МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ (ФИЛИАЛ) ФЕДЕРАЛЬНОГО
ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Гареева Г.А., Григорьева Д.Р.

Теоретические основы среды MatLab

Учебное пособие
для студентов экономических направлений подготовки

Набережные Челны
2017 г.

УДК 004.94
ББК 32.973
О-75

Теоретические основы среды MatLab. Учебное пособие для лабораторных и практических занятий для студентов экономических направлений подготовки / Составители: Гареева Г.А., Григорьева Д.Р. – Набережные Челны: Издательско-полиграфический центр НЧИ К(П)ФУ, 2017. - 140 с.

Учебное пособие предназначено для студентов экономического отделения, изучающих дисциплины «Пакеты прикладных программ», «Разработка Matlab приложений», «Имитационное моделирование социально-экономических систем», «Интеллектуальные математические методы в менеджменте» по образовательным программам бакалавриата и магистратуры.

Рецензенты: кандидат экономических наук, доцент Карамышев А.Н.
кандидат педагогических наук, доцент Гуничева Е.Л.

Печатается по решению учебно-методической комиссии экономического отделения Набережночелнинского института (филиала) К(П)ФУ (протокол №3от 27.11.2017)

Содержание

1. Назначение, принципы функционирования и использования системы MATLAB

СистемаMatLab (Пакет MatLab) разработана специалистами компании MathWorkInc. Само название произошло от сокращения MATrixLABoratory – матричная лаборатория.

Эта система предназначена для осуществления любых численных расчетов и моделирования технических и физических систем, а также выполнения научных и инженерных расчетов. Кроме того, данный математический пакет можно применять в системах автоматизированного проектирования (САПР), потому что из всех математических пакетов MatLab ближе всех к идеологии САПР.

Система MatLab построена по модульному принципу. В ней есть модули для исследования различных объектов, системы визуального моделирования, возможности моделирования в Simulink, что очень удобно для анализа больших систем, которые состоят из отдельных блоков. MatLab может состыковываться с реальным оборудованием, имеет возможность подключать какие-либо программы, написанные с Си. Помимо этого существует возможность данным модулям или библиотекам работать в режиме реального времени. В основе MatLab лежит громадная библиотека научно-технических расчетов, где очень много различных численных и других методов, которые используются при проектировании САПР. Внутри имеется символьный процессор из Maple. Самая сильная сторона Maple – это символьные вычисления, и она практически полностью присутствует в MatLab, т.е., выбирая MatLab, мы одновременно выбираем самые сильные стороны Maple. На сегодняшний день MatLab – самая распространенная математическая программа для технических применений.

Система поддерживает выполнение операций с векторами, матрицами и массивами данных, работу с алгебраическими полиномами, решение нелинейных уравнений и задач оптимизации, численное интегрирование дифференциальных и разностных уравнений, построение различных трафиков, трехмерных поверхностей и линий уровня.

Основной объект системы MatLab – числовой массив или, другими словами, матрица, в которой допускается применение комплексных элементов. Использование матриц не требует явного указания их размеров.

Система MatLab обеспечивает выполнение операций с векторами и матрицами даже в режиме непосредственных вычислений. Ею можно пользоваться как мощнейшим калькулятором, в котором наряду с обычными арифметическими и алгебраическими действиями могут использоваться такие сложные операции, как обращение матрицы, вычисление ее собственных значений и векторов, решение систем линейных алгебраических уравнений и др. Особенностью системы является ее открытость, т.е. возможность ее модификации и адаптации к конкретным задачам пользователя.

MatLab предоставляет широкие возможности для работы с сигналами, для расчета и проектирования аналоговых и цифровых фильтров, включая построение их частотных, импульсных и переходных характеристик. Имеются в системе и средства выполнения спектрального анализа и синтеза, в частности, реализации прямого и обратного преобразования Фурье. Благодаря этому ее довольно удобно использовать при проектировании электронных устройств. С системой MatLab поставляются свыше ста подробно прокомментированных M-файлов, которые содержат демонстрационные примеры и определения новых операторов и функций. Наличие этих примеров и возможность работать в режиме непосредственных вычислений значительно облегчают изучение системы пользователям, заинтересованным в применении математических расчетов.

Система MatLab использует собственный M-язык, который сочетает в себе положительные свойства различных известных языков программирования высокого уровня. С языком Basic систему MatLab роднит то, что она представляет собой интерпретатор (осуществляет пооператорное компилирование и выполнение программы, не образуя отдельного исполняемого файла); M-язык имеет незначительное количество операторов; в нем отсутствует необходимость объявлять типы и размеры переменных. От

языка Pascal система MatLab позаимствовала объектно-ориентированную направленность, т. е. такое построение языка, которое обеспечивает образование новых типов вычислительных объектов на основе типов объектов, уже существующих в языке. Новые типы объектов (в MatLab они называются классами) могут иметь собственные процедуры их преобразования (они определяют методы этого класса), причем новые процедуры могут быть вызваны с помощью обычных знаков арифметических операций и некоторых специальных знаков, которые применяются в математике.

Принципы сохранения значений переменных в MatLab наиболее близки к тем, которые присущи языку Fortran, а именно: все переменные являются локальными – действуют лишь в границах той программной единицы (процедуры, функции или главной, управляющей программы), где им присвоены некоторые конкретные значения. При переходе к выполнению другой программной единицы значения переменных предыдущей программной единицы либо теряются (если выполненная программная единица представляет собой процедуру или функцию), либо становятся недосягаемыми (если выполненная программа является управляющей). В отличие от языков Basic и Pascal в языке MatLab нет глобальных переменных, действие которых распространялось бы на все программные единицы. Но при этом язык MatLab обладает возможностью, которая отсутствует в других языках. Интерпретатор MatLab позволяет в одном и том же сеансе работы выполнять несколько самостоятельных программ, причем все переменные, используемые в этих программах, являются для них общими и образуют единое рабочее пространство. Это дает возможность более рационально организовывать сложные (громоздкие) вычисления по типу оверлейных структур.

Вышеуказанные особенности системы MatLab делают ее весьма гибкой и удобной в использовании вычислительной системой.

2. Простейшие вычисления в MATLAB

2.1. Арифметические вычисления

Для выполнения простейших арифметических вычислений можно воспользоваться командной строкой (**CommandWindow**). Например, для вычисления суммы 5+5 необходимо набрать в командной строке нужное выражение после значков приглашения '>>' и затем нажать '**Enter**'. В результате в командном окне будет выведен следующий результат:

```
>> 5 + 5
ans =
        10
>>
```

В данном случае результат будет записан в специальную переменную **ans** и программа будет готова для дальнейших вычислений.

Если необходим полученный выше результат, то можно воспользоваться переменной **ans**, которая уже была получена. Например, если нужно полученный результат возвести в квадрат (во вторую степень), то в командной строке нужно будет набрать **ans^2** и нажать '**Enter**'.

```
>> ans^2
ans =
       100
>>
```

Для ввода десятичных дробей используется точка. Например, к полученному ответу **ans**, который теперь уже равен 100, нужно прибавить 1.5:

```
>> ans + 1.5
ans =
    101.5000
>>
```

2.2. Формат вывода результата вычислений

Если необходимо получить результат с большей точностью, то можно поменять формат вывода результата. Для этого необходимо зайти в меню **File** и выбрать пункт **Preferences** (рис. 2.1).

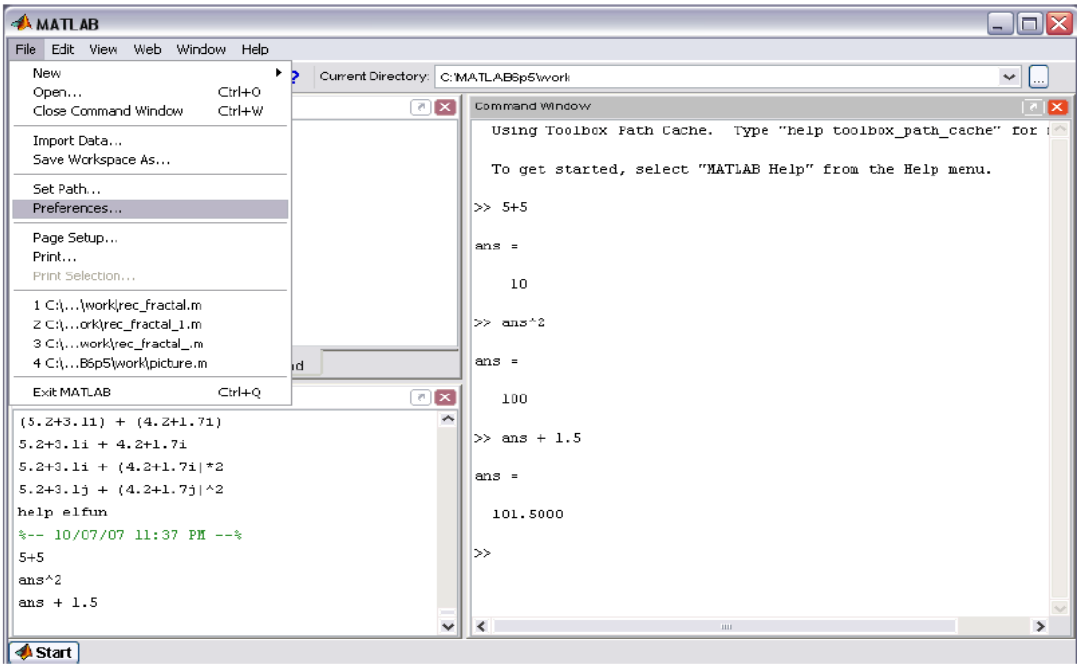


Рис. 2.1

Слева в раскрывающемся списке выбрать пункт **Command Window** и в панели справа в **Text display** найти пункт **Numeric format**, где в раскрывающемся списке и выбрать нужный формат (рис. 2.2).

Например, **short** – это короткий формат с плавающей точкой, где после десятичной точки отображаются только четыре цифры. Например, полученный выше результат будет выглядеть следующим образом:

```
>> ans + 1.5
ans =
       101.5000
>>
```

Однако если выбрать формат **long** – длинный формат с четырнадцатью цифрами после десятичной точки, то результат будет выглядеть следующим образом:

```
>> ans + 1.5
ans =
       1.015000000000000e+002
>>
```

Результат выведен в экспоненциальной форме.

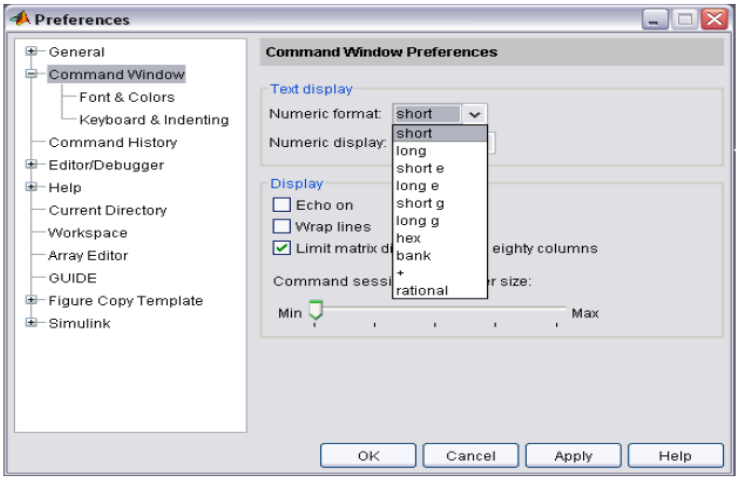


Рис. 2.2

2.3. Вычисление элементарных функций

Например, для вычисления выражения из элементарных функций необходимо ввести это выражение в командной строке и нажать ‘**Enter**’:

$$\sqrt{\sin 2.45\pi + \cos 3.78\pi}.$$

```
>> sqrt (sin (2.45*pi)+cos (3.78*pi))
ans =
        1.3260
```

Если необходимо это выражение изменить, то можно не писать заново все выражение в командной строке, а нажать клавишу ‘↑’ или ‘↓’, так как MatLab запоминает все, что было введено в командной строке.

Изменить предыдущее выражение, записанное выше, в том же месте не получится.

Если необходимо вызвать ранее введенное выражение или команду, то его можно найти в окне **Command History** и нажать два раза по выбранному выражению левой кнопкой мыши. В командной строке появится выбранное выражение, которое сразу же посчитается, как если бы был нажат ‘**Enter**’ (рис. 2.3).

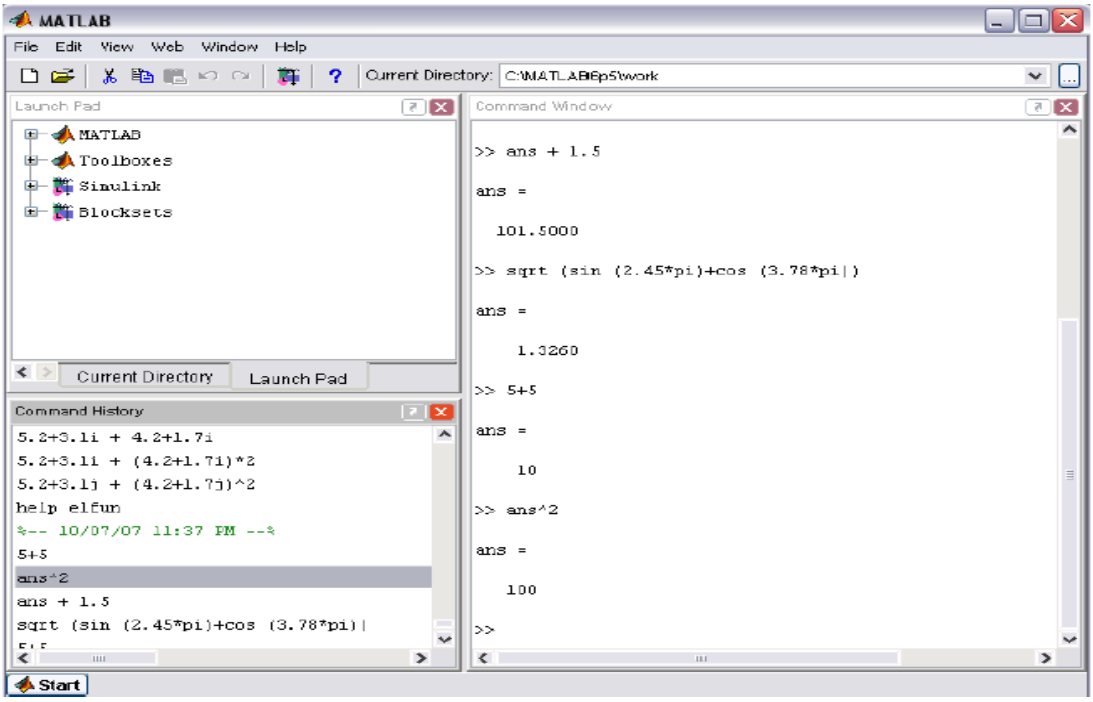


Рис. 2.3

Данное окно содержит также дату и время введенных выражений и команд, так что даже если компьютер вдруг выключится, то все, что набиралось в командной строке, сохранится.

Также в окне **Command History**при нажатии правой клавиши мыши можно скопировать выбранную команду или выражение (**Copy**); выполнить выбранную группу команд или вычислить группу выражений (**Evaluate Selection**), а также удалить выбранную группу (**Delete Selection**) или удалить всю историю (**DeleteEntireHistory**) (рис. 2.4).

В MatLab при делении на ноль получается бесконечность (**Inf**), но при этом выдается предупреждение. Причем если деление осуществляется отрицательного числа на ноль, то результатом будет минус бесконечность (–**Inf**).

Если будет деление ноля на ноль, то MatLab также выдаст предупреждение и в результате выдаст **NaN** (не число), см. рис. 2.5.

Также MatLab способен вычислять и комплексные числа. Например, при вычислении $\sqrt{-1}$ результатом будет

```
>> sqrt (–1.0)
ans =
        0 + 1.0000i
```

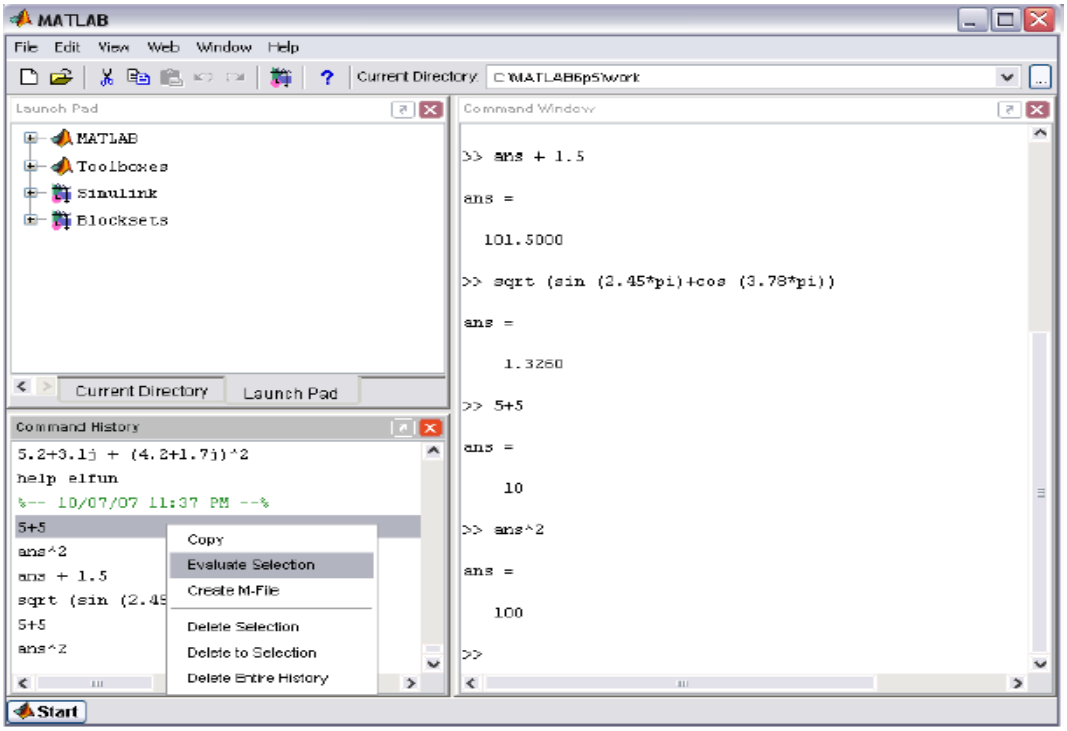


Рис. 2.4

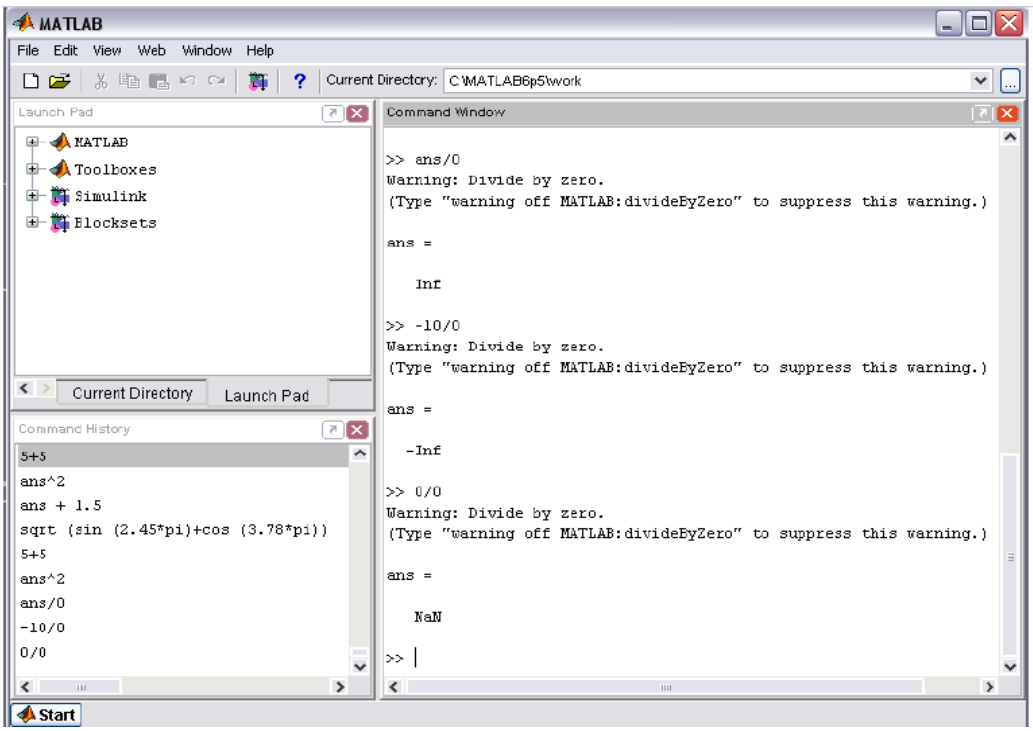


Рис. 2.5

Если необходимо, например, сложить два комплексных числа, то при наборе в командной строке можно писать буквы *i* или *j*. Например:

```
>> 5.2+3.1i + (4.2+1.7i)*2
ans =
        13.6000 + 6.5000i
```

или

```
>> 5.2+3.1j + (4.2+1.7j)^2
ans =
        19.9500 +17.3800i
```

Чтобы вывести все элементарные функции, необходимо набрать в командной строке команду **help elfun**.

2.4. Присвоение переменных

В MatLab знак равенства используется как *оператор присваивания*. Поэтому чтобы переменной *k* назначить число, необходимо написать в командной строке *k* = 3.15 и автоматически будет выведено значение *k*.

Однако вывод значений порой бывает не очень удобен, потому как он может загромождать визуально командное окно данными. Для этого в MatLab предусмотрено после присвоения в самом конце ставить точку с запятой. В результате число 3.15 присвоилось переменной *k* и запомнилось (рис. 2.6).

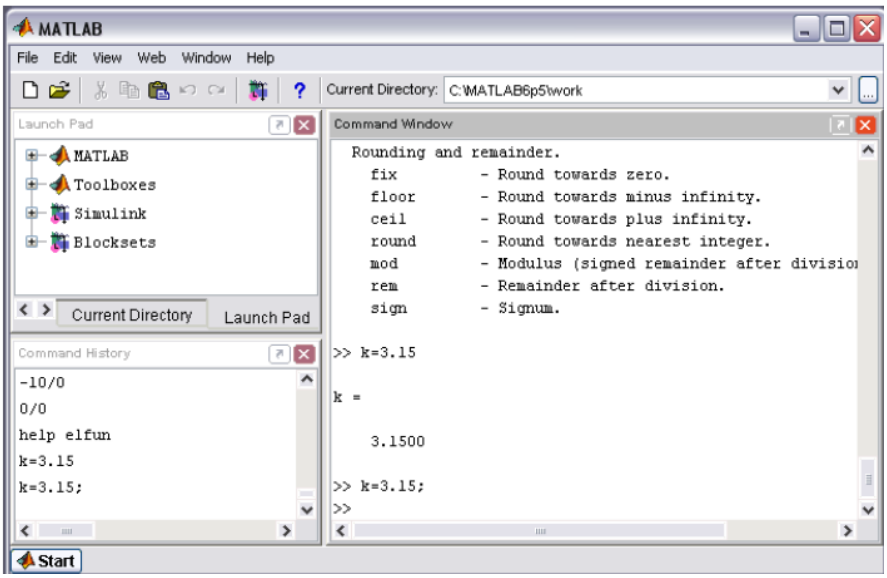


Рис. 2.6

Причем имя переменной может представлять любая последовательность букв. Также в имени переменной могут присутствовать и цифры, но тогда они должны обязательно начинаться с буквы.

Например, чтобы найти значение выражения $\dfrac{\dfrac{1}{\sin 0.6\pi}+\left(\dfrac{5}{\cos 1.2\pi}\right)^2}{\dfrac{5}{\cos 1.2\pi}-\sqrt{\dfrac{1}{\sin 0.6\pi}}}$, можно, воспользовавшись присвоением переменных, значениям $\dfrac{1}{\sin 0.6\pi}$ и $\dfrac{5}{\cos 1.2\pi}$ присвоить соответственно a иb. В результате получим более простое выражение:

```
>> a = 1/sin(0.6*pi);
>> b = 5*cos(1.2*pi);
>> c = (a+b^2)/(b–sqrt(a))
c = –3.4344
```

В окне**Command History** помимо введенных выражений и команд содержится также информация о том, когда и в какое время это выполнялось. Причем не каждая команда или выражение в отдельности, а в течение всего сеанса работы, начиная с момента, когда вы зашли в MatLab, и до того момента, когда вышли. MatLab во время этого сеанса помнит все присвоенные переменные, поэтому можно вывести ранее посчитанную или присвоенную переменную. Для этого необходимо в командной строке написать имя этой переменной и нажать ‘**Enter**’. Если это необходимо, то их можно использовать в дальнейших вычислениях (рис. 2.7).

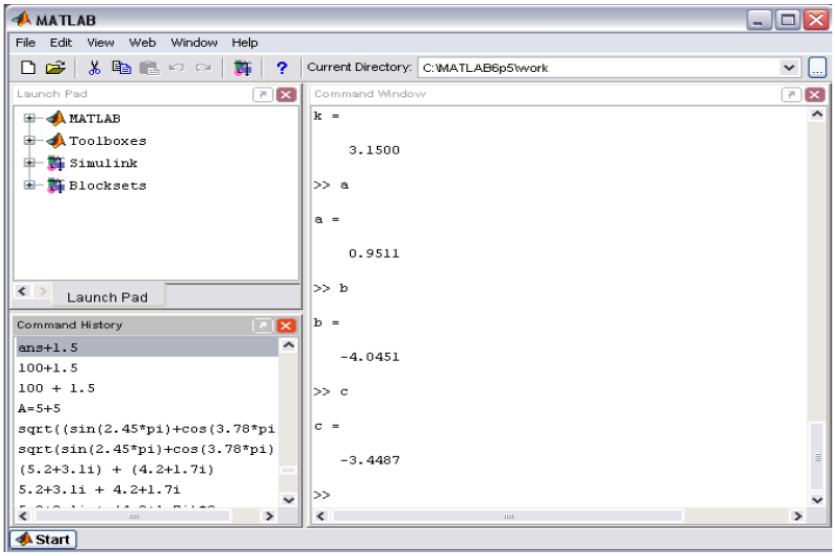


Рис. 2.7

При вводе, например, длинных формул или команд в командной строке можно разделить их на несколько строк. Для этого следует послабить три точки подряд и нажать ‘**Enter**’. При этом не появится знак приглашения ‘>>’ и необходимо просто продолжить набор формулы на следующей строке. Так можно записать и несколько строк (рис. 2.8).

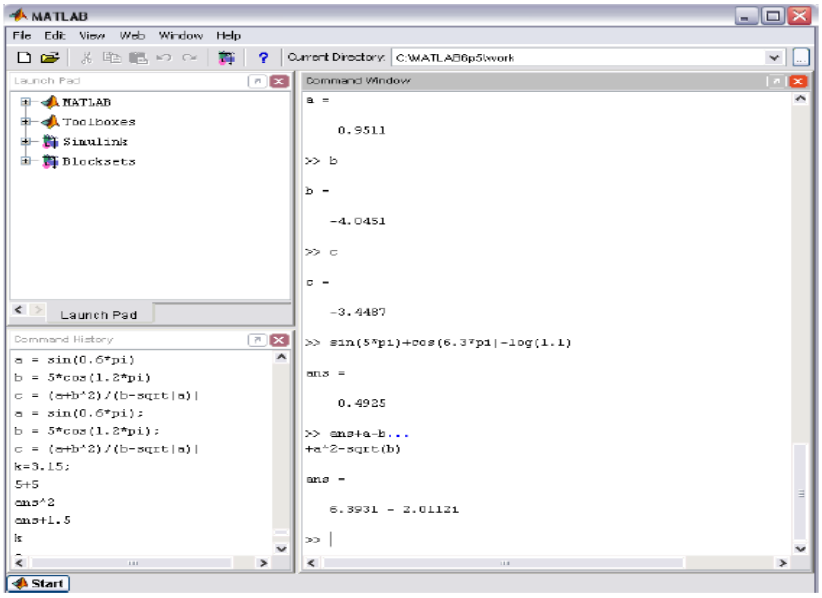


Рис. 2.8

2.5. Сохранение рабочей среды

Чтобы в дальнейшем можно было использовать значения всех переменных, которые использовались в данном сеансе работы, можно сохранить его и в последующем считать. Это можно сделать, написав в командной строке

```
>> save work01-01-08,
```

и при следующем сеансе работы считать ранее сохраненный файл

```
>> load work01-01-08
```

либо воспользоваться меню **File** пункт **Save Workspace As** (рис. 2.9).

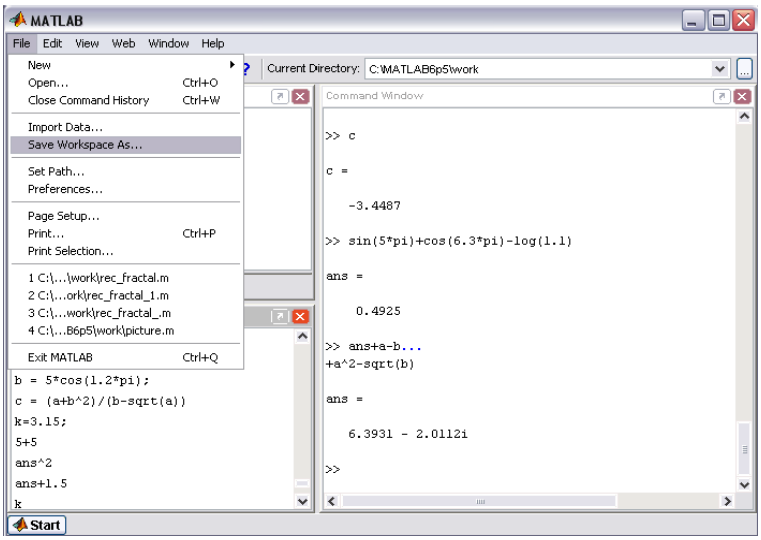


Рис. 2.9

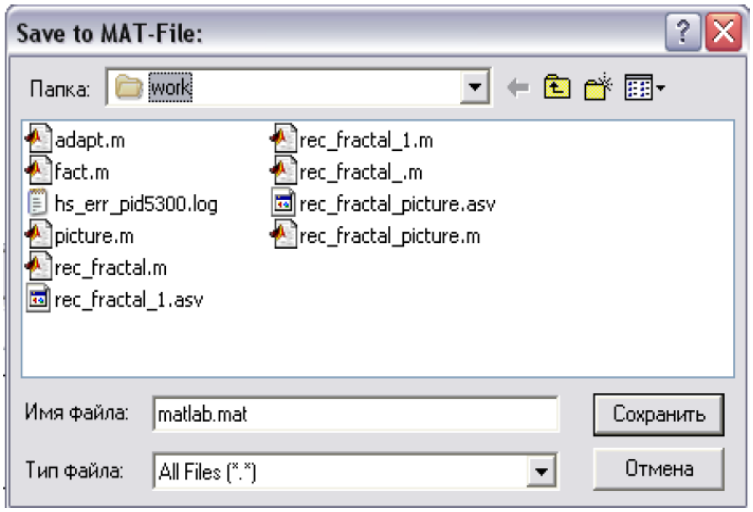


Рис. 2.10

После появления диалогового окна **Save to MAT-File** следует указать каталог и имя файла (рис. 2.10).

2.6. Просмотр переменных

В течение одного сеанса работы может использоваться большое количество переменных. Чтобы узнать, какие были объявлены ранее, можно либо набрать в командной строке команду **who** или **whos** (более подробная информация будетвыведена на экране в виде таблицы), либо увидеть их в **Workspace** в окне слева сверху (рис. 2.11). Панель инструментов окна**Workspace** позволяет удалить лишние переменные, сохранить и открыть рабочую среду (рис. 2.12).

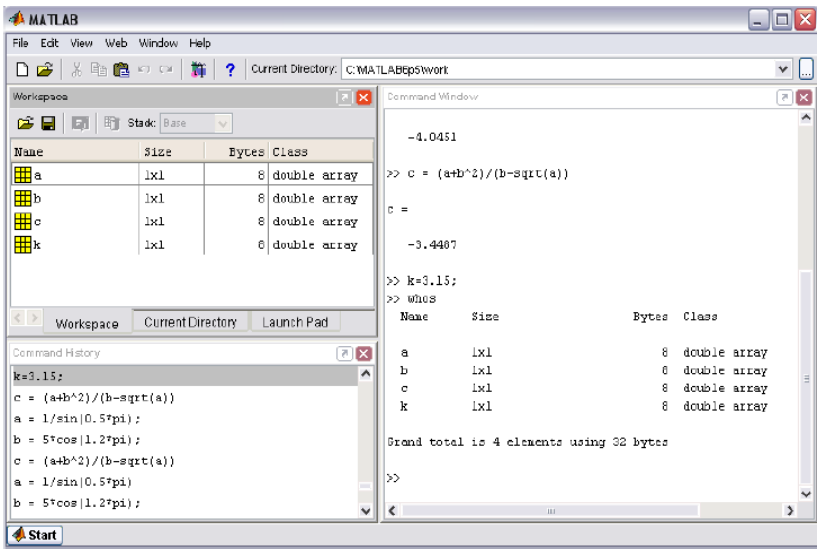


Рис. 2.11

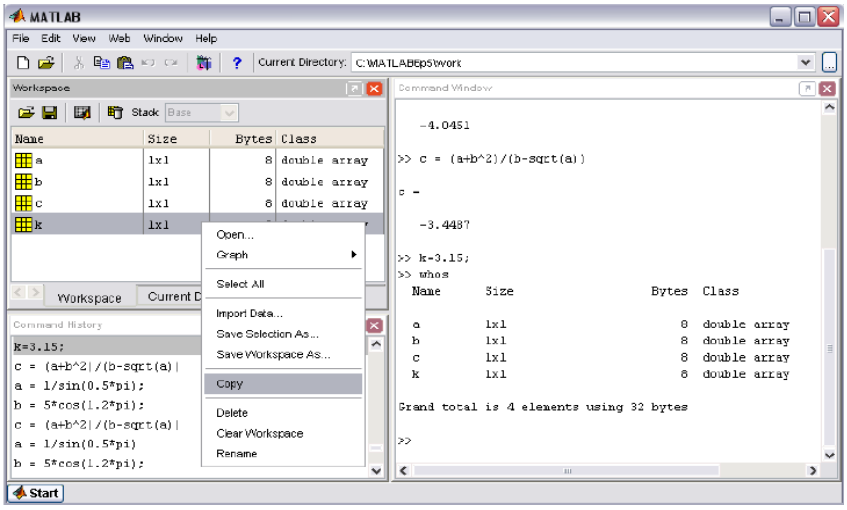


Рис. 2.12

Если необходимо очистить все переменные из памяти MatLab, то в командной строке необходимо написать команду **clear**; если только некоторые,то после команды через пробел указываем, какие именно. Например,

```
>> clear a b
```

3. Работа с массивами. Вектор-столбцы и вектор-строки

3.1. Основные определения

Массив – упорядоченная, пронумерованная совокупность однородных данных. У массива должно быть *имя*. Массивы могут отличаться по числу *измерений*:одномерные, двумерные, многомерные. *Размером* массива называют

число элементов, вдоль каждого из измерений. Доступ к элементам осуществляется при помощи *индекса*. В MatLab нумерация элементов массива начинается с единицы.

Положение элемента в массиве характеризуется двойным индексом (*m*,*n*). Первый индекс означает номер строки (*m)*, второй – номер столбца (*n*).

Матрица размера *1×n*, состоящая из одной строки, называется *вектор-строка*, а матрица размера *m×1*, состоящая из одного столбца, называется *вектор-столбец*.

3.2. Вектор-столбцы и вектор-строки

Ввод, сложение и вычитание векторов

Чтобы вычислить сумму векторов *a* и *b*: $a = \begin{pmatrix} 0.4 \\ 1.6 \\ 8.7 \end{pmatrix}$ $b = \begin{pmatrix} 5.1 \\ 3.9 \\ 7.6 \end{pmatrix}$, необходимо в командной строке ввести сначала вектор *a*, используя квадратные скобки, и между значениями вектора поставить точку с запятой ‘;’, а затем ввести вектор *b* таким же способом.

В конце каждой строки нужно поставить ‘;’, чтобы не загромождать промежуточными данными командное окно. Результат с следует вывести, т.е. в конце *c = a + b* не нужно ставить точку с запятой ‘;’.

Узнать размер, например, массива ***a*** можно при помощи встроенной функции **size**.

Введенные значения можно использовать в дальнейшем для последующих вычислений. Например, найти квадратный корень из элементов *b* (рис. 3.1).

Для вычислений вектор-строк необходимо записать их в командной строке также в квадратных скобках, но между значениями поставить запятые или пробелы (рис. 3.2).

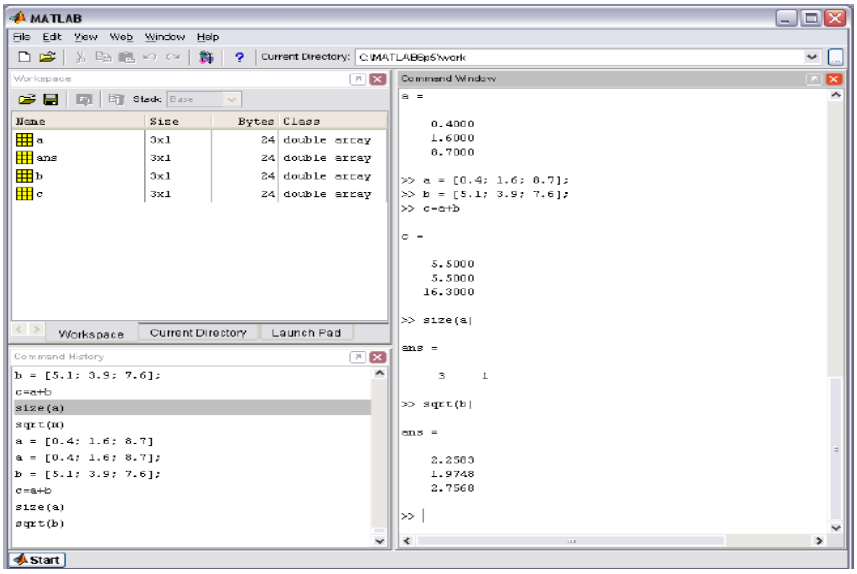


Рис. 3.1

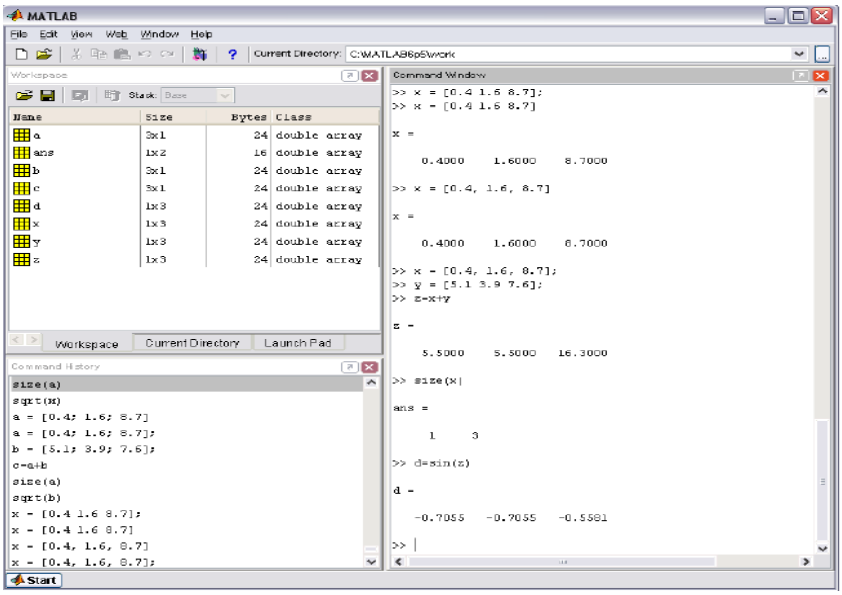


Рис. 3.2

Из нескольких вектор-столбцов (вектор-строк) можно составить один, используя квадратные скобки и разделяя исходные вектор-столбцы точкой с запятой (рис. 3.3).

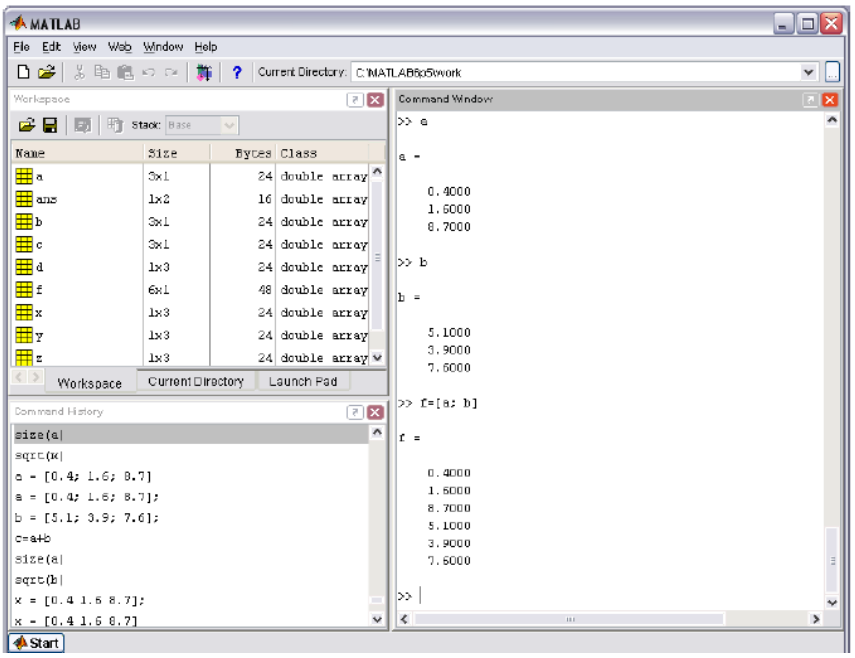


Рис. 3.3

Для сцепления вектор-строк также применяются квадратные скобки, но сцепляемые вектор-строки отделяются пробелами или запятыми (рис. 3.4).

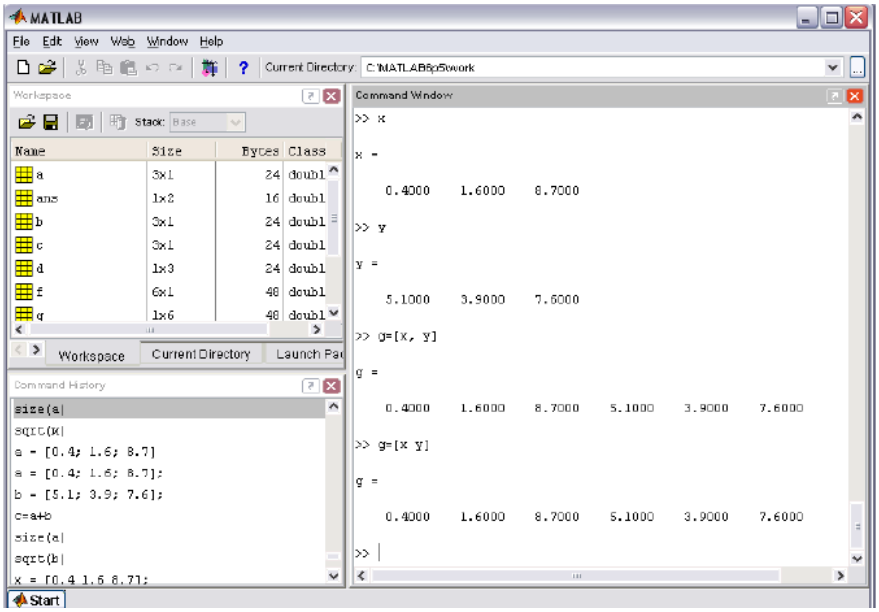


Рис. 3.4

Обращение к элементам вектора

Для того чтобы вывести лишь один элемент из вектор-столбца или вектор-строки, необходимо написать его имя и далее в круглых скобках указать тот элемент, который необходимо вывести. Например, необходимо вывеститретий элемент вектор-строки k и третий элемент вектор-столбца g.

Если нам нужно заменить какой-то из элементов на другое значение, то необходимо после *k*(3) поставить еще знак присваивания и ввести новое значение (рис. 3.5)

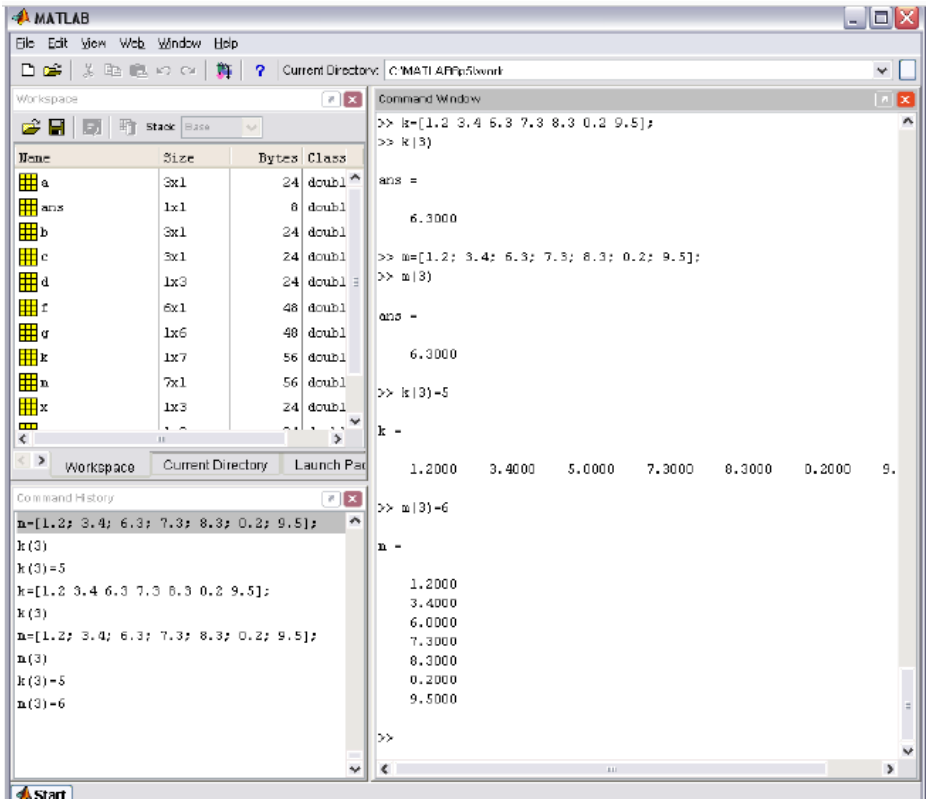


Рис. 3.5

Также из введенных ранее значений можно сформировать новый вектор-столбец или вектор-строку (рис. 3.6).

