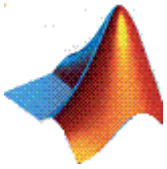


Доля П.Г.
Харьковский Национальный Университет
механико – математический факультет
кафедра геометрии им. А.В. Погорелова

Основы работы в системе MATLAB

Оглавление

Часть 1. Знакомство с MatLab

1.1 Числа, векторы, матрицы, операции, выражения. 3
1.1.1 Знакомство 3
1.1.2 Выражения 6
1.1.3 Работа с матрицами. 8
1.1.4 Командная строка MatLab 10
1.2 Сценарии и функции 11
1.3 Графика 16
1.3.1 Базовые графические объекты 16
1.3.2 Функции и кривые 19
1.3.3 Поверхности 22
1.3.4 Анимация 27
1.4 Символьные вычисления 29
1.4.1 Использование символьных переменных 29
1.4.2 Графическое представление символьных функций 32
1.4.3 Алгебраические преобразования 33
1.4.4 Элементы математического анализа 35
1.5 Алгебраические задачи 38
1.5.1. Полиномы 38
1.5.2. Нахождение корней и решение алгебраических уравнений 40
1.5.3 Решение систем линейных уравнений 45
1.6 Справочная система и среда MatLab 49
1.6.1 Справочная система 49
1.6.2 Среда MatLab 50

Прикладные математики, способные к глубокому общению с другими учеными и инженерами и знакомые с мощью и ограничениями цифровых машин … способны стать вождями завтрашнего математического мира …!

Из книги Гаррета Бирхгоффа "Математика и психология".

Программа MATLAB представляет собой высокоуровневый технический вычислительный язык и интерактивную среду для разработки алгоритмов, визуализации и анализа числовых расчетов. В некотором смысле MATLAB это мощный калькулятор. Даже если вы только поверхностно знакомы с программой, вы можете использовать ее для выполнения невероятных вещей. Его название является сокращением от Matrix Laboratory. В MATLAB реализованы классические численные алгоритмы решения уравнений, задач линейной алгебры, вычисления определенных интегралов, аппроксимации, решения дифференциальных уравнений и их систем. MATLAB обладает хорошо развитыми возможностями визуализации двумерных и трехмерных данных. Его основой являются численные расчеты, но и символьные вычисления MATLAB умеет делать хорошо.

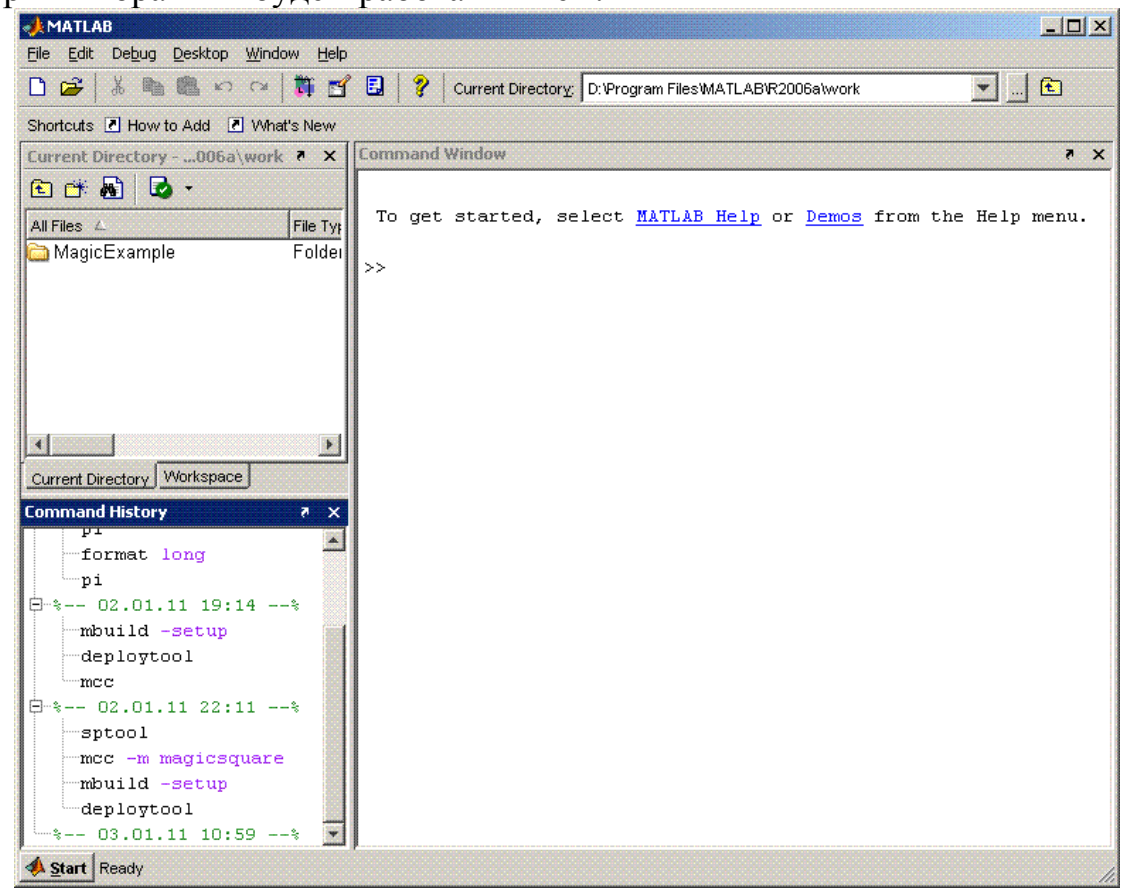
Символьные вычисления в MATLAB основаны на библиотеке, являющейся ядром пакета Maple. Решение уравнений и систем, интегрирование и дифференцирование, вычисление пределов, разложение в ряд и суммирование рядов, поиск решения дифференциальных уравнений и систем, упрощение выражений — вот далеко не полный перечень возможностей MATLAB для проведения аналитических выкладок и расчетов. Обязательным элементом символьной математики является поддержка численных расчетов произвольной точности.

В настоящем пособии мы даем описание только основных возможностей, которыми, как нам кажется, должен владеть каждый пользователь – математик системы MATLAB. Пособие предназначено в первую очередь для знакомства с математическими возможностями программы. Читайте текст и выполняйте примеры. Во многих случаях все пояснения дает само решение задачи. Надеемся, что по окончании выполнения последнего примера вы начнете применять систему MATLAB для решения ваших задач.

1.1 ЧИСЛА, ВЕКТОРЫ, МАТРИЦЫ, ОПЕРАЦИИ, ВЫРАЖЕНИЯ.

1.1.1 Знакомство

После запуска пакета MatLab вы увидите стандартное окно программы. Его основным элементом является командное окно, расположенное справа. Оно предназначено для ввода команд и отображения результатов вычислений. На первых порах мы будем работать в нем.



Внешний вид окна программы

Рис. 1.1

Наберите в командном окне после символа **>>**: **2+2**. Теперь нажмите клавишу **Enter**. Вы увидите:

```
>> 2+2
ans =
4
```

Вероятно строки ввода и вывода будут разделены пустыми строками. Их мы в нашем пособии не печатаем. Символ **>>**, стоящий слева, является символом приглашения на ввод команд. Команды можно вводить справа от него или под ним. Символ **ans** представляет идентификатор результата (answer – ответ).

В нашем пособии текст, набранный шрифтом **Arial-жирный** или **Courier-Bold,** будет представлять команды (то, что мы вводим). Мы будем его печатать без символа приглашения. Текст напечатанный шрифтом `Curier` –

будет представлять то, что мы получим в результате вычисления. Иногда результаты вычисления мы печатать не будем.

MatLab это сокращение от **Mat**rix **Lab**oratory (матричная лаборатория). Числа в MatLab это матрицы. Чтобы ввести матрицу надо ввести ее как список: окружить квадратными скобками, разделять строки точкой с запятой, отделять числа одной строки пробелами. Введите после символа приглашения **>>**

A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]

Выполним строку, нажав Enter. Получим

```
A =
16 3  2 13
5 10 11 8
9  6  7 12
4 15 14 1
```

Теперь идентификатор A есть в рабочем пространстве и к матрице можно обращаться по ее имени A.

Операции + – * / между матрицей и числом означает прибавление (вычитание) числа к каждому элементу матрицы, умножение (деление) на число каждого элемента матрицы.

A+15

A*2

15-A

(но возведение в степень работает по-другому!).

Команды MatLab – это, как правило, вызов одной или нескольких функций по их имени. Аргументы функций набираются в круглых скобках и разделяются запятой. Например

sum(A)

```
ans =
34 34 34 34
```

Ответу всегда присваивается идентификатор. Если вы не ввели его, то MatLab сделает это сам (ans). Здесь мы подсчитали сумму элементов столбцов. MatLab предпочитает работать со столбцами. Чтобы просуммировать строки матрицу надо транспонировать.

A'

```
ans =
16 5 9 4
3 10 6 15
2 11 7 14
13 8 12 1
```

sum(A')'

```
ans =
34
34
34
34
```

Вектор (матрица – столбец) составленный из чисел, стоящих на диагонали матрицы, мы получим с помощью функции **diag**.

diag(A)

sum(diag(A))

Последняя команда просуммировала элементы, стоящие на главной диагонали матрицы.

Функция **fliplr(A)** зеркально отображает матрицу слева направо

fliplr(A)

и делает побочную матрицу главной. Тогда команда

sum(diag(fliplr(A)))

дает сумму элементов побочной диагонали.

Чтобы получить элемент матрицы надо написать **A(i,j)** – первый индекс номер строки, второй номер столбца. Например,

A(1,4) + A(2,4) + A(3,4) + A(4,4)

даст сумму элементов 4-го столбца

Можно ссылаться к элементам двумерной матрицы по одному индексу **A(k)**. В этом случае она рассматривается как одномерный вектор, сформированный из столбцов матрицы. Нумерация индексов начинается с единицы. Например

A(15)

```
ans =
    12
```

это другой способ ссылаться к элементу A(3,4).

Если индекс неправильный, то получим сообщение об ошибке

A(3,5)

```
???  Index exceeds matrix dimensions.
```

Но если такому элементу присвоить значение, то размер матрицы увеличивается

A(3,5)=17

```
A =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12    17
     4    15    14     1     0
```

Оператор двоеточие:

Выражение

1:10

```
ans =
     1     2     3     4     5     6     7     8     9    10
```

это вектор строка, содержащая целые числа от 1 до 10. Между начальным значением и конечным можно ввести приращение

1:2:10

Для получения обращенного интервала вводят отрицательное приращение

100:-10:50

```
ans =
   100    90    80    70    60    50
```

100:-7:50

или

0:pi/4:pi

```
0 0.7854 1.5708 2.3562 3.1416
```

Индекс с двоеточием **A(1:k, j)** выделяет часть матрицы

A(1:3,2)

```
ans =
     3
    10
     6
```

(из 2 – го столбца берем элементы от 1 – го до 3 – го). Например,

sum(A(1:4,4))

суммирует элементы 4 – го столбца.

Слово **end** в индексе матрицы означает выбор последнего столбца (или строки), а двоеточие – выбор всех элементов столбца (или строки).

A(end, :)

```
ans =
     4    15    14     1     0
```

sum(A(:,end)) – суммирует все элементы последнего столбца матрицы.

В MatLab есть функция, которая создает «магическую матрицу» любого размера (матрица у которой сумма элементов по столбцам, строкам, по диагонали и по побочной диагонали одинакова). Создадим такую матрицу размера 4 x 4

B=magic(4)

и переставим местами столбцы

A=B(:,[1 3 2 4])

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Эта запись означает, что для каждой строки элементы переписываются в порядке 1 3 2 4.

Чтобы отменить вывод результатов вычисления в командное окно в конце выражения надо поставить точку с запятой.

B=magic(4);
A=B(:,[1 3 2 4])

1.1.2 Выражения

В MatLab нет необходимости определять тип переменной или размерности матрицы. Например

std=5

создает матрицу размером 1 x 1 с именем **std**. Имена переменных состоят из цифр, букв и символов подчеркивания. Заглавные и строчные буквы различаются. Различаются первые 31 символ в имени переменной.

В числах целая часть от дробной отделяется точкой. Научная система использует букву **e** для определения множителя степени десяти. Мнимые числа используют **i** и **j** как суффикс. Вот примеры правильной записи чисел

```
3           -99             0.0001
9.6397238   1.60210e-20     6.02252e23
1i          -3.14159j       2+4*i
```

Числа хранятся в формате с плавающей точкой, который обладает ограниченной точностью – примерно 16 значащих цифр, и ограничены диапазоном от 10^{-308} до 10^{308} .

Выражения используют обычные арифметические операции и правила старшинства: + – * / \(левое деление) ^(степень) '(комплексно сопряженное транспонирование). Это матричные операции, например, **A*A** это матричное умножение матрицы на себя. Для выполнения поэлементных операций перед знаком операции следует поставить точку, например, **A.*A** – поэлементное умножение матрицы на себя. Сравните

A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1];
A*A
A.*A

Чтобы ввести и выполнить несколько выражений или команд за один раз в командном окне после ввода первой команды следует нажать комбинацию клавиш Shift – Enter, ввести вторую команду, нажать снова комбинацию клавиш Shift – Enter и т.д. После ввода последней команды следует нажать Enter. Все команды будут выполнены и те, которые не заканчивались точкой с запятой, напечатают результат своей работы.

Изменить порядок выполнения операций можно с помощью круглых скобок ( ), которые также используются для указания аргументов функций. Если аргументом функции является вектор (матрица), то функция возвращает вектор или матрицу все элементы которых получены применением этой функции к каждому элементу вектора (матрицы).

x=0:pi/4:2*pi;
y=sin(x);
[x', y']

```
ans =
         0         0
    0.7854    0.7071
    1.5708    1.0000
    2.3562    0.7071
    3.1416    0.0000
    3.9270   -0.7071
    4.7124   -1.0000
    5.4978   -0.7071
    6.2832   -0.0000
```

Здесь мы напечатали таблицу значений функции **sin(x)** от 0 до 2π с интервалом $\pi/4$. Математические функции всегда выполняются поэлементно!

Стандартные функции имеют имена: abs sqrt exp sin и т.д. Список всех элементарных функций можно вывести командой **help elfun.** Команды

help specfun
help elmat

выводят список более сложных математических и матричных функций.

