1.1 Среда разработки MATLAB.

Что такое MATLAB?

MATLAB – это высокопроизводительный язык для технических расчетов. Он включает в себя вычисления, визуализацию и программирование в удобной среде, где задачи и решения выражаются в форме, близкой к математической. Типичное использование MATLAB – это:

- математические вычисления
- создание алгоритмов
- моделирование
- анализ данных, исследования и визуализация
- научная и инженерная графика
- разработка приложений, включая создание графического интерфейса

MATLAB – это интерактивная система, в которой основным элементом данных является массив. Это позволяет решать различные задачи, связанные с техническими вычислениями, особенно в которых используются матрицы и вектора, в несколько раз быстрее, чем при написании программ с использованием "скалярных" языков программирования, таких как Си или Фортран.

Слово MATLAB означает матричная лаборатория (matrix laboratory). MATLAB был специально написан для обеспечения легкого доступа к LINPACK и EISPACK, которые представляют собой современные программные средства для матричных вычислений.

MATLAB развивался в течении нескольких лет, ориентируясь на различных пользователей. В университетской среде, он представлял собой стандартный инструмент для работы в различных областях математики, машиностроении и науки. В промышленности, MATLAB – это инструмент для высокопродуктивных исследований, разработок и анализа данных.

В MATLAB важная роль отводится специализированным группам программ, называемых toolboxes. Они очень важны для большинства пользователей MATLAB, так как позволяют изучать и применять специализированные методы. Toolboxes – это всесторонняя коллекция функций MATLAB (M-файлов), которые позволяют решать частные классы задач. Toolboxes применяются для обработки сигналов, систем контроля, нейронных сетей, нечеткой логики, вэйвлетов, моделирования и т.д.

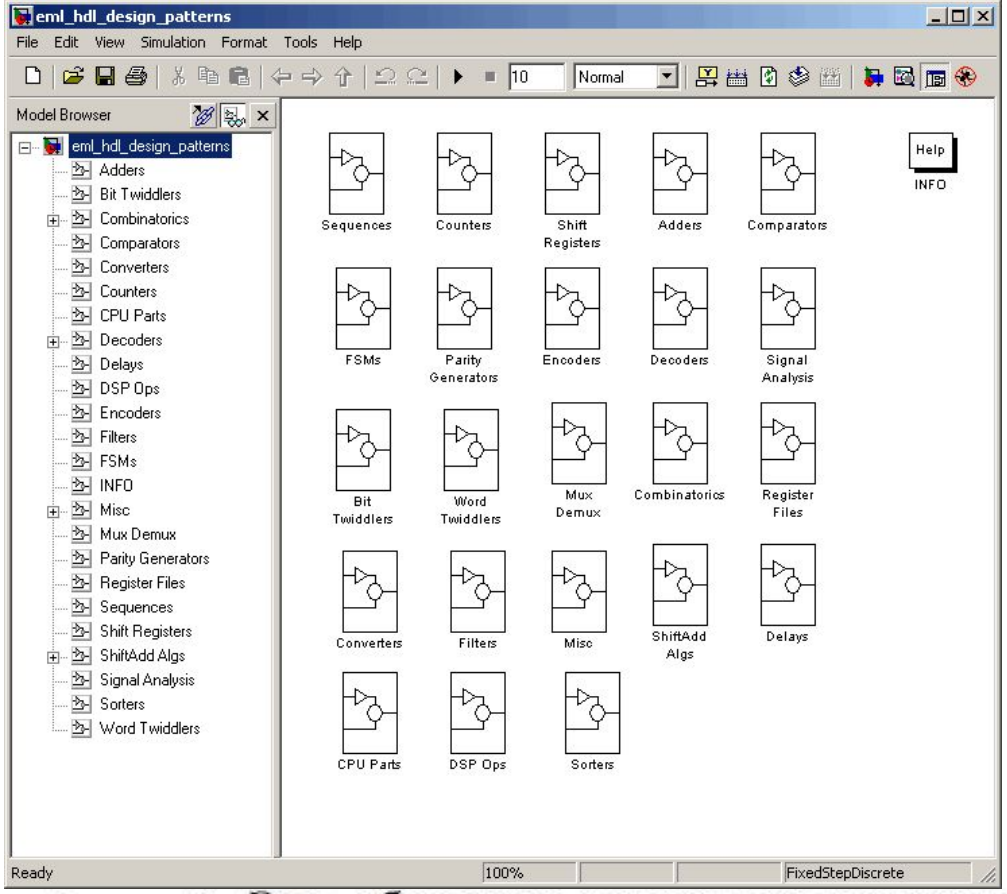
Система MATLAB

Система MATLAB состоит из пяти основных частей:

Язык MATLAB. Это язык матриц и массивов высокого уровня с управлением потоками, функциями, структурами данных, вводом-выводом и особенностями объектно-ориентированного программирования. Это позволяет как программировать в "небольшом масштабе" для быстрого создания черновых программ, так и в "большом" для создания больших и сложных приложений.

Среда MATLAB. Это набор инструментов и приспособлений, с которыми работает пользователь или программист MATLAB. Она включает в себя средства для управления переменными в рабочем пространстве MATLAB, вводом и выводом данных, а также создания, контроля и отладки M-файлов и приложений MATLAB.

Управляемая графика. Это графическая система MATLAB, которая включает в себя команды высокого уровня для визуализации двух- и трехмерных данных, обработки изображений, анимации и иллюстрированной графики. Она также включает в себя команды низкого уровня, позволяющие полностью редактировать внешний вид графики, также как при создании Графического Пользовательского Интерфейса (GUI) для MATLAB приложений.



Библиотека математических функций. Это обширная коллекция вычислительных алгоритмов от элементарных функций, таких как сумма, синус, косинус, комплексная арифметика, до более сложных, таких как обращение матриц, нахождение собственных значений, функции Бесселя, быстрое преобразование Фурье.

Программный интерфейс. Это библиотека, которая позволяет писать программы на Си и Фортране, которые взаимодействуют с MATLAB. Она включает средства для вызова программ из MATLAB (динамическая связь), вызывая MATLAB как вычислительный инструмент и для чтения-записи MAT-файлов.

О Simulink

Simulink, сопутствующая MATLAB программа, - это интерактивная система для моделирования нелинейных динамических систем. Она представляет собой среду, управляемую мышью, которая позволяет моделировать процесс путем перетаскивания блоков диаграмм на экране и их манипуляцией. Simulink работает с линейными, нелинейными, непрерывными, дискретными, многомерными системами.

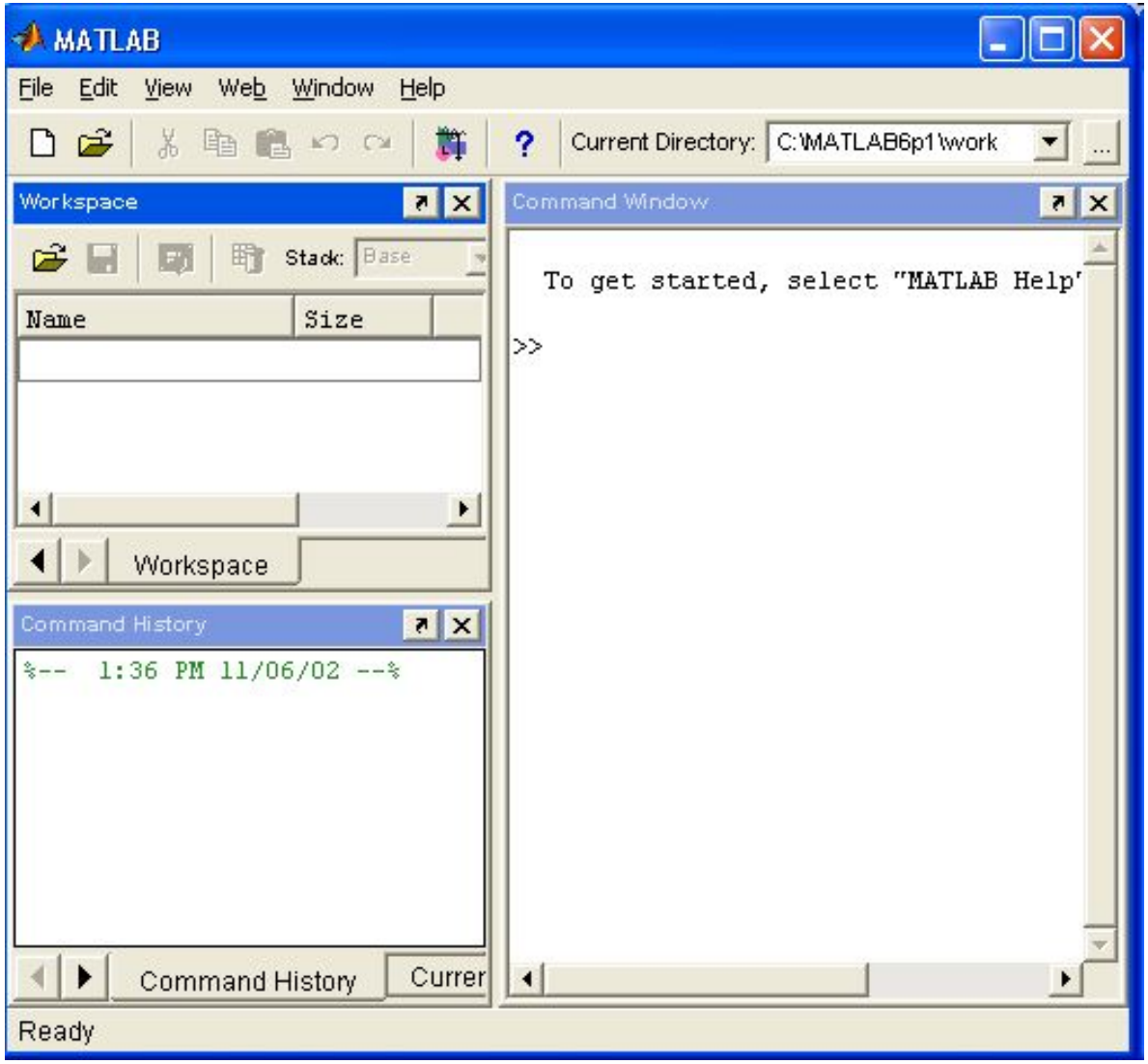
Blocksets – это дополнения к Simulink, которые обеспечивают библиотеки блоков для специализированных приложений, таких как связь, обработка сигналов, энергетические системы.

Real-Time Workshop – это программа, которая позволяет генерировать С код из блоков диаграмм и запускать их на выполнение на различных системах реального времени.

Матрицы и магические квадраты

Лучший способ начать работу с MATLAB — это научиться обращаться с матрицами. В этой главе мы покажем вам, как надо это делать. В MATLAB матрица – это прямоугольный массив чисел. Особое значение придается матрицам 1x1, которые являются скалярами, и матрицам, имеющим один столбец или одну строку, — векторам. MATLAB использует различные способы для хранения численных и не численных данных, однако вначале лучше всего рассматривать все данные как матрицы. MATLAB организован так, чтобы все операции в нем были как можно более естественными. В то время как другие программные языки работают с числами как элементами языка, MATLAB позволяет вам быстро и легко оперировать с целыми матрицами.

Рабочая среда MatLab

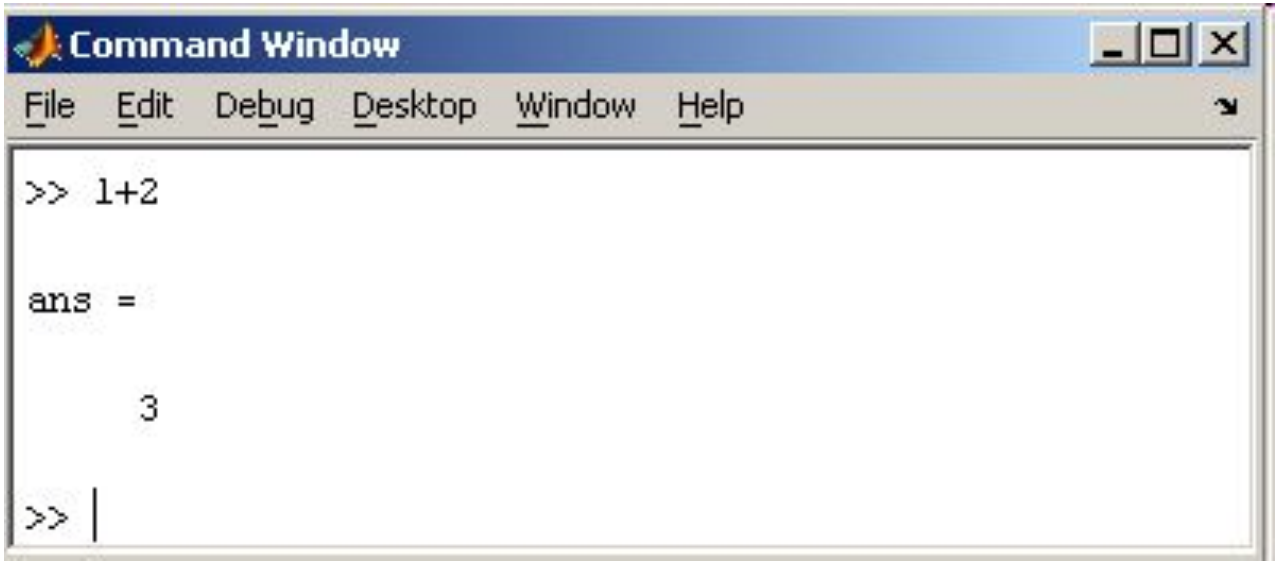


Чтобы запустить программу дважды щелкните на иконку. 

Перед Вами откроется рабочая среда, изображенная на рисунке.

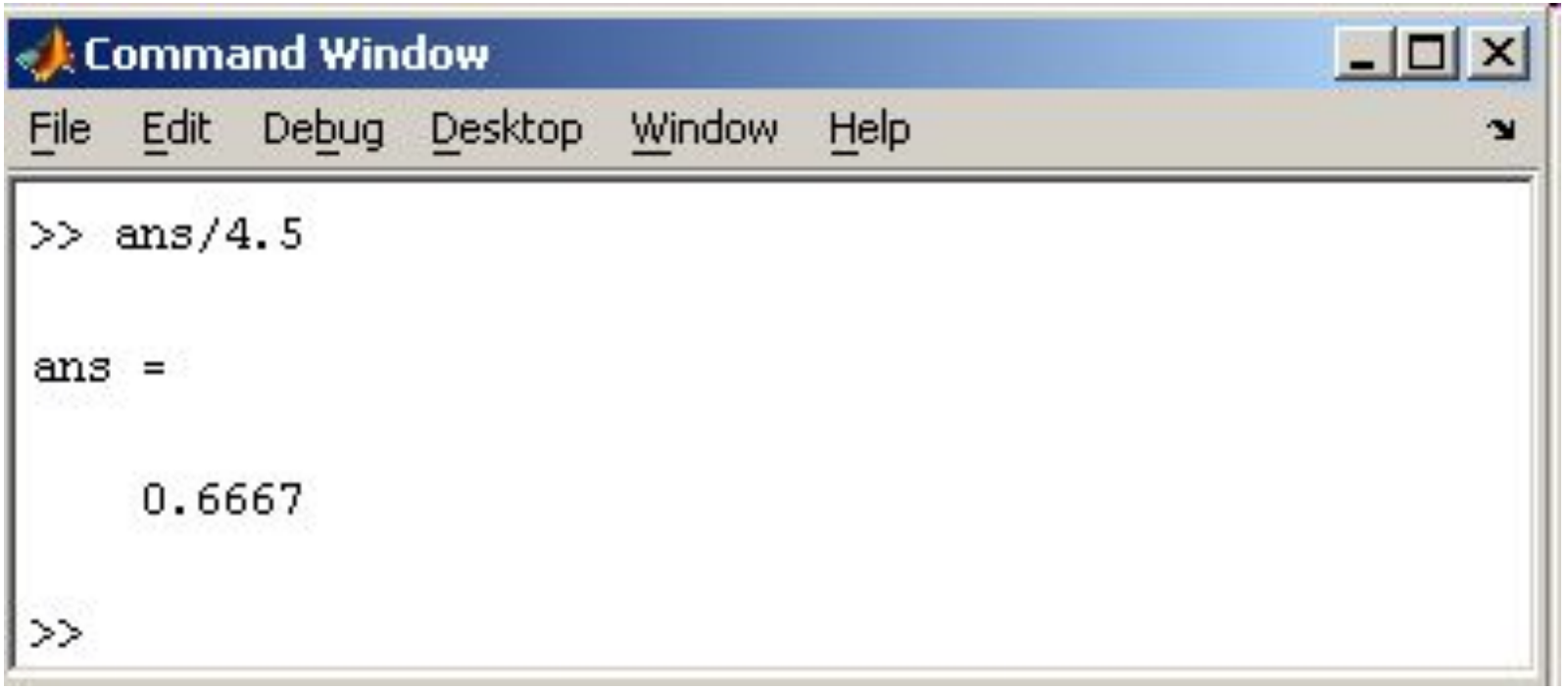
1.2. Простейшие вычисления

Наберите в командной строке 1+2 и нажмите **Enter**. В результате в командном окне MatLab отображается следующее:



Что сделала программа MatLab? Сначала она вычислила сумму 1+2, затем записала результат в специальную переменную ans и вывела ее значение, равное 3, в командное окно. Ниже ответа расположена командная строка с мигающим курсором, обозначающая, что MatLab готов к дальнейшим вычислениям. Можно набирать в командной строке новые выражения и находить их значения.

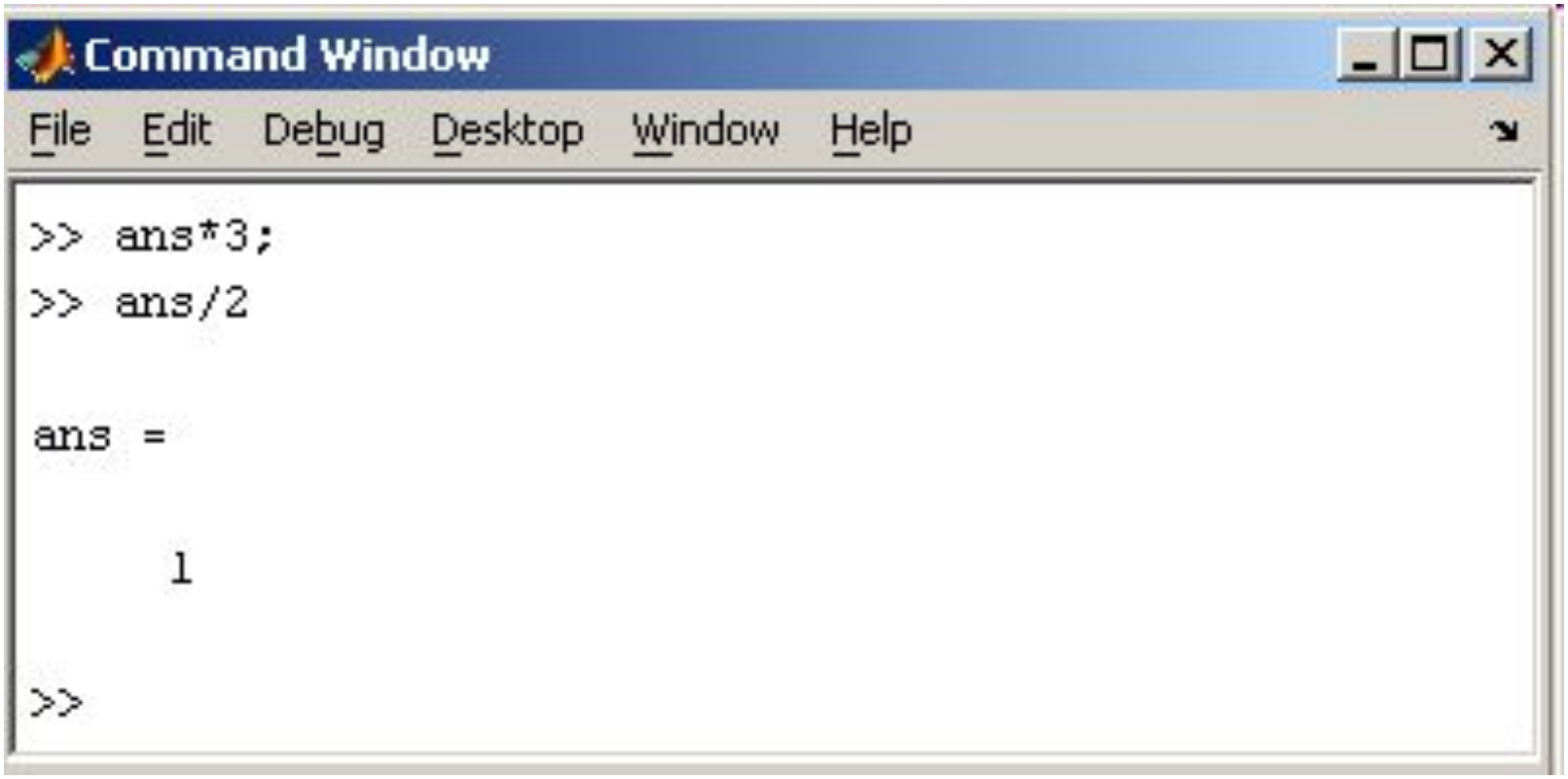
Если требуется продолжить работу с предыдущим выражением, например, вычислить (1+2)/4.5, то проще всего воспользоваться уже имеющимся результатом, который хранится в переменной ans. Наберите ans/4.5 (при вводе десятичных дробей используется точка) и нажмите **Enter**, получается



1.3. Эхо команд

Выполнение каждой команды в MatLab сопровождается эхом. В приведенном выше примере — это ответ ans = 0.6667. Часто эхо затрудняет восприятие работы программы и тогда его можно отключить. Для этого команда должна завершаться символом точка с запятой.

Например



1.4. Сохранение рабочей среды. MAT файлы

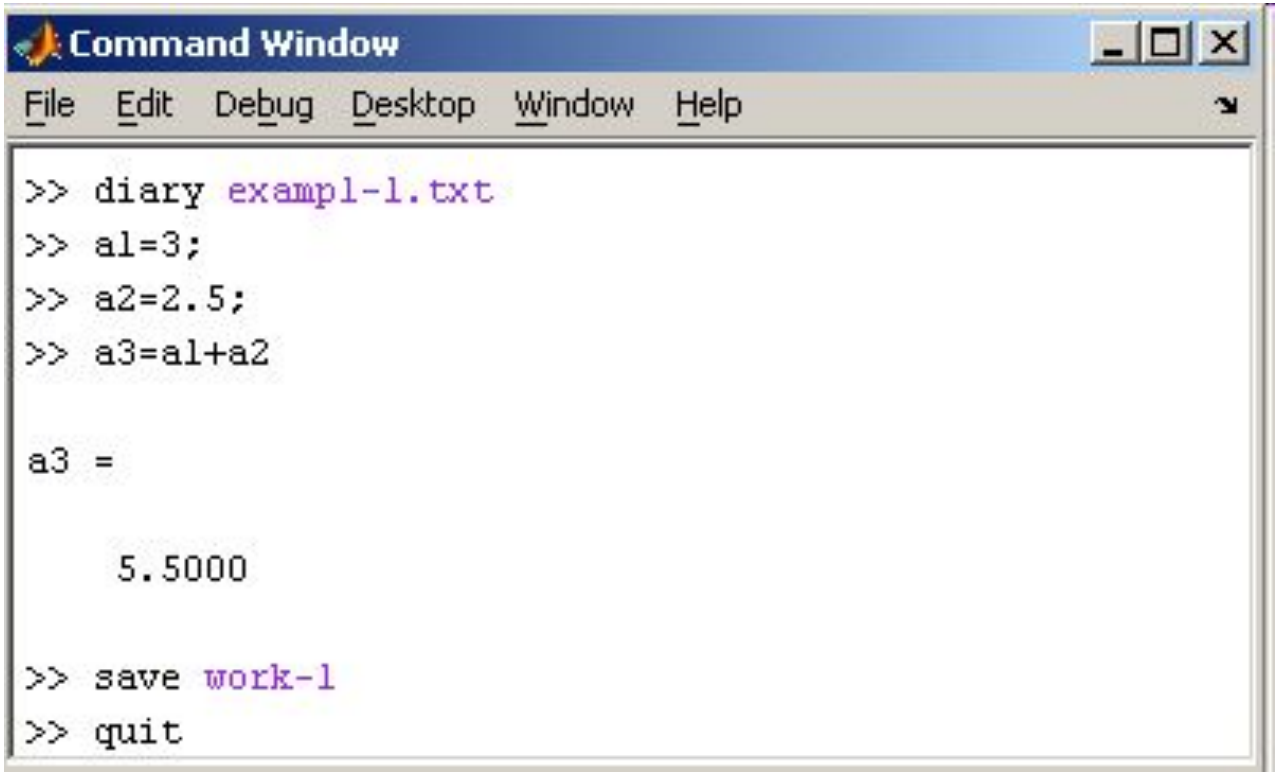
Самый простой способ сохранить все значения переменных — использовать в меню **File** пункт **Save Workspase As.** При этом появляется диалоговое окно **Save Workspase Variables**, в котором следует указать каталог и имя файла.

По умолчанию предлагается сохранить файл в подкаталоге work основного каталога MatLab. Программа сохранит результаты работы в файле с расширением mat. Теперь можно закрыть MatLab.

В следующем сеансе работы для восстановления значений переменных следует открыть этот сохраненный файл при помощи подпункта **Open** меню **File**. Теперь все переменные, определенные в прошлом сеансе, опять стали доступными. Их можно использовать во вновь вводимых командах.

1.5. Журнал

В MatLab имеется возможность записывать исполняемые команды и результаты в текстовый файл (вести журнал работы), который потом можно прочитать или распечатать из текстового редактора. Для начала ведения журнала служит команда diary. В качестве аргумента команды diary следует задать имя файла, в котором будет храниться журнал работы. Набираемые далее команды и результаты их исполнения будут записываться в этот файл, например последовательность команд



Производит следующие действия:

- открывает журнал в файле exampl-1.txt;
- производит вычисления;
- сохраняет все переменные в MAT файле work-1.mat;
- сохраняет журнал в файле exampl-1.txt в подкаталоге work корневого каталога MatLab и закрывает MatLab;

Посмотрите содержимое файла exampl-1.txt в каком-нибудь текстовом редакторе. В файле окажется следующий текст:

```
a1=3;
a2=2.5;
a3=a1+a2

a3 =

    5.5000

save work-1
quit
```

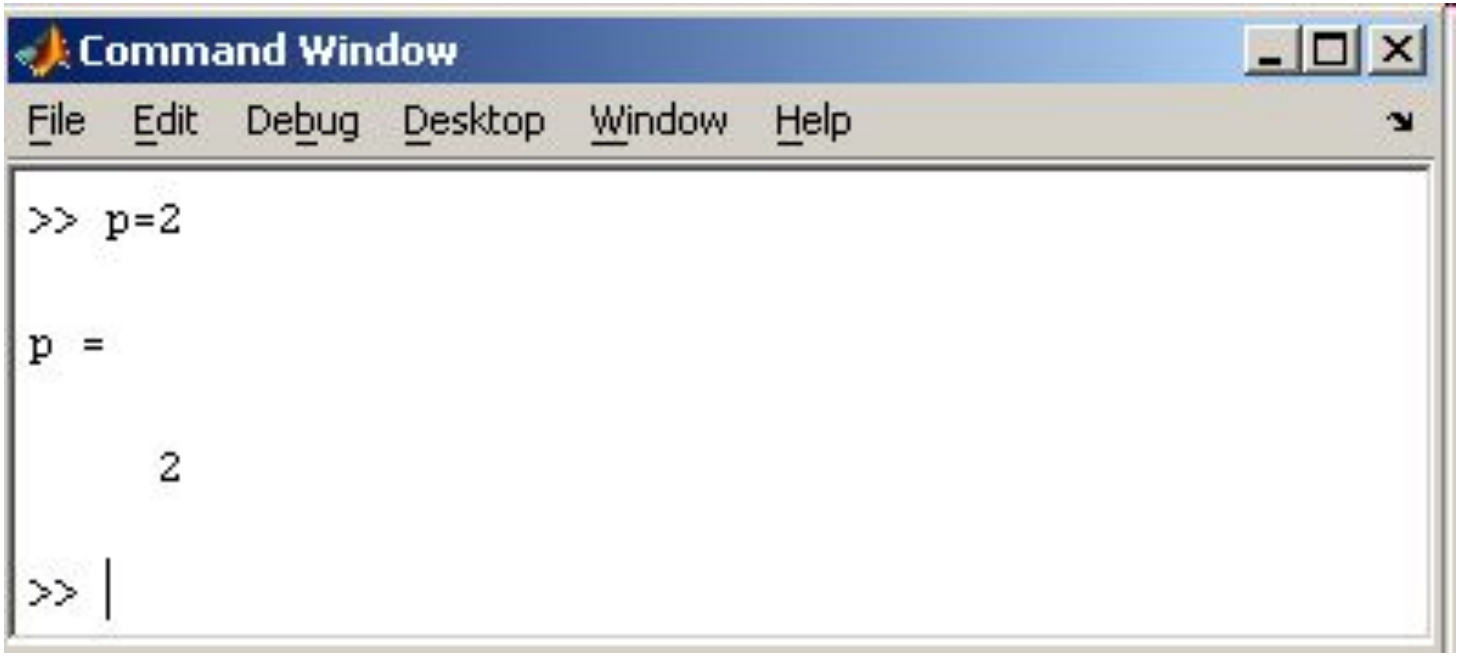
Окно справки MatLab появляется после выбора опции **Help Window** в меню **Help** или нажатием кнопки вопроса на панели инструментов.

Эта же операция может быть выполнена при наборе команды helpwin. Для вывода окна справки по отдельным разделам, наберите helpwin topic.

Окно справки предоставляет Вам такую же информацию, как и команда help, но оконный интерфейс обеспечивает более удобную связь с другими разделами справки.

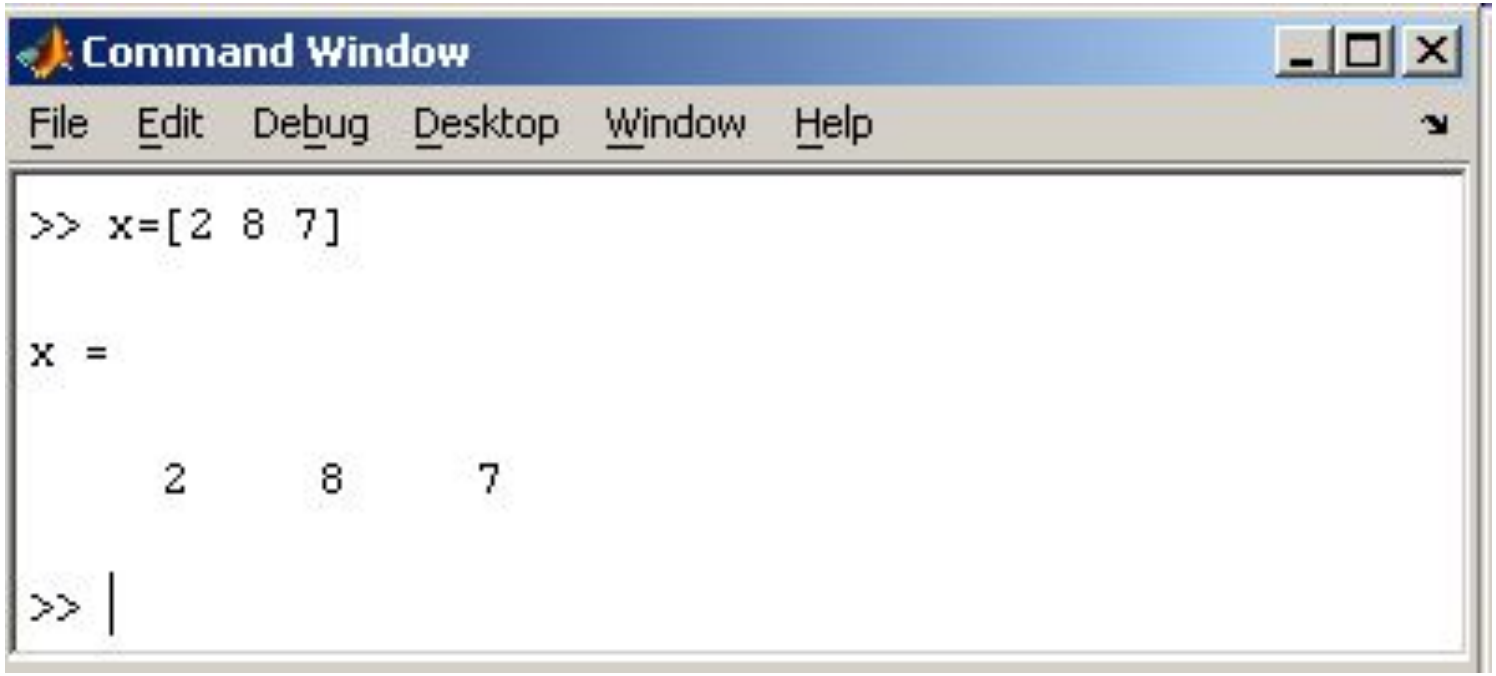
Используя адрес Web-страницы фирмы Math Works Используя адрес Web-страницы фирмы Math Works, вы можете выйти на сервер фирмы и получить самую последнюю информацию по интересующим вас вопросам. Вы можете ознакомиться с новыми программными продуктами Используя адрес Web-страницы фирмы Math Works, вы можете выйти на сервер фирмы и получить самую последнюю информацию по интересующим вас вопросам. Вы можете ознакомиться с новыми программными продуктами или найти ответ на возникшие проблемы на странице технической поддержки.

В MatLab можно использовать скаляры, векторы и матрицы. Для ввода скаляра достаточно приписать его значение какой-то переменной, например

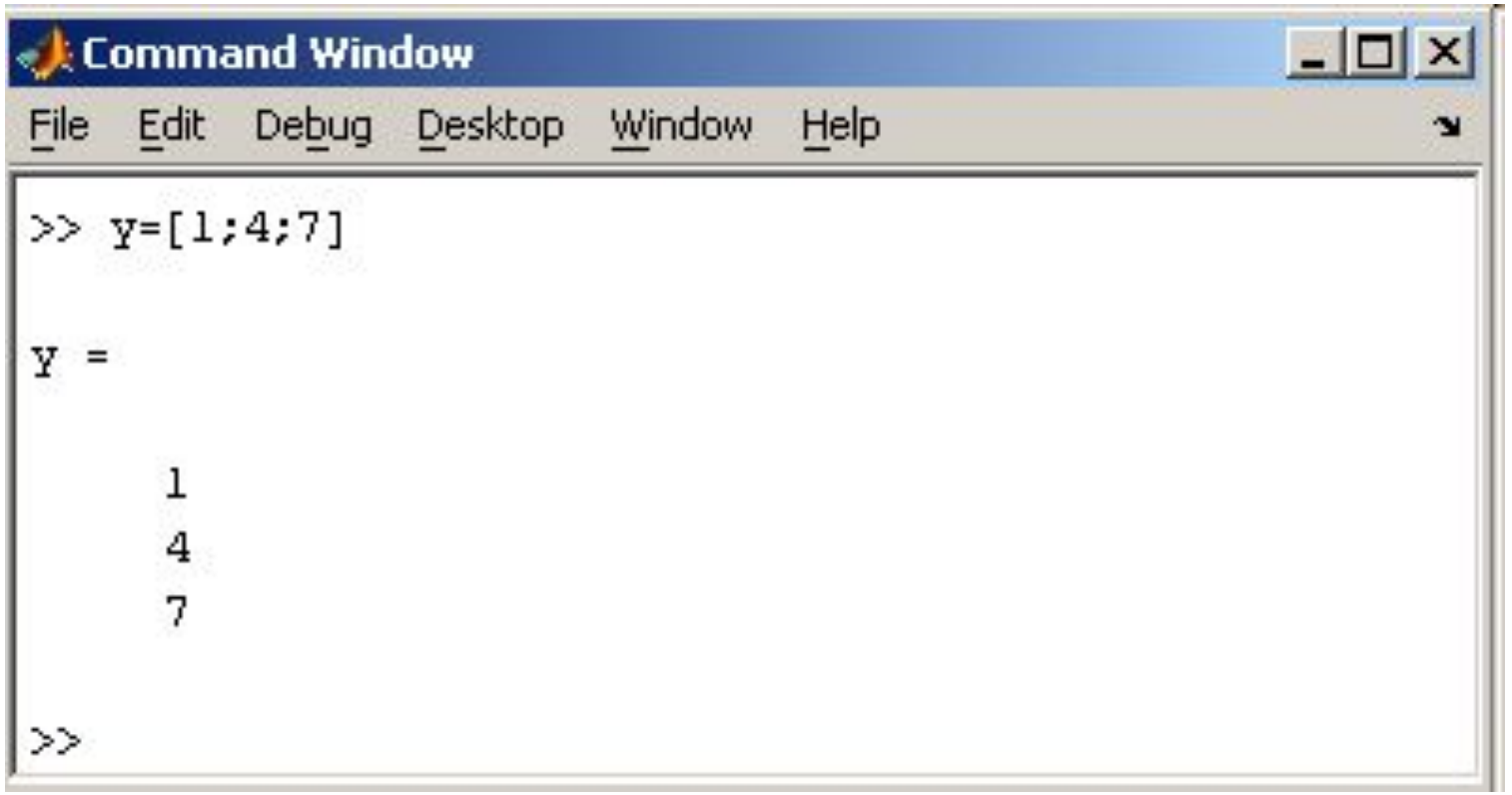


MatLab различает заглавные и прописные буквы, так что p и P — это разные переменные.