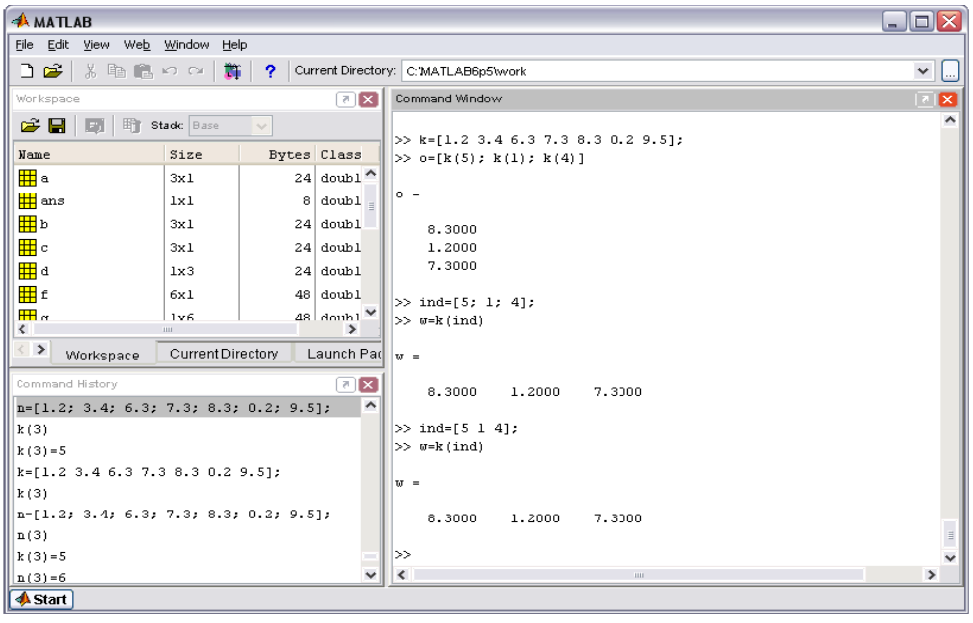


Рис. 3.6

Для помещения определенных элементов вектора в другой вектор в заданном порядке служит *индексация при помощи вектора*. Запись в массив *w* пятого, первого и четвертого элементов *k* производится, как указано на рис. 3.6.

Для обращения к блокам последовательно расположенных элементов вектор-столбца или вектор-строки служит *индексация при помощи знака двоеточия*. Предположим, что в массиве *k*, соответствующем вектор-строке из семи элементов, требуется заменить нулями элементы со второго по пятый.

Присваивание *k* (2:5) = 0 эквивалентно последовательности команд *k*(2) = 0; *k* (3) = 0; *k* (3) = 0; *k* (4) = 0; *k* (5) = 0 (рис. 3.7).

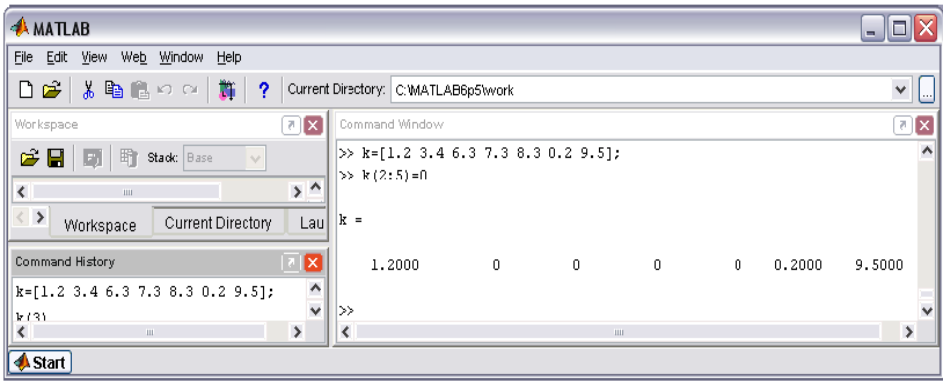


Рис. 3.7

Индексация при помощи двоеточия оказывается удобной при выделении части из большого объема данных в новый массив (рис. 3.8).

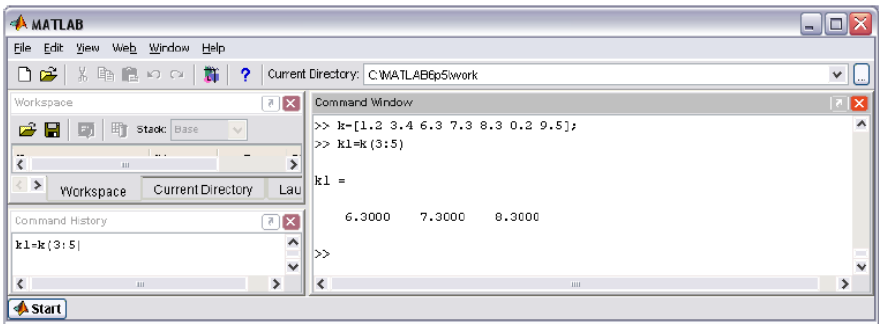


Рис. 3.8

Попробуйте составить массив *k*2, содержащий элементы *k*, кроме четвертого. Используйте двоеточие и сцепление строк.

Применение функций обработки данных к векторам

Перемножение элементов вектор-столбца или вектор-строки осуществляется при помощи функции **prod**. Функция **sum** предназначена для

суммирования элементов вектора. Для нахождения минимума и максимума из элементов вектора служат встроенные функции **min** и **max** (рис. 3.9).

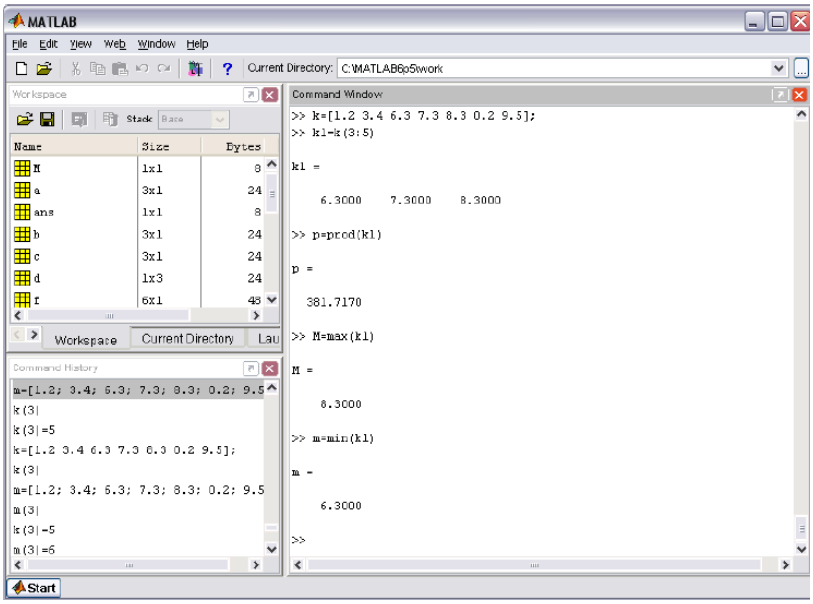


Рис. 3.9

Часто необходимо знать не только значение минимального или максимального элемента в массиве, но и его индекс (порядковый номер). Вызовите, например, функцию **min** с двумя выходными аргументами (рис. 3.10).

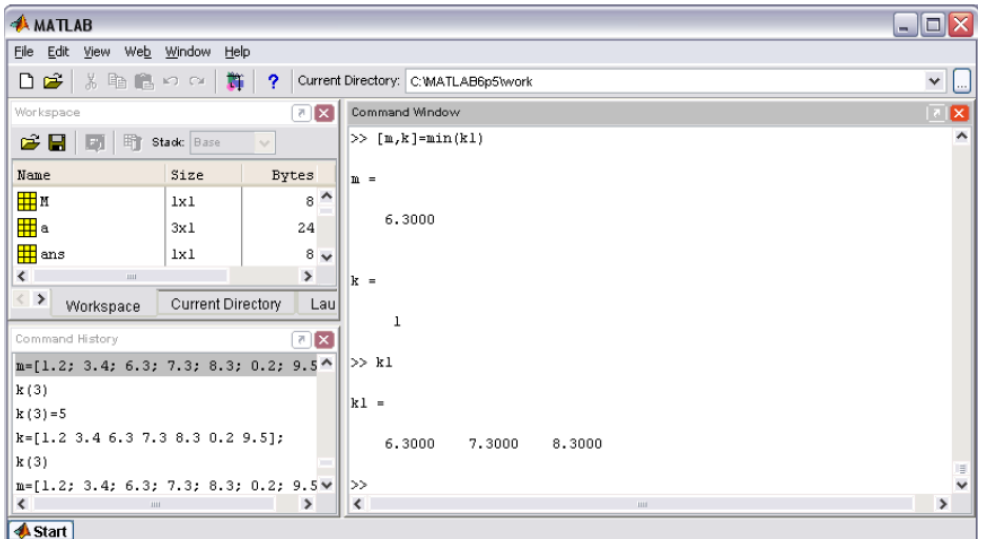


Рис. 3.10

В результате переменной *m* будет присвоено значение минимального элемента массива *k*1, а номер минимального элемента занесен в переменную *k*. Для того чтобы узнать, как именно можно вызвать функцию, следует набрать в командной строке **help** и имя функции.

В число основных функций для работы с векторами входит функция упорядочения вектора по возрастанию его элементов **sort**.

Упорядочение элементов в порядке возрастания их модулей производится с привлечением функции **abs** (рис. 3.11).

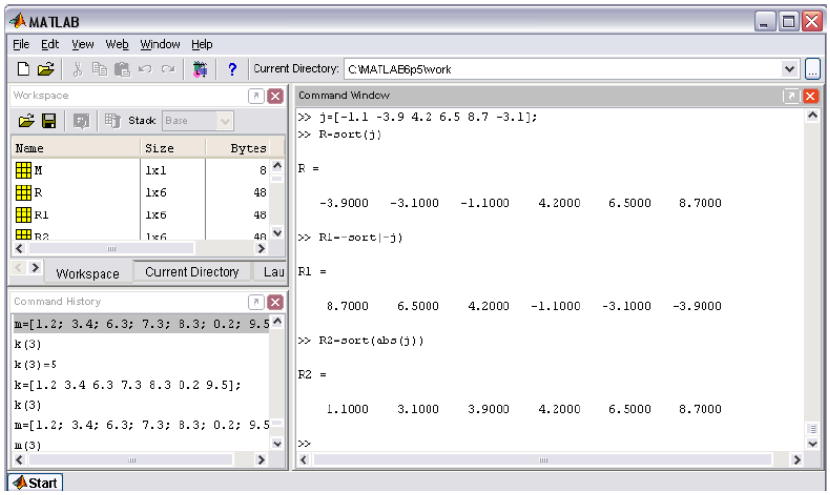


Рис. 3.11

Вызов **sort** с двумя выходными аргументами приводит к образованию массива индексов соответствия элементов упорядоченного и исходного массивов (рис. 3.12).

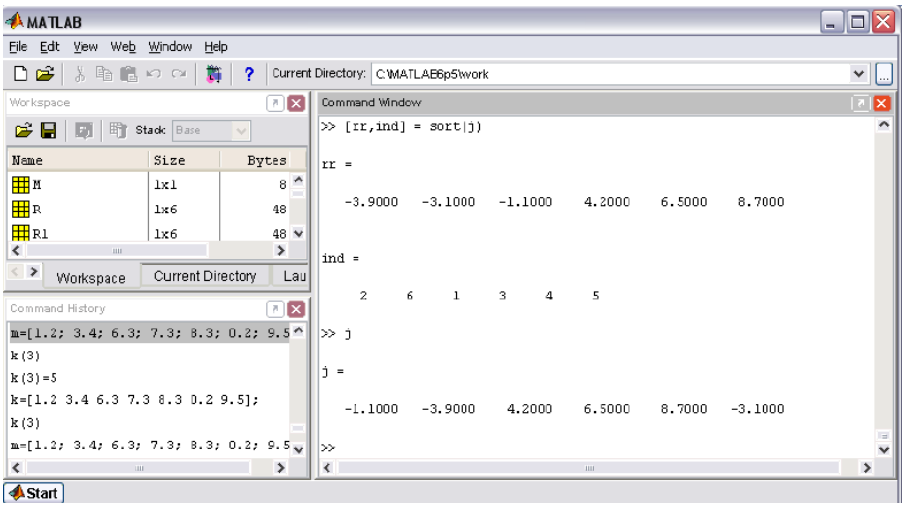


Рис. 3.12

Если аргументом функций **max** и **min** является вектор, состоящий из комплексных чисел, то результатом является максимальный и минимальный по модулю элемент.

Самостоятельно о функциях обработки данных можно узнать, набрав в командной строке команду **help datafun**.

Поэлементные операции с векторами

При поэлементном вычислении векторов или матриц используется точка ‘.’. Например операция .* приводит к поэлементному умножению векторов одинаковой длины. При помощи .^ осуществляется поэлементное возведение в степень. Аналогично при делении и других математических операциях. При этом не нужно ставить между точкой и любой другой математической операцией пробелов, иначе MatLab будет выдавать ошибку (рис. 3.13).

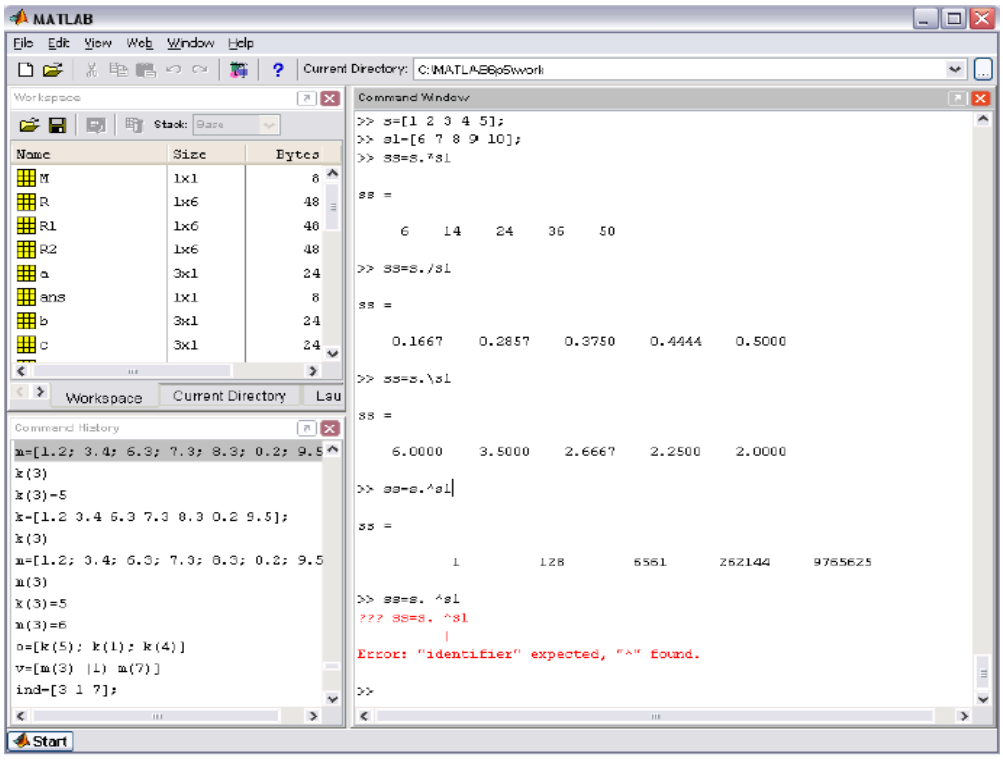


Рис. 3.13

Деление соответствующих элементов векторов одинаковой длины выполняется с использованием операции ./

Обратное поэлементное деление (деление элементов второго вектора на соответствующие элементы первого) осуществляется при помощи операции .\

Точка в MatLab используется не только для ввода десятичных дробей, но и для указания того, что деление или умножение массивов одинакового размера должно быть выполнено поэлементно.

К поэлементным относятся и операции с вектором и числом. Сложение вектора и числа не приводит к сообщению об ошибке. MatLab прибавляет число к каждому элементу вектора. То же самое справедливо и для вычитания.

Делить при помощи знака / можно вектор на число, но не наоборот. Попытка деления числа на вектор приводит к сообщению об ошибке. Это связано с тем, что операция / в MatLab предназначена, в частности, для решения систем линейных алгебраических уравнений. Если требуется разделить число на каждый элемент вектора и записать результат в новый вектор, то следует использовать операцию ./ (рис. 3.14).

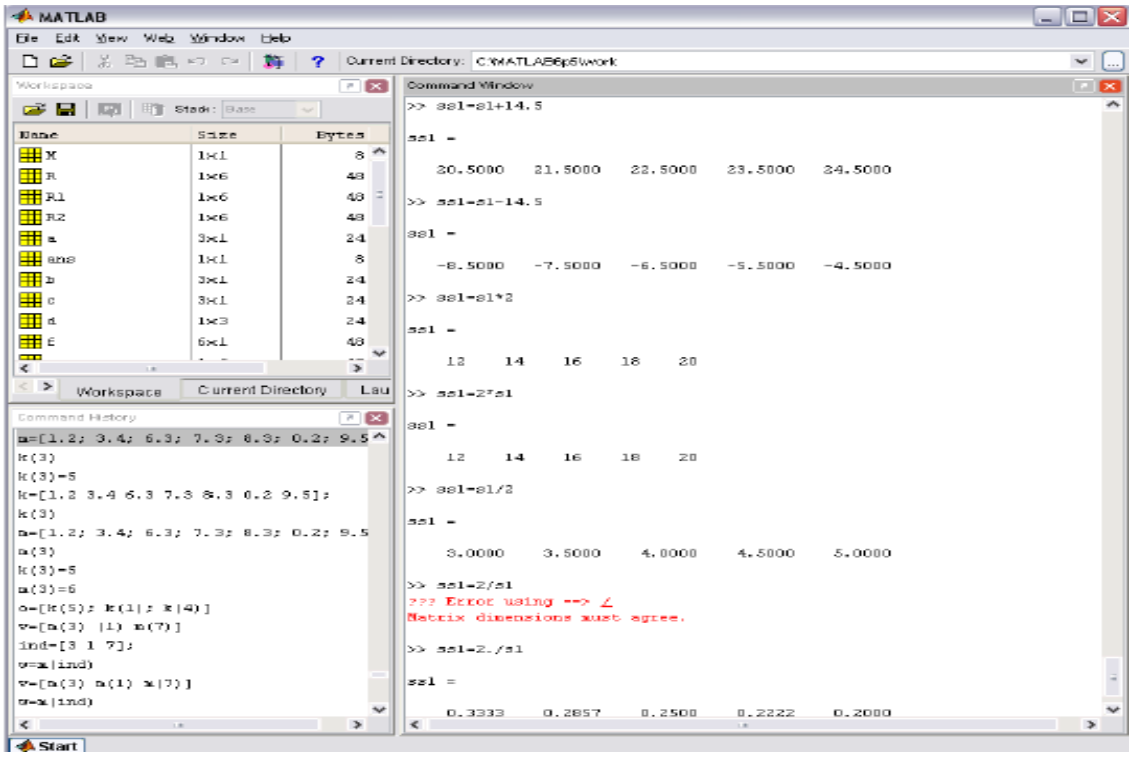


Рис. 3.14

Все вышеописанные операции применимы как к вектор-строкам, так и к вектор-столбцам.

Разберем, как правильно транспонировать и вычислять сопряженные векторы в MatLab. Для вектор-столбца *u* с тремя комплексными элементами(в частности, и с вещественными) *сопряженный* к нему *u** определяется как вектор-строка из его комплексно-сопряженных элементов, а *транспонированныйu*$^{\mathrm{T}}$ – просто как вектор-строка из его элементов, например:

$$u=\begin{bmatrix}2+3i\\1-2i\\3+2i\end{bmatrix}\quad u^{*}=\begin{bmatrix}2-3i & 1+2i & 3-2i\end{bmatrix}\quad u^{T}=\begin{bmatrix}2+3i & 1-2i & 3+2i\end{bmatrix}.$$

Аналогично определяется сопряжение и транспонирование для вектор-строки, приводящее к вектор-столбцу. Для векторов, состоящих только из действительных чисел, операции сопряжения и транспонирования совпадают.

Для нахождения сопряженного вектора в MatLab используется апостроф, а для транспонирования следует применять точку с апострофом (рис. 3.15).

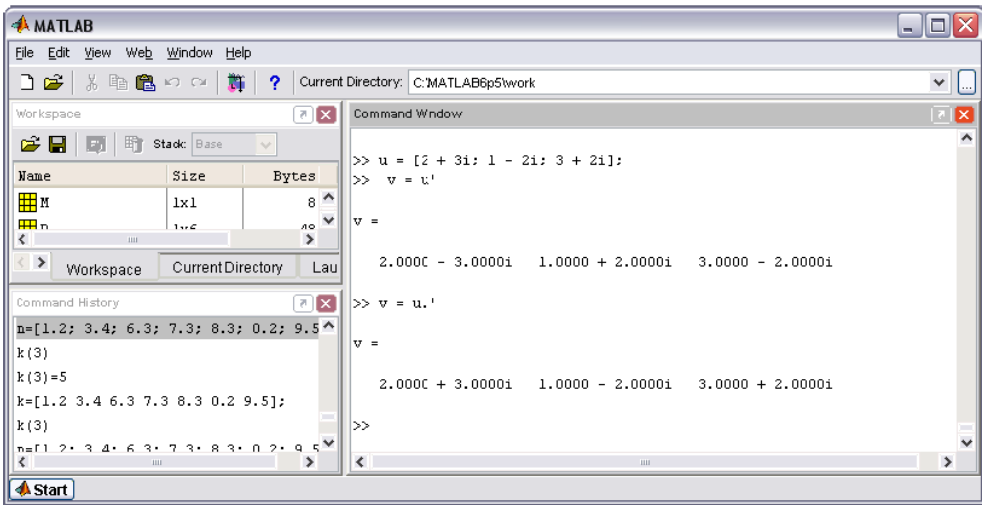


Рис. 3.15

Операции .' и ' над вещественными векторами приведут к одинаковым результатам.

4. Работа с массивами

4.1. Построение таблицы значений функции

Если имеется сравнительно небольшое количество значений функций, то саму функцию удобнее отобразить в виде таблицы.

Например, необходимо вывести таблицу значений функции *y(x)* в точках 0.1, 0.3, 0.4, 0.6, 1.2, 1.5, 2.6:

$$y(x) = \frac{e^{-2x}}{\log x + 5x} + \cos x^2 \cdot \sin 3x$$

Сначала следует записать данные значения в вектор-строку *x*, а затем вычислить значения функции *y(x)* от каждой этой точки, поэтому нужно в записи самой функции использовать поэлементные операции (рис. 4.1).

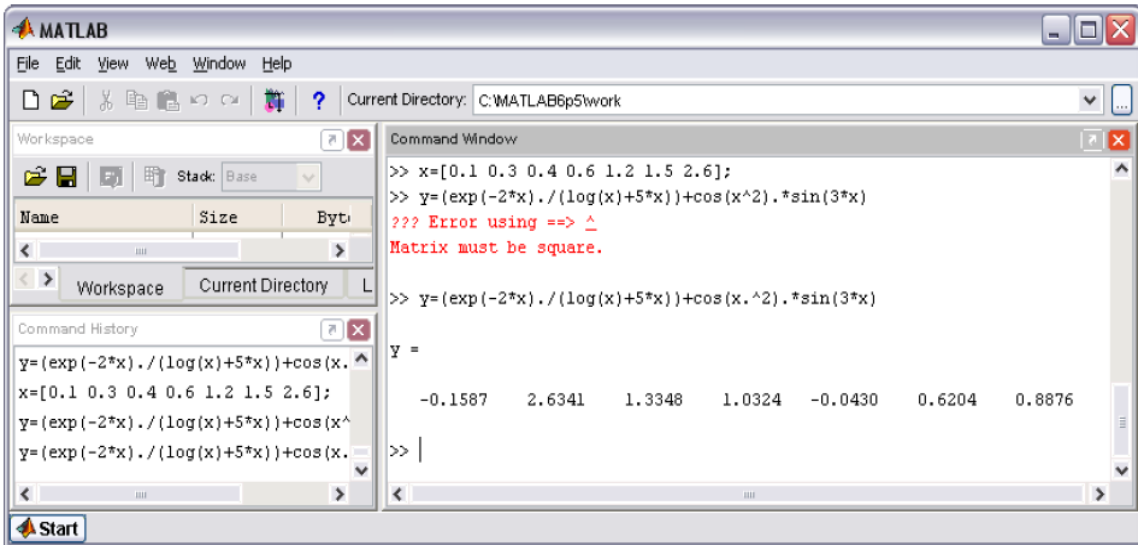


Рис. 4.1

Обратите внимание, что при попытке использования операций возведения в степень Λ, деления / и умножения ∗ (которые не относятся к поэлементным) выводится сообщение об ошибке уже при возведении *cos(x)* в квадрат. Дело в том, что в MatLab операции ∗ и Λ применяются для перемножения матриц соответствующих размеров и возведения квадратной матрицы в степень.

Если требуется вывести значение функции на каком-то отрезке значений, например от 1 до 3, то значения *x* можно записать следующим образом:

```
>> x = [1:3]
x =
     1     2     3
```

В MatLab, если не указан шаг, он по умолчанию равен единице. Номожно самим указать его (рис. 4.2). Например, значения *x* начинаются с 1 и заканчиваются 3, а 0.2 – это шаг, который указывается посередине:

```
>> x = [1:0.2:3]
x =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    2.2000 2.4000    26000    2.8000    3.0000
```

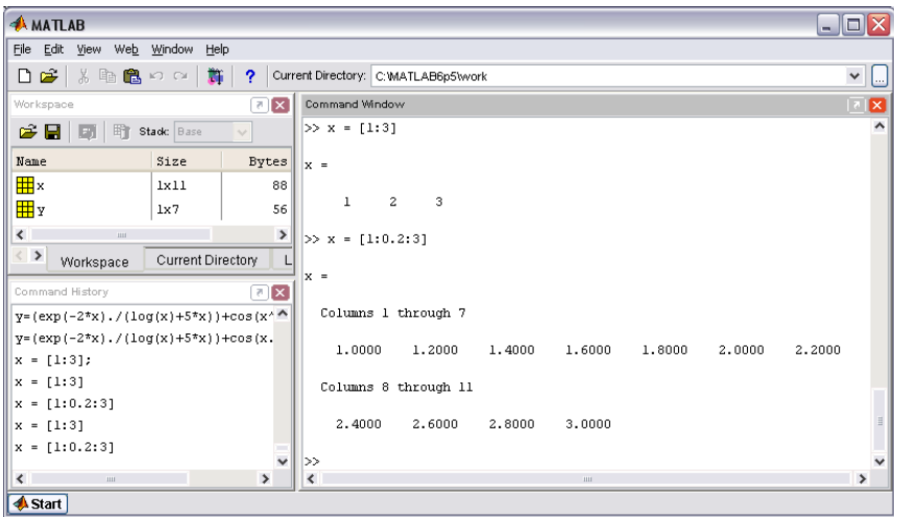


Рис. 4.2

Если задать шаг 0.2 и при этом *x* задать не до 3, а до 2.9, то последним значением будет 2.8 (рис. 4.3).

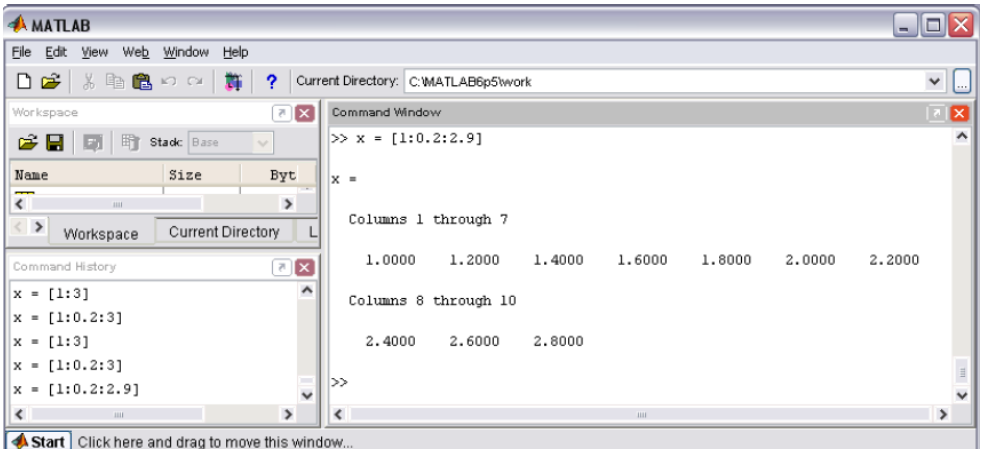


Рис. 4.3

Шаг может быть и отрицательным, но при этом начальное значение должно быть больше конечного. Например, *x* с шагом –0.2 будет выглядеть, как показано на рис. 4.4.

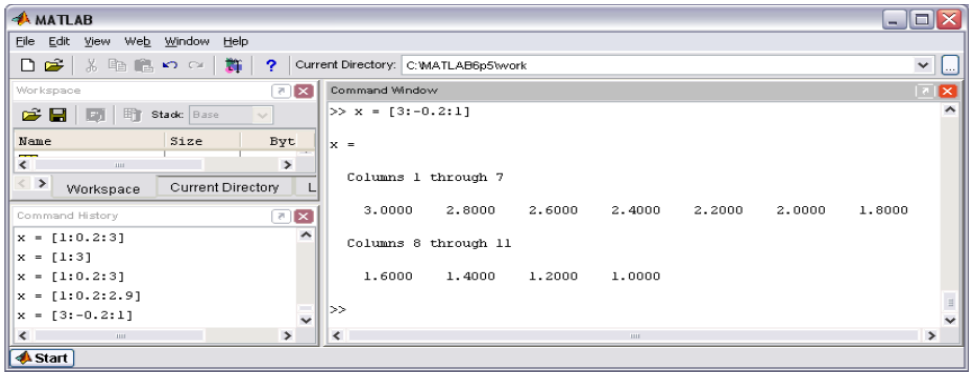


Рис. 4.4

Чтобы заполнить вектор-столбец значениями от 1 до 3 с шагом 0.2, необходимо в конце поставить знак транспонирования (апостроф), см. рис. 4.5.

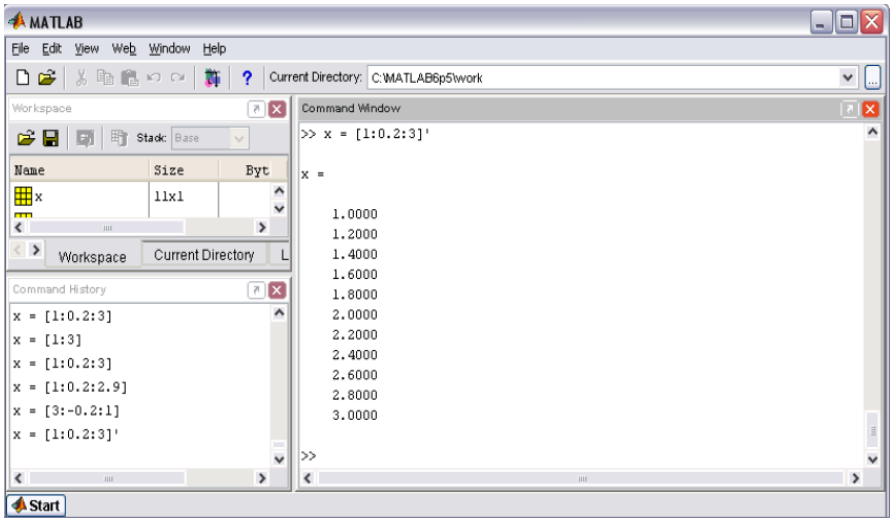


Рис. 4.5

Обратите внимание, что элементы вектора, заполняемого при помощи двоеточия, могут быть только вещественные, поэтому для транспонирования можно использовать апостроф вместо точки с апострофом.

Выведите таблицу значений функции $y(x) = e^{-x} \log 5x$ на промежутке от 1 до 5 с шагом 0.1 (рис. 4.6).

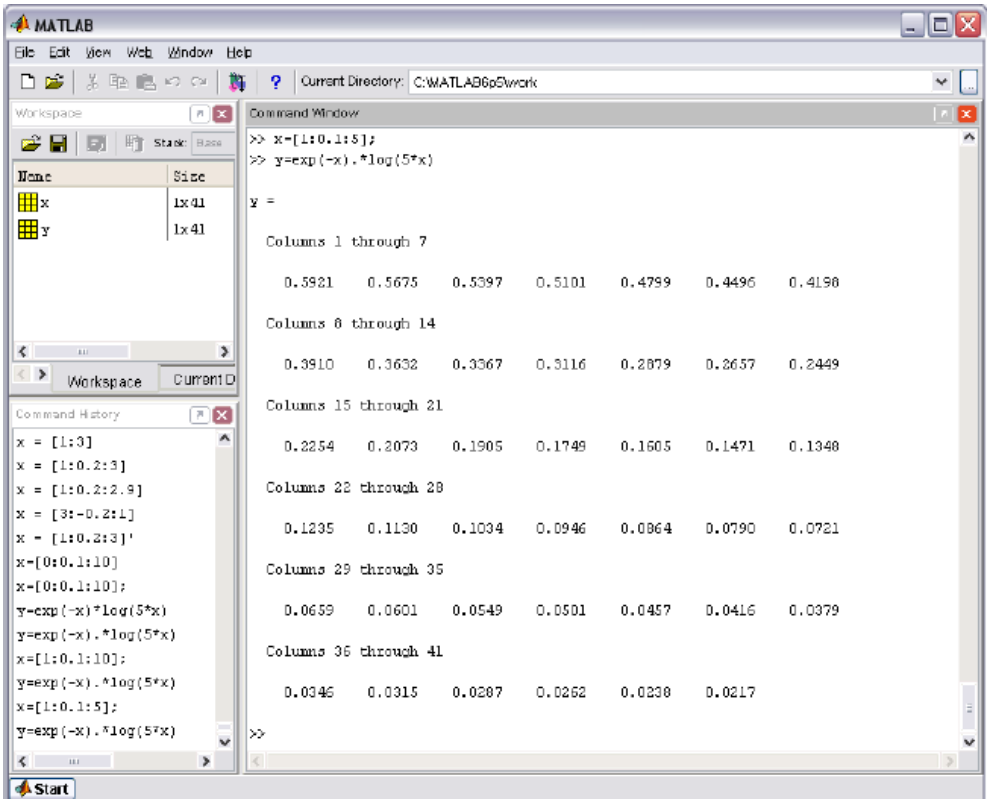


Рис. 4.6

Вектор-строки **x** и **y** состоят из сорока одного элемента, не помещаютсяна экране в одну строку и выводятся по частям. Так как ***x*** и ***y*** хранятся в двумерных массивах размерностью один на сорок один, то выводятся по столбцам, каждый из которых состоит из одного элемента. Сначала выводятся столбцы с первого по седьмой (Columns 1 through 7), затем – с восьмого по четырнадцатый (Columns 8 through 14) и так до сорок первого (Columns 36 through 41). Более наглядным и удобным является графическое представление функции.

4.2. Построение графиков функции одной переменной

Построим график функции одной переменной на примере функции $y(x)=e^{-x}\cdot\sin 10x,$ определенной на отрезке [0, 5]. Вывод отображения функции в виде графика состоит из следующих этапов:

1. Задание вектора значений аргумента *x*.
2. Вычисление вектора *y* значений функции *y(x)*.
3. Вызов команды **plot** для построения графика.

Команды для задания вектора "*x*" и вычисления функции лучше завершать точкой с запятой для подавления вывода в командное окно их значений(после команды **plot** точку с запятой ставить необязательно, так как она ничего не выводит в командное окно). Не забудьте использовать поэлементное умножение .* (рис. 4.7).

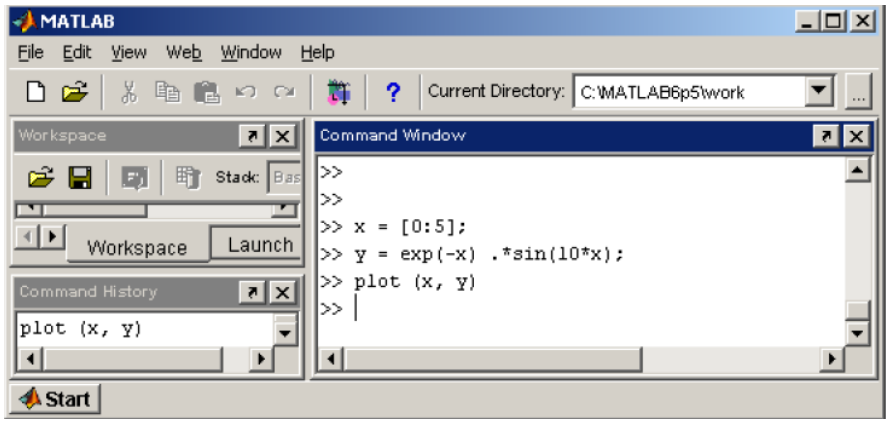


Рис. 4.7

После выполнения команд на экране появляется окно **FigureNo. 1** с графиком функции. Окно содержит меню, панель инструментов и область графика (рис. 4.8).

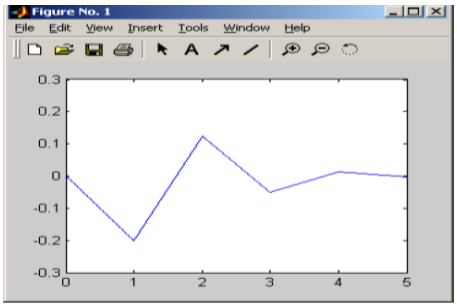


Рис. 4.8

Для построения графика функции должны быть определены два вектора одинаковой размерности, например *x* и *y* (рис. 4.9).

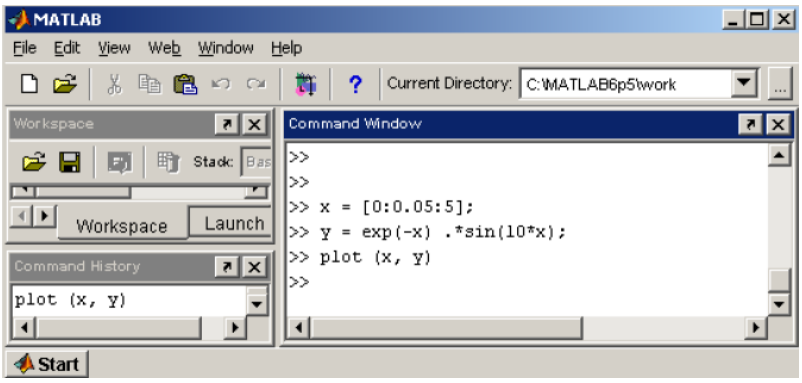


Рис. 4.9

Соответствующий массив *x* содержит значения аргументов, а *y* – значения функции от этих аргументов. Команда **plot** соединяет точки с координатами **(*x(i)*, *y(i)*)** прямыми линиями, автоматически масштабируя оси для оптимального расположения графика в окне. Построенный график функции не должен иметь изломов, так как сама функция гладкая. Для точного построения графика вычислите функцию в большем числе точек на отрезке [0, 5], т. е. задайте меньший шаг при вводе вектора *x*. Не забудьте, что можно занести в командную строку введенные ранее команды при помощи клавиш <↑>, <↓>, затем отредактировать их, нажав <**Enter**>. Следующие команды приводят к построению графика функции в виде плавной кривой (рис. 4.10).

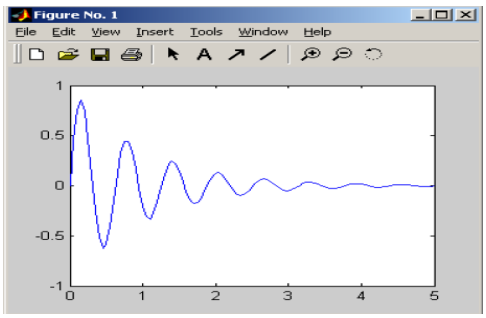


Рис. 4.10

Сравнение нескольких функций удобно производить, отобразив их графики на одних осях. Например, постройте на отрезке [–1, –0.3] графики функций $f(x)=\sin\frac{1}{x^2}$, $g(x)=\sin\frac{1.2}{x^2}$ при помощи последовательности команд, приведенных на рис. 4.11.

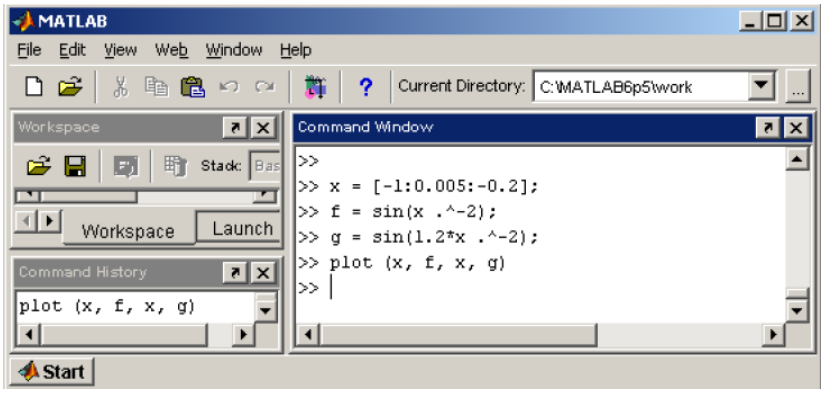


Рис. 4.11

Получившиеся графики дают наглядное представление о поведении исследуемых функций (рис. 4.12).

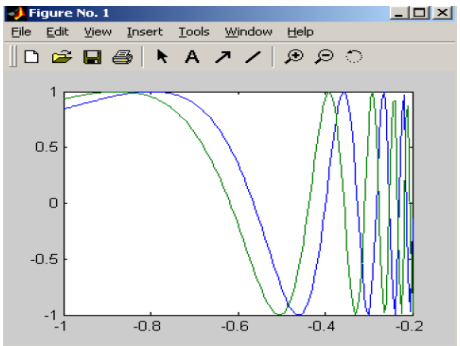


Рис. 4.12

MatLab выводит графики разным цветом. Монохромный принтер напечатает графики различными оттенками серого цвета, что не всегда удобно. Команда **plot** позволяет легко задать стиль и цвет линий, например:

```
>> plot (x, f, 'k–', x, g, 'k:')
```

осуществить построение первого графика сплошной черной линией, а второго – черной пунктирной. Аргументы **'k–'** и **'k:'** задают стиль и цвет первой и второй линий. Здесь **k** означает черный цвет, а дефис или двоеточие – сплошную или пунктирную линию.

4.3. Умножение векторов

Вектор можно умножить на другой вектор скалярно (это произведение ещё называют внутренним), векторно или образовать так называемое внешнее произведение. Результатом скалярного произведения является число, векторного – вектор, а внешнего – матрица.

Скалярное произведение

Скалярное произведение векторов ***a*** и ***b*** длины ***N***, состоящих из действительных чисел, определяется формулой

$$a \cdot b = \sum_{k=1}^{N} a_k \cdot b_k$$

Для вычисления скалярного произведения необходимо просуммировать компоненты вектора, полученного в результате поэлементного умножения ***a*** на ***b***, т. е. надо использовать функцию **sum** и поэлементное умножение. Например, для скалярного произведения векторов $a = \begin{bmatrix} 1.2 \\ -3.2 \\ 0.7 \end{bmatrix}$, $b = \begin{bmatrix} 4.1 \\ 6.5 \\ -2.9 \end{bmatrix}$ приведена требуемая последовательность команд (рис. 4.13).

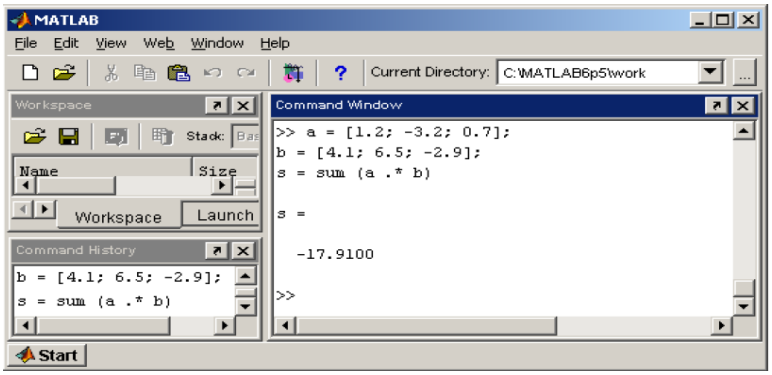


Рис. 4.13

Например, нахождение длины (модуль) вектора ***a:***

$|a| = \sqrt{a \cdot a}$ показано на рис. 4.14.

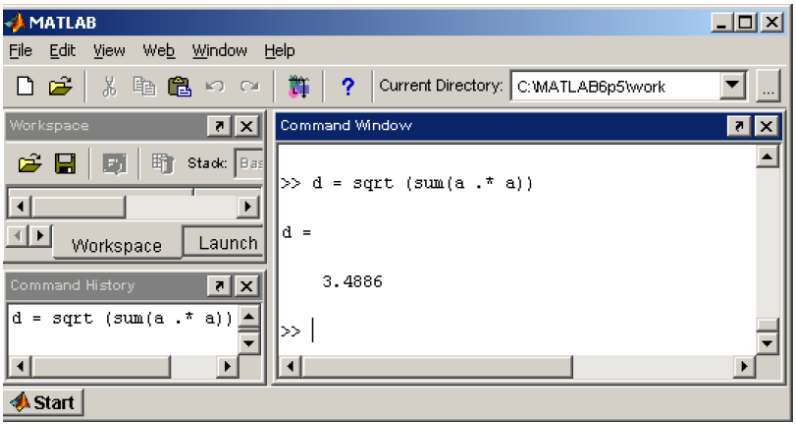


Рис. 4.14

Векторное произведение

Векторное произведение $a \times b$ определено только для векторов из трехмерного пространства, т.е. состоящих из трех элементов. Результатом также является вектор из трехмерного пространства. Для вычисления векторного произведения служит функция **cross** (рис. 4.15).