Стандартные функции встроены, более сложные, например **sinh**, реализованы в M – файлах и можно даже посмотреть их код. Несколько специальных функций представляют значения часто используемых констант, например, **pi** или **realmax** (наибольшее число с плавающей точкой; на моем компьютере оно равно 1.7977e+308), **Inf** (бесконечность), **eps** – относительная точность числа с плавающей точкой 2^{-52}, **NaN** используется для представления не числового значения, например, результат вычисления 0/0 или `Inf - Inf`.

Имена функций не защищены, поэтому можно менять их значения **pi=5.6**. Начальное значение можно восстановить следующим образом **clear pi**.

Приведем несколько примеров выражений MatLab

rho = (1+sqrt(5))/2

```
rho =
1.6180
```

a = abs(3+4i)

```
a =
```

```
5
```
z = sqrt(besselk(4/3,rho-i))
```
z =
0.3730 + 0.3214i
```
huge = exp(log(realmax))
```
huge =
1.7977e+308
```
toobig = pi*huge
```
toobig =
Inf
```
Набранное выражение, как правило, не заканчивается никаким символом. В этом случае результат вычисления будет отображаться в командном окне следом за введенной командой. Если вы желаете выполнить вычисления, а результат не выводить, то завершайте такие выражения точкой с запятой.

1.1.3 Работа с матрицами.

Есть функции, которые генерируют матрицы предопределенного размера и структуры. Например

Z = zeros(2,4) (**zeros** генерирует матрицу со всеми нулями)
```
Z =
0 0 0 0
0 0 0 0
```
F = 5*ones(3,3) (**ones** генерирует матрицу со всеми единицами)
```
F =
5 5 5
5 5 5
5 5 5
```
N = fix(10*rand(1,10)) (**rand** – генерирует матрицу со случайными числами на отрезке [0, 1]; **fix** округление до целого числа в сторону нуля)
```
N =
9 2 6 4 8 7 4 0 8 4
```
Аргументы функций **zeros, ones, rand** задают количество строк и столбцов генерируемой матрицы.

Справку по функции можно получить командой **help** *имя функции*

help fix
```
FIX    Round towards zero.
```
. . .

help rand
```
 RAND   Uniformly distributed random numbers.
```
. . .

Матрицы можно загружать из файлов. Команда **load** может загрузить текстовый файл, содержащий числовые данные, или двоичные файлы с матрицами, созданные в MatLab ранее. Текстовый файл должен быть сформирован в виде прямоугольной таблицы чисел, разделенных пробелами с равным количеством чисел в строке. Например, создадим в «Блокнот»–е файл `Matr1.dat` с таблицей чисел
```
1     2     3     4
5     6     7     8
9     10    11    12
13    14    15    16
```

и выполним команду

load Matr1.dat

Файл будет прочитан и в рабочем пространстве MatLab будет создана переменная с именем Matr1, содержащая матрицу. Содержимое рабочего пространства можно увидеть в окне Workspace. На рис.1.1 его можно увидеть слева от командного окна (левое верхнее). Для этого надо щелкнуть на закладку – корешок Workspace.

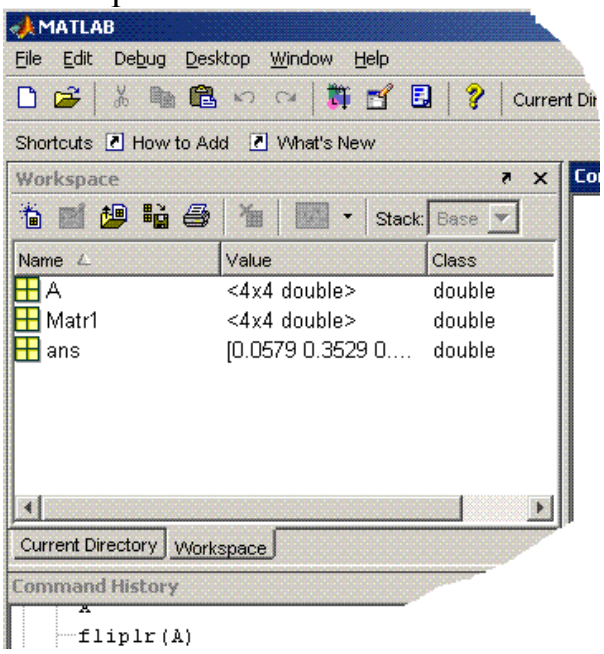


Рис. 1.2

В этом окне отображаются имена, тип и значения (если это возможно) переменных, доступных в текущем сеансе работы MatLab.

Чтобы проверить, что матрица с именем `Matr1` создана, просто наберите ее имя в командном окне

Matr1

Можно создавать матрицы, используя M-файлы, которые представляют текстовые файлы, содержащие код MatLab. Для создания такого файла вызовем встроенный текстовый редактор из меню `File - New - M-file`. В открывшемся окне введем

```
C=[16      3     2     0
   5      10     11    8
   9      6      7     12
   4      15     14    1];
```

Сохраним файл под именем W1.m . Тогда команда

W1

прочитает файл и создаст переменную с именем **C**, которая будет содержать матрицу. Вы можете увидеть имя загруженной матрицы в окне рабочего пространства или, набрав имя матрицы, напечатать ее содержимое.

Заметим, что для того, чтобы не указывать каталог расположения m файлов его название (каталога) должно быть указано в настройках MatLab. Это делается из меню `File - Set Path...`

Матрицы можно конструировать путем объединения уже имеющихся матриц. Это процесс создания матриц из маленьких. Пара квадратных скобок – это оператор объединения. Например

A=magic(4);

B = [A A+32; A+48 A+16]

```
B =
    16     2     3    13    48    34    35    45
     5    11    10     8    37    43    42    40
     9     7     6    12    41    39    38    44
     4    14    15     1    36    46    47    33
    64    50    51    61    32    18    19    29
    53    59    58    56    21    27    26    24
    57    55    54    60    25    23    22    28
    52    62    63    49    20    30    31    17
```

Результатом является матрица 8 x 8, получаемая соединением четырех матриц. Суммы в столбцах

sum(B)

```
ans =
   260   260   260   260   260   260   260   260
```

Сумма по строкам

sum(B')

```
ans =
   196   196   196   196   324   324   324   324
```

Удаление строк и столбцов. Чтобы удалить какой – то элемент матрицы (строку, столбец) его надо присвоить пустым квадратным скобкам.

X=A;

Удалим второй столбец командой

X(:,2) = [ ]

```
X =
    16     3    13
     5    10     8
     9     6    12
     4    15     1
```

Удаление одного элемента приводит к ошибке

X(1,2)=[ ]

```
???  Indexed empty matrix assignment is not allowed.
```

Однако использование одного индекса удаляет отдельный элемент и преобразует матрицу в строку.

X(3)=[ ]

```
X =
    16     5     4     3    10     6    15    13     8    12     1
```

или

X=A

X(2:2:10) = [ ]

```
X =
    16     9     2     7     3     6    15    13     8    12     1
```

1.1.4 Командная строка MatLab

Все приказы программе вводятся в командной строке командного окна, которая начинается символом **>>**. Форматом выводимых результатов можно управлять с помощью команды **format**, используя подходящие аргументы.

Выполнив следующие примеры, вы легко поймете смысл аргументов этой команды.

x = [4/3 1.2345e-6]
format short; x
```
x =
1.3333 0.0000
```
format short e ; x
```
1.3333e+000 1.2345e-006
```
format short g ; x
```
1.3333 1.2345e-006
```
format long ; x
```
1.33333333333333 0.00000123450000
```
format bank ; x
```
1.33 0.00
```
format rat ; x
```
4/3 1/810045
```
format hex ; x
```
3ff5555555555555 3eb4b6231abfd271
```
Команда
format compact
отменяет вывод большого количества пустых линий при выводе.

Нажатие `Enter` в конце строки выполняет команду и выводит результат. Если закончить команду точкой с запятой, то результат в командное окно не выводится. Например, не обязательно смотреть на большую матрицу

A = magic(100);

Если выражение не умещается в одной строке, то используйте троеточие и `Enter` для обозначения продолжения на следующую строку.
```
s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
-1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```
Пробелы вокруг знаков + и – необязательны, но улучшают читаемость строки.

Если вы допустили ошибку в строке, то вместо того, чтобы ее набирать заново, нажмите стрелку ↑. Затем используйте стрелки ← →, чтобы перейти к ошибке и исправить ее. Повторное использование стрелки ↑ вызовет предыдущую команду. Наберите несколько первых символов, тогда клавиша ↑ найдет команду, начинающуюся с них.

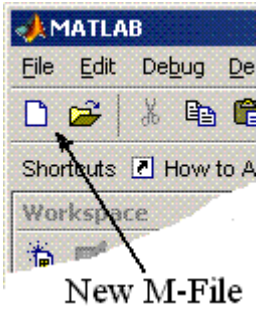
1.2 СЦЕНАРИИ И ФУНКЦИИ

Выше мы рассмотрели достаточно простые примеры, для выполнения которых требуется набрать несколько команд в командной строке. Для более сложных задач число команд возрастает, и работа в командной строке становится непродуктивной. Эффективное решение состоит в оформлении собственных алгоритмов в виде программ (М-файлов), которые можно запустить из рабочей среды или из редактора. Встроенный в MATLAB редактор М-файлов позволяет не только набирать текст программы и запускать ее целиком или частями, но и отлаживать алгоритм.

Файлы, которые содержат код MatLab, называются M – файлами. Вы создаете такие файлы, используя текстовый редактор (можно использовать даже 'Блокнот'). Затем эти файлы используются как любая функция или команда. Существует два вида M-файлов:

- сценарии, которые не имеют входных и выходных аргументов; они используют данные (переменные) из рабочего пространства;
- функции, которые имеют входные и выходные аргументы; они используют локальные переменные.

Раскройте меню `File` рабочей среды MATLAB и в пункте `New` выберите подпункт `M-File` или нажмите кнопку `New M-file` на панели инструментов рабочей среды.



Новый файл открывается в окне редактора M-файлов. Вводите в каждой строке по одной команде MatLab, затем сохраните файл, например, под именем W1.m Этот файл называется сценарием. Чтобы увидеть содержимое M – файла в командном окне надо выполнить команду

type W1

Когда вызывается сценарий, MatLab просто выполняет команды, содержащиеся в файле. Сценарии берут данные из рабочего пространства и создают переменные, которые остаются в рабочем пространстве. В качестве примера создадим сценарий, исследующий ранг магических матриц разного размера. Выберем пункт меню `File – New – M-file`. Откроется текстовый редактор, в котором введите следующий код

```
% Исследование ранга магических матриц
r = zeros(1,32);
for n = 3:32
r(n) = rank(magic(n));
end
r
bar(r)
```

Как видим, строка с комментарием начинается знаком % (процент). Сохраните файл под именем SC_01.m. в одном из рабочих каталогов MatLab (напомним, что добавление/удаление таких каталогов выполняется из пункта меню `File – Set Path…`). Для выполнения сценария в командной строке наберите

SC_01

Сценарий выводит одномерный массив значений рангов матриц и строит столбчатую диаграмму этих значений.

Когда курсор находится в теле сценария, выполнить его код можно нажатием клавиши F5.

В теле файлов сценариев и функций можно использовать управляющие конструкции, такие как **for**, которая организует циклическое вычисление команд, стоящих после нее до первой команды **end**.

Команда **return** выполняет досрочное завершение сценария. Все команды, стоящие в коде после нее, не будут выполняться.

Команды управления потоками (элементы языка программирования MatLab) будут описаны во второй части нашего пособия.

Функции (по – другому называемые файл – функциями) – это М – файлы, которые могут иметь входные аргументы и возвращать значения. Имя М – файла и функции должно быть одинаковым. Функции работают с переменными в пределах собственного рабочего пространства, отличного от того с которым вы оперируете в командной строке.

Хорошим примером является функция **rank**. Ее код находится в файле `toolbox/matlab/matfun/rank.m` Чтобы посмотреть содержимое этого файла выполните команду

type rank

```
function r = rank(A,tol)
%RANK   Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

%   Copyright 1984-2000 The MathWorks, Inc.
%   $Revision: 5.9 $  $Date: 2000/06/01 02:04:15 $

s = svd(A);
if nargin==1
   tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

Первая строка функции в М – файле начинается со слова `function`, после которого задаются выходные переменные, имя функции и входные аргументы. В нашем примере используется один выходной параметр `r` и два входных аргумента `A` и `tol`. Следующие несколько строк до первой пустой строки или первой выполняемой строки являются комментариями, которые выводятся по команде

help rank

При этом самая первая строка комментария несет еще одну нагрузку. Она используется для поиска функций по ключевым словам командой

lookfor rank

Остальное содержимое М – файла представляет исполняемый код. Все переменные: `s` - внутренняя, `r` - возвращаемая, `A` и `tol` – аргументы являются локальными. Функция **rank** может быть использована несколькими способами

rank(A)

r = rank(A)

r = rank(A, 1.e-6)

Обратите внимание, что функция **rank** может вызываться с одним или двумя аргументами. Если нет выходного аргумента, то результат сохраняется в переменной `ans`. Если нет второго входного аргумента, то функция присваивает ему значение по умолчанию. Внутри тела функции можно использовать переменные `nargin` и `nargout`, которые хранят число входных и выходных аргументов при каждом использовании функции. Рассмотренная функция **rank** использует только переменную `nargin`.

Если надо использовать одну копию какой-то переменной в нескольких функциях, то ее надо объявить как `global` во всех функциях. То же надо сделать в командной строке, если вы хотите, чтобы основное рабочее пространство получило доступ к этой переменной. Например, создадим файл `falling.m`, вычисляющий пройденный путь свободно падающего тела за указанное время

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

Затем в командной строке введем следующий код

```
global GRAVITY
GRAVITY = 10;
y = falling((0: .1: 5)' );
```

Определение `global` должно стоять до первого использования ее в теле функции или рабочем пространстве.

Имя файл-функции не обязательно должно совпадать с именем файла, однако обращение к ней происходит по имени файла. Например, если в файле `f22.m` содержится функция с заголовком `g = init(z)`, то ее следует вызывать так:

f = f22(- 0.9)

а вовсе не

f = init(- 0.9) (не работает !)

Чтобы не было путаницы, всегда давайте имя m - файлу такое же, как у функции.

На сеанс можно создавать временные функции с помощью команды inline, которой в качестве аргумента надо передать выражение, заключенное в кавычки

```
ff=inline('x^2/2-x^4/4')
ff =
     Inline function:
     ff(x) = x^2/2-x^4/4
```

С такой функцией можно выполнять любые допустимые операции, например, можно вычислять ее значение в точке

```
ff(1)
ans =
         0.25
```

или строить график с помощью функции **fplot**

```
fplot(ff,[-2 2])
```

О графических функциях мы подробнее поговорим в следующем параграфе.

