12.9) – (12.11) образуют 4n соотношений для нахождения коэффициентов сплайна. Подставляя выражения функций (12.8) и их производных (2.12)

$$\begin{array}{l} g_k'(s)=b_k+2c_k(s-x_k)+3d_k(s-x_k)^2, \\ g_k''(s)=2c_k+6d_k(s-x_k), \end{array} \tag{12.12}$$

в соотношения (12.9) – (12.11) и принимая во внимание соотношение

$$h_k=x_k-x_{k-1}, \quad k=1,2,...,n \quad , \tag{12.13}$$

получим следующую систему уравнений (12.14) – (12.20)

$$a_1-b_1h_1+c_1h_1^2-d_1h_1^3=y_0 \quad , \tag{12.14}$$

$$a_k=y_k, \quad k=1,2,...,n \quad , \tag{12.15}$$

$$a_{k-1}=a_k-b_kh_k+c_kh_k^2-d_kh_k^3, \quad k=2,3,...,n \quad , \tag{12.16}$$

$$b_{k-1}=b_k-2c_kh_k+3d_kh_k^2, \quad k=2,3,...,n \quad , \tag{12.17}$$

$$c_{k-1}=c_k-3d_kh_k, \quad k=2,3,...,n \quad , \tag{12.18}$$

$$c_1-3d_1h_1=0 \quad , \tag{12.19}$$

$$c_n=0 \quad . \tag{12.20}$$

Задача интерполяции свелась к решению системы (12.14-12.20). Из соотношения (12.15) следует, что все коэффициенты $a_k=y_k, \quad k=1,2,...,n$. Подставив соотношения (12.14), (12.15) в (12.16) и используя фиктивный коэффициент $c_0=0$, получим соотношение между

24 Звеном сплайна называется функция $g_i(x)$ на интервале $[x_{i-1}, x_i]$

b_k , c_k и d_k $b_k h_k - c_k h_k^2 + d_k h_k^3 = y_k - y_{k-1}$

Отсюда коэффициенты b_k вычисляются по формуле

$$b_k = \frac{y_k - y_{k-1}}{h_k} + c_k h_k - d_k h_k^2, \quad k = 1,2,\ldots,n \quad . \tag{12.21}$$

Из (12.18) и (12.19) выразим d_k через c_k (с учётом коэффициента $c_0 = 0$)

$$d_k = \frac{c_k - c_{k-1}}{3h_k}, \quad k = 1,2,\ldots,n \quad . \tag{12.22}$$

Подставим (12.22) в (12.21)

$$b_k = \frac{y_k - y_{k-1}}{h_k} + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1}, \quad k = 1,2,\ldots,n \quad . \tag{12.23}$$

Введем обозначение

$$l_k = \frac{y_k - y_{k-1}}{h_k}, \quad k = 1,2,\ldots,n \quad , \tag{12.24}$$

после чего соотношение (12.23) примет вид:

$$b_k = l_k + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1}, \quad k = 1,2,\ldots,n \quad . \tag{12.25}$$

Подставим (12.24) и (12.22) в соотношение (12.17), получим систему относительно c_k

$$b_{k-1} c_{k-2} + 2(h_{k-1} + h_k) c_k = 3(l_k - l_{k-1}), \quad k = 2,3,\ldots,n \quad . \tag{12.26}$$

$$c_0 = 0, \quad c_n = 0 \quad . \tag{12.27}$$

Систему (12.26) можно решить, используя метод прогонки (http://ru.wikipedia.org/wiki/Метод_прогонки). Этот метод сводится к нахождению прогоночных коэффициентов по формулам прямой прогонки.

$$\delta_1 = -\frac{1}{2}\frac{h_2}{h_1 + h_2}, \quad \lambda_1 = \frac{3}{2}\frac{l_2 - l_1}{h_1 + h_2} \quad , \tag{12.28}$$

$$\delta_{k-1} = -\frac{h_k}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}}, \quad \lambda_{k-1} = \frac{3l_k - 3l_{k-1} - h_{k-1}\lambda_{k-2}}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}} \quad k = 3,4,\ldots,n \quad , \tag{12.29}$$

а затем к нахождению искомых коэффициентов c_k по формулам обратной прогонки

$$c_{k-1} = \delta_{k-1} c_k + \lambda_{k-1}, \quad k = n,,n-1,\ldots,2. \quad , \tag{12.30}$$

После нахождения коэффициентов с по формуле (12.7), находим *b* и *d* по формулам (12.22), (12.24).