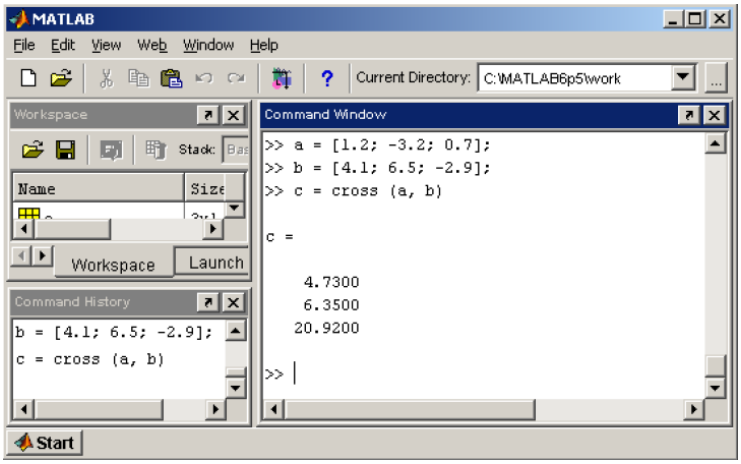


Рис. 4.15

Для тренировки попробуйте вычислить $a \times b + b \times a$. Если получился вектор, состоящий из нулей, то вы все проделали правильно, так как для любых векторов выполняется свойство $a \times b = -b \times a$.

Смешанное произведение векторов a, b, c определяется по формуле $abc = a \cdot (b \times c)$. Модуль смешанного произведения векторов равен объему параллелепипеда, построенного на этих векторах (рис. 4.16).

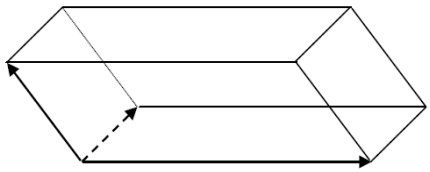


Рис. 4.16

Найдите объем параллелепипеда, если

$$a = \begin{bmatrix} 3.5 \\ 0 \\ 0 \end{bmatrix}, \quad b = \begin{pmatrix} 0.5 \\ 2.1 \\ 0 \end{pmatrix}, \quad c = \begin{pmatrix} -0.2 \\ -1.9 \\ 2.8 \end{pmatrix}.$$

Правильные действия показаны на рис. 4.17.

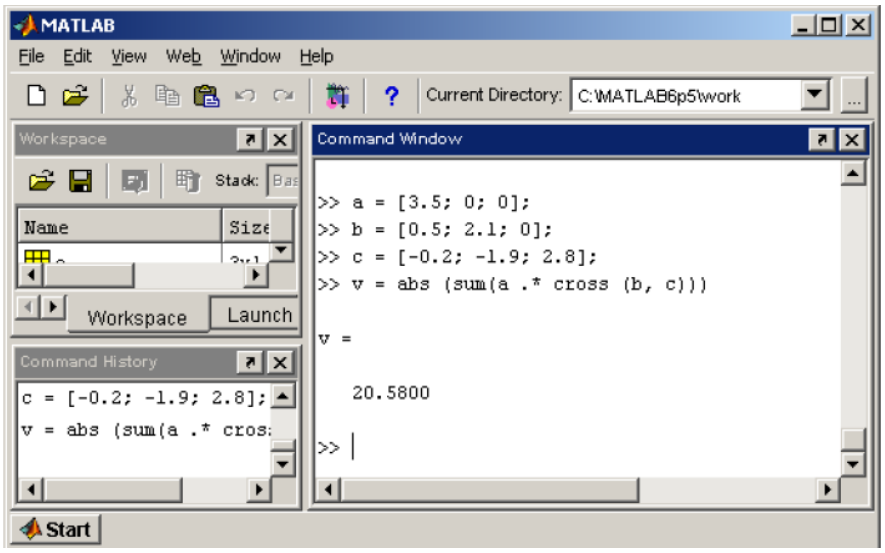


Рис. 4.17

Внешним произведением векторов $a = (a_j)_{j=1,\ldots,N}$ и $b = (b_k)_{k=1,\ldots,M}$ называется матрица $C = (c_{jk})_{j=1,\ldots,N,k=1,\ldots,M}$ размера $N \times M$, элементы которой вычисляются по формуле $c_{jk} = a_j b_k$.

Вектор-столбец **a** в MatLab представляется в виде двумерного массива размера **N** на один. Вектор-столбец **b** при транспонировании переходит в вектор-строку размера один на **M**. Вектор-столбец и вектор-строка есть матрицы, у которых один из размеров равен единице. Фактически, $C = ab^T$, где умножение происходит по правилу *матричного произведения*. Для

вычисления матричного произведения в MatLab используется оператор “звездочка” (рис.4.18).

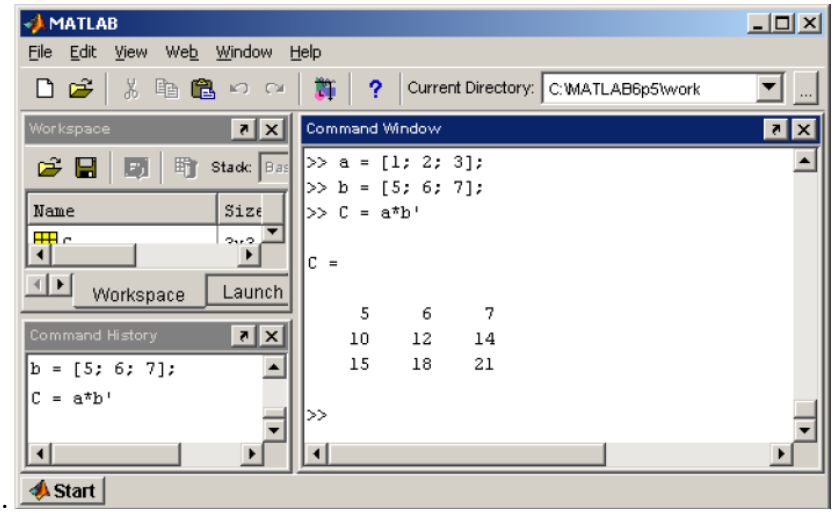


Рис. 4.18

5. Двумерные массивы и матрицы

5.1. Ввод матриц, простейшие операции

Различные способы ввода

Вводить небольшие по размеру матрицы удобно прямо из командной строки. Введите матрицу размерностью два на три:

$$A = \begin{pmatrix} 3 & 1 & -1 \\ 2 & 4 & 3 \end{pmatrix}.$$

Для хранения матрицы используйте двумерный массив с именем A. При вводе учтите, что матрицу A можно рассматривать как вектор-столбец из двух элементов, каждый из которых является вектор-строкой длиной три. Следовательно, строки при наборе отделяются точкой с запятой (рис. 5.1).

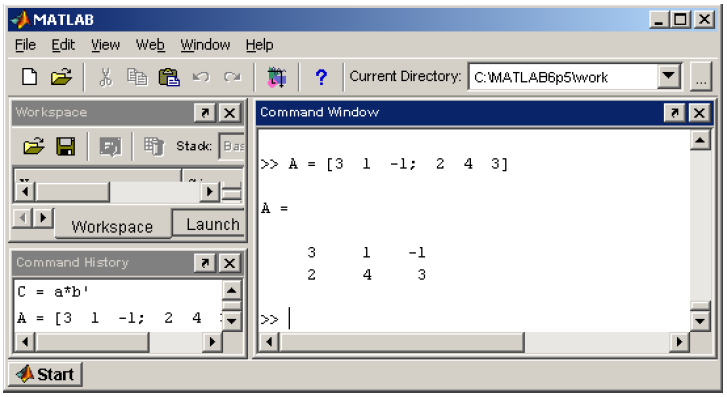


Рис. 5.1

Для изучения простейших операций над матрицами приведем еще несколько примеров. Рассмотрим другие способы ввода. Введите квадратную матрицу размера три так, как описано ниже:

$$B = \begin{pmatrix} 4 & 3 & -1 \\ 2 & 7 & 0 \\ -5 & 1 & 2 \end{pmatrix}.$$

Начните набирать в командной строке

```
>> B = [ 4 3 –1
```

Нажмите клавишу <**Enter**>. Обратите внимание, что MatLab ничего не вычислила. Курсор мигает на следующей строке без символа >>. Продолжите ввод матрицы построчно, нажимая в конце каждой строки <**Enter**>. Последнююстроку завершите закрывающей квадратной скобкой.

```
2 7 0
–5 1 2
B =
     4     3    –1
     2     7     0
    –5     1     2
```

Еще один способ ввода матриц состоит в том, что матрицу можно трактовать как вектор-строку, каждый элемент которой является вектор-столбцом. Например, матрицу

$$C = \begin{pmatrix} 3 & -1 & 7 \\ 4 & 2 & 0 \end{pmatrix}$$

можно ввести при помощи команды, показанной на рис. 5.2.

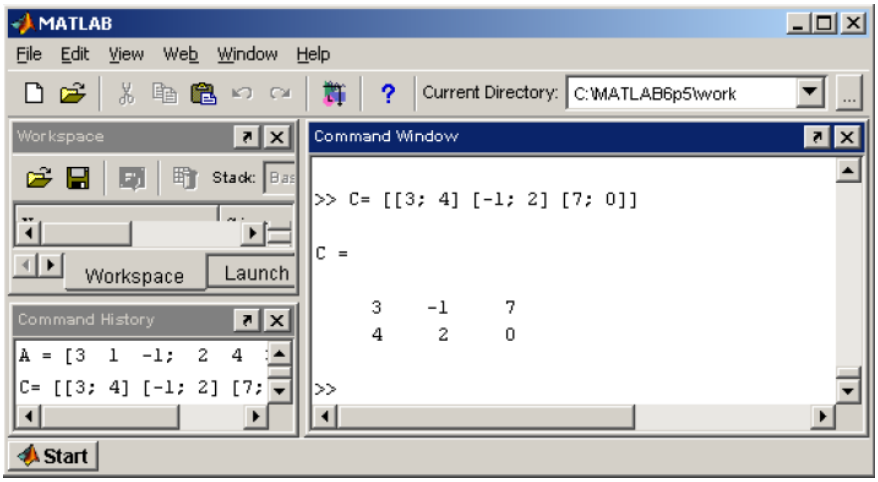


Рис. 5.2

Посмотрите переменные рабочей среды, набрав в командной строке **whos**.

Обращение к элементам матриц

Доступ к элементам матриц осуществляется при помощи двух индексов - номеров строки и столбца, заключенных в круглые скобки (рис. 5.3).

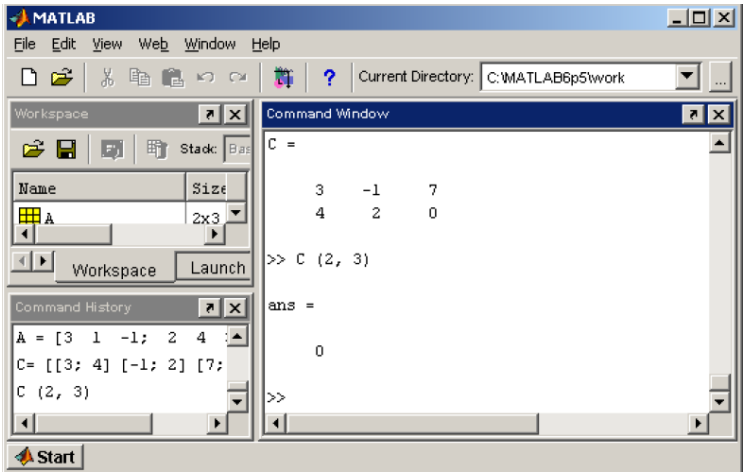


Рис. 5.3

Элементы матриц могут входить в состав разных выражений (рис. 5.4).

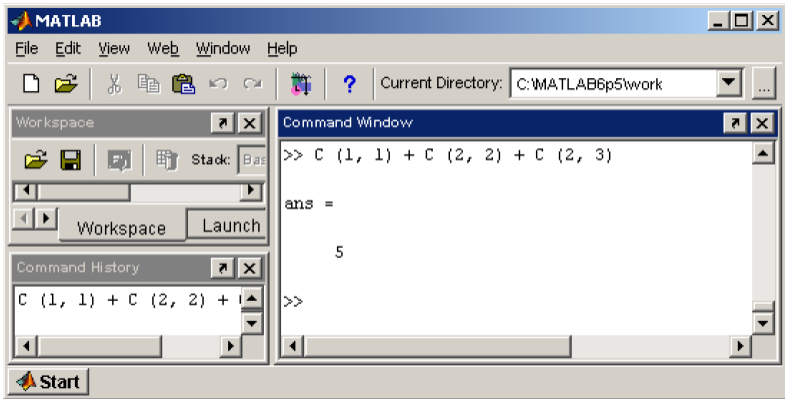


Рис. 5.4

Расположение элементов матрицы в памяти компьютера определяет еще один способ обращения к ним. Матрица *A* размера ***m*** на ***n*** хранится в виде вектора длины ***mn***, в котором элементы матрицы расположены один за другим построчно

[A (1, 1) A (1, 2) … A (1, n) … A (m, 1) A (m, 2) … A (m, n)]

Для доступа к элементам матрицы можно использовать *один индекс*, задающий порядковый номер элемента матрицы в векторе.

Матрица C, определенная в предыдущем разделе, содержится в векторе

[C (1, 1) C (1, 2) C (1, 3) C (2, 1) C (2, 2) C (2, 3)].

Индексация при помощи порядкового номера производится, как показано на рис. 5.5.

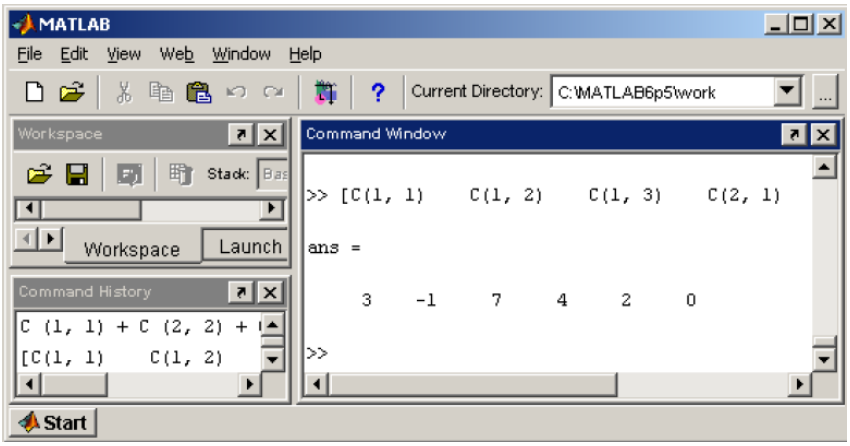


Рис. 5.5

Сложение, вычитание, умножение, транспонирование и возведение в степень

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. Найдите сумму и разность матриц *C* и *A*, определенных выше (рис. 5.6).

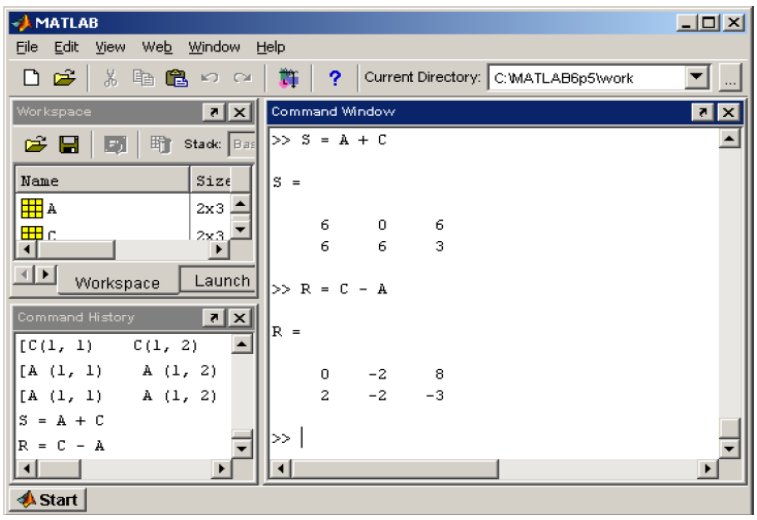


Рис. 5.6

Следите за совпадением размерности, иначе получите сообщение об ошибке.

Для умножения матриц предназначена звездочка (рис. 5.7).Умножение матрицы на число тоже осуществляется при помощи звездочки, причем умножать можно как справа, так и слева (рис. 5.8).

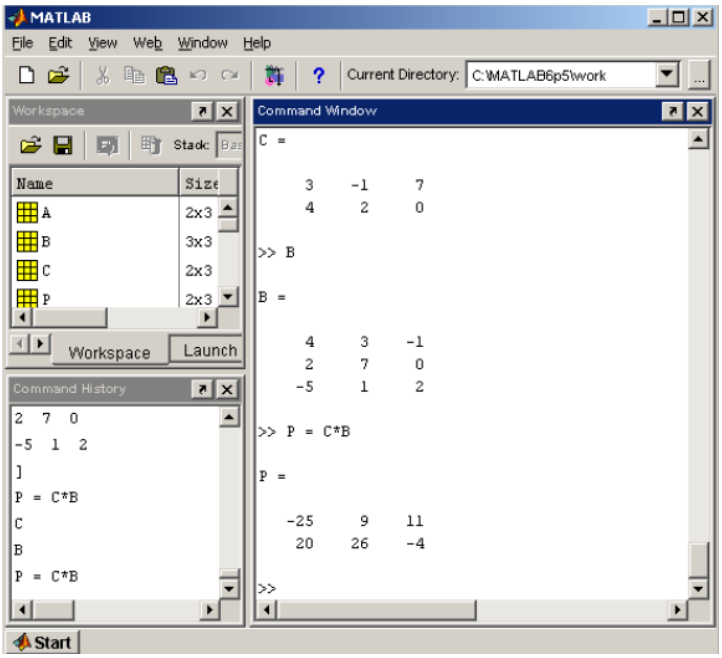


Рис. 5.7

Транспонирование матрицы так же, как и вектора, производится при помощи .', а символ ' означает комплексное сопряжение. Для вещественных матриц эти операции приводят к одинаковым результатам (рис. 5.9).

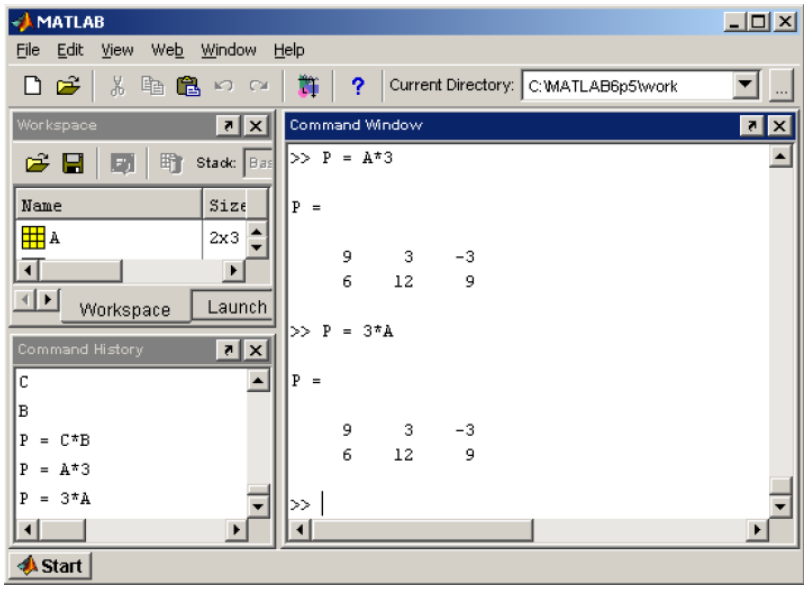


Рис. 5.8

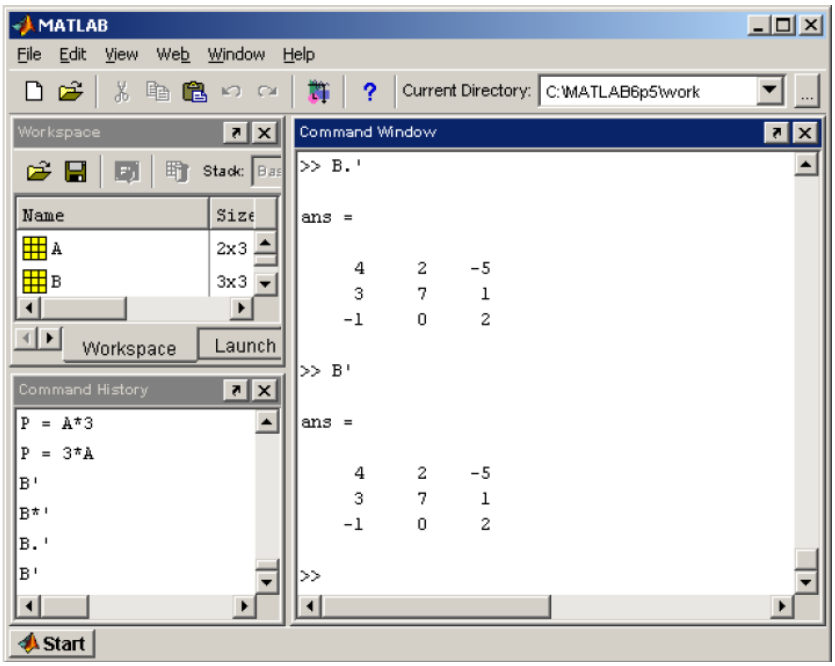


Рис. 5.9

Сопряжение и транспонирование матриц, содержащих комплексные числа, приведут к созданию разных матриц (рис. 5.10).

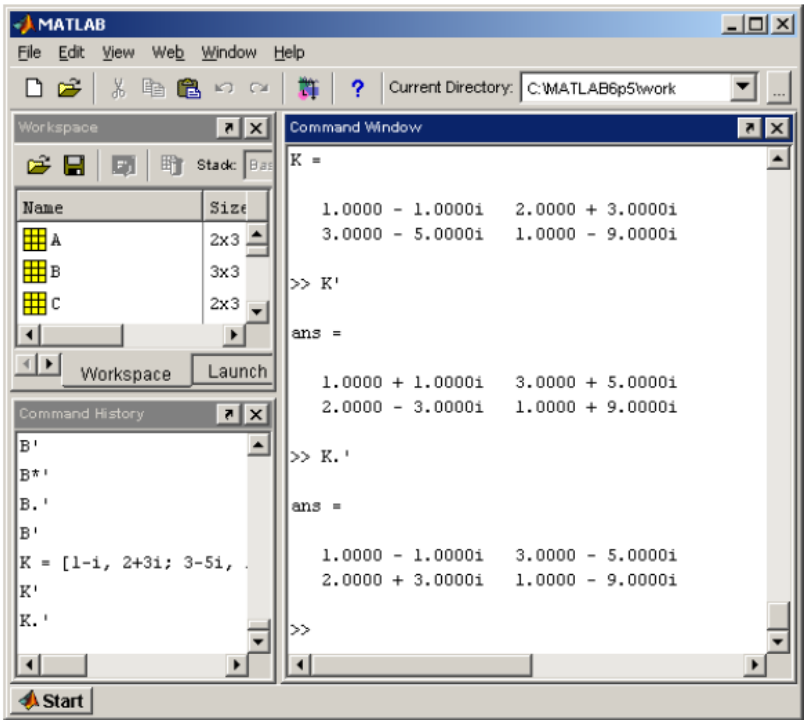


Рис. 5.10

Вспомните, что при вводе вектор-строк их элементы можно разделять или пробелами, или запятыми. При вводе матрицы *K* применены запятые для более наглядного разделения комплексных чисел в строке.

Возведение квадратной матрицы в целую степень производится с использованием оператора ∧ (рис. 5.11).

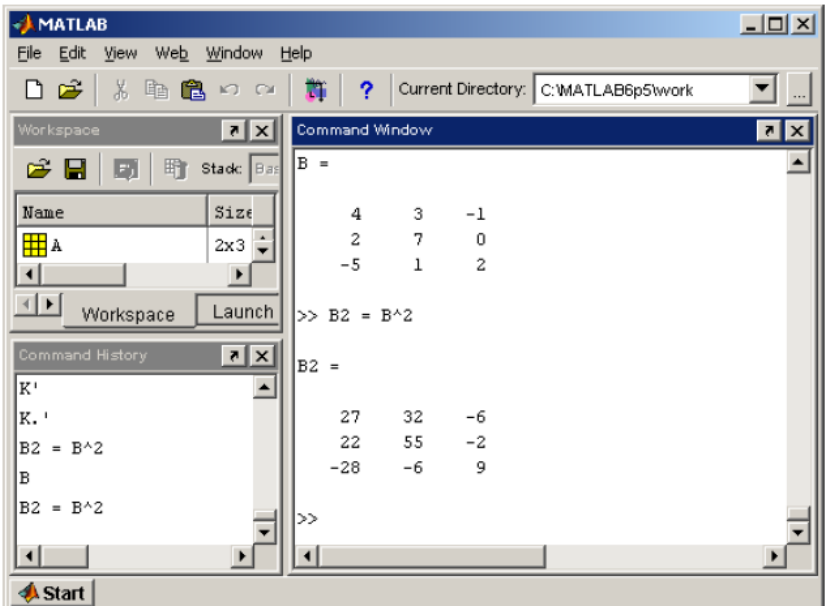


Рис. 5.11

Проверьте полученный результат, умножив матрицу саму на себя.

Найдите значение следующего выражения:

$$(A+C)\cdot B^3\cdot(A-C)^T.$$

Приоритет операций: сначала выполняется транспонирование, потом возведение в степень, затем умножение, а сложение и вычитание производятся в последнюю очередь (рис. 5.12).

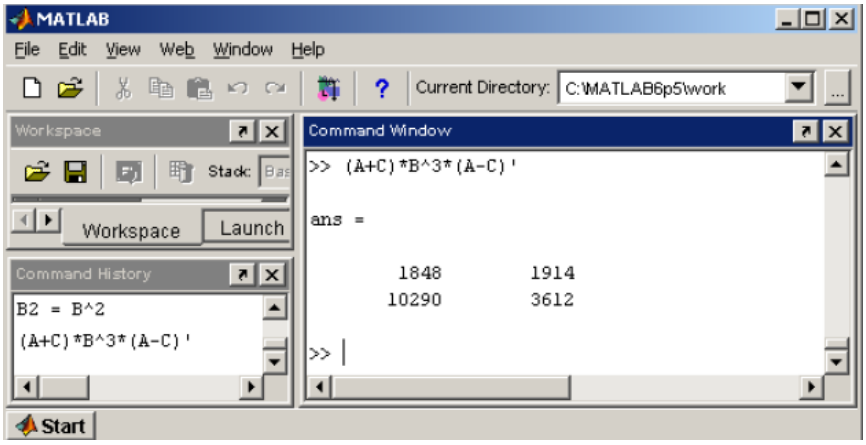


Рис. 5.12

Перемножение матрицы и вектора

Поскольку вектор-столбец или вектор-строка в MatLab являются матрицами, у которых один из размеров равен единице, все вышеописанные операции применимы и для умножения матрицы на вектор или вектор-строки на матрицу. Например, вычисление выражения

$$\begin{bmatrix}1 & 3 & -2\end{bmatrix}\cdot\begin{pmatrix}2 & 0 & 1\\ -4 & 8 & -1\\ 0 & 9 & 2\end{pmatrix}\cdot\begin{bmatrix}-8\\ 3\\ 4\end{bmatrix}$$ на рис. 5.13.

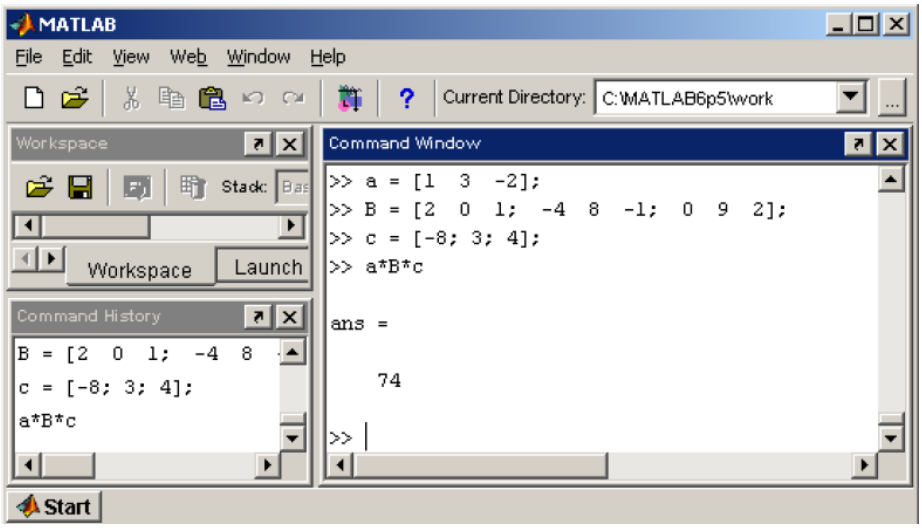


Рис. 5.13

В математике ничего не говорится про деление матриц и векторов, однако в MatLab символ \ используется для решения систем линейных уравнений.

5.2.Решение систем линейных уравнений

Решите небольшую систему, состоящую из трех уравнений, с тремя неизвестными:

$$\begin{cases} 1.2x_1 + 0.3x_2 - 0.2x_3 = 1.3 \\ 0.5x_1 + 2.1x_2 + 1.3x_3 = 3.9 \\ -0.9x_1 + 0.7x_2 + 5.6x_3 = 5.4 \end{cases}$$

Введите матрицу системы в массив *A*, для вектора правой части используйте массив *b*. Решите систему при помощи символа \ (рис. 5.14).

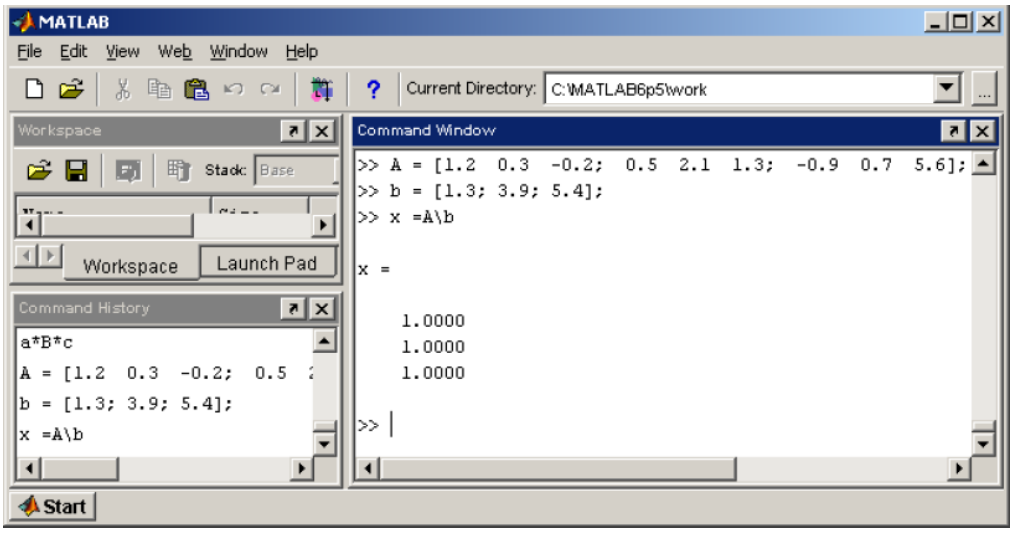


Рис. 5.14

Проверьте, правильный ли получился ответ, умножив *A* на *x*.

Алгоритм решения систем линейных уравнений при помощи операторов \ определяется структурой матрицы коэффициентов системы. В частности, MatLab исследует, является ли матрица треугольной, или может быть приведена перестановками строк и столбцов к треугольному виду, симметричная матрица или нет, квадратная или прямоугольная (MatLab умеет решать системы с прямоугольными матрицами – переопределенные или не доопределённые). Решать системы при помощи \ разумно, когда выбор

алгоритма решения поручается MatLab. Если же имеется информация о свойствах матрицы системы, то лучше использовать специальные методы.

Решение систем небольшой размерности можно выполнить, введя матрицу системы и вектор правой части непосредственно из командной строки. Однако часто требуется найти решение системы, состоящей из большого числа линейных уравнений, причем матрица и вектор правой части системы хранятся в файлах.

5.3. Считывание и запись данных

Перед нами стоит задача – решить систему линейных уравнений, матрица и вектор правой части которой хранятся в текстовых файлах **matr.txt**, **rside.txt**, и записать результат в файл **sol.txt**. Матрица записана в файле построчно, элементы в сроке отделены пробелом, вектор правой части записан в столбик.

Подготовьте файлы с данными, например, в стандартной программе windows-блокнот (NotePad). Скопируйте файлы**matr.txt**, **rside.txt** в подкаталог **work** основного каталога MatLab. Для считывания из файла используйте команду **load**, для записи – **save** (рис. 5.15).

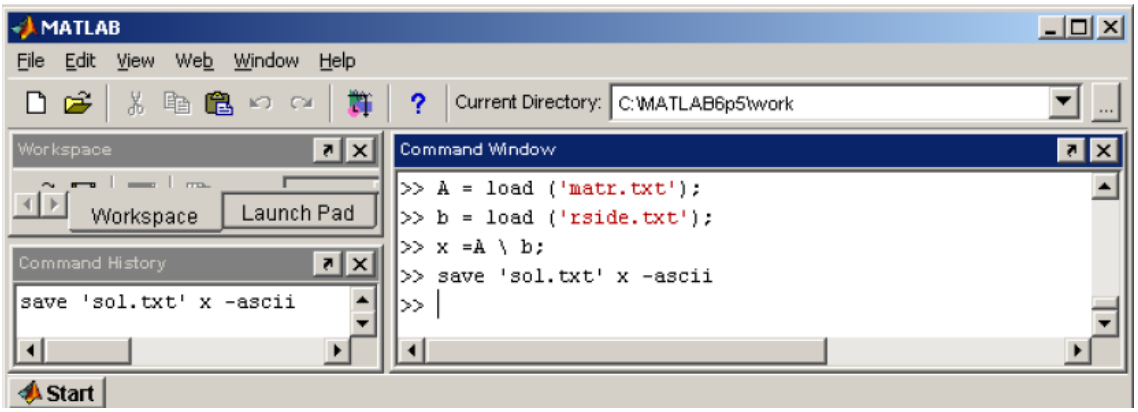


Рис. 5.15

Параметр – **ascii** означает запись в текстовом формате и может использоваться в каталоге **work** для создания файла **sol.txt**, в котором будет записано решение. Посмотреть содержимое файла можно, используя любой текстовый редактор. Для записи результата в файл с двойной точностью следует использовать команду **save**‘**sol.txt**’ x –**ascii–double.**

Аналогично можно записать матрицу в текстовый файл. Запись матрицы *A*, хранящейся в массиве *A*, в файл **sol.txt** осуществляется командой **save'sol.txt'**A – **ascii**.

6. Блочные матрицы

6.1. Работа с блочными матрицами

Блочные матрицы – это матрицы, составленные из непересекающихся подматриц (блоков). Соответствующие размеры блоков должны совпадать.

Конструирование блочных матриц

Введите матрицы

$$A=\begin{pmatrix}-1 & 4\\ -1 & 4\end{pmatrix},\ B=\begin{pmatrix}2 & 0\\ 0 & 5\end{pmatrix},\ C=\begin{pmatrix}3 & -3\\ -3 & 3\end{pmatrix},\ D=\begin{pmatrix}8 & 9\\ 1 & 10\end{pmatrix}$$

и создайте из них блочную матрицу

$$K=\begin{pmatrix}A & | & B\\ -- & + & --\\ C & | & D\end{pmatrix},$$

учитывая, что матрица *K* состоит из двух строк: в первой строке матрицы *A* и *B*, а во второй – *C* и *D*:

```
>> K = [A  B; C  D]
K =
     -1     4     2     0
     -1     4     0     5
      3    -3     8     9
     -3     3     1    10
```

Блочная матрица получена. Можно было поступить и по-другому, например, считать, что матрица *K* состоит из двух столбцов: в первом – матрицы *A* и *C,* а во втором – *B* и *D*. Тогда команду для создания блочной матрицы следовало бы записать так:

```
>> K = [[A;  C] [B;  D]]
```

Рассмотрим еще один пример для проверки знаний о работе с массивами в MatLab. Требуется составить блочную матрицу

$$M = \left(\begin{array}{c|c} S & a \\ -- & -- \\ b & 2.5 \end{array}\right),$$

где

$$S = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix},\ a = \begin{pmatrix} 4 \\ 5 \end{pmatrix},\ b = \begin{pmatrix} -9 & 9 \end{pmatrix}.$$

Решение этой задачи показано на рис. 6.1.

Последний оператор можно заменить на эквивалентный M = [[S;b][a;2.5]]. Обратной задачей к конструированию блочных матриц является выделение блоков.

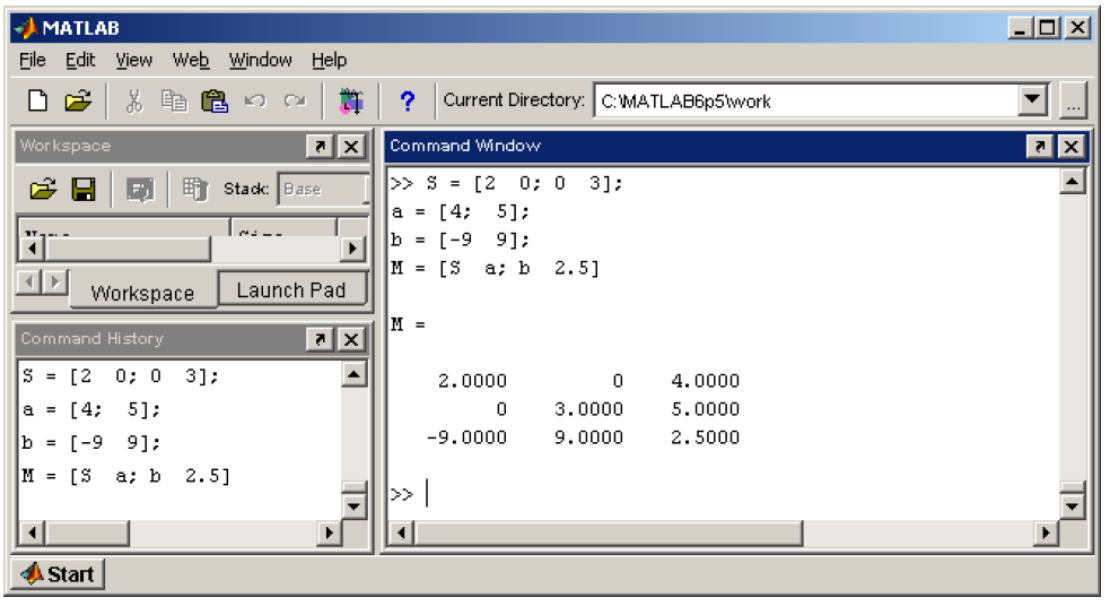


Рис. 6.1

Выделение блоков

Выделение блоков матриц осуществляется индексацией при помощи двоеточия, которая уже использовалась ранее для выделения блоков из векторов. Введите матрицу

$$P = \begin{pmatrix} 1 & 2 & 0 & 2 \\ 4 & 10 & 12 & 5 \\ 0 & 11 & 10 & 5 \\ 9 & 2 & 3 & 5 \end{pmatrix},$$

затем выделите очерченный блок, задав номера строк и столбцов при помощи двоеточия (рис. 6.2).

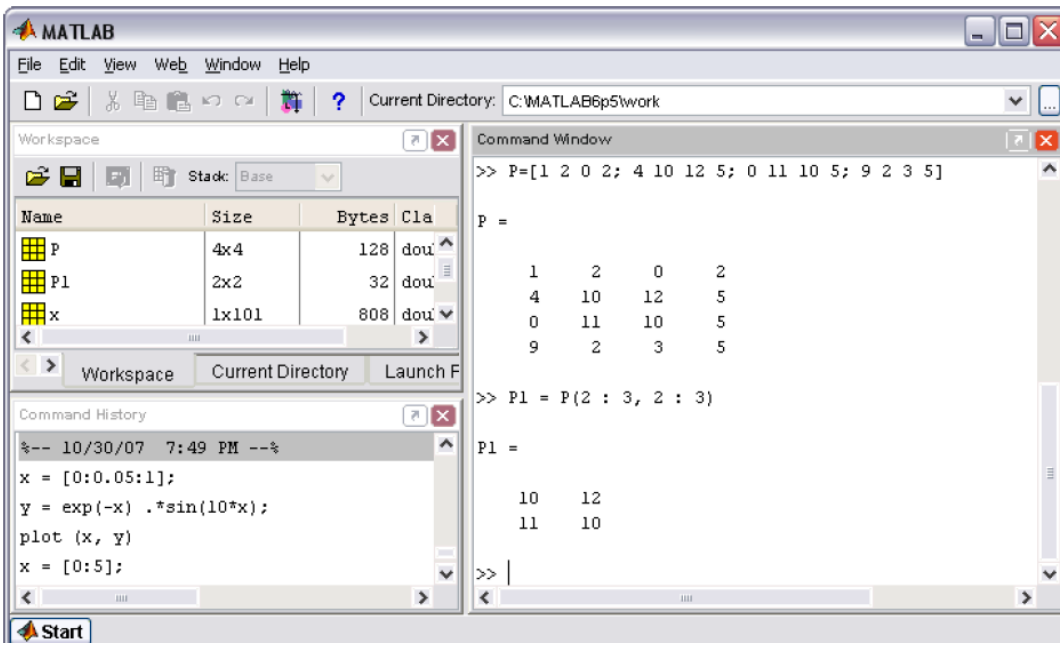


Рис. 6.2

Для выделения из матрицы столбца или строки (т.е. массива, у которого один из размеров равен единице) следует в качестве одного из индексов использовать номер столбца или строки матрицы, а другой индекс заменить на двоеточие без указания пределов. Например, запишите вторую строку P в вектор p (рис. 6.3).

При выделении блока до конца матрицы можно не указывать ее размеры, а использовать элемент **end**:

```
>> p = P(2,  2:end)
p =
        10      12      5
```

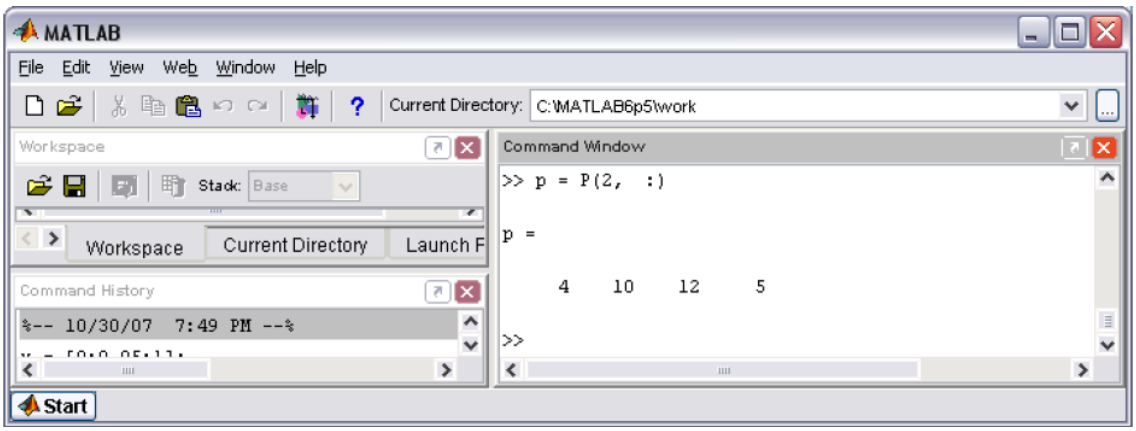


Рис. 6.3

Удаление строк и столбцов

В MatLab парные квадратные скобки [ ] обозначают пустой массив, который, в частности, позволяет удалять строки и столбцы матрицы. Для удаления строки следует присвоить ей пустой массив. Удалите, например, первую строку квадратной матрицы (рис. 6.4).

Обратите внимание на соответствующее изменение размеров массива, которое можно проверить при помощи **size**:

```
>> size(M)
ans =
        2        3
```

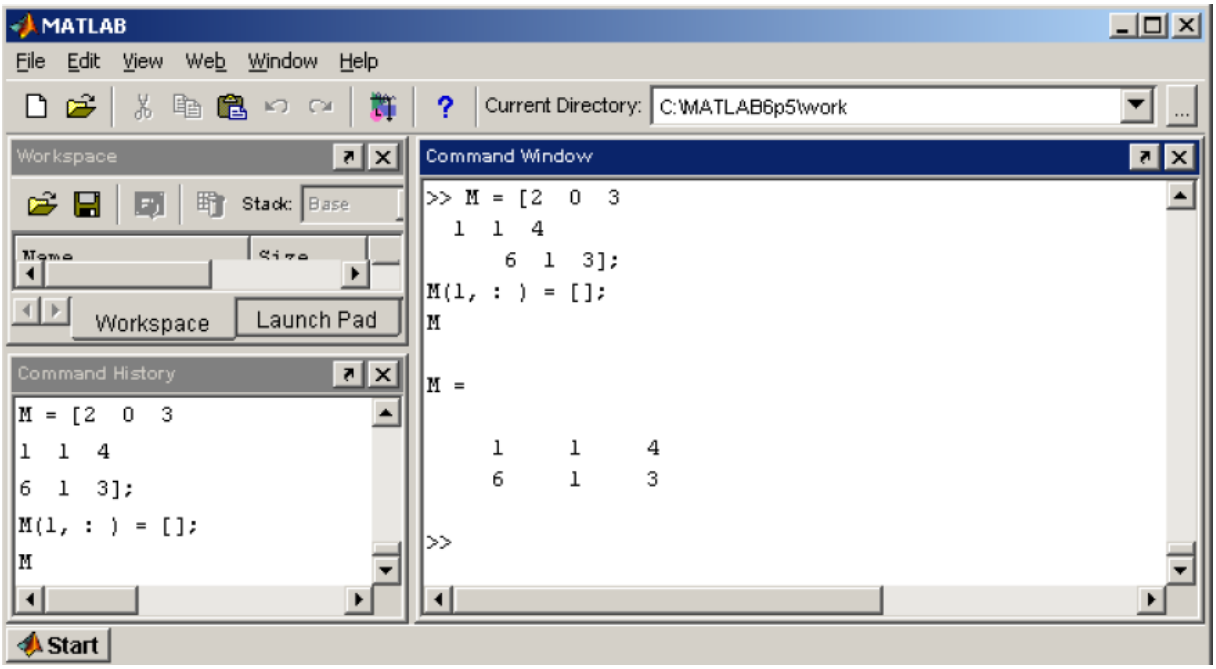


Рис. 6.4

Аналогичным образом удаляются и столбцы. Для удаления нескольких идущих подряд столбцов (или строк) им нужно присвоить пустой массив. Удалите второй и третий столбец в массиве M (рис. 6.5).

