

В серии:

Библиотека ALT Linux

Введение в Octave для инженеров и математиков

Е. Р. Алексеев, О. В. Чеснокова

Москва
ALT Linux
2012

УДК 519.67

ББК 22.1

A47

Введение в Octave для инженеров и математиков:
A47 / Е. Р. Алексеев, О. В. Чеснокова — М.: ALT Linux, 2012. —
368 с.: ил. — (Библиотека ALT Linux).

ISBN 978-5-905167-10-2

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**.

Сайт книги: <http://www.altlinux.org/Books:Octave>

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

УДК 519.67

ББК 22.1

**По вопросам приобретения обращаться: ООО «Альт
Линукс» (495)662-38-83 E-mail: sales@altlinux.ru
<http://altlinux.ru>**

Материалы, составляющие данную книгу, распространяются на условиях лицензии GNU FDL. Книга содержит следующий текст, помещаемый на первую страницу обложки: «В серии “Библиотека ALT Linux”. Название: «Введение в Octave для инженеров и математиков». Книга не содержит неизменяемых разделов. Авторы разделов указаны в заголовках соответствующих разделов. ALT Linux — торговая марка компании ALT Linux. Linux — торговая марка Линуса Торвальдса. Прочие встречающиеся названия могут являться торговыми марками соответствующих владельцев.

ISBN 978-5-905167-10-2

© Е. Р. Алексеев, О. В. Чеснокова, 2012

© ALT Linux, 2012

Оглавление

Введение	6
Глава 1. Общие сведения, установка	8
1.1 Принципы работы с интерпретатором	8
1.2 Установка Octave	14
1.3 Графическая оболочка QtOctave	17
Глава 2. Основы работы	24
2.1 Элементарные математические выражения	24
2.2 Текстовые комментарии	25
2.3 Представление вещественного числа	25
2.4 Переменные	28
2.5 Функции	30
2.6 Массивы	40
2.7 Символьные вычисления	43
Глава 3. Программирование	46
3.1 Основные операторы языка программирования . . .	46
3.2 Обработка массивов и матриц	56
3.3 Обработка строк	62
3.4 Работа с файлами	65
3.5 Функции	79
Глава 4. Построение графиков	87
4.1 Построение двумерных графиков	87
4.2 Построение трёхмерных графиков	109
4.3 Анимация	125
4.4 Графические объекты	127
Глава 5. Задачи линейной алгебры	141
5.1 Ввод и формирование векторов и матриц	141
5.2 Действия над векторами	144

5.3	Действия над матрицами	147
5.4	Функции для работы с матрицами и векторами . . .	152
5.5	Решение некоторых задач алгебры матриц	179
5.6	Решение систем линейных уравнений	185
5.7	Собственные значения и собственные векторы	198
5.8	Норма и число обусловленности матрицы	201
5.9	Задачи линейной алгебры в символьных вычислениях	203
Глава 6. Векторная алгебра и аналитическая геометрия		206
6.1	Векторная алгебра	206
6.2	Аналитическая геометрия	223
Глава 7. Нелинейные уравнения и системы		241
7.1	Решение алгебраических уравнений	241
7.2	Решение трансцендентных уравнений	249
7.3	Решение систем нелинейных уравнений	252
7.4	Решение уравнений и систем в символьных переменных	259
Глава 8. Интегрирование и дифференцирование		262
8.1	Вычисление производной	262
8.2	Исследование функций	266
8.3	Численное интегрирование	269
Глава 9. Решение обыкновенных дифференциальных уравнений		280
9.1	Общие сведения о дифференциальных уравнениях .	280
9.2	Численные методы решения дифференциальных уравнений	282
9.3	Реализация численных методов	289
9.4	Решение систем дифференциальных уравнений . . .	297
9.5	Функции для решения дифференциальных уравнений	298
Глава 10. Решение оптимизационных задач		304
10.1	Поиск экстремума функции	304
10.2	Решение задач линейного программирования	311
Глава 11. Метод наименьших квадратов		326
11.1	Постановка задачи	326

11.2	Подбор параметров экспериментальной зависимости	327
11.3	Уравнение регрессии и коэффициент корреляции . .	332
11.4	Нелинейная корреляция	334
11.5	Подбор зависимостей методом наименьших квадратов	335
Глава 12. Интерполяция функций		346
12.1	Постановка задачи	346
12.2	Интерполяция сплайнами	354
Список литературы		362
Предметный указатель		363

Введение

Книга, которую держит в руках читатель, посвящена **GNU Octave** — одной из самых интересных прикладных программ для решения инженерных и математических задач.

GNU Octave — это свободный интерпретирующий язык для проведения математических вычислений. По возможностям и качеству реализации интерпретатора язык Octave можно сравнивать с проприетарной программой MATLAB, причём синтаксис обоих языков очень схож.

Существуют версии языка для различных дистрибутивов GNU Linux (ALT Linux, Debian, Ubuntu, Mandriva и др.) и для ОС Windows. На наш взгляд, **GNU Octave** больше ориентирован на работу в Linux. Работа в ОС Windows возможна, но пользователю Windows надо быть готовым работать с простым текстовым редактором и командной строкой.

Когда авторы начинали знакомиться с **GNU Octave**, основной проблемой было отсутствие хорошего русскоязычного введения в этот язык. Наша книга является попыткой восполнить этот пробел. Поэтому большое внимание было уделено самому языку (глава 3), операциям с матрицами (глава 5) и графическим возможностям пакета (глава 4).

Наш многолетний опыт преподавания информационных дисциплин в Донецком национальном техническом университете говорит нам о том, что студенту и инженеру наряду с описанием функций, предназначенных для решения той или иной задачи, не лишним будет напомнить и математическую постановку решаемой задачи, а зачастую и численные методы решения задачи. Именно поэтому в ряде глав приведены не только описания функций, но и описаны численные методы решения задач.

Что касается графических оболочек, таких как `qtOctave`, `Xoctave` и `Kalculus`, нами принято решение кратко описать наиболее стабильную из них, `qtOctave`, а основное внимание в книге уделить собственно языку. Мы считаем, что **GNU Octave** — это в первую очередь мощный интерпретирующий язык. Зная его, пользователь сможет работать с любой графической оболочкой.

Авторы выражают благодарность компании ALT Linux за многолетнее сотрудничество и возможность издать очередную книгу.

Авторы заинтересованы в общении с читателями. Мы ждём ваши замечания и отзывы по адресам `EAlekseev@gmail.com` и `chesn_o@list.ru`.

Донецк, апрель 2012

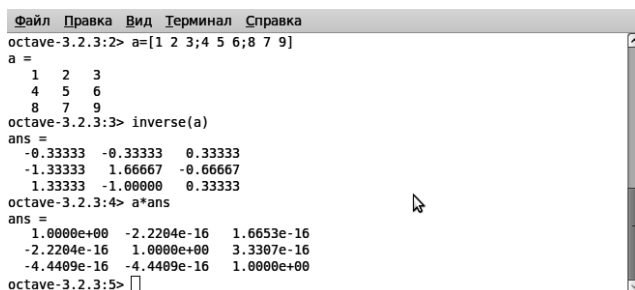
Глава 1

Общие сведения, установка

1.1 Принципы работы с интерпретатором

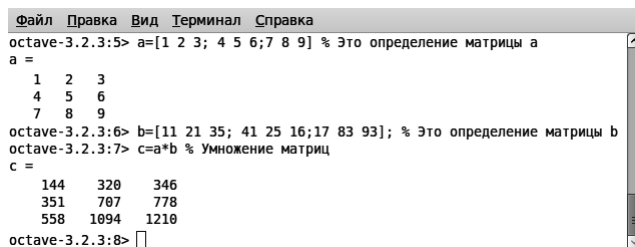
Octave — высокоуровневый интерпретируемый язык программирования, предназначенный для решения задач вычислительной математики. В состав пакета входит интерактивный командный интерфейс (интерпретатор **Octave**). Интерпретатор **Octave** запускается из терминала ОС Linux или из его порта в Windows. После запуска **Octave** пользователь видит окно интерпретатора (см. рис. 1.1).

В окне интерпретатора пользователь может вводить как отдельные команды языка **Octave**, так и группы команд, объединяемые в программы. Если строка заканчивается символом «;», результаты на экран не выводятся. Если же в конце строки символ «;» отсутствует,



```
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**



```

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 b
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**

результаты работы выводятся на экран (см. рис. 1.2). Текст в строке после символа % (процент) является комментарием и интерпретатором не обрабатывается¹ (см. рис. 1.2). Рассмотрим несколько несложных примеров.

Пример 1.1. Решить систему линейных алгебраических уравнений (СЛАУ)

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

Возможны два варианта решения любой задачи в **Octave**:

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

Для решения СЛАУ в окне интерпретатора **Octave** последовательно введём следующие команды (листинг 1.1):

```

% Определение матрицы коэффициентов системы линейных уравнений.
A=[3 5 -7;3 -4 33;22 -11 17];
b=[11; 25; 22]; % Вектор правых частей СЛАУ.
x=A^(-1)*b % Решение системы методом обратной матрицы.
x =
    1.56361
    2.55742

```

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

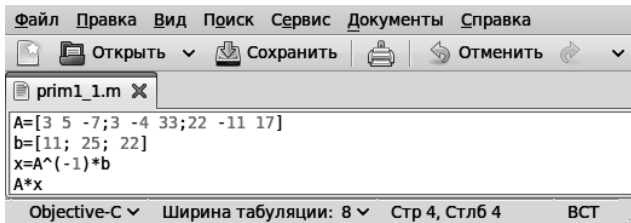


Рис. 1.3. Программа для решения примера 1.1

```
0.92542
octave -3.2.3:27 > A*x % Проверка.
ans =
  11.000
  25.000
  22.000
```

Листинг 1.1. Решение СЛАУ примера 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 редактор², например *gedit*, в окне которого последовательно введём следующие команды:

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

Сохраним введённые 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. Для этого в окне интерпретатора введём команды:

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

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

```

Файл  Правка  Вид  Терминал  Справка
A =
     3     5    -7
     3    -4    33
    22   -11    17
b =
    11
    25
    22
x =
    1.56361
    2.55742
    0.92542
ans =
    11.0000
    25.0000
    22.0000
~
(END)

```

Рис. 1.4. Окно терминала после запуска программы `prim1_1`

Окно интерпретатора примет вид, представленный на рис. 1.4. Просмотрев результаты работы программы, нажмите `q` для возвращения в режим ввода команд терминала.

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

Напомним читателю, что корни квадратного уравнения определяются по формулам $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, 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 из примера 1.2 приведён в листинге 1.2.

```

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 1.2. Решение квадратного уравнения (пример 1.2).

Для запуска программы на выполнения в окне интерпретатора введём текст:

```

cd '/home/evgeniy'
prim1_2

```

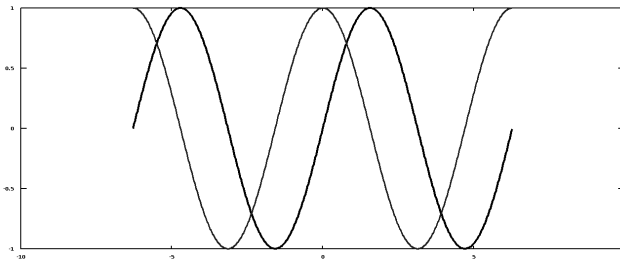


Рис. 1.5. Графики функций $y = \sin(x)$, $z = \cos(x)$

Здесь `/home/evgeniy` — имя папки, где хранится программа, `prim1_2.m` — имя файла в папке `/home/evgeniy`, где хранится листинг 1.2.

Далее пользователь должен ввести значение переменных a , b и c , после чего появятся результаты работы программы:

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

Пример 1.3. Построить графики функций $y = \sin(x)$ и $z = \cos(x)$ на интервале $[-2\pi; 2\pi]$.

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)
```

Листинг 1.3. Построение графиков функций из примера 1.3.

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

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

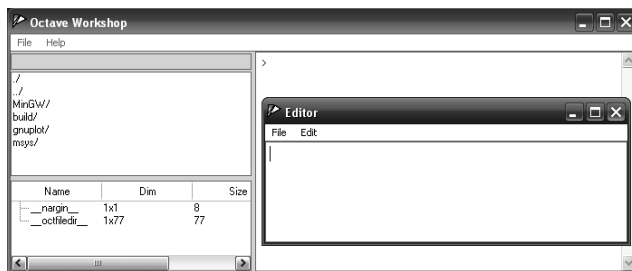


Рис. 1.6. Окно Octave Workshop

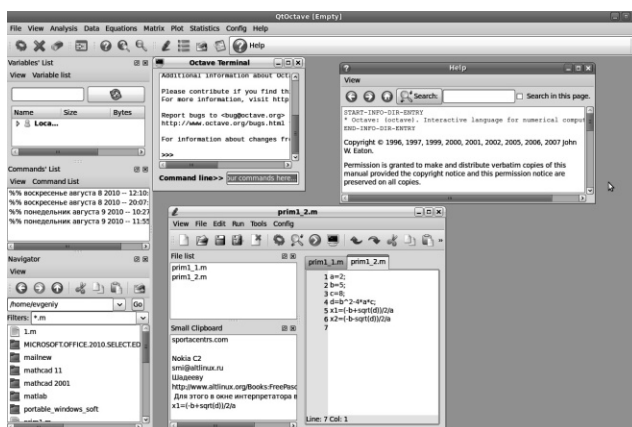


Рис. 1.7. Окно QtOctave

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.

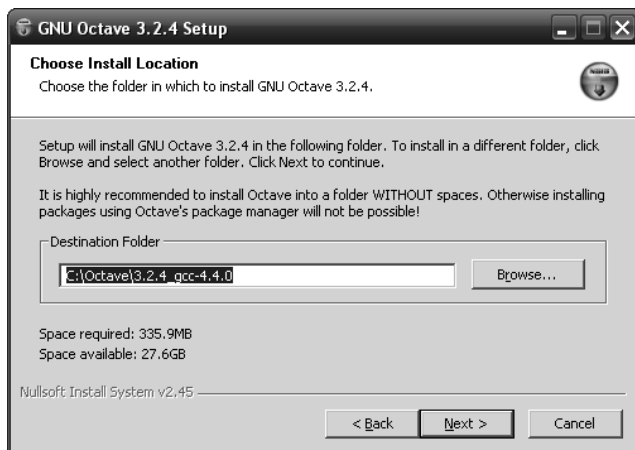


Рис. 1.8. Установка **Octave**. Выбор папки для установки.

1.2 Установка Octave

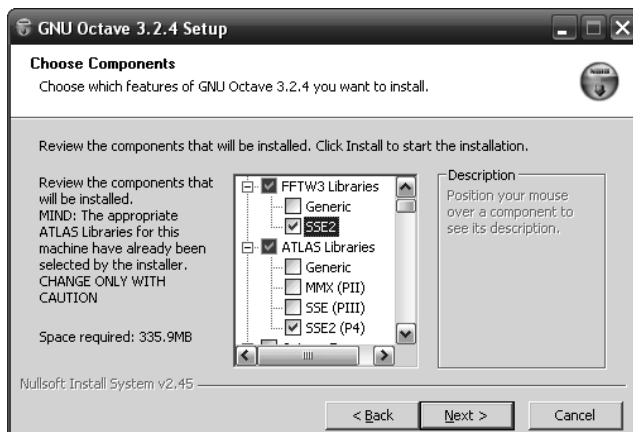
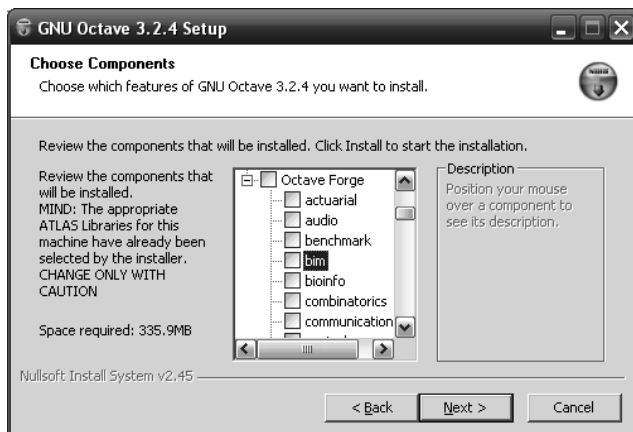
1.2.1 Установка Octave в Windows

Установка **Octave** в ОС Windows проходит стандартным образом. Необходимо с официального сайта <http://www.gnu.org/software/octave/> скачать версию программы для Windows и установить её.

На первом этапе нужно выбрать папку для установки программы (рис. 1.8). На следующем — 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 **Notepad++**, интерпретатор **Octave**, пакеты расширений и 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://sourceforge.net/projects/octave-workshop/>

Рис. 1.9. Установка **Octave**. Выбор архитектуры процессора.Рис. 1.10. Установка **Octave**. Выбор пакетов расширений.

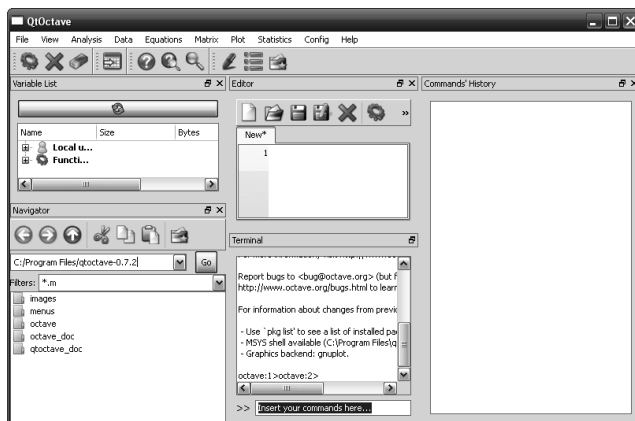


Рис. 1.11. Окно QtOctave под управлением ОС Windows

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

Наиболее мощной графической оболочкой для работы с **Octave** является программа **QtOctave**, которая разработана для ОС Linux. Она портирована в ОС Windows в виде portable-версии, которую можно скачать по адресу <http://portableapps.com/node/14720> (https://forja.rediris.es/frs/download.php/433/qt octave0.6.8_octave2.9.15_Portable_win32.zip)

Программу нужно разархивировать и запустить файл `qt octave.exe`. Окно **QtOctave** в ОС Windows представлено на рис. 1.11.

1.2.2 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).

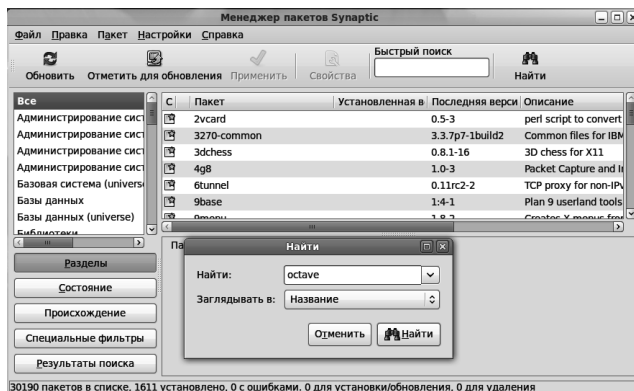


Рис. 1.12. Окно менеджера пакетов Synaptic

В менеджере пакетов **Synaptic** нужно щёлкнуть по кнопке **Найти**, и в строке поиска ввести: **octave**. В результате поиска пользователю будет предложен список, в котором нужно выбрать:

- **qt octave** — 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;
- необходимые пользователю пакеты расширений, например, **octave-linear-algebra**, **octave-optim** и многие другие³.

Установка начнётся после щелчка по кнопке **Применить**. Время установки зависит от количества выбранных пакетов и скорости сетевого соединения. После установки в группе программ Программирование и Наука появятся ярлыки программ **GNU Octave** (интерпретатор **Octave**) и **QtOctave** (графическая оболочка **Octave**).

1.3 Графическая оболочка QtOctave

После запуска **QtOctave** на экране появляется основное окно приложения (рис. 1.13). Это окно содержит меню, панель инструментов и

³Номера 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.

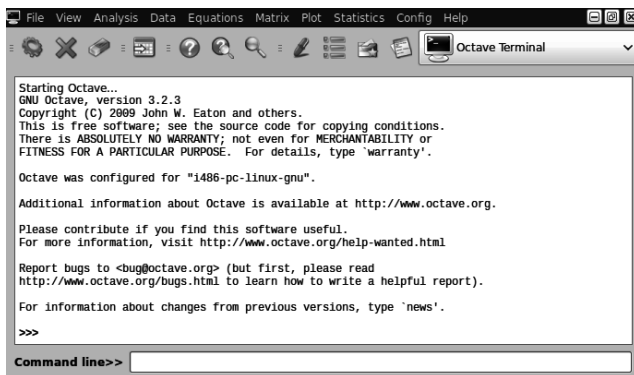


Рис. 1.13. Окно QtOctave

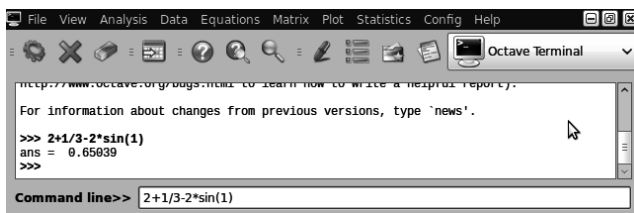


Рис. 1.14. Выполнение элементарной команды

рабочую область Octave Terminal. Окно может иметь другой внешний вид, в зависимости от предпочтений пользователя.

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, что проиллюстрировано на рис. 1.14.

Понятно, что все выполняемые команды не могут одновременно находиться в поле зрения пользователя. Поэтому просмотреть информацию, которая покинула видимую часть окна, можно с помощью полос прокрутки и клавиш **Page Up** и **Page Down**.

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

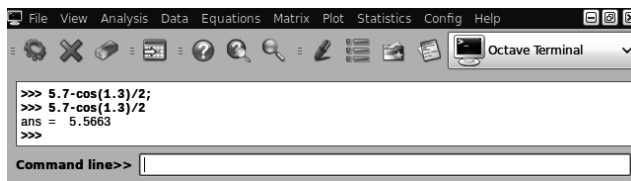


Рис. 1.15. Примеры вывода результатов вычислений

пустой активной командной строке нажать клавишу \uparrow , в ней появится последняя вводимая команда. Повторное нажатие вызовет предпоследнюю команду, и так далее. Клавиша \downarrow выводит команды в обратном порядке.

Таким образом можно сказать, что вся информация в рабочей области находится в зоне просмотра. Важно знать, что в зоне просмотра нельзя ничего исправить или ввести. Единственная допустимая операция — это выделение информации с помощью мыши и копирование её в буфер обмена, к примеру, для дальнейшего помещения в командную строку.

В командной строке действуют элементарные приёмы редактирования:

- \rightarrow — перемещение курсора вправо на один символ;
- \leftarrow — перемещение курсора влево на один символ;
- **Home** — перемещение курсора в начало строки;
- **End** — перемещение курсора в конец строки;
- **Del** — удаление символа после курсора;
- **Backspace** — удаление символа перед курсором.

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

Работа в среде **QtOctave** может осуществляться в так называемом 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.

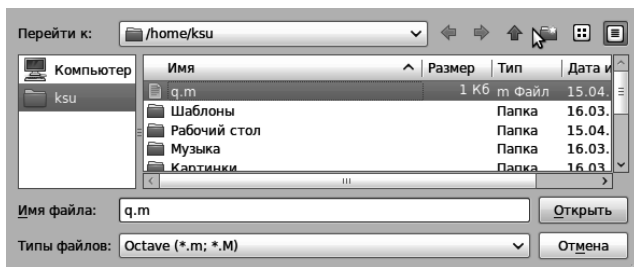


Рис. 1.16. Выбор файла для выполнения в **Octave**

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.

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 terminal**. Команда продублирована кнопкой в виде ластика на панели инструментов.

Команда **View** → **Dock Tools** → **Variable List** открывает окно, показанное на рис. 1.17. Здесь 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.

Окно, представленное на рис. 1.18, содержит список выполненных команд и открывается командой **View** → **Dock Tools** → **Command List**.

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**.

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

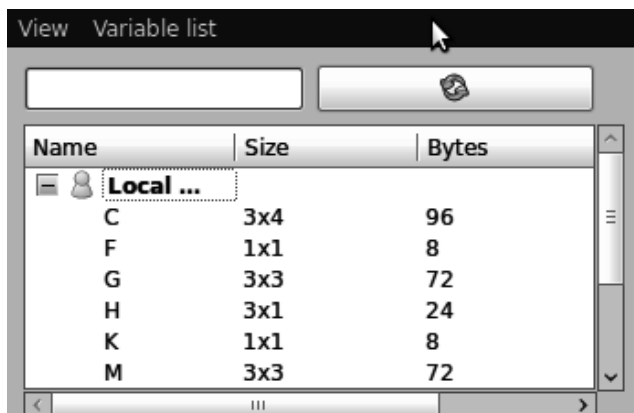


Рис. 1.17. Список переменных, определённых в процессе работы

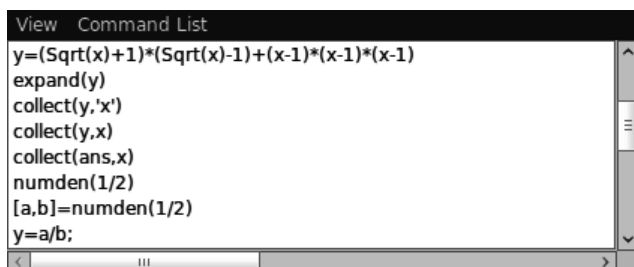


Рис. 1.18. Список выполненных команд

для решения биквадратного уравнения $2x^4 - 9x^2 + 4 = 0$. Нетрудно заметить, что точка с запятой «;» ставится после тех команд, которые не требуют вывода значений.

Для сохранения введённой информации необходимо выполнить команду **File** → **Save** из главного меню редактора. Если информация сохраняется впервые, 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 и нажать **ENTER**.

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

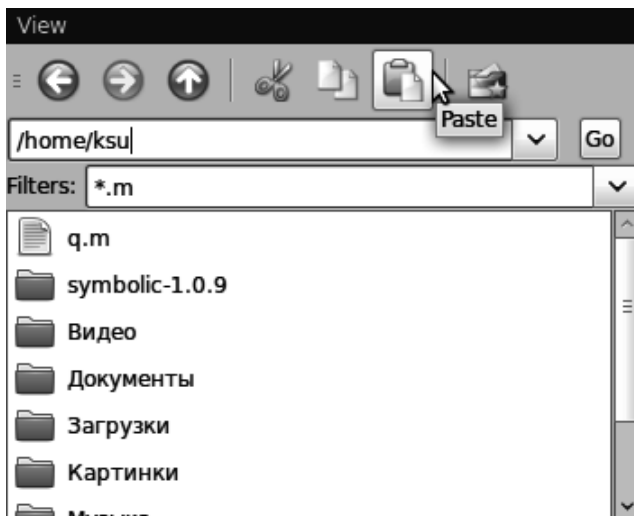


Рис. 1.19. Смена каталога

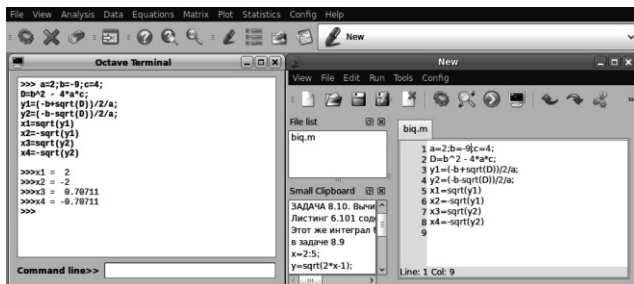


Рис. 1.20. Пример работы в текстовом редакторе

Отметим, что текстовый редактор имеет возможность работы с множеством окон и обладает принятыми для текстовых редакторов приёмами редактирования, подробно останавливаться на которых мы не будем.

Для выхода из режима редактирования можно просто закрыть окно или использовать команду **File** → **Close**. Ранее созданный файл открывает 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 матанализа (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 трёхмерных графиков, форматирование графической области, запись графического изображения в файл);
- **Statistics** — вычисление некоторых статистических функций;
- **Config** — настройка конфигурации системы, установка пакетов расширений;
- **Help** — справочная информация.

Глава 2

Основы работы

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

Простейшие арифметические операции в **Octave** выполняются с помощью следующих операторов:

- $+$ сложение;
- $-$ вычитание;
- $*$ умножение;
- $/$ деление слева направо;
- \backslash деление справа налево;
- $^$ возведение в степень.

Вычислить значение арифметического выражения можно вводом его в командную строку и нажатием клавиши **ENTER**:

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

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

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


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

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

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

2.2 Текстовые комментарии

Правилом хорошего тона в программировании всегда считался легко читаемый программный код, снабжённый большим количеством комментариев. Поскольку далее речь пойдёт о достаточно сложных вычислениях, для наглядности мы также 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 активации следующей командной строки.

Примеры использования текстовых комментариев приведены в следующем разделе.

2.3 Представление вещественного числа

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 $mE \pm p$, 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 умножить *мантиссу* на десять в степени *порядок*. Например,

$$6.42E+2 = 6.42 \cdot 10^2 = 642$$

$$-3.2E-6 = -3.2 \cdot 10^{-6} = -0.0000032$$

Ниже приведён пример ввода вещественного числа.

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

Нетрудно 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
```

- 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
```

- 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
```

- 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 % Ответ в одну строку — здесь две из-за ширины страницы
ans = 010000000000010010010000111111011
      01010100010001000010110100011000
>>> format native-bit
>>> pi % Ответ в одну строку
ans = 00011000101101000010001000101010
      11011111100001001001000000000010
```

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

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

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

- 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
```

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** 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. Таким образом, если в общем виде оператор присваивания записать как **имя_переменной** = **значение_выражения**, то в переменную, имя которой указано слева, будет записано значение выражения, указанного справа.

Имя переменной не должно совпадать с именами встроенных процедур, функций и встроенных переменных системы. Система различает большие и малые буквы в именах переменных. А именно: **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 идёт о символьной или строковой переменной, то выражение в правой части оператора присваивания следует брать в одинарные кавычки.

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

Рассмотрим несколько примеров присвоения значений переменным:

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

Обратите внимание, что если символ ; в конце выражения отсутствует, то в качестве результата выводится имя переменной и её значение. Наличие символа ; передаёт управление следующей командной строке. Это позволяет использовать имена переменных для записи промежуточных результатов в память компьютера.

Листинг ниже содержит пример использования в выражении не определённой ранее переменной.

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

Если команда не содержит знака присваивания, то по умолчанию вычисленное значение присваивается специальной системой переменной *ans*. Причём 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*:

```
>>> 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
>>> ans % Значение системной переменной не изменилось и равно 44
ans = 44
```

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

- *ans* — результат последней операции без знака присваивания;
- *i*, *j* — мнимая единица ($\sqrt{-1}$);
- *pi* — число π (3.141592653589793);
- *e* — число Эйлера ($e = 2.71828183$);
- *inf* — машинный символ бесконечности (∞);
- *NaN* — неопределённый 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 идёт об уничтожении определения одной или нескольких переменных, то можно применить команду: **clear** *имя_переменной*

Пример применения команды **clear**:

```
>>> 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
```

2.5 Функции

Все функции, используемые в **Octave**, можно разделить на два класса *встроенные* и *определённые 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
```

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

2.5.1 Элементарные математические функции

Далее приведены элементарные математические 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
$\operatorname{asin}(x)$	арксинус числа x
$\operatorname{acos}(x)$	арккосинус числа x
$\operatorname{atan}(x)$	арктангенс числа x
$\operatorname{acot}(x)$	арккотангенс числа x
$\operatorname{asec}(x)$	арксеканс числа x
$\operatorname{acsc}(x)$	арккосеканс числа x

Примеры работы с тригонометрическими функциями:

```
>>> x=pi/7
x = 0.44880
>>> 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
```

Таблица 2.2: Экспоненциальные функции

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

Применение экспоненциальных функций:

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

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
$\operatorname{sech}(x)$	гиперболический секанс числа x
$\operatorname{csch}(x)$	гиперболический косеканс числа x

Листинг ниже содержит примеры работы с гиперболическими функциями.

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

Таблица 2.4: Целочисленные функции

Функция	Описание функции
$\operatorname{fix}(x)$	округление числа x до ближайшего целого в сторону нуля

Таблица 2.4 — продолжение

Функция	Описание функции
$\text{floor}(x)$	округление числа x до ближайшего целого в сторону отрицательной бесконечности
$\text{ceil}(x)$	округление числа x до ближайшего целого в сторону положительной бесконечности
$\text{round}(x)$	обычное округление числа x до ближайшего целого
$\text{rem}(x, y)$	вычисление остатка от деления x на y
$\text{sign}(x)$	сигнум-функция (знак) числа x , выдаёт 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

```

Таблица 2.5: Другие элементарные функции

Функция	Описание функции
$\text{sqrt}(x)$	корень квадратный из числа x

Таблица 2.5 — продолжение

Функция	Описание функции
$abs(x)$	модуль числа x
$\log_{10}(x)$	десятичный логарифм от числа x
$\log_2(x)$	логарифм по основанию два от числа x
$pow_2(x)$	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
```

2.5.2 Комплексные числа. Функции комплексного аргумента

Рассмотрим реализацию комплексной арифметики в **Octave**. Как было отмечено выше, для обозначения мнимой единицы зарезерви-

вано 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
```

Кроме того, к комплексным числам применимы элементарные арифметические операции: $+$, $-$, $*$, \backslash , $/$, $^$; например:

```
>>> 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
```

Функции для работы с комплексными числами приведены в таблице 2.6.

Таблица 2.6: Функции работы с комплексными числами

Функция	Описание функции
$real(Z)$	выдаёт действительную часть комплексного аргумента Z
$imag(Z)$	выдаёт мнимую часть комплексного аргумента Z
$angle(Z)$	вычисляет значение аргумента комплексного числа Z в радианах от $-\pi$ до π
$conj(Z)$	Выдаёт число, комплексно сопряжённое Z

Примеры использования функций из таблицы 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
```

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
```

2.5.3 Операции отношения

Операции отношения выполняют сравнение двух операндов и определяют, истинно выражение или ложно (таблица 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

Таблица 2.8. Логические операции¹

A	B	$\neg A$	$A \& B$	$A \vee B$	$A \underline{\vee} B$
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Таблица 2.9. Логические операции отношения

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

2.5.4 Логические выражения

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

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

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 путём 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 $\neg A$ — математическая запись «не A», $A \& B$ — запись «A и B», $A \vee B$ — запись «A или B», $A \underline{\vee} B$ — запись «A исключаящее или B».

1. Логические операторы.
2. Логическая операция \sim .
3. Транспонирование матриц, операции возведения в степень, унарный $+$ и $-$.
4. Умножение, деление
5. Сложение, вычитание.
6. Операции отношения.
7. Логическая операция «и» — $\&$
8. Логическая операция «или» — $|$

Ниже 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
```

2.5.5 Функции, определённые пользователем

В первой главе мы уже рассмотрели создание небольшой программы, которая решает конкретное квадратное уравнение. В этой программе отсутствовал заголовок (первая строка определённого вида), и в неё 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 также имена из списка входных и выходных параметров воспринимаются системой как локальные, то есть эти переменные считаются определёнными 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 её 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. Вы можете вызвать функцию из командной строки или использовать её как один из операторов программы.

Пример 2.1. Создать функцию для решения кубического уравнения.

Кубическое уравнение

$$ax^3 + bx^2 + cx + d = 0$$

после деления на a принимает канонический вид:

$$x^3 + rx^2 + sx + t = 0, \quad (2.1)$$

$$\text{где } r = \frac{b}{a}, s = \frac{c}{a}, t = \frac{d}{a}$$

В уравнении (2.1) сделаем замену

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

и получим следующее приведённое уравнение:

$$y^3 + py + q = 0, \quad (2.2)$$

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

Число действительных корней приведённого уравнения (2.2) зависит от знака дискриминанта $D = \left(\frac{p}{3}\right)^3 + \left(\frac{q}{2}\right)^3$ (табл. 2.10).

Корни приведённого уравнения могут быть рассчитаны по формулам Кардано:

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

Таблица 2.10. Количество корней кубического уравнения

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

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

Далее представлен список команд (листинг 2.1), реализующий описанный выше способ решения кубического уравнения:

```
>>> function [x1,x2,x3]=cub(a,b,c,d)
r=b/a; s=c/a; t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=-(u+v)/2+(u-v)/2*i*sqrt(3);
y3=-(u+v)/2-(u-v)/2*i*sqrt(3);
x1=y1-r/3;
x2=y2-r/3;
x3=y3-r/3;
endfunction
% Вычисляем корни уравнения 3x^3 - 2x^2 - x - 4 = 0
>>> [x1,x2,x3]=cub(3,-2,-1,-4)
x1 = 1.4905
x2 = -0.41191 + 0.85141 i
x3 = -0.41191 - 0.85141 i
```

Листинг 2.1. Нахождение корней кубического уравнения (пример 2.1).

2.6 Массивы

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 ти-

па. Как и любой другой переменной, массиву должно быть присвоено имя.

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

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, 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 котором хранится необходимый элемент.

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

Самый простой способ задать одномерный массив в **Octave** имеет вид **имя массива** = $X_n : dX : X_k$, где X_n — значение первого элемента массива, X_k — значение последнего элемента массива, dX — шаг, с помощью которого формируется каждый следующий элемент массива, то есть значение второго элемента составит $X_n + dX$, третьего — $X_n + dX + dX$, и так далее до X_k .

Если параметр dX в конструкции отсутствует: **имя массива** = $X_n : X_k$, это означает, что по умолчанию он принимает значение равное единице, то есть каждый следующий элемент массива равен значению предыдущего плюс один.

Примеры создания массивов:

```
>>> 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
```

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

```
>>> 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 15:
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
```

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 = -1.2000    3.4000   -0.8000    9.1000    5.6000   -7.3000
```

Элементы вектора-столбца вводятся через точку с запятой:

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

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
```

Ввод элементов матрицы так же осуществляется в квадратных скобках, при этом элементы строки отделяются друг от друга про-

белом 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:

```
>>> 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
```

Подробно работа с векторами и матрицами описана в пятой главе.

2.7 Символьные вычисления

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

Оператор **syms** инициализирует символические операции, с этого оператора должны начинаться любые действия в символьных переменных. Работа с символьными переменными в **Octave** требует их специального объявления: **sym('имя переменной')**. Например, команда **x = sym("x")** объявляет символьную переменную **x**.

Пример 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 2.2).

```
>>> x = sym ("x"); % Объявление
>>> y = sym ("y"); % символьных
>>> z = sym ("z"); % переменных
>>> t = sym ("t");
>>> a = sym ("a");
>>> b = sym ("b");
% Вычисление символьных выражений
>>> x=a+b
```


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

expand(выражение)

Пример 2.4. Раскрыть скобки в выражении $y = (\sqrt{x} + 1)(\sqrt{x} - 1) + (x - 1)^3$ (листинг 2.1).

```
>>> 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 2.4. Решение примера 2.4.

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).

Глава 3

Программирование

3.1 Основные операторы языка программирования

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 её 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 новой переменной (п. 2.4). Для того, чтобы определить новую переменную, достаточно присвоить ей значение: **имя_переменной** = **значение_выражения**

Любую переменную **Octave** воспринимает как матрицу. В простейшем случае матрица может состоять из одной строки и одного столбца:

```
>>> 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
```

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**. Введённое пользователем значение будет присвоено переменной, имя которой указано слева от знака присваивания.

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

Пример 3.1. Создать программу для вычисления значения y по формуле $y = \sin(x)$, при заданном значении x .

Текст программы и результаты её работы показаны в листинге 3.1.

```
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 3.1. Решение к примеру 3.1.

3.1.3 Условный оператор

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

Обычный условный оператор имеет вид:

```
if условие
    операторы_1
else
    операторы_2
end
```

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

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

Как показано на рис. 3.1, фигура на плоскости ограничена линиями $x = -1$, $x = 3$, $y = -2$ и $y = 4$. Значит точка с координатами $(x; y)$ будет принадлежать этой фигуре, если будут выполняться следующие условия: $x \geq -1$, $x \leq 3$, $y \geq -2$ и $y \leq 4$. Иначе точка лежит за пределами фигуры.

Далее приведён текст программы и результаты её работы.

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

Листинг 3.2. Решение к примеру 3.2.

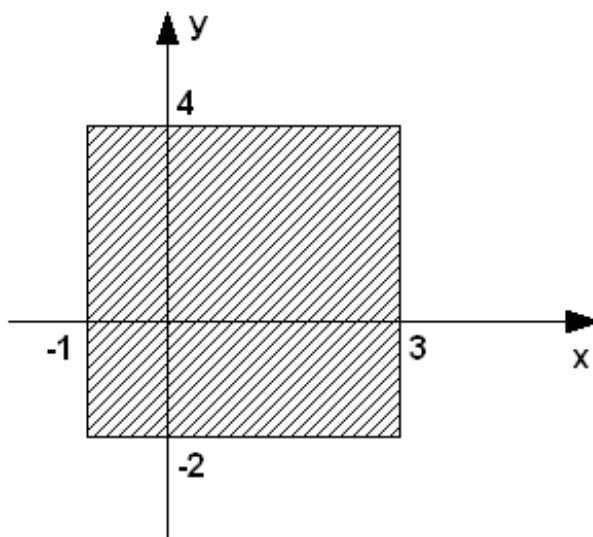


Рис. 3.1. Графическое представление задачи из примера 3.2

Сокращённый условный оператор записывают так:

```
if условие
    операторы
end
```

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

```
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
```

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

```
if условие_1
    операторы_1
elseif условие_2
    операторы_2
elseif условие_3
    операторы_3
...
elseif условие_n
    операторы_n
else
    операторы
end
```

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

Рассмотрим использование расширенного условного оператора на примере.

Пример 3.3. Дано вещественное число x . Для функции, график которой приведён на рис. 3.2 вычислить $y = f(x)$.

Аналитически функцию, представленную на рис. 3.2, можно записать так:

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

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.

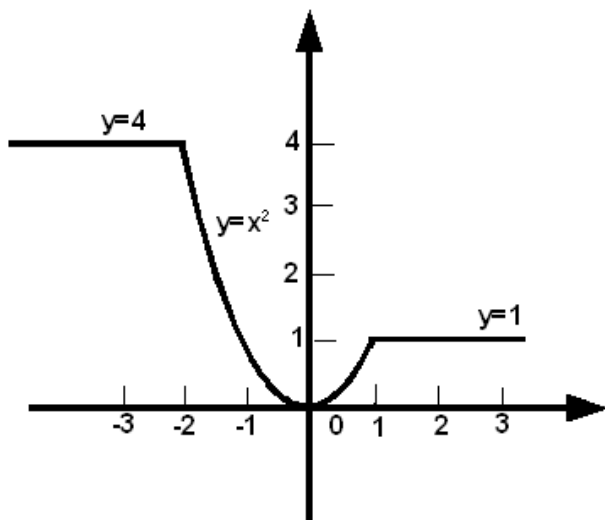


Рис. 3.2. Графическое представление задачи из примера 3.3

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 3.3. Решение к примеру 3.3.

3.1.4 Оператор альтернативного выбора

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

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

Оператор **switch** работает следующим образом: если значение параметра равно **значение1**, то выполняются **операторы1**, иначе если параметр равен **значение2**, то выполняются **операторы2**. В противном случае, если значение параметра совпадает со **значение3**, 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.

Пример 3.4. Вывести на печать название дня недели, соответствующее заданному числу D , при условии, что в месяце 31 день и первое число — понедельник.

Для решения задачи воспользуемся функцией *mod*, позволяющей вычислить остаток от деления двух чисел. Если в результате остаток от деления заданного числа D на семь будет равен единице, то это понедельник (по условию первое число — 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 альтернативного выбора (листинг 3.4).

```
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 3.4. Решение к примеру 3.4.

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

Пример 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 таблицу 3.1.

Текст программы, реализующей данную задачу, можно записать так:

Таблица 3.1. Определение количества цифр числа

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

```

N = input('N=');
M = N; % Сохранить значение переменной N.
kol=1; % Число содержит хотя бы одну цифру.
while round(M/10) > 0 % Выполнять тело цикла, пока частное от деления
    % M на 10, округлённое до целого, больше 0.
    kol = kol + 1; % Счётчик количества цифр
    M = round(M/10); % Изменение числа.
end
disp('kol='); disp(kol);
% Результат работы программы
N= 12345678
kol= 8

```

Листинг 3.5. Решение к примеру 3.5.

3.1.6 Оператор цикла с известным числом повторений

Для записи цикла с известным числом повторений применяют оператор *for*

```

for параметр = начальное_значение:шаг:конечное_значение
    операторы
end

```

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

Если шаг цикла равен 1, то оператор записывают так:

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

Пример 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 делителей $K = 5$.

Для решения поставленной задачи нужно реализовать следующий алгоритм: в переменную K , предназначенную для подсчёта количества делителей заданного числа, поместить значение, которое не влияло бы на результат, т.е. ноль. Далее организовать цикл, в котором изменяющийся параметр i выполняет роль возможных делителей числа N . Если заданное число делится нацело на параметр цикла, это означает, что i является делителем N , и значение переменной K следует увеличить на единицу. Цикл необходимо повторить только $N/2$ раз¹.

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

Листинг 3.6. Количество делителей числа (к примеру 3.6).

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 передаётся оператору, находящемуся непосредственно за циклом. Оператор **continue** начинает новую итерацию цикла, даже если предыдущая не была завершена.

¹Если делитель числа меньше самого числа, значит он не превосходит его половины (*Прим. редактора*).

Пример 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, 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 найдётся ни одного числа, делящего заданное число нацело, то N — простое число, иначе оно таковым не является. Разумно предусмотреть в программе два выхода из цикла. Первый — естественный, при исчерпании всех значений параметра, а второй — досрочный, с помощью оператора **break**. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра.

Текст программы приведён в листинге 3.7.

```
N=input('Введите число ');
pr=1; % Предполагаем, что число N является простым (pr=1).
for i=2:N/2 % Перебираем все возможные делители числа N от 2 до N/2.
    if mod(N,i)==0 % Если N делится на i,
        pr=0; % то число N не является простым (pr=0)
        break; % и прерывается выполнение цикла.
    end
end
if pr==1 % Если pr равно 1, то N — простое число.
    disp('ПРОСТОЕ ЧИСЛО')
else % Если pr равно 0, то N — не является простым.
    disp('НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ')
end
% Результаты работы программы
% Вводим сначала число 12, затем 13
Введите число 12
НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ
%
Введите число 13
ПРОСТОЕ ЧИСЛО
```

Листинг 3.7. Является ли число простым (к примеру 3.7)?

3.2 Обработка массивов и матриц

Octave содержит достаточное количество операций предназначенных для работы с векторами и матрицами (см. гл. 6). В этом па-

раграфе 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
```

Для вывода приглашений вида $x(i) =$ и $a(i,j) =$ в функции *input* использовались функции работы со строками: *strcat*(s_1, s_2, \dots, s_n) и *int2str*(d). Функция *strcat* предназначена для объединения строк s_1, s_2, \dots, s_n 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 добавляют к s остальные элементы массива.