Таким образом, алгоритм расчёта коэффициентов интерполяционного сплайна можно свести к следующим шагам.

Шаг 1. Ввод значений табличной зависимости $y(x)$, массивов *x* и *y*.

Шаг 2. Расчёт элементов массивов *h* и *l* по формулам (12.13) и (12.24).

Шаг 3. Расчёт массивов прогоночных коэффициентов δ и λ по формулам (12.27), (12.25).

Шаг 4. Расчёт массивов коэффициентов c по формуле (12.7).

Шаг 5. Расчёт массивов коэффициентов b по формуле (12.24).

Шаг 6. Расчёт массивов коэффициентов d по формуле (12.22).

После этого в формулу (12.8) можно подставлять любую точку s и вычислять ожидаемое значение.

Расчёт коэффициентов кубического сплайна очень громоздкий и зачастую на 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

$$f_k(s)=a_k+b_k s,\ s\in[x_{k-1},x_k],\ k=1,2,\dots,n\ , \qquad (12.31)$$

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 (12.32), которые получаются из уравнения прямой, проходящей через две точки с координатами (x_{k-1},y_{k-1}) , (x_k,y_k) .

$$a_k=y_{k-1}-\frac{y_k-y_{k-1}}{x_k-x_{k-1}}x_{k-1},\ b_k=\frac{y_k-y_{k-1}}{x_k-x_{k-1}}\ , \qquad (12.32)$$

Найдя коэффициенты линейного сплайна, можно рассчитать значения в любой точке интервала $[x_0,x_n]$. Линейная интерполяция дает достаточно хорошие результаты при практическом счёте внутри интервала $[x_0,x_n]$, когда от получаемой функции не требуют дополнительных свойств (дифференцируемости и т.д.).

Рассмотрим реализацию сплайн-интерполяции в Octave. Это можно сделать запрограммировав рассмотренные выше методы сплайн-интерполирования в Octave или воспользовавшись функцией `interp1`

```
interp1(x, y, xi, '<метод>'),
```

где `x` – массив абсцисс экспериментальных точек, `y` – массив ординат экспериментальных точек, `xi` – точки, в которых необходимо вычислить значение с помощью сплайна, `method` – определяет метод построения сплайна для реализации сплайн-интерполяции. Параметр `method` может принимать одно из следующих значений: `'linear'` – линейная интерполяция, `'spline'` – кубический сплайн.

Рассмотрим несколько практических задач.

ЗАДАЧА 12.2. В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P_0 , Вт) от входного напряжения (U_1 , В) для асинхронного двигателя МТН111-6 (табл. 12.3). Построить график интерполяционной зависимости. Вычислить ожидаемое значение мощности при $U_1=145,155,175\,B$.

Таблица 12.3: Экспериментальные данные

Напряжение U_1 , В	132	140	150	162	170	180
Мощность P_0 , Вт	330	350	385	425	450	485

Реализуем рассмотренный ранее алгоритм кубического сплайна. Решение с комментариями представлено на листинге 12.5.

```
function [b, c,d]=koef_spline(x,y)
% Функция предназначена для вычисления коэффициентов сплайна,
% здесь x,y - массивы абсцисс и ординат экспериментальных
```

```
% точек, b, c,d - коэффициенты сплайна, рассчитываемые по
% формулам (12.21), (12.23), (12.12), (12.7)
n=length(x);
for k=2:n
h(k)=x(k)-x(k-1);
end
for k=2:n
l(k)=(y(k)-y(k-1))/h(k);
end
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)*delt(k-2))/(2*(h(k-1)+...
h(k))+h(k-1)*delt(k-2));
end
c(n)=0;
for k=n:-1:3
c(k-1)=delt(k-1)*c(k)+delt(k-1);
end
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
end
function z=my_spline(x,y,t)
% Вычисление значение кубического сплайна в точке t,
% здесь x,y - массивы абсцисс и ординат экспериментальных
% точек
[b,c,d]=koef_spline(x,y);
n=length(x);
a=y;
%j — номер интервала, которому принадлежит 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
z=a(j)+b(j)*(t-x(j))+c(j)*(t-x(j))^2+d(j)*(t-x(j))^3;
end
%Экспериментальные точки.
U1=[132 140 150 162 170 180];
P0=[330 350 385 425 450 485];
%Точки, в которых надо посчитать ожидаемое значение сплайна.
x=[145 155 175];
for i=1:3
```

```
%Расчёт ожидаемого значения с помощью функции my_spline.
y(i)=(U1,P0,x(i));
end
%Вычисление значений для построения графика слайна.
U2=132:1:180;
for i=1:length(U2)
P2(i)=my_spline(U1,P0,U2(i));
end
x
y
%Построение графика.
plot(U1,P0,'*b;experiment;',U2,P2,...
'-r;spline;',x,y,'pb;points;');
Y
grid on;
%Результаты:
>>>x =
145 155 175
>>>y =
373.19 408.77 471.15
```

Листинг 12.5.