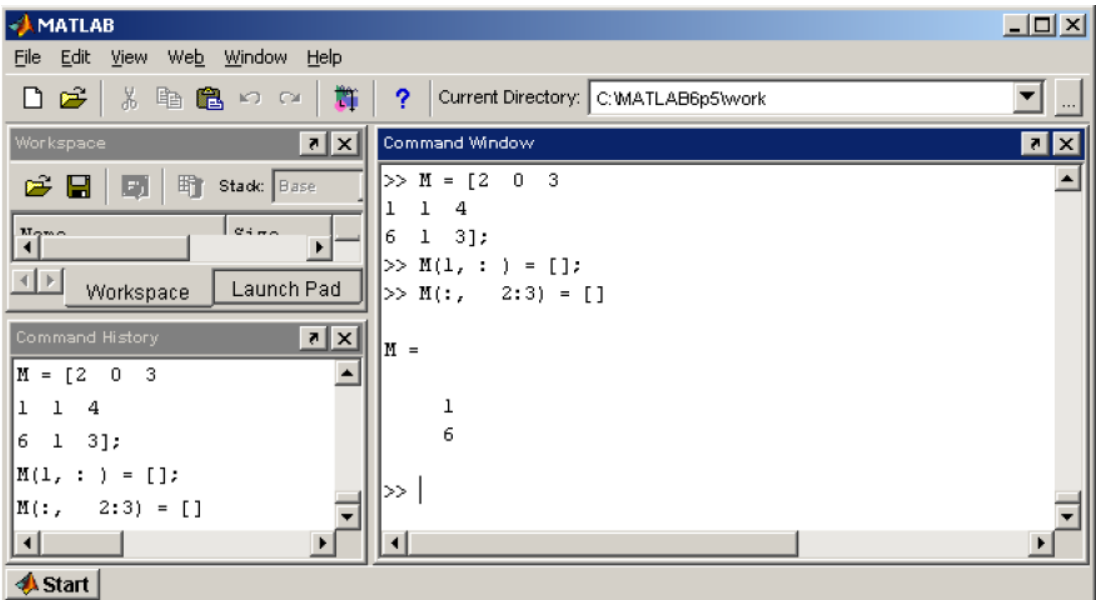


Рис. 6.5

Индексация существенно экономит время при вводе матриц, имеющих определенную структуру.

6.2. Заполнение матриц при помощи индексации

Выше было описано несколько способов ввода матриц в MatLab, в том числе и считывание из файла. Однако бывает проще сгенерировать матрицу, чем вводить ее, особенно если она обладает простой структурой. Рассмотрим пример такой матрицы:

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

Генерация матрицы T осуществляется в три этапа:

1. Создание массива T размера пять на пять, состоящего из нулей.

2. Заполнение первой строки единицами.

3. Заполнение части последней строки минус единицами до последнего элемента.

Соответствующие команды MatLab приведены ниже (они записаны в три колонки с целью экономии места, но набирать их надо последовательно).

Команды не завершаются точкой с запятой для вывода промежуточных результатов в командное окно с целью слежения за процессом формирования матрицы

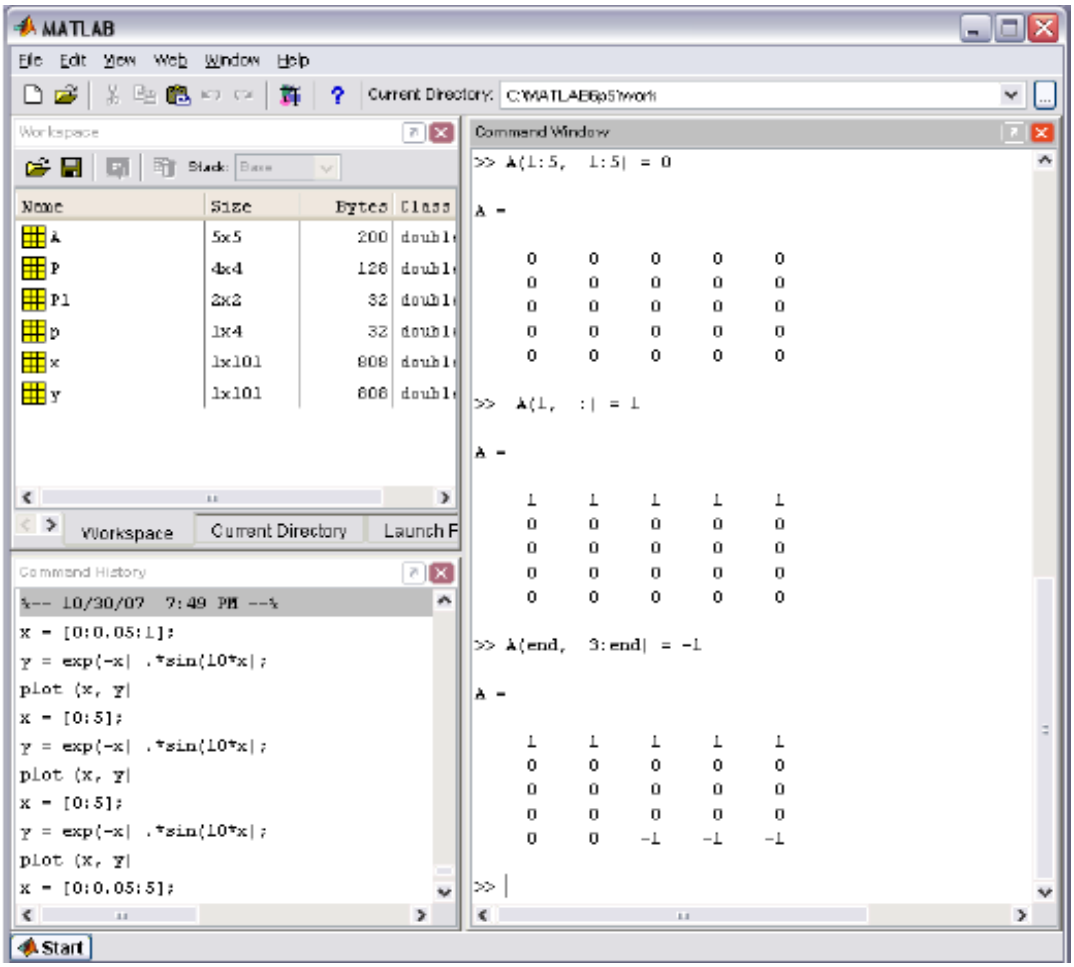


Рис. 6.6

Создание некоторых специальных матриц в MatLab осуществляется при помощи встроенных функций.

6.3. Создание матриц специального вида

Заполнение прямоугольной матрицы нулями производится встроенной функцией **zeros**, аргументами которой являются число строк и столбцов матрицы, показанной на рис. 6.7.

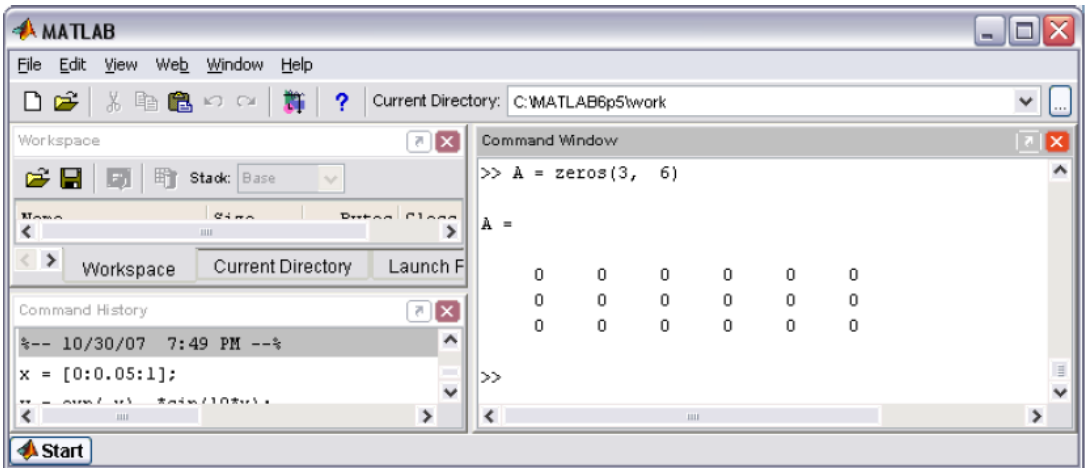


Рис. 6.7

Один аргумент функции **zeros** приводит к образованию квадратнойматрицы заданного размера (рис. 6.8).

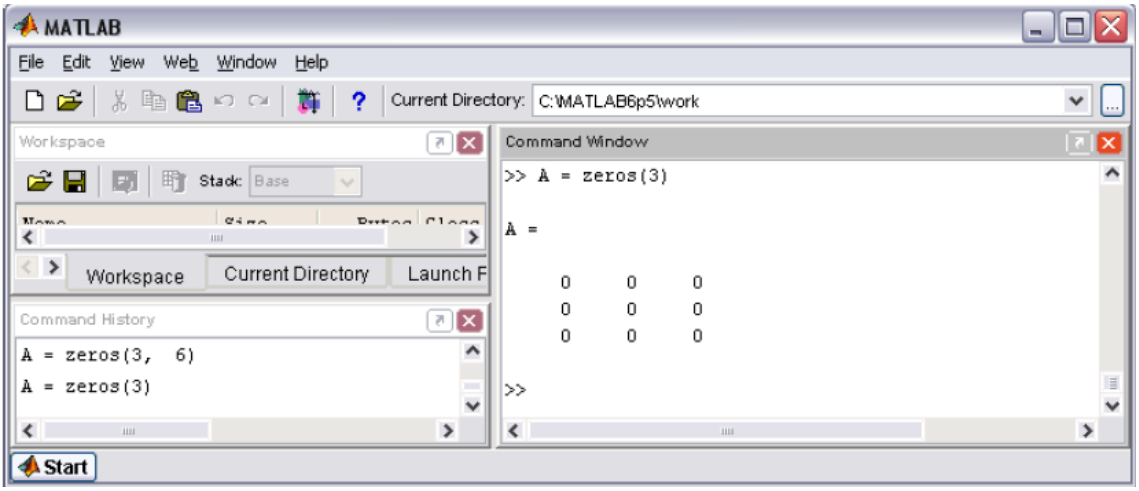


Рис. 6.8

Единичная матрица инициализируется при помощи функции **eye** (рис. 6.9).

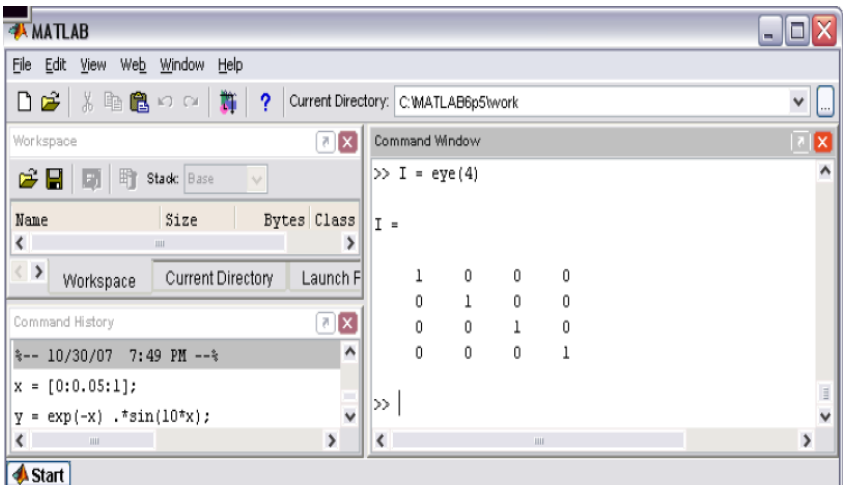


Рис. 6.9

Функция **eye** с двумя аргументами создает прямоугольную матрицу, у которой на главной диагонали стоят единицы, а остальные элементы равны нулю (рис. 6.10).

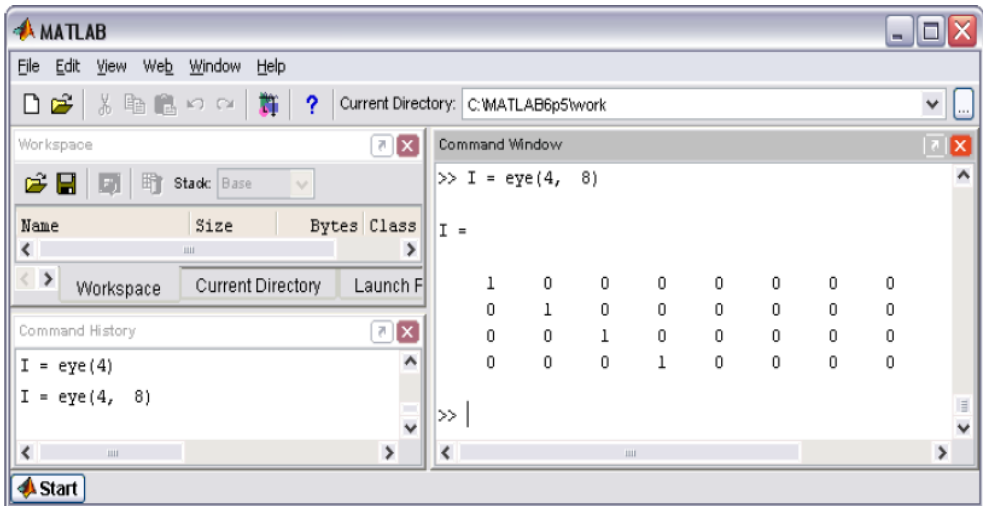


Рис. 6.10

Матрица, состоящая из единиц, образуется в результате вызова функции **ones** (рис. 6.11).

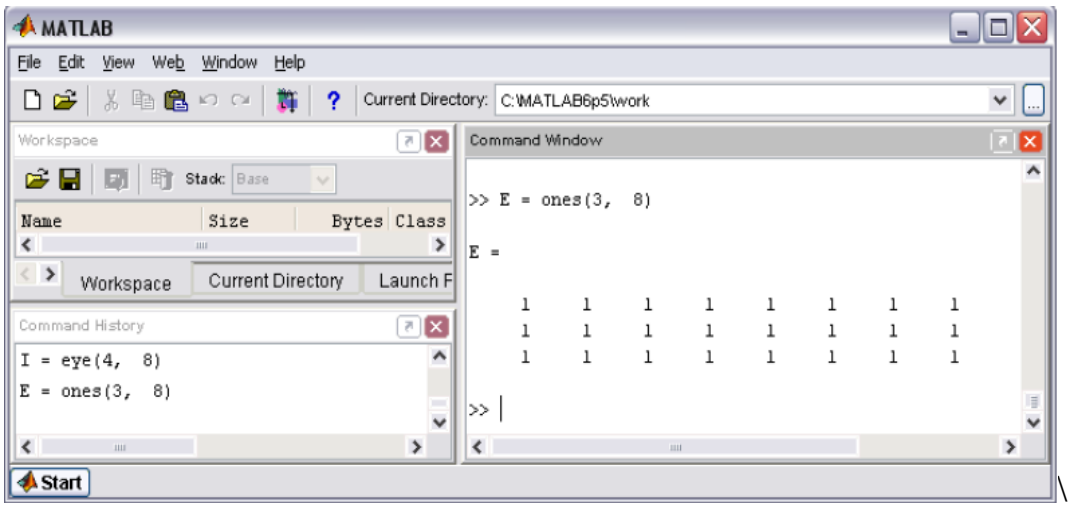


Рис. 6.11

Использование одного аргумента в **ones** приводит к созданию квадратной матрицы, состоящей из единиц.

MatLab предоставляет возможность заполнения матриц случайными элементами. Результатом функции **rand** является матрица чисел, распределенных случайным образом между нулем и единицей, а функции **randn** – матрица чисел, распределенных по нормальному закону (рис. 6.12).

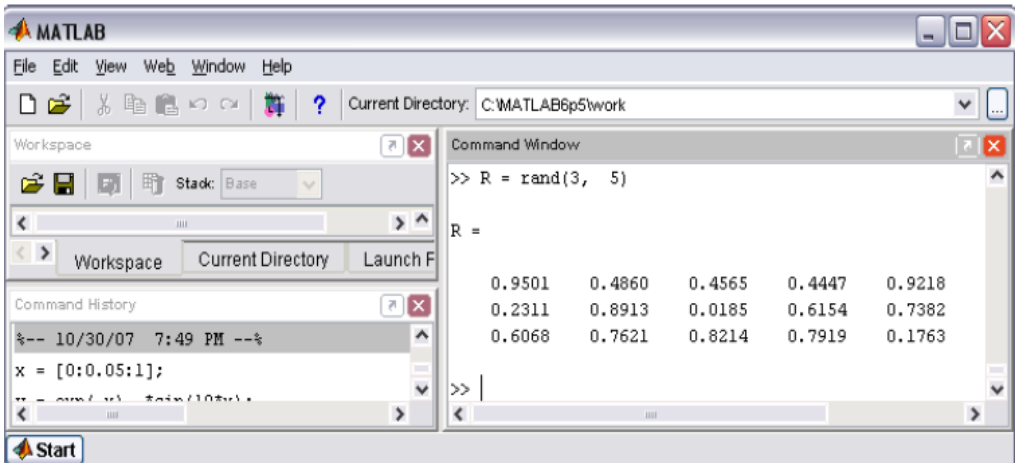


Рис. 6.12

Один аргумент функций **rand** и **randn** приводит к формированию квадратных матриц (рис. 6.13).

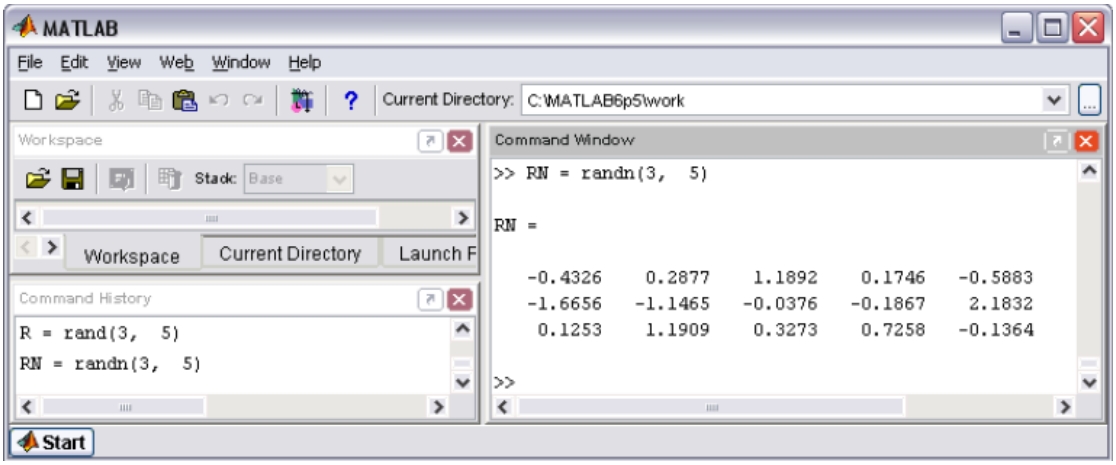


Рис. 6.13

Вопрос об автоматическом заполнении векторов или вектор-строк не должен поставить нас в тупик, поскольку мы знаем что вектор или вектор-строка в MatLab являются матрицей, у которой один из размеров равен единице. Заполните вектор-строку случайными числами.

Проверьте себя, выполнив следующий пример, – см. рис. 6.14.

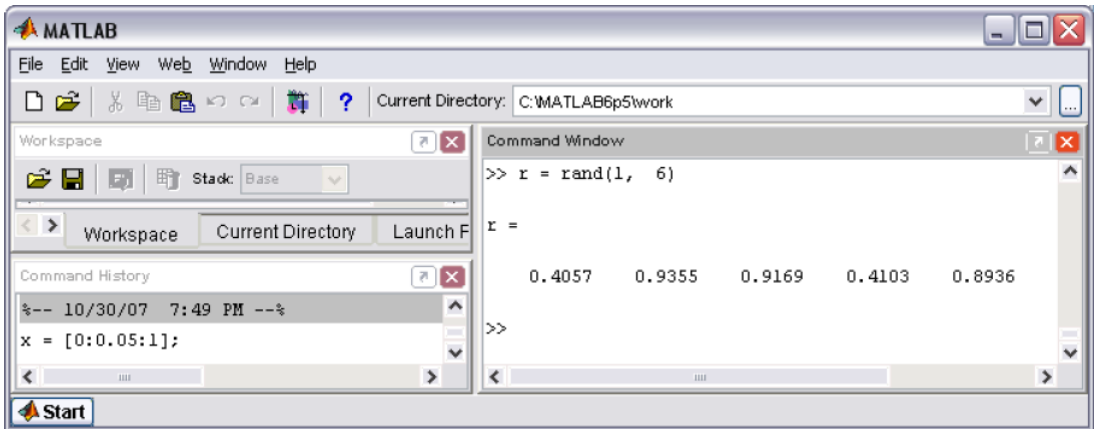


Рис. 6.14

Часто возникает необходимость создания диагональных матриц, т.е. матриц, у которых все внедиагональные элементы равны нулю. Функция **diag** формирует диагональную матрицу из вектора или вектор-строки, располагая их элементы по диагонали матрицы (рис. 6.15).

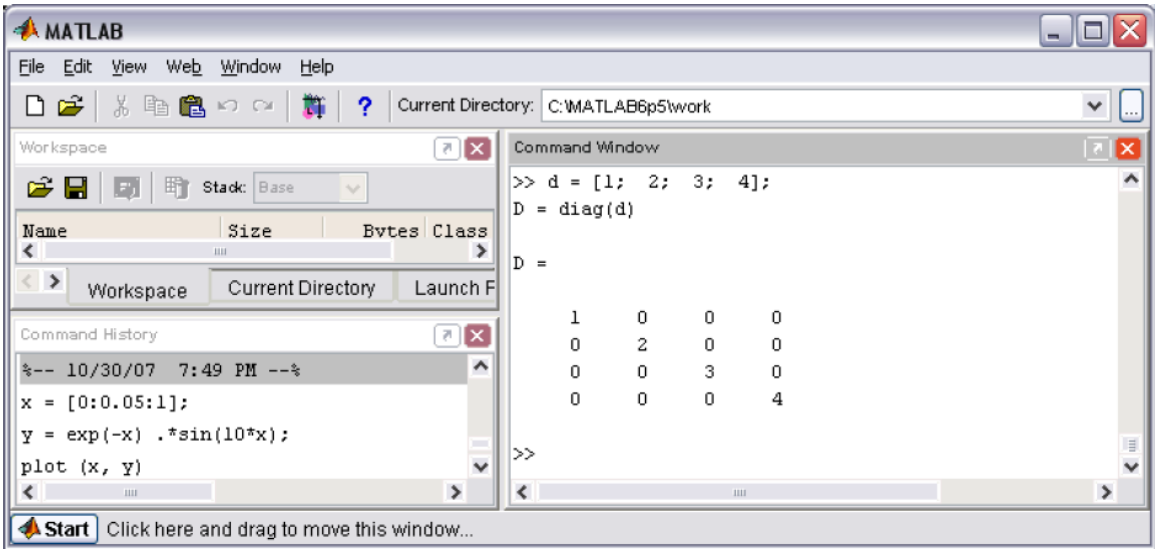


Рис. 6.15

Для заполнения не главной, а побочной диагонали предусмотрена возможность вызова функции **diag** с двумя аргументами. В этом случае второй аргумент означает, насколько побочная диагональ отстоит от главной, а его знак указываетна направление: плюс – вверх, минус – вниз от главной диагонали (рис. 6.16).

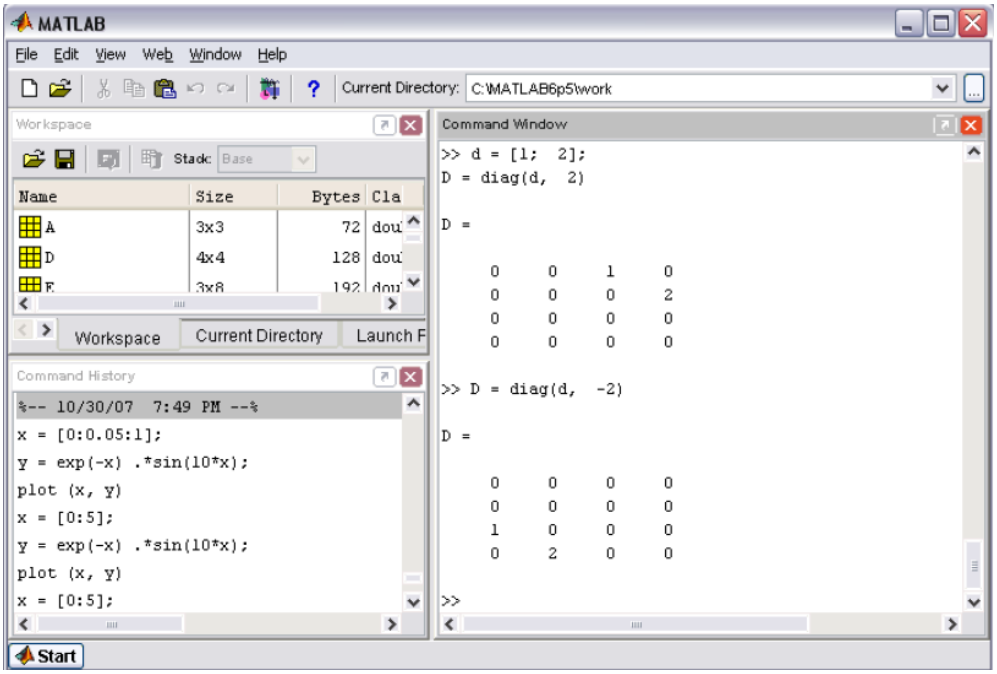


Рис. 6.16

Функция **diag** служит и для выделения диагонали матрицы в вектор (рис. 6.17).

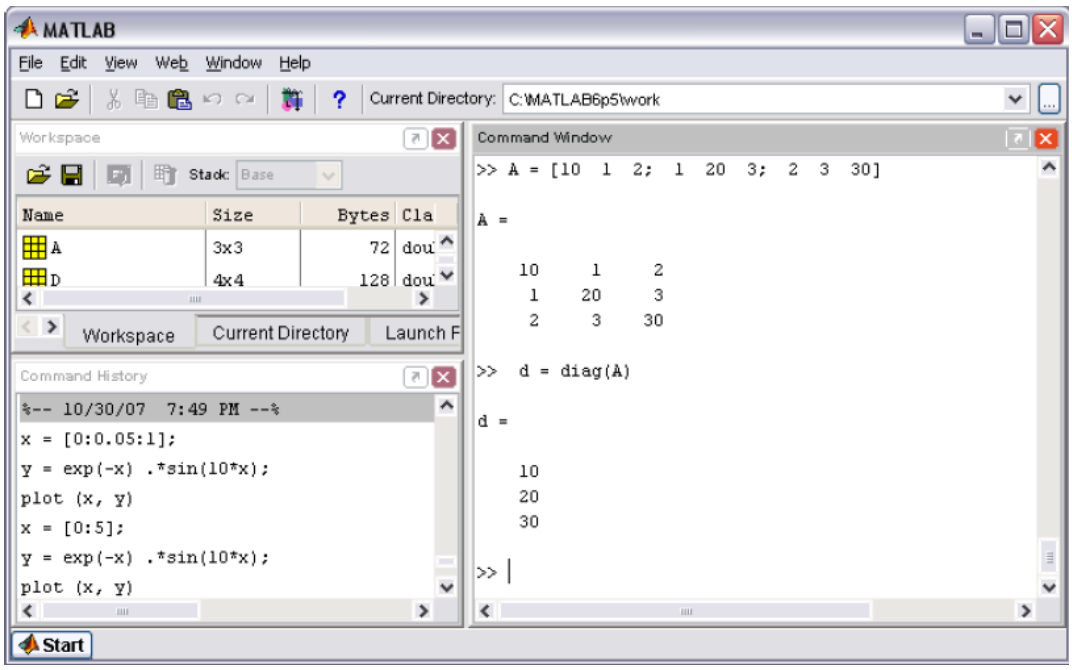


Рис. 6.17

Проделайте следующее упражнение для того, чтобы убедиться, что вы освоили автоматическое заполнение матриц, работу с блочными матрицами и запись данных в файл. Заполните и запишите в файлы матрицы следующее:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & -4 & 0 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & -4 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & -4 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & 0 & -4 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}, \quad G = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}.$$

Матрицу M можно представить в виде блочной матрицы из четырех квадратных блоков размера пять, диагональные блоки формируются при помощи функции **eye**, а внедиагональные – с использованием **ones** (рис. 6.18).

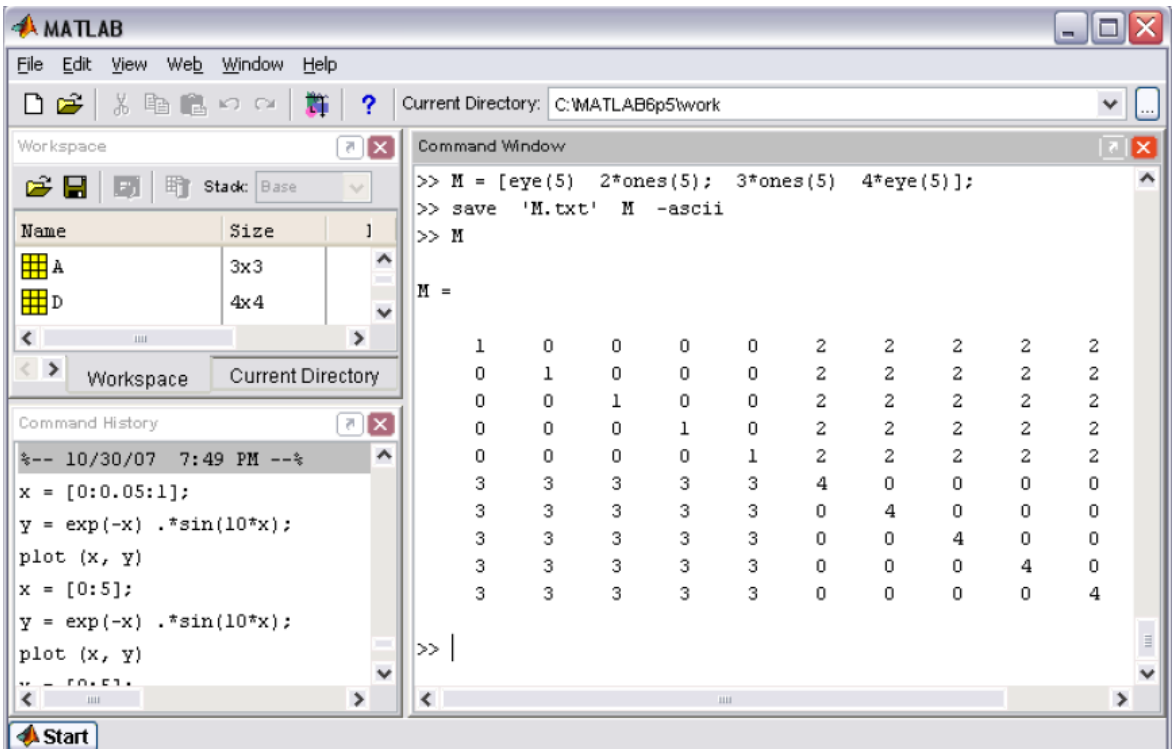


Рис. 6.18

Квадратная матрица G размера семь является трехдиагональной. Заполняя эту матрицу, можно использовать то обстоятельство, что G является суммой диагональной матрицы и двух матриц с шестью ненулевыми элементами над и под главной диагональю (рис. 6.19).

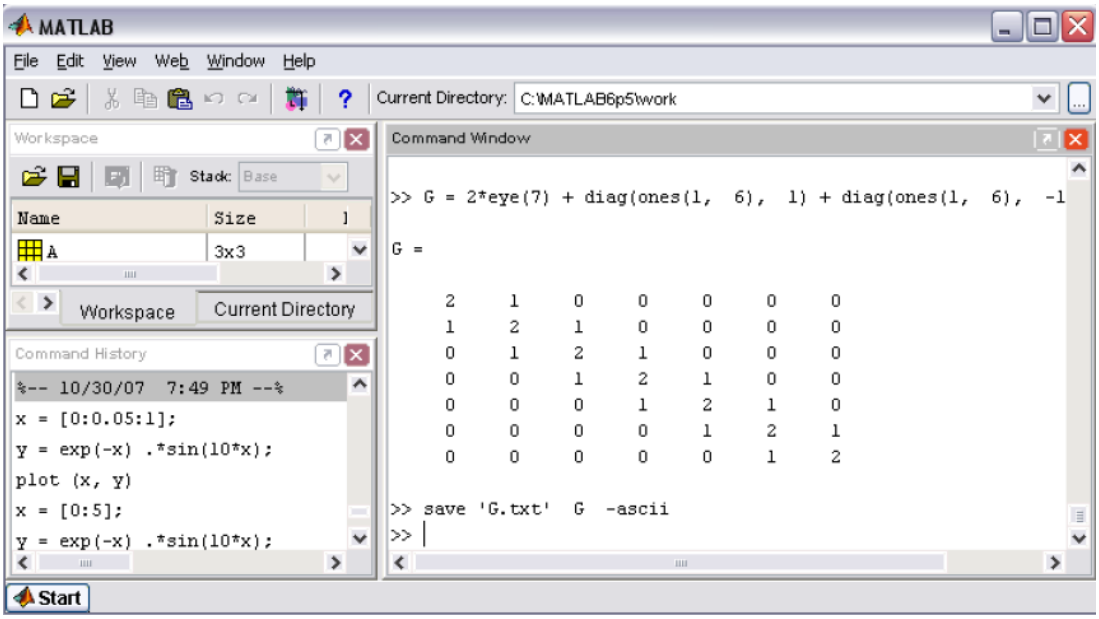


Рис. 6.19

Посмотрите содержание полученных файлов (они должны находиться в подкаталоге **work** основного каталога MatLab). При больших размерах матриц бывает удобно получить ее представление в наглядном виде. В MatLab это можно проделать при помощи визуализации матриц. В следующем разделе разобраны простейшие способы визуализации матричных данных.

7. Визуализация матриц и поэлементные операции над ними

7.1. Визуализация матриц

Матрицы с достаточно большим количеством нулей называются *разреженными*. Часто необходимо знать, где расположены ненулевые элементы, т. е. получить так называемый *шаблон* матрицы. Для этого в MatLab служит функция **spy**. Представим шаблон матрицы G:

$$G = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

```
>> spy(G)
```

После выполнения команды **spy** на экране появляется графическое окно **Figure No. 1**(рис. 7.1).

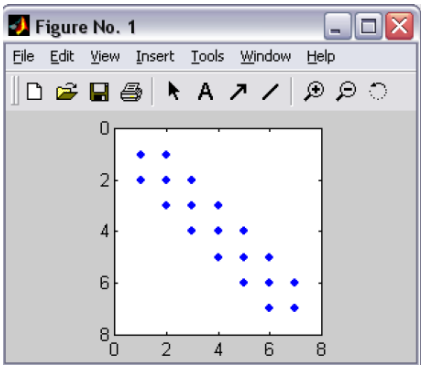


Рис. 7.1

На вертикальной и горизонтальной осях отложены номера строк и столбцов. Ненулевые элементы обозначены маркерами, внизу графического окна указано число ненулевых элементов (nz = 19).

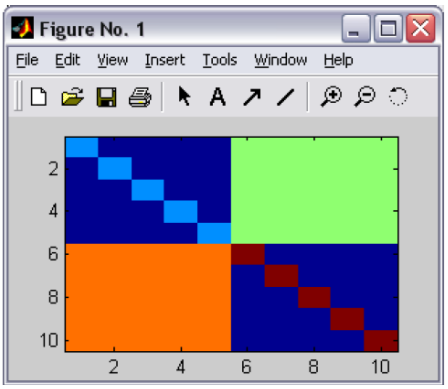
Наглядную информацию о соотношении величин элементов матрицы дает функция **imagesc**, которая интерпретирует матрицу как прямоугольное изображение. Каждый элемент матрицы представляется в виде квадратика, цвет которого соответствует величине элемента. Для того чтобы узнать соответствие цвета и величины элемента следует использовать команду **colorbar**, выводящую рядом с изображением матрицы шкалу цвета. Наконец, для печати на монохромном принтере удобно получить изображение в оттенках серого цвета, используя команду **colormap(gray)**. Мы будем работать с матрицей M:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & -4 & 0 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & -4 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & -4 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & 0 & -4 & 0 \\ 3 & 3 & 3 & 3 & 3 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}.$$

Набирайте команды, указанные ниже и следите за состоянием графического окна.

```
>> imagesc(M)
```

a)




```
>> colorbar
```

б)

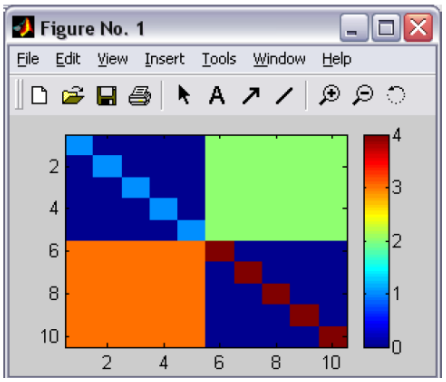


Рис. 7.2

```
>> colormap(gray)
```

в)

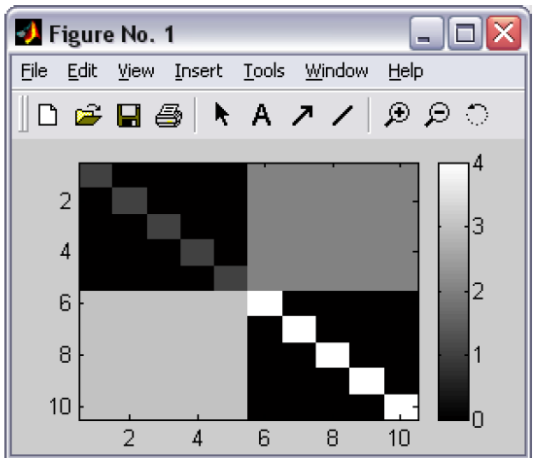


Рис. 7.2.Окончание

В результате получается наглядное представление матрицы (рис. 7.2).

7.2. Поэлементные операции и встроенные функции

Поскольку векторы и матрицы хранятся в двумерных массивах, то применение математических функций к матрицам и поэлементные операции производятся так же, как для векторов. Работа со встроенными функциями (**min**, **max**, **sum** и т. д.) имеет свои особенности в применении к матрицам.

Поэлементные операции с матрицами

Введите две матрицы:

$$A = \begin{pmatrix} 2 & 5 & -1 \\ 3 & 4 & 9 \end{pmatrix} \quad B = \begin{pmatrix} -1 & 2 & 8 \\ 7 & -3 & -5 \end{pmatrix}.$$

Умножение каждого элемента одной матицы на соответствующий элемент другой производится при помощи оператора .* (рис. 7.3).

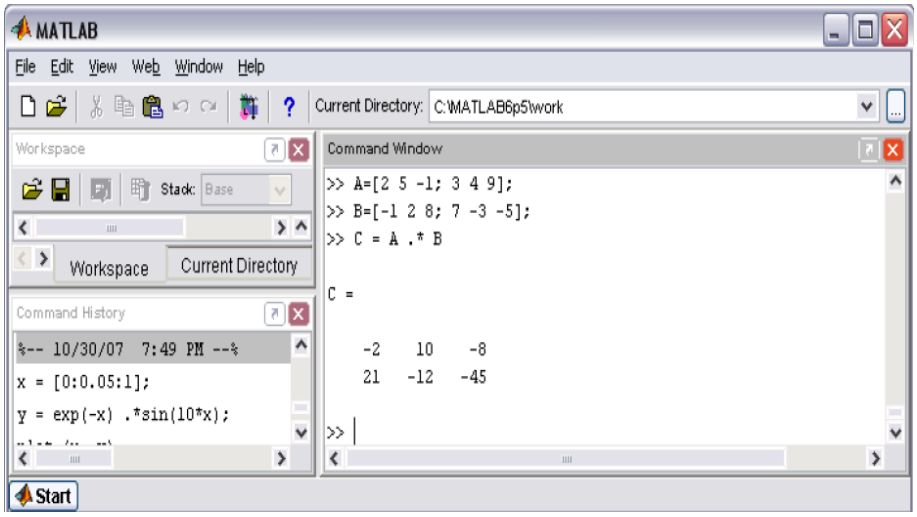


Рис. 7.3

Для деления элементов первой матрицы на соответствующие элементы второй используется ./, а для деления элементов второй матрицы на соответствующие элементы первой служит .\ (рис. 7.4).

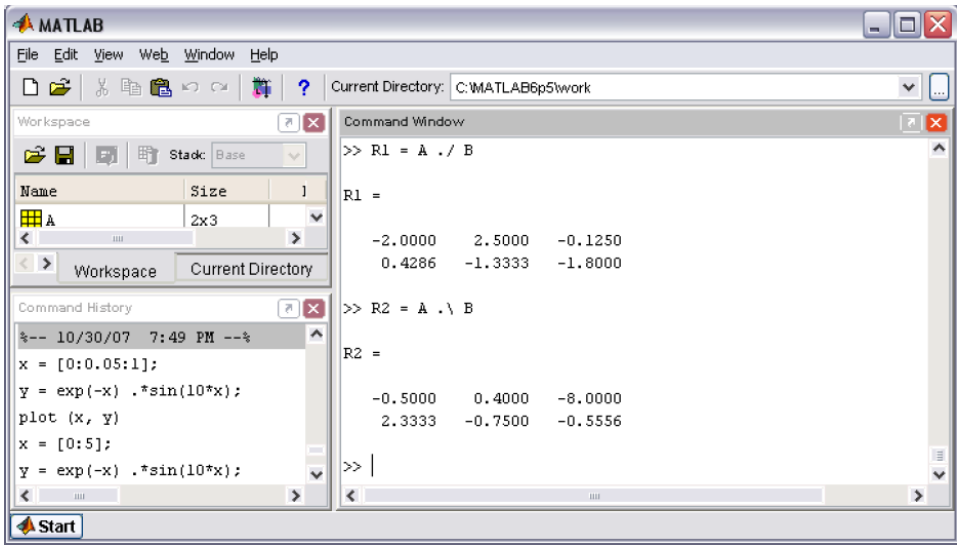


Рис. 7.4

Поэлементное возведение в степень осуществляется при помощи оператора .^ (рис. 7.5).

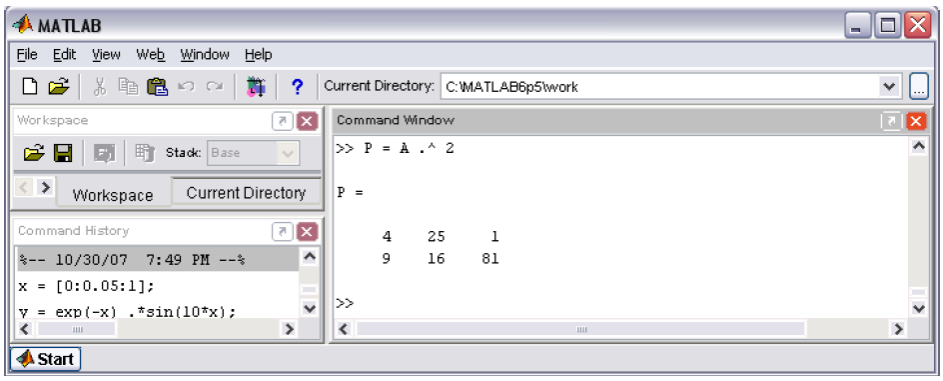


Рис. 7.5

Показатель степени может быть матрицей того же размера, что и матрица, возводимая в степень. При этом элементы первой матрицы возводятся в степени, равные элементам второй матрицы (рис. 7.6).

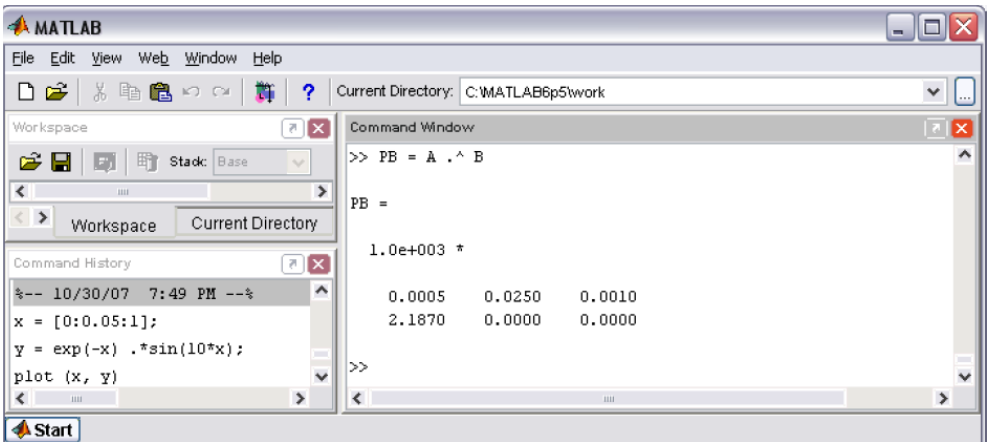


Рис. 7.6

Обратите внимание на форму вывода результата. Во-первых, выделен общий множитель 1.0e+003 для всех элементов результирующей матрицы, т.е. ответ выглядит так:

$$PB = 10^3 \cdot \begin{pmatrix} 0.0005 & 0.0250 & 0.0010 \\ 2.1870 & 0.0000 & 0.0000 \end{pmatrix} = \begin{pmatrix} 0.5 & 25 & 1 \\ 2187 & 9 & 0 \end{pmatrix}.$$

Во-вторых, при возведении 4^{-3} и 9^{-5} получились нули, т. е. произошла потеря точности из-за того, что использовался формат **short**. Дело в том, что $4^{-3} \approx 0.0156$, $9^{-5} \approx 0.000016935$, а эти числа малы по сравнению с остальными, и допустимого числа знаков формата **short** не хватает для их отображения на экране. Если мы хотим получить более точный результат поэлементного возведения в степень, то мы должны задать формат **long** и затем вывести PB снова (рис. 7.7).

