МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ТЕОРИИ СИСТЕМ

Учебно-методическое пособие

Санкт-Петербург
Издательство СПбГЭТУ «ЛЭТИ»
2018

УДК 004.383.3(07)
ББК 3.811.3я7
М34

Авторы: **А. С. Вознесенский, В. В. Гульванский, И. И. Канатов, Д. И. Каплун, С. А. Романов**

М34 Математические основы теории систем: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 76 с.

ISBN 978-5-7629-2419-1

Содержит описание лабораторных работ, которые позволяют на реальных практических задачах приобрести навыки математического моделирования и цифровой обработки сигналов.

Предназначено для студентов, обучающихся по направлению 27.03.04 «Управление в технических системах».

УДК 004.383.3(07)
ББК 3.811.3я7

Рецензент – канд. физ-мат. наук, доц. А.Б. Левина (Университет ИТМО).

Утверждено
редакционно-издательским советом университета
в качестве учебно-методического пособия

ISBN 978-5-7629-2419-1 © СПбГЭТУ «ЛЭТИ», 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Лабораторные работы по курсу «Математические основы теории систем» позволяют на реальных практических задачах приобрести навыки математического моделирования и цифровой обработки сигналов.

Все лабораторные работы проводятся с использованием интерактивной системы инженерных и научных расчетов MATLAB, которая в настоящее время по праву считается основным инструментом разработчика в самых различных областях человеческой деятельности, от космических аппаратов до финансово-экономических исследований. В настоящее время MATLAB является официальным средством проектирования и оформления инженерной документации в таких областях, как автоматика, авиация, военная техника. Математический аппарат MATLAB опирается на векторные и матричные вычисления, что при определенном навыке предельно сокращает трудоемкость расчетов. Богатый арсенал графических средств максимально приспособлен к требованиям на инженерную документацию.

Освоение методов работы в среде MATLAB также является задачей лабораторного практикума. Ограниченный объем лабораторных работ не позволяет рассчитывать на полное освоение системы, включающей несколько десятков пакетов, техническое описание каждого из которых содержит до 800 страниц.

В учебно-методическом пособии приведено краткое описание тех разделов MATLAB, которые будут использоваться на лабораторных занятиях: ввод и редактирование информации, графическое оформление работы, основы программирования, а также описание основных команд пакета Signal Processing Toolbox.

Освоение специфики работы с векторами, матрицами и их визуализация проводится на первом лабораторном занятии.

Следующие три лабораторных работы посвящены изучению классических методов обработки сигналов: корреляционному и спектральному анализу, цифровой фильтрации.

1. ВВЕДЕНИЕ В MATLAB

MATLAB (англ. *Matrix Laboratory*) – пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете. Разработчиком является компания MathWorks.

Релиз первой версии MATLAB 1.0 состоялся в 1984 году. В настоящее время актуальной является версия MATLAB 9.5.

Пакет используют более миллиона инженерных и научных работников, он работает на большинстве современных операционных систем, включая Linux, Mac OS и Windows.

Язык MATLAB является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования.

Программы, написанные на MATLAB, бывают двух типов – функции и скрипты. Функции имеют входные и выходные аргументы, а также собственное рабочее пространство для хранения промежуточных результатов вычислений и переменных. Скрипты же используют общее рабочее пространство. Как скрипты, так и функции сохраняются в виде текстовых файлов и компилируются в машинный код динамически.

Система имеет встроенные средства конвертации исходного кода на языке MATLAB в код на языке C/C++. Нативно поддерживаются параллельные и распределенные вычисления с использованием мощностей CPU и CUDA. Существует возможность конвертации исходного кода в специальные MEX-файлы (MATLAB «обертка» C-кода) с возможностью непосредственного использования внутри MATLAB.

В MATLAB имеется возможность создавать специальные наборы инструментов (англ. *toolbox*), расширяющие его функциональность. Наборы инструментов представляют собой коллекции функций и объектов, написанных на языке MATLAB для решения определённого класса задач. Компания MathWorks поставляет наборы инструментов, которые используются во многих областях: *Signal Processing Toolbox, Image Processing Toolbox, Wavelet Toolbox, Neural Network Toolbox*.

Основной особенностью языка MATLAB являются его широкие возможности по работе с матрицами, которые создатели языка выразили в лозунге «думай векторно» (англ. *think vectorized*).

Конкурентами MATLAB являются такие коммерческие продукты как MathCad, Mathematica, Maple и свободные решения: GNU Octave, SciLab, Python (вместе с библиотеками NumPy, SciPy, matplotlib).

1.1. Рабочие панели MATLAB

По умолчанию после запуска пакета MATLAB на экране появляется комбинированное окно (рис. 1.1), включающее четыре наиболее важные панели – *Command Window* (*Окно команд*), *Editor* (*Редактор*), *Workspace* (*Рабочее пространство*) и *Current Folder* (*Текущий каталог*). Все панели по умолчанию «поставлены на якоря». Они передвигаются вместе с главным окном системы, вместе с ним изменяют свои размеры, границы между окнами можно передвигать. Любую из панелей можно снять с якоря (*Undock*), развернуть (*Maximize*), свернуть (*Minimize*), закрыть (*Close*).

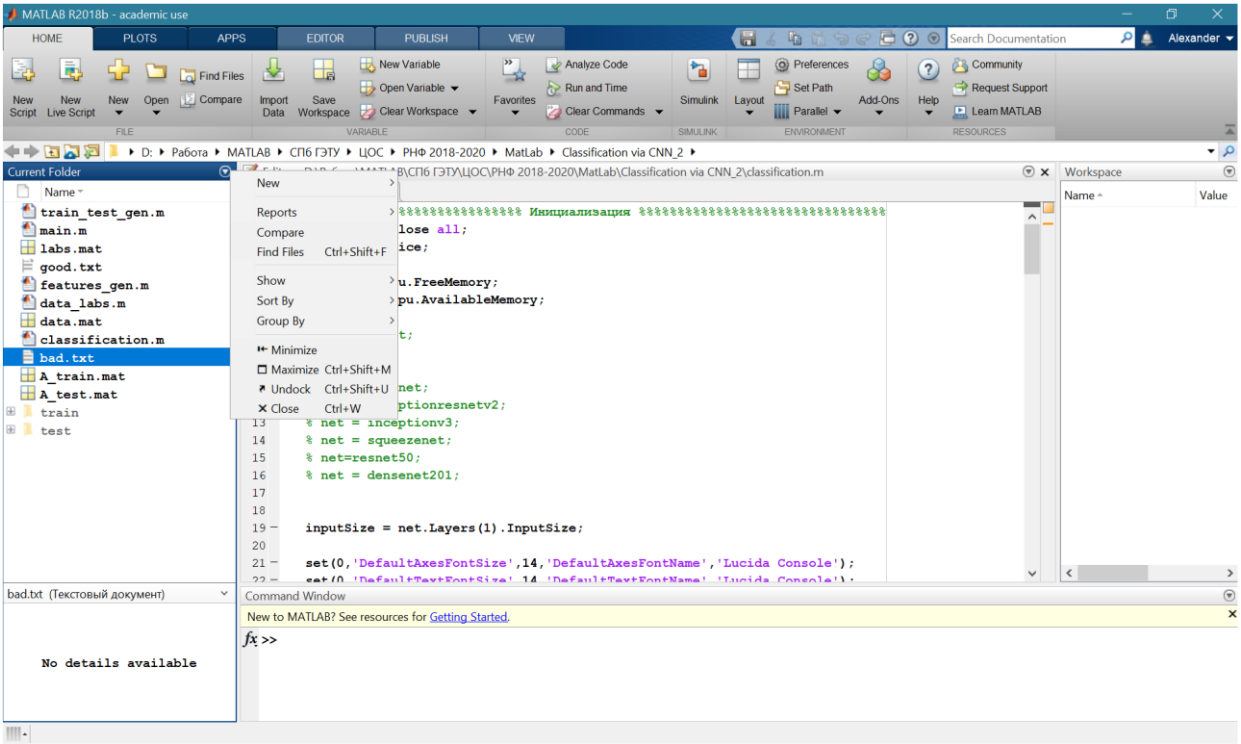

Рис. 1.1. Главное окно системы

В командном окне можно обратиться за помощью по поводу того или иного термина, используя одну из команд – `doc` или `help` (рис. 1.2).

Самой используемой панелью является *Command Window* (*Окно команд*). В ней набираются команды пользователя, подлежащие немедленному исполнению. Результат операции будет выведен на экран, если после команды не поставлена точка с запятой, предотвращающая вывод результата.

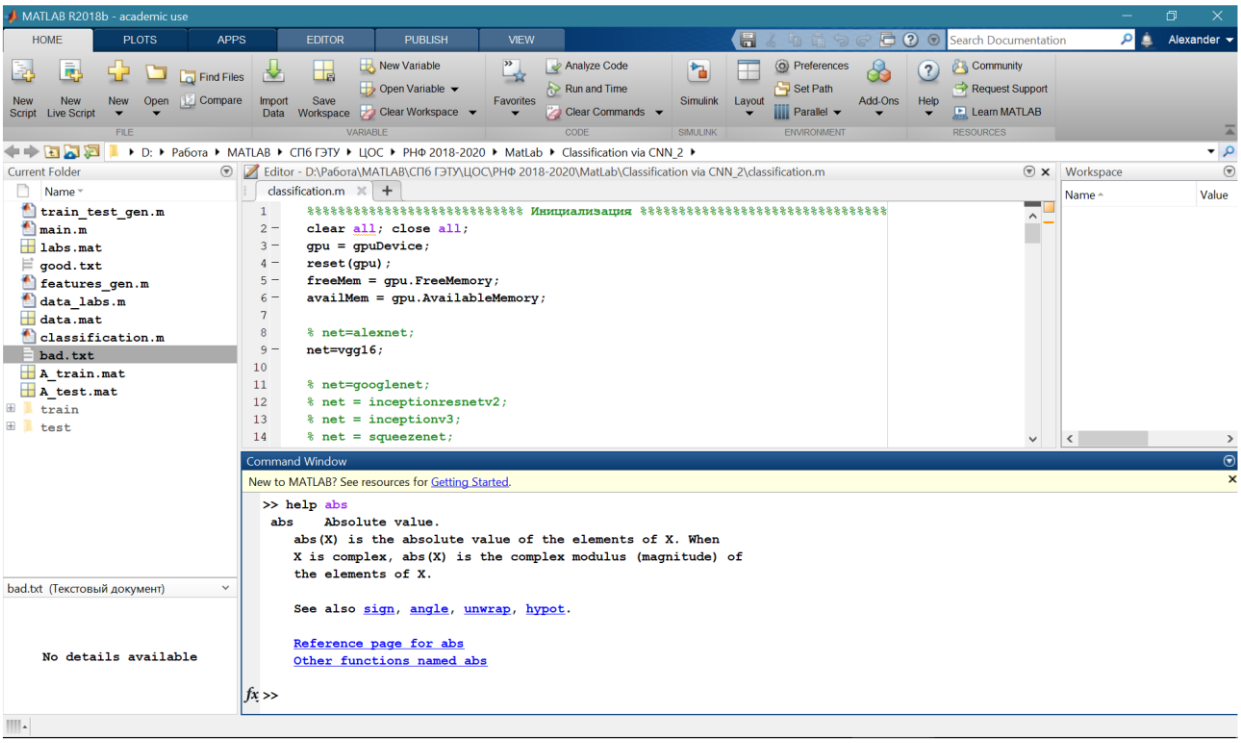

Рис. 1.2. Использование команды help

Еще одной особенностью MATLAB является запрет на исправление предыдущих команд. Просмотреть историю команд, исправить предыдущую команду (или повторить одну из предыдущих) можно курсором, щелкнув по кнопке со стрелкой вверх. Пока не нажата клавиша *Enter*, вы можете вносить любые исправления (рис. 1.3).

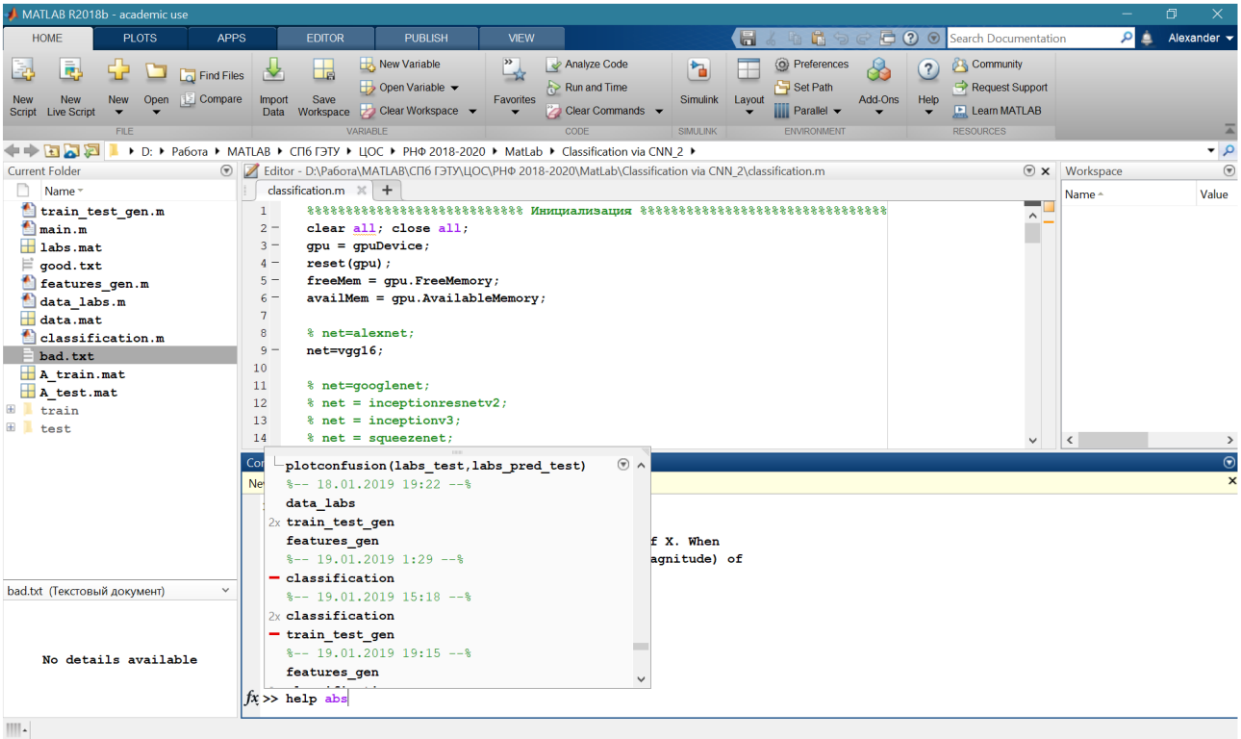

Рис. 1.3. Просмотр истории команд

Еще более удобный способ – писать программу в виде совокупности файл-скриптов (*New->Script*) и файл-функций (*New->Function*) с использованием *Редактора* (рис. 1.4). Обе разновидности файлов имеют расширение .m

С помощью скриптов оформляют основные программы, управляющие от начала до конца организацией всего вычислительного процесса, а также отдельные части основных программ.

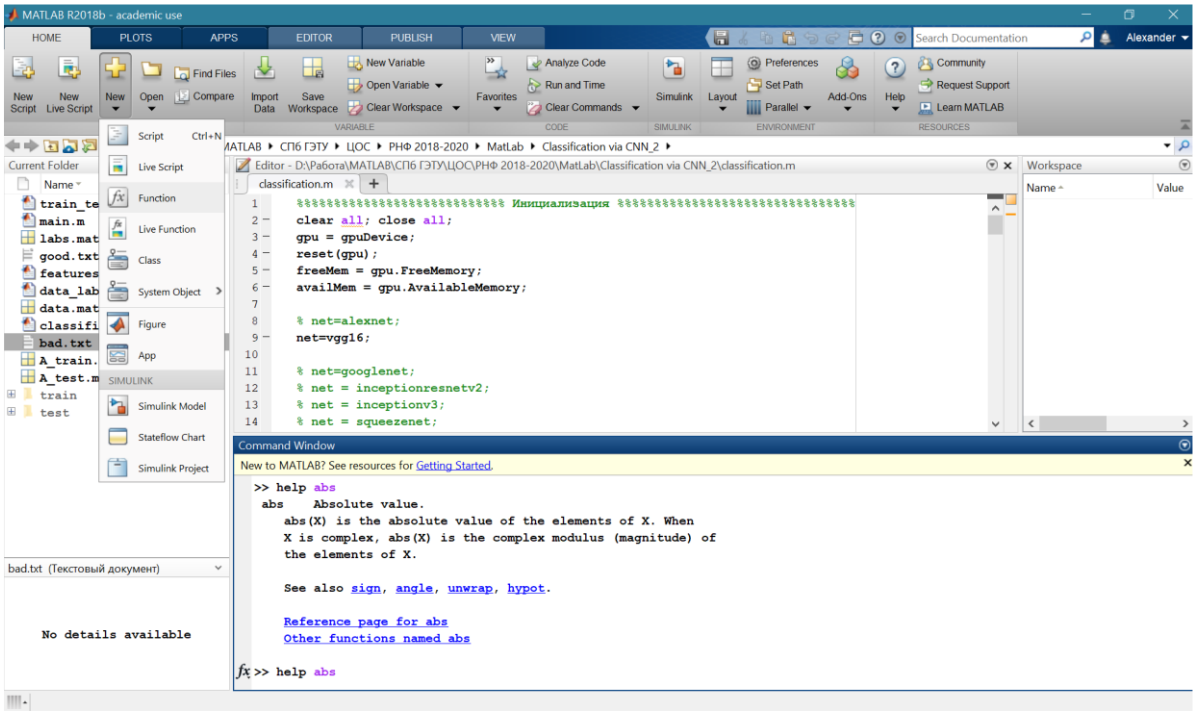

Рис. 1.4. Создание новых скриптов и функций

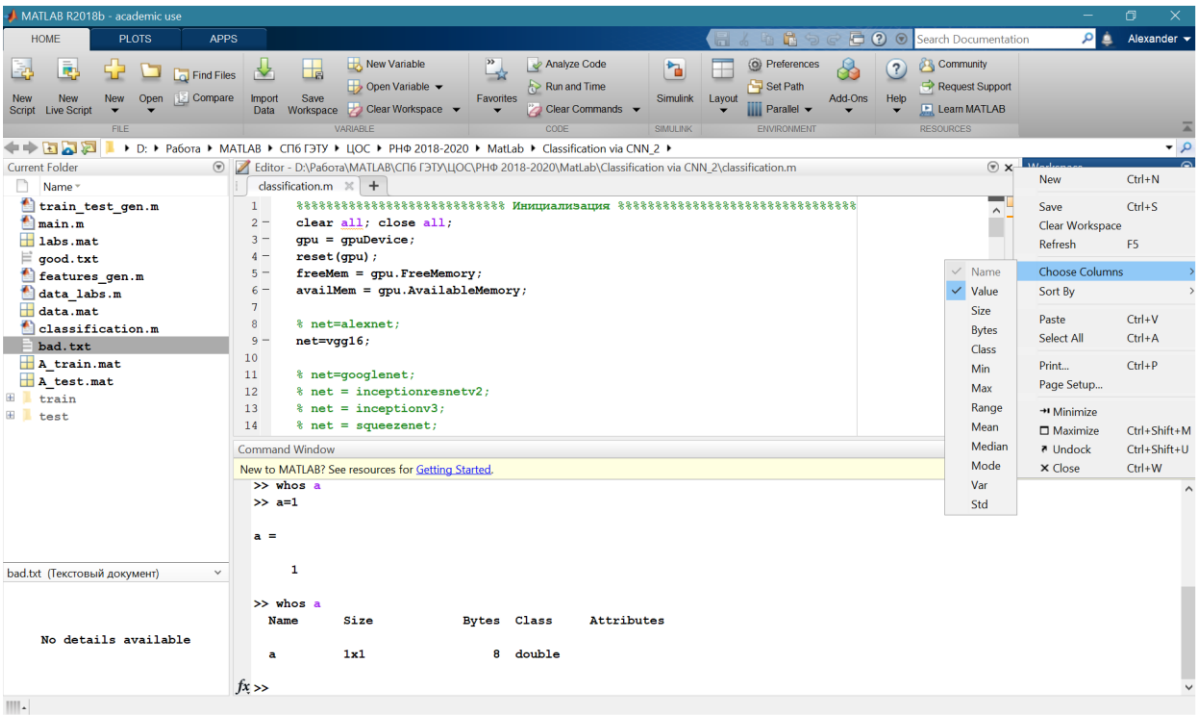

Рис. 1.5. Использование команды whos

Функции служат для оформления отдельных процедур и функций.