С помощью `inline` можно создавать функции 2-х переменных

```
fff=inline('x.^2+y.^2')
fff =
     Inline function:
     fff(x,y) = x.^2+y.^2
```

Тогда график этой функции можно построить следующим образом

```
ezsurf(fff,[-1 1 -1 1]);
```

Замечание. *При создании функций следует всегда ставить точку для указания поэлементных операций. Точку можно не ставить, когда арифметическая операция выполняется между переменной и числом (константой).*

Функции можно создавать с помощью дескриптора @.

```
zz=@(x) exp(-x).*sin(10.*x)./(10*x)
zz =
    @(x) exp(-x).*sin(10.*x)./(10*x)
```

Идентификатор, стоящий в левой части равенства становится именем функции. Справа после знака @ в скобках следует перечислить аргументы функции. С такой функцией также можно выполнять любые допустимые операции – вычислять ее значение в точке, строить график, находить корни, интегрировать и т.д.

```
zz(0.01)
ans =
    0.9884
fplot(zz,[-1 3])
```

Вместо аргументов функциям можно передавать вектора и матрицы. Так выражение

```
g=@(x,y) x.^2+y.^2
g([1 2], [3 4])
ans =
    10    20
```

дает значения функции в точках (1, 3) и (2, 4).

Команды и функции взаимозаменяемы. Пример команд

```
help
```

Многие команды используют управляющий параметр, который определяет последующее действие

```
load имя файла
help magic
type rank
```

Другой способ использования командных параметров состоит в создании строки аргументов функции

```
load( 'имя_файла.dat' )
help( 'magic' )
type( 'rank' )
```

Любая команда типа **command argument** может быть записана в виде **command('argument').** Это называется командно – функциональной двойственностью. Преимущества функционального подхода проявляется, когда командная строка создается из нескольких частей.

1.3 ГРАФИКА

Графические возможности пакета весьма обширны. Для первого знакомства мы приведем краткое описание простейших функций.

1.3.1 Базовые графические объекты

Все графики, кривые и поверхности, которые можно построить в MatLab, являются комбинацией некоторого количества простейших – отрезков, многоугольников и т.д. Комбинируя их определенным образом, мы можем получить практически любой рисунок. Изучение всех возможностей мы отложим до второй части нашего пособия. Здесь же мы рассмотрим основные примитивы, которые могут понадобиться в самом начале вашей работы.

Простейшей функцией является **line**. Она строит отрезки и ломанные. Ее первым аргументом является вектор *x* координат узлов ломаной, а вторым – вектор *y* координат узлов. Для построения пространственной ломаной используется третий аргумент – вектор *z* координат узлов. После задания векторов можно задать свойства – толщину линий, цвет и т.д. в формате

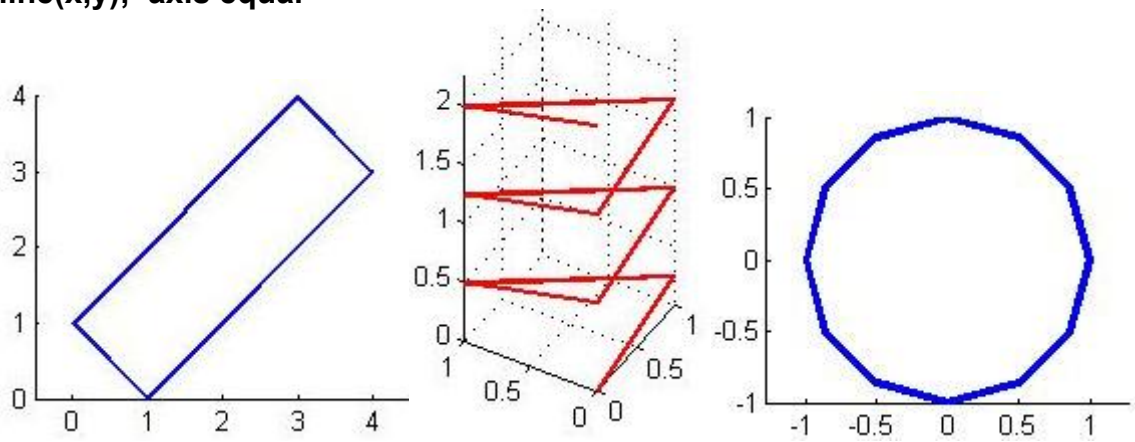
`line(X,Y,Z,'PropertyName',PropertyValue,...).`

Например

```
line([1 4 3 0 1],[0 3 4 1 0]); axis equal
line([0 1 1 0 0 1 1 0 0],[0 0 1 1 0 0 1 1 0],[0 1 2 3 4 5 6 7 8], 'Color','r','LineWidth',2)
grid on;                % рисует сетку на координатных плоскостях
```

или

```
t=0:pi/6:2*pi; x=cos(t); y=sin(t);
line(x,y);  axis equal
```



Команда **axis equal** делает отметки приращений осей одинаковой длины. Каждая команда **line** добавляет графический объект ломаной в текущее графическое окно. Очистка окна от старого рисунка может быть выполнена командой

```
clf
```

Можно также перед построением нового рисунка закрыть графическое окно кнопкой – крестиком в правом верхнем углу окна или командой

```
close
```

Команда **rectangle** рисует прямоугольник, эллипс или фигуру близкую им по форме. Команда `rectangle('Position',[x,y,w,h])` рисует прямоугольник с вершиной в точке (*x,y*) шириной *w* высотой *h*. Свойство **Curvature** определяет кривизны сторон.

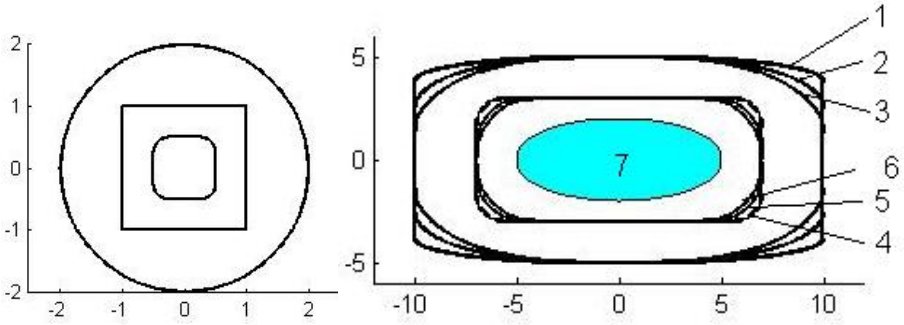
`rectangle('Position',[x,y,w,h], 'Curvature',[cg,cv])`

Если оно опущено или задано равным [0 0], то отрезки сторон будут прямые, и мы получаем прямоугольник. Задание значений кривизны в интервале от 0 до 1 искривляет стороны фигуры – от прямолинейных (при c_g=0 или c_v=0) до эллиптических (при c_g=1 или c_v=1). Первое число в паре $[c_g, c_v]$ управляет кривизной горизонтальных сторон, второе – вертикальных. Слово кривизна используется только для обозначения параметра функции и не является кривизной в точном математическом смысле этого значения. Можно использовать одно число, тогда оно относится к меньшей из сторон фигуры.

```
clf
rectangle('Position',[-2 -2 4 4], 'Curvature',[1,1],'LineWidth',2);
rectangle('Position',[-1 -1 2 2], 'LineWidth',2);
rectangle('Position',[-1/2 -1/2 1 1], 'Curvature',[0.5,0.5],'LineWidth',2);
axis equal
```

Построенные этими командами фигуры показаны на следующем рисунке слева. Фигуры, построенные следующими командами, показаны на рисунке справа. Номера кривых соответствуют номеру строки команды **rectangle** в коде.

```
clf
rectangle('Position',[-10,-5,20,10],'Curvature',[0.8,0.25], 'LineWidth',2);
rectangle('Position',[-10,-5,20,10],'Curvature',[0.8,0.5], 'LineWidth',2);
rectangle('Position',[-10,-5,20,10],'Curvature',[0.8,0.75], 'LineWidth',2);
 rectangle('Position',[-7,-3,14,6],'Curvature',0.2, 'LineWidth',2);
rectangle('Position',[-7,-3,14,6],'Curvature',0.5, 'LineWidth',2);
rectangle('Position',[-7,-3,14,6],'Curvature',1, 'LineWidth',2);
rectangle('Position',[-5,-2,10,4],'Curvature',[1,1],'FaceColor','c')
daspect([1,1,1])
xlim([-12,12])
ylim([-6,6]);
```



Команда **daspect([1,1,1])** определяет одинаковый масштаб единиц по осям. А команда **daspect([1 3 1])** означает, что одна единица по оси *x* равна по длине трем единицам по оси *y* и одной единице по оси *z*, т.е. по оси *y* происходит сжатие.

Опция **'FaceColor','c'** приводит к построению закрашенной фигуры цветом **'c'** – cyan. Команды **xlim([-12,12])** и **ylim([-6,6])** задают границы отображаемой области.

Команда **fill(x,y)** – рисует закрашенный многоугольник с x координатами вершин, заданными в первом векторе, и y координатами – во втором. При этом последняя вершина не обязана совпадать с первой. Третий аргумент задает цвет заливки многоугольника

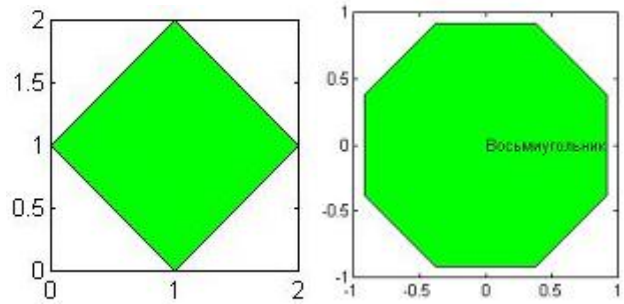
```
clf; fill([1 2 1 0],[0 1 2 1],'g'); daspect([1 1 1])
```

Фигура показана на следующем рисунке слева.

Код, приведенный далее, строит восьмиугольник, в середину графического окна выводит текст '**Восьмиугольник**', а в командное окно выводит форматированный текст с координатами одной из его вершин.

```
clf
t= (1/16:1/8:1)'*2*pi; x = sin(t); y = cos(t);
fill(x,y,'g'); axis square
text(0,0,'Восьмиугольник')
str=sprintf('Координаты второй вершины: x(2)=%6.2f y(2)=%6.2f',x(2),y(2));
disp(str);
```

```
Координаты второй вершины: x(2)=  0.92 y(2)=  0.38
```



Команда **text(x,y,'текст')** в точку с координатами `(x,y)` выводит заданный текст.

Функция **sprintf** создает строку, составленную из текста и отформатированных чисел. Формат задается символами **%** и буквой (в данном примере **f**). Между процентом и буквой могут стоять уточняющие символы – флаги и числа, определяющие ширину поля вывода. Переменные, значения которых будет записываться в строку, являются вторым, третьим и т.д. аргументами функции **sprintf**. Первой форматирующей последовательности символов **%...f** соответствует второй аргумент функции **sprintf** (имя первой переменной), второй последовательности **%...f** – третий аргумент (имя второй переменной) и т.д. В нашем примере мы сказали, что значения `x(2)` и `y(2)` должны быть отображены шестью цифрами, из которых две должны стоять после десятичной точки. Неиспользованные места цифр заполняются пробелами. Кроме буквы **f** для отображения чисел используется большое количество других символов – **g, d, e** и т.д. С ними вы можете познакомиться в справочной системе, например, выполнив команду

```
doc sprintf
```

Команда **disp(str)** выводит строку **str** в командное окно. При формировании текстовой константы ее следует заключать в одинарные кавычки.

1.3.2 Функции и кривые

Самая простая функция **plot** имеет много форм. Для построения одного графика она использует два вектора – первый вектор значений независимой переменной и второй – вектор значений функции в этих точках. Например, для построения графика функции `sin` от 0 до 2π выполним команды

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```

Мы построили два вектора **t** и **y**, которые передали в качестве аргументов функции **plot**. Обратите внимание, что для графика создается отдельное окно. MatLab автоматически присваивает каждому графику свой цвет. Вот как строится 3 графика.

```
y2 = sin(t-.25);
y3 = sin(t-.5);
plot( t, y, t, y2, t, y3)
```

Можно задавать цвет, стиль линий и маркеры следующим образом

plot(x, y, 'цвет стиль маркер')

(пробелы не обязательны). Цвета определяются символами c , m , y , r , g , b, w, k (черный). Тип линии определяются символами: – (минус) сплошная линия, -- (два минуса) разрывная линия, **:** (двоеточие) пунктирная линия, –**.** (минус точка) штрих пунктирная, '**none**' нет линии. Наиболее часто встречающиеся маркеры + , h , * , o.

```
plot(t,y, 'y:+')
```

Функция **plot** автоматически создает новое окно изображений, если до этого оно еще не было открыто. Повторные графики заменяют изображение в существующем окне.

Если надо создать новое окно, то используйте команду **figure** или **figure(n)**, которое n-е окно делает текущим. Номер окна отображается в его заголовке. Команда

```
hold on
```

включает режим добавления графиков в текущее окно

```
x=0:0.01:6;
y=1.2*sqrt(x);
plot(x,y)
```

Команда **hold off** снова включает режим замены графиков.