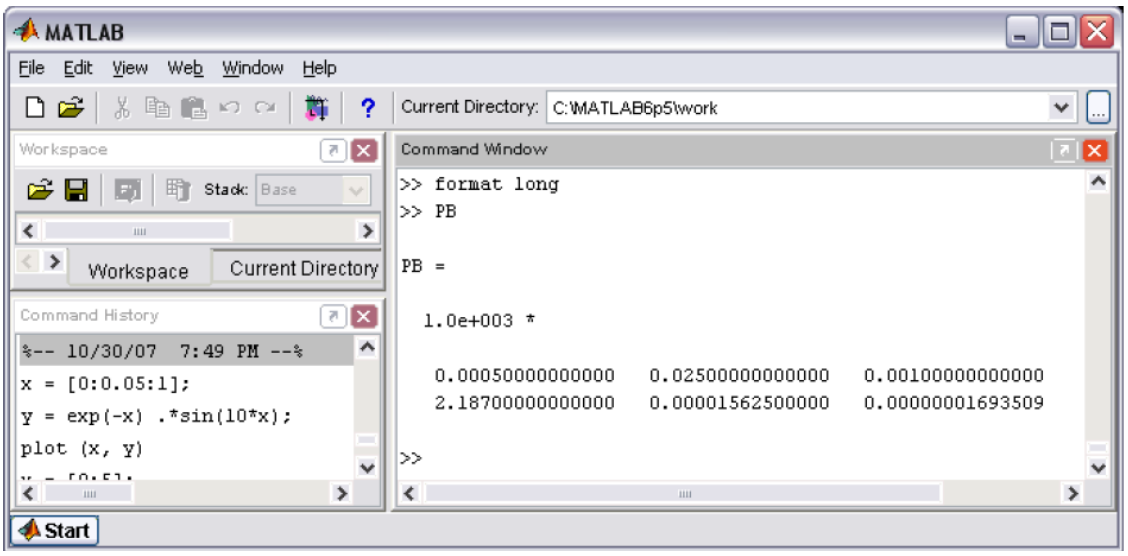


Рис. 7.7

Обратите внимание, что повторного вычисления элементов матрицы PB не потребовалось вне зависимости от установленного формата вывода всех вычислений математических функций от элементов матриц.

Вычисление математических функций от элементов матриц

Очень важно понять, что в книгах по теории матриц формула $\cos A$, где A – квадратная матрица, означает вычисление косинуса от матрицы, которое осуществляется при помощи разложения в ряд. В MatLab имеется возможность вычисления функций от матриц.

Запись $C = \cos A$ приводит к вычислению косинусов от элементов массива A и записи их в массив C того же размера, что и A.

Нахождение, например, косинусов от элементов матрицы

$$A = \begin{pmatrix} \pi/2 & -\pi/2 & 0 \\ \pi & -\pi & 2\pi \\ 0 & 2\pi & \pi/3 \end{pmatrix}$$

производится при помощи команд, указанных на рис. 7.8.

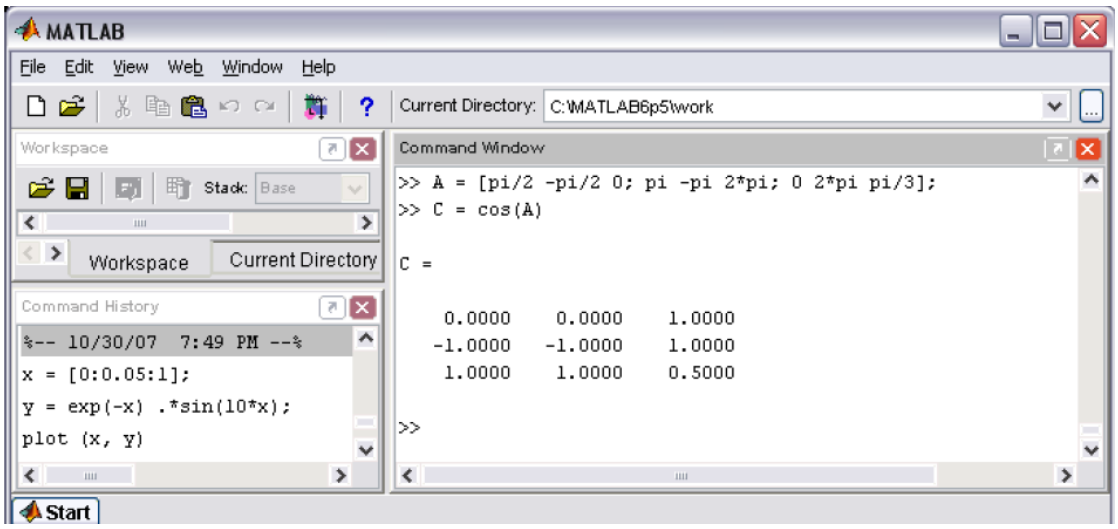


Рис. 7.8

Аналогично вычисляются и другие математические функции. Использование функций обработки данных для матриц (нахождение максимума, минимума, суммы и др.) несколько отличается от их применения при работе с векторами.

7.3. Применение функций обработки данных к матрицам

Функция **sum** вычисляет сумму элементов вектора. С другой стороны, векторы в MatLab так же как, и матрицы, хранятся в двумерных массивах. Возникает вопрос, что же будет, если в качестве аргумента **sum** использовать не вектор, а матрицу. Оказывается, MatLab вычислит вектор-строку, длина которой равна числу столбцов матрицы, а каждый элемент является суммой соответствующего столбца (рис. 7.9).

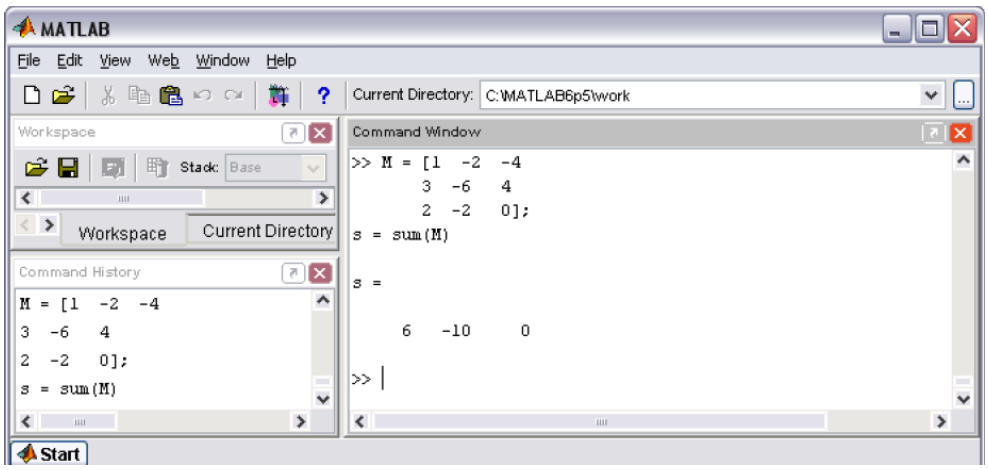


Рис. 7.9

Функция **sum** по умолчанию вычисляет сумму по столбцам, изменяя первый индекс массива при фиксированном втором. Для того чтобы производить суммирование по строкам, необходимо вызвать функцию **sum** с двумя аргументами, указав место индекса, по которому следует суммировать (рис. 7.10).

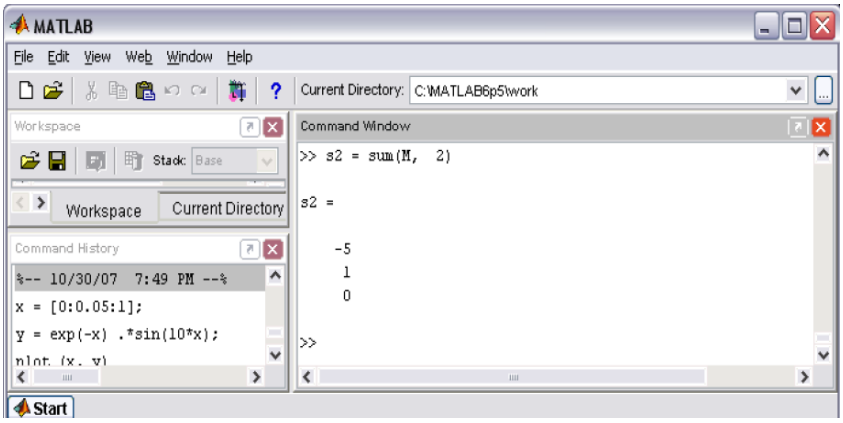


Рис. 7.10

Заметьте, что sum(M) и sum(M, 1) приводят к одинаковым результатам. Итак, функция **sum** суммирует или по строкам, или по столбцам, выдавая результат в виде вектора или вектор-строки. Аналогично работает и функция **prod** (рис. 7.11).

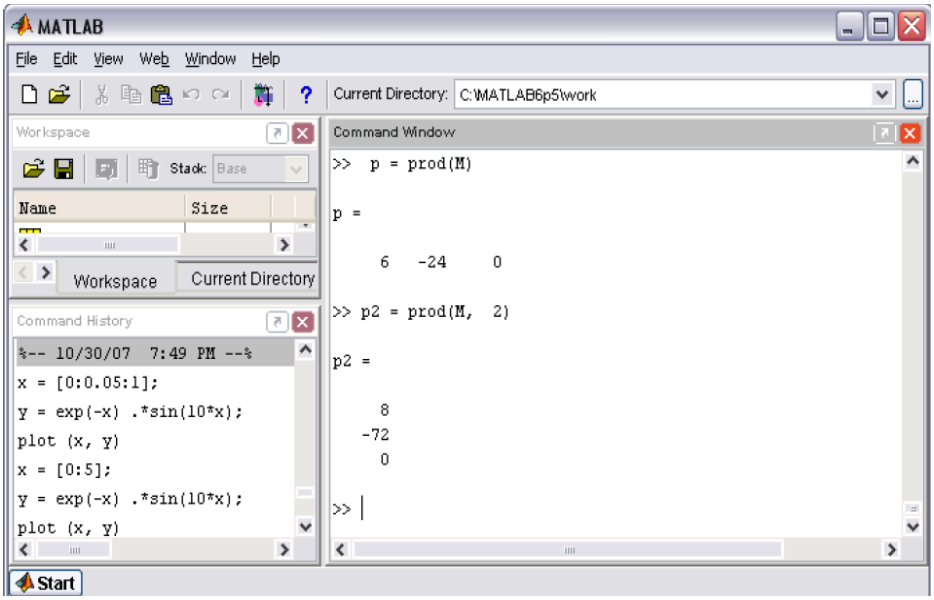


Рис. 7.11

Функция **sort** упорядочивает элементы каждого из столбцов матрицы в порядке возрастания. Вызов **sort** со вторым аргументом, равным двум, приводит к упорядочению элементов строк (рис. 7.12).

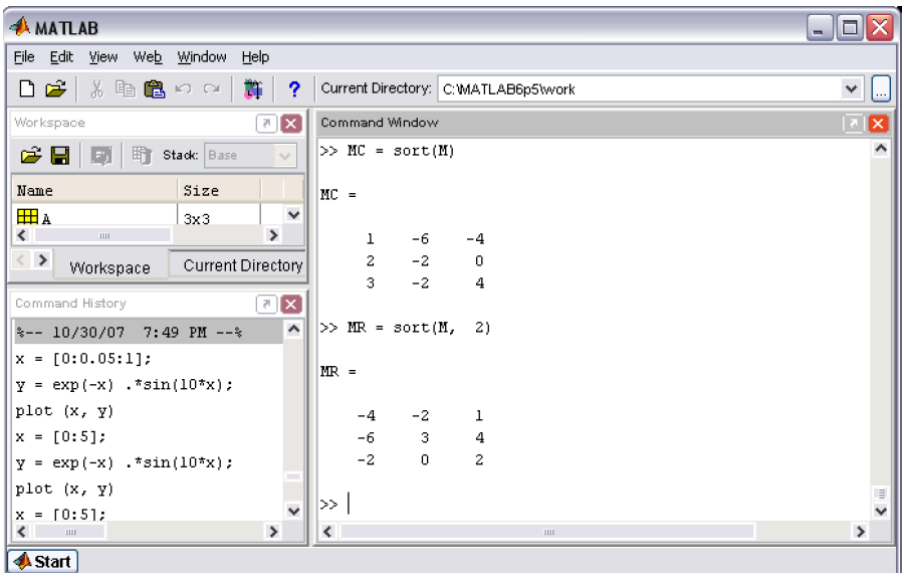


Рис. 7.12

Так же как и для векторов, функция **sort** позволяет получить матрицу индексов соответствия элементов исходной и упорядоченной матриц. Для этого необходимо вызвать **sort** с двумя выходными аргументами (рис. 7.13).

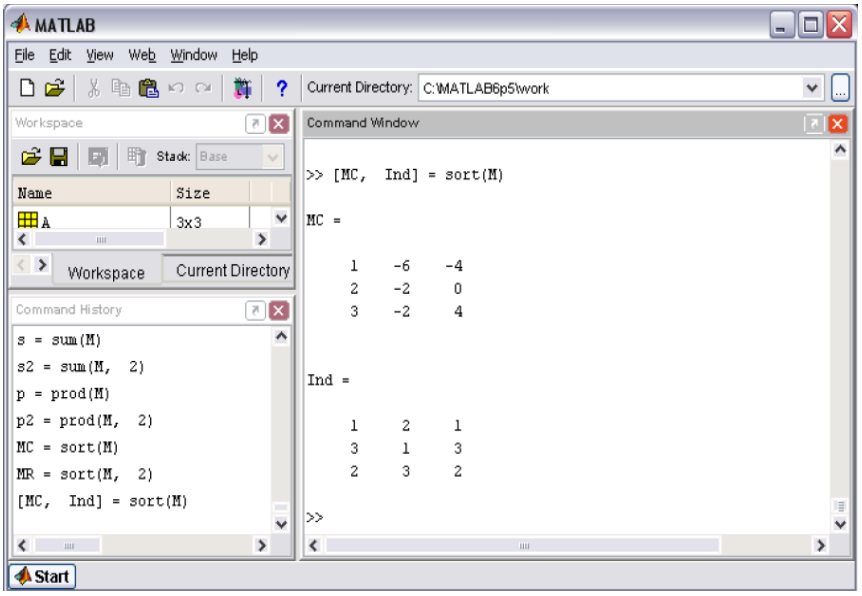


Рис. 7.13

Матрицы M, MC и Ind связаны между собой так: MC(i, j) = M(Ind(i, j), j) , где i и j изменяются от одного до трех.

Функции **max** и **min** вычисляют вектор-строку, содержащую максимальные или минимальные элементы в соответствующих столбцах матрицы (рис. 7.14).

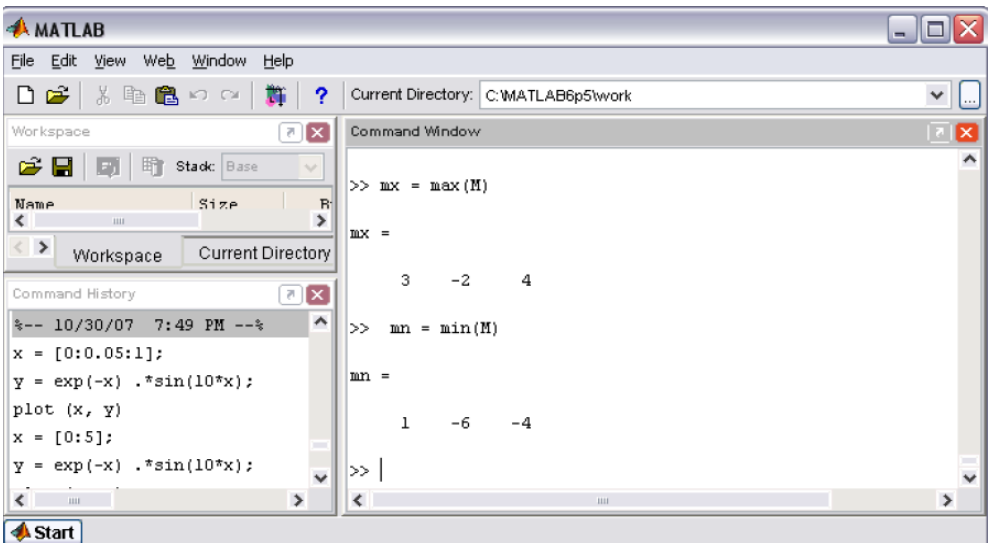


Рис. 7.14

Для того чтобы узнать не только значения максимальных или минимальных элементов, но и их номера в столбцах, следует вызвать **max** или **min (**рис. 7.15).

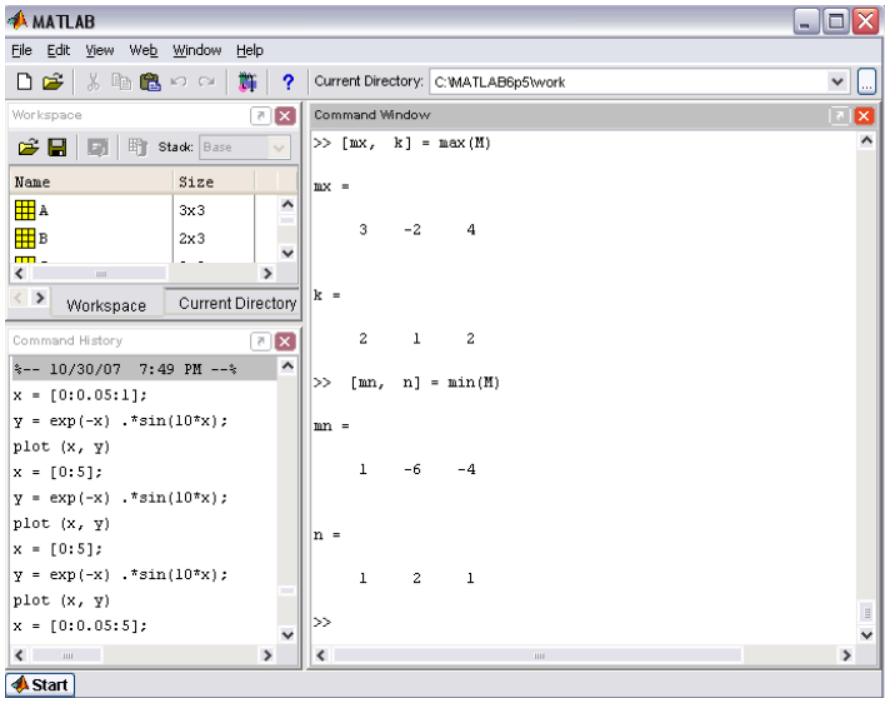


Рис. 7.15

Обратите внимание, что во втором столбце матрицы M два равных максимальных элемента – первый и третий. Всегда возвращается номер первого максимального элемента (второй элемент в вектор-строке равен единице). Точно так же работает и **min** в случае двух равных минимальных элементов в столбце матрицы. Функции **max** и **min** позволяют выделить максимальные или

минимальные элементы из двух матриц одинаковых размеров и записать результат в новую матрицу того же размера, что и исходные (рис. 7.16).

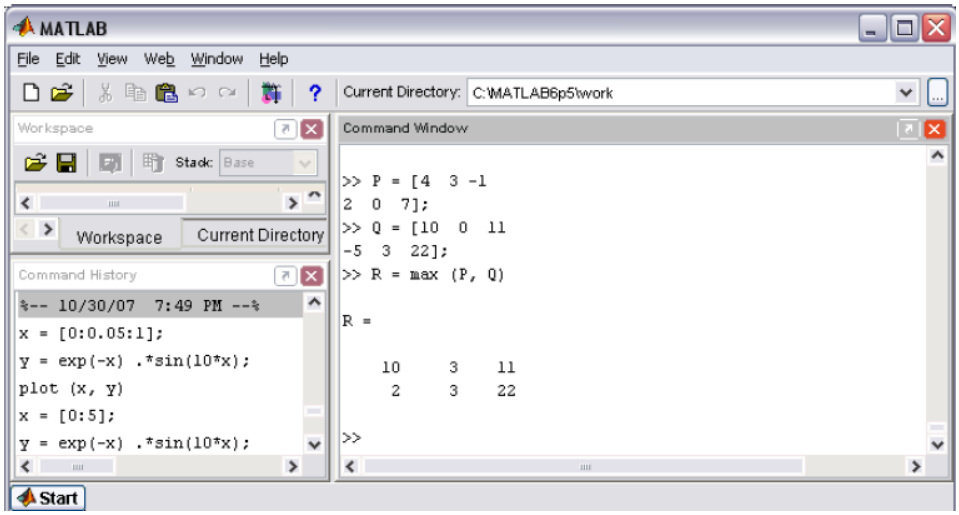


Рис. 7.16

Одним из документов может быть число. В результирующую матрицу записывается максимум из этого числа и соответствующего элемента исходной матрицы (рис. 7.17).

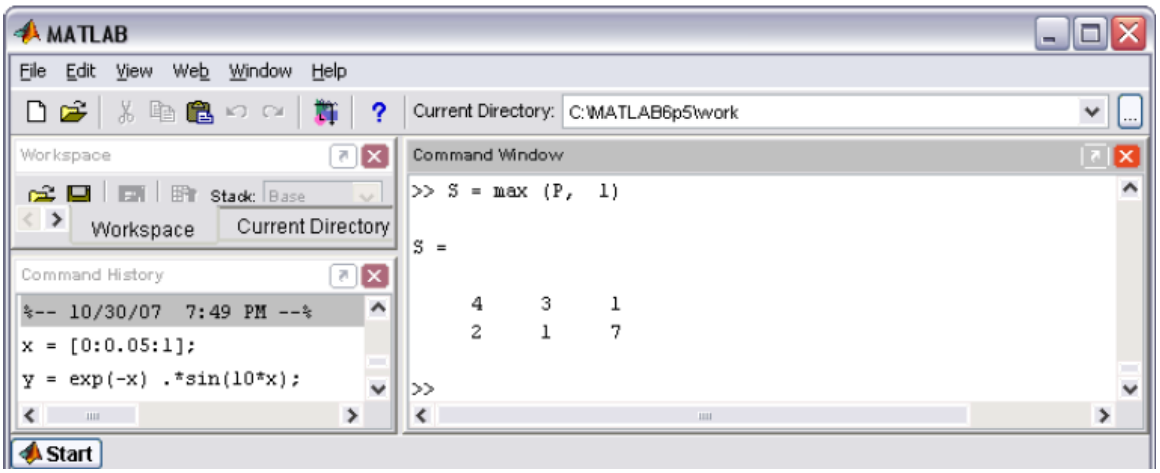


Рис. 7.17

Если оба аргумента функций **min** или **max** являются числами, то возвращается минимальное или максимальное из этих чисел. Для нахождения максимума или минимума не по столбцам матрицы, а по строкам предусмотрена форма вызова со вторым аргументом – пустым массивом (рис. 7.18).

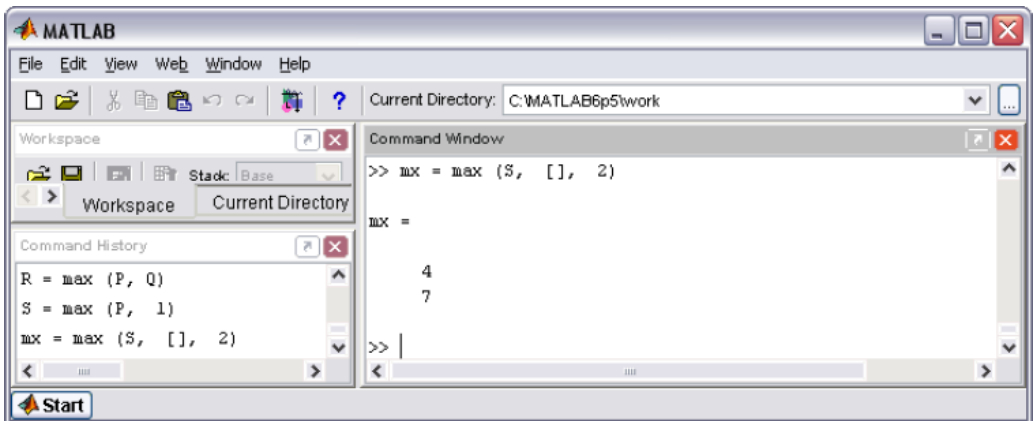


Рис. 7.18

Для того чтобы дополнительно получить номера максимальных элементов в строках, используем вызов **max** с двумя выходными аргументами (рис. 7.19).

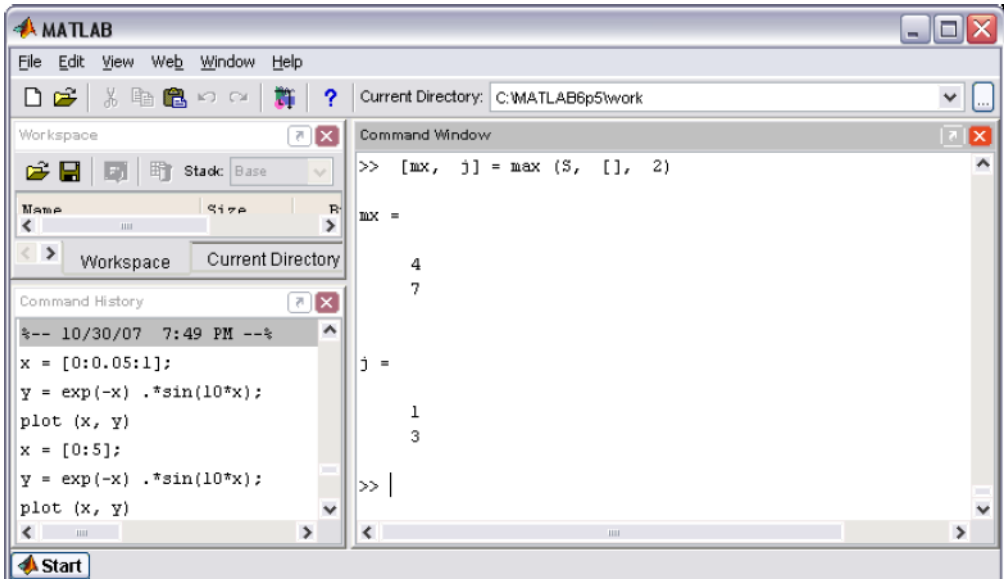


Рис. 7.19

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

Проделайте следующее упражнение, для проверки знаний о применении к матрицам математических функций и функций обработки данных. Задана квадратная матрица A размера n с элементами a_{jk}. Требуется вычислить приведенные ниже величины (нормы матрицы):

$$p = \max_{1\le i\le n}\sum_{k=1}^{n}|a_{jk}|,\ q = \max_{1\le k\le n}\sum_{i=1}^{n}|a_{ik}|,\ r = \left(\sum_{i,k=1}^{n}a_{ik}^2\right)^{1/2},\ s = \sum_{i,k=1}^{n}|a_{ik}|$$

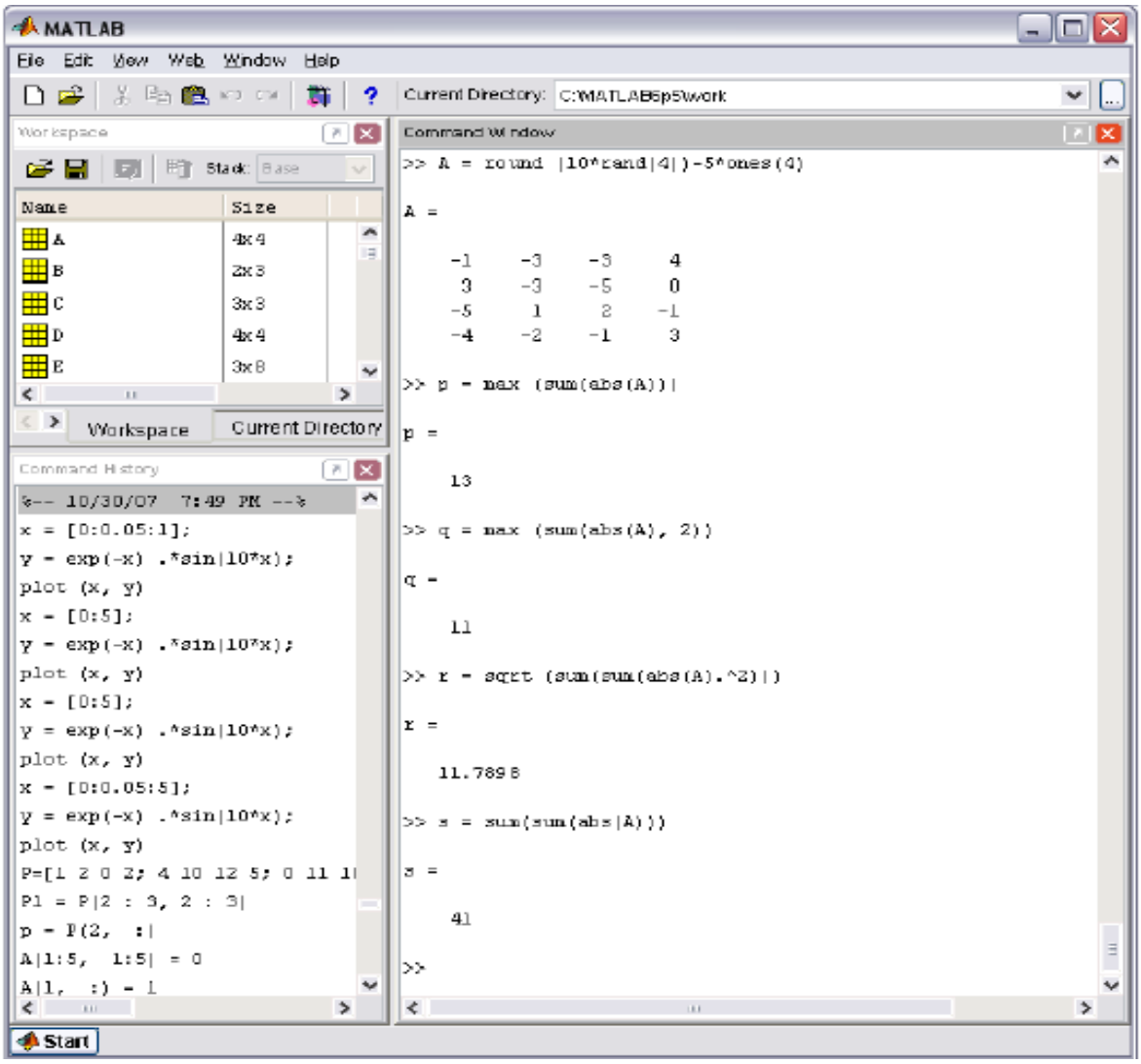


Рис. 7.20

Создайте квадратную матрицу размера четыре, состоящую из случайных целых чисел от нуля до десяти, вычтите из нее матрицу, состоящую из пятерок, и введите необходимые выражения для вычисления норм полученной матрицы. Проверьте себя, выполнив действия, указанные на рис. 7.20.

7.4. Графики двух переменных

Построение графика функции двух переменных в MatLab на прямоугольной области определения переменных включает два предварительных этапа:

1. Разбиение области определения прямоугольной сеткой.

2. Вычисление значений функции в точках пересечения линий сетки и запись их в матрицу.

Построим график функции $z(x,y)=x^2+y^2$ на области определения в виде квадрата x∈[0,1], y∈[0,1] . Необходимо разбить квадрат равномерной сеткой (например, с шагом 0.2) так, как показано на рис. 7.21, и вычислить значения функций в узлах, обозначенных точками.

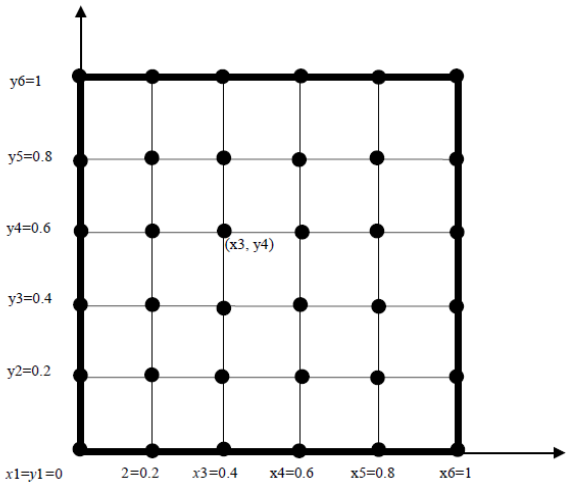
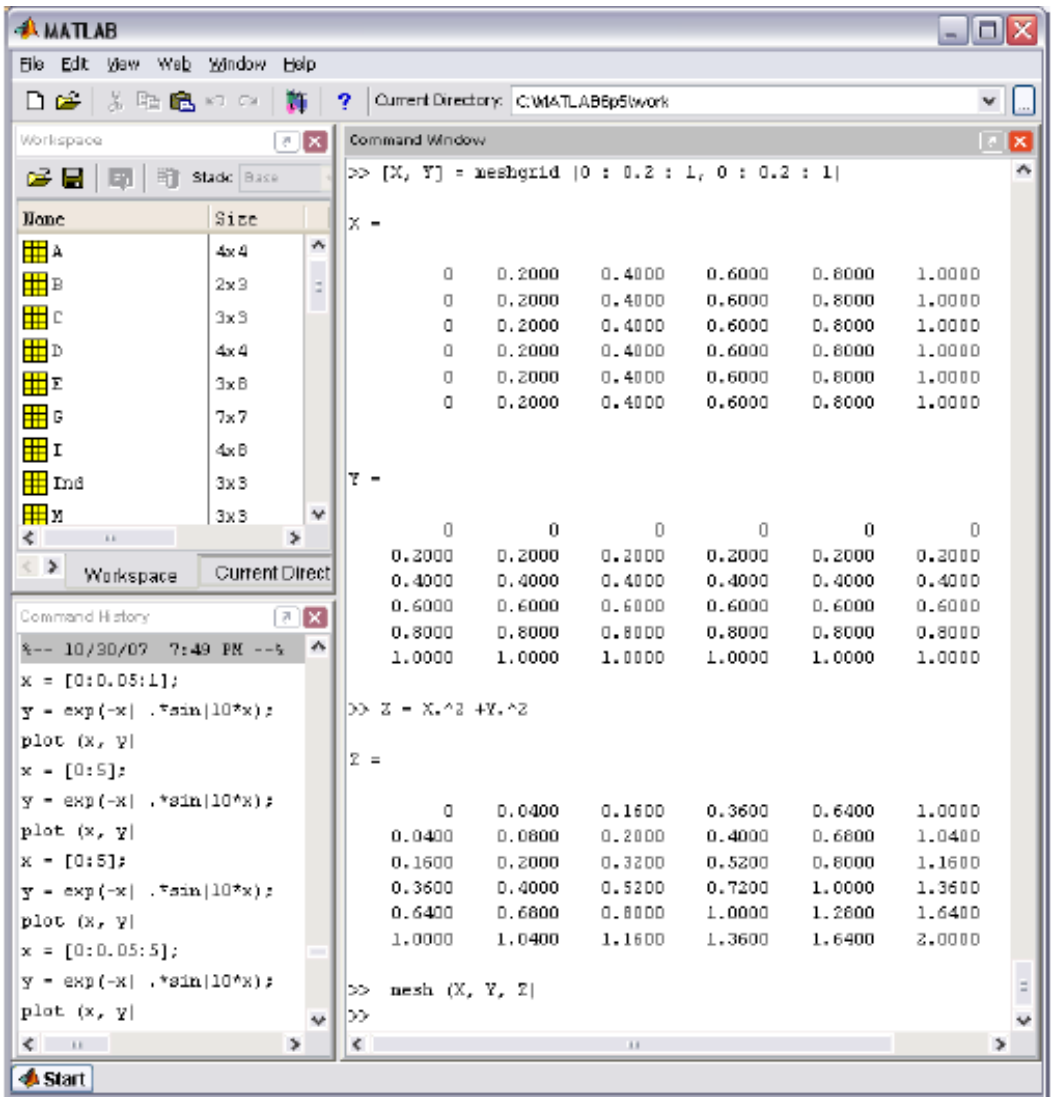


Рис. 7.21

Удобно использовать два двумерных массива *x* и *y* размерностью шесть на шесть для хранения информации о координатах узлов. Массив *x* состоит из одинаковых строк, в которых записаны координаты *x*1, *x*2, …, *x*6, а *y* содержит одинаковые столбцы с *y*1, *y*2, …, *y*6. Значения функции в узлах сетки запишем в матрицу *z* такой же размерности (6×6), причем для вычисления матрицы *Z* используем выражение для функции, но с *поэлементными* матричными операциями. Тогда, например, *z*(3, 4) как раз будет равно значению функции *z*(*x*, *y*) в точке (*x*3, *y*4). Для генерации массивов сетки *x* и *y* по координатам узлов в MatLab предусмотрена функция **meshgrid**, для построения графика в виде каркасной поверхности – функция **mesh**. Следующие операторы приводят к появлению на экране окна с графиком функции (точка с запятой в конце операторов не ставится для того,чтобы проконтролировать генерацию массивов) – см. рис. 7.22.

а)


MATLAB
File Edit View Web Window Help
Current Directory: C:\MATLAB6p5\work
Workspace
Stack: Base
Name Size
A 4x4
B 2x3
C 3x3
D 4x4
E 3x8
G 7x7
I 4x8
Ind 3x3
M 3x3
Workspace
Current Direct
Command History
%-- 10/30/07 7:49 PM --%
x = [0:0.05:1];
y = exp(-x) .*sin(10*x);
plot (x, y)
x = [0:5];
y = exp(-x) .*sin(10*x);
plot (x, y)
x = [0:5];
y = exp(-x) .*sin(10*x);
plot (x, y)
x = [0:0.05:5];
y = exp(-x) .*sin(10*x);
plot (x, y)
Command Window
>> [X, Y] = meshgrid [0 : 0.2 : 1, 0 : 0.2 : 1]
X =
0 0.2000 0.4000 0.6000 0.8000 1.0000
0 0.2000 0.4000 0.6000 0.8000 1.0000
0 0.2000 0.4000 0.6000 0.8000 1.0000
0 0.2000 0.4000 0.6000 0.8000 1.0000
0 0.2000 0.4000 0.6000 0.8000 1.0000
0 0.2000 0.4000 0.6000 0.8000 1.0000
Y =
0 0 0 0 0 0
0.2000 0.2000 0.2000 0.2000 0.2000 0.2000
0.4000 0.4000 0.4000 0.4000 0.4000 0.4000
0.6000 0.6000 0.6000 0.6000 0.6000 0.6000
0.8000 0.8000 0.8000 0.8000 0.8000 0.8000
1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
>> Z = X.^2 +Y.^2
Z =
0 0.0400 0.1600 0.3600 0.6400 1.0000
0.0400 0.0800 0.2000 0.4000 0.6800 1.0400
0.1600 0.2000 0.3200 0.5200 0.8000 1.1600
0.3600 0.4000 0.5200 0.7200 1.0000 1.3600
0.6400 0.6800 0.8000 1.0000 1.2800 1.6400
1.0000 1.0400 1.1600 1.3600 1.6400 2.0000
>> mesh (X, Y, Z)
>>
Start

б)

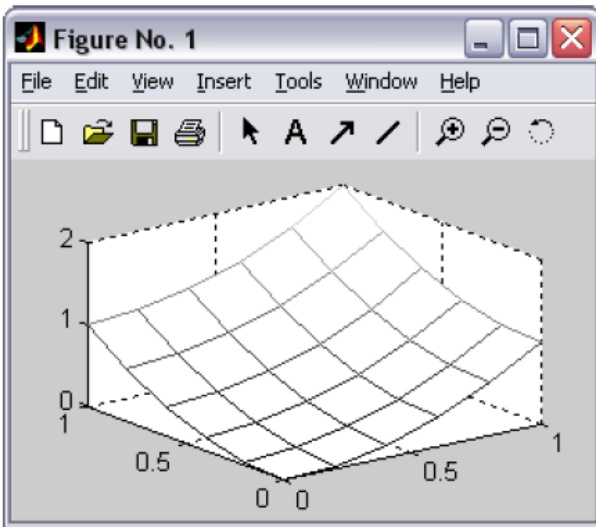


Figure No. 1
File Edit View Insert Tools Window Help
2
1
0
1
0.5
0
0
0.5
1

Рис. 7.22

8. Диаграммы и гистограммы

Наглядным способом представления векторных и матричных данных являются диаграммы и гистограммы. Значение элемента вектора пропорционально высоте столбика диаграммы (в случае столбчатой диаграммы) или площади сектора диаграммы (для круговой диаграммы). Гистограммы используются для получения информации о распределении данных по заданным интервалам.

8.1. Представление векторных данных

Диаграммы векторных данных

Отображение вектора в виде столбчатой диаграммы осуществляется функцией **bar**. Запишите, например, вектор-строку

$$x = [1.2 \quad 1.7 \quad 2.2 \quad 2.4 \quad 2.5 \quad 1.3 \quad 1.1 \quad 0.5 \quad 0.4 \quad 0.1]$$

в переменную **data** и представьте ее столбчатой диаграммой, вызвав функцию **bar** с аргументом *x* (рис. 8.1).

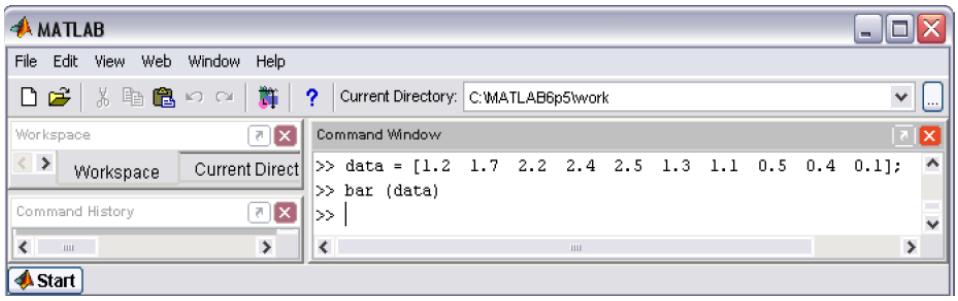


Рис. 8.1

На экране появится графическое окно, содержащее столбчатую диаграмму вектор-строки. По горизонтальной оси откладывается номер элемента вектора, а по вертикальной – его значение. Аргументом функции **bar** может быть как вектор-строка, так и вектор-столбец (рис. 8.2).

Разметку горизонтальной оси можно задать вектором с возрастающими значениями. В этом случае первый аргумент **bar** является вектором с координатами точек разметки, а второй – вектором значений, которые требуется отобразить на диаграмме. Удобно использовать этот способ

построения диаграммы для отображения значений элементов векторов в зависимости не от их номера, а например, от времени, если в вектор записаны результаты измерений в некоторые моменты времени. Важно, чтобы длины этих векторов совпадали, иначе будет выдано сообщение об ошибке (рис. 8.3).

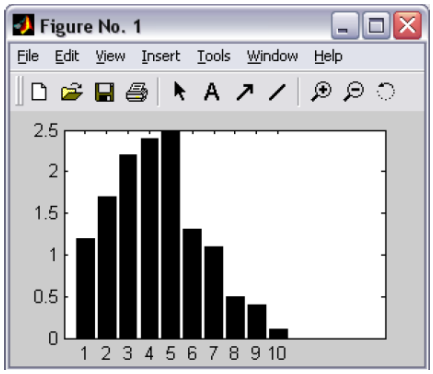


Рис. 8.2

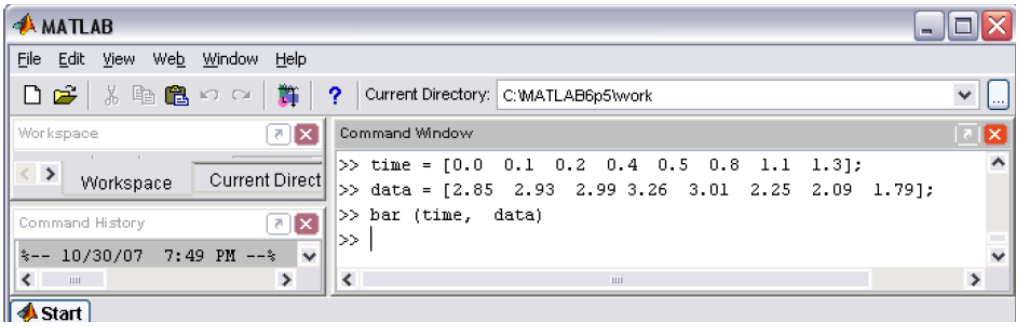


Рис. 8.3

На рис. 8.4пропущенные столбики соответствуют тем моментам времени, в которые измерения не производились.

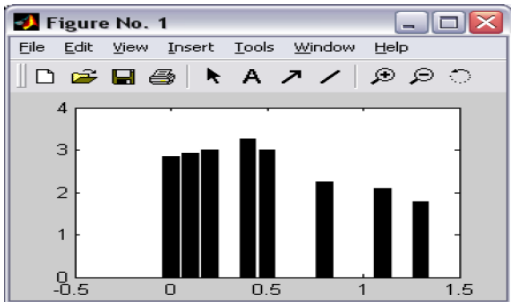
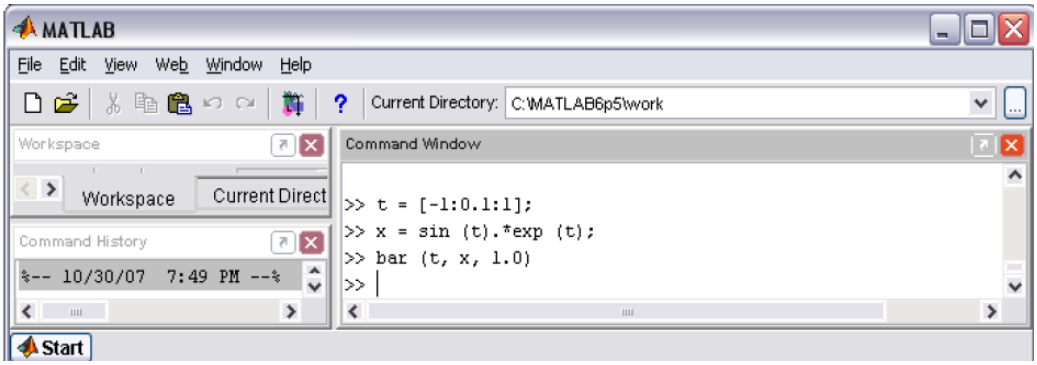


Рис. 8.4

Выбор ширины столбцов осуществляется заданием третьего дополнительного аргумента. По умолчанию ширина равна 0.8. Диаграмма без промежутков между столбиками получается, если установить ширину, равную единице. Выбор значений, больших единицы, приводит к перекрывающимся столбикам. В качестве примера отобразите функцию $x(t) = \sin t \cdot e^t$ на отрезке [–1,

1] в виде столбчатой диаграммы без промежутков, выполнив последовательность операций (рис. 8.5).