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

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
```

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 .

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

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
```

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 .

Ниже 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;
```

Алгоритм поиска минимального элемента в массиве будет отличаться от приведённого выше лишь тем, что в конструкции **if** текста программы знак поменяется с «больше» ($>$) на «меньше» ($<$).

Ниже приведён 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;
```

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений.

Рассмотрим наиболее известный алгоритм сортировки *методом пузырька*. Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Те же действия выполним для второго и третьего, третьего и четвёртого, 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 своё 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 убыванию приведён 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;

```

Для сортировки по убыванию нужно в операторе **if** заменить знак «больше» ($>$) на «меньше» ($<$).

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 $n-1$ -й элемент. Ниже приведён фрагмент программы, реализующей этот алгоритм.

```

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

```

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

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

Листинг 3.8 содержит программу решения этой задачи с подробными комментариями.

```

N = input('N= '); % Ввод размера массива.
for i=1:N          % Цикл для ввода элементов массива.
    x(i)=input(strcat('x(',int2str(i),')= '));
end
pr = 0; % Если pr=0 —простых чисел нет, pr=1, простые числа есть.
for i = 1:N
    % Переменная L используется при проверке является ли данный элемент
    % массива x(i) простым числом, L = 1, пока не встретились делители числа,
    L=1; % L станет равным 0, если встретятся делители числа.
    for j=2:x(i)/2 % Цикл по j от 2 до x(i)/2 для проверки является ли
                    % число простым (поиск возможных делителей числа).
        if mod(x(i),j)==0 % Если x(i) делится на j, то встретился
            L=0; % делитель числа, x(i) не является простым, L = 0 и

```

```

        break; % выходим из цикла по j с помощью оператора break.
    end;
end;
% Проверяем значение переменной L,
if L==1 % если L = 1, то число x(i) — простое.
    if pr==0 % Если при этом pr = 0, то это означает, что встретилось
        % первое простое число
        Min=x(i); % записываем в переменную Min, т.е. предполагаем,
            % что x(i) и является минимальным простым
        k=1; % количество минимумов равно 1,
        pr=1; % записываем в pr 1, т.к. в массиве есть простые числа.
    else % Иначе, если pr=1, т.е. встретилось очередное
        % (не первое) простое число,
        if x(i)<Min % сравниваем x(i) с Min, если x(i)<Min,
            Min=x(i); % этот элемент записываем в переменную Min,
            k=1; % количество минимумов равно 1.
        else
            if x(i)==Min % Если очередной элемент x(i) равен Min,
                k=k+1; % то количество минимумов увеличивается.
            end;
        end;
    end;
end;
end;
if pr == 0 % Если после перебора всех элементов массива, переменная pr
    % осталась равной 0 (простых чисел нет),
    disp('Простых чисел нет!!!!') % то вывод соответствующего сообщения.
else % Если были простые числа,
    disp(Min); % то вывод min (минимальное простое число)
    disp(k); % и k (количество минимумов)
end;

```

Листинг 3.8. Поиск наименьшего простого числа (пример 3.8)

Пример 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 $i + j = n + 1$.

Листинг 3.9 содержит текст программы для решения поставленной задачи.

```

N=input('N='); % Ввод размера квадратной матрицы.
for i=1:N      % Ввод квадратной матрицы.

```

```

    for j=1:N
        A(i,j)=input(strcat('a(',int2str(i),',',int2str(j),')='
        ));
    end
end
for j=1:N % Цикл по всем столбцам матрицы, в каждом из которых ищем
    % максимальный элемент и его номер.
    Max=A(1,j); % Предполагаем, что первый элемент в столбце
    % является максимальным,
    nmax=1; % В переменную nmax, в которой будет храниться номер
    % максимального элемента j-го столбца записываем 1.
    for i=2:N % В цикле по i перебираем все элементы j-го столбца.
        if A(i,j)>Max % Если очередной элемент больше Max,
            Max=A(i,j); % то в переменную Max записываем его,
            nmax=i; % а в переменную nmax — его номер строки.
        end;
    end;
    if nmax==N+1-j % Если в текущем столбце максимальный элемент
    % находится на побочной диагонали,
        for i=1:N % то обнуляем все элементы в этом столбце.
            A(i,j)=0;
        end;
    end;
end;
end;
end;

```

Листинг 3.9. Решение к примеру 3.9.

3.3 Обработка строк

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

Таблица 3.2: Функции для работы со строками.

Функция	Описание функции	Пример использования
<code>char(code)</code>	Возвращает символ по его коду <i>code</i>	<pre>>>> char(100) ans = d >>> char(80:85) ans = PQRSTU</pre>
<code>deblank(s)</code>	Формируется новая строка путём удаления пробелов в конце строки <i>s</i>	<pre>>>> debblank('OCTAVE ') ans = OCTAVE</pre>
<code>int2str(x)</code>	Преобразование чисел, хранящихся в массиве (матрице) <i>x</i> к целому типу и запись результатов в массив символов	<pre>>>> int2str(123.456) ans = 123 >>> int2str([9.8 6.9]) ans = 10 7</pre>

Таблица 3.2 — продолжение

Функция	Описание функции	Пример использования
<i>findstr(str, substr)</i>	Возвращает номер позиции, начиная с которой подстрока <i>substr</i> входит в строку <i>str</i>	<pre>>>> Str='Visual C++'; >>> S='C++'; >>> findstr(Str,S) ans =8</pre>
<i>lower(s)</i>	Возвращает строку путём преобразования строки <i>s</i> к строчным буквам	<pre>>>> S='QtOctave'; >>> lower(S) ans = qt octave</pre>
<i>mat2str(x, n)</i>	Преобразовывает числовую матрицу <i>x</i> в строку; если присутствует необязательный параметр <i>n</i> , то перед преобразованием в строку все элементы матрицы округляются до <i>n</i> значащих цифр в числе	<pre>>>> X=[7.895; -9.325] X = 7.8950 -9.3250 >>> mat2str(X) ans = [7.89499999999999996; -9.32499999999999993] >>> mat2str(X,2) ans = [7.9;-9.3]</pre>
<i>num2str(x, n)</i>	Преобразовывает числовую матрицу (массив) <i>x</i> в массив символов, если присутствует необязательный параметр <i>n</i> , то перед преобразованием в строку все элементы матрицы округляются до <i>n</i> значащих цифр в числе.	<pre>>>>X=[7.89578; -9.32985]; >>>num2str(X) ans = 7.8958 -9.3299 >>>num2str(X,2) ans = 7.9 -9.3 >>>num2str(X,1) ans = 8 -9</pre>
<i>sprintf(format, x)</i>	Формирует строку из чисел, хранящихся в числовой переменной <i>x</i> в соответствии с форматом <i>format</i>	<pre>>>> 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</pre>
<i>sscanf(s, format)</i>	Функция возвращает из строки <i>s</i> числовое значение или массив значений в соответствии с форматом	<pre>>>> s='1234.5' s = 1234.5 >>> x=sscanf(s, '%f') x = 1234.5 >>> x=sscanf(s, '%d') x = 1234</pre>
<i>str2double(s)</i>	Формирование числа из строки <i>s</i> , если это возможно	<pre>>>> s='1.456e-2'; >>> str2double(s) ans = 0.014560</pre>

Таблица 3.2 — продолжение

Функция	Описание функции	Пример использования
<code>str2num(s)</code>	Формирование массива чисел из строки (массива символов) s	<pre>>>> s='pi 2 1.6'; >>> str2num(s) ans=-3.1416 2.0000 1.6000</pre>
<code>strcat(s1, s2, ... sn)</code>	Формируется строка путём объединения строк $s1, s2, \dots, sn$	<pre>>>> s1='Octave'; >>> s2='Qt'; >>> s=strcat(s2,s1) s = QtOctave ;</pre>
<code>strcmp(s1, s2)</code>	Возвращает 1, если строки $s1$ и $s2$ совпадают, 0 — в противном случае	<pre>>>> S1='The first ex.'; >>> strcmp(S1,'The first') ans = 0 >>> S2='The second ex.'; >>> strcmp(S1,S2) ans = 0 >>> strcmp(S1,'The first ex.') ans = 1</pre>
<code>strcmpi(s1, s2)</code>	Сравнение строк $s1$ и $s2$, не различая строчные и прописные буквы	<pre>>>> S1='1-st May'; >>> S2='1-st may'; >>> strcmpi(S2,S1) ans = 1</pre>
<code>strjust(s, direction)</code>	Выравнивание строки s в соответствии с направлением <i>direct</i> : <i>right</i> — выравнивание по правому краю, <i>left</i> — выравнивание по левому краю, <i>center</i> — выравнивание по центру	<pre>>>> S='Pascal 7.0'; >>> strjust(S,'right') 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</pre>
<code>strncmp(s1, s2, n)</code>	Сравнение первых n символов строк $s1$ и $s2$, возвращает 1, если первые n символов строк $s1$ и $s2$ совпадают, 0 — в противном случае	<pre>>>> S1='My name is Vasia'; >>> S2='My name is Petia'; >>> strncmp(S1,S2,10) ans = 1 >>> strncmp(S1,S2,12) ans = 0</pre>

Таблица 3.2 — продолжение

Функция	Описание функции	Пример использования
<i>strrep(s, subs, subsnew)</i>	Формирует новую строку из строки <i>s</i> путём замены подстроки <i>subs</i> на подстроки <i>subsnew</i>	<pre>>>> S='07. 07. 2007'; >>> strrep(S,'7','8') ans = 08. 08. 2008</pre>
<i>strtok(s, delimiter)</i>	Поиск первой подстроки в строке <i>s</i> , отделённой пробелом или символом табуляции (при отсутствии параметра <i>delimiter</i>) или первой подстроки, отделённой от <i>s</i> одним из символов, входящих в <i>delimiter</i> . Функция может возвращать 2 параметра: первый — найденная подстрока, второй — содержит остаток строки <i>s</i> после <i>strtok</i>	<pre>>>> 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!</pre>
<i>upper(s)</i>	Возвращает строку <i>s</i> , преобразованную к прописным буквам	<pre>>>> S='Octave'; >>> upper(S) ans = OCTAVE</pre>

3.4 Работа с файлами

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 все данные преобразуются к символьному типу и хранятся в символьном виде. Этот файл можно просмотреть с помощью любого текстового редактора.

В двоичных файлах информация считывается и записывается в виде блоков определённого размера. Данные в двоичных файлах могут быть любого вида и структуры и не рассчитаны для отображения в виде текста.

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.

Таблица 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
m	Для типов e , E , f выводить m знаков после десятичной точки
Тип	
c	При вводе символьный тип <code>char</code> , при выводе один байт.
d	Десятичное целое со знаком
i	Десятичное целое со знаком
o	Восьмеричное целое без знака
u	Десятичное целое без знака
x , X	Шестнадцатеричное целое без знака, при x используются символы <code>a - f</code> , при X – <code>A - F</code>
f	Значение со знаком вида <code>[-]dddd.dddd</code>
e	Значение со знаком вида <code>[-]d.dddd e[+ -]ddd</code>
E	Значение со знаком вида <code>[-]d.dddd E[+ -]ddd</code>
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, приведённые в табл. 3.4.

Таблица 3.4. Специальные символы

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

При считывании данных из файла можно воспользоваться функцией следующего вида:

$A = \text{fscanf}(f, s1, n)$

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 разделёнными, если между ними есть хотя бы один пробел, символ табуляции или символ перехода на новую строку.

При считывании данных из текстового файла пользователь может следить, достигнут ли конец файла с помощью функции $\text{feof}(f)$ (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:

$\text{fclose}(f)$

Здесь f — идентификатор закрываемого файла. С помощью функции $\text{fclose}('all')$ можно закрыть сразу все открытые файлы кроме стандартных системных файлов.

Рассмотрим использование рассмотренных выше функций на простых примерах.

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

```

Листинг 3.10. Решение к примеру 3.10.

В результате работы программы создан файл `prim_3_10.txt`, который можно открыть при помощи обычного текстового редактора.

ИСХОДНАЯ МАТРИЦА A

1.000000	2.000000	3.000000
4.000000	5.000000	6.000000
7.000000	8.000000	9.000000

МАТРИЦА A после преобразования

3.000000	2.000000	1.000000
----------	----------	----------

```

4.000000    5.000000    6.000000
9.000000    8.000000    7.000000
Файл prim_3_10.txt

```

Обратите внимание, что после записи информации в файл этот файл 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, хранящаяся в буфере файла.

Пример 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 3.11 содержит программу для создания подобного файла.

```

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

```

Листинг 3.11. Запись матрицы в файл (пример 3.11).

После выполнения этой программы будет создан текстовый файл `prim_3_11.txt`:

```

5    3
1    2    3
4    5    6

```

```

7      8      9
0      1      2
3      4      5

```

Файл `prim_3_11.txt`

Пример 3.12. Считать информацию из файла `prim_3_11.txt` в матрицу.

Рассмотрим поэлементное (листинг 3.12) и построчное (листинг 3.13) 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.
N=fscanf(f,'%d',1); % Считываем количество строк в переменную N
M=fscanf(f,'%d',1); % Считываем количество столбцов в переменную M
for i=1:N % Двойной цикл по строкам и столбцам.
    for j=1:M
        A(i,j)=fscanf(f,'%g',1);% Считываем в матрицу один элемент
    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 3.12. Поэлементное чтение матрицы из файла к примеру 3.12

```

f=fopen('primer.txt','rt'); % Открываем файл для чтения.
N=fscanf(f,'%d',1); % Считываем количество строк в переменную N
M=fscanf(f,'%d',1); % Считываем количество столбцов в переменную M
for i=1:N % Открываем цикл по строкам.
    A(i,:)=fscanf(f,'%g',M); % Считываем в i-ю строку из M элементов
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 3.13. Построчное чтение матрицы из файла к примеру 3.12

Пример 3.13. Считать в массив вещественные значения из текстового файла `one.txt`:

```

1.22 3.45 5.6 7.8
9.1 8.2 9.3 7.41 10
Файл one.txt

```

Возможно считывание из файла всего массива целиком (листинг 3.14) или поэлементное считывание данных из файла (листинг 3.15).

```

f=fopen('one.txt');% Открываем файл для чтения.
x=fscanf(f,'%f'); % Считываем содержимое файла целиком в массив.
% 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 3.14. Считывание данных в массив целиком. Пример 3.13

```

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

```

Листинг 3.15. Поэлементное считывание данных. Пример 3.13

Обратите внимание, если данные в файле располагаются в несколько строк, то программы, аналогичные приведённым в листингах 3.12 и 3.13, считывают их как матрицу значений, а программы, приведённые в листингах 3.14 и 3.15 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 **dmlread** предназначена для чтения числовых данных из текстового файла в матрицу. Существуют четыре варианта использования функции.

1. $M = \text{dmlread}('filename')$ — чтение чисел из текстового файла *filename* в матрицу M , числа внутри строки отделяются запятой. В листинге ниже представлен результат чтения данных из файла **one.txt** (см. пример 3.13).

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

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

2. $M = \text{dmlread}('filename', delimiter)$ — чтение чисел из текстового файла *filename* в матрицу M , числа отделяются символом, хранящимся в *delimiter*. Например, $M = \text{dmlread}('ab.txt', '\t')$ — чтение из файла **ab.txt** чисел в матрицу M , внутри строки числа отделяются табуляцией. В качестве примера рассмотрим файл **two.txt**, в котором числа внутри строки разделены двоеточием:

4.3 : 45.78 : 12.90

23.54 : 0.113 : 78

Файл **two.txt**

Ниже представлен результат чтения данных из файла **two.txt**.

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

3. $M = \text{dmlread}('filename', delimiter, R, C)$ — 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. В листинге

представлено чтение матрицы из файла `two.txt` начиная со второй строки и третьего столбца:

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

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 представлено чтение матрицы из файла `one.txt`:

```
>>> 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
```

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');**

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
```

3.4.2 Обработка двоичных файлов

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

$[A, n] = \mathbf{fread}(f, n, type);$, где *f* — идентификатор файла, *n* — количество считываемых из файла элементов, *type* — тип считываемых из файла элементов.

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

Таблица 3.5. Возможные значения параметра *type*

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

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 *n*. 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 $[mk]$, в этом случае данные считываются в матрицу размером $m \times k$.

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 вперёд ($n > 0$) или назад ($n < 0$).

Запись в двоичный файл осуществляется с помощью функции $n = \mathbf{fwrite}(f, A, type)$, где f — идентификатор файла, A — массив (матрица) значений, *type* — тип записываемых в файл элементов.

Функция *fwrite* записывает в заранее открытый файл с идентификатором f массив A , и возвращает количество реально записанных в файл значений n .

Рассмотрим несколько примеров работы с двоичными файлами.

Пример 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 обоими способами представлено в листинге 3.16.

```
% Первый способ
N=input('N='); % Ввод значения переменной N.
f=fopen('abc.dat','wb'); % Открытие двоичного файла abc.dat
                        % в режиме записи2.
fwrite(f,N,'int16'); % Запись числа N в двоичный файл abc.dat.
for i=1:N % Цикл для ввода N вещественных чисел и записи их в файл
    x=input('X='); % Ввод очередного вещественного числа x.
    fwrite(f,x,'float32'); % Запись очередного числа x в файл.
end;
fclose(f); % Закрытие файла.
% Второй способ
N=input('N='); % Ввод значения переменной N.
f=fopen('abc.dat','wb'); % Открытие двоичного файла для записи.
fwrite(f,N,'int16'); % Запись числа N в двоичный файл abc.dat.
for i=1:N % Цикл для ввода массива из N вещественных чисел.
    x(i)=input(strcat('x(',int2str(i),')=')); % Ввод очередного
                                                % вещественного числа в массив x.
end;
fwrite(f,x,'float32'); % Запись массива x в двоичный файл abc.dat.
fclose(f); % Закрытие файла.
```

Листинг 3.16. Запись числа в двоичный файл (пример 3.14).

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

Пример 3.15. Считать данные из файла `abc.dat`, сформированного в задаче из примера 3.14 в массив вещественных чисел.

Программа решения этой задачи представлена в листинге 3.17.

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

Листинг 3.17. Чтение данных из файла в массив (пример 3.15).

²Если файл `abc.dat` не существовал, он создастся, если существовал — все его содержимое будет утеряно. (Прим. редактора)

Пример 3.16. Считать данные из файла `abc.dat`, сформированного в задаче из примера 3.14 в матрицу вещественных чисел.

Зная, что в файле `abc.dat` хранится 20 чисел, в качестве примера запишем их в матрицу размером 4×5 . Программа решения этой задачи представлена в листинге 3.18. В результате работы этой программы будет сформирована матрица вещественных чисел $G(4, 5)$.

```
f=fopen('abc.dat','rb'); % Открытие файла abc.dat в режиме чтения.
N=fread(f,1,'int16'); % Чтение числа N из двоичного файла abc.dat.
G=fread(f,[4 5],'float32'); % Чтение матрицы вещественных чисел
                               % G(4,5) из двоичного файла abc.dat.
fclose(f); % Закрытие файла.
>>> 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 3.18. Чтение данных из файла в матрицу (пример 3.15).

Отдельную задачу представляет чтение данных из двоичного файла, если заранее не известно количество элементов в файле. В листинге 3.19 представлена программа, с помощью которой можно создать файл вещественных чисел.

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

Листинг 3.19. Создание двоичного файла с вещественными числами

Отличие этой программы от представленных в листингах 3.17 и 3.18 состоит в том, что количество записанных в файл `abc2.dat` вещественных чисел в нём не хранится. Поэтому чтение данных из такого файла осуществляется несколько иначе. 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 в листинге 3.20.

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

Листинг 3.20. Чтение из двоичного файла неизвестного размера

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 3.20 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)` попытается считать *N* элементов из двоичного файла, но не считает значений больше, чем их там есть, и остановится в конце файла.

3.5 Функции

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[y_1, y_2, \dots, y_n] = *name_function*(x_1, x_2, \dots, x_m)

Здесь *name_function* — имя функции, x_1, x_2, \dots, x_m — список входных параметров функции, y_1, y_2, \dots, y_n — 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 подфункции
        операторы
    end % конец первой подфункции
    function [y1,y2,...,yn]=f2(x1,x2,...,xm)% начало второй подфункции
        операторы
    end % конец второй подфункции
    ...
    function [y1,y2,...,yn]=fn(x1,x2,...,xm) % начало n-й подфункции
        операторы
    end % конец n-й подфункции
    оператор1;
    оператор2;
    ...
    операторk;
end % здесь заканчивается основная функция

```

Рассмотрим пример.

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

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

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

Заголовок основной функции имеет вид:

function[xN] = *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 [xN] = *udal*(x,m,N)

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

В листинге 3.21 приведено содержимое файла *udal_prostoe.m* с комментариями.

```

% Функция udal_prostoe удаляет из массива  $x(N)$  простые числа и возвращает
% модифицированный массив  $x$  и изменённое значение  $N$  в качестве результата.
function [x N]=udal_prostoe(x, N)
    i=1;
    while i<=N
        L=prostoe(x(i)); % Проверяем является ли число  $x(i)$  простым.
        if L==1 % если число простое ( $L=1$ ),
            [x N]=udal(x, i, N); % то удаляем из массива  $i$ -ый элемент,
        else % иначе переходим к следующему элементу массива.
            i=i+1;
        end;
    end;
end % Окончание основной функции udal_prostoe.
% Функция prostoe проверяет является ли число  $P$  простым,
% она возвращает 1, если  $P$  — простое, 0 — если число  $P$  не является простым.
function pr=prostoe(P)
pr=1
for i=2:P/2
    if 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 путём смещения элементов, начиная с  $m$ -го на одну
% позицию влево. Выходными элементами функции будут массив  $x$ , из которого
% удалён один элемент и уменьшенное на 1 количество ( $N$ ) элементов в массиве.
for i=m:N-1
    x(i)=x(i+1);
end
x(:,N)=[ ]; % После смещения элементов удаляем последний элемент и
N=N-1; % уменьшаем количество элементов в массиве на 1.
end % Окончание функции udal.

```

Листинг 3.21. Файл *udal_prostoe.m*

В листинге 3.21 был использован синтаксис не допускающий вложенность функций друг в друга. Листинг 3.22 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;
            end
        end
    end
end

```

```

                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
end

```

Листинг 3.22. Решение примера 3.17 с использованием вложенных функций.

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

```

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

```

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

В **Octave** есть возможность передавать имя функции как входной параметр, что существенно расширяет возможности программирования. Вообще говоря, имя функции передаётся как строка, а её 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 определённая пользователем функция; параметры этой функции, разделённые 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.

Пример 3.18. Вычислить значение функций $\sin(x)$ и $\cos(x)$ в точке $x = \frac{\pi}{12}$.

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

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

Листинг 3.23. Вычисление значений функции с помощью *feval*

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, то есть 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.

Пример 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 3.24 приведена программа решения задачи с комментариями.

```

% Функция вычисления суммы входных параметров, хранящихся в массиве.
% 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 )
% числами.
pr=1; % pr = 1, если все элементы массива ячеек являются числами.
Sum=0; % До начала суммирования в переменную Sum запишем 0.
% length(varargin) возвращает количество входных параметров sum_var
for i=1:length( varargin ) % цикл по i для перебора всех входных
    % параметров от 1 до length(varargin).
    if isnumeric( varargin{i} )==1 % проверяем, является ли
        % i-ый входной параметр числом,
        Sum=Sum+varargin{i}; % да — добавляем varargin{i} к Sum.
    else % если не является,
        pr=0; % записываем в pr = 0,
        Sum=0; % обнуляем сумму Sum
        break; % и прерываем цикл.
    end;
end
if pr==1 % Если pr=1, то все входные параметры были числами
    % и сумма их найдена
    s=[]; % очищаем переменную s, аварийного сообщения нет.
else % иначе записываем в s аварийное сообщение.
    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 3.24. Нахождение суммы входных параметров функции

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

Пример 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 Фибоначчи (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...).

Текст функции (листинг 3.25):

```
function F=fibonacci(N)
    if (N==0) | (N==1)
        F=N;
    else
        F=fibonacci(N-1)+fibonacci(N-2);
    end
end
% Вызов функции
>>> fibonacci(2)
ans = 1
>>> fibonacci(0)
ans = 0
>>> fibonacci(6)
ans = 8
```

Листинг 3.25. Вычисление n -го числа последовательности Фибоначчи

Глава 4

Построение графиков

В этой главе читатель научится строить графики с помощью **Octave**. Первый раздел посвящён работе с двумерными графиками. Во втором разделе рассмотрено создание различных трёхмерных графиков. Затем описаны анимационные возможности **GNU Octave**. Последний параграф посвящён описанию графических объектов.

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 задаётся двумя перпендикулярными прямыми, называемыми осями координат. Горизонтальная прямая X — ось абсцисс, а вертикальная Y — ось ординат. Точку пересечения осей называют началом координат. Четыре угла, образованные осями координат, носят название координатных углов. Положение точки в прямоугольной системе координат определяется значением двух величин, называемых координатами точки. Если точка имеет координаты x и y , то x — абсцисса точки, y — ордината. Уравнение, связывающее координаты x и y является (называется)

уравнением линии, если координаты любой точки этой линии удовлетворяют ему.

Величина y называется *функцией* переменной величины x , если каждому из тех значений, которые может принимать x , соответствует одно или несколько определённых значений y . При этом переменная величина x называется аргументом функции $y = f(x)$. Говорят также, что величина y зависит от величины x . Функция считается заданной, если для каждого значения аргумента существует соответствующее значение функции. Чаще всего используют следующие способы задания функций:

- табличный — числовые значения функции уже заданы и занесены в таблицу; недостаток заключается в том, что таблица может не содержать все нужные значения функции;
- графический — значения функции заданы при помощи линии (графика), у которой абсциссы изображают значения аргумента, а ординаты — соответствующие значения функции;
- аналитический — функция задаётся одной или несколькими формулами (уравнениями); при этом, если зависимость между x и y выражена уравнением, разрешённым относительно y , то говорят о явно заданной функции, в противном случае функция считается неявной.

Совокупность всех значений, которые может принимать в условиях поставленной задачи аргумент x функции $y = f(x)$, называется областью определения этой функции. Совокупность значений y , которые принимает функция $f(x)$, называется множеством значений функции.

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

Пример 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 в листинге 4.1.

```
x=-10:0.1:10; % Формирование массива x.  
y=sin(x)+sin(3*x)/3+sin(5*x)/5; % Формирование массива y.  
plot(x,y) % Построение графика функции.
```

Листинг 4.1. Построение графика (пример 4.1).

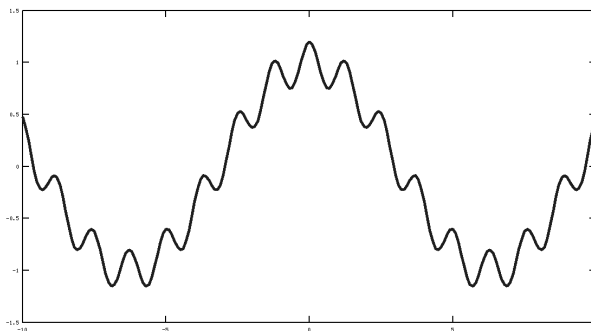


Рис. 4.1. График функции $y = \sin(x) + \frac{1}{3} \sin(3x) + \frac{1}{5} \sin(5x)$

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

График формируется путём 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 стёрт 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 $plot(x1, y1, x2, y2, \dots, xn, yn)$, где $x1, y1$ — массивы абсцисс и ординат первого графика, $x2, y2$ — массивы абсцисс и ординат второго графика, \dots, xn, yn — массивы абсцисс и ординат n -ого графика.
2. Каждый график изображать с помощью функции $plot(x, y)$, но перед обращением к функциям $plot(x2, y2)$, $plot(x3, y3)$, \dots , $plot(xn, yn)$ вызвать команду $hold\ on$ ¹, которая блокирует режим очистки окна.

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

¹Команда $hold$ работает в режиме переключателя: $hold\ on$ — блокирует режим очистки экрана, $hold\ off$ — включает режим очистки экрана.

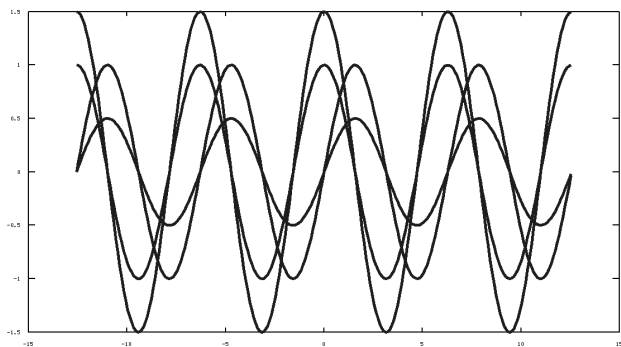


Рис. 4.2. Графики функций $v = \sin x$, $w = \cos x$, $r = \sin \frac{x}{2}$, $p = \frac{3}{2} \cos x$

Пример 4.2. Построить графики функций $v = \sin x$, $w = \cos x$, $r = \sin \frac{x}{2}$, $p = \frac{3}{2} \cos x$ на интервале $[-4\pi; 4\pi]$.

Построение графиков функций первым и вторым способом представлено в листинге 4.2. Получившиеся графики функций представлены на рис. 4.2.

```
% Способ первый
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);
% Способ второй
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 4.2. Два способа построения графика (пример 4.2).

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 $x1, x2, \dots, xn$ — массивы абсцисс графиков; $y1, y2, \dots, yn$ — массивы ординат графиков;

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	жёлтый
m	розовый
c	голубой
r	красный
g	зелёный
b	синий
w	белый

s_1, s_2, \dots, s_n — 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 (см. табл. 4.1).

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

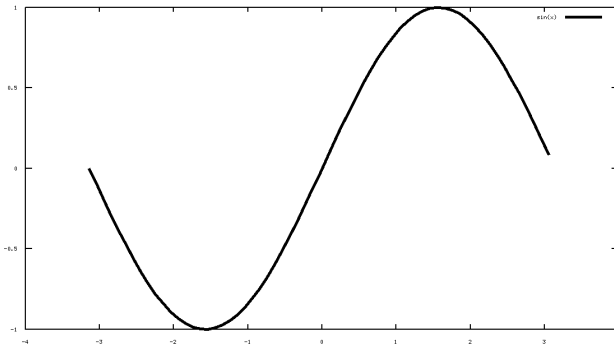


Рис. 4.3. Результат работы функции $\text{plot}(x = -\pi : 0.1 : \pi, \sin(x), " - k; \sin(x); ")$

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 ещё будут помечаться маркером, если символа маркера нет, то соседние точки просто будут соединяться линиями;
- если символ маркера «-» 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 — точечная.

Если необходима легенда для графика, то её следует включить в строку форматов, заключённую в символы «;». Например, команда $\text{plot}(x = -\pi : 0.1 : \pi, \sin(x), " - k; \sin(x); ")$ выведет на экран график функции $y = \sin(x)$ чёрного цвета на интервале $[-\pi; \pi]$ с легендой « $\sin(x)$ » (см. рис. 4.3)

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 $\text{plot}(x = -\pi : 0.1 : \pi, \sin(x), " - ok; \sin(x); ", "markersize", 4);$ выведет на экран график, представленный на рис. 4.4.

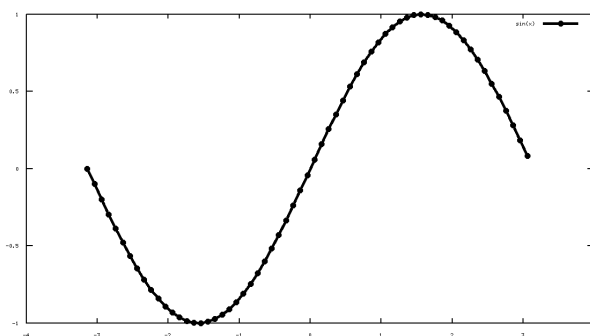


Рис. 4.4. Результаты работы функции `plot(x = -pi : 0.1 : pi, sin(x), "ok; sin(x);", "markersize", 4);`

Для того, 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 уничтожению (закрытию) окна, а к его скрытию. При повторном вызове команды рисования графиков происходит восстановление окна, в котором и изображаются графики. Корректное закрытие графического окна можно осуществить в **Octave** только программно.

Графическое окно создаётся функцией `figure: h = figure();`

Здесь h — переменная, в которой будет храниться дескриптор (номер) окна. Для дальнейших операций с окном надо будет использовать именно переменную, в которой хранится дескриптор.

Уничтожение (закрытие) окна осуществляется с помощью функции `delete(h)`, где h — имя дескриптора закрываемого окна.

В **Octave** есть функция `pause(n)`, которая приостанавливает выполнение программы на n секунд. Её логично вставлять перед функцией закрытия окна.

Octave представляет дополнительные возможности для оформления графиков:

- команда `grid on` наносит сетку на график, `grid off` убирает сетку с графика;

- функция `axis[xmin, xmax, ymin, ymax]` выводит только часть графика, определяемую прямоугольной областью $x_{min} \leq x \leq x_{max}$, $y_{min} \leq y \leq y_{max}$;
- 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', ..., 'легендап', 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 буквы² (см. табл. 4.3), использовать символы верхнего и нижнего индекса. Для вывода текста в верхнем индексе используется символ « \wedge », в нижнем — символ « $_$ ». Например, для вывода $e^{\cos(x)}$ необходимо будет ввести текст `e^{cos(x)}`, а для вывода x_{min} — текст `x_{min}`. При работе с текстом можно также использовать синтаксис \TeX .

Таблица 4.3: Греческие буквы и специальные символы

Команда	Символ	Команда	Символ
<code>\alpha</code>	α	<code>\upsilon</code>	υ
<code>\beta</code>	β	<code>\phi</code>	ϕ
<code>\gamma</code>	γ	<code>\chi</code>	χ
<code>\delta</code>	δ	<code>\psi</code>	ψ
<code>\epsilon</code>	ϵ	<code>\omega</code>	ω
<code>\zeta</code>	ζ	<code>\Gamma</code>	Γ
<code>\eta</code>	η	<code>\Delta</code>	Δ
<code>\theta</code>	θ	<code>\Theta</code>	Θ
<code>\iota</code>	ι	<code>\Lambda</code>	Λ
<code>\kappa</code>	κ	<code>\Xi</code>	Ξ

²Для вывода греческих букв и кириллицы в вашей операционной системе должны быть установлены соответствующие шрифты.

Таблица 4.3 — продолжение

Команда	Символ	Команда	Символ
<code>\lambda</code>	λ	<code>\Pi</code>	Π
<code>\mu</code>	μ	<code>\Sigma</code>	Σ
<code>\nu</code>	ν	<code>\Upsilon</code>	Υ
<code>\xi</code>	ξ	<code>\Phi</code>	Φ
<code>\pi</code>	π	<code>\Psi</code>	Ψ
<code>\rho</code>	ρ	<code>\Omega</code>	Ω
<code>\sigma</code>	σ	<code>\forall</code>	\forall
<code>\varsigma</code>	ς	<code>\exists</code>	\exists
<code>\tau</code>	τ	<code>\approx</code>	\approx
<code>\int</code>	\int	<code>\in</code>	\in
<code>\wedge</code>	\wedge	<code>\sim</code>	\sim
<code>\vee</code>	\vee	<code>\leq</code>	\leq
<code>\pm</code>	\pm	<code>\leftrightarrow</code>	\leftrightarrow
<code>\geq</code>	\geq	<code>\leftarrow</code>	\leftarrow
<code>\infty</code>	∞	<code>\uparrow</code>	\uparrow
<code>\partial</code>	∂	<code>\rightarrow</code>	\rightarrow
<code>\neq</code>	\neq	<code>\downarrow</code>	\downarrow
<code>\nabla</code>	∇	<code>\circ</code>	\circ

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

Пример 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 решения задачи с комментариями приведён в листинге 4.3.

```

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

```

```

plot(x,y,'r'); % Выводим график функции cos(x) красного цвета.
grid on; % Выводим линии сетки.
title('Plot y=cos(x)'); % Выводим заголовок графика.
pause(5); % Приостанавливаем выполнение программы на 5 секунд.
y=cos(x/3);
plot(x,y,'g'); % Выводим график функции cos(x/3) зелёного цвета.
grid on; % Выводим линии сетки.
title('Plot y=cos(x/3)'); % Выводим заголовок графика.
pause(5); % Приостанавливаем выполнение программы на 5 секунд.
delete(okno1); % Закрываем окно с дескриптором okno1.

```

Листинг 4.3. Построение графиков (пример 4.3).

При запуске программы будет создано графическое окно, в котором будет выведен график функции $\sin(x)$ чёрного цвета с линиями сетки и заголовком. Это окно будет находиться на экране в течение 5 секунд, после чего очистится. 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 $\cos(x)$ и $\cos(x/3)$. 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 и посмотреть, что из этого получится.

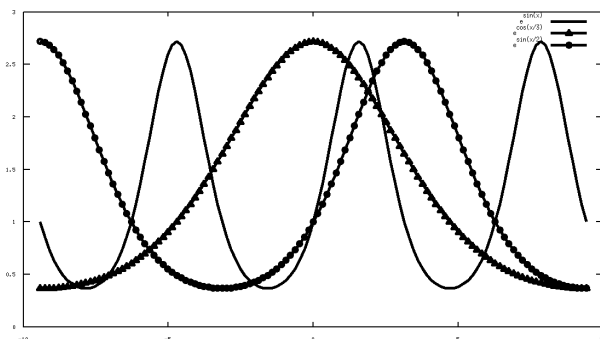
Пример 4.4. Построить графики функций $y = e^{\sin x}$, $u = e^{\cos \frac{x}{3}}$, $v = e^{\sin \frac{x}{2}}$ на интервале $[-3\pi; 3\pi]$.

Рассмотрим два варианта построения графиков (см. листинг 4.4).

```

% -----Способ первый-----
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,
% в качестве легенды выводим e^{sin(x)}.
plot(x, y, "k;e^{sin(x)};")
hold on; % Блокируем режим очистки окна.
% Строим график функции u(x), сплошная чёрная линия, с маркером
% треугольником, 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
% окружностью, размер маркера — 4, в качестве легенды выводим e^{sin(x/2)}.
plot(x, v, "-o;k;e^{sin(x/2)};", "markersize", 4);
% -----Способ второй-----
x=-3*pi():pi()/20:3*pi();

```


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

```

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 в листинге 4.4
plot(x,y,"k;e^{\sin(x)}";x,u,"->k;e^{\cos(x/3)}";x,v,"-o;k;e^{\sin(x/2)}";"markersize",4)

```

Листинг 4.4. Построение графиков (пример 4.4).

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

Пример 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 полуинтервале $[-2; 0)$ и $y(x)$ на полуинтервале $(0; 2]$. Текст программы на **Octave** с комментариями приведён в листинге 4.5, график функции — на рис. 4.6.

```

a=1;b=-0.4;c=0.05;
x1=-2:0.01:-0.1; % Определяем аргумент (массив x) на [-2;-0.1].
x2=0.1:0.01:2; % Определяем аргумент (массив x) на [0.1;2].
y1=a+b./x1+c./x1.^2; % Вычисляем значение функции y(x) на [-2;-0.1].
y2=a+b./x2+c./x2.^2; % Вычисляем значение функции y(x) на [0.1;2].
% Строим график как график двух функций, на интервалах [-2;-0.1], [0.1;2],
% цвет графика чёрный, легенда — .
plot(x1,y1,'k;f(x)=a+b/x+c/x^2';x2,y2,'k');
title('y=f(x)'); % Подпись над графиком.

```

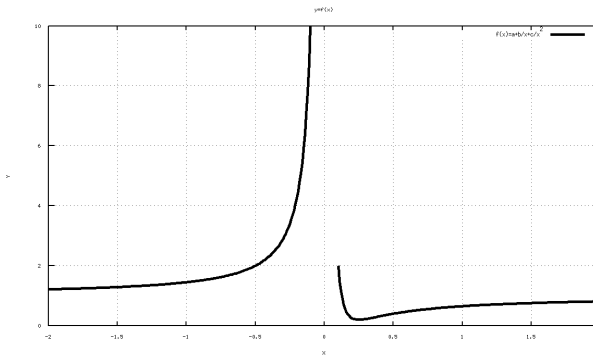


Рис. 4.6. График функции $y(x) = 1 - \frac{0.4}{x} + \frac{0.05}{x^2}$

```
xlabel('X'); % Подпись оси X.
ylabel('Y'); % Подпись оси Y.
grid on; % Рисуем линии сетки.
```

Листинг 4.5. Построение графика (пример 4.5).

Пример 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 4.6 демонстрирует решение примера 4.6, а на рис. 4.7 изображён график функции $y(x) = \frac{1}{x^2 - 2x - 3}$ как результат работы этой программы.

```
% 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;
% 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 чёрного 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.
```

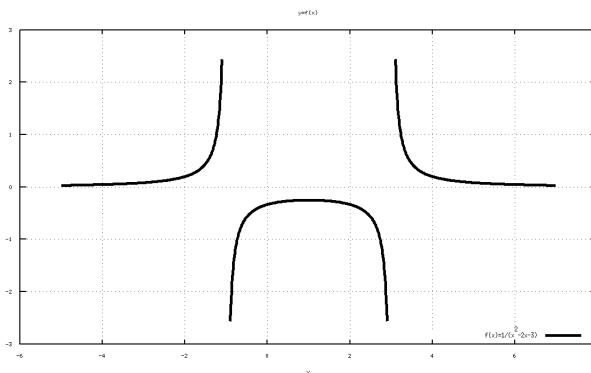


Рис. 4.7. График функции $y(x) = \frac{1}{x^2 - 2x - 3}$

```
ylabel('Y'); % Подпись оси Y.
legend('f(x)=1/(x^2-2x-3)',4); % Вывод легенды.
grid on; % Рисуем линии сетки.
```

Листинг 4.6. Построение графика (пример 4.6).

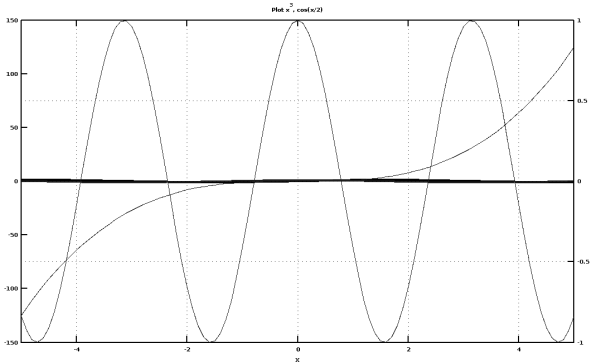
Ещё одну интересную возможность построения двух графиков предоставляет функция *plotyy*, которая позволяет изображать на графике две оси ординат, что очень удобно при построении графиков разных порядков. К сожалению эта функция не позволяет явно задавать типы изображаемых линий. На каждой из осей ординат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

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

Решение этой задачи представлено в листинге 4.7, 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 4.7.

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

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

Пример 4.8. Построить графики функции $y = 2e^{-0.15x^2}$, $z = e^{0.7x-0.25x^2}$, $u = 0.5e^{-0.33x} \sin(2x + \frac{\pi}{3})$, $k = 3 \sin(x - 0.22x^2)$, $v = \cos x$, $w = e^{\cos x}$ на интервале $[-6; 6]$.

Решение задачи представлено в листинге 4.8, 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 графическое окно на 6 частей и объявляем первый график текущим.
subplot(3, 2, 1);
plot(x, y); % Строим график y(x) с линиями сетки и подписями.
grid on; title('Plot y=2e^{-0.15x^2}'); xlabel('x'); ylabel('y');
subplot(3, 2, 2); % Второй график объявляем текущим.
plot(x, z); % Строим график z(x) с линиями сетки и подписями.
```

```

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

```

Листинг 4.8. Построение графиков (пример 4.8).

Для построения графика функции можно использовать функцию *fplot* следующей структуры: *fplot*(@*f*, [*xmin*, *xmax*], *s*).

Здесь *f* — имя функции (стандартной функции **Octave** или функции, определённой 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 отличие от функции *plot*.

Пример 4.9. Используя функцию *fplot*, построить графики функций $e^{\sin x}$, $e^{\cos x}$, $\sin x$, $\cos x$ на интервале $[-3\pi; 2\pi]$.

Текст программы на языке **Octave** с комментариями приведён в листинге 4.9. 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(sin(x))
function z=g(x) z=exp(sin(x)); end
h=figure();
% Делим графическое окно на 4 части и объявляем первый график текущим.
subplot(2,2,1);
% Строим график g = exp(cos(x)) на интервале [-3π; 2π]
% голубого цвета с маркером .
fplot (@g, [-3*pi(), 2*pi()], '-pb');
title('Plot y=e^{-cos(x)}'); % Подписываем график.
grid on; % Проводим линии сетки
subplot(2,2,2); % Второй график объявляем текущим.
% Строим график f=exp(sin(x)) на интервале [-3π; 2π]

```

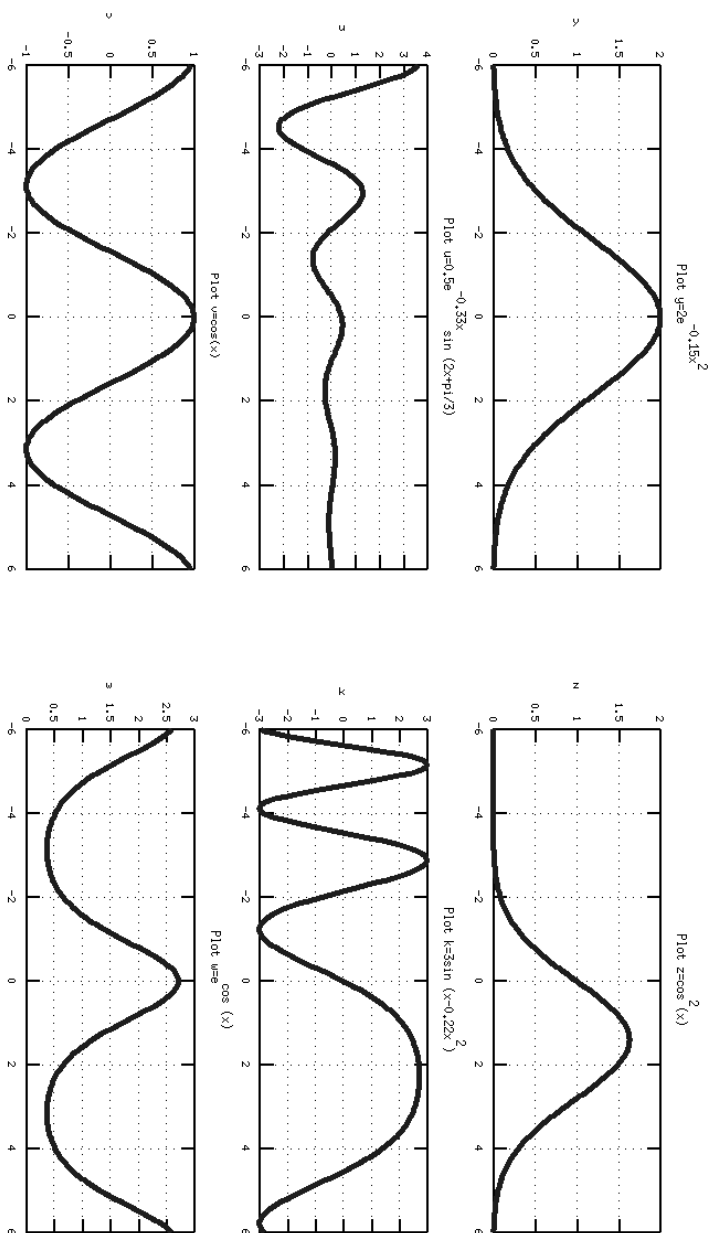


Рис. 4.9. Графики функций $y(x)$, $z(x)$, $u(x)$, $k(x)$, $v(x)$, $w(x)$

```
% красного цвета с маркером .
fplot (@f, [-3*pi(), 2*pi()], 'or');
title ('Plot z=e^{sin(x)}'); % Подписываем график.
grid on; % Проводим линии сетки
subplot (2,2,3); % Третий график объявляем текущим.
% Строим график sin(x) на интервале [-3π; 2π] чёрного цвета
fplot (@sin, [-3*pi(), 2*pi()], 'k');
title ('Plot sin(x)'); % Подписываем график.
grid on; % Проводим линии сетки
subplot (2,2,4); % Четвёртый график объявляем текущим.
% Строим график cos(x) на интервале [-3π; 2π] зелёного цвета
fplot (@cos, [-3*pi(), 2*pi()], 'g');
title ('Plot cos(x)'); % Подписываем график.
grid on; % Проводим линии сетки
```

Листинг 4.9. Построение графиков с помощью *fplot* (пример 4.9).