Команды

```
t=[0:0.1:2*pi];
plot(cos(t),sin(t))
```

рисуют окружность по ее параметрическому уравнению, но она выглядит как овал. Чтобы это исправить, надо управлять осями графика командой axis.

axis square - создает оси одинаковой длины (делает круг кругом)

axis equal - делает отметки приращений осей одинаковой длины (круг тоже будет кругом)

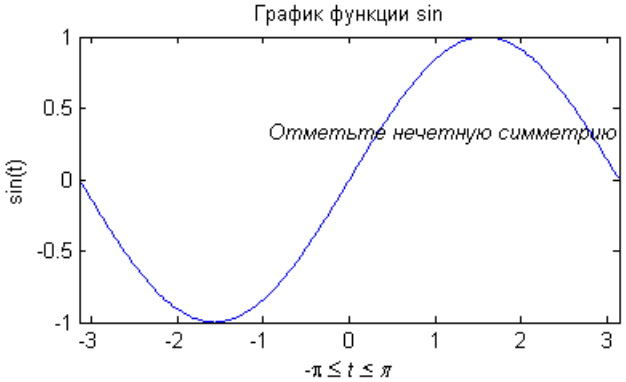
axis([xmin xmax ymin ymax]) - определяет диапазон отображаемой области

axis auto - возвращает значения по умолчанию

axis on, axis off - включают и выключают отображение осей
grid on, grid off - включают и выключают сетку координат

Функции **xlabel, ylabel, zlabel** добавляют подписи к осям, а **title** – добавляет заголовок в верхней части окна.

```
t = -pi:pi/100:pi;
y = sin(t);
plot(t,y)
axis([-pi pi -1 1])
xlabel( ' -\pi \leq \itt \leq \pi ' )
ylabel( ' sin(t) ' )
title('График функции sin')
text(-1,1/3,'\it {Отметьте нечетную симметрию}')
```



Обратите внимание, что функции, выводящие текст – **text, xlabel, ylabel,** распознают теги TeX – а.

Если вы имеете файл – функцию, то для построения ее графика можно использовать функцию **fplot**. При обращении к ней требуется указать имя этой файл – функции (в апострофах) или указатель на нее (с оператором @ перед именем функции) и границы отрезка для построения графика (в векторе из двух элементов).

Создадим в редакторе m – файлов функцию `myfun`

```
function f = myfun(x)
f=exp(-x).*sqrt((x.^2+1)./(x.^4+1));
```

и построим ее график командой

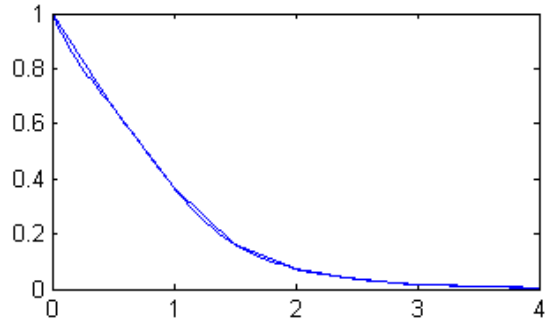
```
fplot('myfun',[0 4])
```

или командой

```
fplot(@myfun,[0 4])
```

Постройте графики **plot** и **fplot** на одних осях (используйте **hold on**) так, как показано на следующем рисунке.

```
x=0:0.5:4;
y=myfun(x);
plot(x,y)
hold on
fplot('myfun',[0 4])
```



График, построенный **fplot**, более точно отражает поведение функции, т. к. алгоритм **fplot** автоматически подбирает шаг аргумента, уменьшая его на участках быстрого изменения исследуемой функции.

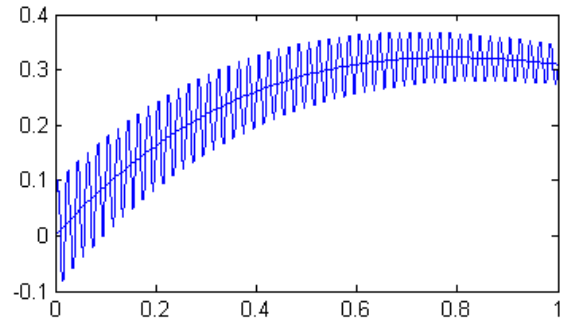
Повторим еще раз. Имеющаяся в нашем распоряжении файл-функция **myfun** позволяет обратиться к специальной функции **fplot**, которой требуется указать имя нашей файл – функции (в апострофах) или указатель на нее (с оператором @ перед именем функции).

При построении графика существенную роль играет выбор шага. Неудачный выбор шага может привести к неверному результату. Создадим, например, следующую файл – функцию

```
function z=myfun2(t)
z=exp(-t).*(sin(t)+0.1*sin(100*pi*t));
```

и построим ее графики с помощью **plot** и **fplot** на отрезке [0, 1]

```
x=0:0.01:1;
y=myfun2(x);
plot(x,y)
hold on
fplot('myfun2',[0 1])
```



Как видим, правильный выбор шага иногда важен.

Третьим аргументом функции **fplot** можно указать тип линии и маркера, так же, как и в **plot**. Положение маркеров покажет те точки, которые выбрала функция **fplot** для построения графика.

```
fplot('myfun',[0 4],'.-')
```

Аргументом функции **fplot** может выступать выражение, заключенное в кавычки

```
fplot('exp(-0.2*x)*sin(x^2)',[0,4*pi])
fplot('sin(x)^2',[0 12])
```

Можно использовать имена стандартных функций

```
fplot('sin',[0 12])
```

Вот пример одновременного построения двух графиков

```
fplot('[cos(x)^2,sin(x)]',[0,4*pi])
```

Отметим, что функция **fplot** бедна на опции форматирования вывода. Однако если мы принимаем результат функции **fplot**, то получаем пару векторов и функция **fplot** не рисует график. Потом график можно нарисовать обычным способом и задать любые опции, которые не поддерживает **fplot**.

```
[X,Y]=fplot('sin(x)/x',[-10,10]);
plot(X,Y,'LineWidth',3)
```

1.3.3 Поверхности

Для построения поверхностей в MatLab есть функции аналогичные **plot**, которые более сложны в использовании. Мы опишем их во второй части нашего пособия.

Есть простые в использовании функции, аналогичные функции **fplot**. Большинство из них начинается с префикса `ez`. Аналогом функции **fplot** в этой группе функций является функция **ezplot**. Ее первым аргументом является строковое выражение функции (в кавычках), а вторым – диапазон изменения независимой переменной.

```
ezplot('x^2*sin(x)',[0 40])
```

Отметим, что в выражениях для этой функции (и всех функций этой группы) следует использовать знаки арифметических операций без точек.

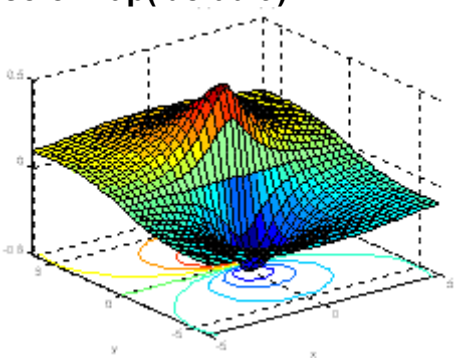
Для построения поверхностей в этой группе имеются функции **ezsurf, ezsurfc, ezmesh** и другие. Первым аргументом они принимают строковое выражение функции (в кавычках) двух переменных, а вторым – диапазон изменения этих переменных. Если диапазона нет, то он назначается системой автоматически. Вот несколько примеров построения поверхностей

```
ezsurfc('sqrt(x^2 + y^2)');
ezsurfc('sqrt(x^2 + y^2)',[-1 1]);
ezsurfc('sqrt(x^2 + y^2)',[-1 1 0 2]);
```

Отметим, что интервалы **[-1 1 0 2]** изменения переменных **[-1 1]** и **[0 2]** определяются порядком следования в алфавите имен аргументов **x** и **y**.

Следующий график строится над областью $-5 < x < 5, -2\pi < y < 2\pi$ на сетке 35 x 35

```
ezsurfc('y/(1 + x^2 + y^2)',[-5,5,-2*pi,2*pi],35)
colormap('default')
```

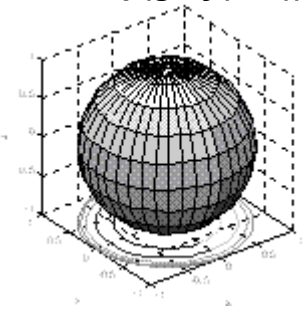


Функция **colormap('default')** определяет цветовую гамму рисунка, в данном случае цветовая гамма устанавливается стандартной.

Поверхность сферы по ее параметрическому уравнению можно построить так

```
ezsurfc('cos(v)*cos(u)','cos(v)*sin(u)','sin(v)',[-pi/2,pi/2,0,2*pi],21);
```

colormap(gray(128))

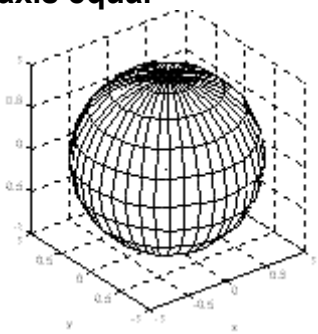


Обратите внимание, что на плоскости XY построены контурные линии. Использование функции **ezsurf** (без последней буквы c) построит ту же поверхность без контурных линий на плоскости XY

ezsurf('cos(v)*cos(u)','cos(v)*sin(u)','sin(v)',[-pi/2,pi/2,0,2*pi],21)

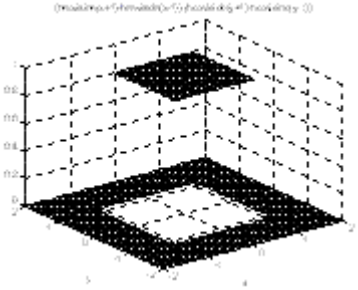
Каркасную поверхность сферы с удаленными невидимыми линиями можно послучить с помощью функции **ezmesh** следующим образом

ezmesh('cos(v)*cos(u)','cos(v)*sin(u)','sin(v)',[-pi/2,pi/2,0,2*pi],21)
colormap([0 0 0])
axis equal



Можно строить даже разрывные поверхности

ezsurfc('(heaviside(x+1)-heaviside(x-1))*(heaviside(y+1)-heaviside(y-1))',[-2,2],51)
ezmesh('(heaviside(x+1)-heaviside(x-1))*(heaviside(y+1)-heaviside(y-1))',[-2,2],51)
colormap([0 0 0])

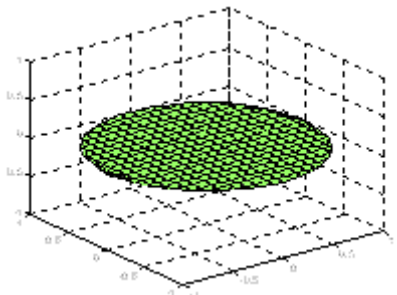


Можно нарисовать плоскую область как плоскую поверхность с краем, например, круг

ezmesh('u','(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2','0',[-1,1,-1,1],21)

или

ezsurf('u','(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2','0',[-1,1,-1,1],21)



или поверхность над ним

ezmesh('u','(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2','u^2+((abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2)^2',[-1,1,-1,1],21)

Но если областью изменения аргументов функции является круг, то проще использовать опцию этих функций **'circ'**

ezmesh('x*y','circ')

Функции группы ez... можно использовать и со стандартными файл – функциями MatLab, но при этом в формулах следует использовать знаки поэлементных арифметических операций (с предшествующей точкой). Для примера создадим следующую m – функцию

```
function x=xellipse(u,v)
x=2*cos(v).*cos(u);
```

Ее график можно построить следующим образом

ezsurf('xellipse',[0 2*pi -pi/2 pi/2])

У этой функции есть один недостаток – вместо числа 2 хотелось бы иметь аргумент, которому можно было бы присваивать произвольные значения. Но использование функции с тремя аргументами при вызове функции **ezsurf** приведет к ошибке. Изменим тело функции следующим образом

```
function x=xel(u,v,a)
x=a.*cos(v).*cos(u);
```

Чтобы построить ее график надо создать локально функцию двух переменных **@(u,v)xel(u,v,2)** и использовать ее при вызове графической функции

ezsurf(@(u,v)xel(u,v,2), [0 2*pi -pi/2 pi/2])

Создадим еще две аналогичные функции

```
function y=yel(u,v,b)
y=b.*cos(v).*sin(u);
```

и

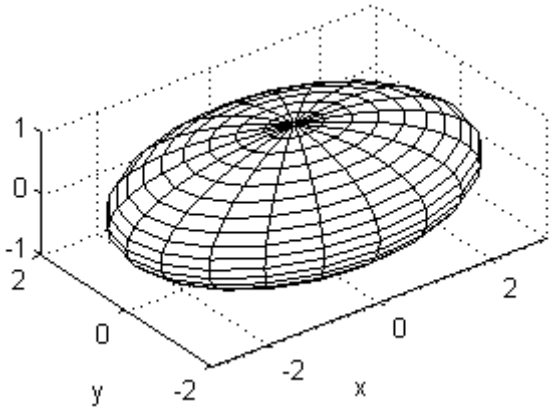
```
function z=zel(u,v,c)
z=c.*sin(v);
```

Тогда можно построить поверхность эллипсоида по его параметрическому уравнению, подставляя произвольные значения полуосей

ezsurf(@(u,v)xel(u,v,3),@(u,v)yel(u,v,2),@(u,v)zel(u,v,2), [0 2*pi -pi/2 pi/2])

или каркасную поверхность эллипса

ezmesh(@(u,v)xel(u,v,3),@(u,v)yel(u,v,2),@(u,v)zel(u,v,1), [0 2*pi -pi/2 pi/2])
axis equal
colormap([0 0 0])



Функции, созданные с помощью дескриптора @, можно присваивать идентификаторам и использовать их имена при вызове графических функций

x=@(u,v)xel(u,v,3) ;
y=@(u,v)yel(u,v,2);
z=@(u,v)zel(u,v,1);
ezmesh(x,y,z, [0 2*pi -pi/2 pi/2],21)

Функции можно создавать сразу по месту, без создания m – файлов и строить графики поверхностей

x=@(u,v) u ;
y=@(u,v) (abs(sqrt(1-u.^2)+v)-abs(sqrt(1-u.^2)-v))/2;
z=@(u,v) u.^2+v.^2;
ezmesh(x,y,z,[-1 1],21)

или, используя построенные функции `x(u,v)`, `y(u,v)`,

ezmesh(x,y,'0',[-1 1],21)

В следующем упражнении построим параллелепипед по его параметрическим уравнениям. Каждые несколько строк будем пояснять рисунком. Чтобы рядом построить несколько графиков используется функция **subplot(m,n,k)**, где **m** – количество рядов графиков, **n** – количество столбцов, **k** – номер текущего графика (тот в который следующая графическая функция будет выводить результат). Графики нумеруются построчно. Результат всех построений мы собрали в следующий файл сценарий