а)



б)

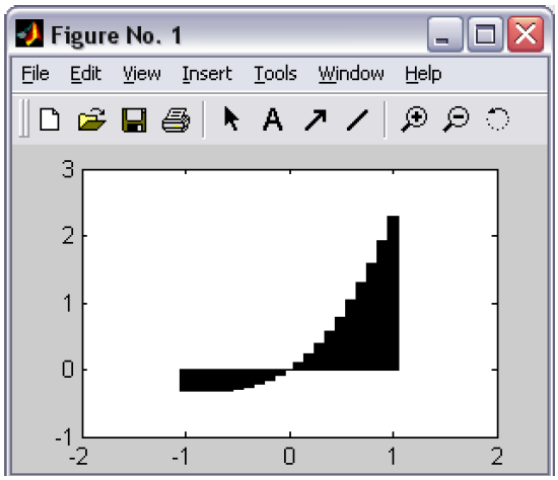
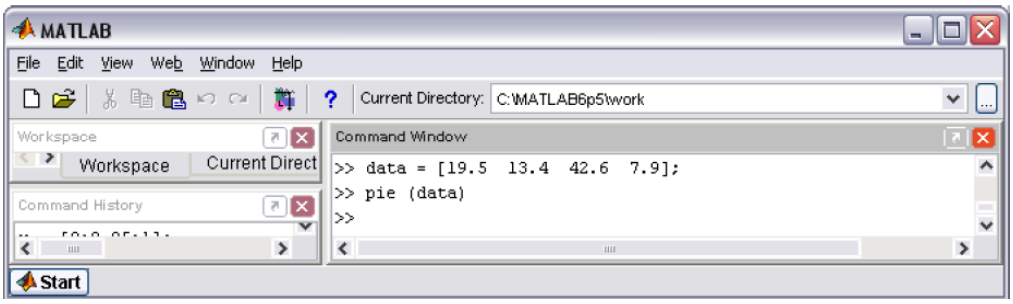


Рис. 8.5

Функция **barh** строит горизонтальную столбчатую диаграмму, т.е. повернутую на 90°. Для построения объемных диаграмм применяется функция **bar3**. Использование **barh** и **bar3** аналогично **bar**.

Если требуется оценить вклад каждого из элементов вектора в общую сумму его элементов, то удобно построить круговую диаграмму при помощи функции **pie** (рис. 8.6.)

а)



б)

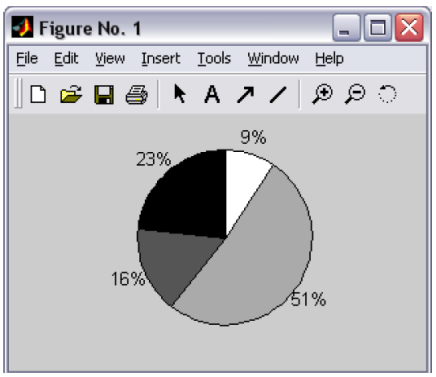


Рис. 8.6

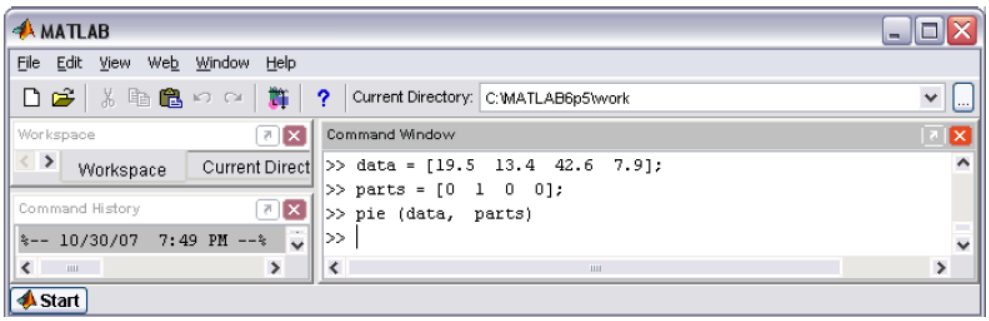
В результате получается диаграмма, в которой площади секторов отвечают процентному вкладу каждого из элементов вектора в общую сумму, т.е. MatLab *нормирует* значения, вычисляя

data / sum (data)

Если сумма элементов вектора (аргумента **pie**) больше или равна единице, то MatLab производит нормировку и строит круг, состоящий из секторов. Если сумма меньше единицы, то нормировка не производится и получается круг с пропущенным сектором.

Часто необходимо отодвинуть от круга диаграммы сектор, соответствующий некоторому элементу. Необходимо задать вторым аргументом функции **pie** вектор, состоящий из единиц и нулей, причем единица стоит в позиции, соответствующей номеру отделяемой части (рис. 8.7).

а)



б)

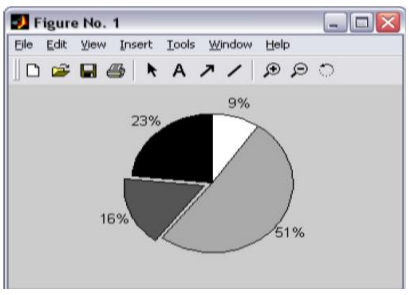
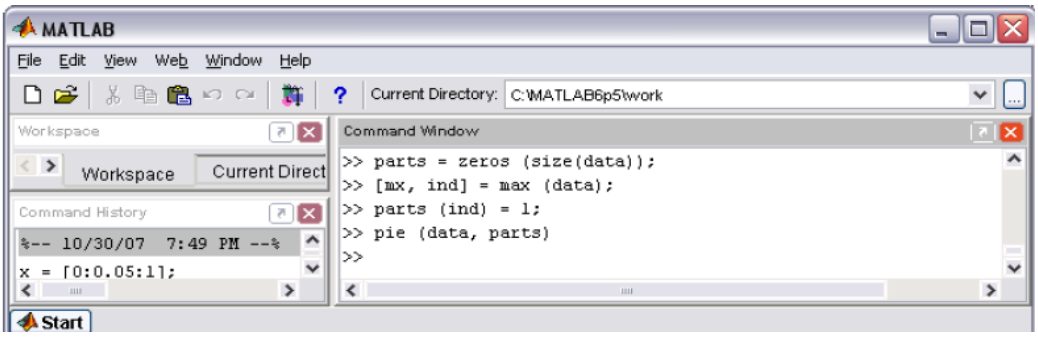


Рис. 8.7

Можно отделить несколько секторов, расположив единицы во вспомогательном векторе на подходящих позициях. Важно, чтобы размеры векторов были одинаковы.

В качестве упражнения напишите команды построения диаграммы с отдельным сектором, соответствующим максимальному значению среди элементов вектора, автоматически создав вспомогательный вектор. Используйте функции **zeros** для создания нулевого вектора, той же длины, что *x*, и **max** с двумя выходными аргументами для поиска номера максимального элемента в векторе *x*. На рис. 8.8 приведена требуемая последовательность команд для построения диаграммы с отдельным сектором.

а)



б)

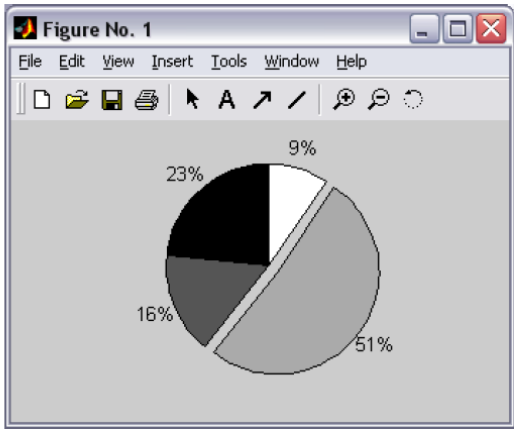
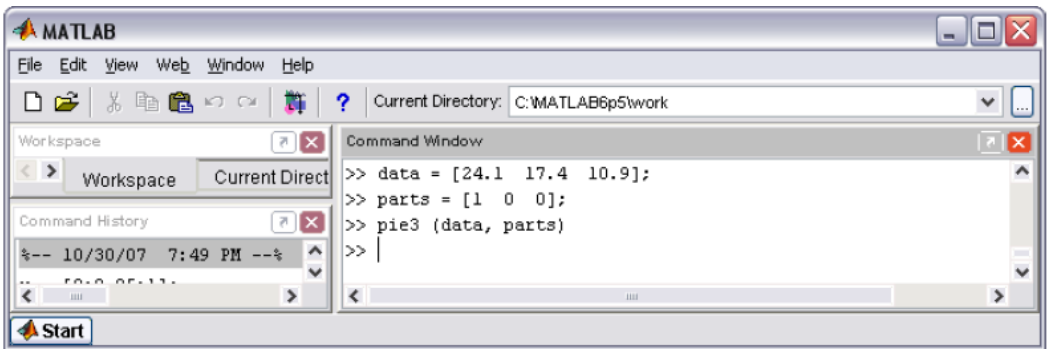


Рис. 8.8.

Визуализация векторных данных может быть осуществлена при помощи команд **pie3** и **bar3**, которые строят трехмерные круговые и столбчатые диаграммы. Например, команды, показанные на рис. 8.9, приводят к появлению трехмерной круговой диаграммы с отделенным сектором.

а)



б)

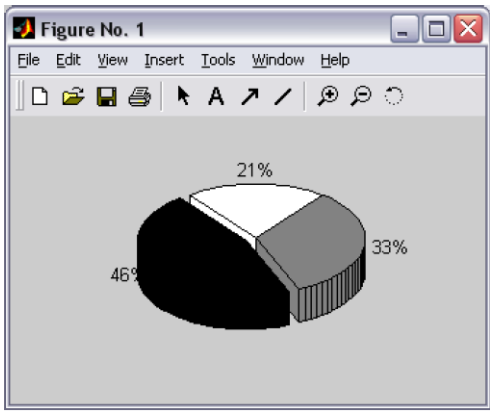
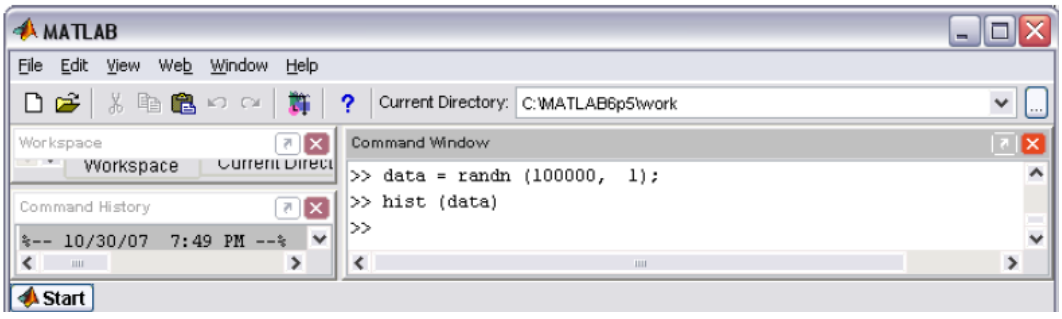


Рис. 8.9

Гистограммы векторных данных

Обработка данных включает вопрос о том, сколько данных попало в тот или иной интервал. Для получения наглядного представления о распределении данных служит функция **hist**. Например, команды, показанные на рис. 8.10, заполняют вектор *x* числами, распределенными по нормальному закону, разбивают интервал, которому они принадлежат, на десять равных частей (по умолчанию) и строят гистограмму попадания чисел в каждый из интервалов.

а)



б)

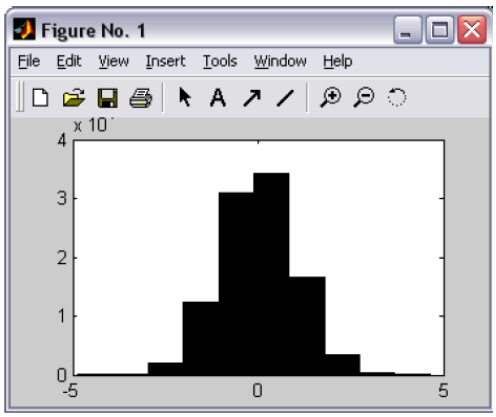
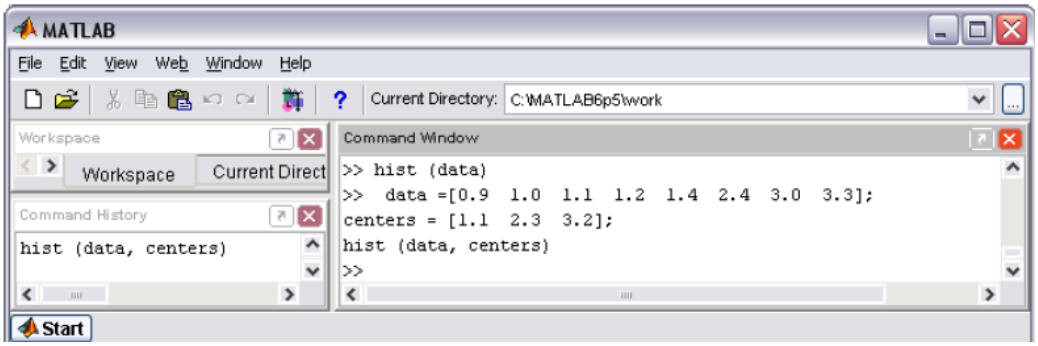


Рис. 8.10

Для увеличения числа интервалов следует в качестве второго аргумента указать число интервалов, например, **hist (data, 50)**. Вместо автоматического разбиения на равные интервалы можно использовать собственное, задав вторым аргументом вектор, содержащий центры интервалов, приводящих кпостроению диаграммы, где звездочки на горизонтальной оси отмечают центры интервалов (рис. 8.11).

а)



б)

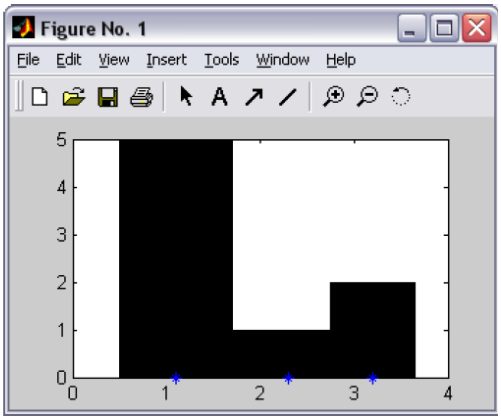
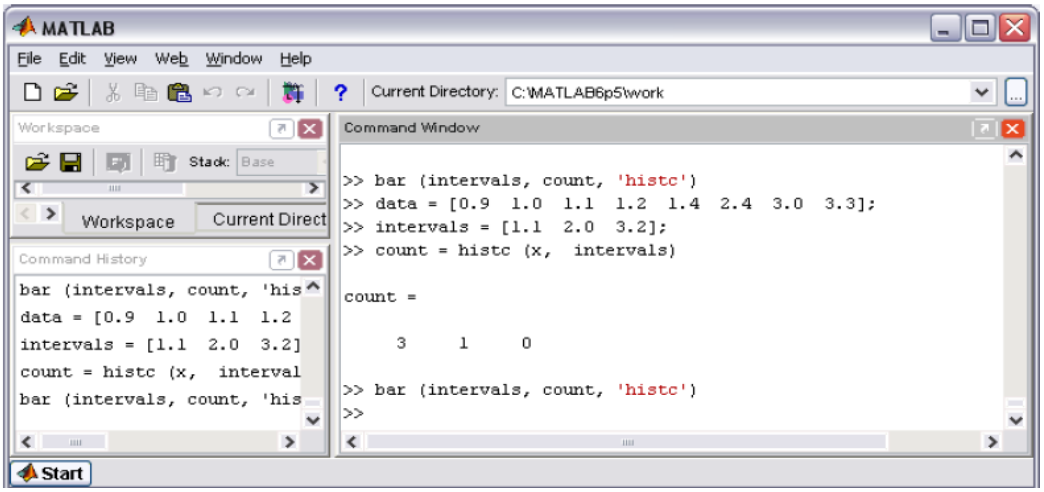


Рис. 8.11

Часто необходимо задать не центры, а границы интервалов. Для построения таких гистограмм следует использовать функцию **histc** в сочетании с вышесказанной функцией **bar**. Функция **histc** возвращает вектор, содержащий

число величин, попавших в заданные интервалы. При помощи функции **bar** с дополнительным аргументом '**histc**' полученный вектор представляется в виде гистограммы (8.12).

а)



б)

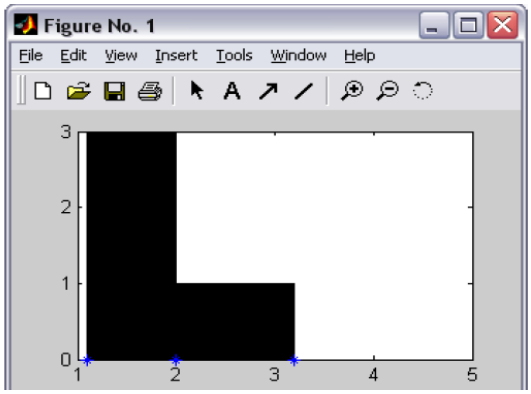
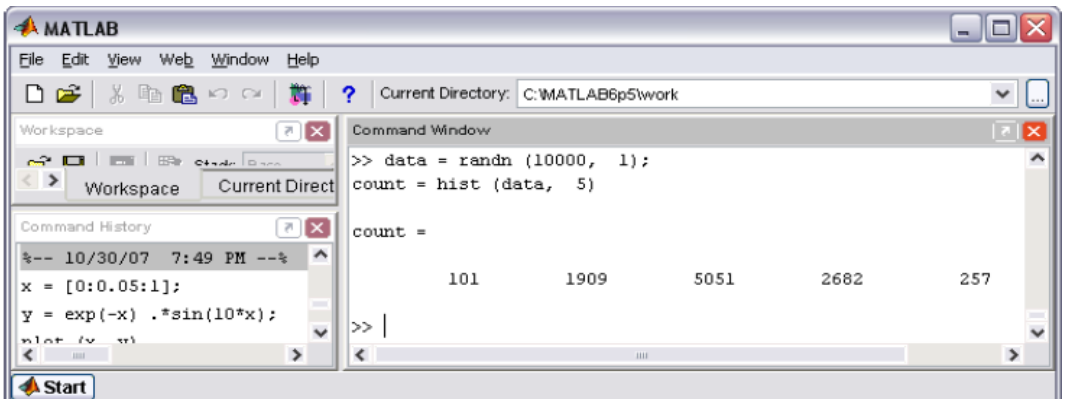


Рис. 8.12

Функцию **hist** можно вызывать с одним или двумя выходными аргументами для получения массивов с информацией о распределении данных, при этом гистограмма не строится. В случае одного аргумента в него записывается вектор, содержащий распределение данных по интервалам. Пример на рис. 8.13демонстрирует создание вектора **count** из пяти элементов, каждый из которых соответствует числу элементов из **data**, попавших в один из пяти интервалов.

а)



б)

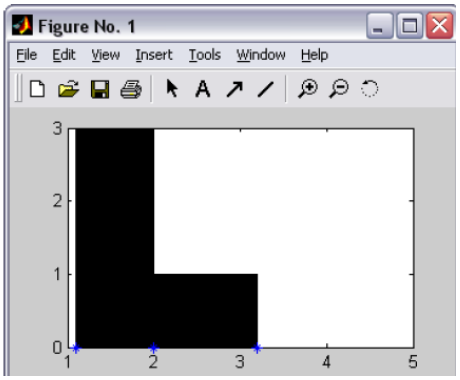


Рис. 8.13

Использование **hist**с двумя аргументами приводит к получению дополнительного вектора с информацией о расположении интервалов (рис. 8.14).

Функция **rose** предназначена для построения угловых гистограмм (в полярных координатах). Аргументом функции **rose** является вектор значений в радианах. Угловые гистограммы дают наглядное представление о данных, связанных с измерениями направлений.

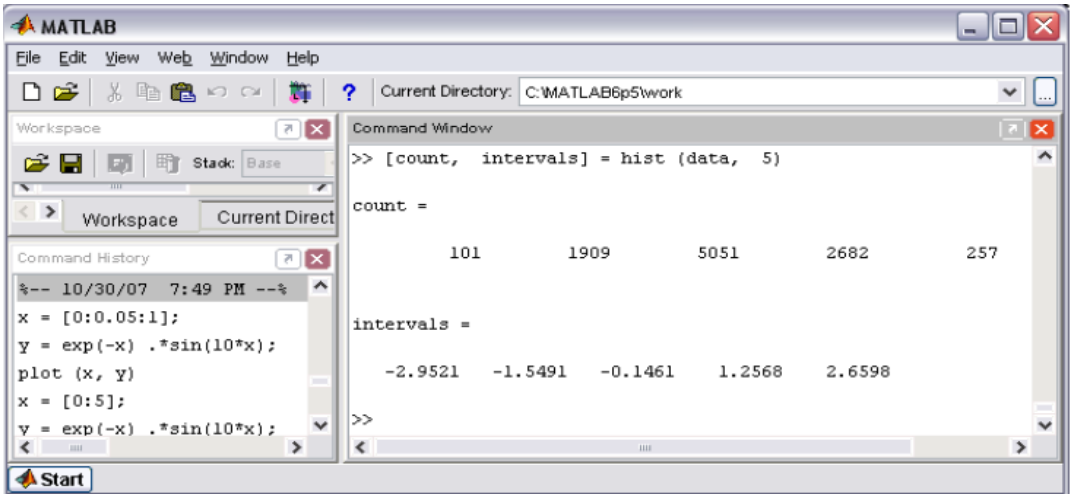


Рис. 8.14

Функция **rose**, так же, как и **hist**, допускает получение информации о распределении по интервалам и границах интервалов при вызове ее с выходными аргументами. Гистограмма в этом случае не отображается.

Если производятся измерения группы величин, то результат представляет собой матрицу. Для отображения матричных данных используются те же функции, что и для векторных данных.

8.2. Представление матричных данных

Предположим, что в матрице DATA, состоящей из четырех строк и трех столбцов содержатся результаты измерений трех величин за четыре момента времени. Для построения столбчатой диаграммы данных примените функцию **bar**, задав в качестве аргумента массив DATA (рис. 8.15).

а)

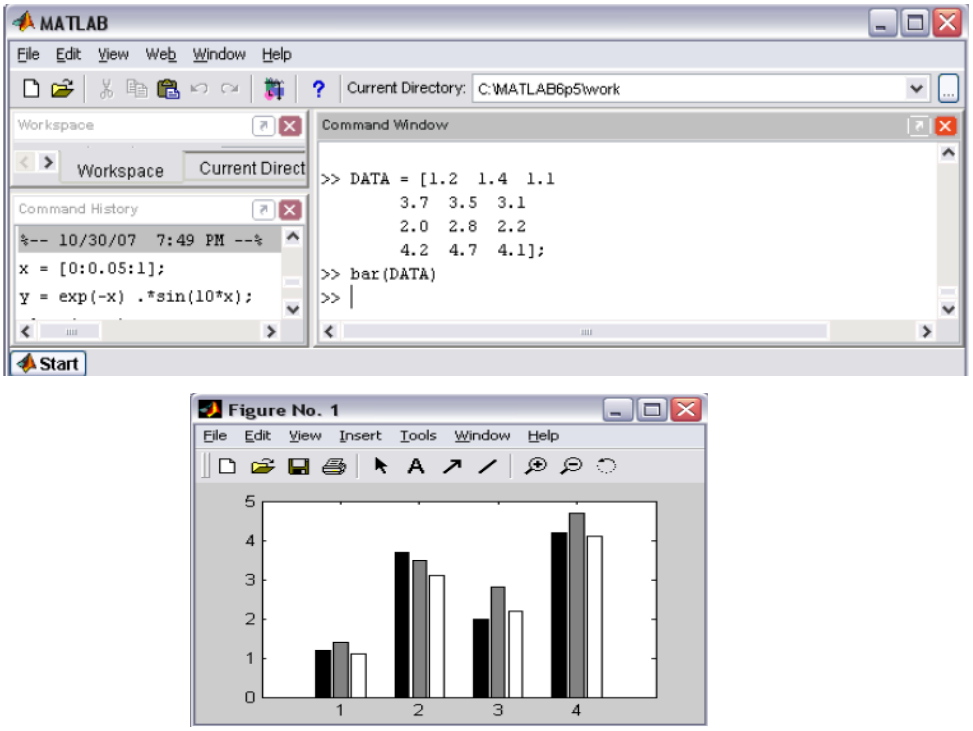


Рис. 8.15

В результате появляется диаграмма сгруппированных данных. На диаграмме расположены четыре группы данных, каждая из которых состоит из трех столбиков, соответствующих измеряемым величинам.

8.3. Графики функций

MatLab предоставляет обширные возможности для визуализации функций одной и двух переменных. Использование функций для построения графиков с минимальным набором задаваемых параметров (остальные MatLab выбирает автоматически) приводит к получению качественных графиков.

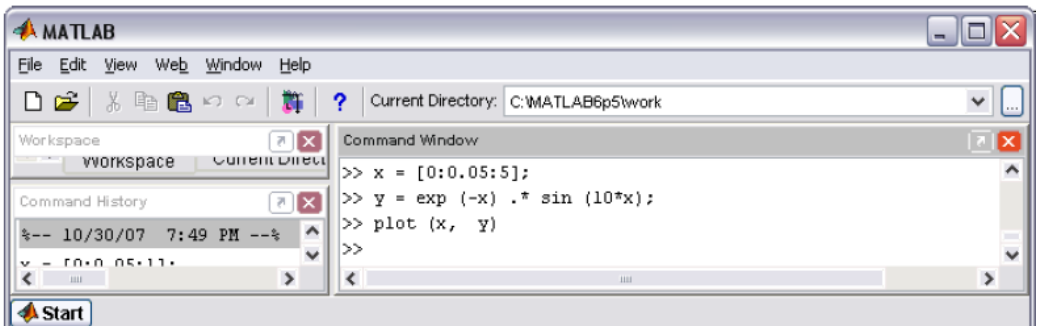
8.4. Графики функций одной переменной

MatLab позволяет строить графики функций в линейном, логарифмическом и полулогарифмическом масштабах. Причем в одном окне можно построить графики нескольких функций, даже определенных на разных отрезках.

Графики в линейном масштабе

Построение графиков функций одной переменной в линейном масштабе осуществляется при помощи функции **plot**. В зависимости от входных аргументов функция **plot** позволяет строить один или несколько графиков, изменять цвет и стиль линий и добавлять маркеры на каждый график (рис. 8.16).

а)



б)

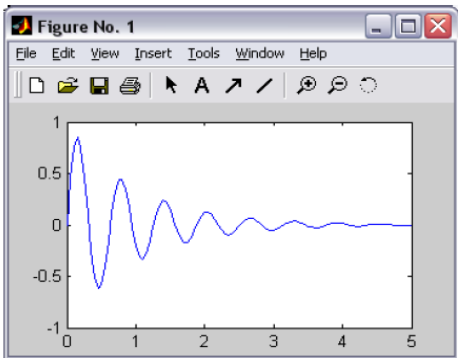
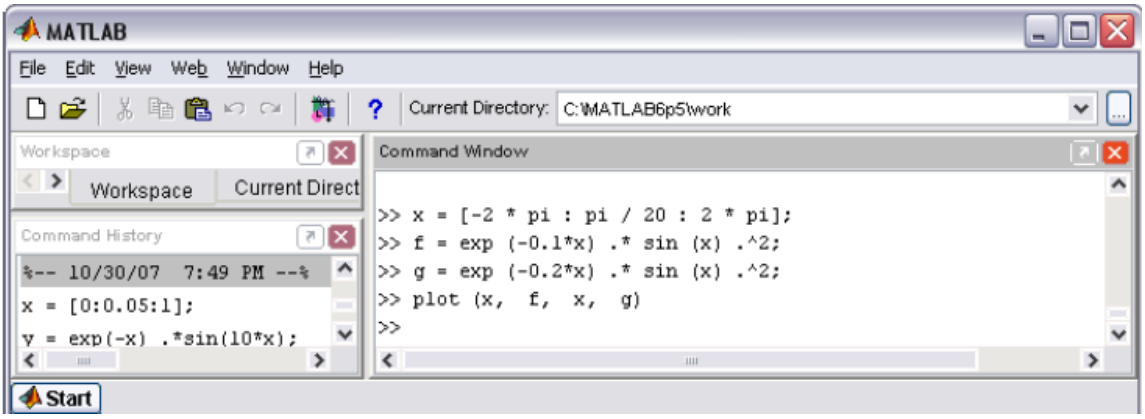


Рис. 8.16

Сравнить несколько функций можно, построив графики на одних координатных осях. Например, построим графики функций $f(x)=e^{-0.1x}\sin^2 x$ и $g(x)=e^{-0.2x}\sin^2 x$ на отрезке [–2π, 2π]. Сгенерируем вектор-строку значений аргумента *x* и вектор-строк *f* и *g*, содержащих значения функций. Команда **plot** с двумя парами аргументов приведет к построению графика (рис. 8.17).

а)



б)

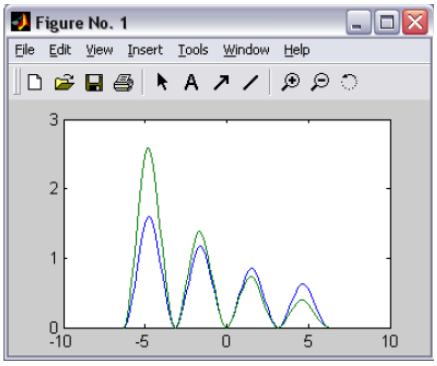


Рис. 8.17

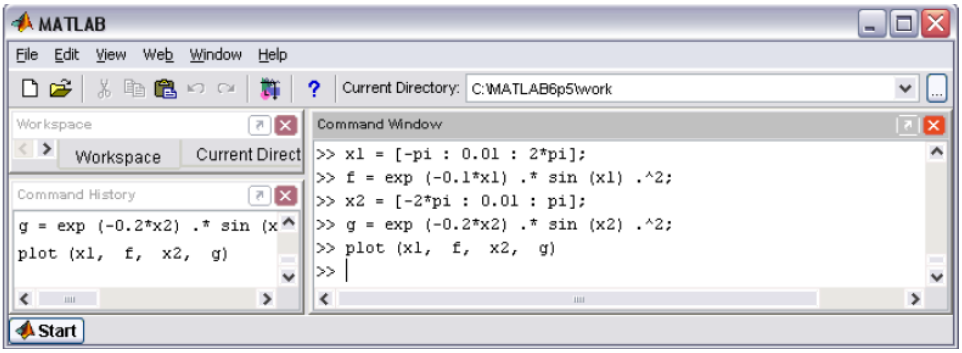
Функции необязательно должны быть определены на одном и том же отрезке. В этом случае при построении графиков MatLab выбирает максимальный отрезок, содержащий остальные. Важно только в каждой паре векторов абсцисс и ординат указать соответствующие друг другу векторы (рис. 8.18).

Аналогично при помощи задания в **plot** через запятую пар аргументов вида *вектор абсцисс*, *вектор ординат* осуществляется построение графиков произвольного числа функций.

Иногда требуется сравнить поведение двух функций, значения которых сильно отличаются друг от друга. Графики функции с небольшими значениями практически сливаются с осью абсцисс, и установить его вид не удается. В этой

ситуации помогает функция **plotyy**, которая выводит графики в окно с двумя вертикальными осями, имеющими подходящий масштаб (рис. 8.19).

а)



б)

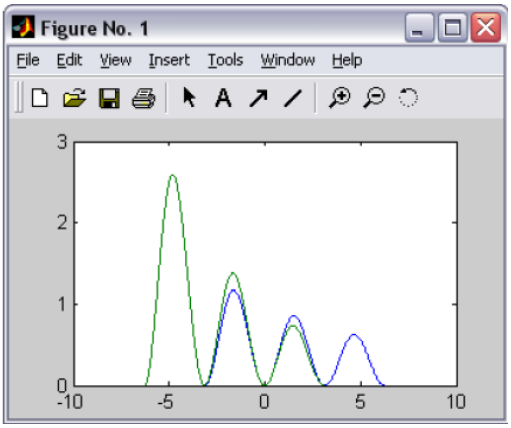
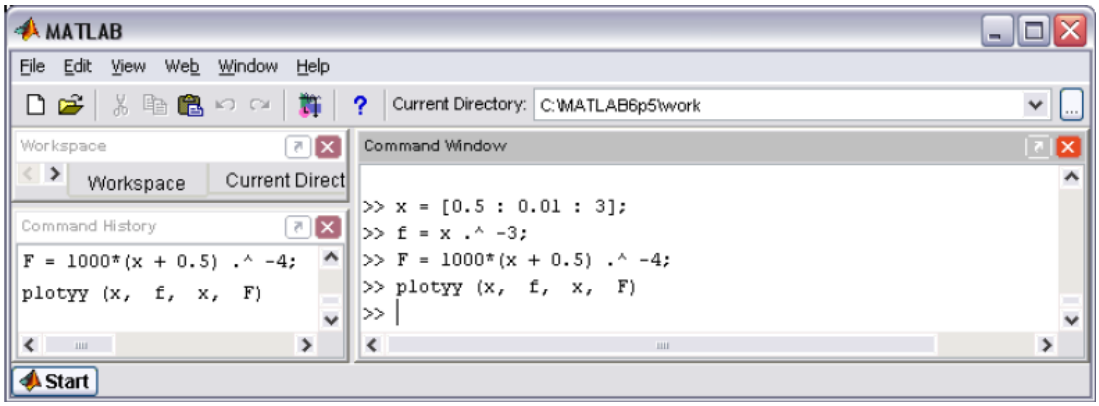


Рис. 8.18

Сравните, например, две функции: $f(x)=\frac{1}{x^3}$ и $F(x)=1000\cdot(x+0.5)^{-4}$

а)



б)

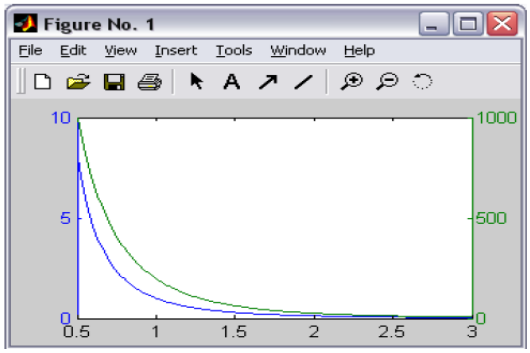


Рис. 8.19

Функция **plot** использует линейный масштаб по обеим координатным осям. Однако MatLab предоставляет пользователю возможность строить графики функций одной переменной в логарифмическом или полулогарифмическом масштабе (рис. 8.19).

Графики в логарифмических масштабах

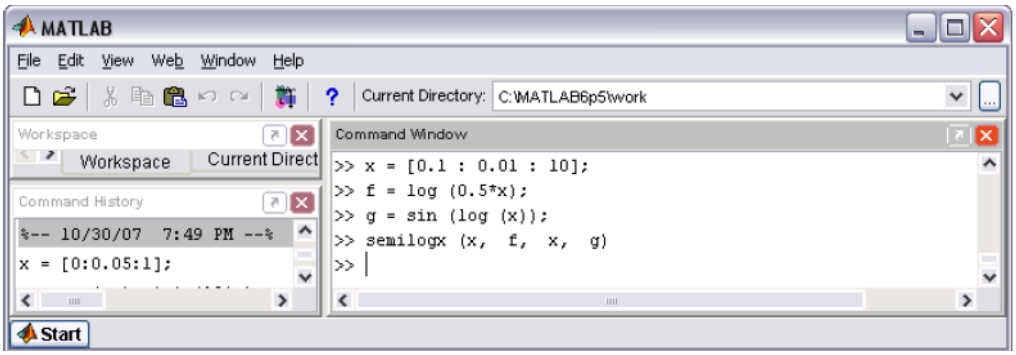
Для построения графиков в логарифмическом и полулогарифмическом масштабах служат следующие функции:

loglog (логарифмический масштаб по обеим осям);

semilogx (логарифмический масштаб только по оси абсцисс);

semilogy (логарифмический масштаб только по оси ординат).

а)



б)

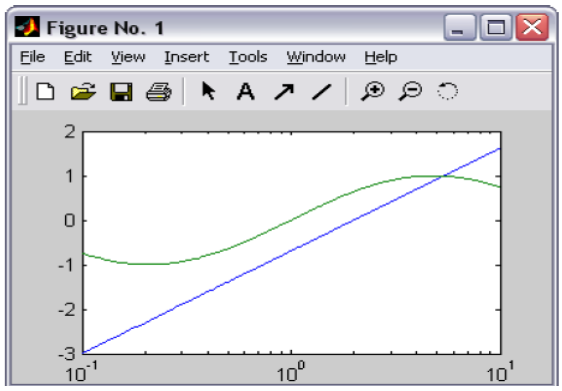


Рис. 8.20

Аргументы **loglog**, **semilogx** и **semilogy** задаются в виде пары векторов значений абсцисс и ординат так же, как для функции **plot**. Постройте, например, графики функций $f(x)=\log 0.5x$ и $g(x)=\sin\ln x$ на отрезке [0.1, 5] в логарифмическом масштабе по оси *x*(рис. 8.20).

Аналогично строятся графики при помощи функций **loglog** и **semilogy**.

Изменение свойств линий

Построенные графики функций должны быть максимально удобными для восприятия. Часто требуется нанести маркеры, изменить цвет линий, а при подготовке к монохромной печати – задать тип линии (сплошная, пунктирная, штрихпунктирная и т. д.). MatLab предоставляет возможность управлять видом графиков, построенных при помощи **plot**, **loglog**, **semilogx** и **semilogy**, для чего служит дополнительный аргумент, помещаемый за каждойпарой векторов. Этот аргумент заключается в апострофы и состоит из трех символов, которые определяют цвет и тип линии. Используется одна, две или три позиции, в зависимости от требуемых изменений. В табл. 8.1 приведены возможные значения данного аргумента с указанием результата.

Таблица 8.1

Цвет		Тип маркера		Тип линии	
1		2		3	
y	желтый	.	точка	–	сплошная
m	розовый	○	кружок	:	пунктирная
c	голубой	×	крестик	–.	штрих-пунктирная
r	красный	+	знак “плюс”	– –.	штриховая
g	зеленый	*	звездочка		
b	синий	s	квадрат		
w	белый	d	ромб		
k	черный	v	треугольник вершиной вниз		
		^	треугольник вершиной вверх		
		<	треугольник вершиной влево		
		>	треугольник вершиной вправо		
		p	пятиконечная звезда		
		h	шестиконечная звезда		

Оформление графиков

Удобство использования графиков во многом зависит от дополнительных элементов оформления: координатной сетки, подписей к осям, заголовка и легенды. Сетка наносится командой **gridon**, подписи к осям размещаются при помощи **xlabel**, **ylabel**, заголовок дается командой **title**. Наличие нескольких графиков на одних осях требует помещения легенды командой **legend** с

информацией о линиях (рис. 8.21). Все перечисленные команды применимы к графикам, как в линейном, так и в логарифмическом и полулогарифмическом масштабах. Следующие команды выводят графики изменения суточной температуры, которые снабжены всей необходимой информацией.

При добавлении легенды следует учесть, что порядок и количество аргументов команды **legend** должны соответствовать линиям на графике. Последним дополнительным аргументом **legend** может быть положение легенды в графическом окне:

–1 – вне графика в правом верхнем углу графического окна;

0 – выбирается лучшее положение в пределах графика так, чтобы как можно меньше перекрывать сами графики;

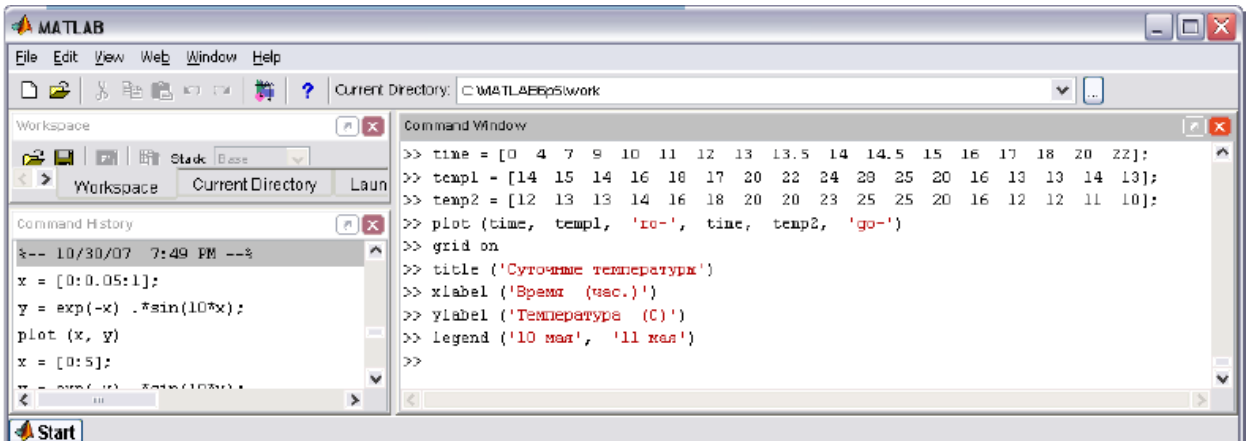
1 – в верхнем правом углу графика (это положение используется поумолчанию);

2 – в верхнем левом углу графика;

3 – в нижнем левом углу графика;

4 – в нижнем правом углу графика.

а)



б)

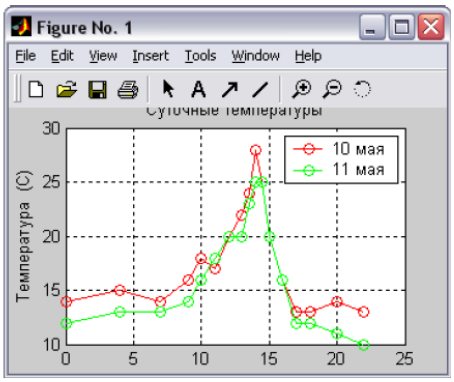


Рис. 8.21

В заголовке графика, легенде и подписях осей допускается добавление формул и изменение стилей шрифта при помощи формата TeX.

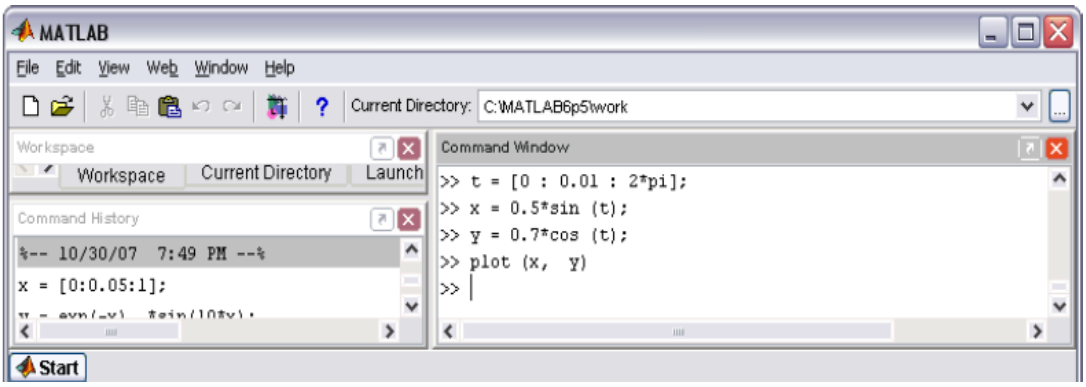
Графики параметрических и кусочно-заданных функций

Для построения функций, заданных параметрически, следует сперва сгенерировать вектор значений аргумента. Затем необходимо вычислить значения функций и записать их в векторы, которые и надо использовать в качестве аргументов **plot**. График функции $x(t) = 0.5 \cdot \sin t$, $y(t) = 0.7 \cdot \cos t$ для $t \in [0, 2\pi]$ (эллипс) получается при помощи команд (рис. 8.22).

Для того чтобы проверить свои знания, постройте график функции, заданной кусочным образом:

$$y(x) = \begin{cases} \pi \cdot \sin x, & -2\pi \le x \le -\pi \\ \pi - |x|, & -\pi < x < \pi \\ \pi \cdot \sin^3 x & \pi \le x \le 2\pi \end{cases}$$

а)



б)

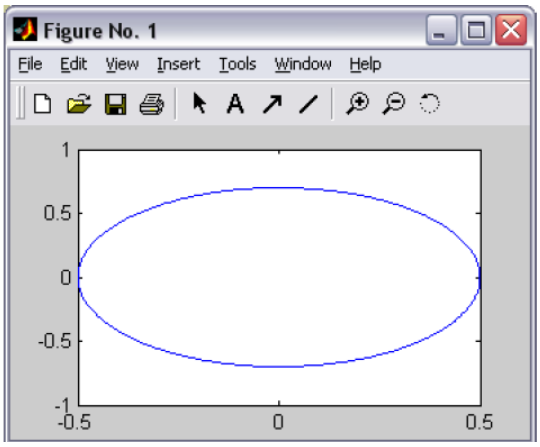


Рис. 8.22

Сначала необходимо вычислить каждую из трех ветвей, т.е. фактически получить три пары массивов $x1$ и $y1$, $x2$ и $y2$, $x3$ и $y3$, затем объединить

значения абсцисс в вектор *x*, а значения ординат в *y* и построить график функции, задаваемой парой массивов *x* и *y*.

8.5. Графики функций двух переменных

MatLab предоставляет различные способы визуализации функций двух переменных – построение трехмерных графиков и линий уровня, параметрически заданных линий и поверхностей.

Для отображения функции двух переменных следует:

1. Сгенерировать матрицы с координатами узлов сетки на прямоугольной области определения функции.

2. Вычислить функцию в узлах сетки и записать полученные значения в матрицу.

3. Использовать одну из графических функций MatLab.

4. Нанести на график дополнительную информацию, в частности, соответствие цветов значениям функции.

Сетка генерируется при помощи команды **meshgrig**, вызываемой с двумя аргументами. Аргументами являются векторы, элементы которых соответствуют сетке на прямоугольной области построения функции – квадрат. Для вычисления функции следует использовать поэлементные операции.

Изучим основные возможности, предоставляемые MatLab для визуализации функций двух переменных, на примере построения графика

$$z(x, y) = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot \left(1 - x^2\right) \cdot y \cdot (1 - y)$$

на прямоугольной области определения $x \in [-1, 1]$, $y \in [0, 1]$. Подготовьте матрицы с координатами узлов сетки и значениями функции, как показано на рис. 8.23.