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 $\rho = |OM|$ (полярный радиус) и числом φ , равным величине угла $\angle XOM$ (полярный угол). Числа ρ и φ называют полярными координатами точки M и обозначают $M(\rho, \varphi)$.

Для формирования графика в полярной системе координат необходимо сформировать массивы значений полярного угла и полярного радиуса и обратиться к функции *polar*: *polar(phi, ro, s)*, где *phi* — массив полярных углов; *ro* — массив полярных радиусов; *s* — строка, состоящая из трёх символов, которые определяют цвет линии, тип маркера и тип линии (см. табл. 4.1, 4.2).

В качестве примера использования функции *polar* рассмотрим решение следующей задачи.

Пример 4.10. Построить график лемнискаты.

Уравнение лемнискаты в полярных координатах имеет вид: $\rho = a\sqrt{2 \cos 2\varphi}$, функция ρ определена при $-\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2}$ ($\cos 2\varphi \geq 0$). Поэтому листинг для изображения графика лемнискаты будет таким.

```
fi=-pi/4:pi/200:pi/4; % Определяем массив полярного угла fi.
% Определяем массив положительных значений полярного радиуса ro.
```

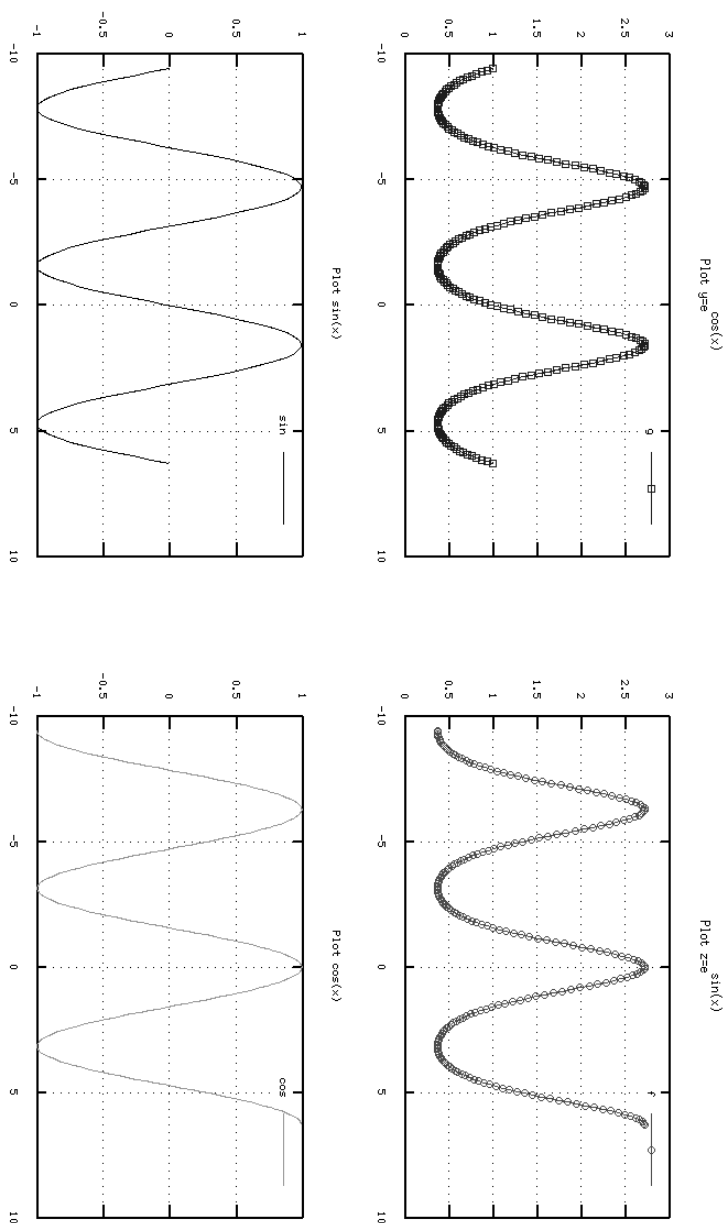


Рис. 4.10. Графики функций $e^{\sin x}$, $e^{\cos x}$, $\sin x$, $\cos x$

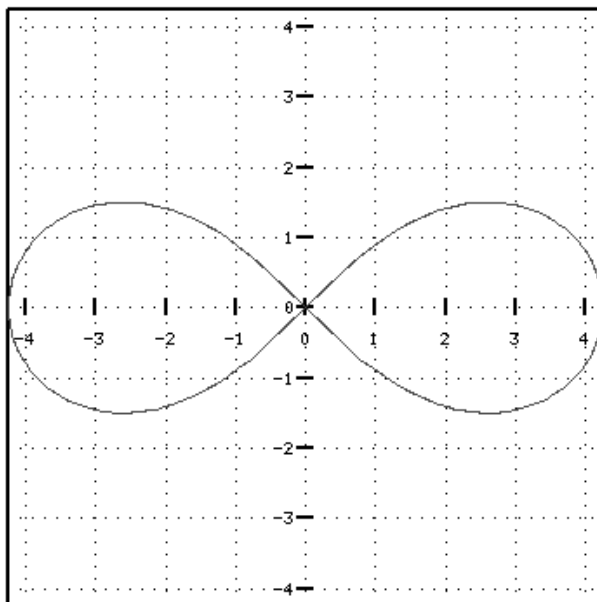


Рис. 4.11. График лемнискаты в полярных координатах

```

ro=3*sqrt(2*cos(2*fi));
polar(fi,ro,'r'); % Рисуем правую часть лемнискаты.
hold on; % Блокируем режим очистки окна.
polar(fi,-ro,'r'); % Рисуем левую часть лемнискаты.
grid on;

```

Листинг 4.10. Построение графика лемнискаты (пример 4.10).

Полученный график изображён на рис. 4.11.

Пример 4.11. Построить графики архимедовой спирали, гиперболической спирали и логарифмической спирали в полярных координатах.

Уравнение архимедовой спирали в полярных координатах имеет вид: $\rho = \alpha\varphi$, гиперболической — $\rho = \frac{\alpha}{\varphi}$. Соотношение $\rho = \alpha e^{k\varphi}$, $k = \operatorname{ctg} \alpha$ является уравнением логарифмической спирали в полярных координатах. Частным случаем логарифмической спирали ($\alpha = \frac{\pi}{2}$, $k = 0$) является уравнение окружности ($\rho = \alpha$).

В листинге 4.11 приведён текст программы, позволяющей построить в одном графическом окне четыре оси координат, в каждом из которых построить свой график — архимедову, гиперболическую и логарифмическую спирали, а также окружность.

График представлен на рис. 4.12.

```
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; ro2=0.5./fi2; 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);
polar(fi1,ro1); % Строим график архимедовой спирали.
title('Graph of Archimedean spiral'); % Подписываем график.
subplot(2,2,2); % Второй график объявляем текущим.
polar(fi2,ro2); % Строим график гиперболической спирали.
title('Graph of the hyperbolic spiral'); % Подписываем график.
subplot(2,2,3); % Третий график объявляем текущим.
polar(fi3,ro3); % Строим график логарифмической спирали.
title('Graph of the logarithmic spiral'); % Подписываем график.
subplot(2,2,4); % Четвёртый график объявляем текущим.
polar(fi4,ro4); % Строим график окружности.
title('Graph the circle'); % Подписываем график.
```

Листинг 4.11. Построение графиков спиралей (пример 4.11).

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 график эписциклоиды и астроида.

Пример 4.12. Построить график эписциклоиды. Уравнение эписциклоиды в параметрической форме имеет вид $x = 4 \cos t - \cos 4t$, $y =$

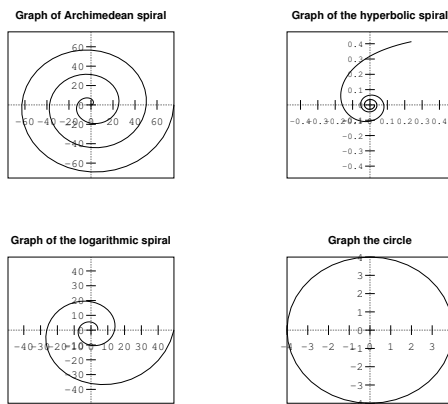


Рис. 4.12. Графики архимедовой, гиперболической и логарифмической спиралей, окружности в полярных координатах

$4 \sin t - \sin 4t, t \in [0; 2\pi]$. В листинге 4.12 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 4.12. Построение графика эпициклоиды (пример 4.12).

Пример 4.13. Построить график астроиды.

Уравнение астроиды в параметрической форме имеет вид $x = 3 \cos^3 t, y = 3 \sin^3 t, t \in [0; 2\pi]$. В листинге 4.13 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 4.13. Построение графика астроиды (пример 4.13).

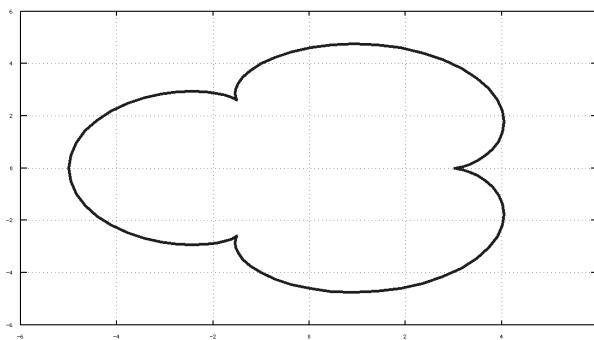


Рис. 4.13. График эпициклоиды

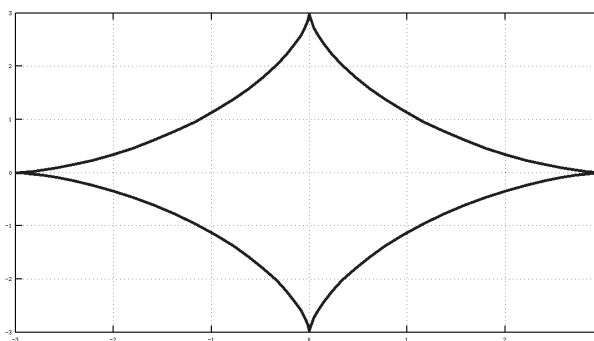
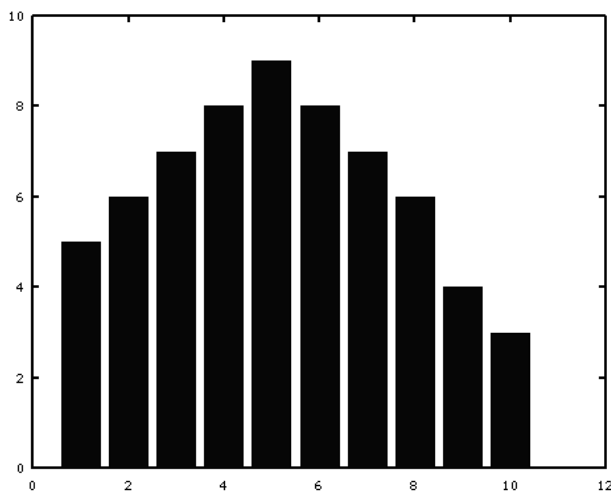


Рис. 4.14. График астроиды

Рис. 4.15. Пример использования функции $\text{bar}(y)$

Функция bar предназначена для построения гистограммы. Функция $\text{bar}(y)$ выводит элементы массива y в виде гистограммы, в качестве массива x выступает массив номеров элементов массива y . Функция $\text{bar}(x, y)$ 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 $y=[5; 6; 7; 8; 9; 8; 7; 6; 4; 3]$; $\text{bar}(y)$; строит гистограмму, представленную на рис. 4.15.

Фрагмент $x1=[-2, -1, 0, 1, 2, 3, 4]$; $y1=\exp(\sin(x1))$; $\text{bar}(x1, y1)$; строит гистограмму, представленную на рис. 4.16.

4.2 Построение трёхмерных графиков

График поверхности (трёхмерный или 3D-график) — это график, положение точки в котором определяется значениями трёх координат.

4.2.1 Построение графиков поверхностей

Дадим определение прямоугольной (или декартовой) системы координат в пространстве.

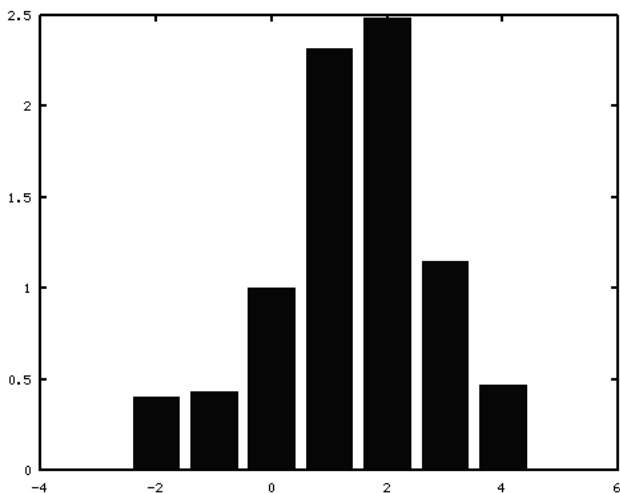


Рис. 4.16. Пример использования функции $\text{bar}(x, y)$

Прямоугольная система координат в пространстве состоит из заданной фиксированной точки O пространства, называемой *началом координат*, и трёх перпендикулярных прямых пространства OX , OY и OZ , не лежащих в одной плоскости и пересекающихся в начале координат, — их называют *координатными осями* (OX — ось абсцисс, OY — ось ординат, OZ — ось аппликат). Положение точки M в пространственной системе координат определяется значением трёх координат и обозначается $M(x, y, z)$. Три плоскости, содержащие пары координатных осей, называются *координатными плоскостями* XY , XZ и YZ .

Величина z называется функцией двух величин x и y , если каждой паре чисел, которые могут быть значениями переменных x и y , соответствует одно или несколько определённых значений величины z . При этом переменные x и y называют аргументами функции $z(x, y)$. Пары тех чисел, которые могут быть значениями аргументов x , y функции $z(x, y)$, в совокупности составляют область определения этой функции.

Для построения графика двух переменных $z = f(x, y)$ необходимо выполнить следующие действия.

1. Сформировать в области построения графика прямоугольную сетку, проводя прямые, параллельные осям $y = y_j$ и $x = x_i$, где $x_i = x_0 + ih$, $h = \frac{x_n - x_0}{n}$, $i = 0, 1, 2, \dots, n$, $y_j = y_0 + j\delta$, $\delta = \frac{y_k - y_0}{k}$, $j = 0, 1, 2, \dots, k$.
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 трёхмерного графика на следующем примере.

Пример 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
```

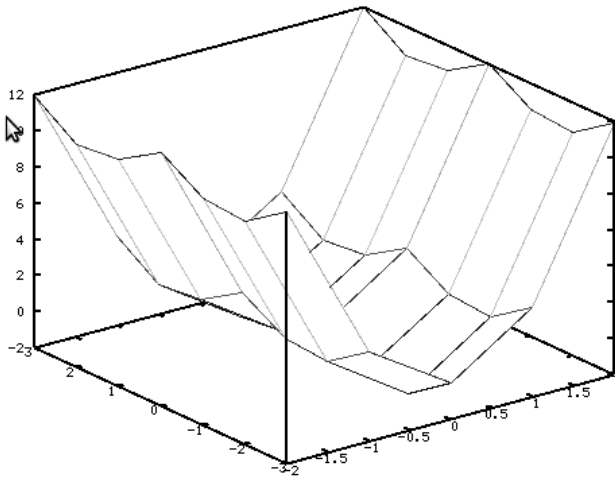


Рис. 4.17. График функции $z(x, y) = 3x^2 - 2\sin^2 y$

Для построения каркасного графика следует обратиться к функции *mesh*: `mesh(x, y, z)`;

После это будет создано графическое окно с трёхмерным графиком (см. рис. 4.17). Как видно, полученный график получился грубым, для получения менее грубого графика следует сетку сделать более плотной (см. листинг 4.14 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 4.14. Построение графика поверхности (пример 4.14).

Любой трёхмерный график можно вращать, используя мышку.

Для построения поверхностей, кроме функции *mesh* построения каркасного графика, есть функция *surf*, которая строит каркасную поверхность, заливая её каждую клетку цветом, который зависит от значения функции в узлах сетки.

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

В листинге 4.15 представлено решение задачи, а на рис. 4.19 изображён получившийся график.

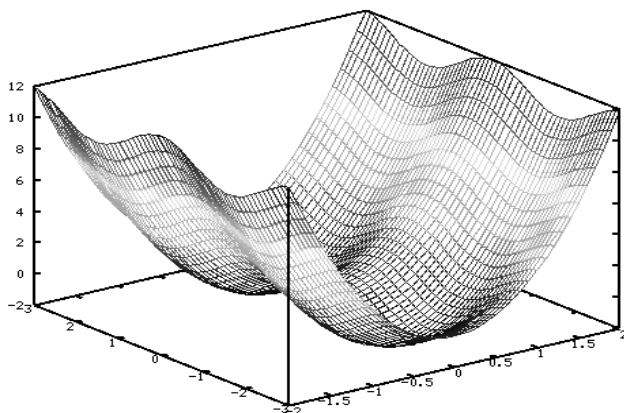


Рис. 4.18. График функции $z(x, y) = 3x^2 - 2\sin^2 y$ с плотной сеткой

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

Листинг 4.15. Построение графика поверхности (пример 4.15).

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*.

Пример 4.16. Построить в одной системе координат графики функций $z(x, y) = \pm(2x^2 + 3y^4) - 1$.

Решение задачи с использованием функции *surf* представлено в листинге 4.16, полученный график изображён 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 4.16. Построение двух графиков одновременно (пример 4.16).

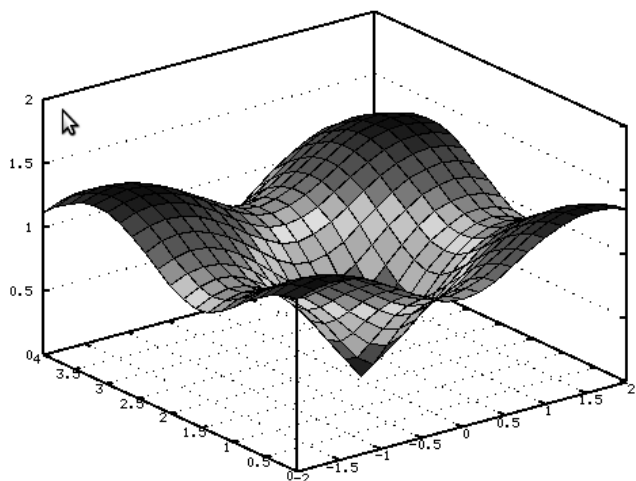


Рис. 4.19. График функции $z(x, y) = \sqrt{\sin^2 x + \cos^2 y}$

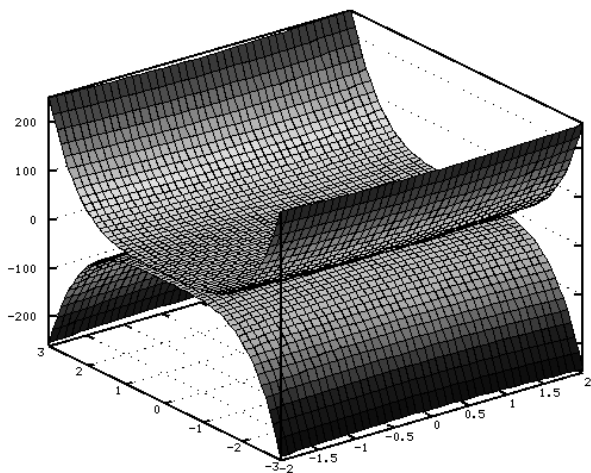


Рис. 4.20. Изображение двух поверхностей в одной системе координат с использованием функции *surf*

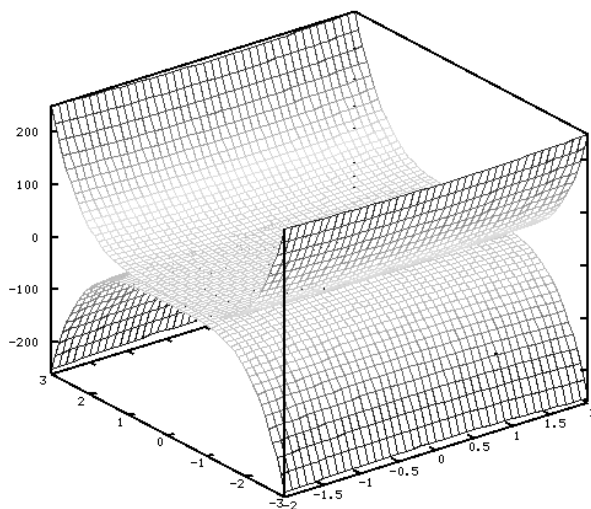


Рис. 4.21. Изображение двух поверхностей в одной системе координат с использованием функции *mesh*

Построение поверхности с помощью функции *mesh* можно осуществить аналогично (см. листинг 4.17), 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 4.17. Построение графиков с помощью *mesh* (пример 4.16).

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 $f(u) \cdot g(v)$, то соответствующие им матрицы X , Y и Z следует формировать в виде матричного умножения $f(u)$ на $g(v)$.
2. Если x , y и z представимы в виде $f(u)$ или $g(v)$, то в этом случае матрицы X , Y и Z следует записывать в виде $f(u)\text{ones}(\text{size}(v))$ или $g(v)\text{ones}(\text{size}(u))$ соответственно.

Рассмотрим несколько задач построения графиков поверхностей заданных параметрически.

Пример 4.17. Построить поверхность однополостного гиперболоида, уравнение которого задано в параметрическом виде $x(u, v) = ch(u)\cos(v)$, $y(u, v) = ch(u)\sin(v)$, $z(u, v) = sh(u)$, $u \in [0, \pi]$, $v \in [0, 2\pi]$.

В листинге 4.18 представлено решение этой задачи, согласно описанному выше алгоритму. График однополостного гиперболоида представлен на рис. 4.22.

```
clear all;
h=3.14/50;
u=[0:h:3.14]'; % Формируем вектор-столбец u.
% Формируем вектор-строку v. Обратите внимание, u — столбец,
% v — строка с одинаковым количеством элементов.
v=[0:2*pi:h:2*pi];
% 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 4.18. Построение поверхности гиперболоида (пример 4.17).

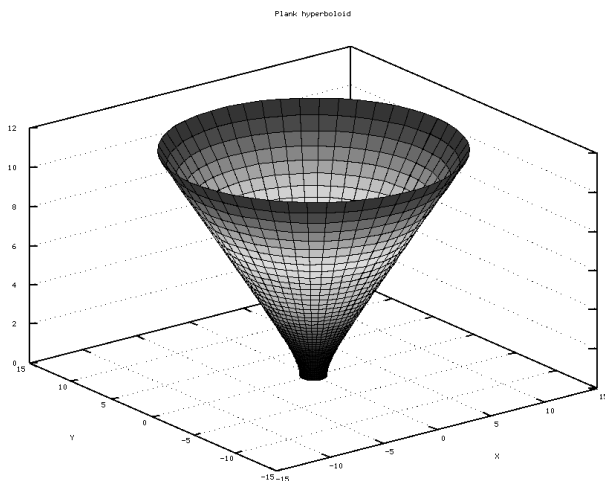


Рис. 4.22. График однополостного гиперboloида

Рассмотрим несколько способов построения сферы в **Octave**.

Пример 4.18. Построить поверхность сферы с центром (x_0, y_0, z_0) и радиусом R .

В декартовой системе координат уравнение сферы имеет вид: $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2$. Его можно записать в параметрическом виде

$$\begin{cases} x(u, v) = x_0 + R \sin(u) \cos(v), \\ y(u, v) = y_0 + R \sin(u) \sin(v), \\ z(u, v) = z_0 + R \cos(u), \end{cases}$$

где $u \in [0, 2\pi]$, $v \in [0, \pi]$.

Методика построения сферы подобна методике построения однополостного гиперboloида, описанной в примере 4.17. В листинге 4.19 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;
u=[-0:h:pi]'; % Формируем вектор-столбец u.
v=[0:2*pi:h:2*pi]; % Формируем вектор-строку v.
```

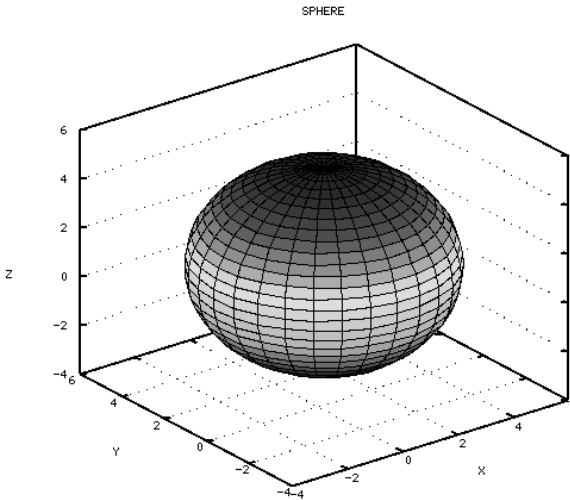


Рис. 4.23. График сферы, построенный с использованием функции *surf*

```
% 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  $\text{ones}(\text{size}(v))$ .
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 4.19. Построение сферы (пример 4.18).

Octave содержит встроенную функцию $[X,Y,Z] = \text{sphere}(n)$, позволяющую формировать матрицы (X,Y,Z) размерности $n + 1$ для построения сферы единичного радиуса (см. пример 4.18) с центром в начале координат.

Для построения сферы единичного радиуса с центром в начале координат достаточно двух команд $[X,Y,Z] = \text{sphere}(n)$; **surf**(X,Y,Z). Чем

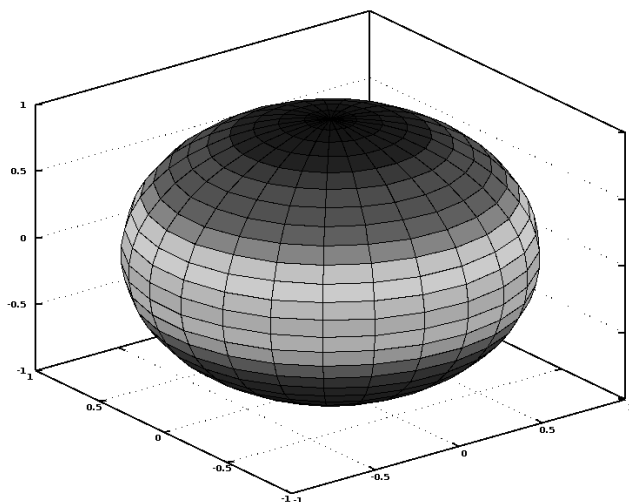


Рис. 4.24. График сферы единичного радиуса, построенный с использованием функции *sphere(25)*

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 . В листинге 4.20 приведено решение примера 4.18 с помощью функции *sphere(n)*.

```
clear all; x0=2; y0=-2; z0=5; R=10 % Определяем центр и радиус сферы.
% 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;
surf(X,Y,Z) % Изображаем сферу
```

Листинг 4.20. Построение сферы с помощью *sphere* (пример 4.18).

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

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

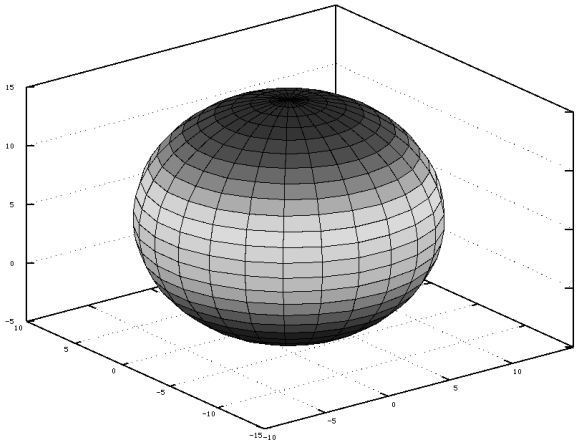


Рис. 4.25. Сфера, построенная с помощью программы, представленной в листинге 4.20

Пример 4.19. Построить поверхность эллипсоида, уравнение которой задано в параметрическом виде:

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

Здесь a, b, c — полуоси эллипсоида, (x_0, y_0, z_0) — центр эллипсоида.

Методика построения эллипсоида подобна тому, как ранее были построены однополостный гиперболоид (пример 4.17) и сфера (пример 4.18). Для этого необходимо сформировать матрицы X , Y и Z , после чего вызвать функцию *surf*. Как это сделать показано в листинге 4.21.

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

```

Листинг 4.21. Построение поверхности эллипсоида (пример 4.19).

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 $[X, Y, Z] = \text{ellipsoid}(xc, yc, zc, xr, yr, zr, n)$, которая позволяет автоматически сформировать матрицы X , Y , Z .

В функции *ellipsoid*:

- 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. 4.23 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');

```

Для построения цилиндров и круговых конусов можно использовать функцию *cylinder* для формирования матриц X , Y , Z . Затем строим саму поверхность (цилиндр, конус) с помощью функции *surf*.

Познакомимся с функцией *cylinder* подробнее. Обращение к функции имеет вид. $[X, Y, Z] = \text{cylinder}(r, n)$;

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 $n + 1$.

Рассмотрим несколько примеров.

Пример 4.20. Построить цилиндр радиуса $R = 4$ и высотой $h = 1$.

Текст программы приведён в листинге 4.22, график — на рисунке 4.27.

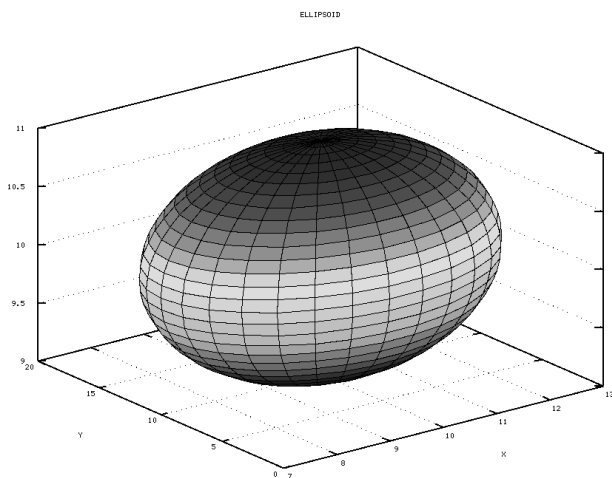


Рис. 4.26. Эллипсоид с центром в точке $(10, 10, 10)$ и полуосями $a = 3$, $b = 7$, $c = 1$

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

Листинг 4.22. Построение цилиндра высотой 1 (пример 4.20).

Пример 4.21. Построить цилиндр радиуса 4 и высотой 20.

Текст программы приведён в листинге 4.23, график — на рисунке 4.28.

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

Листинг 4.23. Построение цилиндра высотой 20 (пример 4.21).

Пример 4.22. Примеры круговых конусов.

Рассмотрим несколько примеров.

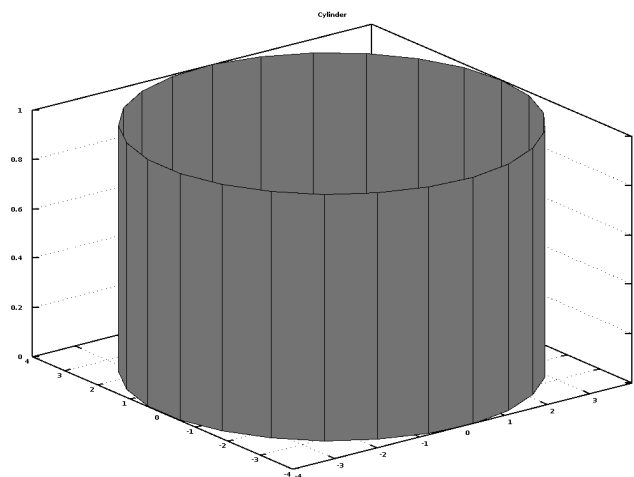


Рис. 4.27. Цилиндр радиуса $R = 4$ и высотой $h = 1$

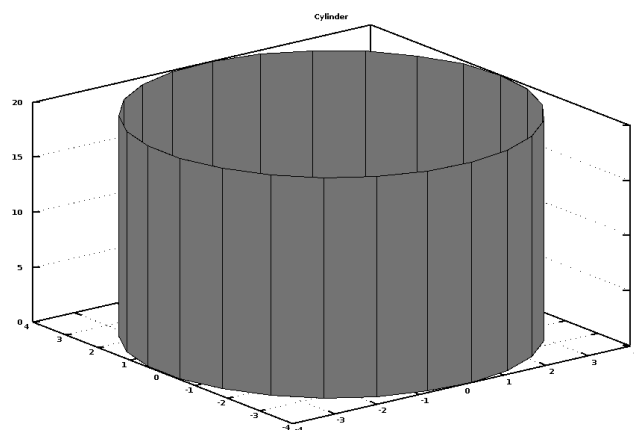


Рис. 4.28. Цилиндр радиуса $R = 4$ и высотой $h = 20$

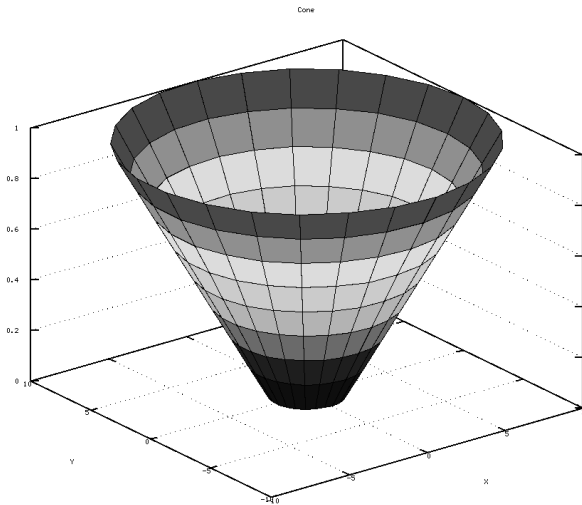


Рис. 4.29. Усечённый круговой конус к листингу 4.24

Усечённый круговой конус, представленный на рис. 4.29, генерируется программой из листинга 4.24.

```
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 4.24. Построение усечённого кругового конуса (пример 4.22).

Круговой конус, представленный на рис. 4.30, генерируется программой в листинге 4.25.

```
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 4.25. Построение кругового конуса (пример 4.22).

В завершении приведён листинг 4.26, который генерирует поверхность, представленную на рис. 4.31.

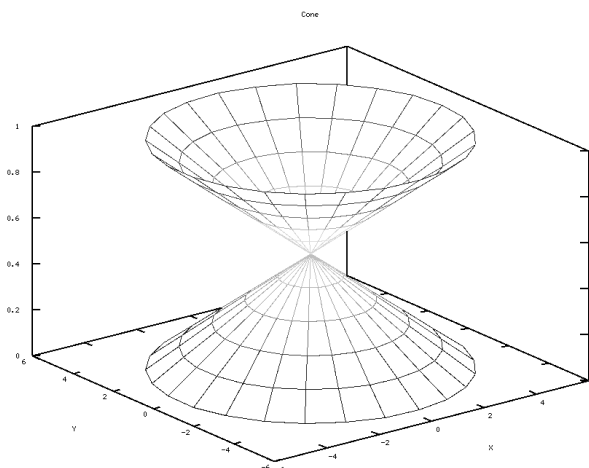


Рис. 4.30. Круговой конус к листингу 4.25

```
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 4.26. Поверхность (см. рис. 4.30)

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);
```

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

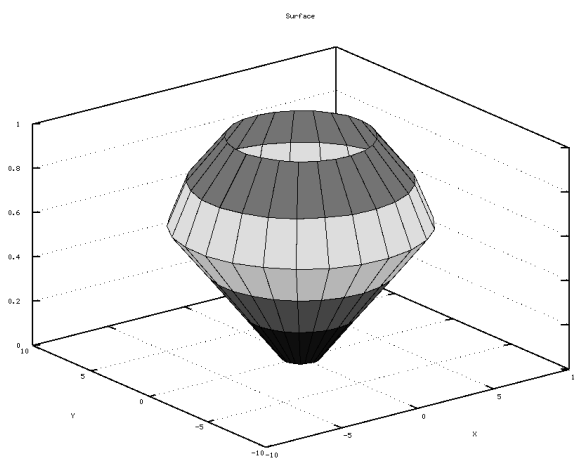


Рис. 4.31. Поверхность к листингу 4.26

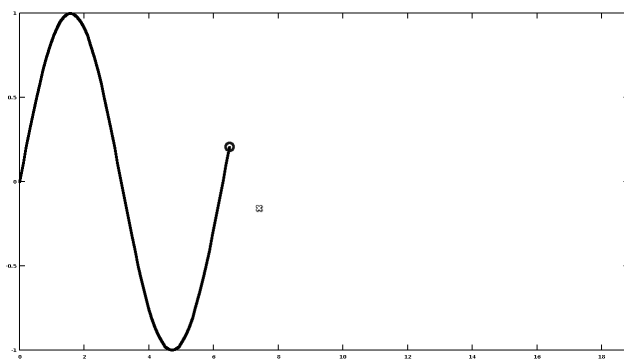


Рис. 4.32. Движение точки вдоль синусоиды

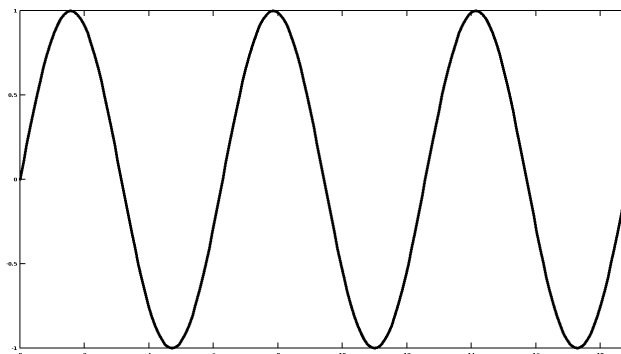


Рис. 4.33. Окончательный вид траектории движения точки

4.4 Графические объекты

Встроенный язык **Octave** — объектно-ориентированный язык программирования. Все объекты находятся в определённой 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 $x(t)$:

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

Окно с графиком синуса на интервале $[-3\pi; 3\pi]$ представлено на рис. 4.34. В результате работы функции *plot* были созданы три графических объекта:

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

При работе с переменными, в которых хранятся объекты, пользователь оперирует ими как обычными переменными. Однако реально это указатели — адреса в памяти, в которых хранятся объекты; в **Octave** в качестве указателя используется номер объекта.

Для получения указателей на объекты в языке **Octave** есть три функции:

- *gcf()* возвращает указатель на текущее графическое окно³;
- *gca()* возвращает указатель на текущие оси;
- *gco()* возвращает указатель на текущий графический объект.

³Синтаксис **Octave** допускает обращение и без скобок (*gcf* — верно)

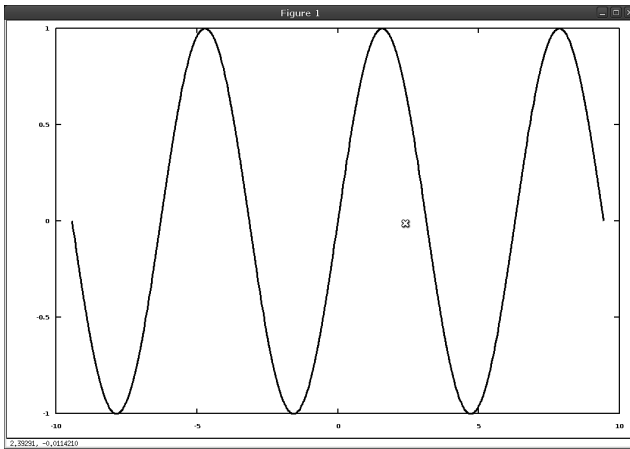


Рис. 4.34. График функции $y = \sin(x)$ на интервале $[-3\pi; 3\pi]$

4.4.1 Свойства графических объектов

Для установки свойств объектов служит функция *set*

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; *Значение1*, *Значение2*, *Значение3*, ... — новые значения свойств.

В простейшем виде функция *set* имеет вид: *set(h, 'Свойство', Значение)*

Для получения свойства объекта служит функция *get*: *get(h, 'Свойство')*; Функция возвращает значения *Свойства* объекта с указателем *h*. Если к функции *get* обратиться с одним параметром *h*, то функция вернёт значения всех свойств объекта в виде *Свойство = Значение*.

4.4.2 Работа с графическим окном

Как уже рассматривалось ранее, для создания графического окна служит функция *figure()*, которая создаёт пустое графическое окно (см. рис. 4.35) и возвращает указатель на него.

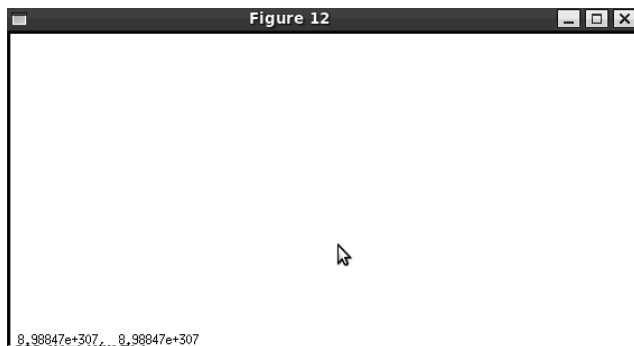


Рис. 4.35. Окно, созданное с помощью функции *figure()*

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 определёнными 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*, хранящее строку, которая будет дописана к имени окна после стандартного имени окна *Figure 1*, *Figure 2*, ...; Например,

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

В результате появится окно, представленное на рис. 4.36. Если свойству *numbertitle* присвоить значение *'off'*, то это позволит отказать от текущей нумерации окон *Figure 1*, *Figure 2*, ... (см. листинг ниже и рис. 4.37).

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

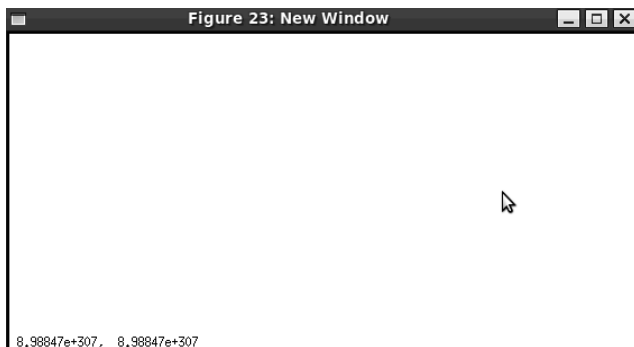


Рис. 4.36. Окно с изменённым заголовком



Рис. 4.37. Окно после выключения стандартной нумерации окон

Как и у многих рассматриваемых графических объектов, у окна есть свойство *Position*, определяющее расположение объекта. *Position* — массив из четырёх 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 «*New Window*» шириной и высотой 400 пикселей, с левым нижним углом с координатами (75, 90):

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

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

4.4.3 Свойства осей графика

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

В листинге 4.27 приведены свойства осей графика, представленного на рис. 4.34.

```
>>>ans =
scalar structure containing
    the fields:
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 =
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 =
```

<pre> 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 </pre>	<pre> 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 plotboxaspectrationmode = auto projection = orthographic tickdir = in tickdirmode = auto ticklength = 0.010000 0.025000 tightinset = 0 0 0 0 units = normalized </pre>
--	---

Листинг 4.27. Свойства осей

Рассмотрим наиболее часто используемые свойства осей:

- *box* — определяет, заключать оси в прямоугольную рамку ('on' — значение по умолчанию) или нет (значение — 'off');
- *color* — определяет цвет фона графика, цвет задаётся в формате *RGB* [*rgb*], где *r*, *g*, *b* — яркость красного, зелёного и синего цветов соответственно, которая меняется от 0 до 1 (см. табл. 4.4) или один из предопределённых цветов.
- *fontangle* — позволяет 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 насыщенность 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 равно 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;
- *axislocation* — 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* — векторы, определяющие координаты разметки соответствующих осей.

В листинге 4.28 представлены команды, изменяющие внешний вид осей графика, изображённого на рис. 4.34. График функции $x = \sin(t)$ на интервале $[-3\pi; 3\pi]$ после их применения представлен на рис. 4.38.

Таблица 4.4: Наиболее распространённые цвета

Цвет	Цвет в формате RGB
Чёрный	[0 0 0]
Синий	[0 0 1]
Тёмно-синий	[0 0 128/255]
Зелёный	[0 1 0]
Тёмно-зелёный	[0 128/255 0]
Голубой	[0 1 1]
Тёмно-голубой	[0 128/255 128/255]
Красный	[1 0 0]
Тёмно-красный	[128/255 0 0]

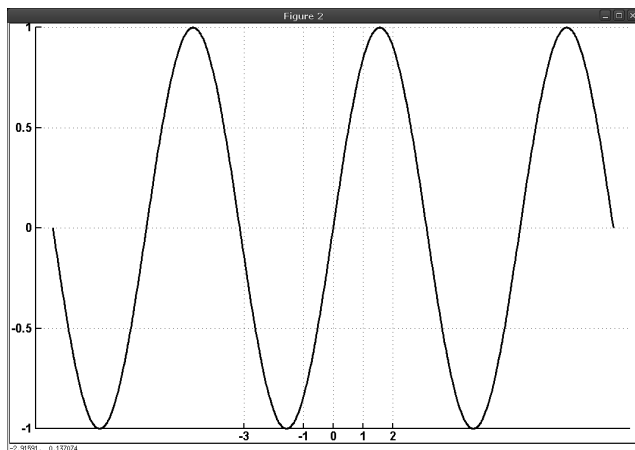


Рис. 4.38. График функции $x = \sin(t)$ на интервале $[-3\pi; 3\pi]$.