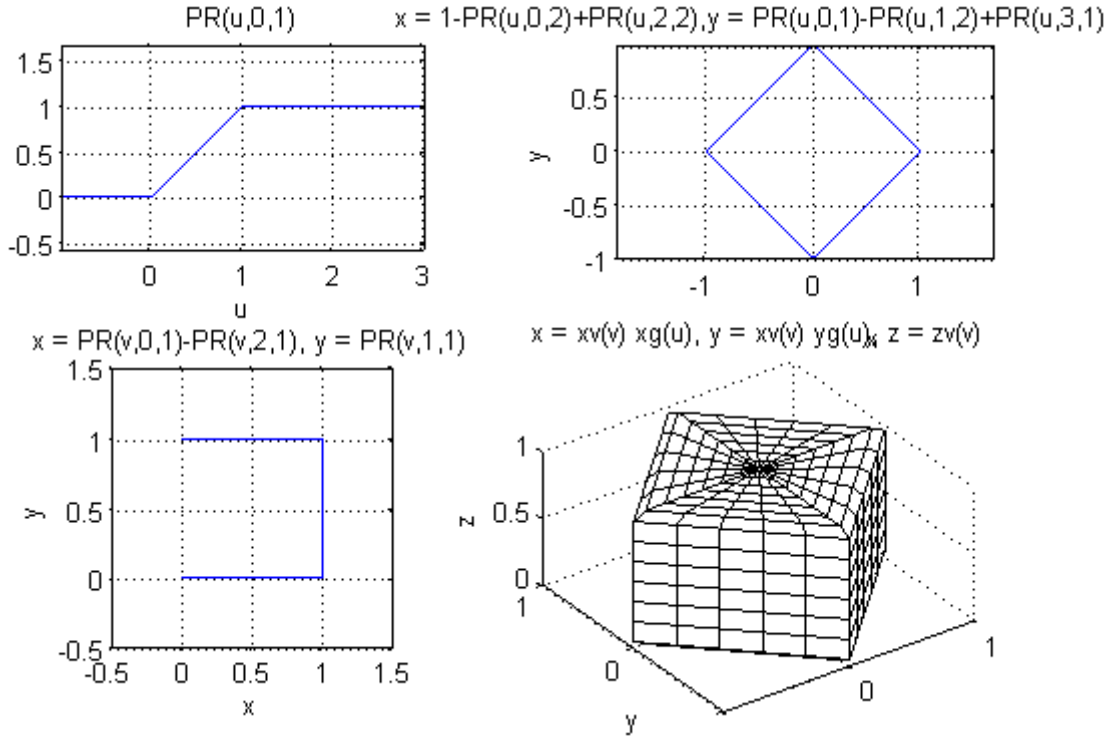
```
PR=@(x,a,w) (w+abs(x-a)-abs(x-a-w))/2;
subplot(2,2,1);ezplot(@(u)PR(u,0,1),[-1 3]);
xg=@(u) 1-PR(u,0,2)+PR(u,2,2);
yg=@(u) PR(u,0,1)-PR(u,1,2)+PR(u,3,1);
subplot(2,2,2);ezplot(xg,yg,[0 4]); axis equal;
xv=@(v) PR(v,0,1)-PR(v,2,1);
zv=@(v) PR(v,1,1);
subplot(2,2,3); ezplot(xv,zv,[0 3]); axis equal;
xs=@(u,v) xv(v).*xg(u);
ys=@(u,v) xv(v).*yg(u);
zs=@(u,v) zv(v);
subplot(2,3,6);ezmesh(xs,ys,zs,[0 4 0 3],21);colormap([0 0 0]);axis equal;
```

Мы рекомендуем вам вначале набирать и выполнять этот код по несколько строк. Вначале введите и выполните первые две строки – получите график функции **PR(u,0,1)**, затем еще три строки – получите квадрат, построенный по его параметрическому уравнению x=xg(u), y=yg(u). Заметим, что полезно построить отдельно график каждой из координатных функций xg(u) и yg(u).

Следующие три строки вводят и строят ломаную в форме скобы. Последние четыре строчки вводят координатные функции параметрического уравнения поверхности параллелепипеда, и по ним функция **ezmesh** строит каркас поверхности. После отладки строк кода скопируйте их в редактор m – файлов и сохраните под именем parallelepiped.m. Затем выполните команду

parallelepiped

В результате вы увидите графическое окно с четырьмя рисунками



Использование функции двух переменных f(x,y) при вызове функции **ezplot** позволяет построить кривую f(x,y)=0 по ее неявному уравнению

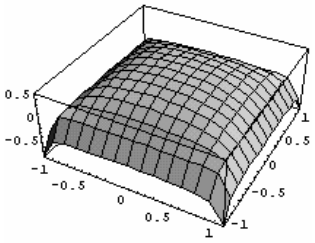
w=@(x,y) 1-x.^2-y.^2;
ezplot(w,[-1 1 -1 1]); axis equal

В следующем примере строится квадрат по его неявному уравнению w(x,y)=0

w=@(x,y) 2-x.^2-y.^2-sqrt((1-x.^2).^2+(1-y.^2).^2);
ezplot(w,[-1.2,1.2])

Заодно, посмотрите как выглядит поверхность функции z=w(x,y)

ezmesh(w,[-1.2 1.2 -1.2 1.2])
colormap([0 0 0])



Несколько полезных графических команд

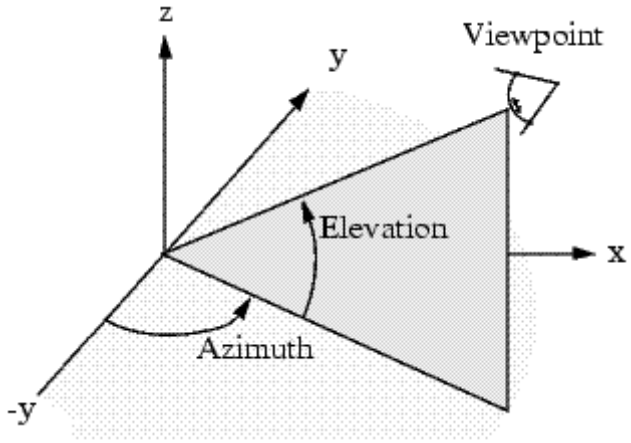
Команда

newplot

создает пустое графическое окно. Команда **view** позволяет посмотреть на трехмерный рисунок с другого направления. Например

view(2) % вид со стандартного направления 2
view(3) % вид со стандартного направления 3

Команда **view(Azimuth,Elevation)** определяет вид с направления, задаваемого углом **Azimuth,** отсчитываемым вокруг оси **z** и углом **Elevation**, отсчитываемым от плоскости XY (см. рисунок). Углы задаются в градусах.



Например
view(60,-45)
Команда
colorbar
выводит рядом с рисунком столбик цветовой палитры. Команда
shading interp
сглаживает цвета на трехмерном графике.

Команда **box on** отображает прямоугольник или прямой параллелепипед вокруг графической области. Команда **box off** стирает охватывающий прямоугольник или параллелепипед. Обычно команду **box** полезно использовать, когда строятся трехмерные фигуры.

1.3.4 Анимация

При изучении движения точки на плоскости или в трехмерном пространстве полезно не только построить траекторию точки, но и следить за движением точки по траектории. MATLAB предоставляет возможность получить анимированный график, на котором кружок, обозначающий точку, перемещается на плоскости или в пространстве, оставляя за собой след в виде линии — траектории движения. График похож на летящую комету с хвостом. Поэтому для построения таких анимированных графиков применяются функции **comet** и **comet3**.

```
t=[0:0.001:10];
x=sin(t)./(t+1);
y=cos(t)./(t+1);
comet(x,y)
```

При выполнении последней команды следите за тем, чтобы окно с графиком было поверх остальных окон. Обратите внимание, что при изменении размеров графического окна или при его минимизации и последующем восстановлении траектория движения пропадает. Это связано со способом, который применяет MATLAB для построения графика.

Скоростью движения кружка можно управлять, задавая различные шаги при создании вектора. Использование **comet** с одним аргументом (вектором) приводит к построению динамически рисующегося графика значений компонентов вектора в зависимости от их номеров.

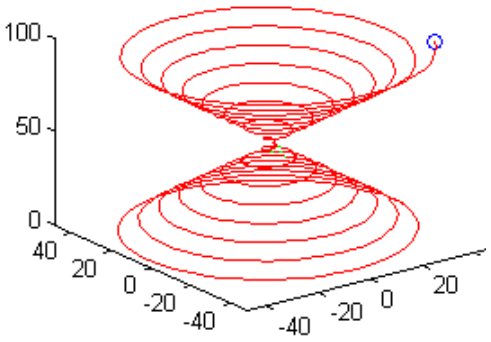
Для построения траектории точки, перемещающейся в пространстве, используется функция **comet3**. Для примера создадим следующий M - файл

```
t=[0:0.1:100];
x=cos(t).*abs(t-50);
y=sin(t).*abs(t-50);
z=t;
comet3(x,y,z)
```

и сохраним его под именем `mycomet.m`. Затем в командном окне выполним команду

mycomet

Последний кадр этой анимации показан на следующем рисунке



Графическая функция **ezplot3** имеет опцию '**animate**', которая приводит к движению небольшого кружка по кривой, построенной с ее использованием

ezplot3('sin(t)', 'cos(t)', 't',[0,6*pi],'animate')

Для построения анимации в общем случае MatLab предполагает использование функции **getframe**, которая захватывает графические кадры и может сохранять их в массив. Другая функция **movie** проигрывает такой массив в графическом окне, создавая, тем самым, анимацию. Вот простой пример

```
figure
t=0:0.01:2*pi;
set(gca,'nextplot','replacechildren');   % устанавливает режим замены кадров
                                         в графическом окне
for j = 1:20
    x=sin(j.*t).*exp(-t);
    plot(t,x);
    F(j)=getframe;          % запоминает текущий графический кадр
end
movie(F,10)                 % Проигрывает массив F десять раз
```

Внутри цикла может стоять любая графическая функция. Созданные с ее помощью кадры будут сохраняться в массив F и проигрываться функцией **movie**.

Вот пример из справочной системы MatLab. Создайте следующий файл – сценарий с именем `mymovie.m`

```
Z= peaks;
surf(Z);
axis tight
set(gca,'nextplot','replacechildren');
% Record the movie
for j = 1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j) = getframe;
end
movie(F,20)       % Play the movie twenty times
```

и выполните команду

mymovie

Некоторые другие графические возможности системы MatLab мы опишем в следующем параграфе. Об операторах циклов и функции **set** мы будем подробно говорить во второй части пособия.

1.4 СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ

1.4.1 Использование символьных переменных

Перед использованием символьные переменные и функции в MatLab должны быть объявлены как символьные. Это отличает MatLab от таких программ как Mathematica или Maple, где переменные, если им не присвоено никаких значений или присвоены значения выражений, составленных из числовых констант, рассматриваются как символьные. Объявление символьных переменных выполняется командой **syms** . Например

syms x a b

Здесь мы создали три символьных переменных. Символьные выражения конструируются из символьных переменных

f=(sin(x)+a)^2*(cos(x)+b)^2/sqrt(abs(a+b))

```
f =
(sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)
```

Выражение **f** автоматически становится символьным. Чтобы увидеть выражение в более привычном виде можно использовать функцию **pretty**

pretty(f)

```
                  2             2
          (sin(x) + a)  (cos(x) + b)
          --------------------------
                        1/2
                | a + b |
```

Символьную функцию можно создать без предварительного объявления переменных при помощи команды **sym**, входным аргументом которой является строка с выражением, заключенная в апострофы

z = sym('c^2 / (d + 1) ')

Эта же функция **sym** может быть использована для объявления символьных переменных. Команда **syms a, b, c** эквивалентна последовательности

a=sym('a');b=sym('b');c=sym('c');

Функция **sym** может использоваться для работы с символьными константами

sym(2) / sym(5) + sym(1) / sym(3)

```
ans =
11/15
```

syms x; x+3.1

```
ans =
x+31/10
```

Символьные переменные могут являться элементами матриц и векторов. Элементы строк матриц при вводе отделяются пробелами или запятыми, а столбцов — точкой с запятой, так же как и для обычных матриц. В результате образуются символьные матрицы и векторы, к которым применимы матричные и поэлементные операции.

syms a b c d e f g h

A=[a b; c d]

```
A =
[ a, b]
[ c, d]
```

B=[e f; g h]

```
B =
[ e, f]
[ g, h]
```

C=A*B

```
C =
[ a*e+b*g, a*f+b*h]
[ c*e+d*g, c*f+d*h]
```

F=A.*B

```
F =
[ a*e, b*f]
[ c*g, d*h]
```

Конструирование блочных символьных матриц не отличается от числовых, требуется следить за размерами соответствующих блоков:

D=[C A; B C]

```
D =
[ a*e+b*g, a*f+b*h,       a,       b]
[ c*e+d*g, c*f+d*h,       c,       d]
[       e,       f, a*e+b*g, a*f+b*h]
[       g,       h, c*e+d*g, c*f+d*h]
```

Обращение к элементам символьных матриц и векторов производится при помощи индексации, в том числе двоеточием и вектором со значениями индексов. Например, выделим вторую строку предыдущей матрицы

d2=D(2,:)

```
d2 =
[ c*e+d*g, c*f+d*h, c, d]
```

Из полученного вектора выберем 1 – й, 3 – й и 4 – й элементы

d=d2([1 3 4])

```
d =
[ c*e+d*g,  c,  d]
```

Для удаления строк или столбцов символьных матриц используется пустой массив

D(1:2,:)=[ ]

```
D =
[   e,   f, a*e+b*g, a*f+b*h]
[   g,   h, c*e+d*g, c*f+d*h]
```

Для преобразования значения числовой переменной в символьную служит функция **sym**. Введите массив типа *double array* (обычная матрица MatLab)

A=[1.3 -2.1 4.9
6.9 3.7 8.5]

```
A =
  1.3    -2.1   4.9
  6.9     3.7   8.5
```

Здесь перенос ввода во вторую строку (в командном окне) выполняется нажатием комбинации клавиш Shift – Enter. Затем образуйте соответствующий символьный массив

B=sym(A)

```
B =
[  13/10, -21/10,  49/10]
[  69/10,  37/10,   17/2]
```

Создайте символьный вектор – столбец **d**

c=[3.2; 0.4; -2.1]

```
c =
    3.2
    0.4
   -2.1
```

d=sym(c)

```
d =
   16/5
    2/5
 -21/10
```

Умножьте матрицу **B** на вектор **d** – результат является символьной переменной, причем все вычисления проделаны над рациональными дробями.

e=B*d

```
e =
 -697/100
  571/100
```

Символьные операции позволяют находить значение выражений и символьных констант со сколь угодно большой точностью. В частности значение рациональных дробей, функций от целых и рациональных аргументов, от числа π и т.д. можно получить с любой степенью точности, т.е. найти сколь угодно много значащих цифр результата. Для этого предназначена функция **vpa**:

c=sym('sqrt(2)'); cn=vpa(c)

```
cn =
1.4142135623730950488016887242097
```

По умолчанию удерживается 32 значащие цифры. Второй дополнительный входной параметр **vpa** служит для задания точности:

cn=vpa(c,60)

```
cn =
  1.41421356237309504880168872420969807856967187537694807317668
```

pp=sym('pi'); vpa(pp,60)

```
ans =
  3.14159265358979323846264338327950288419716939937510582097494
```

Второй аргумент **vpa** задает удерживаемое число значащих цифр только для данного вызова **vpa**. Важно понимать, что выходной аргумент функции **vpa** является символьной переменной. Для перевода символьных переменных в числовые, т. е. в переменные типа *double array*, используется функция **double**

cnum=double(cn)

```
cnum =
  1.4142135623731
```

Переменная **cn** выглядела как число. Однако она являлась символьной переменной и ее можно использовать в арифметических операциях только с другими символьными переменными. Переменную **cnum** можно использовать в обычных выражениях MatLab.