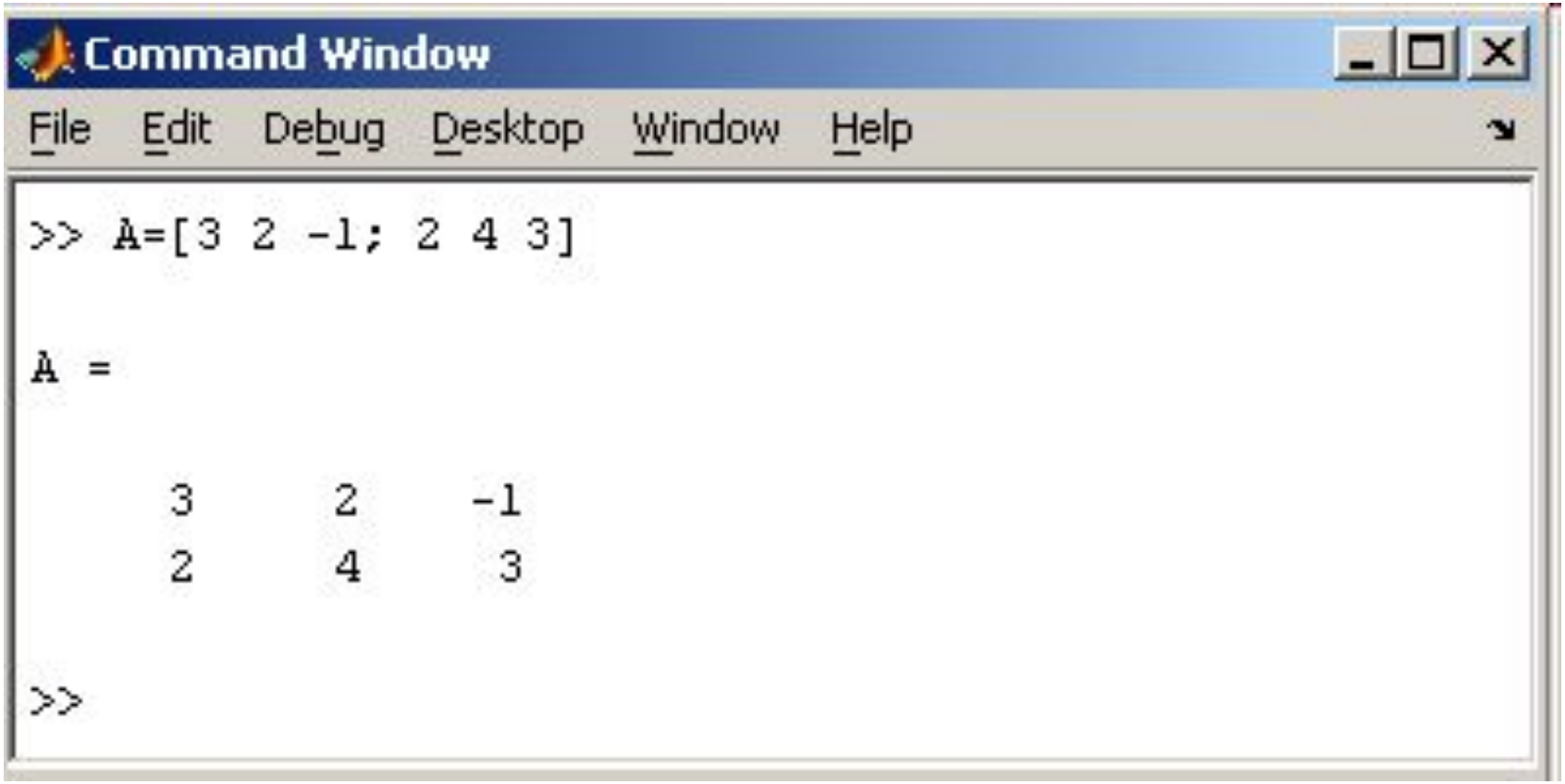
Для ввода массивов (векторов или матриц) их элементы заключают в квадратные скобки. Так для ввода вектора-строки размером 1×3, используется следующая команда, в которой элементы строки отделяются пробелами или запятыми.



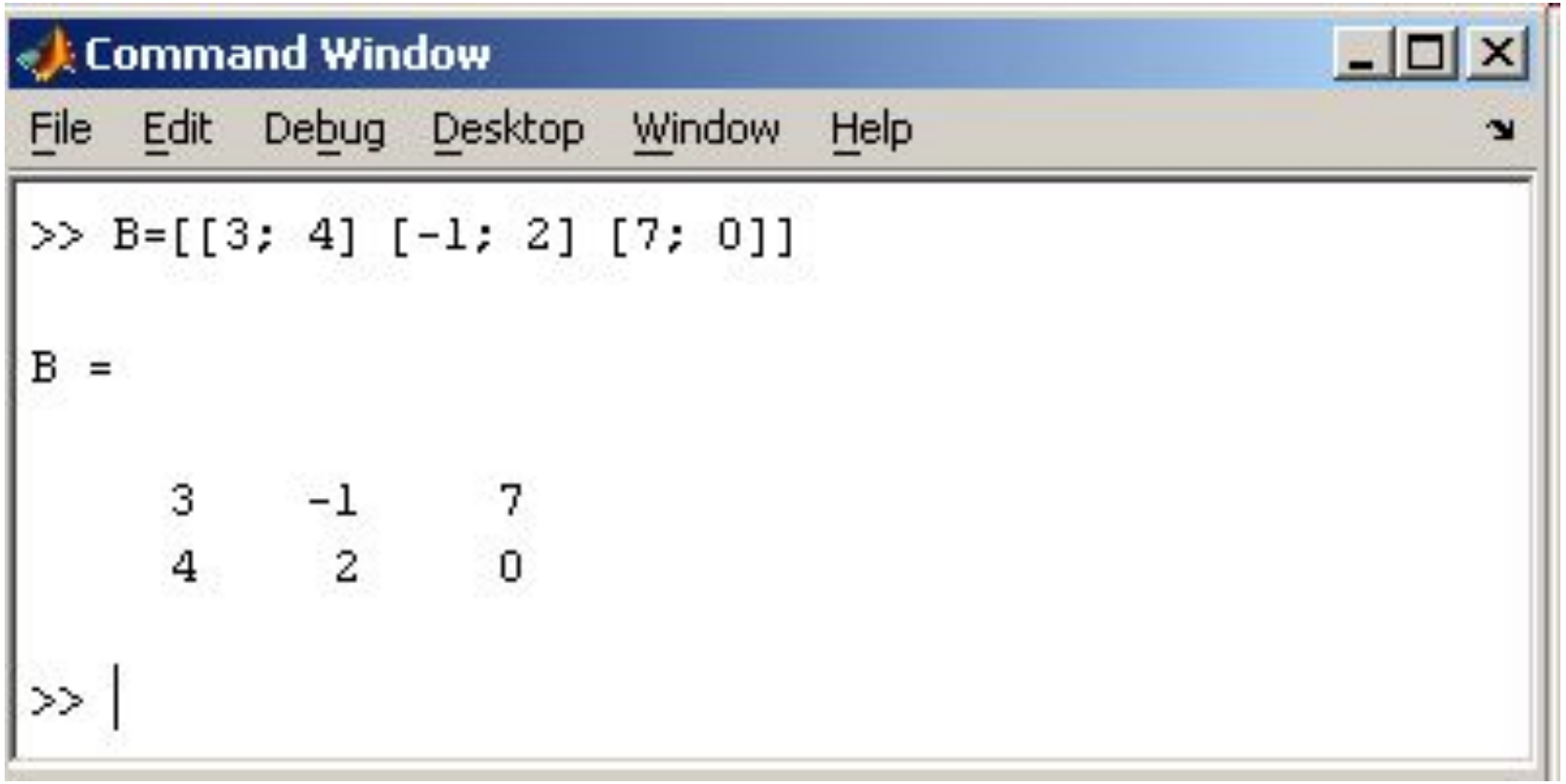
При вводе вектора-столбца элементы разделяют точкой с запятой. Например,



Вводить небольшие по размеру матрицы удобно прямо из командной строки. При вводе матрицу можно рассматривать как вектор-столбец, каждый элемент которого является вектор-строкой.



матрицу можно трактовать как вектор строку, каждый элемент которой является вектор-столбцом.

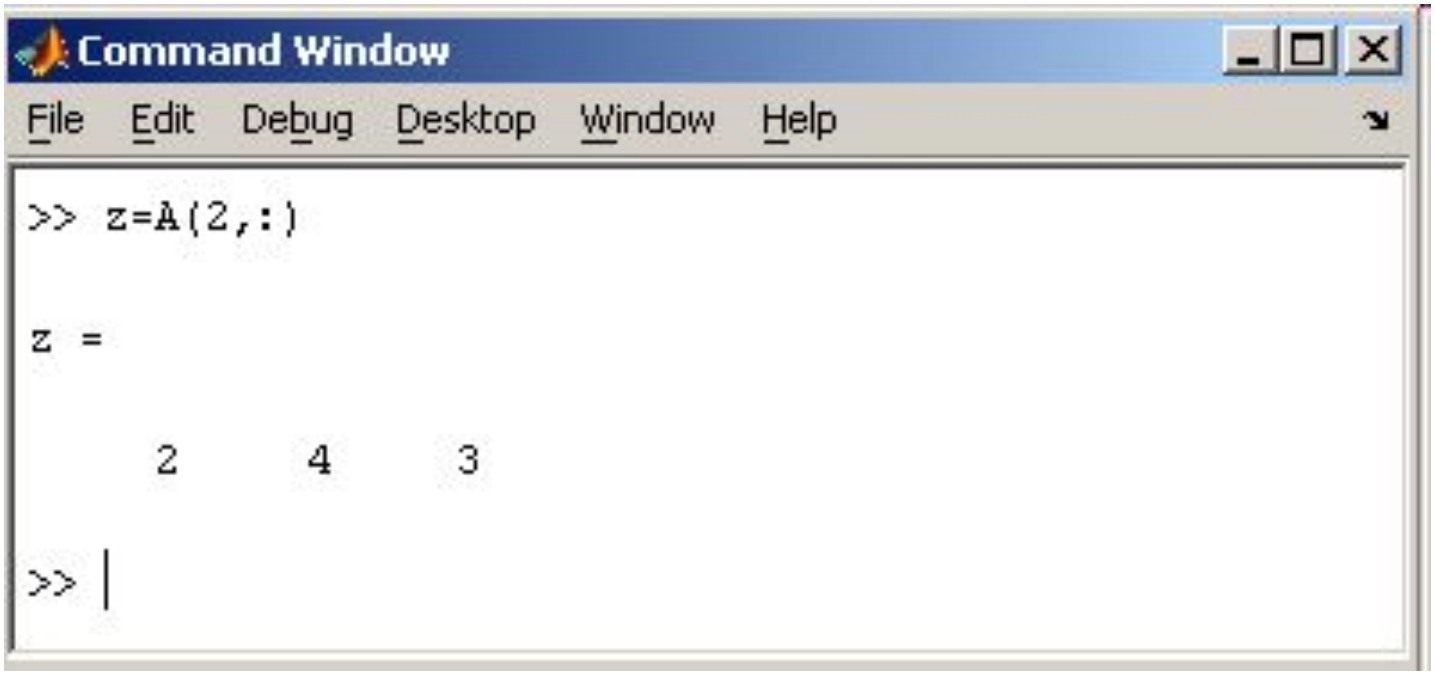


2.2. Доступ к элементам

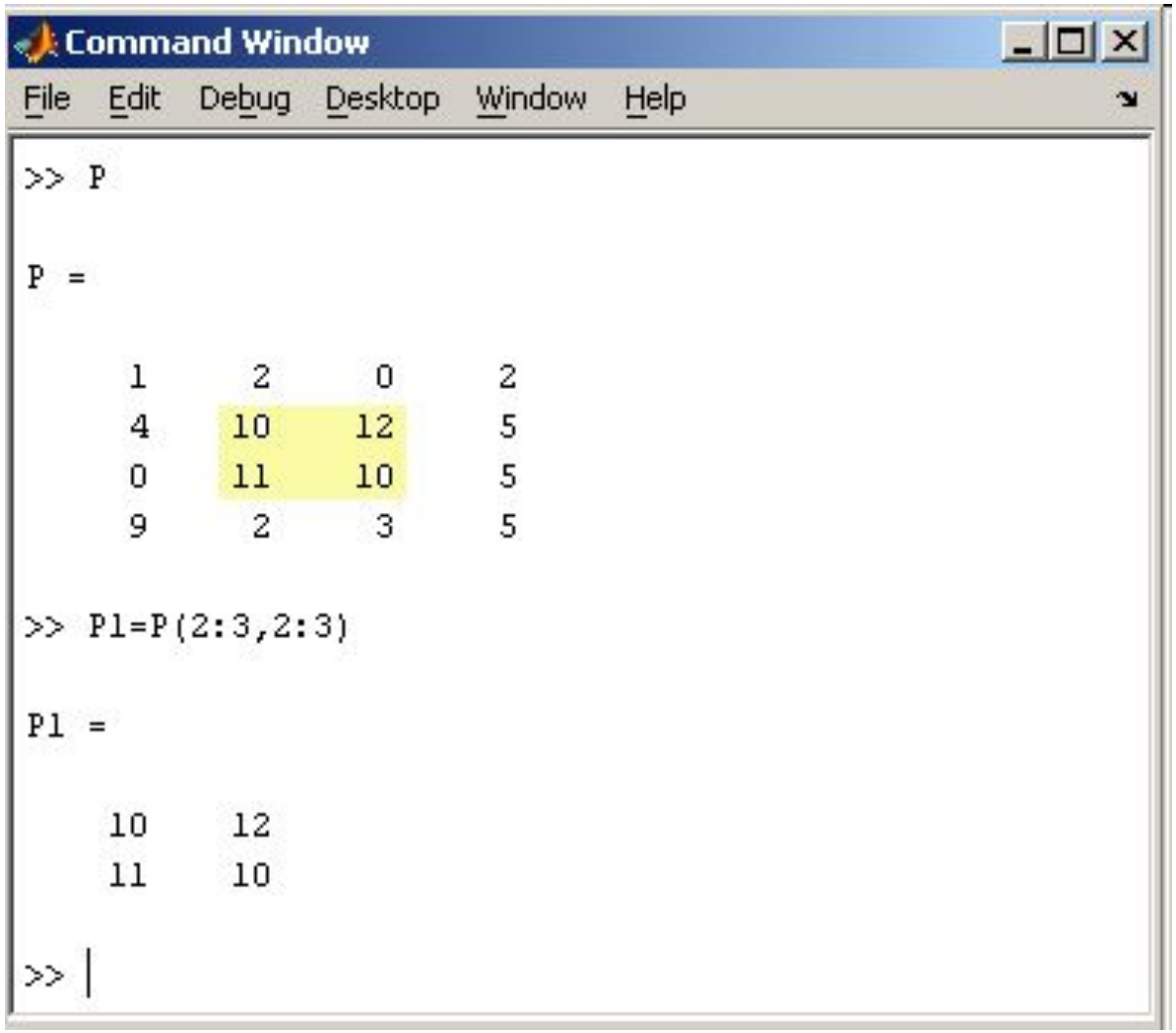
Доступ к элементам матриц осуществляется при помощи двух индексов — номеров строки и столбца, заключенных в круглые скобки,

например команда B(2,3) выдаст элемент второй строки и третьего столбца матрицы B. Для выделения из матрицы столбца или строки следует в качестве одного из индексов использовать номер столбца или строки матрицы, а другой индекс заменить двоеточием.

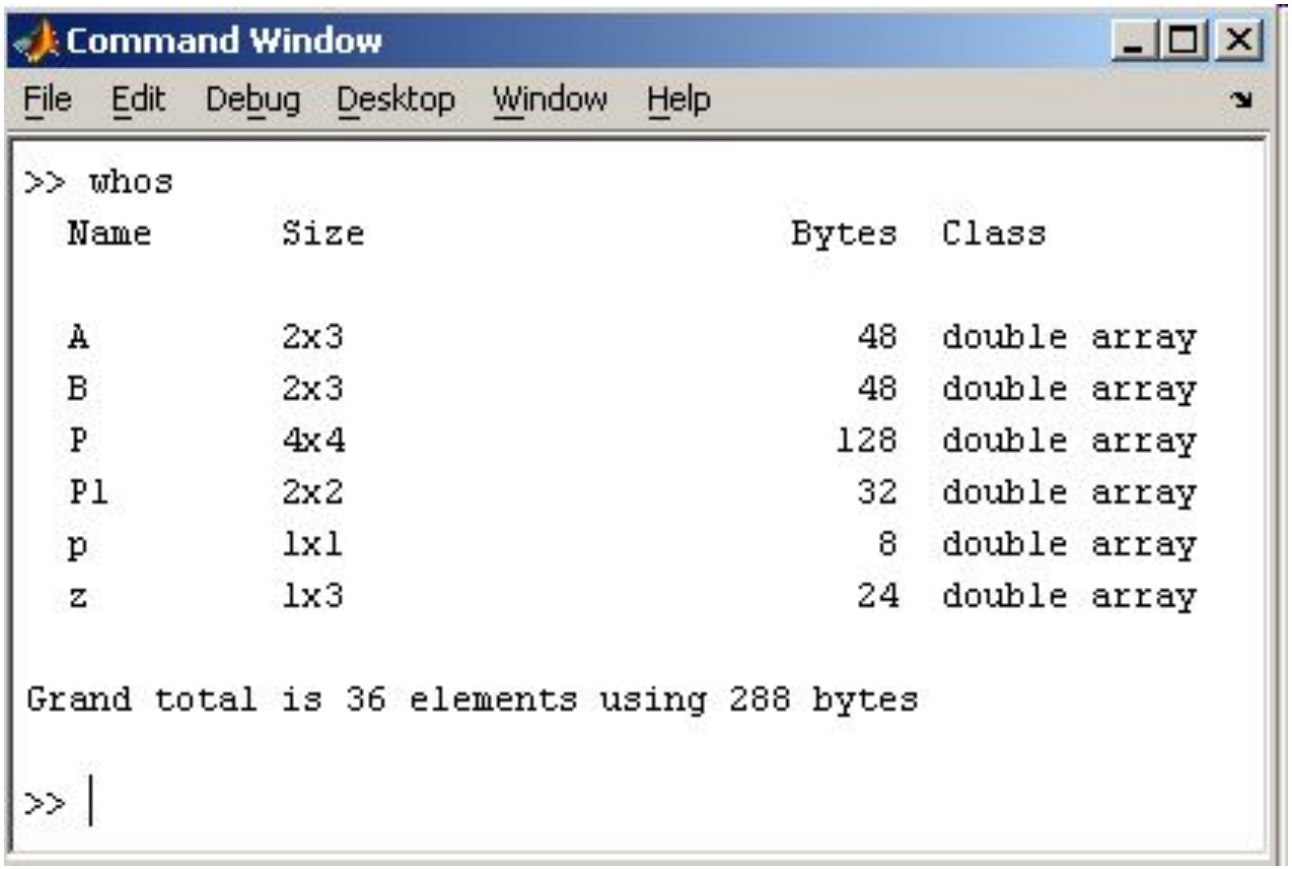
Например, запишем вторую строку матрицы A в вектор z



Также можно осуществлять выделение блоков матриц при помощи двоеточия. Например, выделим из матрицы P блок отмеченный цветом



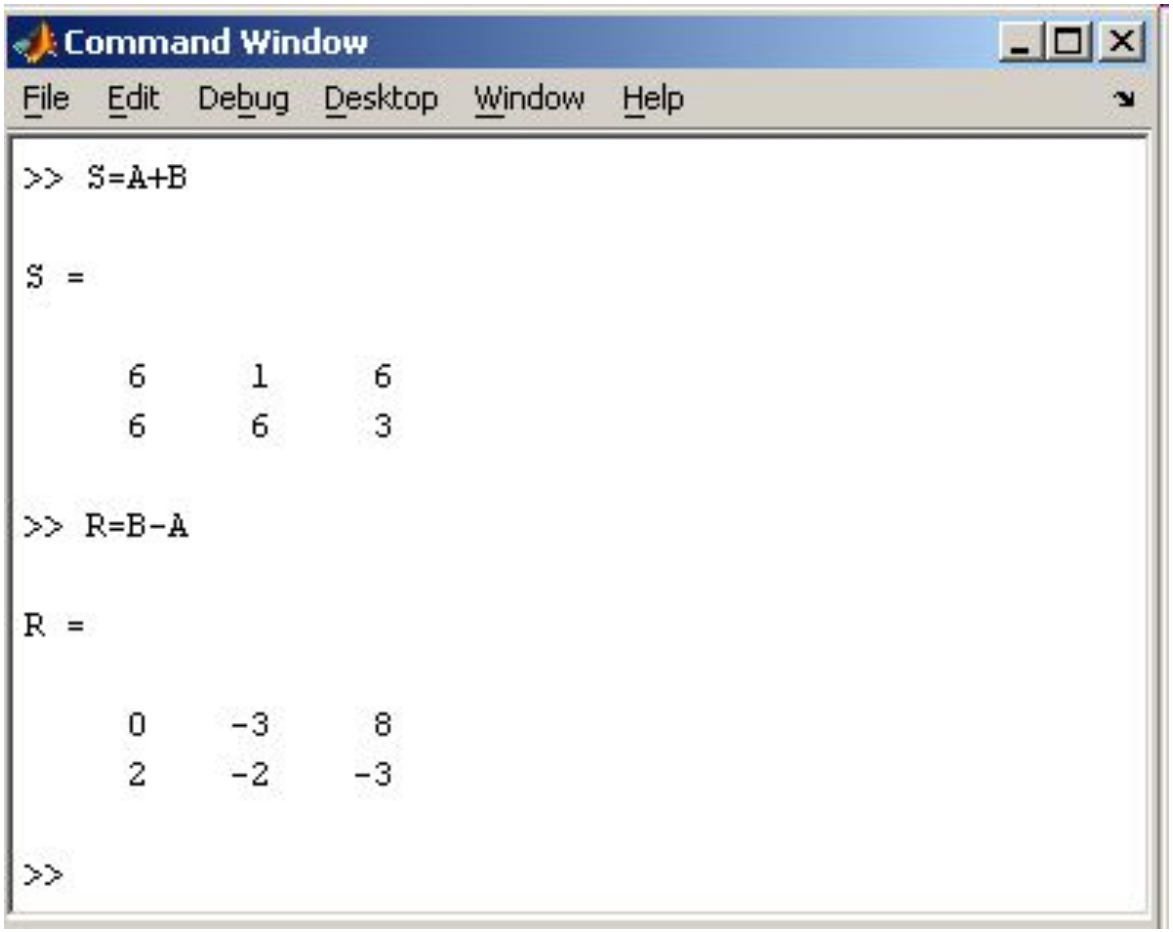
Если необходимо посмотреть переменные рабочей среды, в командной строке необходимо набрать команду whos.



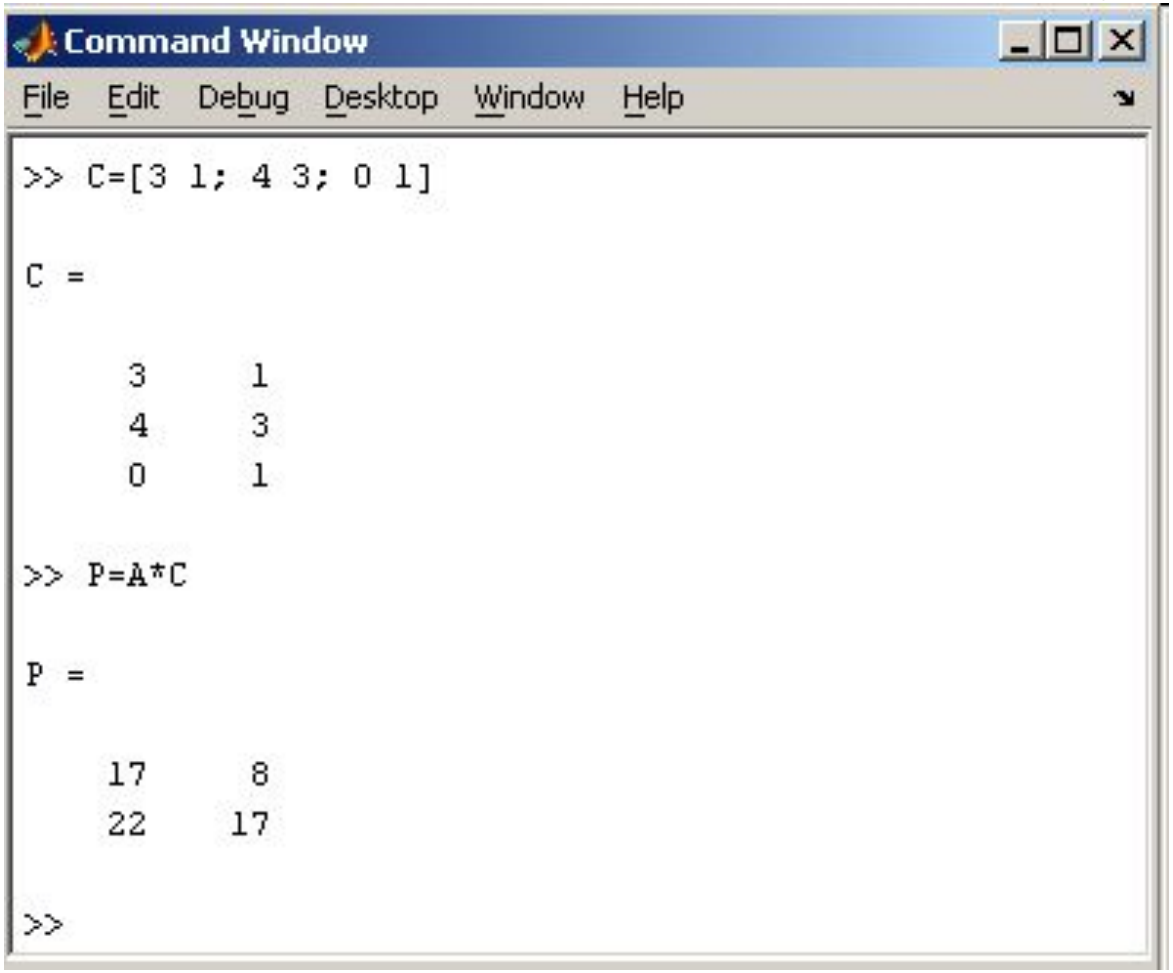
Видно, что в рабочей среде содержатся один скаляр (p), четыре матрицы (A, B, P, P1) и вектор-строка (z).

2.3. Основные матричные операции

При использовании матричных операций следует помнить, что для сложения или вычитания матрицы должны быть одного размера, а при перемножении число столбцов первой матрицы обязано равняться числу строк второй матрицы. Сложение и вычитание матриц, так же как чисел и векторов, осуществляется при помощи знаков плюс и минус

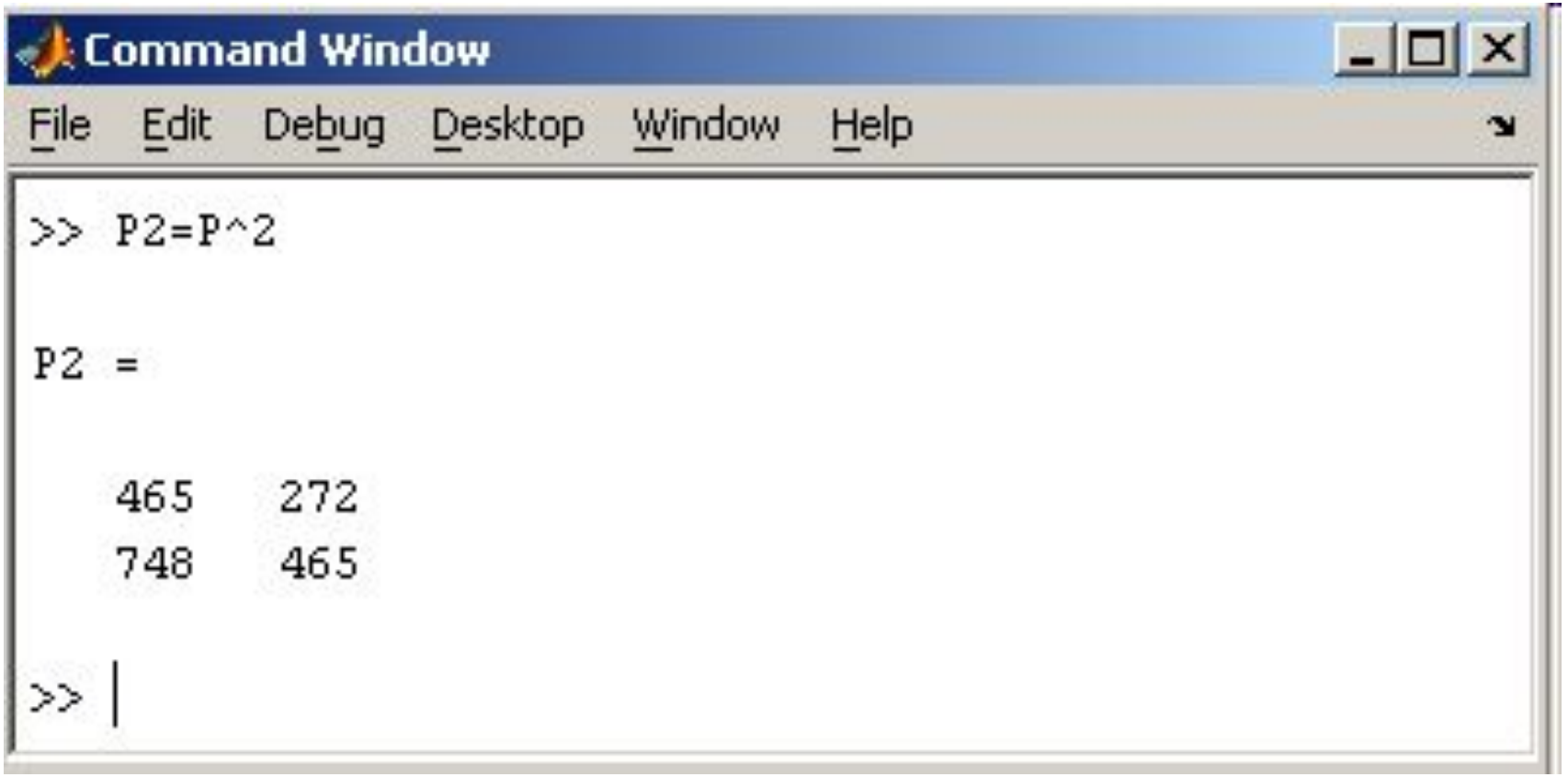


умножение — знаком звездочка *. Введем матрицу размером 3×2



Умножение матрицы на число тоже осуществляется при помощи звездочки, причем умножать на число можно как справа, так и слева.

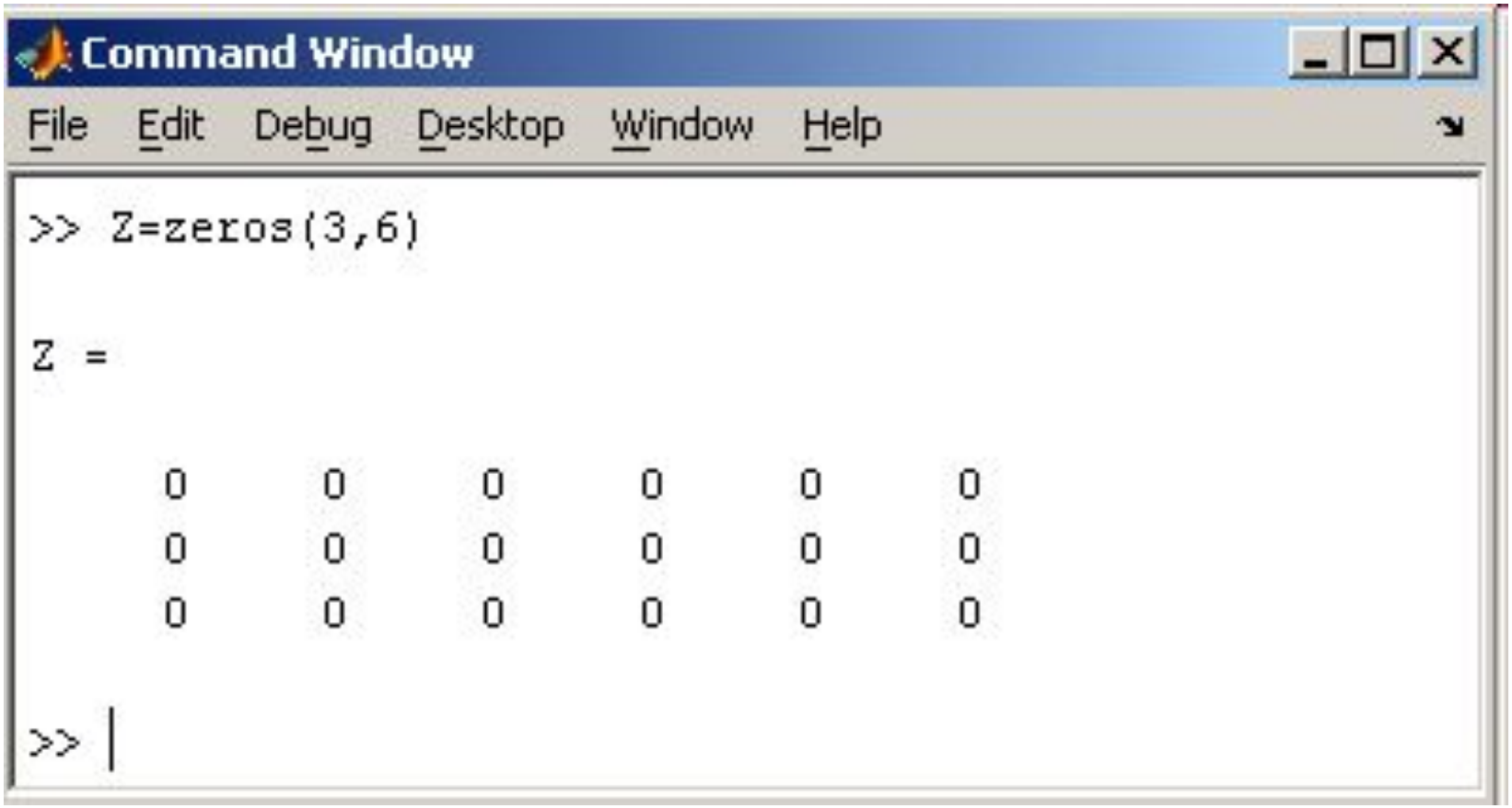
Возведение квадратной матрицы в целую степень производится с использованием оператора ^



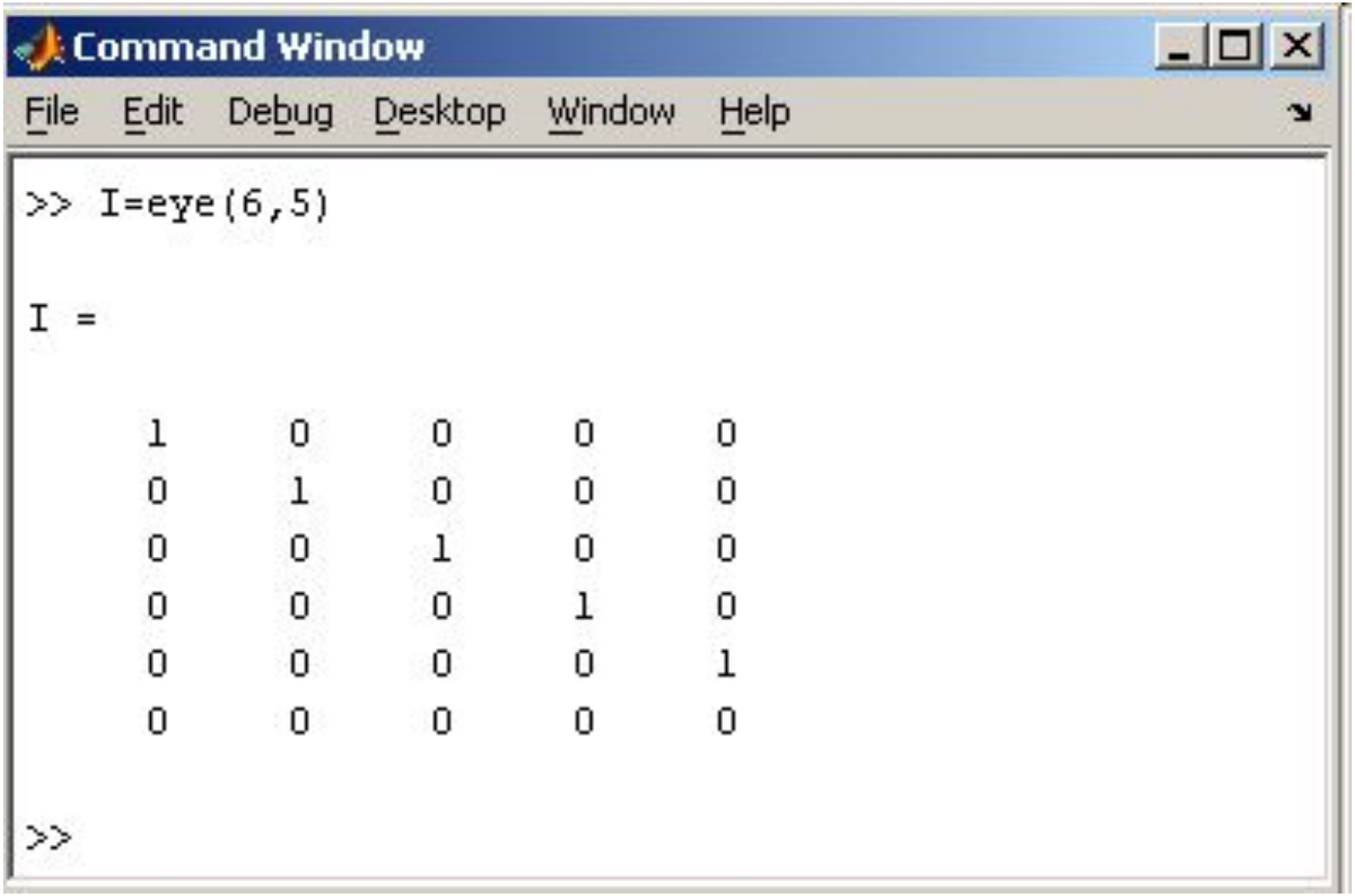
Проверьте полученный результат, умножив матрицу P саму на себя.