Таблица 4.4 — продолжение

Цвет	Цвет в формате RGB
Пурпурный	[1 0 1]
Тёмно-пурпурный	[128/255 0 128/255]
Жёлтый	[1 1 0]
Тёмно-жёлтый	[128/255 128/255 0]
Тёмно-серый	[128/255 128/255 128/255]
Светло-серый	[192/255 192/255 192/255]
Белый	[1 1 1]

```
h=figure(); t=-3*pi:pi/100:3*pi; x=sin(t); plot(t,x);
set(gca,'box','off'); % Убираем прямоугольную сетку вокруг оси.
set(gca,'fontname','Arial'); % Определяем шрифт.
set(gca,'fontsize',20); % Определяем размер шрифта 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 4.28. Изменение вида осей графика

Обращение к функции создания осей с определёнными свойствами имеет вид: `axes('Свойство1', Значение1, 'Свойство2', Значение2,`

'Свойство', Значение, ...); С помощью функции *set* можно также изменять свойства линий, которые формируются с помощью подробно рассмотренной ранее функции *plot*.

Рассмотрим наиболее часто используемые свойства линий:

- *color* — определяет цвет текущей линии в формате RGB или с помощью предопределённого 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.

4.4.4 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 приведёт к исчезновению всех объектов, которые располагались на них.

Очистка текущих осей осуществляется функцией *cla*, очистка текущего окна — функцией *clf*.

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

4.4.5 Примеры

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

В листинге 4.29 приведено решение этой задачи с подробными комментариями.

```
t = -5:0.1:5; x=exp(cos(t)); y=exp(sin(t)); z=cos(t.^2);
% Создаём первое графическое окно, указатель, на которое будет
hfig1=figure; % храниться в переменной hfig1.
% Создаём второе графическое окно, указатель, на которое будет
hfig2=figure; % храниться в переменной hfig2.
```

```

figure(hfig1); % Объявляем первое графическое окно текущим.
% Выводим в нём оси, указатель на на которые будет храниться
hAxes1=axes; % в переменной hAxes1.
% 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);
figure(hfig2); % Объявляем второе графическое окно текущим.
% Выводим в нём оси, указатель на на которые будет храниться
hAxes2=axes; % в переменной hAxes2.
% Выводим в этом окне график функции  $z(t)$ , указатель на который
h_gr2=plot(t,z); % записываем в переменную  $h\_gr2$ .
figure(hfig1); % Объявляем первое графическое окно текущим
% 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
set(hAxes1,'ygrid','on'); % для осей hAxes1.
% 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');
figure(hfig2); % Объявляем второе графическое окно текущим
% 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
set(hAxes2,'xgrid','on'); % для осей hAxes2.
% Включаем отображение линий сетки, перпендикулярной оси OY
set(hAxes2,'ygrid','on'); % для осей hAxes2.
% Устанавливаем зелёный 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 4.29. Программа создания графиков (пример 4.23).

На рис. 4.39 и 4.40 представлены созданные с помощью программы, приведённой в листинге 4.29, окна с графиками функций $x = e^{\cos(t)}$, $y = e^{\sin(t)}$ и $z = \cos(t^2)$ соответственно.

При программировании работы с графическими окнами вызов функции *plot* может осуществляться со всеми параметрами, рассмотренными в этой главе. Для того, чтобы добавить новый график в

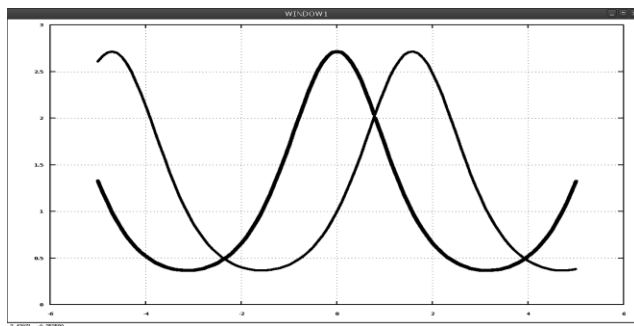


Рис. 4.39. Окно с графиками функций $x = e^{\cos(t)}$ и $y = e^{\sin(t)}$

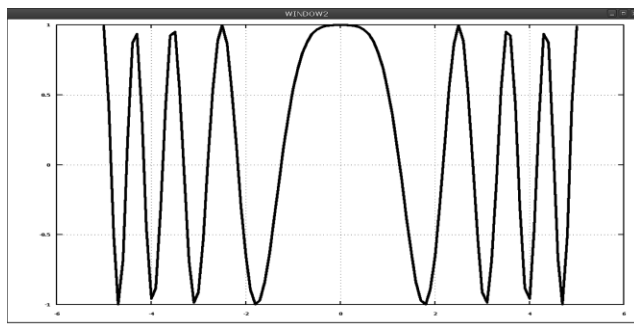


Рис. 4.40. Окно с графиком функции $z = \cos(t^2)$

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.

Пример 4.24. Изобразить графики функций $x_i = \alpha_i e^{\sin(t)}$, $y_i = \sin(\alpha_i t)$, $z_i = \cos(\alpha_i t)$, $v_i = \sin(2\alpha_i t) + \cos(3\alpha_i t)$ на интервале $[-5; 5]$, если коэффициенты $\alpha = 0.5, 0.6, 0.73, 0.79$ хранятся в текстовом файле `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. В листинге 4.30 приведена программа, а на рис. 4.41 представлено полученное в результате графическое окно.

```

f=fopen('gr.txt','rt'); % Открываем файл gr.txt в режиме чтения.
alf=fscanf(f,'%f',4); % Считываем из него данные в массив alf.
% 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);
hfig1=figure; % Создаём графическое окно
% Устанавливаем новое имя первого графического окна.
set(hfig1,'numbertitle','off'); set(hfig1,'name','Plots');
% Выводим в нём 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]);
plot(t,x); % Выводим множество графиков xi(t)
% 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]);
plot(t,y); % Выводим множество графиков yi(t).
% 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]);
plot(t,z); % Выводим множество графиков zi(t)
% 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]);
plot(t,v); % Выводим множество графиков vi(t).
% Выводим линии сетки на осях.
set(haxes4,'xgrid','on','ygrid','on');

```

Листинг 4.30. Построение графиков функций (пример 4.24).

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 гра-

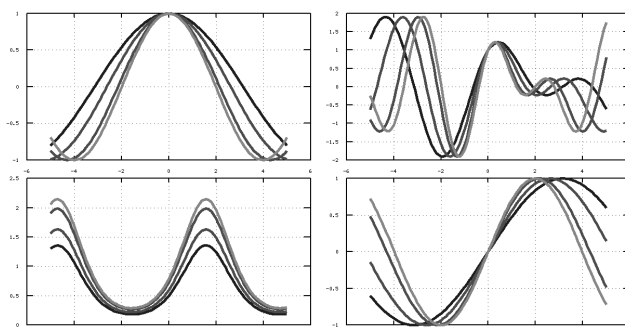


Рис. 4.41. Окно с графиками функций $x_i = \alpha_i e^{\sin(t)}$, $y_i = \sin(\alpha_i t)$, $z_i = \cos(\alpha_i t)$, $v_i = \sin(2\alpha_i t) + \cos(3\alpha_i t)$

фика, нет необходимости пересчитывать их в «экранную» систему координат.

В листинге 4.31 представлена программа построения графика $\sin(x)$ 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. 4.42.

```
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 4.31. Построение графика синуса с подписями

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;
- *backgroundcolor* — 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;

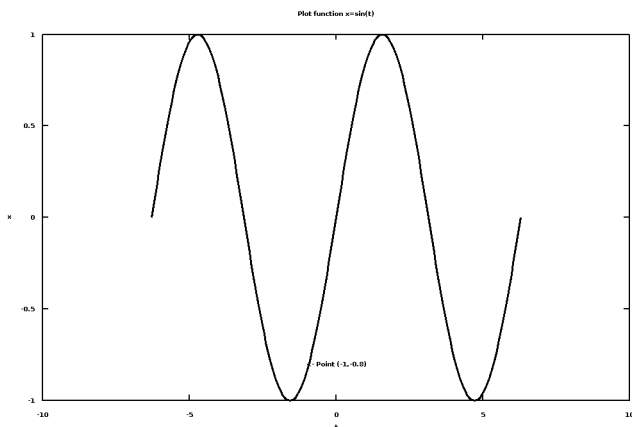


Рис. 4.42. График функции $x = \sin(t)$ с подписями

- *linewidth* — определяет толщину линий прямоугольной рамки.

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

Глава 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

Векторы и матрицы в **Octave** задаются путём 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
```

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
```

Обратиться 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
```

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

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

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
```

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

```

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
>>> Tablica=[M;M;M] % Вертикальная конкатенация матриц
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

```

Важную роль при работе с матрицами играет знак двоеточия «:». Примеры с подробными комментариями приведены в листинге 5.1.

```

>>> 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
>>> Tabl(:,3) % Выделить из матрицы 3-й столбец
ans =
     0.80000
     1.10000
     5.60000
    -8.50000
>>> Tabl(1,:) % Выделить из матрицы 1-ю строку
ans =   -1.20000    3.40000    0.80000
>>> Matr=Tabl(2:3,1:2) % Выделить из матрицы подматрицу
Matr =
     0.90000   -0.10000
     7.60000   -4.50000
% Вставить подматрицу в правый 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
>>> Tabl(:,2)=[] % Удалить из матрицы 2-й столбец
Tabl =
    -1.20000    0.80000
     0.90000    1.10000
     7.60000   -0.10000
     9.00000   -4.50000
>>> Tabl(2,:)=[] % Удалить из матрицы 2-ю строку
Tabl =
    -1.20000    0.80000
     7.60000   -0.10000
     9.00000   -4.50000

```

```
>>> Matr % Представить матрицу в виде вектора-столбца
Matr =
    0.90000    -0.10000
    7.60000    -4.50000
>>> Vector=Matr(:)
Vector =
    0.90000
    7.60000
   -0.10000
   -4.50000
>>> V=Vector(1:3) % Выделить из вектора элементы со 1-го по 3-й
V =
    0.90000
    7.60000
   -0.10000
>>> V(2)=[] % Удалить из массива 2-й элемент
V =
    0.90000
   -0.10000
```

Листинг 5.1. Пример использования знака двоеточия «:»

5.2 Действия над векторами

Рассмотрим действия над векторами, предусмотренные в **Octave**.

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

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

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

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

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


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

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

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

Деление вектора на число определяется аналогично умножению, как деление каждого элемента вектора на это число¹. Знак деления «/» применяют для записи операции деления вектора на число:

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

Умножение векторов определено только для векторов одинакового размера, причём один из них должен быть вектором-столбцом, а второй — вектором-строкой. Если вектор-строку умножать на вектор-столбец, получится скалярное произведение векторов (число)², а если умножать вектор-столбец на вектор-строку, то получится матрица у которой каждая строка представляет собой исходный вектор-строку, умноженный на соответствующие элементы вектора столбца. Операция умножения вектора на вектор записывается с помощью знака «*», также как и операция умножения вектора на число. Примеры умножения векторов:

¹Если делитель будет равен нулю, то в ответе будет Inf или NaN и предупреждение `warning: division by zero.` (Прим. редактора).

²Напомним, что скалярное умножение векторов определено, как сумма произведений соответствующих элементов векторов.

```

>>> a=[2 4 6];b=[1 3 5];
% В результате умножения вектора-строки на вектор-столбец получится число
>>> a*b'
ans = 44
% Результат умножения вектора-столбца на вектор-строку — 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)

```

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:

```

>>> 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

```

Рассмотрим ещё несколько операций поэлементного преобразования вектора. К каждому элементу вектора можно добавить (вычесть) число, используя арифметическую операцию «+» («-»)³:

```

>>> 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

```

Поэлементное умножение векторов выполняется при помощи оператора «.*», результатом такого умножения является вектор, каж-

³Вдумчивый читатель сразу заметит, что это частный случай предыдущего примера — применение к вектору линейного преобразования. (*Прим. редактора*).

дый 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
```

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

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

```
>>> 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
```

Поэлементное возведение в степень выполняет оператор «.^», результатом является вектор, каждый элемент которого это соответствующий элемент заданного вектора, возведённый в указанную степень:

```
>>> a=[2 4 6];b=[1 3 5];
>>> a.^2 % Каждый элемент вектора возвести в квадрат
ans = 4    16    36
>>> b.^(1/2) % Извлечь корень квадратный из каждого элемента вектора
ans = 1.0000    1.7321    2.2361
>>> b.^a % Каждый элемент вектора b возвести в степень a
ans = 1        81    15625
>>> a.^(1./b) % Извлечь корень b-й степени из каждого элемента вектора a
ans = 2.0000    1.5874    1.4310
```

5.3 Действиям над матрицами

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

```

Умножить на число (запись «*») 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

```

Операция *транспонирования* (запись «'») 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

```

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

```

>>> 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)

```

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

```

>>> 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

```

Для *поэлементного преобразования матриц* (листинг 5.2) можно применять операции, описанные ранее, как операции поэлементного преобразования векторов: *добавление (вычитание) числа к каждому элементу матрицы* («+» или «-»), *поэлементное умножение матриц* («.*») *одинакового размера, поэлементное деление матриц одинакового размера* (прямое «./» и обратное «.\»), *поэлементное возведение в степень* («.^») и *применение к каждому элементу матрицы математических функций*.

```

>>> 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 =
  1.333333 -0.666667 -0.333333
  3.000000  2.000000 -0.666667
 -1.000000 -0.333333  0.666667
>>> 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 =
  0.750000 -1.000000 -1.000000
  0.111111  0.166667 -1.000000
 -1.333333 -1.000000  1.500000
>>> M.\N
ans =
  1.333333 -1.000000 -1.000000
  9.000000  6.000000 -1.000000
 -0.750000 -1.000000  0.666667
>>> 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 5.2. Примеры преобразования матриц

Рассмотрим работу с матрицами на следующем примере.

Пример 5.1. Вычислить математическое выражение $(2A + \frac{1}{3}B^T)^2 - AB^{-1}$ для заданных значений A и B .

Решение задачи показано в листинге 5.3.

```
>>> 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 5.3. Вычисление математического выражения (пример 5.1).

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

```
>>> 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
>>> X*A-B % Проверка X · A - B = 0
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
```

Соответственно $A \setminus B$ 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
>>> A*X-B % Проверка A · X - B = 0
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

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:

```
>>> A=[1 2;1 1]; b=[7;6];
>>> x=A\b
x =
     5
     1
>>> A*x % Проверка Ax = b
ans =
     7
     6
```

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:

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

Рассмотрим наиболее часто используемые функции.

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
```


prod(*X*) — вычисляет произведение элементов вектора *X*.

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

cumprod(*X*) — формирует вектор кумулятивного произведения — вектор того же типа и размера, что и *X* вида: $x_1, x_1 \cdot x_2, x_1 \cdot x_2 \cdot x_3, \dots, x_1 \cdot x_2 \cdot \dots \cdot x_n$, каждый элемент которого рассчитывается по формулам $x'_i = x_1 \cdot x_2 \cdot \dots \cdot x_i$, то есть *i*-й элемент вектора *X* умножается на произведение всех предыдущих элементов.

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

sum(*X*) — вычисляет сумму элементов вектора *X*.

```
>>> X=[1 2 3 4 5 6 7 8 9]; sum(X)
ans = 45
```

cumsum(*X*) — формирует вектор кумулятивной суммы — вектор того же типа и размера, что и *X* вида $x_1, x_1 + x_2, x_1 + x_2 + x_3, \dots, x_1 + x_2 + \dots + x_n$, каждый элемент которого рассчитывается по формуле: $x'_i = x_1 + x_2 + \dots + x_i$, 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
```

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

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

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

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

$\text{max}(X)$ — находит максимальный элемент массива X или при вызове вида $[\text{nom}X, \text{nom}] = \text{max}(X)$ 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
```

$\text{mean}(X)$ — 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
```

$\text{dot}(x_1, x_2)$ — вычисляет скалярное произведение векторов x_1 и x_2 .

```
>>> 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
```

$\text{cross}(x_1, x_2)$ — вычисляет векторное произведение векторов x_1 и x_2 .

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

$\text{sort}(X)$ — выполняет сортировку массива X .

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

5.4.2 Функции для работы с матрицами

$\text{eye}(n[, m])$ — 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
```

$\text{ones}(n[, m, p, \dots])$ — 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
```

`zeros($n[m, p, \dots]$)` — 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
```

$\text{diag}(X[k])$ — возвращает квадратную матрицу с элементами X на главной диагонали или на k -й. Функция $\text{diag}(M[k])$, где M — ранее определённая 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    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
 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
```

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

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

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

```
>>> randn
ans = -1.0607
>>> randn
ans = -0.47825
```

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

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

logspace(a, b, n) — формирует массив из 50 (если n не указано) или из n точек, равномерно распределённых в логарифмическом масштабе между значениями 10^a и 10^b ; функция *logspace(a, pi)* даёт равномерное распределение из 50 точек в интервале от 10^a до pi .⁴

```
>>> 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
```

⁴Обратите внимание, что это идёт в разрез с определением функции для произвольного $b \neq pi$, согласно которому интервал должен был бы быть от 10^a до 10^{pi} , сделано это для совместимости с соответствующей функцией *matlab*. (Прим. редактора).

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

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

reshape(*M*, *m*, *n*) — возвращает матрицу размерностью $m \times n$ сформированную из матрицы *M* путём последовательной выборки по столбцам, если матрица *M* не имеет $m \times n$ элементов, то выдаётся 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
```

$\text{cat}(n, A, B, [C, \dots])$ — объединяет матрицы A и B или все входящие матрицы.

```

>>> 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

```

$\text{rot90}(M, k)$ — осуществляет поворот матрицы M на 90 градусов против часовой стрелки или на величину $90 \cdot k$, где k — целое число⁵.

```

>>> 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

```

$\text{tril}(M, k)$ — формирует из матрицы M нижнюю треугольную матрицу начиная с главной или с k -й диагонали.

```

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

```

⁵Отрицательные значения k указывают на поворот по часовой стрелке, проверьте, что $\text{rot90}(M, -1)$ даст тот же результат, что и $\text{rot90}(M, 3)$. (Прим. редактора).

```

    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

```

$\text{triu}(M[,k])$ — 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

```

$\text{size}(M)$ — определяет число строк и столбцов матрицы M , результатом её 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

```

$\text{prod}(M[,k])$ — формирует вектор-строку или вектор-столбец, в зависимости от значения k , каждый элемент которой является произведением элементов соответствующего столбца или строки матрицы M ,

если значение параметра k в конструкции отсутствует, то по умолчанию вычисляются произведения столбцов матрицы; понятно, что результатом работы функции $prod(prod(M))$ 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
```

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

$sum(M[,k])$ — формирует вектор-строку или вектор-столбец, в зависимости от значения k , каждый элемент которой является суммой элементов соответствующего столбца или строки матрицы M , если значение параметра k в конструкции отсутствует, то по умолчанию

вычисляются суммы столбцов матрицы. Сумму всех элементов матрицы вычисляет функция $\text{sum}(\text{sum}(M))$.

```
>>> 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
```

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

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

$\text{diff}(M)$ — 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
```

$\text{min}(M)$ — формирует вектор-строку, каждый элемент которой является наименьшим элементом соответствующего столбца матрицы M . Определить положение этих элементов в матрице можно, если вызвать функцию в формате $[n, m] = \text{min}(M)$, где n — это вектор минимальных элементов столбцов матрицы M , а m — вектор номеров строк матрицы M , в которых находятся эти элементы, конструкция $\text{min}(\text{min}(M))$ позволит отыскать минимум среди всех элементов матрицы. Вызов функции в виде $\text{min}(M, [], k)$ или $[n, m] = \text{min}(M, [], k)$ позволяет управлять направлением поиска, в частности можно отыскать минимальные элементы и их положение в строках матрицы M . И, наконец, функция $\text{min}(A, B)$ сформирует матрицу из строк $\text{min}(A)$ и $\text{min}(B)$.

```
>>> 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))% Минимум среди всех элементов матрицы
ans = -2
>>> [n,m]=min(min(M))
n = -2
m = 1
>>> min(M, [], 1)
```

```

ans =    -2    -1    -2     1
>>> min(M, [], 2) % Как и min(M) формирует вектор-строку, каждый элемент
% которой равен минимальному элементу в соответствующем столбце M
ans =
    -2
    -1
    -1
    -2
>>> [n,m]=min(M, [], 2) % Формирует вектор-столбец, каждый элемент
% которого равен минимальному элементу в соответствующей строке матрицы
% M и их положение в матрице — номера столбцов в которых они находятся.
n =
    -2
    -1
    -1
    -2
m =
     3
     3
     2
     1
>>> 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 % Первая строка — минимумы столбцов матрицы A,
     3     4     5 % а вторая строка — матрицы B
     6     7     8

```

$\max(M)$ — формирует вектор-строку, каждый элемент которой 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:

```

>>> 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

```

$\text{mean}(M, [k])$ — 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 $\text{mean}(\text{mean}(M))$.

```

>>> 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

```

$\text{sort}(M)$ — выдаёт 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

```

$\text{sqrtn}(M)$ — относится к так называемым матричным функциям и возвращает матрицу X , для которой $X * X = M$ (матрица M должна быть квадратной).

```

>>> A=[1 0 -3;0 1 2;2 0 -1]
A =
     1     0    -3
     0     1     2
     2     0    -1
>>> X=sqrtn(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
>>> Y=sqrt(A)% Извлечение квадратного корня из элементов матрицы 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

```

$\expm(M)$ и $\logm(M)$ — взаимобратные матричные функции, первая вычисляет матричную экспоненту e^M , а вторая выполняет логарифмирование по основанию e .

```

>>> A=[1 0 -3;0 1 2;2 0 -1];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

```

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

```

$\text{trace}(M)$ — 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

```

$\text{norm}(M, p)$ — возвращает различные виды норм матрицы M в зависимости от p , если аргумент $p = 1, 2, \text{inf}, \text{fro}$ не задан, то вычисляется вторая норма матрицы M .

```

>>> M=[-1 1 -2 3; 4 5 -1 2; 3 -1 4 1; -2 5 4 3];
>>> norm(M) % Вторая норма
ans =      8.5506
>>> norm(M, 2)
ans =      8.5506
>>> norm(M, 1) % Первая норма
ans =      12
>>> norm(M, inf) % Бесконечная норма
ans =      14
>>> norm(M, 'fro') % Евклидова норма
ans =     11.916

```

$\text{cond}(M, p)$ — 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

```

$\text{rcond}(M)$ — 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

```

$\text{inv}(M)$ — 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
>>> A*invA % Проверка
ans =
     1         0   -5.5511e-17
     0         1         0
     0         0         1
```

$\text{eig}(M)$ — возвращает вектор собственных значений матрицы M , вызов функции в формате $[Matr, D] = \text{eig}(M)$ даст матрицу $Matr$, столбцы которой — собственные векторы матрицы M и диагональную матрицу D , содержащую собственные значения матрицы M , функция $\text{eig}(A, B)$, где A и B квадратные матрицы, выдаёт вектор обобщённых 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
```

$\text{poly}(M)$ — возвращает вектор-строку коэффициентов характеристического полинома матрицы M .

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

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

```
>>> M=[3 -2 1 5;6 -4 2 7;9 -6 3 12]
M =
      3      -2       1       5
      6      -4       2       7
      9      -6       3      12
>>> rref(M)
ans =
      1    -0.66667    0.33333      0
      0       0         0         1
      0       0         0         0
```

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

```
>>> M=[10 1 1;2 10 1;2 2 10]
M =
     10       1       1
       2      10       1
       2       2      10
>>> chol(M)
ans =
      3.1623    0.31623    0.31623
      0         3.1464    0.28604
      0         0         3.1334
% Матрица не симметрическая
>>> A=[1 2;1 1]; chol(A)
error: chol: matrix not positive definite
% Матрица содержит элементы < 0
>>> M=[3 -2 1 5;6 -4 2 7;9 -6 3 12];
>>> chol(M)
error: CHOL requires square matrix
```

$lu(M)$ — выполняет LU -разложение, функция $[L, U, P] = lu(M)$ возвращает три матрицы: L — нижняя треугольная, U — верхняя треугольная и P — матрица перестановок, причём $P \cdot A = L \cdot U$. 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

```

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

```

>>> 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

```

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

```

>>> M=[3 -2 1; 5 6 -4; 2 7 9];
>>> svd(M)
ans =
11.7553
 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

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. Если $m \times n$ выражений расставлены в прямоугольной таблице из m строк и n столбцов, то говорят о матрице размера $m \times n$:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \quad (5.1)$$

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

$$\begin{pmatrix} a_{11} & a_{12} & \dots & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ii} & \dots & a_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & \dots & \dots & a_{mn} \end{pmatrix} \quad (5.2)$$

Матрица размером $m \times n$, (где $m \neq n$) называется прямоугольной (5.1). В случае если $m = n$, матрицу называют квадратной матрицей порядка n .

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (5.3)$$

В частности, матрица типа $1 \times n$ — это вектор-строка: $(a_{11} \ a_{12} \ \dots \ a_{1n})$.

Матрица размером $m \times 1$ является вектором-столбцом: $\begin{pmatrix} a_{11} \\ a_{21} \\ \dots \\ a_{m1} \end{pmatrix}$

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

- *нулевой*, если все её элементы равны нулю: $\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix}$

- *верхней треугольной*, если все элементы, расположенные ниже главной диагонали, равны нулю: $\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$

- *нижней треугольной*, если все элементы, расположенные выше главной диагонали, равны нулю: $\begin{pmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$

- *диагональной*, если все элементы, кроме элементов главной диагонали, равны нулю: $\begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$

- *единичной*, если элементы главной диагонали равны единице, а все остальные нулю: $\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}$

Определителем (детерминантом) матрицы A является число $\det A$ или Δ , вычисляемое по правилу: $\det A = \sum (-1)^{\tilde{\lambda}} a_{1i_1} a_{2i_2} \dots a_{ni_n}$, где сумма распределена на всевозможные перестановки (i_1, i_2, \dots, i_n) элементов $1, 2, \dots, n$ и, следовательно, содержит $n!$ слагаемых, причём $\tilde{\lambda} = 0$, если перестановка чётная, и $\tilde{\lambda} = 1$, если перестановка нечётная.

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

С матрицами можно проводить операции *сравнения*, *сложения* и *умножения*.

Две матрицы $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ считаются *равными*, если они одного типа, 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 типа называется матрица $C = \{c_{ij}\}$ того же типа, элементы которой равны сумме соответствующих элементов матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$: $\{c_{ij}\} = \{a_{ij}\} + \{b_{ij}\}$.

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

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

Произведением матриц $A = \{a_{ij}\}$ размерностью $m \times n$ и $B = \{b_{ij}\}$ размерностью $n \times s$ является матрица $C = \{c_{ij}\}$ размерностью $m \times s$, каждый элемент которой можно представить формулой $\{c_{ij}\} = \{a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}\}$, где $i = 1 \dots m, j = 1 \dots s$.

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*: $A^T = \{a_{ji}\}$.

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

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

Обратной матрицей по отношению к данной матрице $A = \{a_{ij}\}$ размерностью $n \times n$, называется матрица $A^{-1} = \{a_{ij}\}$ того же типа,

которая, будучи умноженной как справа, так и слева на данную матрицу, в результате даёт единичную матрицу $E = \{\delta_{ij}\}$, где $\delta_{ii} = 1$, $\delta_{ij} = 0$, при $i \neq j$: $A \cdot A^{-1} = A^{-1} \cdot A = E$.

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

Перейдём к конкретным примерам.

Пример 5.2. Для матриц A , B и C проверить выполнение следующих тождеств:

$$(A \cdot B) \cdot C = A \cdot (B \cdot C) \quad (5.4)$$

$$(A^T + B) \cdot C = A^T \cdot C + B \cdot C \quad (5.5)$$

Из листинга 5.4 видно, что матрицы, получившиеся в результате вычисления левой и правой частей тождества (5.4), равны, следовательно, первое тождество истинно. Для исследования тождества (5.5) из левой части равенства вычитаем правую и получаем нулевую матрицу, что так же приводит к выводу об истинности тождества.

```
>>> A=[1 -2 0; -3 0 4];B=[3 1;2 0;-1 1];C=[1 2;-1 0];
>>> (A*B)*C % Исследование тождества (5.4)
ans =
-2 -2
-14 -26
>>> A*(B*C)
ans =
-2 -2
-14 -26
>>> (A'+B)*C-(A'*C+B*C) % Исследование тождества (5.5)
ans =
0 0
0 0
0 0
```

Листинг 5.4. Проверка матричных тождеств (пример 5.2).

Пример 5.3. Проверить является ли матрица симметрической. Квадратная матрица называется *симметрической*, если $A^T = A$.

В листинге 5.5 видно, что в результате вычитания из матрицы A транспонированной матрицы A^T получена нулевая матрица, то есть тождество выполнено, и заданная матрица — симметрическая.

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

```
0 0 0
0 0 0
```

Листинг 5.5. Проверка симметрической матрицы (пример 5.3).

Пример 5.4. Проверить, является ли матрица кососимметрической. Квадратная матрица называется *кососимметрической*, если $A^T = -A$.

Проверив равенство для заданной матрицы (листинг 5.6), убеждаемся в его истинности.

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

Листинг 5.6. Проверка кососимметрической матрицы (пример 5.4).

Пример 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 следует транспонировать исходную матрицу и найти матрицу, обратную к ней. Если визуально сложно убедиться в том, что транспонированная матрица равна обратной, можно вычислить их разность. В результате должна получиться нулевая матрица (листинг 5.7).

```
>>> 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
>>> det(A) % Определитель матрицы A отличен от нуля
ans = -0.99998
>>> A'-inv(A) % При вычитании из транспонированной матрицы A обратной
% к ней матрицы получаем нулевую матрицу, значит A — ортогональная.
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 5.7. Проверка ортогональности матрицы (пример 5.5).

Пример 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
```

Пример 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: $A^{-1} \cdot A \cdot X = B \cdot A^{-1}$ или $X \cdot A \cdot A^{-1} = B \cdot A^{-1}$. Так как $A^{-1} \cdot A = A \cdot A^{-1} = E$, а $E \cdot X = X \cdot E = X$, то неизвестную матрицу X можно вычислить так: $X = A^{-1} \cdot B$ или $X = B \cdot A^{-1}$. Понятно, что матричное уравнение имеет единственное решение, если A и B — квадратные матрицы n -го порядка, и определитель матрицы A не равен нулю. Как решить матричное уравнение в **Octave**, показано в листинге 5.8.

```
>>> A=[ 2 3;-2 6];B=[2 5;2/3 5/3];
% Решение уравнения A · X = B
>>> X=A\B % Первый способ
X =
    0.55556    1.38889
    0.29630    0.74074
>>> X=inv(A)*B % Второй способ
X =
    0.55556    1.38889
    0.29630    0.74074
>>> A*X-B % Проверка A · X - B = 0
ans =
    0.0000e+00    0.0000e+00
   -3.3307e-16   -6.6613e-16
% Решение уравнения X · A = B
>>> X=B/A % Первый способ
X =
    1.22222    0.22222
    0.40740    0.074074
>>> X=B*inv(A) % Второй способ
X =
    1.22222    0.22222
    0.40740    0.074074
>>> X*A-B % Проверка X · A - B = 0
ans =
    0.0000e+00    0.0000e+00
    1.1102e-16    0.0000e+00
```

Листинг 5.8. Решение матричного уравнения (пример 5.7).

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

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

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

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

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

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

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

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

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

метических и логических операций. В этом разделе на конкретных примерах будут рассмотрены только точные методы решения систем.

Пример 5.8. Решить систему линейных уравнений

$$\begin{cases} 2x_1 - x_2 + 5x_3 = 1 \\ 3x_1 + 2x_2 - 5x_3 = 1 \\ x_1 + x_2 - 2x_3 = 4 \end{cases} \quad (5.6)$$

при помощи правила Крамера.

Правило Крамера заключается в следующем. Если определитель $\det A$ матрицы системы $Ax = b$ из n уравнений с n неизвестными отличен от нуля, то система имеет единственное решение (x_1, x_2, \dots, 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 . Если определитель матрицы системы равен нулю, это не означает, что система не имеет решений: возможно, её нельзя решить по формулам Крамера.

Итак, для решения поставленной задачи необходимо выполнить следующие действия:

- 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 $x_i = \frac{\det_i}{\det}$.

Листинг 5.9 содержит решение поставленной задачи.

```
>>> disp('Решение СЛАН 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 СЛАН 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 5.9. Решение СЛАУ методом Крамера (пример 5.8).

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

Пример 5.9. Решить систему линейных уравнений 5.6 из примера 5.8 методом обратной матрицы.

Метод обратной матрицы: для системы из n линейных уравнений с n неизвестными $Ax = b$, при условии, что определитель матрицы A не равен нулю, единственное решение можно представить в виде $x = A^{-1}b$ (вывод формулы см. в примере 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 5.10).

```

>>> 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=0
ans =
    -5.3291e-15
     1.0000e+00
     4.0000e+00

```

Листинг 5.10. Решение СЛАУ примера 5.8 методом обратной матрицы (пример 5.9).

Пример 5.10. Решить систему линейных уравнений

$$\begin{cases} 2x_1 + x_2 - 5x_3 + x_4 = 8 \\ x_1 - 3x_2 - 6x_4 = 9 \\ \quad 2x_2 - x_3 + 2x_4 = -5 \\ x_1 + 4x_2 - 7x_3 + 6x_4 = 0 \end{cases} \quad (5.7)$$

методом Гаусса.

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

Метод Гаусса состоит из двух этапов. Первый этап — это *прямой ход*, в результате которого расширенная матрица системы путём элементарных преобразований (перестановка уравнений системы, умножение уравнений на число отличное от нуля и сложение уравнений) приводится к ступенчатому виду:

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

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

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

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

Исходя из выше изложенного, порядок решения задачи в **Octave** (листинг 5.11) 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 $Ax - b$, если в результате получился нулевой вектор, задача решена верно.

```
>>> disp('Решение СЛАУ методом Гаусса');
>>> disp('Матрица системы:'); A=[21 - 51; 1 - 30 - 6; 02 - 12; 14 - 76]
>>> 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 5.11. Решение СЛАУ методом Гаусса (пример 5.10).

Пример 5.11. Решить систему линейных уравнений из примера 5.10 с помощью *LU*-разложения.

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

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

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

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

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

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

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

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

Исходя из средств, которыми располагает **Octave**, решение поставленной задачи будет выглядеть так (листинг 5.12):

- 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);
- поскольку в задаче речь идёт о решении системы, то элементы вектора b должны занять места соответствующие строкам матрицы A , для чего необходимо выполнить действие Pb ;
- решим системы уравнений $Ly = b$ относительно y ;
- зная U и y , найдём решение системы $Ux = y$.


```

>>> disp('Решение СЛАУ методом LU-разложения');
>>> disp('Матрица системы:'); A=[21 - 51; 1 - 30 - 6; 02 - 12; 14 - 76]
>>> 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 =
Перmutation 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 5.12. Решение СЛАУ методом LU-разложения (пример 5.11).

Пример 5.12. Решить систему линейных уравнений

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

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:

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

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

Как решить эту задачу средствами **Octave**, показано в листинге 5.13.

```

>>> 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 5.13. Решение СЛАУ методом QR-разложения (пример 5.12).

Пример 5.13. Исследовать систему на совместность и если возможно решить её:

$$\begin{aligned} \text{а) } & \begin{cases} x_1 + 2x_2 + 2x_3 = -9 \\ x_1 - x_2 + x_3 = 2 \\ 3x_1 - 6x_2 - x_3 = 25 \end{cases} \\ \text{б) } & \begin{cases} x_1 - 5x_2 - 8x_3 + x_4 = 3 \\ 3x_1 + x_2 - 3x_3 - 5x_4 = 1 \\ x_1 - 7x_2 + 2x_4 = -5 \\ 11x_2 + 20x_3 - 9x_4 = 2 \end{cases} \\ \text{в) } & \begin{cases} 4x_1 + x_2 - 3x_3 - x_4 = 0 \\ 2x_1 + 3x_2 + x_3 - 5x_4 = 0 \\ x_1 - 2x_2 - 2x_3 + 3x_4 = 0 \end{cases} \end{aligned}$$

Для решения задачи (листинг 5.14) введём 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 5.14. Исследование системы на совместность (пример 5.13).

5.7 Собственные значения и собственные векторы

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

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

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

о собственных значениях. Для каждого корня \hbar кратности m существует ровно m линейно независимых собственных векторов y .

Пример 5.14. Найти собственные значения и собственные векторы матрицы A .

В листинге 5.15 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 5.15. Нахождение собственных значений (пример 5.14).

Пример 5.15. Привести заданную матрицу к диагональному виду.

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

В листинге 5.16 приведено решение поставленной задачи.

```
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 5.16. Приведение к диагональному виду (пример 5.15).

Пример 5.16. Найти решение обобщённой задачи о собственных значениях для матриц A и B .

Обобщённую задачу о собственных значениях (листинг 5.17) решают при помощи функции $eig(A, B)$, которая в качестве результата выдаёт матрицу обобщённых собственных векторов и диагональную матрицу, содержащую обобщённые собственные значения.

```
disp('Введите матрицу A:'); A=input('A=');
disp('Введите матрицу B:'); B=input('B=');
[X,V]=eig(A,B);
disp('Матрица обобщённых собственных векторов:'); X
disp('Матрица обобщённых собственных значений:'); V
disp('Обобщённые собственные значения:'); v=diag(V)
%
Введите матрицу A:
A= [1 -3;-3 4]
```


Введите матрицу B:

B= [1 2; -3 1]

Матрица обобщённых собственных векторов:

X =

1.00 0.92

0.00 1.00

Матрица, содержащая обобщённые собственные значения:

V =

Diagonal Matrix

1.00 0

0 -0.71

Обобщённые собственные значения:

v =

1.00

-0.71

Листинг 5.17. Обобщённая задача о собственных значениях (пр. 5.16).

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:

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

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

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

$$\|A\|_2 = \sqrt{\lambda_{\max}(AA^T)}, \text{ где } \sqrt{\lambda_{\max}(AA^T)} - \text{максимальное собственное значение матрицы } A = \{a_{ij}\};$$

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

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

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

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

Число обусловленности матрицы A используется для определения меры чувствительности системы линейных уравнений $Ax = b$ к погрешностям задания вектора b . Чем больше число обусловленности, тем более неустойчив процесс нахождения решения системы. Существует несколько вариантов вычисления числа обусловленности, но

все они связаны с нормой матрицы, и равны произведению нормы исходной матрицы на норму обратной:

- число обусловленности матрицы, вычисленное в норме $\|A\|_1$:
 $N = \|A\|_1 \cdot \|A^{-1}\|_1$;
- число обусловленности матрицы, вычисленное в норме $\|A\|_2$:
 $M = \|A\|_2 \cdot \|A^{-1}\|_2$;
- число обусловленности матрицы, вычисленное в норме $\|A\|_i$:
 $P = \|A\|_i \cdot \|A^{-1}\|_i$;
- число обусловленности матрицы, вычисленное в норме $\|A\|_e$:
 $H = \|A\|_e \cdot \|A^{-1}\|_e$.

Пример 5.17. Вычислить нормы и числа обусловленности матрицы A .

В листинге 5.18 приведён фрагмент документа, в котором происходит вычисление норм матрицы A с помощью функции `norm` и по соответствующим формулам. Вычисление чисел обусловленности проведено при помощи функции `cond(A)` и по формулам, отражающим зависимость числа обусловленности от соответствующей нормы матрицы.

```
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 5.18. Вычисление матричных норм (пример 5.17).

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 *символьной матрицы* служит функция `ex_matrix`(число строк, число столбцов, элементы матрицы).

Например:

```
>>> symbols
>>> a = sym ("a"); % Определение символьных переменных
>>> b = sym ("b");
>>> c = sym ("c");
>>> d = sym ("d");
>>> Matr=ex_matrix(1,3,a,b,c) % Матрица строка
Matr = [[a,b,c]]
>>> Matr=ex_matrix(4,1,a,b,c,d) % Матрица столбец
Matr = [[a],[b],[c],[d]]
>>> Matr=ex_matrix(2,2,a,b,c,d) % Матрица 2 на 2
Matr = [[a,b],[c,d]]
>>> Matr=ex_matrix(3,3,a,0,b,c,1,1,d,0,2) % Матрица 3 на 3
Matr = [[a,0.0,b],[c,1.0,1.0],[d,0.0,2.0]]
```

Над символьными матрицами определены операции сложения, вычитания, умножения.

Пример 5.18. Выполнить действия над матрицами (листинг 5.19).

```
>>> symbols
>>> a = sym ("a"); % Определение символьных переменных
>>> 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 5.19. Действия над символьными матрицами (пример 5.18).

К сожалению над символьными матрицами не определены операции вычисления определителя и обратной матрицы.

Для решения *системы линейных алгебраических уравнений* можно воспользоваться функцией *symsolve*.

Пример 5.19. Решить систему линейных алгебраических уравнений $\begin{cases} ax + by = c \\ x + y = d \end{cases}$ относительно переменных x и y (листинг 5.20).

```
>>> x = sym ("x");
>>> y = sym ("y");
>>> a = sym ("a");
>>> b = sym ("b");
>>> c = sym ("c");
>>> d = sym ("d");
>>> sols = symsolve({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 5.20. Решение СЛАУ в символьном виде (пример 5.19).

Пример 5.20. Решить систему линейных алгебраических уравнений (листинг 5.21)

$$\begin{cases} x_1 + 2x_2 + 5x_3 = -9 \\ x_1 - x_2 + 3x_3 = 2 \\ 3x_1 - 6x_2 - x_3 = 25. \end{cases}$$

```
>>> x1 = sym ("x1");
>>> x2 = sym ("x2");
>>> x3 = sym ("x3");
>>> sols = symsolve({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 5.21. Решение СЛАУ используя *symsolve* (пример 5.20).

Глава 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*.

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:

$$|\vec{a}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{X^2 + Y^2}$$

Формулы

$$M_x = \frac{x_1 + x_2}{2}, M_y = \frac{y_1 + y_2}{2}$$

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 путём поэлементного ввода:

```
>>> a=[1 0 3] % Вектор-строка
a = 1 0 3
>>> b=[0;1;4] % Вектор-столбец
b =
0
1
4
```

Пример 6.1. Построить вектор $|\vec{a}| = \{5, 7\}$.

Решение примера показано на рис. 6.1. Листинг 6.1 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 6.1. Построение вектора (пример 6.1).

Пример 6.2. Построить векторы, заданные координатами начала и конца:

$$\vec{a} = \{(2, 3), (4, 6)\}, \quad \vec{b} = \{(9, 7), (6, 5)\}, \quad \vec{c} = \{(1, 8), (4, 8)\}, \\ \vec{d} = \{(6, 7), (6, 9)\}, \quad \vec{k} = \{(8, 4), (8, 1)\}, \quad \vec{p} = \{(7, 3), (5, 3)\}.$$

Решение примера показано в листинге 6.2 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

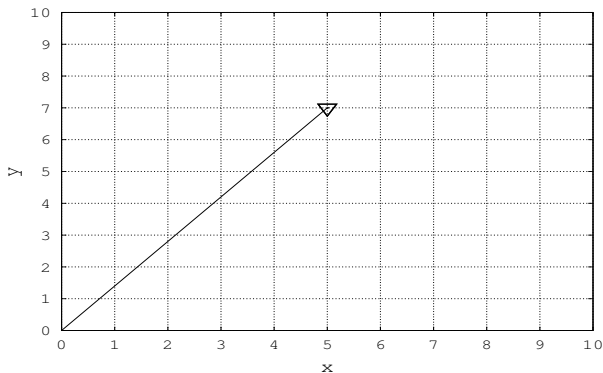


Рис. 6.1. Вектор на плоскости

его середины. В данном случае координаты середины отрезка нужны для нанесения соответствующей надписи, обозначающей вектор на рисунке.

```

clear all;
% Функция рисует направленный отрезок АВ, в качестве результата выдаёт
% координаты середины отрезка АВ.
function [M]=vector (A,B)
    x1=A(1); x2=B(1); y1=A(2); y2=B(2);
    alf=30*pi/180;% Угол в вершине стрелки в радианах
    L=15;% Деление отрезка в заданном отношении
    xm=(x1+L*x2)/(1+L); ym=(y1+L*y2)/(1+L);
    k1=(y2-y1)/(x2-x1); % Угол наклона прямой АВ
    if (k1==Inf)|(k1==inf) % Отрезок перпендикулярен оси Ох
        % Координаты основания треугольника, образующего стрелку
        x4=xm-0.2; y4=ym; x3=xm+0.2; y3=ym;
    elseif k1==0 % Отрезок перпендикулярен оси Оу
        x4=xm; y4=ym-0.2; x3=xm; y3=ym+0.2;
    else
        % Уравнение прямой АВ
        k1=(y2-y1)/(x2-x1); m1=y1-x1*(y2-y1)/(x2-x1);
        % Уравнение прямой перпендикулярной АВ
        k3=-1/k1; m3=1/k1*xm+ym;
    % Уравнение прямой, проходящей через точку В под углом alf к прямой АВ
        k2=(-k1-tan(alf))/(tan(alf)*k1-1); m2=y2-k2*x2;
    % Уравнение прямой, проходящей через точку В под углом -alf к прямой АВ
        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;

```

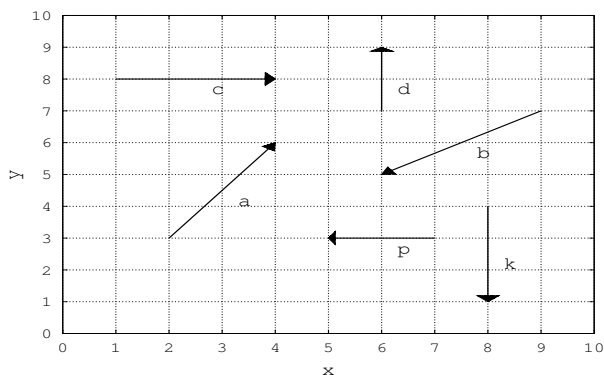



Рис. 6.2. Геометрическое решение примера 6.2

```

end;
% 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');
ma=vector([2,3],[4,6]); % Построение вектора a
T=text(ma(1)+0.3,ma(2)-0.3,'a'); set(T,'FontSize',20)
mb=vector([9,7],[6,5]); % Построение вектора b
T=text(mb(1)+0.3,mb(2)-0.3,'b'); set(T,'FontSize',20)
mc=vector([1,8],[4,8]); % Построение вектора c
T=text(mc(1)+0.3,mc(2)-0.3,'c'); set(T,'FontSize',20)
md=vector([6,7],[6,9]); % Построение вектора d
T=text(md(1)+0.3,md(2)-0.3,'d'); set(T,'FontSize',20)
mk=vector([8,4],[8,1]); % Построение вектора k
T=text(mk(1)+0.3,mk(2)-0.3,'k'); set(T,'FontSize',20)
mp=vector([7,3],[5,3]); % Построение вектора p
T=text(mp(1)+0.3,mp(2)-0.3,'p'); set(T,'FontSize',20)

```

Листинг 6.2. Построение векторов, функция *vector* (пример 6.2).