Окно *Workspace* (Рабочее пространство) отображает все переменные среды, созданные или импортированные. Здесь можно увидеть их имена (*Name*), значения (*Value*) и другие атрибуты.

Точно такую же информацию можно увидеть в командном окне после исполнения команды `whos` (рис. 1.5).

1.2. Ввод и редактирование информации

MATLAB является системой, ориентированной на выполнение векторных и матричных операций. Под вектором в MATLAB понимается одномерный массив чисел, а под матрицей – двумерный массив. При этом по умолчанию принято, что *любая заданная переменная представляет собой вектор или матрицу*. Например, отдельное заданное число система воспринимает как матрицу размером 1×1, а вектор-строку, состоящую из n элементов, – как матрицу размером 1×n.

Ввод значений векторов и матриц. Начальные значения векторов можно вводить с клавиатуры поэлементно. Для этого в строке следует сначала указать имя вектора, потом поставить знак присваивания, далее – *открывающую квадратную скобку,* за ней ввести заданные значения элементов вектора, разделенные пробелами или запятыми, заканчивается строка *закрывающей квадратной скобкой*. Например, строка `v=[1.2 -0.3 1.2e-5]` задает вектор `v`, который состоит из трех элементов со значениями `1.2, -0.3` и `1.2e-5:`

```
>> v=[1.2 -0.3 1.2e-5]

v =

    1.2000   -0.3000    0.0000
```

После ввода вектора система выводит его элементы на экран. В приведенном примере последний элемент представлен значением 0, поскольку в установленном по умолчанию формате `short` отображается не более четырех цифр после десятичной точки.

Длинный вектор можно вводить частями, которые потом следует объединять с помощью операции объединения (конкатенации) векторов в строку: `v=[vl_v2]` (между векторами либо пробел, либо запятая):

```
>> v1=[1 2 3]; v2=[4 5 6]; v=[v1 v2]
```

```
v =

     1     2     3     4     5     6
```

Язык MATLAB предоставляет возможность *сокращенного ввода вектора*, значения элементов которого составляют *арифметическую прогрессию*. Это могут быть, например, порядковые номера дискретных отсчетов или значения временных интервалов. Если обозначить начальное значение этой прогрессии (значение первого элемента вектора) как `start`, конечное значение прогрессии (значение последнего элемента вектора) – как `stop`, а разность прогрессии (шаг) – как `h`, то вектор можно будет ввести с помощью короткой записи: `v=start: h: stop`. Например, ввод строки `v=1:5:31` даст результат:

```
>> v=1:5:31

v =

     1     6    11    16    21    26    31
```

Если средний параметр (шаг) не указан, то он по умолчанию равен 1. Например, ввод выражения `t = -5:2` приводит к формированию такого вектора:

```
>> t = -5:2

t =

    -5    -4    -3    -2    -1     0     1     2
```

Мы показали, как вводятся векторы-строки. Вектор-столбец вводится аналогично, но значения элементов отделяются точкой с запятой. Можно также использовать операцию транспонирования вектора: '.

В MATLAB значения *элементов матрицы* вводятся *в квадратных скобках*, по строкам. При этом элементы строки матрицы разделяются пробелом или запятой, а строки отделяются одна от другой точкой с запятой:

```
>> A=[1 2 3 4; 5.5 7.2 3 9;]
A =
    1.0000    2.0000    3.0000    4.0000
    5.5000    7.2000    3.0000    9.0000
```

1.3. Типы данных в MATLAB

Создание программы начинается с определения переменных и способа представления данных. Следовательно, чтобы правильно организовать описание данных программы, необходимо знать, как задавать переменные MATLAB и какие виды переменных возможны (табл. 1.1).

Таблица 1.1

Типы данных в MATLAB

Типы данных	Описание данных программы
`double`	Вещественный, 64 бит
`single`	Вещественный, 32 бит
`int8`	Знаковый целочисленный, 8 бит
`int16`	Знаковый целочисленный, 16 бит
`int32`	Знаковый целочисленный, 32 бит
`int64`	Знаковый целочисленный, 64 бит
`uint8`	Беззнаковый целочисленный, 8 бит
`uint16`	Беззнаковый целочисленный, 16 бит
`uint32`	Беззнаковый целочисленный, 32 бит
`uint64`	Беззнаковый целочисленный, 64 бит

По умолчанию используется тип `double`, который имеет наибольшую точность представления вещественного числа и является потому универсальным типом. Однако если необходимо экономить память ЭВМ, то можно указывать желаемый тип самостоятельно.

Последнее, что следует знать при задании переменных, – это правило определения их имен. В MATLAB имена переменных могут задаваться только латинскими буквами, цифрами и символом ‘_’. Причем, первый символ в имени должен соответствовать букве латинского алфавита. Также следует отметить, что имена

```
arg = 1;
Arg = 2;
ARG = 3;
```

это три разных имени, т. е. три разные переменные со значениями 1, 2 и 3 соответственно. Данный пример показывает, что MATLAB *различает регистр в именах переменных*.

При программировании лучше всего задавать осмысленные имена переменных, по которым можно было бы понять, какие данные они представляют. Это позволяет избежать ошибок при написании сложных программ.

1.4. Векторы и матрицы в MATLAB

В MATLAB имеется несколько встроенных функций, которые позволяют формировать векторы и матрицы определенного вида. Описание этих функций и примеры их применения приведены ниже.

Функция `zeros (M,N)` создает матрицу размером Mr N с нулевыми элементами.

```
>> zeros(3,5)

ans =

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

Функция `ones(M,N)` создает матрицу из единиц:

```
>> ones(3,5)

ans =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Функция `eye(M,N)` создает единичную матрицу размером Mr N, т. е. матрицу с единицами по главной диагонали и остальными нулевыми элементами:

```
>> eye(3,5)

ans =

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

Функция `rand(M,N)` создает матрицу размером Mr N из случайных чисел, равномерно распределенных в диапазоне от 0 до 1.

Функция `randn(M,N)` создает матрицу размером Mr N из случайных чисел, распределенных по нормальному (гауссову) закону с нулевым математическим ожиданием и стандартным (среднеквадратичным) отклонением, равным 1.

Функция `hadamard(N)` создает матрицу Адамара размером N × N:

```
>> hadamard(4)

ans =

     1     1     1     1
     1    -1     1    -1
     1     1    -1    -1
     1    -1    -1     1
```

В языке MATLAB предусмотрено несколько функций, которые позволяют формировать одну матрицу на основе другой (заданной) или на основе некоторого заданного вектора. К таким функциям принадлежат следующие.

Функция `triu(A)` образует верхнюю треугольную матрицу на основе матрицы A путем обнуления ее элементов ниже главной диагонали:

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

A =

     1     2     3
     4     5     6
     7     8     9

>> triu(A)

ans =

     1     2     3
     0     5     6
     0     0     9
```

Функция `diag(x)` формирует или извлекает диагональ матрицы. Если `x` является вектором, то данная функция создает квадратную матрицу, у которой элементы вектора `x` размещены на главной диагонали:

```
>> v=[1 2 3]

v =

     1     2     3

>> diag(v)

ans =

     1     0     0
     0     2     0
     0     0     3

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

A =

     1     2     3
     4     5     6
     7     8     9

>> diag(A)

ans =

     1
     5
     9
```

1.5. Извлечение и вставка частей матриц

Как и в других алгоритмических языках, к элементам матриц и векторов обращаются, используя индексы. В отличие от языка C, в системе MATLAB индексы отсчитываются от 1 и заключаются в круглые скобки: `x(1)`, `a(2,3)`. Довольно интересной особенностью языка MATLAB является

возможность выполнения однотипных операций над подмножеством элементов векторов или матриц. Для этого вместо индекса указывается диапазон индексов, разделяемых двоеточием.

Прежде всего отметим, что обращение к любому элементу заданной матрицы в MATLAB осуществляется указанием (в круглых скобках, через запятую) после имени матрицы двух целых положительных чисел, определяющих номер строки и столбца матрицы, на пересечении которых расположен этот элемент. Допустим, мы имеем такую матрицу A:

```
>> A=[1 2 3 4; 5 6 7 8; 9 10 11 12]

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Получить значение элемента этой матрицы, расположенного на пересечении второй строки с третьим столбцом, можно таким образом:

```
>> A(2,3)

ans =

     7
```

Если же нужно *поместить на указанное место* некоторое число, например, `pi`, выполните следующее:

```
>> A(2,3)=pi

A =

    1.0000    2.0000    3.0000    4.0000
    5.0000    6.0000    3.1416    8.0000
    9.0000   10.0000   11.0000   12.0000
```

Такая операция называется «вырезкой». Вырезать (вставить) можно как один элемент, так и целую матрицу меньших размеров. Допустим, необходимо вырезать всю вторую строку матрицы:

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

B =

    5.0000    6.0000    3.1416    8.0000
```

Аналогично можно вставить матрицу `B` вместо любой строки `A`:

```
>> A(1,:) = B

A =

    5.0000    6.0000    3.1416    8.0000
    5.0000    6.0000    3.1416    8.0000
    9.0000   10.0000   11.0000   12.0000
```

Если верхней границей изменения номеров элементов матрицы является ее размер в этом измерении, вместо него можно использовать служебное слово `end`. Например:

```
>> A(2:end,2:end)

ans =

    6.0000    3.1416    8.0000
   10.0000   11.0000   12.0000
```

Эти операции удобно использовать для формирования матриц, большинство элементов которых одинаковы, в частности так называемых разреженных матриц, состоящих в основном из нулей. Для примера рассмотрим формирование разреженной матрицы размером 5×7 с единичными элементами в центре:

```
>> A=zeros(5,7);
>> B=ones(3,3);
>> A(2:4,3:5)=B




A =
```

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

«Растянуть» матрицу `A` в единый вектор `v` можно с помощью обычной записи `v=A(:)`. При этом создается вектор-столбец с количеством элементов `m*n`.

Богатые возможности для формирования матриц большой размерности с повторяющимися блоками дает операция *кронекеровского произведения* `kron(A,B)`. При этой операции каждый элемент левой матрацы умножается на всю правую матрицу:

```
>> A=[2 3;4 5];   B=[1 1;1 1];
>> C=kron(A,B)

C =

     2     2     3     3
     2     2     3     3
     4     4     5     5
     4     4     5     5
```

1.6. Специфика выполнения арифметических операций

При составлении алгебраических выражений MATLAB разрешает использование традиционных знаков арифметических операций и символов специальных операций, список которых приведен в табл. 1.2.

Таблица 1.2

Арифметические операции в MATLAB

Символы	Выполняемое действие
+	Покомпонентное сложение числовых массивов одинаковой размерности; добавление скалярной величины к каждому элементу массива
–	Покомпонентное вычитание числовых массивов одинаковой размерности; вычитание скалярной величины из каждого элемента массива
*	Умножение матриц в соответствии с правилами линейной алгебры (число столбцов первого сомножителя должно быть равно числу строк второго сомножителя); покомпонентное умножение всех элементов массива на скаляр
.*	Покомпонентное умножение элементов массивов одинаковой размерности

Окончание табл. 1.2

Символы	Выполняемое действие
/	Деление скаляра на скаляр; покомпонентное деление всех элементов массива на скаляр, правое матричное деление A/B= A*INV(B)
./	Правое покомпонентное деление
\	Левое матричное деление A\B = INV(A)*B
.\	Левое покомпонентное деление
^	Возведение скаляра в любую степень; вычисление целой степени квадратной матрицы
=	Знак присвоения
'	Транспонирование матрицы
[ ]	Задание числовых значений элементов матрицы; задание выходных параметров функции.
( )	Указание индексов (порядковых номеров) элемента вектора или матрицы; задание порядка выполнения операций
.	Десятичная точка; знак поэлементного выполнения операции
..	Переход по дереву каталогов на один уровень вверх
…	Признак продолжения строки
,	Разделение элементов вектора; отделение операторов
;	Подавление вывода на экран; отделение строк матрицы
:	Формирование векторов; выделение строк, столбцов;
%	Указатель начала комментария

1.7. Действия над векторами и матрицами

Сложение векторов. Осуществляется в MATLAB с помощью арифметического знака сложения (+) таким образом: `v=x+y`:

```
>> x=[1 2 3]; y= [4 5 6];
>> v=x+y

v =

     5     7     9
```

Вычитание векторов. Осуществляется в MATLAB с помощью арифметического знака вычитания (–) таким образом: `v=x-y`:

```
>> x=[1 2 3]; y= [4 5 6];
>> v=x-y

v =

    -3    -3    -3
```

Транспонирование вектора. Осуществляется с применением знака апострофа ('), который записывается сразу после имени транспонируемого вектора:

```
>> x'

ans =

     1
     2
     3
```

Умножение вектора на число. Осуществляется в MATLAB с помощью арифметического знака умножения (*) таким образом: `v=x*a` или `v=a*x`, где a – некоторое действительное число:

```
>> v=2*x
v =

     2     4     6
```

Умножение двух векторов. Определено в математике только для векторов одинакового размера (одинаковой длины) и лишь тогда, когда один из векторов-множителей – строка, а второй – столбец. Иначе говоря, если векторы `x` и `y` являются строками, то математический смысл имеют лишь две формы умножения данных векторов: `m=x'*y` и `v=x*y'`. При этом в первом случае результатом будет квадратная матрица, а во втором – число. Во втором случае имеем *скалярное произведение*. В MATLAB умножение векторов задается посредством символа «*», который записывается между множителями-векторами:

```
>> x= [1 2 3]; y= [4 5 6];
>> v = x'*y

v =

     4     5     6
     8    10    12
    12    15    18

>> u=x*y'

u =

    32
```

Векторное произведение двух векторов (для трехкомпонентных векторов). Для выполнения этой операции в MATLAB предусмотрена функция `cross`, которая позволяет найти векторное произведение двух векторов. Если заданы два трехкомпонентных вектора `vl` и `v2`, достаточно ввести команду `cross(vl,v2)`:

```
>> vl = [1 2 3]; v2 = [4 5 6];
>> cross(vl,v2)

ans =

    -3     6    -3
```

Поэлементное преобразование векторов. В языке MATLAB предусмотрено выполнение ряда операций, позволяющих преобразовать заданный вектор в другой вектор, имеющий такой же размер и тип. Подобные операции, строго говоря, не является математическими. К таким операциям относятся, в частности, все операции, осуществляемые с помощью элементарных математических функций, имеющих один аргумент. В языке MATLAB запись вида `y=sin(x)`, где `x` – некоторый известный вектор, приводит к формированию нового вектора `y` (имеющего тот же тип и размер, что и вектор-аргумент), элементы которого равны синусам соответствующих элементов вектора-аргумента `x`:

```
>> x=[-2,-1,0,1.2];
>> y=sin(x)

y =

   -0.9093   -0.8415         0    0.9320
```

Кроме описанных операций в MATLAB предусмотрено несколько операций поэлементного преобразования. Они задаются с помощью обычных знаков арифметических операций и применяются к векторам одинакового типа и размера. Результатом их является вектор аналогичного типа и размера.

Это следующие операции:

– добавление числа к каждому элементу (вычитание числа из каждого элемента) вектора. Осуществляется с помощью символа «+» («-»);

– поэлементное умножение векторов. Производится с помощью комбинации символов «.*», которые записываются между именами перемножаемых векторов. В результате получается вектор, каждый элемент которого является произведением соответствующих элементов векторов-«сомножителей»;

– поэлементное деление векторов. Осуществляется с помощью комбинации символов «./». В результате получается вектор, каждый элемент которого является частным от деления соответствующего элемента первого вектора на соответствующий элемент второго;

– поэлементное деление векторов в обратном направлении. Осуществляется с помощью комбинации символов «.\.». В результате получают вектор, каждый элемент которого является частным от деления соответствующего элемента второго вектора на соответствующий элемент первого;

– поэлементное возведение в степень. Осуществляется с помощью комбинации символов «.^». В результате получается вектор, каждый элемент которого является соответствующим элементом первого вектора, возведенным в степень.

1.8. Комплексные числа

Константам `i` и `j` первоначально присваивается значение, равное `sqrt(-1)`. Они используются для ввода комплексных чисел.

При вводе комплексных чисел допустимы следующие формы записи:
`3+2i, 3+2*i, 3+2j, 3+2*j, 3+2*sqrt(-1), complex(3,2).`

Все формы представления соответствуют одному и тому же комплексному числу `3 + 2i`.

Следует отметить, что с символами `i` и `j` могут быть ассоциированы и другие величины, например, переменная цикла или индекс элемента массива, и в этом случае они *временно* теряют значение мнимой единицы. Поэтому в системе MATLAB, где используется ускоритель JIT, следует помнить, что способ формирования комплексного числа с использованием функции ядра `complex` является самым быстрым.

При использовании комплексного числа в математических выражениях для устранения неоднозначности его заключают в круглые скобки.

Функции `real` и `imag` позволяют получить вещественную и мнимую часть комплексного числа, а функции `abs` и `angle` – соответственно модуль и фазу (в радианах).

Для получения комплексно сопряженного числа используется функция `conj`.

1.9. Ключевые слова

В языке MATLAB зарезервированы следующие 17 ключевых слов, которые используются при формировании операторов: `break, case, catch, continue, else, elseif, end, for, function, global, if, otherwise, persistent, return, switch, try, while.`

Эти слова можно использовать только по прямому назначению при построении конструкций языка. Их нельзя применять для иных целей, это будет порождать сообщения об ошибке.

1.10. Логические операторы

Для сравнения элементов массивов предусмотрены следующие операторы:

< меньше;

<= меньше или равно;

> больше;

>= больше или равно;

= = тождественно равно;

~ = не равно.

При сравнении массивов, допускающих операцию сравнения их элементов, образуется массив логических значений 1 на позициях, где проверяемое отношение выполняется, и 0 – где оно ложно. Если элементы массивов имеют значения 1 и 0 (логические массивы), для них предусмотрены логические операции (табл. 1.3).

Таблица 1.3

Логические операции в MATLAB

Логический оператор	Математическая функция	Функция MATLAB
Отрицание	`not(A)`	`~A`
Логическое И (конъюнкция)	`and(A,B)`	`A&B`
Логическое ИЛИ (дизъюнкция)	`or(A,B)`	`A|B`
Исключающее ИЛИ (сложение по модулю 2)	`xor(A,B)`	`xor(A,B)`
Проверка истинности всех элементов	–	`all(A)`
Проверка истинности хотя бы одного элемента	–	`any(A)`

1.11. Элементарные математические функции

Полный перечень всех математических операций можно найти в справочной системе HELP.

Если вас интересует процедура выполнения конкретной операции, например, вычисление синуса, наберите команду `help`, задав в качестве параметра ключевое слово:

```
>> help sin
```

Если названия команды у вас нет, воспользуйтесь HELP-навигатором (рис. 1.6):

Contents (Содержание);

Search (Поиск);

Examples (Примеры);

Functions (Функции).

