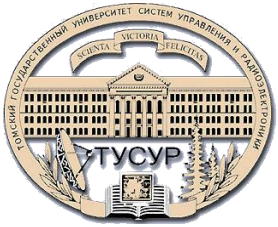
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)



Кафедра радиотехнических систем (РТС)

Д.О. Ноздреватых

НАЧАЛЬНЫЕ СВЕДЕНИЯ О MATLAB

Учебное пособие
для студентов технических вузов

2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра радиотехнических систем (РТС)

Утверждаю:
Зав. кафедрой РТС, проф., д.т.н.
_____ С.В. Мелихов
_____ 2016 г.

Д.О. Ноздреватых

НАЧАЛЬНЫЕ СВЕДЕНИЯ О MATLAB

Учебное пособие
для студентов технических вузов

Разработчик:
Ст. преподаватель каф. РТС
_____ Ноздреватых Д.О.
_____2016 г.

2016

АННОТАЦИЯ

Учебное пособие включает в себя теоретический материал и список вопросов для самоконтроля студентов технических вузов по работе в системе компьютерной математики **MatLab** при изучении дисциплин «Информационные технологии», «Информатика» и других дисциплин, использующих на лабораторных занятиях программирование и моделирование. Учебное пособие поможет студентам выполнить индивидуальные задания при прохождении учебной вычислительной практике и по практике по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

Учебное пособие предназначено для подготовки студентов **направления подготовки 11.05.01 «Радиоэлектронные системы и комплексы», 11.03.02 «Инфокоммуникационные технологии и системы связи», 11.03.01 «Радиотехника»** и другие.

СОДЕРЖАНИЕ

ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ

В данном разделе перечислены основные цели и задачи по дисциплинам «Информационные технологии», «Информатика».

Цели и задачи изучения дисциплины **«Информационные технологии», «Информатика»** заключаются в обеспечении базовой подготовки студентов в области использования средств вычислительной техники и ознакомлении с основами проектирования и программирования.

Курс знакомит студентов с назначением и принципом действия современных персональных компьютеров, основами алгоритмизации и технологии программирования научно-технических задач, языками программирования высокого уровня, технологии обработки и отладки программ, современным программным обеспечением, методами решения типовых инженерных задач и их программной реализацией.

В результате изучения дисциплины студент должен:

Знать: технологию работы на ПК в современных операционных средах, основные методы разработки алгоритмов и программ, структуры данных, используемые для представления типовых информационных объектов, типовые алгоритмы обработки данных.

Уметь: использовать стандартные пакеты прикладных программ для решения практических задач.

Владеть: методами построения современных проблемно-ориентированных прикладных программных средств.

ВВЕДЕНИЕ

Система компьютерной математики **MATLAB (Matrix Laboratory)** переводится с английского как «Матричная лаборатория». Она является одним из эффективнейших средств выполнения научных и инженерных расчетов, их визуализации, обработки результатов эксперимента, анализа и моделирования. Для формулировки и решения задач в среде **MATLAB** используются понятные математические выражения, близкие к традиционным формулам, связывающие векторные или матричные объекты. Система включает ядро, использующее базовые вычислительные (встроенные) функции, и набор общематематических, графических и проблемно – ориентированных пакетов **(Toolboxes)**, позволяющих изучать и применять современную вычислительную технологию в таких областях, как обработка сигналов и изображений, моделирование, системы управления, системы связи, нейронные сети и многие другие.

Эта книга предназначена для освоения основных навыков по самостоятельному выполнению расчетов на компьютере в командном (диалоговом) режиме, возможно, после консультации с преподавателем. В ней на примерах, имеющих в основном математическую направленность, рассматриваются те элементы, команды и операторы, которые в дальнейшем применяются для решения задач с различной проблемной ориентацией. Это задачи высшей математики, теории электрических цепей, теории электро – связи и других дисциплин, изучаемых в технических ВУЗах. Материал представлен сжато. Предполагается, что он усваивается в непосредственном контакте с системой, когда численные и графические результаты расчетов, представленные в примерах, будут получены пользователем прямо на экране.

Предполагается также, что пользователь знаком с каким-либо языком программирования, например **Pascal** или **Basic**, с

приемами, общими для **Windows** – приложений и математическими постановками рассматриваемых задач.

Для реализации всех рассматриваемых примеров на используемом компьютере должны быть установлены следующие компоненты системы: ядро **MATLAB**, пакет **Symbolic Math Toolbox**.

ГЛАВА 1 ЗНАКОМСТВО С MATLAB И ПРОСТЕЙШИЕ ВЫЧИСЛЕНИЯ

Данная глава посвящена описанию рабочей среды **MATLAB** и выполнению элементарных операций с числами и массивами в командном режиме (режиме калькулятора). В этой главе рассматриваются особенности задания в **MATLAB** действительных и комплексных чисел, форматы их представления, правила использования переменных, а также способы задания векторов и матриц. Здесь также рассмотрены встроенные математические функции и способы сохранения переменных для их использования в последующих сеансах работы в **MATLAB**.

1.1 Рабочая среда MATLAB

При работе в **Windows** запустить систему **MATLAB** можно из меню **Пуск** (стартового меню) этой операционной системы. Альтернативным вариантом запуска **MATLAB** является двойной щелчок на ярлыке системы **MATLAB**, расположенном на рабочем столе **Windows.**

Запуск **MATLAB 6.x и выше версии** отображает на экране окно рабочей среды, подобное показанному на рис.1.1.

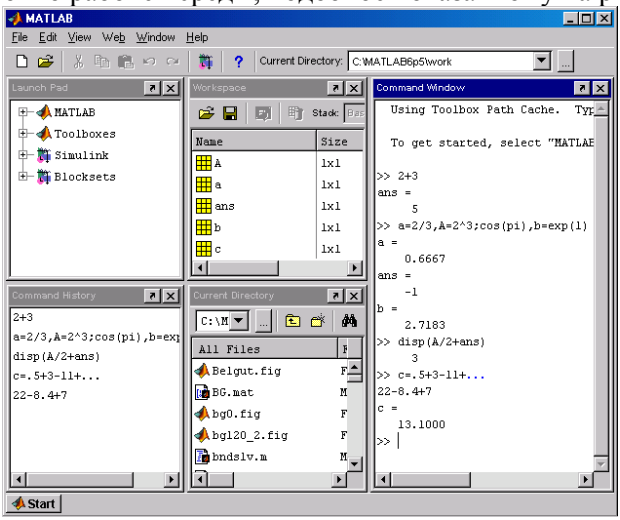


Рис 1.1

Окно рабочей среды (графический интерфейс) состоит из следующих основных элементов:

- строка меню;
- панель инструментов с кнопками и раскрывающимся списком;
- окно с **Launch Pad** (Панель запуска) содержит дерево файловой системы, где отображены только установленные на компьютере разделы расширений системы **MATLAB**. С помощью этого окна можно запустить любой из них;
- окно **Workspase** (Рабочее пространство), из которого можно получить простой доступ к переменным, используемым в данном сеансе работы;
- окно **Command History** (История команд), предназначенное для просмотра и повторного вызова ранее введенных команд;
- окно **Current Directory** (Текущий каталог), в котором отображается список файлов и вложенных папок активного в данный момент каталога;
- окно **Command Window** (Окно команд) предназначено для ввода чисел, переменных, выражений и команд, для промотра результатов вычислений, для отображения текстов выполняемых программ, а также для вывода сообщений об ошибках;
- строка состояния, где отображаются сообщения системы.

Пользователь может настроить окно рабочей среды по своему усмотрению. Можно, например, изменить местоположение и размер внутренних окон приемами, общими для **Windows** – приложений. Отобразить или скрыть соответствующие окна можно с помощью команд меню **View** (Вид) рабочей среды. Например, для отображения полной рабочей среды (рис.1.1) надо набрать команду **View => Desktop Layout => Five Panel**. Любое из внутренних окон полной рабочей среды можно закрыть щелчком по кнопке с крестиком в правом верхнем углу.

1.2. Арифметические вычисления

Работа в среде **MATLAB** может осуществляться либо в *программном* режиме (см. Главу 3), либо в *командном* режиме (режиме *калькулятора*, *диалоговом* режиме) по правилу «задал вопрос, получил ответ». Это превращает **MATLAB** в необычайно мощный калькулятор, который способен производить не только обычные для калькулятора вычисления, но и операции с векторами и матрицами, комплексными числами, рядами и полиномами. Можно почти мгновенно задать и вывести графики различных функций – от простой синусоиды до сложной трехмерной фигуры.

Основным элементом командного режима работы с системой является главное или командное окно **Command Window** . Оно активизируется командой **View => Desktop Layout => Command Window Only** рабочей среды. Структура командного окна аналогична структуре **Windows** – приложений (рис 1.2).

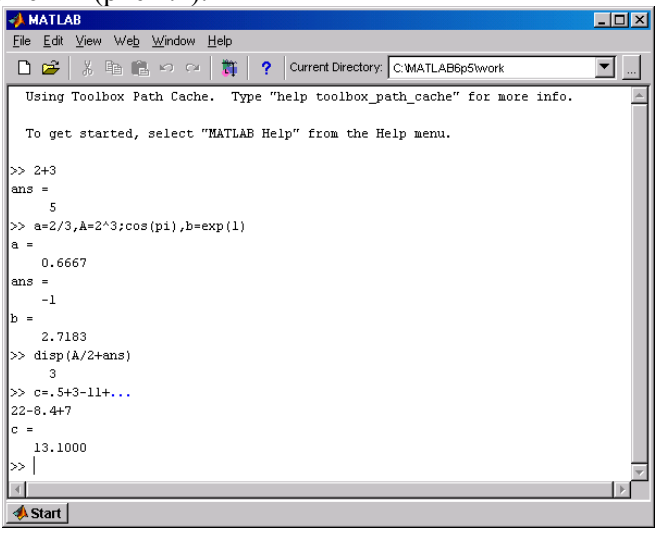


Рис 1.2

Строка в текстовом поле командного окна, отмеченная символом приглашения >> с мигающим курсором , называется *строкой ввода* или *командной строкой*. Она предназначена для ввода с клавиатуры команд, чисел, имен переменных и

знаков операций, составляющих выражение. Для того, чтобы система **MATLAB** выполнила введенную команду или вычислила заданное выражение, следует нажать клавишу **<Enter>** (Ввод).

При вводе курсор может находиться в любом месте командной строки. Введенные выражения вычисляются, а результаты вычислений и выполнения команд появляются в одной или нескольких строках командного окна – строках вывода.

В результате многократных вычислений (нажатий клавиши **<Enter>**) в командном окне автоматически производится *вертикальная протяжка (scrolling)*: строки сдвигаются на одну позицию вверх, а внизу появляется строка ввода с символом приглашения >>. Информация, которая покинула видимую часть окна, не исчезает. В **MATLAB** ранее введенные строки команд представляют собой «историю» и запоминаются в *стеке команд*.

Для просмотра выполненных команд и результатов вычислений, не умещающихся на экране, имеются полосы горизонтальной и вертикальной протяжки. Использование полос протяжки ничем не отличается от других **Windows** – приложений. Можно также осуществлять протяжку командного окна с помощью клавиш **<PageUp>, <PageDown>, <Ctrl+Home>** и **<Ctrl+End>.**

Клавиши **<↑>** и **<↓>**, которые в текстовых редакторах служат для перемещения вверх или вниз по экрану, в **MATLAB** работают иначе. Они используются для возврата в строку ввода ранее выполненных команд с целью их повторного выполнения или редактирования. После первого нажатия клавиши <↑> в строке ввода отобразится последняя введенная команда, при втором нажатии – предпоследняя и т.д. Клавиша <↓> осуществляет прокрутку команд в противоположном направлении.

Иными словами, текстовое поле окна **Command Window** располагается в двух принципиально разных зонах: *зоне просмотра* и *зоне редактирования*. Зона редактирования

находится в командной строке, а вся остальная информация видимой части командного окна – в зоне просмотра.

Пока не нажата клавиша **<Enter>**, вводимое выражение может быть отредактировано или удалено. В зоне просмотра уже ничего нельзя исправить. Если поместить в нее курсор и нажать какую – либо клавишу на клавиатуре, курсор будет автоматически перемещен в строку ввода, расположенную в зоне редактирования В то же время, с помощью клавиш <←> и <→> можно перемещать курсор в командной строке.

Невозможность редактирования ранее введенной команды простой установкой курсора в нужную строку является одной из особенностей системы **MATLAB**.

Сеанс работы с системой **MATLAB** называется *сессией*. Иными словами, сессия – это все то, что отображается в командном окне в процессе работы с системой. Команды сессии автоматически образуют список, который выводится в окне **Command History** , а значения переменных сохраняются в окне **Workspase** (рис.1.1).

Сессия на рис.1.2 содержит четыре команды и результаты их выполнения. Рассмотрим каждую из этих команд и установим некоторые *особенности вычислений в системе* **MATLAB**:

● >> 2+3

ans =

5

Результату выполненной операции не было присвоено имя, поэтому при выводе он был автоматически обозначен символом **ans** (**ansver**). Под этим именем результат вычислений хранится в памяти компьютера и его можно использовать в последующих вычислениях, до тех пор, пока в ходе работы не будет получен новый непоименованный результат. Результат вычислений выводится в строках вывода (без знака >>);

● >> a=2/3,A=2^3;cos(pi),b=exp(1)

a =

0.6667

ans =

```
    -1
b =
    2.7183
```

В одной строке могут быть введены несколько выражений и/или команд. В этом случае они разделяются либо запятыми, либо точками с запятой. Результат вычисления выражения или исполнения команды, за которыми следует запятая, выводится на экран. Результат вычисления выражения, за которым следует символ <**;**>, на экран не выводится, но он сохраняется в памяти и может быть использован в последующих вычислениях.

Знаком присваивания является знак =, а не комбинированный знак **:=**, принятый, например, в языке программирования **Pascal** или в системе символьной математики **Maple.**

В результате выполнения этой команды вычислены и сохранены в памяти переменные $a=2/3=0,6667$, $A=2^3=8$, $ans=\cos\pi=-1$, $b=e^1=e=2,7183$ (*e* – основание натурального логарифма), но переменная *A* не была выведена на экран, так как за ней следует символ <**;**>. При вычислении $\cos\pi$ использовалась системная переменная *pi* – число π. Число *e* системной переменной не является, и для его вычисления использована встроенная элементарная функция **exp** (экспонента). Функции записываются строчными буквами и их аргументы указываются в круглых скобках. Аргумент тригонометрической функции **cos** задан в радианах;

● >> disp(A/2+ans)

```
    3
```

Функция **disp** (от слова «дисплей») выводит в командное окно результат вычисления выражения $2^3/2+\cos\pi$, заключенного в скобки, без сохранения этого результата в памяти. При вычислениях с помощью функции **disp** в командном окне не отображается строка вывода с указанием имени переменной или имени **ans**, как это происходит при обычных вычислениях:

```
>> A/2+ans
ans =
```

3

Применение функции **disp** полезно для создания наглядных документов, у которых предотвращается вывод строк, не несущих полезной информации;

● >> c=.5+3-11+...

22-8.4+7

c =

13.1000

Для ввода длинных формул или команд в командную строку следует поставить три точки (подряд, без пробелов), нажать клавишу **<Enter>** и продолжить набор формулы на следующей строке. Так можно распространить командную строку на несколько физических строк. **MATLAB** вычислит все выражение или выполнит команду после нажатия на клавишу **<Enter>** в последней строке (в которой нет трех идущих подряд точек). Этот прием полезен для создания наглядных документов, у которых предотвращается выход строк в невидимую область окна.

Сессия на рис.1.2 содержит только правильные команды и результаты их выполнения. В общем случае, сессия является результатом проб и ошибок. Ее текст, наряду с правильными определениями, содержит сообщения и предупреждения об ошибках (см. раздел 1.8), переопределения функций и переменных, использованную справочную информацию команды **help** (см. Приложение.1, рис.П1, рис.П.2). Если сессия сильно «засорена» лишней информацией, диалог пользователя с системой затрудняется.

Команда очистки экрана **clc**

>> clc

стирает содержание командного окна **MATLAB** и размещает символ приглашения >> в левом верхнем углу пустого экрана.

Эта команда, однако, оставляет неизменным содержимое окон **Command History** и **Workspase**. Поэтому в «чистом» командном окне можно пользоваться значениями переменных, полученных до ввода команды **clc.**

Если же появится необходимость отредактировать или повторить ранее выполненную команду, то это легко осуществить с помощью окна **Command History**. Подробнее работа с оконами **Command History** и **Workspase** рассматривается в разделах 1.7 и 1.9 соответственно.

Переменные – это именованные объекты, хранящие какие – либо данные.

Переменные могут быть числовыми, матричными или символьными, что зависит от типа хранящихся в них данных. Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваиваивается переменной, т.е. пользователь не должен заботиться о том, какие значения будет принимать переменная (комплексные, вещественные или целые).

Имя переменной (ее *идентификатор*) может содержать до *31* символа и не должно совпадать с именами других переменных, функций, команд и системных переменных **MATLAB**. Имя переменной должно начинаться с буквы, может содержать цифры и символ подчеркивания. **MATLAB** чувствителен к регистру букв (переменные *a* и *A* не идентичны).

В **MATLAB** существует несколько имен переменных, являющихся зарезервированными. Переменные с такими именами называются *системными*. Они задаются после загрузки системы и могут использоваться в арифметических выражениях. Системные переменные могут быть переопределены, т.е. при небходимости им можно присвоить другие значения.

Ниже перечислены основные системные переменные **MATLAB**.

- **ans** – результат вычисления последнего не сохраненного пользователем выражения.
- **i**, **j** – мнимая единица ($\sqrt{-1}$), используемая для задания мнимой части комплексных чисел.
- **Inf** (infinity) – обозначение машинной бесконечности.

● **NaN** – сокращение от слов Not-a-Number (не число), принятое для обозначения неопределенного результата (например, 0/0 или Inf/Inf).

● **pi** – число π *(π=3,141592653589793)*.

● **eps** – погрешность операций над числами с плавающей точкой, т.е интервал между числом *1.0* и следующим ближайшим числом с плавающей точкой равен *2.2204e-16* или 2^{-52}.

● **realmin** – минимальное по модулю вещественное число (*2.2251e-308* или 2^{-1022}).

● **realmax** – наибольшее по модулю вещественное число (*1.7977e+308* или 2^{1023}).

Приоритеты *арифметических операций* системы **MATLAB** в порядке убывания следующие:

1. Возведение в степень <^>.

2. Умножение <*> и деление (слева направо </>, справа налево <\>).

3. Сложение <+> и вычитание <–>.

Выполнение операций одинакового приоритета происходит в порядке слева направо. Для изменения порядка выполнения арифметических операторов следует использовать круглые скобки. Кроме арифметических операторов, в **MATLAB** имеются операторы отношения и логические операторы (см. раздел 4.1).

Полный список операторов и справочную информацию по любому из них можно получить в разделе **ops** справочной системы **MATLAB**, используя команды **help** или **doc** (см. Приложение 1).

Основу большинства расчетов составляют вычисления значений *арифметических выражений*. В них в качестве операндов могут выступать константы, переменные, стандартные и нестандартные функции. В отличие от большинства алгоритмических языков, в **MATLAB** допускается использование операндов–массивов (см. раздел 1.8). В этом случае, результатом вычисления выражения также может быть массив.

Выражения, помещенные между двумя апострофами (заключенные в символьные скобки ' '), рассматриваются как символьные и не вычисляются, даже если в них содержатся математические выражения. Чаще всего они применяются для задания параметров функций и их нечисловых значений, вставки текста в графические объекты, а также для описания символьных переменных и выражений. Так, ввод символьного выражения *'2+3'* приводит к результату:

```
>> '2+3'
ans =
2+3
```

а не *5*.

При выводе графиков символы, помещенные между апострофами, определяют цвет линий графика, их тип и тип маркера, которым метятся линии (см. Главу 5).

1.3 Вещественные числа

Число – простейший объект системы **MATLAB**, представляющий количественные данные. Основным типом данных, с которым производятся вычисления в системе **MATLAB**, являются десятичные числа. Они приближают с заданной точностью произвольные *вещественные* (*действительные*) числа. Десятичные числа, используемые в **MATLAB**, могут быть целыми и дробными.

0
1
-93
7.5674
0.00000047.

Возможно представление чисел в *экспоненциальной форме* с указанием мантиссы и порядка числа.