При необходимости очистить рабочее пространство от всех символьных переменных следует использовать команду

maple restart

1.4.2 Графическое представление символьных функций

Построение графика символьной функции одной переменной осуществляется при помощи функции **ezplot**. Самый простой вариант использования **ezplot** состоит в указании символьной функции в качестве единственного входного аргумента. В этом случае в графическое окно выводится график функции на отрезке [-2л, 2л]

```
f=sym('x^2*sin(x)');
ezplot(f)
```

Обратите внимание, что автоматически создается соответствующий заголовок. По умолчанию в качестве отрезка, на котором строится график, принимается промежуток пересечения области определения функции и интервала $[-2\pi, 2\pi]$. Вторым аргументом может быть задан вектор с границами отрезка, на котором требуется построить график функции

```
ezplot(f,[-3,2])
```

Другие графические функции группы ez… также можно использовать для построения графиков символьных функций. Вот несколько примеров

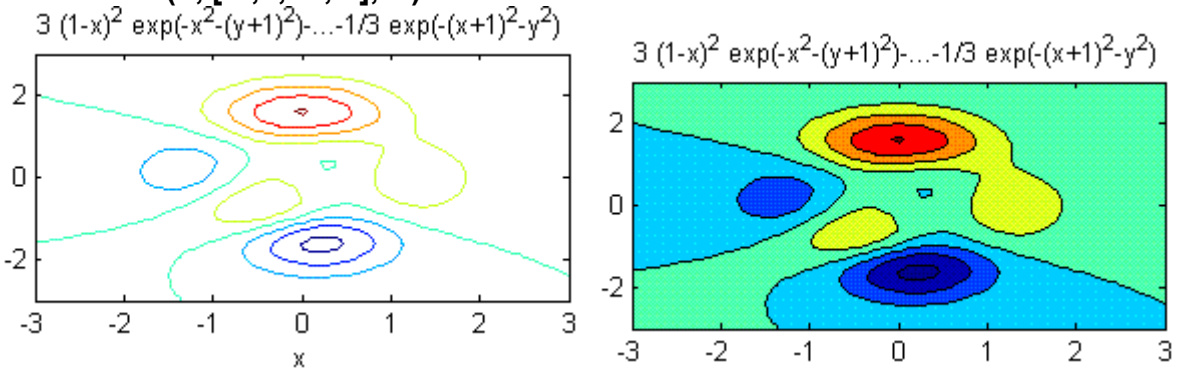
```
syms s t; ezmesh(exp(-s)*cos(t),exp(-s)*sin(t),t,[0,8,0,4*pi])

syms u v; ezmesh(u,(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2,u,[-1,1,-1,1],21)

syms x; ezplot(erf(x)); grid

syms t; ezplot3(sin(t), cos(t), t,[0,6*pi])

syms t; ezpolar(1+cos(t))          % график в полярных координатах

f=sym('x^2 +y^2'); ezsurf(f,[-1 1 -1 1])
```

Функция **ezsurf** отображает график символьной функции только для допустимых значений аргументов, остальные значения отбрасываются, что позволяет исследовать область определения функции двух переменных. Например

```
ezsurf('asin(x^y)',[0 6 -7 7])
```

Функции **ezcontour, ezcontourf** используются для построения контурных графиков – линий уровня функций двух переменных. Второй параметр задает прямоугольную область изменения аргументов, третий `n` – количество ячеек сети `n x n` на которую разбивается область изменения аргументов (по – умолчанию `n=60`)

```
syms x y
f = 3*(1-x)^2*exp(-(x^2)-(y+1)^2) ...
- 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) ...
- 1/3*exp(-(x+1)^2 - y^2);
ezcontour(f,[-3,3],49)
ezcontourf( f, [-3,3, -3, 3],60)
```



1.4.3 Алгебраические преобразования

Алгебраические преобразования с символьными переменными рассмотрим на примерах.

```
syms x; 27*x+12*x                % упрощение символьного выражения
ans =
39*x
```

Операции с полиномами реализуют функции **collect**, **expand**, и **factor**. Функция **expand** раскрывает скобки

```
p=sym('(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3'); expand(p)
ans =
x^4+4*a*x^3+6*x^2*a^2+4*x*a^3+a^4+x^3-4*x^2+4*x-4+a*x-a^2
syms a b; expand((a+b)^5)
ans =
a^5+5*a^4*b+10*a^3*b^2+10*a^2*b^3+5*a*b^4+b^5
```

Аргумент **expand** может быть не только полином, но и символьным выражением, содержащим тригонометрические, экспоненциальную и логарифмическую функции

```
f=sym('sin(arccos(3*x))+exp(2*log(x))'); fe=expand(f)
fe =
(1-9*x^2)^(1/2)+x^2
```

Вычисление коэффициентов при степенях независимой переменной производится с использованием функции **collect**. Введите следующий полином

```
p=sym('(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3')
p =
(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3
pc=collect(p)
pc =
x^4+(1+4*a)*x^3+(-4+6*a^2)*x^2+(4+a+4*a^3)*x+a^4-4-a^2
```

По умолчанию в качестве переменной выбирается x, однако можно было считать, что a — независимая переменная, а x входит в коэффициенты полинома, зависящего от a. Второй аргумент функции **collect** предназначен для указания переменной, при степенях которой следует найти коэффициенты

```
pca=collect(p,'a')
```

```
pca =
a^4+4*x*a^3+(-1+6*x^2)*a^2+(4*x^3+x)*a+x^4+(x-1)^3-x^2-3+x
```

Символьные полиномы разлагаются на множители функцией **factor**, если получающиеся множители имеют рациональные коэффициенты

p=sym('x^5+13*x^4+215/4*x^3+275/4*x^2-27/2*x-18');
pf=factor(p)

```
pf =
1/4*(2*x+1)*(2*x-1)*(x+6)*(x+4)*(x+3)
```

syms a b; factor(a^2+2*a*b+b^2)

```
ans =
(a+b)^2
```

factor(a^2-4/9)

```
ans =
1/9*(3*a-2)*(3*a+2)
```

Функция **factor** умеет работать и с выражениями некоторых других типов, например, с тригонометрическими выражениями

syms x
factor(sin(x)^2-cos(x)^2)

```
ans =
(sin(x)-cos(x))*(sin(x)+cos(x))
```

Преобразование выражений общего вида производится при помощи функции **simplify**, которая реализует алгоритм упрощения выражений, содержащих тригонометрические, экспоненциальную, логарифмическую функции, и некоторые специальные функции. Кроме того, **simplify** способна преобразовывать выражения, содержащие суммирование и интегрирование.

rho = sym('(1 + sqrt(5))/2'); f = rho^2 - rho - 1

```
f =
(1/2+1/2*5^(1/2))^2-3/2-1/2*5^(1/2)
```

simplify(f)

```
ans=
0
```

simplify(sin(x)^2+cos(x)^2)

```
ans =
1
```

simplify((x^2+a*x+b*x+a*b)/(a+x))

```
ans =
x+b
```

Для вычисления значения выражения в точке следует использовать функцию подстановки в виде **R = subs(S,old,new)**

syms x; f = 2*x^2 - 3*x + 1;subs(f,2)

```
ans =
3
```

Здесь MatLab сам определяет имя символьной переменной в выражении **f** и заменяет ее заданным значением 2. Если символьных переменных несколько, то следует указать имя переменной, которая должна быть заменена значением

syms a x; f=a*x^2-x; subs(f,x,2)

```
ans =
4*a-2
```

Одновременная замена нескольких символьных переменных может быть выполнена следующим образом

syms a b; subs(cos(a)+sin(b),{a,b},{sym('c'),2})

```
ans =
cos(c)+sin(2)
```

В символьное выражение вместо символьной переменной можно подставить вектор или матрицу. Результатом будет объект той же размерности (вектор или матрица). Вот примеры табулирования функции

```
y=subs('x^2','x',[0 1 2 3])
y=
    0  1  4  9
syms x y; subs(x^2+y^2,x,[0 1 2])
ans =
[   y^2, 1+y^2, 4+y^2]
syms x y ; subs(x^2+y^2,{x,y},{[0 1 2],[3 4 1]})
ans =
     9    17     5
subs(x*y,{x,y},{[0 1;-1 0],[1 -1;-2 1]})
ans =
     0    -1
     2     0
```

Для вычисления композиции двух функций используется функция **compose**

```
syms x y z t u; f = 1/(1 + x^2); g = sin(y); compose(f,g)
ans =
1/(1+sin(y)^2)
compose(f,g,z)          % возвращает f(g(z))
ans =
1/(1+sin(z)^2)
```

Если задана функция *f(x)*, то обратная функция $g = f^{-1}$ удовлетворяет равенству $g(f(x)) = x$. Символьное представление обратной функции может быть получено с помощью функции **finverse**

```
syms u v ; finverse(exp(u-2*v),u)
ans =
2*v+log(u)
```

Для вычисления целой и дробной части символьного числа можно использовать следующие функции: **fix** – округление до целого в сторону нуля; **floor** – вычисление наибольшего целого, не превосходящего данное; **round** – округление до ближайшего целого; **ceil** – вычисление ближайшего большего целого; **frac** – вычисление дробной части числа **frac(X) = X - fix(X)**

```
x = sym(-5/2); [fix(x) floor(x) round(x) ceil(x) frac(x)]
ans =
[   -2,   -3,   -3,   -2, -1/2]
```

MatLab хорошо умеет работать с разрывными функциями

```
F = 2/3*atan(1/3*tan(1/2*x))
ezplot(F)
J = sym(2*pi/3)*sym('round(x/(2*pi))');
ezplot(J,[-6 6])        % ступенчатая функция
F1 = F+J
ezplot(F1)              % плавно возрастающая
```

1.4.4 Элементы математического анализа

MatLab умеет выполнять основные операции математического анализа, в частности, дифференцирование и интегрирование символьных выражений

```
diff(x^4,x)
ans =
```

```
4*x^3
```
diff(x^4,x,3) % вычислить третью производную
```
ans =
24*x
```
y1=sin(x)/x; limit(y1,x,0) % $\lim_{x\to 0}\frac{\sin x}{x}$
```
ans=1
```
y2=(1-exp(-x))/x; limit(y2,x,inf) % $\lim_{x\to \infty}\frac{1-e^{-1}}{x}$
```
ans=0
```
Для дифференцирования постоянной ее следует объявить символьной
c = sym('5'); diff(c)
```
ans =
0
```
Однако, простое дифференцирование числа дает
diff(5)
```
ans =
     []
```
int((x+a)^3,x) % вычислить неопределенный интеграл
```
ans =
1/4*(x+a)^4
```
int(-2*x/(1+x^2)^2)
```
ans =
1/(1+x^2)
```
syms x z; int(x/(1+z^2),z) % интегрирование по переменной z
```
ans =
x*atan(z)
```
sym s t; int(2*x, sin(t), 1) % здесь `sin(t)` нижний предел интегрирования
```
ans =
1-sin(t)^2
```
При вычислении интегралов может получиться следующая ситуация
syms a x; f = exp(-a*x^2); int(f,x,-inf,inf)
```
ans =
PIECEWISE([1/a^(1/2)*pi^(1/2), csgn(a) = 1],[Inf, otherwise])
```
Не зная ничего о символьной переменной **a**, MatLab вернул два результата – первый получается, когда интеграл сходится, второй – когда расходится. Но если объявить переменную **a** положительной, то сразу получаем результат
syms a positive; f = exp(-a*x^2); int(f,x,-inf,inf)
```
ans =
1/a^(1/2)*pi^(1/2)
```
Функция **diff** умеет дифференцировать выражения любой степени сложности. Однако она не всегда возвращает результат в самой простой форме. Тогда результат следует упростить
diff(sin(x)^2-cos(x)^2,x)
```
ans =
4*sin(x)*cos(x)
```
simplify(diff(sin(x)^2-cos(x)^2,x))
```
ans =
4*sin(x)*cos(x)
```
Функция **simplify** не справилась со своей задачей упрощения. Однако функция **simple**, которая пытается найти представление выражения с меньшим числом символов, чем исходное, получила более простой результат
simple(diff(sin(x)^2-cos(x)^2,x))

```
. . .
ans =
2*sin(2*x)
```
Функция **simplify** срабатывает, когда в ее аргументе стоит неопределенный интеграл
```
simplify(x^3-int(x^2,x))
ans =
2/3*x^3
```
Символьное ядро MatLab хорошо умеет работать с конечными и бесконечными суммами и рядами. Вот примеры суммирования рядов
```
syms x k ; s1 = symsum(1/k^2,1,inf)
s1 =
1/6*pi^2
s2 = symsum(x^k,k,0,inf)
s2 =
-1/(x-1)
symsum(k^2,0,10)
ans =
385
symsum(x^k/sym('k!'), k, 0,inf)
ans =
exp(x)
```
Разложение функций в ряд Тейлора также возможно
```
syms x; f = 1/(5+4*cos(x)); T = taylor(f,8)
T =
1/9+2/81*x^2+5/1458*x^4+49/131220*x^6
```

С некоторыми символьными функциями работать не очень удобно. Например, чтобы вычислить факториал (символьный, следовательно с любым количеством значащих цифр) надо выполнить пару команд
```
kfac = sym('k!');
subs(kfac,k,50)
ans =
30414093201713378043612608166064768844377641568960512000000000000
```
В этом случае можно напрямую обратиться к ядру Maple
```
maple('50!')
ans =
30414093201713378043612608166064768844377641568960512000000000000
```
Для использования стандартных функций Maple их следует передавать функции **maple** как текстовую строку
```
maple('gcd(14, 21)')        % наибольший общий делитель
ans =
7
```
Обычно функция **maple** возвращает результат в символьном виде. Однако, если в результате выполнения возникает ошибка, то функция возвращает вектор из двух элементов: [результат сообщение], где сообщение – это строка с информацией о возникшей ошибке.
Команду **maple** допускается вводить по – разному. Следующие три команды эквивалентны
```
maple('evalf(Pi,50)')
maple evalf Pi 50
maple('evalf','Pi',50)
```

Отметим еще одну функцию, которая переводит символьное выражение в код LaTex

syms x; f = taylor(log(1+x)); latex(f)

```
ans =
x-1/2\,{x}^{2}+1/3\,{x}^{3}-1/4\,{x}^{4}+1/5\,{x}^{5}
```

Возможности символьных вычислений в MatLab весьма обширны. Более подробно с ними вы можете познакомиться по справочной системе. Получить справку по функциям Maple, встроенным в MatLab, или какой – либо теме можно командой

mhelp имя_функции
mhelp тема
mhelp('тема')

Список функций Maple возвращает команда

help mfunlist

1.5 АЛГЕБРАИЧЕСКИЕ ЗАДАЧИ

1.5.1. Полиномы

Полином в MATLAB задается вектором его коэффициентов, начиная с коэффициента при старшей степени. Например, для определения полинома $p = x^7 + 3.2x^5 - 5.2x^4 + 0.5x^2 + x - 3$ следует использовать команду

p=[1 0 3.2 -5.2 0 0.5 1 - 3];

Промежуточные нулевые коэффициенты должны содержаться в векторе.