2.4. Создание матриц специального вида

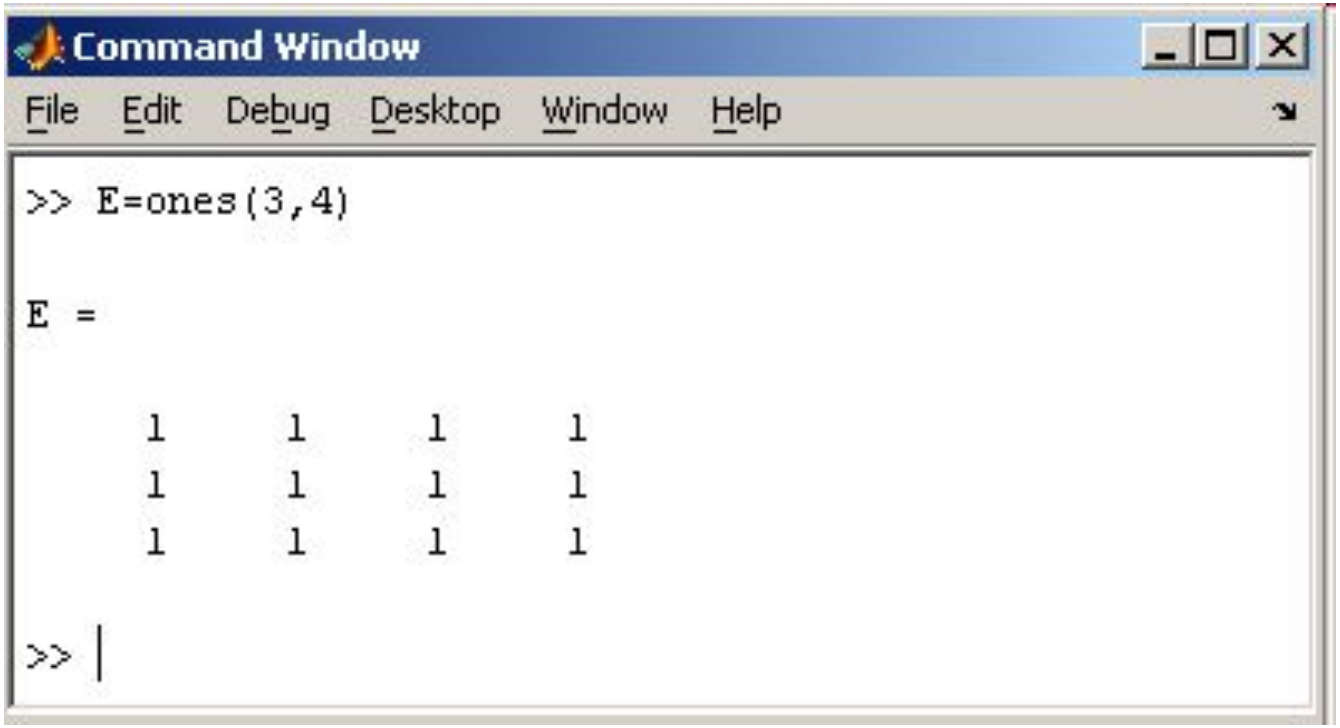
Заполнение прямоугольной матрицы нулями производится встроенной функцией zeros



Единичная матрица создается при помощи функции eye



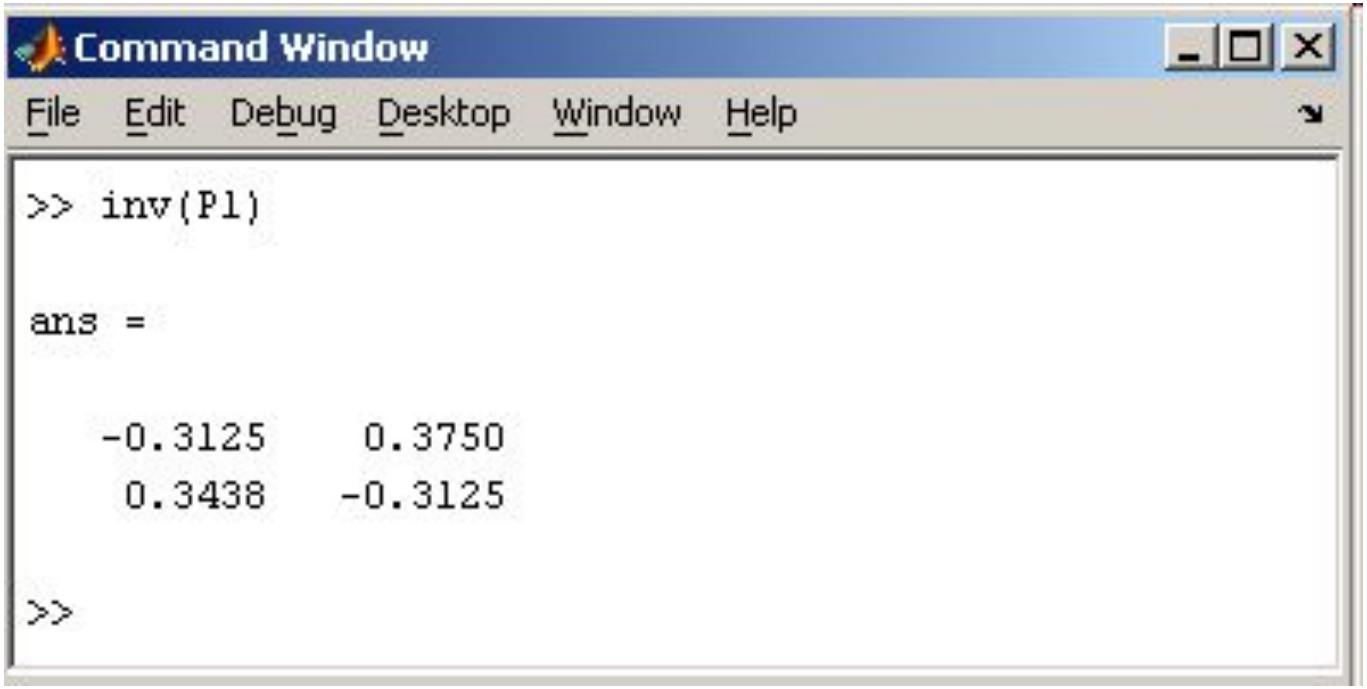
Матрица, состоящая из единиц, образуется в результате вызова функции ones



2.5. Матричные вычисления

MatLab содержит множество различных функций для работы с матрицами. Так, например, транспонирование матрицы производится при помощи апострофа '

Нахождение обратной матрицы проводится с помощью функции inv для квадратных матриц



Более подробно про обработку матричных данных можно узнать, если вывести список всех встроенных функций обработки данных командой help datafun, а затем посмотреть информацию о нужной функции, например help max.

виде прямоугольной таблицы элементов кольца или поля (например, целых — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных чисел), которая представляет собой совокупность строк — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов матрицы задают размер матрицы.

Ввод матриц

Вы можете вводить матрицы в MATLAB несколькими способами:

- вводить полный список элементов
- загружать матрицы из внешних файлов
- генерировать матрицы, используя встроенные функции
- создавать матрицы с помощью ваших собственных функций в M–файлах

Начтем с введения матрицы Дюрера как списка элементов. Вы должны следовать нескольким основным условиям:

- отделять элементы строки пробелами или запятыми
- использовать точку с запятой, ; , для обозначения окончания каждой строки
- окружать весь список элементов квадратными скобками, [ ].

Чтобы ввести матрицу Дюрера просто напишите:

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

MATLAB отобразит матрицу, которую мы ввели,

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

Это точно соответствует числам на гравюре. Если мы ввели матрицу, то она автоматически запоминается средой MATLAB. И мы можем к ней легко обратиться как к A. Сейчас, когда мы имеем A в рабочем пространстве MATLAB,

Операции суммирования элементов, транспонирования и диагонализации матрицы

Вы возможно уже знаете, что особые свойства магического квадрата связаны с различными способами суммирования его элементов. Если вы берёте сумму элементов вдоль какой-либо строки или столбца, или вдоль какой-либо из двух главных диагоналей, вы всегда получите одно и тоже число. Давайте проверим это, используя MATLAB. Первое утверждение, которое мы проверим –

```
sum(A)
```

MATLAB выдаст ответ

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

Когда выходная переменная не определена, MATLAB использует переменную *ans*, коротко от answer – ответ, для хранения результатов вычисления. Мы подсчитали вектор-строку, содержащую сумму элементов столбцов матрицы A. Действительно, каждый столбец имеет одинаковую сумму, магическую сумму, равную 34.

А как насчет сумм в строках? MATLAB предпочитает работать со столбцами матрицы, таким образом, лучший способ получить сумму в строках – это транспонировать нашу матрицу, подсчитать сумму в столбцах, а потом транспонировать результат. Операция транспонирования обозначается апострофом или одинарной кавычкой. Она зеркально отображает матрицу относительно главной диагонали и меняет строки на столбцы. Таким образом

```
A'
```

вызывает

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

А выражение

```
sum(A')'
```

вызывает результат вектор-столбец, содержащий суммы в строках

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

Сумму элементов на главной диагонали можно легко получить с помощью функции *diag*, которая выбирает эту диагональ.

```
diag(A)

ans =
  16
  10
   7
   1
```

А функция

```
sum(diag(A))
```

вызывает

```
ans =
  34
```

Другая диагональ, называемая антидиагональю, не так важна математически, поэтому MATLAB не имеет специальной функции для неё. Но функция, которая вначале предполагалась для использования в графике, *fliplr*, зеркально отображает матрицу слева направо.

```
sum(diag(fliplr(A)))

ans =
    34
```

Таким образом, мы проверили, что матрица на гравюре Дюрера действительно магическая, и научились использовать некоторые матричные операции MATLAB. В последующих разделах мы продолжим использовать эту матрицу для демонстрации дополнительных возможностей MATLAB.

Индексы

Элемент в строке i и столбце j матрицы A обозначается A(i,j). Например, A(4,2) – это число в четвертой строке и втором столбце. Для нашего магического квадрата A(4,2) = 15. Таким образом, можно вычислить сумму элементов в четвертом столбце матрицы A, набрав

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

получим

```
ans =
    34
```

Однако это не самый лучший способ суммирования отдельной строки.

Также возможно обращаться к элементам матрицы через один индекс, A(k). Это обычный способ ссылаться на строки и столбцы матрицы. Но его можно использовать только с двумерными матрицами. В этом случае массив рассматривается как длинный вектор, сформированный из столбцов исходной матрицы.

Так, для нашего магического квадрата, A(8) – это другой способ ссылаться на значение 15, хранящееся в A(4,2).
Если вы пытаетесь использовать значение элемента вне матрицы, MATLAB выдаст ошибку:

```
t=A(4,5)

???  Index exceeds matrix dimensions.
```

С другой стороны, если вы сохраняете значение вне матрицы, то размер матрицы увеличивается.

```
X=A;
X(4,5) = 17

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

Оператор двоеточия

Двоеточие, : , - это один из наиболее важных операторов MATLAB. Он проявляется в различных формах. Выражение

```
1:10
```

- это вектор-строка, содержащая целые числа от 1 до 10

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

Для получения обратного интервала, опишем приращение. Например

```
100:-7:50
```

что дает

```
   100    93    86    79    72    65    58    51
```

или

```
0:pi/4:pi
```

что приводит к

```
0    0.7854    1.5708    2.3562    3.1416
```

Индексное выражение, включая двоеточие, относится к части матрицы.

A(1:k, j)

это первые k элементов j-го столбца матрицы A. Так

sum(A(1:4,4))

вычисляет сумму четвертой строки. Но есть и лучший способ. Двоеточие, само по себе, обращается ко всем элементам в строке и столбце матрицы, а слово *end* – к последней строке или столбцу. Так

```
sum(A(:,end))
```

вычисляет сумму элементов в последнем столбце матрицы A

```
ans =
    34
```

Почему магическая сумма квадрата 4x4 равна 34? Если целые числа от 1 до 16 отсортированы в четыре группы с равными суммами, эта сумма должна быть

```
sum(1:16)/4
```

которая, конечно, равна

```
ans =
    34
```

Функция magic

MATLAB на самом деле обладает встроенной функцией, которая создает магический квадрат почти любого размера. Не удивительно, что эта функция называется *magic*.

```
B=magic(4)

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

Эта матрица почти та же матрица, что и на гравюре Дюрера, и она имеет все те же магические свойства. Единственное отличие заключается в том, что два средних столбца поменялись местами. Для того чтобы преобразовать B в матрицу Дюрера A, переставим их местами.

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

Это означает, что для каждой строки матрицы B элементы переписываются в порядке 1, 3, 2, 4

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

Почему Дюрер переупорядочил столбцы, по сравнению с тем, что использует MATLAB? Без сомнения, он хотел включить дату гравюры, 1514, в нижнюю часть магического квадрата.

Выражения

Как и большинство других языков программирования, MATLAB предоставляет возможность использования математических выражений, но в отличие от многих из них, эти выражения в MATLAB включают матрицы. Основные составляющие выражения:

- переменные
- числа
- операторы
- функции

Переменные

В MATLAB нет необходимости в определении типа переменных или размерности. Когда MATLAB встречает новое имя переменной, он автоматически создает переменную и выделяет соответствующий объем памяти. Если переменная уже существует, MATLAB изменяет ее состав и если это необходимо выделяет дополнительную память. Например,

```
num_students = 25
```

создает матрицу 1x1 с именем num_students и сохраняет значение 25 в ее единственном элементе.

Имена переменных состоят из букв, цифр или символов подчеркивания. MATLAB использует только первые 31 символ имени переменной. MATLAB чувствителен к регистрам, он различает заглавные и строчные буквы. Поэтому A и a – не одна и та же переменная. Чтобы увидеть матрицу связанную с переменной, просто введите название переменной.

Числа

MATLAB использует принятую десятичную систему счисления, с необязательной десятичной точкой и знаками плюс-минус для чисел. Научная система счисления использует букву e для определения множителя степени десяти. Мнимые числа используют i или j как суффикс. Некоторые примеры правильных чисел приведены ниже

3	-99	0.0001
9.6397238	1.60210e-20	6.02252e23
1i	-3.14159j	3e5i

Все числа для хранения используют формат long, определенный стандартом плавающей точки IEEE. Числа с плавающей точкой обладают ограниченной точностью - приблизительно 16 значащих цифр и ограниченным диапазоном – приблизительно от 10^{-308} до 10^{308} (Компьютер VAX использует другой формат чисел с плавающей точкой, но их точность и диапазон приблизительно те же).

Операторы

Выражения используют обычные арифметические операции и правила старшинства.

+	сложение
-	вычитание
*	умножение
/	деление
\	левое деление(описано в разделе Матрицы и Линейная Алгебра в книге "*Using MATLAB*")
^	степень
'	комплексно сопряженное транспонирование
()	определение порядка вычисления

Функции

MATLAB предоставляет большое количество элементарных математических функций, таких как *abs*, *sqrt*, *exp*, *sin*. Вычисление квадратного корня или логарифма отрицательного числа не является ошибкой: в этом случае результатом является соответствующее комплексное число. MATLAB также предоставляет и более сложные функции, включая Гамма функцию и функции Бесселя. Большинство из этих функций имеют комплексные аргументы. Чтобы вывести список всех элементарных математических функций, наберите

```
help elfun
```

Для вывода более сложных математических и матричных функций, наберите

```
help specfun
help elmat
```

соответственно.

Некоторые функции, такие как *sqrt* и *sin*, - встроенные. Они являются частью MATLAB, поэтому они очень эффективны, но их вычислительные детали трудно доступны. В то время как другие функции, такие как *gamma* и *sinh*, реализованы в M-файлах. Поэтому вы можете легко увидеть их код и, в случае необходимости, даже модифицировать его.

Несколько специальных функций предоставляют значения часто используемых констант.

pi	3.14159265…
i	мнимая единица, $\sqrt{-1}$
j	то же самое, что и i
eps	относительная точность числа с плавающей точкой, 2^{-52}
realmin	наименьшее число с плавающей точкой, 2^{-1022}
realmax	наибольшее число с плавающей точкой, $(2-\varepsilon)2^{1023}$
Inf	бесконечность
NaN	не число

Бесконечность появляется при делении на нуль или при выполнении математического выражения, приводящего к переполнению, т.е. к превышению *realmax*. Не число (*NaN*) генерируется при вычислении выражений типа 0/0 или *Inf – Inf*, которые не имеют определенного математического значения.

Имена функций не являются зарезервированными, поэтому возможно изменять их значения на новые, например

```
eps = 1.e-6
```

и далее использовать это значение в последующих вычислениях. Начальное значение может быть восстановлено следующим образом

```
clear eps
```

Выражения

Вы уже познакомились с некоторыми примерами использования выражений в MATLAB. Ниже приведено еще несколько примеров с результатами.

```
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
```

Работа с матрицами

Этот раздел расскажет вам о различных способах создания матриц.

Генерирование матриц

MATLAB имеет четыре функции, которые создают основные матрицы:

zeros	все нули
ones	все единицы
rand	равномерное распределение случайных элементов
randn	нормальное распределение случайных элементов

Некоторые примеры:

```
Z = zeros(2,4)

Z =
     0     0     0     0
     0     0     0     0

F = 5*ones(3,3)

F =
     5     5     5
     5     5     5
     5     5     5
```

Загрузка матриц

Команда *load* считывает двоичные файлы, содержащие матрицы, созданные в MATLAB ранее, или текстовые файлы, содержащие численные данные. Текстовые файлы должны быть сформированы в виде прямоугольной таблицы чисел, отделенных пробелами, с равным количеством элементов в каждой строке. Например, создадим вне MATLAB текстовой файл, содержащий 4 строки:

16.0	3.0	2.0	13.0
5.0	10.0	11.0	8.0
9.0	6.0	7.0	12.0
4.0	15.0	14.0	1.0

Сохраним этот файл под именем magik.dat. Тогда команда

```
load magik.dat
```

прочитает этот файл и создаст переменную magik, содержащую нашу матрицу.

M-файлы

Вы можете создавать свои собственные матрицы, используя M-файлы, которые представляют собой текстовые файлы, содержащие код MATLAB. Просто создайте файл с выражением, которое вы хотите написать в командной строке MATLAB. Сохраните его под именем, заканчивающимся на .m.

Замечание. Для вызова текстового редактора на PC или Mac, выберите **Open** или **New** из меню **File** или нажмите соответствующую кнопку на панели инструментов. Для обращения к текстовому редактору на UNIX используйте символ ! сразу за командой, которую вы используете в строке операционной системы.

Например, создадим файл, включающий следующие 5 строк:

```
A = [ ...
16.0    3.0    2.0    13.0
5.0     10.0   11.0   8.0
9.0     6.0    7.0    12.0
4.0     15.0   14.0   1.0   ];
```

Сохраним его под именем magik.m. Тогда выражение

```
magik
```

прочитает файл и создаст переменную A, содержащую исходную матрицу.

Объединение

Объединение – это процесс соединения маленьких матриц для создания больших. Фактически, вы создали вашу первую матрицу объединением её отдельных элементов. Пара квадратных скобок – это оператор объединения. Например, начнем с матрицы A (магического квадрата 4x4) и сформируем

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

Результатом будет матрица 8x8, получаемая соединением четырех подматриц

```
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
```

Это матрица лишь наполовину является магической. Её элементы представляют собой комбинацию целых чисел от 1 до 64, а суммы в столбцах точно равны значению для магического квадрата 8x8.

```
sum(B)

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

Однако, суммы в строках этой матрицы (*sum(B')'*) не все одинаковы. Необходимо провести дополнительные операции, чтобы сделать эту матрицу действительно магическим квадратом 8x8.

Удаление строк и столбцов

Вы можете удалять строки и столбцы матрицы, используя просто пару квадратных скобок. Рассмотрим

```
X = A;
```

Теперь удалим второй столбец матрицы X.

```
X(:,2) = []
```

Эта операция изменит X следующим образом

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

Если вы удаляете один элемент матрицы, то результат уже не будет матрицей. Так выражение

```
X(1,2) = []
```

результатом вычисления выдаст ошибку. Однако использование одного индекса удаляет отдельный элемент или последовательность элементов и преобразует оставшиеся элементы в вектор-строку. Так

```
X(2:2:10) = []
```

выдаст результат

```
X =
    16     9     3     6    13    12     1
```

Командное окно

До сих пор, мы использовали только командную строку MATLAB, печатая команды и выражения и наблюдая результаты. В этой главе описано несколько способов изменения внешнего вида командного окна. Если ваша система позволяет вам выбирать шрифт, то мы рекомендуем использовать шрифты с фиксированной шириной, такие как Fixedsys или Courier, для обеспечения правильного межстрочного интервала.

Команда format

Команда format управляет численным форматом значений, выводимых MATLAB. Эта операция влияет только на то, как числа изображаются на экране, но не влияет на то, как их вычисляет и сохраняет MATLAB. Ниже представлены различные форматы чисел, используемые для отображения вектора x с компонентами различных величин.

```
x = [4/3  1.2345e-6]

format short

1.3333    0.0000

format short e

1.3333e+000  1.2345e-006

format short g

1.3333  1.2345e-006

format long

1.33333333333333   0.00000123450000
```

Если самый большой элемент матрицы больше 10^3 или самый маленький меньше 10^{-3}, MATLAB применяет общий масштабный коэффициент для форматов *short* и *long*.
В добавление к командам *format*, рассмотренным выше

```
format compact
```

убирает много пустых линий, появляющихся на выходе. Это позволяет вам видеть больше информации на экране. Если вы хотите изменить контроль над форматом выходных данных, используйте функции *sprintf* и *fprintf*.

Сокращение выходных данных

Если вы наберете выражение и нажмете **Return** или **Enter**, MATLAB автоматически выведет результат на экран. Однако если в конце строки вы поставите точку с запятой, MATLAB проведет вычисления, но не отобразит их. Это часто бывает нужно при создании больших матриц. Например,

```
A = magic(100);
```

Длинные командные строки

Если выражение не умещается на одной строке, используйте троеточие, а за ним **Return** или **Enter**, для обозначения того, что выражение продолжается на следующей строке. Например

```
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 =, +, - не обязательны, но улучшают читаемость текста.

Редактор командной строки

Различные стрелки и управляющие клавиши на вашей клавиатуре позволяют вам вызывать, редактировать и многократно использовать команды, набранные ранее. Например, предположим, что вы допустили ошибку при вводе

```
rho = (1 + sqt(5))/2
```

Вы ошиблись в написании *sqrt*. MATLAB ответит вам предупреждением

```
Undefined function or variable 'sqt'.
```

Вместо того, чтобы заново набирать всю строку, просто нажмите клавишу ↑. Тогда на экране изобразится ошибочная команда. Используйте клавишу ← для перемещения курсора и вставки пропущенной буквы r. Повторное использование клавиши ↑ вызовет предыдущие строки. Наберите несколько символов, и тогда клавиша ↑ найдет предыдущую строку, которая начинается с них.

Список доступных клавиш редактирования в командной строке отличается у разных компьютеров. Поэкспериментируйте, чтобы узнать, какие из нижеследующих клавиш доступны на вашей машине. (Многие из этих клавиш будут знакомы пользователям редактора EMACS.)

↑	ctrl-p	Вызов предыдущей строки
↓	ctrl-n	Вызов последующей строки
←	ctrl-b	Движение назад на один символ
→	ctrl-f	Движение вперед на один символ
ctrl-→	ctrl-r	Движение вправо на одно слово
ctrl-←	ctrl-l	Движение влево на одно слово
home	ctrl-a	Переход на начало строки
end	ctrl-e	Переход на конец строки
esc	ctrl-u	Очистка строки
del	ctrl-d	Удаление символа за курсором
backspace	ctrl-h	Удаление символа перед курсором
	ctrl-k	Удаление до конца строки

Графика

MATLAB имеет широкие возможности для графического изображения векторов и матриц, а также для создания комментариев и печати графики. Эта глава описывает несколько наиболее важных графических функций и дает примеры их применения.

Создание графика

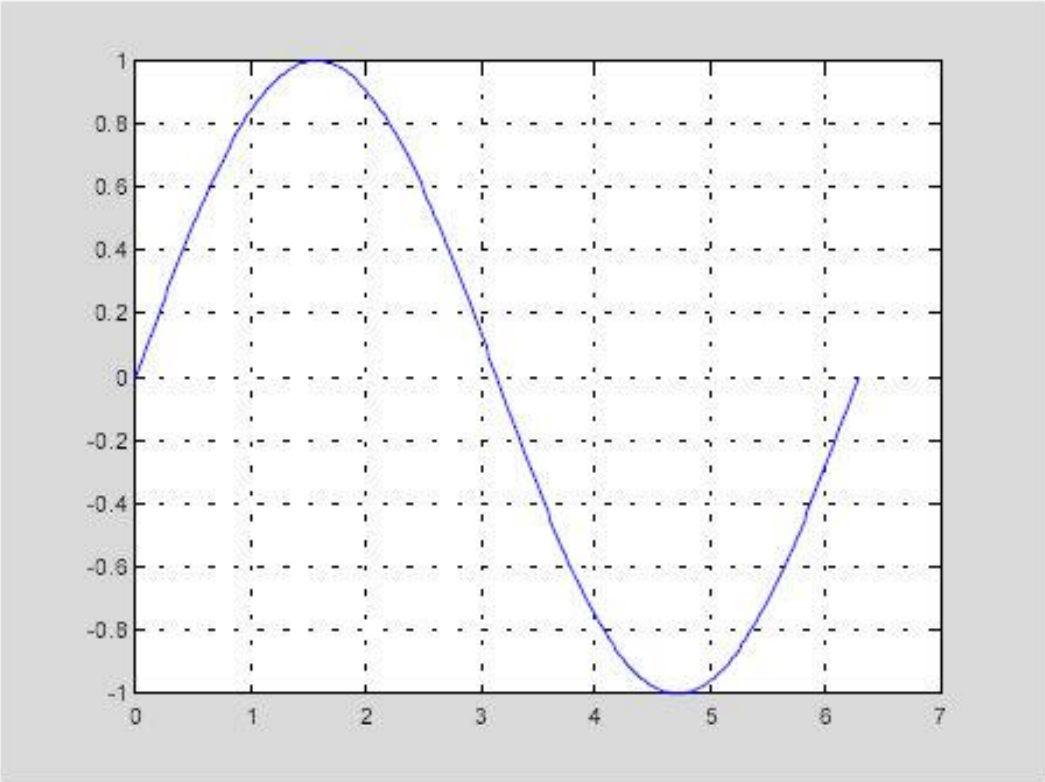
Функция *plot* имеет различные формы, связанные с входными параметрами, например *plot(y)* создает кусочно-линейный график зависимости элементов y от их индексов. Если вы задаете два вектора в качестве аргументов, *plot(x,y)* создаст график зависимости y от x.

Создание графика

Функция *plot* имеет различные формы, связанные с входными параметрами, например *plot(y)* создает кусочно-линейный график зависимости элементов y от их индексов. Если вы задаете два вектора в качестве аргументов, *plot(x,y)* создаст график зависимости y от x.

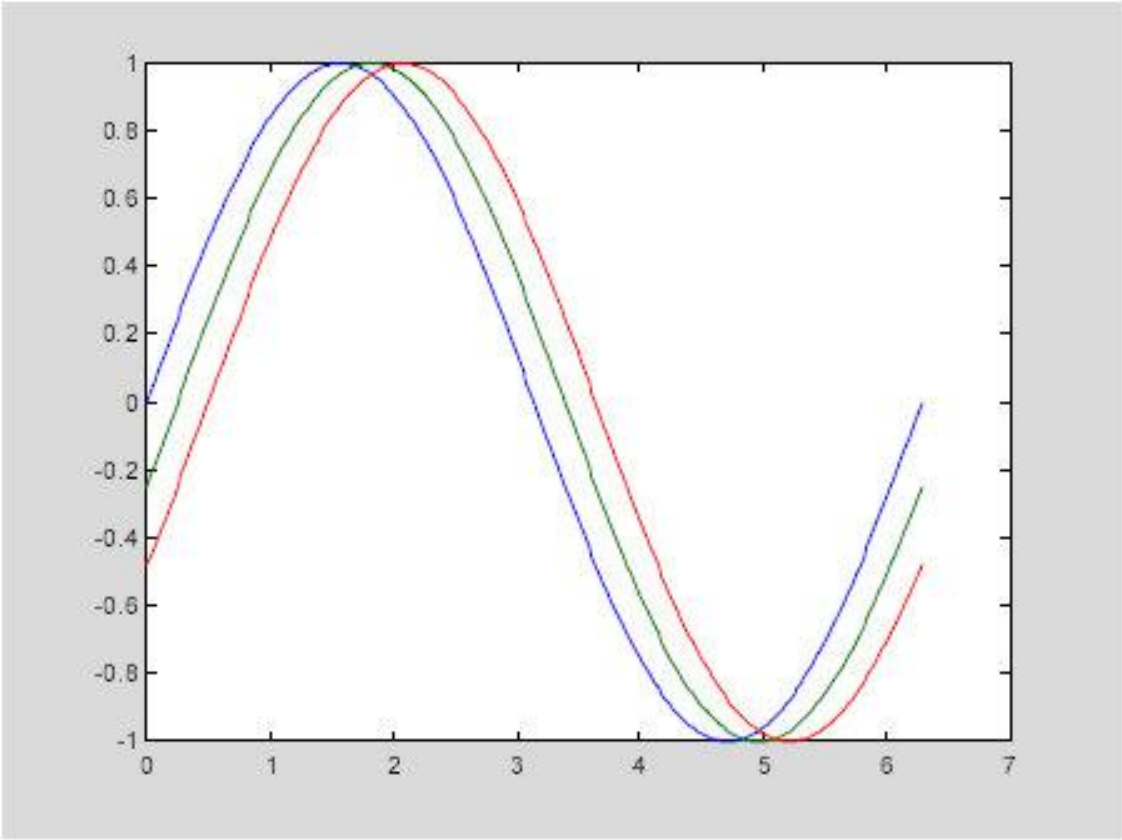
Например, для построения графика значений функции *sin* от нуля до 2π, сделаем следующее

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



Вызов функции *plot* с многочисленными парами x-y создает многочисленные графики. MATLAB автоматически присваивает каждому графику свой цвет (исключая случаи, когда это делает пользователь), что позволяет различать заданные наборы данных. Например, следующие три строки отображают график близких функций, и каждой кривой соответствует свой цвет:

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



Возможно изменение цвета, стиля линий и маркеров, таких как знаки плюс или кружки, следующим образом

```
plot(x, y, 'цвет_стиль_маркер')
```

цвет_стиль_маркер это 1-, 2-, 3- символьная строка (заключенная в одинарные кавычки), составленная из типов цвета, стиля линий и маркеров:

- Символы, относящие к цвету: *'c'*, *'m'*, *'y'*, *'r'*, *'g'*, *'b'*, *'w'* и *'k'*. Они обозначают голубой, малиновый, желтый, красный, зеленый, синий, белый и черный цвета соответственно.
- Символы, относящиеся к типу линий: *'–'* для сплошной, *'– –'* для разрывной, *':'* для пунктирной, *'–.'* для штрихпунктирной линий и 'none' для её отсутствия.
- Наиболее часто встречающиеся маркеры *'+'*, *'o'*, *'*'* и *'x'*.

Например, выражение

```
plot(x,y,'y:+')
```

строит желтый пунктирный график и помещает маркеры *'+'* в каждую точку данных. Если вы определяете только тип маркера, но не определяете тип стиля линий, то MATLAB выведет только маркеры.

Окна изображений

Функция *plot* автоматически открывает новое окно изображения (далее окно), если до этого его не было на экране. Если же оно существует, то *plot* использует его по умолчанию. Для открытия нового окна и выбора его по умолчанию, наберите

```
figure
```

Для того, чтобы сделать существующеё окно текущим -

```
figure(n)
```

где n – это номер в заголовке окна. В этом случае результаты всех последующих команд будут выводиться в это окно.