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*.

Пример 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 6.3. При решении примера была создана функция $dlina(X, Y)$, которая вычисляет длину отрезка \mathbf{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 примере 6.2.

Решение примера показано в конце листинга 6.3 и на рис. 6.3. По полученным числовым результатам и геометрической интерпретации примера 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.

```
function d=dlina(X,Y) % Функция возвращает длину отрезка XY
    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') ; set (A_,'FontSize',20)
B_=text (B(1)+0.3,B(2)-0.3,'B') ; set (B_,'FontSize',20)
% Построение вектора CD
vector (C,D) ;
C_=text (C(1)+0.3,C(2)-0.3,'C') ; set (C_,'FontSize',20)
D_=text (D(1)+0.3,D(2)-0.3,'D') ; set (D_,'FontSize',20)
% Построение вектора OM
vector (O,M) ;
O_=text (O(1)+0.3,O(2)-0.3,'O') ; set (O_,'FontSize',20)
M_=text (M(1)+0.3,M(2)-0.3,'M') ; set (M_,'FontSize',20)
% Построение вектора ON
vector (O,N) ;
O_=text (O(1)+0.3,O(2)-0.3,'O') ; set (O_,'FontSize',20)
N_=text (N(1)+0.3,N(2)-0.3,'N') ; set (N_,'FontSize',20)
% Построение вектора KL
vector (K,L) ;
K_=text (K(1)+0.3,K(2)-0.3,'K') ; set (K_,'FontSize',20)
L_=text (L(1)+0.3,L(2)-0.3,'L') ; set (L_,'FontSize',20)
% Построение вектора PR
vector (P,R) ;
P_=text (P(1)+0.3,P(2)-0.3,'P') ; set (P_,'FontSize',20)
R_=text (R(1)+0.3,R(2)-0.3,'R') ; set (R_,'FontSize',20)
% Построение вектора UV
vector (U,V) ;
U_=text (U(1)+0.3,U(2)-0.3,'U') ; set (U_,'FontSize',20)
V_=text (V(1)+0.3,V(2)-0.3,'V') ; set (V_,'FontSize',20)
% Решение примера 6.3
% Длины отрезков:
AB = 3.6056
CD = 3.6056
ON = 3.1623
OM = 3.1623
KL = 3.1623
PR = 3
UV = 3

```

Листинг 6.3. Сравнение векторов (пример 6.3).

Пример 6.4. Проверить коллинеарны ли векторы \overrightarrow{AB} и \overrightarrow{CD} , \overrightarrow{NM} и \overrightarrow{KL} , \overrightarrow{PR} и \overrightarrow{UV} . Координаты точек: $A(4, 2)$, $B(2, 3)$, $C(3, 2)$, $D(7, 0)$, $M(2, 1)$, $N(6, 1)$, $K(1, 2)$, $L(7, 2)$, $P(3, 3)$, $R(5, 3)$, $U(1, 4)$, $V(5, 4)$.

Текст файла-сценария представлен в листинге 6.4. При решении примера была создана функция $kollin(a, b)$, которая определяет кол-

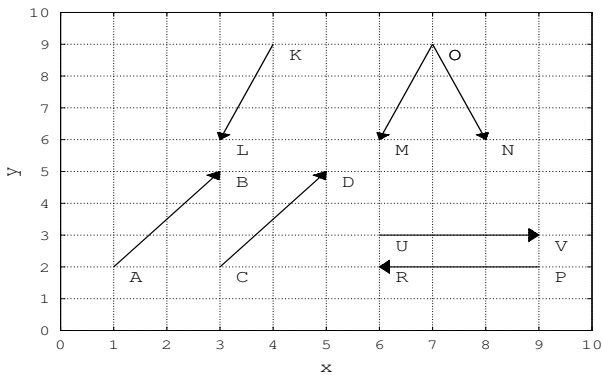


Рис. 6.3. Изображение векторов примера 6.3

линейны 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 листинге 6.2 к примеру 6.2.

Решение примера показано в конце листинга 6.4 и на рис. 6.4. По полученным числовым результатам и геометрической интерпретации примера 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.

```
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)
vector(A,B); % Построение вектора AB
A_ =text(A(1), A(2)+0.3, 'A'); set(A_, 'FontSize', 20)
B_ =text(B(1), B(2)+0.3, 'B'); set(B_, 'FontSize', 20)
vector(C,D); % Построение вектора CD
C_ =text(C(1), C(2)+0.3, 'C'); set(C_, 'FontSize', 20)
D_ =text(D(1), D(2)+0.3, 'D'); set(D_, 'FontSize', 20)
vector(N,M); % Построение вектора NM
N_ =text(N(1), N(2)+0.3, 'N'); set(N_, 'FontSize', 20)
M_ =text(M(1), M(2)+0.3, 'M'); set(M_, 'FontSize', 20)
vector(K,L); % Построение вектора KL
K_ =text(K(1)+0.3, K(2)-0.3, 'K'); set(K_, 'FontSize', 20)
L_ =text(L(1)+0.3, L(2)-0.3, 'L'); set(L_, 'FontSize', 20)
vector(P,R); % Построение вектора PR
P_ =text(P(1)+0.3, P(2)-0.3, 'P'); set(P_, 'FontSize', 20)
R_ =text(R(1)+0.3, R(2)-0.3, 'R'); set(R_, 'FontSize', 20)
vector(U,V); % Построение вектора UV
U_ =text(U(1)+0.3, U(2)-0.3, 'U'); set(U_, 'FontSize', 20)
V_ =text(V(1)+0.3, V(2)-0.3, 'V'); set(V_, 'FontSize', 20)
% Результаты — коэффициенты пропорциональности векторов
AB_CD = -0.50000
MN_KL = Inf
PR_UV = 0.50000

```

Листинг 6.4. Проверка коллинеарности векторов (пример 6.4).

Геометрической проекций вектора \overrightarrow{AB} на ось \mathbf{OX} называется вектор $\overrightarrow{A'B'}$, начало которого A' есть проекция точки A на ось \mathbf{OX} , а конец B' — проекция точки B на ту же ось. Обозначается: $\overrightarrow{A'B'} = \text{Пр}_{\mathbf{OX}} \overrightarrow{AB}$. Алгебраической проекций вектора \overrightarrow{AB} на ось \mathbf{OX} называется длина вектора $\overrightarrow{A'B'}$, взятая со знаком «+», если направление вектора \overrightarrow{AB} совпадает с направлением оси \mathbf{OX} , или со знаком «−» в противном случае.

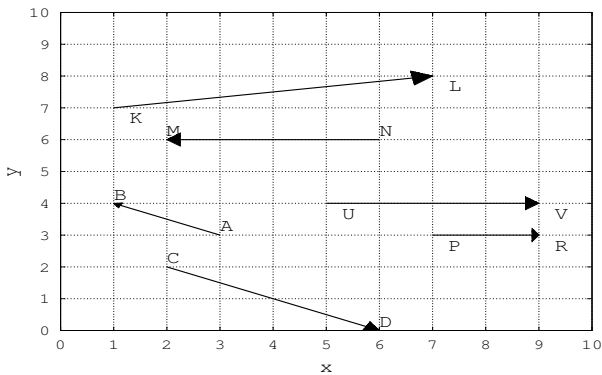
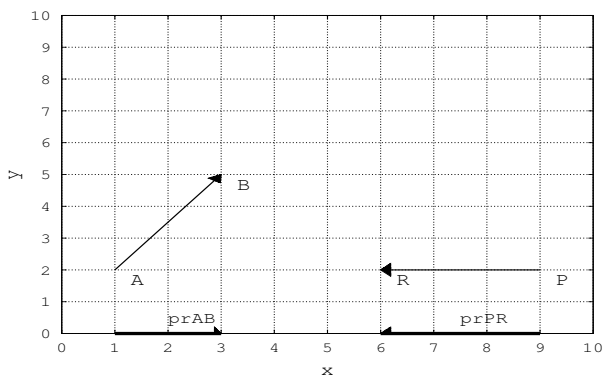


Рис. 6.4. Геометрическая интерпретация примера 6.4

Пример 6.5. Найти проекции векторов \overrightarrow{AB} и \overrightarrow{PR} на ось OX . Координаты точек, задающих векторы: $A(1, 2)$, $B(3, 5)$, $P(9, 2)$, $R(6, 2)$.

Текст файла-сценария и значения алгебраических проекций векторов представлены в листинге 6.5. При решении примера 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 примере 6.2. Геометрические проекции показаны на рис. 6.5.

```
function pr=pr_OX(X) % Длина проекции вектора на ось OX
    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)])
```

Рис. 6.5. Проекция векторов на ось **OX**

```

vector(A,B);% Построение вектора AB
A_=text(A(1)+0.3,A(2)-0.3,'A');set(A_,'FontSize',20)
B_=text(B(1)+0.3,B(2)-0.3,'B');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);
vector(P,R); % Построение вектора PR
P_=text(P(1)+0.3,P(2)-0.3,'P');set(P_,'FontSize',20)
R_=text(R(1)+0.3,R(2)-0.3,'R');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);
% Вычисление проекций
prAB = 2
prPR = -3

```

Листинг 6.5. Нахождение проекций векторов (пример 6.5).

Любые 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: $\lambda \vec{a} = \{\lambda a_1, \lambda a_2\}$, $\frac{\vec{a}}{\lambda} = \{\frac{a_1}{\lambda}, \frac{a_2}{\lambda}\}$.

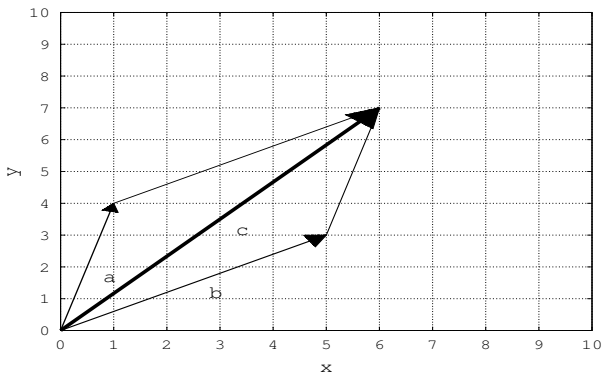


Рис. 6.6. Геометрическое представление суммы векторов

Пример 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 6.7 содержит команды **Octave**, с помощью которых был решён пример и результаты их работы. Функция $vector(A, B)$, описана в примере 6.2. Геометрическое решение примера показано на рис. 6.6.

```
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];
c=[a(1)+b(1), a(2)+b(2)] % Сумма векторов
ma=vector([0, 0], a); % Построение вектора a
text(ma(1)+0.3, ma(2)-0.3, 'a', 'FontSize', 20);
mb=vector([0, 0], b); % Построение вектора b
text(mb(1)+0.3, mb(2)-0.3, 'b', 'FontSize', 20);
mc=vector([0, 0], c); % Построение вектора c=a+b
text(mc(1)+0.3, mc(2)-0.3, 'c', 'FontSize', 20);
```

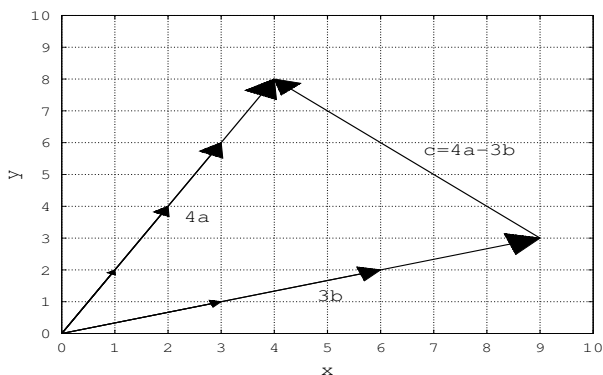



Рис. 6.7. Геометрическое представление действий над векторами

```

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 6.6. Нахождение суммы векторов (пример 6.6).

Пример 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*).

Листинг 6.7 содержит команды **Octave** и результаты работы файла-сценария. Функция $vector(A, B)$, описана в задаче 6.2. Решение примера показано на рис. 6.7.

```

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];
c=[4*a(1)-3*b(1), 4*a(2)-3*b(2)] % Действия над векторами

```

```

for i=1:4 % Построение вектора 4a
    ma=vector([0,0],i*a);
end;
text(ma(1)+0.3,ma(2)-0.3,'4a','FontSize',20);
for i=1:3 % Построение вектора 3b
    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 6.7. Действия над векторами (пример 6.7).

Пример 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 формулам¹: $\cos(\alpha) = \frac{a_1}{|\vec{a}|}$, $\cos(\beta) = \frac{a_2}{|\vec{a}|}$, $\cos(\gamma) = \frac{a_3}{|\vec{a}|}$.

Листинг 6.8 содержит команды **Octave** и результаты работы файла-сценария.

```

function [U]=ugol(X) % Углы, образуемые осями координат с вектором X
    m=sqrt(X(1)^2+X(2)^2+X(3)^2);
    U=acos(X/m);
end;
function gr=rad_gr(rad) % Перевод радиан в градусы и минуты
    gr(1)=round(rad*180/pi); % Градусы
    gr(2)=round((rad*180/pi-gr(1))*60); % Минуты
end;
u=ugol([2,-2,-1]) % Вычисление углов в радианах
% 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 6.8. Нахождение угла между вектором и осями (пример 6.8).

¹ a_i — проекция вектора на соответствующую ось. (Прим. редактора).

Скалярным произведением вектора \vec{a} на вектор \vec{b} называется произведение их модулей и косинуса угла между ними: $\vec{a}\vec{b} = |\vec{a}| \cdot |\vec{b}| \cos(\widehat{a, b})$. Если $\vec{a} = \{a_1, a_2, a_3\}$ и $\vec{b} = \{b_1, b_2, b_3\}$, то $\vec{a}\vec{b} = a_1b_1 + a_2b_2 + a_3b_3$.

Пример 6.9. Найти угол между векторами $\vec{a} = \{-2, 1, 2\}$ и $\vec{b} = \{-2, -2, 1\}$: $\cos(\widehat{a, b}) = \frac{\vec{a}\vec{b}}{|\vec{a}| \cdot |\vec{b}|}$.

Текст файла-сценария и результат его работы представлены в листинге 6.9.

```
function gr=rad_gr(rad) % Перевод радиан в градусы и минуты
    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 6.9. Нахождение угла между векторами (пример 6.9).

Пример 6.10. Проверить, являются ли векторы \overrightarrow{NM} и \overrightarrow{KL} , \overrightarrow{PR} и \overrightarrow{UV} взаимно перпендикулярными. Координаты точек: $M(2, 1)$, $N(6, 1)$, $K(1, 2)$, $L(7, 2)$, $P(3, 3)$, $R(5, 3)$, $U(1, 4)$, $V(5, 4)$.

Если $\vec{a} = \{a_1, a_2\}$ и $\vec{b} = \{b_1, b_2\}$ перпендикулярны, то их скалярное произведение равно нулю: $\vec{a}\vec{b} = a_1b_1 + a_2b_2 = 0$. 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 6.10).

```
% 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]);
```

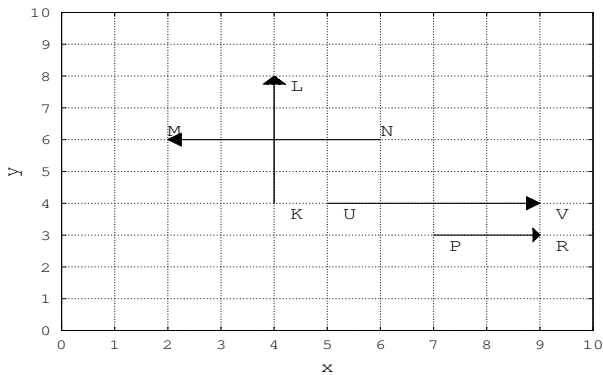


Рис. 6.8. Графическое решение примера 6.10

```

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)
vector(N,M); % Построение вектора NM
N_=text(N(1),N(2)+0.3,'N');set(N_,'FontSize',20)
M_=text(M(1),M(2)+0.3,'M');set(M_,'FontSize',20)
vector(K,L); % Построение вектора KL
K_=text(K(1)+0.3,K(2)-0.3,'K');set(K_,'FontSize',20)
L_=text(L(1)+0.3,L(2)-0.3,'L');set(L_,'FontSize',20)
% Построение вектора PR
vector(P,R); % Построение вектора PR
P_=text(P(1)+0.3,P(2)-0.3,'P');set(P_,'FontSize',20)
R_=text(R(1)+0.3,R(2)-0.3,'R');set(R_,'FontSize',20)
vector(U,V); % Построение вектора UV
U_=text(U(1)+0.3,U(2)-0.3,'U');set(U_,'FontSize',20)
V_=text(V(1)+0.3,V(2)-0.3,'V');set(V_,'FontSize',20)
% Результат работы программы
MN_KL = 0
PR_UV = 8

```

Листинг 6.10. Проверка перпендикулярности векторов (пример 6.10).

Векторным произведением вектора \vec{a} на не коллинеарный с ним вектор \vec{b} называется вектор \vec{c} , модуль которого численно равен пло-

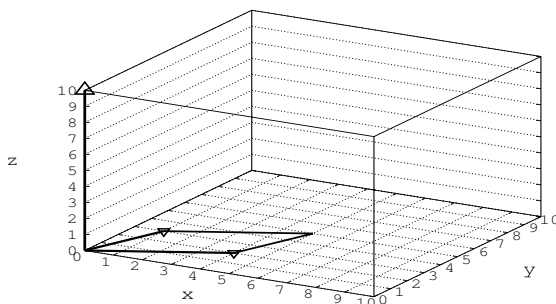


Рис. 6.9. Геометрическая интерпретация векторного произведения

щади параллелограмма построенного на векторах \vec{a} и \vec{b} : $\vec{a} \times \vec{b} = |\vec{a}| \cdot |\vec{b}| \sin(\widehat{a, b})$. Направление вектора \vec{c} перпендикулярно плоскости параллелограмма. Если векторы $\vec{a} = \{x_1, y_1, z_1\}$ и $\vec{b} = \{x_2, y_2, z_2\}$, то $\vec{a} \times \vec{b} = \left\{ \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix}, \begin{vmatrix} z_1 & x_1 \\ z_2 & x_2 \end{vmatrix}, \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} \right\}$.

Пример 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 6.11 содержит текст программы и результаты её работы.

```
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;
function D=dlin(x) % Длина вектора
    D=(x(1)^2+x(2)^2+x(3)^2)^(1/2);
end;
function gr=rad_gr(rad) % Перевод радиан в градусы и минуты
    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];
% Расчёт 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));
% Расчёт 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);
% Расчёт 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');
c % Координаты векторного произведения
alfa=rad_gr(alf) % Угол между векторами a и b
% Результаты работы программы — координаты векторного произведения
c =      0      -0      10
% Угол между векторами
alfa =      45      -0

```

Листинг 6.11. Векторное произведение векторов (пример 6.11).

Три вектора называют *компланарными*, если они, будучи приведены к общему началу, лежат в одной плоскости. *Смешанным* или *векторно-скалярным произведением* трёх 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 $\vec{a} = \{x_1, y_1, z_1\}$, $\vec{b} = \{x_2, y_2, z_2\}$ и $\vec{c} = \{x_3, y_3, z_3\}$ даны своими координатами, то смешанное произведение вычисляют по формуле:

$$\overrightarrow{abc} = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}.$$

Необходимым и достаточным *условием компланарности векторов* \vec{a} , \vec{b} и \vec{c} является равенство нулю их смешанного произведения $\vec{a}\vec{b}\vec{c} = 0$.

Пример 6.12. Проверить компланарность векторов (листинг 6.12):

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 6.12. Проверка компланарности векторов (пример 6.12).

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* $A(x - x_0) + B(y - y_0) + C(z - z_0) = 0$ или $Ax + By + Cz + D = 0$.

Вектор $\vec{N}\{A, B, C\}$ называется *нормальным вектором* плоскости.

Пример 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 примера 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:

$$Ax + By + Cz + D = 0 \Rightarrow z(x, y) = -\frac{A}{C}x - \frac{B}{C}y - \frac{D}{C},$$

$$z(x, y) = b_0 + b_1x + b_2y, b_0 = -\frac{A}{C}, b_1 = -\frac{B}{C}, b_2 = -\frac{D}{C}.$$

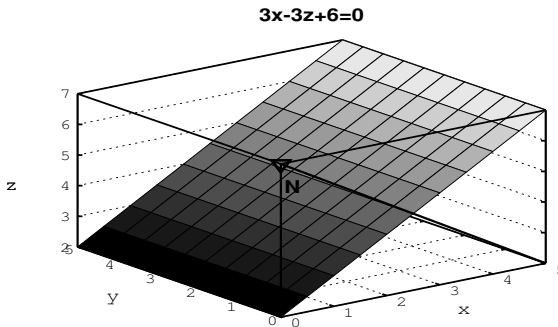


Рис. 6.10. Изображение плоскости и её направляющего вектора

Решение примера представлено в листинге 6.13 и на рис. 6.10

```
clf; cla;
set(gcf, 'Position', [20, 20, 400, 400]);
axis([0, 10, 0, 10, 0, 10])
A=3; B=0; C=-3; D=6; N=[A, B, C]; % Параметры плоскости
b0=-D/C; b1=-A/C; b2=-B/C; % Параметры уравнения плоскости, преобразо-
                             % ванного к функции двух переменных

% 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);
title('3x-3z+6=0', 'FontSize', 20) % Заголовок
```

Листинг 6.13. Построение плоскости по точке и вектору (пример 6.13).

Если плоскость проходит через заданную точку $M_1(x_1, y_1, z_1)$ и параллельна плоскости $Ax + By + Cz + D = 0$, то её уравнение записывают так $A(x - x_1) + B(y - y_1) + C(z - z_1) = 0$.

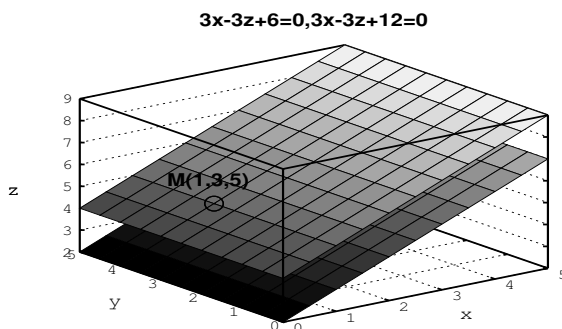


Рис. 6.11. Параллельные плоскости

Пример 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 примера показано на рис. 6.11 и в листинге 6.14.

```
function p=plos(A,B,C,D)
% Параметры уравнения плоскости, преобразованного
b0=-D/C; b1=-A/C; b2=-B/C; % к функции двух переменных
% 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])
A1=3;B1=0;C1=-3;D1=6; % Параметры плоскости
% 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');
M=[1,3,5]; % Изображение точки
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 6.14. Построение плоскости по параллельной ей плоскости и точке (пример 6.14).

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:

$$\begin{vmatrix} x - x_0 & y - y_0 & z - z_0 \\ x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \end{vmatrix} = 0.$$

Пример 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 иллюстрируют это утверждение. Изображение на рис. 6.12 получено в результате работы программы показанной в листинге 6.15. Рис. 6.13 получен из рис. 6.12 путём поворота² в графическом окне.

```

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;
M0=[1,2,3]; M1=[2,1,2]; M2=[3,3,1]; % Исходные данные
% 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);

```

² Аналитически поворот можно осуществить с помощью функции `view`, например, `view(37.5,30)`; вернуть рисунок в исходное состояние можно командой `view(2)` (прим. редактора).

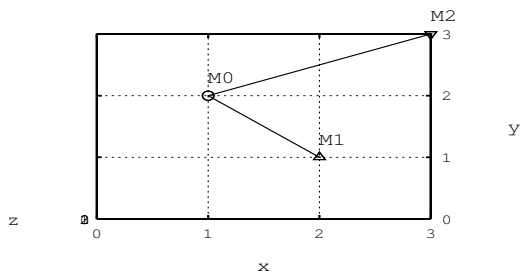


Рис. 6.12. Иллюстрация к примеру 6.15 (векторы на плоскости)

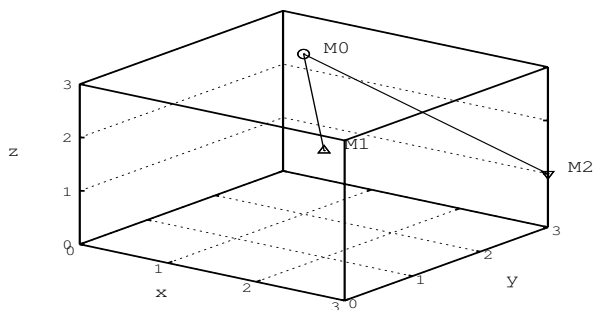


Рис. 6.13. Иллюстрация к примеру 6.15 (векторы в пространстве)

```

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 6.15. Построение векторов по трём точкам (пример 6.15).

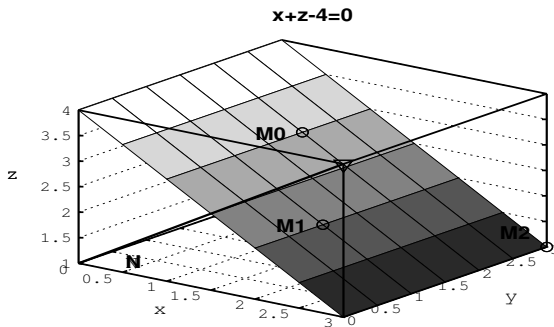


Рис. 6.14. Плоскость, проходящая через три точки

Плоскость представлена уравнением:

$$\begin{vmatrix} x-1 & y-2 & z-3 \\ 1 & -1 & -1 \\ 2 & 1 & -2 \end{vmatrix} = 0 \Rightarrow x + z - 4 = 0.$$

Графическое изображение плоскости показано на рис. 6.14. Текст программы в листинге 6.16.

```

clf; cla;
set(gcf, 'Position', [20, 20, 400, 400]); axis([0, 10, 0, 10, 0, 10])
A=1;B=0;C=1;D=-4;N=[A,B,C]; % Параметры плоскости
% 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 6.16. Построение плоскости по трём точкам (пример 6.15).

Если плоскость $Ax + By + Cz + D = 0$ не параллельна оси \mathbf{OX} ($A \neq 0$), то она отсекает на этой оси отрезок $a = -\frac{D}{A}$. Аналогично отрезки на осях \mathbf{OY} , \mathbf{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*.

Пример 6.16. Написать уравнение плоскости $3x - 6y + 2z - 12 = 0$ в отрезках и построить эту плоскость.

Найдём длины отрезков: $a = -\frac{D}{A} = -\frac{-12}{3} = 4$, $b = -\frac{D}{B} = -\frac{-12}{-6} = -2$, $c = -\frac{D}{C} = -\frac{-12}{2} = 6$.

Запишем уравнение плоскости в отрезках:

$$\frac{x}{4} + \frac{y}{-2} + \frac{z}{6} = 1.$$

Решение примера показано на рис. 6.15 и в листинге 6.17.

```

clf; cla;
a=4;b=-2;c=6;
set(gcf,'Position',[50,50,400,400]);
axis([0,a,0,b,0,c]);
X=0:0.2:a;Y=b:0.2:0;
[x y]=meshgrid(X,Y);

```

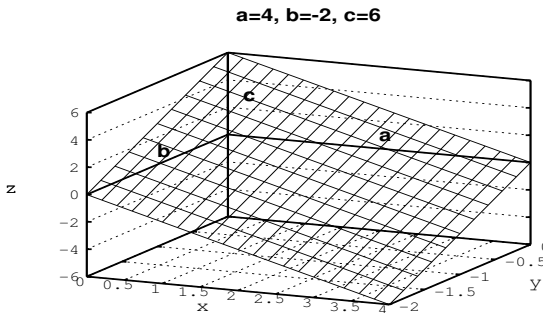


Рис. 6.15. Плоскость, заданная отрезками

```

z=-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)
xlabel('x');ylabel('y');zlabel('z');
set(gca,'Position',[.1,.1,.8,.8]);
set(gca,'View',[25 22])

```

Листинг 6.17. Уравнение плоскости в отрезках (пример 6.16).

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 $Bu + D = 0$, ($A = 0, C = 0$) — плоскость, параллельную плоскости **XOZ**, уравнение $CZ + D = 0$, ($A = 0, B = 0$) — плоскость, параллельную плоскости **XOY**.
- Уравнения $X = 0$, $Y = 0$, $Z = 0$ представляют собой плоскости **YOZ**, **XOZ** и **XOY**.

Пример 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 примера описан в листинге 6.18. Графическое решение показано на рис. 6.16.

```
function p=plosl(A,B,C,D)
M=[A,B,C]; % Параметры плоскости
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 % Плоскость не параллельна OZ
% Уравнение плоскости преобразовано 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 % Плоскость не параллельна OY
% Уравнение плоскости преобразовано 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 % Плоскость не параллельна OX
% Уравнение плоскости преобразовано 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(2)]);
```

```

    set(gca, 'ztick', [0:M(3)]); set(gca, 'box', 'on');
    p=f1;
end;% конец функции
% Изображение плоскостей заданных в примере 6.17
clf; cla; subplot(3,2,1);
A1=1;B1=1;C1=0;D1=-1; % Плоскость x+y-1=0
plot(A1,B1,C1,D1);
title('x+y-1=0, (C=0)');
subplot(3,2,2);
A2=1;B2=0;C2=-1;D2=1; % Плоскость x-z+1=0
plot(A2,B2,C2,D2);
title('x-z+1=0, (B=0)');
subplot(3,2,3);
A3=0;B3=1;C3=1;D3=2; % Плоскость y+z+2=0
plot(A3,B3,C3,D3);
title('y+z+2=0, (A=0)');
set(gca, 'View', [130 30]);
subplot(3,2,4);
A4=1;B4=-1;C4=1;D4=0; % Плоскость x-y+z-2=0
plot(A4,B4,C4,D4);
set(gca, 'View', [40 30]);
title('x-y+z=0, D=0');
subplot(3,2,5);
A5=2;B5=0;C5=0;D5=3; % Плоскость 2x+3=0
plot(A5,B5,C5,D5);
set(gca, 'View', [40 30]);
title('2x+3=0, (B=0,C=0)');
subplot(3,2,6);
A6=0;B6=3;C6=0;D6=-2; % Плоскость 3y-2=0
plot(A6,B6,C6,D6);
set(gca, 'View', [40 30]);
title('3y-2=0, (A=0,C=0)');

```

Листинг 6.18. Построение нескольких плоскостей (пример 6.17).

Расстояние от точки $M_1(x_1, y_1, z_1)$ до плоскости $Ax + By + Cz + D = 0$ равно абсолютному значению величины

$$d = \frac{|Ax_1 + By_1 + Cz_1 + D|}{\sqrt{A^2 + B^2 + C^2}}.$$

Пример 6.18. Найти расстояние от точки $M_1(3, 9, 1)$ до плоскости $x - 2y + 2z + 3 = 0$.

Решение показано в листинге 6.19.

```

% 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

```

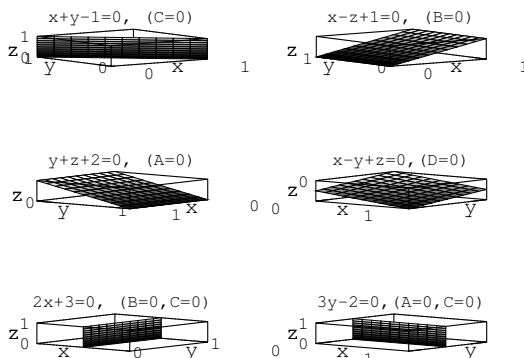



Рис. 6.16. Особые случаи положения плоскости относительно системы координат

```
d=abs(M*N+D)/norm(N)
d = 5.3333
```

Листинг 6.19. Вычисление расстояния от точки до плоскости.

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

$$\cos(\phi) = \pm \frac{A_1A_2 + B_1B_2 + C_1C_2}{\sqrt{A_1^2 + B_1^2 + C_1^2} \sqrt{A_2^2 + B_2^2 + C_2^2}},$$

причём, выбирая «+» получаем $\cos(\angle \vec{N}_1 \vec{N}_2)$, выбирая «-» получаем $\cos(180 - \angle \vec{N}_1 \vec{N}_2)$.

Пример 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 6.20.

```
% 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 6.20. Вычисление угла между плоскостями (пример 6.19).

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.

Пример 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 6.21. Для построения плоскости применялась функция *plos1*(A, B, C, D), описанная в примере 6.17.

```

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;

```

```

end;
if (k1! = k2) | (k2 != k3)
    flag=0
    clf; cla;
    plos1(N1(1),N1(2),N1(3),N1(4));
    hold on
    plos1(N2(1),N2(2),N2(3),N2(4));
elseif (k1 == k2) & (k2 == k3) & (k3 == k4)
    flag=1;
    clf; cla;
    plos1(N1(1),N1(2),N1(3),N1(4));
elseif (k1 == k2) & (k2 == k3) & (k3 != k4)
    flag=2;
    disp('Геометрическая фигура не определена!')
end;
end;
% Случай а)
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]);
% Случай b)
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]);
% Случай c)
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 в случае c)
Геометрическая фигура не определена!

```

Листинг 6.21. Построение прямых, заданных уравнениями плоскостей (пример 6.20).

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.

Пример 6.21. Найти направляющие коэффициенты прямой $2x - 2y - z + 8 = 0$ и $x + 2y - 2z + 1 = 0$ (листинг 6.24).

```

N1=[2,-2,-1];N2=[1,2,-2];
% Расчёт координат векторного произведения

```

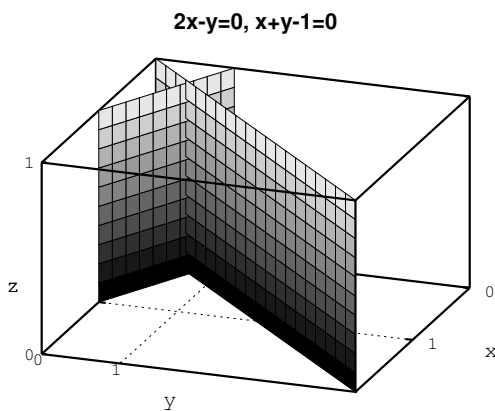


Рис. 6.17. Геометрическая интерпретация примера 6.20, случай а)

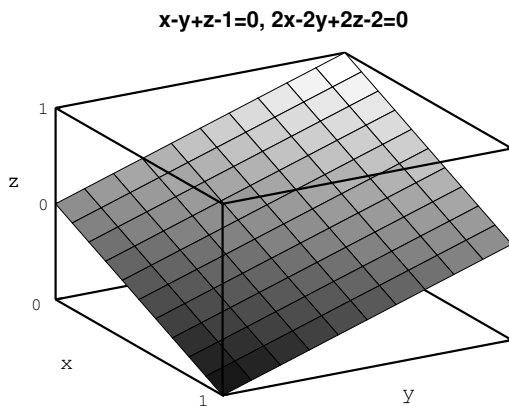


Рис. 6.18. Геометрическая интерпретация примера 6.20, случай б)

```

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 6.22. Расчёт направляющих коэффициентов прямой (пример 6.21).

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.

Пример 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 примера (рис. 6.19) показаны в листинге 6.23.

```

clf; cla;
set(gcf,'Position',[20,20,400,400]);
set(gca,'Position',[.1,.1,.8,.8]);
xlabel('x'); ylabel('y'); zlabel('z');
axis([3,5,-3,2,-2,3]);
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);

```

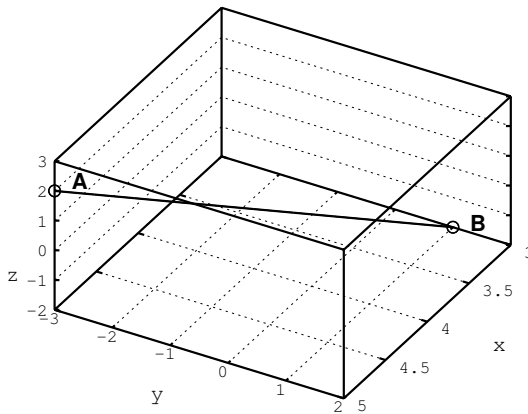


Рис. 6.19. Прямая, проходящая через две точки

```
% 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 6.23. Построение прямой по параметрическим уравнениям (пример 6.22).

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

$$\cos(\phi) = \pm \frac{ll' + mm' + nn'}{\sqrt{l^2 + m^2 + n^2} \sqrt{l'^2 + m'^2 + n'^2}}.$$

Пример 6.23. Найти угол между прямыми $x = t, y = 2t, z = 3t$ и $x = -1 + 2t, y = 1 + t, z = -1 + 4t$ (листинг 6.24, рис. 6.20).

```
a=[1,2,3];b=[2,1,4]; % Исходные данные
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');
```

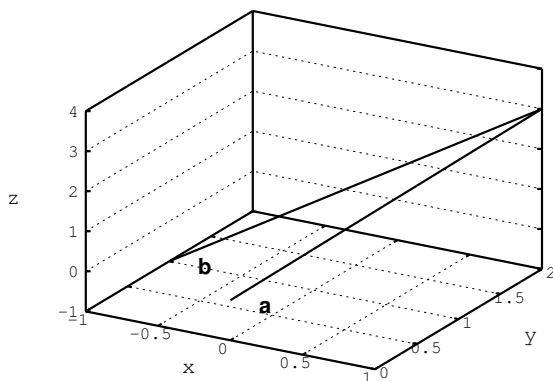


Рис. 6.20. Угол между прямыми

```

xlabel('x'); ylabel('y'); zlabel('z');
axis([-1,1,0,2,-1,4]);
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',[30 30])
% Результат
fi_1 = 21

```

Листинг 6.24. Нахождение и построение угла между прямыми (пример 6.23).

Прямая $x = x_0 + lt, y = y_0 + mt, z = z_0 + nt$ 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 .

Пример 6.24. Найти точку пересечения прямой $x = -5 + 3t, y = 3 - t, z = -3 + 2t$ с плоскостью $2x + 3y + 3z - 8 = 0$.

Выполним расчёты в технике символьных вычислений (листинг 6.25).

```
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 = 1.00000
z = 1.00000
```

Листинг 6.25.

Глава 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 функция $f(x)$ имеет на интервале $[a, b]$ хотя бы один корень, если она поменяла на этом интервале знак: $f(a) \cdot f(b) < 0$. Границы интервала a и b называют *пределами интервала изоляции*¹. На втором этапе проводят уточнение отделённых корней, т.е. находят корни с заданной точностью.

7.1 Решение алгебраических уравнений

Любое уравнение $P(x) = 0$, где $P(x)$ это многочлен (*полином*), отличный от нулевого, называется *алгебраическим уравнением* относительно переменной x . Всякое алгебраическое уравнение относительно x можно записать в виде

$$a_0 + a_1x + a_2x^2 + \dots + a_nx^n = 0, \quad \text{где } a_0 \neq 0, n \geq 1$$

¹Графический метод весьма приблизителен, отделить корни для функции $f(x) = x \sin(\frac{1}{x})$, при $x \neq 0$ и $f(x) = 0$ при $x = 0$ ни на каком интервале, содержащем нуль ($x = 0$) таким способом не удастся. (*Прим. редактора*).

a_i — коэффициенты алгебраического уравнения n -й степени. Например, линейное уравнение это алгебраическое уравнение первой степени, квадратное — второй, кубическое — третьей и так далее.

В **Octave** определить алгебраическое уравнение можно в виде вектора его коэффициентов $p = \{a_n, a_{n-1}, \dots, a_1, a_0\}$. Например, полином $2x^5 + 3x^3 - 1 = 0$ задаётся вектором:

```
>>> p=[2,0,3,0,-1]
p = 2 0 3 0 -1
```

Рассмотрим функции, предназначенные для действий над многочленами.

Произведение двух многочленов вычисляет функция $q = \text{conv}(p1, p2)$, 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 $p2$.

Пример 7.1. Определить многочлен, который получится в результате умножения выражений $3x^4 - 7x^2 + 5$ и $x^3 + 2x - 1$.

Как видно из листинга 7.1 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 7.1. Умножение многочленов (пример 7.1).

Частное и остаток от деления двух многочленов находит функция $[q, r] = \text{deconv}(p1, p2)$, 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 $p2$.

Пример 7.2. Найти частное и остаток от деления многочлена $x^6 - x^5 + 3x^4 - 8x^2 + x - 10$ на многочлен $x^3 + x - 1 = 0$.

В результате имеем (см. листинг 7.2):

$$\frac{x^6 - x^5 + 3x^4 - 8x^2 + x - 10}{x^3 + x - 1} = x^3 - x^2 + 2x + 2 + \frac{1}{-11x^2 + x - 8}$$

.

```
>>> 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 7.2. Деление многочленов (пример 7.2).

Выполнить *разложение частного двух многочленов*, представляющих собой правильную дробь на простейшие рациональные дроби вида

$$\frac{P_1(x)}{P_2(x)} = \sum_{j=1}^M \frac{a_j}{(x - b_j)^{k_j}} + \sum_{i=1}^N c_i x^{N-i}$$

можно с помощью функции $[a, b, c, k] = \text{residue}(p1, p2)$, где $p1$ — многочлен степени n (числитель), $p2$ — многочлен степени m (знаменатель), причём $n < m$.

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.

Пример 7.3. Разложить выражение $\frac{x^3+1}{x^4-3x^3+3x^2-x}$ на простейшие дроби.

Проанализировав листинг 7.3 запишем решение:

$$\frac{x^3 + 1}{x^4 - 3x^3 + 3x^2 - x} = \frac{2}{x - 1} + \frac{1}{(x - 1)^2} + \frac{2}{(x - 1)^3} + \frac{-1}{x}.$$

Значение вектора $c = [] (0 \times 0)$ 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 = [] (0 x 0)
k =
1
2
3
1

```

Листинг 7.3. Разложение на простейшие дроби (пример 7.3).

Пример 7.4. Разложить выражение $\frac{7x^2+26x-9}{x^4+4x^3+4x^2-9}$ на простейшие дроби.

Из листинга 7.4 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 $\text{residue}(p1, p2)$ выполняет разложение только на простейшие дроби вида $\frac{a}{(x-b)^k}$.

```

>>> p1=[1 26 -9];
>>> p2=[1 4 4 -9];
>>> [a,b,c,k]=residue(p1,p2)
a =
-0.100000 - 6.120680 i
-0.100000 + 6.120680 i
1.200000 + 0.000000 i
b =
-2.50000 + 1.65831 i
-2.50000 - 1.65831 i
1.00000 + 0.00000 i
c = [] (0 x 0)
k =
1
1
1

```

Листинг 7.4. Неудача в разложении на дроби (пример 7.4).

Вычислить значение многочлена в заданной точке можно с помощью функции $\text{polyval}(p, x)$, где p — многочлен степени n , x — значение, которое нужно подставить в многочлен.

Пример 7.5. Вычислить значение многочлена $x^6 - x^5 + 3x^4 - 8x^2 + x - 10$ в точках $x_1 = -1, x_2 = 1$ (листинг 7.5).

```
>>> p=[1 -1 3 0 -8 1 -10];
>>> x=[-1,1];
>>> polyval(p,x)
ans = -14 -14
```

Листинг 7.5. Вычисление значения многочлена (пример 7.5).

Вычислить производную от многочлена позволяет функция $\text{polyder}(p)$, где p — многочлен степени n . Функция формирует вектор коэффициентов многочлена, являющегося производной от p .

Производную произведения двух векторов вычисляет функция $\text{polyder}(p1, p2)$, где $p1$ и $p2$ — многочлены.

Вызов функции в общем виде $[q, r] = \text{polyder}(p1, p2)$ приведёт к вычислению производной от частного $p1$ на $p2$ и выдаст результат в виде отношения полиномов q и r .

Пример 7.6. Вычислить производную от многочлена $x^6 - x^5 + 3x^4 - 8x^2 + x - 10$

Листинг 7.6 показал, что решением примера является многочлен $6x^5 - 5x^4 + 12x^3 - 16x + 1$.

```
>>> p=[1 -1 3 0 -8 1 -10];
>>> polyder(p)
ans = 6 -5 12 0 -16 1
```

Листинг 7.6. Вычисление производной многочлена (пример 7.6).

Пример 7.7. Вычислить производную от произведения многочленов $(3x^4 - 7x^2 + 5)(x^3 + 2x - 1)$.

Листинг 7.7 показал, что решением примера является многочлен $21x^6 - 5x^4 - 12x^3 - 27x^2 + 14x + 10$.

```
>>> p1=[3 0 -7 0 5];
>>> p2=[1 0 2 -1];
>>> polyder(p1,p2)
ans = 21 0 -5 -12 -27 14 10
```

Листинг 7.7. Вычисление производной произведения (пример 7.7).

Пример 7.8. Вычислить производную от выражения

$$\frac{x^3 + 1}{x^4 - 3x^3 + 3x^2 - x}.$$

Из листинга 7.8 видно, что решение примера имеет вид

$$\frac{-x^4 - 2x^3 - 4x + 1}{x^6 - 4x^5 + 6x^4 - 4x^3 + x^2}.$$

```
>>> 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 7.8. Вычисление производной частного (пример 7.8).

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 .

Пример 7.9. Найти $\int (x^2 + 2x + 3) dx$.

Согласно листингу 7.9 решение имеет вид:

$$\int (x^2 + 2x + 3) dx = \frac{1}{3}x^3 + x^2 + 3x.$$

Если определить значение постоянной интегрирования (вторая часть листинга 7.9), то решение будет таким:

$$\int (x^2 + 2x + 3) dx = \frac{1}{3}x^3 + x^2 + 3x + 5.$$

```
>>> p=[1 2 3];
>>> polyint(p)
ans = 0.33333 1.00000 3.00000 0.00000
% Указываем постоянную интегрирования
>>> polyint(p,5)
ans = 0.33333 1.00000 3.00000 5.00000
```

Листинг 7.9. Нахождение неопределённого интеграла (пример 7.9).

Построить многочлен по заданному вектору его корней позволяет функция *poly(x)*, где x — вектор корней искомого полинома.

Пример 7.10. Записать алгебраическое уравнение, если известно, что его корни $x_1 = -2, x_2 = 3$.

Согласно листингу 7.10 решение примера имеет вид: $x^2 - x - 6 = 0$

```
>>> x=[-2 3];  
>>> poly(x)  
ans = 1 -1 -6
```

Листинг 7.10. Построение многочлена по корням (пример 7.10).

Решить *алгебраическое уравнение* $P(x) = 0$ можно при помощи встроенной функции $\text{roots}(p)$, где p — многочлен степени n . Функция формирует вектор, элементы которого являются корнями заданного полинома.

Пример 7.11. Решить алгебраическое уравнение $x^2 - x - 6 = 0$.

Из листинга 7.11 видно, что значения $x_1 = -2$, $x_2 = 3$ являются решением уравнения.

```
>>> p=[1 -1 -6];  
>>> roots(p)  
ans =  
 3  
 -2
```

Листинг 7.11. Решение алгебраического уравнения (пример 7.11).

Пример 7.12. Найти корни полинома $2x^3 - 3x^2 - 12x - 5 = 0$.

Найдём корни полинома, так как показано в листинге 7.12.