3.3333e-4
-75.8e13.

Для отделения порядка числа от мантиссы применяется символ *e*, т.е. запись *3.3333e-04* соответствует записи 3.3333×10^{-4} или *0.00033333*.

Ввод чисел возможен в любом удобном для пользователя виде. Например, проще набрать *10e8* или *1.0e9*, чем *1 000 000 000*, а результат будет тот же самый. Пробел между цифрами и символом *e* при вводе не допускается, так это приводит к сообщению об ошибке:

```
>> 10 e8
??? 10 e8
Error: Missing operator, comma, or semicolon.
```

В качестве разделителя целой и дробной частей числа используется точка, а не запятая.

При вводе числа с нулевой целой частью ноль может опускаться: .5 равнозначно 0. 5.

Хотя задавать вещественные числа можно в любой из указанных выше форм, на машинном уровне системы **MATLAB** они представляются в форме с мантиссой и показателем степени. Этот основной тип данных называется **double** (формат *с двойной точностью*). Он задается по умолчанию, и даже целые числа представляются системой **MATLAB** на машинном уровне в той же форме, что и дробные числа.

Под мантиссу и показатель степени (на машинном уровне используется двоичная система записи) отводится *8* байт памяти. В результате для десятичных чисел достигается точность порядка *15* значащих цифр. Они принимает по модулю значения от *2.2250738507201e-308* до *1.797693134862316e+308*. Для этих чисел зарезервированы имена **realmin** и **realmax**.

Все операции над числами **MATLAB** выполняет по умолчанию в формате **double**. Такой формат удовлетворяет подавляющему большинству требований к численным расчетам, но совершенно не подходит для символьных вычислений с произвольной (абсолютной) точностью (см. Гл.7).

В **MATLAB** также существует тип данных **single**, который снижает требования к памяти в два раза (под мантиссу и показатель степени отводится 4 байта). Но при

сложных вычислениях в этом случае возрастает вероятность получить результат с большой погрешностью.

Существуют и целые типы данных: **int8, uint8, int16, uint16, int32, uint32**, **int64, uint64.,** под которые отводится *8, 16, 32* и *64* байта соответственно. Буква **u** соответствует беззнаковым типам данных с диапазоном от *0* до некоторого максимального положительного значения.

Для того, чтобы переменная получила тип данных, отличный от **double**, применяется явный квалификатор , совпадающий с названием типа.

Например,

>> x=int32(3.2)

определяет переменную целого типа **int32.**

Подробную информацию о перечисленных типах данных можно получить по справкам **doc double**, **doc single**, **doc int8**, **doc uint8**.

1.4 Форматы вывода результата вычислений

Формат числа определяет вид результата вычислений в командном окне. Чтобы не перегружать подробностями командное окно, **MATLAB** по умолчанию использует формат **short** (укороченный), при котором на экране отображаются только четыре цифры после десятичной точки (рис 1.2). Однако формат вывода может быть и другим. Он может быть установлен с помощью изменения параметров **Command Window.**

Выберем в меню **File** пункт **Preferences** (Предпочтения). На экране появится диалоговое окно **Preferences**, изображенное на рис. 1.3.

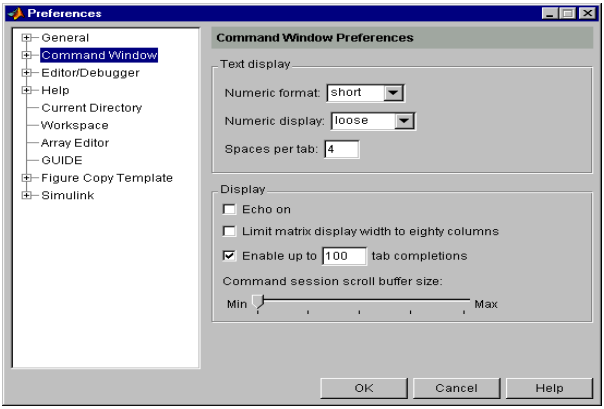


Рис 1.3

Для установки формата вывода следует убедиться, что в списке левой панели выбран пункт **Command Window** (как показано на рис.1.3). Задание формата производится из раскрывающегося списка **Numeric format** панели **Text display**.

Выберем **short** в раскрывающемся списке **Numeric format**. Закроем диалоговое окно, нажав кнопку **ОК**. Сейчас установлен короткий формат с плавающей точкой для вывода, при котором на экране отображаются только четыре цифры после десятичной точки. Наберем в командной строке *200/3* и нажмем **<Enter>.** Результат выводится в формате **short**:

```
>> 200/3
ans =
   66.6667
```

Этот формат вывода сохранится для всех последующих вычислений, если только не будет установлен другой формат. В **MATLAB** возможна ситуация, когда при отображении слишком большого или малого числа результат не укладывается в формат **short**. При вычислении *100000/3* и *1/3000* результаты выводятся в экспоненциальной форме:

```
>> 100000/3
ans =
  3.3333e+004
>> 1/3000
ans =
```

3.3333e-004

Однако, первоначальная установка формата **short** сохраняется при дальнейших вычислениях.

Если требуется получить результат вычислений более точно, то в раскрывающемся списке диалогового окна **Preferences** следует выбрать **long**. При этом результат будет отображаться в длинном формате с плавающей точкой **long** с четырнадцатью цифрами после десятичной точки. Форматы **short e** и **long e** предназначены для вывода результата в экспоненциальной форме с четырьмя и четырнадцатью цифрами мантиссы после десятичной точки соответственно. Информацию о форматах можно получить, набрав в командной строке команду **doc format.**

Задать формат вывода можно непосредственно из командной строки при помощи команды **format**. Например:

```
>> format long e
>> 1.33/5.13
ans =
   2.592592592592593e-001
>> format long g
>> ans
ans =
       0.259259259259259
>> format rat
>> 0.03333333
ans =
    1/30
```

Отметим, что **format rat** устанавливает формат вывода, при котором вещественные числа приближённо представляются отношением двух небольших целых чисел.

Для того, чтобы выделить результат, **MATLAB** выводит его через строку после вычисляемого выражения. Иногда требуется разместить больше строк на экране. Для этого в диалоговом окне **Preferences** следует выбрать **compact** из раскрывающегося списка **Numeric display**. В приводившихся ранее примерах пустой строки нет.

Добавление пустых строк обеспечивается выбором **loose** из списка **Numeric display**.

Применение команды **format** без параметров восстанавливает используемое по умолчанию состояние **format short** и **format loose**.

Все промежуточные вычисления **MATLAB** *производит с* ***двойной точностью****, независимо от того, какой формат вывода установлен.*

1.5 Комплексные числа и комплексные функции

Комплексными называются числа вида $z=x+iy$, где x и y – действительные числа, а $i=\sqrt{-1}$. Числа x и y называются соответственно *действительной* и *мнимой* частями комплексного числа z. Они обозначаются $x=Re\ z$, $y=Im\ z$. Комплексные числа в системе MATLAB записываются в следующем виде:

3+2i; 7-4j; -3.8952+1.23e-5i; 5+i*7.

По умолчанию они имеют тип **double**. Для записи комплексного числа требуется в два раза больше памяти, чем для записи вещественного числа, так как по *8* байт памяти отводится для *Re z* и *Im z*. При вводе комплексных чисел мнимая единица ***i*** или ***j*** может быть записана до или после мнимой части. При записи мнимой единицы перед *Im z* между ними ставится знак умножения **<*>**. При записи мнимой единицы после *Im z* ставить знак умножения необязательно. Например, следующие записи эквивалентны:

*1+i*2~1+j*2~1+2*i~1+2*j~1+2i~1+2j*.

Если коэффициентом при мнимой единице является не число, а переменная, нельзя писать просто *x+yi*, а необходимо использовать знак умножения, т.е. *x+y*i*.

Если в командную строку ввести ***i***, получим

```
>> i
ans =
      0 + 1.0000i
```

Тот же результат получим при вводе буквы ***j***:

```
>> j
ans =
```

```
    0 + 1.0000i
```

Кроме того, комплексное число можно представить в другом формате:

```
>> format long
>> 4-9j
ans =
  4.00000000000000 - 9.00000000000000i
```

При выводе мнимая единица всегда обозначается буквой ***i*** и выводится после мнимой части.

Число $\overline{z} = x - iy$ называется к*омплексно-сопряженным* числу $z=x+iy$. Два комплексных числа считаются *равными*, если равны отдельно их действительные и мнимые части. Алгебраические действия над комплесными числами выполняются по формулам:

$$z_1 \pm z_2=(x_1+iy_1) \pm (x_2+iy_2)=(x_1 \pm x_2)+i(y_1 \pm y_2);$$

$$z_1 z_2=(x_1+iy_1)(x_2+iy_2)=(x_1x_2 - y_1y_2)+i(x_1y_2 +x_2y_1);$$

$$\frac{z_1}{z_2} = \frac{z_1 \overline{z_2}}{z_2 \overline{z_2}}.$$

Примеры:

```
>> 1+2i+3-4j
ans =
  4.0000 - 2.0000i
>> (1+2i)*(3-4j)
ans =
  11.0000 + 2.0000i
>> (1+2i)/(3-4j)
ans =
  -0.2000 + 0.4000i
>> z=(3+2i)^3
z =
  -9.0000 +46.0000i
```

Функции **real** и **imag** выделяют вещественную и мнимую части комплексного значения:

```
>> real(z)
ans =
```

```
   -9
>> imag(z)
ans =
   46
```

Функция **complex** формирует комплексное число по паре вещественных:

```
>> z=complex(3,-4)
z =
  3.0000 - 4.0000i
```

Функция **conj** возвращает *комплексно–сопряженное* число:

```
>> conj(z)
ans =
  3.0000 + 4.0000i
```

Такой же результат получим, поставив апостроф после комплексного значения:

```
>> z=1+2*i'
z =
  1.0000 - 2.0000i
>> z'
ans =
  1.0000 + 2.0000i
```

Возникающий в процессе вычислений с вещественными переменными комплексный результат не является ошибкой. Вычислить $\sqrt{-1}$, оставаясь в рамках только вещественных чисел, нельзя. **MATLAB** автоматически перейдет к комплексным вычислениям и в итоге возвратит результат, равный ***i***:

```
>> sqrt(-1)
ans =
      0 + 1.0000i
```

В математике используют и другие формы представления комплексных чисел:

$z=x+iy=\rho e^{i\varphi}=\rho(\cos\varphi+i\sin\varphi)$.

Здесь $\rho = |z| = \sqrt{x^2+y^2}$ – модуль комплексного числа, а $\varphi = arg\ z$ – фаза, или главное значение его аргумента (величина угла измеряется в радианах, $\pi < \varphi \leq \pi$), причем $tg\ \varphi = \frac{y}{x}$.

Значения этих параметров можно определить с помощью стандартных функций ρ=**abs(z)** и $\varphi = arg\ z$ = **angle(z)**.

Основные элементарные функции комплексного переменного:

показательная функция e^{iz};

тригонометрические функции

$$\mathbf{\cos z = \frac{e^{iz} + e^{-iz}}{2}, \sin z = \frac{e^{iz} - e^{-iz}}{2i};}$$

гиперболические функции

$$\mathbf{ch\ z = \frac{e^{z} + e^{-z}}{2}, sh\ z = \frac{e^{z} - e^{-z}}{2};}$$

главное значение логарифма

$ln\ z = ln\ |z| + iarg\ z$;

главные значения обобщенных показательной и степенной функций

$a^z = e^{z ln\ a}$ и $z^\alpha = e^{\alpha ln\ z}$,

z, α, a – любые комплексные числа, $a \neq 0$.

Примеры:

```
>> exp(1+i)
ans =
  1.4687 + 2.2874i
>> sin(1+j)
ans =
  1.2985 + 0.6350i
>> i^i
ans =
   0.2079
>> isreal(i^i)
ans =
    1
>> (2+i)^(1-3i)
ans =
```

```
  -3.3307 - 8.3459i
>> log(-1)
ans =
        0 + 3.1416i
```

1.6 Использование элементарных функций

Встроенные элементарные функции (например, **sin**) записываются строчными буквами, их аргументы указываются в *круглых скобках.* Имена наиболее употребительных элементарных функций, аргументы которых могут быть и *комплексными*, сведены в таблицу 1.1.

Таблица 1.1. Основные элементарные функции

exp	экспонента	**abs**	модуль
log	натуральный логарифм	**asin**	арксинус
log10	десятичный логарифм	**acos**	арккосинус
sqrt	квадратный корень	**atan**	арктангенс
sin	синус	**sinh**	гиперболический синус
cos	косинус	**cosh**	гиперболический косинус
tan	тангенс	**tanh**	гиперболический тангенс
cot	котангенс	**asinh**	гиперболический арксинус
sec	секанс	**acosh**	гиперболический арккосинус
csc	косеканс	**atanh**	гиперболический арктангенс

Аргументы перечисленных тригонометрических функций задаются в радианах. Обратные к ним функции возвращают результат также в радианах.

Упомянем еще функции, связанные с целочисленной арифметикой. Например, функции округления: **round** (округление до ближайшего целого), **fix** (усечение дробной части числа), **floor** (округление до меньшего целого), **ceil** (округление до большего целого).

Кроме того, есть еще функции **mod** (остаток от деления с учетом знака), **rem** (остаток в смысле модульной арифметики), **sign** (знак числа), **factor** (разложение числа на простые множители), **isprime** (истинно, если число простое), **primes** (формирование списка простых чисел), **rat** (приближение числа в виде отношения двух небольших целых чисел), **lcm** (наименьшее общее кратное), **gcd** (наибольший общий делитель). В этих случаях о комплексных аргументах не может быть и речи.

И, наконец, есть функции, предназначенные для решения стандартных задачи комбинаторики: функция **perms** вычисляет число перестановок, а функция **nchoosek** – число сочетаний. Например, C_{12}^{4} – число сочетаний из *12* по *4*, легко находится находится вызовом функции **nchoosek**

```
>> nchoosek(12,4)
ans =
  495
```

Способы получения справочной информации по встроенным элементарным функциям (раздел **elfun** справочной системы **MATLAB**) подробно рассмотрены в Приложении 1 (рис. П.1 – П.3).

В **MATLAB** имеются встроенные специальные математические функции. Например, функции Бесселя **besselj, bessely**, полиномы Лежандра **legendre** и др. (см. раздел **specfun** справочной системы **MATLAB**).

В **MATLAB 7** появились тригонометрические функции, аргументы которых можно задавать в градусах. Обратные к ним функции возвращают результат также в

градусах. Имена таких функций заканчивются буквой **d** (**sind, cosd, acosd, …**).

1.7 Сообщения об ошибках и их исправление

При наличии ошибок в выражениях или командах **MATLAB** указывает на наличие ошибки. Из текста иногда можно понять сущность ошибки, но часто комментарии бывают настолько общими, что трудно установить место и содержание ошибки. Надо отличать *предупреждение* об ошибке от *сообщения* о ней. *Предупреждения* (обычно после слова **Warning**) не останавливают вычисления и лишь предупреждают о том, что ответ может быть ошибочным. Например:

```
>> sin(0)/0
Warning: Divide by zero.
ans =
   NaN
```

При *сообщение об ошибке* красного цвета (после знаков ???) **MATLAB** не выдает решение.

Вычислим, например, значение $\sqrt{2}$ с помощью встроенной элементарной функции **sqrt** (квадратный корень), введя выражение

```
>> sqr(2)
??? Undefined function or variable 'sqr'.
```

Это сообщение говорит о том (на английском языке), что не определена переменная или функция, и указывает, какая именно – **sqr.**

Устранение ошибки наиболее целесообразно не путем набора нового правильного выражения, а редактированием ошибочного.

Существует несколько способов возврата в строку ввода ранее введенных команд.

Первый способ – с помощью клавиш **<↑>** и **<↓>.** Он рассматривался в разделе 1.2

При вычислении значения $\sqrt{2}$ обнаружена *синтаксическая ошибка* – не определена функция **sqr.** Клавишей <↑> вернем команду

```
>> sqr(2)
```

в командную строку. Отредактируем ее – после **sqr** вставим **t**, и нажмем клавишу <**Enter>.**

```
>> sqrt(2)
ans =
   1.4142
```

Второй способ – копирование из содержимого текстового поля рабочего окна.

В текстовом поле можно выделить с помощью мыши любую команду и копировать ее в буфер обмена операционной системы **Windows**, а затем вставить в командную строку. Выделение и вставка производится теми же средствами, что и в других **Windows** – приложениях. (см. третий способ).

Третий способ – копирование из окна **Command History.**

Для активизации окна **Command History** необходимо войти в меню **View** командного окна, выбрать вкладку с одноименным названием и щелкнуть на ней левой кнопкой мыши (поставить галочку). В этом окне отображаются дата и время каждого сеанса работы в **MATLAB**, а также перечень команд, вводимых в течение каждого сеанса (рис.1.4).

Если в окне **Command History** дважды щелкнуть левой кнопкой мыши на какой – либо команде, эта команда будет выполнена. Это равнозначно вводу данной команды в командное окно и последующему нажатию клавиши <**Enter>** (рис.1.4).

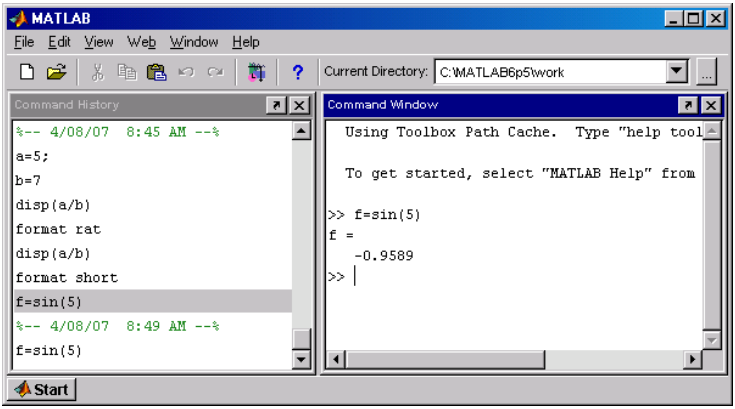


Рис.1.4

Если щелкнуть на какой – либо команде окна **Command History** левой кнопкой мыши, то данная команда становится текущей (на синем фоне). Можно выделить нужную последовательность команд при помощи комбинации клавиш **<Shift>+<↑>, <Shift>+<↓>**. При щелчке правой кнопкой мыши на выделенной области окна **Command History** появляется всплывающее меню. Выбор пункта **Copy** приводит к копированию выделенной последовательности в буфер обмена **Windows**. При щелчке правой кнопкой мыши на области окна **Command Window** появляется всплывающее меню. Выбор пункта **Paste** приводит к вставке скопированной последовательности команд в командную строку. Весь вставленный в командную строку набор команд отправляется на выполнение нажатием клавиши <**Enter**>.

До нажатия клавиши <**Enter**> содержимое набора можно редактировать, используя обычные приемы редактирования, общие для **Windows** – приложений, в том числе и с помощью мыши. Можно вносить в команды необходимые изменения, удалять лишние команды и добавлять новые. При редактировании клавиши <↑> и <↓> могут использоваться для перемещения между строками набора.

При вычислениях любое арифметическое выражение набирают с клавиатуры в командной строке. Если выражение

задано формулой, операндами которой являются степени, дроби, элементарные функции, то по записи этого выражения в одну строку трудно определить, правильно набрана формула, или нет. Редактор **MATLAB** в состоянии отыскать только синтаксические ошибки ввода. Но он не может обнаружить так называемые ***семантические ошибки***, когда, например, пользователь вместо одного оператора ошибочно применит другой, или неправильной расстановкой скобок изменит порядок выполнения операций и т.д.

MATLAB совместно с пакетом **ToolBox Symbolic Math** (см. Главу 7) дает возможность визуальной проверки соответствия набранного выражения исходной формуле.

Рассмотрим на примере, как можно обнаружить, а затем и устранить семантические ошибки. Пусть требуется вычислить значение выражения, заданного формулой

$$F = \frac{\ln x + \frac{1}{y}}{\sqrt{x} + tg\left(\frac{sin5}{1+y^2}\right)}$$

при $x=0,1$, $y=0,2$.

Введем значения переменных x и y. Наберем с клавиатуры арифметическое выражение F и вычислим его значение, нажав клавишу <**Enter**>. Редактор **MATLAB** синтаксических ошибок ввода не обнаружил. В результате вычислено значение выражения $F= 7,2111$.

