Ожикенов К.А.

ОСНОВЫ ПРОГРАММИРОВАНИЯ В СРЕДЕ MATLAB

Учебное пособие

Алматы, 2012

УДК 004.43 (075.8)
ББК 32.973-018.1 я73
О - 45

Рецензенты:

Утепбергенов Е.Т. Доктор технических наук, профессор
Джомартова Ш.А. Доктор технических наук, доцент

О - 45 **Ожикенов К.А.**

Основы программирования в среде MATLAB: Учебное пособие. – Алматы: 2012. – 148 с.

ISBN 9965-885-92-3

В пособии изложены первоначальные сведения о языке программирования интегрированного математического пакета MATLAB. Рассмотрена методика решения задач линейной алгебры, линейных и нелинейных дифференциальных уравнений. Изложение сопровождается большим количеством иллюстративных примеров.

Учебное пособие адресовано студентам вузов, обучающихся по техническим специальностям, а также может быть полезным и для всех желающих изучить язык программирования MATLAB.

УДК 004.43 (075.8)
ББК 32.973-018.1 я73

ISBN 9965-885-91-5

ВВЕДЕНИЕ

В связи с повсеместным развитием современных компьютерных технологий, существенно изменились и подходы решения математических задач разной сложности. Для решения сложных задач можно применять ряд математических пакетов, одним из которых и является MATLAB (*MATrix LABoratory* - матричная лаборатория). Пакет MATLAB широко используется во всем мире при решении задач, связанных с матричными вычислениями. Операции и команды в MATLAB достаточно естественны и аналогичны математической записи формул на бумаге. MATLAB создавался как пакет программ, реализующих наиболее эффективные вычислительные алгоритмы линейной алгебры. Он организован таким образом, чтобы пользователь имел возможность применять при работе обычный математический язык.

В настоящее время пакет MATLAB представляет собой развитую интегральную программную среду, включающую собственный язык программирования. Он дает пользователю возможность быстро выполнять различные операции над векторами и матрицами, такие как умножение и обращение матриц, вычисление определителей, нахождение собственных чисел и векторов. Кроме того, в MATLAB входят операции вычисления обычных функций (алгебраических, тригонометрических, логических), решения алгебраических и дифференциальных уравнений, операции построения графиков и ряд других.

MATLAB является языком высокого уровня. По отдельным его командам можно выполнять такие сложные операции, как нахождение корней полиномов, решение линейных и нелинейных алгебраических уравнений, моделирование линейных динамических систем. Указанные операции являются элементарными функциями MATLAB.

Появление данного учебного пособия вызвано с потребностью в компактном изложении первоначальных сведений о языке программирования MATLAB, и здесь автор постарался изложить

минимальный объем сведений о функциях пакетов. Изложение снабжено иллюстрирующими примерами.

Автор старался подать изучаемый материал таким образом, чтобы настоящее пособие можно было применить в качестве самоучителя. При этом старался отметить особенности языка, функций библиотеки стандартных программ и использованных в ней вычислительных методов, не очевидные для неискушенного читателя. От читателя пособия, пытающегося реализовать полученные сведения в своей практической деятельности, не требуется досконального изучения каких-либо языков программирования и численных методов математики. Тем не менее, автор предполагает, что читатель достаточно будет иметь общие знания, требующихся при работе на компьютере и, естественно, знаний по той предметной области, в которой он работает.

Пособие ориентировано на использование его студентами технических специальностей вузов, на начальных стадиях своей научно-практической деятельности.

1 СРЕДА MATLAB. РАБОТА В КОМАНДНОМ РЕЖИМЕ

1.1 Среда MATLAB

Командное окно системы MATLAB представлено на рис. 1.1. Оно содержит строку меню, панель инструментов, рабочую область и строку состояния. В рабочей области вводятся команды и выводятся результаты вычислений. Команды вводятся в строке ввода, отмеченной знаком приглашения (>>). В строке состояния, расположенной в нижней части окна, отображаются сообщения системы.

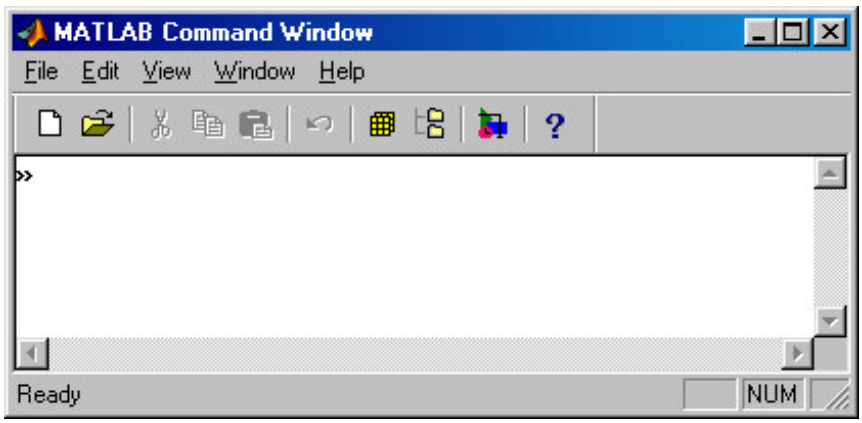


Рисунок 1.1 – Командное окно системы MATLAB

1.2 Работа в командном режиме

В MATLAB вычисления можно выполнять в программном режиме или в командном режиме. В программном режиме вызывается специальная программа, составленная на языке MATLAB. Эта программа осуществляет ввод исходных данных, выполнение расчетов, вывод результатов.

В командном режиме команды и вычисляемые выражения вводятся в строку ввода командного окна. Результаты расчетов выводятся в командное окно.

Например, после ввода в командную строку выражения *4+5* и нажатия клавиши Enter, на экране появится результат:

```
» 4+5
ans =
    9
»
```

Значение рассчитанного выражения присваивается специальной переменной *ans* и выводится в отдельной строке. В переменной *ans* сохраняется результат последнего вычисленного выражения, которое не было присвоено никакой другой переменной. Это значение можно использовать для выполнения последующих расчетов:

```
» 4+5
ans =
    9
» ans/3
ans =
    3
»
```

Если после введенного в командную строку выражения поставить символ «;» (точка с запятой), результат расчета выведен не будет. Например (в столбце справа введены те же команды, но без символа «;» в конце):

```
» 4+5;
» ans/3;
» ans
ans =
    3
»
```

```
» 4+5
ans =
    9
» ans/3
ans =
    3
» ans
ans =
    3
»
```

Символ «;» можно также использовать для разделения выражений, вводимых в командной строке:

```
» g=8; h=5;k=g>h;
» g
g =
    8
```

```
» h
h =
     5
» k
k =
     1
»
```

Все команды и выражения, введенные в командную строку, сохраняются. Для их повторного ввода используются клавиши ↑ и ↓, которые обеспечивают просмотр команд в прямой или обратной последовательности.

Для очистки командного окна используется команда *clc*.

Если вводимое выражение не помещается в одной строке, для переноса используется знак многоточия «…»(три или более точек):

```
» 3+...
5-7+9/...
3
ans =
     4
»
```

Сеанс работы в MATLAB называется *сессией*. Сессия состоит из строк ввода, вывода, предупреждений и сообщений об ошибках. Сообщение об ошибке предваряется тремя вопросительными знаками и останавливает выполнение вычислений:

```
» 1h=5
??? 1
    |
Missing operator, comma, or semi-colon.

»
```

Предупреждение предваряется словом *Warning*. Предупреждение не останавливает выполнение расчетов, а только информирует пользователя о том, что найденная ошибка может повлиять на результат:

```
» 1/0
Warning: Divide by zero.
ans =
  Inf
»
```

1.3 Математические выражения

В математическое выражение могут входить числовые константы, имена переменных, функции, знаки арифметических операторов, круглые скобки.

1.3.1 Действительные числа

При записи чисел для отделения целой части числа от дробной используется символ «.» (точка). Для отделения мантиссы от порядка применяется символ *e*. Например:

```
7    -2    5.8    2.456e-5    9.914e12
```

В записи числа не должно быть пробелов.

1.3.2 Комплексные числа

В MATLAB можно выполнять расчеты с действительными и комплексными числами. Для обозначения мнимой единицы ($\sqrt{-1}$) используются переменные с именами *i* или *j*.

При вводе комплексного числа в алгебраической форме знак умножения между мнимой частью, представленной числовой константой, и мнимой единицей можно не вводить:

```
» 7+5i
ans =
  7.0000 + 5.0000i
```

Если мнимая часть представлена переменной, при вводе комплексного числа следует ввести знак умножения между мнимой частью и мнимой единицей:

```
» 7
ans =
    7
» 5+ans*i
ans =
  5.0000 + 7.0000i
»
```

Функции для работы с комплексными числами приведены в п.1.3.8.

1.3.3 Переменные

Переменные могут быть числовыми, символьными, матричными или векторными.

Правила формирования имен переменных:

1. Имя переменной может состоять из любого количества символов, но *запоминаются и идентифицируются* только первые 31.

2. Имя должно быть уникальным (не должно совпадать с другими именами переменных, функций).

3. Имя может содержать буквы, цифры и символ подчеркивания.

4. Имя должно начинаться с буквы.

5. В именах переменных заглавные и строчные буквы различаются.

Для проверки допустимости использования имени переменной существует специальная функция, вызов которой записывается так

isvarname имя переменной

Эта функция возвращает значение *1*, если имя переменной допустимо и *0* – в противном случае. Например:

```
» isvarname alfa
ans =
    1
» isvarname 1a
```

```
ans =
    0
»
```

Для присваивания значений переменным используется операция присваивания

имя переменной = выражение

Например:

```
» a=5
a =
    5
»
```

Тип переменной определяется выражением, присвоенным переменной.

Чтобы вывести на экран значение переменной, следует ввести ее имя в командной строке и нажать клавишу Enter.

1.3.4 Системные переменные

В MATLAB используются следующие системные переменные:

i или *j*	мнимая единица
realmax	максимальное по модулю действительное число *(1.7977e+308)*
realmin	минимальное по модулю действительное число *(2.2251e-308)*
Inf	бесконечность (результат деления на ноль)
ans	результат последнего вычисленного выражения, которое не было присвоено никакой переменной
NaN	не число. Принято для обозначения неопределенного результата (получается в результате операций *0/0* или *Inf/Inf*)
pi	число π
eps	погрешность операций над числами с плавающей точкой (интервал между числом *1.0* и следующим ближайшим числом с плавающей точкой, равен 2^{-52})

Системные переменные задаются системой, но могут быть переопределены.

1.3.5 Арифметические операции

Над действительными и комплексными числами можно производить операции сложения, вычитания, умножения, деления и возведения в степень (они обозначаются соответственно знаками + - * / ^).

В соответствии с правилами приоритета действует следующий порядок выполнения арифметических операций:

1. Возведение в степень (^).
2. Умножение и деление (* /).
3. Сложение и вычитание (+ -).

Операции одинакового приоритета выполняются слева направо. Порядок выполнения операций можно изменить с помощью круглых скобок.

Для вычисления значения выражения его надо ввести в командной строке и нажать клавишу Enter

```
» 4^2-5/(2+3^2)
ans =
  15.5455
»
```

При выполнении расчетов с комплексными числами операнды нужно заключать в круглые скобки:

```
» 4+3i-(5-7i)/(3+6i)
ans =
  4.6000 + 4.1333i
»
```

Для вычисления комплексно-сопряженного числа используется апостроф (‘):

```
» d=7-4i
d =
  7.0000 - 4.0000i
```

```
» d'
ans =
   7.0000 + 4.0000i
»
```

1.3.6 Операции отношения

Для сравнения операндов применяются следующие операции отношения:

==	*равно*
~=	*не равно*
<	*меньше*
>	*больше*
<=	*меньше или равно*
>=	*больше или равно*

Результатом операции отношения является «истина» или «ложь». «Истина» в MATLAB обозначается *1*, «ложь» - *0*.

Приоритет операций отношения ниже, чем приоритет арифметических операций.

1.3.7 Логические операции

Логические операции и соответствующие им логические функции приведены в табл. 1.1

Таблица 1.1 – Логические операции и функции

Операция	Название	Функция
&	*логическое И*	*and*
|	*логическое ИЛИ*	*or*
~	*логическое НЕ*	*not*
	исключающее ИЛИ	*xor*

При выполнении логических операций истинными считаются отличные от нуля операнды, ложными – равные нулю. Работу логических операторов и функций иллюстрирует табл. 1.2.

Таблица 1.2 – Работа логических операторов и функций

Операнды		Логические операции и функции			
x	*y*	*x&y* *and(x,y)*	*x|y* *or(x,y)*	*~x* *not(x)*	*xor(x,y)*
1	*1*	*1*	*1*	*0*	*0*
1	*0*	*0*	*1*	*0*	*1*
0	*1*	*0*	*1*	*1*	*1*
0	*0*	*0*	*0*	*1*	*0*

Логические операции имеют более низкий приоритет, чем операции отношения. Приоритет логической операции *И* выше, чем операции *ИЛИ*.

1.3.8 Элементарные функции

Функции MATLAB делятся на встроенные и внешние. Перечень встроенных элементарных функций приведен в табл. 1.3. Внешние функции находятся в отдельных файлах с расширением *m*.

Таблица 1.3 – Встроенные элементарные функции MATLAB

Имя функции	Назначение	Имя функции	Назначение
Тригонометрические			
sin(x)	синус	*asin(x)*	арксинус
sinh(x)	гиперболический синус	*asinh(x)*	гиперболический арксинус
cos(x)	косинус	*acos(x)*	арккосинус
cosh(x)	гиперболический косинус	*acosh(x)*	гиперболический арккосинус
tan(x)	тангенс	*atan(x)*	арктангенс
tanh(x)	гиперболический тангенс	*atanh(x)*	гиперболический арктангенс
sec(x)	секанс	*asec(x)*	арксеканс
sech(x)	гиперболический секанс	*asech(x)*	гиперболический арксеканс
csc(x)	косеканс	*acsc(x)*	арккосеканс

Имя функции	Назначение	Имя функции	Назначение
csch(x)	гиперболический косеканс	*acsch(x)*	гиперболический арккосеканс
cot(x)	котангенс	*acot(x)*	арккотангенс
coth(x)	гиперболический котангенс	*acoth(x)*	гиперболический арккотангенс
atan2(y,x)	арктангенс *y/x* в диапазоне от $-\pi$ до π		
Экспоненциальные, показательные и логарифмические			
exp(x)	экспонента	*log(x)*	натуральный логарифм
log10(x)	десятичный логарифм	*nextpow2(x)*	степень, в которую надо возвести число *2*, чтобы получить ближайшее число, большее или равное *x*
sqrt(x)	квадратный корень		
pow2(x)	возведение числа 2 в степень x	*log2(x)*	логарифм по основанию *2*
Функции для работы с комплексными числами			
real(x)	действительная часть комплексного числа	*abs(x)*	модуль комплексного числа
imag(x)	мнимая часть комплексного числа	*angle(x)*	аргумент комплексного числа
complex(a,b)	создает комплексное число *a+jb*	*conj(x)*	возвращает комплексно сопряженое к *x* число
Функции округления и вычисления остатка от деления			
fix(x)	округление до ближайшего целого в сторону	*floor(x)*	округление до ближайшего целого в отрицательной

Имя функции	Назначение	Имя функции	Назначение
	нуля		бесконечности
ceil(x)	округление до ближайшего целого в положительной бесконечности	*round(x)*	округление до ближайшего целого
mod(x,y)	остаток от целочисленного деления с учетом знака	*rem(x,y)*	остаток от целочисленного деления по модулю
sign(x)	определение знака числа		

Для вызова функции необходимо после ее имени указать в круглых скобках список параметров через запятую:

```
» sin(pi/2)
ans =
    1
»
```

Аргументы тригонометрических функций должны задаваться в радианах, обратные тригонометрические функции возвращают результат также в радианах.

Элементарные функции предназначены для работы с действительными и комплексными аргументами.

Команда *help elfun* выводит список всех элементарных функций с их кратким описанием. Более подробную информацию о функции можно получить, применив команду *help* имя функции.

1.4 Рабочая область

Созданные в процессе работы переменные располагаются в рабочей области памяти (*workspace*). Содержимое рабочей области можно просмотреть, применив команды *who* и *whos*. Команда *who*

используется для вывода на экран списка имен используемых переменных, а команда *whos* – для вывода на экран информации о размерах массивов, объеме памяти, занимаемом переменными, типах переменных:

```
» a=5
a =
     5
» d=2
d =
     2
» who

Your variables are:

a        d

» whos
  Name      Size          Bytes  Class

  a         1x1               8  double array
  d         1x1               8  double array

Grand total is 2 elements using 16 bytes

»
```

Рабочую область можно также просмотреть в окне *Workspace Browser* (рис. 1.2), вызываемом командой *workspace*, введенной в строке ввода, кнопкой  (*Workspace Browser*) или командой *File/Show Workspace*. В окне *Workspace Browser* можно удалить переменную (выделив ее и нажав кнопку Delete), а если нажать кнопку Open, значение выделенной переменной можно просмотреть в окне Editor/Debugger (редактор/отладчик).

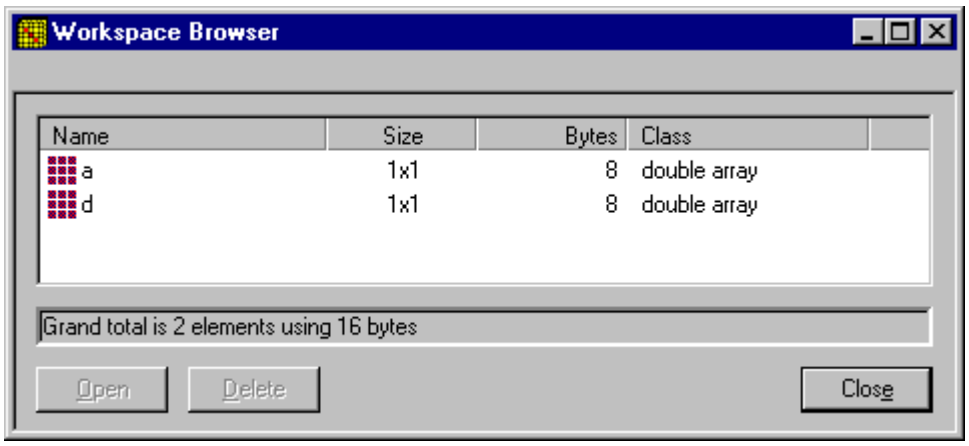


Рисунок 1.2 – Окно Workspace Browser

Для полной очистки рабочей области используется команда *clear*. Несколько переменных можно удалить выборочно командой

clear список имен переменных (через пробел)

Например:

» who

Your variables are:

j q r u y

» clear q
» who

Your variables are:

j r u y

» clear u j
» who

Your variables are:

r y

» clear

```
» who
»
```

По мере задания и удаления переменных рабочая область становится фрагментированной. Это может привести к снижению скорости выполнения расчетов. Дефрагментация рабочей области выполняется командой *pack*.

Рабочая область может быть сохранена командой *save*:

save имя файла	запись всех переменных рабочей области в файл с указанным именем и расширением *.mat*
save имя файла список имен переменных (через пробел)	запись указанных переменных рабочей области в файл

Загрузка ранее сохраненной рабочей области осуществляется командой *load:*
load имя файла или *load* имя файла список имен переменных (через пробел)

Переменные можно загружать выборочно из разных файлов. Второй вариант команды *load* используется для выборочной загрузки переменных.

Для сохранения рабочей области и ее загрузки могут быть также использованы команды *File/Save Workspace As* и *File/Load Workspace* соответственно.

Примеры применения команд записи и загрузки рабочей области:

```
» save fg
» what

MAT-files in the current directory C:\MATLABR11\work

fg
```

```
» clear
» who
» load fg
» who

Your variables are:

ans      h        u

»
```

В приведенном примере использована команда *what* для вывода имен файлов текущего каталога.

1.5 Ведение дневника

Дневник сессии отображает в текстовом файле все команды и результаты расчетов. Такой файл нельзя впоследствии запустить, его можно только просмотреть. Файл дневника по умолчанию имеет расширение m, но может быть задано любое другое расширение.

Для начала записи дневника применяется команда

diary имя файла

Команда *diary off* приостанавливает запись в файл, а команда *diary on* возобновляет запись дневника. Вывести на экран текст файла дневника можно командой *type* имя файла.

Следующий пример иллюстрирует использование дневника:

```
» diary d1
» k=5
k =
     5
» diary off
» h=k/2
h =
   2.5000
» diary on
» whos
```

```
  Name      Size         Bytes  Class

  h         1x1              8  double array
  k         1x1              8  double array

Grand total is 2 elements using 16 bytes

» diary off
» type d1

k=5
k =
     5
diary off
whos
  Name      Size         Bytes  Class

  h         1x1              8  double array
  k         1x1              8  double array

Grand total is 2 elements using 16 bytes

diary off

»
```

В примере выделено содержимое файла дневника. Этот файл можно просмотреть любым способом, применимым для текстовых файлов.

1.6 Настройка параметров системы

Команда *File/Preferences* позволяет вызвать диалоговое окно, в котором осуществляется настройка параметров. В окне имеется три вкладки (рис. 1.3). Назначение полей вкладки *General* приведено в табл. 1.4.

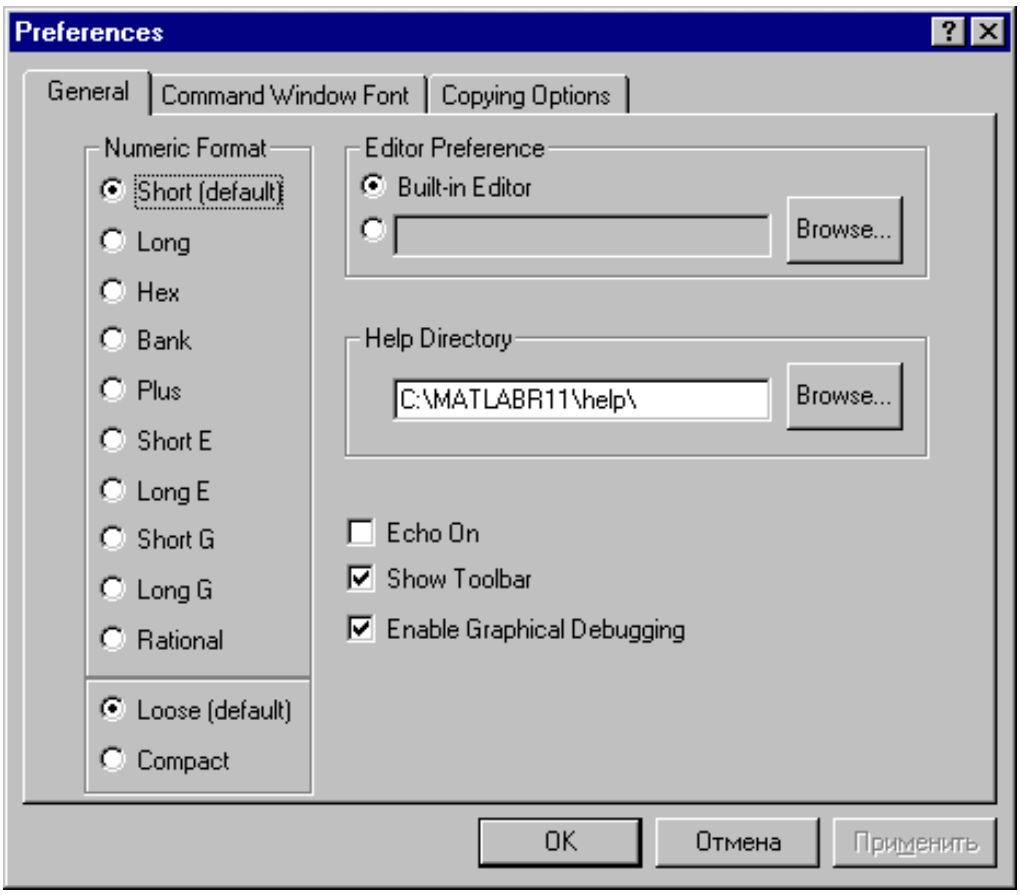


Рисунок 1.3 – Вкладка *General* окна *Preferences*

Таблица 1.4 – Назначения полей вкладки **General**

Поле	Назначение
Numeric Format	выбор формата представления чисел (табл. 1.5) и формы вывода результатов: *loose* – свободное (вывод результатов чередуется с пустыми строками) и compact (компактное, без вывода пустых строк).
Editor Preference	задается используемый текстовый редактор (по умолчанию - выстроенный)
Help Directory	каталог, в котором размещаются файлы справки
Echo On	выводить ли на экран команды исполняемого *Script*-файла сценария
Show Toolbar	отображение панели инструментов
Enable Graphical Debugging	поддержка отладки графики

Таблица 1.5 – Форматы представления чисел

Формат	Описание формата	Пример
short	краткое представление числа в формате с фиксированной точкой (5 знаков)	*1.2345*
long	длинное представление числа в формате с фиксированной точкой (15 знаков)	*0.00045634487671*
hex	представление числа в шестнадцатеричной системе счисления	*3f3de8325237f974*
bank	с двумя знаками после десятичной точки (используется для денежных единиц)	*1.23*
plus	выводится только знак числа (+ или -)	+
short e	краткое представление числа в формате с плавающей точкой (5 знаков мантиссы и 3 знака порядка)	*1.2345e+000* *4.5634e-004*
long e	длинное представление числа в формате с плавающей точкой (15 знаков мантиссы и 3 знака порядка)	*4.56344876713453e-004*
short g	MATLAB выбирает более удачный краткий формат представления числа: с фиксированной или плавающей точкой	*1.2345*
long g	MATLAB выбирает более удачный длинный формат представления числа: с фиксированной или плавающей точкой	
rational	представление в виде рациональной дроби	*10/3*

Формат представления чисел также можно задать командой

format имя формата

например,

format long

Чтобы вывести на экран информацию обо всех доступных форматах, надо использовать команду

help format

На вкладке *Command Window Font* задаются параметры шрифта командного окна, опции копирования могут быть выбраны на вкладке *Copying Options*.

1.7 Использование справочной системы

Справка по MATLAB может быть получена следующими способами:

- команда *help*;
- команда *lookfor*;
- меню *Help*;
- в формате *PDF*;
- на Web-сервере фирмы *The Math Works*.

Для получения справочной информации по использованию функции необходимо ввести команду

help имя функции

Следует иметь в виду, что хотя в тексте справки имена функций написаны заглавными буквами, в фактических именах функций используются строчные буквы.

Функции MATLAB сгруппированы по назначению. Команда **help** выводит на экран перечень групп функций. Для получения справки по определенной группе используется команда

help имя группы

Команда

lookfor ключевое слово или *lookfor* ‘ ключевое словосочетание ’ применяется для поиска М-функции по ключевому слову в первой строке комментария. На экран выводится найденная строка в случае наличия в ней искомого ключевого слова. Добавление опции – *all*

lookfor ключевое слово - *all* или *lookfor* ‘ ключевое словосочетание ’ – *all*

обеспечивает вывод полного текста комментария.

Назначение команд меню *Help* приведено в табл. 1.6.

Таблица 1.6 – Назначение команд меню *Help*

Команда	Назначение
Help Window	окно справки
Help Tips	окно справки для получения подсказки
Help Desk (HTML)	доступ к справочной системе, размещенной на жестком диске
Examples and Demos	демонстрация возможностей
About MATLAB	информация об установленной версии

Окно *Help Window* (рис. 1.4) можно вызвать командами *Help/Help Window, Help/Help Tips*, кнопкой ? панели инструментов или командой *helpwin*. Для перехода к требуемому разделу необходимо дважды щелкнуть мышью на его заголовке или применить команду *helpwin* имя раздела. Кнопки Back , Forward , Home позволяют перейти к предыдущему, следующему и начальному активированным окнам справки.

Кнопка Tips выводит окно подсказок. В этом режиме активен список *See also* с командами *lookfor, which, demo, general* и др.

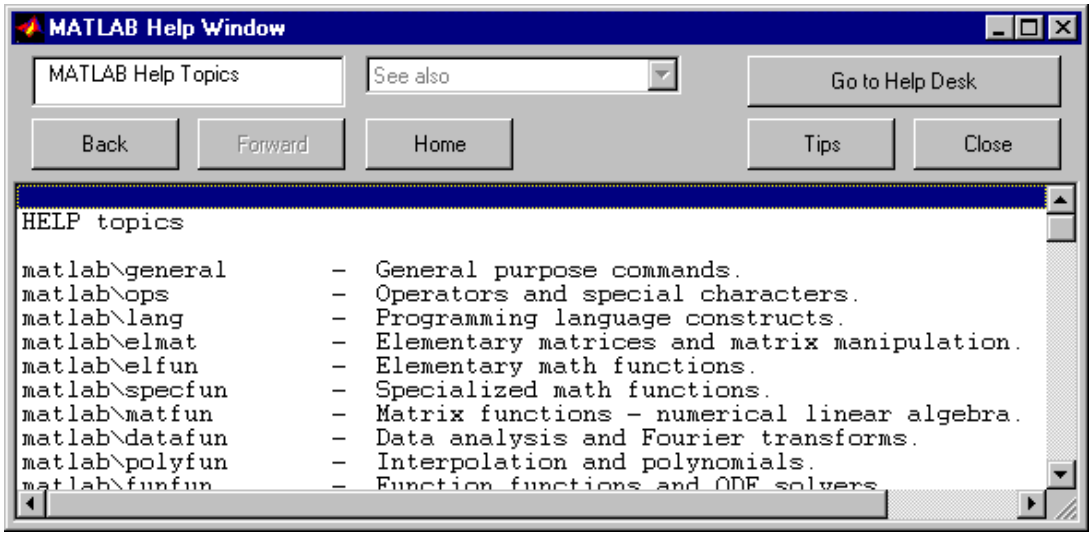


Рисунок 1.4 – Окно справки

Команда *Help/Help Desk (HTML)* обеспечивает доступ к справочной информации в формате HTML. В том же окне имеется кнопка для доступа к информации в формате PDF.