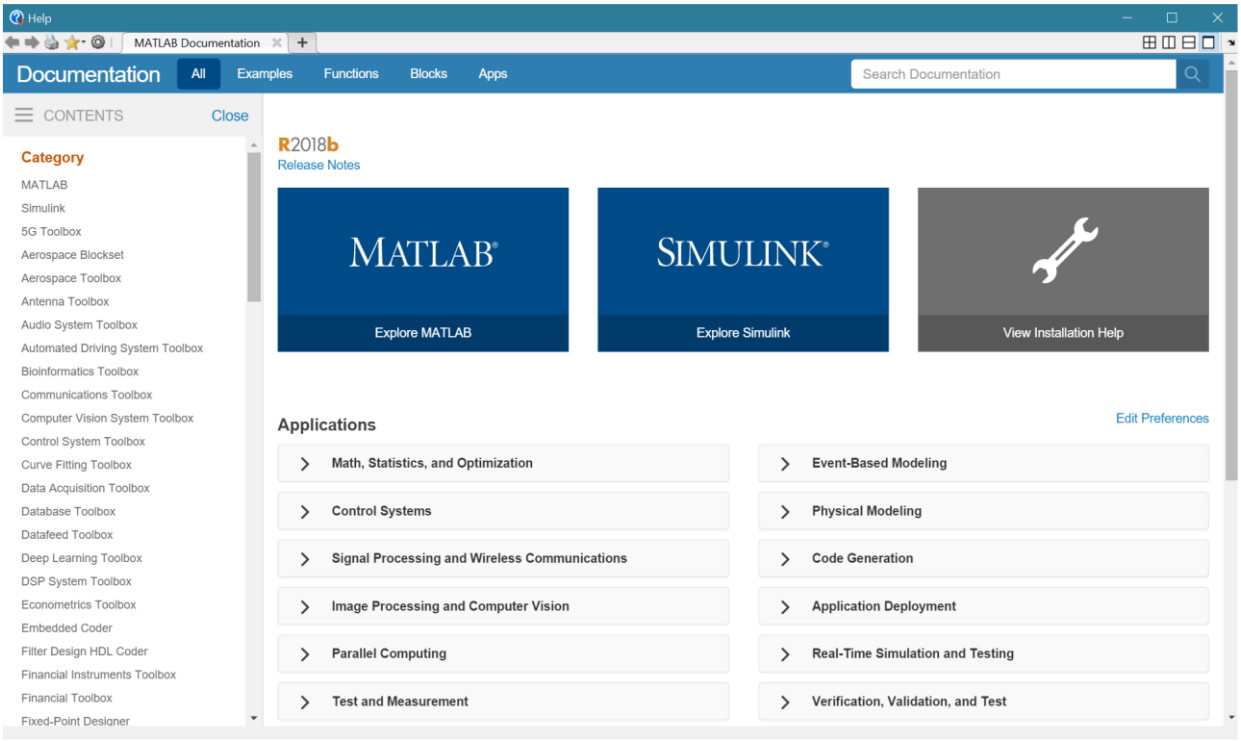

Рис. 1.6. Справочная система MATLAB

В табл. 1.4 приведены только часто встречающиеся элементарные функции.

Таблица 1.4

Элементарные функции в MATLAB

Категория функций	Наименование функций
Тригонометрические	`sin, cos, tan, cot, sec, csc`
Обратные тригонометрические	`arcsin, arccos, arctan, arccot, arcsec, arccsc`
Гиперболические	`sinh, cosh, tanh, coth, sech, csch`
Обратные гиперболические	`arcsinh, arccosh, arctanh, arccoth, arcsech, arccsch`
Степени логарифмы, корни	`exp, ln, log, log10, log2, ^, sqrt, surd`
Округления	`ceil, fix, floor, round`
Наибольший общий делитель	`gcd`
Наименьшее общее кратное	`lem`
Модуль числа	`abs`

Окончание табл. 1.4

Категория функций	Наименование функций
Знак числа	`Sign`
Остаток от деления с учетом знака делимого	`mod`
Остаток от деления	`rem`
Разложение числа на простые множители	`factor`
Вычисление факториала	`factorial`

1.12. Оформление графиков

Одна из ключевых возможностей MATLAB – возможность визуализации результатов вычислений с использованием широкого спектра функций.

Основная функция построения графика одной переменной – `plot(x,y)`.

Пример:

```
>> x = -4*pi : pi/100 : 4*pi;
>> y= 3* sin(x+ pi/2);
>> plot (x,y)
```

Можно и проще – `plot(y)`. Но если аргумент не указан в явном виде, по умолчанию в качестве аргумента принимается номер отсчета вектора `y`.

Масштаб по оси x потерян, теперь это просто картинка. Сравните два графика (рис. 1.7).

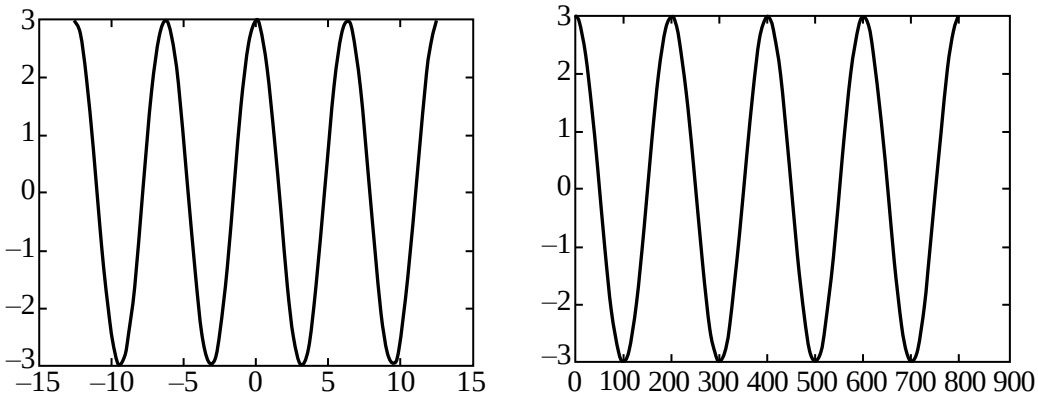

Рис. 1.7. Графики

Это далеко не все возможности MATLAB. Разберем следующий пример (рис. 1.8) (комментарии в MATLAB выделяются знаком `%`):

```
set(0,'DefaultAxesFontSize',14,'DefaultAxesFontName','L
ucida Console'); % установка шрифтов
set(0,'DefaultTextFontSize',14,'DefaultTextFontName','L
ucida Console'); % установка шрифтов
```

```
x=-4*pi:pi/20:4*pi; % задание аргумента (отсчетов фазы)
y1=4*sin(x+pi/2);   % первая функция
y2=2*sin(3*x);      % вторая функция
figure(1);           % если не писать эту команду, все графики будут выводиться в одно окно; при этом предыдущий график будет потерян
axis([-2*pi 2*pi -4 4]);    % задание масштаба осей по форме [xmin xmax ymin ymax]
xlabel('phase, rad');           % наименование горизонтальной оси
ylabel('amplitude');            % наименование вертикальной оси
title('y1=4sin(x) y2=2sin(3x)'); % наименование графика
hold on;                    % команда включения наложения (совмещения) графиков
plot(x,y1,'r');         % точки графика соединены прямыми красного цвета ('r')
plot(x,y2,'g'); % точки графика соединены прямыми зеленого цвета ('g')
legend('4*sin(x+pi/2)','2*sin(3*x)')    % формирует окно в правом углу графика с названиями и видом линий всех переменных.
grid on; % наложение сетки
```

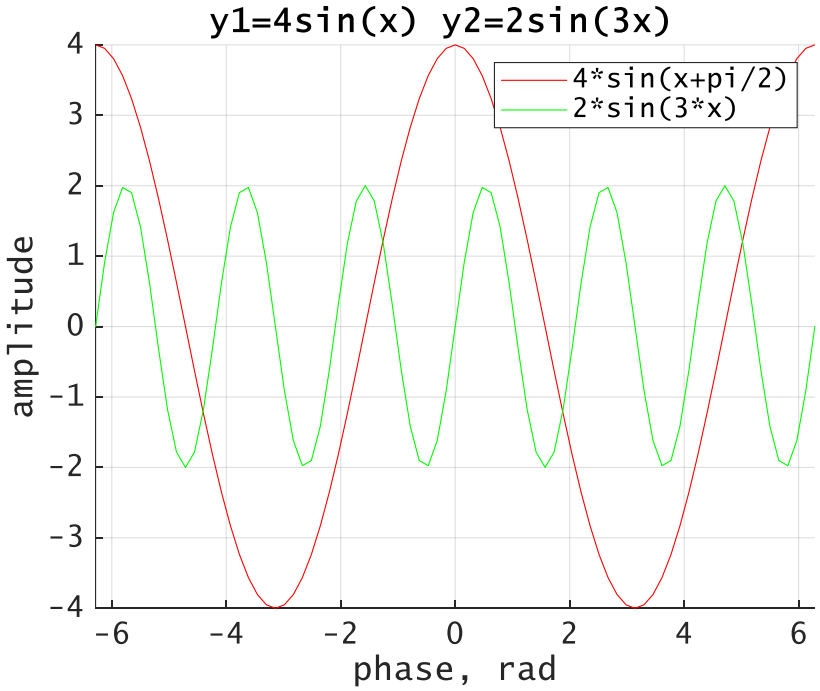

Рис. 1.8. Результат работы программы

Скопировать график в текстовый редактор (Word) можно выполнив команды Edit->Copy Figure (рис. 1.9). Вставка осуществляется путем нажатия Ctrl+V либо Shift+Ins. Также возможно вставить рисунок стандартными средствами Word.

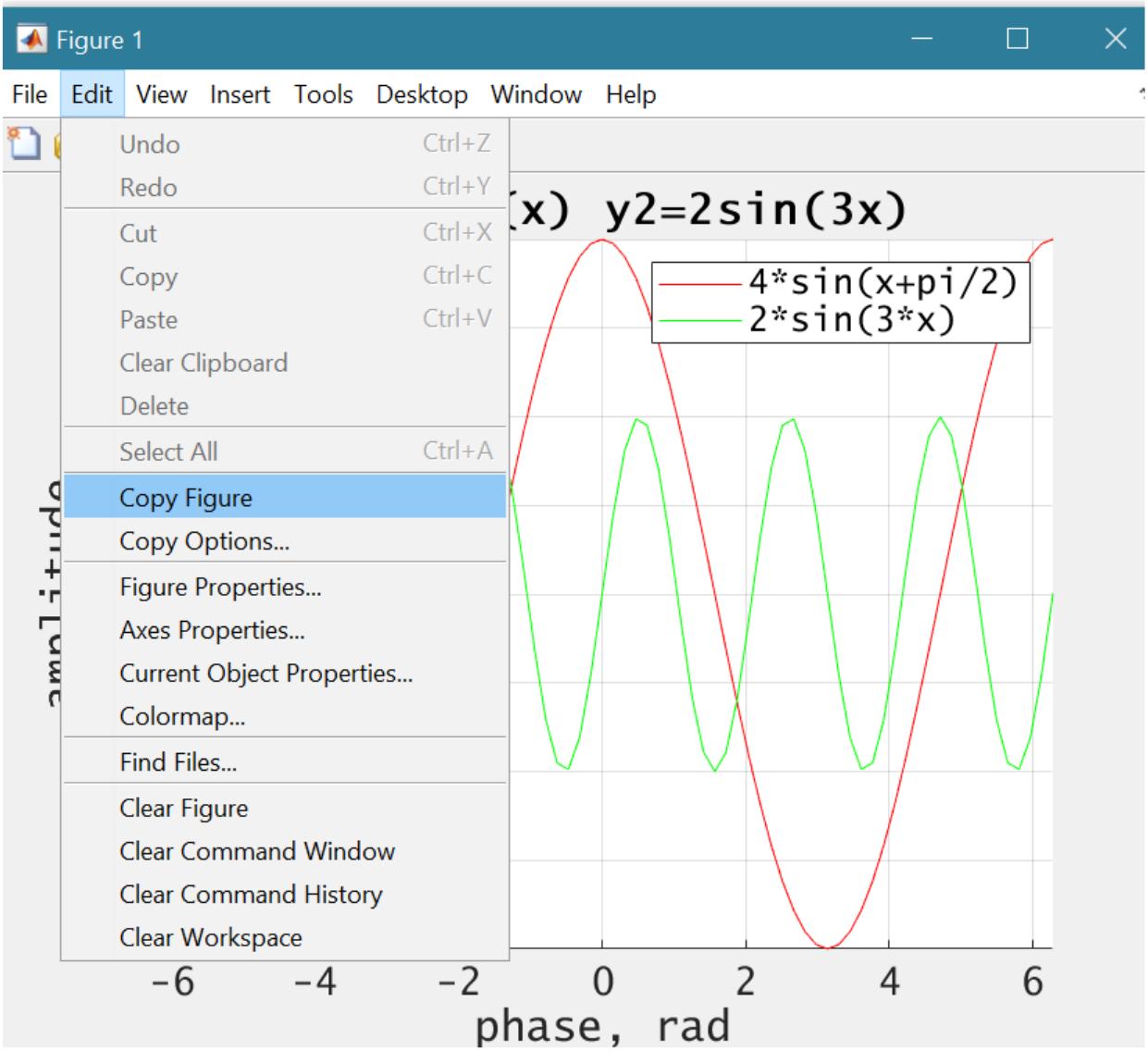

Рис. 1.9. Копирование графика в текстовый редактор

Приведем наиболее часто употребляемые команды графики (табл. 1.5):

Таблица 1.5

Графические функции в MATLAB

Команды	Содержание команд
`plot`	График в линейном масштабе
`loglog`	График в логарифмическом масштабе
`semilogx, semilogy`	График в полулогарифмическом масштабе
`polar`	График в полярных координатах
`plot3`	Построение линий и точек в трехмерном пространстве
`meshgrid`	Формирование двумерных массивов X и Y

Окончание табл. 1.5

Команды	Содержание команд
`mesh, meshc, meshz`	Трехмерная сетчатая поверхность
`surf, surfc`	Затененная сетчатая поверхность
`axis`	Масштабирование осей и вывод на экран
`grid`	Нанесение сетки
`hold`	Управление режимом сохранения текущего графического окна
`subplot`	Разбиение графического окна
`zoom`	Управление масштабом графика
`colormap`	Палитра цветов
`title`	Заголовки для двух- и трехмерных графиков
`xlabel, ylabel, zlabel`	Обозначение осей
`text`	Добавление к текущему графику текста
`legend`	Добавление к текущему графику легенды
`colorbar`	Шкала палитры
`bar`	Столбцовые диаграммы
`hist`	Построение гистограммы
`stem`	Дискретные графики
`stairs`	Ступенчатый график

Проиллюстрируем некоторые из команд (рис. 1.10–1.12):

```
>>[X,Y]=meshgrid(-1:0.05:1,0:0.05:1);
>>Z=4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
>>surfc(X,Y,Z)
```

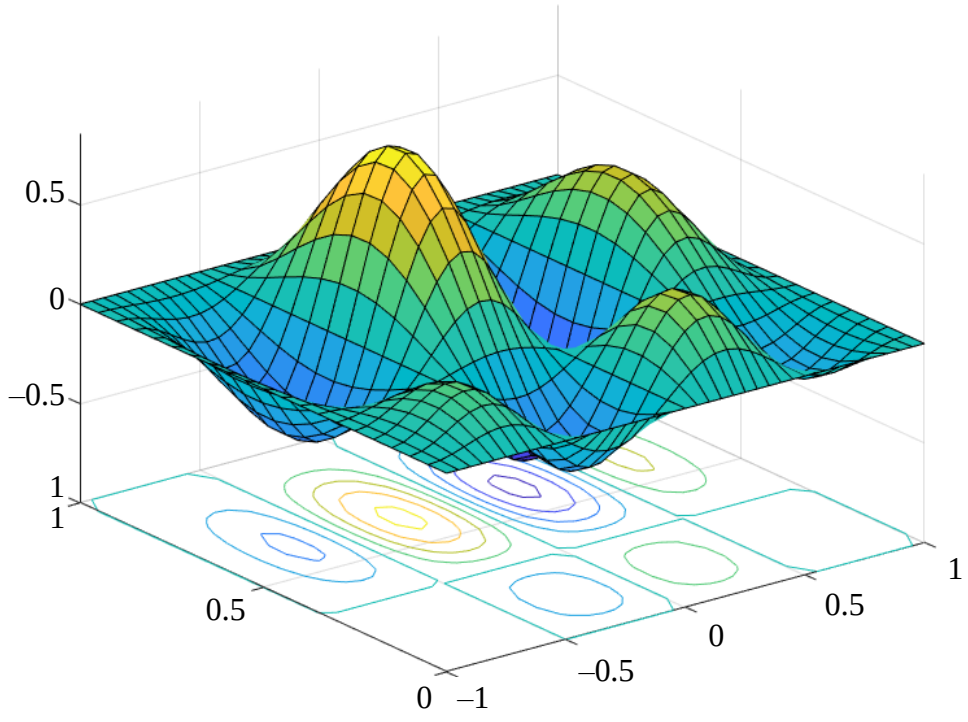

Рис. 1.10. Результат работы программы

```
>> R=2;
>> [X,Y,Z]=cylinder(R);
>> surf(X,Y,Z)
>> shading interp   % вместо проволочного каркаса – по-
верхность, плавн залитая цветом
>> colormap(hot)     % палитра
```

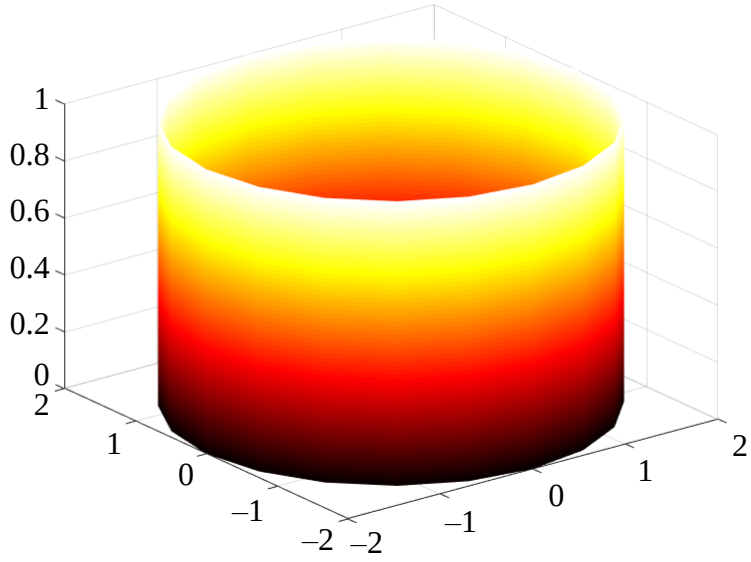

Рис. 1.11. Построение цилиндра с помощью графических средств MATLAB

Однако оценить подлинные возможности MATLAB можно, если в качестве аргумента использовать не скалярную величину, а функцию.

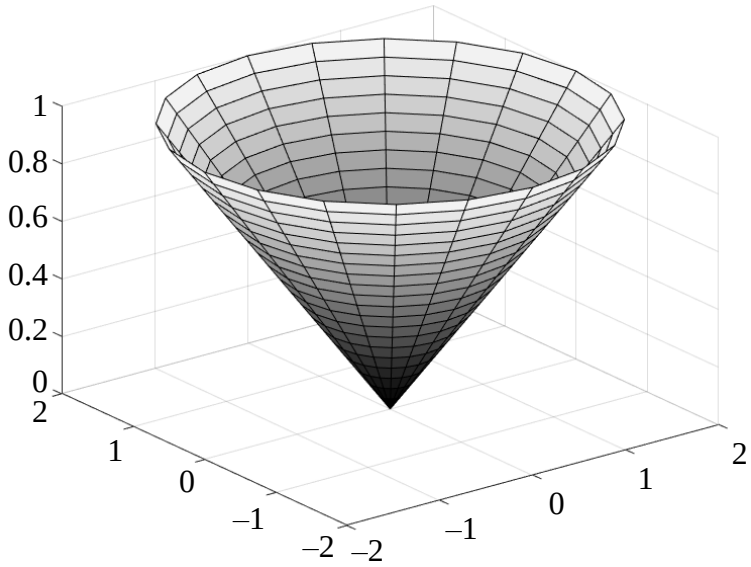

Рис. 1.12. Построение конуса с помощью графических средств MATLAB

Пусть R меняется от нуля до 2 с шагом 0.1:

```
>> R=0:0.1:2;
>> [X,Y,Z]=cylinder(R);
>> surf(X,Y,Z)
>> shading faceted % тип поверхности – «проволочный каркас»
>> colormap(gray)
```

1.13. Основы программирования в среде MATLAB

1.13.1. Операторы управления вычислительным процессом

Все операторы циклов и условных переходов начинаются служебным словом `for, while, if, elseif, switch, case` и заканчиваются служебным словом *end*. Операторы, расположенные между ними, воспринимаются системой как составные части одного сложного оператора. Поэтому нажатие клавиши Enter при переходе к следующей строке не приводит к немедленному выполнению цикла.