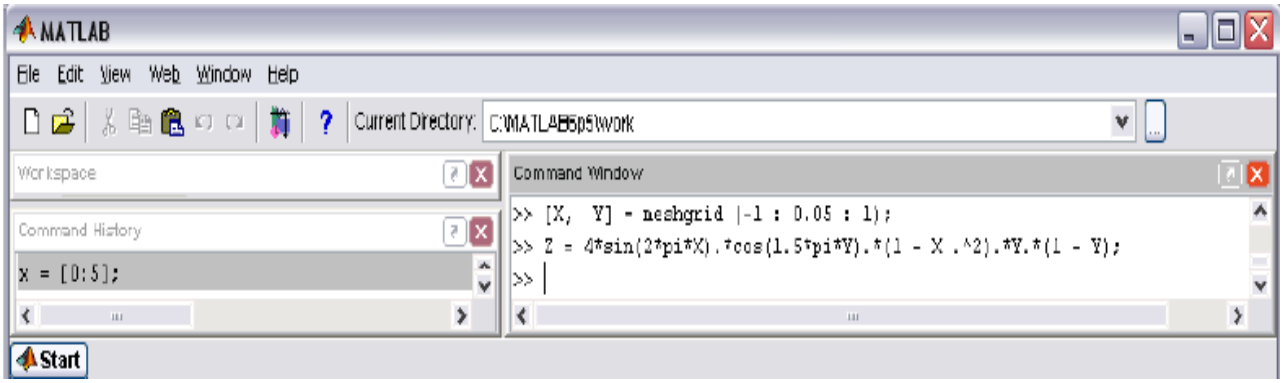


Рис. 8.23

Для построения каркасной поверхности используется функция **mesh**, вызываемая с тремя аргументами:

```
>> mesh (X, Y, Z)
```

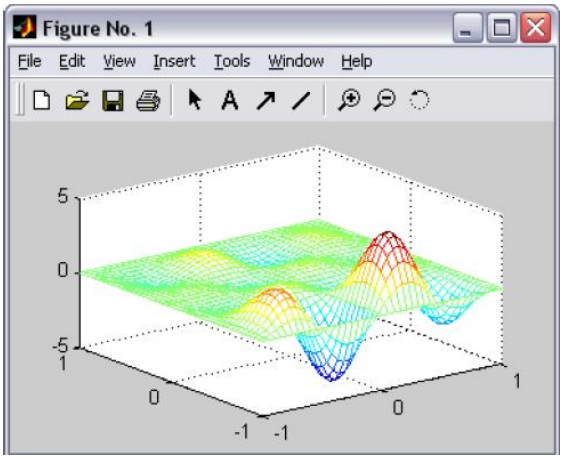


Рис. 8.24

Цвет линий поверхности соответствует значениям функции рис. 8.24. MatLab рисует только видимую часть поверхности. При помощи команды **hiddenoff** можно сделать каркасную поверхность “прозрачной”, добавив скрытую часть. Команда **hiddenon** убирает невидимую часть поверхности, возвращая графику прежний вид.

Функция **surf** строит каркасную поверхность графика функции и заливает каждую клетку поверхности определенным цветом, зависящим от значения функции в точках, соответствующих углам клетки (рис. 8.25):

```
>> surf (X, Y, Z)
```

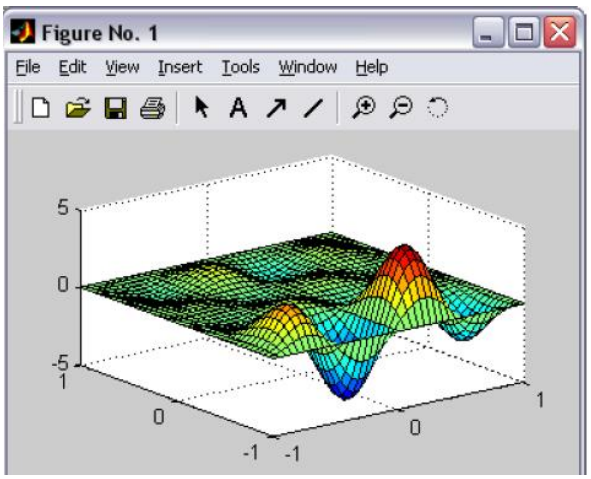


Рис. 8.25

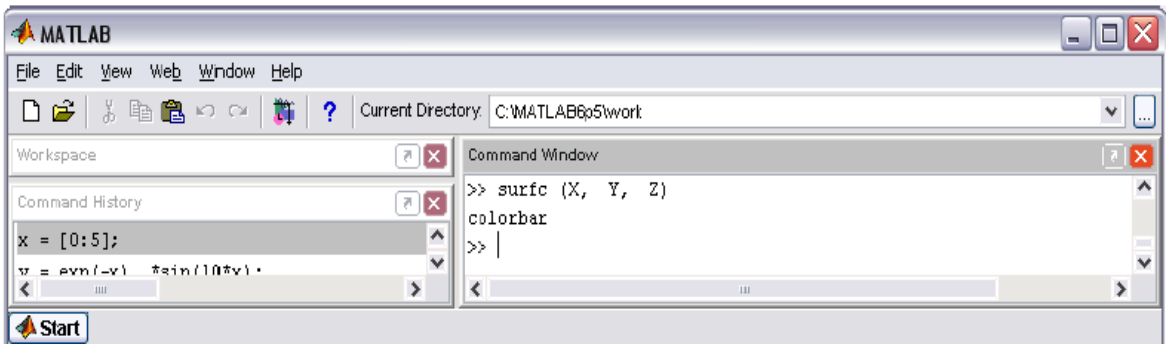
В пределах каждой клетки цвет постоянный. Команда **shadingflat** позволяет убрать каркасные линии. Для получения поверхности, плавно залитой цветом, предназначена команда **shadinginterp**.

При помощи **shadingfaceted** можно вернуться к виду с каркасными линиями.

В MatLab определена команда **colorbar**, которая выводит рядом с графиком столбик, устанавливающий соответствие между цветом и значением функции. Постройте при помощи **surf** график поверхности и дополните его информацией о цвете.

Команды **meshc** или **surfc** позволяют получить более точное представление о поведении функции. Эти команды строят каркасную поверхность или залитую цветом каркасную поверхность и размещают на плоскости *xy* линии уровня функции (линии постоянства значений функции) – см. рис. 8.26.

а)



б)

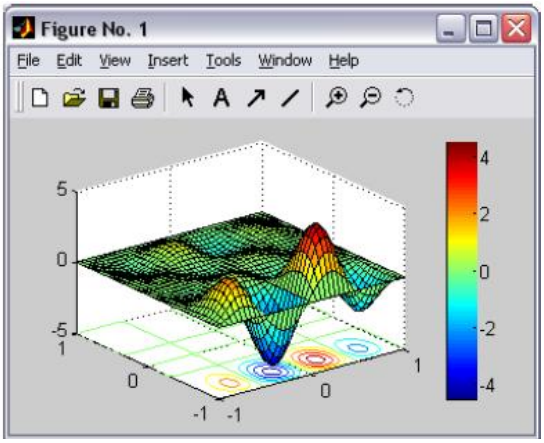
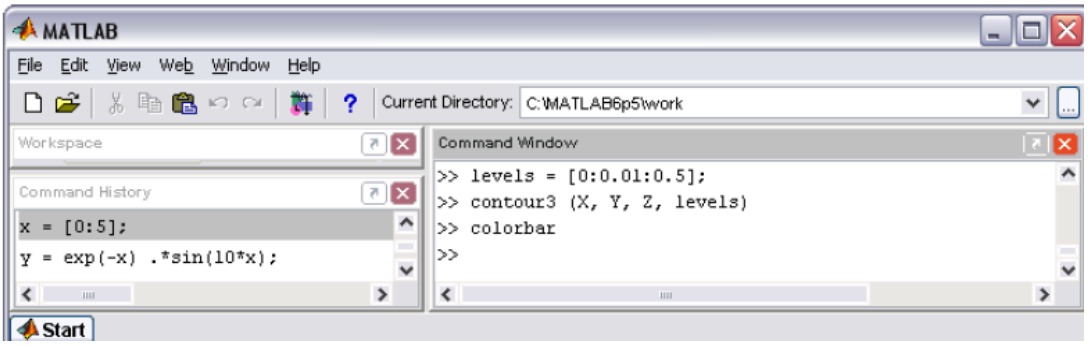


Рис. 8.26

а)



б)

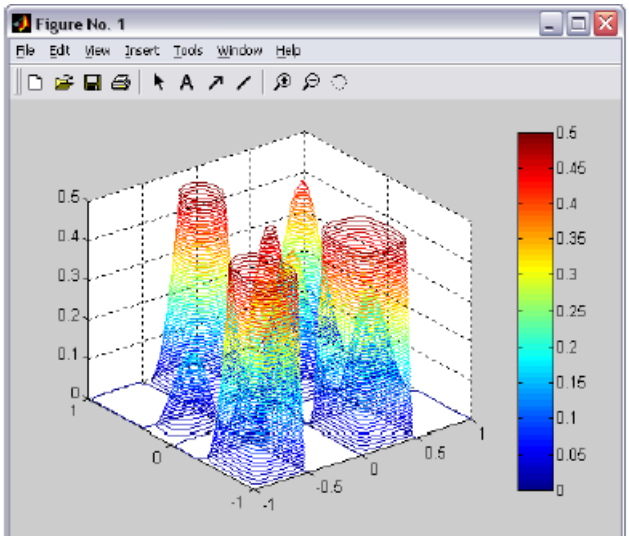


Рис. 8.27

MatLab позволяет построить поверхность, состоящую из линий уровня, при помощи функции **contour3**. Эту функцию можно использовать так же, как и описанные выше **mesh**, **surf**, **meshc** и **surfc** с тремя аргументами. При этом число линий уровня выбирается автоматически. Имеется возможность задать четвертым аргументом в **contour3** либо число линий уровня, либо вектор, элементы которого равны значениям функции, отображаемым в виде линий уровня. Задание вектора удобно, когда требуется исследовать поведение

функции в некоторой области ее значений (срез функции). Постройте, например поверхность, состоящую из линий уровня, соответствующих значениям функции от 0 до 0.5 с шагом 0.01 (рис. 8.27).

9. Графики функций

MatLab предоставляет обширные возможности для визуализации функций одной и двух переменных. Использование функций для построения графиков с минимальным набором задаваемых параметров (остальные MatLab выбирает автоматически) приводит к получению качественных графиков.

9.1. Графики функций двух переменных

MatLab предоставляет различные способы визуализации функций двух переменных – построение трехмерных графиков и линий уровня, параметрически заданных линий и поверхностей.

Построение параметрически заданных поверхностей и линий

MatLab позволяет строить трехмерные линии, определенные формулами

$$x = x(t),\ y = y(t),\ z = z(t),\ t \in [a,b],$$

и поверхности, задаваемые зависимостями

$$x = x(u,v),\ y = y(u,v),\ z = z(u,v),\ u \in [c,d],\ v \in [c,d].$$

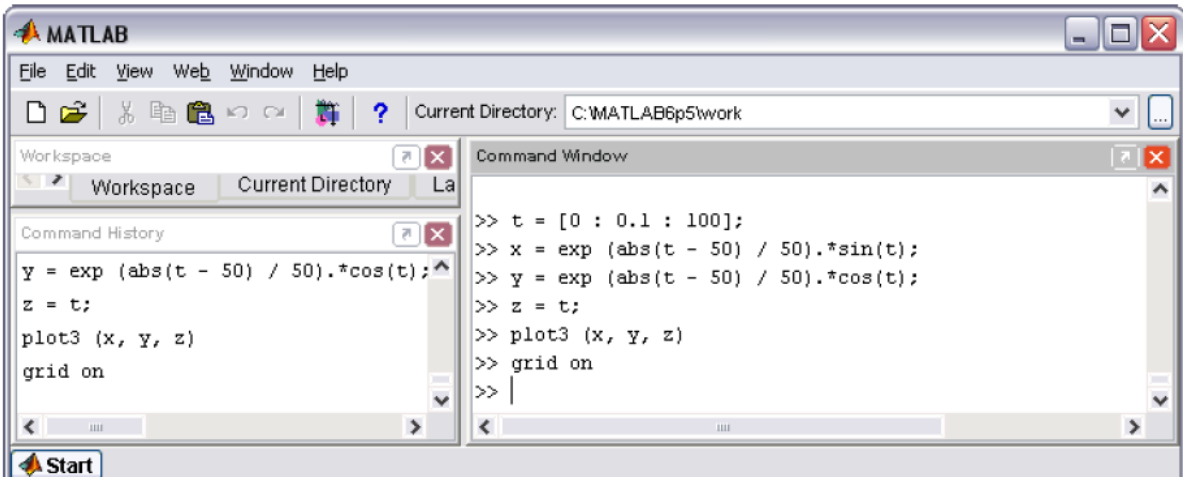
Функция **plot3** визуализирует параметрически заданные линии, используя в качестве аргументов векторы, содержащие значения функций *x(t)*, *y(t)* и *z(t)*, вычисленные для значений параметра *t*. Сначала следует сформировать вектор *t*, используя заполнение с постоянным шагом при помощи двоеточия, а затем вычислить и записать в векторы соответствующие значения функции.

Например, для графика линии

$$x = e^{-|t-50|/50}\sin t,\ \ y = e^{-|t-50|/50}\cos t,\ z = t,\ t \in [0,100]$$

использованы команды, показанные на рис. 9.1.

а)



б)

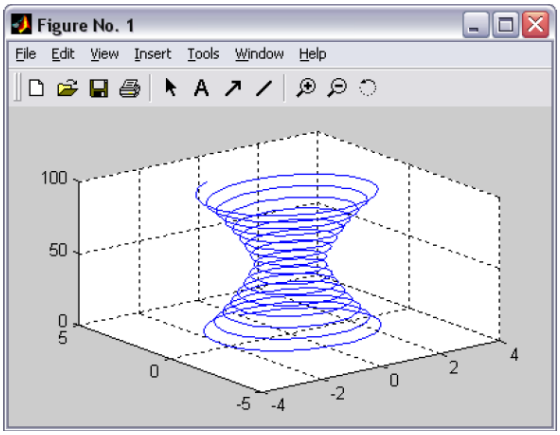


Рис. 9.1

Параметрически заданную поверхность можно построить при помощи любой из функций, предназначенных для отображения трехмерных графиков. Важно только правильно подготовить аргументы. Дело в том, что функции *x(u, v)* и *y(u, v)* могут быть многозначны, что надо учесть при создании матриц с информацией о расположении узлов сетки на области построения и матрицы, содержащей значения функций *z(u, v)* в этих точках. Например, отобразите поверхность (конус), определенную зависимостями

$$x(u,v)=0.3\cdot u\cdot\cos v,\ y(u,v)=0.3\cdot u\cdot\sin v,\ z(u,v)=0.6\cdot u,\ u,v\in[-2\pi,2\pi].$$

Сгенерируйте при помощи двоеточия вектор-столбец и вектор-строку, содержащие значения параметров на заданном интервале (*u* – вектор-столбец, *v* – вектор-строка) – рис. 9.2.

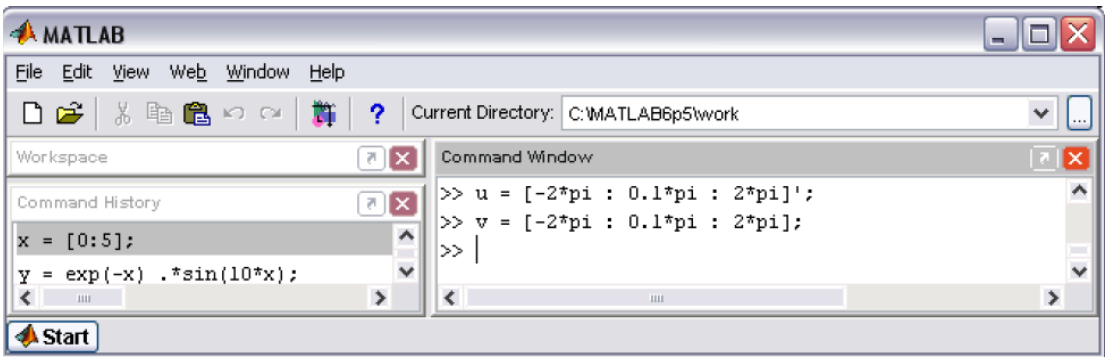


Рис. 9.2

Далее сформируйте матрицы X, Y, содержащие значения функций *x(u, v)*, *y(u, v)* в точках, соответствующих значениям параметров при помощи *внешнего произведения векторов* (звездочка без точки) – рис. 9.3.

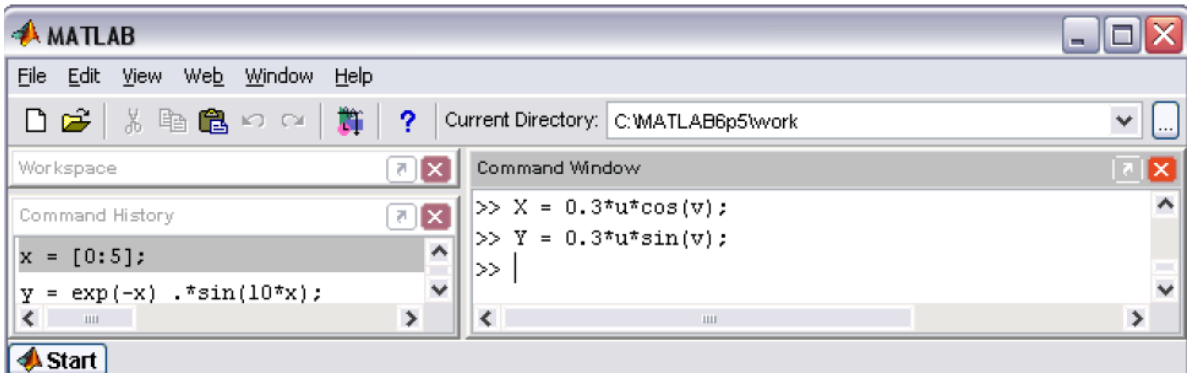


Рис. 9.3

Матрица Z должна быть того же размера, что X и Y и, кроме того, она должна содержать значения, соответствующие значениям параметров. Если бы в функцию *z(u,v)* входило произведение *u* и *v*, то матрицу Z можно было заполнить аналогично X и Y при помощи внешнего произведения. С другой стороны, функцию *z(u, v)* можно представить в виде $z(u,v)=0.6\cdot u\cdot g(v)$, где *g(v)* ≡ 1. Поэтому для вычисления Z снова примените внешнее произведение на вектор-строку той же размерности, что V, состоящую из единиц (рис. 9.4).

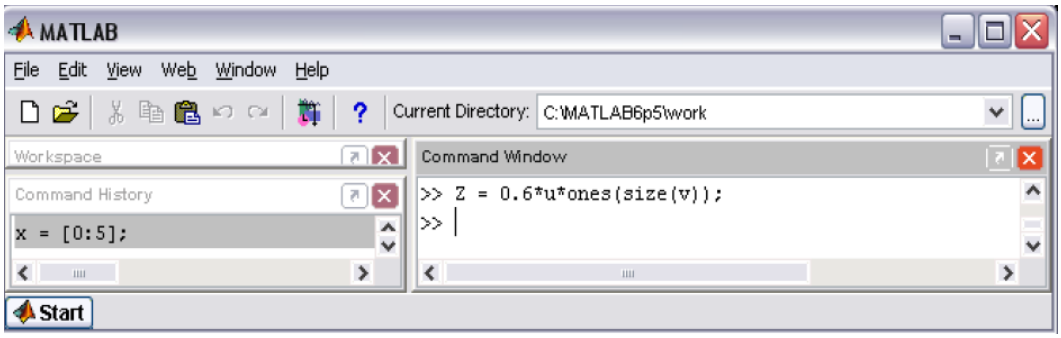
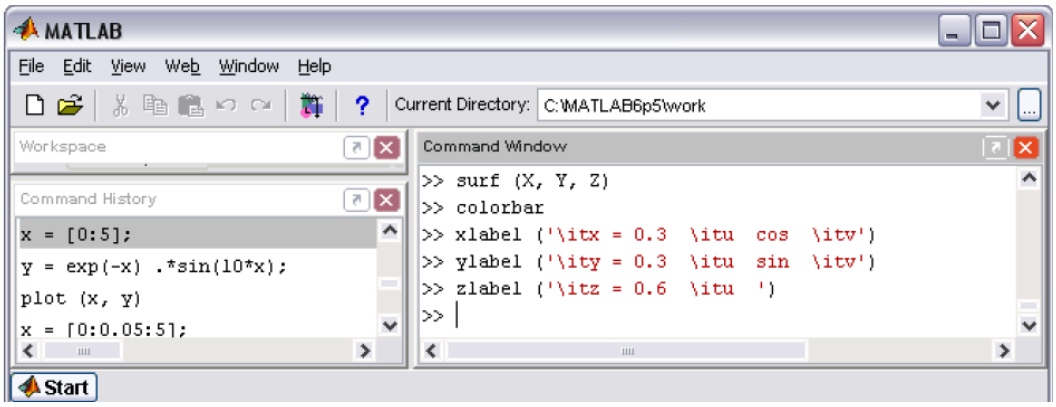


Рис. 9.4

Все требуемые матрицы созданы. Используйте теперь любую из описанных выше функций для построения трехмерных графиков (рис. 9.5).

а)



б)

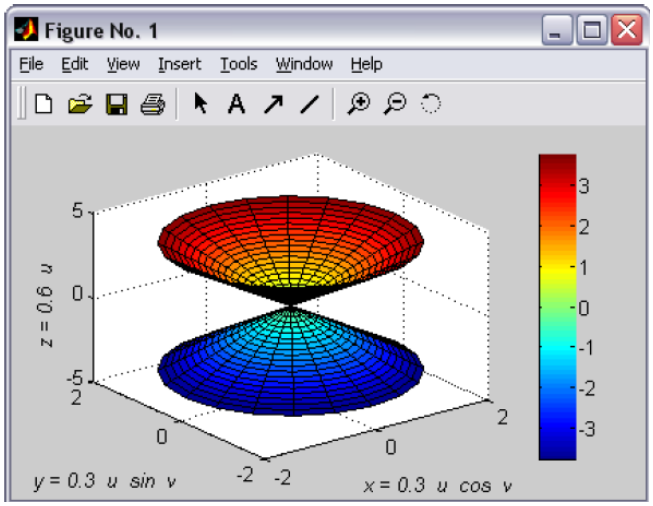


Рис. 9.5

Постройте самостоятельно прозрачную каркасную поверхность эллипсоида, заданного соотношениями

$$x(u,v)=\cos u\cdot\cos v,\quad y(u,v)=0.7\cdot\cos u\cdot\sin v,\quad z(u,v)=0.8\cdot\sin u,\quad u,v\in[-\pi,\pi].$$

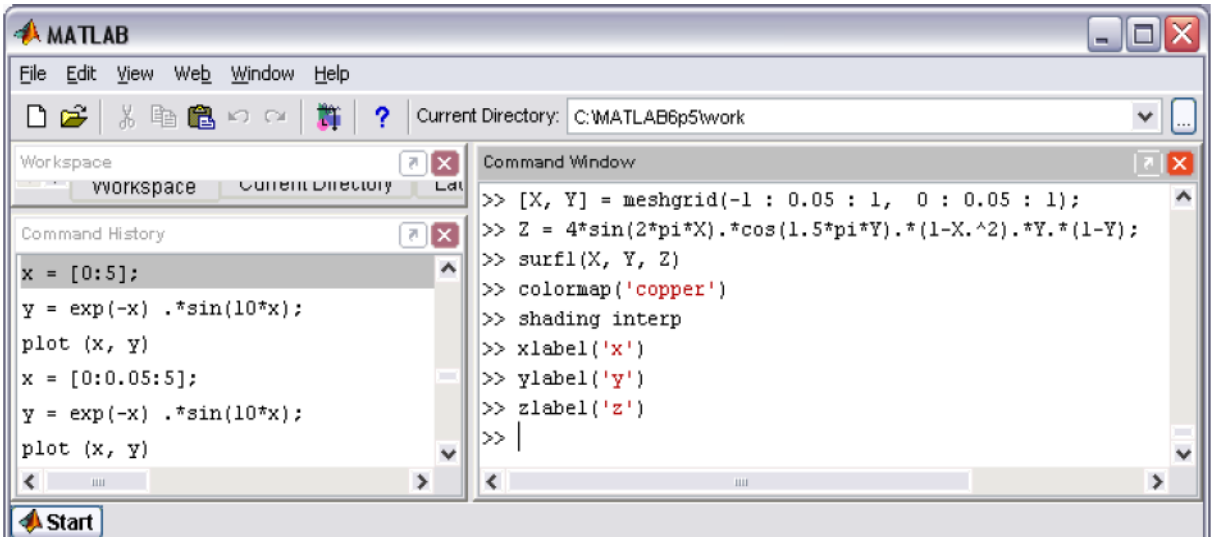
Построение освещенной поверхности

Предположим, что поверхность графика функции сделана из материала с определенными свойствами отражения и поглощения света и, кроме того, можно управлять расположением источника света. Эти две возможности вместе с поворотом графика позволяют получить естественно выглядящую поверхность, повернутую и освещенную под нужным углом. Для построения освещенной поверхности применяется функция **surfl**.

Постройте освещенную поверхность, задаваемую на прямоугольной области $x\in[-1\ 1]$, $y\in[0\ 1]$ формулой

$$z(x,y)=4\cdot\sin 2\pi x\cdot\cos 1.5\pi y\cdot\left(1-x^2\right)\cdot y\cdot(1-y).$$

а)



б)

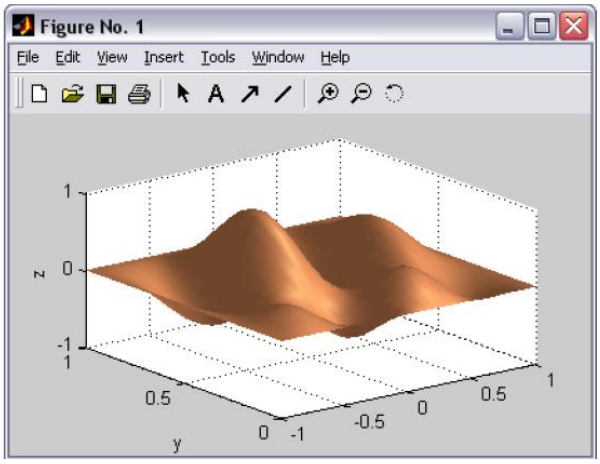
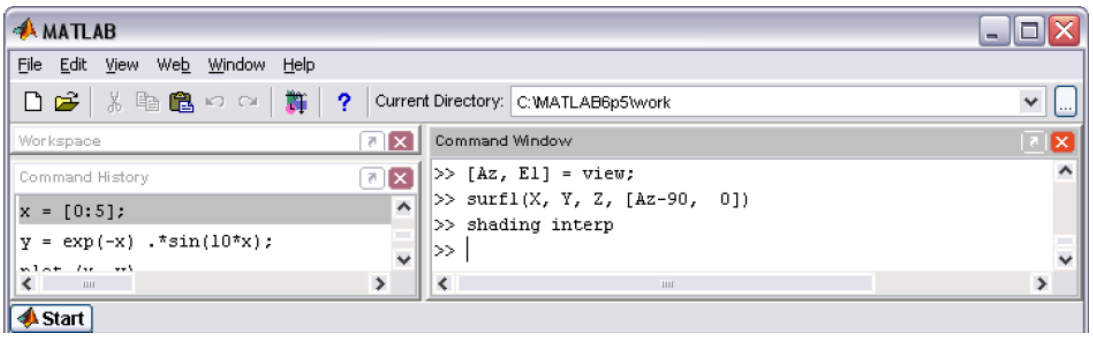


Рис. 9.6

При использовании **surfl** удобно задавать цветовые палитры **copper**, **bone**, **gray**, **pink**, в которых интенсивность цвета изменяется линейно. Для получения плавно изменяющихся оттенков следует использовать **shadinginterp**. Команды (рис. 9.6), приводят к получению требуемой освещенной поверхности.

По умолчанию источник света имеет азимут, больший на 45°, чем наблюдатель, и тот же угол возвышения. Дополнительным четвертым аргументом **surfl** может быть вектор-строка из двух элементов – азимута и угла возвышения источника света. Например, азимут источника изменить на –90° по отношению к наблюдателю, а угол возвышения установите в ноль (рис. 9.7).

а)



б)

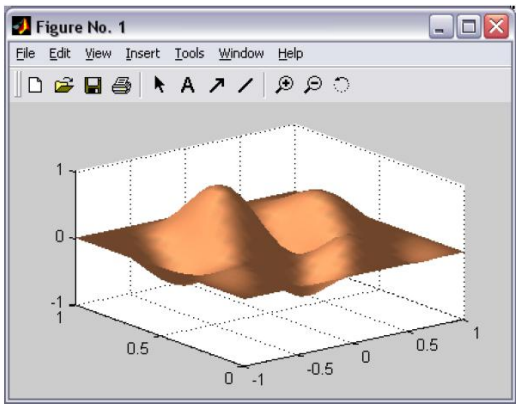


Рис. 9.7

9.2. Анимированные графики

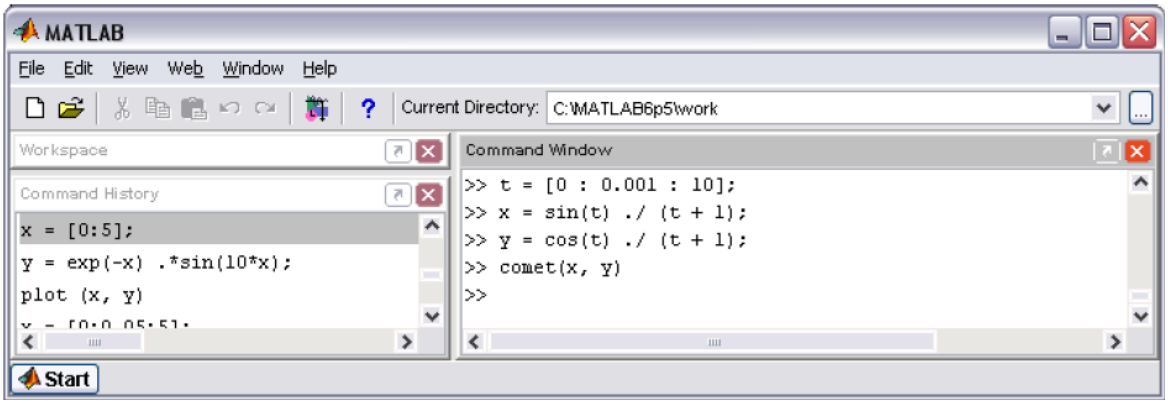
При изучении движения точки на плоскости или в трехмерном пространстве полезно не только построить траекторию точки, но и следить за движением точки по траектории. MatLab предоставляет возможность получить анимированный график, на котором кружок, обозначающий точку, перемещается на плоскости или в пространстве, оставляя за собой след в виде линии – траектории движения. График похож на летящую комету с хвостом.

Для построения анимированных графиков применяются функции **comet** и **comet3**. Постройте, например, траекторию движения точки в течение 10 секунд, координаты которой изменяются по закону

$$x(t)=\frac{\sin t}{t+1}\ y(t)=\frac{\cos t}{t+1}.$$

Действуйте точно так же, как при построении графика параметрически заданной функции, но для визуализации результата используйте **comet** (рис. 9.8).

а)



б)

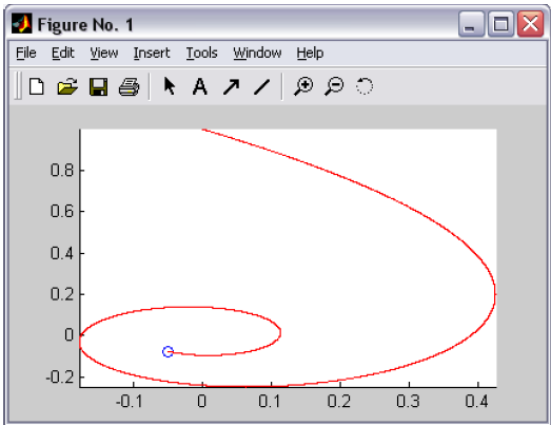


Рис. 9.8

При выполнении последней команды следите за тем, чтобы окно с графиком было поверх остальных окон.

Скоростью движения кружка можно управлять, задавая различные шаги при автоматическом заполнении вектора, соответствующего времени. Использование **comet** с одним аргументом (вектором) приводит к построению динамически рисующегося графика значений элементов номера в зависимости от их номеров. Функцию **comet** можно вызвать и с третьим дополнительным числовым параметром, который задает длину хвоста кометы. По умолчанию он равен 0.1. Обратите внимание, что при изменении размеров графического окна или при его минимизации и последующем восстановлении траектория движения пропадает. Это связано со способом, который применяет MatLab для построения графика.

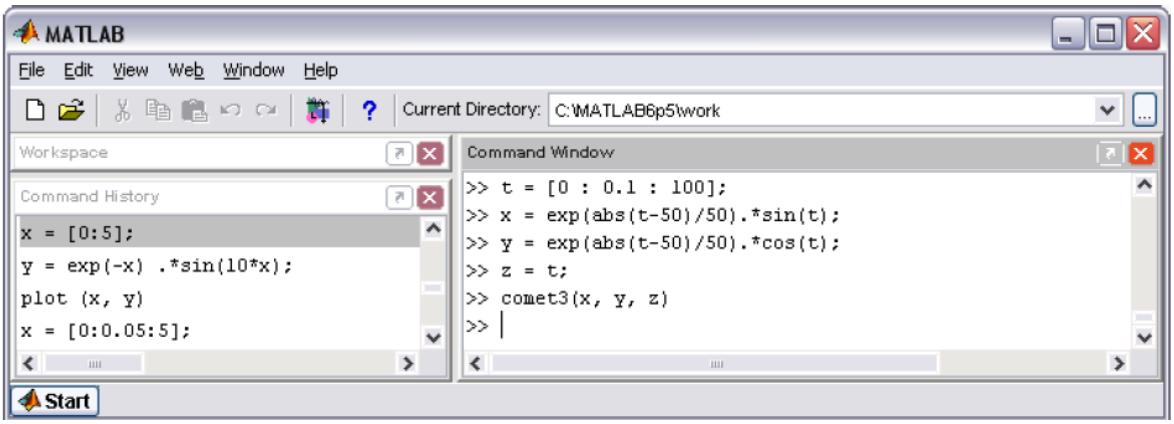
Получите самостоятельно траекторию движения фиксированной точки на окружности, катящейся по прямой (циклоиду). Циклоида описывается параметрическими зависимостями $x(t) = t - \sin t,\ y(t) = 1 - \cos t.$

Для построения траектории точки, перемещающейся в пространстве, используется функция **comet3**. Пусть координаты точки в течение 100 секунд изменялись по следующему закону:

$$x = e^{-|t-50|/50}\sin t,\ y = e^{-|t-50|/50}\cos t,\ z = t.$$

Отобразите траекторию движения точки, применяя команды, показанные на рис. 9.9.

а)



б)

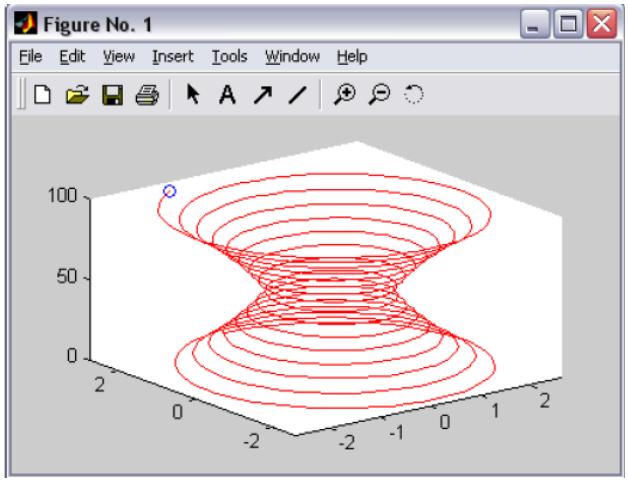


Рис. 9.9

Функцию **comet3** можно вызывать с четвертым числовым аргументом, который так же, как и в случае **comet**, задает длину хвоста кометы.

9.3. Работа с несколькими графиками

Во всех примерах графики выводились в специальное графическое окно с заголовком **Figure No.1**. При следующем построении графика предыдущий пропадал, а новый выводился в то же самое окно. MatLab предоставляет следующие возможности работы с несколькими графиками:

• вывод каждого графика в свое окно;

• вывод нескольких графиков в одно окно (на одни координатные оси);

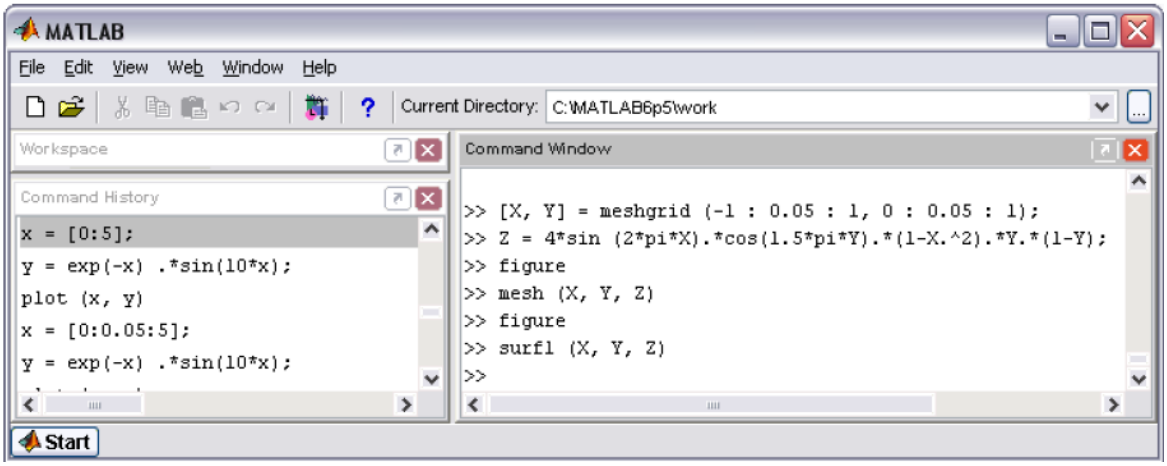
• отображение в пределах одного окна нескольких графиков, каждого на своих осях.

Вывод графиков в отдельные окна

Команда **figure**, определенная в MatLab, служит для создания пустого графического окна и отображения его на экране. Окно становится текущим, т.е. все последующие графические функции будут осуществлять построение графиков в этом окне. Для построения нового графического окна следует снова использовать **figure**. На рис. 9.10 показано построение поверхности и вывод двух графических окон с использованием команды **figure.**

Последовательность команд (рис. 9.10), приводит к появлению на экране двух графических окон: **Figure No.1**, содержащего каркасную поверхность, и **Figure No.2**с освещенной поверхностью. Окно **Figure No.2** является текущим, так как было создано последним. Команды, набираемые далее(рис. 9.11), приведут к изменениям именно в этом окне.

а)



б)

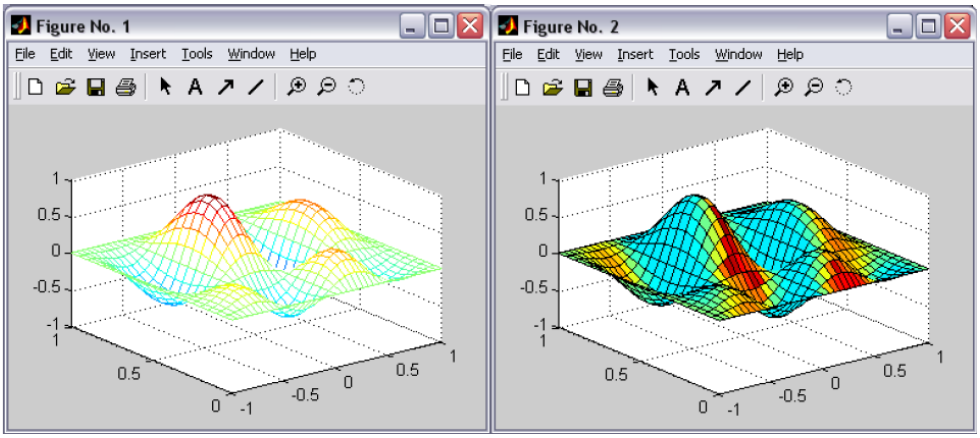
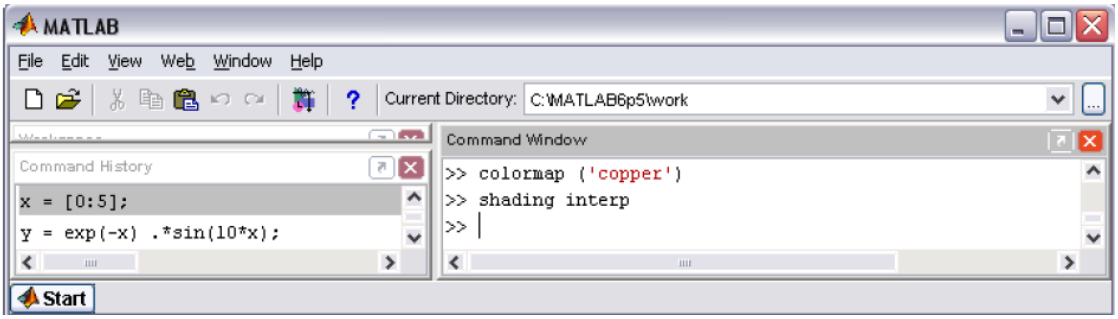


Рис. 9.10

а)



б)

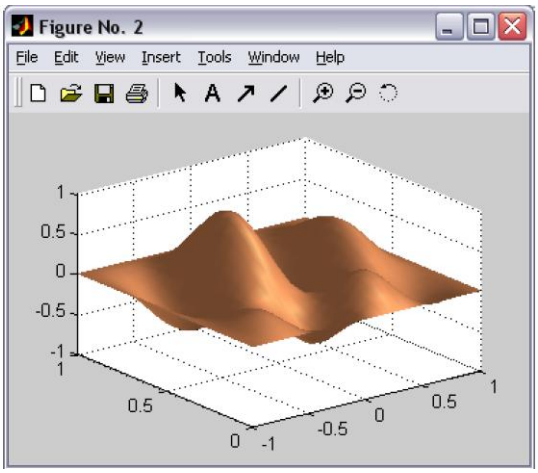


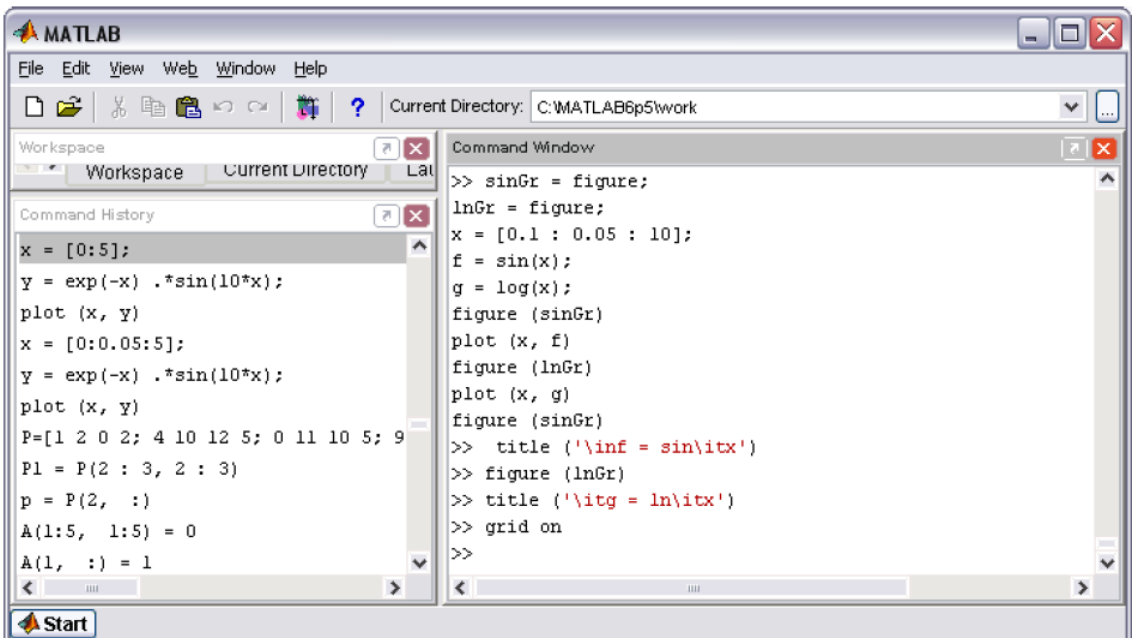
Рис. 9.11

Для того чтобы сделать графическое окно **Figure No.1** текущим, следует щелкнуть на нем мышкой, вернуться в рабочую среду MatLab и продолжать ввод команд. Команды повлекут изменения в окне **Figure No.1**. Для очистки всего текущего окна используется команда **clf** (сокращение от **clearfigure**), а для того, чтобы убрать только график, но оставить оси, заголовок иназвание осей, следует применить **cla** (сокращение от **clearaxes**).

Вышеописанным способом можно получить сколько угодно графических окон и вывести в них графики различных функций или визуализировать векторные и матричные данные. Однако для изменения того или иного графика придется искать его окно на экране и делать его текущим при помощи щелчка мыши. Есть более универсальный и удобный способ работы с несколькими окнами. При создании каждого нового графического окна при помощи **figure** следует вызвать ее с выходным аргументом. Этот аргумент называется в MatLab *указателем* на графическое окно. Значением выходного аргумента является число, совпадающее с номером графического окна. Для того чтобы сделать графическое окно текущим, следует вызвать **figure**, применив в качестве входного аргумента указатель на требуемое графическое окно. Создайте два графических окна, постройте в них графики функций $f = \sin x$ и $g = \ln x$, а затем дайте заголовки и нанесите сетку на второй график (рис. 9.12).

Для того чтобы очистить графическое окно с указателем **lnGr**, следует использовать команду **clf(lnGr)**. Удаление графика из первого окна, на которое указывает **sinGr**, производится при помощи команды **cla(sinGr)**.