```
>>> p=[2 -3 -12 -5];  
>>> x=roots(p)  
x =  
 3.44949  
 -1.44949  
 -0.50000
```

Листинг 7.12. Нахождение корней полинома (пример 7.12).

Графическое решение заданного уравнения показано в листинге 7.13 и на рис. 7.1. Точки пересечения графика с осью абсцисс и есть корни уравнения. Не трудно заметить, что графическое решение совпадает с аналитическим (листинг 7.12).

```
cla; okno1=figure();  
x=-2:0.1:5.5;  
y=2*x.^3-3*x.^2-12*x-5;
```

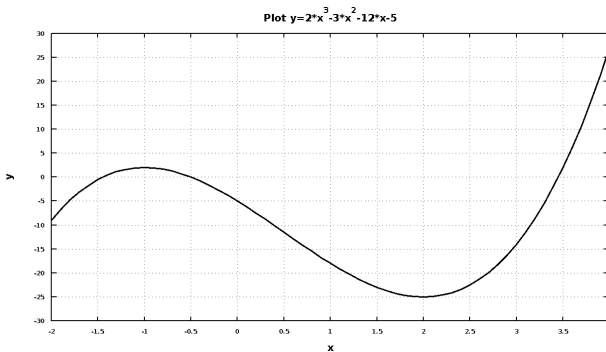


Рис. 7.1. Графическое решение примера 7.12

```
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 7.13. Графическое нахождение корней (пример 7.12).

Пример 7.13. Найти решение уравнения $x^4 + 4x^3 + 4x^2 - 9 = 0$.

Графическое решение примера было получено при помощи последовательности команд приведённых в первой части листинга 7.14. На рис. 7.2 видно, что заданное алгебраическое уравнение имеет два действительных корня. Аналитическое решение примера, представленное во второй части листинга 7.14 показывает не только действительные, но и комплексные корни.

```
% Графическое нахождение корней
cla; okno1=figure();
x=-4:0.1:2;
y=x.^4+4*x.^3+4*x.^2-9;
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]);
```

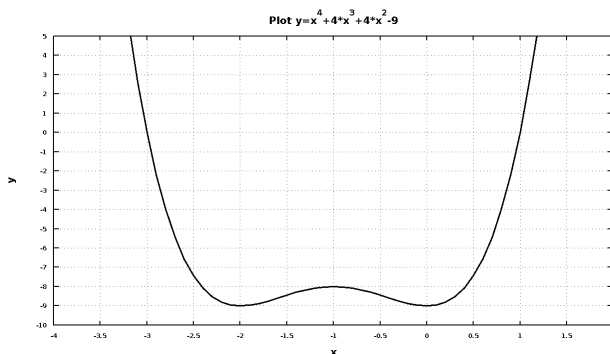



Рис. 7.2. Графическое решение примера 7.13

```
set(gca, 'ytick', [-10:1:5]);
grid on;
xlabel('x'); ylabel('y');
title('Plot y=x^4+4*x^3+4*x^2-9');
% Аналитическое нахождение корней
>>> 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 7.14. Графическое и аналитическое нахождение корней уравнения (пример 7.13).

7.2 Решение трансцендентных уравнений

Уравнение, в котором неизвестное входит в аргумент трансцендентных функций, называется *трансцендентным уравнением*. К трансцендентным уравнениям принадлежат показательные, логарифмические, тригонометрические.

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, *[a, b]* — интервал изоляции корня.

Если функция вызывается в формате: $[x, y] = fzero(name, x0)$, то здесь x — корень уравнения, y — значение функции в точке x .

Пример 7.14. Найти решение уравнения:

$$\sqrt[3]{(2x-3)^2} - \sqrt[3]{(x-1)^2} = 0.$$

Начнём решение данного трансцендентного уравнения с определения интервала изоляции корня. Воспользуемся для этого графическим методом. Построим график функции, указанной в левой части уравнения (листинг 7.15), создав предварительно функцию для её 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)
cla; okno1=figure();
x=-1:0.1:3;
y=f1(x);
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 7.15. Графическое отделение корней (пример 7.14).

На графике (рис. 7.3) видно, что функция $f(x)$ дважды пересекает ось Ox . Первый раз на интервале $[1, 1.5]$, второй — $[1.5, 2.5]$.

Уточним корни, полученные графическим методом. Воспользуемся функцией, вычисляющей левую часть заданного уравнения из листинга 7.15 и обратимся к функции *fzero*, указав в качестве параметров имя созданной функции и число (1.5) близкое к первому корню:

```
>>> x1=fzero('f1', 1.5)
x1 = 1.3333
```

Теперь применим функцию *fzero*, указав в качестве параметров имя функции, и интервал изоляции второго корня:

```
>>> x2=fzero('f1', [1.5 2.5])
x1= 2
```

Не трудно заметить, что и в первом и во втором случае функция *fzero* правильно нашла корни заданного уравнения.

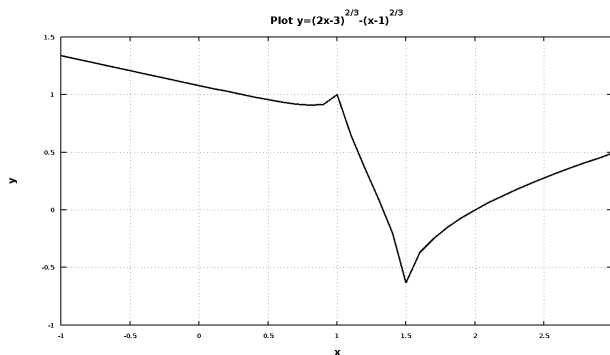


Рис. 7.3. Графическое решение примера 7.14

Ниже приведён 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/3.4.0/m/optimization/fzero.m at line 170, column 5
```

В следующем листинге приведён 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 % Решение уравнения
X = 1.3333    2.0000
>>> Y % Значения функции в точке X
Y = -2.3870e-15    0.0000e+00
```

Пример 7.15. Найти решение уравнения $x^4 + 4x^3 + 4x^2 - 9 = 0$.

Как видим, левая часть уравнения представляет собой полином. В примере 7.13 было показано, что данное уравнение имеет четыре корня: два действительных и два мнимых (листинг 7.14).

Листинг 7.16 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 7.16. Решение уравнения с помощью *fzero* (пример 7.15).

7.3 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*.

Несложную систему элементарной подстановкой можно привести к нелинейному уравнению. Рассмотрим несколько примеров, в которых описан такой приём решения нелинейной системы.

Пример 7.16. Решить систему уравнений:

$$\begin{cases} x^2 + 2y^2 = 1 \\ x - y = 1. \end{cases}$$

Найдём графическое решение с помощью команд листинга 7.17. На рис. 7.4 видно, что система имеет два решения.

```

% 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

```

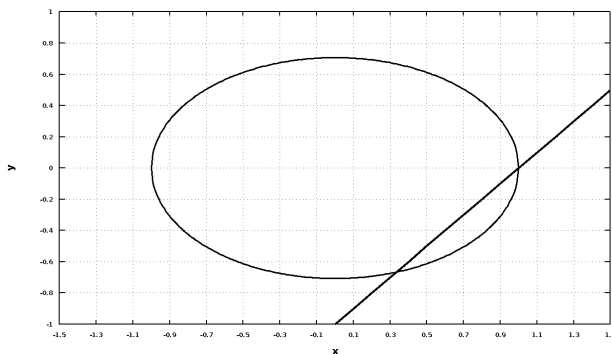


Рис. 7.4. Решение системы нелинейных уравнений из примера 7.16

```

cla; okno1=figure ();
x1=-1:0.01:1; x2=-1.5:0.1:1.5;
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 7.17. Построение графика системы уравнений (пример 7.16).

Не сложно убедиться, что данная система легко сводится к одному уравнению: $\{y = x - 1, x^2 + 2y^2 = 1\} \Rightarrow x^2 + 2(x - 1)^2 - 1 = 0 \Rightarrow 3x^2 - 4x + 1 = 0$.

Решив это уравнение с помощью функции *roots*, найдём значения x . Затем подставим их в одно из уравнений системы, например во второе, и тем самым вычислим значения y :

```

>>> p=[3 -4 1];
>>> x=roots(p)
x =
    1.00000
    0.33333
>>> y=x-1
y =
   -1.1102e-16
   -6.6667e-01

```

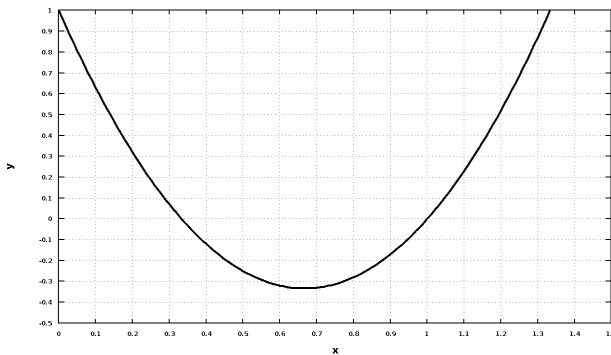


Рис. 7.5. Графическое решение примера 7.16

Понятно, что система имеет два решения $x_1 = 1$, $y_1 = 0$ и $x_2 = \frac{1}{3}$, $y_2 = -\frac{2}{3}$ (рис. 7.4). Графическое решение уравнения (листинг 7.18), к которому сводится система, показано на рис. 7.5.

```
function y=f(x)
    y=3*x.^2-4*x+1;
end;
cla; okno1=figure();
x=0:0.01:1.5;
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 7.18. График упрощённой системы (пример 7.16).

Пример 7.17. Решить систему уравнений:

$$\begin{cases} \sin(x+1) - y = 1.2 \\ 2x + \cos(y) = 2. \end{cases}$$

Проведём 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 7.19), к которому сводится система, его удобно использовать для выбора начального приближения функции *fzero*. Вторая часть листинга 7.19

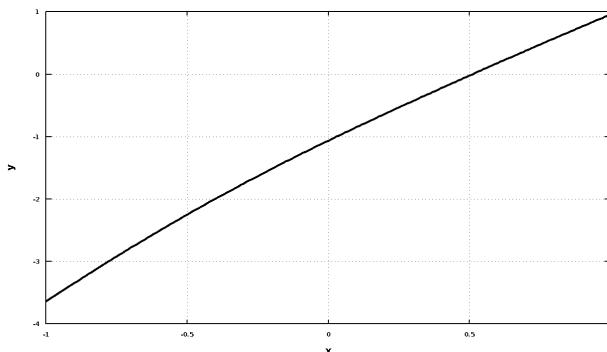


Рис. 7.6. Графическое решение примера 7.17

содержит решение заданной системы. Здесь значения x вычисляются при помощи функции *fzero*, а значение y определяется из первого уравнения системы.

```
% Строим график
function y=fun(x)
    z=sin(x+1)-1.2;
    y=2*x+cos(z)-2;
end;
cla; okno1=figure();
x=-1:0.01:1;
L=plot(x,fun(x)); set(L,'LineWidth',2,'Color','k')
grid on; xlabel('x'); ylabel('y');
% Находим аналитическое решение системы вблизи x = 0
>>> X=fzero('fun',0)
>>> Y=sin(X+1)-1.2
X = 0.51015
Y = -0.20184
```

Листинг 7.19. Графическое и аналитическое решение (пример 7.17).

Решить систему нелинейных уравнений, 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 $F(x) = 0$ (она должна принимать на входе вектор аргументов и возвращать вектор значений), $x0$ — вектор приближений, относительно которого будет осуществляться поиск решения.

Пример 7.18. Найти решение системы нелинейных уравнений:

$$\begin{cases} \cos(x) + 2y = 2 \\ \frac{x^2}{3} - \frac{y^2}{3} = 1. \end{cases}$$

Решим систему графически, для чего выполним перечень команд указанных в листинге 7.20. 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
cla; okno1=figure();
x1=-5:0.001:-sqrt(3);
x2=sqrt(3):0.001:5;
x3=-5:0.1:5;
% 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 7.20. Графическое решение системы (пример 7.18).

Составим функцию, соответствующую левой части системы. 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;
```

Теперь решим систему, указав в качестве начального приближения сначала вектор $[-3, -1]$, затем $[1, 3]$.

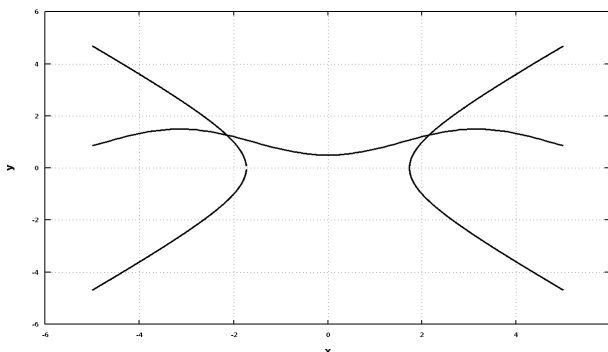


Рис. 7.7. Графическое решение примера 7.18

```
>>> [X1_Y1]=fsolve('fun', [-3 -1])
X1_Y1 = -2.1499    1.2736
>>> [X2_Y2]=fsolve('fun', [1 3])
X2_Y2 = 2.1499    1.2736
```

Понятно, что решением примера 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.

Пример 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 7.21 содержит функцию заданной системы и её решение. Обратите внимание на выходные параметры функции *fsolve*. В нашем случае значения функции f для найденного решения x близки

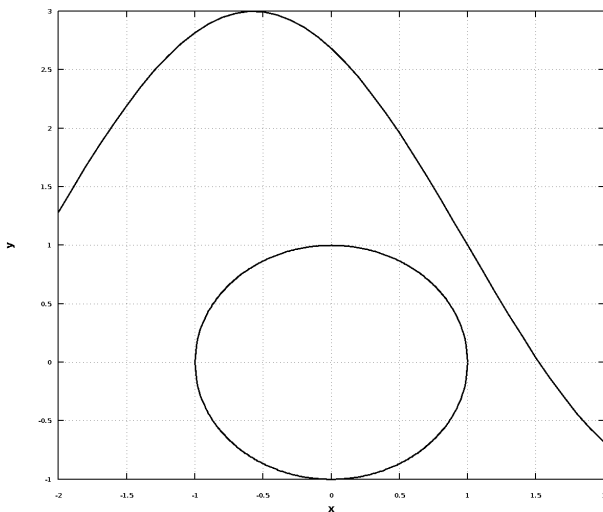


Рис. 7.8. Графическое решение примера 7.20

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 7.21. Решение системы с помощью *fsolve* (пример 7.19).

Пример 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 7.22.

```
% 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(); cla;
x1=-1:0.01:1; x3=-2:0.1:2;
% 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 7.22. Графическое решение системы (пример 7.20).

Однако применение к системе функции *fsolve* даёт положительный ответ, что видно из листинга 7.23. 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 её 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 7.23. Решение с помощью *fsolve* (пример 7.20).

7.4 Решение нелинейных уравнений и систем в символьных переменных

Напомним, что для работы с символьными переменными в **Octave** должен быть подключён специальный пакет расширений **octave-**

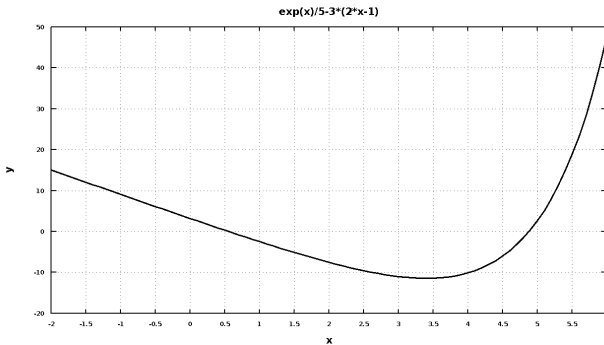


Рис. 7.9. Графическое решение примера 7.21

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*.

Пример 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 7.24.

```
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 = symsolve(y,0)
>>> q2 = symsolve(y,4)
q1 = 0.55825
q2 = 4.8777
```

Листинг 7.24. Решение системы с помощью *symsolve* (пример 7.21).

Пример 7.22. Решить систему:

$$\begin{cases} x^2 + y^2 + 3x - 2y = 4 \\ x + 2y = 5. \end{cases}$$

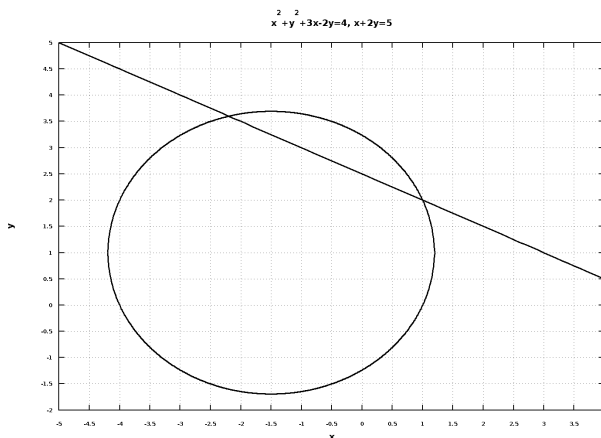


Рис. 7.10. Графическое решение примера 7.22

Решение системы (листинг 7.25) 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 7.25. Решение системы с помощью `symfsolve` (пример 7.22).

Глава 8

Интегрирование и дифференцирование

Дифференцирование в **Octave** осуществляется в технике символьных переменных¹. 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 .

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 её угловой коэффициент, будет равен значению производной в этой точке $k = f'(x)$. Следовательно, уравнение касательной к линии в заданной точке имеет вид: $y(x) = f'(x)(x - x_0) + f(x_0)$.

¹Для работы с символьными переменными в **Octave** подключите специальный пакет расширений `octave-symbolic`. Установка пакетов расширений описана в первой главе, техника работы с символьными переменными — в п. 2.7.

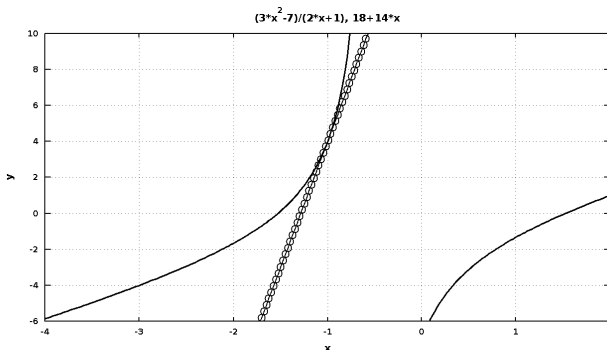


Рис. 8.1. График функции и её касательной

Пример 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);
f1=differentiate(f,x)% Первая производная от заданной функции
% Уравнение касательной: k=subs(f1,x,x0), f(x0)=subs(f,x,x0)
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+(14.0-6.0198691364212032297E-19*I)*x
```

Листинг 8.1. Получение уравнения касательной (пример 8.1).

На рис. 8.1 представлены графики заданной функции и её касательной. Рисунок построен с помощью команд из листинга 8.2.

```
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. График функции и её касательной (пример 8.1).

Пример 8.2. Найти а) $f'(x) = \frac{5 \sin(2x)}{\sqrt{\cos(2x)}}$ и б) $f'(x) = \tan(\sqrt[3]{\ln(x)})$.

Решение примера показано в листинге 8.3.

```
>>> clear all;
>>> symbols
>>> x = sym ("x");
% Пример а)
>>> 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))
% Пример б)
>>> 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 8.3. Нахождение производных (пример 8.2).

Если функция $y(x)$ задана параметрическими уравнениями $x = \phi(t)$, $y = \psi(t)$, то производная вычисляется по формуле $y'(x) = \frac{\psi'(t)}{\phi'(t)} = \frac{y'_t}{x'_t}$.

Пример 8.3. Найти производную функции, заданной параметрически:

$$\begin{cases} x(t) = 3 \cos^3(t) \\ y(t) = 3 \sin^3(t) \end{cases}$$

Листинг 8.4 содержит решение примера.

```
>>> 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 8.4. Производная параметрической функции (пример 8.3).

Пример 8.4. Найти производные а) $f''(x) = \ln(\cos(x))$,
б) $f^{IV}(x) = \tan(x)$ (листинг 8.5).

```
>>> clear all;
>>> symbols
>>> x = sym ("x") ;
% Пример а)
>>> f=Log(Cos(x));
>>> differentiate(f,x,2)
ans = -1-cos(x)^(-2)*sin(x)^2
% Пример б)
>>> 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 8.5. Производные высших порядков (пример 8.4).

Пример 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: $y''(x) = \frac{\left(\frac{y_t'}{x_t'}\right)'}{x_t'} = \frac{y_t''x_t' - x_t''y_t'}{(x_t')^3}$.

В листинге 8.6 представлено решение примера

```
>>> 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);
>>> z=(yt2*xt-xt2*yt)/xt^3
z = -(1-cos(t))^(3.0)*(cos(t)*(-1+cos(t))+sin(t)^2)
```

Листинг 8.6. Вторая производная параметрической функции (пр. 8.5).

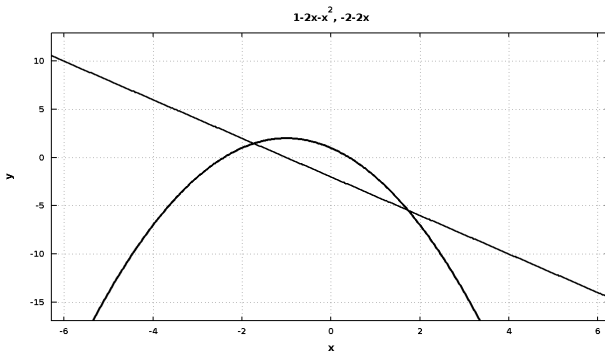


Рис. 8.2. Исследование функции на возрастание и убывание

8.2 Исследование функций

Понятие производной тесно связано с *задачей исследования функции*. Из курса математического анализа известно, что если производная функции $f(x)$ положительна на всём интервале $[a, b]$, то функция на нём *возрастает*, если всюду отрицательна, то $f(x)$ *убывает*.

Пример 8.6. Построить график функции $f(x) = 1 - 2x - x^2$ и её производной. Исследовать функцию на возрастание и убывание.

Вычислим производную заданной функции и построим оба графика в одном окне :

```
>>> symbols
>>> x=sym("x");
>>> f=1-2*x-x^2;
>>> differentiate(f,x)
ans = -2.0-(2.0)*x
% построим график заданной функции и её 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 8.7. Исследование функции (пример 8.6).

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.

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.

Пример 8.7. Исследовать функцию $f(x) = \frac{x^3}{3} - 2x^2 + 3x + 1$ на экстремум.

Найдём производную функции и отобразим её на графике:

```
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
% Изобразим функцию и её производную на графике
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, 3x^2-4x')
% Корни уравнения
>>> x1 = symfsolve(y,1)
>>> x2 = symfsolve(y,3)
x1 = 1
x2 = 3
```

Листинг 8.8. Исследование функции (пример 8.7).

На рис. 8.3 и в листинге 8.8 видно, что первая производная обращается в нуль в точках $x = 1$ и $x = 3$. При переходе через точку $x = 1$ $f'(x)$ меняет знак с плюса на минус, следовательно, это точка максимума функции $f(x)$, а в точке $x = 3$ знак первой производной меняется с минуса на плюс, то есть это точка минимума.

График функции называется *выпуклым на промежутке* $[a, b]$, если он расположен выше касательной, проведённой в любой точке этого интервала. Если же график функции лежит ниже касательной, то он называется *вогнутым*. Функция будет *выпуклой на интервале*

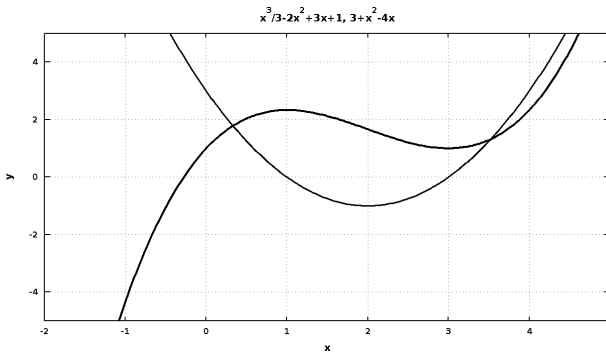


Рис. 8.3. Исследование функции на экстремум

$[a, b]$, если вторая производная $f''(x)$ на нём 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 неё имеет значения разных знаков, то точка a — *точка перегиба*.

Пример 8.8. Определить точки перегиба функции $f(x) = \frac{3x-2}{x^2+1}$.

Найдём вторую производную заданной функции. Построим графики функции и её второй производной. Определим точки в которых вторая производная обращается в ноль (листинг 8.9).

```
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
```

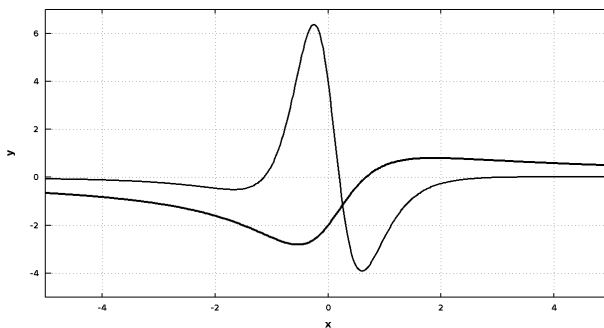


Рис. 8.4. Определение точки перегиба функции

```
x2 = 0.19855
x3 = 2.9425
```

Листинг 8.9. Точки перегиба функции (пример 8.8).

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 трёх точках $x_1 = -1.1411$, $x_2 = 0.19855$, $x_3 = 2.9425$, причём 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 $f''(x) > 0$.

8.3 Численное интегрирование

Пусть дана функция $f(x)$, известно, что она непрерывна на интервале $[a, b]$ и уже определена её первообразная $F(x)$, тогда *определённый интеграл* от этой функции можно вычислить в пределах от a до b по *формуле Ньютона-Лейбница*:

$$\int_a^b f(x) dx = F(b) - F(a), \quad \text{где } F'(x) = f(x).$$

Пример 8.9. Вычислить определённый интеграл

$$I = \int_2^5 \sqrt{2x-1} dx.$$

К сожалению в **Octave** не предусмотрены средства символьного интегрирования, поэтому обратимся к таблице интегралов и найдём, что

$$I = \int \sqrt{2x-1} dx = \frac{1}{3} \sqrt[3]{(2x-1)^2} + C.$$

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 8.10. Вычисление определённого интеграла (пример 8.9).

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 Ньютон–Лейбница. Кроме того, нередко подынтегральная функция задаётся таблицей или графиком и тогда понятие первообразной вообще теряет смысл. В этом случае большое значение имеют *численные методы интегрирования*, основная задача которых заключается в вычислении значения определённого интеграла на основании значений подынтегральной функции.

Численное вычисление определённого интеграла называют *механической квадратурой*. Формулы, соответствующие тому или иному численному методу приближённого интегрирования, называют *квадратурными*. Подобное название связано с *геометрическим смыслом определённого интеграла*: значение определённого интеграла

$$y = \int_a^b f(x) dx, \quad f(x) \neq 0,$$

равно площади криволинейной трапеции с основаниями $[a, b]$ и $f(x)$.

Вообще говоря, классические учебники по численной математике предлагают немало методов интегрирования, но здесь мы рассмотрим только те методы, которые имеют непосредственное отношение к функциям **Octave**.

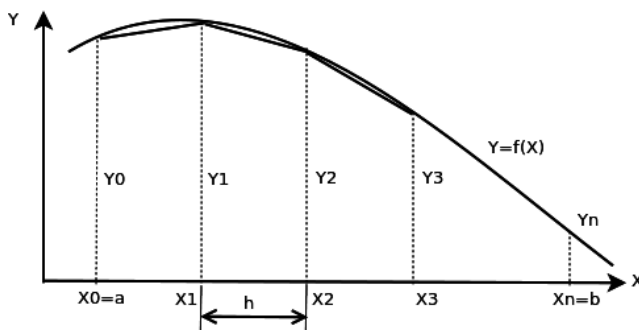


Рис. 8.5. Геометрическая интерпретация метода трапеций

8.3.1 Интегрирование по методу трапеций

Изложим геометрическую интерпретацию *интегрирования по методу трапеций*. Для этого участок интегрирования $[a, b]$ разобьём точками на n равных частей (рис. 8.5), причём $x_0 = a$, $x_n = b$.

Тогда длина каждой части будет равна $h = \frac{b-a}{n}$, а значение абсциссы каждой из точек разбиения можно вычислить по формуле $x_i = x_0 + ih$, $i = 1, 2, \dots, n-1$. Теперь из каждой точки x_i проведём перпендикуляр до пересечения с кривой $f(x)$, а затем заменим каждую из полученных криволинейных трапеций прямолинейной. Приближённое значение интеграла будем рассматривать как сумму площадей прямолинейных трапеций, причём площадь отдельной трапеции составляет $S_i = \frac{y_{i-1} + y_i}{2} h$, следовательно, площадь искомой фигуры вычисляются по формуле:

$$S = \int_a^b f(x) dx = \sum_{i=1}^n S_i = \frac{h}{2} \sum_{i=1}^n (y_{i-1} + y_i) = h \left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i \right).$$

Таким образом, получена *квадратурная формула трапеций* для численного интегрирования:

$$I = \int_a^b f(x) dx = h \left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i) \right).$$

Функции *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).

Таблица 8.1. Значения функции $y(x) = \cos(x)$

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

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 , причём по умолчанию элементы вектора x принимают значения номеров элементов вектора y .

Пример 8.10. Вычислить интеграл от функции $y(x) = \cos(x)$. Значения функции представлены в табл. 8.1.

Решение примера представлено в листинге 8.11.

```
>>> 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 8.11. Вычисление интеграла методом трапеций (пример 8.10).

Пример 8.11. Вычислить интеграл $I = \int_2^5 \sqrt{2x-1} dx$.

Листинг 8.12 содержит несколько вариантов решения данного примера. В первом случае интервал интегрирования делится на отрезки с шагом 1, во втором 0.5, в третьем 0.1 и в четвёртом 0.05. Не трудно заметить, что чем больше точек разбиения, тем точнее значение искомого интеграла. Решение можно сравнить с результатом полученным в задаче 8.9, где этот же интеграл был найден по формулам Ньютона-Лейбница (листинг 8.10).

```
clear all;
% Вариант 1. h=1
>>> x=2:5;y=sqrt(2*x-1);I1=trapz(x, y)
% Вариант 2. h=0.5
>>> x=2:0.5:5;y=sqrt(2*x-1);I2=trapz(x, y)
% Вариант 3. h=0.1
>>> x=2:0.1:5;y=sqrt(2*x-1);I3=trapz(x, y)
% Вариант 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 8.12. Вычисление интеграла с разной точностью (пр. 8.11).

В листинге 8.13 приведён 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 8.12, первый вариант). То есть мы нашли сумму площадей трёх 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)* приведёт к вычислению суммы площадей шестидесяти трапеций с основанием $h = 1$ и боковыми сторонами, заданными вектором y . Таким образом, в первом и втором примерах листинга 8.13 вычисляются площади совершенно разных фигур.

```
% Пример 1.  
>>> x=2:5; y=sqrt(2*x-1); I=trapz(y)  
I = 7.2478  
% Пример 2.  
>>> x=2:0.05:5; y=sqrt(2*x-1); I=trapz(y)  
I = 145.36
```

Листинг 8.13. Особенности вычисления интеграла через *trapz(y)*.

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 её работы является вектор, состоящий из промежуточных вычислений. То есть, если общая площадь S криволинейной трапеции сформирована из суммы площадей $\sum_{i=1}^n S_i$ прямолинейных трапеций, то элементы вектора представляют собой следующую последовательность $S_1 = 0$, $S_2 = S_1 + S_2$, $S_3 = S_1 + S_2 + S_3, \dots$, $S_n = S_1 + S_2 + S_3 + \dots + S_n$.

Таким образом, последний элемент вектора будет равен искомой площади фигуры S . Функцию интегрирования с накоплением можно

вызывать в форматах $\text{cumtrapz}(x, y)$ и $\text{cumtrapz}(y)$, где x и y векторы, определяющие функцию $y(x)$.

Пример 8.12. Вычислить интеграл $I = \int_0^{\frac{\pi}{2}} \frac{1}{5 + \sin(x)} dx$.

Листинг 8.14 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. Интегрирование на левой части интервала от 0 до 1
>>> 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 8.14. Вычисление интеграла через cumtrapz (пример 8.12).

8.3.2 Интегрирование по методу Симпсона

Изложим идею *интегрирования по методу Симпсона*. Пусть $n = 2m$ — чётное число, а $y_i = f(x_i)$ ($i = 0, 1, \dots, n$) — значения функции $y = f(x)$ для равноотстоящих точек $a = x_0, x_1, x_2, \dots, x_n = b$ с шагом $h = \frac{b-a}{n} = \frac{b-a}{2m}$. На паре соседних участков (рис. 8.6) кривая $y = f(x)$ заменяется параболой $y = L(x)$, коэффициенты которой подобраны так, что она проходит через точки Y_0, Y_1, Y_2 .

Площадь криволинейной трапеции, ограниченной сверху параболой, составит: $S_i = \frac{h}{3}(y_{i-1} + 4y_i + y_{i+1})$.

Суммируя площади всех криволинейных трапеций, получим:

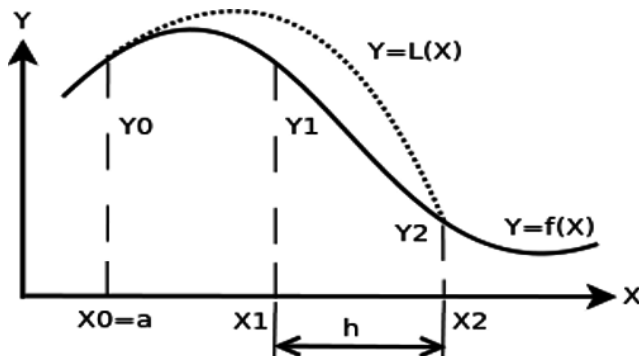


Рис. 8.6. Геометрическая интерпретация интегрирования по методу Симпсона

$$S = \int_a^b f(x) dx \approx \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \dots + 2y_{2m-2} + 4y_{2m-1} + y_{2m}) = \\ = \frac{h}{3} \left(y_0 + y_{2m} + \sum_{i=1}^{2m-1} p y_i \right), \text{ где }^2 \quad p = 3 - (-1)^i$$

Следовательно, формула Симпсона для численного интегрирования имеет вид: $I = \int_a^b f(x) dx = \frac{h}{3} \left(f(a) + f(b) + \sum_{i=1}^{2m-1} p y_i \right)$.

Методы трапеций и Симпсона являются частными случаями *квадратурных формул Ньютона-Котеса*, которые, вообще говоря, имеют вид

$$\int_a^b y dx = (b-a) \sum_{i=0}^n H_i y_i,$$

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*.

²Как нетрудно заметить $p = 2$ при чётном i и $p = 4$ при нечётном i .

Для $n > 2$ получают квадратурные формулы Ньютона–Котеса высших порядков.

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 $[F, K] = \text{quadv}(\text{name}, a, b, [\text{tol}, \text{trace}])$, 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.

Пример 8.13. Вычислить интеграл $\int_0^1 \sqrt{4-x^2} dx$.

Решение примера с применением функции *quadv* приведено в листинге 8.15.

```
% Подынтегральная функция
function y=G(x)
    y=(4-x^2).^(1/2);
end;
>>> format long
% Вычисление интеграла по методу Симпсона
% 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
```


Как известно, корни полинома Лежандра существуют при любом n , различны и принадлежат интервалу $[-1; 1]$.

Итак, *квадратурной формулой Гаусса* называют выражение

$$\int_a^b f(x) dx = \frac{b-a}{2} \sum_{i=1}^n A_i f\left(\frac{a+b}{2} + \frac{b-a}{2} t_i\right),$$

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.

Пример 8.14. Вычислить интеграл $\int_0^1 t^2 \sqrt{\left(3 + \sin\left(\frac{1}{t}\right)\right)} dt$.

Обратите внимание, что в нижней границе интегрирования подынтегральная функция терпит разрыв. Решение примера с применением функции *quad* приведено в листинге 8.16.

```
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 8.16. Вычисление интеграла через *quad* (пример 8.14).

Функции

$F = \text{quadl}(f, a, b[, tol, trace])$ и $[F, err] = \text{quadgk}(f, a, b[, tol, trace])$, где: *name* — 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 Гаусса-Конрада.

Пример 8.15. Вычислить интеграл $\int_{-\frac{\pi}{3}}^{\frac{\pi}{3}} tg^4(x) dx$.

Решение примера с применением функций *quadl* и *quadgk* приведено в листинге 8.17.

```

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 8.17. Вычисление интеграла через *quadl* и *quadgk* (пр. 8.15).

Глава 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

$$H(t, x, x', x'', \dots, x^{(n)}) = 0. \quad (9.1)$$

Решением дифференциального уравнения называется функция $x(t)$, которая обращает уравнение в тождество.

Системой дифференциальных уравнений n -го порядка называется система вида:

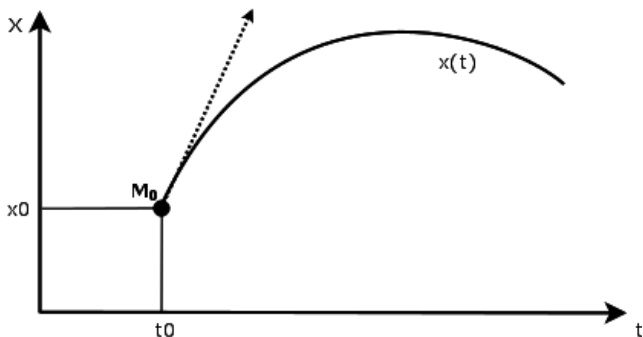


Рис. 9.1. Интегральная кривая, проходящая через точку $M_0(t_0, x_0)$

решение. Но даже для уравнений с известным аналитическим решением очень часто необходимо вычислить числовое значение при определённых исходных данных. Поэтому широкое распространение получили численные методы решения обыкновенных дифференциальных уравнений.

9.2 Численные методы решения дифференциальных уравнений и их реализация

Численные методы решения дифференциального уравнения первого порядка будем рассматривать для следующей задачи Коши. Найти решение дифференциального уравнения

$$x' = f(x, t) \quad (9.4)$$

удовлетворяющее начальному условию

$$x(t_0) = x_0 \quad (9.5)$$

иными словами, требуется найти интегральную кривую $x = x(t)$, проходящую через заданную точку $M_0(t_0, x_0)$ (рис. 9.1).

Для дифференциального уравнения n -го порядка

$$x^{(n)} = f(t, x, x', x'', \dots, x^{(n-1)}) \quad (9.6)$$

задача Коши состоит в нахождении решения $x = x(t)$, удовлетворяющего уравнению (9.6) и начальным условиям

$$x(t_0) = x_0, x'(t_0) = x'_0, \dots, x^{(n-1)}(t_0) = x_0^{(n-1)} \quad (9.7)$$

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

$$t_i = t_0 + ih, \quad i = 0, 1, \dots, n, \quad h = \frac{t_n - t_0}{n} \quad (9.8)$$

Для вычисления значения функции в точке t_1 разложим функцию $x = x(t)$ в окрестности точки t_0 в ряд Тейлора [2]

$$x(t_1) = x(t_0 + h) = x(t_0) + x'(t_0)h + x''(t_0)\frac{h^2}{2} + \dots \quad (9.9)$$

При достаточно малом значении h членами выше второго порядка можно пренебречь и с учётом $x'(t_0) = f(x_0, t_0)$ получим следующую формулу для вычисления приближённого значения функции $x(t)$ в точке t_1

$$x_1 = x_0 + hf(x_0, t_0) \quad (9.10)$$

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

$$x_2 = x_1 + hf(x_1, t_1).$$

Повторяя этот процесс, сформируем последовательность значений x_i в точках t_i по формуле

$$x_{i+1} = x_i + hf(x_i, t_i), \quad i = 0, 1, \dots, n-1. \quad (9.11)$$

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 $M_0, M_1, M_2, \dots, M_n$ с вершинами $M_i(x_i, y_i)$. Звенья ломаной Эйлера $M_i M_{i+1}$ в каждой вершине M_i имеют направление $y_i =$

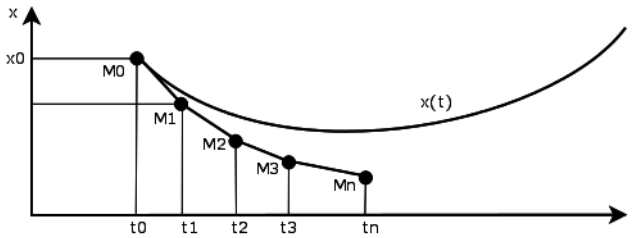


Рис. 9.2. Геометрическая интерпретация метода Эйлера

$f(t_i, x_i)$, совпадающее с направлением интегральной кривой $x(t)$ уравнения (9.4), проходящей через точку M_i (рис. 9.2). Последовательность ломанных Эйлера при $h \rightarrow 0$ на достаточно малом отрезке $[x_i, x_i + h]$ стремится к искомой интегральной кривой.

На каждом шаге решение $x(t)$ определяется с ошибкой за счёт отбрасывания членов ряда Тейлора выше первой степени, что в случае быстро меняющейся функции $f(t, x)$ может привести к быстрому накоплению ошибки. В методе Эйлера следует выбирать достаточной малый шаг h .

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]

$$t_p = t_i + \frac{h}{2}, \quad x_p = x_i + \frac{h}{2} f(x_i, t_i) \quad (9.12)$$

после чего находят значение x_{i+1} по формуле

$$x_{i+1} = x_i + h f(x_p, t_p), \quad i = 0, 1, \dots, n-1 \quad (9.13)$$

9.2.3 Решение дифференциальных уравнений методами Рунге-Кутты

Рассмотренные выше методы Эйлера (как обычный, так и модифицированный) являются частными случаями явного *метода Рунге-Кутты*

Кутта k -го порядка. В общем случае формула вычисления очередного приближения методом Рунге-Кутта имеет вид [2]:

$$x_{i+1} = x_i + h\varphi(t_i, x_i, h), \quad i = 0, 1, \dots, n-1 \quad (9.14)$$

Функция $\varphi(t, x, h)$ приближает отрезок ряда Тейлора до k -го порядка и не содержит частных производных $f(t, x)$ [2].

Метод Эйлера является *методом Рунге-Кутта первого порядка* ($k = 1$) и получается при $\varphi(t, x, h) = f(t, x)$.

Семейство *методов Рунге-Кутта второго порядка* имеет вид [2]

$$x_{i+1} = x_i + h \left((1 - \alpha)f(t_i, x_i) + \alpha f \left(t_i + \frac{h}{2\alpha}, x_i + \frac{h}{2\alpha} f(t_i, x_i) \right) \right), \\ i = 0, 1, \dots, n-1 \quad (9.15)$$

Два наиболее известных среди методов Рунге-Кутта второго порядка [2] — это метод Хойна ($\alpha = \frac{1}{2}$) и модифицированный метод Эйлера ($\alpha = 1$).

Подставив $\alpha = \frac{1}{2}$ в формулу (9.15), получаем расчётную формулу *метода Хойна* [2]:

$$x_{i+1} = x_i + \frac{h}{2} (f(t_i, x_i) + f(t_i + h, x_i + hf(t_i, x_i))), \quad i = 0, 1, \dots, n-1$$

Подставив $\alpha = 1$ в формулу (9.15), получаем расчётную формулу уже рассмотренного выше модифицированного метода Эйлера

$$x_{i+1} = x_i + h \left(f \left(t_i + \frac{h}{2}, x_i + \frac{h}{2} f(t_i, x_i) \right) \right), \quad i = 0, 1, \dots, n-1$$

Наиболее известным является *метод Рунге-Кутта четвёртого порядка*, расчётные формулы которого можно записать в виде [2]:

$$\left\{ \begin{array}{l} x_{i+1} = x_i + \Delta x_i, \quad i = 0, 1, \dots, n-1 \\ \Delta x_i = \frac{h}{6}(K_1^i + 2K_2^i + 2K_3^i + K_4^i) \\ K_1^i = f(t_i, x_i) \\ K_2^i = f(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_1^i) \\ K_3^i = f(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_2^i) \\ K_4^i = f(t_i + h, x_i + hK_3^i) \end{array} \right.$$

Одной из модификаций метода Рунге-Кутты является *метод Кутты-Мерсона* (или *пятиэтапный метод Рунге-Кутты четвёртого порядка*), который состоит в следующем [2].

1. На i -м шаге рассчитываются коэффициенты

$$\begin{aligned} K_1^i &= f(t_i, x_i) \\ K_2^i &= f(t_i + \frac{h}{3}, x_i + \frac{3}{2}K_1^i) \\ K_3^i &= f(t_i + \frac{h}{3}, x_i + \frac{h}{6}K_1^i + \frac{h}{6}K_2^i) \\ K_4^i &= f(t_i + \frac{h}{2}, x_i + \frac{h}{8}K_1^i + \frac{3h}{2}K_2^i) \\ K_5^i &= f(t_i + h, x_i + \frac{h}{2}K_1^i - \frac{3h}{2}K_3^i + 2hK_4^i) \end{aligned} \quad (9.16)$$

2. Вычисляем приближённое значение $x(t_{i+1})$ по формуле

$$\tilde{x}_{i+1} = x_i + \frac{h}{2}(K_1^i - 3K_3^i + 4K_4^i) \quad (9.17)$$

3. Вычисляем приближённое значение $x(t_{i+1})$ по формуле

$$x_{i+1} = x_i + \frac{h}{6}(K_1^i + 4K_3^i + K_5^i) \quad (9.18)$$

4. Вычисляем оценочный коэффициент по формуле

$$R = 0.2|x_{i+1} - \tilde{x}_{i+1}| \quad (9.19)$$

5. Сравниваем R с точностью вычислений ε . Если $R \geq \varepsilon$, то уменьшаем шаг вдвое и возвращаемся к п.1. Если $R < \varepsilon$, то значение, вычисленное по формуле (9.18), и будет вычисленным значением $x(t_{i+1})$ (с точностью ε).

6. Перед переходом к вычислению следующего значения x , сравниваем R с $\frac{\varepsilon}{64}$. Если $R < \frac{\varepsilon}{64}$, то дальнейшие вычисления можно проводить с удвоенным шагом $h = 2h$.

Рассмотренные методы Рунге-Кутты относятся к классу одношаговых методов, в которых для вычисления значения в очередной точке x_{k+1} нужно знать значение в предыдущей точке x_k .

Ещё один класс методов решения задачи Коши — *многошаговые методы*, в которых используются точки $x_{k-3}, x_{k-2}, x_{k-1}, x_k$ для вычисления x_{k+1} . В многошаговых методах первые четыре начальные точки $(t_0, x_0), (t_1, x_1), (t_2, x_2), (t_3, x_3)$ должны быть получены заранее любым из одношаговых методов (метод Эйлера, Рунге-Кутта и т.д.). Наиболее известными *многошаговыми методами* являются методы *прогноза-коррекции Адамса* и *Милна*.

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 $t_0, t_1, t_2, \dots, t_i$ 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 [2]

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(t, x(t)) dt \quad (9.20)$$

При вычислении интеграла, входящего в (9.20), вместо функции $f(t, x(t))$ будем использовать интерполяционный полином Лагранжа, построенный по точкам $(t_{i-3}, x_{i-3}), (t_{i-2}, x_{i-2}), (t_{i-1}, x_{i-1}), (t_i, x_i)$. Подставив полином Лагранжа в (9.20), получаем первое приближение (прогноз) \tilde{x}_{i+1} для значения функции в точке t_{i+1}

$$\begin{aligned} \tilde{x}_{i+1} = x_i + \frac{h}{24} & (-9f(t_{i-3}, x_{i-3}) + 37f(t_{i-2}, x_{i-2}) - \\ & - 59f(t_{i-1}, x_{i-1}) + 55f(t_i, x_i)) \end{aligned} \quad (9.21)$$

Как только \tilde{x}_{i+1} вычислено, его можно использовать. Следующий полином Лагранжа для функции $f(t, x(t))$ построим по точкам

(t_{i-2}, x_{i-2}) , (t_{i-1}, x_{i-1}) , (t_i, x_i) и новой точке $(t_{i+1}, \tilde{x}_{i+1})$, после чего подставляем его в (9.20) и получаем второе приближение (корректор)

$$x_{i+1} = x_i + \frac{h}{24}(f(t_{i-2}, x_{i-2}) - 5f(t_{i-1}, x_{i-1}) + 19f(t_i, x_i) + 9f(t_{i+1}, \tilde{x}_{i+1})) \quad (9.22)$$

Таким образом, для вычисления значения $x(t_{i+1})$ методом Адамса необходимо последовательно применять формулы (9.21), (9.22) [2], а первые четыре точки можно получить методом Рунге-Кутты.

9.2.5 Решение дифференциальных уравнений методом Милна

Отличие *метода Милна* от метода Адамса состоит в использовании в качестве интерполяционного полинома Ньютона.

Подставив в (9.20) вместо функции $f(t, x(t))$ интерполяционный полином Ньютона, построенный по точкам (t_{k-3}, x_{k-3}) , (t_{k-2}, x_{k-2}) , (t_{k-1}, x_{k-1}) , (t_k, x_k) получаем первое приближение — прогноз Милна \tilde{x}_{k+1} для значения функции в точке t_{k+1} [2]

$$\tilde{x}_{k+1} = x_{k-3} + \frac{4h}{3}(2f(t_{k-2}, x_{k-2}) - f(t_{k-1}, x_{k-1}) + 2f(t_k, x_k)) \quad (9.23)$$

Следующий полином Ньютона для функции $f(t, x(t))$ построим по точкам (t_{k-2}, x_{k-2}) , (t_{k-1}, x_{k-1}) , (t_k, x_k) и новой точке $(t_{k+1}, \tilde{x}_{k+1})$, после чего подставляем его в (9.20) и получаем второе приближение — корректор Милна [2]

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, \tilde{x}_{k+1})) \quad (9.24)$$

В методе Милна для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (9.23), (9.24), а первые четыре точки можно получить методом Рунге-Кутты.

Существует *модифицированный метод Милна*. В нём сначала вычисляется первое приближение по формуле (9.23), затем вычисляется управляющий параметр [2]

$$m_{k+1} = \tilde{x}_{k+1} + \frac{28}{29}(x_k - \tilde{x}_k) \quad (9.25)$$

после чего вычисляется значение второго приближения — корректор Милна по формуле

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, m_{k+1})) \quad (9.26)$$

В модифицированном методе Милна первые четыре точки можно получить методом Рунге-Кутты, а для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (9.23), (9.25), (9.26).

9.3 Реализация численных методов

Ниже приведены тексты функций, реализующие рассмотренные в п. 9.2 численные методы решения дифференциальных уравнений.