2 РАБОТА С МАССИВАМИ

Все данные в MATLAB интерпретируются как массивы. По умолчанию предполагается, что каждая переменная является массивом, например, запись *R=45* означает, что *R* – вектор, состоящий из одного элемента со значением *45*. Обратите внимание: на рис. 1.2 в окне *Workspace Browser* указан размер *1x1* переменных *a* и *d*.

В MATLAB можно работать с одномерными (вектор-столбец или вектор-строка), двумерными (матрица) и многомерными массивами. Нумерация элементов в массивах начинается с единицы. Размер массива – это количество элементов вдоль каждого измерения.

2.1 Формирование векторов и матриц

Ввод вектора-строки осуществляется следующим образом:

имя вектора=[значения элементов вектора через запятую или пробел]

Для ввода вектора-столбца элементы следует разделять точкой с запятой или после ввода значения каждого элемента нажимать клавишу Enter:

```
» a=[2 7 3 9 4];
» s=[3;5;7];
» h=[3
4]
h =
     3
     4
» a
a =
     2     7     3     9     4
» s
s =
     3
     5
     7
»
```

При вводе матрицы элементы строк разделяются пробелами или запятыми, а строки отделяются друг от друга точкой с запятой или нажатием клавиши Enter:

```
» s=[1,2,3; 4 5 6
7 8,9]
s =
    1    2    3
    4    5    6
    7    8    9
»
```

Элементами массивов могут быть выражения и комплексные числа:

```
» h=[8*sin(0.54) 3-9 sqrt(-1) 3+4i]
h =
  4.1131          -6.0000                0 + 1.0000i   3.0000 + 4.0000i
»
```

Объединение нескольких матриц или векторов A, B, C, D встык по горизонтали (объединяемые массивы должны иметь равное число строк) выполняется следующим образом:

[A,B,C,D] или *[A B C D]*

Для объединения массивов *A, B, C, D* встык по вертикали (объединяемые массивы должны иметь равное число столбцов) их следует разделить точкой с запятой или нажатием клавиши Enter.

Пример объединения массивов:

```
» a=[1 2;3 4];
» b=[5 6 7;8 9 10];
» d=[11 12 13];
» c=[8 5 6 4];
» f=[a b]
f =
    1    2    5    6    7
    3    4    8    9   10
» h=[d c]
```

```
h =
   11    12    13     8     5     6     4
» k=[d;b
d]
k =
   11    12    13
    5     6     7
    8     9    10
   11    12    13
»
```

Кроме списка значений, элементы массивов могут быть заданы диапазоном следующим образом:

начальное значение : шаг : конечное значение

Если шаг равен единице, его можно не задавать.

В следующих примерах формируется вектор *k*, элементы которого принадлежат диапазону от *3* до *7* с единичным шагом. Каждая строка матрицы *p* задана диапазоном (с шагом *3* и разными начальными и конечными значениями). При задании матрицы *u* диапазон задает первые четыре элемента первой строки и последние пять элементов второй строки.

```
» k=[3:1:7]
k =
    3     4     5     6     7
» p=[0:3:9; 1:3:10]
p =
    0     3     6     9
    1     4     7    10
» u=[0:3:9 100 30; 200 1:2:9 ]
u =
    0     3     6     9   100    30
  200     1     3     5     7     9
»
```

Если все элементы вектора задаются диапазоном, квадратные скобки можно не использовать:

```
» w=8:4:32
w =
    8    12    16    20    24    28    32
»
```

Единичный шаг можно не указывать:

```
» q=3:10
q =
    3     4     5     6     7     8     9    10
»
```

2.2 Обращение к элементу массива

Для обращения к элементам вектора используется один индекс (как для вектора-строки, так и для вектора-столбца), который записывается после имени вектора в круглых скобках, например, *g(7)* – седьмой элемент вектора *g*.

Для обращения к элементам матрицы используется два индекса – номер строки и номер столбца, которые записываются в круглых скобках через запятую, например, *k(3,7)* – элемент матрицы *k*, расположенный на пересечении третьей строки и седьмого столбца.

Кроме способов задания массивов, перечисленных в п. 2.1, элементам массива можно просто присвоить значения. При этом автоматически создается матрица требуемого размера. Элементы, которым не присвоено значение, будут равны нулю:

```
» r(3,2)=7
r =
     0     0
     0     0
     0     7
» r(5,1)=5
r =
     0     0
     0     0
```

```
     0     7
     0     0
     5     0
»
```

В примере автоматически создана матрица размером *3x2*, а после очередного присваивания ее размер был увеличен до *5x2*.

Начинать присваивать значения элементов массива лучше с элемента, расположенного в правом нижнем углу массива. В этом случае по мере добавления элементов не будет изменяться его размер (выделение памяти для массива буде выполняться один раз).

К элементам матрицы можно обратиться, используя один индекс, определяющий положение элемента в одномерном массиве, состоящем из следующих друг за другом столбцов матрицы. Например, выражения *f(2,2)* и *f(6)* задают один и тот же элемент массива *f*:

```
» f=[1 2 ;3 4; 5 6; 7 8]
f =
     1     2
     3     4
     5     6
     7     8
» f(2,2)
ans =
     4
» f(6)
ans =
     4
»
```

2.3 Применение оператора «двоеточие»

При задании диапазона индексов элементов массива используется оператор «:» («двоеточие»).

С помощью оператора «двоеточие» можно выделить фрагмент массива, например, выражение вида

```
m(in:ik,jn:jk)
```

выделяет подматрицу матрицы *m* с *in* по *ik* строку и с *jn* по *jk* столбец.

Например:

```
» h=[1 2 3 ;4 5 6;7 8 9]
h =
    1    2    3
    4    5    6
    7    8    9
» h(2:3,1:2)
ans =
    4    5
    7    8
»
```

Выделение строки и столбца матрицы *M* с помощью оператора двоеточие выполняется как:

M(номер строки , :) и M(: , номер столбца)

Например:

```
» d=[1 2 3; 4 5 6; 7 8 9]
d =
    1    2    3
    4    5    6
    7    8    9
» d(:,2)
ans =
    2
    5
    8
» d(1,:)
ans =
    1    2    3
»
```

При использовании оператора двоеточие вместо номера последней строки (столбца) можно использовать слово *end*. В примере на экран выводится фрагмент матрицы, начиная со второй строки и со второго столбца до конца массива:

```
» d=[1 2 3; 4 5 6; 7 8 9]
d =
    1    2    3
    4    5    6
    7    8    9
» d(2:end, 2:end)
ans =
    5    6
    8    9
»
```

2.4 Удаление строк и столбцов существующего массива

Для удаления строк или столбцов массива используется пустой массив (парные квадратные скобки *[]*). Размер пустого массива *0*x*0*. Для удаления строки или столбца необходимо присвоить пустой массив:

```
» d=[1 2 3; 4 5 6; 7 8 9]
d =
    1    2    3
    4    5    6
    7    8    9
» d(1,:)=[]
d =
    4    5    6
    7    8    9
»
```

Можно сразу удалить и несколько строк или столбцов. В примере удалены первый и второй столбцы:

```
» d=[1 2 3; 4 5 6; 7 8 9]
d =
```

```
     1     2     3
     4     5     6
     7     8     9
» d(:,1:2)=[]
d =
     3
     6
     9
»
```

2.5 Применение элементарных функций к векторам и матрицам

Аргументами элементарных функций MATLAB могут быть массивы. В этом случае функция возвращает массив того же размера, что и ее аргумент. Каждый элемент возвращаемого массива равен значению функции от соответствующего элемента массива-аргумента. Например:

```
» h=[1 2 3 ;4 5 6;7 8 9]
h =
     1     2     3
     4     5     6
     7     8     9
» sqrt(h)
ans =
    1.0000    1.4142    1.7321
    2.0000    2.2361    2.4495
    2.6458    2.8284    3.0000
»
```

2.5.1 Матричные и векторные операции

Матричные и векторные операции (табл. 2.2) выполняются в соответствии с правилами векторного и матричного исчисления.

Таблица 2.2 – Матричные и векторные операции

Операция	Описание	Пример
+	вычисление суммы матриц (векторов) одинакового размера. Результатом операции является массив того же размера, каждый элемент которого равен сумме соответствующих элементов складываемых массивов	*» b=[1 2 3;4 5 6];* *» c=[7 2 4; 4 8 3];* *» b+c* *ans =* *8 4 7* *8 13 9* *»*
-	разность массивов одного размера	*» c=[7 2 4; 4 8 3];* *» b=[1 2 3;4 5 6];* *» b-c* *ans =* *-6 0 -1* *0 -3 3* *»*
*	умножение матриц, скалярное произведение векторов (перемножить можно массивы, если число столбцов левого операнда равно числу строк правого операнда)	*» a=[1 2;3 4];* *» b=[1 2 3;4 5 6];* *» a*b* *ans =* *9 12 15* *19 26 33*
‘	(апостроф) – транспонирование массивов с действительными элементами, транспонирование и комплексное сопряжение массивов с комплексными элементами	*» c=[7 2 4; 4 8 3]* *c =* *7 2 4* *4 8 3* *» c'* *ans =* *7 4* *2 8* *4 3* *»* *» d* *d =* *3.0000 + 1.0000i*

		3.0000 - 7.0000i *» d'* *ans =* *3.0000 - 1.0000i* *3.0000 + 7.0000i*
.’	транспонирование массивов	*» d=[3.0000 + 1.0000i 3.0000 - 7.0000i]* *d =* *3.0000 + 1.0000i* *3.0000 - 7.0000i* *» d.'* *ans =* *3.0000 + 1.0000i* *3.0000 - 7.0000i* *» c=[5 6 7];c.'* *ans =* *5* *6* *7* *»*
^	возведение матрицы в степень (выполняется умножением матрицы на саму себя). Возведение матрицы в степень – *1* дает обратную матрицу. Функция *inv(A)* также предназначена для вычисления матрицы, обратной *A*. Возведение матрицы в отрицательную степень *(-n)* равно произведению обратной матрицы на саму себя *n* раз.	*» a=[2 3; 6 7]* *a =* *2 3* *6 7* *» a^3* *ans =* *206 255* *510 631* *» a*a*a* *ans =* *206 255* *510 631* *» a^-1* *ans =*

```
   -1.7500    0.7500
    1.5000   -0.5000
» ans*a
ans =
     1     0
     0     1
» a^-2
ans =
    4.1875   -1.6875
   -3.3750    1.3750
» inv(a)
ans =
   -1.7500    0.7500
    1.5000   -0.5000
»
```

Если один из операндов операций сложения, вычитания, умножения или правый операнд операции деления является числом, над каждым элементом массива и числом выполняется соответствующая операция. Например, найдем сумму матрицы *A* и числа *2* и разделим полученную матрицу на *5*:

```
» A=[1 2; 3 4]
A =
     1     2
     3     4
» A+2
ans =
     3     4
     5     6
» ans/5
ans =
    0.6000    0.8000
    1.0000    1.2000
»
```

Для вычисления векторного произведения векторов d и q (состоящих из трех элементов) используется функция **cross**(d,q):

```
 » d=[1 2 3];
» q=[4 5 6];
» cross(d,q)
ans =
   -3    6   -3
»
```

Найти скалярное произведение векторов *d* и *q* одного размера можно с помощью функции *dot(d,q)*. При ее использовании векторы должны иметь одинаковое количество элементов и быть строками или столбцами в любой комбинации:

```
» a=[1 2 3 4];
» b=[4 3 2 1];
» dot(a,b)
ans =
   20
» a*b
??? Error using ==> *
Inner matrix dimensions must agree.

» a*b'
ans =
   20
»
```

В примере операция *a*b* привела к ошибке, поскольку оба операнда – строки. После транспонирования вектора *b* причин для возникновения ошибки нет.

2.6 Поэлементные операции над массивами

Поэлементные операции (табл. 2.2) над матрицами и векторами выполняются только над массивами одного размера и типа (если их

оба операнда являются массивами). В результате получается массив того же размера и типа.

Таблица 2.2 - Поэлементные операции над матрицами и векторами

Операция	Описание	Пример
.*	Поэлементное умножение одного массива на другой. Результатом является массив, каждый элемент которого равен произведению соответствующих элементов перемножаемых массивов.	*» a=[1 4;2 5]* *a =* *1 4* *2 5* *» d=[1 2; 3 4]* *d =* *1 2* *3 4* *» a.*d* *ans =* *1 8* *6 20* *»*
./	Поэлементное деление левого операнда на правый	*» a./d* *ans =* *1.0000 2.0000* *0.6667 1.2500*
.\	Поэлементное деление правого операнда на левый	*» a.\d* *ans =* *1.0000 0.5000* *1.5000 0.8000*
.^	Поэлементное возведение в степень	*» a.^d* *ans =* *1 16* *8 625* *» a.^2* *ans =* *1 16* *4 25* *» a^2*

		ans = *9 24* *12 33* *»*

Поэлементные операции над массивами позволяют рассчитать значения элементов массива по одной и той же формуле без использования циклов. Например, для расчета таблицы значений функции достаточно с помощью оператора двоеточие сформировать вектор аргументов. Затем, используя матричные операции и элементарные функции, вычисляемые также поэлементно, рассчитать вектор значений функции. Например, рассчитаем таблицу значений функции

$f(x) = \dfrac{\cos(2x)}{\sqrt{1 + x \cdot \sin^2(x)}}$ на интервале от *5* до *7* с шагом *0.5*:

```
» x=5:0.5:7
x =
   5.0000   5.5000   6.0000   6.5000   7.0000
» f=cos(2*x)./sqrt(1+x.*sin(x).^2)
f =
  -0.3546   0.0023   0.6964   0.7956   0.0682
»
```

2.7 Функции для работы с массивами

Функции формирования и преобразования массивов приведены в табл. 2.1. Если при вызове функции указываются размеры массива, вызов функции с одним аргументом подразумевает квадратную матрицу.

Таблица 2.1 – Функции для работы с массивами

Функция	Выполняемое действие	Пример
zeros(n,m) *zeros(n)*	формирование матрицы размером *n*x*m* из нулевых элементов. Это же можно сделать, присвоив элементу, расположенному в правом	*» s=zeros(2,2)* *s =* *0 0* *0 0* *» h(3,2)=0*

	нижнем углу матрицы значение *0*. Если при вызове функции задать один аргумент, будет сформирована квадратная матрица.	*h =* *0 0* *0 0* *0 0* *»*
ones(n,m) *ones(n)*	формирование матрицы размером *n*x*m* каждый элемент которой равен *1*.	*» ones(2,3)* *ans =* *1 1 1* *1 1 1* *»*
eye(n,m) *eye(n)*	формирование матрицы размером *n*x*m*, элементы главной диагонали которой равны *1*, остальные равны *0*	*» eye(3)* *ans =* *1 0 0* *0 1 0* *0 0 1* *» eye(2,3)* *ans =* *1 0 0* *0 1 0*
rand(n,m) *rand(n)*	формирование матрицы размером *n*x*m*, элементы которой являются случайными числами, распределенными равномерно в интервале от *0* до *1*. Если задан один аргумент, формируется квадратная матрица порядка *n*.	*» rand(2,3)* *ans =* *0.8913 0.4565 0.8214* *0.7621 0.0185 0.4447* *»*
randn(n,m) *randn(n)*	формирование матрицы, элементы которой являются случайными числами, распределенными по нормальному закону (с математическим ожиданием *0* и среднеквадратичным	*» randn(2)* *ans =* *-0.4326 0.1253* *-1.6656 0.2877* *»*

	отклонением *1*). Аргументы задаются аналогично функции rand.	
magic(n)	формирование магической матрицы – квадратной матрицы порядка *n*, в которой суммы элементов каждого столбца, строки и диагоналей равны между собой	*» magic(3)* *ans =* *8 1 6* *3 5 7* *4 9 2*
diag(A) *diag(A,k)*	Если аргументом *A* функции является матрица, функция возвращает вектор элементов ее главной диагонали. Если *A* – вектор, функция возвращает диагональную матрицу, элементы главной диагонали равны соответствующим элементам вектора. Если при вызове функции задаются два параметра, второй параметр определяет номер диагонали (точка отсчета - главная диагональ, она имеет номер *0*. Диагонали, расположенные выше нее имеют положительные номера, ниже главной диагонали - отрицательные).	*» f=[1 2 3; 4 5 6; 7 8 9]* *f =* *1 2 3* *4 5 6* *7 8 9* *» diag(f)* *ans =* *1* *5* *9* *» g=[11 22 33];* *» diag(g,-2)* *ans =* *0 0 0 0 0* *0 0 0 0 0* *11 0 0 0 0* *0 22 0 0 0* *0 0 33 0 0* *»
cat(q, A, B, C)	объединение матриц *A, B, C* встык по вертикали (при *q=1*) или по горизонтали (при *q=2*)	*» f=[1 2 3];* *» k=[4 4 5];* *» cat(1,f,k)*

		ans = *1 2 3* *4 4 5* *» cat(2,f,k)* *ans =* *1 2 3 4 4 5* *»*
fliplr(A)	меняет порядок следования столбцов матрицы *A* на обратный (переставляет столбцы матрицы относительно вертикальной оси)	*» d=[1 2 3; 4 5 6; 7 8 9]* *d =* *1 2 3* *4 5 6* *7 8 9* *» fliplr(d)* *ans =* *3 2 1* *6 5 4* *9 8 7*
flipud(A)	меняет порядок следования строк матрицы *A* на обратный	*» d=[1 2 3; 4 5 6; 7 8 9]* *d =* *1 2 3* *4 5 6* *7 8 9* *» flipud(d)* *ans =* *7 8 9* *4 5 6* *1 2 3* *»*
rot90(A)	поворот матрицы A на угол 90° против часовой стрелки	*» d* *d =* *1 2 3* *4 5 6*

		7 8 9 *» rot90(d)* *ans =* *3 6 9* *2 5 8* *1 4 7* *»*
inv(A)	вычисление обратной матрицы	*» a=[2 3; 6 7]* *a =* *2 3* *6 7* *» inv(a)* *ans =* *-1.7500 0.7500* *1.5000 -0.5000* *»*
det(A)	вычисление определителя квадратной матрицы	*» a=[2 3; 6 7]* *a =* *2 3* *6 7* *» det(a)* *ans =* *-4* *»*
ndims(A)	возвращает количество размерностей многомерного массива	*» s=[4 5; 7 8]* *s =* *4 5* *7 8* *» ndims(s)* *ans =* *2*
size(A)	возвращает вектор первый элемент которого – количество строк, второй – количество столбцов матрицы *A*	*» h=[2 3 4; 4 5 6]* *h =* *2 3 4* *4 5 6*

		» s=size(h) *s =* *2 3* *» s(1)* *ans =* *2* *» s(2)* *ans =* *3* *»*
length(V)	возвращает количество элементов вектора *V*	*» s=[1 8 3 7]* *s =* *1 8 3 7* *» length(s)* *ans =* *4* *»*
sum(A) *sum(A, k)*	возвращает сумму элементов вектора *A*. Если *A* – матрица, функция возвращает вектор сумм элементов каждого столбца. Функция может быть вызвана с двумя аргументами: первый – массив, второй аргумент определять будут рассчитываться суммы элементов строк (при *k=2*) или столбцов (при *k=1*, это значение принимается по умолчанию)	*» s=[1 4 3 7];sum(s)* *ans =* *15* *» h=[1 2 3; 4 5 6];sum(h)* *ans =* *5 7 9* *» h=[1 2 3; 4 5 6];sum(h,2)* *ans =* *6* *15* *»
prod(A) *prod(A, k)*	возвращает произведение элементов вектора *A*. Если *A* – матрица, функция возвращает вектор произведений элементов каждого столбца. Второй	*» s=[1 4 3 7];prod(s)* *ans =* *84* *» h=[1 2 3; 4 5 6];prod(h)*

	параметр, как и для функции sum, задает расчет произведений строк или столбцов массива.	*ans =* *4 10 18* *»*
sort(A) *sort(A, k)*	Сортировка элементов вектора (элементов столбцов матрицы (при *k=1* или если второй аргумент отсутствует) или элементов строк матрицы (при *k=2*)) по возрастанию. Если вызвать функцию с двумя выходными параметрами, в первый из них будет записан отсортированный массив, во второй – старые значения индексов элементов отсортированного массива.	*» f=[3 6 1 3];sort(f)* *ans =* *1 3 3 6* *» h=[7 3 5; 1 5 2]* *h =* *7 3 5* *1 5 2* *» sort(h)* *ans =* *1 3 2* *7 5 5* *» sort(h,2)* *ans =* *3 5 7* *1 2 5* *» [a,b]=sort(h,2)* *a =* *3 5 7* *1 2 5* *b =* *2 3 1* *1 3 2* *»
sortrows(A) *sortrows(A,k)*	*sortrows(A)* - сортировка строк матрицы в порядке возрастания элементов первого столбца. *sortrows(A,k)* – сортировка элементов *k*–го столбца по возрастанию путем перестановки строк. Если *k* – вектор, сортировка выполняется по	*» a=[1 7 -5; -4 7 9; 4 -1 0]* *a =* *1 7 -5* *-4 7 9* *4 -1 0* *» sortrows(a)* *ans =*

	столбцу, номер которого равен первому элементу вектора, затем (при равенстве элементов) по второму и т.д.	*-4 7 9* *1 7 -5* *4 -1 0* *» sortrows(a,[2,1])* *ans =* *4 -1 0* *-4 7 9* *1 7 -5* *» sortrows(a,2)* *ans =* *4 -1 0* *1 7 -5* *-4 7 9* *»*
min(A) *max(A)* *min(A, [], k)* *max(A, [], k)*	возвращает минимальный (максимальный) элемент вектора или вектор минимальных (максимальных) элементов каждого столбца матрицы. Форма вызова функции с тремя параметрами используется для поиска минимальных (максимальных) элементов строк матрицы (при *k=2*) или ее столбцов (при *k=1*). Функция может быть вызвана с двумя возвращаемыми параметрами: первый – минимальный (максимальный) элемент или их вектор; второй – индексы минимальных (максимальных) элементов (например, для строк матрицы – номера столбцов, в которых расположены их наименьшие и	*» f* *f =* *3 6 1 3* *» min(f)* *ans =* *1* *» [a,b]=min(f)* *a =* *1* *b =* *3* *» h* *h =* *7 3 5* *1 5 2* *» [a,b]=min(h)* *a =* *1 3 2* *b =* *2 1 2* *» max(h,[],2)*

	наибольшие элементы). Если минимальных (максимальных) элементов в векторе (строке, столбце) несколько, функция возвращает индекс первого из них.	*ans =* *7* *5* *»*
any(A) *any(A, k)*	возвращает *1*, если в заданном векторе есть хотя бы один ненулевой элемент. Если аргументом является матрица, функция возвращает вектор, элементы которого соответствуют столбцам матрицы. Функция может быть вызвана с двумя параметрами: первый массив, второй – *(k)* определяет в столбцах (при *k=1*) или строках (при *k=2*) следует определить наличие ненулевых элементов.	*» c=[0 0; 4 6; 2 0]* *c =* *0 0* *4 6* *2 0* *» any(c)* *ans =* *1 1* *» any(c,2)* *ans =* *0* *1* *1* *»
all(A) *all(A, k)*	возвращает *1*, если в заданном векторе нет нулевых элементов. Если аргументом является матрица, функция возвращает вектор, элементы которого соответствуют столбцам матрицы. Функция может быть вызвана с двумя параметрами: первый массив, второй – *(k)* определяет в столбцах (при *k=1*) или строках (при *k=2*) следует определить отсутствие нулевых элементов.	*» c* *c =* *0 0* *4 6* *2 0* *» all(c)* *ans =* *0 0* *» all(c,2)* *ans =* *0* *1* *0* *»
reshape(A,n,	возвращает массив размера *n*x*m*,	*» f=[2 4; 5 6; 4 5]*

m)	сформированный из элементов массива *A* путем их последовательной выборки по столбцам	*f =* *2 4* *5 6* *4 5* *» s=reshape(f,1,6)* *s =* *2 5 4 4 6 5*
repmat(A,n, m)	возвращает двумерный массив, сформированный из *n* блоков *A* по вертикали и m по горизонтали	*a =* *1 2* *3 4* *» A=repmat(a,2,2)* *A =* *1 2 1 2* *3 4 3 4* *1 2 1 2* *3 4 3 4*
k=find(x) *k=find(x*условие*)* *[i,j]=find(X)* *[i,j]=find(X* условие*)*	*k=find(x)* и *[i,j]=find(X)* возвращает индексы ненулевых элементов одномерного *x* и двумерного *X* массивов. *k=find(x*условие*)* и *[i,j]=find(X*условие) возвращает индексы элементов одномерного *x* и двумерного *X* массивов, удовлетворяющие заданному условию	*» a* *a =* *1 2 -7* *8 4 -23* *56 0 3* *» [i,j]=find(a>5)* *i =* *2* *3* *j =* *1* *1* *» *

Полный список функций обработки данных может быть вызван командой *help datafun*.

2.8 Решение систем линейных уравнений

Для решения системы линейных уравнений вида $A \cdot x = B$, где A - матрица коэффициентов, B - вектор свободных членов, x - вектор решений применяется операция \: $x = A \setminus B$.

Система линейных уравнений может быть также решена путем обращения матрицы коэффициентов: $x = inv(A) * B$, следует отметить, что это нерационально.

Например, решим систему уравнений

$$\begin{cases} 2 \cdot x1 + 4 \cdot x2 + 7 \cdot x3 = 3 \\ 4 \cdot x1 + 5 \cdot x2 + 8 \cdot x3 = 5 \\ 3 \cdot x1 + 5 \cdot x2 + x3 = 1 \end{cases}$$

```
» a=[2 4 7; 4 5 8; 3 5 1]
a =
     2     4     7
     4     5     8
     3     5     1
» b=[3; 5; 1]
b =
     3
     5
     1
» x1=a\b
x1 =
    1.0444
   -0.5111
    0.4222
» x2=inv(a)*b
x2 =
    1.0444
   -0.5111
    0.4222
» a*x1
ans =
    3.0000
    5.0000
```

```
    1.0000
»
```

Операция *B/A* эквивалентна выражению *B*inv(A)*.

2.9 Особенности применения операций сравнения и логических операций к массивам

В качестве операндов операций отношения могут выступать числа и массивы. Сравниваемые массивы должны иметь одинаковые размеры. Результатом операции сравнения массивов является массив того же размера, что и операнды, состоящий из нулей и единиц (каждый элемент является результатом сравнения соответствующих элементов сравниваемых массивов):

```
» a=[1 5; 2 6]
a =
    1    5
    2    6
» b=[4 6; 1 4]
b =
    4    6
    1    4
» a>b
ans =
    0    0
    1    1
»
```

Если один из операндов является скаляром, а другой массивом, скаляр сравнивается с каждым элементом массива, результатом операции является массив того же размера, что и массив-операнд:

```
» a
a =
    1    5
    2    6
» a<=2
ans =
```

```
     1     0
     1     0
»
```

При выполнении операций >, >=, <, <= над массивами с комплексными элементами, сравниваются только действительные части комплексных элементов массивов. При выполнении операций == и ~= учитываются значения действительной и мнимой частей:

```
» c=[3+1i 3-7i]
c =
  3.0000 + 1.0000i   3.0000 - 7.0000i
» d=[1-3i 3-7i]
d =
  1.0000 - 3.0000i   3.0000 - 7.0000i
» c>d
ans =
     1     0
» c==d
ans =
     0     1
»
```

Выполнение логических операций (табл 1.2) над массивами одного размера производится поэлементно:

```
» a=[1 3 0 0];b=[0 1 3 2]; a&b
ans =
     0     1     0     0
» a|b
ans =
     1     1     1     1
»
```

2.10 Приоритет выполнения операций

Ниже перечислены операции в порядке убывания их приоритета:

1. Круглые скобки.

2. Транспонирование (.’), транспонирование с комплексным сопряжением (‘), возведение в степень (^), поэлементное возведение в степень (.^).

3. Унарный плюс (+), унарный минус (-). логическое отрицание (~).

4. Умножение и деление (.* , ./ , .\ , * , / , \ ).