Функция **polyval** предназначена для вычисления значения полинома от некоторого аргумента

polyval(p,1)

```
ans =
   -2.5000
```

Аргумент может быть матрицей или вектором, в этом случае производится поэлементное вычисление значений полинома и результат представляет матрицу или вектор того же размера, что и аргумент

polyval(p,[1 2;3 4])

```
ans =
  1.0e+003 *
   -0.0025    0.0726
    0.8473    4.5824
```

Функция **roots** возвращает вектор корней полинома, в том числе и комплексных. Аргументом функции **roots** является вектор коэффициентов полинома

roots(p)

```
ans =
  -0.5962 + 2.0565i
  -0.5962 - 2.0565i
   1.2450
   0.3103 + 0.6513i
   0.3103 - 0.6513i
  -0.6732
```

Убедимся в правильности работы функции roots, вычислив значение полинома в одном из корней

polyval(p,r(3))
```
ans =
 -1.3323e-015
```
Умножение двух полиномов осуществляется при помощи функции **conv**. Например, для вычисления произведения полиномов $p(x) = x^5 + x^3 + 1$ и $q(x) = x^2 + 2x + 3$ следует создать два вектора коэффициентов и использовать их в качестве аргументов **conv**:
p= [1 0 1 0 0 1] ;
q = [1 2 3] ;
s=conv(p,q)
```
s =
     1     2     4     2     3     1     2     3
```

Функция **deconv** осуществляет деление полиномов с остатком. Она вызывается с двумя выходными аргументами — частным и остатком от деления:
[d r]=deconv(p,q)
```
d =
     1    -2     2     2
r =
     0     0     0     0   -10    -5
```
Размер вектора, содержащего коэффициенты остатка, равен максимальному из размеров векторов, соответствующих делимому полиному и его делителю.

Для сложения и вычитания полиномов в MATLAB нет специальной функции. В то же время использование знака "+" для нахождения суммы полиномов разной степени приведет к ошибке, т. к. нельзя складывать векторы разных размеров. Вы можете самостоятельно написать функцию сложения полиномов. Она должна приводить полиномы к общей степени, и затем вектора одинакового размера, представляющие полиномы, можно сложить.

Функция **polyder** предназначена для вычисления производной не только от полинома, но и от произведения и частного двух полиномов. Вызов **polyder** с одним аргументом — вектором, соответствующим полиному, приводит к вычислению вектора коэффициентов производной полинома:
polyder(p)
```
ans =
     5     0     3     0     0
```
Для вычисления производной от произведения полиномов следует использовать **polyder** с двумя входными аргументами:
polyder(p,q)
```
ans =
     7    12    20     8     9     2     2
```
Для нахождения производной отношения двух полиномов, следует вызвать **polyder** с двумя выходными аргументами **[q, d] = polyder(b,a)**. Производная частного возвращается в виде двух полиномов `q` – числителя и `d` - знаменателя отношения **q / d**
b= [1 0 1 0 0 1] ;
a = [1 2 3] ;
[q,d]=polyder(b,a)

```
q =
     3     8    16     4     9    -2    -2
d =
     1     4    10    12     9
```

Для задания полинома в символьном виде используется команда **poly2sym**. Она принимает вектор коэффициентов и возможно имя символьной переменной

```
syms x
r = poly2sym([1 3 2],x)
ans =
x^2+3*x+2
```

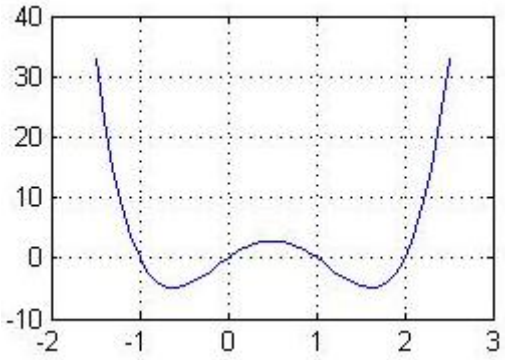
и возвращает символьное представление полинома.

Функция **poly** возвращает вектор (представление полинома в виде вектора) по его корням

```
pp=poly([1 2])
pp =
     1    -3     2
syms x;  poly2sym(pp,x)
ans =
x^2-3*x+2
```

Чтобы нарисовать график полинома надо задать вектор значений независимой переменной, вычислить его значения в этих точках и обратиться к команде **plot**

```
p=poly([-1 0 1 2])
x=-2:0.1:3;
y=polyval(p,x);
plot(x,y)
```



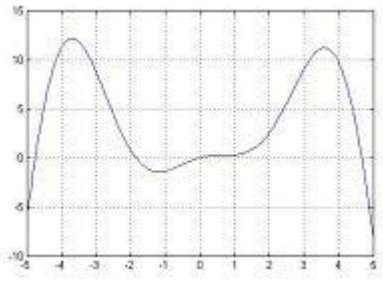
1.5.2. Нахождение корней и решение алгебраических уравнений

Функция **fzero** позволяет приближенно вычислить корень уравнения на некотором интервале или ближайший к заданному начальному приближению. В простейшем варианте **fzero** вызывается с двумя входными и одним выходным аргументом **x=fzero('funcname',x_0)**, где `funcname` имя файл – функции (левая часть уравнения $f(x)=0$), x_0 – начальное приближение корня. Для примера решим уравнение $\sin x - x^2 \cos x = 0$. Создадим m - функцию

```
function y = myf(x)
y = sin(x)-x.^2.*cos(x);
```

Посмотрим на ее график

```
fplot('myf',[-5 5])
```



Видим, что на отрезке [- 5 5] имеется 4 корня. Чтобы уточнить значение корня, например, вблизи 5 выполним команду

fzero('myf',5)

```
ans =
    4.6665
```

Алгоритм функции **fzero** по умолчанию находит корень уравнения с точностью **eps**

eps

```
ans =
  2.2204e-016
```

т.е. с точностью до двойки в шестнадцатом знаке после десятичной точки. Чтобы убедиться в этом, измените формат представления данных на **long e** и выведите значение корня

x=fzero('myf',5);
format long e
x

```
x =
    4.666499563444645e+000
```

Вместо начального приближения вторым параметром **fzero** можно задать интервал, на котором следует найти корень

x2=fzero('myf',[-3 -1])

```
x2 =
   -1.853927459696150e+000
```

На границах указываемого интервала функция должна принимать значения разных знаков, иначе появится сообщение об ошибке!

Функцию, корень которой определяется, можно задать при помощи указателя на нее

format long
x1 = fzero(@myf, [-5 -4])

```
x1 =
  -4.75655940570290
```

Вместо создания m – функции можно создать анонимную функцию

fun=@(x) sin(x)-x.^2.*cos(x)
x2 = fzero(fun, -2)

```
x2 =
  -1.85392745969615
```

На сеанс можно создавать временные функции одной переменной

ff=inline('x^2/2-x^4/4')

```
ff =
     Inline function:
     ff(x) = x^2/2-x^4/4
```

fplot(ff,[-2 2])

и определять корень с помощью функции **fzero**

fzero(ff,1)

```
ans =
    1.4142
```

Важной особенностью **fzero** является то, что она вычисляет только те корни, в которых функция меняет знак, а не касается оси абсцисс. Найти корень уравнения $x^2=0$ при помощи **fzero** не удается

fzero(@(x) x^2,0.001)

```
Exiting fzero: aborting search for an interval ...
```

Функция **fsolve** решает нелинейные алгебраические уравнения и их системы, заданные в виде $F(x)=0$, где F вектор – функция, а x – неизвестный вектор. Функция **fsolve(fun,x_0)** принимает имя функции **fun** и начальное значение искомого вектора $\mathbf{x_0}$. Для примера сравните корни функции **fun=@(x) sin(x)-x.^2.*cos(x)**, полученные с помощью **fzero** и **fsolve.**

x1=fzero(fun,5)

x2=fsolve(fun,5)

```
x1 =
    4.666499563444645e+000
Optimization terminated: first-order optimality is less than options. TolFun.
x2 =
    4.666499563444647e+000
```

Для решения системы следует создать функцию, возвращаемым значением которой является вектор, т.е. вектор – функцию. Решим, например, систему

$$\begin{cases} x(2-y)-e^{y}\cdot\cos x=0 \\ x-y-\cos x-e^{y}=-2 \end{cases}$$

Создаем функцию, которая возвращает вектор из двух значений

```
function F=FN(x)
% x  вектор аргументов, F - вектор значения
F(1)=x(1)*(2-x(2))-cos(x(1))*exp(x(2));
F(2)=2+x(1)-x(2)-cos(x(1))-exp(x(2));
```

Решаем систему командой

[x,f]=fsolve(@FN, [0 0])

```
Optimization terminated: first-order optimality is less than options. TolFun.
x =
    0.7391    0.4429
f =
  1.0e-011 *
   -0.4702   -0.6404
```

Здесь **[0 0]** начальное приближение. Найденное решение возвращается в векторе x, а значение – в векторе f. Посмотреть параметры настроек (точность и т.д.) функции **fsolve** можно командой

optimset fsolve

```
ans =
        Display: 'final'
    MaxFunEvals: '100*numberofvariables'
        MaxIter: 400
         TolFun: 1.000000000000000e-006
           TolX: 1.000000000000000e-006
                 ...
```

Здесь мы не приводим всех опций, поскольку их более 50. Это могут быть указания на метод решения, точность, количество итераций и т.д. За

подробным описанием смысловой нагрузки опций обращайтесь к справочной системе.

Для изменения одной или нескольких опций мы также используем функцию **optimset** в формате **opt = optimset('OptionName','OptionValue',…).** Возвращенное функцией множество параметров **opt** нужно передать третьим аргументом функции **fsolve.**

Найдем матрицу, удовлетворяющую уравнению

$$X \cdot X \cdot X = \begin{bmatrix} 37 & 54 \\ 81 & 118 \end{bmatrix}$$

Создаем m – функцию

```
function F = FNM(x)
F = x*x*x-[37,54;81,118];
```

Следующая последовательность команд решает нашу задачу

```
x0 = ones(2,2)                  % Начальное приближение
opt = optimset('Display','off'); % Отключить вывод сообщения
[x,F] = fsolve(@FNM,x0,opt)
x0 =
     1     1
     1     1
x =
    9.999999999041618e-001    2.000000000043856e+000
    3.000000000065749e+000    3.999999999969842e+000
F =
   -4.013145371573046e-011    1.733724275254645e-011
    2.506794771761634e-011   -1.691091711109038e-011
```

Полученный вектор **x** близок к точному значению $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ (установлен формат вывода **long e**). Если вы установите формат вывода **short**, то увидите точное решение

```
format short
x
x =
    1.0000    2.0000
    3.0000    4.0000
```

Можно решать системы линейных уравнений с использованием функции **fsolve**. Решим для примера систему

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

Создадим m – функцию

```
function F = FNL(x)
% x  вектор аргументов, F - вектор значения
F(1)=x(1)+x(2)+x(3)-6;
F(2)=2*x(1)-x(2)-x(3)+3;
F(3)=3*x(1)-2*x(2)+x(3)-2;
```

Тогда последовательность команд напечатает нам решение задачи

```
x0=[0 0 0]
opt = optimset('Display','off');  % Отключить вывод сообщения
x = fsolve(@FNL,x0,opt)
```

```
x =
    1.0000    2.0000    3.0000
```
Однако для решения систем линейных уравнений лучше использовать другие возможности MatLab, о которых мы поговорим в следующем параграфе.

Для нахождения корней системы нелинейных уравнений можно использовать команду **solve** из пакета символьной математики

syms a b c x
S = a*x^2 + b*x + c;
solve(S)
```
ans =
 1/2/a*(-b+(b^2-4*a*c)^(1/2))
 1/2/a*(-b-(b^2-4*a*c)^(1/2))
```
t = solve('x^2-3*x+2 = 0') % находит оба корня
```
t =
 2
 1
```
solve(x^2-5*a*x+4*a^2,x) % находит оба корня
```
ans =
   a
 4*a
```
syms a x
s = solve(x^3+a*x+1)
```
s =. . .
```
Здесь мы не выписали длинную формулу решения кубического уравнения.

Функция **solve** находит все корни (если это возможно) и возвращает решение в символьном виде. В частности это значит, что можно получить любую, сколь угодно большую точность решения. Однако следует помнить, что не для всех уравнений можно построить решение в символьном виде.