Это результат выполненения двух первых команд на наглядном документе – сессии, отображенной на рис.1.5.

Для того, чтобы проверить выражение F на наличие семантических ошибок, ему необходимо придать статус символьного. Символьное выражение F создается с помощью функции **sym** (см. раздел 7.1), входным аргументом которой является строка с арифметическим выражением , заключенная в апострофы.

Третья командная строка получена редактированием предыдущей с целью придания арифметическому выражению F статуса символьного. Результат выполнения этой команды – символьное выражение F, выведенное в командное окно в

одну строку. Очевидно, что арифметическое и символьное выражения *F* синтаксически совпадают.

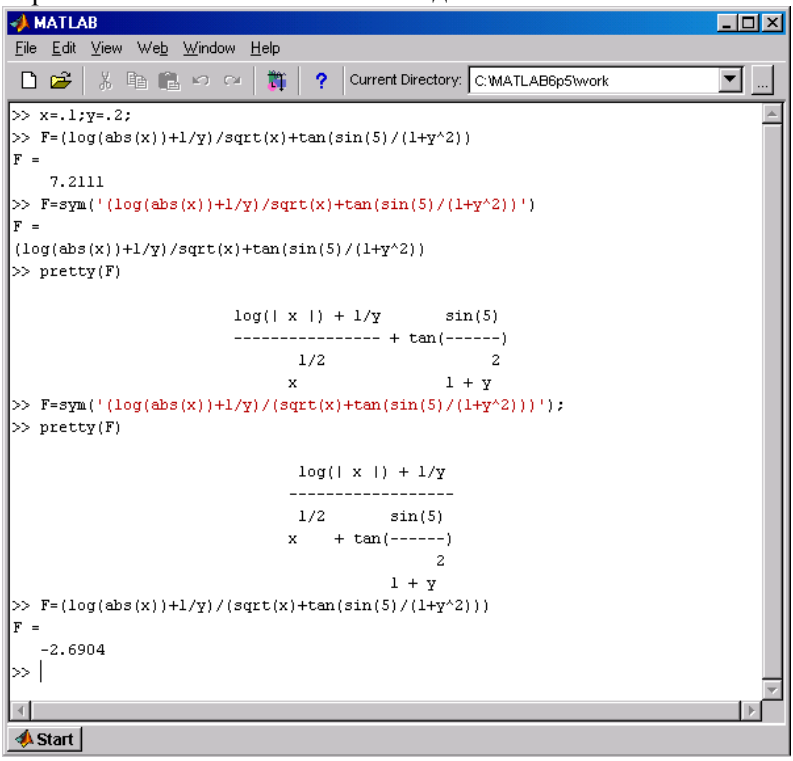


Рис.1.5

Функции **pretty**(см. раздел 7.1) выводит в командное окно символьное выражение *F* в виде, близком к математической формуле.

Сравнив формулу, выведенную в командное окно в результате выполнения четвертой команды и исходную формулу *F*, видим, что это разные формулы. Очевидно, что при вводе формулы *F* нарушен порядок выполнения операций. Предполагаемая семантическая ошибка – при вводе знаменатель дроби в исходной формуле *F* не заключен в скобки.

Пятая командная строка получена редактированием третьей командной строки с целью исправления ошибки ввода.

Результат выполнения пятой команды – новое символьное выражение *F* (без вывода в командное окно).

В шестой команде функции **pretty** выводит в командное окно новое символьное выражение *F* в формульном виде. Выведенная формула совпадает с исходной формулой *F*.

Седьмая командная строка получена редактированием пятой командной

строки с целью отмены для исправленного выражения *F* статуса символьного. Результат выполнения седьмой команды – правильное значение выражения *F= -2,6904*.

1.8 Ввод векторов и матриц

Фундаментальным принципом построения системы **MATLAB** является ее ориентация на операции с *массивами*. Массив – упорядоченная, пронумерованная совокупность однородных данных. Такими данными могут быть действительные и комплексные числа, переменные либо арифметические выражения. У массива должно быть имя. Массивы различаются по числу *размерностей* или *измерений*: одномерные, двумерные, многомерные. *Размером* массива называют число элементов вдоль каждого из измерений. Под *вектором* в **MATLAB** понимается одномерный массив данных, а под *матрицей* – двумерный или многомерный массив. Если элементы вектора представлены в виде столбца, такой вектор называется *вектором-столбцом*, а если в виде строки, вектор называется *вектором-строкой*.

Исходные значения векторов можно задавать с клавиатуры путем поэлементного ввода. Для этого в командной строке следует вначале указать имя вектора, потом поставить знак присваивания < = >, далее – открывающую *квадратную* скобку, а за ней ввести заданные значения элементов вектора, отделяя их пробелами или запятыми. Завершается строка закрывающей квадратной скобкой. Так, ввод

```
V=[1 2 3]
V =
    1    2    3
```

задает вектор-строку *V*, содержащую три элемента со значениями *1,2,3*.

Параллельное использование запятых и пробелов допустимо даже в пределах ввода одного вектора: записи *V=[1,2,3]* и *V=[1 2,3]* эквивалентны предыдущей.

Оператор **<:>** *(двоеточие)* дает возможность простого создания векторов, каждый элемент которых отличается от предшествующего на постоянную величину, т.е. шаг. Например:

```
>> V=-0.1:0.3:1.4
V =
   -0.1000    0.2000    0.5000    0.8000    1.1000    1.4000
```

При автоматическом заполнении шаг, равный единице, допускается не указывать. Шаг может быть и отрицательным.

Возможен ввод векторов в виде арифметических выражений, содержащих любые встроенные функции. Например:

```
>> V=[2+2/(3+4),exp(5),sqrt(10)]
V =
   2.2857  148.4132    3.1623
```

Вектор-столбец *A* вводится аналогично, но значения элементов в перечне отделяются знаком **<;>**:

```
>> A=[1.3;5.4;6.9]
A =
   1.3000
   5.4000
   6.9000
```

Вектор-столбец можно создать транспонированием вектор-строки (см. п.1.10).

Знак **<;>** также используется для разделения строк при задании матриц. Один из способов ввода матрицы основан на том, что матрицу можно рассматривать как вектор-столбец, каждый элемент которого является строкой матрицы. Поскольку точка с запятой используется для разделения элементов вектор-столбца, то комбинируя оба варианта разделителей, можно сформировать матрицу

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

```
>> B=[1 3 0;-2 -2 5]
B =
    1    3    0
   -2   -2    5
```

Матрицы небольших размеров удобно вводить из командной строки. Вначале ставится открывающая квадратная скобка. Затем элементы каждой строки матрицы набираются через пробел, а ввод строки завершается нажатием на клавишу **<Enter>**. При вводе последней строки в конце ставится закрывающая квадратная скобка. Если после закрывающей квадратной скобки не ставить точку с запятой для подавления вывода в командное окно, то матрица выведется в виде таблицы.

Так можно сформировать предыдущую матрицу *B*:

```
>> B=[1 3 0
-2 -2 5]
B =
    1    3    0
   -2   -2    5
```

Еще один способ формирования векторных и матричных массивов в окне рабочего пространства **Workspace** рассмотрен в разделе1.10.

Обращение к элементам вектора

Доступ к элементам вектора осуществляется при помощи *индекса (целое положительное число)*, заключаемого в круглые скобки после имени массива, в котором хранится вектор. Если введен массив *V*, определенный вектор-строкой

```
>> V=[3.1 4.5 7.1 2.2 0.8];
```

то для вывода, например, его третьего элемента, используется *индексация*:

```
>> V(3)
ans =

   7.1000
```

Если нужно, наоборот, вставить на это место некоторое число, например π, то это можно сделать так:

```
>> V(3)=pi
v =
   3.1000    4.5000    3.1416    2.2000    0.8000
```

Обращение к элементам матриц

Доступ к элементам матриц осуществляется при помощи *двух индексов* – номеров строки и столбца, заключенных в круглые скобки и расположенных после имени, в котором хранится массив данных, например:

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

Если в приведенной матрице на место элемента, обозначенного цифрой *7*, нужно вставить другое число, например *e*, то это можно сделать так:

```
>> M(2,3)=exp(1)
M =
   1.0000    2.0000    3.0000    4.0000
   5.0000    6.0000    2.7183    8.0000
```

Обратите внимание на то, что после операции присваивания нового значения хотя бы одному элементу матрицы, не завершенной точкой с запятой, матрица *M* целиком выводится в командное окно.

Присвоить можно и значение элементу матрицы, индексы которого превышают текущие ее размеры. В этом случае эти размеры будут соответственно увеличены, а остальные элементы такой расширенной матрицы будут заполнены нулями:

```
>> A=[1 2]
A =
    1     2
>> A(2,3)=5
A =
    1     2     0
```

```
    0     0     5
```

Элементы массивов могут входить в состав выражений.

```
>> A(1,1)/A(1,2)-A(2,3)
ans =
  -4.5000
```

Функция **size(B)** позволяет установить размер массива *B*, она возвращает результат в виде вектора, первый элемент которого равен числу строк, а второй – столбцов:

```
>> B=[1 2 3;4 5 6];V=[1 2 3];A=[1;2;3];a=5;
>> disp(size(B))
    2     3
>> disp(size(V))
    1     3
>> disp(size(A))
    3     1
>> disp(size(a))
    1     1
```

Итак, *B, V, A ,a* – массивы размеров *2×3, 1×3, 3×1, 1×1* соответственно.

Результатом применения функции **length(V)** к вектору *V* является длина вектора, т.е. число его элементов:

```
>> disp(length(V))
    3
```

Функция **ndims(B)** отображает размерность массива *B.*

```
>> disp(ndims(B))
    2
>> disp(ndims(a))
    2
>> disp(ndims(V))
    2
```

Итак, *B, a, V* – массивы размерности *2* установленных выше размеров.

В **MATLAB** можно создавать массивы размерности и выше *2*.

Пример:

```
>> p(1,1,2)=1
p(:,:,1) =
```

```
     0
p(:,:,2) =
     1
>> disp(size(p))
     1     1     2
>> disp(ndims(p))
     3
```

Итак, введен массив *p* размерности *3* размера *1×1×2* с двумя элементами *p(1,1,1) =0* и *p(1,1,2) =1*.

1.9 Просмотр и сохранение переменных

Можно вывести значение любой переменной *a* в командное окно, для чего следует набрать имя переменной в командной строке и нажать **<Enter>**, либо вызвать функцию **disp(a)**.

Просмотр текущих переменных рабочей среды производится при помощи команды **whos**. Предположим, что ранее переменным *a* , *b* и *c* были присвоены значения:

```
>> a= -1.34;
>> b=2.98+3.86i;
c=[1 2 3;-5 6 -1]
c =
     1     2     3
    -5     6    -1
```

Вызовем команду **whos**, указав через пробелы имена переменных

```
>> whos a b c
```

В командное окно выводится таблица, приведенная ниже.

Name	Size	Bytes	Class
a	1x1	8	double array
b	1x1	16	double array (complex)
c	2x3	48	double array

В столбике **Class** указан тип переменной, в **Bytes** – число байт, выделенных под хранение значения, а столбик **Size** содержит информацию о размере переменных. После таблицы размещена строка с указанием суммарного объема памяти в байтах.

MATLAB запоминает значения всех переменных, определенных во время сеанса работы, даже если применена команда очистки экрана **clc** (см. раздел1.2).

После ввода этой команды экран будет очищен. Для того, чтобы узнать, какие переменные уже использованы, а какие нет, служит команда **who**, выводящая в комадное окно список используемых переменных:

```
>> who
Your variables are:
a  b  c
```

Для удаления из памяти всех переменных используется команда **clear**. Если за ней указать список переменных (через пробел), то только они будут удалены из памяти, например:

```
>> clear b
>> who
Your variables are:
a  c
```

По завершении сеанса работы с системой **MATLAB** все использованные переменные теряются. Чтобы сохранить содержимое рабочего пространства в файле на диске компьютера, надо выполнить команду меню

File │ Save Workspace As…

После выбора раскроется диалоговое окно **Save to MAT – files(*.mat).** Введем имя файла, (например, **R.mat**) и щелкнем по кнопке **Save**. Теперь значения использованных переменных сохранены в одноименном файле текущего рабочего каталога.

Чтобы восстановить значения переменных в следующем сеансе работы, выполним команду

File │ Open…

Затем в диалоговом окне **Open** необходимо выбрать вкладку с названием **R.mat** и щелкнуть на ней дважды левой кнопкой мыши. Произойдет загрузка в текущее рабочее пространство ранее сохраненных переменных.

Другой способ: сохранение – **save R.mat**; восстановление – **load R.mat.**

Начиная с версии 6.0 в **MATLAB** появилось удобное средство для просмотра переменных рабочей среды – окно **Workspace,** которое активизируется с помощью команды **View => Workspace** меню командного окна (рис.1.6).

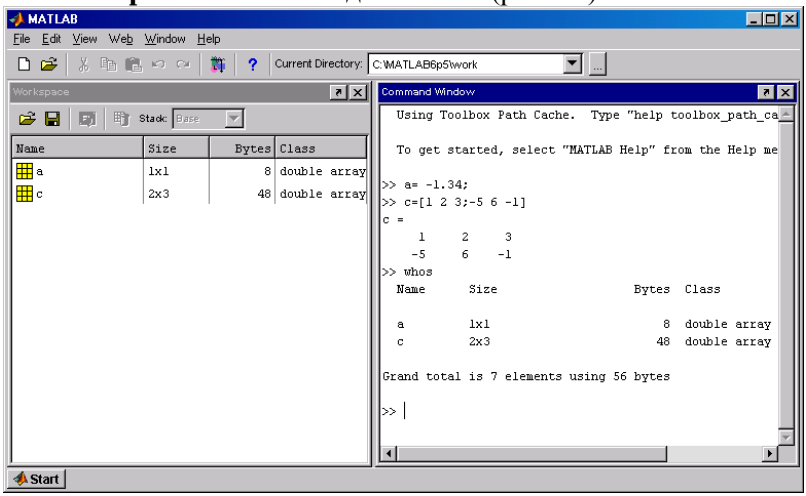


Рис.1.6

Окно **Workspace** содержит таблицу, аналогичную той, что выводится командой **whos.**

Двойной щелчок левой кнопкой мыши на строке с именем переменной в окне **Workspace** отображает в отдельном окне редактора массивов **Array Editor** ее матричное представление. На рис.1.7 показано отображение переменной *c* .

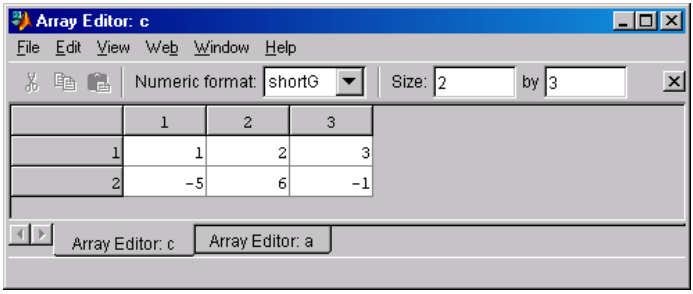


Рис.1.7

В окне **Array Editor** в соответствующих полях задается размер матрицы – число ее строк и столбцов – и далее вводятся числа, выражения, переменные и функции в отдельные клетки появившейся таблицы. Этот способ обеспечивает быстрый доступ к элементам матрицы и позволяет их вводить или редактировать в произвольном порядке.

При щелчке правой кнопкой мыши на строке с именем переменной в окне **Workspace** появляется всплывающий список команд. Команды списка позволяют переименовать переменные, удалить лишние, сохранить рабочую среду и т.д. Так, щелчок на строке с именем команды **Open** (Открыть) приводит к отображению этой переменной в окне **Array Editor**.

Один из способов формирования векторных и матричных массивов состоит в создании пустого массива

```
>> M=[]
M =
    []
```

размера 0×0, который затем заполняется с использованием редактора массивов **Array Editor**.

1.10 Матричные операции над векторами и матрицами

MATLAB позволяет выполнять мощные групповые вычисления над массивами, используя обычные математические функции, которые в традиционных языках программирования работают только со скалярными аргументами. Групповые операции позволяют компактно задавать выражения, при которых выпололняется гигантский объем работы. Кроме того, матричные вычисления выполняются намного быстрее, чем скалярные. Пример сравнения по быстродействию *матричного* и *скалярного* умножения двух матриц приведен в разделе 4.6.

К матричным действиям с матрицами относятся такие операции, которые используются в матричном исчислении в математике.

Базовые действия с матрицами (векторами) – *сложение, вычитание, транспонирование, умножение матрицы на число, умножение матрицы на матрицу, возведение квадратной матрицы в степень* – осуществляются в **MATLAB** с помощью обычных знаков арифметических операций. Условия, при которых эти операции возможны, таковы:

- при сложении или вычитании матриц они должны иметь одинаковые размеры;
- при умножении матриц число столбцов первого множителя должно совпадать с числом строк втрого множителя.

Невыполнение этих условий приводит к появлению сообщения об ошибке.

Приведем несколько примеров действий с матрицами

$$A=\begin{bmatrix} 0 & -2 & 4 \\ 3 & 2 & 1 \end{bmatrix}, D=\begin{bmatrix} -5 & 4 & 2 \\ 1 & 3 & 1 \end{bmatrix}, B=\begin{bmatrix} -1 & -2 & -3 \\ 1 & 3 & 1 \\ 0 & 2 & 2 \end{bmatrix}.$$

```
>> A=[0 -2 4;3 2 1];D=[-5 4 2;1 3 1];B=[-1 -2 -3;1 3 1;0 2
2];
```

Пример сложения и вычитания

```
>> disp(A+D)
   -5    2    6
    4    5    2
>> disp(D-A)
   -5    6   -2
   -2    1    0
```

Пример умножения на число

```
>> disp(3*D)
  -15   12    6
    3    9    3
```

Пример транспонирования матрицы, при котором ее строки становятся столбцами с теми же номерами, а столбцы – строками, осуществляется с помощью оператора апостроф <'>:

```
>> disp(A')
    0    3
   -2    2
```

4 1

В математике транспонированная матрица A обозначается A^T.

Знак **<*>** закреплен за *матричным умножением* векторов и матриц в *смысле линейной алгебры*. При этом число столбцов первой матрицы обязано равняться числу строк второй матрицы. Произведение прямоугольной матрицы $A_{n\times k}$ (таблицы чисел, расположенных в n строках и в k столбцах) на матрицу $B_{k\times m}$ определяется следующим образом: для того, чтобы получить элемент c_{ij} матрицы - произведения $C=AB$, следует элементы i -й строки матрицы A умножить на соответствующие элементы j -го столбца матрицы B и результаты сложить, т. е.

$$c_{ij} = \sum_{p=1}^{k} a_{ip} b_{pj}.$$

Матрица $C = C_{n\times m}$ занимает n строк и m столбцов.

Пример умножения матрицы на матрицу

```
>> C=A*B
C =
   -2    2    6
   -1    2   -5
```

Умножение двух векторов определено в математике только для векторов одинакового размера и лишь тогда, когда один из векторов сомножителей

является строкой, а второй – столбцом. Иначе говоря, если векторы X и Y являются строками, то математический смысл имеют только две формы умножения этих векторов: $U=X*Y'$ и $V=X'*Y$. Причем в первом случае результатом будет *скалярное произведение векторов X и Y* (число), а во втором – *внешнее произведение векторов X и Y* (квадратная матрица). Пример:

```
>> x=[1 2 3];y=[4 5 6];
>> v=x*y'
v =
   32
>> v=x'*y
```

```
v =
    4    5    6
    8   10   12
   12   15   18
```

Скалярное произведение двух векторов возвращает функция **dot**.

```
>> s=dot(x,y)
s =
   32
```

Векторное произведение. Для трехкомпонентных векторов в **MATLAB** существует функция **cross**, которая позволяет найти векторное произведение двух векторов. Пример:

```
>> v1=[1 2 3];v2=[4 5 6];
>> cross(v1,v2)
ans =
   -3    6   -3
```

Функция **det(B)** – вычисляет определитель $|B|$ квадратной матрицы B.

```
>> d=det(B)
d =
   -6
```

Функция *обращения матрицы* – **inv(B)** – вычисляет матрицу, обратную заданной матрице B. Исходная матрица B должна быть квадратной, и ее определитель не должен быть равен нулю.