5. Сложение (+) и вычитание (-).

6. Операции отношения (>, >=, < , <=, == , ~=).

7. Логическое И (&).

8. Логическое ИЛИ (|).

2.11 Примеры работы с матрицами

Пример 1. Отсортировать элементы побочной диагонали матрицы в порядке убывания.

```
» a=[1 2 7; 8 4 23; 56 3 9]
a =
    1    2    7
    8    4   23
   56    3    9
»
a1=rot90(a);d=diag(a1);d1=flipud(sort(d));b1=diag(d);c1=diag(d1);a1=
a1-b1+c1;
» a=rot90(rot90(rot90(a1)))
a =
    1    2   56
    8    7   23
    4    3    9
»
```

Пример 2. Дан одномерный массив *A*. Все положительные элементы поместить в массив *P*, а отрицательные и нулевые – в массив *N*. При формировании массивов *P* и *N* сохранить

первоначальный порядок следования элементов. Вывести на экран количество элементов в полученных массивах.

```
» a=[1 2 -7 8 4 -23 56 0 3 9]
a =
    1    2   -7    8    4  -23   56    0    3    9
»
at=a.';at(:,2)=at(:,1)>0;as=sortrows(at,2);np=sum(at(:,2));n=length(a);
» N=as(1:n-np,1).'
N =
   -7  -23    0
» P=as(n-np+1:n,1).'
P =
    1    2    8    4   56    3    9
» length(P)
ans =
    7
» length(N)
ans =
    3
» n
n =
   10
» np
np =
    7
»
```

Пример 3. Дан массив *A*. Удалить строку и столбец, на пересечении которых расположен максимальный элемент.

```
» a=[1 2 -7; 8 4 -23; 56 0 3 ]
a =
    1    2   -7
    8    4  -23
   56    0    3
» maxa=max(max(a))
```

```
maxa =
   56
» [i,j]=find(a==maxa)
i =
    3
j =
    1
» a(i,:)=[];  a(:,j)=[];
» a
a =
    2   -7
    4   -23
»
```

Пример 4. Дан массив *A*. Определить количество строк матрицы, совпадающих с ее первой строкой.

```
» a=[1 2 -7; 8 4 -23;1 2 -7; 56 0 3;1 2 -7; ]
a =
    1    2   -7
    8    4   -23
    1    2   -7
   56    0    3
    1    2   -7
» v=size(a)
v =
    5    3
» n=v(1)
n =
    5
» af=repmat(a(1,:),n-1,1)
af =
    1    2   -7
    1    2   -7
    1    2   -7
    1    2   -7
```

```
» y=a(2:end,:)==af
y =
     0     0     0
     1     1     1
     0     0     0
     1     1     1
» ns=sum(all(y,2))
ns =
     2
»
```

3 СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ М-ФАЙЛОВ: СЦЕНАРИЕВ И ПРОСТЕЙШИХ ФАЙЛ-ФУНКЦИЙ

В MATLAB можно работать как в командном режиме (команды и операторы вводятся в командном окне, немедленно выполняются и выводится результат), так и в режиме выполнения последовательности команд, записанной в текстовый файл. Файлы, содержащие команды и операторы MATLAB имеют расширение *.m* и называются М-файлами.

Существует два типа М-файлов: *сценарии* и *функции*. Сценарии и функции могут содержать любые команды и операторы MATLAB.

Для создания М-файла необходимо воспользоваться командой *File/New/M-file* (кнопка  *New M-file* панели инструментов). Эта команда открывает окно редактора М-файлов, предназначенного для ввода и отладки программ. После окончания ввода М-файл должен быть сохранен (в текущем или в любом другом каталоге). Для использования М-файла должен быть установлен путь поиска, ведущий к этому файлу.

Если требуется внести изменения в созданный М-файл, следует его открыть, внести изменения и сохранить модифицированный файл (команда *File/Save (File/Save as* – сохранить под другим именем) или аналогичная кнопка на панели инструментов редактора).

В М-файле в одной строке могут записываться несколько команд, разделенных символом; (точка с запятой), но для наглядности их удобнее располагать в отдельных строках. При работе с М-файлами точка с запятой также используется для подавления вывода в командное окно результатов выполнения отдельных команд. Длинное выражение может занимать несколько строк. При переносе строки применяется знак многоточия.

Типы переменных в М-файлах не декларируются, а определяются автоматически после присваивания им значений.

При вводе программы рекомендуется использовать комментарии, которые при выполнении программы игнорируются. Комментарий – это строка или ее часть, расположенная за знаком процента (%). Таким образом, комментарий может занимать отдельную строку или располагаться после операторов.

Комментарии, предшествующие первому оператору или первой пустой строке, используются справочной системой. Первая строка комментария выводится командами

lookfor ключевое слово

и

help имя каталога

Команда

help имя функции

выводит на экран все строки комментария, расположенные до пустой строки или первого оператора.

3.1 Сценарии

Сценарии предназначены для автоматизации работ, которые необходимо выполнять многократно (их использование избавляет от необходимости повторного ввода команд в командную строку).

Сценарии не имеют входных и выходных аргументов. При выполнении сценария его командам доступны данные рабочей области. Если при выполнении сценария создаются переменные, они также размещаются в рабочей области и остаются в ней после окончания выполнения сценария. Таким образом, сценарии работают только с переменными рабочего пространства и не имеют локальных переменных. Поэтому при выполнении нескольких сценариев, необходимо следить за согласованностью их общих переменных.

Для выполнения сценария в командной строке следует ввести имя файла, в котором он сохранен, и нажать Enter или использовать команду

run имя файла

Например, для запуска программы, сохраненной в файле ex1.m, необходимо ввести в командной строке *ex1* или команду *run ex1*. Программа также может быть запущена в окне редактора/отладчика.

При вызове М-файла из командной строки, выполняются следующие действия:

1. Проверяется наличие в рабочем пространстве переменной с таким именем.

2. Если переменная в рабочем пространстве переменная отсутствует, выполняется поиск встроенной функции с заданным именем.

3. Если предыдущие шаги не привели к положительному результату, выполняется поиск М-файла с введенным именем, причем поиск сначала выполняется в текущем рабочем каталоге системы, а потом в каталогах, установленных в пути поиска.

4. Если файл найден, он выполняется, в противном случае выводится сообщение об ошибке.

Пример 1. Составить сценарий для взаимной перестановки максимального и минимального элементов первого столбца исходной матрицы.

Текст сценария:

```
a=[3 4 5;6 4 2;9 7 5] % исходная матрица
[maxa,imax]=max(a(:,1)); % определяем максимальный элемент первого столбца
[mina,imin]=min(a(:,1)); % определяем минимальный элемент первого столбца

% выполняем взаимную перестановку
% максимального и минимального элементов первого столбца
v= a(imax,1);
a(imax,1)=a(imin,1);
a(imin,1)=v;

a  % матрица после перестановки
```

3.2 Установка путей поиска

Любой файл, который необходимо выполнить, должен находиться в текущем каталоге или в каталоге, указанном в списке путей поиска (доступа). Иначе требуемый файл не будет найден. Поэтому, если файл сохранен в каталоге, не указанном в списке путей поиска, следует внести в список путей поиска путь к этому каталогу.

Следующие команды предназначены для работы со списком путей поиска:

path	вывод на экран списка путей поиска
addpath путь поиска	добавить каталог в список путей поиска, например: *addpath 'c:\matlab_w'*
rmpath путь поиска	удалить каталог из списка путей поиска, например: *rmpath 'c:\matlab_w'*

Кроме перечисленных команд, есть более удобное средство просмотра путей доступа – *Path Browser*, окно которого, вызываемое командой *File/ Set Path…* или кнопкой  *Path Browser* командного окна, представлено на рис. 3.1.

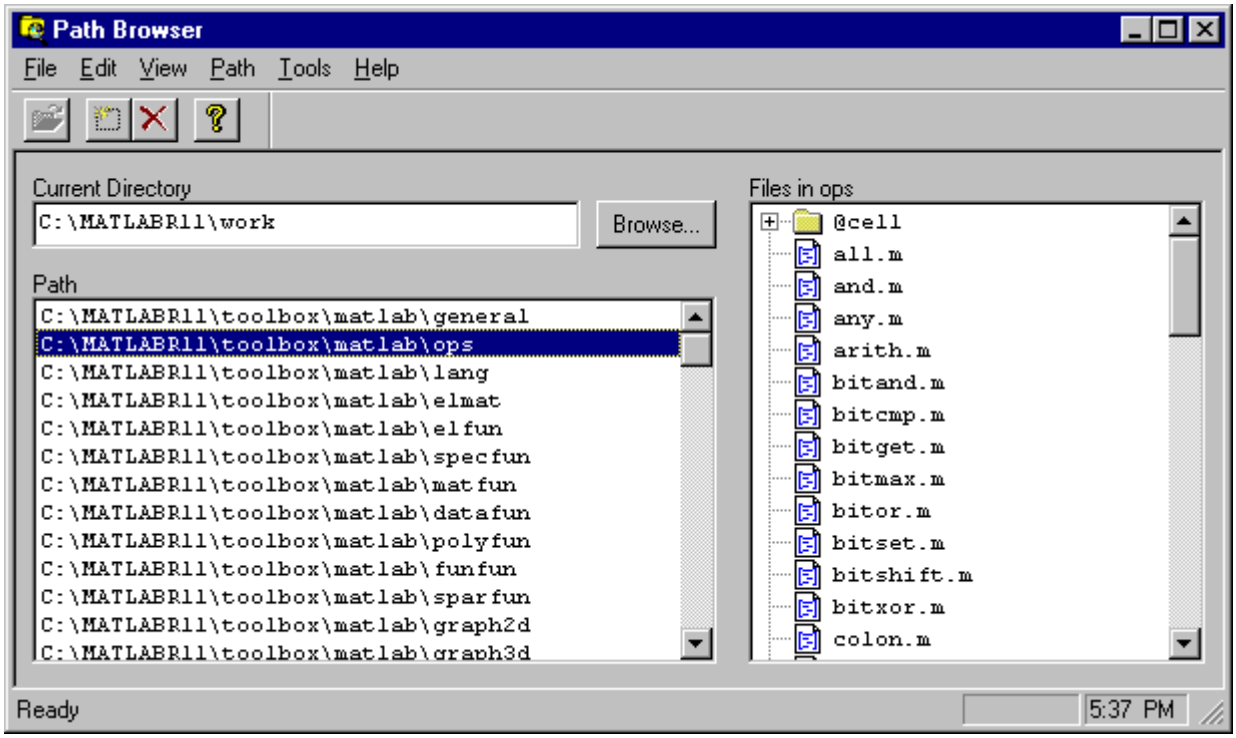


Рисунок 3.1 – Окно **Path Browser**

Path Browser позволяет посматривать, удалять и назначать пути поиска, изменять их последовательность в списке, просматривать все файлы MATLAB.

Поле *Current Directory* предназначено для отображения и изменения (кнопка Browse правее поля) текущего каталога. Поле *Path* содержит список путей доступа. Поле *Files in* имя каталога,

выделенного в поле *Path* содержит список файлов и подкаталогов выделенного каталога.

Команды меню *Path Browser* и соответствующие им кнопки панели инструментов приведены в таблице 3.1.

Таблица 3.1 – Команды меню *Path Browser*

Меню	Команда	Кнопка панели инструментов	Назначение
File	*Open*		Открыть файл
	Save Path		Сохранить путь доступа для использования в последующих сеансах работы.
	Exit Path Brouser		Выход из *Path Brouser*
Edit	*Undo*		Отменить последнюю операцию
	Redo		Вернуть последнюю операцию
View	*Toolbar*		Вывести панель инструментов
	Status Bar		Вывести панель состояния
	Editor Debugger		Открыть окно редактора/отладчика М-файлов
	Path Browser		Показать *Path Browser*
	Workspace Browser		Открыть окно *Workspace Browser*
Path	*Add to Path*		Добавить каталог в список путей поиска
	Remove from Path		Удалить каталог из списка путей поиска
	Refresh		Обновить пути поиска (после внесения изменений командами из командного окна)

Меню	Команда	Кнопка панели инструментов	Назначение
	Restore Defaults		Восстановить установки, принятые по умолчанию
Tools	*Run*		Выполнить М-файл
	Customize		Выбрать настройки
	Options		Установить опции
	Font		Установить шрифт
Help	*About Matlab Path Browser*		Вывод информации о *Path Browser*

Для перемещения каталога в другую позицию списка путей поиска в области *Path* следует захватить его левой кнопкой мыши и переместить.

3.3 Функции

Функции предназначены для реализации изолированного фрагмента программы, решающего определенную задачу. Они могут иметь входные и выходные аргументы (иметь параметры и возвращать одно или несколько значений). Параметрами функций и возвращаемыми значениями могут быть массивы различных типов, размерностей и размеров. Функции, как и сценарии, могут состоять из команд и операторов MATLAB и хранятся в текстовых файлах с расширением *.m*. Имя М – файла, в котором хранится функция, должно совпадать с именем функции. Функция должна быть размещена в одном из каталогов, входящих в список доступа MATLAB.

Функция может быть вызвана из командной строки, других функций или сценария. Если функция возвращает значение, ее вызов может входить в выражение, если тип возвращаемого значения совместим с этим выражением.

Текст функции (ее определение) начинается с заголовка функции, после которого следует тело функции. Заголовок определяет интерфейс функции и имеет следующий вид:

function [y1, y2, y3, ...]= имя функции *(x1, x2, x3, ...)*

где *y1, y2, y3, ...* - возвращаемые значения; *x1, x2, x3, ...* - параметры функции. Например, заголовок функции с именем *F1*, одним параметром a и тремя возвращаемыми значениями *x, y, z* имеет вид:

function [x,y,z]=F1(a)

Заголовок функции *F2* с двумя параметрами *a, b* и одним возвращаемым значением *x* будет выглядеть так:

function x=F2(a,b)

Функция *F3* имеет три параметра *a,b,c* и не имеет возвращаемых значений

function F3(a,b,c)

Функция *F4* не имеет параметров и возвращаемых значений

function F4

Тело функции состоит из команд и операторов MATLAB. Возвращаемым переменным в теле функции должны быть присвоены значения (в противном случае при вызове функции будет выведено соответствующее сообщение).

Комментарий, помещенный после заголовка функции до первой пустой строки или до первого оператора, выводится командой *help* имя функции. Первая строка этого комментария используется аналогично первой строке комментариев сценариев.

Вызов функции, возвращающей несколько значений (которые должны быть присвоены переменным *Y1, Y2, Y3, ...*) записывается следующим образом:

*[Y1, Y2, Y3, ...]=*имя функции (список параметров через запятую)

В качестве примера запишем вызовы функций *F1, F2, F3, F4*, заголовки которых рассматривались ранее. При этом значения параметров и имена переменных, которым должны быть присвоены возвращаемые значения, для наглядности будем записывать заглавными буквами:

[X,Y,Z]=F1(A)

X=F2(A,B) или F2(A,B) – в выражении

F3(A,B,C)

F4

Пример 2. Составить функцию для определения количества элементов матрицы, больших заданного числа. Определить индексы этих элементов.

Определение функции (сохранено в файле *l3_p2.m*) имеет вид:

```
function [k, ind]=fk(A,Y)
%Определение количества элементов матрицы A, больших заданного числа Y
k=sum(sum(A>Y));%Определение количества
[i,j]=find(A>Y);%индексы элементов матрицы A, больших заданного числа Y
ind=[i,j];%индексы записываем в матрицу (первый столбец - номера строк, второй - столбцов)
```

Формируем матрицу a и вызываем функцию из командной строки:

```
» a=[6 8 2; 5 9 12; 4 2 78]
a =
    6    8    2
    5    9   12
    4    2   78
» [g,h]=l3_p2(a,7)
g =
    4
h =
    1    2
    2    2
    2    3
    3    3
»
```

В М-файл можно поместить определения нескольких функций. Одна из этих функций (основная) должна иметь такое же имя, как и имя файла. При этом извне (из командной строки, сценария или функции, определение которой размещено в другом файле) может быть вызвана только основная функция. Остальные функции (они

называются вспомогательными или подфункциями) могут быть вызваны основной или друг другом.

Пример 3. Длина отрезка задана в дюймах (*1 дюйм =2,54 см*). Перевести значение длины в метрическую систему, то есть выразить ее в метрах, сантиметрах и миллиметрах. Так, например, *21 дюйм = 0 м 53 см 3,4 мм.*

Для решения задачи составим три функции: основную с именем pr и две вспомогательные *D* (для перевода дюймов в миллиметры) и *M* (для перевода миллиметров в метры, сантиметры, миллиметры). Текст *M*-файла *pr.m* с функциями *pr, D* и *M*:

```
function [m, sm, mm]=pr(d)
%pr- перевод длины отрезка в дюймах в метры, сантиметры, миллиметры
mm=D(d);
[m,sm,mm]=M(mm);

function mm=D(d)
%перевод  дюймов в миллиметры
mm=d*25.4;

function [m,sm,mm]=M(mm)
%перевод миллиметров в метры, сантиметры, миллиметры
m=fix(mm/1000);%метры
mm=rem(mm,1000);%отбрасываем целое число метров
sm=fix(mm/10);%сантиметры
mm=rem(mm,10);%миллиметры
```

Для иллюстрации переведем отрезок длиной *121* дюйм в метры, сантиметры, миллиметры (обратите внимание на вызов функции *pr*):

```
» [m1, sm1,mm1]=pr(121)
m1 =
     3
sm1 =
     7
mm1 =
```

```
    3.4000
» whos
  Name      Size        Bytes  Class

  m1        1x1             8  double array
  mm1       1x1             8  double array
  sm1       1x1             8  double array

Grand total is 3 elements using 24 bytes

» help pr

 pr- перевод длины отрезка в дюймах в метры, сантиметры,
миллиметры

» D(12)
??? Undefined function or variable 'D'.

»
```

При попытке вызова вспомогательной функции *D* из командной строки выводится сообщение об ошибке.

Все переменные, используемые в функциях, являются локальными (в рабочем пространстве и для других функций они недоступны). Это иллюстрирует результат выполнения команды *whos* в предыдущем примере. Переменные разных функций этого примера, имеющие одинаковые имена, фактически являются разными переменными.

Если необходимо ограничить возможность вызова функции только функциями заданного каталога, в нем может быть создан подкаталог с именем private и поместить туда файл с функцией. Все функции из каталога private видны только функциям из родительского каталога. Каталог *private* невозможно добавить в список путей поиска командой *File/Set Path*…

При вызове функции выполняются поиск переменной с таким именем или функции (до первой встретившейся функции с таким именем) в следующей последовательности:

1. Встроенная функция.
2. Подфункции в том же М-файле.
3. М-функция в подкаталоге private.
4. М-функция в текущем рабочем каталоге.
5. М-функция в каталогах, установленных в пути поиска (по порядку сверху вниз списка).

В случае обнаружения функции она выполняется, в противном случае выводится сообщение об ошибке.

3.4 Глобальные переменные

В функциях используются локальные переменные, а в сценариях - переменные рабочего пространства. В случае необходимости одноименные переменные рабочего пространства и одной или нескольких функций могут быть объявлены глобальными с помощью команды

global список имен пременных через запятую

Такое объявление должно быть помещено в каждую функцию, где будут использоваться глобальные переменные, и сделано в командном окне, если глобальная переменная должна быть доступна и в рабочей области.

В примере глобальная переменная *n* используется в качестве счетчика общего числа вызовов функций *r1* и r2. Определения функций:

```
function r1
global n
n=n+1;
```

```
function r2
global n
n=n+1;
```

Фрагмент сессии:

```
» global n
» n=0;
» r1
» n
n =
    1
» r2
» n
n =
    2
» r2
» n
n =
    3
» whos
  Name      Size            Bytes  Class

  n         1x1                 8  double array (global)

Grand total is 1 elements using 8 bytes

»
```

3.5 Редактор/отладчик М-файлов. Отладка программ

Редактор/отладчик М-файлов Editor/Dubugger (рис. 3.2) предназначен для создания, редактирования и отладки М-файлов. Редактор/отладчик может быть вызван из командной строки командой *edit* или командой *File/New/M-file* (кнопкой *New M-file*).

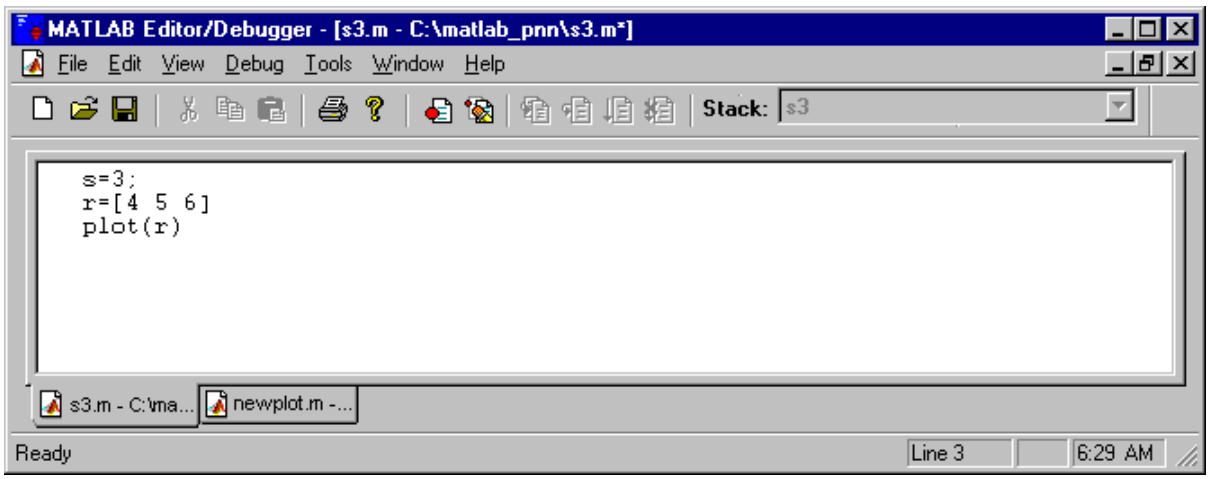


Рисунок 3.2 - Окно Редактора/отладчика М-файлов

Если в редакторе открыто несколько файлов, он становится многооконным. Каждый файл располагается на отдельной вкладке, переключение между которыми осуществляется щелчком мышью на корешке вкладки. В строке заголовка редактора М-файлов отображается имя активного файла и путь к нему на диске. Если после имени отображается символ «*», это означает, что файл после внесения изменений не был сохранен.

Меню *File* содержит команды для работы с М-файлами (создание, открытие, закрытие, сохранение, печать).

Меню *Edit* включает команды отмены выполненных действий, повтора отмены, выделения, копирования, вырезания и вставки фрагментов текста М-файла, поиска и замены заданного текста. Команда *Edit/Go To Line* позволяет немедленно перейти к строке с заданным номером (номера строк отображаются в правом нижнем углу строки состояния).

Команды меню *View* позволяют управлять отображением панели инструментов (*Toolbar*) и строки состояния (*Status Bar*) содержат команды вызова *Path Browser* и *Workspace Browser*. Команда *Evaluate Selection* предназначена для выполнения выделенного фрагмента М-файла. Команда *Auto Indent Selection* служит для автоматического выполнения отступов в выделенном фрагменте.

Команды меню *Debug* и соответствующие им кнопки панели инструментов предназначены для отладки программ. Они приведены в таблице 3.2.

Таблица 3.2 – Команды меню *Debug*

Команда	Кнопка	Описание
Continue		Продолжить выполнение М-файла до его завершения или первой точки останова
Single Step		Выполнить текущую строку
Step in		Выполнить текущую строку, но если она содержит вызов М-функции, осуществить переход к первой выполняемой строке вызываемой функции
Quit Debugging		Выйти из режима отладки
Set/Clear Breakpoint		Установить/удалить контрольную точку в строке, где находится курсор
Clear All Breakpoints		Удалить все контрольные точки
Stop if Error		Останов в случае обнаружения ошибки
Stop if Warning		Останов в случае предупреждения
Stop if NaN or Inf		Останов если результат *NaN* или *Inf*

Меню *Tools* содержит команды настройки параметров редактора, шрифта, команду *Run* – выполнить М-файл.

Для отладки М-файлов применяются точки останова и пошаговое выполнение программы (табл. 3.2). Строки, в которых назначены точки останова, слева помечаются красным кружком. После задания точек останова функция (сценарий) запускается на выполнение командой *Tools/Run* или из командного окна**.**

В процессе выполнения программы произойдет ее останов в первой назначенной точке останова. Когда выполнение программы останавливается в точках останова, левее строки появляется желтая стрелка. Стрелка, направленная вправо, означает, что эта строка будет выполняться следующей, а стрелка, направленная вниз, означает, что достигнут конец функции. При остановке в точке останова в

командной строке изменяется знак приглашения к вводу команд с >> на *K*>>.

В точках останова можно проанализировать значения переменных, параметров функции, выражений. Чтобы увидеть значение переменной, достаточно подвести курсор к ее имени в тексте программы, после чего на экране появится всплывающий желтый прямоугольник со значением переменной внутри его. В процессе отладки значение переменной также может быть выведено в командное окно. Для этого следует выделить переменную и воспользоваться командой *View/ Evaluate Selection.* Кроме этого, перед выполнением программы можно временно удалить символ; в конце операторов для просмотра результатов в процессе ее выполнения.

В процессе отладки значения переменных могут быть изменены. Для этого в месте, где следует внести изменения, необходимо назначить точку останова и во время паузы в командном окне присвоить переменной требуемое значение.

Продолжить выполнение программы можно одной из команд *Continue, Single Step* или *Step in.*

Кроме отладочных команд редактора/отладчика, вводимых с помощью меню и панели инструментов, существуют аналогичные команды, которые вводятся в строке ввода командного окна [3].

4 ПОСТРОЕНИЕ ГРАФИКОВ

В MATLAB есть средства для построения двух и трехмерных графиков и их форматирования.

4.1 Построение двухмерных графиков

Двухмерные графики могут быть построены в декартовой и полярной системах координат в линейном и логарифмическом масштабах.

Для построения графика функции необходимо задать вектор значений аргумента, рассчитать вектор значений функции и вызвать функцию построения графика.

4.1.1 Построение графиков в декартовой системе координат

Для построения графика функции в линейном масштабе используется функция

plot (X, Y).

В вектор *X* должен быть записан массив значений аргумента функции, а в вектор *Y* – массив значений функции. Функция *plot* соединяет точки графика отрезками прямых линий.

Например, построим график функции $f(x)=\frac{1}{x^2+1}$ на интервале от *–5* до *5*. Для этого в командную строку вводим:

```
» x=-5:0.1:5; y=1./(x.^2+1); plot(x,y)
```

На экране появится окно с графиком, представленное на рис. 4.1. Окно, в которое выводится график, включает область графика, панель инструментов и меню. При использовании команды построения графика графическое окно создается автоматически.

Если аргументом функции *plot(Y)* является один одномерный массив, будет построена зависимость значения элементов вектора y от их индексов. Если элементы вектора *Y* комплексные, будет построен график *plot(real(Y),imag(Y))*. Если *Y* – двумерный массив – будут построены зависимости элементов столбцов от номера строки.

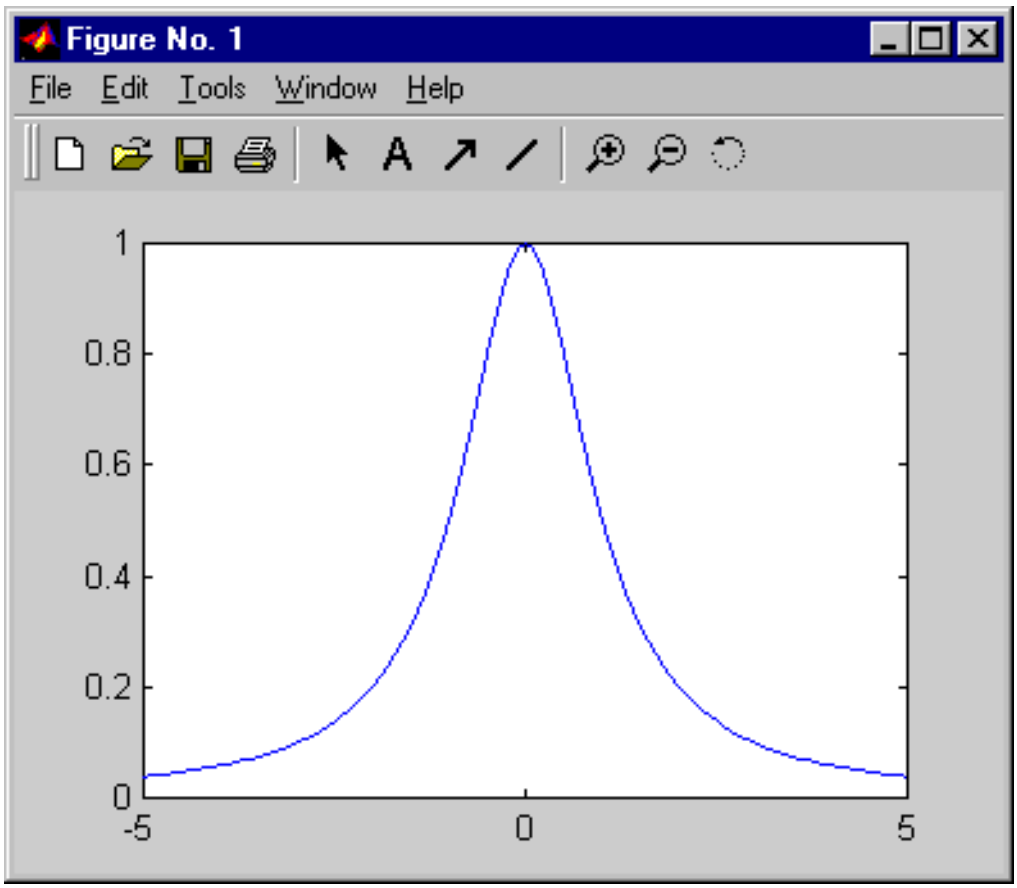


Рисунок 4.1 – График функции $f(x)=\frac{1}{x^2+1}$.