Оператор цикла `while`

```
while <условие>
    <операторы>
end
```

Операторы внутри цикла выполняются до тех пор, пока выполняется условие после слова `while`. При этом среди операторов обязательно должны быть такие, которые изменяют переменную, записанную в условии цикла.

Пример:

```
function sum_i
S = 0;                  % начальное значение суммы
i=1;                    % счетчик суммы
while i <= 20           % цикл (работает пока i< = 20)
     S=S+i;             % подсчитывается сумма
     i=i+1;             % увеличивается счетчик на 1
end                     % конец цикла
disp(S);                % отображение суммы 210 на
экране
```

Оператор цикла `for`

```
for <имя>= <нач. значение> : <шаг> : <конечное значение>
    <операторы>
end
```

Сравните два оператора цикла. В цикле `for` нет ни команд, изменяющих переменную цикла, ни условий проверки на его окончание. Все это автоматически делает счетчик цикла.

Пример:

```
function search_max
a = [3 6 5 3 6 9 5 3 1 0];
m = a(1);                          % текущее максимальное
значение
for i=1:length(a)      % цикл от 1 до конца вектора с
                                   % шагом 1 (по умолчанию)
    if m < a(i)                    % если a(i) > m,
         m = a(i);                 % то m = a(i)
    end
end                                % конец цикла for
disp(m);
```

Чтобы досрочно выйти из цикла (например, при выполнении какого-нибудь условия) применяют оператор `break`.

Условный оператор `if`

```
if <выражение>
<операторы 1>        % выполняются, если истинно условие
else
<операторы 2>        % выполняются, если условие ложно
end
```

Условие может быть составным, т. е. состоять из нескольких простых условий, объединенных знаками логических операций: `&` (и), `|` (или), `~` (не). Можно усложнить конструкцию оператора, введя дополнительные условия командой `elseif`:

```
if <выражение 1>
<операторы 1>    % выполняются, если истинно выражение 1
elseif <выражение 2>
<операторы 2>    % выполняются, если истинно выражение 2
...
elseif <выражение N>
<операторы N>    % выполняются, если истинно выражение N
end
```

Пример:

```
function my_if
x = 1;
if x >= 0 & x <= 2
   disp('x принадлежит диапазону от 0 до 2');
else
   disp('x  не принадлежит диапазону от 0 до 2');
end
```

Условный оператор switch

```
switch expr
          case case_expr1
            <операторы1>
          case case_expr2
            <операторы2>



         ...
          otherwise,
            <операторы>
end
```

Пример:

```
function upper_symbol
ch='c';
switch ch
    case 'a', ch='A';
    case 'b', ch='B';
    case 'c', ch='C';
    case 'd', ch='D';
    case 'e', ch='E';
   ...
    case 'z', ch='Z';
end

disp(ch);
```

1.13.2. Создание файл-функций

При написании программ удобным способом упрощения повторяющихся процедур является создание собственных файл-функций (рис. 1.13, 1.14).

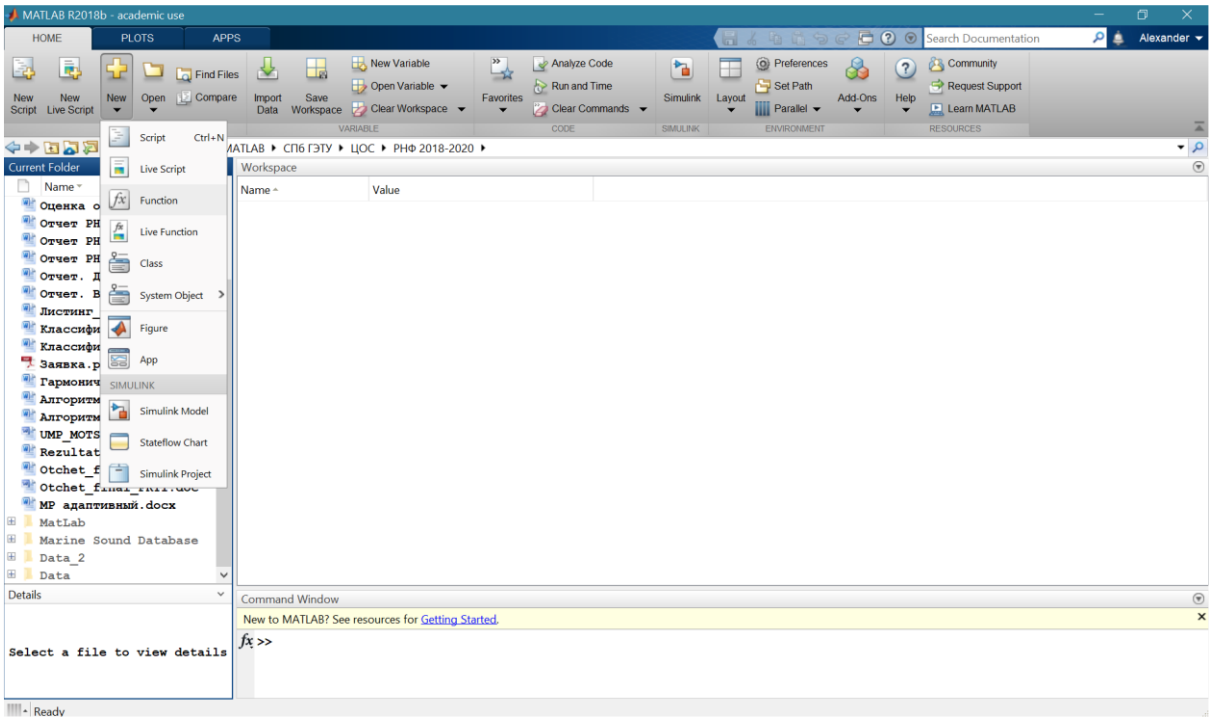

Рис. 1.13. Создание новой функции

Выполните команды *New->Function*. На экране появится окно текстового редактора (*Editor*).

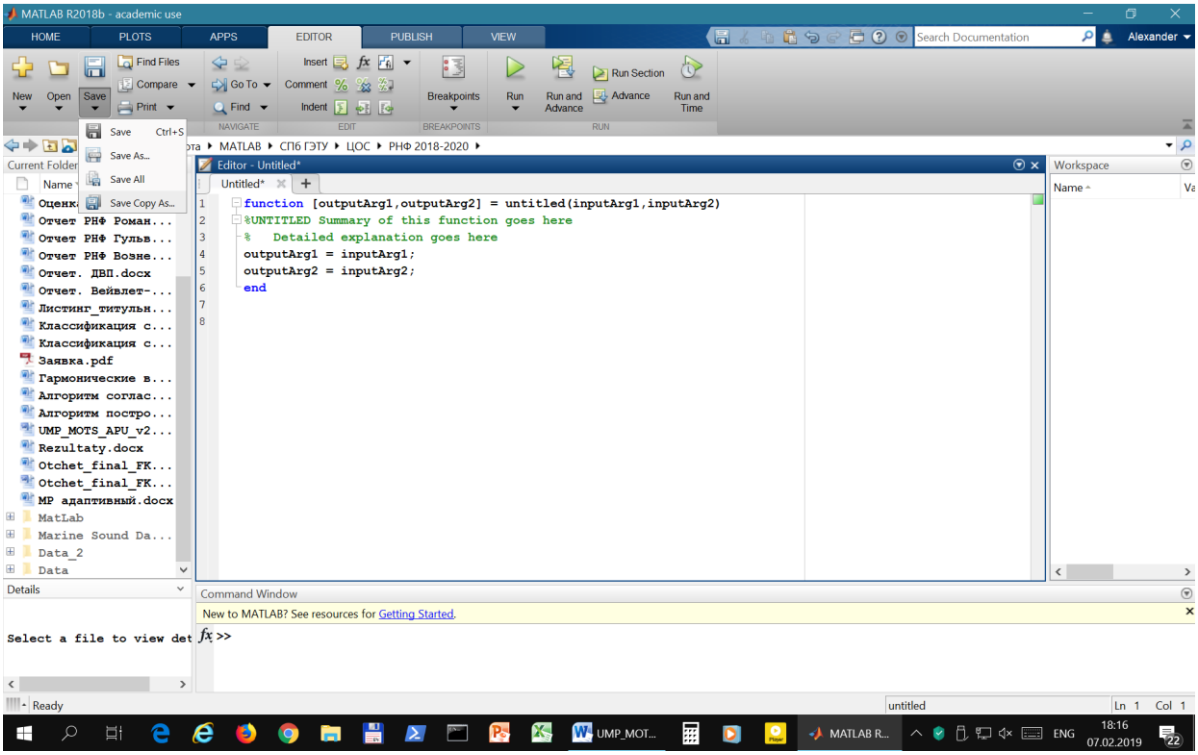

Рис. 1.14. Окно редактора

В нем наберите заголовок функции по форме:

```
function [y1,y2,…yn]=<имя процедуры>(<входные переменные>).
```

Теперь осталось только сохранить созданную функцию. В меню редактора выберите *Save->Save as*, и подтвердите свое согласие в открывшемся каталоге. Теперь вы можете пользоваться своей функцией точно так же, как раньше пользовались, например, функцией sin.

Для примера создадим файл-функцию, вычисляющий сразу три функции:

```
y1=400 sin(x)/x;
y2=x2;
y3=400-x2;
```

Назовем эту функцию «трио», а все «y» объединим в одну матрицу размером ($size(x)\times 3$):

```
function y= trio(x)
y(:,1)=400*sin(x)./x;
y(:,2)=x.^2;
y(:,3)=400-x.^2;
```

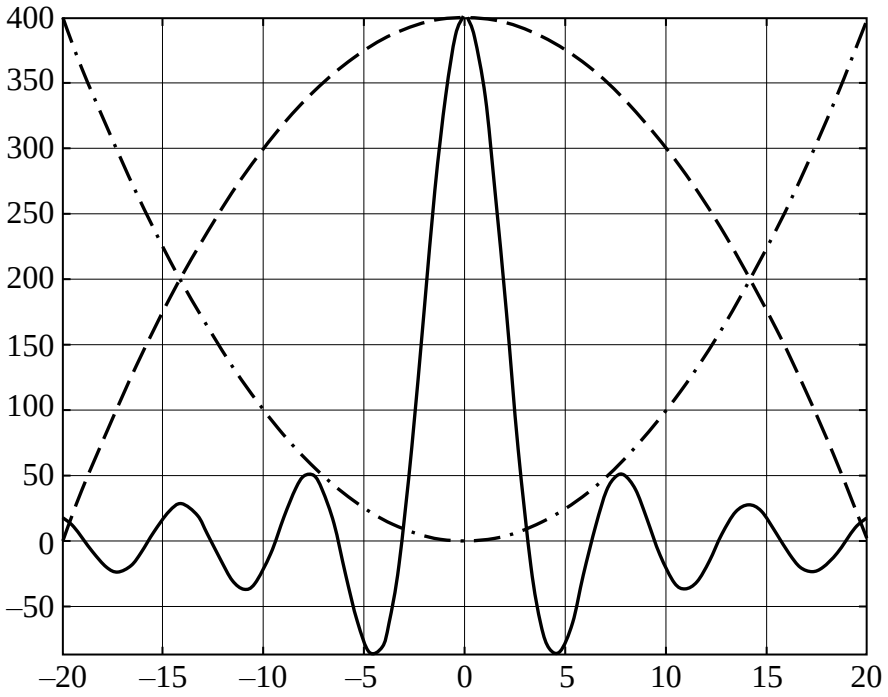

Рис. 1.15. График функции «трио»

На рис.1.15 приведены графики функций:

```
fplot('trio',[-20 20]), grid
title('График функции "trio"').
```

2. МЕТОДЫ ОПТИМИЗАЦИИ

Рассмотрим следующую задачу оптимизации, называемую также задачей математического программирования:

$$\begin{aligned} & f(x) \to \min; \\ & h(x) = 0; \\ & g(x) \le 0, \end{aligned} \tag{2.1}$$

где $x \in \mathbb{R}^n$ – переменная (параметр) оптимизации, $h: \mathbb{R}^n \to \mathbb{R}^m$ и $g: \mathbb{R}^n \to \mathbb{R}^p$ – функциональные ограничения, $f: \mathbb{R}^n \to \mathbb{R}$ – целевая функция.

Множество $D = \left\{x \in \mathbb{R}^n : h(x) = 0, g(x) \le 0\right\}$ называют допустимым множеством. Точка $x^* \in D$ называется решением, если для всех точек x из некоторой её окрестности $Q \cap D$ выполняется $f(x^*) \le f(x)$. Если $Q = D$, то говорят о глобальном минимуме, в противном случае – о локальном. Напомним также, что задачу максимизации можно свести к приведенному выше виду заменой знака целевой функции.

В общем случае поиск глобального и даже локального минимума – чрезвычайно сложная задача. Однако для некоторых классов задач существуют эффективные методы решения. Задачи, в которых целевая функция и ограничения являются выпуклыми, могут быть эффективно решены даже при достаточно больших размерностях.

В этой главе мы сделаем акцент на задачах условной оптимизации: рассмотрим необходимое условие экстремума, опишем идею методов внутренней точки для выпуклых задач, подробнее остановимся на задачах линейного и квадратичного программирования. Задачи этого класса возникают в различных областях, и для них существуют эффективные методы решения. Также они обладают огромной теоретической значимостью. Так, многие общие методы решения задач нелинейного программирования используют квадратичное программирование в качестве подзадачи (методы внутренней точки, последовательное квадратичное программирование и другие).

2.1. Необходимое условие экстремума

Предположим, что функции f, g, h непрерывно-дифференцируемые. Тогда *необходимое* условие экстремума задается теоремой Каруша–Куна–Такера.

Введем функцию Лагранжа $L(x,\lambda,\nu) = f(x) + \lambda^T h(x) + \nu^T g(x)$, где $\lambda \in \mathbb{R}^m$, $\nu \in \mathbb{R}^p$. Если x_0 – точка экстремума, тогда существуют такие векторы λ и ν, что

$$\nabla_x L(x_0,\lambda,\nu) = 0;$$
$$h_i(x_0) = 0, i = 1,\ldots,m;$$
$$g_i(x_0) \le 0, j = 1,\ldots,p;$$
$$\nu_i \ge 0, i = 1,\ldots,m;$$
$$\nu_i g_i(x_0) = 0, i = 1,\ldots,m.$$

Для справедливости теоремы, в точке x_0 должны также выполняться условия регулярности, однако мы не будем на них останавливаться, считая здесь и далее, что они выполнены.

Ограничение-неравенство $g_i(x) \le 0$ называется активным в точке x_0, если $g_i(x_0) = 0$. Если в точке минимума неравенство неактивно, то его можно не учитывать при минимизации. Это обеспечивает условие дополняющей нежесткости $\nu_i g_i(x_0) = 0$.

Частными случаями теоремы Каруша–Куна–Такера являются метод множителей Лагранжа для задач с ограничениями типа «равенство» и теорема Ферма для безусловной оптимизации.

Для задач выпуклой оптимизации условия ККТ являются также и достаточными условиями глобального минимума.

Теорема Каруша–Куна–Такера задает условия экстремума. Возникает вопрос – почему бы нам не искать минимум, непосредственно решая полученную систему? Оказывается, что в общем случае это чрезвычайно сложная задача. Даже если присутствуют только ограничения-равенства, то получим систему нелинейных алгебраических уравнений. Если же присутствуют ограничения-неравенства, то возможны 2^p наборов активных неравенств, для каждого из которых приходится решать систему уравнений. Несмотря на это, условия ККТ являются очень полезными для проверки решения и, как мы увидим далее, используются при разработке алгоритмов оптимизации.

2.2. Методы внутренней точки

Возможные способы решения задач условной оптимизации базируются на предположении, что мы умеем эффективно решать безусловные задачи. Тогда можно попытаться приблизить исходную задачу последовательностью

безусловных. Эту идею используют метод штрафных функций и метод барьеров. В методе штрафных функций штраф осуществляется при выходе за границы допустимой области, в барьерном методе – при подходе к границе изнутри, при этом начальная точка должна принадлежать допустимой области. По этим причинам класс таких методов называют «методами внутренней точки»?

Рассмотрим следующую задачу выпуклой оптимизации:

$$f(x) \to \min;$$
$$Ax = b;$$
$$g(x) \le 0.$$

где функции f и g – выпуклые и дважды дифференцируемые, а допустимая область непуста. Выпуклые задачи обладают множеством хороших свойств, в частности, любое решение является глобальным.

Для описания метода решения этой задачи последовательно изложим алгоритмы решения частных подзадач, при этом делая акцент на основных идеях, не останавливаясь на технических деталях.

2.3. Метод Ньютона

Рассмотрим задачу безусловной оптимизации $f(x) \to \min$.

Одним из способов решения таких задач является метод Ньютона, к которому можно прийти разными способами.

Пусть мы находимся в точке x, достаточно близкой от решения, и хотим сделать шаг d, такой чтобы прийти в минимум. Тогда должно выполняться условие экстремума $\nabla f(x+d) = 0$. Линеаризуя его, получим

$$\nabla f(x+d) \approx \nabla f(x) + \nabla^2 f(x)d = 0,$$

откуда

$$d = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x).$$

Рассмотрим также другой способ вывода шага Ньютона. Для этого аппроксимируем функцию $f(x)$ первыми тремя членами разложения в ряд Тейлора:

$$f(x+d) \approx f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d.$$

Мы хотим найти такое направление d, при котором функция наиболее уменьшится. Получили задачу безусловной квадратичной оптимизации:

$$\nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d \to \min_d.$$

Разрешая относительно d необходимое условие экстремума, придем к шагу Ньютона $d = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$.

Таким образом, следующую точку x_{k+1} можно найти по текущей x_k как $x_{k+1} = x_k + \alpha_k d_k$, где α_k – длина шага, в оригинальном методе равная единице.

Метод Ньютона сходится квадратично, т. е. значительно быстрее градиентных методов, однако в общем случае обладает малой областью сходимости. Также недостатком является необходимость вычисления и обращения функции Гессиана на каждой итерации.

2.4. Метод Ньютона с ограничениями типа равенств

Рассмотрим теперь задачу с линейными ограничениями типа «равенство»:

$$f(x) \to \min;$$

$$Ax = b.$$

Воспользуемся тем же приемом – аппроксимируем целевую функцию членами вплоть до квадратичного в разложении в ряд Тейлора:

$$f(x + d) \approx f(x) + \nabla f(x) d + d^T \nabla^2 f(x) d \to \min;$$

$$A(x + d) = b.$$

Предположим, что текущая точка x принадлежит допустимой области, т. е. $Ax = b$, тогда уравнение $A(x + d) = b$ эквивалентно $Ad = 0$. Таким образом, получили задачу квадратичной оптимизации с линейными ограничениями типа «равенство»:

$$\nabla f(x) d + d^T \nabla^2 f(x) d \to \min_d;$$

$$Ad = 0.$$

Уравнения ККТ для таких задач представляют собой линейную систему уравнений:

$$\begin{pmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} d \\ \lambda \end{pmatrix} = \begin{pmatrix} \nabla f(x) \\ 0 \end{pmatrix}.$$

На каждой итерации метода необходимо решать систему линейных уравнений, что может быть сделано вычислительно эффективно.

Заметим, что тот же результат можно получить, линеаризуя необходимые условия экстремума.

Еще раз отметим, что начальная точка в методе должна принадлежать допустимой области. В качестве упражнения предлагается модифицировать систему, чтобы устранить это условие.