Пример:

```
>> B1=inv(B)
B1 =
  -0.6667    0.3333   -1.1667
   0.3333    0.3333    0.3333
  -0.3333   -0.3333    0.1667
```

Матрица, обратная матрице B, обозначается B^{-1} и удовлетворяет соотношениям $(B^{-1})^{-1} = B$, $BB^{-1} = B^{-1}B = E$, где E – единичная матрица того же порядка n, что и B.

Проверим правильность результата выполнения операции обращения матрицы B:

```
>> disp(inv(B1))
  -1.0000   -2.0000   -3.0000
   1.0000    3.0000    1.0000
        0    2.0000    2.0000
```

В результате получили матрицу B, т.е. соотношение $(B^{-1})^{-1} = B$ выполняется.

```
>> disp(B*B1)
   1.0000         0         0
  -0.0000    1.0000   -0.0000
        0         0    1.0000
>> disp(B1*B)
   1.0000         0         0
        0    1.0000         0
        0   -0.0000    1.0000
```

В результате получили единичную матрицу E, т.е. соотношения $BB^{-1} = B^{-1}B = E$ также выполняются.

Примеры возведения квадратной матрицы в степень

```
>> disp(B^2)
   -1   -10    -5
    2     9     2
    2    10     6
>> disp(B^-1)
  -0.6667    0.3333   -1.1667
   0.3333    0.3333    0.3333
  -0.3333   -0.3333    0.1667
```

При возведении матрицы в целую положительную степень происходит *матричное* умножение матрицы на саму себя столько раз, каков показатель степени, в отличие от *поэлементного возведения матрицы в степень* при помощи оператора **<.^>** (см. раздел 2.7). Для отрицательных степеней вычисляется степень обратной матрицы.

Возможно использование дробных степеней. Извлечем из матрицы B кубический корень $D=\sqrt[3]{B}$.

```
>> D=B^(1/3)
D =
  0.5355 + 0.7217i  -0.3647 - 0.0000i  -0.7244 + 0.7217i
```

```
   0.2356 - 0.2165i   1.4422            0.2356 - 0.2165i
  -0.0355 + 0.1443i   0.3647            1.2244 + 0.1443i
```

Проверим результат возведением в куб.

```
>> D^3
ans =
  -1.0000 + 0.0000i  -2.0000 - 0.0000i  -3.0000 + 0.0000i
   1.0000 - 0.0000i   3.0000 - 0.0000i   1.0000 - 0.0000i
   0.0000 + 0.0000i   2.0000 + 0.0000i   2.0000 - 0.0000i
>> real(ans)
ans =
   -1.0000   -2.0000   -3.0000
    1.0000    3.0000    1.0000
    0.0000    2.0000    2.0000
```

Соотношение $D^3 = B$ выполняется, т.е. кубический корень $D=\sqrt[3]{B}$ найден верно.

Если требуется извлечь квадратный корень из матрицы, то лучше применить встроенную функцию **sqrtm**. Матричные экспонента и логарифм вычисляются при помощи функций **expm** и **logm**.

В **MATLAB** вводятся две новые функции (они не относятся к традиционным математическим операциям над векторами и матрицами) *деления матриц слева направо и справа налево*. Первая операция записывается при помощи знака < / >, а вторая – при помощи знака < \ >, которые помещаются между именами матриц – делимого и делителя. Операция *B/A* эквивалентна операции *B*inv(A)* и ее удобно использовать для решения матричного уравнения

*X*A=B,*

а *A\B* эквивалентна *inv(A)*B* и является решением матричного уравнения

*A*X=B.*

1.11 Решение систем линейных алгебраических уравнений

Пример компактных групповых вычислений системы **MATLAB** – решение систем линейных уравнений.

Решить квадратную систему линейных алгебраических уравнений

$$\begin{cases} x_1+3x_2 = -2; \\ -2x_1-2x_2+5x_3 =10; \\ x_1 - 5x_3 = -9. \end{cases}$$

Решение:

В матричном виде система имеет вид $Ax = b$, где

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

– матрица из коэффициентов при незвестных и вектор-столбцы, составленные соответственно из свободных членов и из неизвестных. Если $|A| = det(A) \neq 0,$ то применяя оператор обратного деления матриц < \ >, получим:

```
>> A=[1 3 0;-2 -2 5;1 0 -5]
A =
     1     3     0
    -2    -2     5
     1     0    -5
>> b=[-2;10;-9]
b =
    -2
    10
    -9
>> d=det(a)
d =
    -5
>> x=A\b
x =
     1
    -1
     2
```

Отметим, что за операторами ввода *A, b, c* не следует символ <**;**>. Благодаря этому легко осуществить контроль правильности ввода *A, b, c*.

Решение $x_1=1,\ x_2=-1,\ x_3=2$ легко проверить его подстановкой в систему уравнений:

```
>> disp(A*x)
  -2.0000
  10.0000
  -9.0000
```

В результате получен вектор-столбец свободных членов. Система решена верно.

Найдем обратную матрицу, а затем решение системы с помощью обратной матрицы:

```
>> A1=inv(A)
A1 =
  -2.0000   -3.0000   -3.0000
   1.0000    1.0000    1.0000
  -0.4000   -0.6000   -0.8000
>> A1*b
ans =
   1.0000
  -1.0000
   2.0000
```

Отметим, что такой способ требует больше времени и памяти, к тому же он может дать большую погрешность решения. Поэтому для решения систем следует применять знак < \ >.

Решить квадратную систему линейных алгебраических уравнений

$$\begin{cases} x_1+3x_2 = -2; \\ -2x_1-2x_2+5x_3 = 10; \\ x_1+5x_3 = -9. \end{cases}$$

Решение:

```
>> a=[1 3 0;-2 -2 5;1 0 5]
a =
    1    3    0
   -2   -2    5
    1    0    5
>> b=[-2 10 -9]
b =
   -2   10   -9
>> disp(det(a))
```

```
    35
>> disp(inv(a))
   -0.2857   -0.4286    0.4286
    0.4286    0.1429   -0.1429
    0.0571    0.0857    0.1143
>> x=a\b'
x =
   -7.5714
    1.8571
   -0.2857
```

Заметим, что в последнем операторе произведено обратное деление на вектор-столбец *b'*, поскольку вектор из свободных членов *b* введен как вектор-строка. Получили приближенное решение системы.

Точное решение системы и точная обратная матрица имеют вид:

```
>> format rat
>> x=a\b'
x =
   -53/7
    13/7
    -2/7
>> disp(inv(a))
    -2/7        -3/7         3/7
     3/7         1/7        -1/7
     2/35        3/35        4/35
```

ГЛАВА 2 РАБОТА С МАССИВАМИ

Эта глава знакомит с другими способами создания векторов и матриц. В ней рассмотрены способы формирования массивов определенного вида, создания новых массивов на основе существующих. Мы узнаем, что представляют собой поэлементные операции над массивами и чем они отличаются от векторных и матричных операций, выполняемых в соответствии с правилами векторного и матричного исчисления в математике.

2.1 Создание векторов

Длинный вектор можно вводить частями, которые затем объединять с помощью операции *сцепления строк*:

```
>> V1=[1 2 3];V2=[4 5 6];
>> V=[V1 V2]
V =
    1    2    3    4    5    6
```

Для создания нового вектора из определенных в заданном порядке элементов другого вектора применяется *индексация при помощи вектора*. Запись в массив *W* пятого, второго, первого и третьего элементов *V* производится следующим образом:

```
>> ind=[5 2 1 3];
>> W=V(ind)
W =
    5    2    1    3
```

Пусть в массиве *P*, соответствующем вектору-строке из девяти элементов, требуется заменить нулями элементы с третьего по седьмой. Эту задачу легко решить *индексацией с помощью двоеточия*.

Например,

```
>> P=[-1 0.1 2.2 3.4 5.6 3.1 6.8 9.7 5.5];
>> P(3:7)=0
P =
   -1.0000    0.1000         0         0         0         0         0
9.7000    5.5000
```

Индексация при помощи двоеточия удобна при выделении части из большого объема данных в новый массив. Пусть задан массив *W*

```
>> W=[0.1 2.2 3.4 5.6 3.1 6.8 9.7];
```

Составим массив *P*, состоящий из всех элементов массива *W*, кроме третьего, используя двоеточие и сцепление строк:

```
>> P=[W(1:2) W(4:7)]
P =
   0.1000    2.2000    5.6000    3.1000    6.8000    9.7000
```

Указание номеров элементов вектора можно использовать и при вводе векторов, последовательно добавляя новые элементы (не обязательно в порядке возрастания их номеров). Команды:

```
>> h=10;
>> h(2)=20;
>> h(4)=40;
```

приводят к образованию вектора:

```
>> h
h =
10 20 0 40
```

Заметим, что для ввода первого элемента *h* не обязательно указывать его индекс, т.к. при выполнении оператора *h=10* создается вектор (массив размера один на один). Следующие операторы присваивания приводят к автоматическому увеличению длины вектора *h*, а пропущенные элементы (в нашем случае *h(3))* получают значение ноль.

2.2 Поэлементные операции над векторами и матрицами

Векторы могут использоваться как аргументы различных математических функций (таких как **sin**, **cos**, **exp** и т.д). В результате вычисления этих функций получается вектор того же размера и типа, элементы которого равны значениям функции от соответствующих элементов вектора, заданного в качестве аргумента. Пример:

```
>> a=1:4
a =
    1    2    3    4
>> b=sin(a)
b =
   0.8415    0.9093    0.1411   -0.7568
```

Кроме преобразования векторов с помощью математических функций, в **MATLAB** можно выполнять поэлементные преобразования векторов с помощью арифметических операторов. Такие операции не относятся к традиционным математическим операциям над векторами, они лишь преобразуют элементы вектора как элементы обычного одномерного массива.

Операции поэлементного преобразования векторов могут выполнятся только над векторами *одинакового размера и типа*. В результате получается вектор такого же размера и типа.

Поэлементное умножение векторов осуществляется следующим образом:

```
>> x=[1,2,3,4,5];y=[-2,1,4,0,5];
>> z=x.*y
z =
   -2    2   12    0   25
```

Результатом поэлементного умножения векторов *x* и *y* являеся вектор *z*, каждый элемент которого представляет собой произведение соответствующих элементов векторов *x* и *y*. Оператор поэлементного умножения – совокупность знаков **<.*>** (без пробела между точкой и звездочкой).

Поэлементное деление векторов выполняется с помощью оператора **<./>**.

```
>> u=x./y
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to suppress this warning.)
u =
  -0.5000    2.0000    0.7500       Inf    1.0000
```

Результат – вектор *u*, элементы которого являются частным от деления соответствующих элементов векторов *x* и *y*.

Обратное поэлементное деление векторов (т.е. деление элементов второго вектора на соответствующие элементы первого) выполняется с помощью оператора <.\>.

```
>> v=x.\y
v =
  -2.0000    0.5000    1.3333         0    1.0000
```

Поэлементное возведение в степень выполняется с помощью оператора <.^>.

```
>> t=x.^y
t =
         1         2        81         1      3125
```

При поэлементном возведении в степень каждый элемент вектора *x* возводится в степень, равную соответствующему элементу вектора *y*.

Оригинальной в **MATLAB** является операция *прибавления к вектору числа*. Она записывается таким образом: *A*+*x* или *x*+*A* (где *A* – вектор, а *x* – число). Такая операция не относится к традиционным математическим операциям над векторами. Например,

```
>> a=[1 2 3 4]
a =
    1    2    3    4
>> disp(a+2)
    3    4    5    6
>> disp(2-a)
    1    0   -1   -2
```

В **MATLAB** поэлементные операции над матрицами аналогичны поэлементным операциям над векторами. То есть матрицы могут использоваться как аргументы различных математических функций. Пример:

```
>> A=[1 2 3 4 5;-2 3 1 4 0]
A =
    1    2    3    4    5
   -2    3    1    4    0
```

```
>> B=[-1 3 5 -2 1;1 8 -3 -1 2]
B =
   -1    3    5   -2    1
    1    8   -3   -1    2
>> disp(sin(A))
   0.8415   0.9093   0.1411  -0.7568  -0.9589
  -0.9093   0.1411   0.8415  -0.7568        0
```

Над матрицами *одинакового размера и типа* определены операции *поэлементного умножения*

```
>> disp(A.*B)
   -1    6   15   -8    5
   -2   24   -3   -4    0
```

поэлементного деления

```
>> disp(A./B)
  -1.0000   0.6667   0.6000  -2.0000   5.0000
  -2.0000   0.3750  -0.3333  -4.0000        0
>> disp(A.\B)
Warning: Divide by zero.
  -1.0000   1.5000   1.6667  -0.5000   0.2000
  -0.5000   2.6667  -3.0000  -0.2500      Inf
```

поэлементного возведения в степень

```
>> disp(A.^B)
 1.0e+003 *
   0.0010   0.0080   0.2430   0.0001   0.0050
  -0.0020   6.5610   0.0010   0.0003        0
```

Обратим внимание на результат, полученный при выполнении операции *A.^B*. Система **MATLAB** выделила общий множитель *1.0e+003* * для всех элементов результирующей матрицы.

прибавления к матрице числа

```
>> disp(A+2)
    3    4    5    6    7
    0    5    3    6    2
>> disp(4-B)
    5    1   -1    6    3
    3   -4    7    5    2
```

В результате этих операций получается матрица такого же размера и типа.

При поэлементном возведении в степень показателем степени может быть не только матрица того же размера, что и исходная, но и число.

```
>> disp(A.^3)
    1    8   27   64  125
   -8   27    1   64    0
```

2.3 Применение функций обработки данных к векторам и матрицам

Результатом применения функции **length(V)** к некоторому вектору *V* является длина вектора – число его элементов.

Функция **max(V)** выдает значение максимального по значению элемента вектора *V*. Аналогично, функция **min(V)** извлекает минимальный элемент вектора *V*.

Функции **mean(V)** и **std(V)** определяют, соответственно, среднее значение и среднеквадратическое отклонение вектора *V*.

Функция сортировки **sort(V)** формирует вектор, элементы которого распределены в порядке возрастания их значений.

Функция **sum(V)** вычисляет сумму элементов вектора *V*.

Функция **prod(V)** выдает произведение всех элементов вектора *V*.

Функция **cumsum(V)** формирует вектор того же типа и размера, любой элемент которого является суммой всех предыдущих элементов вектора *V* (вектор кумулятивной суммы).

Функция **cumprod(V)** создает вектор, элементы которого являются произведением всех предыдущих элементов вектора *V*.

Функция **diff(V)** выдает вектор, имеющий размер на единицу меньший, чем размер вектора *V*, элементами которого является разность между соседними элементами вектора *V*.

Применение описанных функций проиллюстрировано ниже на одном и том же векторе *v*.

```
>> v=[1 .1 .5 .1 .1 .4]
v =
   1.0000    0.1000    0.5000    0.1000    0.1000    0.4000
>> disp(length(v))
    6
>> disp(max(v))
    1
>> disp(min(v))
   0.1000
>>  disp(mean(v))
   0.3667
>>  disp(std(v))
   0.3559
>>  disp(sort(v))
   0.1000    0.1000    0.1000    0.4000    0.5000    1.0000
>>  disp(sum(v))
   2.2000
>>  disp(prod(v))
  2.0000e-004
>>  disp(cumsum(v))
   1.0000    1.1000    1.6000    1.7000    1.8000    2.2000
>>  disp(cumprod(v))
   1.0000    0.1000    0.0500    0.0050    0.0005    0.0002
>>  disp(diff(v))
  -0.9000    0.4000   -0.4000         0    0.3000
```

Часто необходимо знать не только значение минимального или максимального элемента в массиве, но и его индекс (порядковый номер). Если указать второй выходной параметр, то можно получить дополнительную информацию о первом индексе элемента, значение которого является максимальным или минимальным:

```
>> [M,n]=max(v)
M =
    1
n =
```

```
    1
>> [M,n]=min(v)
M =
   0.1000
n =
    2
```

Для того, чтобы упорядочить вектор *v* по убыванию, также используем функцию **sort**:

```
>> R=-sort(-v)
R =
   1.0000    0.5000    0.4000    0.1000    0.1000    0.1000
```

Вызов **sort** с двумя выходными аргументами приводит к образованию массива индексов соответствия элементов упорядоченного и исходного массивов:

```
>> [r,ind]=sort(v)
r =
   0.1000    0.1000    0.1000    0.4000    0.5000    1.0000
ind =
    2    4    5    6    3    1
```

Если аргументом функций **max** и **min** является вектор, состоящий из комплексных чисел, то результатом является максимальный или минимальный по модулю элемент. Функция **sort** также упорядочивает комплексный вектор по модулю, а компоненты с равными модулями располагаются в порядке возрастания фаз.

Применение встроенных функций обработки данных к некоторым последовательно расположенным элементам вектора не представляет труда. Следующий вызов функции **prod** вычисляет произведение элементов вектора *z* со второго по шестой:

```
>> p=prod(z(2:6))
```

Сумма элементов произвольного вектора *V* с четными индексами находится следующим образом:

```
>> ind=2:2:length(V);
>> s=sum(V(ind))
```

Результатом применения функции **size(M)** к некоторой матрице *M* является вектор-строка из двух элементов, первым

из которых является число строк матрицы *M*, а вторым – число ее столбцов.

Те же функции **max, min, mean, std, sort, sum, prod, cumsum, cumprod, diff**, что применялись к векторам, могут быть применены и к матрицам. Основным отличием использования в качестве аргументов этих функций именно матриц является то, что соответствующие описанные выше операции производятся по отношению к каждому из столбцов заданной матрицы (за исключением, конечно, функции **size**). Так, в результате применения функций **max, min, mean, std** получаются векторы-строки с числом элементов, равным числу столбцов заданной матрицы. Каждый элемент результата содержит, соответственно, максимальное, минимальное, среднее или среднеквадратическое значение элементов соответствующего столбца заданной матрицы.

Приведем примеры.

Пусть имеем *3* величины *y1*, *y2* и *y3*, измеренные при некоторых пяти значениях аргумента (которые не указаны). Тогда данные измерений образуют *3* вектора по *5* элементов, наример

```
>> y1=[5.5 6.3 6.8 8 8.6];
>> y2=[-1.2 0.5 -0.6 1 0.1];
>> y3=[3.4 5.6 0 8.4 10.3];
```

Сформируем из них матрицу измерений так, чтобы векторы *y1, y2, y3* образовывали столбцы этой матрицы:

```
>> A=[y1',y2',y3']
A =
   5.5000   -1.2000    3.4000
   6.3000    0.5000    5.6000
   6.8000   -0.6000         0
   8.0000    1.0000    8.4000
   8.6000    0.1000   10.3000
```

Применим к этой матрице функции **size, max, min, mean, std**.

```
> disp(size(A))
     5     3
>> disp(max(A))
```

```
   8.6000    1.0000   10.3000
>> disp(min(A))
   5.5000   -1.2000         0
>> disp(mean(A))
   7.0400   -0.0400    5.5400
>> disp(std(A))
   1.2582    0.8735    4.0655
```

Если при обращении к функциям **max** и **min** указать второй выходной параметр, то он даст информацию о номере строки, где находится в соответствующем столбце первый элемент с максимальным (или минимальным) значением. Например:

```
>> [M,n]=max(A)
M =
   8.6000    1.0000   10.3000
n =
    5     4     5
>> [M,n]=min(A)
M =
   5.5000   -1.2000         0
n =
    1     1     3
```

Если в качестве второго входного аргумента **sum** указать *2*, то суммирование произойдет по строкам.

```
>> M=[1 2;3 4; 5 6]
M =
    1     2
    3     4
    5     6
>> disp(sum(M,2))
    3
    7
   11
```

Для вычисления суммы всех элементов матрицы требуется дважды применить **sum**.

```
>> disp(sum(sum(M)))
   21
```

Самостоятельно о функциях обработки данных можно узнать в разделе **datafun** справочной системы **MATLAB**.