```
function [x,t]=eiler(a,b,n,x0)
% Функция решения задачи Коши  $x'(t) = g(t, x)$   $x(a) = x_0$  методом Эйлера.
% n — количество отрезков, на которые разбивается интервал  $[a, b]$ .
h=(b-a)/n; % Вычисление шага h.
x(1)=x0;
for i=1:n+1 % Формирование системы равноотстоящих узлов  $t_i$ 
    t(i)=a+(i-1)*h;
end
% Вычисление значений функции в узловых точках 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 методом Эйлера.

```
function [x,t]=eiler_m(a,b,n,x0)
% Функция решения задачи Коши  $x'(t) = g(t, x)$   $x(a) = x_0$ 
% модифицированным методом Эйлера.
% n — количество отрезков, на которые разбивается интервал  $[a, b]$ .
h=(b-a)/n; % Вычисление шага h.
x(1)=x0;
for i=1:n+1 % Формирование системы равноотстоящих узлов  $t_i$ 
    t(i)=a+(i-1)*h;
end
% Вычисление значений функции 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. Функция решения задачи Коши модифицированным методом Эйлера.

```

function [x,t]=runge_kut(a,b,n,x0)
% Функция решения задачи Коши  $x'(t) = g(t,x)$   $x(a) = x_0$  методом Рунге-Кутта
% n — количество отрезков, на которые разбивается интервал [a,b].
h=(b-a)/n;% Вычисление шага h.
x(1)=x0;
for i=1:n+1 % Формирование системы равноотстоящих узлов  $t_i$ 
    t(i)=a+(i-1)*h;
end
% Вычисление значений функции формуле (9.16).
for i=2:n+1
    % Расчёт 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);
    % Расчёт приращения delt
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
end
end

```

Листинг 9.3. Функция решения задачи Коши методом Рунге-Кутта.

```

function [x,t,j]=kut_merson(a,b,n,eps,x0)
% Функция решения задачи Коши  $x'(t) = g(t,x)$   $x(a) = x_0$  методом
% Кутта-Мерсона на интервале интегрирования [a,b] с точностью eps,
% n — количество отрезков, на которые вначале разбивается интервал [a,b].
h=(b-a)/n; % Вычисление шага h.
x(1)=x0; t(1)=a; i=2;
while (t(i-1)+h)<=b
    R=3*eps;
    while R>eps
        % Расчёт 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
end
% В переменной j возвращается количество элементов в массивах x и t
j=i-1
end

```

Листинг 9.4. Функция решения задачи Коши методом Кутта-Мерсона.

```

function [x,t]=adams(a,b,n,x0)
% Функция решения задачи Коши  $x'(t) = g(t, x)$   $x(a) = x_0$  методом Адамса
% n — количество отрезков, на которые разбивается интервал  $[a, b]$ .
h=(b-a)/n; % Вычисление шага h
x(1)=x0;
for i=1:n+1 % Формирование системы равноотстоящих узлов  $t_i$ .
    t(i)=a+(i-1)*h;
end
% Вычисление значений функции в трёх узловых точках по формуле(9.16)
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);
    % Расчёт приращения delt
    delt=h/6*(K1+2*K2+2*K3+K4);
    x(i)=x(i-1)+delt;
end

```

```

for i=4:n % Вычисление значений в остальных точках методом Адамса
% 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))
+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  $x'(t) = g(t, x)$   $x(a) = x_0$  методом Милна
% n — количество отрезков, на которые разбивается интервал [a, b].
h=(b-a)/n; % Вычисление шага h
x(1)=x0; xp(1)=x(1);
for i=1:n+1 % Формирование системы равноотстоящих узлов  $t_i$ 
t(i)=a+(i-1)*h;
end
% Вычисление значений функции в трёх узловых точках по формуле (9.16)
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);
% Расчёт приращения delt
delt=h/6*(K1+2*K2+2*K3+K4);
x(i)=x(i-1)+delt;
xp(i)=x(i);
end
for i=4:n % Вычисление значений в остальных точках методом Адамса
% 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. Функция решения задачи Коши методом Милна¹.

¹Написать функцию модифицированного метода Милна авторы предоставляют читателю.

Рассмотрим использование приведённых выше функций на примере решения следующей задачи Коши.

Пример 9.1. Решить задачу Коши

$$\begin{cases} y'(x) = 6y - 13x^3 - 22x^2 + 17x - 11 + \sin(x); \\ y(0) = 2. \end{cases}$$

Известно точное решение задачи 9.1:

$$y(x) = \frac{119}{296}e^{6x} + \frac{1}{24}(52x^3 + 114x^2 - 30x + 39) - \frac{6\sin(x)}{37} - \frac{\cos(x)}{37}.$$

В листинге 9.7 представлено решение уравнения методами:

- модифицированным методом Эйлера;
- Рунге-Кутты;
- 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]=euler_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-Кутты.
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);
        del_t=h/6*(K1+2*K2+2*K3+K4);
        x(i)=x(i-1)+del_t;
    end
end
% Функция решения задачи Коши методом Кутты-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);
        end
        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
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,ME]=euler_m(0,1,10,2);
% Решение дифференциального уравнения методом Рунге-Кутты.
[YR,XR]=runge_kut(0,1,10,2);
% Решение дифференциального уравнения методом Кутты-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=f1(x1);
% Построение графиков.

```

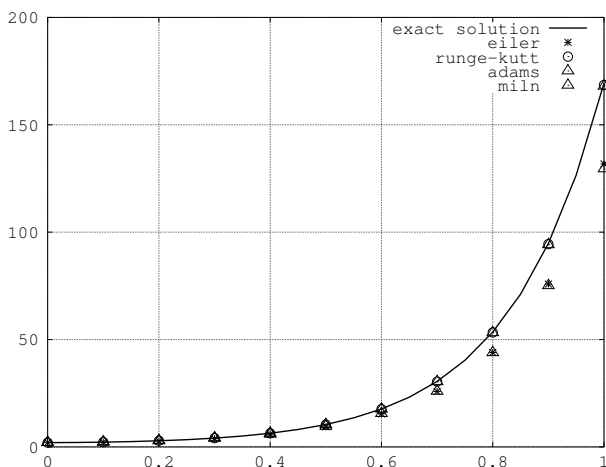


Рис. 9.3. Графики решения модифицированным методом Эйлера, методами Рунге-Кутты, Адамса, Милна и точного решения

```
plot(x1,y1,'-g;exact solution','XE_M,YE_M','*b;euler','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. Различные методы решения задачи Коши (пример 9.1).

На рис. 9.3–9.4 приведены графики решения задачи модифицированным методом Эйлера, методами Рунге-Кутты, 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-Кутты и Кутта-Мерсона.

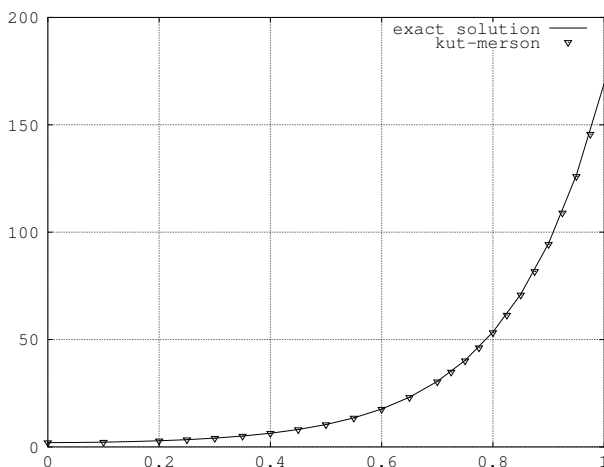


Рис. 9.4. Графики решения методом Кутты-Мерсона и точного решения

9.4 Решение систем дифференциальных уравнений

Все рассмотренные методы решения дифференциальных уравнений применимы и для систем дифференциальных уравнений. Рассмотрим на примере метода Рунге-Кутты, как рассмотренные методы можно обобщить для систем.

Пусть дана система дифференциальных уравнений в матричном виде:

$$\begin{cases} \frac{d\bar{x}}{dt} = \bar{f}(t, \bar{x}) \\ \bar{x}(t_0) = \bar{x}_0 \quad (\text{начальное условие}), \end{cases} \quad (9.27)$$

$$\text{где } \bar{x} = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{pmatrix}, \bar{f}(t, \bar{x}) = \begin{pmatrix} f_1(t, x_1, x_2, \dots, x_n) \\ f_2(t, x_1, x_2, \dots, x_n) \\ \dots \\ f_n(t, x_1, x_2, \dots, x_n) \end{pmatrix}, \bar{x}_0 = \begin{pmatrix} x_1^0 \\ x_2^0 \\ \dots \\ x_n^0 \end{pmatrix}.$$

Задавшись некоторым шагом h и введя стандартные обозначения $t_i = t_0 + ih$, $x_i = x(t_i)$, $\Delta x_i = x_{i+1} - x_i$, $i = 1, 2, \dots, n$ получим формулы метода Рунге-Кутты для системы:

$$\left\{ \begin{array}{l} \bar{x}_{i+1} = \bar{x}_i + \Delta \bar{x}_i, i = 0, 1, \dots, n-1 \\ \Delta \bar{x}_i = \frac{h}{6} (\bar{K}_1^i + 2\bar{K}_2^i + 2\bar{K}_3^i + \bar{K}_4^i) \\ \bar{K}_1^i = \bar{f}(t_i, \bar{x}_i) \\ \bar{K}_2^i = \bar{f}(t_i + \frac{h}{2}, \bar{x}_i + \frac{h}{2} \bar{K}_1^i) \\ \bar{K}_3^i = \bar{f}(t_i + \frac{h}{2}, \bar{x}_i + \frac{h}{2} \bar{K}_2^i) \\ \bar{K}_4^i = \bar{f}(t_i + h, \bar{x}_i + h \bar{K}_3^i) \end{array} \right.$$

9.5 Функции 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-Кутты 2-3-го и 4-5-го порядка точности соответственно;
- *ode5r(@f, interval, X0, options)*, *ode2r(@f, interval, X0, options)* — функции решений обыкновенных жёстких дифференциальных уравнений (или систем).

Функции решают систему дифференциальных уравнений автоматически подбирая шаг для достижения необходимой точности.

Входными параметрами этих функций являются:

- *f* — вектор-функция для вычисления правой части дифференциального уравнения или системы²;
- *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.

²При обращении к функциям *odeXX* используется указатель @*f* на функцию. (Прим. редактора).

Для определения параметров управления ходом решения дифференциальных уравнений используется функция *odeset* следующей структуры:

```
options = odeset('namepar'_1, val_1, 'namepar'_2, val_2, ..., 'namepar'_n, val_n);
```

Здесь

- *namepar_i* — имя *i*-го параметра;
- *val_i* — 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 .

Решим задачу 9.1 с использованием функций *ode23*, *ode45*. Текст программы с комментариями представлен в листинге 9.8.

```
% 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 уравнения.
% RelTol — относительная точность решения 1E-5,
% AbsTol — абсолютная точность решения 1E-5,
% InitialStep — начальное значение шага изменения переменной 0.025,
% MaxStep — максимальное значение шага изменения переменной 0.1.
par=odeset('RelTol', 1e-5, 'AbsTol', 1e-5, 'InitialStep',
    ,0.025, 'MaxStep', 0.1);
% Решение дифференциального уравнения методом Рунге-Кутты 2-3 порядка.
```

```

[X23, Y23]=ode23(@g,[0 0.25],2,par);
% Определение параметров управления ходом решения уравнения.
% RelTol — относительная точность решения 1E-4,
% AbsTol — абсолютная точность решения 1E-4,
% InitialStep — начальное значение шага изменения переменной 0.005,
% MaxStep — максимальное значение шага изменения переменной 0.2.
par=odeset('RelTol',1e-4,'AbsTol',1e-4,'InitialStep',0.05,'
    MaxStep',0.2);
% Решение дифференциального уравнения методом Рунге-Кутты 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. Решение задачи 9.1 с помощью *ode23*, *ode45*.

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-Кутты более высокой точности шаг изменения переменной x намного меньше.

Рассмотрим пример решения жёсткой системы дифференциальных уравнений. Напомним читателю определение жёсткой системы дифференциальных уравнений. Система дифференциальных уравнений n -го порядка

$$\frac{dx}{dt} = Bx \quad (9.28)$$

называется жёсткой [2], если выполнены следующие условия:

- действительные части всех собственных чисел матрицы $B(n)$ отрицательны $|Re(\lambda_k)| < 0, k = 1, 2, \dots, n$;
- величина $s = \frac{\max_{1 \leq k \leq n} |Re(\lambda_k)|}{\min_{1 \leq k \leq n} |Re(\lambda_k)|}$, называемая числом жёсткости системы, велика. При исследовании на жёсткость нелинейной си-

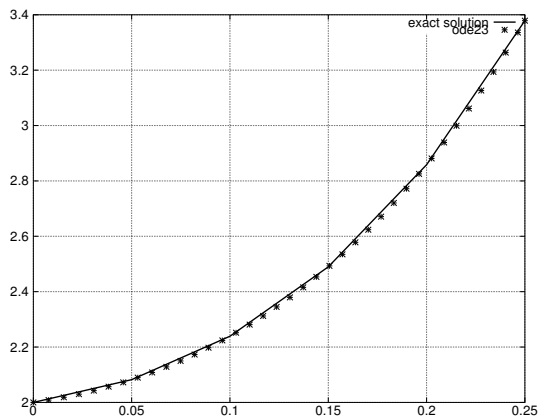


Рис. 9.5. Графики точного решения задачи 9.1 и решения, найденного с помощью функции ode23

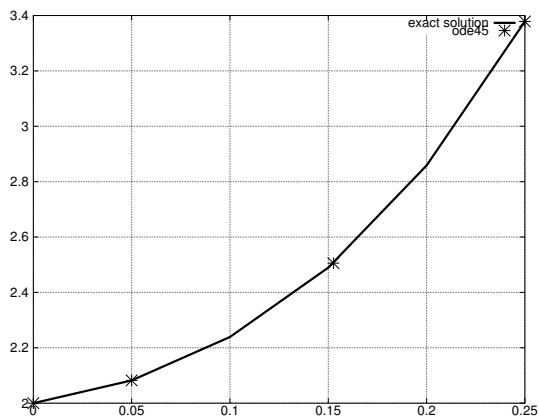


Рис. 9.6. Графики точного решения задачи 9.1 и решения, найденного с помощью функции ode45

стемы дифференциальных уравнений (9.27) в роли матрицы B будет выступать матрица частных производных.

Пример 9.2. Решить задачу Коши для жёсткой системы дифференциальных уравнений:

$$\begin{cases} \frac{dx}{dt} = \begin{pmatrix} 119.46 & 185.38 & 126.88 & 121.03 \\ -10.395 & -10.136 & -3.636 & 8.577 \\ -53.302 & -85.932 & -63.182 & -54.211 \\ -115.58 & -181.75 & -112.8 & -199 \end{pmatrix} x, \\ x(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{cases}$$

Решение задачи с комментариями представлено в листинге 9.9, на рис. 9.7 можно увидеть график решения.

```
% Функция правой части жёсткой системы дифференциальных 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
% 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 переменной 0.02,
% MaxStep — максимальное значение шага изменения 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],par);
% Построение графика решения.
plot(A,B,'-k');grid on;
```

Листинг 9.9. Решение задачи Коши для жёсткой системы дифференциальных уравнений (пример 9.2)

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 функ-

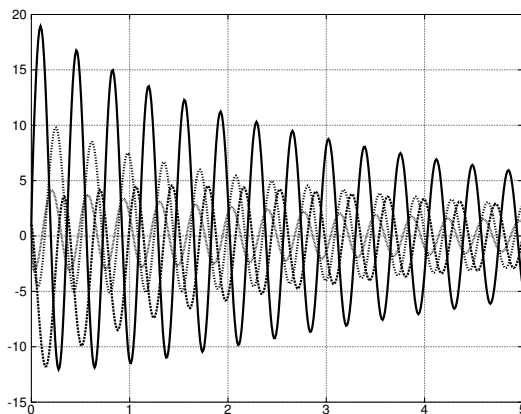


Рис. 9.7. График решения задачи 9.2

ций **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 приложения³.

Множество функций для решения 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>.

³Ещё раз напоминаем читателю, что справка по **Octave**, доступная из оболочки **qt octave** недостаточно полная.

Глава 10

Решение оптимизационных задач

В данной главе рассматриваются решение задач поиска минимума (максимума) в **Octave**. В первой части на примерах решения практических задач рассматривается функция *qp* предназначенная для поиска минимума функции одной или нескольких переменных с ограничениями. Вторая часть 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 $[x, obj, info, iter] = sqp(x_0, phi, g, h, lb, ub, maxiter, tolerance)$, которая предназначена для решения следующей оптимизационной задачи.

Найти минимум функции $\varphi(x)$ при следующих ограничениях $g(x) = 0$, $h(x) \geq 0$, $lb \leq x \leq ub$. Функция *sqp* при решении задачи оптимизации использует метод квадратичного программирования.

Аргументами функции *sqp* являются:

- x_0 — начальное приближение значения x ,
- phi — оптимизируемая функция $\varphi(x)$,
- g и h — функции ограничений $g(x) = 0$ и $h(x) \geq 0$,
- 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 *sqr* 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 *sqr* возвращает значение *info* = 101, то задача решена правильно),
- *iter* — реальное количество итераций при решении задачи.

Рассмотрим несколько примеров использования функции *sqr* при решении задач поиска экстремума функции одной переменной без ограничений.

Пример 10.1. Найти минимум функции $\varphi(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$

При решении задачи оптимизации с помощью функции *sqr* 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 $x_0 = -3$. Решение задачи представлено в листинге 10.1.

```
function obj = phi(x)
    obj = x^4+3*x^3-13*x^2-6*x+26;
endfunction
[x,obj,info,iter]=sqr(-3,@phi)
% Результаты решения
x = -3.8407
obj = -95.089
info = 101
iter = 5
```

Листинг 10.1. Поиск минимума функции (пример 10.1)

Минимум функции $\varphi(x) = -95.089$ достигается в точке $x = -3.8407$, количество итераций равно 5, параметр *info* = 101 свидетельствует о корректном решении задачи поиска минимума $\varphi(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$.

Рассмотрим пример поиска минимума функции нескольких переменных.

Пример 10.2. Найти минимум функции Розенброка¹ $f(x, y) = N(y - x^2)^2 + (1 - x^2)^2$.

¹В классическом определении функции Розенброка $N = 100$ ($N > 0$, достаточно большое число), авторы используют $N=20$. (Прим. редактора.)

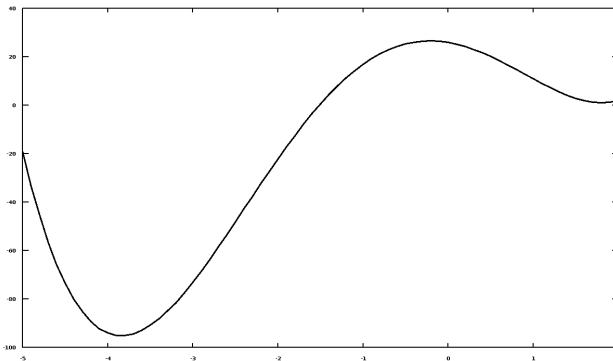


Рис. 10.1. График функции $\varphi(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$

Построим график функции Розенброка для $N = 20$ (см. листинг 10.2). График полученной поверхности приведён на рис. 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 функции Розенброка для $N = 20$

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 учитывать 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)
```

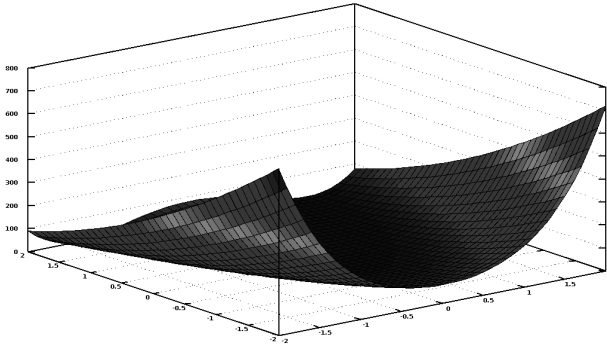


Рис. 10.2. График функции Розенброка

```
% Результаты вычислений
x =
    1.00000
    1.00000
obj = 7.1675e-13
info = 101
iter = 14
```

Листинг 10.3. Вычисление минимума функции Розенброка ($N = 20$)

Как и следовало ожидать, функция *sqr* 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 *sqr* предназначена для поиска минимума функций (как одной, так и нескольких переменных) с различными ограничениями.

Рассмотрим несколько задач поиска экстремума с ограничениями

Пример 10.3. Найти максимум и минимум функции ([1])

$$F = (x - 3)^2 - (y - 4)^2 \text{ при ограничениях: } \begin{cases} 3x + 2y \geq 7 \\ 10x - y \leq 8 \\ -18x + 4y \leq 12 \\ x \geq 0 \\ y \geq 0 \end{cases}$$

В функции *sqr* все ограничения должны быть вида ≥ 0 . Поэтому второе и третье ограничение умножим на -1 , и перенесём всё в ле-

вую часть неравенств. В результате этих несложных преобразований система ограничений примет вид:

$$g(x) = \begin{cases} 3x + 2y - 7 \geq 0 \\ -10x + y + 8 \geq 0 \\ 18x - 4y + 12 \geq 0 \\ x \geq 0 \\ y \geq 0 \end{cases}$$

Последовательно рассмотрим задачу на минимум (листинг 10.4) и максимум (листинг 10.5).

```
% В задаче на минимум функция, в которой хранится F(x) будет такой
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
% Вычисляем с помощью функции sqp
x0=[0;0];[x, obj, info, iter]=sqp(x0,@f,[],@g)
minimum=f(x)
% Результаты
x =
    1.2178
    4.1782
obj = 3.2079
info = 101
iter = 5
minimum=3.2079
```

Листинг 10.4. Нахождение минимума функции (пример 10.3)

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 итераций.

Теперь рассмотрим решение задачи на максимум. Функция *sqp* может искать только минимум. Поэтому вспомним, как задача на максимум сводится к задаче на минимум: $\max f(x) = -\min f(-x)$. Это учтено в листинге 10.5.

```
function y=f(x)
    y=(x(1)-3)^2+(x(2)-4)^2;
endfunction
% Определяем функцию, для которой минимум будет максимумом функции f
```

```

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. Нахождение максимума (пример 10.3)

Максимум достигается в точке (2,12), его величина равна 65.

Пример 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 $x_1 + x_2 + x_3 = 120, x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$.

```

% Оптимизируемая функция f
function y=f(x)
    y=4*x(1)+x(1)*x(1)+8*x(2)+x(2)*x(2)+x(3)*x(3);
endfunction
% Функция ограничения g(x) = 0
function z=g(x)
    z=x(1)+x(2)+x(3)-120;
endfunction
% Функция ограничения phi(x) >= 0
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. Решение задачи оптимизации из примера 10.4

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.

Пример 10.5. Найти максимум функции $f = -x_1^2 - x_2^2$ при ограничениях $(x_1 - 7)^2 + (x_2 - 7)^2 \leq 18$, $x_1 \geq 0$, $x_2 \geq 0$ [1].

В этой задаче 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** представлен в листинге 10.7. Функция достигает своего максимального значения -32 в точке $(4, 4)$.

```
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. Решение задачи из примера 10.5

Следующим классом оптимизационных задач, рассматриваемых в этой главе, будут задачи линейного программирования (ЗЛП).

Таблица 10.1. Содержимое белков, углеводов и жиров в продуктах

Элемент	белки	углеводы	жиры
$P1$	a_{11}	a_{12}	a_{11}
$P2$	a_{21}	a_{22}	a_{23}
$P3$	a_{31}	a_{32}	a_{33}
$P4$	a_{41}	a_{42}	a_{43}

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 начнём на примере задачи об оптимальном рационе.

Задача об оптимальном рационе. Имеется четыре вида продуктов питания: $P1$, $P2$, $P3$, $P4$. Известна стоимость единицы каждого продукта c_1, c_2, c_3, c_4 . Из этих продуктов необходимо составить пищевой рацион, который должен содержать не менее b_1 единиц белков, не менее b_2 единиц углеводов, не менее b_3 единиц жиров. Причём известно, в единице продукта $P1$ содержится a_{11} единиц белков, a_{12} единиц углеводов и a_{13} единиц жиров и т.д. (см. таблицу 10.1).

Требуется составить пищевой рацион, чтобы обеспечить заданные условия при минимальной стоимости.

Пусть x_1, x_2, x_3, x_4 — количества продуктов $P1, P2, P3, P4$. Общая стоимость рациона равна

$$L = c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 = \sum_{i=1}^4 c_i x_i \quad (10.1)$$

Сформулируем ограничение на количество белков, углеводов и жиров в виде неравенств. В одной единице продукта $P1$ содержится a_{11} единиц белков, в x_1 единицах — $a_{11}x_1$, в x_2 единицах продукта $P2$

содержится $a_{21}x_2$ единиц белка и т.д. Следовательно общее количество белков во всех четырёх типов продукта равно $\sum_{j=1}^4 a_{j1}x_j$ и должно быть не больше b_1 . Получаем первое ограничение

$$a_{11}x_1 + a_{21}x_2 + a_{31}x_3 + a_{41}x_4 \leq b_1 \quad (10.2)$$

Аналогичные ограничения для жиров и углеводов имеют вид:

$$\begin{aligned} a_{12}x_1 + a_{22}x_2 + a_{32}x_3 + a_{42}x_4 &\leq b_2 \\ a_{13}x_1 + a_{23}x_2 + a_{33}x_3 + a_{43}x_4 &\leq b_3 \end{aligned} \quad (10.3)$$

Принимаем во внимание, что x_1, x_2, x_3, x_4 положительные значения, получим ещё четыре ограничения

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0 \quad (10.4)$$

Таким образом задачу о оптимальном рационе можно сформулировать следующим образом: найти значения переменных x_1, x_2, x_3, x_4 , удовлетворяющие системе ограничений (10.2) — (10.4), 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) — (10.4) 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, \dots, x_n , при которых функция цели L (10.5) достигает своего минимального значения и удовлетворяет системе линейных ограничений (10.6).

$$L = c_1x_1 + c_2x_2 + \dots + c_nx_n = \sum_{i=1}^n c_ix_i \quad (10.5)$$

$$\sum_{j=1}^n a_{ij}x_j \leq b_i, \quad i = 1, 2, \dots, m. \quad (10.6)$$

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, \dots, x_n , при которых функция цели L (10.5) достигает своего минимального (максимального) значения и удовлетворяет системе линейных ограничений. Система ограничений может быть представлена неравенствами (10.7) или (10.8). При этом значения x могут быть, как вещественными, так и целочисленными, как положительными, так и отрицательными.

$$\sum_{j=1}^n a_{ij}x_j \leq b_i, \quad i = 1, 2, \dots, m \quad (10.7)$$

$$\sum_{j=1}^n a_{ij}x_j \geq b_i, \quad i = 1, 2, \dots, m \quad (10.8)$$

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

$[xopt, fmin, status, extra] = glpk(c, a, b, lb, ub, ctype, vartype, sense, param)$

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 содержит 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 значения вектора ub равны $+\infty$;

- *ctype* — массив символов размерности n , определяющий тип ограничения (например, (10.7) или (10.8)), элементы этого вектора могут принимать одно из следующих значений:
 - «F» — ограничение будет проигнорировано,
 - «U» — ограничение с верхней границей ($(A(i,:)*x \leq b(i))$),
 - «S» — ограничение в виде равенства ($(A(i,:)*x = b(i))$),
 - «L» — ограничение с верхней границей ($(A(i,:)*x \geq b(i))$),
 - «D» — двойное ограничение ($(A(i,:)*x \leq b(i) \text{ и } (A(i,:)*x \geq b(i))$);
- *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 ко 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 оптимизация, при *status* = 180, решение найдено и задача оптимизации решена полностью²;
- *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 4.15³ и выше).

²Если *status* \neq 180, то полученному решению доверять нельзя, подробнее о значениях переменной *status* в этом случае можно прочитать в справке.

³В последней на момент написания книги версии Octave использовалась библиотека GLPK версии 4.38.

Рассмотрим несколько примеров решения задач линейного программирования.

Пример 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:

$$\begin{cases} 3x_1 - x_2 \leq 2 \\ x_2 - 2x_3 \leq -1 \\ 4x_3 - x_4 \leq 3 \\ 5x_1 + x_4 \geq 6 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0. \end{cases}$$

Сформируем параметры функции *glpk*

$$c = \begin{pmatrix} 0 \\ -1 \\ -2 \\ 4 \end{pmatrix} \text{ — коэффициенты при неизвестных функции цели,}$$

$$a = \begin{pmatrix} 3 & -1 & 0 & 0 \\ 0 & 1 & -2 & 0 \\ 0 & 0 & 4 & -1 \\ 5 & 0 & 0 & 1 \end{pmatrix} \text{ — матрица системы ограничений,}$$

$$b = \begin{pmatrix} 2 \\ -1 \\ 3 \\ 6 \end{pmatrix} \text{ — свободные члены системы ограничений,}$$

ctype = "UUUL" — массив, определяющий тип ограничения⁴,

vartype = "CCCC" — массив, определяющий тип переменной, в данном случае все переменные вещественные,

sense = 1 — задача на минимум.

Текст программы решения задачи приведён 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)
% Результаты решения
xmin =
    1.00000
    1.00000
    1.00000
```

⁴Первые три ограничения типа «меньше», четвёртое — типа «больше»

```

1.00000
fmin = 1.00000
status = 180

```

Листинг 10.8. Решение задачи из примера 10.6

Минимальное значение $L = 1$ достигается при $x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$. Значение

переменной *status* равно 180, что свидетельствует о корректном решении задачи линейного программирования.

Пример 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:

$$\begin{cases} x_1 + x_2 \geq 2 \\ x_1 - x_2 \leq 0 \\ x_1 + x_3 \geq 2 \\ x_1 + x_2 - x_3 \leq 3 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0. \end{cases}$$

Сформируем параметры функции *gplk*:

$c = \begin{pmatrix} 1 \\ -1 \\ -3 \end{pmatrix}$ — коэффициенты при неизвестных функции цели,

$a = \begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & -1 \end{pmatrix}$ — матрица системы ограничений (три переменных

и четыре ограничения),

$b = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 3 \end{pmatrix}$ — свободные члены системы ограничений,

ctype = "LULU" — массив символов, определяющий тип ограничения⁵,

vartype = "CCC" — массив, определяющий тип переменной, в данном случае все переменные вещественные,

sense = -1 — задача на максимум.

⁵Первое и третье ограничения типа «больше», второе четвёртое — типа «меньше».

Текст программы решения задачи приведён в листинге 10.9.

```
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)
% Результаты решения
xmax =
    1.66667
    1.66667
    0.33333
fmax = -1.0000
status = 180
```

Листинг 10.9. Решение задачи из примера 10.7

Минимальное значение ⁶ $L = -6$ достигается при $x = \begin{pmatrix} 1.66667 \\ 1.66667 \\ 0.33333 \end{pmatrix}$.

Значение переменной *status* равно 180, что свидетельствует о корректном решении задачи линейного программирования.

Решим задачу 10.7, как задачу целочисленного программирования (см. 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. Решение задачи из примера 10.7 в целых числах

Значение *status* = 171 свидетельствует о корректном решении задачи целочисленного программирования, значение $L = -8$ достигается при $x = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$.

Пример 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.

Пример 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.

⁶ Авторы обращают внимание читателей в **Octave** *fmin* было равно -1 , а потом к нему необходимо было прибавить -5

Таблица 10.2. Стоимость одного посещения

Турбаза	Дельфинарий	Ботанический сад	Поход в горы
1	5	9	20
2	10	12	24

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 таблицей 10.2.

Для решения задачи введём следующие обозначения:

x_1 — число туристов первой турбазы, посещающих дельфинарий;
 x_2 — число туристов первой турбазы, посещающих ботанический сад;
 x_3 — число туристов первой турбазы, отправляющихся в поход;
 x_4 — число туристов второй турбазы, посещающих дельфинарий;
 x_5 — число туристов второй турбазы, посещающих ботанический сад;
 x_6 — число туристов второй турбазы, отправляющихся в поход.

Составим функцию цели, заключающуюся в минимизации стоимости мероприятий фирмы: $Z = 5x_1 + 9x_2 + 20x_3 + 10x_4 + 12x_5 + 24x_6$.

Руководствуясь условием задачи, определим ограничения:

$$\begin{cases} x_1 + x_4 \leq 90; \\ x_2 + x_5 \leq 170; \\ x_3 + x_6 \leq 105; \\ x_1 + x_2 + x_3 = 195; \\ x_4 + x_5 + x_6 = 165. \end{cases}$$

Количество туристов, участвующих в мероприятиях не может быть отрицательным: $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, $x_4 \geq 0$, $x_5 \geq 0$, $x_6 \geq 0$.

Кроме того, необходимо помнить, что это задача целочисленного программирования (количество туристов — число целое!!!).

В массиве x будут храниться значения x_1 , x_2 , x_3 , x_4 , x_5 и x_6 .

Сформируем параметры функции *gplk*:

$$c = \begin{pmatrix} 15 \\ 9 \\ 20 \\ 10 \\ 12 \\ 24 \end{pmatrix} \text{ — коэффициенты при неизвестных функции цели,}$$

$$a = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$
 — матрица системы ограничений шесть пе-

ременных и пять ограничений,

$$b = \begin{pmatrix} 90 \\ 170 \\ 105 \\ 195 \\ 165 \end{pmatrix}$$
 — свободные члены системы ограничений,

ctype = "UUUSS" — массив, определяющий тип ограничения⁷,

vartype = "IIIIII" — 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];
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];
b=[90; 170; 105; 195; 165];
ctype="UUUSS"; vartype="IIIIII"; sense=1;
[xmin, fmin, status]=glpk(c,a,b,[], [], ctype, vartype, sense)
% Результаты решения
xmin =
    90
     5
   100
     0
   165
     0
fmin = 4475
status = 171

```

Листинг 10.11. Решение задачи из примера 10.8

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.

⁷Первые три ограничения типа «меньше», четвёртое и пятое — типа «равно».

Таблица 10.3. Данные к задаче 10.9

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	

Пример 10.9. Для изготовления трёх видов изделий (А, Б, В) используется токарное, фрезерное, шлифовальное и сварочное оборудование. Затраты времени на обработку одного изделия каждого типа представлены в таблице 10.3. Общий фонд рабочего времени каждого вида оборудования и прибыль от реализации изделий каждого типа представлены этой же таблице. Составить план выпуска изделий для достижения максимальной прибыли [1].

Пусть x_1 — количество изделий вида А, x_2 — количество изделий вида Б, x_3 — вида В.

Прибыль от реализации всех изделий составляет

$$L = 10x_1 + 14x_2 + 12x_3 \quad (10.9)$$

Общий фонд рабочего времени фрезерного оборудования составляет $2x_1 + 4x_2 + 5x_3$. Эта величина не должна превышать 120 часов.

$$2x_1 + 4x_2 + 5x_3 \leq 120 \quad (10.10)$$

Запишем аналогичные ограничения для фонда рабочего времени токарного, сварочного, шлифовального оборудования

$$\begin{cases} x_1 + 8x_2 + 6x_3 \leq 280 \\ 7x_1 + 4x_2 + 5x_3 \leq 240 \\ 4x_1 + 6x_2 + 7x_3 \leq 360 \end{cases} \quad (10.11)$$

Таким образом получаем следующую задачу линейного программирования.

Найти такие положительные значения x_1 , x_2 , x_3 при которых функция цели L (10.9) достигает максимального значения и выполняются ограничения (10.10)–(10.11).

Теперь решим эту задачу в **Octave** с помощью функции *gplk*.

Сформируем параметры функции *gplk*:

$c = \begin{pmatrix} 10 \\ 14 \\ 12 \end{pmatrix}$ — коэффициенты при неизвестных функции цели,

$a = \begin{pmatrix} 2 & 4 & 5 \\ 1 & 8 & 6 \\ 7 & 4 & 5 \\ 4 & 6 & 7 \end{pmatrix}$ — матрица системы ограничений (три переменных и

четыре ограничений),

$b = \begin{pmatrix} 120 \\ 280 \\ 240 \\ 360 \end{pmatrix}$ — свободные члены системы ограничений,

ctype = "UUUU" — массив, определяющий тип ограничения⁸,

vartype = "III" — массив, определяющий тип переменной, в данном случае все переменные целые,

sense = -1 — задача на максимум.

Решение задачи в **Octave** представлено в листинге 10.12

```
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. Решение задачи из примера 10.9

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.

Пример 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

⁸Все четыре ограничения типа «меньше».

Таблица 10.4. Нормы затрат ресурсов к примеру 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		

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.

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

$$L = 315x_1 + 278x_2 + 573x_3 + 370x_4 \quad (10.12)$$

Ограничения на фонд рабочего времени формируют следующие ограничения

$$\begin{cases} 550x_1 + 620x_3 \leq 64270 \\ 40x_1 + 30x_2 + 20x_3 + 20x_4 \leq 4800 \\ 86x_1 + 110x_2 + 150x_3 + 52x_4 \leq 22360 \\ 160x_1 + 92x_2 + 158x_3 + 128x_4 \leq 26240 \\ 158x_2 + 30x_3 + 50x_4 \leq 7900 \end{cases} \quad (10.13)$$

Ограничение на возможное использование комплектующих изделий

$$3x_1 + 4x_2 + 3x_3 + 3x_4 \leq 520 \quad (10.14)$$

Ограничение на выполнение сборочно-наладочных работ

$$4.5x_1 + 4.5x_2 + 4.5x_3 + 4.5x_4 \leq 720 \quad (10.15)$$

Ограничения на возможный выпуск изделий каждого вида

$$x_2 \geq 40, x_3 \leq 120, x_1 \geq 0, x_3 \geq 0, x_4 \geq 0 \quad (10.16)$$

Сформулируем задачу линейного программирования.

Найти значения x_1, x_2, x_3 и x_4 при которых функция цели L (10.12) достигает своего максимального значения и выполняются ограничения (10.13)–(10.16).

Рассматриваемая задача из широко известной книги [1] была интересна авторам в связи с тем, что ещё 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 её в **Octave** и посмотрим сколько времени у нас на это уйдёт.

Сформируем параметры функции *gplk*:

$$c = \begin{pmatrix} 315 \\ 278 \\ 573 \\ 370 \end{pmatrix} \text{ — коэффициенты при неизвестных функции цели,}$$

$$a = \begin{pmatrix} 550 & 0 & 620 & 0 \\ 40 & 30 & 20 & 20 \\ 86 & 110 & 150 & 52 \\ 160 & 92 & 158 & 128 \\ 0 & 158 & 30 & 50 \\ 3 & 4 & 3 & 3 \\ 4.5 & 4.5 & 4.5 & 4.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ — матрица системы ограничений (четыре}$$

переменных и двенадцать ограничений),

$$b = \begin{pmatrix} 64270 \\ 4800 \\ 22360 \\ 26240 \\ 7900 \\ 520 \\ 720 \\ 40 \\ 120 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{— свободные члены системы ограничений,}$$

ctype = "UUUUUUULULLL"— массив символов, определяющий тип ограничения⁹,

vartype = "IIII"— массив, определяющий тип переменной, в данном случае все переменные целые (задача целочисленного программирования),

sense = -1 — задача на максимум.