На рис. 12.2 можно увидеть графическое решение задачи 12.2.

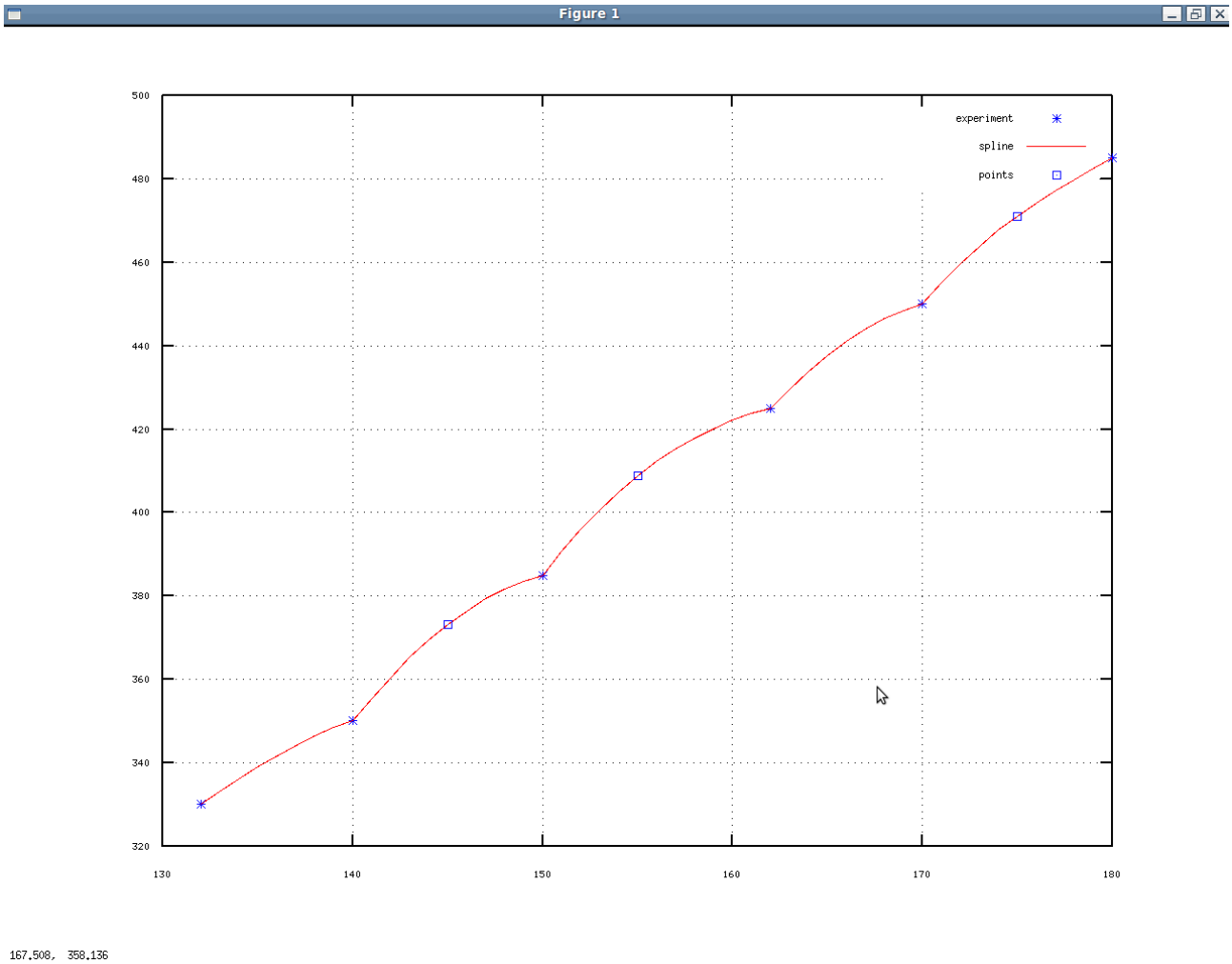


Рис. 12.2. Графическое решение задачи 12.2.