2.4 Формирование массивов определенного вида

В **MATLAB** предусмотрены специальные встроенные функции для создания векторов и матриц определенного вида. Рассмотрим некоторые из них.

zeros(M,N) – создает матрицу размером $M \times N$ с нулевыми элементами:

```
>> zeros(3,4)
ans =
   0   0   0   0
   0   0   0   0
   0   0   0   0
```

ones(M,N) – создает матрицу размером $M \times N$ с единичными элементами:

```
>> ones(3,4)
ans =
   1   1   1   1
   1   1   1   1
   1   1   1   1
```

eye(M,N) – создает матрицу размером $M \times N$ с единицами по главной диагонали и остальными нулевыми элементами:

```
>> eye(3,4)
ans =
   1   0   0   0
   0   1   0   0
   0   0   1   0
```

rand(M,N) – создает матрицу размером $M \times N$ из случайных чисел, равномерно распределенных в диапазоне от 0 до 1:

```
>> rand(3,4)
ans =
   0.9501   0.4860   0.4565   0.4447
   0.2311   0.8913   0.0185   0.6154
   0.6068   0.7621   0.8214   0.7919
```

randn(M,N) – создает матрицу размером $M \times N$ из случайных чисел, распределенных по нормальному закону с нулевым математическим ожиданием и стандартным (среднеквадратическим) отклонением, равным единице 1:

```
>> randn(3,4)
ans =
    1.1908   -0.1567   -1.0565    0.5287
   -1.2025   -1.6041    1.4151    0.2193
   -0.0198    0.2573   -0.8051   -0.9219
```

Для всех перечисленных функций можно задавать один аргумент M в случае квадратной матрицы (M=N). Например,

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

rot90 – осуществляет разворот матрицы на 90^o против часовой стрелки:

```
>> Q=[1 2;3 4]
Q =
1 2
3 4
>> R=rot90(Q)
R =
2 4
1 3
```

diag

1) диагональная матрица, элементы которой задаются во входном аргументе – векторе *V*

D=diag(V)

2) диагональная матрица со смещенной на *k* позиций диагональю (положительные *k* – смещение вверх, отрицательные – вниз), результатом является квадратная матрица размера *length(V)+abs(k)*

D=diag(V,k)

3) выделение главной диагонали из матрицы *A* в вектор *d*

d=diag(A)

4) выделение *k*-ой диагонали из матрицы *A* в вектор *d*

d=diag(A,k)

Разберем, как получить трехдиагональную матрицу *B* размера *5×5,* приведенную ниже, с использованием функций **MATLAB**.

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

Введем вектор *v* с целыми числами *от одного до пяти* и используйте его для создания диагональной матрицы и матрицы со смещенной на единицу вверх диагональю. Вектор длины шесть, содержащий *двойки*, заполняется, например, так: *2*ones(1,4).* Этот вектор укажем в первом аргументе функции **diag**, а минус единицу – во втором и получим третью вспомогательную матрицу. Теперь достаточно вычесть из первой матрицы вторую и сложить с третьей:

```
>> v=1:5;
>> B=diag(v)-diag(v(1:4),1)+diag(2*ones(1,4),-1)
B =
    1   -1    0    0    0
    2    2   -2    0    0
    0    2    3   -3    0
    0    0    2    4   -4
    0    0    0    2    5
```

Справочная информация о функциях, используемых для создания массивов определенного вида, находится в разделе **elmat** справочной системы **MATLAB**.

2.5 Создание новых массивов на основе существующих

Выделение блоков матриц осуществляется индексацией с помощью двоеточия. Введем матрицу

```
>> P=[1 2 0 2;4 10 12 5;0 11 10 5;9 2 3 5]
P =
```

```
    1     2     0     2
    4    10    12     5
    0    11    10     5
    9     2     3     5
```

Выделим из нее матрицу размером *2×2*, находящуюся в центре:

```
>> P1=P(2:3,2:3)
P1 =
   10    12
   11    10
```

Вставим на это же место матрицу *F* размером *2×2*:

```
>> F=[3 -7;4 11]
F =
    3    -7
    4     5
>> P(2:3,2:3)=F
P =
    1     2     0     2
    4     3    -7     5
    0     4    11     5
    9     2     3     5
```

Для удаления отдельных столбцов и строк матрицы используются пустые квадратные скобки *[ ]*. Удалим второй столбец матрицы *P*:

```
>> P(:,2)=[]
P =
    1     0     2
    4    12     5
    0    10     5
    9     3     5
```

А теперь удалим вторую строку:

```
>> P(2,:)=[]
P =
    1     0     2
    0    10     5
    9     3     5
```

«Растянуть» матрицу P в единый вектор-столбец V можно с помощью простой записи:

```
>> V=P(:)
V =
     1
     0
     9
     0
    10
     3
     2
     5
     5
```

«Расширять» матрицу, составляя ее из отдельных заданных матриц («блоков») можно тоже довольно просто. Если заданы несколько матриц – блоков *A1, A2,…,AN* с одинаковым числом строк, то из них можно «слепить» единую матрицу A, объединяя блоки в одну «строку» операцией *горизонтального сцепления A=[A1,A2,…,AN]*. Аналогично, *вертикальное сцепление* матриц можно реализовать при условии, что все составляющие блоки–матрицы имеют одинаковое число столбцов, применяя для отделения блоков вместо запятой – точку с запятой: *A=[A1;A2;…,AN]*.

Пример горизонтального сцепления:

```
>> A1=[1 2 3;4 5 6;7 8 9];A2=[10;11;12];A3=[14 15;16 17;18 19];A=[A1,A2,A3]
A =
     1     2     3    10    14    15
     4     5     6    11    16    17
     7     8     9    12    18    19
```

Пример вертикального сцепления:

```
>> B1=[1 2 3 4 5];B2=[6 7 8 9 10;11 12 13 14 15];B3=[17 18 19 20 21];B=[B1;B2;B3]
B =
     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
```

```
   17    18    19    20    21
```

Построение таблицы значений функции

Пусть требуется вывести в командное окно таблицу значений функции

$y=3e^{-0,5x}\cdot\sin x$ ***при изменении аргумента*** **x** ***от*** **0** ***до*** **5** ***с шагом*** **0,5***. Вычисление массива значений этой функции в указанных условиях можно осуществить с помощью простых операторов:*

```
>>  a=3;h=0.5;x=0:.5:5;y=a*exp(-h*x).*sin(x);
>> x
x =
  Columns 1 through 10
         0    0.5000    1.0000    1.5000    2.0000    2.5000
3.0000    3.5000    4.0000    4.5000
  Column 11
    5.0000
>> y
y =
  Columns 1 through 10
         0    1.1201    1.5311    1.4135    1.0035    0.5144
0.0945   -0.1829   -0.3073   -0.3091
  Column 11
   -0.2361
```

Результат, отображенный на экране, не похож на таблицу. Построим таблицу в виде столбцов значений переменной и функции, используя вертикальное сцепление:

```
>> disp([x' y'])
         0         0
    0.5000    1.1201
    1.0000    1.5311
    1.5000    1.4135
    2.0000    1.0035
    2.5000    0.5144
    3.0000    0.0945
    3.5000   -0.1829
    4.0000   -0.3073
    4.5000   -0.3091
```

5.0000 -0.2361

2.6 Собственные числа и векторы матрицы. Матричные функции линейной алгебры

В **MATLAB** существует большое число функций, реализующих разнобразные операции линейной алгебры.

К таким функциям относятся **det** и **inv**, рассмотренные в разделе 1.10.

Важной задачей линейной алгебры является задача на собственные значения квадратной матрицы *A*.

Собственные числа (значения) λ_i и *собственные векторы* ($u_i \neq 0$) квадратной матрицы *A* удовлетворяют равенствам

$Au_i = \lambda_i u_i$.

Пусть дана квадратная матрица

$A = \begin{bmatrix} 3 & 4 \\ 5 & 2 \end{bmatrix}$.

Функция **eig** с входным аргументом матрицей и выходным – вектором записывает в него все собственные числа матрицы:

```
>> A=[3 4;5 2];
>> Lambda=eig(A)
Lambda =
    7
   -2
```

Функция **eig** с двумя выходными аргументами возвращает одновременно все собственные векторы и числа:

```
>> [U,Lam]=eig(A)
U =
   0.7071   -0.6247
   0.7071    0.7809
Lam =
    7     0
    0    -2
```

Первый выходной аргумент *U* является матрицей, составленной по столбцам из собственных векторов. Для

доступа к первому собственному вектору u_1 используем индексацию при помощи двоеточия

```
>> u1=U(:,1)
u1 =
   0.7071
   0.7071
```

Аналогично, для вектора u_2

```
>> u2=U(:,2)
u2 =
  -0.6247
   0.7809
```

Вторым выходным аргументом *Lam* возвращается диагональная матрица, содержащая собственные числа исходной матрицы $\lambda_1 = 7$, $\lambda_2 = -2$.

Итак, собственным числам $\lambda_1 = 7$ и $\lambda_2 = -2$ соответствуют собственные векторы

$$u_1 = \begin{bmatrix} 0{,}7071 \\ 0{,}7071 \end{bmatrix} \text{ и } u_2 = \begin{bmatrix} -0{,}6247 \\ 0{,}7809 \end{bmatrix}.$$

Проверим выполнение равенств $Au_i = \lambda_i u_i$, $i=1,2$:

```
>> L1=7;L2=-2;
>> disp(A*u1-L1*u1)
     0
     0
>> disp(A*u2-L2*u2)
  1.0e-015 *
   -0.2220
   -0.4441
```

Равенства выполняются, т.е. задача на собственные значения для рассмотренной выше матрицы A решена верно.

Собственные значения λ матрицы A являются корнями ее характеристического уравнения $|A-\lambda E| = 0$, где E – единичная матрица того же порядка n, что и A. Многочлен $|A-\lambda E|$ называется характеристическим полиномом матрицы A. В общем случае имеется n различных комплексных или вещественных корней характеристического уравнения. Однако в отдельных случаях при наличии кратных корней их число уменьшается.

Составим характеристическое уравнение для рассмотренной выше матрицы A:

$n = 2, E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, |A-\lambda E| = \begin{vmatrix} 3-\lambda & 4 \\ 5 & 2-\lambda \end{vmatrix} = (3-\lambda)(2-\lambda)-20 = \lambda^2-5\lambda-14 = 0.$

В этом случае характеристический полином $\lambda^2-5\lambda-14$. Решив уравнение $\lambda^2-5\lambda-14 = 0$, получим: $\lambda_1 = 7, \lambda_2 = -2$.

Функции **poly** возвращает массив коэффициентов характеристического полинома матрицы A:

```
>> poly(A)
ans =
    1   -5   -14
```

Из равенств $Au_i = \lambda_i u_i$ следует, что собственные векторы определяются с точностью до числового множителя, поэтому одну из координат собственного вектора можно фиксировать на каком-либо конкретном значении. Так, если зафиксировать в векторе u_1 первую координату равной 1, то это равносильно делению вектора u_1 поэлементно на $u_1(1)$:

```
>> v1=u1/u1(1)
v1 =
    1
    1
```

Следовательно, вектор с целочисленными координатами, соответствующий вектору u_1, равен

$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Проверка:

```
>> disp(A*v1-L1*v1)
    0
    0
```

Найдем отношение координат вектора u_2:

```
>> format rat
>> disp(u2(1)/u2(2))
   -4/5
```

Таким образом, вектор с целочисленными координатами, соответствующий вектору u_2, равен

$v_2 = \begin{bmatrix} -4 \\ 5 \end{bmatrix}$.

Проверка:

```
>> v2=[-4;5];
>> disp(A*v2-L2*v2)
     0
     0
```

Умножением v_1 и v_2 на целочисленные множители (не равные *0*) можно получить и другие собственные векторы с целочисленными координатами.

Найденные с помощью функции **eig** собственные вектора отличаются от всех других тем, что являются *единичным* или *нормированными по евклидовой норме*.

Одна из норм $||V||_p$ вектора V длины n определяется в MATLAB следующим образом:

$$||V||_p = \sqrt[p]{\sum_{k=1}^{n} |v_k|^p} \ (p=1, 2, \ldots).$$

Если $p=2$ – норма называется *евклидовой*.

Функция **norm(V,p)** возвращает норму $||V||_p$ вектора V.

Функция **norm(V)** возвращает евклидову норму вектора V по умолчанию.

Для вещественных векторов V длины $n=2,3$ *norm(V)* совпадает с длиной вектора V.

Если *norm(V)=1*, вектор V называется *единичным* или *нормированным*.

Найдем норму (длину) каждого из векторов u_1, u_2, v_1 , v_2:

```
>> disp(norm(u1))
     1
>> disp(norm(u2))
     1
>> disp(norm(v1))
   1.4142
>> disp(norm(v2))
   6.4031
```

Вектор V нормируется следующим образом:

$V_{norm} = \frac{V}{norm(V)}$.

Пример нормировки вектора v_1:

```
>> Vn1=v1/norm(v1)
Vn1 =
   0.7071
   0.7071
```

В результате получили нормированный вектор u_1 .

Вывод:

В результате нормировки любой собственный вектор матрицы A совпадает с соответствующим ему нормированным вектором u_i с точностью до знака.

Норма вектора **norm(V,inf)** равна наибольшему из модулей элементов вектора *V*.

Норма вектора **norm(V,–inf)** равна наименьшему из модулей элементов вектора *V*.

Для матрицы *A* функция **norm** имеет в **MATLAB** четыре модификации, описание которых получим, набрав в командной строке команду **doc norm**.

Ранг матрицы A есть такое число $r = r_A$, что по крайней мере один из определителей r - го порядка, получаемый из этой матрицы при удалении строк и/или столбцов, отличен от нуля, а все определители $(r+1)$ – го порядка равны нулю. Ранг матрицы равен наибольшему числу линейно независимых строк (или столбцов). *Квадратная матрица A порядка n является невырожденной в том и только в тот случае, когда ее ранг* $r_A = n$, т.е. $det(A) \neq 0$.

Функция **rank(A)** возвращает ранг матрицы *A*.

Найдем ранг и рассмотренной выше матрицы *A*:

```
>> disp(rank(A))
    2
```

Поскольку ранг и порядок матрицы равны *2*, ее определитель ненулевой:

```
>> det(A)
ans =
  -14
```

Перечислим некоторые из функций, реализующих операции линейной алгебры:

cond – число обусловленности матрицы;

lu – разложение на треугольные матрицы;

svd – сингулярное разложение матрицы и т.д.

Справочная информация по матричным функциям находится в разделе **matfun** справочной системы **MATLAB**.

ГЛАВА 3 М-ФАЙЛЫ

До сих пор все вычисления и операции в **MATLAB** мы выполняли в «режиме калькулятора»: набирали в командной строке очередную команду, нажимали клавишу **<Enter>** – система выполняла заданное действие и, при необходимости, выдавала результат этого действия. Для повторного выполнения команд или создания на их основе похожих команд, мы использовали клавиши <↑> и <↓>, а также окно **Command History** (История команд), из которого можно скопировать отдельные команды и даже наборы команд (см. раздел 1.7).

Однако, такой пошаговый режим работы хорош только для разового расчета. Если та же вычислительная схема требует небольшой модификации алгоритма или исходных данных, работа в пошаговом режиме становится неэффективной.

*Более эффективным способом выполнения команд в системе **MATLAB** является применение М-файлов (текстовых файлов с расширением **.m**), содержащих набор инструкций на языке системы.*

В системе **MATLAB** различают два типа М-файлов: файл-программы (Script M-Files) и файл-функции (Function M-Files). М-файлы являются примерами программ в системе **MATLAB**. Подробнее о программировании – в Главе 4.

Для написания М-файлов предназначен специальный редактор, служащий для создания, редактирования и отладки программ. Умение создавать собственные файл-функции и файл-программы необходимо как при программировании в **MATLAB**, так и при решении различных задач средствами **MATLAB** Например, поиска корней уравнений, интегрирования, оптимизации и др. Справочную информация по М-файлам можно получить, набрав в командной строке **doc lang**.

3.1 Файл-программы

Файл-программы (их также называют *скриптами* или *сценариями*) являются самым простым типом М-файлов. При повторном вводе большого набора инструкций не только тратится много времени, но и возникает вероятность появления ошибок. В простейшем случае можно включить любую цепочку команд в текстовый файл, записать его на диск под произвольным именем с расширением **.m**, а затем вызвать и уже автоматически выполнить, не нажимая клавишу **<Enter>** после очередной команды.

В **MATLAB** имеется редактор М-файлов, который вызывается из меню **File** по команде **New => M-file**. На экране появляется окно редактора. Наберем в нем какие-либо команды, например, для нахождения длины *d* радиус–вектора точки *(x;y;z)* трехмерного пространства (см. рис. 3.1):

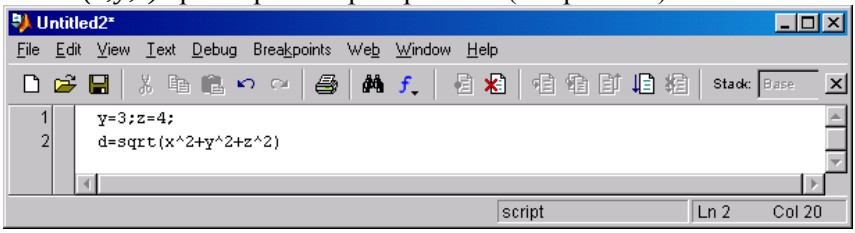


Рис.3.1

Заметим, что точка с запятой **<;>** предотвращает вывод в командное окно результатов действий отдельных команд при выполнении программы. Оператор *2* в сценарии **mysum** не завершен точкой с запятой, поэтому выводит результат *d* в командное окно.

Для запуска сценария или его части есть несколько способов. Первый, самый простой – выделить операторы при помощи мыши, удерживая левую кнопку, или при помощи клавиши **<Shift>** со стрелками **<↑>** и **<↓>** и нажать **<F9>**. Выделенные операторы выполняются последовательно, точно так же, как если бы они были набраны в командной строке. Очевидно, что работать в М-файле удобнее, чем в командной строке, поскольку можно сохранить программу, можно редактировать ее, используя обычные приемы редактирования,

общие для **Windows** – приложений, в том числе с помощью мыши и клавиш <↑> и <↓>.

Для сохранения программы, набранной в окне редактора М-файлов, можно воспользоваться командой **File => Save As (Сохранить как)**. В появившемся диалоговом окне **Save fili as** (Сохранить файл как) раскроется подкаталог **work** основного каталога **MATLAB**, который по умолчанию является текущим (его содержимое можно увидеть в окне **Current Directory**). В поле **File name (Имя файла)** введем имя файл-программы (например, **myprog.m**) вместо **Untitled.m** и щелкнем на кнопке **Save**, чтобы сохранить его. М-файлы не следует путать с МАТ-файлами, в которых хранятся переменные из рабочего пространства системы **MATLAB.**

После того, как программа сохранена в М-файле, к примеру в **myprog.m**, для ее запуска можно использовать пункт **Run** меню **Debug**, либо набрать в командной строке имя М-файла (без расширения) и нажать **<Enter>**, то есть выполнить, как команду **MATLAB**. При таких способах запуска программы следует учесть важное обстоятельство – путь к каталогу с М-файлом должен быть известен **MATLAB.** Сделаем каталог с файлом **myprog** текущим.

В **MATLAB** установка текущего каталога производится из окна **Current Directory** рабочей среды. Если это окно отсутствует, то следует выбрать пункт **Current Directory** меню **View** рабочей среды. Для выбора желаемого каталога на диске нажмем кнопку, расположенную слева от раскрывающегося списка. Созданный М-файл можно сохранить не только в текущем, но и в любом другом каталоге. В этом случае перед запуском файла на выполнение нужно установить пути поиска, ведущие к нему.

Когда текущий каталог установлен, то все М-файлы, находящиеся в нем, могут быть запущены из командной строки, либо из редактора М-файлов.

Например, если ввести команды

```
>> x=2;
>> myprog
d =
```

5.3852

то сценарий **myprog.m**, текст которого содержится на рис 3.1, вычислит длину радиус–вектора точки *(2;3;4)*.

Сценарий **myprog.m** обрабатывает как свои собственные переменные *y, z*, так и переменную *x* , определенную до вызова сценария в командном окне системы **MATLAB** и хранящуюся в ее рабочем пространстве. Это возможно потому, что переменные, *определяемые в сценариях*, и переменные, *определяемые в командном окне*, составляют *единое рабочее пространство* системы **MATLAB**. В свою очередь, все переменные, созданные во время работы сценария, остаются в рабочем просранстве **MATLAB** и после окончания его выполнения, т. е. являются *глобальными*. Можно убедится в этом, выполнив команду **whos**.