Функция **solve** возвращает решение в виде структуры

R=solve(2*x+3*y-13,3*x-2*y,x,y)
```
R =
    x: [1x1 sym]
    y: [1x1 sym]
```
Чтобы увидеть значения надо выполнить команды

R.x
```
ans =
2
```
R.y
```
ans =
3
```
Вот пример решения нелинейной системы

R=solve(x^2+y^2-16,x^2-y-4,x,y);
R.x'
```
ans =
[  0,  0, 7^(1/2), -7^(1/2)]
```
R.y'
```
ans =
[ -4, -4,  3,  3]
```
Такой способ возврата результата удобен, если у системы несколько решений

S = solve('u^2-v^2 = a^2','u + v = 1','a^2-2*a = 3')
```
S =
    a: [2x1 sym]
    u: [2x1 sym]
```

```
    v: [2x1 sym]
```
Чтобы их увидеть можно выполнить следующие команды

S.a' % напечатать оба значения *a*

```
ans =
[  3, -1]
```

s2 = [S.a(2), S.u(2), S.v(2)] % напечатать второе решение

```
s2 =
[ -1,  1,  0]
```

Вот, что получается, если решения нет

R=solve(x^2-3,x^2-4,x,y)

```
R =
[ empty sym ]
```

В следующем примере мы подсказываем MatLab, что переменная **a** вещественное число

syms a real; syms x y
[x,y] = solve(x^2*y^2, x-y/2-a);
[x';y']

```
ans =
[    0,    0,    a,    a]
[ -2*a, -2*a,    0,    0]
```

Уравнения для функции **solve** можно передавать в виде строки

eq = 'x^2*y^2=1, x-y/2-a';
[x,y] = solve(eq)

С помощью функции **solve** можно решать линейные системы

clear u v x y; syms u v x y
S = solve(x+2*y-u, 4*x+5*y-v);
sol = [S.x;S.y]

```
sol =
 -5/3*u+2/3*v
  4/3*u-1/3*v
```

Функция **solve** сама определяет имена переменных, относительно которых нужно решать уравнение или систему. Но если символьных переменных много, то вторым аргументом можно явно указать имена, относительно которых следует решать задачу.

solve('a*x^2 + b*x + c','b')

```
ans =
-(a*x^2+c)/x
```

solve('a*x^2 + b*x + c','x')

```
ans =
 1/2/a*(-b+(b^2-4*a*c)^(1/2))
 1/2/a*(-b-(b^2-4*a*c)^(1/2))
```

1.5.3 Решение систем линейных уравнений

Если надо решить большую систему линейных уравнений, лучше всего использовать встроенные возможности MatLab для решения линейных систем. Функция **X=A \ B** находит решение системы уравнений вида **A·X = B**, где **A** – прямоугольная матрица размера `m x n` и **B** – матрица размера `n x k`. В частности, если **b** столбец длины `n`, то выражение **A \ b** возвращает решение системы уравнений вида **A·X = b**. Решим для примера систему вида

$$\begin{bmatrix} 3 & 2 & 1 \\ 0 & 4 & -1 \\ -2 & -5 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \\ -9 \end{bmatrix}$$

```
A=[3 2 1; 0 4 -1; -2 -5 1];
b=[10; 5; -9]
x=A \ b
x =
    1.0000
    2.0000
    3.0000
```

Операцию **** называют операцией левого деления (умножения слева на обратную матрицу **inv(A)**). Она эквивалентна вызову функции **A \ B = mldivide(A,B)**. Рассмотрим еще пример

```
A=[1, -2, 3, -1;2, 3, -4, 4; 3, 1, -2, -2;1, -3, 7, 6]
A =
     1    -2     3    -1
     2     3    -4     4
     3     1    -2    -2
     1    -3     7     6
b=[6;-7;9;-7];
y=A \ b
y =
    2.0000
   -1.0000
         0
   -2.0000
```

Проверим, что вектор **y** удовлетворяет системе

```
A*y
ans =
    6.0000
   -7.0000
    9.0000
   -7.0000
```

Решить систему можно командой **inv(A)*B**, где функция **inv** вычисляет обратную матрицу.

```
inv(A)
ans =
    0.2857    0.1429    0.1429    0.0000
   -5.0286   -2.3143    2.6857    1.6000
   -2.8571   -1.4286    1.5714    1.0000
    0.7714    0.4857   -0.5143   -0.2000
```

Проверим

```
A*inv(A)                % получаем единичную матрицу
```

Решение системы, используя обратную матрицу, можно получить следующим образом

```
z=inv(A)*b; z'
ans =
    2.0000   -1.0000    0.0000   -2.0000
```

Если результат обращения матрицы не является надежным (определитель матрицы коэффициентов равен или близок к нулю), то выдается сообщение. Например,

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

y=A \ b

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.541976e-018.
y =
   -0.3333
    0.6667
         0
```

Решение с помощью оператора \ эффективнее, чем использование обратной матрицы. Это объясняется тем, что алгоритм решения систем линейных уравнений при помощи оператора \ определяется структурой матрицы коэффициентов системы. В частности, MATLAB исследует, является ли матрица треугольной, или может быть приведена перестановками строк и столбцов к треугольному виду, симметричная матрица или нет, квадратная или прямоугольная (MATLAB умеет решать системы с прямоугольными матрицами — переопределенные или недоопределенные). Поэтому решать системы при помощи \ разумно, когда выбор алгоритма решения поручается MATLAB. Если же имеется информация о свойствах матрицы системы, то следует использовать специальные методы.

В MatLab есть еще одна операция **/** - это операция правого деления матриц. Выражение **B / A** эквивалентно **B*inv(A).** Значит эта операция позволяет решать системы линейных уравнений вида **Y·A = B**. Вместо операции **B / A** можно вызывать эквивалентную ей функцию **mrdivide(B,A)**. Поясним различие между операциями **** и **/** следующей таблицей

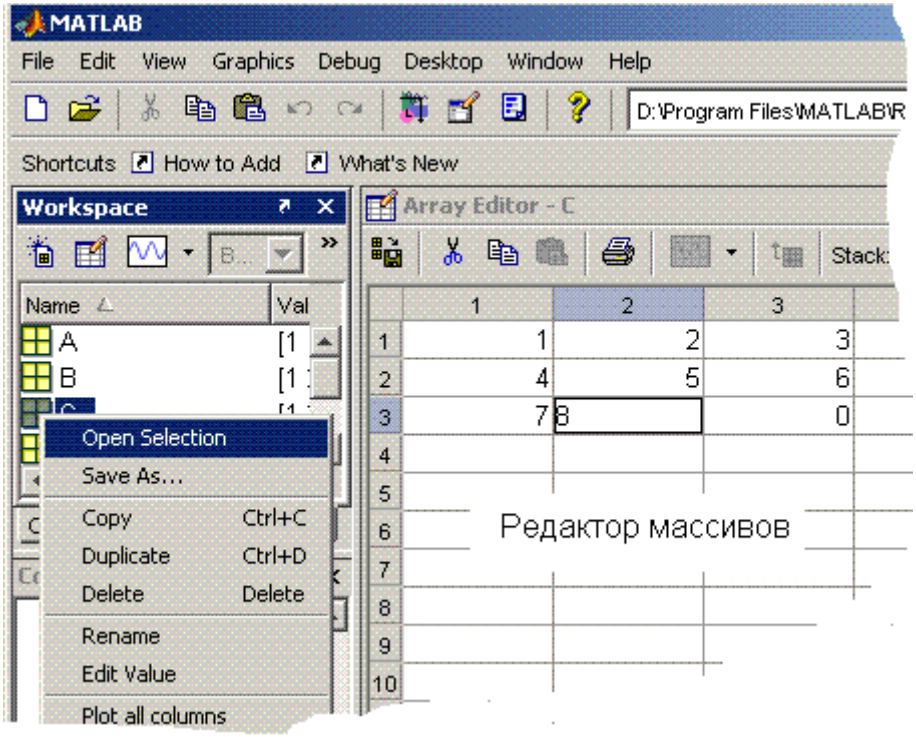
A=[1 1; 0 1] A = 1 1 0 1	**inv(A)** ans = 1 -1 0 1	**inv(A)*B** ans = -2 -2 3 4	**A \ B** ans = -2 -2 3 4
B=[1 2; 3 4] B = 1 2 3 4		**B*inv(A)** ans = 1 1 3 1	**B / A** ans = 1 1 3 1

Правило для запоминания: верх значка операции матричного деления **** или **/** всегда наклонен в сторону обращаемой матрицы. При этом размерности массивов при использовании операций **** и **/** должны быть согласованы.

Решение систем небольшой размерности можно выполнить, введя матрицу системы и вектор правой части непосредственно из командной строки. Однако часто требуется найти решение системы, состоящей из большого числа линейных уравнений. Для ввода данных можно воспользоваться редактором массивов. Выполните команду

C=[ ]

Пустой массив появится в окне рабочего пространства (Workspace). Щелкните в этом окне правой кнопкой мыши по имени массива **C** и выберите пункт Open Selection. Можно также выполнить двойной щелчок по имени переменной. Откроется редактор массивов, показанный на следующем рисунке



В этом окне вы можете вводить новый или редактировать уже существующий массив.

Можно также загрузить массив из файла командой **load** так, как мы это делали в п. 1.1.3.

Отметим, что имеются и поэлементные операции левого и правого деления. Операция **A .\ B** эквивалента поэлементному делению **B(i,j) / A(i,j)**, а операция **A ./ B** эквивалентна **A(i,j) / B(i,j)**.

A=[1 1; 2 4]	**B=[1 2; 3 4]**	**A .\ B**	**A ./ B**
A =	B =	ans =	ans =
1 1	1 2	1.0 2.0	1.0 0.5
2 4	3 4	1.5 1.0	0.6667 1.0

Операции \ и / могут решать недоопределенные системы. Например, решим систему, матрица коэффициентов которой имеет нулевой определитель

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

det(A)

```
ans =
     0
```

b=[6; 15; 24]

x1=A \ b

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.541976e-018.
x1 =
     0
     3
     0
```

Проверим, что полученный вектор `x1` является решением системы

A*x1-b % получаем нуль – вектор

Но легко проверить, что другой вектор – столбец, например, **[1 1 1]'** также является решением системы.

x2=[1 1 1]'

A*x2-b % получаем нуль – вектор

В данном случае решений бесконечно много и MatLab возвращает тот вектор решений, который содержит больше нулевых компонент.

Однако сообщение и возвращаемый вектор могут быть другими. Например

```
A=[1 2 3; 4 5 6; 6 6 6]
det(A)
ans =
     0
b=[6 15 18]';
A \ b
Warning: Matrix is singular to working precision.
ans =
   NaN
   NaN
   NaN
```

Функции **mldivide(A,B)** и **mrdivide(A,B)** (операции **** и **/**) довольно сложны. Для квадратных матриц они находят точное решение, для переопределенных систем находят решение с использованием метода наименьших квадратов, а для недоопределенных находят решение с минимальным числом ненулевых компонент. За более подробной информацией обращайтесь к справочной системе.

Можно решать линейные системы в символьном виде, используя те же операции левого и правого деления

```
syms u v
A = [1 2; 4 5];
b = [u; v];
z = A \ b
z =
 -5/3*u+2/3*v
  4/3*u-1/3*v
```

1.6 СПРАВОЧНАЯ СИСТЕМА И СРЕДА MATLAB

1.6.1 Справочная система

Простой способ получить справку по какой – либо функции это выполнить команду

help *имя функции*

Например

help magic

Информация о функциях по команде **help** выводится в командное окно. Все имена функций строчные, несмотря на то, что MatLab выводит их заглавными буквами. Можно получить справку по целой теме, если вы знаете ее имя

help matfun

Следующая команда выводит список тем

help

Команда **helpwin** или выбор пункта меню `Help – MatLab Help` открывает справочное окно MatLab с именами тем, а команда **helpwin** *topic* открывает окно справки по соответствующей теме. Например

helpwin matfun

Команда **lookfor** *keyword* позволяет искать функции по ключевому слову. Например

lookfor inverse

Аналогом команды **helpwin** является команда **doc**. Однако справка по одной и той же теме или функции отображается по – разному. Например, команды

doc ezplot

helpwin ezplot

откроют разные страницы справки об одной и той же функции.

1.6.2 Среда MatLab

Среда MatLab включает в себя совокупность переменных, созданных во время сеанса, и набор файлов, содержащих программы и данные, существующие между сеансами.

Рабочее пространство – это область памяти, доступная из командной строки. Команды

who % выводит список переменных, доступных в текущем сеансе

whos % выводит более подробную информацию о переменных, доступных в текущем сеансе

показывают текущее содержание рабочего пространства. Окно `Workspace` содержит ту же информацию, которая выводится по команде **whos**. Но при этом двойной щелчок по имени переменной открывает окно, в котором можно редактировать содержимое переменной (матрицы) прямо в памяти. Команда

save

сохраняет содержание рабочего пространства в MAT – файле, который может быть прочитан командой **load** в последующих сеансах работы. Например,

save 12_03_2010

сохранит содержание рабочего пространства в файл `12_03_2010.mat`. Можно сохранить только определенные переменные, указывая их имена после имени файла.

MatLab использует маршрут поиска файлов. Когда мы вызываем функцию, то выполняется первый M – файл на этом пути. Команда

path

показывает маршруты поиска файлов. Меню `Set Path...` из меню `File` показывает и позволяет добавлять новые каталоги в маршрут поиска. Команда

clear ValName или **clear('ValName')**

очищает содержимое переменной **ValName**. Команда **clear** без параметров очищает все переменные текущего сеанса. Команда **pack** сохраняет все переменные на диске, очищает память и снова загружает переменные.

Комбинация клавиш `Ctrl - Break` прерывает процесс вычислений.

Ниже мы приводим список основных специальных символов.

; (точка с	– разделяет строки матрицы;

запятой)	– завершает команды, если вывод результатов в командное окно не нужен;
[ ] (квадратные скобки)	– используются при задании матриц и векторов;
, (запятая)	– разделяет элементы строк матриц; – разделяет операторы в строке ввода;
: (двоеточие)	– используется для указание диапазона (интервала изменения величины);
( ) (круглые скобки)	– задают порядок выполнения матема – тических операций; – указывают аргументы функций; – указывают индексы массивов;
… (три точки и более)	– являются символом продолжения выражения на следующую строку;
. (точка)	– отделяет дробную часть от целой; – применяется в составе комбинирован – ных знаков поэлементных операций (./ .\ .^ .*)
% (процент)	– начало комментария
' (апостроф)	– начинает и завершает символьные строки (для включения самого апострофа в строку надо поставить два апострофа подряд).

Команда **type** `'FileName'` печатает в командном окне текст m – файла. Команда

clc

очищает командное окно, оставляя знак приглашения (переменные в памяти остаются). Команда **home** делает то же самое.