Команда *plot(X, Y)* если один или оба аргумента - двумерные массивы, задает следующие построения:

- если массив *X* – одномерный, а *Y* – двумерный, строятся графики зависимости столбцов массива *Y* от элементов вектора *X*;

- если массив *Y* – одномерный, а *X* – двумерный, строятся графики зависимости элементов вектора *Y* от столбцов массива *X*;

- если массивы *X* и *Y* двумерные, строятся зависимости столбцов массива *Y* от столбцов массива *X*.

Команда *plot(X,Y,LS)* позволяет с помощью строковой переменной *LS* (заключается в апострофы) задать способ отображения линии, ее цвет, маркеры точек (табл. 4.1). Например, команда

```
» a=-10:0.5:10; plot(a,a.^2,'ro-')
```

задает построение параболы на интервале от *–10* до *10* сплошной красной линией с маркерами в виде кружков (рис. 4.2)

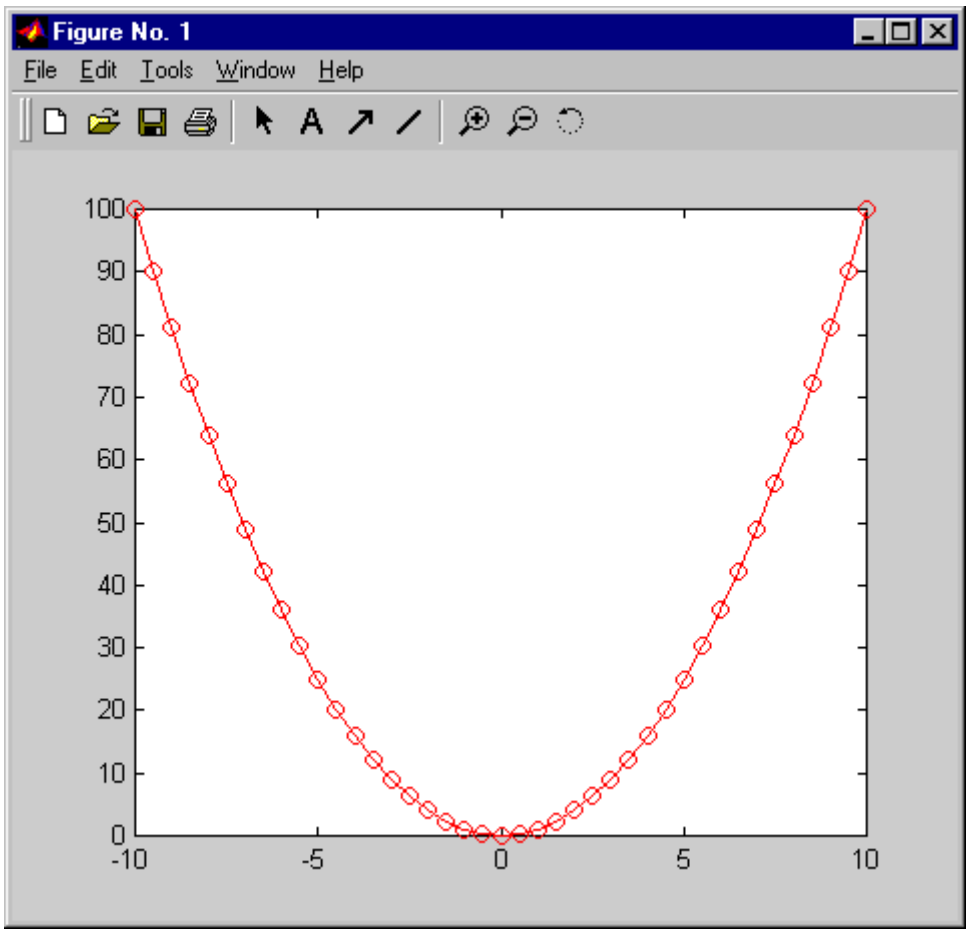


Рисунок 4.2 – Пример задания формата линии

Таблица 4.1 – Параметры линии

Тип линии	символ	Цвет линии	символ	Тип маркера	символ
сплошная	-	*желтый*	*y*	*точка*	.
штриховая	--	*фиолето-вый*	*m*	*плюс*	+
пунктирная	:	*голубой*	*c*	*звездочка*	*
штрих-пунктирная	-.	*красный*	*r*	*кружок*	*o*
отсутствие символа при наличии маркера	*нет линии*	*зеленый*	*g*	*крест*	*x*
		синий	*b*	*квадрат*	*s*
		белый	*w*	*ромб*	*d*
		черный	*k*	*пятиугольник*	*p*
				шестиугольни	*h*

Тип линии	символ	Цвет линии	символ	Тип маркера	символ
				к	
				стрелка влево	<
				стрелка вправо	>
				стрелка вниз	*v*
				стрелка вверх	^

Команда *plot(X1,Y1,LS1, X2,Y2,LS2, ...)* выполняет построение нескольких групп графиков линиями с разными параметрами.

Команда *plot(X,Y,LS, 'PN',PV ...)* применяется для задания свойств линии (**'***PN***'** – имя свойства, *PV* – устанавливаемое значение свойства), используемой для построения графика. Например, при задании значения *3* для толщины линии команда предыдущего примера будет выглядеть так:

```
» a=-10:0.5:10; plot(a,a.^2,'r-h','LineWidth',3)
```

Для построения графиков в логарифмическом масштабе по обеим координатным осям вместо функции *plot* используется функция *loglog* с тем же набором аргументов. Построить график функции, используя логарифмический масштаб по оси *X* и линейный масштаб по оси *Y* можно с помощью функции *semilogx*, а функция *semilogy* реализует построение в логарифмическом масштабе по оси *Y* и линейном по оси *X*. Например:

```
» x=0:0.2:5; loglog(x,exp(x))
» figure;
»  semilogx(x,exp(x))
» figure;
»  semilogy(x,exp(x))
»
```

Результаты представлены на рис. 4.3

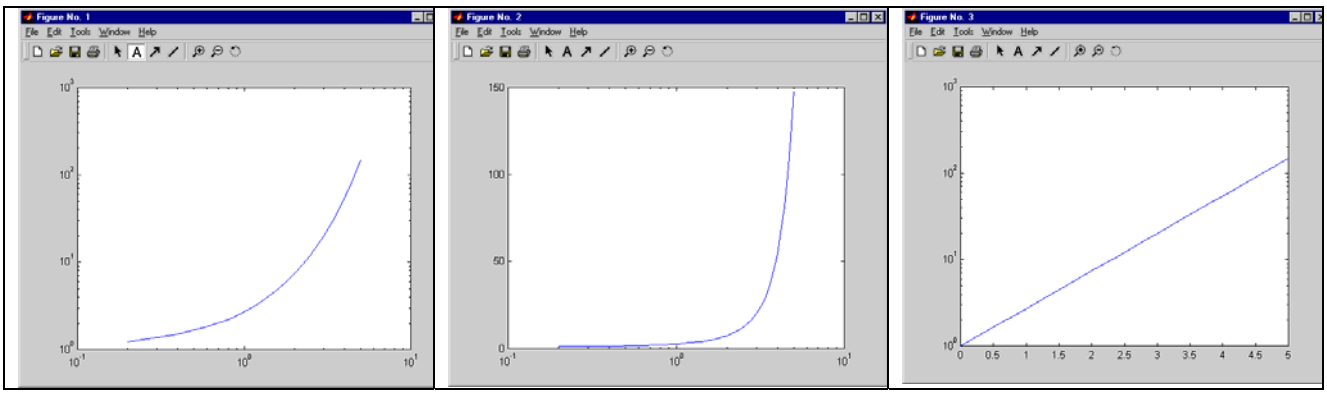


Рисунок 4.3 – Графики функций, построенные с использованием логарифмического масштаба

Функция

plotyy(x1,y1,x2,y2)

строит два графика функций, заданных массивами *x1, y1* и *x2, y2* с двумя осями координат (левая ось соответствует первому графику, правая - второму). Если *y1, y2* – двумерные массивы, количество графиков равно числу столбцов в них.

Например, построим синусоиду и параболу на интервале от -π до π:

```
» x=-pi:0.01:pi; plotyy(x,sin(x),x,x.^2)
```

Результат представлен на рис. 4.4.

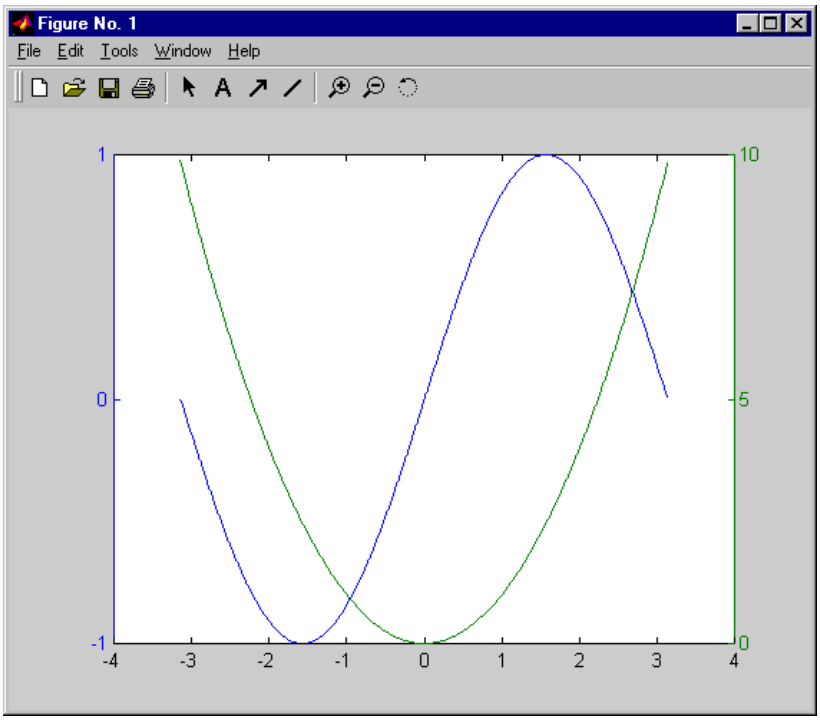


Рисунок 4.4 – Пример построения графиков с двумя координатными осями

Кнопки  ,  и  панели инструментов графического окна предназначены соответственно для добавления текста, стрелки и линии в области графика. Редактирование графика, его отдельных составляющих, задание параметров осей, линий, легенды и т.д. в графическом окне после построения графика осуществляется командами меню *Edit* и *Tools*.

Для сохранения графического окна используется команда *File/Save* в этом случае графическое окно сохраняется полностью, а файл имеет расширение *fig*. Открыть графическое окно можно командой *File/Open*.

Для сохранения графика в одном из стандартных графических форматов, следует указать его расширение. Кроме команд *File/Save* и *File/Save As* для экспорта графиков можно воспользоваться командой *File/Export* и при сохранении в списке «Тип файла» выбрать требуемый формат.

4.1.2 Построение нескольких графиков в одном окне

При задании очередной команды построения графика, в окне будет построен новый график, а предыдущий исчезнет. Для того, чтобы в одном окне построить несколько графиков, применяется команда hold on (включает режим сохранения предыдущего графика). Для отмены режима сохранения предыдущего графика используется команда *hold off* (отключает режим сохранения предыдущего графика). Команда *hold* реализует переключение с одного режима на другой.

Например:

```
» x=-5:0.1:5; y=1./(x.^2+1); plot(x,y); hold on; plot(x,0.5*sin(2*x)); hold off
```

Результат построения представлен на рис. 4.5

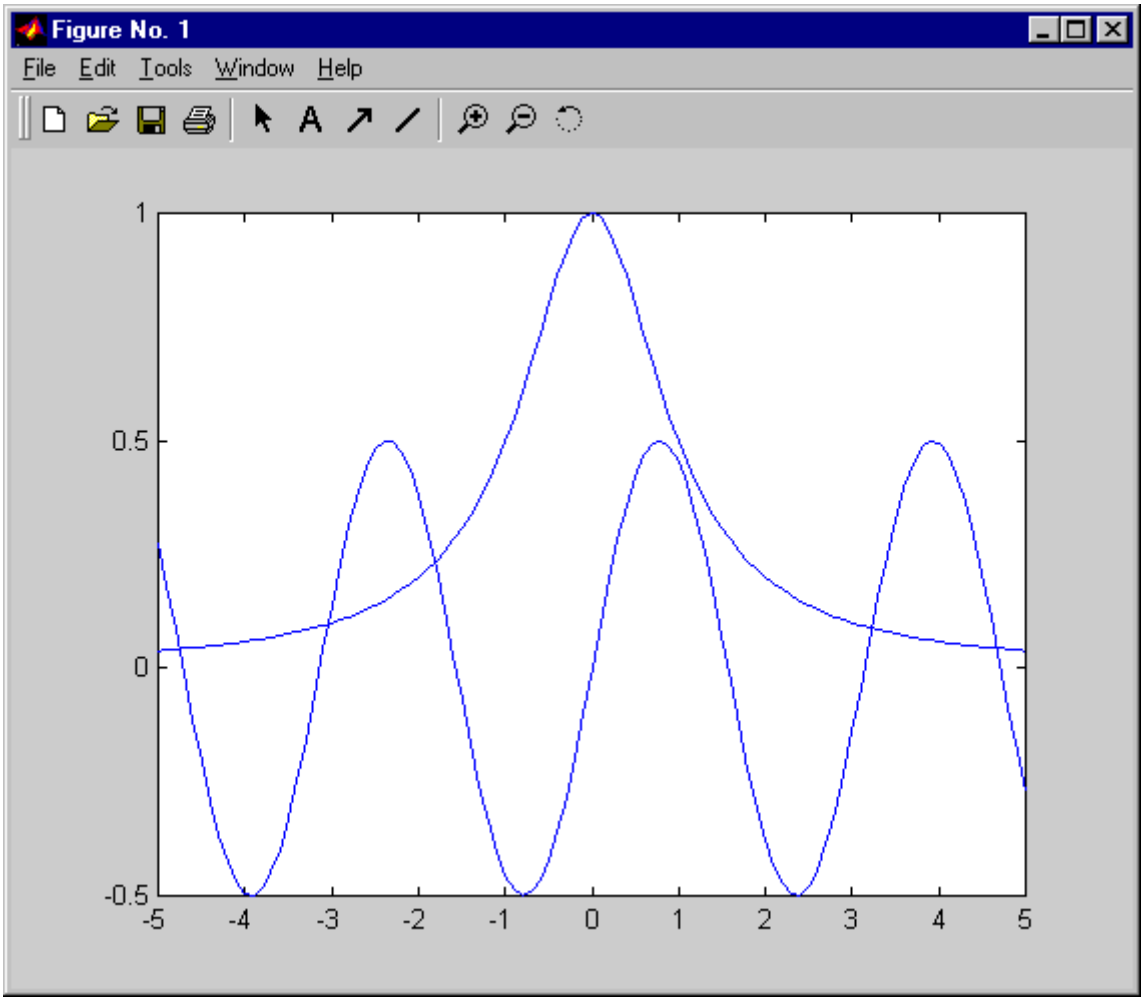


Рисунок 4.5 – Несколько графиков в одном окне построены с помощью команды hold on

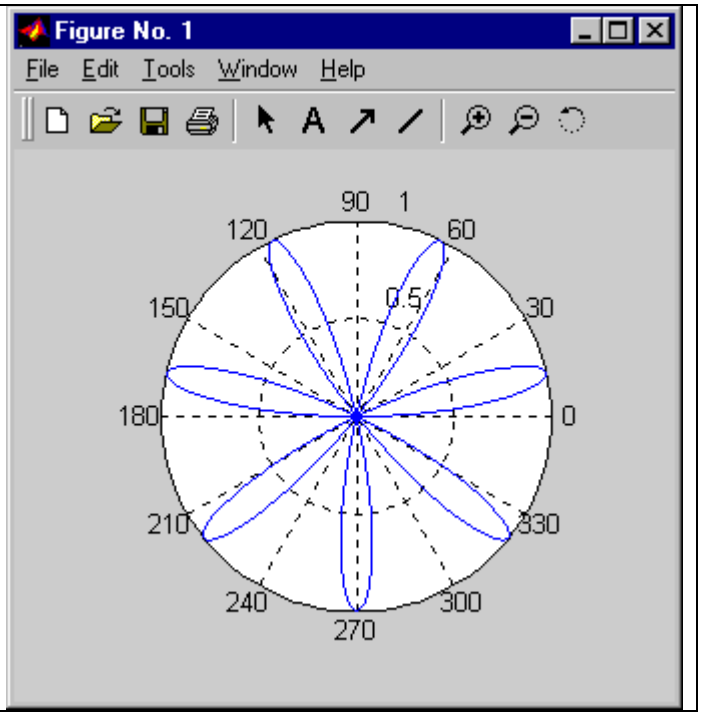
4.1.3 Построение графиков в полярной системе координат

Функция

polar(F,R,LS)

выполняет построение графика в полярной системе координат, задаваемого полярным углом, и полярным радиусом, значения которых записаны в массивы *F* и *R* соответственно. Необязательная строковая переменная *LS* задает свойства линии (как для функции *plot*). Если *F* и *R* – двумерные массивы одного размера, будут построены графики по каждому их столбцу.

Например, построим график функции $\rho(\varphi) = \sin(7\varphi)$ на интервале от *0* до π:

*» x=0:0.001:pi; polar(x,sin(7*x))*	

4.2 Управление графическими окнами

Для создания нового или перехода к существующему графическому окну используется команда *figure (n),* где *n* – номер графического окна. Если графическое окно с заданным номером не существует, будет создано новое окно. Если окно с заданным номером существует, оно становится активным, т.е. последующие графические команды будут относиться к нему. Команда *figure* без параметров создает новое окно с наименьшим несуществующим номером.

Команда *close(n)* или *close n* закрывает графическое окно с номером *n*. Команда *close all* закрывает все открытые графические окна.

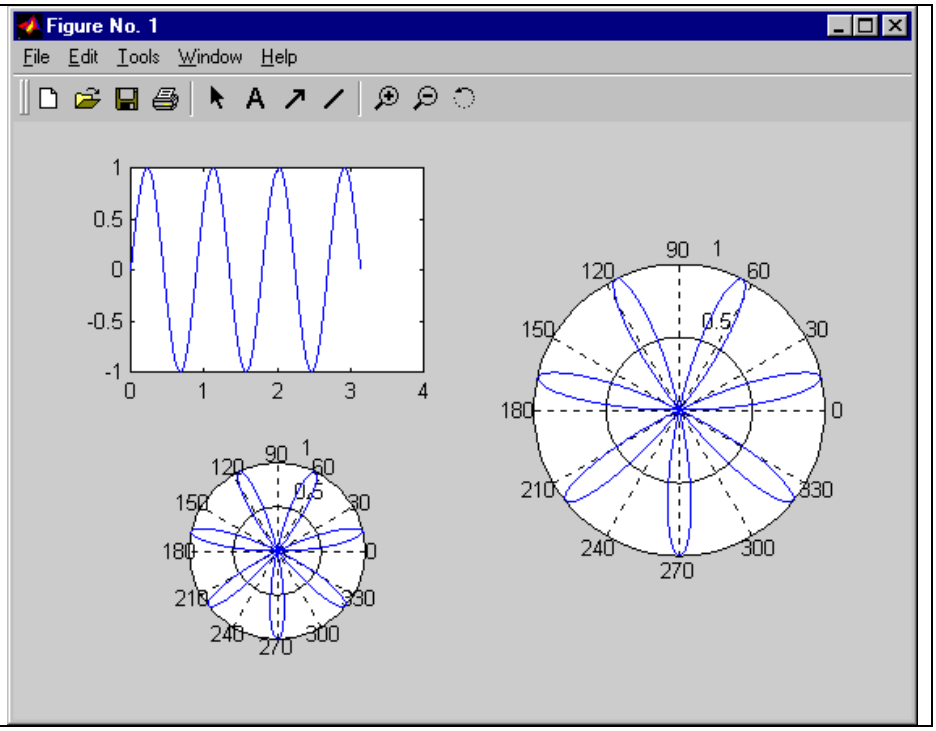
4.3 Разбивка графических окон на подокна

Графическое окно можно разделить на подокна и в каждом из них вывести графики различных функций. Для разделения окна на *n* подокон по вертикали и *m* подокон по горизонтали и организации вывода в *k*-е окно (если считать слева направо сверху вниз), используется функция

subplot(n, m, k)

Например:

```
» x=0:0.001:pi;
y=sin(7*x);
» subplot(2,2,1);
plot(x,y);
» subplot(2,2,3);
polar(x,y);
» subplot(1,2,2);
polar(x,y);
»
```



4.4 Построение трехмерных графиков

Для построения линий в пространстве используется функция **plot3(X, Y, Z)**

Эта функция является трехмерным аналогом функции **plot** и предназначена для соединения прямыми линиями точек, абсциссы, ординаты и аппликаты которых записаны соответственно в массивах X, Y, Z.

Следующий фрагмент позволяет построить в пространстве кривую, приведенную на рис. 4.6.

```
» z=0:0.01:70;  x=(1-z/70).*cos(z);  y=(1-z/70).*sin(z);  plot3(x,y,z)
```

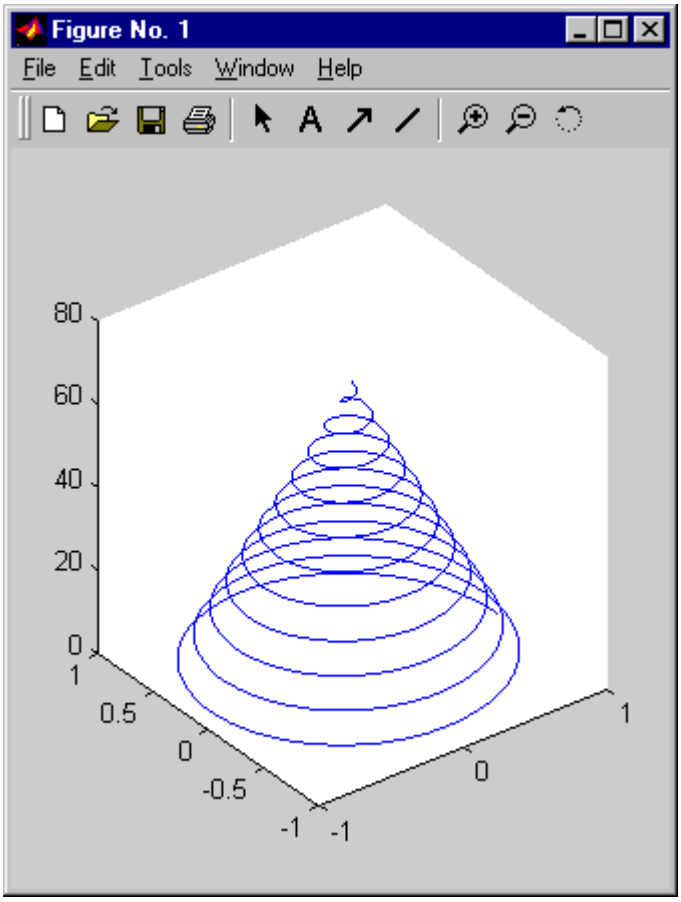


Рисунок 4.6 – Кривая, построенная с использованием функции *plot3*

Для построения графика функции двух переменных необходимо:

1. Сформировать две матрицы со значениями абсцисс и ординат каждой узловой точки графика. Для этого с помощью функции

[X, Y]=meshgrid(x, y)

задав в качестве аргументов векторы значений абсцисс x и ординат y узловых точек, формируются матрицы *X, Y* значений абсцисс и ординат. При этом каждая строка матрицы *X* является копией вектора *x*, а каждый столбец матрицы *Y* является копией вектора *y*.

2. Рассчитать матрицу *Z* значений функции двух переменных в каждой узловой точке.

3. Применить одну из функций построения трехмерных графиков (табл. 4.2).

Таблица 4.2 – Функции для построения трехмерных графиков

Функция	Описание функции
mesh(x, y, z)	построение каркасной поверхности, координаты узловых точек которой содержатся в матрицах *x, y, z*
meshc(x, y, z)	построение каркасной поверхности и линий уровня функции, спроектированных на плоскость *XY*
meshz(x, y, z)	построение каркасной поверхности и перпендикуляров, опущенных из граничных точек поверхности на плоскость *XY*
surf(x, y, z)	построение закрашенной поверхности, координаты узловых точек которой содержатся в матрицах *x, y, z*
surfc(x, y, z)	построение закрашенной поверхности и линий уровня функции, спроектированных на плоскость *XY*
contour(x, y, z)	построение линий уровня поверхности, координаты узловых точек которой содержатся в матрицах *x, y, z* . Если параметры, возвращаемые функцией *contour,* передать в функцию *clabel: clabel(contour(x, y, z))* на график будут нанесены значения функции, соответствующие линиям уровня.
contourf(x, y, z)	построение линий уровня поверхности с закрашиванием промежутков между ними разными цветами
contour3(x, y, z)	построение линий уровня поверхности в пространстве

Для управления внешним видом построенного графика используются различные функции, представленные в табл. 4.3.