3.2 Файл-функции

В файл-функциях описываются функции, определяемые пользователем. Они отличаются от сценариев тем, что могут иметь входные и выходные аргументы, а все переменные, определенные внутри файл-функции, являются *локальными* и не видны в рабочей среде. Они доступны только в теле данной функции. После выполнения данной файл-функции все значения этих переменных исчезают, а область оперативной памяти, в которой они хранились, освобождается. М-файл, содержащий файл-функцию, должен начинаться с *заголовка*, после которого следует *тело функции*, где записываются инструкции на языке системы, с помощью которых в итоге вычисляютя возвращаемые значения. Заголовок состоит из слова **function**, возвращаемых (выходных) значений, имени файл-функции и списка входных аргументов. Входные аргументы в списке разделяются запятой. Простейший пример файл-функции с тремя входными и одним выходным аргументами – нахождение длины радиус–вектора точки *(x;y;z)*. трехмерного пространства.

```
function d=mysum(x,y,z)
d=sqrt(x^2+y^2+z^2);
```

Наберем этот пример в редакторе М-файлов, который вызывается по команде **File => New => M-file**, и сохраним его. Заметим, что **MATLAB** предлагает в качестве имени М-файла название файл-функции, т.е. **mysum.m**. Указанное в заголовке имя функции должно совпадать с именем файла (без учета расширения **.m**), в который записывается текст функции. Рассогласование имени функции и имени файла не допускается. Убедимся, что каталог с файлом **mysum.m** является текущим и вызовем файл-функцию **mysum** из командной строки:

```
>> s=mysum(2,3,4)
s =
   5.3852
```

При вызове файл-функции **mysum** произошли следующие события:

- входной аргумент *x* получил значение *2*;
- входной аргумент *y* стал равен *3*;
- входной аргумент ***z*** стал равен *4*;
- величина $\sqrt{x^2+y^2+z^2}$ записалась в выходной аргумент *d*;
- значение выходного аргумента *d* получила переменная *s* рабочей среды и результат вывелся в командное окно.

Заметим, что оператор *2* в файл-функции **mysum** завершен точкой с запятой для подавления вывода локальной переменной *d* в командное окно. Для просмотра значений локальных переменных при отладке М-функций, очевидно, не следует подавлять вывод на экран значений требуемых переменных.

При необходимости, отдельные части М-файла можно дополнять комментариями, которые начинаются со знака процента *(%)* и не являются исполняемыми инструкциями, а служат лишь для целей документирования процесса программирования. Цель их размещения в тексте программы – пояснить смысл той или иной части программного кода. Например, файл-функцию **mysum** можно снабдить таким комментарием:

function d=mysum(x,y,z)

%Вычисление длины d=sqrt(x^2+y^2+z^2) радиус-вектора точки (x;y;z)

d=sqrt(x^2+y^2+z^2);

Комментарии, располагающиеся сразу за заголовком определения файл-функции, воспринимаются системой **MATLAB** как краткая справка об используемой функции. Эта справка появится в командном окне, если ввести команду

>> help *имя функции*

Аргументами **mysum** могут быть не только числа или скалярные переменные, но и массивы требуемых размеров. Так, если изменить синтаксис оператора *2* файл-функции **mysum** – поставить точки перед операциями возведения в степень и сохранить в **mysum1.m** текст

function d=mysum1(x,y,z)

d=sqrt(x.^2+y.^2+z.^2);

то входные аргументы *x,y,z* теперь могут быть массивами. При вызове

>> x=[1 2 3];y=[-1 0 2];z=[3 2 1];

>> s=mysum1(x,y,z)

s =

3.3166 2.8284 3.7417

компоненты переменной *s* рабочей среды (массива размера *1×3*) получили значения длин радиус–векторов точек *(1;-1;3), (2;0;2)* и *(3;2;1)* соответственно.

Запуск **mysum1(2,3,4**) и **mysum(2,3,4**) со скалярными входными аргументами дает одинаковый результат *5.3852.*, а при запуске **mysum(x,y,z**) с входными аргументами–массивами *x,y,z* из предыдущего примера появится сообщение об ошибке – неправильном возведении массива в степень в операторе *2*. Поэтому в создаваемых М-файлах желательно предусмотреть процедуру *векторизации кода* – вместо операторов <^>, <*****>, </> следует применять операторы <.^>, <.*****>, <./>.

Разберем теперь, как создать файл-функцию с несколькими выходными аргументами. Список выходных

аргументов в заголовке файл-функции заключается в квадратные скобки, сами аргументы отделяются запятой.

Пример файл-функции **mysum2** с тремя входными и двумя выходными аргументами, выдающей длину *d* и квадрат длины *d2* радиус–вектора точки трехмерного пространства *(x;y;z)*:

```
function [d,d2]=mysum2(x,y,z)
d2=x.^2+y.^2+z.^2;
d=sqrt(d2);
```

При запуске файл-функции **mysum2** используем квадратные скобки для указания переменных *m* и *n*, в которые будут занесены значения *d* и *d2*:

```
>> [m,n]=mysum2(2,3,4)
m =
   5.3852
n =
   29
```

Практически все функции **MATLAB** являются файл-функциями и хранятся в одноименных М-файлах. Функция ***cos*** допускает два варианта вызова: *cos(x)* и *y=cos(x),* в первом случае результат записывается в *ans***,** а во втором – в переменную *y*. Наша функция **mysum2** ведет себя точно так же.

Файл-функцию **mysum2** можно вызвать без выходных аргументов, или только с одним выходным аргументом. В этом случае вернется только первый аргумент – длина *d*.

Передача информации из командного окна **MATLAB** в файл-функцию осуществляется с помощью параметров функции. Другой механизм передачи информации – глобальные переменные. Файл-функция может и не иметь входных или выходных аргументов, заголовки таких файл-функций приведены ниже:

function noout(a,b), function [v,u]=noin, function noarg().

Для того, чтобы рабочая область **MATLAB** и файл-функция могли совместно использовать некоторую

переменную с заданным именем, ее всюду нужно объявить как глобальную с помощью ключевого слова **global**.

Пример файл-функции **mysum3** без входных аргументов:

```
function [d,d2]=mysum3
global x y z
d2=x.^2+y.^2+z.^2;
d=sqrt(d2);
```

Вызов ее осуществляется следующим образом:

```
>> global x y z
>> x=2;y=3;z=4;
>> [m,n]=mysum3
m =
   5.3852
n =
   29
```

С помощью ключевого слова **global** перечисленные в командном окне переменные рабочего пространства *x=2, y=3, z=4* сохраняют свое значение и имя внутри файл-функции **mysum3**.

ГЛАВА 4 ПРОГРАММИРОВАНИЕ

Для освоения методов программирования в **MATLAB** не нужно обладать какими-то специальными знаниями. Встроенный язык программирования **MATLAB** (М-язык) достаточно прост, он содержит необходимый минимум конструкций. И хотя работа в «режиме калькулятора» не является программированием, все рассмотренные ранее команды, операторы и функции являются типичными инструментами М-языка.

К основному набору конструкций М-языка относятся также операторы отношения, логические операторы, операторы ветвления и цикла, рассматриваемые в этой главе.

4.1 Операторы отношения и логические операторы

Операторы отношения служат для поэлементного сравнения двух операндов, в качестве которых могут выступать числа, векторы или матрицы. При этом сравниваемые векторы или матрицы должны иметь одинаковые размеры. Если операнды одинаковы, то программа возвращает *1* (**True –** Истина), в противном случае **–** *0* (**False –** Ложь). Перечень операторов отношения с соответствующими им функциями представлен в табл. 4.1.

Таблица 4.1. Операторы отношения и их функции

Оператор	***Название***	***Функция***
==	Равно	eq
~=	Не равно	ne
<	Меньше	lt
>	Больше	gt
<=	Меньше или равно	le
>=	Больше или равно	ge

Операторы = = и ~= сравнивают действительные и коплексные переменные. При этом сравниваются действительные и комплексные части числа.

Операторы <, <=, >, >= при сравнении комплексных чисел сравнивают только действительные части числа.

Примеры приведены в табл. 4.2.

Таблица 4.2. Примеры использования операторов отношения

Выражение	***Функция***	***Результат***
>> 3==3	>> eq(3,3)	ans = 1
>> 5~=5	>> ne(5,5)	ans = 0
>> 4+2i==4+i	>> eq(4+2i,4+i)	ans = 0
>> 7.2<8.3	>> lt(7.2,8.3)	ans = 1
>> 1.4+5i<1.5+i	>> lt(1.4+5i,1.5+i)	ans = 1
>> 3<=2.33	>> le(3,2.33)	ans = 0

Если при вычислениях надо формально определить, является ли переменная *x* комплексной, можно вызвать функцию **isreal(x)**, возвращающую *1*, если *x* не является комплексной, и *0* в противном случае.

В выражениях, вводимых в командном окне системы **MATLAB**, операторы отношения могут использоваться наряду с арифметическими операторами. Рассмотрим пример вычисления выражения, содержащего операторы отношения.

```
>> a=1;b=-1;c=2;
>> (a>=c)+(b==a)+(c>a)
ans =
    1
```

Здесь значения выражений *(a>=c)* и *(b==a)* равны *0* (Ложь), значение выражения *(c>a)* равно *1* (Истина). В результате переменная *ans*, являющаяся суммой значений этих трех выражений, оказывается равной *1*.

Операции отношения имеют более низкий приоритет, чем арифметические операции. Но в этом примере переменная *ans* равна сумме значений трех операций отношения только

потому, что эти операции заключены в круглые скобки. Если же скобки опустить, результат будет иным.

```
>> a>=c+b==a+c>a
ans =
    0
```

При поэлементном сравнении двух массивов одинаковых размеров с помощью операторов отношения результат будет представлен в виде массива того же размера, состоящего из нулей и единиц. Пример:

```
>> A=[1 0;-2 3];B=[2 3;-3 2];
>> A>B
ans =
    0    0
    1    1
```

В операторах отношения допустимо сравнение массива и числа. В этом случае происходит сравнение каждого элемента массива с числом. Результатом является массив того же размера, что и исходный. Пример:

```
>> A=[1 0;-2 3];b=0.5;
>> A>b
ans =
    1    0
    0    1
```

Логические операторы предназначены для выполнения поэлементных логических операций над массивами одинаковых размеров. Логические операторы и соответствующие им функции приведены в табл. 4.3.

Таблица 4.3. Логические операторы и их функции

Оператор	***Название***	***Функция***
&	Логическое И	and
|	Логическое ИЛИ	or
Отсутствует	Исключающее ИЛИ	xor
~	Логическое НЕ	not

Первые три операции являются двухоперандными (бинарными), а операция "Не" является унарной (однооперандной).

При выполнении логических операций «истинными» считаются операнды, не равные нулю, а «ложными» – операнды, равные нулю. При этом результатом операции "И" будет *1*, если оба операнда не равны нулю, и *0*, если хотя бы один из операндов нулевой. Операция "ИЛИ" дает *1*, если хотя бы один операнд не равен нулю. Операция "исключающее ИЛИ" выдает *1* лишь тогда, когда один из операндов равен нулю, а другой не равен, в остальных случаях она выдает *0*. Операция "Не" выдает *1*, если ее единственный операнд равен нулю, и *0* в противном случае.

Примеры использования логических операторов:

```
>> A=[1 0;-2 3];B=[2 3;-3 2];
>> and(A,B)                    или >> A&B
ans =
    1    0
    1    1
>> or(A,B)                     или >> A|B
ans =
    1    1
    1    1
>> xor(A,B)
ans =
    0    1
    0    0
>> not(A)                      или >> ~A
ans =
    0    1
    0    0
```

Элементами логических операторов могут быть массив и число. В этом случае происходит поэлементное выполнение логической операции для каждого элемента массива и числа. Результатом является массив того же размера, что и исходный. Пример:

```
>> A=[1 0;-2 3];b=3;
>> xor(A,b)
ans =
    0    1
    0    0
```

Поскольку логические и арифметические операции могут входить в одно выражение, порядок выполнения этих операций зависит от их приоритета. Выполнение операций одинакового приоритета происходит в порядке слева направо. Приоритет операций можно изменить с помощью круглых скобок.

Приоритеты операций системы **MATLAB** в порядке убывания приведены ниже:

1. Круглые скобки <**( )**>.

2. Транспонирование <**.'**>, транспонирование с комплексным сопряжением<'>, возведение в степень <^>, поэлементное возведение в степень <**.^**>.

3. Унарный плюс <+>, унарный минус <–>, логическое отрицание <~>.

4. Умножение и деление (в том числе поэлементное) <*****>, </>, <\>, <**.***>, <**./**>, <**.**>.

5. Сложение <+> и вычитание <–>.

6. Операции отношения **<**, **<=**, >, >=, ==, **~=**.

7. Логическое И <**&**>.

8. Логическое ИЛИ <|>.

Отметим, что сначала выполняются операции над агументами функций **eq**, **ne**, **lt**, **gt**, **le**, **ge**, **and**, **or**, **not**, если использовать их вместо соответствующих им операторов. Например, два выражения *and(A,B)+F* и *A&B+F* не эквивалентны.

Справка – **doc ops**.

4.2 Операторы цикла

Цикл **for** используется для повторения операторов в случае, когда число повторений заранее известно. Синтаксис цикла **for** имеет следующий вид:

for *var = b1:b2:b3*

Операторы (текст программы)

end

Здесь *var* – переменная (счетчик) цикла, которая при каждом повторении цикла изменяется от начального значения *b1* до конечного значения *b3* с шагом *b2* (если параметр *b2* не указан, по умолчанию его значение принимается равным *1*). Переменная цикла может принимать не только целые, но и вещественные значения с любым знаком. Операторы разделяются запятой <**,**>, точкой с запятой <**;**> или нажатием клавиши **<Enter>** Набор операторов завершается служебным словом **end**. Цикл завершается, как только значение *var* превысит *b3*. Операторы между **for** и **end** воспринимаются системой как части одного сложного оператора. Поэтому нажатие клавиши **<Enter>** для перехода к следующей строке не приводит в данном случае к выполнению этих операторов. Выполнение операторов начинается только тогда, когда введена "закрывающая скобка" сложного оператора в виде служебного слова **end.**

Если несколько сложных операторов такого типа вложены один в другой, вычисления начинаются лишь тогда, когда записан конец (**end**) наиболее охватывающего (внешнего) сложного оператора.

Например, для поиска суммы элементов матрицы, расположенных ниже главной диагонали, следует использовать два цикла **for**, причем начальное значение счетчика внутреннего цикла зависит от текущего значения счетчика внешнего цикла.

```
function s=Sn(A)
[n m]=size(A);
s=0;
for j=1:m
for i=j+1:n
   s=s+A(i,j);
end
end
```

В том случае, когда число повторений заранее неизвестно и определяется в ходе выполнения блока

операторов, следует организовать цикл **while**. Цикл **while** работает, пока выполнено условие цикла.

Пример файл-функции **polsum** , которая находит сумму всех первых положительных элементов вектора:

```
function s=polsum(x)
s=0;
k=1;
while x(k)>0
s=s+x(k);
k=k+1;
end
```

Вызовем файл-функцию **poslum** из командной строки:

```
>> a=[1 -2 3];
>> S=polsum(a)
S =
    1
```

Файл-функция **poslum** имеет один недостаток: если все элементы вектора – положительные числа, то *k* становится больше длины вектора *x*, что приводит к ошибке, например:

```
>> a=[1 2 3];
>> S=polsum(a)
???  Index exceeds matrix dimensions.
```

Кроме проверки значения *x(k),* следует позаботиться о том, чтобы значение *k* не превосходило длины вектора *x*. Вход в цикл должен осуществляться только при одновременном выполнении условий *k<= length(x)* и *x(k)>0*, т. е. необходимо применить логический оператор "И", обозначаемый в **MATLAB** символом **&**. Заменим в файл-функции **poslum** условие цикла *while x(k)>0* на составное *while k<=length(x)&x(k)>0*. Если первое из условий не выполняется, то второе условие проверяться не будет, именно поэтому выбран такой порядок операндов. Теперь файл-функция **polsum** будет работать верно для любых векторов:

```
>> a=[1 2 3];
>> S=polsum(a)
S =
    6
```

Справка – **doc lang**.

4.3 Операторы ветвления

Ветвление в ходе работы программы осуществляется при помощи конструкции **if-elseif-else**. Самый простой вариант ее использования (без **elseif** и **else**) реализован в файл-функции **possum** , которая предназначена для нахождения суммы всех элементов вектора, больших *2*.

```
function s=possum2(x)
s=0;
for k=1:length(x)
if x(k)>2
s=s+x(k);
end
end
```

Если ход программы должен изменяться в зависимости от нескольких условий, то следует использовать полную конструкцию **if-elseif-else**. Каждая из ветвей **elseif** в этом случае должна содержать условие выполнения блока операторов, размещенных после нее. Важно понимать, что условия проверяются подряд, первое выполненное условие приводит к работе соответствующего блока, выходу из конструкции **if-elseif-else** и переходу к оператору, следующему за **end**. У последней ветви **else** не должно быть никакого условия. Операторы, находящиеся между **else** и **end**, работают в том случае, если все условия оказались невыполненными.

Предположим, что требуется написать файл-функцию для вычисления кусочно-заданной функции:

$$f(x)=\begin{cases}\sqrt[3]{x}, & x<-1;\\ x, & -1\le x\le 3;\\ 2-x, & x>3.\end{cases}$$

Она имеем вид

```
function f=pwf(x)
if x<-1
f=x^(1/3);
elseif x<=3
```

```
f=x;
else
f=2-x;
end
```

Первое условие $x<-1$ проверяется в ветви **if**. Отметим, что условие $-1\le x$ не требуется включать в следующую ветвь **elseif** , поскольку в эту ветвь программа заходит, если предыдущее условие ($x<-1$) не выполнено. Условие $x>3$ также проверять не надо. Если не выполнены два предыдущих условия $x<-1$ *и* $x\le 3$, то x будет больше трех.

Справка – **doc lang**.

4.4 Оператор переключения switch

Ход работы программы может определяться значением некоторой переменной (переключателя). Такой альтернативный способ ветвления программы основан на использовании оператора переключения **switch**. Переменная-переключатель помещается после **switch** через пробел. Оператор **switch** содержит блоки, начинающиеся со слова **case.** После каждого **case** через пробел записывается то значение переключателя, при котором выполняется данный блок. Последний блок начинается со слова **otherwise**, его операторы работают в том случае, когда ни один из блоков **case** не был выполнен. Если хотя бы один из блоков **case** выполнен, то происходит выход из оператора **switch** и переход к оператору, следующему за **end**.

Предположим, что требуется найти количество единиц и минус единиц в заданном массиве и, кроме того, найти сумму всех элементов, отличных от единицы и минус единицы. Создадим файл-функцию, которая по заданному массиву возвращает число минус единиц в первом выходном аргументе, число единиц – во втором, а сумму – в третьем. В ней следует перебрать все элементы массива в цикле, причем в роли переменной-переключателя будет выступать текущий элемент массива.

```
function [m,p,s]=mpsum(x)
m=0;
```

```
p=0;
s=0;
for i=1:length(x)
switch x(i)
case -1
m=m+1;
case 1
p=p+1;
otherwise
s=s+x(i);
end
end
```

Блок **case** может быть выполнен не только при одном определенном значении переключателя, но и в том случае, когда переключатель принимает одно из нескольких допустимых значений. В этом случае значения указываются после слова **case** в фигурных скобках через запятую, например, **case {1,2,3}**.

Справка – **doc lang**.

4.5 Оператор прерывание цикла break

Досрочное завершение циклов **while** или **for** осуществляется при помощи оператора **break**.

Пусть, например, требуется по заданному массиву *x* образовать новый массив *y* по правилу *y(k)=x(k+1)/x(k)* до первого нулевого элемента *x(k)*, т.е. до тех пор, пока имеет смысл операция деления. Номер первого нулевого элемента в массиве *x* заранее неизвестен, более того, в массиве *x* может и не быть нулей. Решение задачи состоит в последовательном вычислении элементов массива *y* и прекращении вычислений при обнаружении нулевого элемента в *x*. Следующая файл-функция демонстрирует работу оператора **break**.

```
function y=div(x)
for k=1:length(x)-1
if x(k)==0
break
end
```

```
y(k)=x(k+1)/x(k);
end
```

Справка – **doc lang**.