2.5. Прямой метод внутренней точки

В предыдущих разделах мы научились решать задачи с ограничениями в виде линейных уравнений. Поэтому попытаемся свести общую задачу к этому случаю, например, представив в следующем виде:

$$f(x) + \sum_{i=1}^{p} I(g_i(x));$$

$$Ax = b,$$

где

$$I(y) = \begin{cases} 0, & y \le 0 \\ \infty, & y > 0. \end{cases}$$

К сожалению, индикаторная функция $I(y)$ недифференцируема, поэтому мы не можем воспользоваться ранее рассмотренными методами. Возможный выход состоит в аппроксимации, например: $I(y) \approx -\frac{1}{\tau}\log(-y)$. Тогда, введя обозначение $\varphi(x) = -\sum_{i=1}^{p} I(g_i(x))$ и домножая целевую функцию на τ, получим следующую аппроксимацию исходной задачи:

$$\tau f(x) + \varphi(g(x)) \to \min;$$

$$Ax = b. \qquad (2.2)$$

Полученная задача является выпуклой, и точность аппроксимации увеличивается при увеличении τ. Очевидным способом решения представляется решение полученной аппроксимации при достаточно большом τ. Однако при больших τ решение полученной задачи методом Ньютона сопряжено с некоторыми трудностями, поскольку функция Гессиана возле границы изменяется

чрезвычайно быстро. Поэтому в прямом методе внутренней точки решают последовательность полученных при аппроксимации задач (2.2) при увеличивающихся τ. При этом начальная точка x_0 для каждой итерации по τ берется как оптимальное значение x^* для предыдущего τ.

2.6. Прямо-двойственный метод внутренней точки

Можно показать, что для фиксированного τ точка x является решением задачи (2.1), если существуют такие λ и ν, что

$$\begin{aligned} & \nabla f(x) + \nu^T g(x) + A^T \lambda = 0; \\ & Ax = b; \\ & \nu \geq 0; \\ & g(x) \leq 0; \\ & \nu_i g_i(x) = -\frac{1}{\tau}. \end{aligned} \qquad (2.3)$$

Эта система совпадает с системой ККТ для задачи (2.1), за исключением условия дополняющей нежесткости, которое в исходной задаче имеет вид $\nu_i g_i(x) = 0$, т. е. введение логарифмического барьера эквивалентно возмущению условий ККТ.

Прямо-двойственный метод стартует из внутренней точки, а далее на каждой итерации производится решение линеаризованной возмущенной системы ККТ (2.3) (без учета неравенств), которое задает шаг метода Ньютона по прямой переменной x и двойственным λ, ν. Этот метод для многих задач эффективнее прямого.

2.7. Линейное программирование

Линейное программирование – частный случай математического программирования, когда целевая функция и ограничения линейные. Это наиболее простой класс выпуклых задач оптимизации. Для задач линейного программирования существуют эффективные алгоритмы и программы, позволяющие решать задачи с большим числом переменных.

Рассмотрим задачу линейного программирования в форме, принятой в команде linprog системы MATLAB:

$$\begin{aligned} & f^T x \to \min; \\ & Ax \leq b; \end{aligned}$$

$$A_{eq}x = b_{eq};$$

$$lb \le x \le ub.$$

Допустимое множество D, задаваемое ограничениями, является многогранником, не обязательно ограниченным, а линии уровня целевой функции – гиперплоскостями. На рис. 2.1 приведен пример для задачи:

$$x_1 + x_2 \to \max;$$

$$x_1 + 2x_2 \le 5;$$

$$3x_1 + 2x_2 \le 11;$$

$$-x_1 + x_2 \le 1;$$

$$x_1 \ge 0;$$

$$x_2 \ge 0.$$

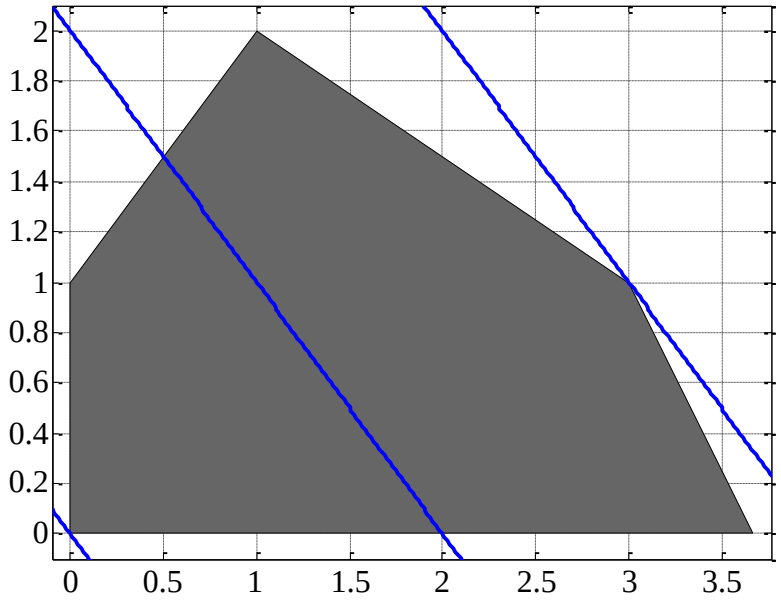

Рис. 2.1. Допустимое множество и линии уровня

Исходя из геометрии допустимой области, возможны следующие случаи:

– нет решения (когда допустимое множество пусто);

– целевая функция неограничена;

– единственное решение (вершина многогранника);

– континуум решений (ребро многогранника; однако в этом случае оптимальное значение функции единственно).

Основными алгоритмами решения задач линейного программирования являются методы внутренней точки и симплекс-метод. Симплекс-метод использует тот факт, что решение, если оно существует, обязательно находится в вершине (или на ребре) многоугольника. По сути симплекс-метод является «умным перебором» вершин многогранника.

Для задач больших размерностей использование прямо-двойственного метода внутренней точки предпочтительней. Вывести для него шаг Ньютона предлагается в качестве упражнения.

В пакете Optimization Toolbox системы MATLAB для решения задач линейного программирования используется функция linprog. Она имеет следующий синтаксис:

```
[xval,fval,exitflag] = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

Здесь x0 и options являются необязательными параметрами, задающими начальную точку и настройки решателя. Если ограничения какого-нибудь типа отсутствуют, то соответствующие матрица и вектор заменяются на пустой массив. Так, если отсутствуют ограничения-равенства, то это можно указать как `Aeq = []; beq = []`.

Выходные переменные `xval` и `fval` возвращают оптимальное значение аргумента и функции соответственно. Переменная `exitflag` отображает статус решения задачи. Различные ее значения можно посмотреть в документации, здесь же отметим наиболее важные случаи:

– решение найдено;

– допустимая область пуста;

– целевая функция неограничена.

Покажем как можно решить задачу (см. рис. 2.1):

```
f = -[1 1];
lb = [0 0];
ub = [];
beq = [];
Aeq = [];
A = [1 2; 3 2; -1 1];
b = [5 11 1];
[xval, fval, exitflag] = linprog(f, A, b, Aeq, beq, lb, ub);
```

Подробнее про использование параметра options будет рассказано далее, здесь же проиллюстрируем установку в качестве метода решения метод внутренней точки (по умолчанию используется симплекс-метод):

```
options = optimset('Algorithm', 'interior-point');
[x,fval,exitflag] = linprog(f, A, b, Aeq, beq, lb, ub, [],
options);
```

Рассмотрим пример задачи, приводящей к линейному программированию.

Пример. Транспортная задача. Есть два завода, которые производят товар для трех магазинов. Стоимость перевозки из заводов в магазины:

20	25	55
35	15	40

Известно, что спрос товара в первом магазине составляет 100 единиц, во втором 150, в третьем 200. В то же время первый завод производит 200 единиц товара, второй 250. Требуется минимизировать затраты на перевозку.

Решение. Обозначим как x_{ij} количество товара, перевозимое из i-го завода в j-й магазин. Тогда требуется минимизировать затраты $20x_{11} + 25x_{12} + 55x_{13} + 35x_{21} + 15x_{22} + 40x_{23}$ при ограничениях, обеспечивающих выполнение спроса $x_{11} + x_{21} \geq 100$, $x_{12} + x_{22} \geq 150$, $x_{13} + x_{23} \geq 200$ и ограничениях на производство $x_{11} + x_{12} + x_{13} \leq 200$, $x_{21} + x_{22} + x_{23} \leq 250$. Также очевидно следует наложить требование неотрицательности всех x_{ij}.

Полученную задачу можно решить следующим скриптом:

```
c = [20 25 55 35 15 40];
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 = [-100; -150; -200; 200; 250];
lb = zeros(6, 1);
[x,fval,exitflag] = linprog(c, A, b, [], [], lb, [])
```

Откуда $x_{11} = x_{12} = 100$, $x_{13} = x_{21} = 0$, $x_{22} = 50$, $x_{23} = 200$.

2.8. Квадратичное программирование

Квадратичным программированием называют задачи с линейными ограничениями и квадратичной целевой функцией:

$$\frac{1}{2}x^T Qx + q^T x \to \min;$$

$$Ax \leq b;$$

$$Cx = d.$$

Предполагается, что матрица Q симметричная, т. е. $Q = Q^T$. Если же это не так, то ее можно привести к симметричному виду заменой $Q_{\text{new}} = \frac{Q + Q^T}{2}$, при этом решение не изменится. В системе MATLAB такая замена производится автоматически.

Если матрица Q является положительно определенной, то задача выпуклая, решение является глобальным и может быть эффективно найдено. В противном случае могут быть локальные решения, а поиск глобального – NP-трудная задача. Далее будем рассматривать только случай положительно-определенной матрицы.

Для решения задач квадратичного программирования может использоваться большое число алгоритмов. Так, например, метод Франка-Вульфа основан на линеаризации целевой функции и последовательном решении задач линейного программирования.

Для задач большой размерности, как правило, наиболее эффективным является прямо-двойственный метод Ньютона. Вывести шаг метода Ньютона предлагается в качестве упражнения.

В пакете Optimization Toolbox системы MATLAB для решения задач квадратичного программирования используется функция quadprog. Она имеет следующий синтаксис:

```
[x, fval, exitflag] = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
```

Здесь матрицы и векторы соответсвуют следующей форме записи задачи квадратичного программирования:

$$\frac{1}{2}x^T Hx + f^T x \to \min;$$

$$Ax \le b;$$

$$A_{\text{eq}}x = b_{\text{eq}};$$

$$lb \le x \le ub,$$

а необязательные параметры x0 и options задают начальную точку и параметры. Выходные переменные такие же, как для линейного программирования.

Рассмотрим в качестве примера следующую задачу:

$$4x_1^2 + 2x_1x_2 + x_2^2 + 2x_1 + 3x_2 \to \min;$$

$$x_1 - x_2 \le 0;$$

$$x_1 + x_2 \text{ J } 4;$$

$$x_1 \text{ J } 3.$$

Ее решение можно найти следующей последовательностью команд:

```
H = 2*[4 1; 1 1];
f = [2; 3];
A = [-1 1; 1 1; 1 0];
b = [0 4 3];
[x, fval, exitflag] = quadprog(H, f, A, b);
```

Если мы будем решать эту же задачу, но пытаться максимизировать функцию

```
H = -2*[4 1; 1 1];
f = -[2; 3];
A = [-1 1; 1 1; 1 0];
b = [0 4 3];
options = optimset('Algorithm', 'active-set');
[x, fval, exitflag] = quadprog(H, f, A, b, [], [], [], [], [],
options),
```

то MATLAB выдаст решение и `exitflag=1`. Однако теперь задача невыпуклая и целевая функция неограничена. Если использовать алгоритм внутренней точки

```
H = -2*[4 1; 1 1];
f = -[2; 3];
A = [-1 1; 1 1; 1 0];
b = [0 4 3];
optio optimset('Algorithm','interior-point-convex');
[x, fval, exitflag] = quadprog(H,f,A,b,[],[],[],[],[], op-
tions),
```

то MATLAB выдаст предупреждение о невыпуклости задачи и возвратит `exitflag=-6`.

Таким образом, рекомендуется заранее проверять задачу на выпуклость или пользоваться алгоритмом `interior-point-convex`. Также рекомендуется для любых задач оптимизации всегда проверять решение, которое выдается решателем.

Рассмотрим пример задачи оптимального управления, приводящей к квадратичному и линейному программированию.

Пример. Задана линейная система $x_{k+1} = Ax_k + Bu_k$ с начальным условием x_0, где $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}$, $k = 1, \ldots, N-1$. Требуется найти управление, переводящее систему в точку x_N^{des} при ограничениях $|u_k| \le u_{\max}$ и минимизирующее функционал:

а) $J = \sum_{k=0}^{N-1} u_k^2$;

б) $J = \sum_{k=0}^{N-1} |u_k|$.

Решение. Одним из подходов к решению таких задач является последовательный; зная x_0 и u_k для всех k, можем вычислить x_N:

$$x_1 = Ax_0 + Bu_0;$$

$$x_2 = Ax_1 + Bu_1 = A(Ax_0 + Bu_0) + Bu_1 = A^2x_0 + ABu_0 + Bu_1;$$

$$\vdots$$

$$x_N = A^N x_0 + \sum_{k=1}^{N} A^{N-k} B u_{k-1}.$$

В матричном виде это выражение можно записать как $x_N = A^N x_0 + Hu$, где $H = [A^{N-1}B \quad A^{N-2}B \quad \ldots \quad AB \quad B]$.

Таким образом, для квадратичного функционала $J = \sum_{k=0}^{N-1} u_k^2$ получили задачу квадратичного программирования:

$$u^T u \to \min;$$

$$Hu = x_N - A^N x_0;$$

$$u \le u_{\max};$$

$$u \ge -u_{\max}.$$

Решим эту задачу в системе MATLAB при $A = \begin{pmatrix} 1 & 0.4 & 0.8 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 1 \\ 0 \\ 0.3 \end{pmatrix}$, $x_0 = (1 \quad 1 \quad 1)^T$, $x_N^{des} = (7 \quad 2 \quad -6)^T$, $N = 30$:

```
n = 3;
N = 30;
```

```
A = [-1 0.4 0.8; 1 0 0; 0 1 0];
b = [1 0 0.3]';
xdes = [7 2 -6]';
u_max = 5;
x0 = ones(n, 1);
for k = 1:N
  H(:, k) = A^(N-k)*b;
end
Aeq = H;
beq = xdes - A^N*x0;
Q = 2*eye(N);
f = zeros(N, 1);
lb = -u_max*ones(N, 1);
ub = u_max*ones(N, 1);
[u, fval, exitflag] = quadprog(Q, f, [], [], Aeq, beq,
lb, ub)
stairs(0:N-1, u)
axis tight
xlabel('t')
ylabel('u')
```

Рассмотрим теперь функционал $J = \sum_{k=0}^{N-1} |u_k|$. Для того, чтобы прейти к задаче линейного программирования, заметим, что задача минимизации модуля скалярной величины $|x| \to \min$ эквивалентна задаче

$$s \to \min;$$

$$x \le s;$$

$$x \ge -s.$$

Таким образом, исходная задача оптимального управления сводится к задаче линейного программирования с переменными оптимизации $u \in \mathbb{R}^N$ и $s \in \mathbb{R}^N$:

$$\sum_{k=0}^{N-1} s \to \min;$$

$$Hu = x_N - A^N x_0;$$

$$u \in J \; s;$$

$$u \in - s;$$

$$u \in J \; u_{\max};$$

$$u \in - u_{\max}.$$

Для тех же данных решение в системе MATLAB:

```
n = 3;
N = 30;
A = [-1 0.4 0.8; 1 0 0; 0 1 0];
b = [1 0 0.3]';
xdes = [7 2 -6]';
u_max = 5;
x0 = ones(n,1);
for k = 1:N
  H(:, k) = A^(N-k)*b;
end
Aeq = [H zeros(n, N)];
beq = xdes – A^N*x0;
c = [zeros(N, 1); ones(N, 1)];
lb = -u_max*ones(2*N, 1);
ub = u_max*ones(2*N, 1);
A = [-eye(N) -eye(N); eye(N) -eye(N)];
b = [zeros(N, 1); zeros(N, 1)];
[xval, fval, exitflag] = linprog(c, A, b, Aeq, beq, lb, ub)
u = xval(1:N);
stairs(0:N-1, u)
axis tight
xlabel('t')
ylabel('u')
```

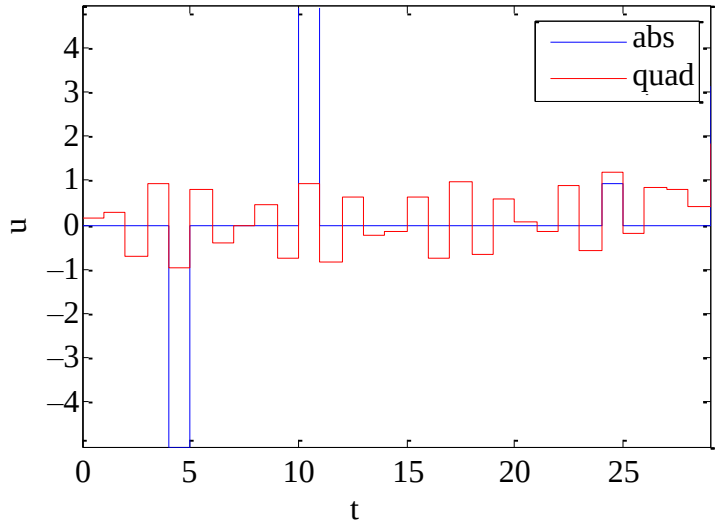

Рис. 2.2. Оптимальные управления

График найденных управлений представлен на рис. 2.2.

2.9. Решение задач нелинейной оптимизации в системе MATLAB

Перейдем к решению нелинейных задач математического программирования в системе MATLAB. Для этих целей существует большое число решателей и пакетов, среди которых особо отметим CVX и YALMIP за простоту использования и большие возможности. Однако здесь остановимся на использовании пакета Optimization toolbox. Подробное описание можно посмотреть в документации, мы же отметим лишь основные моменты.

Optimization toolbox позволяет решать различные задачи, в том числе условной, безусловной и многокритериальной оптимизации. При решении задачи необходимо сначала определить, к какому классу она относится, затем выбрать подходящую функцию и алгоритм.

2.10. Безусловная оптимизация

Для решения задач безусловной оптимизации рекомендуется использовать функцию `fminunc`. Можно применять также функцию `fminsearch`, однако она использует методы нулевого порядка (безградиентные) и в общем случае медленнее. В то же время она позволяет решать негладкие задачи.

Команда `fminunc` имеет следующий синтаксис:

```
[x, f, exitflag] = fminunc(fun, x0, options)
```

В отличие от функций `linprog` и `quadprog`, для этой команды обязательно указывать начальную точку `x0`. Поскольку функция ищет локальный минимум, то различные начальные точки могут приводить к различным решениям.

Входной аргумент `fun` является ссылкой на минимизируемую функцию. Ее можно записать, например, как функцию в отдельном файле или анонимную функцию. Рассмотрим оба варианта на стандартном примере функции Розенброка: $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$.

В первом случае создается функция:

```
function f = rosenbrock(x)
f = 100*(x(2) – x(1)^2)^2 + (1 – x(1))^2;
```

Далее в командной строке или в другом файле осуществляется минимизация:

```
x0 = [1; 0.5] % пример начальных условий
[x, f, exitflag] = fminunc(@rosenbrock, x0, options)
```

Во втором случае все команды могут быть записаны в одном файле:

```
rosenbrock = @(x)(100*(x(2) – x(1)^2)^2 + (1-x(1))^2);
x0 = [1; 0.5] % пример начальных условий
[x fval] = fminunc(rosenbrock, x0, options)
```

Многие алгоритмы для этой и других функций используют градиент функции. По умолчанию используется конечно-разностная аппроксимация. Для увеличения скорости и точности рекомендуется, если это возможно, указывать явную формулу для градиента. Так, функция Розенброка с градиентом может быть записана как

```
function [f g] = rosen_grad(x)
f = 100*(x(2) – x(1)^2)^2 + (1-x(1))^2;
if nargout > 1
 g = [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
 200*(x(2)-x(1)^2)];
End
```

Теперь укажем, что градиент будет вычисляться функцией и запустим процесс оптимизации:

```
Options = optimoptions(@fminunc,'Algorithm','quasi-newton',
'GradObj','on');
x0 = [-1; 2];
[x fval] = fminunc(@rosen_grad, x0, options);
```

Рекомендуется всегда проверять правильность вычисления градиента. Для этого присутсвует опция DerivativeCheck. Так, тот же пример с проверкой вычисления градиента:

```
options = optimoptions(@fminunc, 'Algorithm', 'quasi-
newton', ...
            'GradObj', 'on', 'DerivativeCheck', 'on');
x0 = [-1; 2];
[x fval] = fminunc(@rosen_grad, x0, options);
```

Перед началом итераций метода выведено сообщение DerivativeCheck successfully passed, подтверждающее правильность вычисления градиента.

Рассмотрим теперь подробнее настройки options. Основные используемые опции, помимо вышеперечисленных, это отображение информации Display и критерии останова.

Параметр Display устанавливает настройки отображения работы алгоритма. По умолчанию установлено значение `final`, при котором информация выводится только после окончания работы. Другим часто используемым значением является `iter`, при котором выводится информация после каждой итерации.

Для настроек точности алгоритмов используются следующие параметры:

– `TolFun` – нижняя оценка на изменение $f(x)$; алгоритм заканчивает работу, если текущее изменение меньше;

– `TolX` – нижняя оценка на шаг по x; алгоритм заканчивает работу, если текущий шаг меньше;

– `MaxFunEvals` – максимальное число вычислений функции;

– `MaxIter` – максимальное число итераций.

Параметры `TolX` и `TolFun` не рекомедуется выставлять меньше $1e-14$.

2.11. Условная оптимизация

Для задач условной оптимизации, не принадлежащих к задачам линейного и квадратичного программирования, могут использоваться функции `fminbnd`, `fmincon` и `fseminf`.

Функция `fminbnd` находит минимум функции $f(x)$ скалярного аргумента x на некотором интервале $[x_{\min}, x_{\max}]$.

Функция fmincon является основной для решения задач условной оптимизации. Она имеет следующий синтаксис:

```
[x,fval,exitflag] =
fmicon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

где A, b, Aeq, beq, lb, ub задают линейные ограничения, nonlcon – нелинейные. Задание начальной точки x0 обязательно.