В следующей задаче воспользуемся встроенными функциями Octave.

ЗАДАЧА 12.3. В результате опыта холостого хода определена зависимость $u(x)$ (табл. 12.1). Построить график интерполяционной зависимости. Вычислить ожидаемое значение мощности при $x = 0.308, 0.325, 0.312$.

Таблица 12.4: Результаты эксперимента

x	0.298	0.303	0.31	0.317	0.323	0.33
u	3.25578	3.17639	3.1218	3.04819	2.98755	2.9195

Для решения задачи (листинг 12.6) воспользуемся функцией `interp1`.

```
%Экспериментальные точки
x=[0.298 0.303 0.31 0.317 0.323 0.33];
u=[3.25578 3.17639 3.1218 3.04819 2.98755 2.9195];
%Точки, в которых надо посчитать ожидаемое значение.
X1=[0.308 0.312 0.325];
%Расчёт значений в точках 0.308, 0.312, 0.325 с помощью
% кубического сплайна.
u1s=interp1(x,u,x1,'spline')
%Расчёт значений в точках 0.308, 0.312, 0.325 с помощью
% линейного сплайна.
u1l=interp1(x,u,x1,'linear')
%Вычисление значений для построения графиков линейного и
% кубического слайнов.
xi=0.298:0.002:0.33;
uxis=interp1(x,u,xi,'spline');
uxil=interp1(x,u,xi,'linear');
%Построение графика
plot(x,u,'*b;experimentl;',xi,uxil,'-r;linear spline;',...
xi,uxis,'-b;cubic spline;',x1,u1s,...
'pr;points (cubic spline);',x1,u1l,...
'<b;points (linear spline);');
axis([0.29,0.34,2.8,3.3]);
grid on;
%Результаты:
>>>u1s =
3.1370 3.1031 2.9685
>>>u1l =
3.1374 3.1008 2.9681
```

Листинг 12.6.

Рис. 12.3 представляет графическое решение задачи.

Как и при решении задачи аппроксимации (подбора кривой методом наименьших квадратов), при построении интерполяционных зависимостей необходимо первоначально поставить и решить задачу с точки зрения математики и только потом использовать функции пакета Octave.

Этой задачей мы завершаем краткое введение в Octave. В книге были рассмотрены только некоторые функции и методы решения математических и инженерных задач. Далее читатель может самостоятельно продолжить изучение Octave. Официальная страница справки http://www.gnu.org/software/octave/doc/interpreter/. Следует помнить, что существует огромное количество расширений к пакету. Описание пакетов расширений приведено на странице http://octave.sourceforge.net.