4.6 Пример сравнения быстродействия матричных и скалярных операций

За *умножением* векторов и матриц в *смысле линейной алгебры* в **MATLAB** закреплен знак **<*>**. Определение произведения прямоугольных матриц дано в разделе 1.10.

Пример умножения матриц в *матричной* форме с помощью знака **<*>**:

```
>> A=ones(3)
A =
    1    1    1
    1    1    1
    1    1    1
>> B=ones(3)
B =
    1    1    1
    1    1    1
    1    1    1
>> C=A*B
C =
    3    3    3
    3    3    3
    3    3    3
```

В традиционных языках программирования умножение матриц в *смысле линейной алгебры* осуществляется в *скалярной* форме с помощью вложенных циклов.

Ниже приведен текст файл-функции умножения матриц *D* и *E* размера $n \times n$ в скалярной форме:

```
function G=Matr(D,E,n)
for i=1:n
for j=1:n
s=0;
for k=1:n
s=s+D(i,k)*E(k,j);
```

```
end
G(i,j)=s;
end
end
```

Сохраним этот текст в **Matr.m**. Теперь с помощью файл-функции **Matr.m** скалярное умножение матриц *A* и *B* можно реализовать следующим образом:

```
>> A=ones(3);B=ones(3);
>> Matr(A,B,3)
ans =
    3    3    3
    3    3    3
    3    3    3
```

В этом примере умножение матриц осуществлено в *скалярной* форме. Результаты умножения матриц в *матричной* и *скалярной* формах совпадают.

Сравним теперь быстродействие обеих форм умножения матриц. Система **MATLAB** имеет удобное средство для приблизительного замера (оценки) быстродействия выполнения оператора или М-функции – набор команд **tic** и **toc**. В командной строке вызов некоторой функции обрамляется с двух сторон этими командами:

```
>> tic, Fun, toc
```

После вычисления функции в командное окно выдается также оценка времени вычислений (сек.), которая зависит от быстродействия конкретного компьютера. Это время может несколько отличаться при повторном замере. Поэтому делают несколько измерений, а в качестве итоговой оценки используют среднее арифметическое значение затраченного времени.

Найдем оценку времени выполнения этой файл-функции на используемом компьютере:

```
>> A=ones(3);B=ones(3);
>> tic,Matr(A,B,3);toc
elapsed_time =
   0
```

Найдем теперь оценку времени выполнения умножения этих же матриц *A* и *B* в *матричной* форме:

```
>> tic,A*B;,toc
elapsed_time =
    0
```

Так как матрицы *A* и *B* имеют небольшой размер *3×3*, то оценочное время выполнения умножения в обеих формах практически равно нулю.

Проведем теперь сравнение по быстродействию операции умножения матриц *A* и *B* размером *300×300* с использованием этой операции в *скалярной* и в *матричной* формах:

```
>> A=ones(300);B=ones(300);
>> tic,Matr(A,B,300);,toc
elapsed_time =
  504.1600
>> tic,A*B;toc
elapsed_time =
    0.2700
```

В этом случае оценка времени выполнения умножения в *скалярной* форме *504,16* сек. (≈ *8,4* мин.) и *0,27* сек. – в *матричной форме*.

Следовательно, умножение в *матричной* форме матриц *A* и *B* размером *300×300* осуществляется в MATLAB примерно в $\frac{504,16}{0,27} \approx 1900$ раз быстрее, чем в *скалярной форме* (независимо от быстродействия компьютера).

Для матриц *A* и *B* размера *1000×1000* оценка времени выполнения умножения в *скалярной* форме (на используемом компьютере) исчисляется часами, а в *матричной* – равно *8,24* сек.:

```
>> A=ones(1000);B=ones(1000);
>> tic,A*B;,toc
elapsed_time =
    8.2400
```

В рассмотренных примерах использованы единичные матрицы. Это связано с простотой генерации матриц. Матрицы

A и *B* автоматически созданы в результате вызова функции **ones** (см. раздел 2.4).

Теперь используем матрицы со случайными значениями элементов. Для этого используем встроенный генератор **rand** (см. раздел 2.4) равномерно распределенных случайных чисел, и создадим из этих чисел матрицы A и B размером *1000×1000*.

При этом оценка времени, которое затрачивается на выполнения их умножения в *матричной* форме составляет:

```
>> A=rand(1000);B=rand(1000);
>> tic,A*B;,toc
elapsed_time =
   8.1200
```

Результат *8,12* сек. практически такой же, как и для единичных матриц того же размера (*8,24* сек.).

Вывод

Там, где это возможно, *вместо операторов цикла лучше применять матричные операции над массивами*, которые исполняются в **MATLAB** намного быстрее.

ГЛАВА 5 СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ

В предыдущих главах рассматривались численные процедуры **MATLAB**, выполняемые с аргументами, заданными в виде чисел или числовых массивов. В состав **MATLAB** входит пакет **Symbolic Math ToolBox**, который добавил к системе возможность *символьных вычислений*. Помимо типовых *аналитических вычислений* (таких, как упрощение математических выражений, подстановка, нахождение пределов, дифференцирование, интегрирование, интегральные преобразования, получение решений уравнений и систем уравнений, включая дифференциальные и т.д.) пакет **Symbolic** позволяет реализовывать *арифметические операции с любой точностью*. Функции пакета **Symbolic** реализуют интерфейс между средой **MATLAB** и ядром **Maple IV** системы аналитических вычислений, или системы компьютерной алгебры, канадского университета *Waterlooo*, причем работа в **MATLAB** не требует установки **Maple**. Версия системы **Maple IV** в своем ядре и в расширениях имеет около *3000* функций. Система **MATLAB** с пакетом **Symbolic**, включающим в себя около сотни символьных команд и функций, намного уступает **Maple** по их количеству. Однако, пользователь, имеющий опыт работы в **Maple**, может напрямую обращаться ко всем функциям и процедурам этой системы (кроме графических), написанным на встроенном языке **Maple** (см. раздел 7.17). Способы получения справочной информации по пакету **Symbolic** рассмотрены в Приложении 1.

5.1 Символьные переменные, константы и выражения

Поскольку переменные системы **MATLAB** по умолчанию не определены и традиционно задаются как векторные, матричные, числовые и т.д., то есть не имеющие отношения к символьной математике, для реализации символьных вычислений нужно прежде всего позаботиться о создании специальных *символьных переменных*.

Для создания символьных переменных или объектов используется функция **sym**.

Например, команда

```
>>x=sym('x')
x =
x
```

возвращает символьную переменную с именем *'x'* и записывает результат в *x*.

Команда **x=sym('x','real')** дополнительно определяет *x* как вещественную переменную. Аналогично, **x=sym('x','positive')** определяет *x* как положительную (вещественную) переменную, а **x=sym('x','unreal')** определяет *x* как чисто формальную переменную (то есть не обладающую никакими дополнительными свойствами).

Для создания группы символьных объектов служит функция **syms**.

Команда

```
>> syms a b c
```

создает символьные переменные с именами *a, b, c*.

Команда

```
>> Pi=sym('pi');
```

создает символьное число *Pi=π*, не обладающее погрешностью представления числа π в формате с плавающей запятой. Результаты операций с символьным *Pi* выражаются не в числовой, а в символьной форме. Следовательно, пакет **Symbolic** позволяет получить точные значения тригонометрических функций (и их рациональных комбинаций) от аргумента π в *виде выражений*, включающих квадратные корни из рациональных чисел, если такие выражения существуют и могут быть найдены системой. Например, точное значение $sin\frac{\pi}{5}$ имеет вид:

```
>> S=sin(Pi/5)
S =
1/4*2^(1/2)*(5-5^(1/2))^(1/2)
```

Символьное выражение *S* выведено в командное окно в одну строку.

Функция **pretty(S)** выводит в командное окно символьное выражение **S** в формате, приближенном математическому.

```
>> pretty(S)
                    1/2        1/2 1/2
               1/4 2     (5 - 5    )
```

Теперь очевидно, что $sin\frac{\pi}{5} = \frac{\sqrt{2}}{4}\sqrt{5-\sqrt{5}}$.

Символьное выражение можно создать при помощи функции **sym**, входным аргументом которой является строка с выражением, заключенным в апострофы. Например,

```
>> F=sym('x+y')
F =
x+y
```

Команда **syms** без аргументов выводит список символьных объектов, имеющихся в рабочем пространстве.

При запросе о наличии символьных переменных в памяти после выполнения предыдущего примера

```
>> syms
   'F'
```

получен ответ *'F'*, то есть входящие в выражение переменные *x* и *y* не являются символьными. Их нельзя использовать в качестве аргументов в дальнейших символьных вычислениях.

Изменим рассмотренный выше ввод символьного выражения *F* следующим образом:

```
>> syms x y
>> F=x+y
F =
x+y
>> syms
   'F'    'x'    'y'
```

Теперь переменные *x* и *y* вначале получили статус символьных, а сконструированное из них выражение *F* приобрело статус символьного автоматически.

5.2 Вычисления с использованием арифметики произвольной точности

MATLAB обычно ведет вычисления с числами, представленными в формате плавающей точки с двойной точностью. Это довольно высокая точность, обеспечивающая потребности практических вычислений в прикладных задачах. Однако, ряд задач теории чисел, численного кодирования и некоторых других требует выполнения вычислений вообще без какой – либо погрешности или со сколь угодно малой погрешностью. Такие вычисления не очень удачно называют *арифметикой произвольной точности* – правильнее говорить о *точной* арифметике.

Для проведения вычислений в арифметике произвольной точности служит функция **vpa**:

R=vpa(S) – возвращает результат вычислений каждого элемента символьного массива *S* , используя арифметику произвольной точности с текущим числом цифр *D* , установленным функцией **digits.** Результат *R* имеет тип **sym**;

R=vpa(S,D) - возвращает результат вычислений каждого элемента массива *S* с текущим числом цифр *D.*

Примеры:

```
>>vpa(exp(1),50)
ans =
2.7182818284590455348848081484902650117874145507
813
>>vpa([2*pi,exp(1),log(2)],10)
ans =
[ 6.283185308, 2.718281828, .6931471806]
```

Функция **digits** служит для установки числа цифр в числах арифметики произвольной точности. Она используется в одном из следующих вариантов:

digits – возвращает число значащих цифр в числах арифметики произвольной точности (по умолчанию *32*);

digits(D) – устанавливает заданное число цифр *D* для арифметики произвольной точности.

Примеры вычисления числа π с *32* и *6* значащими цифрами:

```
>>digits
Digits = 32
>>vpa pi
ans =
3.1415926535897932384626433832795
>> digits 6
>> vpa pi
ans =
3.14159
```

5.3 Функции упрощения выражений – simplify, simple

Функция **simplify(S)** поэлементно упрощает символьные выражения массива *S*. Если упрощение невозможно, то возвращается исходное выражение. Примеры:

```
>>syms a b x
>>V=[sin(x)^2+cos(x)^2, log(a*b)]
V =
[ sin(x)^2+cos(x)^2,        log(a*b)]
>>simplify(V)
ans =
[       1, log(a*b)]
>>simplify((a^2-2*a*b+b^2)/(a-b))
ans =
a-b
```

Возможности проведения упрощений с помощью команды **simplify** в **Symbolic** не обладают возможностями системы **Maple** в полной мере в связи с отсутствием опций, определяющих путь упрощения. Дополнительные возможности упрощения обеспечивает функция **simple.**

Функция **simple(S)** выполняет различные упрощения символьных выражений массива *S* и выводит как промежуточные результаты, так и самый короткий результат. В другой форме – **[R,HOW] = simple(S)** – промежуточные результаты не выводятся. Результаты упрощений содержатся в векторе *R* , а в строковом векторе *HOW* указывается

выполняемое преобразование. Следующие примеры иллюстрируют работу функции:

S

R

HOW

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

1

combine

2*cos(x)^2-sin(x)^2

3*cos(x)^2-1

simplify

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

cos(2*x)

combine

```
[R,HOW]=simple(cos(x)^2-sin(x)^2)
R =
cos(2*x)
HOW =
combine
```

5.4 Функция расширения выражений – expand

Функция **expand(S)** расширяет символьные выражения массива *S*. Рациональные выражения она раскладывает на простые дроби, полиномы – на полиномиальные выражения и т.д. Функция работает со многими алгебраическими и тригонометрическими функциями. Примеры:

```
>>syms a b x;
>>S=[(x+2)*(x+3)*(x-4),sin(2*x)];
>>expand(S)
ans =
[ x^3+x^2-14*x-24, 2*sin(x)*cos(x)]
>>expand(sin(a+b))
ans =
sin(a)*cos(b)+cos(a)*sin(b)
>>expand((a+b)^3)
ans =
a^3+3*a^2*b+3*a*b^2+b^3
```

5.5 Разложение выражений на простые множители – функция factor

Функция **factor(S)** поэлементно разлагает символьные выражения массива *S* на простые множители, а целые числа – на произведение простых чисел. Примеры:

```
>>x=sym('x');
>>factor(x^7-1)
ans =
(x-1)*(x^6+x^5+x^4+x^3+x^2+x+1)
>>factor(sym('123456789'))
ans =
(3)^2*(3803)*(3607)
```

Пусть требуется найти определитель *D* (функция **det**) и обратную матрицу A^{-1} (функция **inv**) символьной матрицы

$A = \begin{bmatrix} a & b \\ a^2 & b^2 \end{bmatrix}$.

```
>> syms a b
>> A=[a b;a^2 b^2]
A =
[   a,   b]
[ a^2, b^2]
>> D=det(A)
D =
a*b^2-b*a^2
>> factor(D)
ans =
-a*b*(-b+a)
>> A1=inv(A)
A1 =
[ -b/a/(-b+a),  1/a/(-b+a)]
[  a/b/(-b+a), -1/b/(-b+a)]
```

5.6 Приведение подобных членов – функция collect

Функция **collect(S, v)** работает с символьными полиномами *S* нескольких переменных, где *v* – одна из переменных полинома. Эта функция возвращает разложение

полинома *S* по степеням *v* (*S* может быть массивом полиномов). Примеры:

```
>>syms x y
>>S=[x^3*y^2+x^2*y+3*x*y^2,x^4*y-y*x^2];
>>collect(S,x)
ans =
[ x^3*y^2+x^2*y+3*x*y^2,          x^4*y-x^2*y]
>>collect(S,y)
ans =
[ (x^3+3*x)*y^2+x^2*y,        (x^4-x^2)*y]
```

5.7 Обеспечение подстановок – функция subs

Одной из самых эффективных и часто используемых операций символьной математики является операция подстановки. Она реализуется с помощью функции **subs** со следующими формами записи:

- **subs(S)** – заменяет в символьном выражении *S* все переменные их символьными значениями, которые берутся из вычисляемой функции или рабочей области среды **MATLAB**.
- **subs(S, NEW)** – заменяет все свободные символьные переменные в *S* из списка *NEW*.
- **subs(S, OLD, NEW)** – заменяет *OLD* на *NEW* в символьном выражении *S*. При одинаковых размерах массивов *OLD* и *NEW* замена идет поэлементно. Если *S* и *OLD* – скаляры, а *NEW* - числовой массив или массив ячеек, то скаляры расширяются до массива результатов.

Примеры:

```
>>syms a b x y;
>>subs(x-y,y,1)
ans =
x-1
>>subs(sin(x)+cos(y),[x y],[a b])
ans =
sin(a)+cos(b)
```

Подстановка вместо переменной ее числового значения приводит к вычислению символьной функции от значения аргумента, например:

```
>> s=sym('x^(x+1)');
>> f=subs(s,'x',1.5)
f =
   2.7557
```

Число можно заменить его символьным представлением и затем найти значение функции с произвольной точностью при помощи функции **vpa**:

```
>> f=subs(s,'x','1.5')
f =
(1.5)^((1.5)+1)
>> vpa(f,40)
ans =
2.755675960631075360471944584044 1
```

5.8 Функция вычисления пределов – limit

Для вычисления пределов функции *F(x)*, заданной в аналитическом (символьном) виде, служит функция **limit**, которая используется в одном из следующих вариантов:

- **limit(F,x,a)** – возвращает предел символьного выражения *F* в точке *x=a*;
- **limit(F,x,a,'right')** или **limit(F,x,a,'left')** – возвращает предел в точке *a* справа или слева.

Продемонстрируем приемы вычисления пределов на следующих примерах:

а) $\lim\limits_{x\to 0}\frac{\sin ax}{ax}$; **б)** $\lim\limits_{x\to\infty}\left(1+\frac{3}{x}\right)^{2x}$ **с)** $\lim\limits_{x\to 1}\frac{1}{1-x}$; **д)** $\lim\limits_{x\to 1+0}\frac{1}{1-x}$;

е) $\lim\limits_{x\to 1-0}\frac{1}{1-x}$; **ж)** $\lim\limits_{x\to\infty}\frac{x(\ln(a+x)-\ln x)}{5}$.

Решения в указанном порядке будут иметь вид

```
>> syms a x
>> limit(sin(a*x)/(a*x),x,0)
ans =
1
>> limit((1+3/x)^(2*x),x,inf)
ans =
exp(6)
>> limit(1/(1-x),x,1)
```

```
ans =
NaN
```

Здесь переменная **NaN** означает, что предела функции $\frac{1}{1-x}$ в точке $x=1$ не существует.

```
>> limit(1/(1-x),x,1,'right')
ans =
-inf
```

Правосторонний предел функции $\frac{1}{1-x}$ в точке $x=1$ существует и равен $-\infty$.

```
>> limit(1/(1-x),x,1,'left')
ans =
inf
```

Левосторонний предел функции $\frac{1}{1-x}$ в точке $x=1$ существует и равен $+\infty$.

```
>> limit(x*(log(a+x)-log(x))/5,x,inf)
ans =
1/5*a
```

5.9 Функция вычисления производных – diff

Для вычисления в символьном виде производных от выражения *S* служит функция **diff**, записываемая в форме **diff(S, x, n)**. Она возвращает символьное значение *n*-ой производной (производной степени n) от символьного выражения или массива символьных выражений *S* по переменной *x*, т.е.

$$\mathbf{S^n(x) = \frac{d^n(S)}{dx^n}.}$$

В формате **diff(S, x)** находится первая производная (*n=1* по умолчанию).

Пусть требуется найти первую и третью производные функции $y = x^2 sinx$.

Решения будут иметь вид:

```
>> syms x
>> y=x^2*sin(x);
```

```
>> diff(y,x)
ans =
2*x*sin(x)+x^2*cos(x)
>> diff(y,x,3)
ans =
6*cos(x)-6*x*sin(x)-x^2*cos(x)
```

Если выражение *S* зависит от нескольких переменных, например, *S=S(x,y)*, то ее частная производная $\frac{\partial S}{\partial x}$ (или $S'_x(x,y)$)по аргументу *x* есть производная этой функции по *x* при постоянном значении *y*.

В декартовой системе координат на плоскости *xOy* градиент функции *S(x,y)* есть вектор

$$grad\ S = \left(\frac{\partial S}{\partial x}, \frac{\partial S}{\partial y}\right).$$

Частными производными второго порядка функции *S=S(x,y)* называются частные производные от ее первых производных $\frac{\partial S}{\partial x}, \frac{\partial S}{\partial y}$, т.е.

$$\frac{\partial^2 S}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial S}{\partial x}\right), \frac{\partial^2 S}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial S}{\partial y}\right), \frac{\partial^2 S}{\partial y \partial x} = \frac{\partial}{\partial y}\left(\frac{\partial S}{\partial x}\right), \frac{\partial^2 S}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial S}{\partial y}\right).$$

Частные производные второго порядка обозначаются также символами

$S''_{xx}(x,y), S''_{xy}(x,y), S''_{yx}(x,y), S''_{yy}(x,y)$.

Аналогично определяются и обозначаются частные производные более высоких порядков. Смешанные производные второго порядка, (например, $S''_{xy}(x,y)$ и $S''_{yx}(x,y)$), отличающиеся только порядком дифференцирования, равны между собой при условии их непрерывности.

Функция *S* может быть массивом. В этом случае результат вычисления будет представлен в виде массива, элементами которого являются производные от исходных функций, образующих массив. Например,

```
>> syms a x
>> y=[a*log(x);x^a];
>> diff(y,x)
ans =
```

```
[     a/x]
[ x^a*a/x]
```

Рассмотрим еще один пример. Для функции двух переменных *f(x,y) = arcsinxy* вычислить *gradf(x;y)* в точке *(0;0)* и найти смешанную прозводную $\frac{\partial^2 f}{\partial x \partial y}$.