а)



б)

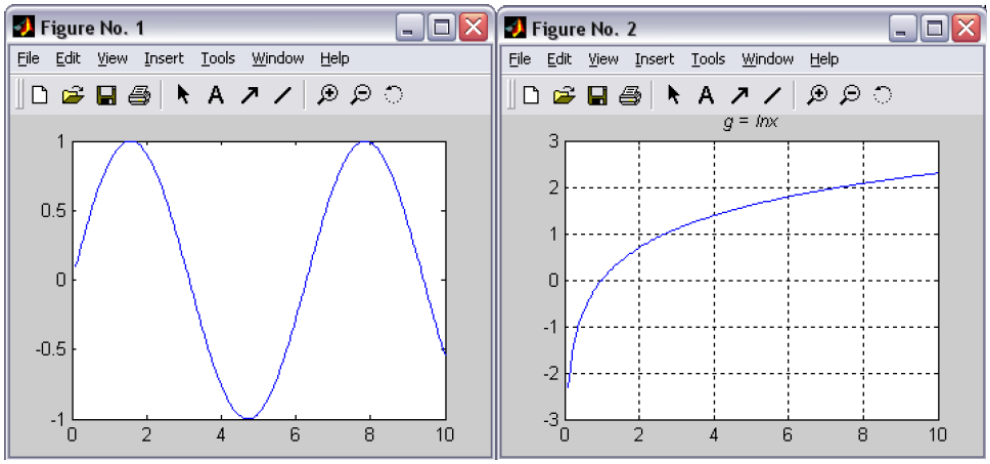
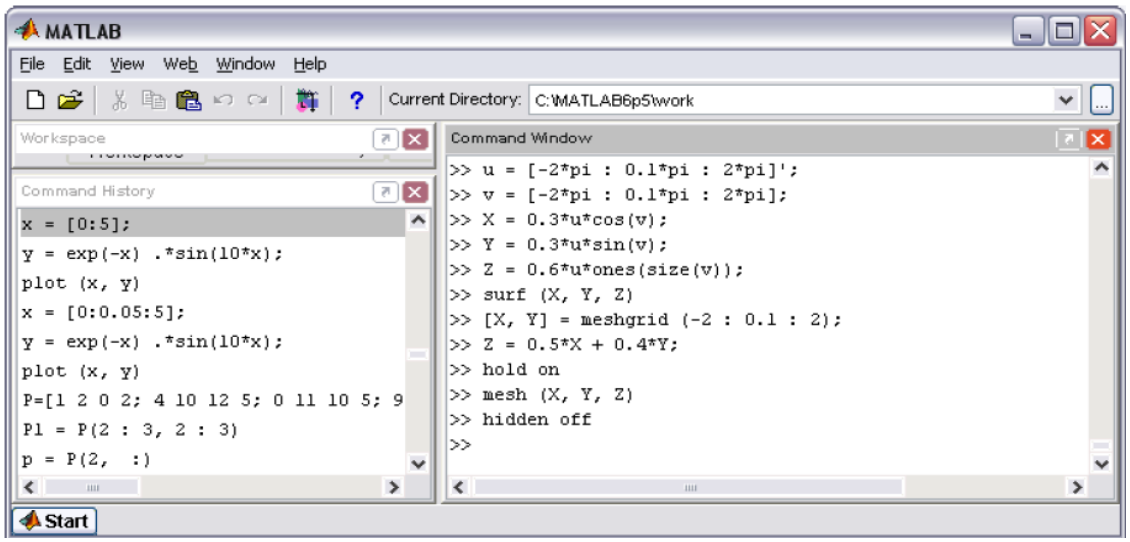


Рис. 9.12

Вывод нескольких графиков на одни оси

Возможность отображения нескольких графиков функций одной переменной на одних осях использовалась при изучении функций **plot**, **plotyy**, **semilogx**, **semilogy**, **loglog**. Они позволяют выводить графики нескольких функций, задавая соответствующие векторные аргументы парами, например **plot**(*x*, *f*, *x*, *g*). Однако, при построении трехмерных графиков или различных типов графиков, объединять их на одних осях не было возможности. Для объединения графиков предназначена команда **holdon**, которую нужно задать перед построением следующего графика. На рис. 9.13показано, как выводится пересечение плоскости и конуса, заданного параметрически.

а)



б)

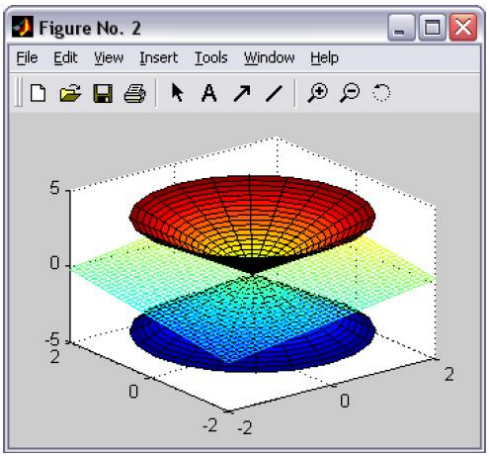


Рис. 9.13

Команда **hiddenoff** применена для того, чтобы показать часть конуса, находящуюся под плоскостью.

Обратите внимание, что **holdon** распространяется на все последующие выводы графиков в текущее окно. Для размещения графиков в новых окнах следует выполнить команду **holdoff**. Команда **holdon** может применяться и для расположения нескольких графиков функций одной переменной, например,

```
>> plot (x, f, x, g)
```

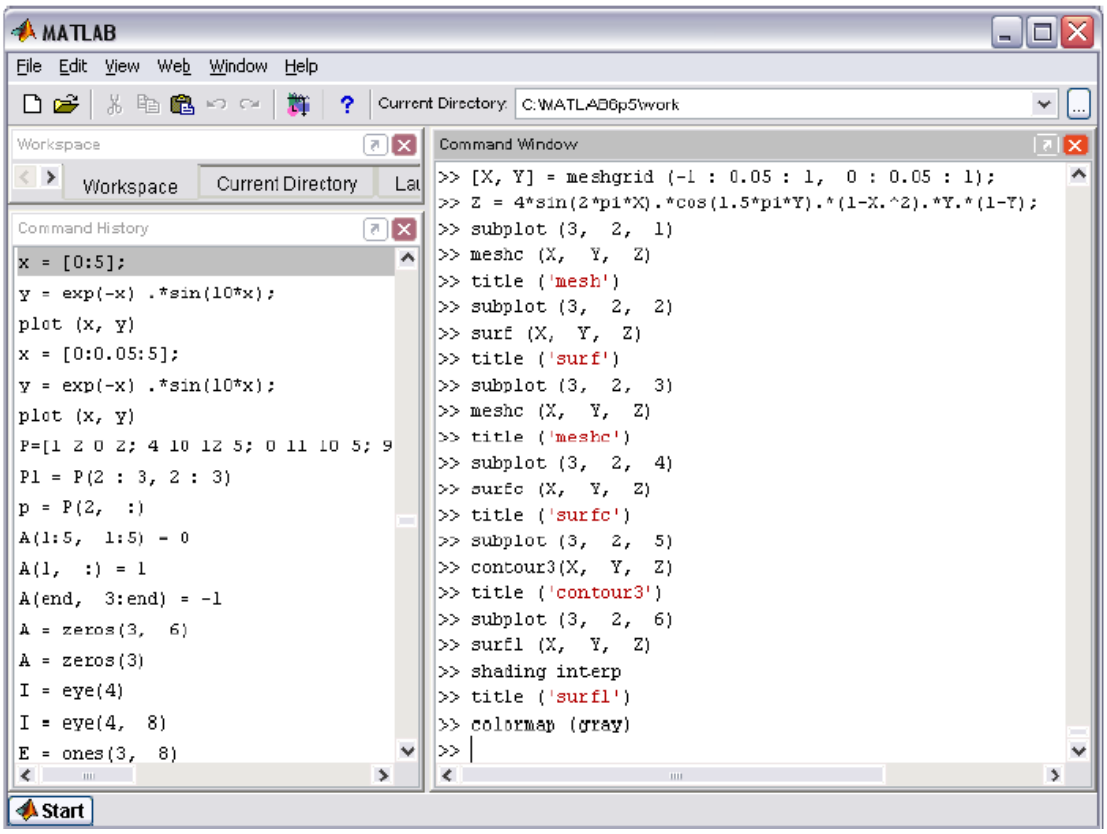
эквивалентно последовательности

```
>> plot (x, f)
>> hold on
>> plot (x, g)
```

Построение нескольких графиков в одном графическом окне

MatLab позволяет разбить графическое окно на несколько подграфиков со своими осями. Для этого служит команда **subplot**, которая располагает подграфики в виде матрицы и используется с тремя параметрами: **subplot** (*i*, *j*, *n*). Здесь *i* и *j* – число подграфиков по вертикали и горизонтали, а n – номер подграфика, который надо сделать текущим. Номер отсчитывается от левого верхнего угла построчно. Например, команда **subplot** (3, 2, 4) предполает наличие шести подграфиков и делает четвертый текущим (рис. 9.14). После выполнения **subplot** (3, 2, 4) все графические функции будут осуществлять вывод именно в этот подграфик.

а)



б)

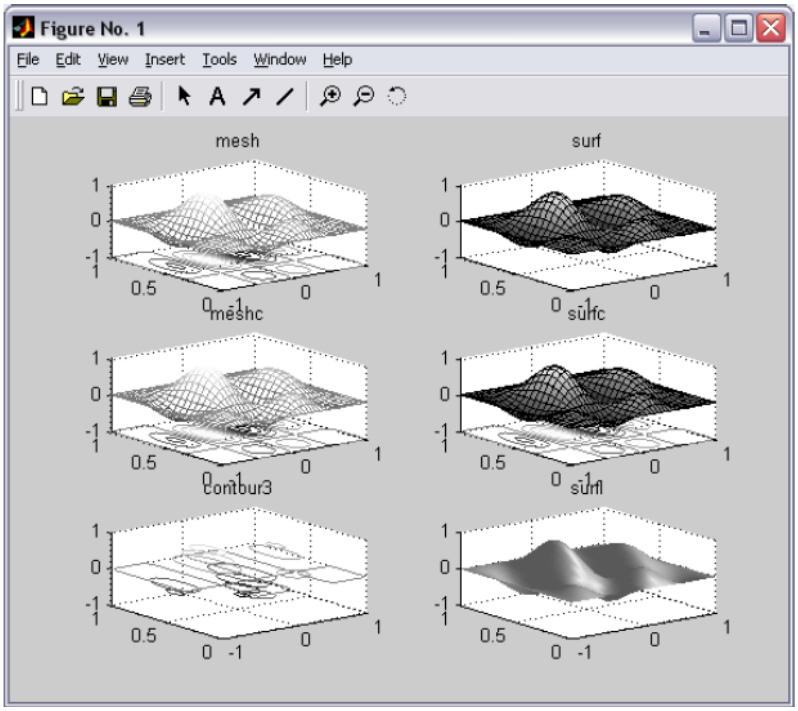


Рис. 9.14

Постройте графики функции

$$z(x,y) = 4 \cdot \sin 2\pi x \cdot \cos 1.5\pi y \cdot \left(1 - x^2\right) \cdot y \cdot (1 - y)$$

на прямоугольной области определения $x \in [-1, 1]$, $y \in [0, 1]$ всеми известными способами, размещая их на отдельных подграфиках. Названия команд, применяемых для построения графиков, включите в заголовки подграфиков.

В результате получается графическое окно, которое содержит шесть подграфиков, наглядно демонстрирующих способы построения трехмерных графиков в MatLab (рис. 9.14). Последняя команда **colormap (gray)** изменяет палитру всего графического окна, а не подграфиков по отдельности.

10. М-ФАЙЛЫ

10.1. Общие сведения

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 облегчает работу. Самым удобным способом выполнения 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. При помощи редактора М-файлов можно создавать собственные функции и вызывать их, в том числе и из командной строки.

10.2. Работа в редакторе М-файлов

Раскройте меню **File** основного окна MatLab и в пункте **New** выберите подпункт **M-file**. Наберите в редакторе команды, приводящие к построению двух графиков на одном графическом окне. Необязательно набирать много команд – наша цель сейчас состоит в том, чтобы научиться выполнять команды из редактора М-файлов. Например,

```
x = [0 ; 0.1 ; 7];
f = exp (–x);
subplot (1,  2,  1)
plot (x,  f)
g = sin (x);
subplot (1,  2,  2)
plot (x,  g)
```

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 No.1**, содержащее графики функций. Если вы решили построить график косинуса вместо синуса, то просто измените строку $g = \sin x$ в М-файле на $g = \cos x$ и запустите все команды снова.

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

Очень удобной возможностью, предоставляемой редактором М-файлов, является выполнение части команд. Закройте графическое окно **Figure No.1**. Выделите при помощи мыши, удерживая левую кнопку, первые четыре команды и выполните их из пункта **Evaluate Selection** меню **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 **<F9>**. Выполните оставшиеся три команды и проследите за состоянием графического окна. Потренируйтесь самостоятельно, наберите какие-либо примеры из предыдущих глав в редакторе М-файлов и запустите их.

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

```
%построение графика sin(x) в отдельном окне
```

В редакторе М-файлов может быть открыто несколько файлов. Переход между файлами осуществляется при помощи закладок с именами файлов, расположенных внизу окна редактора. Открытие существующего М-файла производится при помощи пункта **Open** меню **File** рабочей среды либо редактора М-файлов. Открыть файл в редакторе можно и командой MatLab **edit** из командной строки, указав в качестве аргумента имя файла, например:

```
>> edit mydemo
```

Команда **edit** без аргумента приводит к созданию нового файла.

Типы М-файлов

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

Файл-функции с одним входным аргументом

Предположим, что в вычислениях часто необходимо использовать функцию

$$e^{-x} \cdot \sqrt{\frac{x^2+1}{x^4+0.1}}.$$

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

```
function f = myfun (x)
f = exp (–x)*sqrt ((x^2+1)/(x^4+0.1));
```

Слово **function** в первой строке определяет, что данный файл содержит файл-функцию. Первая строка является заголовком функции, в которой размещается имя функции и списки входных и выходных аргументов. В примере, приведенном выше, имя функции **myfun**, один входной аргумент – *x* и

один выходной – *f*. После заголовка следует *тело функции* (оно в данном примере состоит из одной строки), где и вычисляется ее значение. Важно, что вычисленное значение записывается в *f*. Не забудьте поставить точку с запятой для предотвращения вывода лишней информации на экран.

Теперь сохраните файл в рабочем каталоге. Выбор пункта **Save** или **Saveas** меню **File** приводит к появлению диалогового окна сохранения файла, в поле **Filename** которого уже содержится название**myfun**. 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, например, из командной строки:

```
>> y = myfun (1.3)
y =
        0.2600
```

Вызов собственных функций может осуществляться из файл-программы и из другой файл-функции.

Файл-функция, приведенная выше, имеет один существенный недостаток. Попытка вычисления значений функции от массива приводит к ошибке, а не к массиву значений, как это происходит при вычислении встроенных функций.

Необходимо просто при вычислении значения функции использовать поэлементные операции.

Измените тело функции на

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

(не забудьте сохранить изменения в файле **myfun.m**).

Теперь аргументом функции **myfun** может быть как число, так и вектор или матрица значений, например:

```
>> x = [1.3  7.2];
>> y = myfun (x)
y =
        0.2600         0.0001
```

Переменная *y*, в которую записывается результат вызова функции**myfun**, автоматически становится вектором нужного размера.

Постройте график функции **myfun** на отрезке [0, 4] из командной строки или при помощи файл-программы:

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

MatLab предоставляет еще одну возможность работы с файл-функциями – использование их в качестве аргументов некоторых команд. Например, для построения графика служит специальная функция **fplot**, заменяющая последовательность команд, приведенную выше. При вызове **fplot** имя функции, график которой требуется построить, заключается в апострофы, пределы построения указываются в вектор-строке из двух элементов:

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

Постройте график **myfun** разными способами при помощи команд **plot** и**fplot** на одних осях, используя **hold on**. Обратите внимание, что график, построенный при помощи **fplot**, более точно отражает поведение функции, так как **fplot** сама подбирает шаг аргумента, уменьшая его на участках быстрого изменения отображаемой функции.

Файл-функции с несколькими входными аргументами

Все входные аргументы размещаются в списке через запятую. Например, файл-функция, вычисляющая длину радиус-вектора точки трехмерного пространства $\sqrt{x^2 + y^2 + z^2}$:

```
function r = radius3 (x,  y,  z)
r = sqrt (x.^2 + y.^2 + z.^2);
```

Для вычисления длины радиус-вектора теперь можно использовать функцию **radius3**, например:

```
>> R = radius3(1,  1,  1)
R =
        1.732
```

Файл-функции с несколькими выходными аргументами

Файл-функции с несколькими выходными аргументами удобны при вычислении функций, возвращающих несколько значений (в математике они называются *вектор-функции*). 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:

```
function [hour, minute, second] = hms (sec)
hour = floor (sec/3600);
minute = floor ((sec-hour*3600)/60);
second = sec-hour*3600-minute*60;
```

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

```
>> [H,  M,  S] = hms (10000)
H =
        2
M =
        46
S =
        40
```

Если список выходных аргументов пуст, т. е. заголовок выглядит так: **function myfun (a, b)** или **function [ ] = myfun (a, b)**, то файл-функция не будет возвращать никаких значений. Такие функции тоже иногда оказываются полезными.

Предусмотрена также возможность создавать файл-функции, которые сами приспосабливаются к числу входных и выходных аргументов. Большинство встроенных функций работают именно таким образом.

11. Численные методы и программирование

Решение уравнений

Нахождение корней произвольных уравнений осуществляет встроенная функция **fzero**, для определения всех корней полиномов применяется **roots**.

Встроенная функция **fzero** позволяет приближенно вычислить корень уравнения по заданному начальному приближению. В самом простом варианте **fzero** вызывается с двумя входными и одним выходным аргументом ***x*** = **fzero (‘myf’, *x*0)**, где **myf** – имя файл-функции, вычисляющей левую часть уравнения; ***x*0** – начальное приближение к корню; ***x*** – найденное приближенное значение корня. Решите, например, на отрезке [–5,5] уравнение $\sin x - x^2 \cos x = 0$.

Перед нахождением корней полезно построить график функции, входящей в левую часть уравнения. Конечно построить график можно при помощи **plot**, но все равно понадобится написать файл-функцию, поэтому имеет смысл воспользоваться **fplot**, которая к тому же позволяет получить более точный график по сравнению с **plot**.

Ниже приведен текст требуемой файл-функции:

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

Теперь постройте график **myf**, используя **fplot**, и нанесите сетку.

```
fplot (‘myf’,  [–5  5])
grid on
```

Из графика **myf** видно, что функция на этом отрезке имеет четыре корня. Один корень равен нулю, в чем нетрудно убедиться, подставив $x = 0$ в уравнение.

Уточните значение корня, расположенного вблизи $x = -5$, при помощи **fzero**:

```
>> x1 = fzero (‘myf’, –5)
Zero found in the interval: [–4.7172,   –5.2].
x1 =
        –4.7566
```

Приближенное значение корня равно –4.7566. Проверьте ответ, вычислив значение функции **myf** в точке *x*1:

```
>> myf (x1)
ans =
        2.6645e –015
```

Значение функции близко к нулю, не означает, что приближенное значение корня расположено достаточно близко к его точному значению.

Важной особенностью **fzero** является то, что она вычисляет только те корни, в которых функция меняет знак, а не касается оси абсцисс.

В качестве исследуемой функции может выступать и встроенная математическая функция, например,

```
>> fzero ('sin', [2  4])
Zero found the interval:  [2,  4].
ans =
        3.14159265358979
```

Вычисление всех корней полинома

Полином в MatLab задается вектором его коэффициентов. Например, для определения полинома

$$p = x^7 + 3.2x^5 - 5.2x^4 + 0.5x^2 + x - 3$$

следует использовать команду

```
>> p = [1  0  3.2  –5.2  0  0.5  1  –3];
```

Число элементов вектора, т.е.число коэффициентов полинома, всегда на единицу больше его степени, нулевые коэффициенты должны содержаться в векторе.

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

```
>> polyval (p,  1)
ans =
        –2.5000
```

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

Нахождение сразу всех корней полиномов осуществляется при помощи функции **roots**, в качестве аргумента которой указывается вектор с коэффициентами полинома. Функция **roots** возвращает вектор корней полинома, в том числе и комплексных:

```
>> r = roots (p)
r =
        –0.5668 + 2.0698i
        –0.5668 – 2.0698i
         1.2149
         0.5898 + 0.6435i
         0.5898 – 0.6435i
        –0.6305 + 0.5534i
        –0.6305 – 0.5534i
```

Число корней полинома, как известно, совпадает со степенью полинома. Убедитесь в правильности работы **roots**, вычислив значение полинома от вектора его корней:

```
>> polyval (p,  r)
ans =
        1.0e–012  *
        –0.1008 + 0.0899i
        –0.1008 – 0.0899i
        –0.0666
         0.0027 – 0.0018i
         0.0027 + 0.0018i
         0.0102 – 0.0053i
         0.0102 + 0.0053i
```

Обратите внимание, что в верхней строке результата содержится общий множитель 1.0e–012, на который следует помножить каждое число получившегося вектора.

12. Операторы цикла

Выполнение схожих повторяющихся действий в MatLab осуществляется при помощи операторов циклов **for** и **while**. Цикл **for** предназначен для выполнения заданного числа повторяющихся действий, а **while** – для действий, число которых заранее не известно, но известно условие продолжения цикла.

12.1. Цикл for

Самое простое использование **for** осуществляется следующим образом:

```
for count = start : step : final
        команды MatLab
end
```

Здесь **count** – переменная цикла, **start** – ее начальное значение, **final** – конечное значение, а **step** – шаг, на который увеличивается count при каждом следующем заходе в цикл. Цикл заканчивается, как только значение count становится больше **final**. Переменная цикла может принимать не только целые, но и вещественные значения любого знака. Разберем применение цикла**for** на некоторых характерных примерах.

Пусть требуется вывести семейство кривых для $x \in [0, 2\pi]$, которое задано функцией, зависящей от параметра $y(x,a) = e^{-ax} \sin x$, для значений параметра от –0,1 до 0,1. Можно, конечно, последовательно вычислять *y(x,a)* и строить ее графики для различных значений *a* от –0,1 до 0,1, но гораздо удобнее использовать цикл **for**. Наберите текст файл-процедуры в редакторе М-файлов:

```
figure
x = [0 : pi / 30 : 2*pi];
for a = –0.1 : 0.02 : 0.1
        y = exp (-a*x) .* sin(x);
        hold on
        plot(x,  y)
end
```

Сохраните в файле **FORdem1.m** и запустите его на выполнение (или из редактора М-файлов, или из командной строки, набрав в ней команду **FORdem1** и нажав <**Enter**>).

В результате выполнения **FORdem1** появится графическое окно, которое содержит требуемое семейство кривых.

Напишите файл-программу для вычисления суммы

$$S = \sum_{k=1}^{10} \frac{1}{k!}$$

Алгоритм вычисления суммы использует накопление результата, т.е. сначала сумма равна нулю, затем в переменную *k* заносится единица, вычисляется 1/*k*! (т.е. 1/1!), добавляется к *S* и результат снова заносится в *S*. Далее*k* увеличивается на единицу и процесс продолжается, пока последним слагаемым не станет 1/10!.

Файл-программа **FORdem2** вычисляет искомую сумму.

```
% ФАЙЛ-ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ СУММЫ
%       1/1! + 1/2! + … + 1/10!
% обнуление S для накопления суммы
S = 0;
% накопление суммы в цикле
for  k = 1 : 10
        S = S + 1 / factorial (k);
end
% вывод результата в командное окно
S
```

Если шаг цикла равен 1, то его можно не указывать.

Наберите файл-программу в редакторе М-файлов, сохраните в текущем каталоге в файле **FORdem.m** и выполните ее. Результат отображается в командном окне, так как в последней строке файл-программы содержится *S* без точки с запятой для вывода значения переменной *S*

```
S =
        1.7183
```

Остальные строки файл-программы, которые могли бы повлечь вывод на экран промежуточных значений, завершаются точкой с запятой для подавления вывода в командное окно.

Первые две строки с комментариями не случайно отделены пустой строкой от остального текста программы. Именно они выводятся на экран, когда пользователь при помощи команды **help** из командной строки получает информацию о том, что делает **FORdem2**

При написании файл-программ и файл-функций, предназначенных для дальнейшей работы с ними, не пренебрегайте комментариями!

Важно понять, что все переменные, использующиеся в файл-программе, становятся доступными в рабочей среде. Они являются так называемыми *глобальными переменными*. Например, для получения значения *k* после выполнения **FORdem2** следует просто набрать *k* в командной строке и нажать <**Enter**>. Результат очевиден, так как последний раз цикл **for** выполнялся как раз для *k*, равного десяти. С другой стороны, в файл-программе могут использоваться все переменные, введенные в рабочей среде.

Вычислим, например, сумму, похожую на предыдущую, но зависящую еще от переменной *x*:

$$S(x)=\sum_{k=1}^{10}\frac{x^k}{k!}$$

Для вычисления данной суммы в файл-программе **FORdem** требуется изменить строку внутри цикла **for** на

```
S = S + x .^ k / factorial(k);
```

Перед запуском программы следует определить переменную *x* в командной строке. Вычисление, например *S*(1.5) производится из командной строки при помощи следующих команд:

```
>> x = 1.5;
>> FORdem2
S =
        3.4817
```

В качестве *x* может быть вектор или матрица, поскольку в файл-программе **FORdem2** при накоплении суммы использованы поэлементные операции.

Перед запуском **FORdem2** нужно обязательно присвоить переменной *x* некоторое значение, а для вычисления суммы, например, из пятнадцати слагаемых, придется внести изменения в текст файл-программы. Гораздо лучше написать универсальную файл-функцию, у которой в качестве входных аргументов будут значение *x* и верхнего предела суммы, а выходным – значение суммы *S(x)*. Использование операторов, в частности циклов, в файл-функциях производится так же, как и в файлах-программах.

Файл-функция sumN, вычисляющая *S(x)*, имеет вид

```
function s = sumN(x,  N)
% ФАЙЛ-ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ СУММЫ
%       x/1! + x^2/2! + … + x^N/N!
% использование:    S = sum(x,  N)

% обнуление S для накопления суммы
S = 0;
% накопление суммы в цикле
for m = 1 : 1 : N
        S = S + x.^m/factorial(m);
end
```

Об использовании функции **sumN** пользователь может узнать, набрав в командной строке **helpsumN**. В командное окно выведутся первые три строки с комментариями, отделенные от текста файл-функции пустой строкой. Для *x* = 1.5 и *N* = 10 функция **sumN** даст тот же результат, что и **FORdem2**:

```
>> S = sumN (1.5,  10)
S =
        3.4817
```

Обратите внимание, что внутренние переменные файл-функции не являются глобальными (*m* в файл-функции **sumN**). Попытка просмотра значения переменной *m* из командной строки приводит к сообщению о том, что

m не определена. Если в рабочей среде имеется *глобальная* переменная с тем же именем *m*, определенная из командной строки или в файл-программе, то она никак не связана с *локальной* переменной *m* в файл-функции. Как правило, лучше оформлять собственные алгоритны в виде файл-функций для того, чтобы переменные, используемые в алгоритме, не портили значения одноименных глобальных переменных рабочей среды. Впрочем, при необходимости, файл-функция может использовать глобальные переменные.

Циклы **for** могут быть *вложены* друг в друга, при этом переменные вложенных циклов должны быть *разными*. Вложенные циклы удобны для заполнения матриц. Например, во встроенной справке MatLab по оператору **for**содержится пример заполнения матрицы Гильберта при помощи вложенных циклов. Элементы матрицы Гильберта порядка *n* определяются формулами $a_{i,j} = 1/(i + j - 1)$ для $i, j = 1,2,...,n$.

Скопируйте пример в новый М-файл при помощи буфера обменаWindows, сохраните с именем **HILdem.m** и продолжите работу с ним, как с файл-программой.

Перед инициализацией матрицы следует задать ее размер. Для вывода матрицы на экран достаточно добавить в конце файл-программы имя массива, содержащего матрицу. Следует снабдить части файл-программы комментариями. После небольших измерений в скопированном примере файл-программа **HILdem** выглядит так:

```
% Задание размера матрицы
n = 4;
% Инициализация матрицы и заполнение ее нулями
a = zeros (n,  n)
% Вычисление матрицы Гильберта порядка n
for i = 1 : n
        for j = 1 : n
                a(i,  j) = 1/(I + j – 1);
        end
end
% Вывод матрицы Гильберта на экран
A
```

Запуск **HILdem** выводит в командное окно матрицу Гильберта четвертого порядка и приводит к появлению в рабочей среде новой глобальной переменной – массива ″a″ размера четыре на четыре, содержащего элементы матрицы Гильберта.

Перед заполнением матриц или векторов следует сначала создать их и заполнить нулями командой **zeros** для увеличения скорости алгоритма. Команда ***a*= zeros (n, n)** выполняется быстрее, чем последовательность ***a* (*i, j*)= 0** для всех *i* и *j* от единицы до *n*.

Исследуем границы спектра матрицы Гильберта для различных порядков матрицы (от первого до некоторого N-го) и отобразим результат в виде графиков значений максимального и минимального собственных чисел в зависимости от размера матрицы. Для решения этой задачи требуется N раз выполнить операторы файл-функции **HILdem**, изменяя *n*от единицы до N. После вычисления матрицы Гильберта порядка *n* необходимо:

1. Найти ее спектр при помощи встроенной функции **eig.**

2. Определить границы спектра, т. е. максимальный и минимальный элемент вектора – аргумента **eig.**

3. Запомнить границы в элементах с индексом *n* некоторых вспомогательных векторов.

В конце следует визуализировать результат.

Пример. Напишите файл-функцию **HilSpect**, которая выводит в графическое окно на разные графики зависимости максимального и минимального собственных чисел матрицы Гильберта от порядка матрицы (для минимального собственного числа используйте логарифмическую шкалу по оси ординат).

Входным аргументом файл-функции **HilSpect** должен быть максимальный порядок исследуемых матриц, выходные аргументы в данном случае не нужны.

Пример. Исследуйте границы спектра матрицы Гильберта до двенадцатого порядка.

В результате вызова **HilSpect(12)** появится графическое окно.

При решении практических задач, требующих много вычислений, становится актуальной проблема *временных затрат* компьютера. Оказывается, что заполнение матриц лучше осуществлять при помощи индексации или встроенных функций MatLab, если структура матрицы позволяет это. Например, для получения матрицы Гильберта предназначена функция **hilb**.

```
function HilSpect (N)
% Исследование границ спектра матрицы Гильберта
% использование: HilSpect (N), N – максимальный порядок
% инициализация массивов для границ спектра
Lmax = zeros (1,  N);
Lmin = zeros (1,  N);
% вычисления для матриц от первого порядка до N-го
for n =  1 : N
        % заполнение матрицы Гильберта, вместо вложенных циклов
        % можно использовать встроенную функцию A = hilb (n);
        A = zeros (n);
        for k = 1 : n
                for j = 1 : n
                        A (k, j) = 1/(k + j – 1);
                end
        end
        % вычисление спектра и его границ
        Lambda = eig (A);
        Lmax (n) = max (Lambda);
        Lmin (n) = min (Lambda);
end
% вычисление спектра и границ спектра от порядка матрицы
figure;
subplot (2,  1,  1)
plot (Lmax,  'ko–')
title ('Максимальное собственное число матрицы Гильберта')
xlabel (' \ itN ')
grid on
subplot (2,  1,  2)
semilogy (Lmin,  'k*–')
title ('Минимальное собственное число матрицы Гильберта')
xlabel (' \ itN ')
grid on
```

Цикл **for** оказывается полезным при выполнении повторяющихся похожих действий в том случае, когда их число заранее определено. Обойти это ограничение позволяет более гибкий цикл **while**,способ применения которого приведен ниже.

12.2. Цикл while, суммирование рядов

Рассмотрим пример на вычисление суммы. Требуется найти сумму для заданного *x* (разложение в рядsin x):

$$S(x)=\sum_{k=0}^{\infty}(-1)^k\frac{x^{2k+1}}{(2k+1)!}.$$

Конечно, до бесконечности суммировать не удастся, но можно накапливать сумму, пока слагаемые являются не слишком маленькими, скажем больше10^{-10}(по модулю). Циклом **for** здесь не обойтись, так как заранее неизвестнозначение *k*, обеспечивающее малость текущего слагаемого. Выход состоит вприменении цикла **while**, который работает, пока выполняется *условие цикла*:

```
hile условие цикла
        команды MatLab
end
```

В данном примере условием цикла является то, что текущее слагаемое $\frac{x^k}{k!}$ больше 10^{-10}. Для записи условия в формате, понятной MatLab, следует использовать знак больше (>).

Конечно, малость слагаемого – понятие относительное, слагаемое можетбыть, скажем, порядка10^{-10}, но и сама сумма должна быть того же порядка. В этом случае нельзя останавливать суммирование. Пока не будем обращать на это внимания – нашей задачей является изучение работы с циклами.

Текст файл-функции **mysin**, вычисляющей сумму ряда:

```
function s = mysin (x)
% Вычисление синуса разложением в ряд
% Использование: y = mysin (x),  –pi < x < pi
s = 0;
k = 0;
while  abs (x .^ (2*k+1) / factorial (2*k+1)) > 1.0e –10
        s = s + (–1) ^k*x. ^ (2*k+1) / factorial (2*k+1);
        k = k + 1;
end
```

Обратите внимание, что у цикла **while**, в отличие от **for**, нет переменной, поэтому пришлось до начала цикла *k* присвоить ноль, а внутри увеличивать *k* на единицу.

Сравните теперь результат, построив графики функций **mysin** и **sin** на отрезке $[-\pi,\pi]$ на одних осях, например, при помощи **fplot** (команды можно задать из командной строки):

```
>> fplot ('mysin', [–pi, pi])
>> hold on
>> fplot ('sin', [–pi, pi],  'k.')
```

Условие цикла **while** может содержать не только знак >. Для задания условия выполнения цикла допустимы также другие *операции отношения*:

```
= =    – равенство.
<      – меньше.
< =    – меньше или равно.
> =    – больше или равно.
~ =    – не равно.
```

Задание более сложных условий производится с применением логических операторов. Например, условие $-1 \le x < 2$ состоит в одновременном выполнении неравенств $x \ge -1$ и $x < 2$ и записывается при помощи логического оператора **and**:

```
and (x >= –1, x < 2)
```

или эквивалентным образом с символом **&**:

```
(x >= –1) & (x < 2)
```

Логические операторы и примеры их использования приведены в табл. 12.2.

Таблица 12.2

Оператор	Условие	Запись в MatLab	Эквивалентная запись
Логическое “И”	x < 3 и k = 4	and (x < 3, k == 4)	(x < 3) & (k == 4)
Логическое “Или”	x = 1, 2	or (x == 1, x == 2)	(x == 1) | (x == 2)
Отрицание “Не”	a ≠ 1.9	not (a == 1.9)	~ (a == 1.9)

При вычислении суммы бесконечного ряда имеет смысл ограничить число слагаемых. Если ряд сходится из-за того, что его члены не стремятся к нулю, то условие на малость текущего слагаемого может никогда не выполниться и программа зациклится. Выполните суммирование, ограничив число слагаемых. Добавьте в условие цикла **while** файл-функции **mysin** ограничение на малость слагаемого:

```
(abs(x.^(2*k+1) / factorial (2*k+1)) > 1.0e –10) ((k <= 100000)
```

или в эквивалентной форме:

```
and(abs(x.^(2*k+1) / factorial (2*k+1)) > 1.0e –10, k <= 100000).
```

Для задания порядка выполнения логических операций следует использовать круглые скобки, например записи (*x* = = 1) / (*x* = = 2) & (*y* = = 3) и (*x* = =1) | ((*x* = = 2) & (*y* = = 3)) не эквивалентны в MatLab, в отличие от многих языков программирования.

13. Операторы ветвления. Исключительные ситуации

Условный оператор **if** и оператор переключения **switch** позволяют создать гибкий разветвляющийся алгоритм выполнения команд, в котором при выполнении определенных условий работает соответствующий блок операторов или команд MatLab. Практически во всех языках программирования имеются аналогичные операторы.

13.1. Условный оператор if

Оператор **if** может применяться в простом виде для выполнения блока команд при удовлетворении некоторого условия или в конструкции **if–elseif–else** для написания разветвляющих алгоритмов.

Проверка входных аргументов

Начнем с простейшего примера – файл-функции для вычисления выражения $\sqrt{x^2-1}$.

Создание файл-функции не должно вызвать затруднений. Она работает для любых значений *x*, причем для –1 <*x*< 1 результат является комплексным числом. Предположим, что вычисления происходят в области действительных чисел и требуется вывести предупреждение о том, что результат является комплексным числом. Перед вычислением функции следует произвести проверку значения аргумента *x* и вывести в командное окно предупреждение, если модуль *x* не превосходит единицы. Здесь уже не обойтись без условного оператора **if**, применение которого в самом простом случае выглядит так:

```
if условие
        команды MatLab
end
```

Если условие верно, то выполняются команды MatLab, размещенные между **if** и **end**, а если условие неверно, то происходит переход к командам, расположенным после **end**. При записи условия используются операции следующие отношения:

```
= =   – равенство.
<     – меньше.
< =   – меньше или равно.
> =   – мольше или равно.
~ =   – не равно.
```

Команда **warning** служит для вывода предупреждения в командное окно.

Файл-функция, проверяющая значение аргумента:

```
function f = Rfun (x);
% вычисляет sqrt (x^2–1)
% выводит предупреждение, если результат комплексный
% использование y = Rfun (x)
% проверка аргумента
if abs (x) < 1
        warning ('результат комплексный')
end
% вычисление функции
f = sqrt (x^2 – 1);
```

Теперь вызов **Rfun** от аргумента, меньшего единицы по модулю, приведет к выводу в командное окно предупреждения:

```
>> y = Rfun (0.2)
результат комплексный
y =
        0 + 0.97979589711327i
```

Файл-функция **Rfun** только предупреждает о том, что ее значение комплексное, все вычисления с ней продолжаются. Если же комплексный результат означает ошибку вычислений, то следует прекратить выполнение функции, используя команду **error** вместо **warning**.

Пример. Напишите файл-функцию **root2**, которая по коэффициентам квадратного уравнения находит только вещественные его корни, а для комплексных выдает ошибку.

При составлении файл-функций следует предусмотреть еще один вид контроля – проверку количества входных и выходных параметров. Если пользователь вызовет функцию **root2** с двумя входными параметрами, то получит сообщение об ошибке при выполнении того оператора файл-функции, который содержит неопределенный параметр. В случае вызова функции **root2** с одним выходным аргументом или без аргументов будет вычислен только первый корень квадратного уравнения, что также введет пользователя в заблуждение. Лучше заранее предупредить пользователя о характере ошибки и прекратить работу файл-функции. Кроме того, следует учесть, что файл-функция **root2** не может принимать массивы в качестве входных аргументов.

Если даже использовать поэлементные операции при вычислениях, то дискриминант уравнения **D** будет массивом, а что такое **D**> 0, для массива пока неизвестно.

Дополните функцию **root2** вышеописанными видами контроля, предотвращающими неправильное ее использование. Встроенные функции **nargin** и **nargout** возвращают число входных и выходных аргументов соответственно. Для проверки, являются ли входные аргументы числами, следует сначала найти размеры соответствующих переменных при помощи **size,** а затем проверить их на равенство единице. Вывод текста в командное окно в ходе выполнения файл-программы или файл-функции осуществляется оператором **disp**, сам текст указывается в апострофах: **disp (‘текст’).**

Организация ветвления

В общем виде оператор ветвления представляет конструкцию **if–elseif–else,** работу которой хорошо поясняет пример файл-функции **ifdem**:

```
function ifdem (a);
% Пример использования структуры if–elseif–else
if  (a  = =  0)
        disp (‘a равно нулю’)
elseif  a  = =  1
        disp (‘a равно единице’)
elseif  a  = =  2
        disp (‘a равно двум’)
elseif  a  >=  3
        disp (‘a больше или равно трем’)
else
        disp (‘a меньше трех, но не ноль, не единица и не двойка’)
end
```

В зависимости от выполнения того или иного условия работает соответствующая ветвь программы, если все условия неверны, то выполняются команды, размещенные после **else**. Вызовы функции **ifdem** с различными аргументами позволяют убедиться в вышесказанном:

```
>> ifdem (1)
а равно единице
>> ifdem (1.2)
а меньше трех, но не ноль, не единица и не двойка
>> ifdem (2)
а равно двум
>> ifdem (1.2)
а меньше трех, но не ноль, не единица и не двойка
>> ifdem (2)
а равно двум
```

Ветвей может быть сколько угодно (добавьте несколько **elseif** с похожими условиями в **ifdem**) или только две, например:

```
if (a == 0)
        disp ('а равно нулю')
else
        disp ('а не равно нулю')
end
```

В случае двух ветвей используется завершающее **else**, а **elseif** пропускается. Оператор **if** должен заканчиваться **end**.

Файл-функция **ifdem** хорошо демонстрирует работу оператора **if**, но на практике оказывается бесполезной. Действительно полезный пример – вычисление кусочно-заданной функции

$$y(x)=\begin{cases}\sin x, & x<-\pi,\\ x/\pi, & -\pi\le x<\pi,\\ -\cos x, & x\ge\pi,\end{cases}$$

которая реализуется файл-функцией **pwfun**. Ее текст приведен ниже:

```
function y = pwfun (x);
% вычисляет кусочно-линейную функцию
%       sin(x) –1,              если x< –pi
% y(x) =                x,      если –pi <=x <0
%       pi*cos(x),      если x >= 0
% использование y = pwfun(x), x – число;
if  x < –pi
        y = sin (x)  –1
elseif  x < pi  % проверка x > –pi не нужна!
        y = x / pi;
else            % здесь x > pi
        y = –cos (x);
end
```

Внимание: число ветвей **if–elseif–else** равно трем.

Во второй ветви достаточно только проверить, что $x< \pi$, условие $x \geq -\pi$ *уже выполнено* (иначе бы отработала первая ветвь в **if–elseif–else**, и оператор **if** закончил работу).

В последней ветви нет смысла проверять никакие условия, она работает, если все предыдущие условия неверны, что как раз соответствует $x \geq \pi$.

Для построения графика кусочно-заданной функции **pwfun** следует воспользоваться командой **fplot ('pwfun', [–3*pi, 3*pi])**.

Построение графика **pwfun** не случайно осуществляется при помощи **fplot**. Функцией **plot** воспользоваться не удастся, так как требуется предварительно вычислить вектор значений функции от вектора аргументов, а **pwfun**не умеет работать с входным *аргументом-вектором*. Убедиться в этом можно, построив график **pwfun** командами

```
x = [–3*pi : 0.1 : 3*pi];
y = pwfun(x);
plot (x, y)
```

Никакой ошибки при выполнении файл-функции не возникает, однако график строится неправильно. Дело в том, что вектор *x* входит в условия оператора **if**. Операции отношения (<, <=, >, >=, ~=) могут *специальным образом*применяться и к векторам, а при обычном пользовании в данном

примере не дают ожидаемого результата. Выход состоит в применении следующего алгоритма вычисления кусочно-заданной функции, для реализации которого достаточно понимания вышеописанного материала.

1. *Проверка числа входных аргументов: если число входных аргументов не равно единице, то происходит завершение работы файл-функции с сообщением об ошибке (выход по ошибке).*

2. *Проверка, является ли входной аргумент вектором или числом, один из размеров которого должен быть равен единице. Если это условие не выполняется, то происходит выход по ошибке.*

3. *Нахождение длины входного аргумента*.

4. *Создание вектора выходного аргумента того же размера, что и входной аргумент, и заполнение его нулями.*

5. *Перебор всех элементов входного вектора с использованием цикла for, вычисление от них значений кусочно-заданной функции и запись в соответствующие элементы выходного вектора.*

Попытайтесь составить файл-функцию**pwfun1** самостоятельно (входной аргумент – число или вектор).

Входными аргументами файл-функции **pwfun1** могут быть как число, так и вектор, причем если входной аргумент является вектор-строкой (вектор-столбцом), то результат тоже вектор-строка (вектор-столбец).

В качестве завершающего упражнения попытайтесь улучшить **pwfun1**(создав новый файл-функцию **pwfun2**) так, чтобы ее входным аргументом могла быть и матрица. Учтите, что вектор в MatLab, так же как и матрица, является двумерным массивом, у которого один из размеров равен единице. Очевидно, что для перебора элементов входного аргумента придется использовать вложенные циклы **for**.

13.2. Onepamop switch

Предположим, что при работе с функцией двух аргументов

$$e^{-|x\cdot y|}\sin\pi x\cdot\cos\pi x^2$$

приходится визуализировать ее четырьмя различными способами: каркасной поверхностью, сплошной поверхностью, выводить линии уровня, строить освещенную поверхность. Удобно создать файл-функцию, один из входных аргументов которой **vis** будет определять способ визуализации. Если **vis** равен единице, то строится каркасная поверхность, для **vis,** равного двум, – сплошная и т. д. Можно, конечно, использовать оператор **if** в полном виде **if-elseifelse**, который в зависимости от значения **vis** выполняет нужную ветвь программы, выводящую соответствующий график. Однако оператор переключения **switch** позволяет написать более наглядную программу см. следующий фрагмент:

```
switch a
        case  –1
                disp('a = –1')
        case   0
                disp('a = 0')
        case  (2,  3,  4)
                disp('a равно 2 или 3 или 4')
        otherwise
                disp('a не равно-1, 0, 1')
end
```

Каждая ветвь определяется оператором **case**, переход в нее выполняетсятогда, когда переменная оператора **switch** (в данном примере **''a''**) принимает значение, указанное после **case**, или одно из значений из списка **case**. После выполнения какой-либо из ветвей происходит выход из **switch**, при этом значения, заданные в других **case**, уже *не повторяются*. Если подходящих значений для **''a''** не нашлось, то выполняется ветвь программы, соответствующая **otherwise**.

Оператор **switch** как нельзя лучше подходит для решения поставленной задачи о выводе различных графиков исследуемой функции. Попытайтесь написать файл-функцию (**myplot3D**) самостоятельно. Входными аргументами являются границы построения исследуемой функции по каждой из переменных **xmin**, **xmax**, **ymin**, **ymax** и способ построения графика, определяемый **vis.** Все

пять входных аргументов должны быть числами, причем **xmin** меньше **xmax** и **ymin** меньше **ymax** – не забудьте сделать соответствующую проверку! Выходные аргументы в данном случае не требуются.

Оператор **switch** удобно применять тогда, когда есть соответствие между*дискретными* значениями некоторой переменной и последующими действиями.

Список литературы

1. Якунина С.В. Математическое программное обеспечение. Версия 1.0 [Электронный ресурс] : электрон. учеб. пособие / С. В. Якунина. – Электрон. дан. (5 Мб). – Красноярск : ИПК СФУ, 2008.
2. Ануфриев, И. Е. Самоучитель MatLab 5.3/6.x / И. Е. Ануфриев. – СПб.: БХВ-Петербург, 2002. – 736 с.
3. Леоненков, А. Нечёткое моделирование в среде MATLAB и fuzzyTECH / А.Леоненков. – СПб.: БХВ-Петербург, 2003. – 736 с.

Подписано в печать 17.10.2017 г.
Формат 60х84/15 Бумага офсетная Печать ризографическая
Уч.изд.л. 8,4Усл.-печ.л. 8,4 Тираж 50 экз.
Заказ №723
Полиграфический центр «Астэрия»

423800, г. Набережные Челны, Новый город, проспект Академика Рубаненко,12
Тел./факс (8552) 36-28-88, e-mail:asterio@mail.ru