Программа решения задачи в **Octave** представлена в листинге 10.13.

```
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;4800;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. Решение задачи из примера 10.10

Для получения максимальной прибыли ($fmax = 492$) необходимо произвести 65 единиц изделий первого типа, 40 — второго, 46 — тре-

⁹Первые три ограничения типа «меньше», четвёртое и пятое — типа «равно».

тьего и 4 — четвёртого. Значение параметра *status* = 171 говорит о корректности решения задачи линейного программирования.

Для написания программы и решения довольно сложной задачи в **Octave** понадобилось буквально пару минут.

Подобным образом можно решать всевозможные 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 гибкой и мощной функцией *gplk* для решения задач линейного программирования из всех свободных и проприетарных программ.

Рассмотренных двух 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 ему обратиться к пакету расширений **Minimization** для GNU **Octave**. Краткое описание функций этого пакета на английском языке с некоторыми примерами приведено на странице <http://octave.sourceforge.net/optim/overview.html>.

Глава 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 четырёх 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 были получены некая экспериментальная зависимость $y(x)$, представленная в таблице 11.1. 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

Таблица 11.1. Данные эксперимента

X	x_1	x_2	x_3	\dots	x_{n-1}	x_n
Y	y_1	y_2	y_3	\dots	y_{n-1}	y_n

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 (см. рис. 11.1):

$$\begin{aligned}
 S(a_1, a_2, \dots, a_k) &= \sum_{i=1}^n (y_i - Y_i)^2 = \\
 &= \sum_{i=1}^n (y_i - f(x_i, a_1, a_2, \dots, a_k))^2 \rightarrow \min
 \end{aligned}
 \tag{11.1}$$

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 *sqr*.
 2. Использовать специализированную функцию *polyfit*(x, y, n).
 3. Используя аппарат высшей математики, составить и решить систему алгебраических уравнений для определения коэффициентов a_i .

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

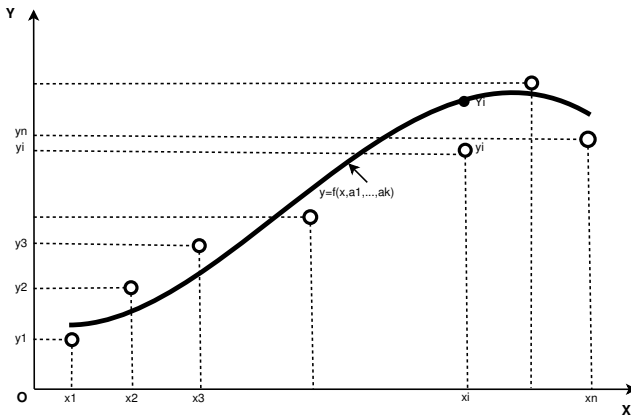


Рис. 11.1. Геометрическая интерпретация МНК

алгебраических уравнений:

$$\begin{cases} \frac{\partial S}{\partial a_1} = 0 \\ \frac{\partial S}{\partial a_2} = 0 \\ \dots\dots\dots \\ \frac{\partial S}{\partial a_k} = 0 \end{cases} \quad (11.2)$$

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.

$$\begin{cases} \sum_{i=1}^n 2(y_i - f(x_i, a_1, a_2, \dots, a_k)) \frac{\partial f}{\partial a_1} = 0 \\ \sum_{i=1}^n 2(y_i - f(x_i, a_1, a_2, \dots, a_k)) \frac{\partial f}{\partial a_2} = 0 \\ \dots\dots\dots \\ \sum_{i=1}^n 2(y_i - f(x_i, a_1, a_2, \dots, a_k)) \frac{\partial f}{\partial a_k} = 0 \end{cases} \quad (11.3)$$

Составим систему (11.3) для наиболее часто используемых функций.

11.2.1 Подбор коэффициентов линейной зависимости

Для подбора параметров линейной функции $Y = a_1 + a_2x$, составим функцию (11.1) для линейной зависимости:

$$S(a_1, a_2) = \sum_{i=1}^n (y_i - a_1 - a_2x_i)^2 \rightarrow \min. \quad (11.4)$$

Продифференцировав функцию S по a_1 и a_2 , получим систему уравнений:

$$\begin{cases} 2 \sum_{i=1}^n (y_i - a_1 - a_2 x_i) (-1) = 0 \\ 2 \sum_{i=1}^n (y_i - a_1 - a_2 x_i) (-x_i) = 0 \end{cases} \Rightarrow \begin{cases} a_1 n + a_2 \sum_{i=1}^n x_i = \sum_{i=1}^n y_i \\ a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n y_i x_i \end{cases}, \quad (11.5)$$

решив которую, определим коэффициенты функции $Y = a_1 + a_2 x$:

$$\begin{cases} a_1 = \frac{1}{n} \left(\sum_{i=1}^n y_i - a_2 \sum_{i=1}^n x_i \right) \\ a_2 = \frac{n \sum_{i=1}^n y_i x_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2} \end{cases}. \quad (11.6)$$

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 $S(a_1, a_2, a_3)$ (11.1):

$$S(a_1, a_2, a_3) = \sum_{i=1}^n (y_i - a_1 - a_2 x_i - a_3 x_i^2)^2 \rightarrow \min. \quad (11.7)$$

После дифференцирования S по a_1 , a_2 и a_3 получим систему линейных алгебраических уравнений:

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^n x_i + a_3 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n y_i \\ a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 + a_3 \sum_{i=1}^n x_i^3 = \sum_{i=1}^n y_i x_i \\ a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 + a_3 \sum_{i=1}^n x_i^4 = \sum_{i=1}^n y_i x_i^2 \end{cases} \quad (11.8)$$

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 $S(a_1, a_2, a_3, a_4)$:

$$S(a_1, a_2, a_3, a_4) = \sum_{i=1}^n (y_i - a_1 - a_2 x_i - a_3 x_i^2 - a_4 x_i^3)^2 \rightarrow \min \quad (11.9)$$

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 прологарифмируем¹ выражение $Y = ax^b e^{cx}$: $\ln Y = \ln a + b \ln x + cx$. Сделаем замену $Y1 = \ln Y$, $A = \ln a$, после этого функция примет вид: $Y1 = A + b \ln x + cx$.

Составим функцию $S(A, b, c)$ по формуле (11.1):

$$S(A, b, c) = \sum_{i=1}^n (Y1_i - A - b \ln x_i - cx_i)^2 \rightarrow \min \quad (11.15)$$

После дифференцирования получим систему трёх линейных алгебраических уравнений для определения коэффициентов A , b , c .

$$\begin{cases} nA + b \sum_{i=1}^n \ln x_i + c \sum_{i=1}^n x_i = \sum_{i=1}^n Y1_i \\ A \sum_{i=1}^n \ln x_i + b \sum_{i=1}^n (\ln x_i)^2 + c \sum_{i=1}^n x_i \ln x_i = \sum_{i=1}^n Y1_i \ln x_i \\ A \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i \ln x_i + c \sum_{i=1}^n x_i^2 = \sum_{i=1}^n Y1_i x_i \end{cases} \quad (11.16)$$

После решения системы (11.16) 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 её прологарифмировать $\ln Y = \ln ax^b = \ln a + b \ln x$. После чего замена $Z = \ln Y$, $X = \ln x$, $A = \ln a$ приводит заданную функцию к линейному виду $Z = bX + A$, где коэффициенты A и b вычисляются по формулам (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) найдём A и b , а затем вычислим $a = e^A$.

Рассмотрим ещё ряд зависимостей, которые сводятся к линейной.

¹Можно и не проводить предварительное логарифмирование выражения $Y = ax^b e^{cx}$, однако в этом случае получаемая система уравнений будет нелинейной, которую решать сложнее.

Для подбора параметров функции $Y = \frac{1}{ax+b}$ сделаем замену $Z = \frac{1}{Y}$. В результате получим линейную зависимость $Z = ax + b$. Функция $Y = \frac{x}{ax+b}$ заменами $Z = \frac{1}{Y}$, $X = \frac{1}{x}$ сводится к линейной $Z = a + bX$. Для определения коэффициентов функциональной зависимости $Y = \frac{1}{ae^{-x}+b}$ необходимо сделать следующие замены $Z = \frac{1}{Y}$, $X = e^{-x}$. В результате также получим линейную функцию $Z = aX + b$.

Аналогичными приёмами (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 *sqr*.

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

$$S = \sum_{i=1}^n (y_i - f(x_i, a_1, a_2, \dots, a_k))^2 \quad (11.17)$$

называемая *суммарной 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 *коэффициентами регрессии* и определяются формулами (11.6).

Чем меньше величина $S = \sum_{i=1}^n (y_i - a_1 - a_2x_i)^2$, тем более обоснованно предположение, что экспериментальные данные описываются линейной функцией. Существует показатель, характеризующий тесноту линейной связи между x и y , который называется *коэффициентом корреляции* и рассчитывается по формуле:

$$r = \frac{\sum_{i=1}^n (x_i - M_x)(y_i - M_y)}{\sqrt{\sum_{i=1}^n (x_i - M_x)^2 \sum_{i=1}^n (y_i - M_y)^2}}, \quad M_x = \frac{\sum_{i=1}^n x_i}{n}, \quad M_y = \frac{\sum_{i=1}^n y_i}{n} \quad (11.18)$$

Значение коэффициента корреляции удовлетворяет соотношению $-1 \leq r \leq 1$.

Чем меньше отличается абсолютная величина r от единицы, тем ближе к линии регрессии располагаются экспериментальные точки. Если $|r| = 1$, то все экспериментальные точки находятся на линии регрессии. Если коэффициент корреляции близок к нулю, то это означает, что между x и y не существует линейной связи, но между ними может существовать зависимость, отличная от линейной.

Для того, чтобы проверить, значимо ли отличается от нуля коэффициент корреляции, можно использовать *критерий Стьюдента*. Вычисленное значение критерия определяется по формуле:

$$t = r \sqrt{\frac{n-2}{1-r^2}} \quad (11.19)$$

Рассчитанное по формуле (11.19) значение t сравнивается со значением, взятым из *таблицы распределения Стьюдента* (см. 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

$k \backslash p$	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

Таблица 11.2: — продолжение

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 в случае нелинейной зависимости пользуются *индекс корреляции*. Он показывает тесноту связи между фактором x и зависимой переменной y и рассчитывается по формуле:

$$R = \sqrt{1 - \frac{\sum_{i=1}^n (y_i - Y_i)^2}{\sum_{i=1}^n (y_i - M_y)^2}}, \quad (11.20)$$

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 R практически равен нулю. Если коэффициент корреляции r является мерой тесноты связи только для линейной формы

Таблица 11.3. Растворимость азотнокислого натрия

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

связи, то индекс корреляции R — как для линейной, так и для нелинейной. При прямолинейной связи коэффициент корреляции по своей абсолютной величине равен индексу корреляции: $|r| = R$.

11.5 Подбор зависимостей методом наименьших квадратов в Octave

11.5.1 Функции Octave, используемые для подбора зависимости МНК

Для решения задач подбора аналитических зависимостей по экспериментальным данным можно использовать следующие функции Octave:

- $\text{polyfit}(x, y, k)$ — 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;
- $\text{sqp}(x0, phi, g, h, lb, ub, maxiter, tolerance)$ — функция поиска минимума (функция подробно описана в десятой главе);
- $\text{cor}(x, y)$ — функция вычисления коэффициента корреляции (x — массив абсцисс экспериментальных точек, y — массив ординат экспериментальных точек);
- $\text{mean}(x)$ — функция вычисления среднего арифметического.

11.5.2 Примеры решения задач

Пример 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 $t = 32^\circ\text{C}$ в случае линейной зависимости и найти коэффициент корреляции.

Решение задачи 11.1 с комментариями приведено в листинге 11.1.

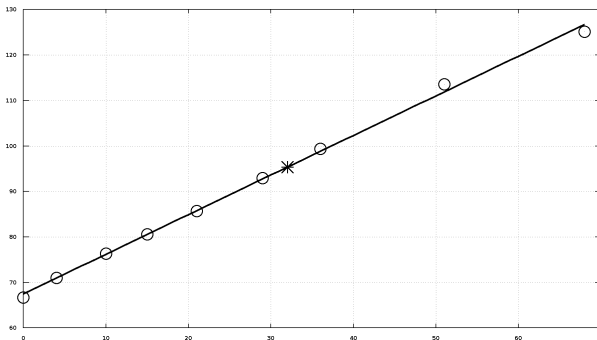


Рис. 11.2. Иллюстрация к примеру 11.1

```
% 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,'*k')
grid on;
% Вычисление коэффициента корреляции
k=cov(x,y)
% Результаты вычислений
a = 0.87064    67.50779
yt = 95.368
k = 1
```

Листинг 11.1. Решение к примеру 11.1

На рис. 11.2 приведено графическое решение этой задачи, изображены экспериментальные точки, линия регрессии $y = a_1x + a_2$, на котором отмечена точка $t = 32$.

Пример 11.2. В результате эксперимента получена табличная зависимость $y(x)$ (см. табл. 11.4). Подобрать аналитическую зависимость $Y = ax^be^{cx}$ методом наименьших квадратов. Вычислить ожидаемое значение в точках 2, 3, 4. Вычислить индекс корреляции. Ре-

Таблица 11.4. Данные к примеру 11.2

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

шение задачи подбора параметров функции $f(x) = ax^b e^{cx}$ в **Octave** возможно двумя способами:

1. Решение задачи путём поиска минимума функции (11.15)². После чего надо пересчитать значение коэффициента a по формуле $a = e^A$.
2. Формирование системы линейных алгебраических уравнений (11.16)³ и её решение.

Рассмотрим последовательно оба варианта решения задачи.

Способ 1.

Функция (11.15) реализована в **Octave** с помощью функции `f_mnk`. Полный текст программы решения задачи способом 1 с комментариями приведён на листинге 11.2. Вместо коэффициентов A , b , c из формул (11.15)–(11.16) 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];
```

²Может не получится решать задачу подбора зависимости «в лоб» путём оптимизации функции $S(a, b, c) = \sum_{i=1}^n (y_i - ax_i^b e^{cx_i})^2$, это связано с тем, что при решении задачи оптимизации с помощью *sqr* итерационными методами может возникнуть проблема возведения отрицательного числа в дробную степень [3, с.70–71]. Да и с точки зрения математики, если есть возможность решать линейную задачу вместо нелинейной, то лучше решать линейную.

³Следует помнить, что при отрицательных значениях y необходимо будет решать проблему замены $Y = \ln y$.

```

% 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 11.15 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,'sk');
% Вычисление индекса корреляции.
R=sqrt(1-sum((y-yr).^2)/sum((y-mean(y)).^2))
% Результаты вычислений
c =
    0.33503
    0.90183
   -0.69337
sum1=0.090533
x2= 2 3 4
y2= 0.65272 0.47033 0.30475
R= 0.99533

```

Листинг 11.2. Решение к примеру 11.2 способ 1.

Таким образом подобрана зависимость $Y = 0.33503x^{0.90183}e^{-0.9337x}$. 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 11.16. Решение с комментариями приведено в 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;
% 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 (11.16)
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 gfef.
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.
yг=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,'sk');
% Вычисление индекса корреляции.
R=sqrt(1-sum((y-yг).^2)/sum((y-mean(y)).^2))

```

Листинг 11.3. Решение к примеру 11.2 способ 2.

Пример 11.3. В результате эксперимента получена табличная зависимость $y(x)$ (см. табл. 11.5). Подобрать аналитические зависимости $f(x) = b_1 + b_2x + b_3x^2 + b_4x^3 + b_5x^4 + b_6x^5$, $g(x) = a_1 + a_2x + a_3x^2 + a_4x^3 + a_5x^5$ и $\varphi(x) = c_1 + c_2x + c_4x^3 + c_5x^5$ методом наименьших квадратов. Пользуясь значением индекса корреляции выбрать наилучшую из них, 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.

Как рассматривалось ранее, решать задачу подбора параметров полинома методом наименьших квадратов в **Octave** можно тремя способами.

1. Сформировать и решить систему уравнений (11.3).

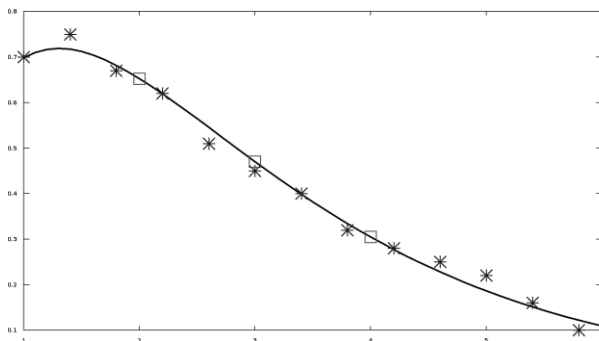


Рис. 11.3. График к примеру 11.2: экспериментальные точки и подобранный методом наименьших квадратов зависимость

Таблица 11.5. Данные к примеру 11.3

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

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 *sqr*.

3. Использовать функцию *polyfit*.

Чтобы продемонстрировать использование всех трёх 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 *sqr*.

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 $S(a_1, a_2, a_3, a_4, a_5) = \sum_{i=1}^n (y_i - a_1 - a_2x_i - a_3x_i^2 - a_4x_i^3 - a_5x_i^5)^2$. 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:

$$\left\{ \begin{array}{l} a_1 n + a_2 \sum_{i=1}^n x_i + a_3 \sum_{i=1}^n x_i^2 + a_4 \sum_{i=1}^n x_i^3 + a_5 \sum_{i=1}^n x_i^5 = \sum_{i=1}^n y_i \\ a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 + a_3 \sum_{i=1}^n x_i^3 + a_4 \sum_{i=1}^n x_i^4 + a_5 \sum_{i=1}^n x_i^6 = \sum_{i=1}^n y_i x_i \\ a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 + a_3 \sum_{i=1}^n x_i^4 + a_4 \sum_{i=1}^n x_i^5 + a_5 \sum_{i=1}^n x_i^7 = \sum_{i=1}^n y_i x_i^2 \\ a_1 \sum_{i=1}^n x_i^3 + a_2 \sum_{i=1}^n x_i^4 + a_3 \sum_{i=1}^n x_i^5 + a_4 \sum_{i=1}^n x_i^6 + a_5 \sum_{i=1}^n x_i^8 = \sum_{i=1}^n y_i x_i^3 \\ a_1 \sum_{i=1}^n x_i^5 + a_2 \sum_{i=1}^n x_i^6 + a_3 \sum_{i=1}^n x_i^7 + a_4 \sum_{i=1}^n x_i^8 + a_5 \sum_{i=1}^n x_i^{10} = \sum_{i=1}^n y_i x_i^5 \end{array} \right. \quad (11.21)$$

Решив систему (11.21), найдём коэффициенты a_1, a_2, a_3, a_4 и a_5 функции $g(x) = a_1 + a_2x + a_3x^2 + a_4x^3 + a_5x^5$.

Для поиска функциональной зависимости вида $\varphi(x) = c_1 + c_2x + c_4x^3 + c_5x^5$ необходимо будет найти такие значения c_1, c_2, c_3, c_4 , при которых функция

$$S(c_1, c_2, c_3, c_4) = \sum_{i=1}^n (y_i - c_1 - c_2x_i - c_3x_i^3 - c_4x_i^5)^2 \quad (11.22)$$

принимала бы наименьшее значение.

После вывода необходимых формул приступим к реализации в **Octave**. Текст программы с очень подробными комментариями приведён в листинге 11.4.

```
% Функция для подбора зависимости  $f_i(x)$  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 (11.22).
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
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 храниться в переменной В.
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 (11.21) для подбора функции g(x). Здесь GGL —
% матрица коэффициентов, H — вектор правых частей системы (11.21),
% G — первые 4 строки и 4 столбца матрицы коэффициентов, G1 — пятый
% столбец матрицы коэффициентов, G2 — пятая строка матрицы коэфф-тов.
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 (11.21) из матриц
% G, G1 и G2.
GGL=[G G1'; G2]
H(5)=sum(y.*x.^5);
% Решение системы (11.21) методом обратной матрицы и
% формирование коэффициентов A функции g(x).
A=inv(GGL)*H'
% Подбор коэффициентов зависимости fi(x) методом наименьших квадратов,
% используя функцию sqr. Коэффициенты функции будут храниться в перемен-
% ной C. Задание начального значения вектора C, при неправильном его опре-
% делении, экстремум функции может быть найден неправильно.
C=[2;1;3;1];
% Поиск вектора C, при котором функция (11.22) достигает своего
% минимального значения, вектор C — коэффициенты функции fi.
C=sqr(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 первой функции f(x) в заданных точках.
yr1=polyval(B,x);
% Вычисление значений 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 третьей 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 точках 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;experiment";,X1,Y1,'-b;f(x);',X1,Y2,'dr;g(x);',X1
,Y3,'ok;fi(x);',z,yz,'sb;f(z);');
grid();
% 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

```

Листинг 11.4. Решение к примеру 11.3

На рисунке 11.4 представлено графическое решение задачи.

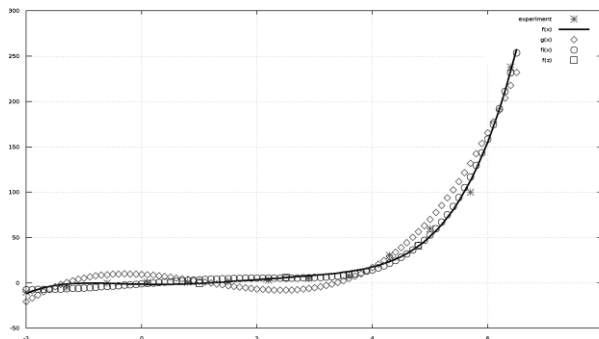


Рис. 11.4. Графическое решение к примеру 11.3

Рассмотренная задача демонстрирует основные приёмы подбора зависимости методом наименьших квадратов. Авторы рекомендуют внимательно рассмотреть её для понимания методов решения подобных задач в **Octave**.

В заключении авторы позволяют 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 *sqr* — довольно мощная, лучше использовать методы и функции решения систем линейных алгебраических уравнений, функцию *polyfit*, чем функцию *sqr*. Этот совет связан с тем, что функция *sqr* — приближённые итерационные алгоритмы, поэтому получаемый результат иногда может быть менее точен, чем при точных методах решения систем линейных алгебраических уравнений. Но, иногда, именно функция *sqr* — 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).

Глава 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 $n + 1$ точка $x_0, x_1, x_2, \dots, x_n$ ($a = x_0, b = x_n$), называемые узлами интерполяции, и значения неизвестной функции $f(x)$ в этих точках

$$f(x_0) = y_0, f(x_1) = y_1, f(x_2) = y_2, \dots, f(x_n) = y_n \quad (12.1)$$

Требуется построить интерполирующую функцию $F(x)$, которая в узлах интерполяции принимает те же значения, что и $f(x)$

$$F(x_0) = y_0, F(x_1) = y_1, F(x_2) = y_2, \dots, F(x_n) = y_n \quad (12.2)$$

Таблица 12.1. Таблица разделённых 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}	...	$y_{012...n}$

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: $A_1 = \frac{y_0 - y_1}{x_0 - x_1} = y_{01}$, где y_{01} — разделённая разность первого порядка, которая стремится к первой производной функции при $x_1 \rightarrow x_0$. По аналогии вводятся и другие разделённые разности первого порядка: $y_{02} = \frac{y_0 - y_2}{x_0 - x_2}$, $y_{03} = \frac{y_0 - y_3}{x_0 - x_3}$, ..., $y_{0n} = \frac{y_0 - y_n}{x_0 - x_n}$.

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 $A_2 = y_{012} = \frac{y_{01} - y_{02}}{x_1 - x_2}$, здесь y_{012} — разделённая разность второго порядка, эта величина стремится ко второй производной при $x_1 \rightarrow x_2$. Аналогично вводятся

$$y_{013} = \frac{y_{01} - y_{03}}{x_0 - x_3}, y_{014} = \frac{y_{01} - y_{04}}{x_1 - x_4}, \dots, y_{01n} = \frac{y_{01} - y_{0n}}{x_1 - x_n}.$$

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 $A_n = y_{012...n} = \frac{y_{012...n-1} - y_{012...n}}{x_{n-1} - x_n}$.

Полученные результаты запишем в табл. 12.1

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

Ещё одно представление интерполяционного полинома степени n предложил Лагранж:

$$F(t) = \sum_{i=0}^n y_i \prod_{\substack{j=0 \\ j \neq i}}^n \frac{t - x_j}{x_i - x_j} \quad (12.6)$$

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

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 $k = \text{length}(x) - 1$) для вычисления коэффициентов полинома, после чего с помощью функции *polyval*(A, t) вычислить значение полинома в необходимых точках.

```
% x — массив абсцисс экспериментальных точек, y — массив ординат
% экспериментальных точек, t — точка в которой требуется найти значение.
function s=kanon(x,y,t)
    n=length(x); % Вычисление количества точек в массивах x и y
    % Формирование коэффициентов системы уравнений (12.4)
```

```

    for i=1:n
        for j=1:n
            A(i,j)=x(i).^(j-1);
        end
    end
    a=A^(-1)*y'; % Решение системы уравнений (12.4)
    % 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. Функция *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=newton(x,y,t)
    n=length(x); % Вычисление количества точек в массивах x и y
    % Запись в первый столбец матрицы разделённых разностей вектора y
    for i=1:n
        C(i,1)=y(i);
    end
    for i=2:n % Формирование матрицы разделённых разностей
        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
    for i=1:n % Формирование массива коэффициентов полинома Ньютона
        A(i)=C(i,i);
    end
    s=0; % Расчёт значения полинома в точке t по формуле (12.5)
    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. Функция *newton* для вычисления полинома Ньютона в точке t по экспериментальным значениям (x_i, y_i)

Таблица 12.2. Данные к примеру 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

```

%  $x$  — массив абсцисс экспериментальных точек,  $y$  — массив ординат
% экспериментальных точек,  $t$  — точка в которой требуется найти значение.
function s=lagrang(x,y,t)
    n=length(x); % Вычисление количества точек в массивах  $x$  и  $y$ 
    % Расчёт суммы произведений по формуле (12.6)
    % для вычисления значения полинома Лагранжа в точке  $t$ 
    s=0;
    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. Функция *lagrang* для вычисления полинома Лагранжа в точке t по экспериментальным значениям (x_i, y_i)

Пример 12.1. В результате эксперимента получена табличная зависимость $y(x)$ (см. табл. 12.2). Построить интерполяционный полином. Вычислить ожидаемое значение в точках 0.5, 0.6 и 0.7, построить график зависимости.

Решение с подробными комментариями представлено в листинге 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 примера 12.1. Подобным образом можно подбирать коэффициенты интерполяционного полинома и для других задач.

```

function s=kanon(x,y,t)
    n=length(x); % Вычисление количества точек в массивах  $x$  и  $y$ 
    for i=1:n % Формирование коэффициентов системы уравнений (12.4)
        for j=1:n
            A(i,j)=x(i).^(j-1);

```

```

        end
    end
    a=A^(-1)*y'; % Решение системы уравнений (12.4)
    s=0; % Вычисление значения полинома в точке  $t$  по формуле (12.3)
    for i=1:n
        s=s+a(i)*t^(i-1);
    end
end
function s=newton(x,y,t)
    n=length(x); % Вычисление количества точек в массивах  $x$  и  $y$ 
    for i=1:n % Запись в первый столбец  $C$  разделённых разностей  $y$ 
        C(i,1)=y(i);
    end
    for i=2:n % Формирование матрицы разделённых разностей
        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
    for i=1:n % Формирование массива коэффициентов полинома Ньютона
        A(i)=C(i,i);
    end
    s=0; % Расчёт значения полинома в точке  $t$  по формуле (12.5)
    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)
    n=length(x); % Вычисление количества точек в массивах  $x$  и  $y$ 
    s=0; % Расчёт значения полинома Лагранжа в точке  $t$  по формуле (12.6)
    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
%  $x$  и  $y$  — массивы абсцисс и ординат экспериментальных точек примера 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];

```

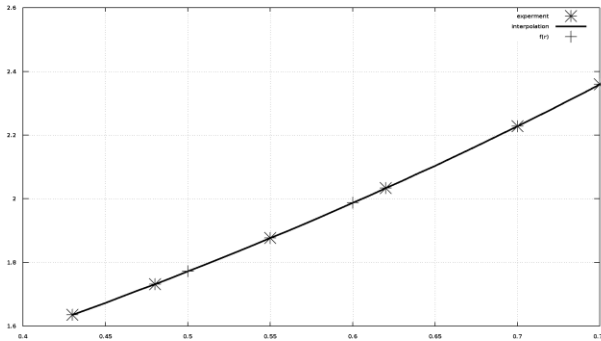



Рис. 12.1. Полиномиальная интерполяция (пример 12.1)

```

r=[0.5 0.6 0.7] % точки для которых надо вычислить ожидаемые значения
for i=1:3
% Вычисление i-го ожидаемого значения интерполяционного полинома Ньютона
rsn(i)=newton(x,y,r(i));
% Вычисление i-го значения канонического интерполяционного полинома
rsk(i)=kanon(x,y,r(i));
% Вычисление i-го значения интерполяционного полинома Лагранжа
rsl(i)=lagrang(x,y,r(i));
end
rsn
rsk
rsl
% Вычисление ожидаемых значений 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 вычислений
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. Решение примера 12.1

Полиномиальная интерполяция не всегда даёт удовлетворительные результаты при аппроксимации зависимостей. При представлении полиномами возможна большая погрешность на концах этих кривых. Несмотря на выполнение условий в узлах, интерполяционная функция может иметь значительное отклонение от аппроксимируемой кривой между узлами. При этом повышение степени интерполяционного полинома приводит не к уменьшению, а к увеличению погрешности. Решение этой проблемы предложено теорией сплайн-интерполяции (от английского слова *spline* — рейка, линейка).

12.2 Интерполяция сплайнами

Рассмотрим один из наиболее распространённых вариантов интерполяции кубическими сплайнами. Было установлено [2], что недеформируемая линейка между соседними углами проходит по линии, удовлетворяющей уравнению

$$\varphi^{IV}(x) = 0. \quad (12.7)$$

Функцию $\varphi(x)$ будем использовать для интерполяции зависимости $y(x)$, заданной на интервале (a, b) в узлах $a = x_0, x_1, \dots, x_n = b$ значениями y_0, y_1, \dots, y_n .

Кубическим сплайном, интерполирующим на отрезке $[a, b]$ данную функцию $y(x)$, называется функция [2]

$$g_k(s) = a_k + b_k(s - x_k) + c_k(s - x_k)^2 + d_k(s - x_k)^3, \quad (12.8)$$

$$s \in [x_{k-1}, x_k], k = 1, 2, \dots, n,$$

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 $g_1''(a) = g_n''(b) = 0$.

Для построения интерполяционного сплайна необходимо найти $4n$ коэффициента $a_k, b_k, c_k, d_k, (k = 1, 2, \dots, n)$.

Из определения сплайна получаем $n + 1$ соотношение (12.9)

$$g_1(x_0) = y_0, \quad g_k(x_k) = y_k, \quad k = 1, 2, \dots, n \quad (12.9)$$

Из условий гладкой стыковки звеньев сплайна (во внутренних узловых точках совпадают значения двух соседних звеньев сплайна¹, их первые и вторые производные) получаем ещё ряд соотношений (12.10–12.11) [2]:

$$\begin{aligned} g_{k-1}(x_{k-1}) &= g_k(x_k) \\ g'_{k-1}(x_{k-1}) &= g'_k(x_k), \\ g''_{k-1}(x_{k-1}) &= g''_k(x_k) \\ k &= 2, 3, \dots, n \end{aligned} \quad (12.10)$$

$$g''_1(x_0) = 0, \quad g''_n(x_n) = 0 \quad (12.11)$$

Соотношения (12.9)–(12.11) образуют $4n$ соотношений для нахождения коэффициентов сплайна. Подставляя выражения функций (12.8) и их производных (12.12)

$$\begin{aligned} g'_k(s) &= b_k + 2c_k(s - x_k) + 3d_k(s - x_k)^2, \\ g''_k(s) &= 2c_k + 6d_k(s - x_k) \end{aligned} \quad (12.12)$$

в соотношения (12.9)–(12.11) и принимая во внимание соотношение

$$h_k = x_k - x_{k-1}, \quad k = 1, 2, \dots, n \quad (12.13)$$

получим следующую систему уравнений (12.14)–(12.20)

$$a_1 - b_1 h_1 + c_1 h_1^2 - d_1 h_1^3 = y_0 \quad (12.14)$$

$$a_k = y_k, \quad k = 1, 2, \dots, n \quad (12.15)$$

$$a_{k-1} = a_k - b_k h_k + c_k h_k^2 - d_k h_k^3, \quad k = 2, 3, \dots, n \quad (12.16)$$

$$b_{k-1} = b_k - 2c_k h_k + 3d_k h_k^2, \quad k = 2, 3, \dots, n \quad (12.17)$$

$$c_{k-1} = c_k - 3d_k h_k, \quad k = 2, 3, \dots, n \quad (12.18)$$

$$c_1 - 3d_1 h_1 = 0 \quad (12.19)$$

$$c_n = 0 \quad (12.20)$$

Задача интерполяции свелась к решению системы (12.14–12.20). Из соотношения (12.15) следует, что все коэффициенты $a_k = y_k$, $k = 1, 2, \dots, n$. Подставив соотношения (12.14), (12.15) в (12.16) и используя фиктивный коэффициент $c_0 = 0$, получим соотношение между b_k , c_k и d_k : $b_k h_k - c_k h_k^2 + d_k h_k^3 = y_k - y_{k-1}$

¹Звеном сплайна называется функция $g_i(x)$ на интервале $[x_{i-1}, x_i]$

Отсюда коэффициенты b_k вычисляются по формуле

$$b_k = \frac{y_k - y_{k-1}}{h_k} + c_k h_k - d_k h_k^2, \quad k = 1, 2, \dots, n \quad (12.21)$$

Из (12.18) и (12.19) выразим d_k через c_k (с учётом коэффициента $c_0 = 0$)

$$d_k = \frac{c_k - c_{k-1}}{3} h_k, \quad k = 1, 2, \dots, n \quad (12.22)$$

Подставим (12.22) в (12.21)

$$b_k = \frac{y_k - y_{k-1}}{h_k} + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1}, \quad k = 1, 2, \dots, n \quad (12.23)$$

Введём обозначение

$$l_k = \frac{y_k - y_{k-1}}{h_k}, \quad k = 1, 2, \dots, n \quad (12.24)$$

после чего соотношение (12.23) примет вид:

$$b_k = l_k + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1}, \quad k = 1, 2, \dots, n \quad (12.25)$$

Подставим (12.25) и (12.22) в соотношение (12.17), получим систему относительно c_k

$$b_{k-1} c_{k-2} + 2(h_{k-1} + h_k) c_k = 3(l_k - l_{k-1}), \quad k = 2, 3, \dots, n \quad (12.26)$$

$$c_0 = 0, \quad c_n = 0 \quad (12.27)$$

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

$$\delta_1 = -\frac{1}{2} \frac{h_2}{h_1 + h_2}, \quad \lambda_1 = \frac{3}{2} \frac{l_2 - l_1}{h_1 + h_2}, \quad (12.28)$$

$$\begin{aligned} \delta_{k-1} &= -\frac{h_k}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}}, \\ \lambda_{k-1} &= \frac{3l_k - 3l_{k-1} - h_{k-1}\lambda_{k-2}}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}}, \end{aligned} \quad k = 3, 4, \dots, n \quad (12.29)$$

а затем к нахождению искоемых коэффициентов c_k по формулам обратной прогонки

$$c_{k-1} = \delta_{k-1}c_k + \lambda_{k-1}, \quad k = n, n-1, \dots, 2. \quad (12.30)$$

После нахождения коэффициентов c по формуле (12.30), находим b и d по формулам (12.22), (12.25).

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 (12.25).
- Шаг 3. Расчёт массивов прогоночных коэффициентов δ и λ по формулам (12.28), (12.29).
- Шаг 4. Расчёт массивов коэффициентов c по формуле (12.30).
- Шаг 5. Расчёт массивов коэффициентов b по формуле (12.25).
- 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

$$f_k(s) = a_k + b_k s, \quad s \in [x_{k-1}, x_k], \quad k = 1, 2, \dots, n, \quad (12.31)$$

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) .

$$a_k = y_{k-1} - \frac{y_k - y_{k-1}}{x_k - x_{k-1}} x_{k-1}, \quad b_k = \frac{y_k - y_{k-1}}{x_k - x_{k-1}} \quad (12.32)$$

Найдя коэффициенты линейного сплайна, можно рассчитать значения в любой точке интервала $[x_0, x_n]$. Линейная интерполяция даёт достаточно хорошие результаты при практическом счёте внутри интервала $[x_0, x_n]$, когда от получаемой функции не требуют дополнительных свойств (дифференцируемости и т.д.).

Таблица 12.3. Данные к примеру 12.2

Напряжение U_1 , В	132	140	150	162	170	180
Мощность P_0 , Вт	330	350	385	425	450	485

Рассмотрим реализацию сплайн-интерполяции в **Octave**. Это можно сделать запрограммировав рассмотренные выше методы сплайн-интерполирования или воспользовавшись функцией *interp1*:

interp1(x, y, xi, method), 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.

Пример 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 (см. табл. 12.3). Построить график интерполяционной зависимости. Вычислить ожидаемое значение мощности при $U_1 = 145, 155, 175$ В.

Реализуем рассмотренный ранее алгоритм кубического сплайна. Решение с комментариями представлено в листинге 12.5, а на рис. 12.2 можно увидеть графическое решение примера.

```
function [b, c, d]=coef_spline(x,y)
% Функция вычисляет коэффициенты сплайна, здесь x,y — массивы абсцисс
% и ординат экспериментальных точек, b, c, d — коэффициенты сплайна,
% рассчитываемые по формулам (12.21), (12.23), (12.27), (12.30)
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)
% Вычисляет значение кубического сплайна в точке t,
% здесь x,y — массивы абсцисс и ординат экспериментальных точек
    [b,c,d]=coef_spline(x,y);
    n=length(x);
    a=y;
    % определяем 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)=my_spline(U1,P0,x(i));
end
% Вычисление значений для построения графика сплайна.
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;')
grid on;
% Результаты работы программы
x = 145 155 175
y = 373.19 408.77 471.15

```

Листинг 12.5. Решение примера 12.2

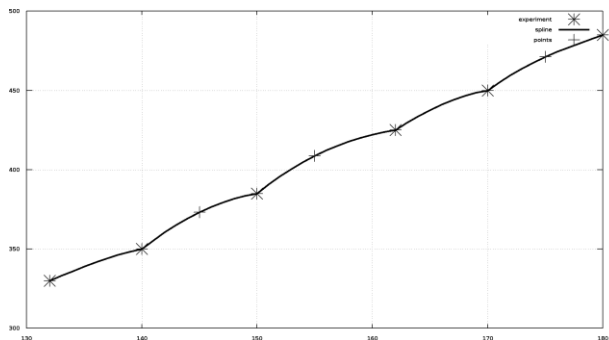


Рис. 12.2. Кубический сплайн к примеру 12.2

Таблица 12.4. Данные к примеру 12.3

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

В следующей задаче воспользуемся встроенными функциями **Octave**.

Пример 12.3. В результате эксперимента определена функция $u(x)$ (см. табл. 12.4). Построить график интерполяционной зависимости. Вычислить ожидаемое значение функции при $x = 0.308, 0.325, 0.312$.

Для решения задачи воспользуемся функцией *interp1*. В листинге 12.6 представлено решение примера 12.3, а на рис. 12.3 — графическая иллюстрация.

```
% 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 графиков линейного и кубического сплайнов.
xi=0.298:0.002:0.33;
uxis=interp1(x,u,xi,'spline');
uxil=interp1(x,u,xi,'linear');
```

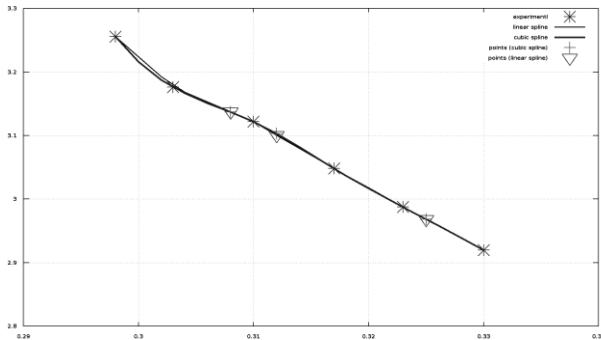



Рис. 12.3. Кубический и линейный сплайны к примеру 12.3

```
% Построение графика
plot(x,u,'*b;experiment1','xi,uxil','-r;linear spline','xi,uxis
','-b;cubic spline','x1,uls,'pr;points (cubic spline)','x1,
u1l,'<b;points (linear spline)');
axis([0.29,0.34,2.8,3.3]);
grid on;
% Результаты работы программы,
uls = 3.1370 3.1031 2.9685
u1l = 3.1374 3.1008 2.9681
```

Листинг 12.6. Решение примера 12.3

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>.

Список литературы

- [1] Акулич И.Л. Математическое программирование в примерах и задачах. —М.: Высшая школа, 1986. — 319с.
- [2] Вержбицкий В.М. Основы численных методов. —М.: Высшая школа, 2002. — 840с.
- [3] Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Free Pascal и Lazarus: Учебник по программированию. М.: Альт Линукс, 2010. -448с (электронная версия <http://docs.altlinux.org/books/2010/freepascal.pdf>).

Предметный указатель

Дифференциальные уравнения, 280

метод

Адамса, 288

Эйлера, 283

Эйлера модифицированный,
284

Хойна, 285

Кутта-Мерсона, 286

Милна, 288

Милна модифицированный,
288

Рунге-Кутта, 284

Рунге-Кутта четвёртого по-
рядка, 285

Рунге-Кутта второго поряд-
ка, 285

система, 280

задача Коши, 282

жёсткая система, 300

Дифференцирование, 262

Экстраполирование, 347

Функция

Файлы

dlmread, 73

dlmwrite, 74

fclose, 68

feof, 68

fopen, 66

fprintf, 66

fread, 75

frewind, 76

fscanf, 68

fseek, 76

fteail, 76

fwrite, 76

type, 74

целочисленная

ceil, 33

fix, 32

floor, 33

rem, 33

round, 33

sign, 33

элементарная

abs, 34

gcd, 34

lcm, 34

log10, 34

log2, 34

pow2, 34

rats, 34

sqrt, 33

команда

function, 80

комплексные

angle, 35

conj, 35

imag, 35

real, 35

строки

char, 62

deblank, 62

findstr, 63

int2str, 57, 62

lower, 63

mat2str, 63

num2str, 57, 63

sprintf, 63

sscanf, 63

str2double, 63

str2num, 64

strcat, 57, 64

strcmp, 64

strcmpi, 64

- strjust, 64
- strncmp, 64
- strrep, 64
- strtok, 65
- upper, 65
- aCos, 44
- acos, 31
- acot, 31
- ascs, 31
- asec, 31
- aSin, 44
- asin, 31
- aTan, 44
- atan, 31
- axes, 131
- axis, 94
- bar, 108
- cla, 135
- clf, 135
- comet, 125
- conv, 242
- cor, 335
- Cos, 44
- cos, 31
- cosh, 32
- cot, 31
- coth, 32
- csc, 31
- csch, 32
- cumtrapz, 271, 273
- cylinder, 121
- deconv, 242
- delete, 93, 129, 135
- differentiate, 262
- ellipsoid, 121
- ex_matrix, 203
- Exp, 44
- exp, 32
- expand, 45
- feval, 83
- figure, 93, 128
- fplot, 101
- fsolve, 255, 257
- fzero, 249
- gca, 127
- gcf, 127
- gco, 127
- get, 128
- glpk, 313
- hold, 114
- interp1, 358
- legend, 94
- length, 84
- Log, 44
- log, 32
- mean, 335
- mesh, 111
- meshgrid, 111
- mod, 52
- ode23, 298
- ode2r, 298
- ode45, 298
- ode5r, 298
- odeset, 299
- pause, 93
- Pi, 44
- plot, 88, 90
- plotyy, 99
- polar, 103
- poly, 246
- polyder, 245
- polyfit, 335, 349
- polyint, 246
- polyva, 245
- quad, 278
- quadgk, 279
- quadl, 279
- quadv, 276
- residue, 243
- roots, 247
- sec, 31
- sech, 32
- set, 128, 139
- Sin, 44
- sin, 31
- sinh, 32
- sphere, 118
- sqp, 304, 335
- Sqrt, 44
- subplot, 100
- subs, 44
- surf, 112, 118
- symfsolve, 260
- symlsolve, 204
- Tan, 44
- tan, 31
- tanh, 32
- text, 94, 139
- title, 94
- trapz, 271

- xlabel, 94
- ylabel, 94
- Индекс корреляции, 334
- Интерполирование, 346
- Канонический полином, 347
- Коэффициенты регрессии, 332
- Команда
 - clear, 30
 - format, 26
 - grid, 93
 - pkg load symbolic, 43
 - sym, 43
 - symbols, 43
- Критерий Стьюдента, 333
- Кубический сплайн, 354
- Линейная интерполяция, 357
- Линейное программирование
 - задача, 311, 315
- Линия регрессии, 332
- Матрица, 142
 - число обусловленности, 201
 - действие
 - $*$, 149
 - $+$, 149
 - $-$, 149
 - $./$, 149
 - $.\backslash$, 149
 - $.\wedge$, 149
 - $/$, 151
 - \backslash , 151
 - поэлементное преобразование, 149
 - сложение, 147
 - степень, 149
 - транспонирование, 148
 - умножение, 148
 - умножение на число, 148
 - вычитание, 147
- диагональная, 180
- единичная, 180
- элемент, 142
- функция
 - cat, 162
 - chol, 176
 - cond, 174
 - cumprod, 167
 - cumsum, 168
 - det, 173
 - diff, 168
 - eig, 175
 - expm, 173
 - eye, 155
 - inv, 175
 - linspace, 160
 - logm, 173
 - logspace, 160
 - lu, 176
 - max, 170
 - mean, 171
 - min, 169
 - norm, 174
 - ones, 155
 - poly, 175
 - prod, 166
 - qr, 177
 - rand, 159
 - randn, 159
 - rcond, 174
 - repmat, 161
 - reshape, 161
 - rot90, 164
 - rref, 176
 - size, 166
 - sort, 171
 - sqrtm, 172
 - sum, 167
 - trace, 173
 - tril, 164
 - triu, 165
 - zeros, 156, 157
- характеристический многочлен, 198
- характеристическое уравнение, 198
- инволютивная, 183
- коэффициентов, 185
- кососимметрическая, 183
- минор, 186
- невыврожденная, 181
- нижняя треугольная, 180
- норма, 201
- нулевая, 180
- обратная, 181
- определитель, 180
- ортогональная, 183, 194
- перестановочная, 181
- произведение, 181
- ранг, 186
- расширенная, 185
- равенство, 181

- разность, 181
- решение линейных систем, 152
- симметрическая, 182
- символьная, 203
- системы, 185
- собственный вектор, 198
- собственное подпространство, 198
- собственное значение, 198
- сумма, 181
- транспонированная, 181
- умножение на число, 181
- уравнение, 184
- вектор–столбец, 181
- вектор–строка, 181
- верхняя треугольная, 180, 194
- вырожденная, 181
- LU-факторизация, 192
- LU-разложение, 192
- QR-разложение, 194
- Метод наименьших квадратов, 326
- Оператор
 - break, 55
 - continue, 55
 - disp, 47
 - for-end, 54
 - if-else-end, 48
 - if-elseif-end, 50
 - if-end, 49
 - input, 47
 - switch-case-end, 52
 - while-end, 53
- Определённый интеграл, 269
- Переменная
 - ans, 10, 29, 30
 - e, 29
 - i, 29, 35
 - inf, 29
 - j, 29, 35
 - NaN, 29
 - pi, 29
 - realmax, 30
 - realmin, 29
- Полином Лагранжа, 349
- Полином Ньютона, 347
- Производная функции, 262
- Производная параметрической функции, 264
- Расстояние от точки до плоскости, 232
- СЛАУ, 9, 185, 204
- Символ
 - ;, 8, 25, 29
 - %, 9, 25
- Система линейных алгебраических уравнений, 9, 185
- Система линейных уравнений
 - базисное решение, 186
 - частное решение, 186
 - эквивалентные, 185
 - метод Гаусса, 190
 - метод обратной матрицы, 188
 - множество решений, 185
 - неоднородная, 185
 - неопределённая, 186
 - несовместная, 185
 - общее решение, 186
 - однородная, 185
 - определённая, 186
 - правило Крамера, 187
 - решение, 185
 - совместная, 186
 - тривиальное решение, 185
- Система уравнений, 252
- Сортировка
 - метод пузырька, 59
- Суммарная квадратичная ошибка, 332
- Уравнение
 - трансцендентное, 249
- Вектор
 - действие, 144
 - .*, 146
 - ./, 147
 - .\, 147
 - .^, 147
 - деление на число, 145
 - поэлементное преобразование, 146
 - сложение, 144
 - транспонирование, 144
 - умножение, 145
 - умножение на число, 145
 - вычитание, 144
- элемент, 142
- функция
 - cross, 154
 - cumprod, 153
 - cumsum, 153
 - diff, 153

- dot, 154
- length, 152
- max, 154
- mean, 154
- min, 153
- prod, 153
- sort, 154
- sum, 153
- направляющий, 235
- неизвестных, 185
- нормальный, 223
- правых частей, 185
- свободных членов, 185
- вектор-столбец, 141, 180
- вектор-строка, 141, 180
- Векторная алгебра, 206
 - деление отрезка, 207
 - коэффициент пропорциональности векторов, 212
 - коллинеарные векторы, 210
 - компланарные векторы, 222
 - модуль вектора, 206
 - нуль-вектор, 206
 - проекция вектора, 213
 - алгебраическая, 213
 - противоположные векторы, 210
 - равные векторы, 210
 - середина отрезка, 207
 - смешанное произведение, 222
 - условие компланарности, 223
 - вектор, 206
 - действия, 215
 - деление на число, 215
 - правило параллелограмма, 216
 - правило треугольника, 217
 - скалярное произведение, 219
 - сложение, 215
 - угол между векторами, 219
 - умножение на число, 215
 - векторное произведение, 220
 - вычитание, 215
 - векторно-скалярное произведение, 222
- каноническое уравнение прямой, 237
- коэффициент корреляции, 332
- параметрическое уравнение прямой, 237
- уравнение плоскости в отрезках, 229

Научное издание

Серия «Библиотека ALT Linux»

Алексеев Евгений Ростиславович, Чеснокова Оксана Витальевна

Введение в Octave для инженеров и математиков

Оформление обложки: А. С. Осмоловская

Вёрстка: А. В. Прокудин, В. Л. Черный

Редактура: В. Л. Черный

Подписано в печать 26.08.12. Формат 60х90/16.

Гарнитура Computer Modern. Печать офсетная. Бумага офсетная.

Усл. печ. л. 23,0. Уч.-изд. л. 25,67 . Тираж 999 экз. Заказ

ООО «Альт Линукс»

Адрес для переписки: 119334, Москва, 5-й Донской проезд, д. 15,

стр. 6

Телефон: (495) 662-38-83. E-mail: sales@altlinux.ru

<http://altlinux.ru>