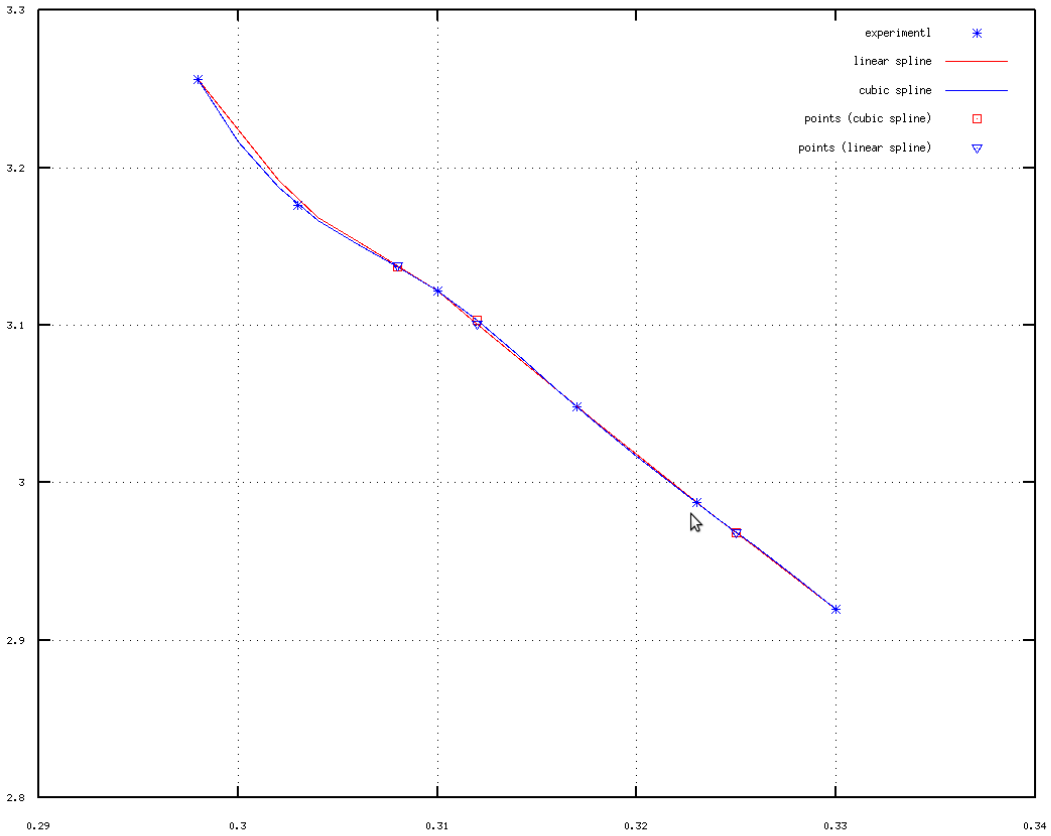


Рис. 12.3 Графическое решение задачи 12.3

Алфавитный указатель

Список литературы

1. Акулич И.Л. Математическое программирование в примерах и задачах. - М.: Высшая школа, 1986. - 319с.
2. Алексеев Е. Р., Чеснокова О. В. Matlab 7. Самоучитель. М.: НТ Пресс, 2006. – 464с.
3. Алексеев Е. Р., Чеснокова О. В. Mathcad 12. Самоучитель. М.: НТ Пресс, 2005. – 345с.
4. Алексеев Е. Р., Чеснокова О. В. Решение задач вычислительной математики в пакетах MathCad, Matlab и Maple. Самоучитель. М.: НТ Пресс, 2006. – 496с.
5. Алексеев Е. Р., Чеснокова О. В., Павлыш В. Н., Славинская Л. В. Турбо Паскаль 7.0. М.: НТ Пресс, 2004, – 270с.
6. Березин И. С., Жидков Н. П. Методы вычислений. М.:Наука, 1966. – 632с.
7. Бронштейн И. Н., Семендяев К. А. Справочник по математике для инженеров и учащихся вузов. – М.: Наука, 1981. – 720 с.
8. Вержбицкий В. М. Основы численных методов. – М: Высшая школа, 2002. – 840с.
9. Демидович Б. П., Марон И.А., Шувалова В.З. Численные методы анализа. – М.: Наука, 1967. – 368с.
10. Scilab: Решение инженерных и математических задач/ Е.Р. Алексеев, О.В. Чеснокова, Е.А. Рудченко. – М.: ALT Linux; БИНОМ. Лаборатория знаний, 2008. – 260с.: 8с. ил.:.- (Библиотека ALT Linux).
11. Octave. URL: http://www.gnu.org/software/octave (дата обращения 19.09.2011).
12. Octave - Wikipedia, the free encyclopedia. URL: http://en.wikipedia.org/wiki/Octave (дата обращения 19.09.2011).
13. GNU Octave - Википедия. URL: http://ru.wikipedia.org/wiki/GNU_Octave (дата обращения 19.09.2011).

Для заметок

ISBN 978-966-8248-30-6

А 47 **Алексеев Е.Р., Чеснокова О.В.** GNU Octave для студентов и преподавателей. - Донецк.: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 330с.

ISBN 978-966-8248-30-6 **ООО «**Технопарк ДонНТУ УНИТЕХ**»**

Издательство: ООО «Технопарк ДонНТУ УНИТЕХ»

Свидетельство о внесении субъекта издательского дела в государственный реестр издателей, изготовителей и распространителей издательской продукции:Дк №1017 от 21.08.2002.

83000, г. Донецк, ул. Артема, 58, к.1.311

Тел. : (062) 304 90 19

Технический редактор: Аноприенко А.Я.

Дизайн обложки: Гутаревич Н.

Подписано к печати: 13 сентября 2011г.
Формат 70×90¼
Усл.печ.л. 25.38
Печать лазерная.
Заказ № 454
Тираж 200 экз.

Отпечатано в типографии «Друк-Инфо»
Адрес: 83000, г.Донецк, ул. Артема, 58
Тел.: (062) 335 64 55