Добавление кривых на существующий график

Команда *hold* позволяет добавлять кривые на существующий график. Когда вы набираете

```
hold on
```

MATLAB не стирает существующий график, а добавляет в него новые данные, изменяя оси, если это необходимо. Например, следующий элемент кода вначале создает контурные линии функции *peaks*, а затем накладывает псевдоцветной график той же функции:

```
[x,y,z] = peaks;
contour(x,y,z,20,'k')
hold on
pcolor(x,y,z)
shading interp
```

Команда *hold on* является причиной того, что график *pcolor* комбинируется с графиком *contour* в одном окне

Подграфики

Функция *subplot* позволяет выводить множество графиков в одном окне или распечатывать их на одном листе бумаги.

```
subplot(m,n,p)
```

разбивает окно изображений на матрицу m на n подграфиков и выбирает p-ый подграфик текущим. Графики нумеруются вдоль первого в верхней строке, потом во второй и т.д. Например, для того, чтобы представить графические данные в четырех разных подобластях окна необходимо выполнить следующее:

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1)
mesh(X)
subplot(2,2,2); mesh(Y)
subplot(2,2,3); mesh(Z)
subplot(2,2,4); mesh(X,Y,Z)
```

Мнимые и комплексные данные

Если аргумент функции *plot* комплексное число, то мнимая часть игнорируется, за исключением случая, когда комплексный аргумент один. Для этого специального случая происходит построение графика зависимости реальной части аргумента от мнимой. Поэтому

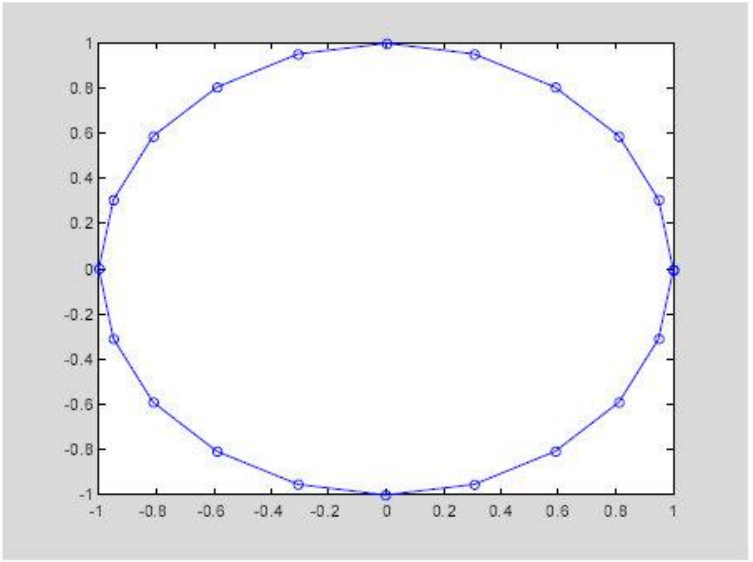
```
plot(Z)
```

где Z комплексный вектор или матрица, эквивален

```
plot(real(Z),imag(Z))
```

Например,

```
t = 0:pi/10:2*pi;
plot(exp(i*t),'-o')
```



отобразит двадцатисторонний многоугольник с маленькими кружками на вершинах.

Управление осями

Функция *axis* имеет несколько возможностей для настройки масштаба, ориентации и коэффициента сжатия.

Обычно MATLAB находит максимальное и минимальное значение и выбирает соответствующий масштаб и маркитирование осей. Функция *axis* заменяет значения по умолчанию предельными значения, вводимыми пользователем.

```
axis( [xmin xmax ymin ymax] )
```

В функции *axis* можно также использовать ключевые слова для управления внешним видом осей. Например

```
axis square
```

создает x и y оси равной длины, а

```
axis equal
```

создает отдельные отметки приращений для x и y осей одинаковой длины. Так функция

```
plot(exp(i*t))
```

следующая либо за *axis square*, либо за *axis equal* превращает овал в правильный круг.

```
axis auto
```

возвращает значения по умолчанию и переходит в автоматический режим.

```
axis on
```

включает обозначения осей и метки промежуточных делений.

```
axis off
```

выключает обозначения осей и метки промежуточных делений.

```
grid off
```

выключает сетку координат, а

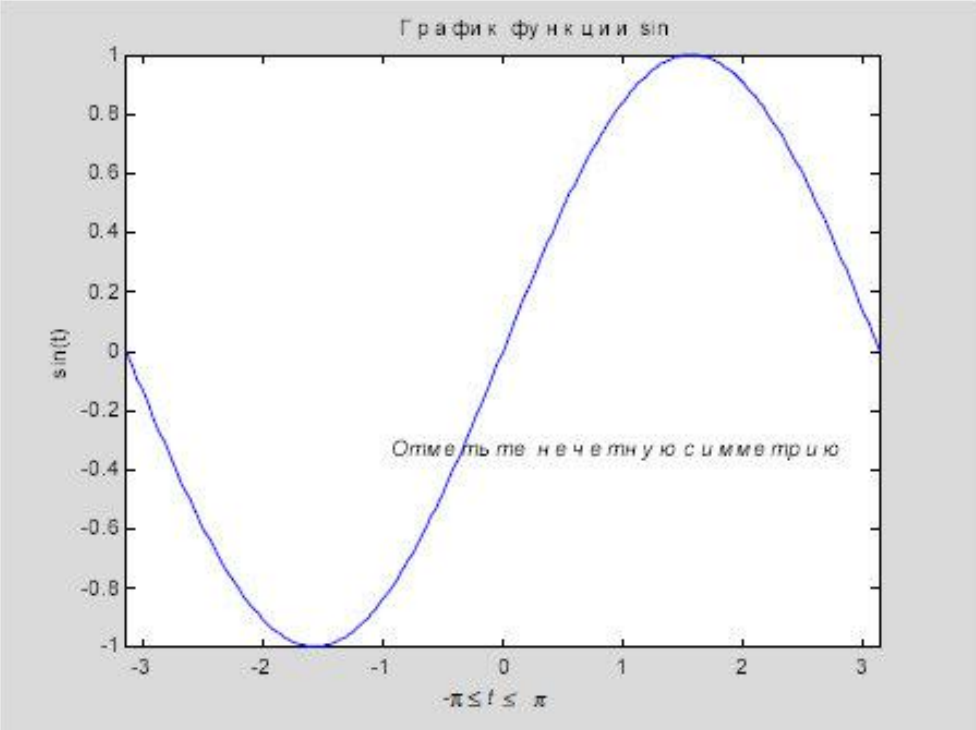
```
grid on
```

включает её заново.

Подписи к осям и заголовки

Функции *xlabel*, *ylabel*, *zlabel* добавляют подписи к соответствующим осям, функция title добавляет заголовок в верхнюю часть окна, а функция text вставляет текст в любое место графика. Использование TEX представления позволяет применять греческие буквы, математические символы и различные шрифты. Следующий пример демонстрирует эту возможность.

```
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} ' )
```



Графика

MATLAB имеет широкие возможности для графического изображения векторов и матриц, а также для создания комментариев и печати графики. Эта глава описывает несколько наиболее важных графических функций и дает примеры их применения.

Создание графика

Функция *plot* имеет различные формы, связанные с входными параметрами, например *plot(y)* создает кусочно-линейный график зависимости элементов y от их индексов. Если вы задаете два вектора в качестве аргументов, *plot(x,y)* создаст график зависимости y от x.

1. Построение двумерных графиков функций

В результате вычислений в системе MATLAB обычно получается большой массив данных, который трудно анализировать без наглядной визуализации.

Поэтому система визуализации, встроенная в MATLAB, придаёт этому пакету особую практическую ценность.

Графические возможности системы MATLAB являются мощными и разнообразными. В первую очередь целесообразно изучить наиболее простые в использовании возможности. Их часто называют высокоуровневой графикой.

Например, нет ничего проще, чем построить график функции одной вещественной переменной.

Следующие команды

x = 0 : 0.01 : 2;

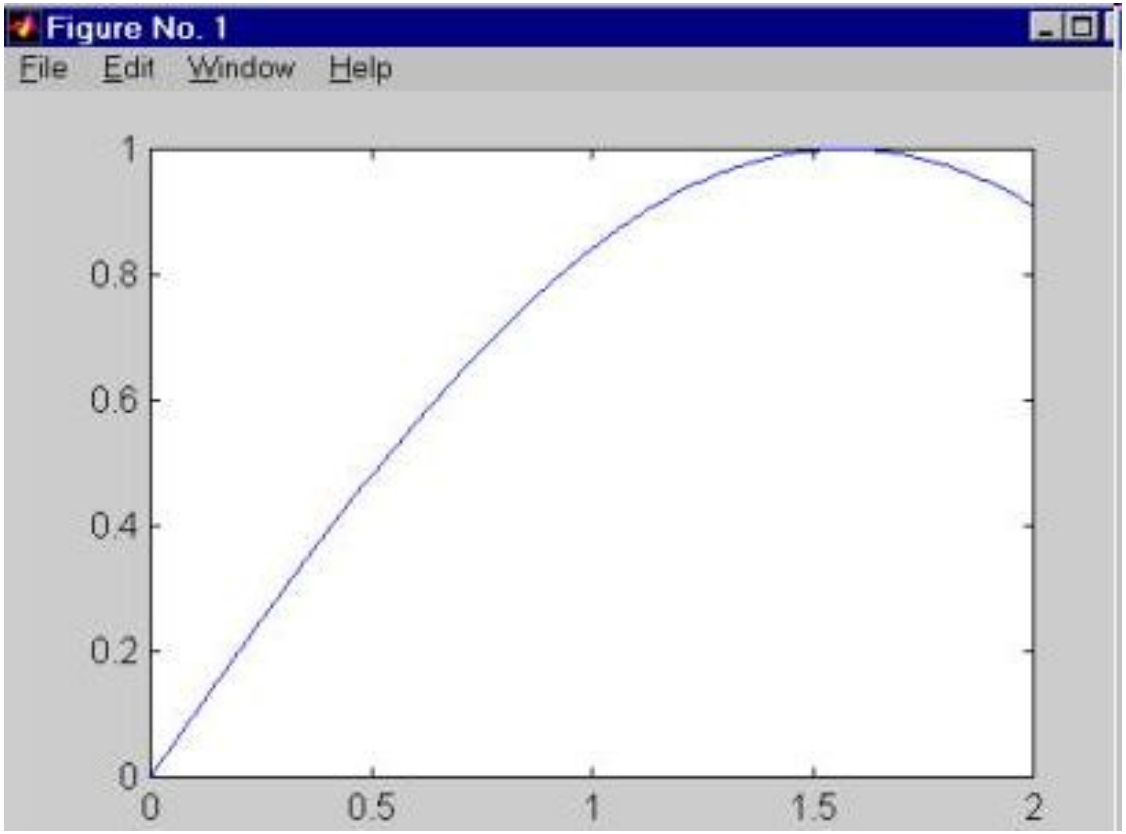
y = sin(x);

вычисляют массив y значений функции sin для заданного набора аргументов.

После этого командой

plot(x , y)

получим график функции:



MATLAB показывает графические объекты в специальных графических окнах, имеющих в заголовке слово Figure (изображение, внешний вид, фигура).

При построении графиков функций сразу проявляется тот факт, что очень большую часть работы MATLAB берёт на себя.

Мы в командной строке ввели лишь одну команду, а система сама создала графическое окно, построила оси координат, вычислила диапазоны изменения переменных x и y;

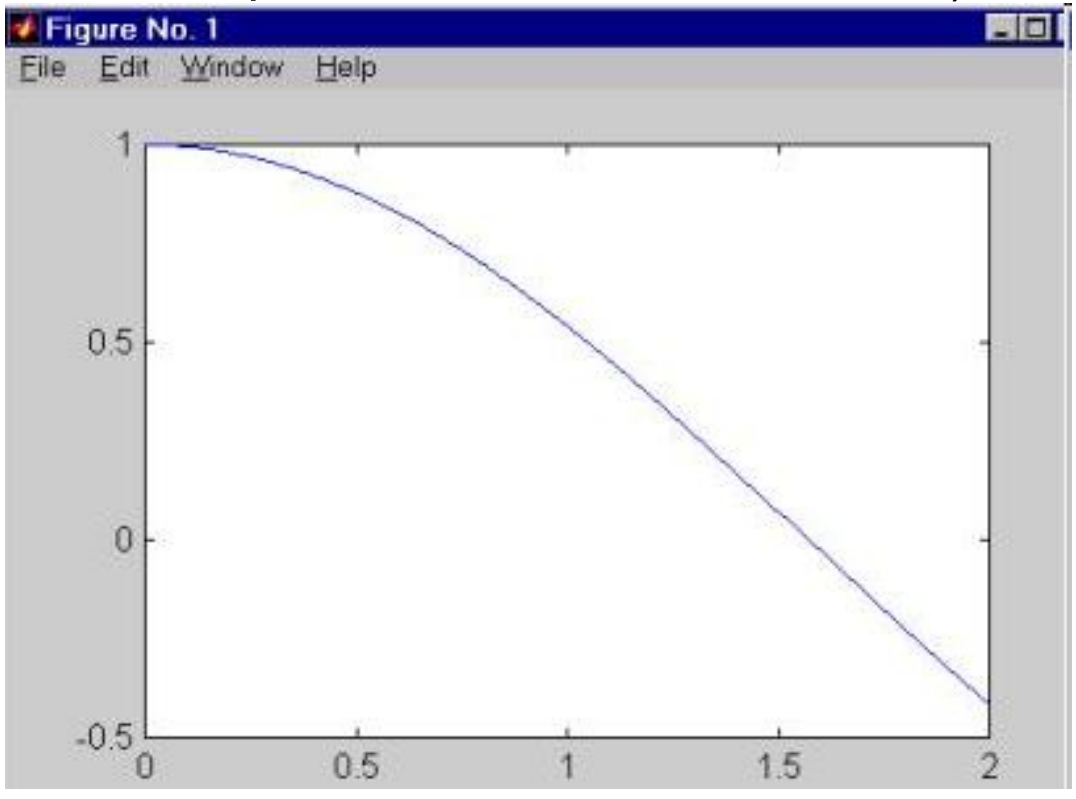
проставила на осях метки и соответствующие им числовые значения, провела через опорные точки график функции некоторым, выбранным по умолчанию, цветом;

в заголовке графического окна надписала номер графика в текущем сеансе работы.

Если мы, не убирая с экрана дисплея первое графическое окно, вводим и исполняем ещё один набор команд:

```
 x = 0 : 0.01 : 2;
z = cos( x );
plot( x , z )
```

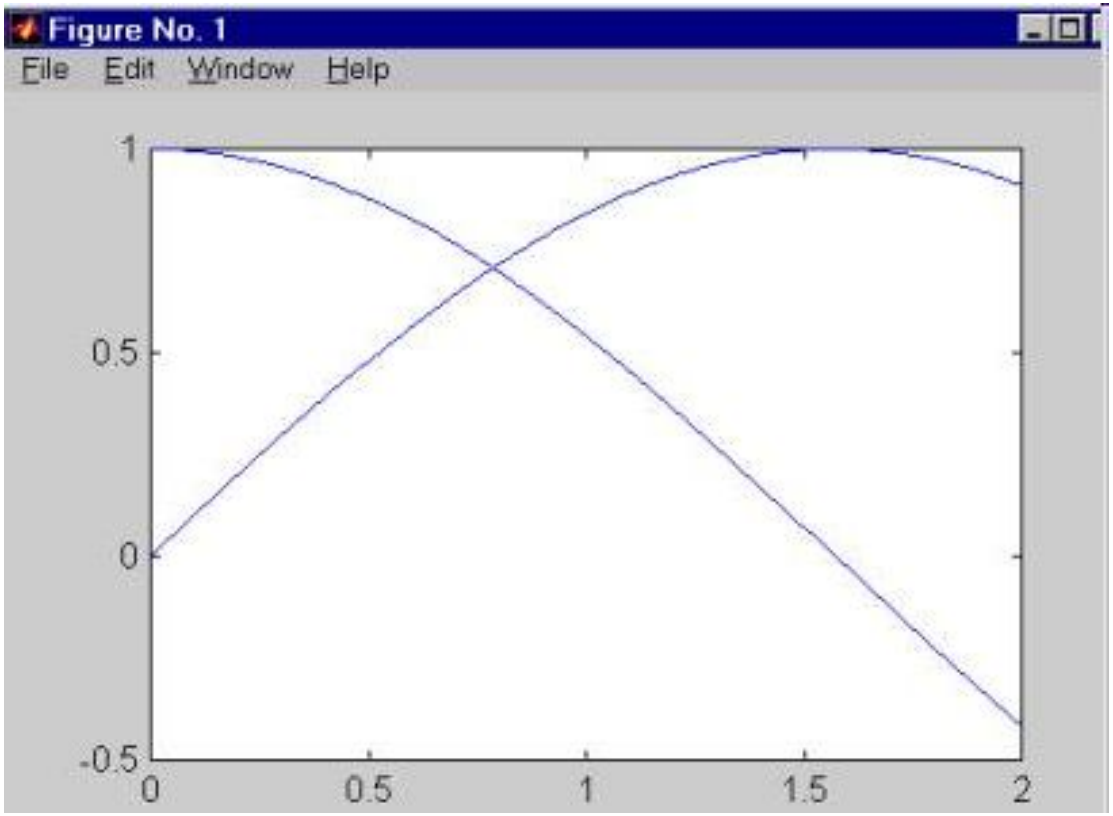
то получаем новый график функции в том же самом графическом окне (при этом старые оси координат и график в нём пропадают - этого можно также добиться командой clf, а командой cla удаляют только график с приведением осей координат к их стандартным диапазонам от 0 до 1):



Если нужно второй график провести "поверх первого графика", то перед исполнением второй графической команды plot, нужно выполнить команду

hold on

которая предназначена для удержания текущего графического окна. В результате будет получено следующее изображение:



Того же самого можно добиться, потребовав от функции plot построить сразу несколько графиков в рамках одних и тех же осей координат:

```
x = 0 : 0.01 : 2;
y = sin( x ); z = cos( x );
plot( x , y , x , z )
```

У такого способа есть ещё одно (кроме экономии на команде hold on) преимущество, так как разные графики автоматически строятся разным цветом.

К недостаткам указанных способов построения нескольких графиков в пределах одних и тех же осей координат относится использование одного и того же диапазона изменения координат, что при несопоставимым значениях двух функций приведёт к плохому изображению графика одной из них.

Если всё же нужно одновременно визуализировать несколько графиков так, чтобы они не мешали друг другу, то это можно сделать двумя способами.

Во-первых, можно построить их в разных графических окнах.

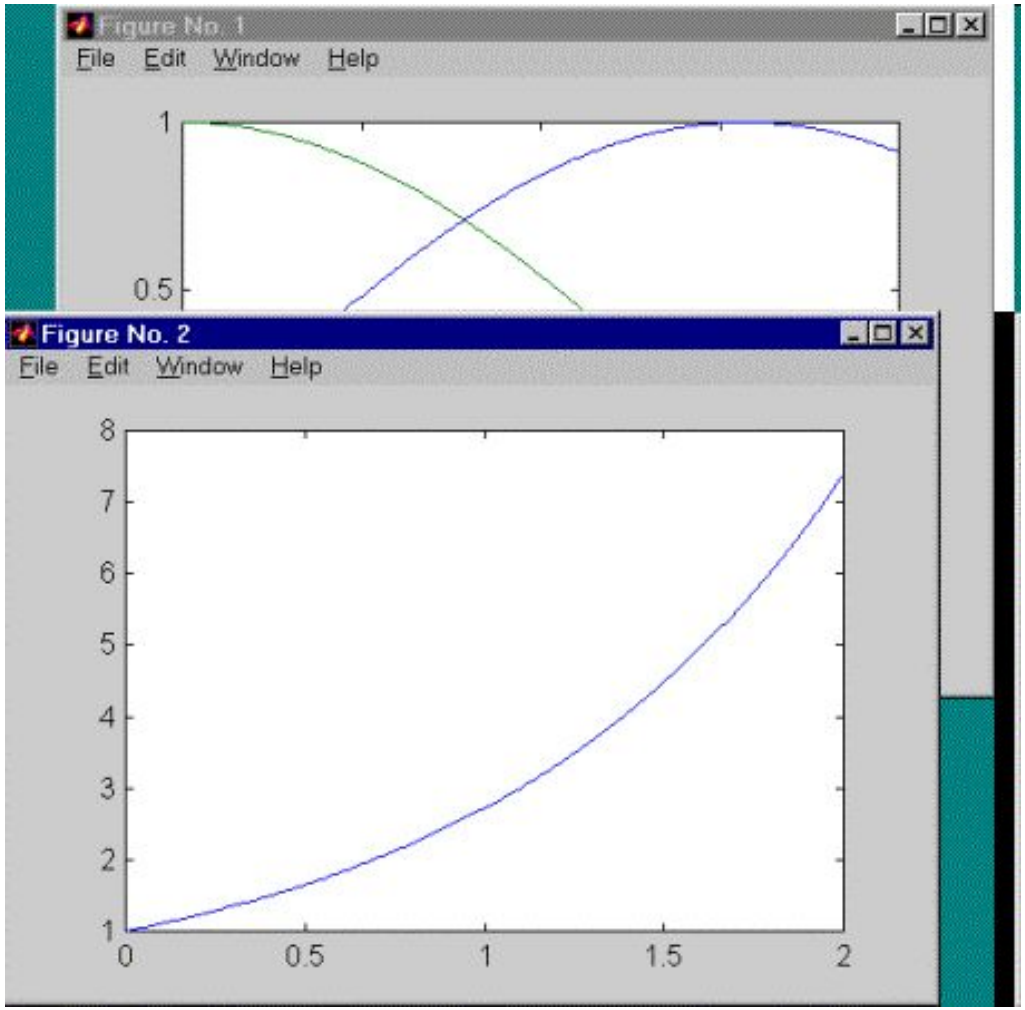
Например, построив графики функций sin и cos в пределах одного графического окна (показано выше), вычисляем значения для функции exp:

```
w = exp( x );
```

После этого выполняем команды

```
figure; plot( x , w )
```

которые построят график функции exp в новом графическом окне, так как команда figure создаёт новое (добавочное) графическое окно, и все последующие за ней команды построения графиков выводят их в новое окно:



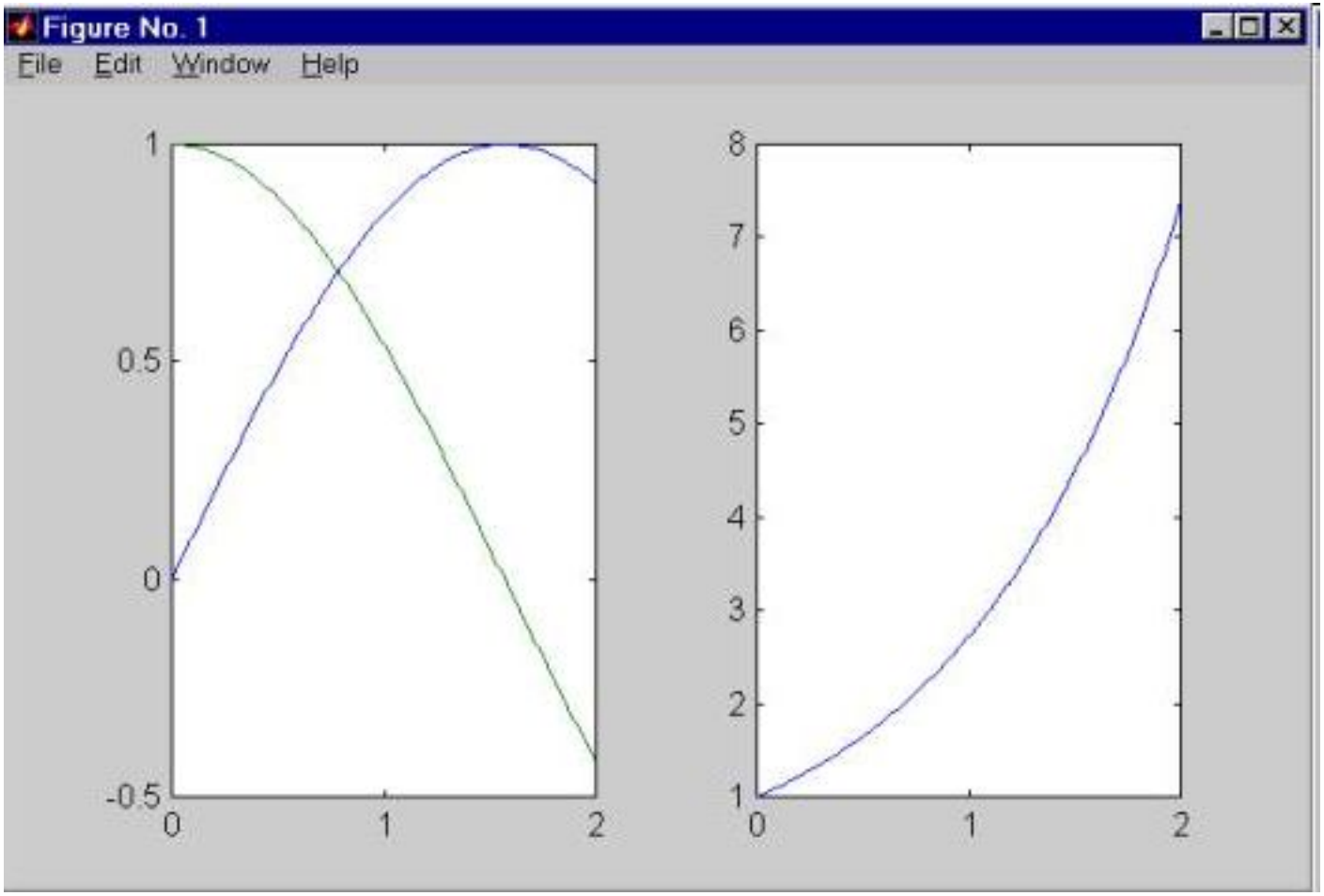
В результате в первом графическом окне (Figure No. 1) по вертикальной оси переменные изменяются в диапазоне от -0.5 до 1, а во втором графическом окне (Figure No. 2) - от 1 до 8.

Вторым решением рассматриваемой задачи показа сразу нескольких графиков без конфликта диапазонов осей координат является использование функции subplot.

Эта функция позволяет разбить область вывода графической информации на несколько подобластей, в каждую из которых можно вывести графики различных функций.

Например, для ранее выполненных вычислений с функциями sin, cos и exp, строим графики первых двух функций в первой подобласти, а график третьей функции - во второй подобласти одного и того же графического окна:

subplot(1,2,1); plot(x,y,x,z)
subplot(1,2,2); plot(x,w)
в результате чего получаем графическое окно следующего вида:



Диапазоны изменения переменных на осях координат этих подобластей независимы друг от друга.

Функция subplot принимает три числовых аргумента, первый из которых равен числу рядов подобластей, второе число равно числу колонок подобластей, а третье число - номеру подобласти (номер отсчитывается вдоль рядов с переходом на новый ряд по исчерпанию).

Если для одиночного графика диапазоны изменения переменных вдоль одной или обоих осей координат слишком велики, то можно воспользоваться функциями построения графиков в логарифмических масштабах.

Для этого предназначены функции semilogx, semilogy и loglog. Подробную информацию по использованию этих функций всегда можно получитьпри помощи команды

help имя_функции

набираемой с клавиатуры и выполняемой в командном окне системы MATLAB.

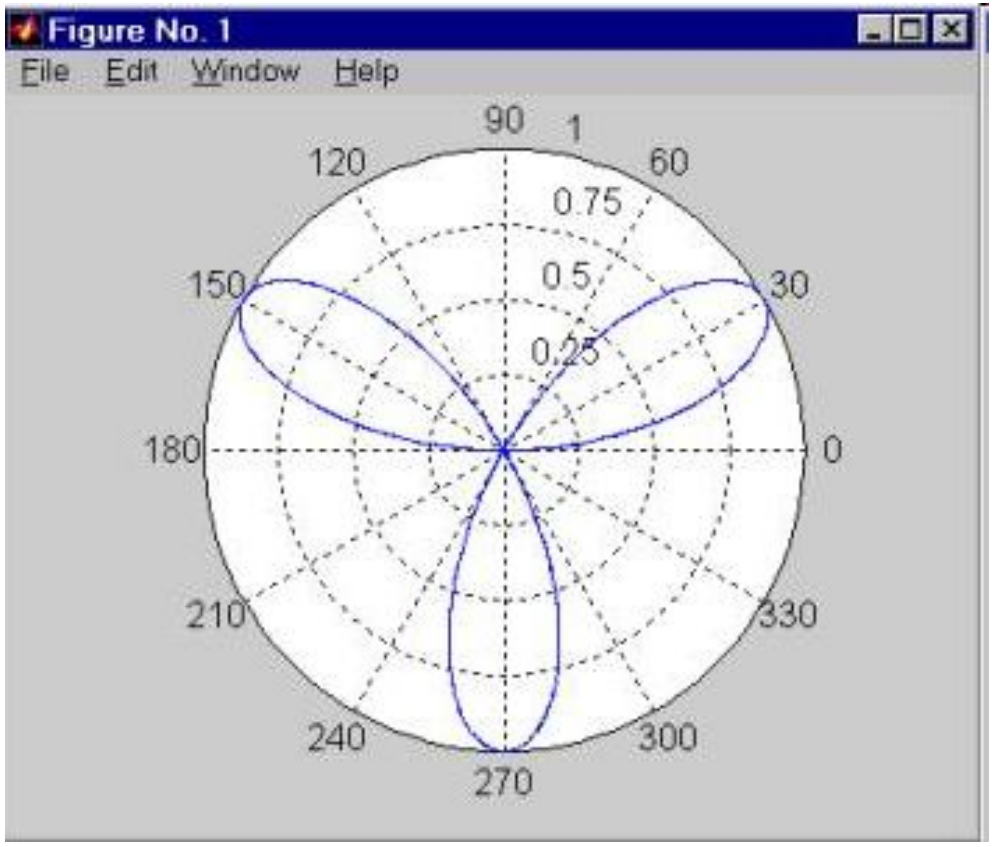
Итак, уже рассмотренные примеры показывают, как подсистема высокоуровневой графики MATLABa легко справляется с различными случаями построения графиков, не требуя слишком большой работы от пользователя.

Ещё одним таким примером является построение графиков в полярных координатах. Например, если нужно построить график функции r = sin(3f) в полярных координатах, то следующие несколько команд

phi = 0 : 0.01 : 2*pi; r = sin(3* phi);

polar(phi , r)

решают эту задачу:



2. Оформление графиков функций

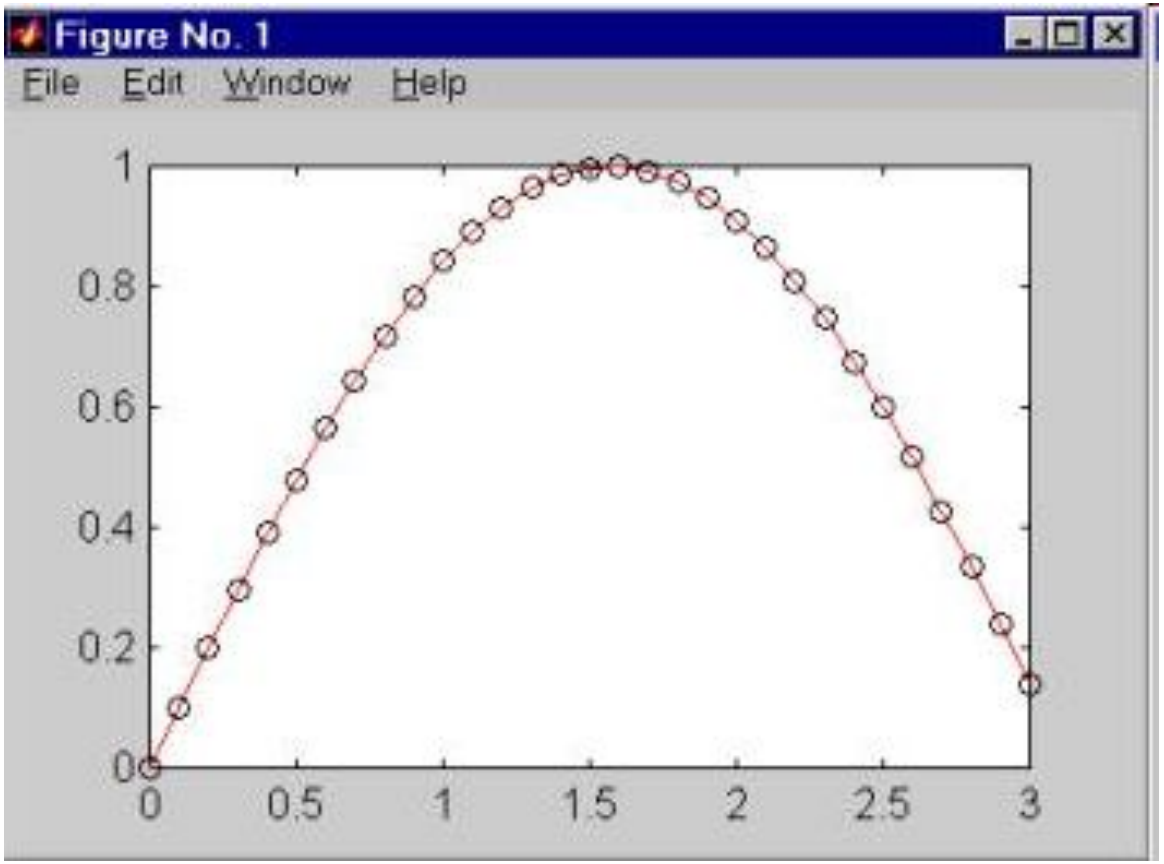
Рассмотрим ряд вопросов, связанных с внешним видом графиков функций - цветом и стилем линий, которым проведены сами графики, а также различными надписями в пределах графического окна.

Например, следущие команды
x = 0 : 0.1 : 3; y = sin(x);
plot(x, y, 'r-', x, y, 'ko')
позволяют придать графику вид красной сплошной линии, на которой в дискретных вычисляемых точках проставляются чёрные окружности.

Здесь функция plot дважды строит график одной и той же функции, но в двух разных стилях.

Первый из этих стилей отмечен как 'r-', что означает проведение линии красным цветом (буква r), а штрих означает проведение сплошной линии.

Второй стиль, помеченный как 'ko' означает проведение чёрным цветом (буква k) окружностей (буква o) на месте вычисляемых точек.



В общем случае, функция

plot(x1, y1, s1, x2, y2, s2, …)
позволяет объединить несколько графиков функций y1(x1), y2(x2),…, проведя их со стилями s1, s2, …

В случае функции вида

plot(x1, y1, s1, x1, y1, s2)

мы можем провести линию графика единственной функции y1(x1) одним цветом, а точки на нём (вычисляемые точки) - другим цветом.

Стили s1, s2,… задаются в виде набора трёх символьных маркеров, заключенных в одиночные кавычки.

Первый (не обязательно по порядку) из этих маркеров задаёт тип линии:

Маркер	Тип линии
-	непрерывная
--	штриховая
:	пунктирная
-.	штрих-пунктирная

Второй маркер задаёт цвет:

Маркер	Цвет линии
c	голубой
m	фиолетовый
y	жёлтый
r	красный
g	зелёный
b	синий
w	белый
k	чёрный

Последний маркер задаёт тип проставляемых "точек":

Маркер	Тип точки
.	точка
+	плюс
*	звёздочка
o	кружок
x	крестик

Можно указывать не все три маркера. Тогда используются необходимые маркеры, установленные "по умолчанию".

Порядок, в котором указываются маркеры, не является существенным, то есть 'r+-' и '-+r' приводят к одинаковому результату.

Если в строке стиля поставить маркер типа точки, но не проставить маркер на тип линии, то тогда отображаются только вычисляемые точки, а непрерывной линией они не соединяются.

Наиболее мощным способом оформления графиков функций (и выполнения других графических работ) является дескрипторный метод, полное изучение которого относится к так называемой низкоуровневой графике системы MATLAB.

Выше мы оформляли график функции sin с помощью непрерывной красной линии и чёрных кружков. Теперь попробуем ограничиться лишь непрерывной линией, но очень толстой. Как это можно сделать? Вот простое решение на базе дескрипторной графики:

```
 x = 0 : 0.1 : 3; y = sin( x );
hPlot = plot( x, y );
set( hPlot, 'LineWidth', 7 );
```

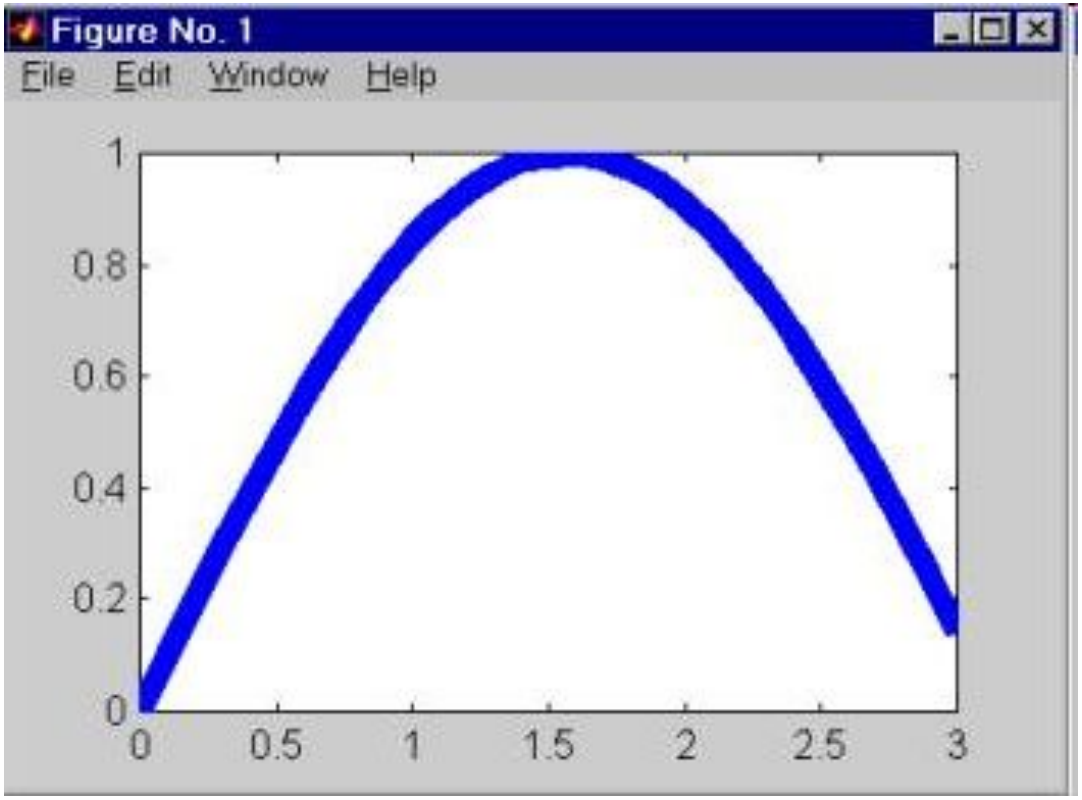
Функция plot через опорные (вычисленные) точки с координатами x, y проводит отрезки прямых линий. Прямые линии в системе MATLAB представляют собой графические объекты типа Line.

Эти объекты имеют огромное число свойств и характеристик, которые можно менять. Доступ к этим объектам осуществляется по их описателям (дескрипторам; handles).

Описатель объекта Line, использованного для построения нашего графика, возвращается функцией plot. Мы его запоминаем для дальнейшего использования в переменной hPlot.

Затем этот описатель предлагается функции set для опознания конкретного графического объекта. Именно для такого опознанного объекта функция set изменяет характеристики, которые указаны в других аргументах при вызове функции set.

В нашем примере мы указали свойство 'LineWidth' (толщина линии), для которого задали новое значение 7 (а по умолчанию - 0.5). В результате получается следующая картина:



Текущее значение любого параметра (атрибута; характеристики) графического объекта можно узнать с помощью функции get.

Например, если после получения показанного на рисунке графика ввести и исполнить команду
width = get(hPlot, 'LineWidth')

то для переменной width будет получено значение 7.

Теперь от оформления непосредственно линий перейдём к оформлению осей системы координат, к надписям на осях и так далее.

MATLAB выбирает пределы на горизонтальной оси равными указанным для независимой переменной. Для зависимой переменной по вертикальной оси MATLAB вычисляет диапазон изменения значений функции.

Затем этот вычисленный диапазон приписывается вертикальной оси системы координат, так что график функции оказывается как бы вписанным в прямоугольник.

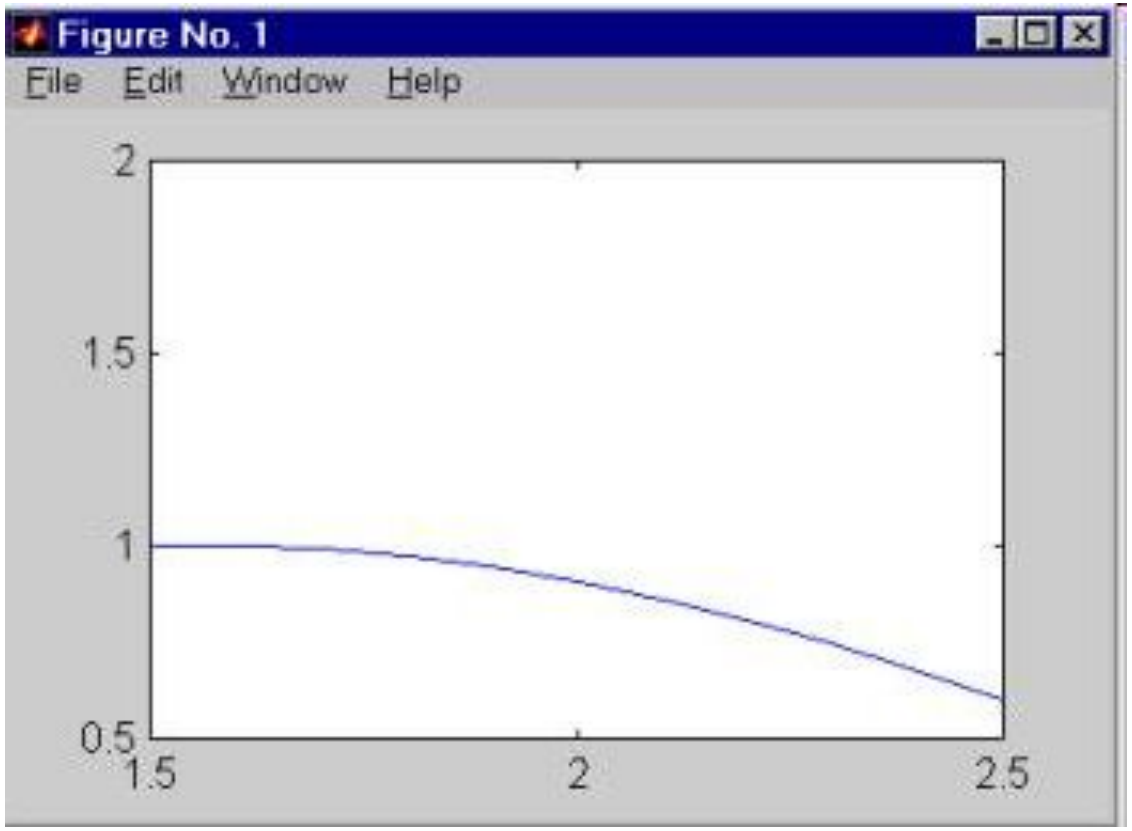
Если мы хотим отказаться от этой особенности масштабирования при построении графиков в системе MATLAB, то мы должны явным образом навязать свои пределы изменения переменных по осям координат. Это делается с помощью функции
axis([xmin, xmax, ymin, ymax])

причём команду на выполнение этой функции можно вводить с клавиатуры сколько угодно раз уже после построения графика функции, чтобы, глядя на получающиеся визуальные изображения, добиться наилучшего восприятия.

Такое масштабирование позволяет получить подробные изображения тех частей графика, которые вызывают наибольший интерес в конкретном исследовании.

Например, для ранее полученного графика функции sin, можно сузить пределы по осям координат

axis([1.5, 2.5, 0.5, 2])
чтобы получше разглядеть вершину синусоиды:



Чаще всего этот приём увеличения масштаба изображения применяют при графическом решении уравнений с тем, чтобы получить более высокую точность приближения к корню.

Теперь изменим количество числовых отметок на осях. Их может показаться недостаточно (на горизонтальной оси последнего рисунка их всего три - для значений 1.5 , 2 и 2.5).

Изменить отметки на осях координат можно с помощью функции set, обрабатывающей графический объект Axes. Это объект, который содержит оси координат и белый прямоугольник, внутри которого и проводится сам график функции.

Для получения описателя такого объекта применяют функцию gca, которую вызывают без параметров.

В итоге, следующий фрагмент кода

```
hAxes = gca;
set( hAxes, 'xtick', [ 1.5, 1.75, 2.0, 2.25, 2.5 ] )
```

выполняющийся после построения графика, устанавливает новые метки на горизонтальной оси координат (пять штук).

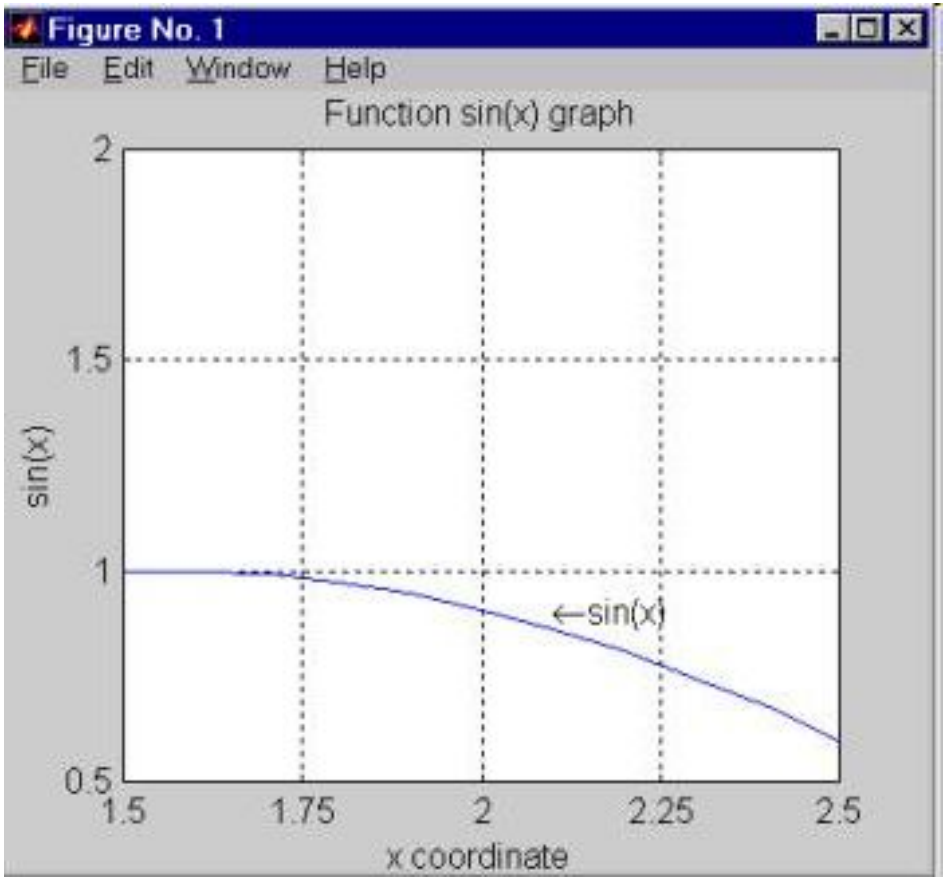
Для проставления различных надписей на полученном рисунке применяют функции xlabel, ylabel, title и text.

Функция xlabel предназначена для проставления названия горизонтальной оси, функция ylabel - то же для вертикальной оси (причём эти надписи ориентированы вдоль осей координат).

Если требуется разместить надпись в произвольном месте рисунка - применяем функцию text:
text(x, y, 'some text')

Общий заголовок для графика проставляется функцией title. Кроме того, используя команду
grid on
можно нанести измерительную сетку на всю область построения графика.

Применяя все эти средства
title('Function sin(x) graph');
xlabel('x coordinate'); ylabel('sin(x)');
text(2.1, 0.9, '\leftarrowsin(x)'); grid on;



Надпись функцией text помещается, начиная от точки с координатами, указанными первыми двумя аргументами. Специальные символы вводятся внутри текста после символа \ ("обратная косая черта").

В примере мы ввели таким образом специальный символ "стрелка влево".

3. Трёхмерная графика.

Графики функций двух переменных представляют из себя куски поверхностей, нависающие над областями определения функций.

Отсюда ясно, что изображение графиков функций двух переменных требует реализации "трёхмерной графики" на экране дисплея. Высокоуровневая графическая подсистема MATLABa автоматически реализует трёхмерную графику без специальных усилий со стороны пользователя.

Пусть в точке с координатами x1,y1 вычислено значение функции z=f(x,y) и оно равно z1. В некоторой другой точке (то есть при другом значении аргументов) x2,y2 вычисляют значение функции z2.

Продолжая этот процесс, получают массив (набор) точек (x1,y1,z1), (x2,y2,z2), … (xN,yN,zN) в количестве N штук, расположенных в трёхмерном пространстве.

Специальные функции системы MATLAB проводят через эти точки гладкие поверхности и отображают их проекции на плоский дисплей компьютера.

Чаще всего точки аргументов расположены в области определения функции регулярно в виде прямоугольной сетки (то есть матрицы).

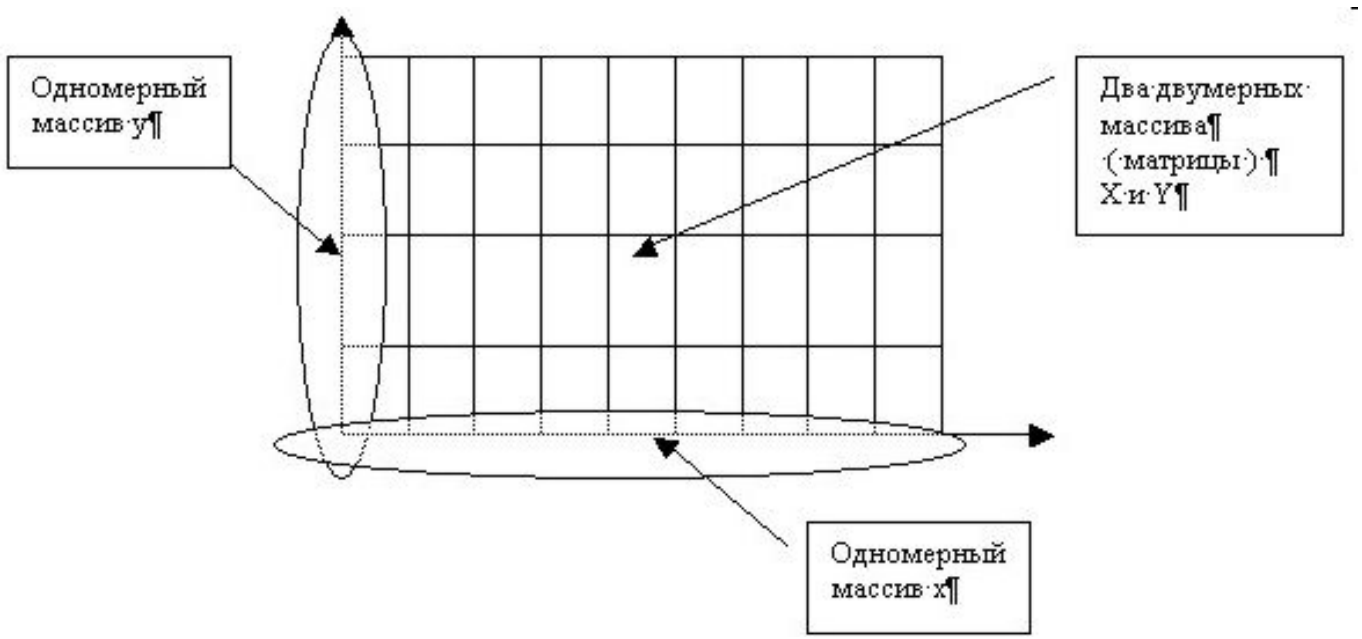
Такая сетка точек порождает две матрицы одной и той же структуры: первая матрица содержит значения первых координат этих точек (x - координат), а вторая матрица содержит значения вторых координат (y - координат).

Обозначим первую матрицу как X, а вторую - как Y. Есть ещё и третья матрица - матрица значений функции z=f(x,y)при этих аргументах. Эту матрицу обозначим буквой Z.

Простейшей функцией построения графика функции двух переменных в системе MATLAB является функция

plot3(X , Y , Z)
где X, Y и Z - матрицы одинаковых размеров, смысл которых мы только что объяснили.

В системе MATLAB имеется специальная функция для получения двумерных массивов X и Y по одномерным массивам x, y.



Пусть по оси x задан диапазон значений в виде вектора

u = -2 : 0.1 : 2

а по оси y этот диапазон есть

v = -1 : 0.1 : 1

Для получения матриц X и Y, представляющих первые и вторые координаты получающейся прямоугольной сетки точек используют специальную функцию системы MATLAB:

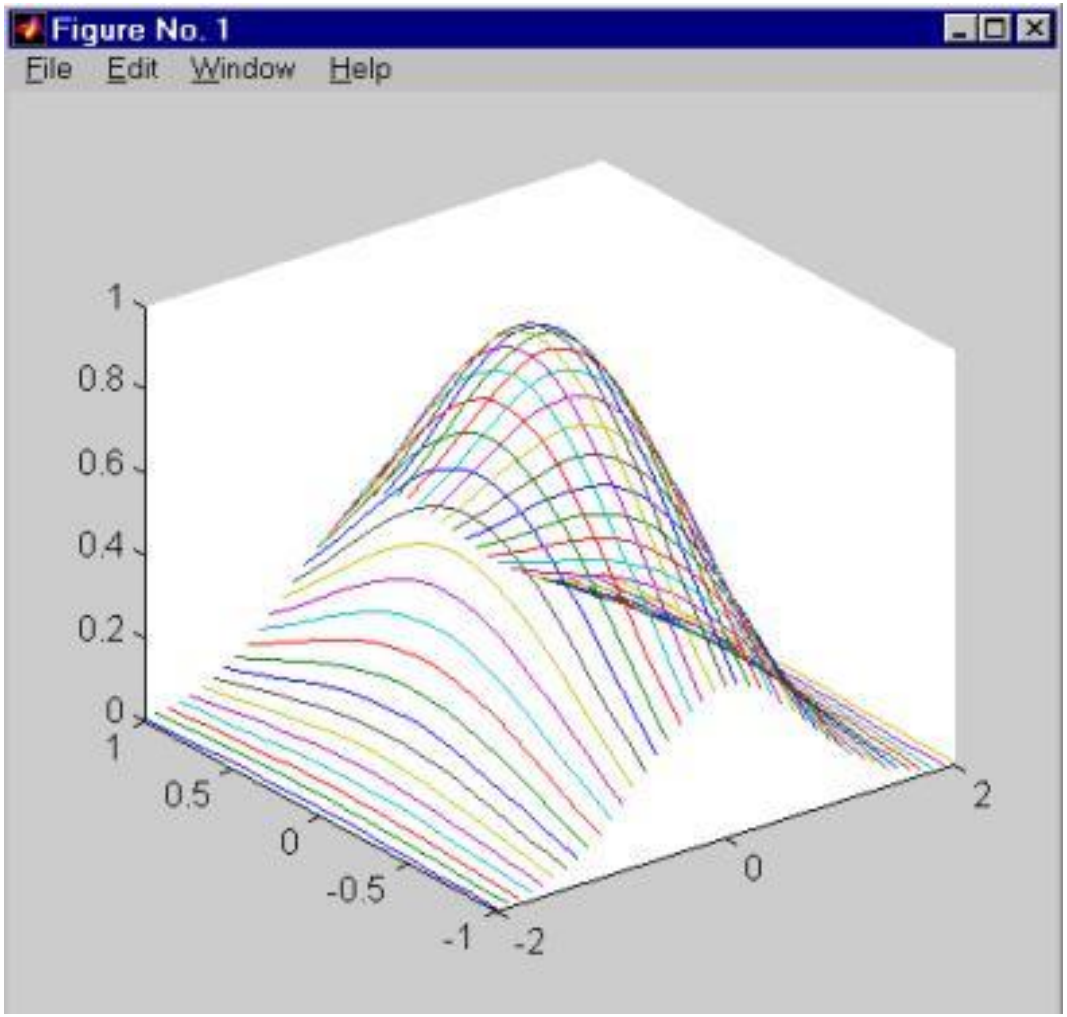
[X , Y] = meshgrid(u, v)

Как мы видим, эта функция получает на входе два одномерных массива (вектора), представляющие массивы точек на осях координат, и возвращает сразу два искомых двумерных массива.

На прямоугольной сетке точек вычисляем значения функции, например функции exp:

Z = exp(- X.^2 - Y.^2)

функцию plot3,получаем следующее изображение трёхмерного графика этой функции:



функция plot3 строит график в виде набора линий в пространстве, каждая из которых является сечением трёхмерной поверхности плоскостями, параллельными плоскости yOz. можно сказать, что каждая линия получается из отрезков прямых, соединяющих набор точек, координаты которых берутся из одинаковых столбцов матриц X, Y и Z. То есть, первая линия соответствует первым столбцам матриц X, Y Z; вторая линия - вторым столбцам этих матриц и так далее.

Для построения трёхмерных линий, задаваемых параметрически применяется другая форма вызова функции plot3:

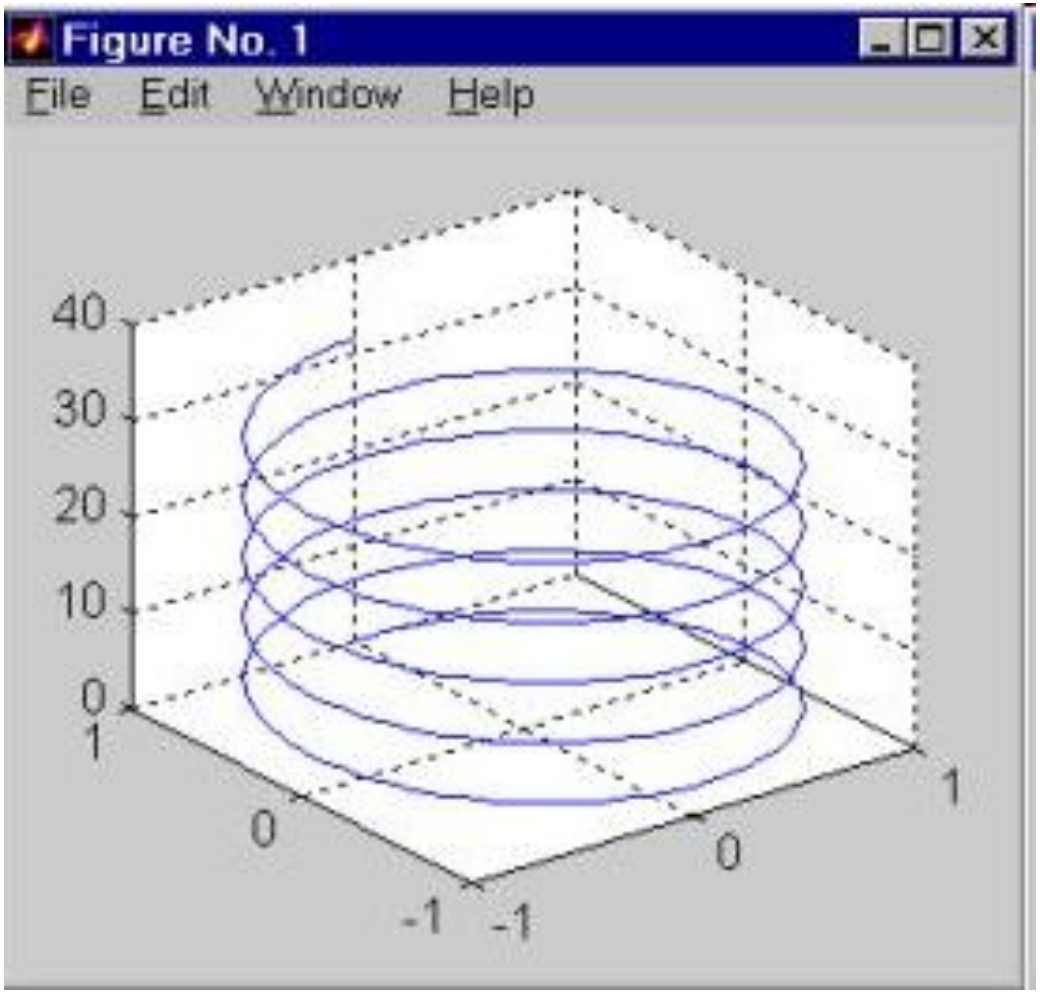
plot3(x, y, z)

где x, y и z являются одномерными массивами координат точек, которые и нужно последовательно соединить отрезками прямых.

Например, следующий фрагмент кода

```
t = 0 : pi/50 : 10*pi ;
 x = sin( t );
 y = cos( t );
 plot3( x , y , t );
 grid on
```

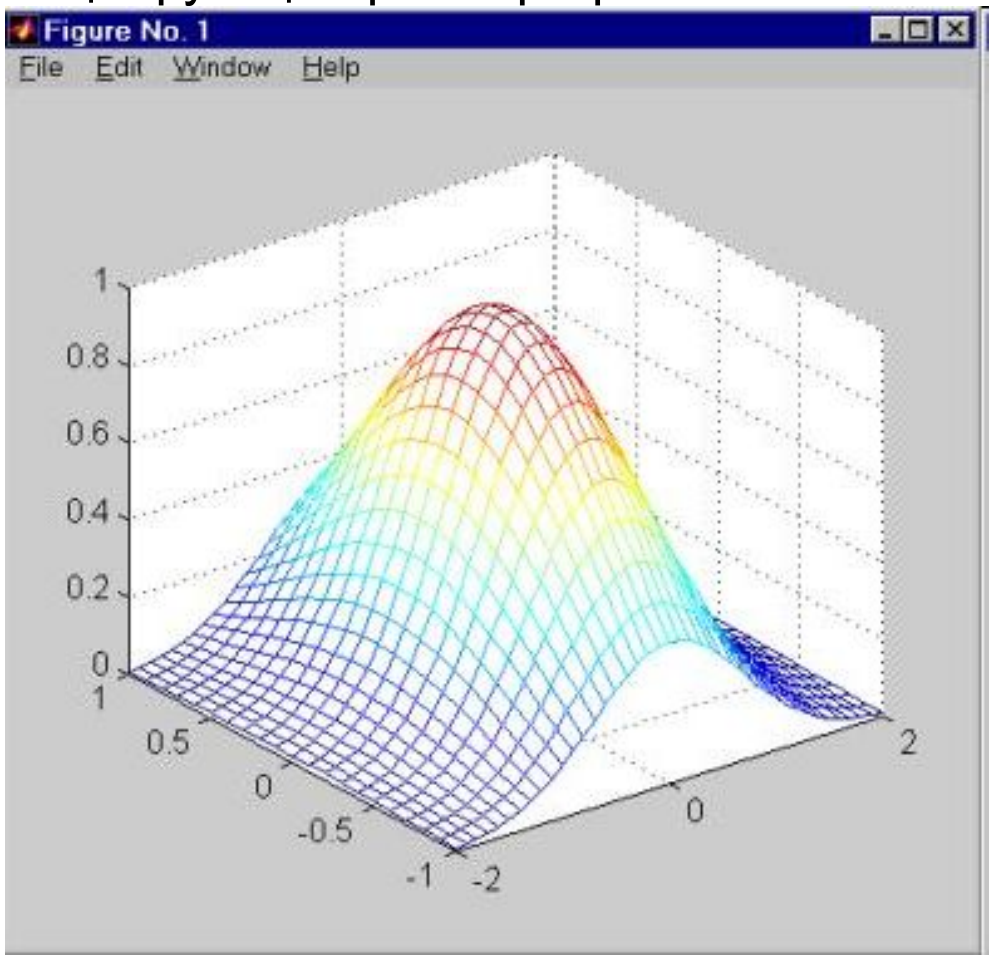
для проставления сетки координатных значений в области построения графика (также допустимо использовать команды и функции по оформлению графиков, ранее рассмотренные для "плоского" случая), позволяет построить винтовую линию, изображение которой показано на следующем рисунке:



Помимо этой простейшей функции система MATLAB располагает ещё рядом функций, позволяющих добиваться большей реалистичности в изображении трёхмерных графиков. Это функции mesh, surf и surfl.

Функция mesh соединяет вычисленные соседние точки поверхности графика отрезками прямых и показывает в графическом окне системы MATLAB плоскую проекцию такого объёмного "каркасно-ребристого" (по-английски зовётся wireframe mesh) тела.

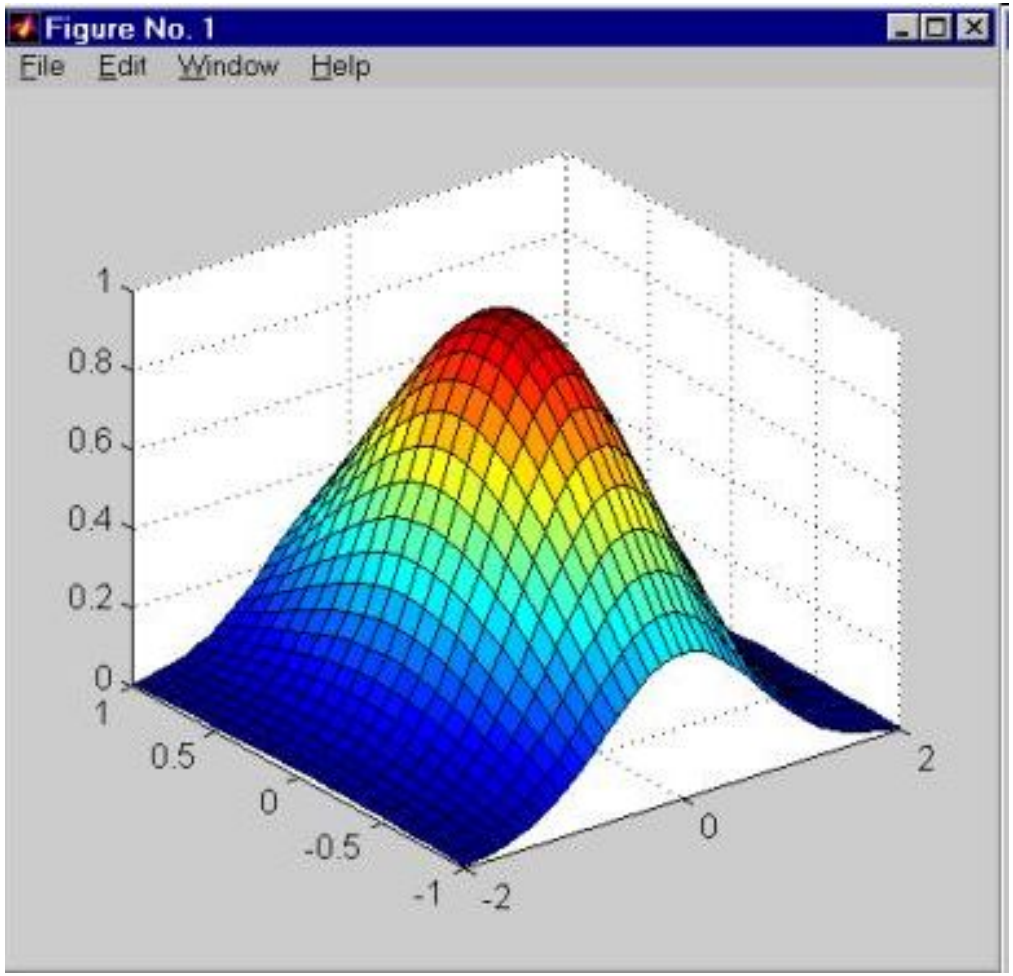
Вместо ранее показанного при помощи функции plot3 графика функции
exp(- X.^2 - Y.^2)



Для лучшего восприятия "объёмности" изображения разные рёбра автоматически окрашиваются в разные цвета. Кроме того (в отличие от функции plot3) осуществляется удаление невидимых линий.

Если вы считаете, что изображённое ребристое тело является прозрачным и не должно скрывать задних линий, то можно ввести команду hidden off , после чего такие линии появятся на изображении.

Более плотного изображения поверхности можно добиться, если вместо функции mesh применить функцию surf(X, Y, Z).



В результате получается следующее изображение представляющее плотную (непрозрачную) сетчатую поверхность,
причём отдельные ячейки (грани) этой сетчатой поверхности (плоские четырёхугольники) автоматически окрашиваются в разные цвета.

С помощью функции surf получаются хотя и искусственно раскрашенные, но весьма наглядные изображения.

Если же мы хотим добиться более естественных и объективных способов окрашивания поверхностей, то следует использовать функцию surfl.

Функция surfl трактует поверхность графика как материальную поверхность с определёнными физическими свойствами по отражению света.

По умолчанию задаётся некоторый источник света, освещающий такую материальную поверхность, после чего рассчитывается траектории отражённых лучей, попадающих в объектив условной камеры.

Изображение в такой камере и показывается в графическом окне системы MATLAB.

Так как разные материалы по-разному отражают падающие лучи, то можно подобрать некоторый материал, чтобы получить наилучшее (с точки зрения пользователя) изображение.

В частности, можно использовать функцию
colormap(copper)

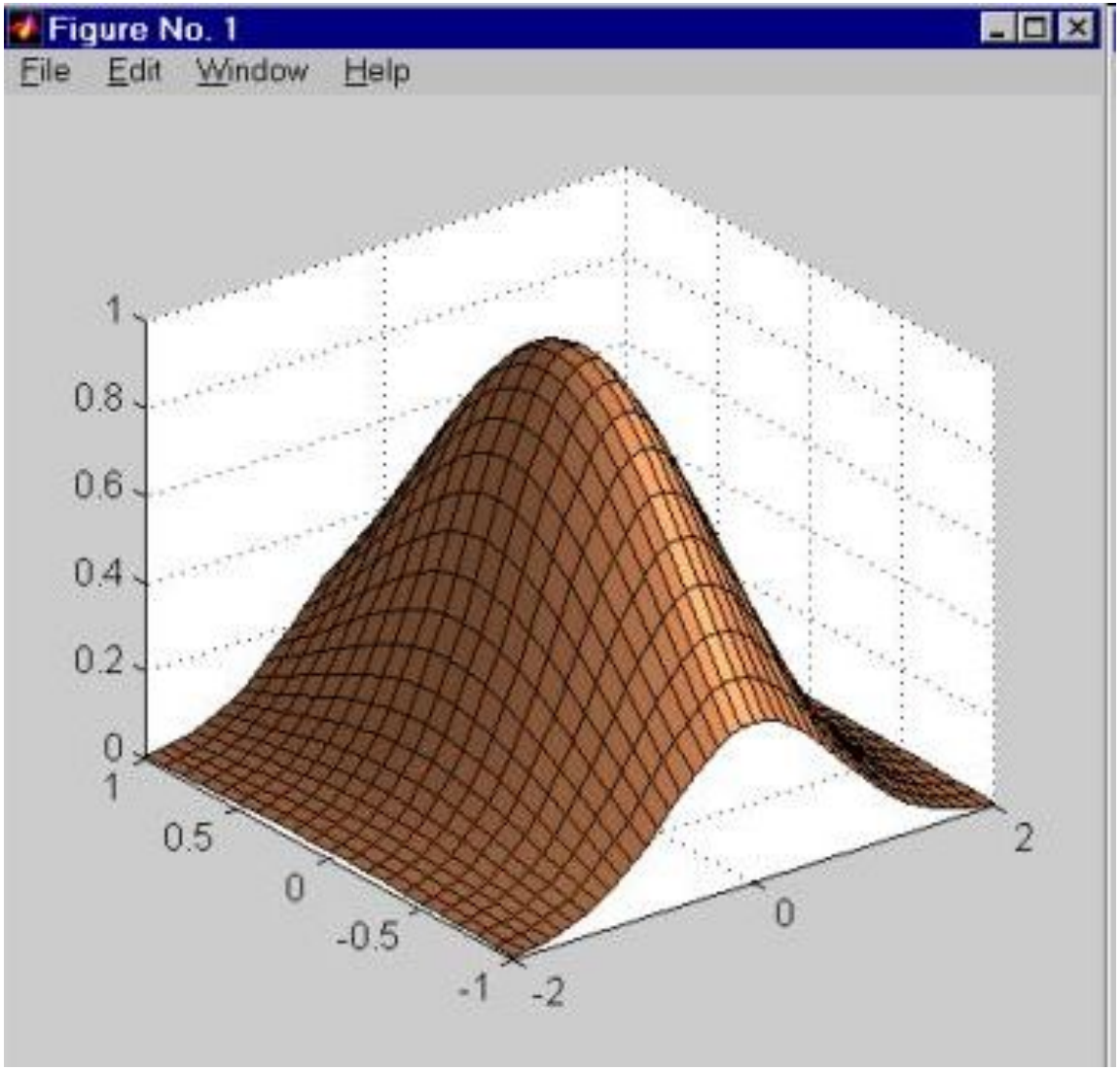
с помощью которой для изображения графика выбирается набор цветов (по-английски - colormap), который характерен для света, отражающегося от медной поверхности (медь по-английски - copper). После этого применение функции

surfl(X, Y, Z)
вместо surf(X,Y,Z) приводит к получению очень реалистически выглядящего и очень наглядного графика:

Можно с такого графика убрать чёрные линии, изображающие рёбра, а также добиться ещё более плавного перехода освещения поверхности, если выполнить команду

shading interp

означающую, что теперь цвет (освещённость) будет меняться даже внутри отдельных граней (ячеек).



В итоге будет получаться совсем уж реальное изображение некоторой объёмной фигуры. Лучше это или хуже для задачи изображения графиков функций двух переменных - судить конкретному пользователю.

4. Положение камеры и вращение трёхмерных графиков.

Многие приёмы оформления трёхмерных графиков совпадают с теми, что были рассмотрены при изучении плоских графиков функций одного переменного.

В частности, для масштабирования удобно использовать функцию axis, которая в трёхмерном случае принимает уже три пары скалярных аргументов:
axis([xmin, xmax, ymin, ymax, zmin, zmax])

По-прежнему можно использовать функции text, xlabel, ylabel, zlabel, title, а также можно наносить отметки на осях координат с помощью функции set.

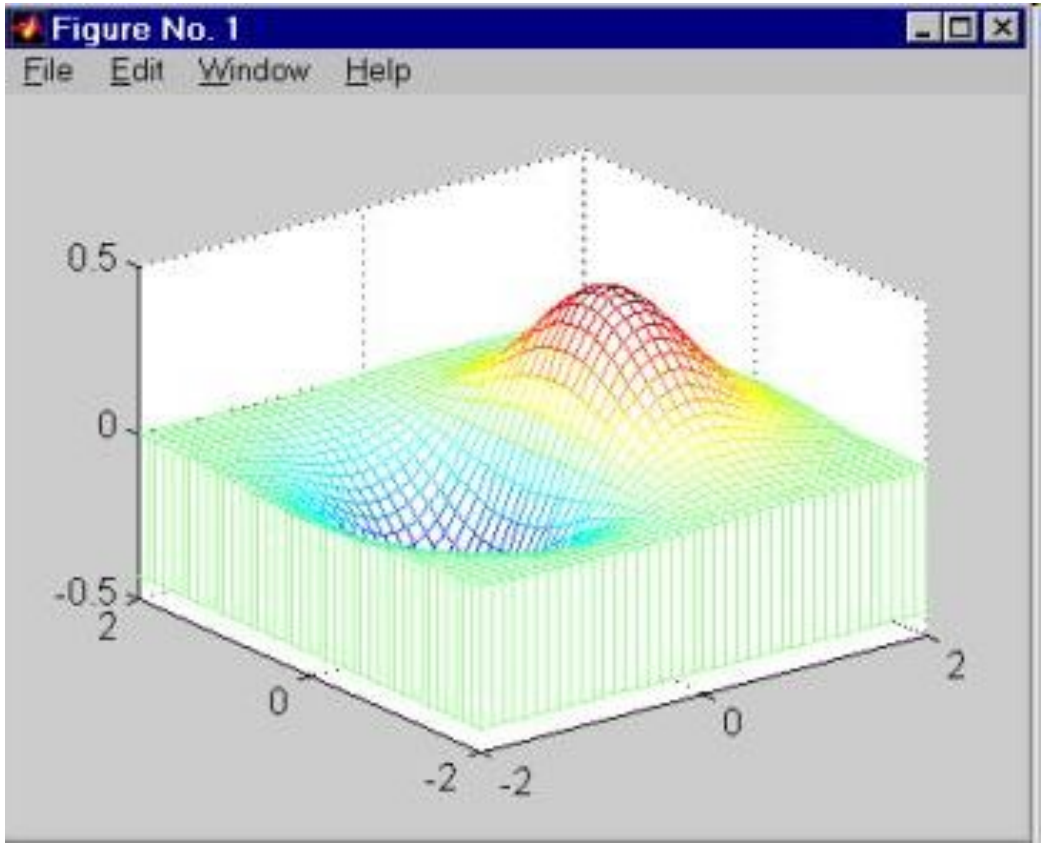
Можно также с помощью функции subplot разместить в одном графическом окне несколько трёхмерных графиков.

К новым методам дополнительного оформления трёхмерных графиков можно отнести возможность вызывать функцию mesh с суффиксами z и c (meshz и meshc), а функцию surf с суффиксом c (surfc).

Использование суффикса z приводит к построению "графика с основанием". Например, фрагмент кода