Таблица 4.3 – Функции для изменения внешнего вида графика

Функция	Описание функции
hidden on (hidden off)	включение (отключение) режима удаления невидимых линий на графике
shading	устанавливают способ затенения поверхностей: *shading faceted* – равномерная раскраска ячеек с нанесением черных граней; *shading flat* – раскраска ячеек или граней в зависимости от цвета узла сетки; *shading interp* – раскраска цветом, определяемым интерполяцией цветов в узлах сетки
view	управление точкой обзора: *view (A, E)*– устанавливает азимут *A* и угол возвышения *E* (азимут – угол поворота вокруг оси *Z* от отрицательного направления оси *Y* против часовой стрелки, угол возвышения – угол между отрезком, направленным из начала координат в точку обзора и плоскостью *XY*); *view([x, y, z])* задает точку обзора с координатами *(x,y,z);* *view(2), view(3)* – задать положение точки обзора, принятое по умолчанию, для двухмерных и трехмерных графиков
colorbar	выводит вертикальный столбец с текущей цветовой палитрой и шкалой оси *Z*
colormap(MN)	изменить цветовую палитру на палитру с именем *MN.* Имена стандартных и дополнительных палитр: *cool* (оттенки голубого и пурпурного), *gray* (оттенки серого), *hot* (белый-желтый-оранжевый-красный-черный), *hsv* (цвета радуги), *jet* (красный-желтый-зеленый-голубой-синий, используется по умолчанию), *copper* (оттенки медного), *autumn* (желтый-оранжевый-красный), *winter* (синий-зеленый), *summer* (желтый-зеленый), *spring* (желтый-пурпурный), *pink* (оттенки коричневого). Возврат к палитре, используемой по умолчанию, осуществляется командой

Функция	Описание функции
	colormap('default').
camlight	создание источника света: *camlight (A, E)* – с заданными координатами *A* - азимут, *E* - угол возвышения; *camlight headlight* – в точке обзора; *camlight right* – правее и выше точки обзора; *camlight left* – левее и выше точки обзора
material	задание свойств материала поверхности: *material shiny* – блестящая; *material dull* – матовая; *material metal* – металлическая; *material default* – принятая по умолчанию
brighten(B)	изменяет яркость палитры: при *0<B<1* яркость увеличивается, при *-1<B<0* уменьшается

Продемонстрируем возможности трехмерной графики на примере построения графика функции двух переменных $z(x,y) = \sin(x) * \sin(y)$.Для построения графиков, приведенных на рис. 4.7, 4.8 применены следующие команды:

```
» x=-pi:0.1:pi; y=x; [x1, y1]=meshgrid(x,y); z=sin(x1).*sin(y1);
» subplot(2,2,1); mesh(x1, y1, z);
» subplot(2,2,2); meshz(x1, y1, z);
» subplot(2,2,3); surf(x1, y1, z); view ([0,-pi,0]); colorbar; view ([0,-pi,0])
» subplot(2,2,4); surfc(x1, y1, z);
» figure(2)
» subplot(2,1,1); clabel(contour(x1, y1, z));
» subplot(2,1,2); contourf(x1, y1, z);
»
```

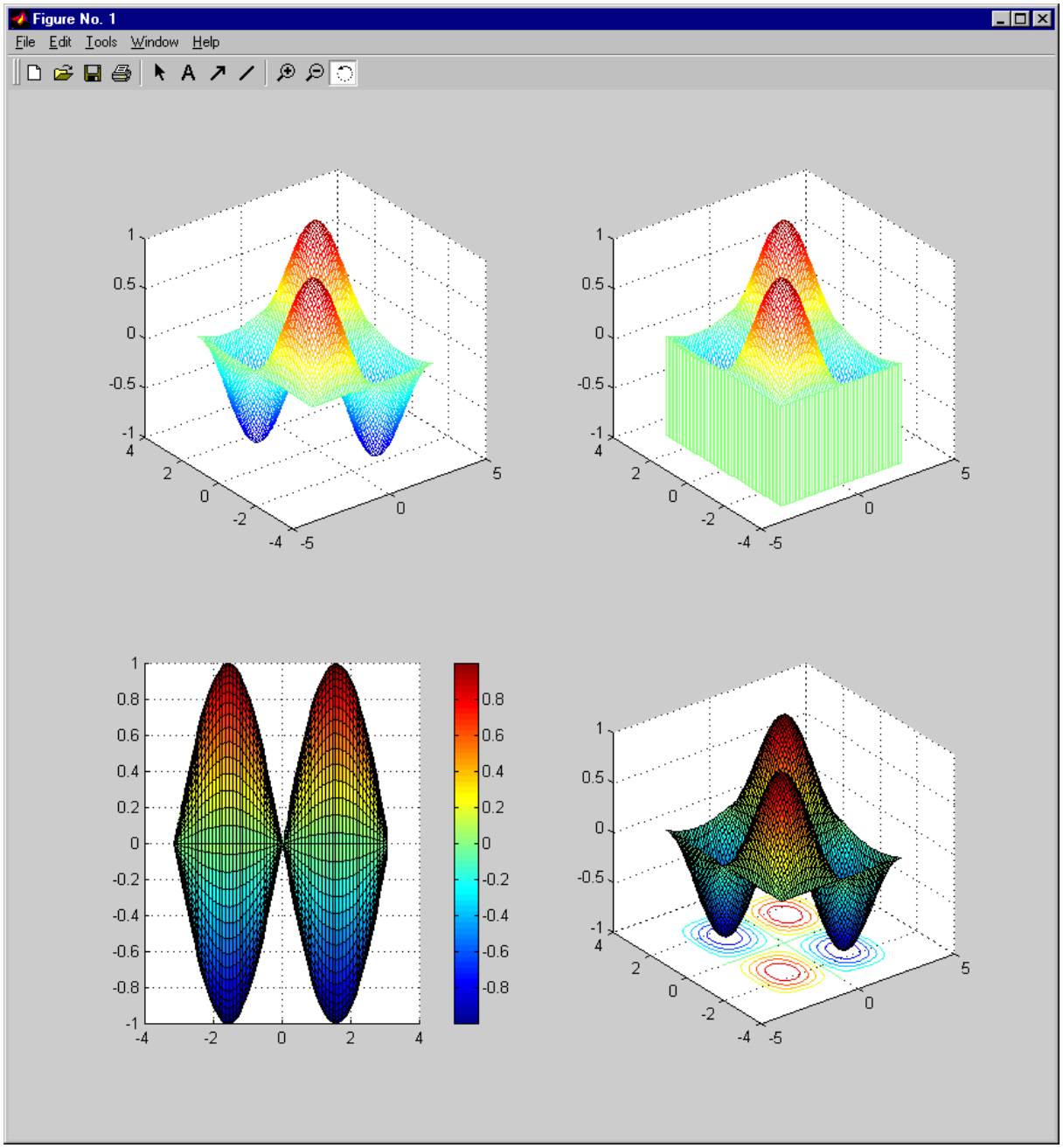


Рисунок 4.7 – Графики функции $z(x,y) = \sin(x) * \sin(y)$

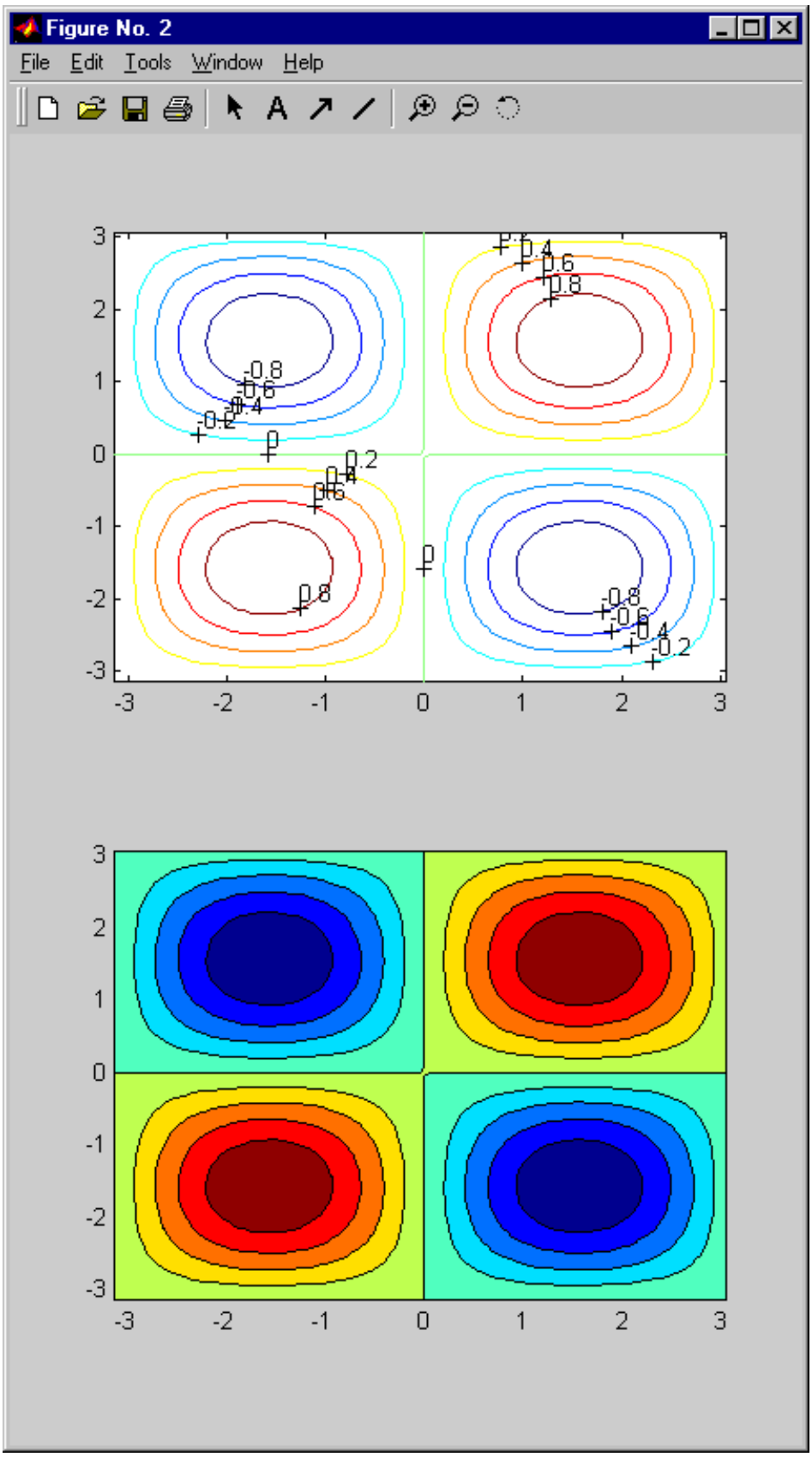


Рисунок 4.8 – Графики функции $z(x, y) = \sin(x) * \sin(y)$

4.5 Вывод в графическом окне надписей, заголовков, координатной сетки, легенды

Для отображения координатной сетки служит команда *grid on*. Убрать линии координатной сетки можно командой *grid off*. Команда *grid* выполняет функцию переключателя из одного состояния в другое.

В области графика вывод надписей (задаваемых при вызове функции в виде заключенной в апострофы текстовой переменной *SV*) осуществляется функциями:

xlabel(SV) – название оси x;

ylabel(SV) – название оси y;

zlabel(SV) – название оси z;

title(SV) – заголовок графика, размещаемый над графиком;

text(X, Y, Z, SV) - текст, начиная с точки с координатами *(X, Y, Z)*. Координаты точки задаются в той системе координат, в которой построен график. Для графика на плоскости координата *Z* отсутствует;

gtext(SV) – команда высвечивает перекрестие, перемещение которого позволяет место вывода строки *SV*. Для завершения ввода необходимо нажать любую клавишу или щелкнуть мышью.

legend(SV1, SV2, SV3 ...pos) – добавляет в области графика легенду, в которой отображается тип линии или маркера и поясняющий текст к каждому графику, записанный в текстовых переменных *SV1, SV2, SV3*. Параметром *pos* задается местоположение легенды:

Значение *pos*	Местоположение легенды
-1	*вне графика справа*
0	*в области графика так, чтобы легенда скрывала минимальное количество точек графика*
1	*в правом верхнем углу графика*
2	*в левом верхнем углу графика*
3	*в левом нижнем углу графика*
4	*в правом нижнем углу графика*

Для написания текста с буквами греческого алфавита и другими символами во всех строковых переменных, выводимых на экран, может использоваться синтаксис языка *TeX*.

Команда *box on (box off)* применяется для изображения (удаления) контура параллелепипеда, в котором размещается трехмерный объект. Команда *box* выполняет роль переключателя из одного состояния в другое.

4.6 Изменение масштаба графика

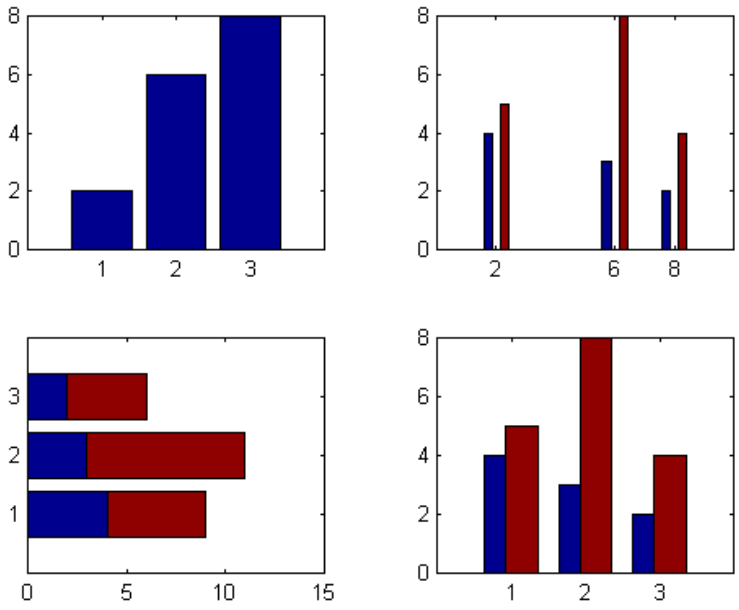
Команда *axis([xmin, xmax, ymin, ymax])* устанавливает масштаб по осям *x, y* двумерного графика, а команда *axis([xmin, xmax, ymin, ymax, zmin, zmax])* – по осям *x, y, z* трехмерного графика. Параметры *xmin, ymin, zmin* задают минимальные значения, *xmax, ymax, zmax* – максимальные значения соответственно по осям *x, y, z*. Команда *axis* без параметров возвращает вектор с предельными значениями по всем осям: *[xmin, xmax, ymin, ymax, zmin, zmax]).*

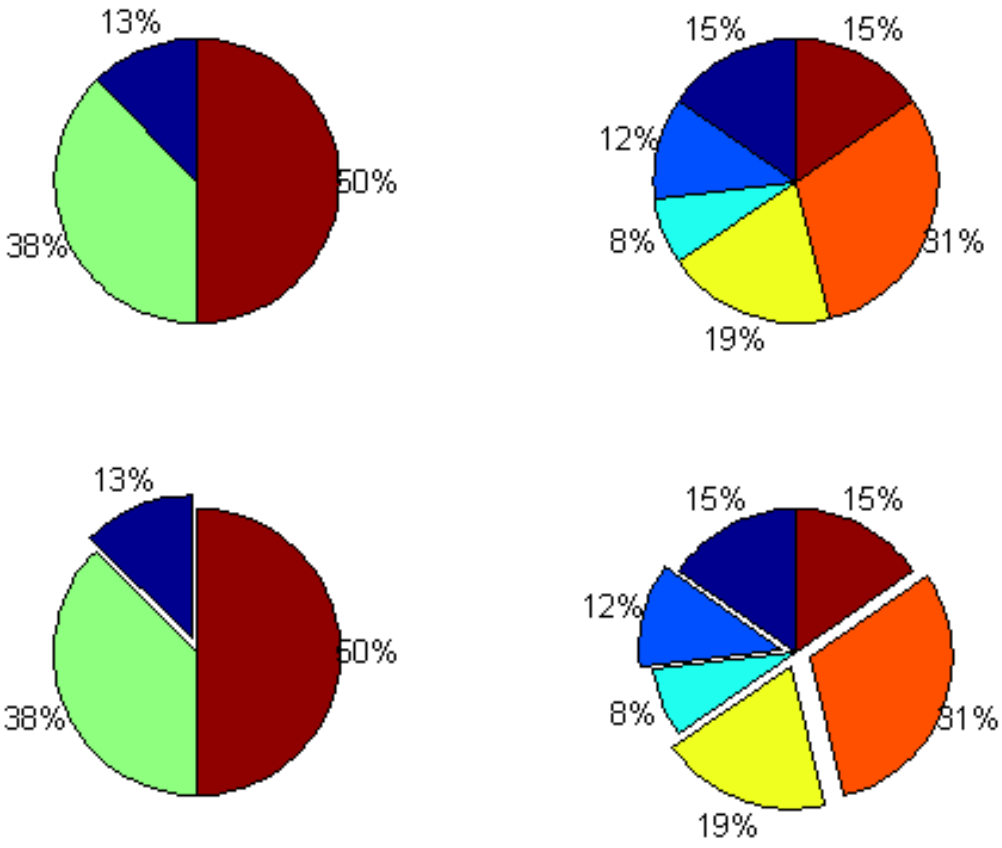
Команда *axis('auto')* задает автоматическую установку предельных значений по всем осям, команда *axis('auto x')* – по оси *x*, команда *axis('auto xy')* – по осям *x, y*. Команда *axis equal* устанавливает одинаковые масштабные коэффициенты по всем осям.

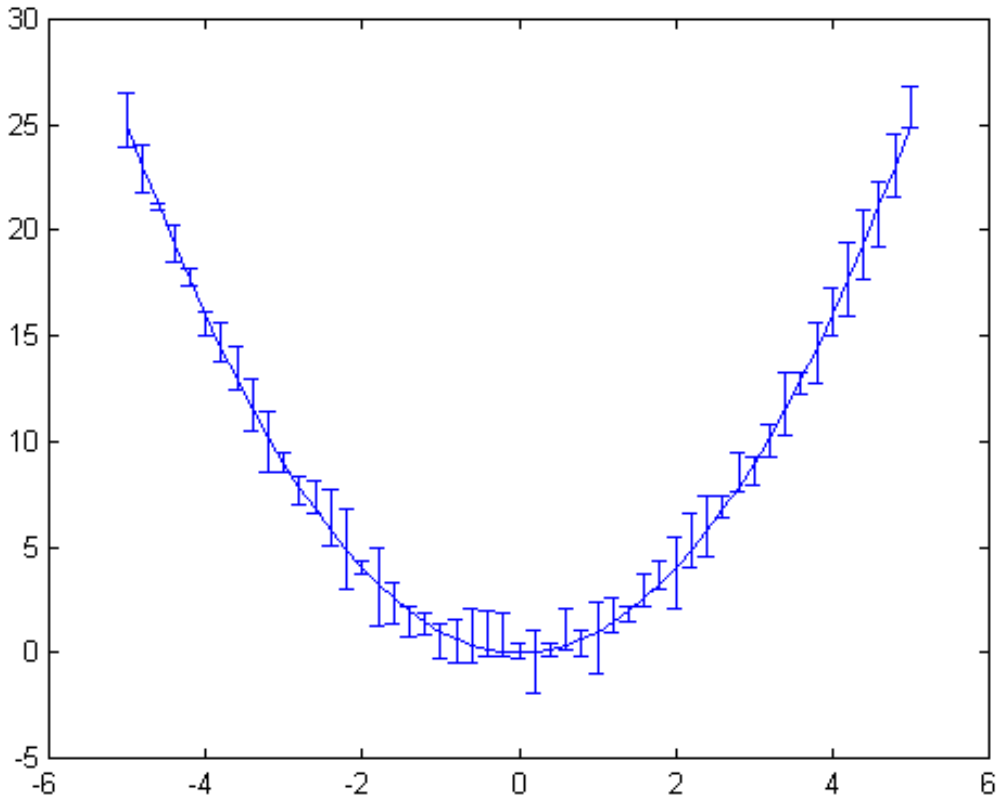
4.7 Специальная графика

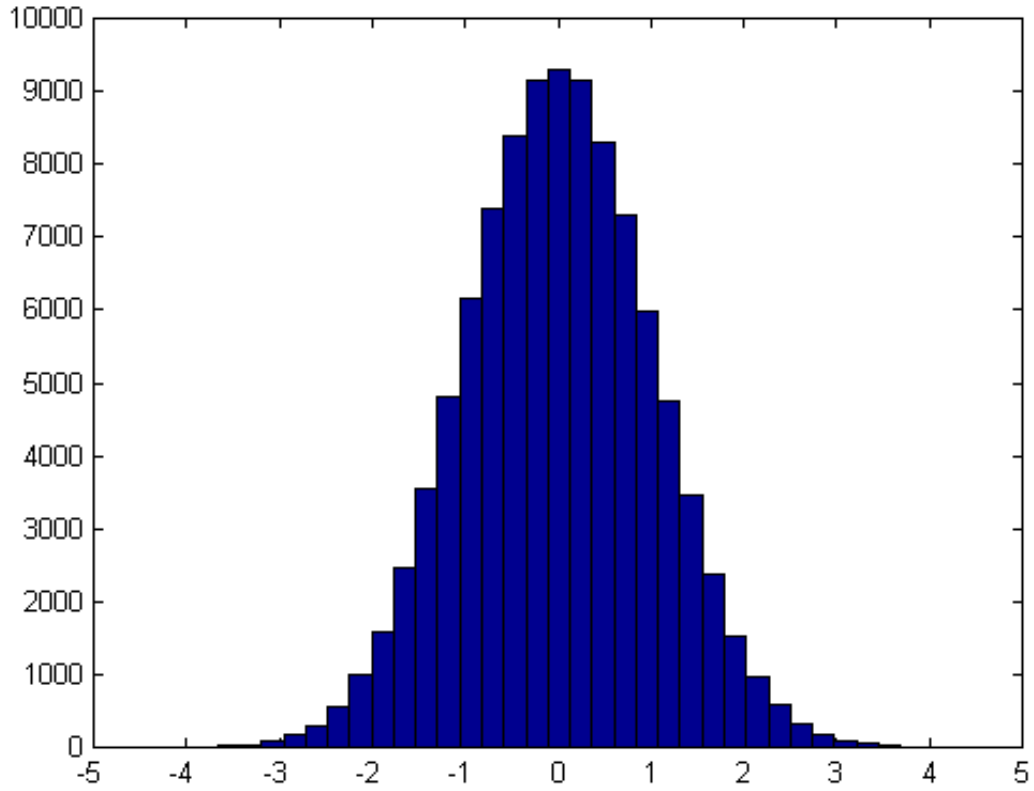
Функции специальной двумерной графики приведены в табл. 4.4.

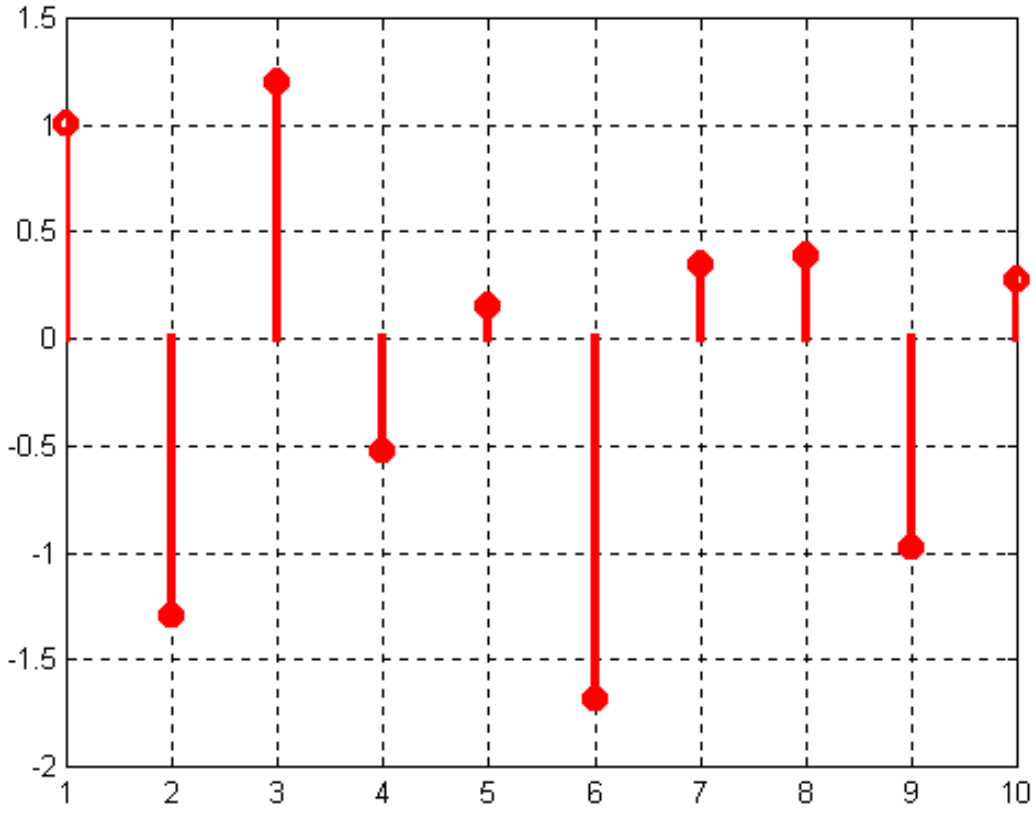
Таблица 4.4 – Специальная двумерная графика

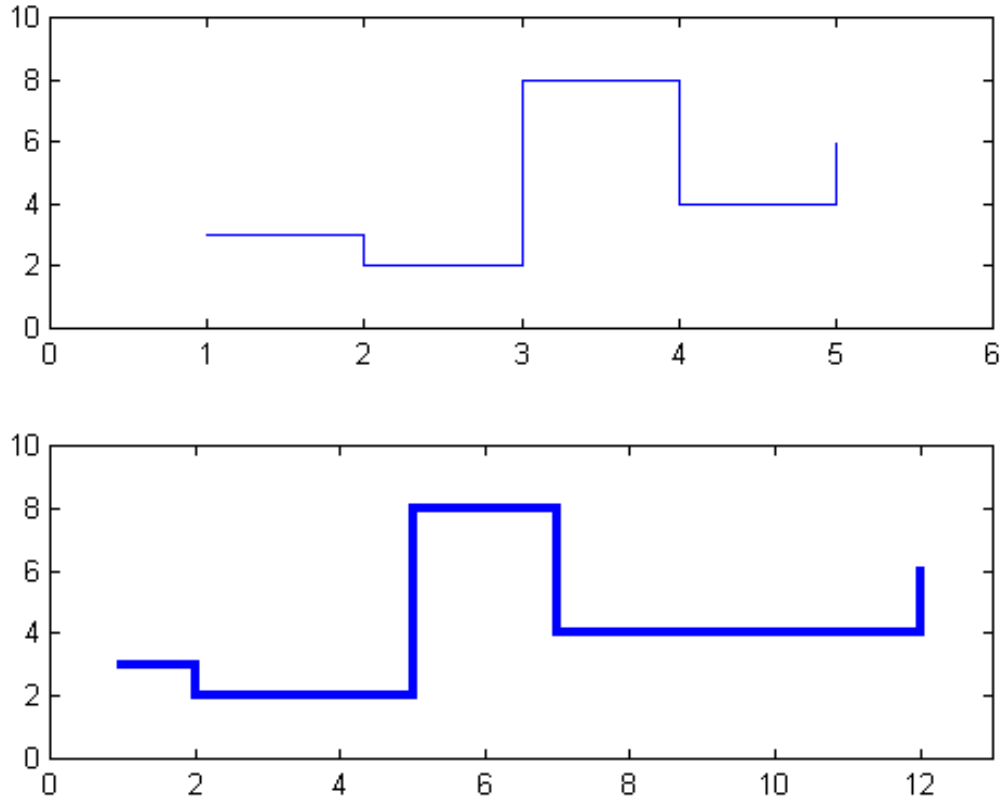
Функция	Описание
bar (Y,h,s) *bar (X,Y,h,s)*	построение вертикальных (*bar*) и горизонтальных (*barh*) столбцовых диаграмм с относительной шириной столбца *h* (по умолчанию *h=0.8*) и стилем *s* (ширина столбца и стиль при вызове функции могут не задаваться): *bar (Y,h,s)* - если – *Y* - одномерный массив, каждому элементу вектора соответствует столбец диаграммы, если *Y* – матрица, выводятся группы столбцов, соответствующие значениям элементов строки (а ось *x* размечается метками, соответствующими номеру строки); *bar (X,Y,h,s)* – если *Y* - одномерный массив, выводит элементы вектора *Y*, в позициях, определяемых значениями элементов вектора *X*, если *Y* –матрица, в позициях, определяемых значениями элементов вектора *X* размешаются группы столбцов, соответствующие элементам строк матрицы *Y* возможные стили: *'group'* – диаграмма строится в виде групп столбцов (этот стиль используется по умолчанию), *'stack'* – каждой строке матрицы *Y* соответствует один составной столбец, длина каждой части которого определяется значением соответствующего элемента массива *Y* Пример: *» x=[2 6 8];y=[4 5; 3 8; 2 4]; subplot(2,2,1); bar(x); subplot(2,2,2); bar(x,y,0.5)* *» subplot(2,2,3); barh(y,'stack'); subplot (2,2,4); bar(y,1.5)* 

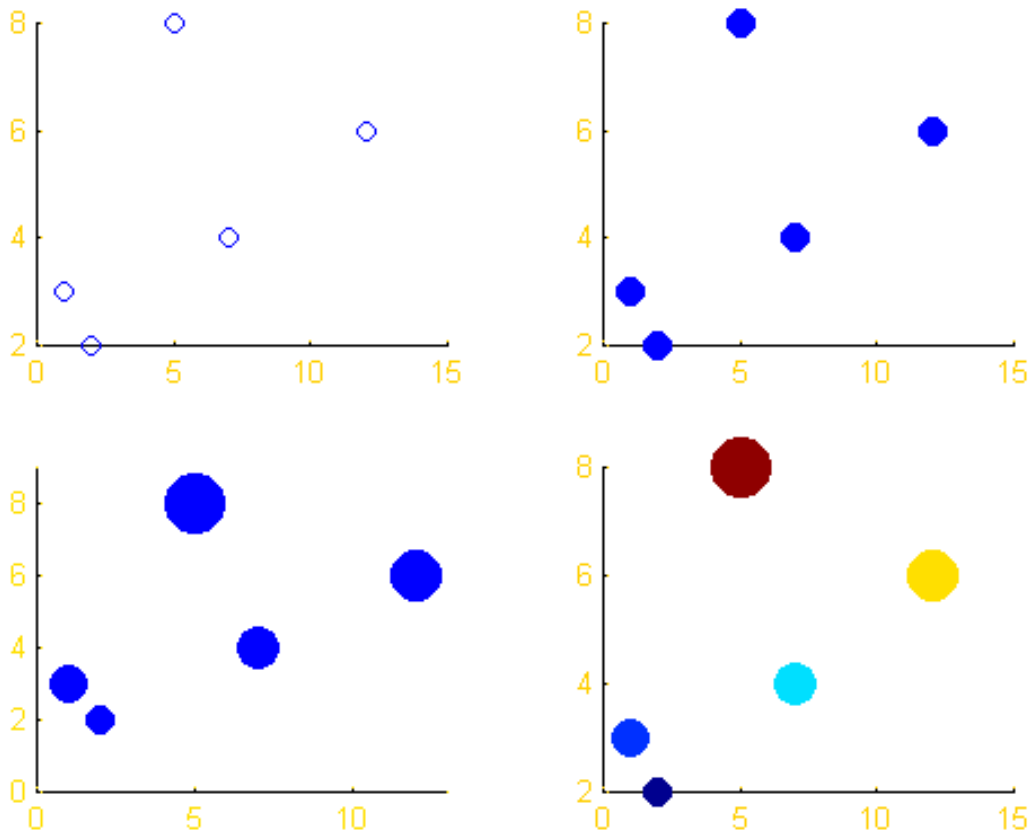
Функция	Описание
pie	построение круговой диаграммы: *pie(x)* – отображает каждый элемент массива *x* в виде сектора круговой диаграммы. *pie(x, v)* – на круговой диаграмме дополнительно отделяются секторы, которым соответствуют ненулевые элементы массива *v* (массив *v* должен быть того же размера, что и массив *x*). Пример: *» x=[2 6 8];y=[4 5; 3 8; 2 4]; subplot(2,2,1); pie(x); subplot(2,2,2); pie(y)* *» c=[1 0 0]; d=[0 1; 1 1; 0 0];subplot(2,2,3); pie(x,c); subplot (2,2,4); pie(y,d)* 
errorbar	Построение графика с указанием интервала погрешности. Функции *errorbar(y,e)* и *errorbar(x,y,e)* предназначены для построения графика с погрешностью *±e* относительно каждой точки графика Функция *errorbar(x,y,L,U)* строит график зависимости *y* от *x* с