Нелинейные ограничения nonlcon задаются в виде отдельного файла или анонимной функции. Аналогично целевой функции возможно использование градиента нелинейных ограничений.

При использовании функции fmincon рекомендуется сперва использовать алгоритм внутренней точки interior-point, во вторую очередь – последовательное квадратичное программирование sqp и метод активных множеств active-set.

Рассмотрим пример из документации: минимизация функции Розенброка при ограничении $x_1^2 + x_2^2 \text{ J } 1$.

Целевая функция:

```
function f = rosenbrock(x)
f = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
Ограничения:
function [c, ceq] = unitdisk(x)
c = x(1)^2 + x(2)^2 - 1;
ceq = [ ];
```

Функция, задающая нелинейные ограничения, должна возвращать значения как неравенств $g(x) \text{ J } 0$, так и равенств $h(x) = 0$ даже в том случае, когда какое-нибудь из этих ограничений отсутствует. В этом случае возвращается пустой массив.

После задания целевой функции и ограничений запустим алгоритм минимизации:

```
options = optimoptions(@fmincon, 'Display', 'iter', 'Algo-
rithm','interior-point');
[x, fval, exitflag] = fmincon(@rosenbrock,[0 0],
[],[],[],[],[],[],@unitdisk,options);
```

Функция fseminf решает задачи полубесконечного программирования. Они называются так, потому что в них конечное число переменных и бесконечное число ограничений. В общем виде такие задачи можно записать как

$$f(x) \to \min;$$

$$K_i(x, w_i) \geq 0;$$

$$w_i \in I_i,$$

где I_i – интервал.

Такие задачи возникают во множестве приложений. Рассмотрим, например, робастное линейное программирование:

$$cx \to \min;$$

$$ax \geq b,$$

где c, b – заданные числа; x – скалярная переменная оптимизации; $a \in [a_{\min}, a_{\max}]$, т. е. неравенство $ax \geq b$ задает бесконечное число ограничений. В рассмотренном примере из оптимального управления такая задача возникнет, если какой-нибудь элемент матрицы A известен с точностью до принадлежности интервалу.

Подробнее с функцией и ее использованием можно ознакомиться в документации.

2.12. Упражнения

1. Найти точку экстремума, используя ККТ: $x_1 + x_2 \to \min$ при $x_1^2 + x_2^2 \geq 2$.

2. Реализовать в системе MATLAB метод Ньютона для системы $\log(\exp(x) + \exp(-x)) \to \min$. Запустить из начальных точек $x_0 = 1$ и $x_0 = 1.1$. Когда метод сходится, а когда нет?

3. За сколько итераций метод Ньютона сходится из произвольной точки для функции $x_1^2 + x_2^2 \to \min$?

4. Реализовать метод Ньютона для задачи $x_1^4 + x_2^4 + x_1^2 + x_2^2 \to \min$ при ограничении $x_1 + x_2 = 1$ с начальными точками $x_0 = [-1 \quad 1]$ и $x_0 = [1 \quad 1]$. Как вы можете объяснить увиденное?

5. Решить в системе MATLAB задачу линейного программирования:

$$2x_1 - x_2 - 5x_3 \to \max;$$

$$4x_1 + 3x_2 + x_3 = 4;$$

$$-x_1 + 6x_2 - x_3 \leq 2;$$

$$7x_1 - x_2 + 2x_3 \geq 5;$$

$$x_1 \text{ i } 0;$$

$$x_2 \text{ i } 0.$$

6. Решите в системе MATLAB задачу квадратичного программирования:

$$x_1^2 + 2x_2^2 - 2x_1 - 6x_2 - 2x_1x_2 \text{ ® min};$$

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 \text{ J } 1;$$

$$- x_1 + 2x_2 \text{ J } 2;$$

$$x_1 \text{ i } 0;$$

$$x_2 \text{ i } 0.$$

7. Имеется завод, выпускающий шкафы и тумбочки. На сборку шкафа уходит 2,5 ч, тумбочки – 1 ч. Покраска и шкафа, и тумбочки занимает 3 ч. Упаковка шкафа занимает 1 ч, тумбочки – 2 ч. Требуется максимизировать прибыль, если на сборку можно потратить не более 20 ч, сушку – 30 ч, упаковку – 16 ч, а прибыль от продажи шкафа и тумбочки составляет 3 и 4 у. е. Свести к задаче оптимизации и найти ответ.

8. Сравнить быстродействие реализации симплекс-метода и метода внутренней точки команды `linprog`. Данные можно сгенерировать, например, следующим образом:

```
rng(1234)
n = 400;
m = 800;
A = [];
b = [];
lb = zeros(n, 1);
ub = [];
Aeq = randn(m, n);
x = 5*rand(n, 1) + 0.01;
beq = A*x;
c = ones(n, 1);
```

Время выполнения команды можно замерить, например, поместив ее между командами `tic` и `toc`. Попробуйте решить эту же задачу командой `fmincon` с разными настройками алгоритмов.

Дополнительные упражнения

9. Модифицируйте систему уравнений для шага метода Ньютона с ограничениями, чтобы он работал из начальных точек не из допустимой области. Протестируйте на задаче 4.

10. Выведите шаг прямо-двойственного метода для задачи линейного программирования.

11. Выведите шаг прямо-двойственного метода для задачи квадратичного программирования.

12. Другим способом решения задачи из примера 2 является оптимизация по x и u, не выражая в явном виде x_N. Таким образом, решите задачу из примера 2 с функционалом $J = \sum_{k=0}^{N-1} u_k^2$.

3. ЛАБОРАТОРНЫЕ РАБОТЫ

Предлагаемые лабораторные работы ориентированы на освоение студентами тех разделов математики и системного анализа, которые широко используются в инженерной практике и научных исследованиях. Предпринята попытка с использованием средств MATLAB охватить все этапы проектирования сложных технических систем: от операций над матрицами, векторами, визуализации и преобразования исходной информации в цифровой вид до ее статистической обработки и построения модели системы.

ЛАБОРАТОРНАЯ РАБОТА 1.

МАТРИЧНЫЕ ПРЕОБРАЗОВАНИЯ И ТРЕХМЕРНАЯ ГРАФИКА

Цель работы: освоение специфики матричных преобразований MATLAB и сравнительный анализ различных форм графического отображения результатов.

Теоретические положения. Пусть дана функция двух аргументов $z = f(x, y)$. В области определения $x \in [x_{\min}, x_{\max}]$, $y \in [y_{\min}, y_{\max}]$ командой `[X,Y] = meshgrid(xgv,ygv)` создайте координатную сетку с заданным шагом. Теперь можно создавать саму функцию, например:

```
>>[x,y]=meshgrid(-10:0.3:10,-10:0.3:10);
>> z=(sin(x)./x).*(sin(y)./y);
>> surf(x,y,z)
```

Полученная поверхность приведена на рис. 3.1. Можно произвольно выбрать шкалу цветовых оттенков (см. `help colormap`).

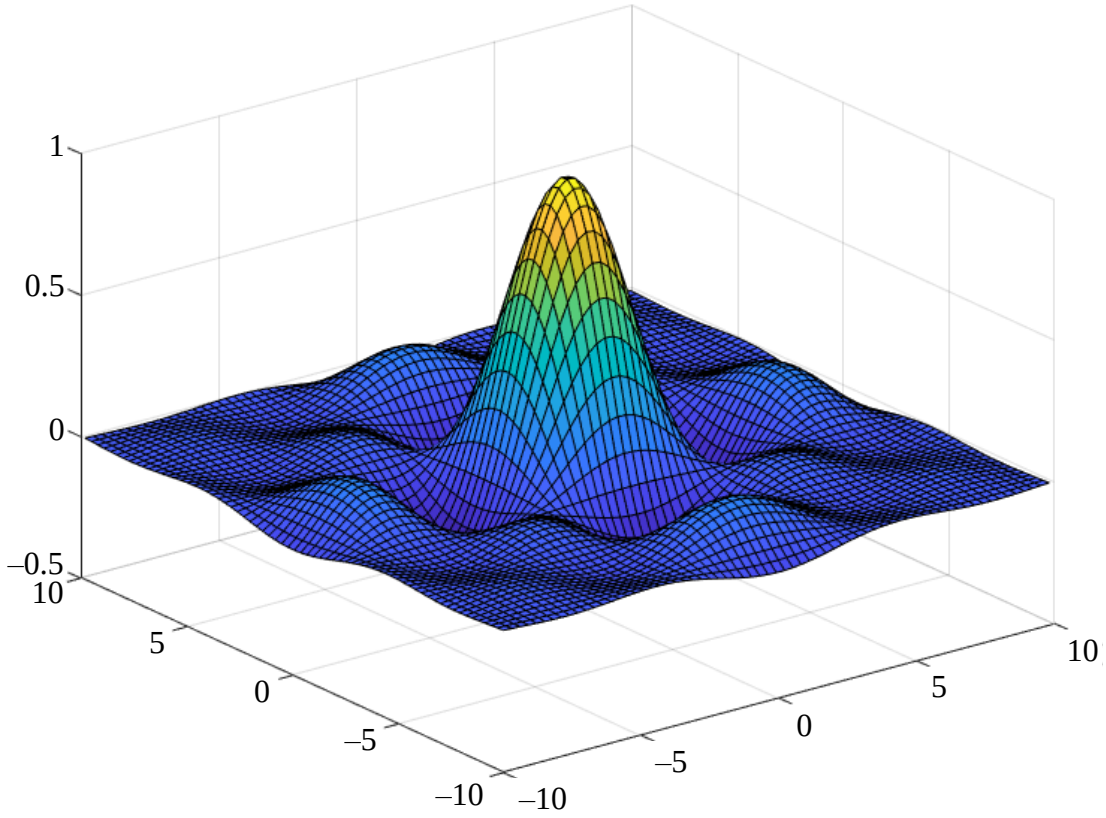

Рис. 3.1. Трехмерная поверхность

Второй способ состоит в формировании двух взаимно перпендикулярных плоскостей X и Y – аналогов двумерных осей ординат:

```
>> x=-10:10;
>> y=ones(1,21);
>> X=x'*y;
>> Y=y'*x;
```

Теперь используем координатные плоскости для построения конуса:

```
>> R1=sqrt(X.^2+Y.^2);
>> surf(X,Y,-R1);
```

или пирамиды (рис. 3.2):

```
>> R2=max(abs(X),abs(Y))
>> surf(X,Y,-R2);
```

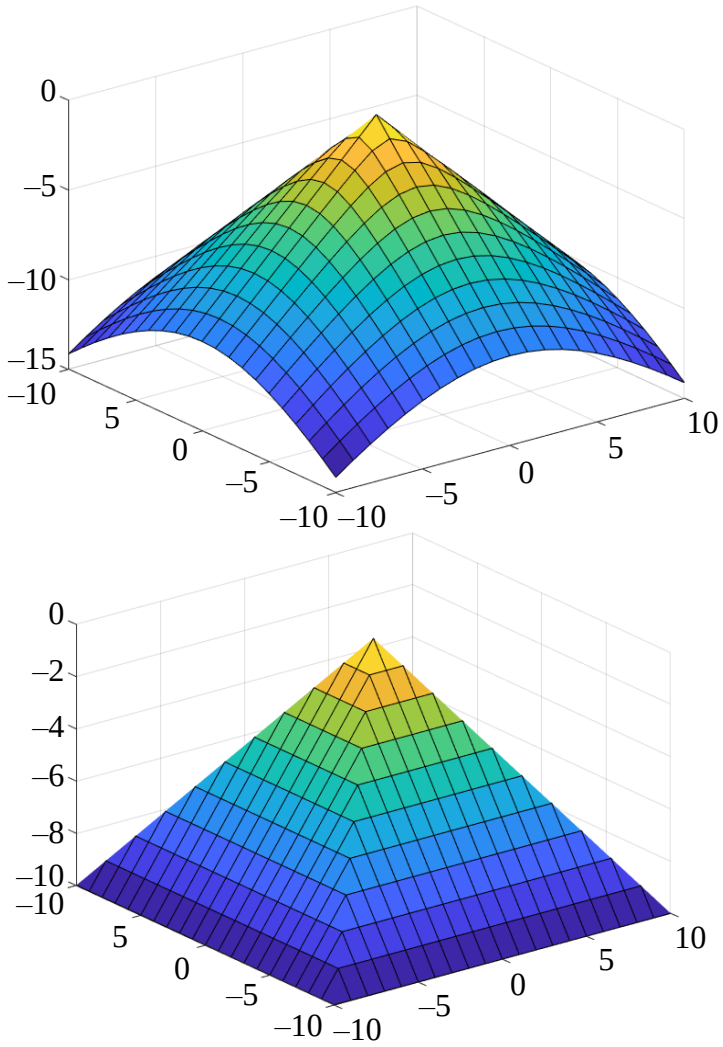

Рис. 3.2. Трехмерные фигуры

Матричные операции, вычисление функций от матриц. Кратко напомним основные матричные операции.

Определитель Δ:

$\Delta = \sum_{i=1}^{n} a_{ij} A_{ij}$ $j = 1 \text{K } n$, A_{ij} – алгебраическое дополнение;

Обратная матрица A^{-1}:

$A^{-1} = \dfrac{A^*}{\Delta A}$ A^* – союзная матрица;

Умножение матриц $\omega = \alpha\beta$. Элемент результирующей матрицы ω_{jk} на пересечении строки j и столбца k равен *сумме попарных произведений элементов строки* j матрицы α и столбца k матрицы β: $\omega_{jk} = \sum_{l} \alpha_{jl}\beta_{lk}$

Возведение в степень $n = \omega^p = \omega\omega\text{K}\,\omega$ осуществляется p-кратным умножением матрицы на себя.

Порядок проведения работы

Используемые команды MATLAB:

– `det(A)` – определитель матрицы A;

– `eig(A)` – собственные числа матрицы A;

– `eye(n)` – единичная матрица порядка n с единицами на главной диагонали;

– `inv(A)` – обратная матрица A^{-1};

– `sqrtm(A)` – матричный квадратный корень $A^{1/2}$;

– `logm(A)` – матричный логарифм;

– `expm(A)` – матричная экспонента e^{A};

– `kron` – кронекеровское умножение матриц.

1. В качестве исходной фигуры, на которой будем изучать матричные преобразования, выберем пирамиду (см. рис. 3.2). Присвоим ей имя R. Симметрия матрицы R относительно главной диагонали и антидиагонали делает такую матрицу вырожденной – ее определитель равен нулю (проверьте). Соответственно, большинство матричных операций для нее невыполнимо. Поэтому добавим к элементам на главной диагонали по единице, сложив ее с единичной матрицей (eye) того же размера. Теперь над матрицей можно производить как поэлементные, так и матричные операции.

2. Сравните (по графикам) результаты двух операций – обращения матрицы командой inv и поэлементного деления матрицы ones(n,n) на R.

3. Сравните матричные операции sqrtm(A), logm(A), expm(A) с аналогичными операциями, выполняемыми поэлементно.

4. Преобразуйте пирамиду R операциями врезки. «Отрежьте» какой-нибудь из углов, приравняв нулю выбранные элементы. Например:

```
R1(:,1:5)=0;
figure(2)
mesh(-R1)
R1(10:15,:)=4;
figure(3)
mesh(-R1)
surfl(-R1)           % освещенная поверхность (без кар-
касной сетки)
shading interp    % линейное изменение цвета
colormap('gray')  % палитра серого
```

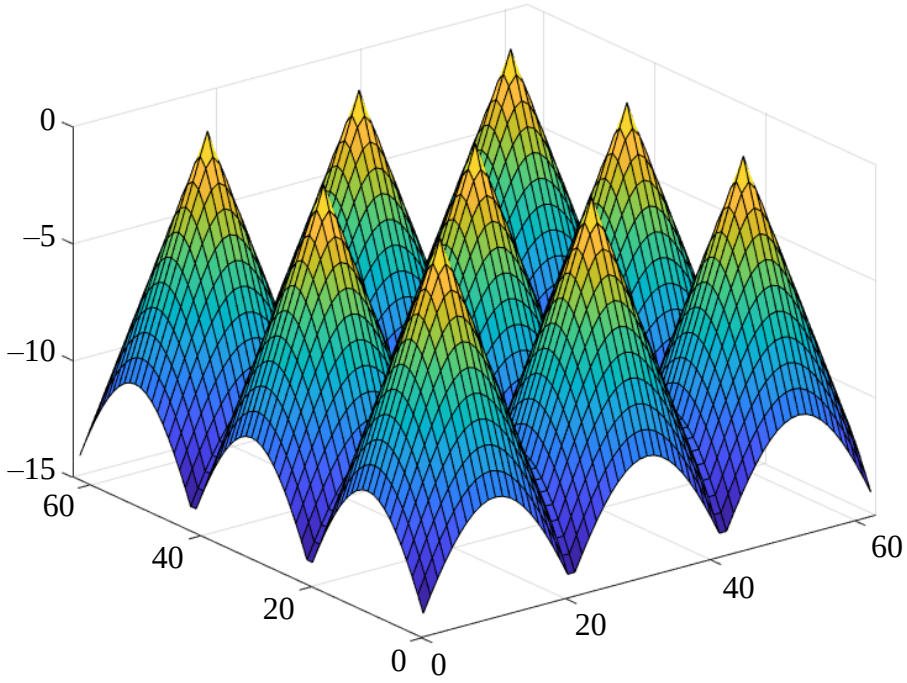

Рис. 3.3. Мультипликация операцией kron

5. Очень полезной операцией размножения массивов (мультиплицирования) является операция кронекеровского умножения матриц, в которой вся правая матрица умножается на каждый элемент левой:

$$\begin{bmatrix} 1 & 3 \\ -1 & 5 \end{bmatrix} \otimes \begin{bmatrix} 2 & 3 & 2 & 3 \\ 4 & 5 & 4 & 5 \\ 2 & 3 & -2 & -3 \\ 4 & 5 & -4 & -5 \end{bmatrix}$$

Создайте 9 конусов с помощью матрицы 3*3 (рис. 3.3).