```
[X,Y] = meshgrid( -2 : 0.1 : 2 );
Z = X .* exp( - X.^2 - Y.^2 );
meshz( X, Y, Z )
```

строит следующий график:

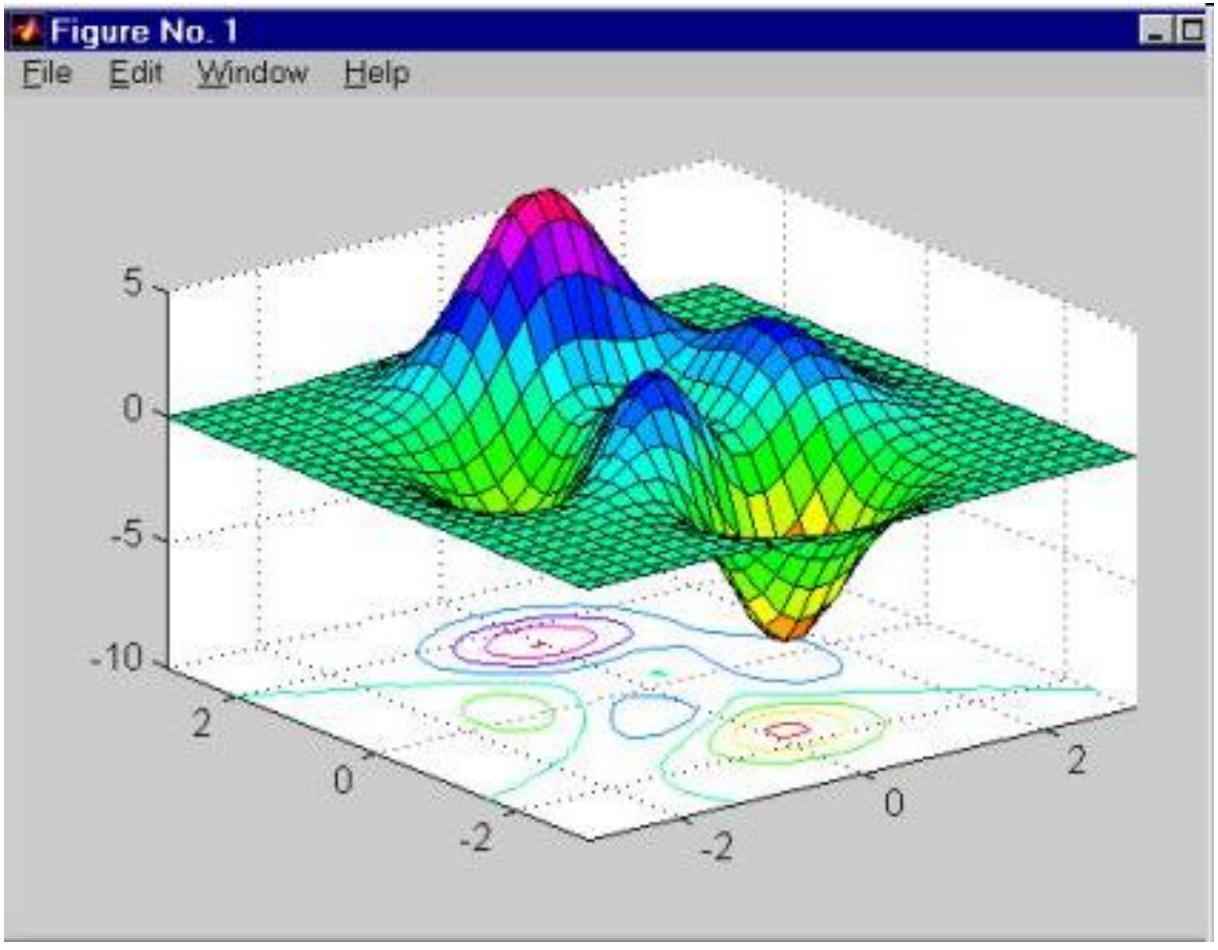


Функции с суффиксом с помимо собственно трёхмерного графика строят ещё и так называемые линии уровня.

Например, следующий фрагмент

```
[X,Y,Z] = peaks(30); surfc(X,Y,Z);
 colormap( hsv ); axis([-3 3 -3 3 -10 5]);
```

приводит к изображению:



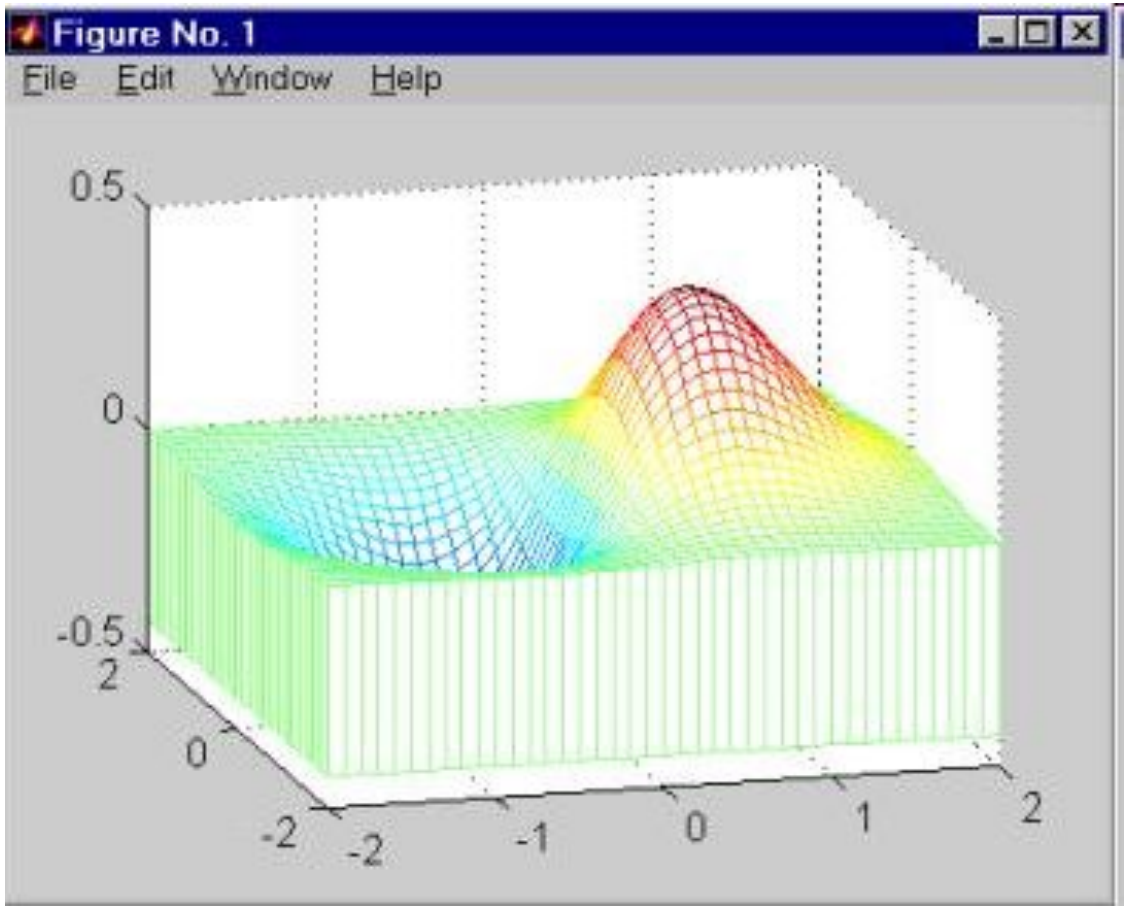
Функция с именем peaks (является некоторой масштабированной комбинацией стандартных гауссовых функций) часто применяется в справочной системе MATLABa для наглядной иллюстрации графических функций.

Наконец, для трёхмерных графиков существует возможность изменять точку обзора графика, то есть положение условной камеры. Положение камеры опередеяется углом азимута (часто обозначают az) и углом возвышения (часто обозначают el).

Изменение первого угла означает вращение плоскости xOy вокруг оси Oz против часовой стрелки. Угол возвышения есть угол между направлением на камеру и плоскостью xOy.

Когда выполняются функции mesh или surf, то по умолчанию устанавливаются значения az = -37.5°, el = 30°. Эти значения в любой момент времени можно изменить функцией
view([az , el])

В частности, если после построения показанного выше графика функции
X .* exp(- X.^2 - Y.^2)
выполнить команду
view([-15 , 20])
то график изменит свой вид:



Комбинируя вызов различных функций системы MATLAB и выбирая различные варианты закраски и угла просмотра, можно добиться оптимального вида трёхмерных графиков

5. Сохранение в файлах и передача в другие программы графических изображений MATLABa.

Получив удачное изображение мы можем захотеть сохранить его для будущих просмотров в системе MATLAB, а также, возможно, и в других приложениях.
Самым простым способом сохранения графического изображения является использование команды меню Edit | Copy Figure, в результате чего изображение (это клиентская часть графического окна MATLABa) будет сохранено в буфере обмена операционной системы Windows (Clipboard).

После этого вы можете вставить это изображение в документ редактора Word командой меню Paste последнего. Вместе со всем документом это изображение, полученное первоначально в системе MATLAB, можно будет распечатать на принтере.
Вместо использования команды меню графического окна можно из командного окна системы MATLAB выполнить команды

print -dbitmap или **print -dmeta**
причём вторая команда сохранит изображение в буфере обмена в формате Windows Metafile вместо формата Bitmap.

Аналогично созданное в системе MATLAB изображение можно будет командой Paste вставить в редактируемое изображение простого графического редактора Paint, входящего в поставку операционной системы Windows.

В результате вы можете смонтировать изображение, полученное в системе MATLAB с другими изображениями программы Paint.

Часто, бывает удобно сохранить полученное в MATLABe изображение в файле некоторого известного графического формата. Это легко сделать командами
print -options FileName
где вместо options надо подставить заданный идентификатор для конкретной ситуации.

Например, если мы хотим создать графический файл в формате популярного пакета векторной графики Illustrator, то вместо options надо будет подставить dill :
print -dill FileName

В результате выполнения этой команды на диске будет записан файл FileName.ai, где расширение ai характерно для пакета Illustrator. Далее этот файл можно открыть в пакете Illustrator и осуществлять его дальнейшее редактирование уже в рамках этого мощного пакета векторной графики.

Много других популярных графических форматов файлов можно получить, применяя команду capture и функцию imwrite. Например, следующий код

```
 [X,map]=capture(1);
imwrite(X,map,'myfile1.jpg')
```

Создаёт файл myfile1.jpg, который хорошо включать в Internet-страницы для их просмотра браузером Internet Explorer.
Функция capture возвращает матрицу X, соответствующую точкам изображения, и матрицу цветов map (три столбца в формате RGB), использованную в изображении. Каждый элемент матрицы X равен номеру одной из строк матрицы map.
В системе MATLAB по матрицам X и map можно восстановить графическое изображение, применив команды

```
colormap( map );
 image( X );
```

6. Показ произвольных растровых изображений.

Произвольное изображение на экране компьютера представляет собой массив пикселов, каждый из которых характеризуется своим цветом. Цвет пиксела определяется тремя составляющими: красным, зелёным и синим (Red, Green, Blue - RGB). Для задания величины составляющей цвета пиксела достаточно одного байта памяти (8 битов), где можно записать целые числа от нуля до 255 (всего 256 значений).

Каждому пикселу экрана должны соответствовать три целых числа в диапазоне от 0 до 255. В системе MATLAB таким целым числам соответствует тип данных, обозначаемый как uint8. Под такой тип данных отводится в памяти всего один байт, вместо 8 байт для обычных вещественных (дробных) чисел типа double. По умолчанию любой переменной в системе MATLAB ставится в соответствие тип double независимо от числовых значений, которые вы присваиваете переменным.
Например, в результате следующей строки кода
iVar1 = 128
создаётся переменная с именем iVar1 и типом double, которой присваивается значение 128. Для хранения такого значения достаточно одного байта памяти, однако для переменной iVar1 типа double отводится 8 байт памяти.

Чтобы избежать такого перерасхода переменную нужно явно объявлять как целую, используя модификатор uint8:

iVar2 = uint8(128);

Так созданная переменная iVar2 считается целой переменной (а не вещественной), и под неё отводится один байт памяти. Такие переменные в системе MATLAB специально предназначены для хранения целых значений от 0 до 255 (с целью экономии памяти) и не предназначены для вычислений!

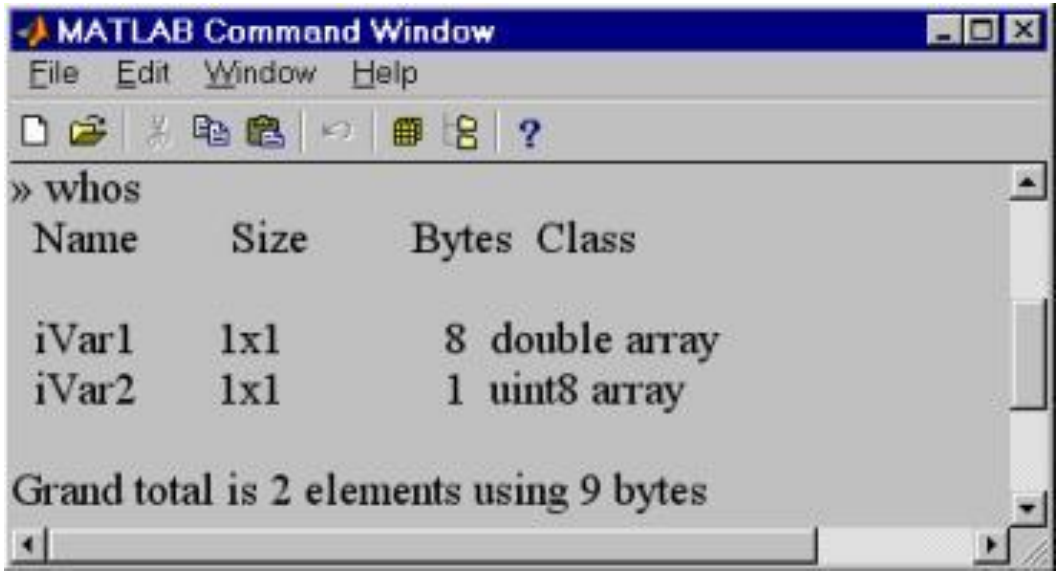
По-крайней мере в версии MATLAB 5.2 это ещё так. В результате для следующего фрагмента
iVar2 = iVar2 + 1;
получаем сообщение об ошибке:
??? Function '+' not defined for variables of class 'uint8'.

Дословно означающее, что операция "сложение" для переменных типа uint8 не определена.

Чтобы узнать (если забыли), какой тип имеет та или иная переменная из рабочего пространства системы MATLAB, нужно ввести и выполнить команду

whos

в результате в командном окне MATLABa появится следующее сообщение:



из которого видно, что iVar1 является массивом размера 1x1 (то есть фактически скаляром) типа double и занимает в памяти 8 байт, а iVar2 имеет тип uint8 и занимает в памяти только 1 байт (в 8 раз меньше).

При этом обе переменные имеют одинаковые значения.

Некоторый набор цветов (в количестве m штук), называемый палитрой или colormap, можно оформить в виде матрицы размером m x 3 типа double.

Например, матрица map1

map1(1,1) = 0.12; map1(1,2) = 0.123; map1(1,3) = 0.987;
map1(2,1) = 0.456; map1(2,2) = 0.7; map1(2,3) = 0.22;
map1(3,1) = 0.88; map1(3,2) = 0.19; map1(3,3) = 0.611;
map1(4,1) = 0.255; map1(4,2) = 0.298 ; map1(4,3) = 0.128;
map1(5,1) = 0.01; map1(5,2) = 0.78; map1(5,3) = 0.60;

задаёт набор из пяти цветов. Каждая строка соответствует одному цвету. Элементы строки (слева - направо) задают красную, зелёную и синию составляющие цвета.

Далее сформируем матрицу k x L типа uint8, каждый элемент которой будет равен одному из номеров (минус единица) строк таблицы цветов map1.

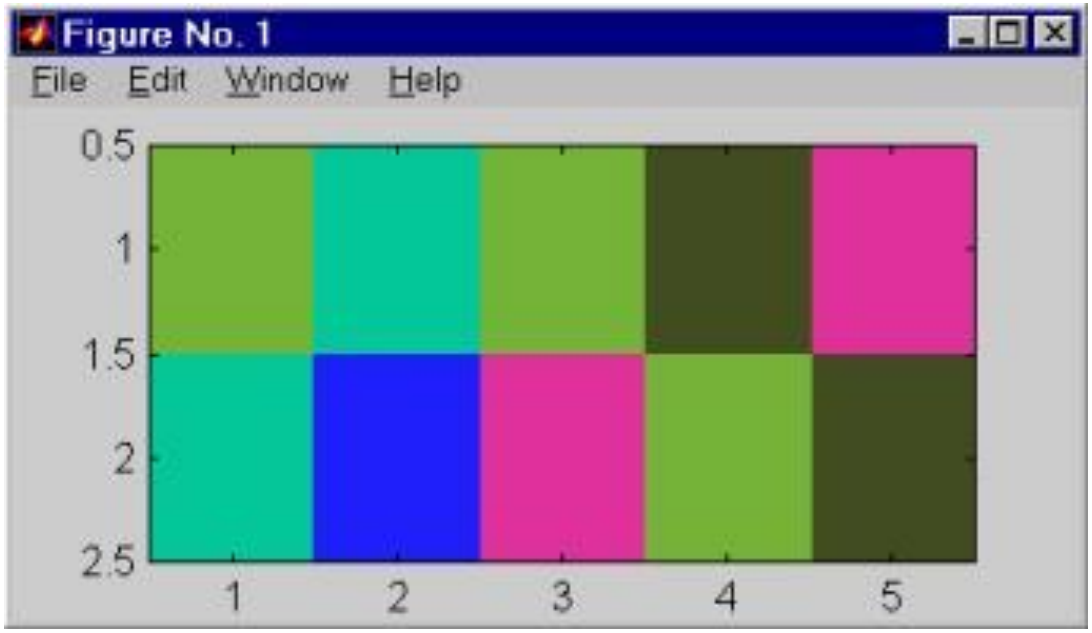
Такой матрицы вместе с матрицей цветов будет достаточно, чтобы показать на экране компьютера массив пикселов, то есть произвольное изображение.

Например, матрица X1
X1=uint8([1 4 1 3 2; 4 0 2 1 3])
задаёт массив типа uint8 размером 2 x 5 пиксел. Этот массив всего занимает в памяти 10 байт, а не 80 байт, как было бы в случае массива типа double.

Первый пиксел в первом ряду имеет цвет, задаваемый второй строкой матрицы map1, второй пиксел в этом же ряду соответствует пятой строке матрицы map1, и так далее.

Чтобы заставить систему MATLAB реально отобразить произвольную картину пикселов, нужно вызвать функцию image.

Например, с помощью функций
image(X1); colormap(map1);
создаётся графический объект Image системы MATLAB, которому в графическом окне MATLABa соответствует реальное изображение:



Поскольку мы не управляем размером графического окна системы MATLAB, то оно появляется на экране с некоторым разумным размером, заданным по умолчанию.

Далее, так как наше изображение состоит из двух рядов по пять пиксел в каждом, а это очень мелкое изображение (физический размер пиксела экрана примерно равен 0.2 мм), то MATLAB по умолчанию масштабирует его (увеличивает), чтобы можно было разглядеть это изображение.

Если требуется отменить такое масштабирование, то следует указать явно нужные размеры:

```
[ m , n ] = size( X1 );
figure( 'Units', 'pixels', 'Position', [100 100 n m] );
 image( X1 ); colormap( map1 );
```

Здесь размеры m и n изображения X1 навязываются в качестве физического размера картинки в графическом окне системы MATLAB. Для слишком маленьких картинок при этом ничего хорошего не получится.

Если мы не будем создавать новые изображения, манипулируя явно матрицами, а будем пытаться отобразить в графическом окне системы MATLAB уже готовые картинки, записанные в файлах, то тогда нам потребуется прочесть содержимое этих файлов функцией imread.

В частности, ранее мы записывали трёхмерные изображения в файлы с помощью функции imwrite. Теперь их можно прочесть

[X2, map2] = imread('myfile1.jpg')

и показать в графическом окне. Заметим только, что файл должен быть в текущем каталоге системы MATLAB, иначе его нужно указать вместе с полным путём к нему.

Рассмотренное нами строение данных для объекта Image, состоящее из двух матриц, одна из которых построчно задаёт цвета, а вторая своими элементами указывает входы в таблицу (матрицу) цветов, называется более точно как Indexed Image (индексированное изображение).

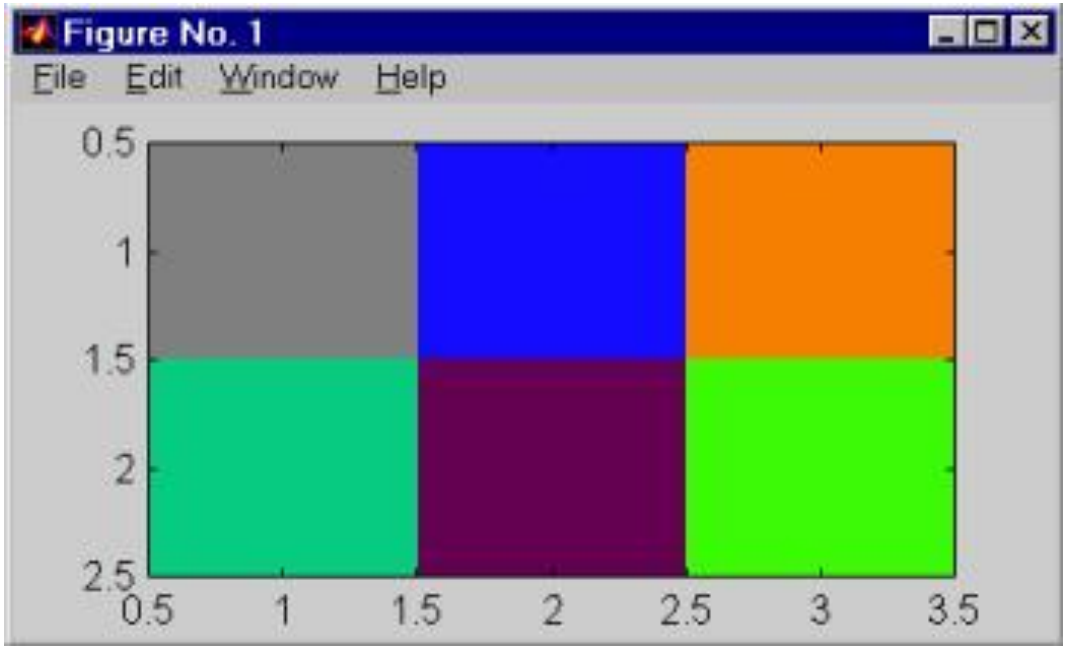
Есть и другой тип объекта Image - так называемый Truecolor Image (картинки с очень большим количеством цветов - до 16 миллионов). Этот второй тип объектов Image устроен по-другому.

Для TruecolorImage- объектов таблица цветов не требуется, так как массивы данных таких объектов непосредственно определяют цвета. Эти массивы имеют размер m x n x 3 (это массивы размерности 3). Величины m и n определяют размер картинки на экране (m x n пикселов), а вдоль третьего направления располагаются RGB-составляющие цвета каждого пиксела.

Зададим для примера следующий массив для изображения TrueColor:

xTrue(1,1,1) = uint8(127);xTrue(1,1,2) = uint8(127); xTrue(1,1,3) = uint8(127);
xTrue(1,2,1) = uint8(19); xTrue(1,2,2) = uint8(12);xTrue(1,2,3) = uint8(255);
xTrue(1,3,1) = uint8(245);xTrue(1,3,2) = uint8(127);xTrue(1,3,3) = uint8(1);
xTrue(2,1,1) = uint8(6); xTrue(2,1,2) = uint8(203);xTrue(2,1,3) = uint8(128);
xTrue(2,2,1) = uint8(100);xTrue(2,2,2) = uint8(1);xTrue(2,2,3) = uint8(80);
xTrue(2,3,1) = uint8(60);xTrue(2,3,2) = uint8(249);xTrue(2,3,3) = uint8(5);

Массив xTrue создаёт изображение 2 x 3 пиксела с помощью вызова одной функции image(xTrue):



Если изображение находится в файле и вы заранее не знаете какой оно имеет тип (индексное, то есть с палитрой цветов, или имеет тип TrueColor), то можете читать его следующим образом:

[X, map] = imread('name.xxx')

В случае TrueColor изображений здесь матрица X получит размер m x n x 3 а матрица палитры map будет пустой:

size(map) = 0 0

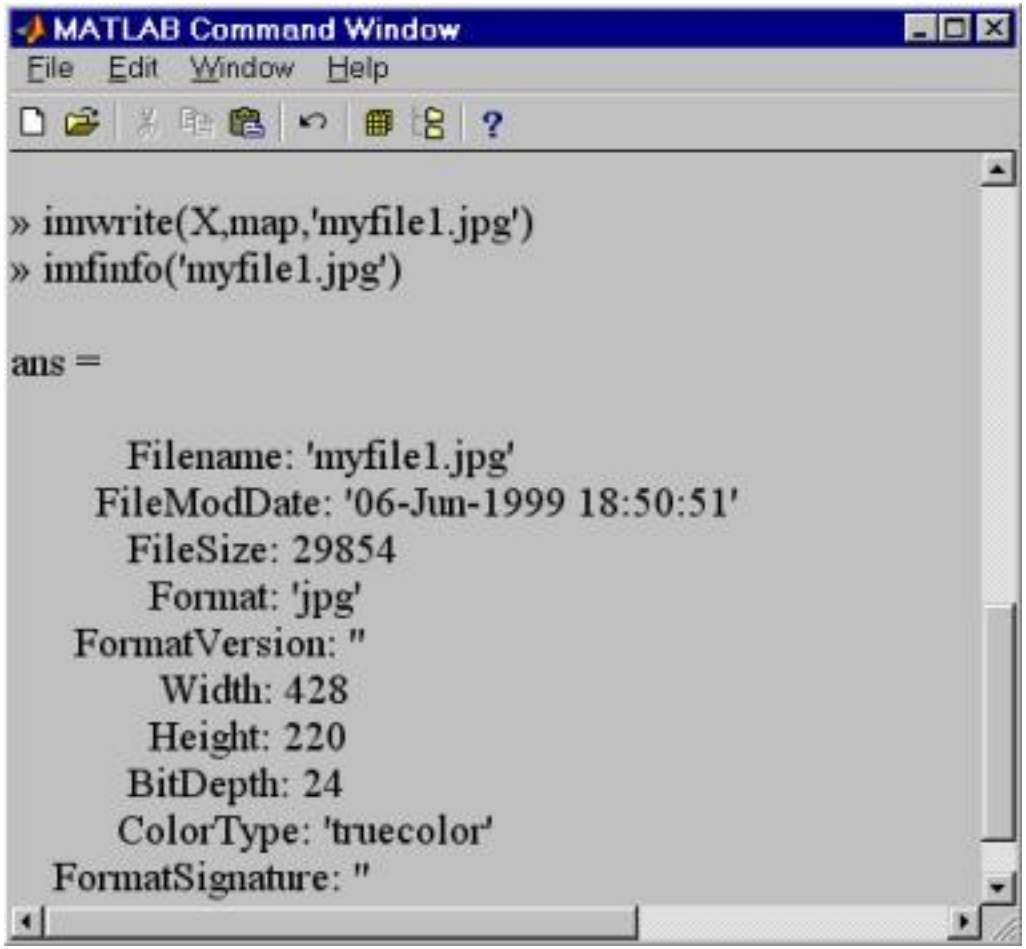
В дальнейшем функция image автоматически по размеру матрицы X распознаёт типы изображений и действует в обоих случаях как надо, а функция colormap в случае пустого массива map не делает ничего, так что оба этих случая могут быть обработаны одинаково.
Однако, если бы мы заранее знали, что в файле содержится изображение типа TrueColor, то мы бы выполняли для его чтения более короткий код

X = imread('name.xxx')

а показывали бы изображение вызовом только одной функции image(X). Чтобы заранее узнать тип изображения в файле, нужно вызвать функцию

imfinfo('name.xxx')

В частности для файла 'myfile1.jpg', созданного в предыдущем разделе данного пособия, функция imfinfo выдаст следующую информацию:



Отсюда видно, что тип изображения в файле (ColorType) есть truecolor. Это означает, что можно полностью обойтись без матрицы цветов

3. Интегрирование MatLab и Excel

Интегрирование MatLab и Excel позволяет пользователю Excel обращаться к многочисленным функциям MatLab для обработки данных, различных вычислений и визуализации результата.

Надстройка excllink.xla реализует данное расширение возможностей Excel. Для связи MatLab и Excel определены специальные функции.

3.1. Конфигурирование Excel

Перед тем как настраивать Excel на совместную работу с MatLab, следует убедиться, что Excel Link входит в установленную версию MatLab. В подкаталоге exclink основного каталога MatLab или подкаталога toolbox должен находиться файл с надстройкой excllink.xla.

Запустите Excel и в меню **Tools** выберите пункт **Add-ins**. Откроется диалоговое окно, содержащее информацию о доступных в данный момент надстройках.

Используя кнопку **Browse**, укажите путь к файлу excllink.xla. В списке надстроек диалогового окна появтится строка **Excel Link 2.0 for use with MatLab** с установленным флагом. Нажмите **OK**, требуемая надстройка добавлена в Excel.

Обратите внимание, что в Excel теперь присутствует панель инструментов **Excel Link**, содержащая три кнопки: **putmatrix**, **getmatrix**, **evalstring**.

Эти кнопки реализуют основные действия, требуемые для осуществления взаимосвязи между Excel и MatLab — обмен матричными данными, и выполнение команд MatLab из среды Excel. При повторных запусках Excel надстройка excllink.xla подключается автоматически.

Согласованная работа Excel и MatLab требует еще нескольких установок, которые приняты в Excel по умолчанию (но могут быть изменены).

В меню **Tools** перейдите к пункту **Options**, открывается диалоговое окно **Options**. Выберите вкладку **General** и убедитесь, что флаг **R1C1 reference style** выключен, т.е. ячейки нумеруются A1, A2 и т.д.

На вкладке **Edit** должен быть установлен флаг **Move selection after Enter**.

3.2. Обмен данными между MatLab и Excel

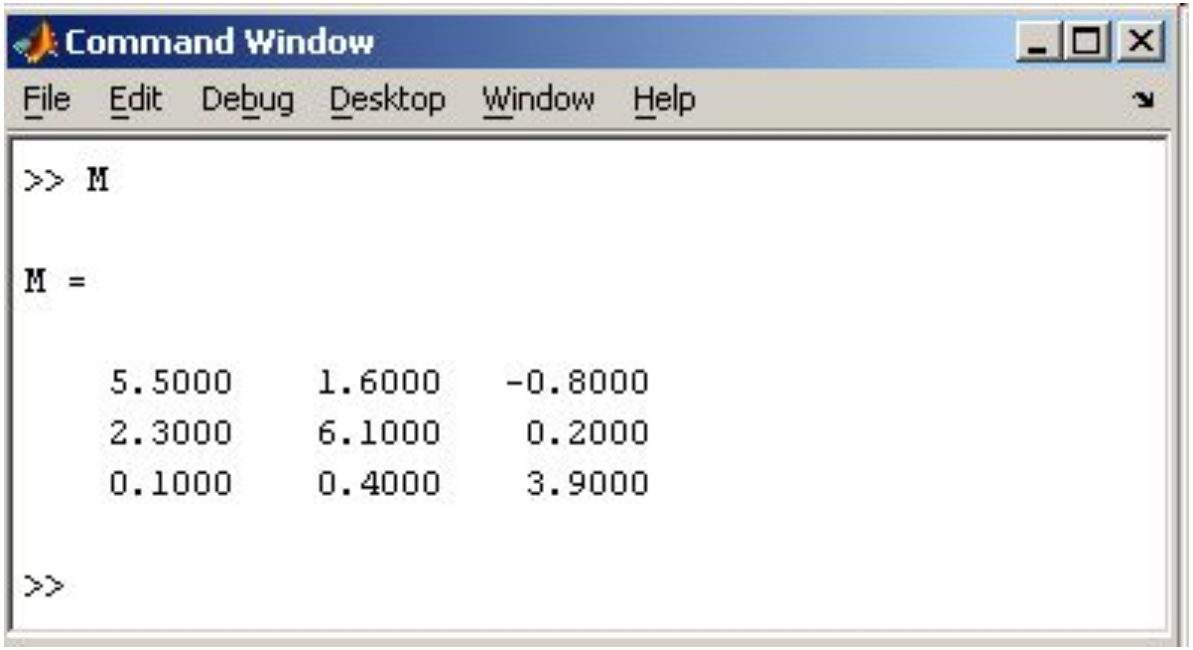
Запустите Excel, проверьте, что проделаны все необходимые настройки так, как описано в предыдущем разделе (MatLab должен быть закрыт).

Введите в ячейки с A1 по C3 матрицу, для отделения десятичных знаков используйте точку в соответствии с требованиями Excel.

	A	B	C
1	5.5	1.6	-0.8
2	2.3	6.1	0.2
3	0.1	0.4	3.9

Выделите на листе данные ячейки и нажмите кнопку **putmatrix**, появляется окно Excel с предупреждением о том, что MatLab не запущен. Нажмите **OK**, дождитесь открытия MatLab.

Появляется диалоговое окно Excel со строкой ввода, предназначенной для определения имени переменной рабочей среды MatLab, в которую следует экспортировать данные из выделенных ячеек Excel. Введите к примеру, M и закройте окно при помощи кнопки **OK**. Перейдите к командному окну MatLab и убедитесь, что в рабочей среде создалась переменная M, содержащая массив три на три:



Проделайте некоторые операции в MatLab с матрицей M, например, обратите ее.

Вызов inv для обращения матрицы, как и любой другой команды MatLab можно осуществить прямо из Excel.

Нажатие на кнопку **evalstring**, расположенную на панели **Excel Link**, приводит к появлению диалогового окна, в строке ввода которого следует набрать команду MatLab

IM=inv(M).

Результат аналогичен полученному при выполнении команды в среде MatLab.

Вернитесь в Excel, сделайте текущей ячейку A5 и нажмите кнопку **getmatrix**.

Появляется диалоговое окно со строкой ввода, в которой требуется ввести имя переменной, импортируемой в Excel.

В данном случае такой переменной является IM. Нажмите OK, в ячейки с A5 по A7 введены элементы обратной матрицы.

Итак, для экспорта матрицы в MatLab следует выделить подходящие ячейки листа Excel, а для импорта достаточно указать одну ячейку, которая будет являться верхним левым элементом импортируемого массива.

Остальные элементы запишутся в ячейки листа согласно размерам массива, переписывая содержащиеся в них данные, поэтому следует соблюдать осторожность при импорте массивов.

Вышеописанный подход является самым простым способом обмена информацией между приложениями — исходные данные содержатся в Excel, затем экспортируются в MatLab, обрабатываются там некоторым образом и результат импортируется в Excel.

Пользователь переносит данные при помощи кнопок панели инструментов **Excel Link**.

Информация может быть представлена в виде матрицы, т.е. прямоугольной области рабочего листа. Ячейки, расположенные в строку или столбец, экспортируются, соответственно, в вектор-строки и вектор-столбцы MatLab.

Аналогично происходит и импорт вектор-строк и вектор-столбцов в Excel.

4.5 Печать графиков

Пункт **Print** в меню **File** и команда print печатают графику MatLab.

Меню **Print** вызывает диалоговое окно, которое позволяет выбирать общие стандартные варианты печати.

Команда print обеспечивает большую гибкость при выводе выходных данных и позволяет контролировать печать из M-файлов.

Результат может быть послан прямо на принтер, выбранный по умолчанию, или сохранен в заданном файле.

5. Программирование

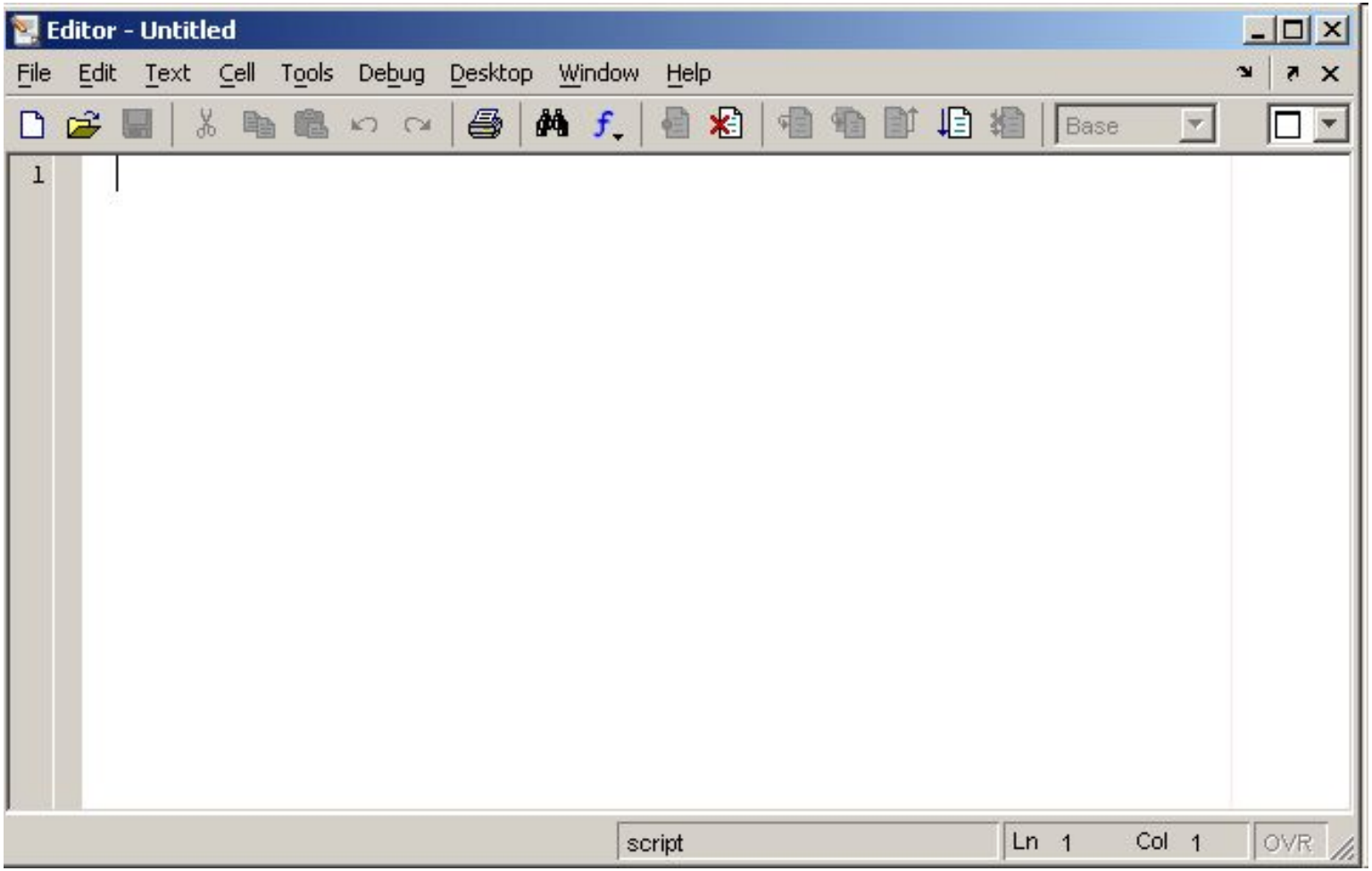
М-файлы

Работа из командной строки MatLab затрудняется, если требуется вводить много команд и часто их изменять. Ведение дневника при помощи команды diary и сохранение рабочей среды незначительно облегчают работу.

Самым удобным способом выполнения групп команд MatLab является использование М-файлов, в которых можно набирать команды, выполнять их все сразу или частями, сохранять в файле и использовать в дальнейшем.

Для работы с М-файлами предназначен редактор М-файлов. С его помощью можно создавать собственные функции и вызывать их, в том числе и из командного окна.

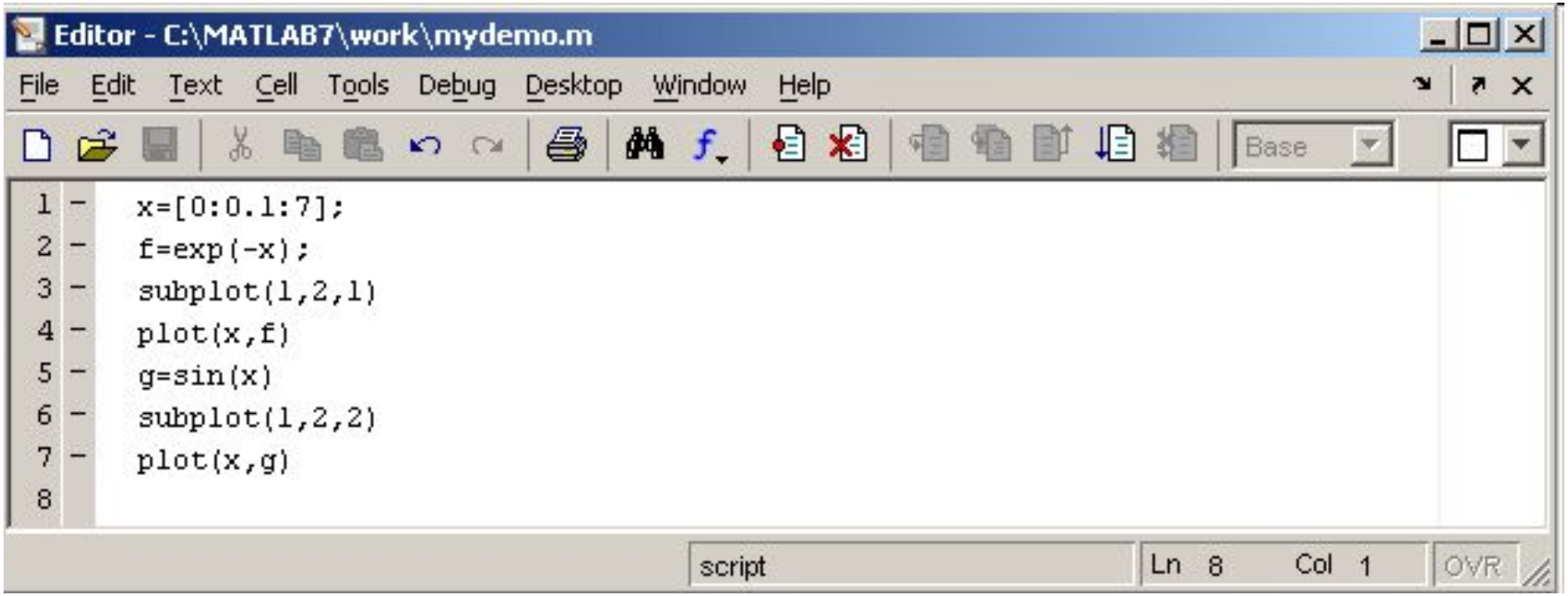
Раскройте меню **File** основного окна MatLab и в пункте **New** выберите подпункт **M-file**. Новый файл открывается в окне редактора М-файлов, которое изображено на рисунке.



М-файлы в MatLab бывают двух типов: файл-программы (*Script M-Files*), содержащие последовательность команд, и файл-функции, (*Function M-Files*), в которых описываются функции, определяемые пользователем.

Файл-программа

Наберите в редакторе команды, приводящие к построению двух графиков на одном графическом окне



Сохраните теперь файл с именем mydemo.m в подкаталоге work основного каталога MatLab, выбрав пункт **Save as** меню **File** редактора.

Для запуска на выполнение всех команд, содержащихся в файле, следует выбрать пункт **Run** в меню **Debug**.

На экране появится графическое окно *Figure 1*, содержащее графики функций.

Команды файл-программы осуществляют вывод в командное окно. Для подавления вывода следует завершать команды точкой с запятой.

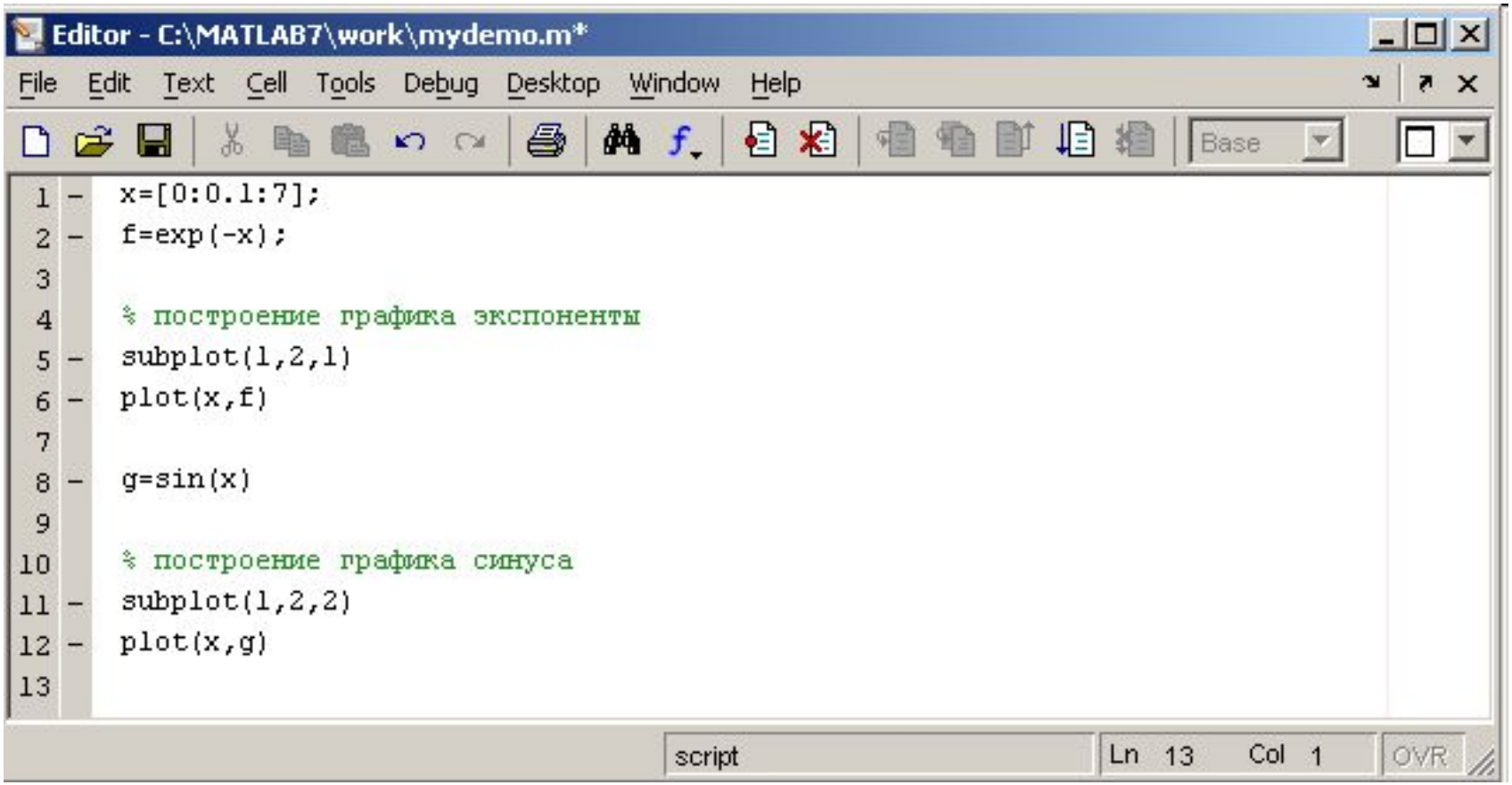
Если при наборе сделана ошибка и MatLab не может распознать команду, то происходит выполнение команд до неправильно введенной, после чего выводится сообщение об ошибки в командное окно.

Очень удобной возможностью, предоставляемой редактором М-файлов, является выполнение части команд. Закройте графическое окно *Figure 1*.

Выделите при помощи мыши, удерживая левую кнопку, или клавишами со стрелками при нажатой клавише **Shift**, первые четыре команды и выполните их из пункта **Text**.

Обратите внимание, что в графическое окно вывелся только один график, соответствующий выполненным: командам. Запомните, что для выполнения части команд их следует выделить и нажать клавишу **F9**.

Отдельные блоки М-файла можно снабжать комментариями, которые пропускаются при выполнении, но удобны при работе с М-файлом. Комментарии начинаются со знака процента и автоматически выделяются зеленым цветом, например:



Открытие существующего М-файла производится при помощи пункта **Open** меню **File** рабочей среды, либо редактора М-файлов.

Файл-функция

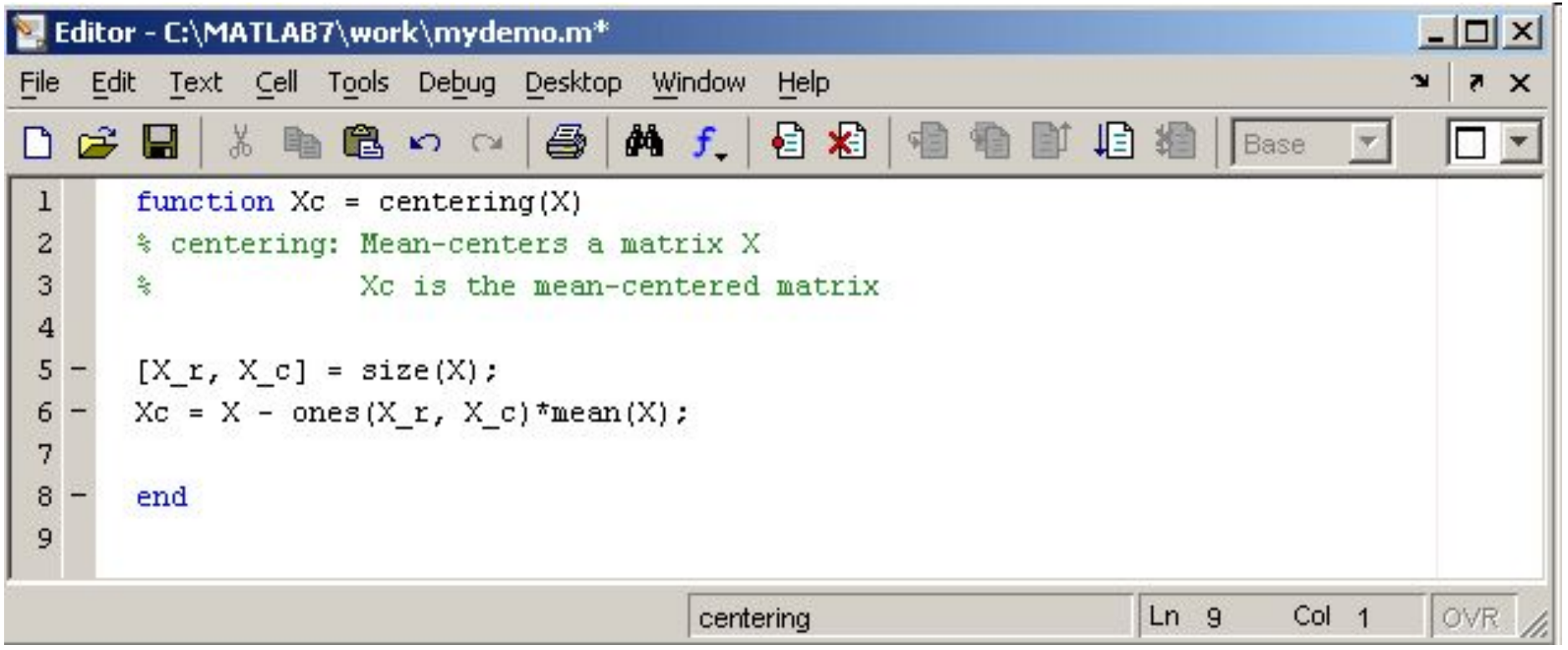
Рассмотренная выше файл-программа является только последовательностью команд MatLab, она не имеет входных и выходных аргументов.

Для использования численных методов и при программировании собственных приложений в MatLab необходимо уметь составлять файл-функции, которые производят необходимые действия с входными аргументами и возвращают результат действия в выходных аргументах.

Разберем несколько простых примеров, позволяющих понять работу с файл-функциями.

Проводя предобработку данных многомерного анализа часто применяет центрирование.

Имеет смысл один раз написать файл-функцию, а потом вызывать его всюду, где необходимо производить центрирование. Откройте в редакторе М-файлов новый файл и наберите



Слово function в первой строке определяет, что данный файл содержит файл-функцию.

Первая строка является заголовком функции, в которой размещается имя функции и списка входных и выходных аргументов.

В примере имя функции centering, один входной аргумент X и один выходной — Xc. После заголовка следуют комментарии, а затем — тело функции (оно в данном примере состоит из двух строк), где и вычисляется ее значение. Важно, что вычисленное значение записывается в Xc.

Не забудьте поставить точку с запятой для предотвращения вывода лишней информации на экран. Теперь сохраните файл в рабочем каталоге. Обратите внимание, что выбор пункта **Save** или **Save as** меню **File** приводит к появлению диалогового окна сохранения файла, в поле **File name** которого уже содержится название centering. Не изменяйте его, сохраните файл функцию в файле с предложенным именем!

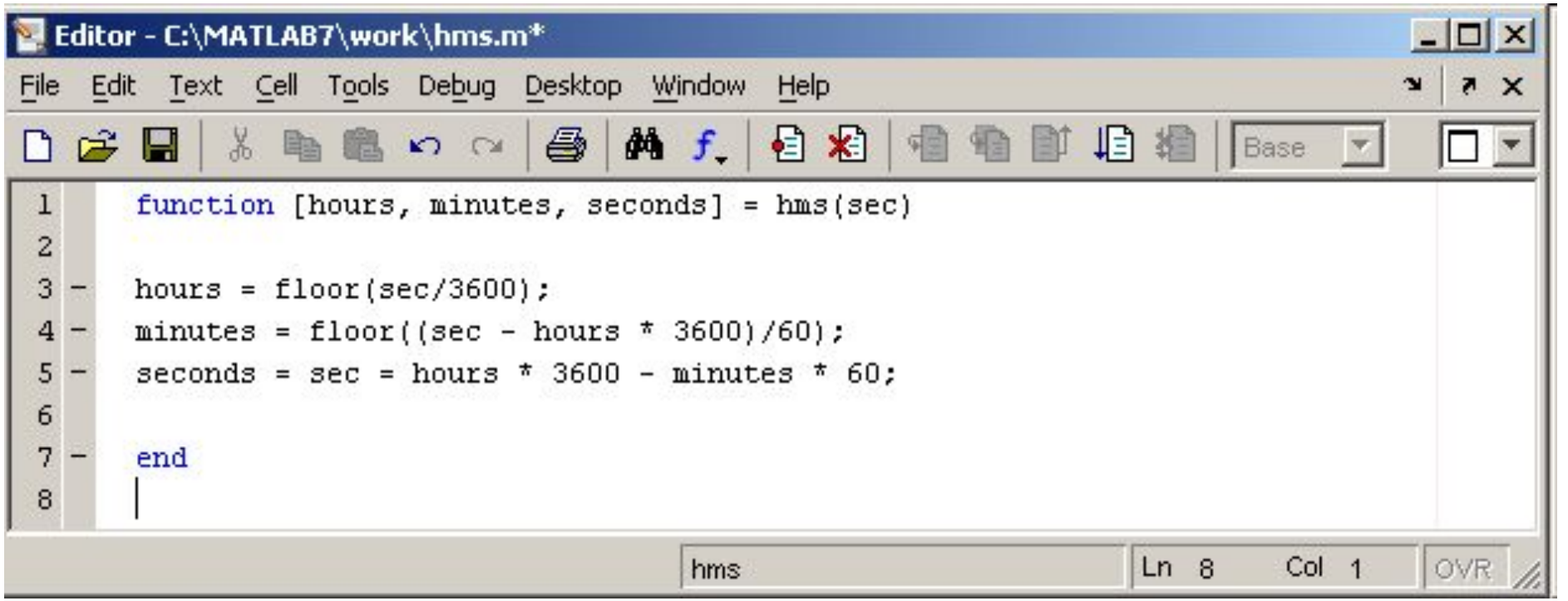
Теперь созданную функцию можно использовать так же, как и встроенные sin, cos и другие.

Вызов собственных функций может осуществляться из файл-программы и из другой файл-функции. Попробуйте сами написать файл-функцию, которая будет шкалировать матрицы, т.е. делить каждый столбец на величину среднеквадратичного отклонения по этому столбцу.

Можно написать файл-функции с несколькими входными аргументами, которые размещаются в списке через запятую. Можно также создавать и функции, возвращающие несколько значений.

Для этого выходные аргументы добавляются через запятую в список выходных аргументов, а сам список заключается в квадратные скобки.

Хорошим примером является функция, переводящая время, заданное в секундах, в часы, минуты и секунды.



При вызове файл-функций с несколькими выходными аргументами результат следует записывать в вектор соответствующей длины.

Программирование М-функций

1. **Понятия функции и сценария.**

Работая в интерактивном режиме, приходится всё время вводить нужные команды с клавиатуры. Затем, в следующих сеансах работы с системой MATLAB, можно командой load ввести из MAT-файла ранее сохранённую информацию о переменных, с которыми ранее осуществлялись вычисления. Однако команды для её обработки потребуется заново вводить с клавиатуры.

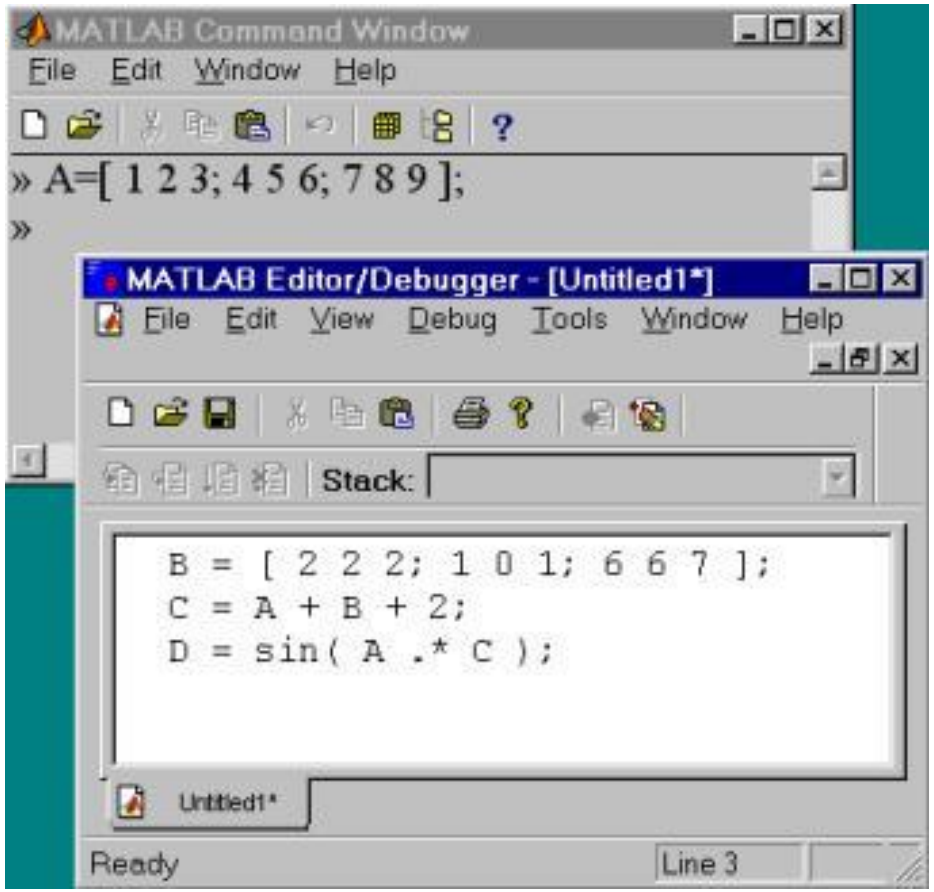
Это вполне приемлемо, когда заранее неизвестен порядок обработки информации, или когда объём такой обработки невелик и повторяется редко.

Если же порядок обработки информации заранее известен и её надо осуществлять многократно, то лучше последовательность команд оформить в виде сценария.

Сценарий - это текстовый файл, в котором в нужном порядке записаны команды, подлежащие последовательному выполнению.

Создать такой файл можно с помощью любого текстового редактора, но система MATLAB располагает собственным текстовым редактором для этих целей, который к тому же обеспечивает дополнительные удобства.

В результате лучше всего пользоваться именно этим редактором. Он вызывается командой меню File | New | M-file и работает в своём собственном окне:



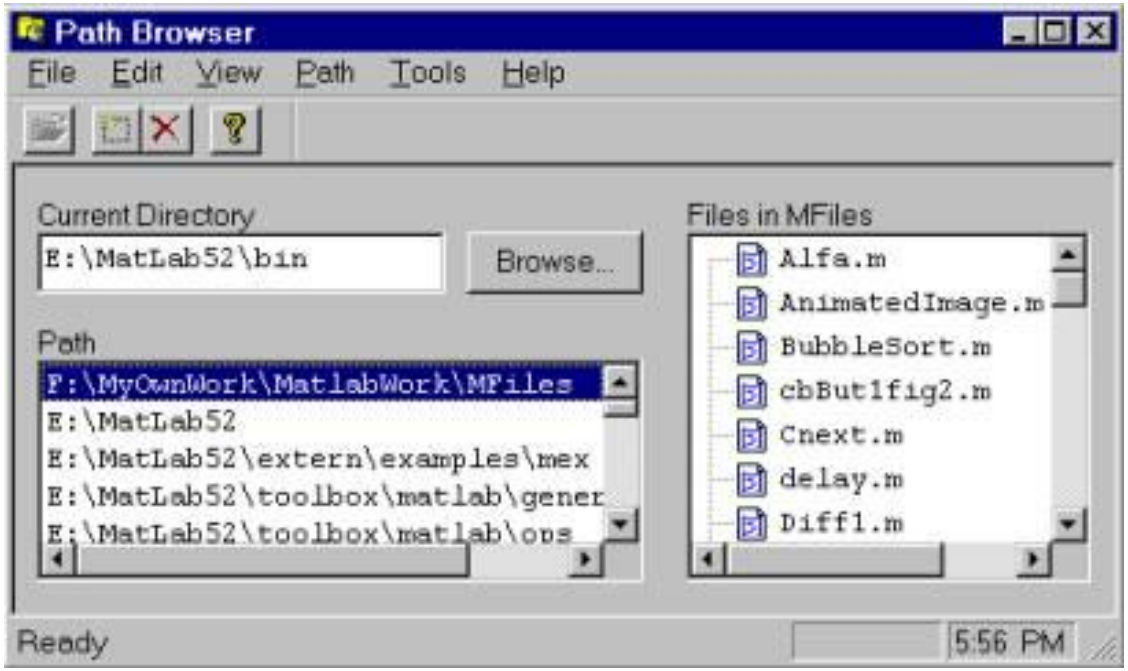
Сценарий производит вычисления как с переменными, которые определяются непосредственно внутри него, так и с переменными, ранее определёнными в рабочем пространстве системы MATLAB. Таким образом, пространство переменных у них общее.

После создания сценария его надо сохранить в файле на диске. Этот файл может иметь произвольное имя (желательно, чтобы оно не совпало с именами файлов, входящих в стандартную поставку системы MATLAB), но расширение имени должно состоять из одной буквы m (например, myScript1.m и тому подобное).

По этой причине принято говорить, что сценарии хранятся в M-файлах. Их не надо путать с MAT-файлами, изученными ранее и в которых хранятся переменные из рабочего пространства системы MATLAB.

Каталог на диске для хранения M-файлов может быть любым, но путь к этому каталогу обязательно должен быть известен MATLABу. MATLAB хранит сведения обо всех таких каталогах.

Для нового каталога надо выполнить команду меню (командного окна) File | Set Path…, с помощью которого вызывается диалоговое окно с именем Path Browser (Просмотрщик путей доступа к файлам):



В этом окне показывается список всех зарегистрированных путей доступа. Для добавления нового каталога служит команда меню этого окна

Path | Add to path..

После того, как мы добавили новый каталог, в нём можно сохранить файл с созданным в редакторе сценарием. Сохранёный таким образом сценарий в любой момент можно выполнить целиком (то есть выполнить за один раз все команды, входящие в этот сценарий), если набрать в командном окне имя файла (без расширения), содержащего сценарий, и нажать клавишу Enter.

Сценарий обрабатывает как свои собственные переменные, так и переменные, определённые до вызова сценария в командном окне системы MATLAB и хранящиеся в её рабочем пространстве.

Сценарию нельзя в момент его вызова передать для обработки в виде параметров дополнительную информацию, не содержащуюся в рабочем пространстве.

Чтобы сделать возможным передачу для дальнейшей обработки входных параметров, а также разделить рабочие пространства, нужно вместо сценария написать функцию.

Функции реализуют определённый алгоритм обработки входной информации и идеально приспособлены для решения отдельных задач, которые в совокупности позволяют разрешить некоторую крупную проблему.

2. Синтаксис определения и вызова М-функций.

Текст М-функции должен начинаться с *заголовка*, после которого следует *тело функции*.

Заголовок определяет " интерфейс" функции (способ взаимодействия с ней) и устроен следующим образом:
function [RetVal1, RetVal2,…] = FunctionName(par1, par2,…)

Здесь провозглашается функция (с помощью неизменного "ключевого" слова function) с именем FunctionName, которая принимает входные параметры par1, par2,…, и вырабатывает (вычисляет) выходные (возвращаемые) значения RetVal1, RetVal2…

По-другому говорят, что *аргументами функции* являются переменные par1, par2,.., а *значениями функции* (их надо вычислить) являются переменные RetVal1, RetVal2,… .

Указанное в заголовке имя функции (в приведённом примере - FunctionName) должно служить именем файла, в который будет записан текст функции.

Для данного примера это будет файл FunctionName.m (расширение имени, по-прежнему, должно состоять лишь из одной буквы m).

Рассогласования имени функции и имени файла не допускается!
Тело функции состоит из команд, с помощью которых вычисляются возвращаемые значения.

Тело функции следует за заголовком функции. Заголовок функции плюс тело функции в совокупности составляют определение функции.

Как входные параметры, так и возвращаемые значения могут быть в общем случае массивами (в частном случае - скалярами) различных размерностей и размеров.

Например, функция MatrProc1