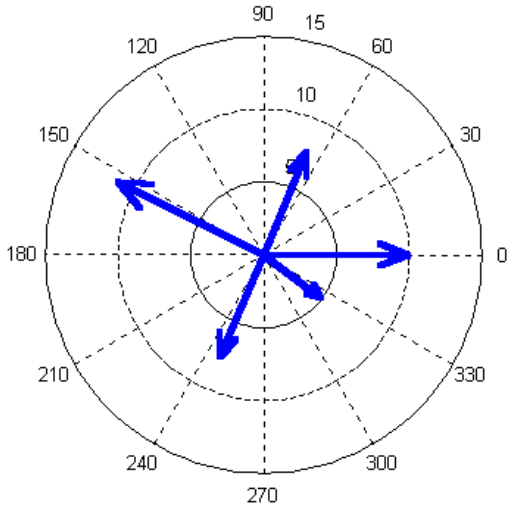
Функция	Описание
	погрешностью *–L* и *+U* относительно каждой точки графика Пример: *» x=-5:0.2:5; y=x.^2; l=rand(1,length(y))*2;u=rand(1,length(y))*2;* *» errorbar(x,y,l,u)* 

Функция	Описание
hist	рассчитывает и строит гистограмму. *hist(y)* - строит гистограмму для*10* интервалов, *hist(y,n)* – строит гистограмму для *n* интервалов Пример: *» x=randn(100000,1);* *» hist(x,40)* 
stem	*stem(y,LS)* - построение графика элементов одномерного массива *y* в зависимости от индекса в виде вертикальных линий, которые заканчиваются маркером. Строковая переменная *LS* задает параметры линии. *stem(x,y,LS)* - построение графика элементов одномерного массива *y* в зависимости от значений элементов вектора *x*. Пример: *» x=randn(10,1); k=stem(x,'r'); grid on; set(k,'LineWidth',3)*

Функция	Описание
	
stairs	*stairs(Y,LS)* – строит ступенчатый график, ординатами которого являются значения элементов одномерного массива *Y*, а абсциссами – их индексы. Строковая переменная *LS* задает параметры линии; *stairs(X,Y,LS)* - абсциссами ступенчатого графика являются значения элементов массива *X*, а ординатами - элементы массива *Y*. Пример: *» x=[1 2 5 7 12];y=[3 2 8 4 6];* *» figure(2);subplot(2,1,1);stairs(y);axis([0,6,0,10]);* *» subplot(2,1,2);h=stairs(x,y);set(h,'LineWidth',3); axis([0,13,0,10]);*

Функция	Описание
	
scatter	построение группы точек на плоскости, абсциссы и ординаты которых записаны в векторы *X* и *Y*: *scatter(X,Y)* – в виде окружностей синего цвета; *scatter(X,Y, S)* – размеры окружностей задаются вектором *S*; *scatter(X,Y, S, C)* – цвета окружностей задаются вектором *C*; *scatter(… , 'filled')* – окружности закрашены. Пример: *» x=[1 2 5 7 12];y=[3 2 8 4 6];* *» figure;* *» subplot(2,2,1); scatter(x,y)* *» subplot(2,2,2); scatter(x,y,100,'filled')* *» subplot(2,2,4); scatter(x,y,50*y,y,'filled')* *» subplot(2,2,4); scatter(x,y,50*y,y,'filled')* *» subplot(2,2,3); scatter(x,y,50*y,'filled')* *» axis([0,13,0,9])*

Функция	Описание
	
compass	*compass(Z, LS)* - строит векторы, являющиеся геометрической интерпретацией комплексных чисел, записанных в массиве *Z*; *compass(X,Y, LS)* – равносильна *compass(X+j*Y);* Необязательный параметр *LS* является строковой переменной и задает параметры линии. Пример: *» z=[3+7j 4- 3j -10+5j -3-7j 10];* *» h=compass(z); set(h,'LineWidth',3)*

Функция	Описание
	
comet	Отображает движение точки по траектории в виде головы и хвоста кометы: *comet(Y)* - траектория задана одномерным массивом *Y* (абсциссы точек, принадлежащих траектории равны индексам элементов массива *Y*, ординаты – значению); *comet(X,Y)* - траектория задана одномерными массивами *X* и *Y*; *comet(X,Y,P)* – параметром *P* задается относительная длина «хвоста» кометы (длина «хвоста» равна *P*length(Y))*. Пример: комета движется по спирали. *» f=0:0.001:10*pi; r=f*10; x=r.*cos(f); y=r.*sin(f);* *» figure; comet(x,y)* Функция *comet3(X, Y, Z, P)* рисует движение точки по пространственной траектории, заданной одномерными массивами *X, Y, Z*. Параметром *P* задается относительная длина «хвоста» кометы.

4.8 Дополнительные примеры

Пример 1. Составить функцию для построения графика функции

$$y(x)=\begin{cases} 50\,\dfrac{\sin(x)^2}{x} & \text{при} \quad x<-4 \\ \dfrac{10\cdot e^{x}}{x^2+1} & \text{при} \quad -4<x<4 \\ x^2/5 & \text{при} \quad 4<x<10 \\ x & \text{при} \quad 10<x \end{cases}$$

на заданном интервале *[xn; xk]* (рассчитав ее значения с шагом *dx*).

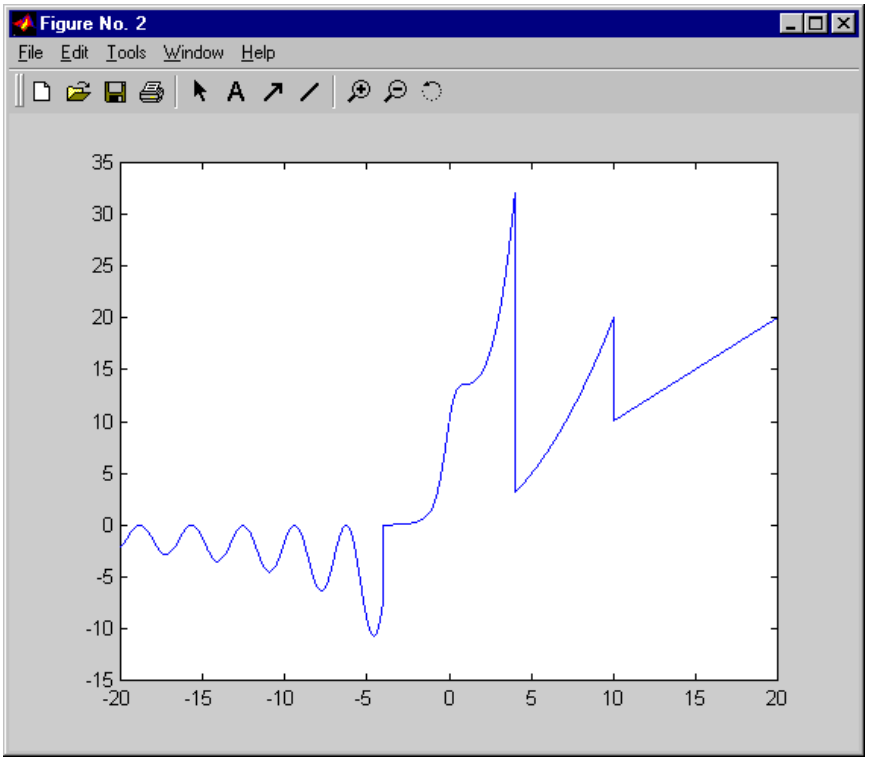
Текст функции:

```
function v=Vg(xn,dx,xk)
x=xn:dx:xk;

y=(x<=-4).*(50*sin(x).^2./x)+(x>-4&x<=4).*(10*exp(x)./(x.^2+1))...
  +(x>4&x<=10).*(x.^2/5)+(x>10).*x;

plot(x,y)
```

Вызов функции:

```
» Vg(-20,0.0001,20)
```

Результат:



Пример 2. Определить количество точек пересечения функции $y(x)$ из предыдущего примера и функции $f(x)=x^3-7x+6$, границы интервалов, содержащие их. Построить окружности с центрами в

серединах найденных интервалов. Используя комету обвести часть плоскости, ограниченную графиками заданных функций и найденными точками пересечения. Движение кометы начать и закончить во второй точке пересечения: обвести первую область по ходу часовой стрелки, вторую – против хода часовой стрелки.

Функция:

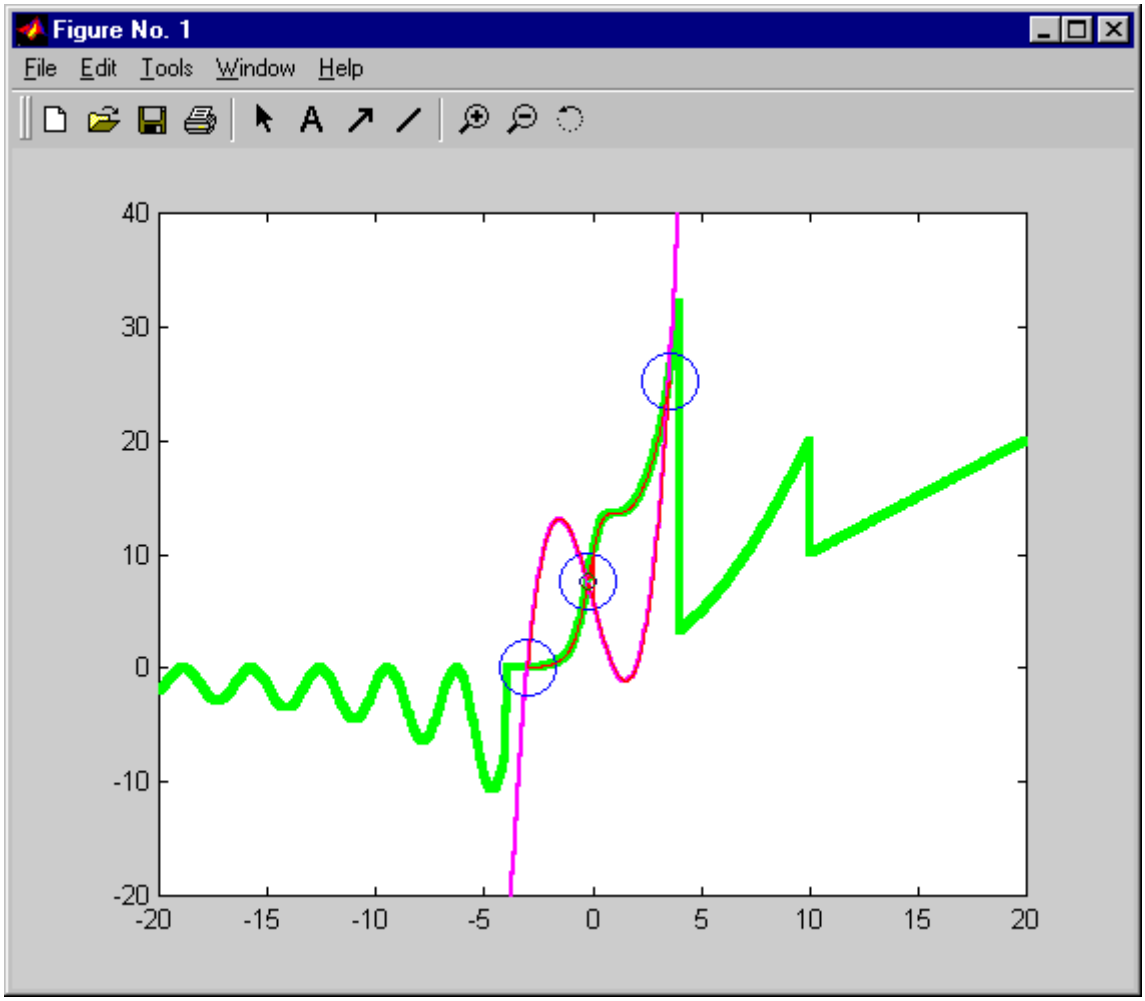
```
function l4_p2_3(xn,xk,dx)
%лекция 4 пример 2
% расчет и построение графиков
hold off
x=xn:dx:xk;
y1=f1(x);
y2=f2(x);
plot(x,y1,'g','LineWidth',3)
hold on
plot(x,y2,'m','LineWidth',2)
axis([-20 20 -20 40]) % изменяем масштаб

% приближенное определение координат точек пересечения графиков
y1_y2=y1-y2;
r=[x(1:end-1); y1_y2(1:end-1).*y1_y2(2:end)<0];
rt=r';
rs=flipud(sortrows(rt,2));
nt=sum(rs(:,2))
xn=rs(1:nt,1)
xk=xn+dx
xr=xn+dx/2;
xr=sort(xr);
yr=f1(xr);
% строим окружности с центрами в приближенно найденных точках пересечения графиков
scatter(xr,yr,dx*10000)

dx=dx/2; %уменьшаем шаг
```

```
% задаем координаты траектории кометы
xc1= xr(2):-dx:xr(1);
xc2=xr(1):dx:xr(2);
xc3=xr(2):dx:xr(3);
xc4=xr(3):-dx:xr(2);
xc=[xc1 xc2 xc3 xc4];
yc=[min([f1(xc1);  f2(xc1)])  max([f1(xc2);  f2(xc2)])  min([f1(xc3);
f2(xc3)]) max([f1(xc4); f2(xc4)]) ];
comet(xc,yc) % движение кометы по траектории
function y=f1(x)
y=(x<=-4).*(50*sin(x).^2./x)+...
  (x>-4&x<=4).*(10*exp(x)./(x.^2+1))+...
  (x>4&x<=10).*(x.^2/5)+...
  (x>10).*x;

function y=f2(x)
y=x.^3-7*x+6;
```

Результат (без предупреждений о делении на ноль):

```
» l4_p2_3(-20,20,0.05)
nt =
     3
xn =
    3.5000
   -0.2500
   -3.0000
xk =
    3.5500
   -0.2000
   -2.9500
```

Полученный график:



Пример 3. Случайным образом задать *n* точек на плоскости с абсциссами от *x1* до *x2* и ординатами от *y1* до *y2*. Определить количество точек, принадлежащих прямоугольнику (включая его границу), координаты левого верхнего угла которого (*xp1, yp1*), правого нижнего угла (*xp2, yp2*). Построить прямоугольник, точки, принадлежащие ему – в виде кругов одного цвета, не принадлежащие – в виде кругов другого цвета меньшего радиуса.

Функция:

```
function nt=l4_p3(x1,x2,y1,y2,n,xp1,yp1, xp2,yp2)
hold off

xp=[xp1,xp2,xp2,xp1 xp1];
yp=[yp1, yp1,yp2,yp2, yp1];
plot(xp,yp)%построение прямоугольника
hold on
```

```
axis([x1,x2,y1,y2])
grid on

x=fv(x1,x2,n);
y=fv(y1,y2,n);

r=x>=xp1&x<=xp2&y>=yp2&y<=yp1;% принадлежность точки прямоугольнику
xy=[x y r]
nt=sum(xy(:,3)); % количество точек, принадлежащих прямоугольнику

scatter(xy(:,1),xy(:,2),50+50*xy(:,3),4+xy(:,3)*9,'filled') %построение точек

function k=fv(x1,x2,n)
k=x1+(x2-x1)*rand(n,1);
```

Результат:

```
» l4_p3(-10,10,-20,20,10,-3,8, 5,-2)
xy =
   6.5773   -5.3480         0
  -6.6746   -7.8986         0
  -2.1219   14.0738         0
   0.4151   10.3792         0
   4.3625   17.9904         0
   1.3838    2.3175    1.0000
  -0.7839  -19.4307         0
  -1.0939    3.8471    1.0000
  -8.2451   12.6482         0
  -1.1303   19.0837         0
ans =
     2
»
```

График:

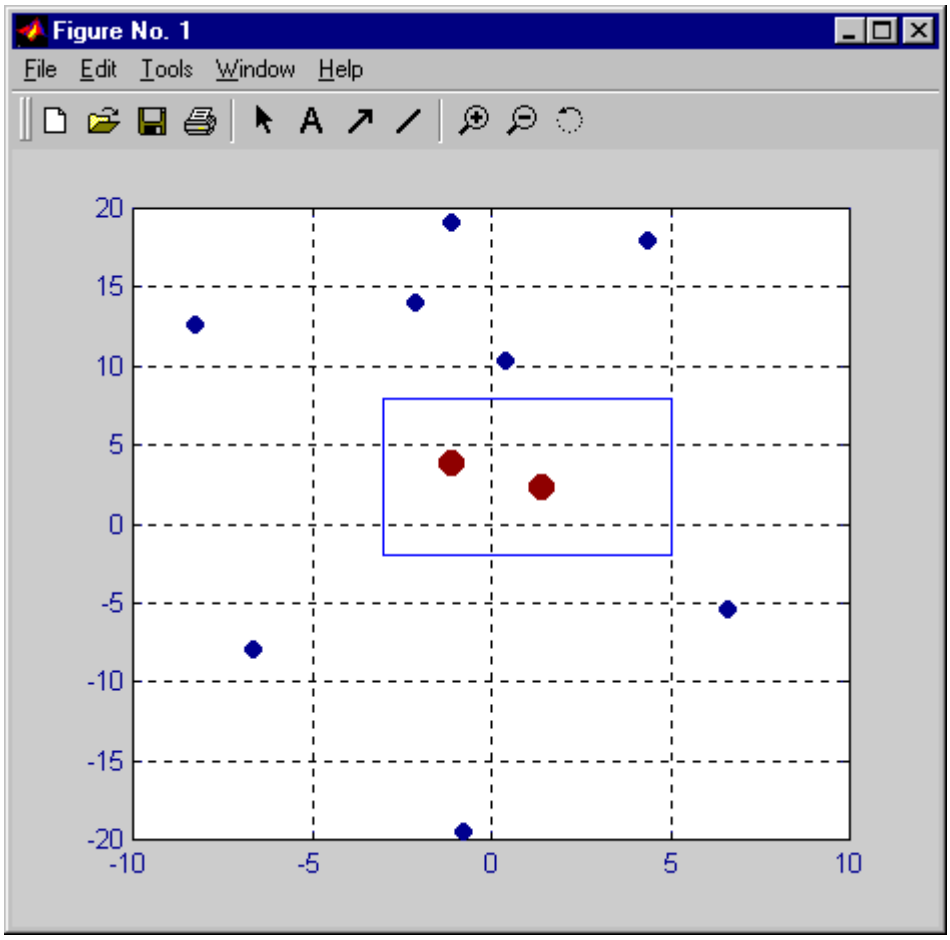


Figure No. 1
File Edit Tools Window Help
20
15
10
5
0
-5
-10
-15
-20
-10
-5
0
5
10

5 ПРОГРАММИРОВАНИЕ

5.1 Функции ввода и вывода

Функция

disp(имя переменной)

используется для вывода на экран значения переменной без указания ее имени. Аргументом функции *disp* может быть и текстовая строка, заключенная в апострофы:

disp(‘ символьная строка ‘)

После каждого вызова функции *disp* происходит переход на новую строку.

Функция *input* предназначена для вывода на экран запроса и ввода значения одной переменной или допустимого в системе MATLAB выражения (вводимое выражение может содержать вызовы функций), которое она возвращает. Вызов функции записывается следующим образом

имя переменной =*input*(‘приглашение’)

Возвращаемое функцией значение присваивается переменной, имя которой задано при ее вызове (при вводе массива следует его элементы заключить в квадратные скобки, в конце строк матриц вводить «;»). Например, текст сценария, сохраненного в файле *inp.m*:

```
x=input('введите значение переменной x ')
s=input('введите значение переменной s ')
v=input('введите вектор v ')
h=input('введите матрицу h ')
```

Результат:

```
» inp
введите значение переменной x 4
x =
     4
введите значение переменной s sin(0.3)/2+x^2-7
s =
    9.1478
введите вектор v [3 5 1]
v =
```

```
    3     5     1
введите матрицу h [2 4; 7 8]
h =
    2     4
    7     8
»
```

Если вызов функции *input* записывается так

имя переменной =*input*(‘приглашение’, ‘s’)

она возвращает строку, введенную пользователем. Например:

```
» h=input('Введите Ваше имя ','s')
Введите Ваше имя МАРАТ
h =
МАРАТ
»
```

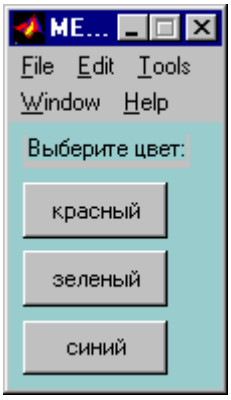
Функция *menu*, вызов которой записывается следующим образом

имя переменной=*menu* (‘ заголовок меню ’, ‘ выбор 1’, ‘ выбор 2 ’, … ,’ выбор n ’)

выводит на экран меню с заголовком и кнопками выбора, функция возвращает номер выбранной кнопки (в списке, заданном при вызове функции). Например:

```
» k=menu('Выберите цвет:','красный','зеленый','синий')
k =
    3
»
```

При выполнении функции на экране появилось меню:



после щелчка на кнопке «синий» функция вернула значение 3.

5.2 Операторы цикла

В MATLAB имеется два оператора цикла:

- цикл *for* служит для организации выполнения тела цикла фиксированное число раз,
- цикл *while* выполняет группу операторов в зависимости от значения логического выражения. Операторы цикла могут применяться при выполнении расчетов в командном окне, в сценариях и функциях.

5.2.1 Оператор цикла for

Оператор цикла

for [*параметр цикла*] = [*начальное значение*] : [*шаг*] : [*конечное значение*]

[операторы (тело цикла)]

end

выполняет тело цикла, пока параметр цикла принимает значения от начального до конечного с заданным шагом. Если шаг равен единице, его можно не указывать. Операторы, принадлежащие телу цикла, могут располагаться в разных строках или в одной строке (в этом случае их необходимо разделять запятой или точкой с запятой).

Например, составим функцию для определения суммы и произведения чисел в заданном диапазоне с заданным шагом:

```
function [s,p]=sp(in,ik,di)
s=0;
p=1;
for i=in:di:ik
   s=s+i;
   p=p*i;
end
```

Результат:

```
» [S,P]=sp(3,15,2)
S =
   63
P =
    2027025
»
```

Для задания значений параметра цикла можно использовать вектор-строку:

for параметр цикла = *массив A*

операторы (тело цикла)

end

если *A* – вектор-строка, параметр цикла последовательно принимает значения элементов вектора. В качестве примера составим функцию для определения суммы значений элементов вектора

```
function s=sv(v)
s=0;
for i=v
  s=s+i;
end
```

Результат:

```
» sv([4 6 3])
ans =
   13
»
```

Если *A* - матрица, параметром цикла будет каждый столбец матрицы *A*, цикл выполнится столько раз, сколько столбцов в матрице.

Составим функцию для определения суммы значений матрицы, использовав составленную ранее функции расчета суммы значений элементов вектора:

```
function s=sm(a)
s=0;
for i=a
   s=s+sv(i');%столбец транспонируем
end
```

Результат:

```
» f=[3 4; 5 7; 2 1]
f =
    3    4
    5    7
    2    1
» sm(f)
ans =
   22
```

5.2.2 Оператор цикла while

Оператор

while *логическое выражение*

операторы (тело цикла)

end

организует многократное выполнение тела цикла, пока логическое выражение истинно. Если значением логического выражения является массив, его значение истинно, если все элементы массива истинны (отличны от нуля) и массив не является пустым (*[]* – пустой массив).

Например, задание:

Составить функцию для выражения

$$\sum_{i=1}^{\infty}\frac{1}{i(i+1)}$$

с заданной точностью ε ($\varepsilon > 0$) при ε *=0,01 і 0,00000005*. Текст функции (кроме суммы функция возвращает количество итераций):

```
function [s,i]=sr1(eps)
s=0;
i=0;
y=1;
while abs(y)>eps
   i=i+1;
   y=1/i/(i+1);
   s=s+y;
end
```

Результат:

```
» [s1,n1]=sr1(0.01)
s1 =
    0.9091
n1 =
    10
» [s2,n2]=sr1(0.0000000005)
s2 =
    1.0000
n2 =
      44721
```

Если не сформирован вектор, которому присваиваются возвращаемые значения, функция возвращает только первое из них:

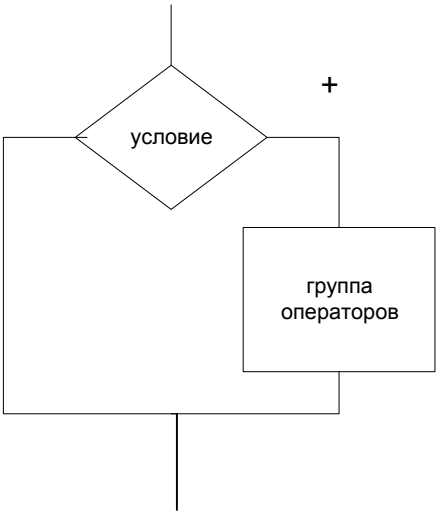
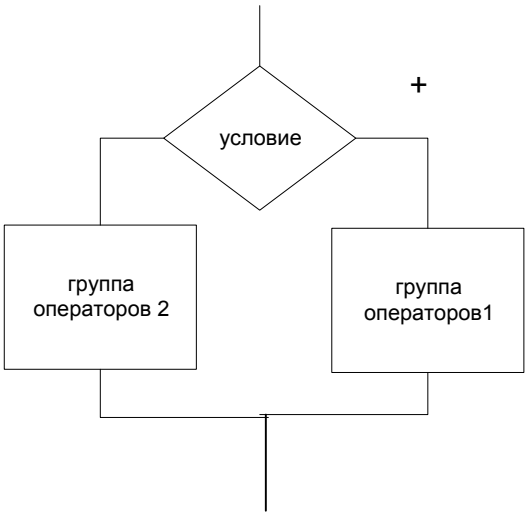
```
» sr1(0.0000000005)
ans =
    1.0000
```

5.3 Операторы ветвления

К операторам ветвления относятся условный оператор и оператор переключения.

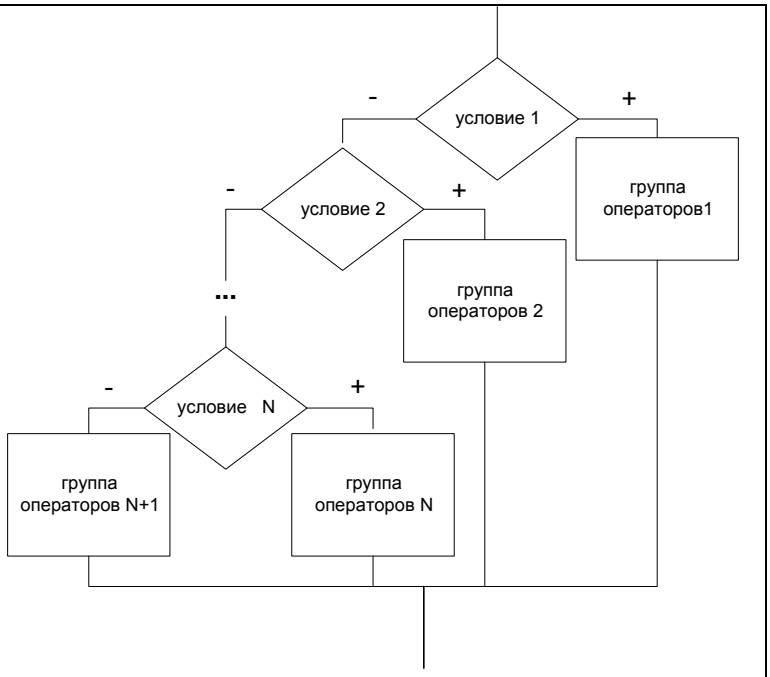
5.3.1 Условный оператор

Условный оператор может быть записан в одной из форм:

Синтаксис	Блок-схема
if *условие* *группа операторов* *End*	
if *условие* *группа операторов* 1 *else* *группа операторов* 2 *end*	
if *условие* 1 *группа операторов* 1 *elseif* *условие* 2 *группа операторов* 2 … *elseif* *условие N* *группа операторов N* *else*	

группа операторов $N+1$

end



В качестве условия может быть использовано любое выражение (как правило, используется логическое выражение). Условие считается истинным, если оно принимает ненулевое значение. Если в качестве условия используется массив, истинным оно будет, только если этот массив не пустой и все его элементы отличны от нуля.