Поменяйте порядок матриц в последней операции.

ЛАБОРАТОРНАЯ РАБОТА 2.

КОРРЕЛЯЦИОННЫЙ МЕТОД ИЗМЕРЕНИЯ ЗАДЕРЖКИ СИГНАЛА

Цель работы: анализ возможностей корреляционных методов выделения сигнала из шума и измерения его параметров.

Теоретические положения. *Корреляция* (correlation) и ее частный случай для центрированных сигналов (*ковариация*) – это мера схожести двух сигналов (рис. 3.4). Приведем один из вариантов использования корреляционной функции. Допустим, нам надо найти в тексте некоторый символ. Скользя по тексту, прикладываем символ к каждой букве, проверяя их сходство. Аналогично решается задача обнаружения некоторой последовательности известной формы и конечной длины T в принимаемом сигнале $y(t)$.

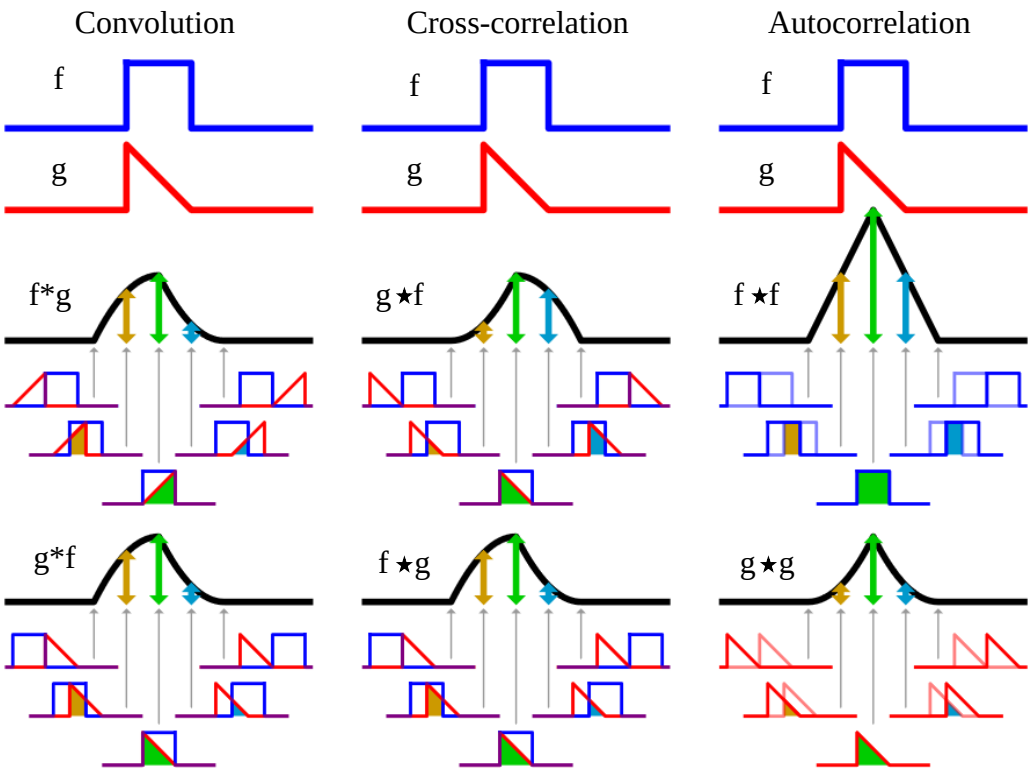

Рис. 3.4. Сравнение операций свертки, кросс-корреляции и автокорреляции

Для поиска этой последовательности в скользящем по сигналу $y(t)$ временном окне длиной T вычисляется интеграл от произведения процессов $y(t)$ и $x(t)$:

$$Z_{xy} = \int_0^T x(t) * y(t) dt.$$

В дискретной форме Z есть *скалярное произведение* векторов X и Y:

$$Z_{xy} = X * Y = \sum_{i=1}^{N} x(i) y(i).$$

Скалярное произведение используется как мера взаимосвязи двух векторов (процессов). Равенство нулю скалярного произведения означает крайнюю меру их независимости – их ортогональность. Чтобы проверить взаимосвязь векторов при различных сдвигах одного из них на время τ, необходимо Z_{xy} вычислять многократно, для каждого τ. Если еще ввести нормирующий множитель $1/T$ для исключения зависимости Z от длины реализации T, получим выражение, называемое *взаимной корреляционной функцией* (ВКФ):

$$R_{xy}(\tau) = \frac{1}{T} \int_J^{J+T} x(t) * y(t+\tau) dt,$$

Аналогично для непрерывного сигнала:

$$R_{xy}(\tau) = \frac{1}{N} \sum_{i=J}^{J+N} x(i) * y(i+\tau)$$

Теперь скалярное произведение превратилось в *свертку* двух векторов.

$R(\tau)$ широко используется в различных областях человеческой деятельности. Например, в медицине корреляционная функция позволяет обнаружить минимальные изменения состояния организма от воздействия какого-либо неблагоприятного фактора или оценить длительность воздействия лекарства на пациента. Социологи используют ВКФ для оценки влияния на население новых законодательных актов. Астрономам ВКФ позволила обнаружить периодичность в изменении свечения звезд (пульсары). В приведенном примере поиска в тексте заданного символа корреляционная функция использовалась для распознавания или обнаружения сигнала. Еще более часто корреляционная функция используется для определения времени прихода (задержки) сигнала. Поскольку максимальное сходство (следовательно, и максимальное скалярное произведение) будет в момент полного совпадения $x(t)$ и его копии в $y(t)$, по положению максимума корреляционной функции можно судить о времени прихода сигнала. Этот метод лежит в основе измерения расстояния до цели (ракеты, самолета), определения высоты полета самолета или глубины погружения подводной лодки. Чтобы определить направление прихода радиосигнала (задача пеленгации), определяют разность времени прихода сигнала пеленгуемого объекта на две разнесенные антенны. Во всех этих примерах производится определение скалярного произведения сигнала $x(t)$ *с собственной копией*, скользящей по аргументу, а функцию $R_{xx}(\tau)$ называют *автокорреляционной функцией – АКФ*:

$$R_{xx}(\tau) = \frac{1}{T_т} \int_0^T x(t) * x(t+\tau) dt .$$

Автокорреляция позволяет оценить среднестатистическую зависимость текущих отсчетов сигнала от своих предыдущих и последующих значений (так называемый радиус корреляции значений сигнала), а также выявить в сигнале наличие периодически повторяющихся элементов.

В корреляционном анализе часто вид корреляционной функции не имеет значения: интерес представляет один из ее параметров. В приведенных примерах хотя и вычисляется вся корреляционная функция, однако в первой задаче анализируется только величина максимального значения, а во второй –

положение ее максимума на временной оси. Для анализа длительности лечебного воздействия прививок в медицине анализируют ширину корреляционной функции и т. д.

КФ, вычисленная по центрированному значению сигнала $x(t)$, представляет собой *ковариационную* функцию сигнала:

$$C_{xy}(\tau)= \frac{1}{T}\int_0^T (x(t)- m_x)^*(y(t+\tau)- m_y)dt .$$

Порядок проведения работы

1. Рассмотрим задачу измерения высоты полета самолета. Проще всего послать вертикально вниз короткий радиоимпульс и измерить при помощи корреляционной функции задержку импульса, отраженного от земли. Создайте такие импульсы из длинной нулевой последовательности `zeros(n,m)` присвоением группе выбранных отсчетов некоторого значения (амплитуды импульса):

```
X=zeros (1,1000);
Y=zeros (1,1000);
X(1:100)=5;              % излучаемый импульс с амплитудой 5 и
                         % длительностью 100 отсчетов
Y(301:400)=1;    % принятый импульс запоздал на 300 отсчетов
```

Постройте $R_{xx}(\tau)$, можно воспользоваться командой `conv(X,Y)`, которая для выполнения свертки (скольжения по τ) предварительно увеличит вдвое длину вектора *Y*.

У такого способа измерения высоты полета есть два серьезных недостатка: малая помехозащищенность (можно поймать чужой импульс) и пологая форма $R_{xx}(\tau)$, не позволяющая гарантировать точность определения задержки при наличии шума. Проверьте, как изменится вид $R_{xx}(\tau)$, если посылать более сложный сигнал из двух или трех импульсов. Объясните, чем вызвано появление боковых лепестков и их временное положение.

2. Создайте еще более сложный сигнал, в котором нет периодических повторов импульсов. Его можно создать из случайной последовательности `rand(1,m)` логической операцией сравнения с порогом $x > P$. Сымитируйте задержку распространения сигнала и получите графики $R_{xx}(\tau)$ и $C_{xy}(\tau)$. Объясните, чем вызвано их отличие.

3. Наложите на принятый (задержанный) сигнал шум с соотношением *SNR* 3ё 10 дБ. Определите минимальное соотношение *SNR*, при котором еще возможно измерение задержки.

4. В качестве посылаемого сигнала возьмите отрезок синусоиды (5–10 периодов). Наложите шум и постройте $R_{xx}(\tau)$. Как теперь определить величину задержки?

ЛАБОРАТОРНАЯ РАБОТА 3. СПЕКТР. РЯД ФУРЬЕ

Цель работы:

– знакомство со спектральным представлением периодических и случайных процессов;

– изучение взаимосвязи преобразований сигналов во временной и частотной областях;

– оценка дефектов дискретного преобразования Фурье и методы их подавления.

Теоретические положения. Поставим задачу: представить некоторый сложный периодический процесс $x(t)$, заданный на интервале $[0,T]$, в виде суммы простых периодических функций вида $\psi_\omega(t) = A\sin(\omega t + \varphi) = a\cos(\omega t) + b\sin(\omega t)$.

Периодичность $x(t)$ гарантирует, что в таком разложении будут присутствовать только гармоники кратных частот $\omega = n\omega_0$, n – целое, $\omega_0 = \frac{2\pi}{T}$. При $n = 0$ получим постоянную составляющую, обычно записываемую в форме $\frac{a_0}{2}$, а все разложение будет иметь следующий вид:

$$x(t) = \frac{a_0}{2} + (a_1\cos\omega_0 t + b_1\sin\omega_0 t) + (a_2\cos 2\omega_0 t + b_2\sin 2\omega_0 t) + \text{K} =$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n\cos n\omega_0 t + b_n\sin n\omega_0 t.$$

Данное разложение называется *рядом Фурье.*

Коэффициенты этого ряда рассчитываются по формулам:

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} x(t)dt;$$

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} x(t)\cos n\omega_0 dt;$$

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} x(t)\sin n\omega_0 dt.$$

Если воспользоваться формулой Эйлера $\exp(j\varphi) = \cos\varphi + j\sin\varphi$, получим более простую запись ряда Фурье в комплексной форме:

$$x(t) = \sum_{n=-\infty}^{\infty} F_n e^{jn\omega_0 t},$$

где $F_n = \frac{1}{T}\int_{-T/2}^{T/2} x(t)e^{-jn\omega_0 t}dt$ – коэффициенты ряда Фурье, или спектр процесса $x(t)$.

Совокупность функций $e^{-jn\omega_0 t}$ называется *базисом Фурье.*

Комплексные коэффициенты F_n обычно представляют в форме $F_n = S_n e^{j\varphi} = U_n + jV_n$, где $S_n = \sqrt{U_n^2 + V_n^2}$ – амплитудный спектр; $\varphi_n = \operatorname{arctg}\frac{V_n}{U_n}$ – фазовый спектр; U_n – реальная составляющая спектра; V_n – мнимая составляющая спектра.

Для анализа непериодических процессов ($T \to \infty$) служит преобразование Фурье:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt.$$

Широкое распространение спектрального представления сигналов объясняется следующими причинами:

– гармонические функции – единственные, не меняющие своей формы при прохождении через линейную систему: может измениться только их амплитуда и фаза, но не форма, а, значит, не частота;

– простота синтеза гармонического колебания – для этого достаточно иметь колебательный контур или любую другую резонансную систему. Разложить в спектр Фурье оптический сигнал может любая двояковыпуклая линза, радиосигналы в эфире тоже представлены электромагнитными волнами – гармониками ряда Фурье;

– графическое представление спектральных коэффициентов на частотной оси – спектра сигнала – позволяет получить наглядную картину распределения в сигнале низких и высоких частот;

– частотные характеристики используются не только для анализа сигналов, но и для анализа свойств динамических систем.

Чтобы построить спектр с помощью ДПФ (БПФ), надо определить следующие параметры:

– количество спектральных составляющих N;

– шаг между соседними частотами – разрешение по частоте Δf;

– частоту дискретизации f_S;

– минимальную (нижнюю) частоту спектра $f_{\text{нижн}}$;

– верхнюю частоту $f_{\text{верх}}$;

– временной интервал анализа T.

На самом деле эти параметры жестко связаны друг с другом, и для однозначного построения спектра достаточно задать всего две величины.

Как правило, анализ начинается с выбора временной базы анализа T и частоты дискретизации f_S. При этом оказываются определенными и количество отсчетов сигнала $N = T \text{г } f_S$, и минимальная частота спектра: $f_{\text{нижн}} = 1/T$. А поскольку количество спектральных коэффициентов равно количеству отсчетов сигнала N, оказываются определенными и верхняя частота преобразования $f_{\text{верх}} = f_{\text{нижн}} \text{г } N = f_S$, и шаг между соседними частотами $\Delta f = f_{\text{нижн}}$.

Спектр сигнала (сейчас мы рассматриваем только вещественные функции) будет состоять из двух зеркально отраженных картин. Для комплексного спектра этого не происходит, однако в спектре появляются отрицательные частоты.

Теорема Котельникова – фундаментальное утверждение в области цифровой обработки сигналов, связывающее непрерывные и дискретные сигналы.

Теорема. Любой сигнал $s(t)$, спектр которого не содержит составляющих с частотами выше некоторого значения $\omega_в = 2\pi f_в$, может быть без потерь информации представлен своими дискретными отсчетами $\{s(kT)\}$, взятыми с интервалом T, удовлетворяющим следующему неравенству: $T < \dfrac{1}{2 f_в} = \dfrac{\pi}{\omega_в}$.

Преобразование Фурье в дискретной форме (ДПФ) имеет и другие недостатки. Главный из них – растекание спектра. Растекание спектра (англ. *spectrum leakage*) – эффект, возникающий вследствие финитности анализируемого сигнала (фактически бесконечный сигнал взвешивается финитным прямоугольным окном). Для подавления этого эффекта используют взвешивание сигнала специальными оконными функциями (окна Чебышева, Ханна, Парзена и т. д.).

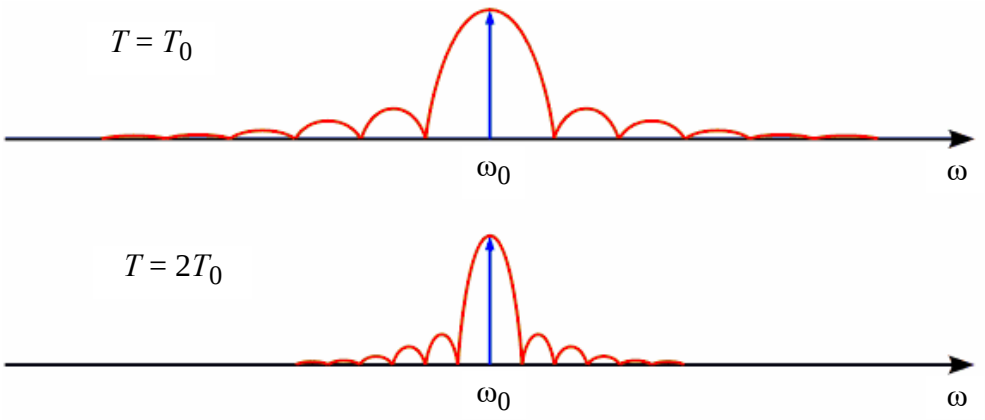

Рис. 3.5. Растекание спектра

При спектральном преобразовании сигнала могут использоваться и другие (не гармонические) функции. В этом случае говорят об *обобщенном преобразовании Фурье:* $F_n = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\psi_n(t)dt$.

Главные требования к функциям ψ_n – их *ортогональность и полнота*.

Два вектора X_i, X_j называют *взаимноортогональными,* если их скалярное произведение Y равно нулю: $X_i = [x_i(1), x_i(2), \ldots, x_i(N)]$; $X_j = [x_j(1), x_j(2), \ldots, x_j(N)]$; $Y = \sum_{n=1}^{N} X_i(n) * X_j(n) = 0$.

Если при этом норма каждого вектора равна 1, $\|Y\| = \sqrt{\sum_{n=1}^{N} X_i(n)X^*_i(n)} = 1$, векторы называют *ортонормальными.*

Для любого вектора размерности (длины) N можно найти N и только N ортогональных векторов. Такой набор векторов называют *полной ортонормальной системой,* или *базисом* N-мерного линейного пространства.

Рассмотрим для примера два базиса в пространстве $N = 2^3$.