```
function [ A, B ] = MatrProc1( X1, X2, x )
A = X1 .* X2 * x;
B = X1 .* X2 + x;
```

рассчитана на "приём" двух массивов одинаковых (но произвольных) размеров и одного скаляра.

Эти массивы в теле функции сначала перемножаются поэлементно, после чего результат такого перемножения ещё умножается на скаляр.

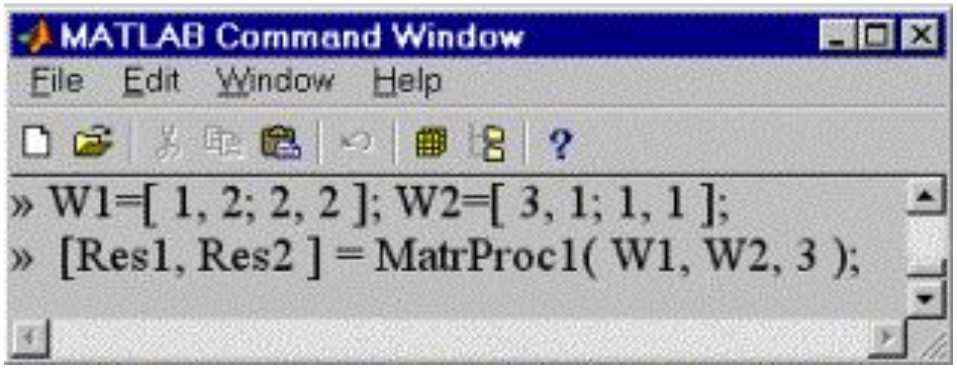
Таким образом порождается первый из выходных массивов. Одинаковые размеры входных масивов X1 и X2 гарантируют выполнимость операции их поэлементного умножения. Второй выходной массив (с именем B) отличается от первого тем, что получается сложением со скаляром (а не умножением).

Вызов созданной нами *функции* осуществляется из командного окна системы MATLAB (или из текста какой-либо другой функции) обычным образом: записывается имя функции, после которого в круглых скобках через запятую перечисляются *фактические входные параметры*, со значениями которых и будут произведены вычисления.

Фактические параметры могут быть заданы числами (массивами чисел), именами переменных, уже имеющими конкретные значения, а также выражениями.

Если фактический параметр задан именем некоторой переменной, то реальные вычисления будут производиться с копией этой переменной (а не с ней самой). Это называется *передачей параметров по значению*.

Ниже показан вызов из командного окна MATLABa ранее созданной нами для примера функции MatrProc1.



Здесь имена фактических входных параметров (W1 и W2) и переменных, в которых записываются результаты вычислений (Res1 и Res2), не совпадают с именами аналогичных переменных в определении функции MatrProc1.

Очевидно, что совпадения и не требуется, тем более, что у третьего входного фактического параметра нет имени вообще! Чтобы подчеркнуть это возможное отличие, имена входных параметров и выходных значений в определении функции называют формальными.

В рассмотренном примере вызова функции MatrProc1 из двух входных квадратных матриц 2 x 2 получаются две выходные матрицы Res1 и Res2 точно таких же размеров:

Res1 =

9 6

6 6

Res2 =

6 5

5 5

Вызвав функцию

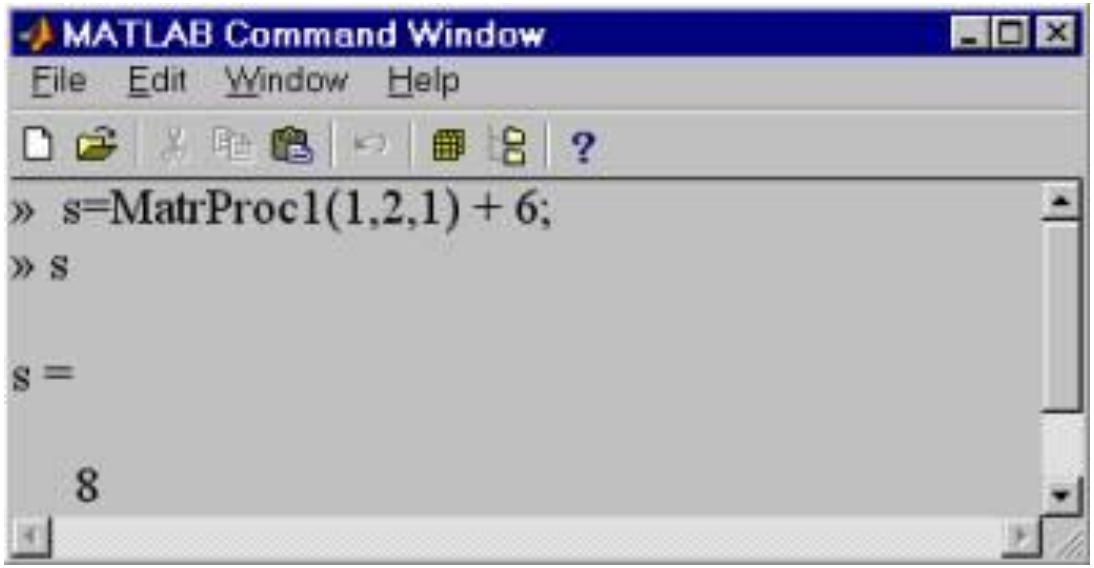
MatrProc1 [r1,r2] = MatrProc1([1 2 3; 4 5 6], [7 7 7; 2 2 2], 1);

с двумя входными массивами размера 2x3, получим две выходные матрицы размера 2x3. То есть, одна и та же функция MatrProc1 может обрабатывать входные параметры различных размеров и размерностей!

Можно вместо массивов применить эту функцию к скалярам (это всё равно массивы размера 1x1).

Теперь рассмотрим вопрос о том, можно ли использовать эту функцию в составе выражений так, как это делается с функциями, возвращающими единственное значение?

Оказывается это делать можно, причём в качестве значения функции, применяемого для дальнейших вычислений, используется первое из возвращаемых функцией значений. Следующее окно системы MATLAB иллюстрирует это положение:



При вызове с параметрами 1,2,1 функция MatrProc1 возвращает два значения: 2 и 3. Для использования в составе выражения используется первое из них.

3. Конструкции управления.

В любом языке программирования, в том числе и в языке программирования, встроенном в систему MATLAB (его часто называют М-языком), имеются специальные конструкции, которые задаются с помощью зарезервированных ключевых слов этого языка и служат для управления порядком выполнения операций.

Такие конструкции часто называют операторами управления. К джентльменскому набору операторов управления относятся *операторы ветвления* и *операторы цикла.* Начнём с операторов ветвления.

К операторам ветвления в М-языке относятся *условный оператор* и *оператор переключения*.

Условный оператор использует ключевые слова
if ("если"), else ("иначе")
elseif ("иначе если"), end ("конец всей конструкции")
и может выступать в одной из следующих трёх форм.

Во-первых,
if условие
…
end

Во-вторых,

```
if условие
    …
else
…
end
```

И, наконец, в форме

```
if условие1
    …
elseif условие2
…
else
    …
end
```

в которой ветвей с ключевым словом elseif может быть много.

Область действия условного оператора начинается ключевым словом if, а заканчивается ключевым словом end.

Под *условием* понимается произвольное выражение (чаще всего это выражение включает в себя операции сравнения и логические операции), истинность или ложность которого понимается как отличие или равенство нулю.

Если условие истинно, то выполняются команды, стоящие после строки с ключевым словом if.

Если условие ложно, то эти команды пропускаются и переходят либо к следующему за условным оператору (первая форма), либо проверяют ещё одно условие в строке с ключевым словом elseif (третья форма условного оператора), либо выполняются без дополнительных проверок команды, стоящие после строки с ключевым словом else (вторая из приведённых выше форм).

Их можно использовать и в условных выражениях, входящих в условные операторы MATLABa. В тех случаях, когда значением таких выражений будет массив, *истинность условия* наступает, когда *истинны (не равны нулю) все элементы массива.*

Если хоть один элемент такого массива будет равен нулю, то условие считается ложным. Кроме того, ложность имеет место при использовании *пустых массивов*.

Приведём иллюстрирующий работу условного оператора фрагмент кода

```
A = [ 1 2; 4 0 ];
     if A
          b = 1;
    else
          b =2;
```

в результате выполнения которого переменная b получит значение 2, так как матрица A содержит один нулевой элемент, и всё условие считается ложным.

Оператор переключения использует ключевые слова switch ("переключить"), case ("случай"), otherwise ("иначе") и имеет следующую конструкцию:

```
switch выражение
        case значение1
       …
        case { значение2, значение3 }
           …
        …
       otherwise
       …
    end
```

Сначала вычисляется вырабатывающее *скалярное* числовое значение *выражение*, а затем полученный результат сравнивается с набором значений значение1, значение2, значение3 и так далее. В случае совпадения с одним из значений, выполняется нижестоящая ветка. Если нет совпадения ни с каким из перечисленных значений, то выполняется ветка, стоящая после ключевого слова **otherwise**. Строк с ключевым словом case может быть много, а строка с ключевым словм otherwise - одна.

Теперь рассмотрим операторы цикла, призванные циклически повторять участки программного кода.

В зависимости от способа определения условия останова повторов различают два вида операторов цикла в М-языке системы MATLAB.

Первый из них использует ключевые слова for ("для") и end. Он имеет вид

while *выражение*

…

end

Здесь повтор участка кода, обозначенного многоточием, продолжается пока *выражение* истинно (не равно нулю). В случае массивов истинность наступает, когда все элементы массива истинны.

Следующий фрагмент вычисляет сумму отрезка ряда:

```
S=0; k=1; u=1;
 while u > 1e-8
 S = S + u;
k = k + 1; u = 1/k^2;
end
```

Условием останова служит требование к очередным слагаемым быть больше некоторого заранее определённого числа: как только слагаемое станет меньше этого числа, суммирование прекратится.

Другой вариант оператора цикла использует ключевые слова for (" для") и end. Он имеет вид:

for varName = *выражение*
…
end
где **varName** - произвольно выбираемое программистом имя так называемой *переменной цикла*.

В отличие от первого варианта оператора цикла здесь легко прогнозировать количество итераций (повторов), так как тело цикла (обозначено многоточием) выполняется для всех возможных значений переменной **varName**.

Набор возможных значений для переменной цикла поставляет *выражение*.

Чаще всего *выражение* представлено с помощью ранее изученной операции "*диапазон значений*".

В следующем фрагменте кода осуществляется сложение 57 членов ряда:

```
 S=0;
for k = 1 : 1 : 57
    S = S + 1/k^2;
end
```

При каждом новом проходе переменная цикла k увеличивается на единицу. Как легко заметить, здесь осуществляется суммирование того же ряда, что и в примере, посвящённом оператору цикла while…end.

В предыдущем примере условием останова было требование к величине очередного слагаемого, а теперь этим условием является исчерпание всех возможных значений переменной цикла.

В итоге можно сделать вывод, что использование того или иного варианта оператора цикла диктуется особенностями конкретной математической задачи.

Вместо операции задания диапазона можно явно указывать весь набор возможных значений в виде вектор-строки, например

for m = [2, 5, 7, 8, 11, 23]

что приведёт к шести итерациям.

При первой итерации переменная цикла m будет равна 2, при втором - 5 и так далее до исчерпания всех возможных значений.

Достаточно необычным вариантом может показаться использование матриц в управляющем выражении:

A = [1 2; 3 4]; f

or k = A

Такой цикл будет повторяться ровно столько раз, сколько столбцов в матрице A, то есть два раза для данного конкретного случая.

При каждом проходе переменная цикла принимает значение очередного столбца матрицы, то есть является вектор-столбцом.

Например, следующий фрагмент

```
S=0; A = [ 1 2; 3 4];
for k = A
S = S + sqrt( k(1)^2 + k(2)^2 );
end
```

вычисляет сумму "длин" столбцов матрицы A.

Оба вида операторов цикла можно прервать, если применить оператор break внутри тела цикла.

Для *повышения эффективности* программы везде, где это возможно, вместо операторов цикла лучше применять эквивалентные по результатам операции с массивами, так как последние исполняются в системе MATLAB быстрее.

Например, вместо

```
 k=0;
for x = 0 : 0.1 :100
k=k+1; y( k ) = cos( x );
end
```

лучше использовать операции с массивами:

```
x = 0 : 0.1 : 100; y = cos( x );
```

так как они быстрее исполняются и записываются короче.

4. Проверка входных параметров и выходных значений М-функции.

Как мы говорили выше, несовпадение типов и числа фактических и формальных параметров в М-функции приводит к их неправильной работе.

Но "пользователь" М-функции всегда может ошибиться при её вызове. Поэтому желательно встраивать внутрь кода М-функций проверки входных параметров.

Ранее рассмотренная в пункте 2 функция MatrProc1 предполагала использовать в качестве первого и второго аргументов массивы одинаковых размеров.

Если пользователь по ошибке задаст фактические параметры в виде массивов разных размеров, то в процессе выполнения функции возникнет ошибка.

Чтобы избежать этого, можно в теле функции MatrProc1 организовать проверку размеров первого и второго параметров:

```
 function [ A, B ] = MatrProc1( X1, X2, x )
if size( X1 ) ~= size( X2 )
     error( 'Bad 1st and 2nd parameters' )
end
A = X1 .* X2 * x;
B = X1 .* X2 + x;
```

Теперь при вызове функции MatrProc1 с неправильными размерами первого и второго аргументов, стандартная функция системы MATLAB error будет корректно останавливать всю работу и выводить в командное окно MATLABa наше диагностическое сообщение (аргумент функции error), после чего пользователю останется лишь повторно вызвать функцию MatrProc1, но уже с правильными параметрами.

Затем нужно добавить ещё проверку третьего параметра на скалярность, что можно выполнить следующим фрагментом кода:

```
[ m ,n ] = size( x );
if ( m ~=1 | n ~= 1 )
   error( 'Bad 3d parameter' )
end
```

Наконец, неплохо проверить общее число параметров, с которыми функция была вызвана.

Для этой цели в системе MATLAB специально предусмотрена переменная с именем nargin.

Её значением является количество аргументов, фактически переданное функции при вызове.

Тогда проверка на число параметров выполняется следующим образом:

```
if nargin ~= 3
   error( 'Bad number of parameters' )
end
```

Более того, в системе MATLAB предусмотрена переменная nargout, содержащая число возвращаемых значений, предполагающихся в реальной форме вызова этой функции.

Например, вызов
[s1, s2, s3] = MatrProc1(x1, x2, x)
предполагает получить аж три возвращаемых значения, в то время как из определения функции следует, что возвращаемых значений у этой функции два.

Чтобы предупредить пользователя функции о несовпадении числа ожидаемых возвращаемых значений их номинальному числу, нужно в теле функции осуществить проверку переменной nargout:

```
if nargout ~= 2
   error( 'Must be 2 return values' )
 end
```

Осуществлённые нами проверки приводят к тому, что функцию можно вызвать только с правильным числом входных параметров и возвращаемых значений.

Однако ранее мы встречались со встроенными функциями системы MATLAB, которые могли быть вызваны с разным числом входных параметров (и это очень типично).

В результате фактически разные работы бликого типа выполняются под одним и тем же именем функции, что весьма наглядно и удобно. Таковой, например, является функция plot, имя которой говорит о построении графиков функций.

Если бы разные варианты вызовов этой функции пришлось бы осуществлять под разными именами, то от наглядности не осталось бы и следа.

5. Видимость имён переменных и имён функций.

Локальные и глобальные переменные. Функция располагает собственным, изолированным от рабочего пространства системы MATLAB, пространством переменных.

Поэтому, если перед вызовом М-функции в командном окне MATLABa была определена переменная с именем, например, varName1, то нельзя рассчитывать на то, что переменная в теле функции с этим же именем уже имеет некоторое значение.

Это совсем другая переменная (хотя у неё и то же самое имя varName1) и располагается она в памяти машины в другой области памяти.

Переменные, которые используются в теле М-функции и не совпадают с именами формальных параметров этой функции, называются локальными. По-другому говорят, что они видимы лишь в пределах М-функции.

Извне они не видны (не достижимы). Внутри функции не видны переменные, определённые в командном окне MATLABa - они являются внешними по отношению к функции и не видны в ней.

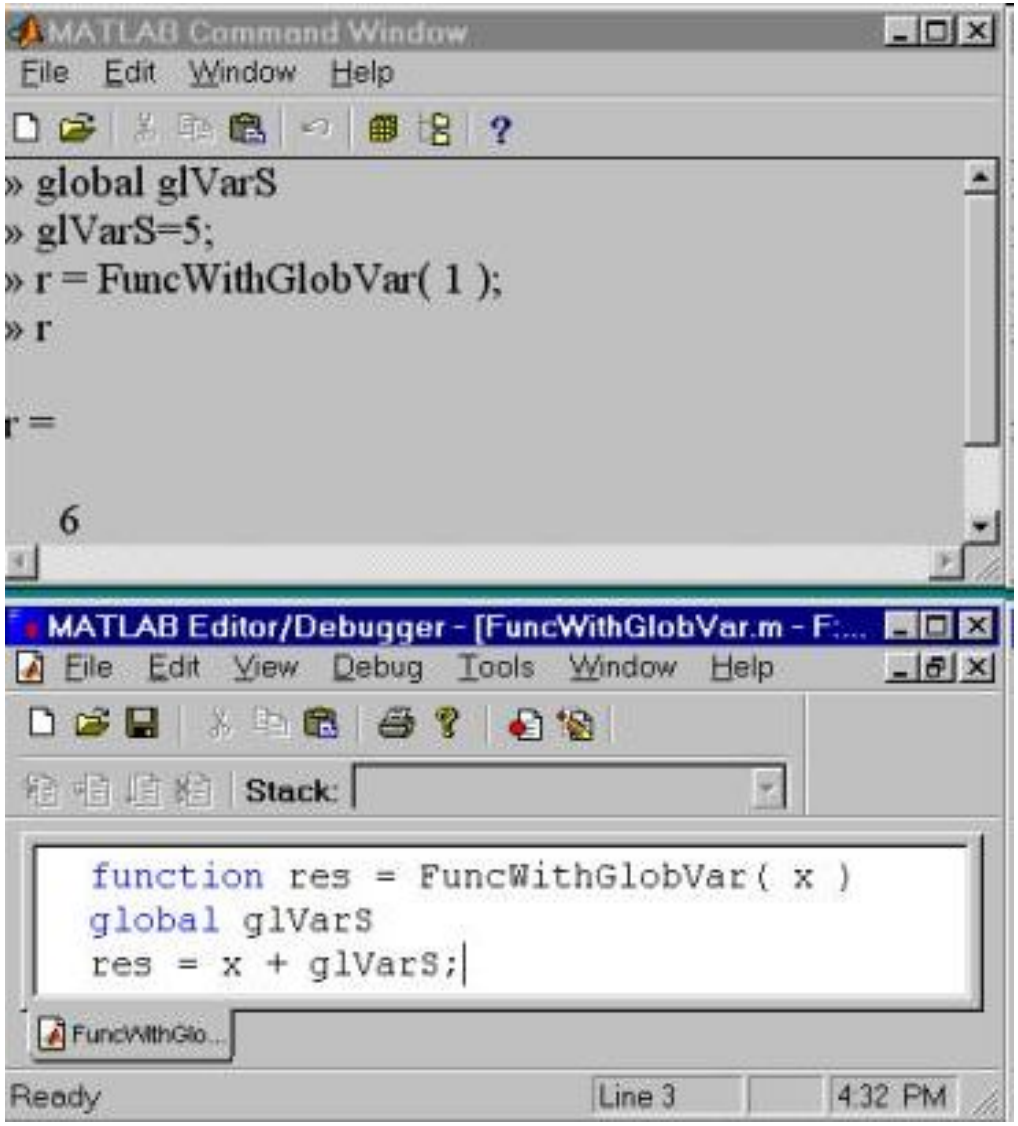
Аналогично, локальные внутри некоторой функции переменные не видны внутри другой М-функции.

Одним из каналов передачи информации из командного окна системы MATLAB в М-функцию и из одной функции в другую является механизм параметров функции.

Другим таким механизмом являются глобальные переменные.

Чтобы рабочая область системы MATLAB и несколько М-функций могли совместно использовать переменную с некоторым именем, её всюду нужно объявить как глобальную с помощью ключевого слова global.

К примеру, переменная glVarS, участвующая в вычислениях в рабочем пространстве и в функции FuncWithGlobVar является одной и той же переменной (единственный участок памяти) повсюду - поэтому её можно использовать в функции без дополнительного присваивания её какого-либо значения:



Так как у глобальных переменных "глобальная" область действия, то чтобы случайно (по ошибке) не переопределить её где-либо, желательно давать таким переменным более мнемонические (более длинные и осмысленные) имена.

Теперь рассмотрим вопрос о видимости имён функций. Если мы сохранили функцию с некоторым именем в файле с этим же именем и расширением m, и кроме того если системе MATLAB известен путь к этому файлу на диске, то эту функцию можно вызывать как из командного окна, так и из других функций.

Однако в тексте М-функции можно поместить опеределения нескольких функций, причём только одна из них может совпадать по имени с именем файла.

Именно эта функция и будет видна из командного окна и других функций. Все остальные функции будут внутренними - их могут вызывать только функции из того же файла.

Например, если в файле ManyFunc.m будет содержаться следующий текст

```
 function ret1 = ManyFunc( x1, x2 )
ret1 = x1 .* x2 + AnotherFunc( x1 )
 function ret2 = AnotherFunc( y )
ret2 = y .* y + 2 * y + 3;
```

состоящий из определений двух функций с именами ManyFunc и AnotherFunc, то извне можно вызывать только функцию ManyFunc.

По-другому можно сказать, что извне видны только функции с именами, совпадающими с именами М-файлов.

Остальные функции должны вызываться этой функцией и другими внутренними функциями.

6. Разработка и отладка М-функций.

Разрабатывая функцию, вы в первую очередь разрабатываете алгоритм решения некторой задачи, после чего переводите его на формальный язык кодирования, которым и является М-язык.

Несмотря на довольно высокую наглядность М-языка (отсутствуют низкоуровневые конструкции, близкие к машинным командам), всё равно это формальный язык. По прошествии времени детали разработок забудутся и для модификации функции придётся всё вспоминать снова.

Чтобы упростить процесс дальнейшей модификации функции, а также её отладки на стадии, когда ещё не удалось добиться правильной работы, в текст функции вставляют комментарии.
Комментарии могут занимать отдельные строки, начинающиеся с символа %, после которого следует текст комментария.

Также комментарии можно располагать в конце любой строки кода, поскольку интерпретатор М-языка, встретив знак %, считает все символы после него просто комментарием (а не командами, подлежащими переводу в машинную форму и исполнению).

Особую роль в системе MATLAB имеют комментарии, располагающиеся в смежном наборе строк сразу за заголовком определения функции. Весь этот набор строк выводится в командное окно системы MATLAB при исполнении команды

help имя_M-функции

Поскольку такую команду в первую очередь будут применять пользователи функции (а не разработчики), то желательно расположить в этих комментариях описательную информацию и сведения о правильном вызове этой функции.

Теперь подробно остановимся на вопросе об отладке M-функций, то есть на приёмах, с помощью которых можно выявить месторасположение ошибок и их причину.

Система MATLAB осуществляет серьёзную помощь в этом процессе. В частности, при возникновении ошибки в процессе выполнения M-функции, в командное окно выводится приблизительное диагностическое сообщение (не следует переоценивать качество такой диагностики) и номер строки, в котором по мнению MATLABa произошла ошибка.

Другим, более развитым способом отладки функции является применение точек останова и пошаговое выполнения тела функции. Для этого применяют встроенные возможности редактора-отладчика системы MATLAB.

То, что уже многократно применяемый нами редактор (в нём набираем текст функций и с помощью меню сохраняем в файле) заодно является и отладчиком, говорит даже заголовок его окна:

Matlab Editor / Debugger

так как ***debugger*** в переводе с английского означает "***отладчик***".
Чтобы поставить "точку останова" на какой-либо строке кода функции, туда нужно поместить курсор и нажать клавишу F12 (повторное нажатие этой клавиши убирает точку останова).

Вместо нажатия этой клавиши можно выполнить команду меню

Debug | Set/Clear Breakpoint

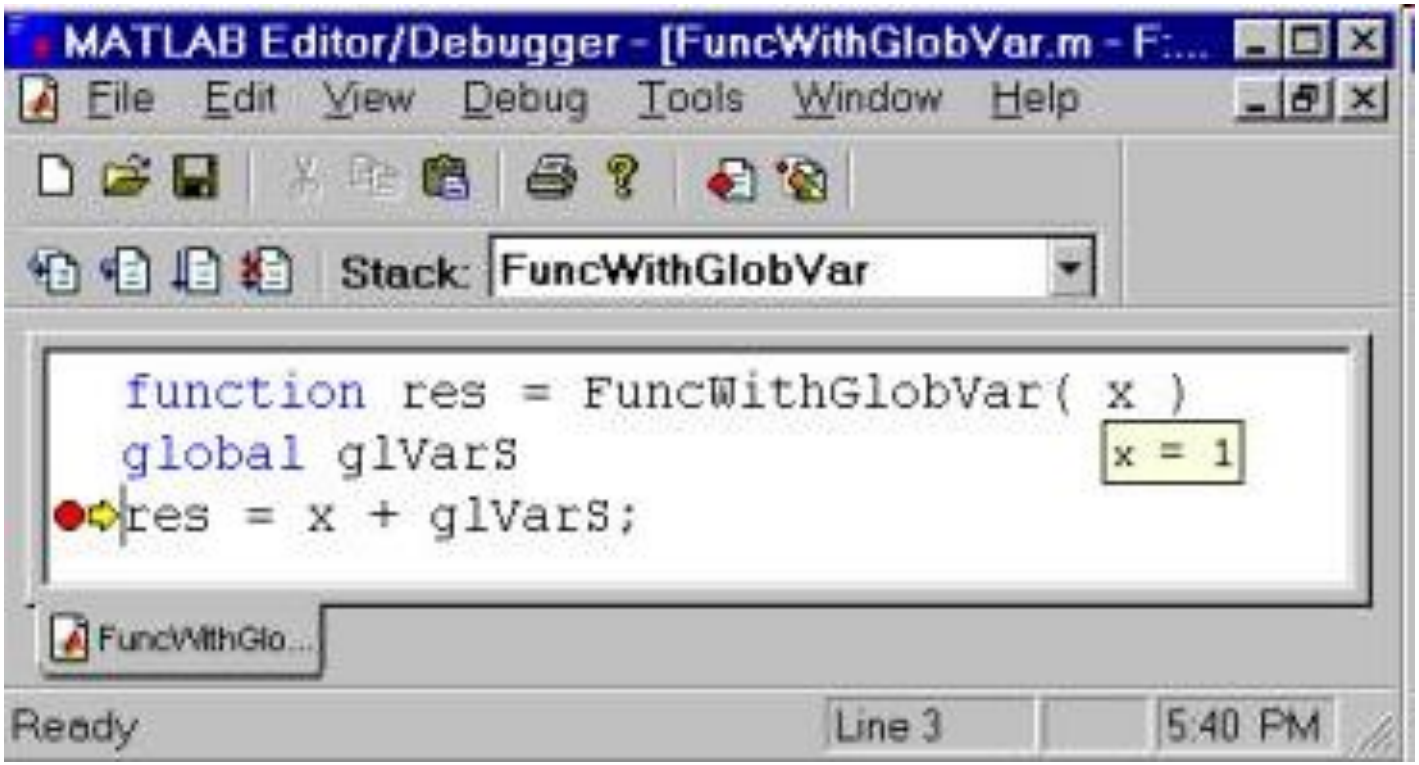
но всё же быстрее это можно выполнить нажатием клавиши. После этого в строке слева появляется красный кружок, указывающий на то, что в данной строке проставлена точка останова.

После этого, не закрывая окна Редактора/Отладчика (Editor/Debugger), переключаем фокус ввода с клавиатуры в командное окно MATLABa и запускаем обычным образом функцию на выполнение.

После этого и произойдёт останов выполнения функции прямо на строке, в которой поставлена точка останова (Breakpoint).

Теперь мы можем просматривать фактические значения входных параметров функции, текущие значения глобальных и локальных переменных, а также значения выражений.

Чтобы просмотреть значение переменной, достаточно подвести курсор к её имени в тексте функции, после чего на экране появится всплывающий жёлтый прямоугольник со значением переменной внутри него:



Далее, нажимая клавишу F10 мы можем выполнять функцию построчно, каждый раз проверяя результаты такой пошаговой работы функции.

В результате всегда можно "окружить ошибку" и выявить её причину.

Изменив текст функции и устранив выявленную ошибку,запускаем функцию на выполнение, в результате чего либо удостовериваемся в её правильной работе, либо находим новую ошибку.

Желательно продумать методику отладки, запуская функцию на выполнение с разными значениями аргументов и разными значениями глобальных функций.

В результате такого итерационнного отладочного процесса приходят к правильно работающим функциям.

7. Массивы символов.

До сих пор мы мели дело с единственным типом данных - массивами вещественных чисел.

Это действительно основной тип данных системы MATLAB, предназначенный для вычислений. В то же время, при рассмотрении графики MATLABa мы столкнулись с типом данных "*короткое целое*", обозначаемое с помощью ключевого слова uint8.

Этот тип данных специально предназначен для компактного хранения больших массивов целых чисел, что очень характерно для графических задач.

Однако производить вычисления с типом данных uint8 нельзя (по крайней мере в версии системы MATLAB 5.2).

Если всё же нужно произвести вычисления, то сначала тип данных uint8 приводят явно к типу double, производят вычисления и возвращаются к типу uint8 для дальнейшего хранения.

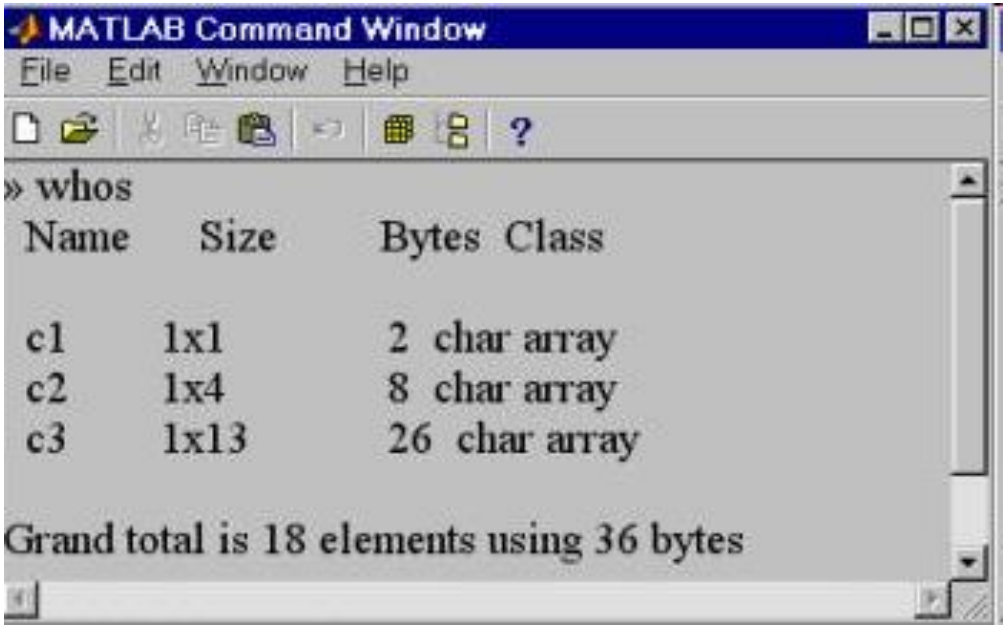
Во всех языках программирования, и MATLAB здесь не исключение, большую роль играет обработка текстовых данных.

Для этой цели в системе MATLAB предусмотрен тип данных char (то есть "символ").

Текстовые данные, в том числе и одиночный символ, должны заключаться с обеих сторон апострофами:

c1 = 'a'; c2='abcd'; c3 = 'Hello, World!';

В результате таких присваиваний создаются переменные (естественно, это массивы - в системе MATLAB всё является массивами) типа char:



Из рисунка видно, что текстовые данные в системе MATLAB являются вектор-строками типа char (одна строка и несколько столбцов по числу содержащихся символов).

Например, переменная c3 является символьным массивом (часто говорят - строкой символов) из 13 символов, занимающим 26 байт.

Таким образом, под каждый символ отводится 2 байта. Каждый символ кодируется целым числом в соответствии со стандартной системой кодировки ASCII.

Легко практически выяснить, какой код соответствует тому или иному символу. Следующий фрагмент

```
code = double( c1( 1 ) )
code=
             9 7
```

показывает, что символу 'a' соответствует десятичное число 97.

Если после того, как переменная c3 получила текстовое значение 'Hello, World!', написать

```
c3 = 3.14;
```

то переменная c3 станет уже переменной типа double. Так как в сложных и громоздких M-функциях могут возникнуть ситуации, когда заранее неизвестен тип переменной в какой-либо момент времени исполнения функции, то с целью определения типа переменной следует применить функцию isa.

Например, следующий вызов этой функции
isa(s3, 'char')

вернёт истину (единицу), если переменная s3 является в этот момент строковой (символьной), и вернёт ложь (нуль) в противоположном случае.
По отношению к массивам символов справедливы также все операции, которые мы ранее рассмотрели для случая массивов типа double.

Например, вместо группового присваивания c2 = 'abcd' можно организовать поэлементное присваивание с помощью операции индексации:

c2(1)='a'; c2(2)='b'; c2(3)='c'; c2(4)='d';
или осуществить операцию конкатенации
c2 = ['abc' , 'd']; c2 = [c2 , ' QWERTY'];

В тесной связи с рассмотренной операцией конкатенации текстовых строк находится стандартная функция int2str, которая преобразует целые числовые значения в символы, отображающие эти целые числа.

Например, вызов функции

res = int2str(2)

приведёт к появлению текстовой переменной res со значением '2'. В итоге, мы имеем возможность сформировать в цикле набор нескольких имён функций, отличающихся только последним символов - их номером:

```
name = 'function'; arg = 10.7;
 for k = 1 : 10
Name = [ name ,int2str( k ) ];
 res( k ) = feval( Name, arg );
end
```

и даже вычислить значения всех таких функций при значении аргумента arg. Это осуществляется с помощью стандартной функции системы MATLAB feval, которая принимает в качестве своего первого аргумента текстовую строку с именем M-функции, подлежащей вычислению.

Второй и последующие аргументы этой функции служат для передачи в качестве аргументов вычисляемым функциям.

В вышеприведённом фрагменте результаты вычислений десяти функций запоминаются в массиве res.
Если требуется в одной переменной запомнить несколько имён функций (это возможно в случае их одинаковой длины) для последующего их исполнения с помощью feval, то можно сформировать текстовый массив размерности 2;

```
Names( 1, : ) = 'function1';
Names( 2, : ) = 'function2';
```

Первая строка этого массива содержит имя первой функции, вторая строка - второй функции. Размер этого массива типа char есть 2 x 9. Часто текстовые строки используются для вывода в командное окно системы MATLAB для информирования пользователя о ходе выполнения M-функции.

Это осуществляется с помощью функции disp, принимающей в качестве аргумента текстовую строку:

```
x = 7;
message = [ ' Variable x = ', int2str( x ) ];
disp( message );
```

Кроме того, в командное окно надо выводить сообщения, предупреждающие пользователя о необходимости ввода с клавиатуры значения переменной:

VarX = input(' VarX = ? ');

Функция input выводит в командное окно текст, являющийся её аргументом, после чего ожидает ввода значения с клавиатуры и нажатия клавиши Enter. Таким образом можно ввести с клавиатуры числовое значение и запомнить её значение в переменной VarX.

Внутри строки-аргумента функции input может присутствовать специальный набор из двух символов /n, приводящий к показу сообщения на двух строках (часть сообщения после /n показывается на новой строке).

Для ввода текстового значения, а не числового, требуется вызывать функцию input с двумя аргументами:

VarStr = input(' StringVar = ', 's');

В результате выполнения этой функции на экране появляется надпись
StringVar =
после чего можно набирать необходимый текст с клавиатуры, заканчивая ввод нажатием клавиши Enter.

Если нажать Enter, не введя с клавиатуры никакого текста, то переменная VarStr примет значение пустого массива. Желательно перед использованием этой переменной проверять её на этот случай функцией isempty(VarStr), возвращающей единицу, когда аргумент является пустым массивом.

Система MATLAB располагает также полным набором функций для "классической" обработки текстов. К таким функциям относятся функции findstr, blanks, deblank, num2str, str2num, strcat, strcmp, strcmpi, strrep, strtok.

Функции num2str, str2num производят преобразования из строк в действительные числа и обратно, функции blanks, deblank, strrep работают с пробелами и повторением символов, функция strcat осуществляет конкатенацию, функции strcmp и strcmpi сравнивают значения двух строк, функции findstr и strtok находят или выделяют в строках подстроки.
Например, в следующем фрагменте находится массив позиций вхождения слова Hello в текст, содержащийся в переменной vStr:

```
 innerStr = 'Hello';
vStr='Hello is the word. Hello is opposite to bye.';
 positions = findstr( vStr, innerStr );
```

В результате переменная (массив) positions принимает следующее значение:

```
 positions =
                1 20
```

В итоге функция findstr обнаружила два вхождения переменной innerStr в текст Vstr. Первое вхождение имеет место начиная с самого первого символа, второе вхождение имеет место на 20-ом символе (включая пробелы, разумеется).