Например, составим функцию *max* для определения наибольшего из трех чисел:

```
function max=l5_p1(a,b,c)
if a>b
  if a>c
    max=a;
  else
    max=c;
  end
elseif b>c
  max=b;
else
  max=c;
end
```

Результат:

```
» l5_p1(13, 4, 5)
ans =
   13
```

5.3.2 Оператор переключения

Оператор

switch выражение или переменная *B1*
case значение 1
группа операторов 1
case { значение 2, значение 3, значение 4, … }
группа операторов 2
…

otherwise
группа операторов *n*
end
оператор *Z*

Оператор организует разветвление в зависимости от значения переменной или выражения, записанного после ключевого слова *switch* (результатом должен быть скаляр или строка). Количество групп *case* произвольное. После ключевого слова case записывается возможное значение выражения или несколько значений, заключенных в фигурные скобки.

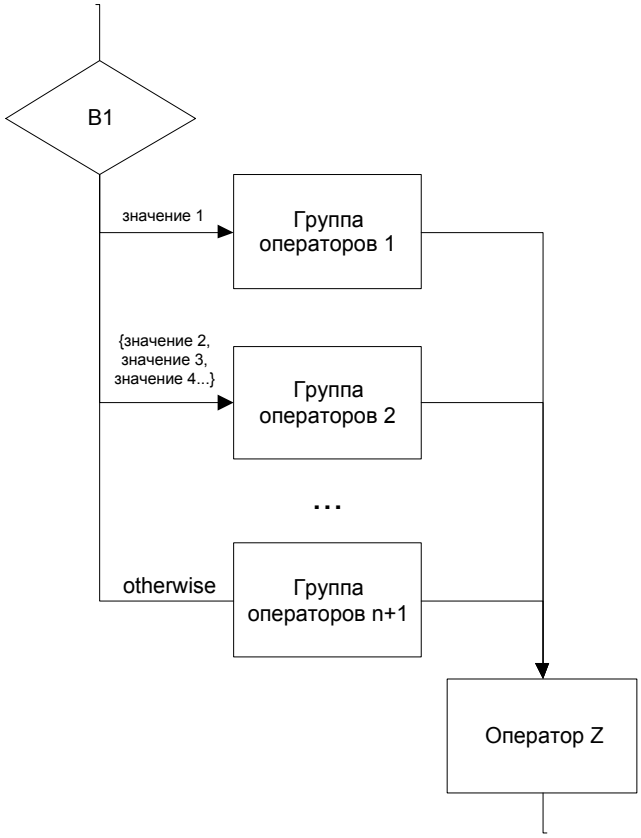
Оператор работает следующим образом:

- рассчитывается значение выражения, записанного после ключевого слова *switch*;
- если результат совпадает со значением, записанным после одного из ключевых слов *case*, выполняется группа операторов, следующая за ним. Далее происходит переход к оператору, следующему за ключевым словом *end*. Если после ключевого слова *case* записано несколько значений, заключенных в фигурные скобки, переход к соответствующей группе

операторов осуществляется, если значение выражения совпадает с одним из этих значений;

- если значение выражения не совпало ни с одним из значений, следующих за ключевыми словами *case*, выполняется группа операторов, записанная после *otherwise* (если эта ветвь существует).

Блок-схема:



Например, составим функцию для построения графика одной из функций: $\sin(x)$, x^2, e^x на заданном пользователем интервале. Текст функции *graph3*:

```
function graph3
v=menu('Выберите функцию','sin(x)','x^2', 'exp(x)');
xn=input('введите начальное значение аргумента');
xk=input('введите конечное значение аргумента');
dx=(xk-xn)/500;
x=xn:dx:xk;
switch v
case 1
  y=sin(x);plot(x,y)
```

```
case 2
  y=x.^2;plot(x,y)
case 3
  y=exp(x);plot(x,y)
otherwise %если после вызова меню значение переменной v будет изменено
  disp('неверный выбор')
end
```

Этот же пример со строковой переменной:

```
function graph4
v=input('введите функцию','s');
xn=input('введите начальное значение аргумента');
xk=input('введите конечное значение аргумента');
dx=(xk-xn)/500;
x=xn:dx:xk;
switch v
case 'sin(x)'
  y=sin(x);;
  plot(x,y)
case 'x^2'
  y=x.^2;
  plot(x,y)
case 'exp(x)'
  y=exp(x);
  plot(x,y)
otherwise
 disp('неверный выбор')
end
```

Пример организации выполнения разветвления в зависимости от выбранного пункта меню и возврата к меню до выбора выхода из него:

```
 function l5_p2
v=1
```

```
while v~=3
  v=menu('выбор','1','2','выход')
  switch v
  case 1
    disp (1)
  case 2
    disp(2)
  case 3
   disp( 'выход')
  end
end
disp('КОНЕЦ')
```

5.4 Другие операторы

Оператор *break* прерывает выполнение циклов *for* и *while*.

Пример:

```
for i=1:1:5
  for j=1:5
    if i==2& j==3
      break
    end
  end
   if i==2
   break
  end
end
i
j
```

Оператор *return* выполняет возврат в вызывавшую функцию или к режиму работы с клавиатурой.

```
function [i,j]=return1
for i=1:1:5
  for j=1:5
    if i==2& j==3
      i
```

```
      j
       return
     end
  end
end
```

Результат:

```
» [z,x]=return1
i =
    2
j =
    3
z =
    2
x =
    3
»
```

Команда *pause* используется для создания паузы при выполнении программы до нажатия любой клавиши. Команда *pause(n)* приостанавливает работу системы на n секунд. Команда *pause off* отключает работу пауз, а *pause on* включает.

Пример:

```
function pause1

xn=input('введите начальное значение аргумента');
xk=input('введите конечное значение аргумента');
dx=(xk-xn)/500;
x=xn:dx:xk;
for v=1:3
  subplot(2,2,v)
   switch v
   case 1
     y=sin(x);
     plot(x,y)
   case 2
     y=x.^2;
```

```
    plot(x,y)
  case 3
    y=exp(x);
    plot(x,y)
  end
  pause
end
```

5.5 Примеры

Пример 1. На плоскости заданы *n* точек. С некоторого момента точка с наименьшей массой исчезает, передавая свою массу ближайшей к ней точке. Так продолжается до тех пор, пока не останется одна точка. Определить координаты этой точки.

Тексты функций:

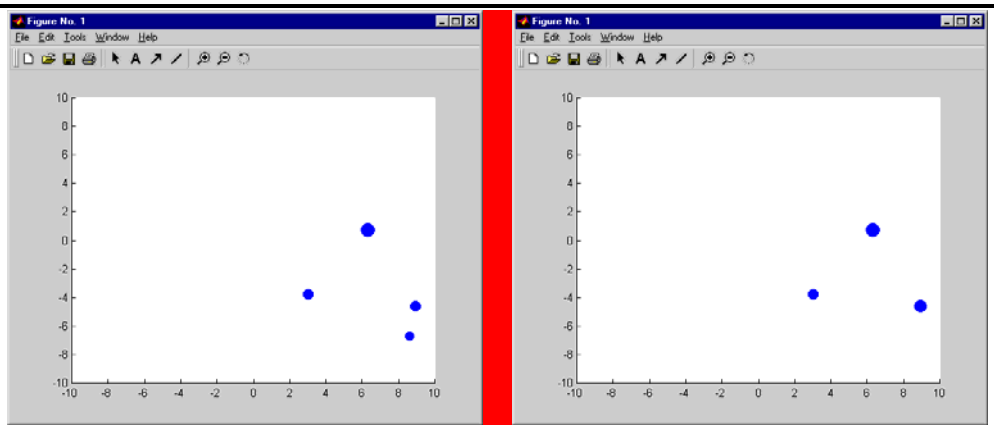
```
function l5_p3_s_1(n,x1,x2,y1,y2,m1,m2)

x=fv(x1,x2,n);
y=fv(y1,y2,n);
m=fv(m1,m2,n);
[X,Y]=l5_p3_1(x,y,m,x1,x2,y1,y2)

function k=fv(x1,x2,n)
k=x1+(x2-x1)*rand(n,1);
```

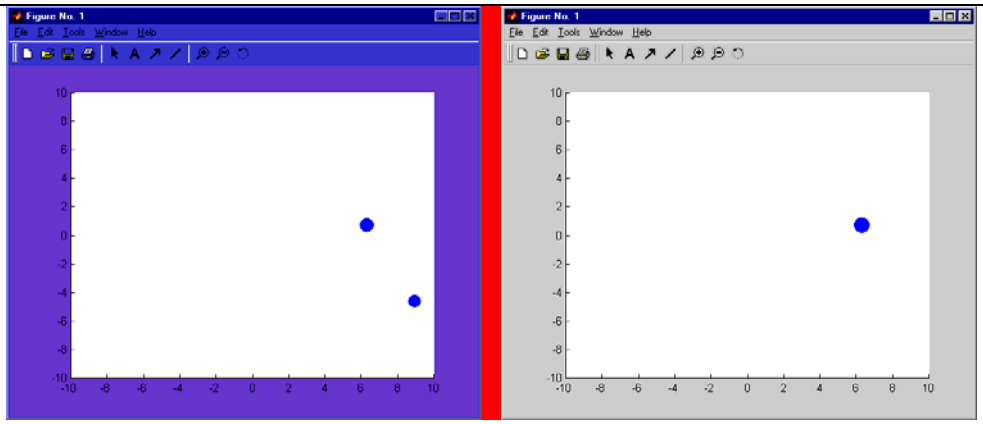
```
function [X,Y]=l5_p3_1(x,y,m,xn,xk,yn,yk)

a=[x y m];
s=size(a);
scatter(x,y,sqrt(m/pi)*100,'filled')
axis([xn,xk,yn,yk])
for i=1:s(1)-1
  [min,imin]=min(a(:,3));
  b=a(imin,:);

  pause
```

```
   a(imin,:)=[];

   a(:,4)=sqrt( (a(:,1)-b(1)).^2+ (a(:,2)-b(2)).^2 );
   [rmin,ib]=min(a(:,4));
   a(ib,3)=a(ib,3)+b(3)
   scatter(a(:,1),a(:,2),sqrt(a(:,3)/pi)*100,'filled')
   axis([xn,xk,yn,yk])
end
X=a(1);
Y=a(2);
```

Результат:

```
» l5_p3_s_1(4,-10,10,-10,10,1,10)
a =
   3.0310   -3.8012    2.8989    6.2986
   8.9217   -4.6237    4.4263    2.1344
   6.3177    0.7290    6.8661    7.8061
a =
   8.9217   -4.6237    4.4263    5.9479
   6.3177    0.7290    9.7650    5.5969
a =
   6.3177    0.7290   14.1913    5.9525
X =
   6.3177
Y =
   0.7290
»
```





Пример 2. Определить количество и индексы нулевых элементов исходной матрицы.

Функция (количество нулевых элементов, в массиве a –индексы нулевых элементов, в первом столбце – номера строк, во втором столбце – номера столбцов):

```
function [a,k]=l5_p4(b)
s=size(b);
k=0;
for i=1:s(1)
  for j=1:s(2)
    if b(i,j)==0
      k=k+1;
      a(k,1)=i;
      a(k,2)=j;
    end
  end
end
```

Результат:

```
» h=[4 0 -3; 6 8 0; 0 0 0],[R,K]=l5_p4(h)
h =
    4    0   -3
    6    8    0
    0    0    0
R =
    1    2
    2    3
```

```
    3     1
    3     2
    3     3
K =
    5
»
```

6 РЕШЕНИЕ СИСТЕМ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ В MATLAB

Дифференциальное уравнение – уравнение, содержащее неизвестную функцию (одной или нескольких переменных), независимые переменные, производные неизвестной функции по независимым переменным. Дифференциальное уравнение (содержащее функцию одной переменной) имеет вид

$$\frac{dy}{dt} = f(t, y).$$

Система дифференциальных уравнений *n*-го порядка имеет вид:

$$\begin{cases} y_1' = f_1(t, y_1, y_2 ... y_n) \\ y_2' = f_2(t, y_1, y_2 ... y_n) \\ \quad ... \\ y_n' = f_n(t, y_1, y_2 ... y_n) \end{cases},$$

где t - независимая переменная, как правило - время.

Для определения частного решения (задача Коши) системы дифференциальных уравнений должны быть заданы начальные условия: $y_1(t_0)$, $y_2(t_0)$, $y_3(t_0)$, $\dots y_n(t_0)$. Решение системы дифференциальных уравнений при заданных начальных условиях сводится к определению зависимостей $y_1(t)$, $y_2(t)$, $y_3(t)$, $\dots y_n(t)$.

Каждое дифференциальное уравнение *n*-го порядка, заданное в явной форме

$$y^{(n)}(t) = f(t, y, y', y'', ... y^{(n-1)})$$

путем введения новых неизвестных функций $y_1 = y$, $y_2 = y'$, $y_3 = y''$, $\dots y_n = y^{(n-1)}$ можно преобразовать в систему n дифференциальных уравнений:

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ \quad ... \\ y_{n-1}' = y_n \\ y_n' = f(t, y_1, y_2, ..., y_n) \end{cases}.$$

В MATLAB для решения систем дифференциальных уравнений численными методами предназначены функции *ode45, ode23, ode113,*

ode15s, ode23s, ode23t, ode23tb. Функции используют различные численные методы, например *ode45* - метод Рунге-Кутта 4-го и 5-го порядков, *ode23* - метод Рунге-Кутта 2-го и 3-го порядков. Функции *ode15s, ode23s, ode23t, ode23tb* предназначены для решения жестких систем дифференциальных уравнений. Жесткие – это системы дифференциальных уравнений, решения которых на различных интервалах изменения независимых переменных ведут себя по-разному: на одних участках наблюдается очень быстрое их изменение, на других – очень медленное.

Рассмотрим применение функций решения системы дифференциальных уравнений на примере функции *ode45*. Введем векторы $Y = [y_1, y_2, \ldots y_n]$ и $F = [f_1, f_2, \ldots f_n]$. Тогда система дифференциальных уравнений принимает вид

$$Y' = F(t, Y).$$

Для применения функции решения системы дифференциальных уравнений необходимо составить специальную М-функцию, вычисляющую правую часть системы уравнений ($F(t, Y)$).

Вызов функции записывается следующим образом:

[t,Y]=ode45('имя функции',ts,Y0)

Функция *ode45* интегрирует систему дифференциальных уравнений на интервале времени *ts=[tn, tk]* с начальными условиями *Y0*. Первый параметр – имя М-функции, вычисляющей правые части системы дифференциальных уравнений. Каждая строка возвращаемого массива *Y* – решение в моменты времени, определяемые вектором *t* (в первом столбце – y_1, во втором - y_2 и так далее).

В качестве примера решим систему дифференциальных уравнений

$$\begin{cases} \dfrac{dy_1}{dt} = 5\,y_2 * \sin(t) \\ \dfrac{dy_2}{dt} = t + 1 - y_1 - t \cdot y_2 \end{cases}$$

с использованием функций *ode45* и *ode15s*. Построим полученные зависимости в разных подокнах.

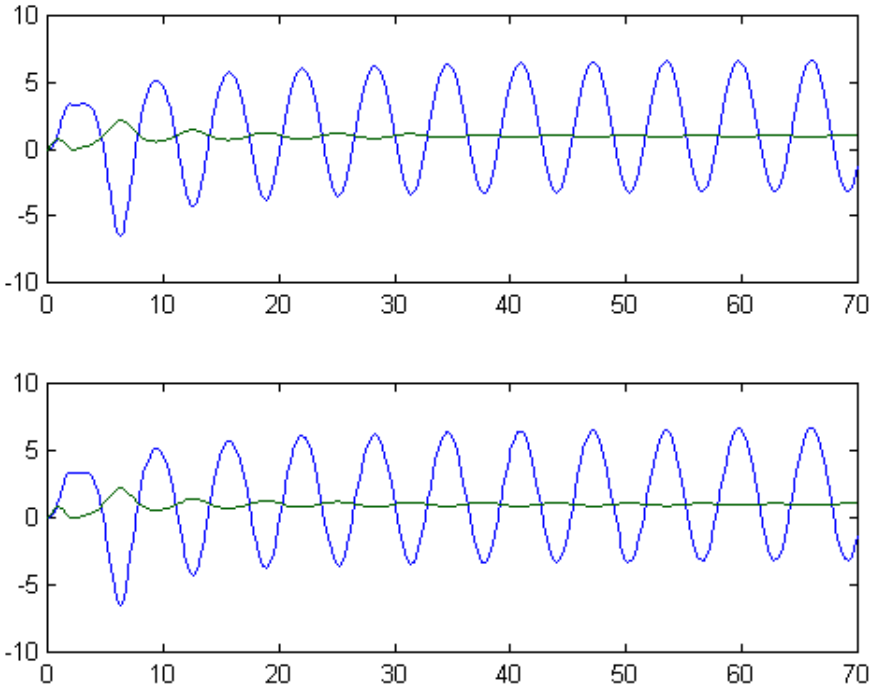
М-функция для вычисления правых частей системы дифференциальных уравнений:

```
function Y=l7_p7f(t,y)
 Y=[ 5*y(2)*sin(t);t+1-y(1)-t*y(2) ];
```

Текст сценария:

```
y0=[0,0];
[T,Y]=ode45('l7_p7f',[0,70],y0);
subplot(2,1,1)
plot(T,Y);
[T1,Y1]=ode15s('l7_p7f',[0,70],y0);
subplot(2,1,2)
plot(T1,Y1);
```

Полученные графики:



7 РЕШЕНИЕ ТИПИЧНЫХ МАТЕМАТИЧЕСКИХ ЗАДАЧ В MATLAB

7.1 Решение нелинейных уравнений и систем нелинейных уравнений

Для определения корня уравнения вида $f(x)=0$ используется функция

fzero(fun, x0, tol)

где *fun* – функция, корень которой следует определить, *x0* начальное приближение корня. Функция *fzero* возвращает приближенное значение корня; *tol* – погрешность определения значения корня (если погрешность не задана, она принимается равной *eps*).

Функция *fun* задается как формула, записанная по правилам MATLAB, заключенная в апострофы или как имя М-функции (в апострофах), вычисляющей левую часть уравнения. Начальное приближение может быть задано скалярным значением или интервалом, на концах которого функция имеет разные знаки (в виде вектора координат начала и конца интервала).

Например, определим корень уравнения $2^x+\sin(x)+x=0$. Для этого составим функцию *f7*, вычисляющую левую часть уравнения:

```
function y=f7(x)
y=2.^x+sin(x)+x;
```

и выполним вызов функции *fzero* с разным набором параметров:

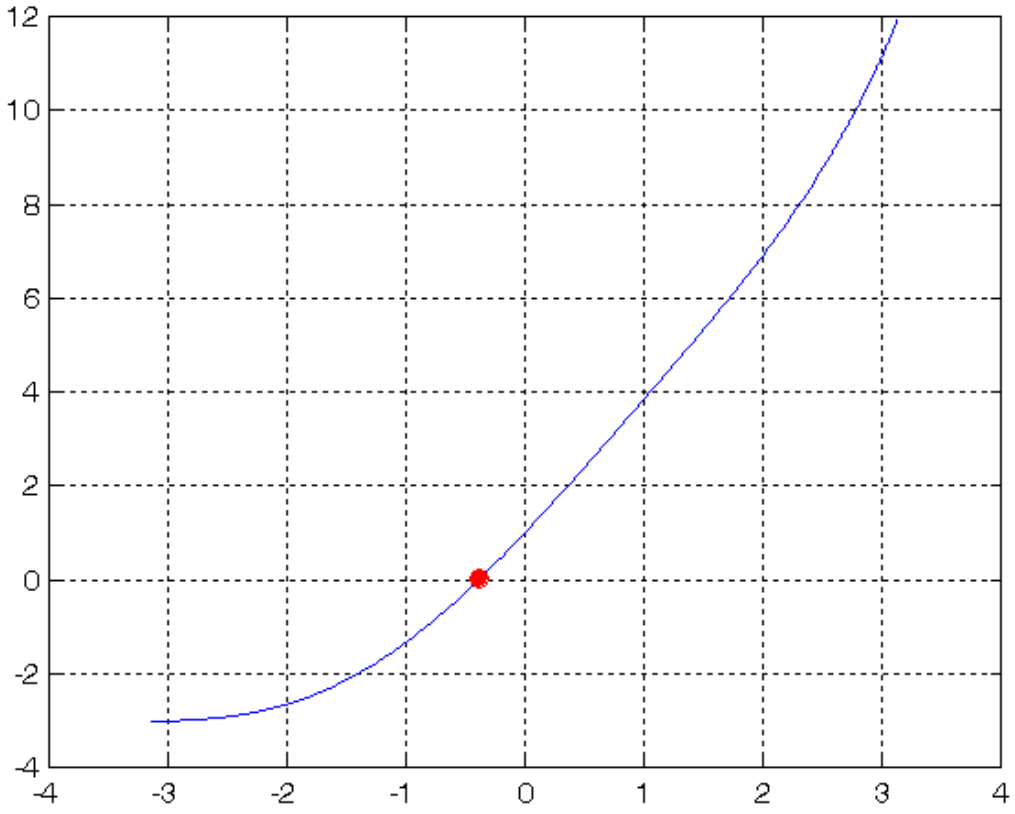
```
x=-pi:0.01:pi;
plot(x,f7(x));
grid on
fzero('2.^x+sin(x)+x',-0.5)

fzero('2.^x+sin(x)+x',-0.5,1e-3)

fzero('f7',-0.5,1e-4)

xr=fzero('f7',[-1,0],1e-4);

hold on;
```

```
scatter(xr,f7(xr),40,'r','filled');
```

Результат:

```
» l7_p0
Zero found in the interval: [-0.38686, -0.58].
ans =
   -0.3871
ans =
   -0.3869
ans =
   -0.3871
»
```

График:



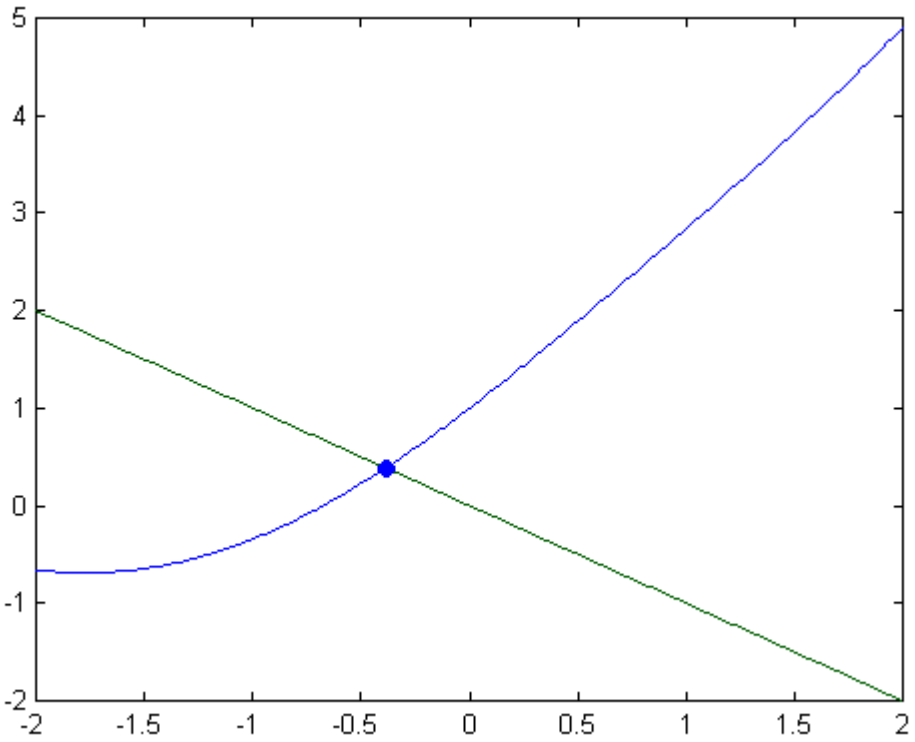
Функция *fzero* определяет только точки пересечения заданной функцией оси абсцисс. Точки, в которых функция только касается оси абсцисс, нулями не считаются.

Для символьного решения уравнений и систем уравнений используется функция *solve*. При ее вызове уравнения задаются в круглых скобках в апострофах через запятую.

Например, решение уравнения $2^x + \sin(x) = -x$ с графической иллюстрацией найденного решения можно выполнить следующим образом:

```
» xr=solve('2^x+sin(x)=-x'); x=-2:0.01:2;y1=2.^x+sin(x); y2=-x;...
plot(x,y1,x,y2);hold on; scatter(double(xr),double(-xr),40,'filled')
» xr
xr =
-.38712465392757487234848391052801
»
```

На рисунке показаны графики функций $y = 2^x + \sin(x)$ и $y = -x$, а также найденная их точка пересечения:



В приведенном примере используется символьная переменная *xr*, поэтому при построении точки применяется преобразование ее типа к типу *double (double(xr))*.

Следующий пример иллюстрирует решение системы уравнений

$$\begin{cases} \sin(x+y) - 1.2x = 0.2 \\ x^2 + y^2 = 1 \end{cases}$$

```
» [xr,yr]=solve('sin(x+y)-1.2*x=0.2','x^2+y^2=1')
```

```
xr =
-.95682502589007371450373183654668
yr =
-.29066453142834570132077207025201
»
```

Проверка:

```
» x=double(xr)
x =
    -0.95683
» y=double(yr)
y =
    -0.29066
» abs(sin(x+y)-1.2*x-0.2)<eps
ans =
    1
» abs(x^2+y^2-1)<eps
ans =
    1
```

7.2 Определение минимума функции одной и нескольких переменных

Поиск минимума функции одной переменной осуществляется функцией *fmin*, вызов которой записывается следующим образом:

xmin=fmin('имя функции', x1,x2)

xmin=fmin('имя функции', x1,x2, options)

xmin=fmin('имя функции', x1,x2, options, p1, p2, ..., p10)

где имя функции – имя встроенной или М-функции, минимум которой необходимо найти; *x1, x2* - начало и конец интервала, в котором необходимо определить минимум функции; *options* – вектор управляющих параметров, предназначенных для настройки алгоритма оптимизации, если используются опции по умолчанию, вместо этого аргумента следует задать пустой массив;

p1, p2,..., p10 – параметры оптимизируемой функции.

Пример. Определить минимумы функций на интервалах:

Функция	Интервал
cos(x)	*[1 , 4]*
*cos(p1*x+p2) при p1=2, p2=1*	*[1 , 4]*
*cos(p1*x+p2) при p1=1, p2=2*	*[5 , 8]*

Оптимизируемая М-функция:

```
function y=l7_p9f1(x,p1,p2)
y=cos(x*p1+p2);
```

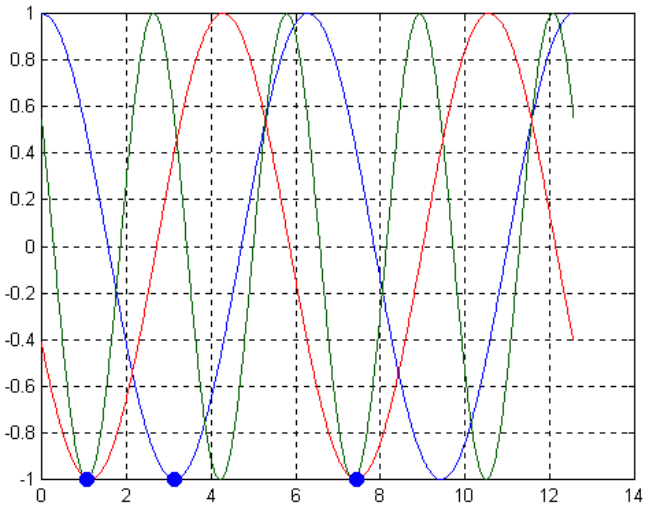
Текст сценария:

```
xmin1=fmin('cos',1,4)
xmin1_2=fmin('l7_p9f1',1,4,[],1,0)
xmin2=fmin('l7_p9f1',1,4,[],2,1)
xmin3=fmin('l7_p9f1',5,8,[],1,2)
x=0:0.01:pi*4;
plot(x,[cos(x);l7_p9f1(x,2,1);l7_p9f1(x,1,2)])
hold on
grid on
scatter([xmin1;xmin2;xmin3],[cos(xmin1);l7_p9f1(xmin2,2,1);l7_p9f1(xmi
n3,1,2)],70,'filled')
```

Результат:

```
» l7_p9
xmin1 =
        3.14158640896816
xmin1_2 =
        3.14158640896816
xmin2 =
        1.07078246937276
xmin3 =
        7.42479085012525
```

График:



Минимизация функции нескольких переменных осуществляется функцией

xmin=fmins('имя функции',x0)

Функция возвращает вектор *xmin* значений переменных, при которых функция принимает минимальное значение в окрестностях начального приближения, задаваемого вектором *x0*.