Первый базис задается матрицей:

$$W_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

или в графической форме (каждая функция – одна из строк матрицы W_3) (рис. 3.5):

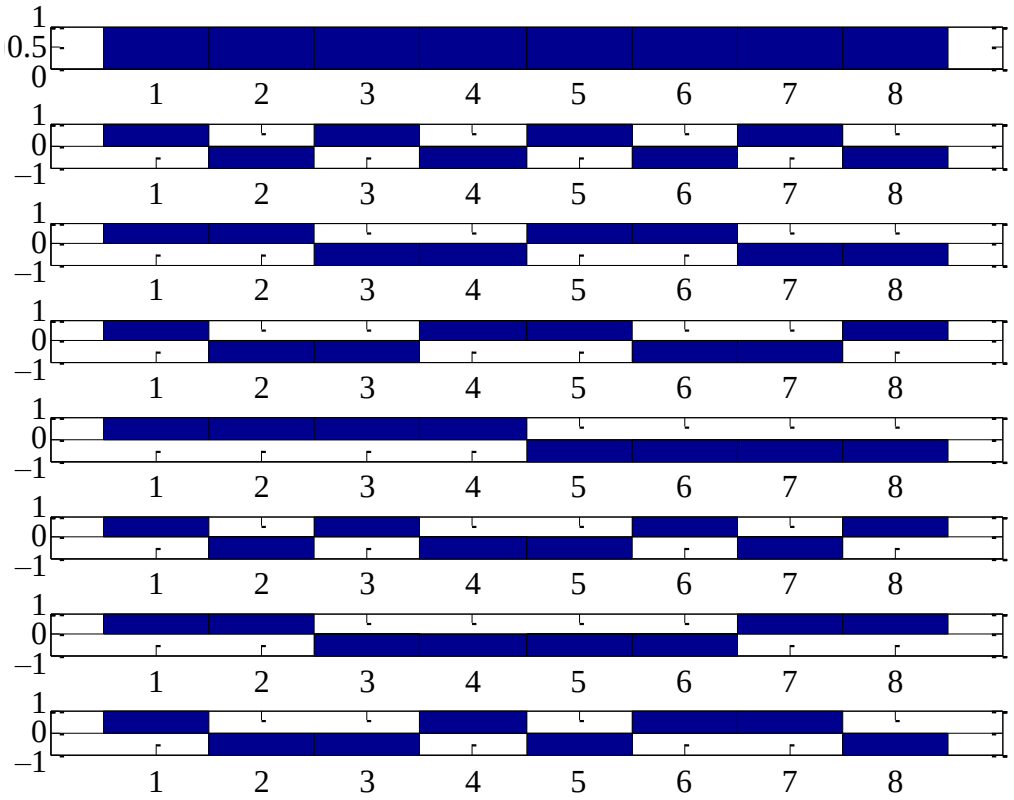

Рис. 3.5. Первый базис

Второй базис зададим матрицей:

```
A =
    0.1250   0.5026   0.4275   0.8550   0.2138   0.6917
0.4275   0.8550
    0.2138  -0.2939   0.7310  -0.5000   0.3655  -0.4045
0.7310  -0.5000
    0.2513   0.4755  -0.2939  -0.8090   0.4297   0.6545
-0.2939  -0.8090
    0.3458  -0.3455  -0.4045   0.5878   0.5914  -0.4755
-0.4045   0.5878
    0.5026   0.8090   0.8090   1.0000  -0.2939  -0.5878
-0.8090  -1.0000
    0.5026  -0.8090   0.8090  -1.0000  -0.2939   0.5878
-0.8090   1.0000
    0.6917   0.8090  -0.5878  -1.0000  -0.4045  -0.5878
0.5878   1.0000
    0.6917  -0.8090  -0.5878   1.0000  -0.4045   0.5878
0.5878  -1.0000
```

или графически (рис. 3.6):

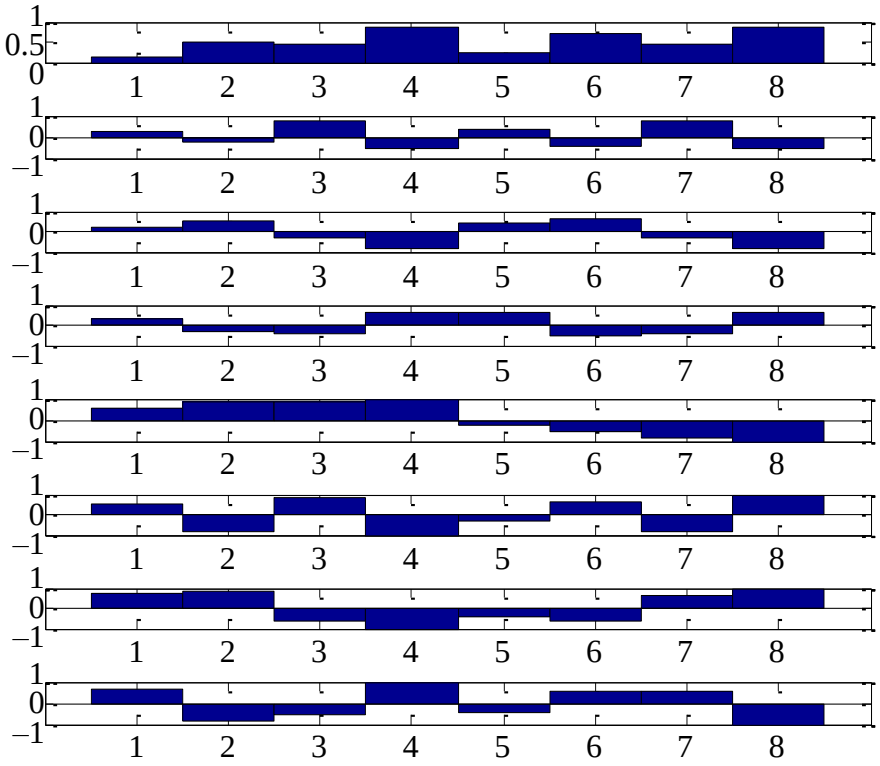

Рис. 3.6. Второй базис

Сравним эти базисы. Прежде всего, бросается в глаза равенство ±1 всех элементов матрицы W_3. Это очень удобное свойство базисных функций: при умножении на такую функцию временные затраты минимальны. Во втором базисе это свойство потеряно. Первая строка базиса обычно служит для выделения постоянной составляющей из исследуемого сигнала, и в классических базисах все ее элементы равны единице. При этом все остальные функции не реагируют на постоянную составляющую вследствие симметричности их относительно нуля. Этого во втором базисе тоже нет. Поэтому первый базис используется очень широко и носит название базиса Уолша, а второй базис кроме вас никто не видел.

Приведем еще один весьма широко используемый базис – базис Хаара (рис. 3.7):

$$H_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

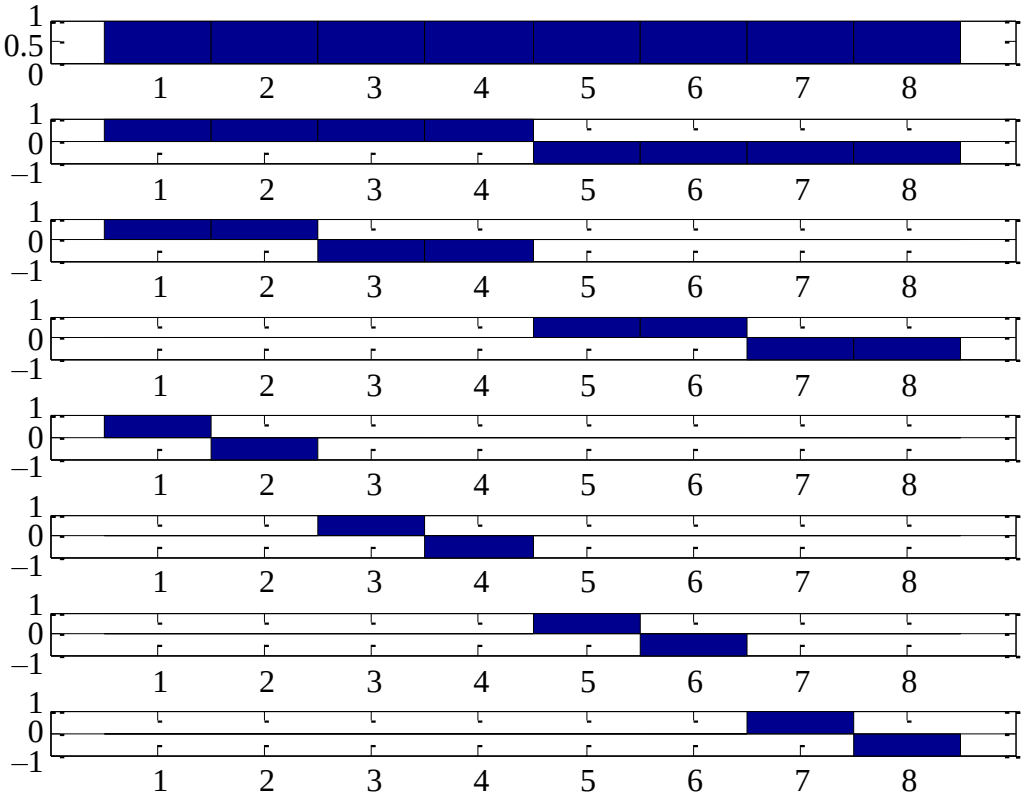

Рис. 3.7. Базис Хаара

Его главное достоинство – большое количество нулевых элементов. Это не только ускоряет вычисления спектра, но и обеспечивает сжатое представление сигналов с резкими перепадами. Это объясняется тем, что каждая функция (за исключением первых двух) выделяет только некоторую локальную область существования сигнала.

Для упрощения здесь базис Хаара приведен в ненормированной форме: чем у́же импульс, тем меньше его энергия.

Порядок проведения работы

Используемые функции MATLAB:

– `fft (x)` – преобразование Фурье;

– `ifft(y)` – обратное преобразование Фурье;

– `abs(fft(x))` – модуль спектра;

– `angle(y), phase(y)` – фаза спектра;

– `real(y), imag(y)` – реальная и мнимая составляющая;

– `kron(A,B)` – кронекеровское произведение матрицы B на A, при котором каждый элемент матрицы A умножается на всю матрицу B.

1. Создайте два сигнала `x1=cos(2*pi*f1*t)`; `x2=4*cos(2*pi*f2*t)`. Частоты `f1` и `f2` задаются преподавателем. Временной интервал – базу анализа – выберите так, чтобы более низкочастотный сигнал имел на нем 3–5

периодов. Задайте частоту дискретизации так, чтобы на периоде сигнала высокой частоты укладывалось 4–10 отсчетов. Получите модуль спектра сигнала, постройте его график. Объясните, чем определяется номер гармоники в спектре.

2. Создайте еще два сигнала: `x3=x1+x2;  x4=x1.*x2` и постройте их спектры. Объясните полученный результат.

3. На временном интервале $N = 2^7$ отсчетов создайте δ-импульс и получите его спектр (модуль и фазу). Как изменится спектр, если сдвинуть δ-импульс?

4. В цикле `for` последовательно увеличивайте ширину импульса, наблюдая соответствующие изменения его спектра. Для произвольной ширины импульса рассчитайте спектр вручную. Сделайте выводы.

5. На том же временном интервале создайте периодический прямоугольный сигнал со скважностью 2 (меандр) и количеством периодов, кратным двум. Постройте его спектр. Рассчитайте спектр вручную.

6. Операцией `kron(A,A)`, где $A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, создайте базис Уолша для $N = 2^7$. Постройте спектр того же сигнала. Сравните спектр Фурье и Уолша.

7. Определите форму и ширину частотной характеристики двух соседних каналов анализатора Фурье. Это можно сделать в цикле `for`, изменяя частоту анализируемого сигнала с достаточно малым шагом (0.1 – 0.2) и выделяя из спектра только отчет, принадлежащий выбранному каналу. Оцените, как меняется спектр моногармонического сигнала при его смещении по частотной оси. Для улучшения качества анализа используйте вместо прямоугольного временного окна, обрезающего сигнал, окно Хеннинга (`hanning`). Сравните результаты.

ЛАБОРАТОРНАЯ РАБОТА 4. ДИСКРЕТНЫЕ СИСТЕМЫ

Цель работы:

– знакомство с дискретными системами и операцией дискретной свертки;

– расчет импульсной и частотных характеристик;

– изучение визуальных средств проектирования цифровых фильтров.

Теоретические положения. Дискретный фильтр – это произвольная система обработки дискретного сигнала $T[x(n)] = z(n)$, обладающая свойствами:

– линейности: $T[\alpha x(n) + \beta y(n)] = \alpha T[x(n)] + \beta T[z(n)]$;

– стационарности: $T[x(n + k)] = z(n + k)$.

Существуют также фильтры с переменными параметрами, не обладающие свойством стационарности, – адаптивные фильтры.

В общем случае дискретный фильтр суммирует (с весовыми коэффициентами) некоторое количество входных отчетов (включая последний) и некоторое количество предыдущих выходных отсчетов:

$$y(k) = b_0 x(k) + b_1 x(k-1) + \text{K} + b_P x(k-P) - \text{K}$$
$$- a_1 y(k-1) - a_2 y(k-2) - \text{K} - a_Q y(k-Q).$$

Выделяют фильтры с *конечной импульсной характеристикой* (*КИХ-фильтры*) и с *бесконечной импульсной характеристикой* (*БИХ-фильтры*). Для КИХ-фильтров выполняется условие $a_j = 0 \,"j$. Таким образом, главным отличием БИХ-фильтров является *наличие обратной связи и потенциальная неустойчивость*.

Уравнение БИХ-фильтра:

$$y(k) = \sum_{i=0}^{P} b_i x(k-i) - \sum_{j=1}^{Q} a_j y(k-j),$$

где a_j и b_i – вещественные коэффициенты.

Уравнение КИХ-фильтра:

$$y(k) = \sum_{i=0}^{P} b_i x(k-i),$$

где b_i – вещественные коэффициенты.

Пусть $x(k) = \delta(k)$. Тогда:

Импульсная функция БИХ-фильтра записывается следующим образом:

$$h(k) = \sum_{i=0}^{P} b_i \delta(k-i) - \sum_{j=1}^{Q} a_j h(k-j).$$

Z-преобразование импульсной функции дает передаточную функцию БИХ-фильтра:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{P} b_i z^{-i}}{1 + \sum_{j=1}^{Q} a_j z^{-j}}.$$

Импульсная функция КИХ-фильтра записывается следующим образом:

$$h(k) = \sum_{i=0}^{P} b_i \delta(k-i).$$

Z-преобразование импульсной функции дает передаточную функцию КИХ-фильтра:

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{i=0}^{P} b_i z^{-i}.$$

Комплексный коэффициент передачи определяется следующим образом:

$$\dot{K}(\omega) = H(e^{j\omega T}).$$

Амплитудно-частотная характеристика (АЧХ) фильтра – модуль комплексного коэффициента передачи: $A = \left|\dot{K}(\omega)\right|$.

Фазо-частотная характеристика (ФЧХ) фильтра – фаза комплексного коэффициента передачи: $F = \arg\left[\dot{K}(\omega)\right]$.

По виду АЧХ выделяют следующие фильтры (рис. 3.8):

– нижних частот (ФНЧ);

– верхних частот (ФВЧ) частот;

– полосовые (ПФ);

– режекторные (РФ).

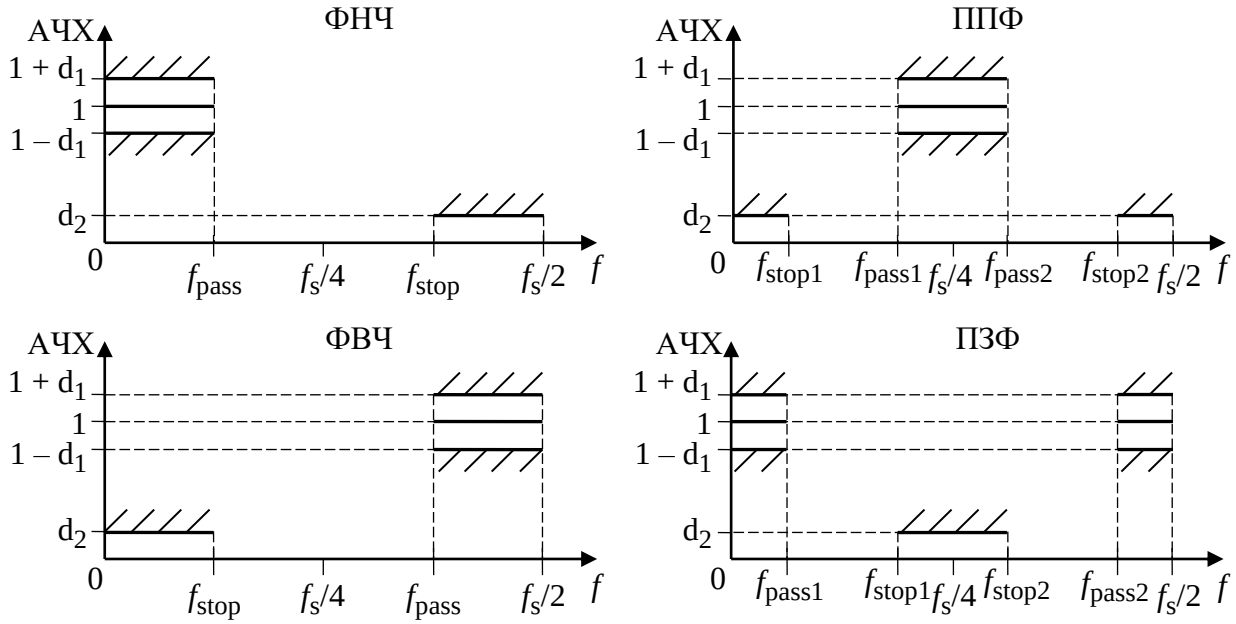

Рис. 3.8. Классификация фильтров по виду АЧХ

На рис. 3.8 введены следующие обозначения:

f_s – частота дискретизации;

f_{pass}, f_{pass1}, f_{pass2} – граничные частоты полосы пропускания (ПП);

f_{stop}, f_{stop1}, f_{stop2} – граничные частоты полосы задерживания (ПП);

d_1 – неравномерность в полосе пропускания;

d_2 – неравномерность в полосе задерживания.

КИХ-фильтр обладает рядом полезных свойств, из-за которых он иногда более предпочтителен в использовании, чем БИХ-фильтр. Вот некоторые из них:

– КИХ-фильтры устойчивы;

– КИХ-фильтры при реализации не требуют наличия обратной связи;

– фаза КИХ-фильтров может быть сделана линейной.

Различают два вида реализации цифрового фильтра: *аппаратный* и *программный*.

Аппаратные цифровые фильтры реализуются на элементах интегральных схем, ПЛИС, тогда как программные реализуются с помощью программ, выполняемых процессором или микроконтроллером. Преимуществом программных методов перед аппаратными является лёгкость воплощения, настройки и изменений, а также то, что в себестоимость такого фильтра входит только труд программиста. Недостаток – низкая скорость, зависящая от быстродействия процессора, а также трудная реализуемость цифровых фильтров высокого порядка.

Порядок проведения работы

Используемые функции MATLAB:

– `fft (x)` – преобразование Фурье;

– `ifft(y)` – обратное преобразование Фурье;

– `abs(fft(x))` – модуль спектра;

– `angle(y), phase(y)` – фаза спектра;

– `filter` – фильтрация;

– `conv` – свертка;

– `fdatool` – синтез цифровых фильтров.

1. Рассчитать реакцию КИХ-фильтра 3-го порядка, заданного разностным уравнением:

$$y(n) = 0.2x(n) + 0.3x(n-1) + 0.5x(n-2) + 0.7x(n-3),$$

$$n = 0\text{j}\ 64;$$

$$x(n) = 0.1\sin(\omega Tn),\ \omega T = 0.7\text{рад}.$$

2. Рассчитать реакцию БИХ-фильтра 3-го порядка, заданного разностным уравнением:

$$y(n) = x(n) + x(n-1) + x(n-2) + x(n-3) + 0.3y(n-1) -$$
$$- 0.25y(n-2) - 0.4y(n-3),$$
$$n = 0 \text{j} \; 64;$$
$$x(n) = 0.1\sin(\omega Tn), \; \omega T = 0.7 \text{рад}.$$

3. Вычислить импульсную характеристику БИХ-фильтра, заданного в п. 2.

4. Вычислить импульсную характеристику БИХ-фильтра, заданного в п. 2, используя функцию `impz()`, при $N = 80$, $Fs = 5000$ Гц.

5. Используя утилиту `fdatool` синтезировать полосовой КИХ-фильтр со следующими параметрами (нормированная полоса частот):

– полоса пропускания 0,35–0,7;

– полоса задерживания 0–0,15 0,8–1;

– неравномерности по 0,025 в ПП и ПЗ.

Получить ИХ, АЧХ и ФЧХ фильтра.

6. По заданному набору коэффициентов КИХ-фильтра получить его АЧХ, ФЧХ и ИХ:

```
0
0,026477
0
0
0
-0,044116
0
0
0
0,093436
0
0
0
-0,31394
0
0,5
```

```
0
-0,31394
0
0
0
0,093436
0
0
0
-0,044116
0
0
0
0,026477
0
```

Сравните результат с п. 5, сделайте выводы и синтезируйте полосовой КИХ-фильтр с полученными частотными параметрами.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Boyd S., Vandenberghe L. Convex Optimization / Cambridge University Press, 2004. 243 с.

Ветчинкин А. С., Зуев В. А., Лукомский Ю. А. Теория оптимального управления. СПб.: Изд-во СПб ГЭТУ «ЛЭТИ», 2011. 96 с.

Коршунов Ю. М. Математические основы кибернетики: учеб. пособие. М.: Энергия, 1987. 423 с.

Лазарев Ю. Моделирование процессов и систем в MATLAB: учебный курс. СПб.: БХВ-Петербург, 2005. 512 с.

Малышенко А. М. Математические основы теории систем: учеб. пособие для втузов. Томск: Изд-во ТПУ, 2004. 334 с.

Nocedal J., Wright J. Numerical optimization / Springer, 2006. 324 с.

Основы цифровой обработки сигналов: курс лекций / А. И. Солонина, Д. А. Улахович, С. М. Арбузов, Е. Б. Соловьева. СПб.: БХВ-Петербург, 2005. 768 с.

Вознесенский Александр Сергеевич
Гульванский Вячеслав Викторович
Канатов Иван Иванович
Каплун Дмитрий Ильич
Романов Сергей Алексеевич

Математические основы теории систем

Учебно-методическое пособие

Компьютерная верстка О. А. Маткеримова

Редактор М. Б. Шишкова

Подписано в печать ??.??.??. Формат 60×84 1/16.
Бумага офсетная. Печать цифровая. Печ. л. 4,75.
Гарнитура «Times New Roman». Тираж 50 экз. Заказ .

Издательство СПбГЭТУ «ЛЭТИ»
197376, С.-Петербург, ул. Проф. Попова, 5