Найдем минимум функции $f(x,y)=\cos(x)*\sin(y)$ в окрестностях точки с координатами (3,-5). Для графической иллюстрации построим график функции в виде поверхности и линий уровня и нанесем на него найденную точку минимума (с помощью функции *scatter3*, работающей аналогично *scatter*, с той лишь разницей, что надо задать три координаты точки).

Сценарий:

```
close all;
x=-2*pi:0.3:2*pi;
y=x;
x0=[3 ,-5];
xmin=fmins('l7_p10f2',x0)
[xx,yy]=meshgrid(x,y);
subplot(1,2,1)
mesh(xx,yy,l7_p10f1(xx,yy))
axis([-2*pi,2*pi,-2*pi,2*pi,-1,1])
hold on
```

```
scatter3(xmin(1),xmin(2),l7_p10f1(xmin(1),xmin(2)),70,2,'filled')
subplot(1,2,2)
contour(xx,yy,l7_p10f1(xx,yy))
grid on
hold on
scatter3(xmin(1),xmin(2),l7_p10f1(xmin(1),xmin(2)),70,2,'filled')
```

Функции *l7_p10f1 и l7_p10f2:*

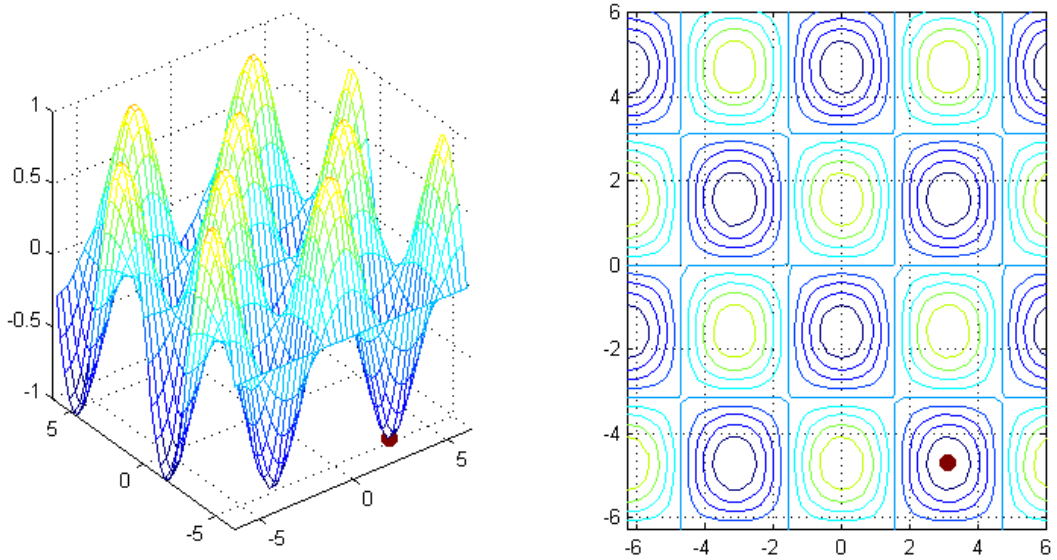
```
function f=l7_p10f1(x,y)
f=cos(x).*sin(y);
```

```
function f=l7_p10f2(x)
f=cos(x(1)).*sin(x(2));
```

Результат:

```
» l7_p10
xmin =
        3.14155102269033        -4.71242727650315
```

Полученный график:



7.3 Вычисление конечных разностей и численное дифференцирование

Функция *diff(x)* вычисляет конечные разности. Если ее аргумент *x* – одномерный массив, состоящий из *n* элементов, функция

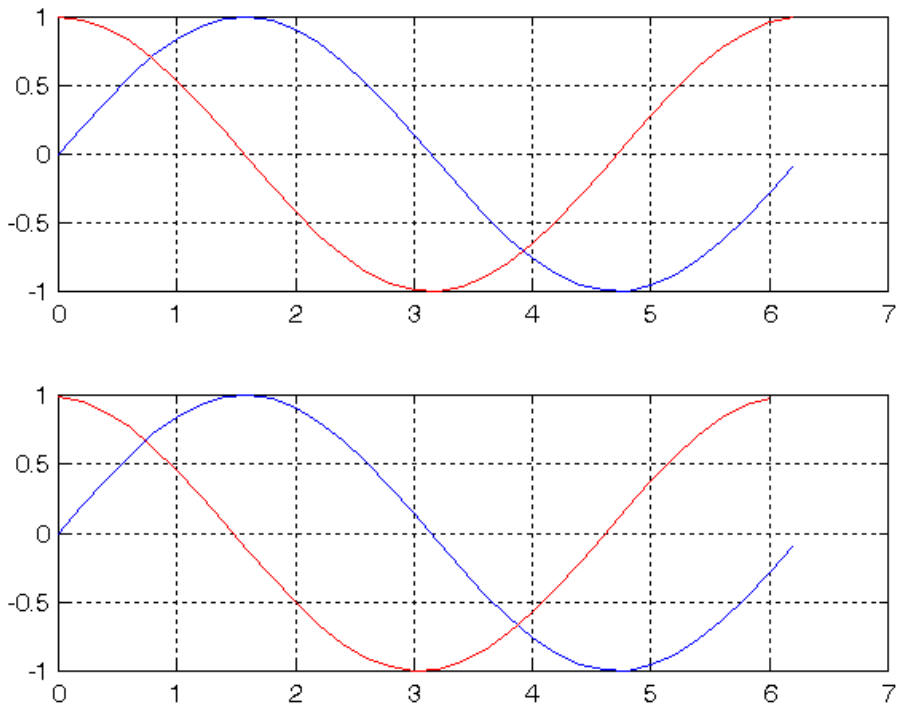
возвращает одномерный массив *[x(2)-x(1) x(3)-x(2) ... x(n)-x(n-1)]*. Количество элементов в возвращаемом массиве *n-1*.

Функция *diff(x,n)* вычисляет конечные разности порядка *n* согласно соотношению: *diff(x,n)=diff(diff(x,n-1))*, например:

```
» y=[4 3 7 1]; dy=diff(y)
dy =
   -1    4   -6
» ddy=diff(dy)
ddy =
    5  -10
» ddyy=diff(y,2)
ddyy =
    5  -10
»
```

Приближенное значение первой производной функции *y(x)*, заданной таблично, вычисляется согласно выражению: *diff(y)./diff(x)*.

В примере в первом подокне построены график функции *sin(x)* и ее первая производная, рассчитанные аналитически, а во втором – приближенно, с использованием функции *diff*.



Сценарий:

```
close all
x=0:0.2:2*pi; y= sin(x);
r=size(x);
n=r(2);

dy=cos(x);d2y=-sin(x);
subplot(2,1,1);
plot(x,y);
hold on;
grid on;
plot(x,dy,'r');
plot(x,d2y,'g');

d1=diff(y)./diff(x);  d2=diff(d1)./diff(x(1:n-1));
subplot(2,1,2);
plot(x,y);
hold on;
grid on;
plot(x(1:n-1),d1,'r');
plot(x(1:n-2),d2,'g');
```

7.4 Численное интегрирование

Функция *trapz(x,y)* возвращает значение интеграла от функции *y* по переменной *x,* вычисленного методом трапеций. Например, вычислим значение интеграла функции $y = x$ на интервале от *1* до *2*:

```
x=1:0.1:2;
y=x;
i=trapz(x,y)%численное интегрирование
ik=2^2/2-1/2%проверка
```

Результат:

```
» l7_p2
i =
       1.5
```

```
ik =
        1.5
```

Если функция вызывается с одним аргументом *trapz(y),* вычисляется значение интеграла при постоянном шаге, равном *1* (очевидно, чтобы перейти к произвольному значению шага, надо умножить полученное значение интеграла на величину шага интегрирования). Для того же примера:

```
x=1:0.1:2;
y=x;
i=trapz(y)%численное интегрирование
i=i*0.1
ik=2^2/2-1/2%проверка
```

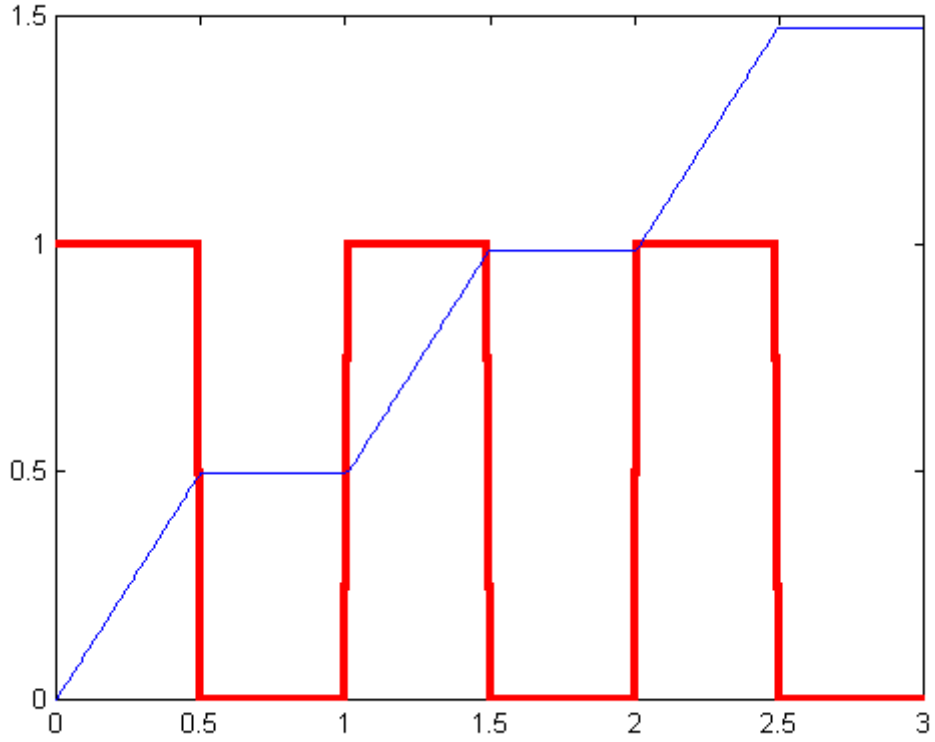
Результат:

```
» l7_p3
i =
   15
i =
        1.5
ik =
        1.5
```

Функция *cumtraps (x,y) (cumtraps (y))* дополнительно вычисляет значение промежуточных результатов. Для функции, представляющей собой прямоугольные импульсы (на графике утолщенная линия красного цвета), функция *cumtraps* позволила построить функцию, отражающую процесс накопления искомого интеграла. Ее значение в произвольный момент времени *t1* равно значению интеграла на интервале от *0* до *t1*. Текст сценария:

```
x=0:0.01:3;
y=(x<0.5|x>1&x<1.5|x>2&x<2.5);
i=cumtrapz(x,y)%численное интегрирование
plot(x,y,'r','LineWidth',3)
hold on
plot(x,i)
```

График:



Функции *quad* и *quad8* реализуют вычисление интеграла методом квадратур (функция *quad* с использованием формул Ньютона-Котеса 2-го порядка (метод Симпсона), функция *quad8* использует формулы 8-го порядка).

Вызов этих функций записывается однотипно

функция quad:	*функция quad8:*
quad('имя функции', a, b) *quad('имя функции', a, b, tol)* *quad('имя функции', a, b, tol, trace)* *quad('имя функции', a, b, tol, trace, p1, p2, ...)*	*quad8('имя функции', a, b)* *quad8(...)*

где имя функции – имя М-функции для вычисления значения подынтегрального выражения; *a, b* – нижний и верхний пределы интегрирования; *tol* – относительная погрешность интегрирования, по умолчанию *1e-3*; если *trace* ≠ *0* – дополнительно выполнять построение точечного графика подынтегральной функции; *p1, p2, ...*

- параметры подынтегральной функции. Применим эти функции для вычисления значения интеграла $\int_{-\pi/2}^{\pi/2}\cos^2(x)dx$. Текст сценария:

```
close all;
i1=quad('l7_p5f',-pi,pi);
i1_8=quad8('l7_p5f',-pi,pi);
i2=quad('l7_p5f',-pi,pi,1e-1);
i2_8=quad8('l7_p5f',-pi,pi,1e-10,1);
disp(['i1=',num2str(i1,20),'   i1_8=',num2str(i1_8,20),'
i2='num2str(i2,20),'   i2_8='num2str(i2_8,20)])
```

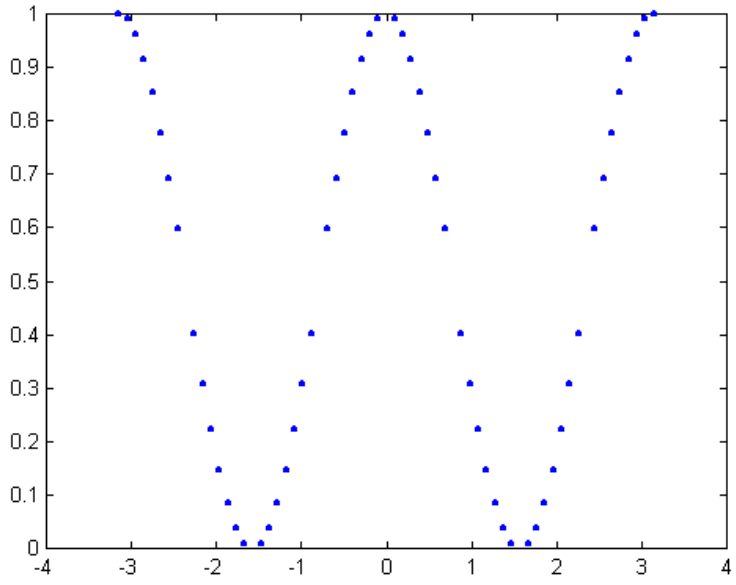
М-функция для вычисления значения подынтегрального выражения:

```
function y=l7_p5f(x)
y=cos(x).^2;
```

Результат:

```
» l7_p5
i1=3.1415926535897931   i1_8=3.1415926535897931
i2=3.1415926535897931   i2_8=3.1415926535897931
```

Точечный график подынтегральной функции:



Вычисление двойных интегралов вида $\int_{y1}^{y2}\int_{x1}^{x1} f(x,y)dxdy$ реализует функция *dblquad*:

dblquad('имя функции', x1, x2, y1, y2)

dblquad('имя функции', x1, x2, y1, y2, tol)

dblquad('имя функции', x1, x2, y1, y2, tol, metod)

где 'имя функции'- имя М-функции, вычисляющей подынтегральное выражение. Эта функция должна иметь два параметра, причем первый из них – вектор значений переменной внутреннего интеграла, второй - скалярное значение переменной внешнего интеграла. Интегрируемая функция должна возвращать вектор; *x1, x2* – нижний и верхний пределы внутреннего интеграла; *y1, y2* - нижний и верхний пределы внешнего интеграла; *metod* – имя одной из функций *quad, quad8* или функции пользователя, которая имеет такой же вызов и имеет такое же возвращаемое значение, как и функции *quad, quad8*.

Например, найдем значение интеграла $\int_0^{2\pi}\int_0^{\pi} \sin(x)^2 \cdot \cos(x)^2 \cdot y \, dx \, dy$.

Текст М-функции:

```
function f=l7_p6f(x,y)
f=sin(x).^2.*cos(x).^2*y;
```

Вызов функции и результаты расчета:

```
» i=dblquad('l7_p6f',0,pi,0,2*pi)
i =
      7.75156917007495
» i=dblquad('l7_p6f',0,pi,0,2*pi,'quad8')
i =
      7.75156917007495
```

7.5 Операции над полиномами

Полином или многочлен – это выражение вида

$P(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + ... + a_n x + a_{n+1}$.

Полином задается вектором коэффициентов (в порядке убывания степени переменной) $A = [a_1, a_2, ...a_{n+1}]$ (количество элементов вектора коэффициентов на 1 больше степени полинома, нулевые коэффициенты не могут быть пропущены).

Функция

polyval(A,x)

возвращает значение степенного полинома с вектором коэффициентов *A* при значении аргумента *x*:

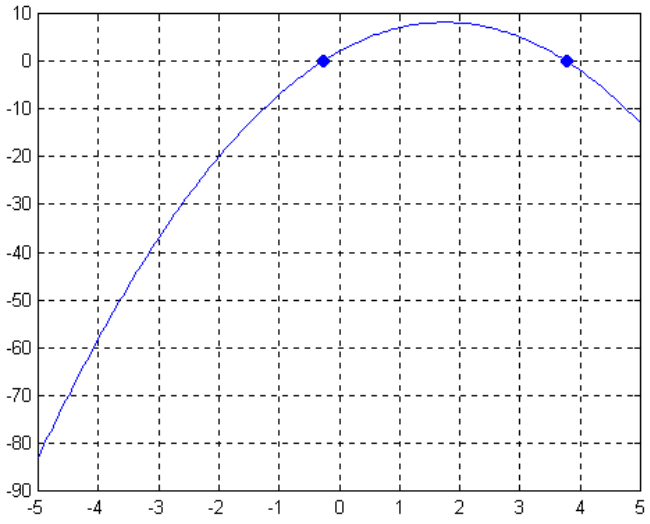
```
» h=[-2 7 2];polyval(h,3)
ans =
     5
» -2*3^2+7*3+2
ans =
     5
```

Если *x* – массив, функция *polyval* возвращает массив того же размера значений степенного полинома при всех значениях элементов массива *x*.

Вычисление корней полинома с с вектором коэффициентов *A* осуществляет функция *roots(A):*

```
» r=roots(h)
r =
    3.7656
   -0.26556
» X=-5:0.05:5;Y=polyval(h,X); plot(X,Y); hold on;
scatter(r,polyval(h,r),40,'filled'); grid on
```

Полученный график:



Функция *conv* возвращает вектор *w* коэффициентов полинома, являющегося произведением двух полиномов, заданных векторами коэффициентов *a* и *b*:

w=conv(a,b)

Функция *deconv* возвращает векторы *u* и *v* коэффициентов полиномов, которые являются частным и остатком от деления

полинома, заданного вектором коэффициентов *a* на полином с вектором коэффициентов *b*:

[u,v]=deconv(a,b)

Например:

```
» a=[4 5 8]; b=[2 3 4 7]; d=conv(a,b)
d =
    8    22    47    72    67    56
» deconv(d,b)
ans =
    4     5     8
» d(1)=d(1)+3
d =
   11    22    47    72    67    56
» deconv(d,b)
ans =
        5.5        2.75       8.375
» [x,y]=deconv(d,b)
x =
        5.5        2.75       8.375
y =
         0          0          0      -2.625       14.25      -2.625
```

Дифференцирование полиномов выполняет функция *polyder*. Аргументом этой функции является вектор коэффициентов дифференцируемого полинома. Функция возвращает вектор коэффициентов соответственно полинома - производной.

```
» a=[4 5 7]; d= polyder(a)
d =
    8     5
»
```

В MATLAB есть функция интегрирования полинома.

7.6 Аппроксимация

Для аппроксимации табличной функции y(x) полиномом степени n (методом наименьших квадратов) используется функция

B=polyfit(x, y, n),

возвращающая вектор B коэффициентов степенного полинома. При вызове функции задаются следующие параметры: *x, y* – векторы значений аргумента и функции соответственно, *n* –степень полинома. Рассмотрим результаты аппроксимации функции степенными полиномами *2, 3, 4* и *5* степени. Сценарий:

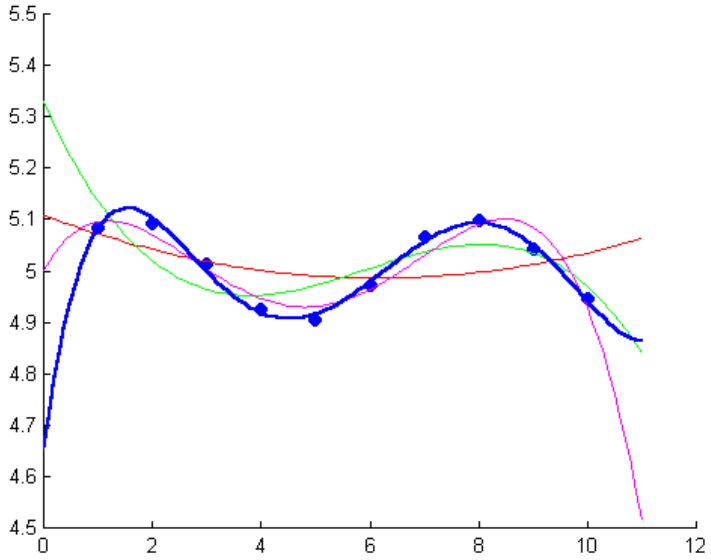
```
x=1:1:10;
y=5+(0.1*sin(x));
scatter(x,y,50,'filled')

b2=polyfit(x,y,2)
b3=polyfit(x,y,3)
b4=polyfit(x,y,4)
b5=polyfit(x,y,5)

xp=0:0.1:11;
hold on
plot(xp,polyval(b2,xp),'r')
plot(xp,polyval(b3,xp),'g')
plot(xp,polyval(b4,xp),'m')
plot(xp,polyval(b5,xp),'b','LineWidth',2)
```

Результат:

```
» l7_p11
b2 =
0.00323858317961304      -0.0397381205521077      5.10798609433367
b3 =
  Columns 1 through 3
  -0.00258158641063002     0.0458347589550084    -0.236196846401052
  Column 4
        5.32948620836573
b4 =
  Columns 1 through 3
-0.00095461460177518 5    0.0184199348284241    -0.107858191930798
  Columns 4 through 5
       0.183833578380035          5.00186247703647
```

```
b5 =
  Columns 1 through 3
0.000245186199226074   -0.00769723508049221   0.0862547832809714
  Columns 4 through 6
-0.411276113473067      0.761819178688985      4.65124621214316
»
```

График:



7.7 Интерполяция

Для интерполяции функции одной переменной *y(x)*, заданной таблично, кубическим сплайном применяется функция

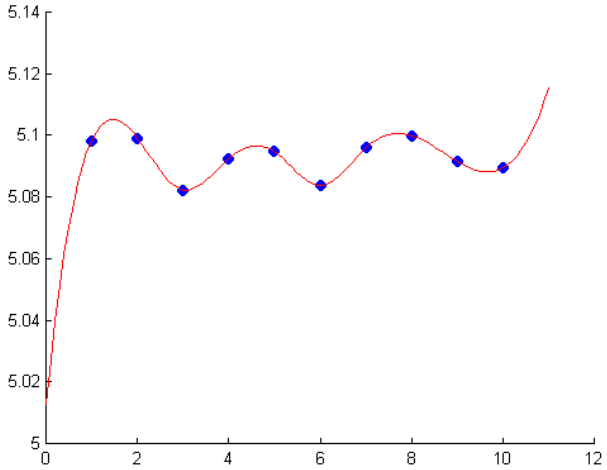
yi=spline(x, y, xi)

где *x, y* – векторы значений аргумента и функции. Функция возвращает значение функции *yi* для заданного значения аргумента *xi*.

Пример:

```
%задаем табличную функцию
x=1:1:10;
y=5+(0.1*sin(x).^0.1);
scatter(x,y,50,'filled')
hold on
xp=0:0.1:11;
%применяем интерполяцию сплайнами для построения графика функции
plot(xp,spline(x,y,xp),'r')
```

График:



Функция

yi=interp1(x, y, xi, 'метод')

применяется для интерполяции табличной функции *y(x)* указанным методом. Функция *interp1* возвращает значение табличной функции *yi* при заданном значении аргумента *xi*. При вызове функции может быть задан один из методов:

– *'nearest'* – ступенчатая интерполяция;

– *'linear'* - линейная интерполяция (выполняется по умолчанию, если метод не указан);

– *'spline'* - интерполяция кубическими сплайнами (аналогично функции *spline*);

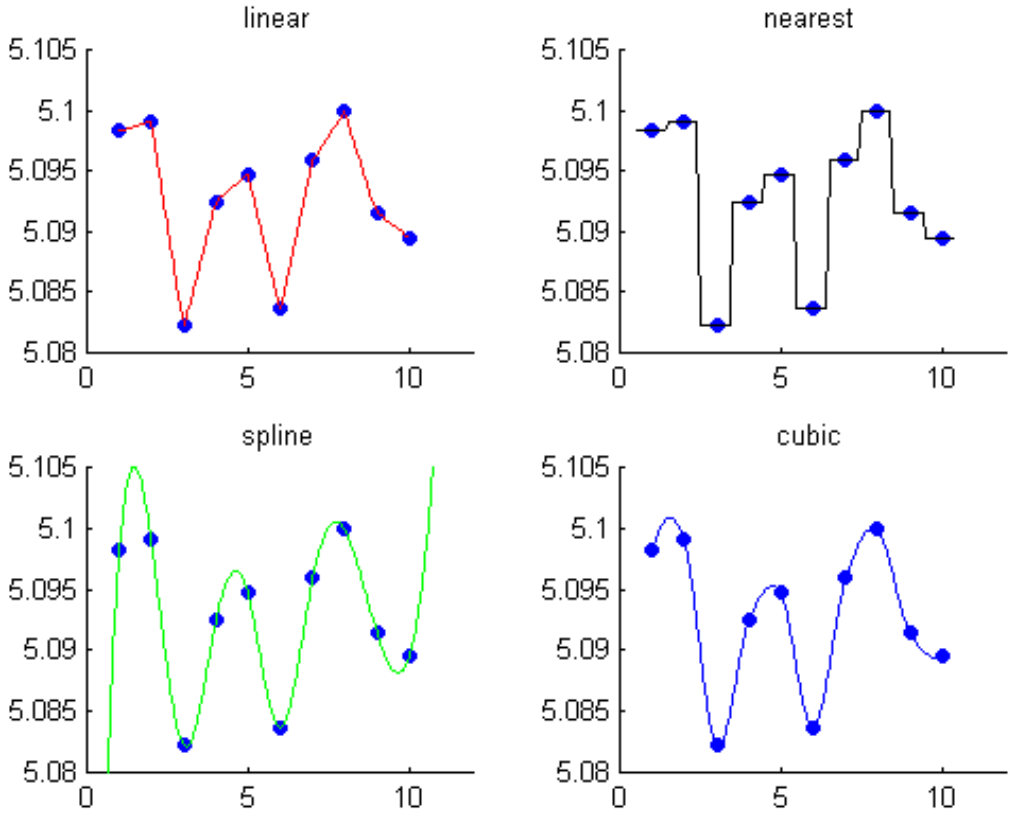
– *'cubic'* - кубическая интерполяция.

Пример:

```
close all;
%задаем табличную функцию
x=1:1:10;
y=5+(0.1*sin(x).^0.1);
xp=0.5:0.1:11;
subplot(2,2,1)
scatter(x,y,30,'filled')
hold on
plot(xp,interp1(x,y,xp),'r')
title('linear')
axis([0,12,5.08,5.105])
```

```
subplot(2,2,2)
scatter(x,y,30,'filled')
hold on
plot(xp,interp1(x,y,xp,'nearest'),'k')
title('nearest')
axis([0,12,5.08,5.105])
subplot(2,2,3)
scatter(x,y,30,'filled')
hold on
plot(xp,interp1(x,y,xp,'spline'),'g')
title('spline')
axis([0,12,5.08,5.105])
subplot(2,2,4)
scatter(x,y,30,'filled')
hold on
plot(xp,interp1(x,y,xp,'cubic'),'b')
title('cubic')
axis([0,12,5.08,5.105])
```

Полученные графики иллюстрируют результаты интерполяции разными методами:



ЛИТЕРАТУРА

1. Дьяконов В. П., Абраменкова И.В. MATLAB 5.0/5.3. Система символьной математики. М.: Нолидж. – 1999. – 640 с.

2. Потемкин В.Г. Система MATLAB 5 для студентов. Справочное пособие. – М.: Диалог-МИФИ, 1998. – 314 с.

3. Потемкин В.Г. Система инженерных и научных расчетов MATLAB 5.x: - в 2-х т. Том 1. – М.: Диалог-МИФИ, 1999. – 366 с.

4. Потемкин В.Г. Система инженерных и научных расчетов MATLAB 5.x: - в 2-х т. Том 2. – М.: Диалог-МИФИ, 1999. – 304 с.

5. Потемкин В.Г. Система MATLAB. Справочное пособие. – М.: Диалог-МИФИ, 1998. – 350 с.

6. Мартынов Н.Н. Matlab 7. Элементарное введение. – М.: КУДИЦ-ОБРАЗ, 2005. – 416 с.

7. Курбатова Е.А. Matlab 7. Самоучитель. – М.: Издательский дом «Вильямс», 2006. – 256 с.

СОДЕРЖАНИЕ

Касымбек Адильбекович Ожикенов

ОСНОВЫ ПРОГРАММИРОВАНИЯ В СРЕДЕ MATLAB

Учебное пособие

Подписано в печать 25.09.2012 г. Тираж 500 экз.
Формат изд. 60x84/16. Объем 9 усл. печ. л.

Отпечатано в типографии "ИП Волков А.И."
Райымбека 212/1, оф. 319. Тел.: 330-03-12, 330-03-13