Решение:

```
>> syms x y
>> f=asin(x*y);
>> x1=diff(f,x)
x1 =
y/(1-x^2*y^2)^(1/2)
>> y1=diff(f,y)
y1 =
x/(1-x^2*y^2)^(1/2)
>> subs([x1 y1],[x y],[0 0])
ans =
    0     0
>> xy=diff(diff(f,x),y)
xy =
1/(1-x^2*y^2)^(1/2)+y^2/(1-x^2*y^2)^(3/2)*x^2
>> [m n]=simple(xy)
m =
1/(1-x^2*y^2)^(3/2)
>> pretty(m)
                    1
                --------------
                     2  2 3/2
                (1 - x  y )
```

Итак, $grad(arcsinxy)\big|_{(0;0)} = (0;0)$ и $\frac{\partial^2 arcsinxy}{\partial x \partial y} = \frac{1}{\sqrt{(1-x^2y^2)^3}}$.

5.10 Функция вычисления интегралов – int

В ряде случаев возникает необходимость вычисления неопределенных и определенных интегралов

$I = \int f(x)dx$ и $I = \int_a^b f(x)dx$.

Здесь *f(x)* - подынтегральная функция независимой переменной *x*, *a* – нижний и *b* – верхний пределы интегрирования для определенного интеграла.

Функция **int(f, x)** – возвращает неопределенный интеграл (первообразную функцию) от символьного выражения *f* по переменной *x*.

Функция **int(f, x, a, b)** – возвращает значение определенного интеграла от символьного выражения *f* по переменной *x* с пределами от a до *b*.

Вычислить неопределенный интеграл:

$$\int \frac{x}{1-x^2}\,dx.$$

Решение:

```
>> syms x
>> int(x/(1-x^2))
ans =
-1/2*log(x-1)-1/2*log(1+x)
```

Полученное решение с помощью элементарных преобразований приводится в виду $\frac{1}{2}\,ln\frac{1}{x^2-1}$. Для этого можно попробовать упростить полученный ответ с помощью функций **simple**, **simplify** или **factor**.

Проверим результат дифференцированием:

```
>> diff(-1/2*log(x-1)-1/2*log(1+x),x)
ans =
-1/2/(x-1)-1/2/(1+x)
>> [q,w]=simple(ans)
q =
-x/(x^2-1)
```

В результате получена подинтегральная функция, т.е. интеграл вычислен правильно.

Вычислить неопределенный интеграл:

$$\int \frac{x}{a+bx^2}\,dx.$$

Решение:

```
>> syms x a b
>> int(x/(a+b*x^2),x)
ans =
1/2/b*log(a+b*x^2)
```

В этом случае подинтегральная функция была задана в аналитическом виде с символьными переменными *a*, *b*, *x*.

Вычислить определенный интеграл:

$$\int_c^d \frac{x}{a+bx^2}\,dx.$$

Решение:

```
>> syms x a b c d
>> int(x/(a+b*x^2),x,c,d)
ans =
1/2*(log(a+d^2*b)-log(a+c^2*b))/b
>> pretty(ans)
                                2                  2
                           log(a + d  b) - log(a + c  b)
                      1/2 ---------------------------------
                                          b
```

В данном случае подинтегральная функция была задана в аналитическом виде с символьными переменными, а пределы интегрирования – также в виде символьных переменных.

Вычислить определенный интеграл:

$$\int_0^{\pi/4} arcsinx\,dx.$$

Решение:

```
>> syms x
>> int(asin(x),x,0,pi/4)
ans =
```

```
1/4*pi*asin(1/4*pi)+1/4*(16-pi^2)^(1/2)-1
```

Для получения решения в числовой форме надо активизировать с помощью мыши строку ответа и нажать клавишу **<Enter>**. Будет получен следующий ответ:

```
ans =
  0.32847177096777
```

Вычислить несобственный интеграл:

$$\int_0^{\infty} xe^{-x}dx.$$

Решение:

```
>> syms x
>> int(x*exp(-x),x,0,inf)
ans =
1
```

Вычисления интегралов с бесконечными пределами с помощью функции **int** имеют некоторые особенности. Вычислим интеграл

$$\int_0^{\infty} t^n e^{-at}dt.$$

Решение:

```
>> syms a n t
>> int(t^n*exp(-a*t),t,0,inf)
ans =
int(t^n*exp(-a*t),t = 0 .. inf)
```

Решение в явном виде не найдено. Предположим теперь, что переменная *n* – положительная. Тогда программа будет иметь вид:

```
>> syms a t
>> n=sym('n','positive');
>> int(t^n*exp(-a*t),t,0,inf)
ans =
1/(a^n)/a*gamma(n)*n
```

Если $n \geq 0$ целое, то $gamma(n) \times n = n!$, и в этом случае $\int_0^\infty t^n e^{-at} dt = \frac{n!}{a^{n+1}}$.

Функция **int** не позволяет получить неопределенный интеграл от произвольной функции. Например, при вычислении интеграла $\int e^{\sin x} dx$ получим следующий результат:

```
>> syms x
>> int(exp(sin(x)),x)
ans =
int(exp(sin(x)),x)
```

В некоторых случаях **int** возвращает выражение для первообразной через специальные функции, например, при вычислении интеграла $\int \frac{\sin x}{x} dx$ получим

```
>> syms x
>> int(sin(x)/x,x)
ans =
sinint(x)
```

Первообразной является так называемый интегральный синус

$$Si(x) = \int_0^x \frac{\sin t}{t} dt,$$

информацию о котором можно получить, введя из командной строки **doc sinint**.

Попробуем теперь вычислить определенные интегралы

$$\int_0^1 e^{\sin x} dx \text{ и } \int_0^1 \frac{\sin x}{x} dx.$$

```
>> syms x
>> I=int(exp(sin(x)),x,0,1)
```

```
I =
int(exp(sin(x)),x = 0 .. 1)
>> vpa(ans,15)
ans =
1.63186960841805
>> int(sin(x)/x,x,0,1)
ans =
sinint(1)
>> vpa(ans,15)
ans =
.946083070367183
```

Для проверки вычислим первый интеграл с помощью функции **quad**.

```
>> quad('exp(sin(x))',0,1,1e-13)
ans =
  1.63186960841805
```

Следующий пример относится к вычислению тройного интеграла:

$$\int_0^a \int_0^a \int_0^a (x^2+y^2)z\,dx\,dy\,dz.$$

Здесь можно трижды использовать функцию **int**:

```
>> syms a x y z
>> int(int(int((x^2+y^2)*z,x,0,a),y,0,a),z,0,a)
ans =
1/3*a^6
```

Вычислим тройной интеграл

$$\int_0^\pi \int_0^\pi \int_0^\pi \sin(x^2+y^2+z^2)\,dx\,dy\,dz,$$

который не выражается в элементарных функциях (в этом легко убедиться, убрав точку с запятой в операторе S=int(int(int(sin(x^2+y^2+z^2),x,0,pi),y,0,pi),z,0,pi);).

```
>> syms a x y z
>> S=int(int(int(sin(x^2+y^2+z^2),x,0,pi),y,0,pi),z,0,pi);
>> vpa(S,5)
```

```
ans =
.28051
```

5.11 Функция разложения выражения в ряд Тейлора – taylor

В задачах аппроксимации и приближения функций *f(x)* важное место занимает их разложение в ряд Тейлора в окрестности точки *a*:

$$f(x) = \sum_{n=0}^{\infty}(x-a)^n \frac{f^{(n)}(a)}{n!}.$$

Частным случаем этого ряда при *a=0* является ряд Маклорена:

$$f(x) = \sum_{n=0}^{\infty}x^n \frac{f^{(n)}(0)}{n!}.$$

Для получения разложения аналитической функции *f* в ряд Тейлора (и Маклорена) служит функция **taylor**:

taylor(f) – возвращает шесть членов ряда Маклорена (ряд Тейлора в точке *x=0*). В общем случае, когда небходимо задать число членов разложения *n*, и точку *a*, относительно которой ищется разложение, используется функция **taylor(f,n,x,a)**.

Примеры:

```
>> syms x
>> taylor(cos(x))
ans =
1-1/2*x^2+1/24*x^4
>> taylor(cos(x),5,x,2)
ans =
cos(2)-sin(2)*(x-2)-1/2*cos(2)*(x-2)^2+1/6*sin(2)*(x-
2)^3+1/24*cos(2)*(x-2)^4
```

Получим разложение в степенной ряд по степеням *x* «функции ошибок», определяемой равенством

$$Erf(x)= \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

и не выражающейся в конечном виде через элементарные функции.

Решение:

```
>> syms x t
>> Pi=sym('pi');
>> taylor(2/sqrt(Pi)*int(exp(-t^2),t,0,x))
ans =
2/pi^(1/2)*x-2/3/pi^(1/2)*x^3+1/5/pi^(1/2)*x^5
>> [m n]=simple(ans)
m =
1/15*x*(30-10*x^2+3*x^4)/pi^(1/2)
>> pretty(m)
                                  2      4
                          x (30 - 10 x  + 3 x  )
                  1/15 ------------------------
                                   1/2
                                 pi
```

Команда **taylortool** приводит к появлению окна приложения, изображенного на рис.5.1.

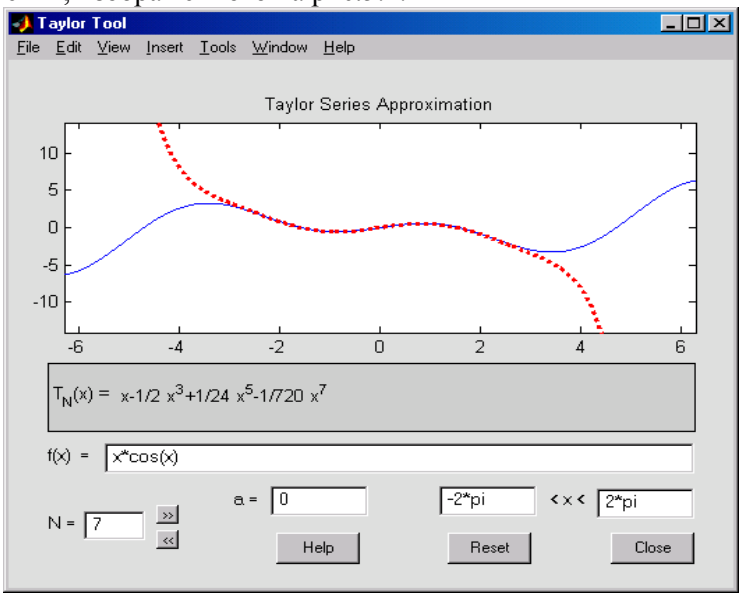


Рис.5.1.

Пользователь может вводить формулы различных функций в строке *f(x)*= и

исследовать приближение функции на произвольном интервале отрезком ряда Тейлора требуемой длины.

7.12 Функция вычисления сумм рядов – symsum

В математическом анализе часто приходится вычислять сумму значений функции *f(i)* при условии, что аргумент *i* этой функции принимает целочисленные значения, изменяющееся в замкнутом интервале *[a;b]*, т.е.

$$Sum = \sum_{i=a}^{b} f(i).$$

Такие суммы принято называть *конечными*. При $b = \infty$ можно говорить о *бесконечной* сумме (в смысле бесконечного числа членов ряда).

Для аналитического вычисления суммы ряда служит функция

symsum(f, i, a, b), возвращающая сумму ряда $\sum_{i=a}^{b} f(i)$.

Справка по функции – **doc symsum**.

Рассмотрим применение этой функции на примерах.

Найти сумму числового ряда:

$$\sum_{n=1}^{\infty} \frac{1}{n^2}..$$

Решение:

```
>> syms n
>> symsum(1/n^2,n,1,inf)
ans =
1/6*pi^2
```

Таким образом,

$$\sum_{n=1}^{\infty}\frac{1}{n^2}=\frac{\pi^2}{6}.$$

Найти сумму числового ряда:

$$\sum_{n=1}^{\infty}\frac{(-1)^n}{n}.$$

Решение:

```
>> syms n
>> symsum((-1)^n/n,n,1,inf)
ans =
-log(2)
```

В результате получили

$$\sum_{n=1}^{\infty}\frac{(-1)^n}{n}=-ln2.$$

Найти сумму числового ряда:

$$\sum_{n=1}^{\infty}\frac{1}{n^3}..$$

Решение:

```
>> syms n
>> symsum(1/n^3,n,1,inf)
ans =
zeta(3)
>> vpa(ans,8)
ans =
1.2020569
```

Результат выражается через дзета – функцию Римана (справка по этой функции – **doc zeta**) и равен

$$\sum_{n=1}^{\infty}\frac{1}{n^3} = zeta(3) \approx 1{,}2020569.$$

Найти сумму числового ряда:

$$\sum_{n=1}^{\infty}\frac{cosn\pi}{n^2}.$$

Решение:

```
>> syms s n
>> symsum(cos(n*pi)/n^2,n,1,inf)
ans =
-hypergeom([1, 1, 1],[2, 2],-1)
>> vpa(ans,8)
ans =
-.82246703
```

Результат выражается через гипергеометрическую функцию (справка по этой функции – **doc hypergeom**) и равен

$$\sum_{n=1}^{\infty}\frac{cosn\pi}{n^2} = -hypergeom([1,\ 1,\ 1],\ [2,\ 2],\ -1) \approx -0{,}82246703.$$

Если в слагаемые входит знак факториала, то в этом случае следует использовать функцию **sym.**

Найдем сумму функционального ряда:

$$\sum_{k=0}^{\infty}\frac{x^k}{k!}.$$

Решение:

```
>> syms x k
>> symsum(x^k/sym('k!'), k, 0,inf)
ans =
exp(x)
```

Таким образом,

$$\sum_{k=0}^{\infty}\frac{x^k}{k!}=e^x.$$

Найдем сумму числового ряда:

$$\sum_{n=1}^{\infty}\frac{cosn}{n^2}.$$

Решение:

```
>> syms n
>> symsum(cos(n)/n^2,n,1,inf)
ans =
sum(cos(n)/n^2,n = 1 .. inf)
>> vpa(ans,8)
ans =
sum(cos(n)/n^2,n = 1 .. inf)
```

Результат свидетельствует о том, что среда **MATLAB** не нашла сумму ряда.

Найдем конечную сумму:

$$\sum_{k=0}^{10}k^2.$$

Решение:

```
>> syms k
>> symsum(k^2,k,0,10)
ans =
385
```

Сумма равна *385*.

Найдем конечную сумму с переменным верхним пределом *n:*

$$\sum_{k=0}^{n}k^2.$$

Решение:

```
>> syms k n
>> symsum(k^2,k,0,n)
ans =
1/3*(n+1)^3-1/2*(n+1)^2+1/6*n+1/6
>> pretty(ans)
                    3              2
        1/3 (n + 1)  - 1/2 (n + 1)  + 1/6 n + 1/6
```

Найдем конечную сумму с переменным верхним пределом *n:*

$$\sum_{k=1}^{n} cos\frac{k\pi}{2}.$$

Решение:

```
>> syms k n
>> symsum(cos(k*pi/2),k,1,n)
ans =
1/2*sin(1/2*(n+1)*pi)-1/2*cos(1/2*(n+1)*pi)-1/2
simple(ans)
ans =
1/2*cos(1/2*n*pi)+1/2*sin(1/2*n*pi)-1/2
```

Для аналитического вычисления произведения ряда служит функция **product**. Справка по функции – **mhelp product**.

5.13 Решение алгебраических уравнений и систем– функция solve

Для решения систем алгебраических уравнений и одиночных уравнений служит функция **solve**:

- **solve(expr1, expr2,..., exprN, var1, var2,..., varN)** – возвращает значения переменных *varI*, при которых соблюдаются равенства, заданные выражениями *exprI*. Если в выражениях не используются знаки равенства, то полагается, что *exprI=0.*

Результат может быть возвращен в следующих формах:

● для одного уравнения и одной переменной решение возвращается в виде одномерного или многомерного массива ячеек;

● при одинаковом числе уравнений и переменных решение возвращается в упорядоченном по именам переменных виде;

Функция **solve** позволяет найти не только вещественные, но и комплексные решения систем алгебраических уравнений и одиночных уравнений. Справка по функции – **doc solve**.

Рассмотрим возможности этой функции на примерах.

Решим уравнение $x^3-1=0$.

Решение:

```
>> syms x,y=x^3-1;S=solve(y)
S =
[                1]
[ -1/2+1/2*i*3^(1/2)]
[ -1/2-1/2*i*3^(1/2)]
```

В результате получены три разных значения корня $x_1=1$, $x_2=\frac{-1+i\sqrt{3}}{2}$, $x_3=\frac{-1-i\sqrt{3}}{2}$, которые хранятся соответственно в элементах *S(1), S(2), S(3)* массива *S*.

Для проверки правильности решения подставим полученные корни в приведенном порядке вместо *x* в исходное выражение $y=x^3-1$.

```
>> subs(y,x,S)
ans =
[                    0]
[ (-1/2+1/2*i*3^(1/2))^3-1]
[ (-1/2-1/2*i*3^(1/2))^3-1]
>> [m,n]=simple(ans)
m =
[ 0]
[ 0]
[ 0]
```

Выражение $y=x^3-1$ принимает значение 0 при подстановке любого из корней, т.е. $x_1=1$, $x_2=\frac{-1+i\sqrt{3}}{2}$, $x_3=\frac{-1-i\sqrt{3}}{2}$, являются точными корнями уравнения $x^3-1=0$.

Функция **roots** (см. раздел 6.1) нашла бы только приближенные значения корней уравнения $x^3-1=0$. В общем случае полиномиальное уравнение степени выше 4 не может иметь точного решения, выраженного с помощью радикалов.

Функция **solve** позволяет решать уравнения, представленные в аналитическом виде.

Решим квадратное уравнение $ax^2+bx+c=0$.

Решение:

```
>> S=solve('a*x^2+b*x+c=0',x)
S =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

Это известные выражения корней $x_{1,2} = \frac{-b\pm\sqrt{b^2-4ac}}{2}$ квадратного уравнения $ax^2+bx+c=0$ (обратная теорема Виета). Точно также можно выразить с помощью радикалов решения кубического уравнения $ax^3+bx^2+cx+d=0$, хотя эти выражения достаточно сложные.

Решим трансцендентное уравнение $x^{\ln x+1}-1=0$.

Решение:

```
>> syms x
>> S=solve('x^(log(x)+1)-1',x)
```

```
S =
[  exp(0)]
[ exp(-1)]
```

Проверка:

```
>> subs(x^(log(x)+1)-1,x,S)
ans =
[ 0]
[ 0]
```

В данном случае найдены точные решения $x_1=1$, $x_2=e^{-1}$.

Найдем решение трансцендентного уравнения

$lnx+3-x=0$.

Решение:

```
>> solve('log(x)+3-x=0')
ans =
[    -lambertw(-exp(-3))]
[ -lambertw(-1,-exp(-3))]
>> vpa(ans,8)
ans =
[ .52469097e-1]
[    4.5052415]
```

Найдены приближенные решения $x_1=0,0524691$ *и* $x_2=4,5052415$, которые выражаются через функцию Ламберта.

Решения этой системы были получены в разделе 6.2 с помощью функции **fzero**. Но в данном случае функция **solve** находит оба корня одновременно, причем с любой степенью точности. При этом не требуется графически определять интервалы изоляции корней.

Решение любого трансцендентного уравнения, в том числе и тригонометрического (см. раздел 7.17), достаточно сложная и серьезная проблема. Иногда простое трансцендентное уравнение может и не решаться в **MATLAB** функцией **solve**, или она может выдать не все решения.

Решим трансцендентное уравнение:

$sinx+lnx+e^x-1=0$.

Решение:

```
>> syms x
>> Y=sin(x)+log(x)+exp(x)-1;
```

```
>> S=solve(Y);
>> vpa(S,5)
ans =
-3.0553-1.7145*i
>> subs(Y,S)
ans =
-.8e-31-.1e-30*i
```

Найдя комплексный корень уравнения $x_1 = -3{,}0553 - 1{,}7145i$, функция **solve** не определила вещественный корень. С помощью функции **ezplot** (см. раздел 7.16) графически определяем (рис.5.2), что он находится вблизи значения *0,4*:

```
>> ezplot('sin(x)+log(x)+exp(x)-1',[0 ,1,-1, 3])
>> grid
```

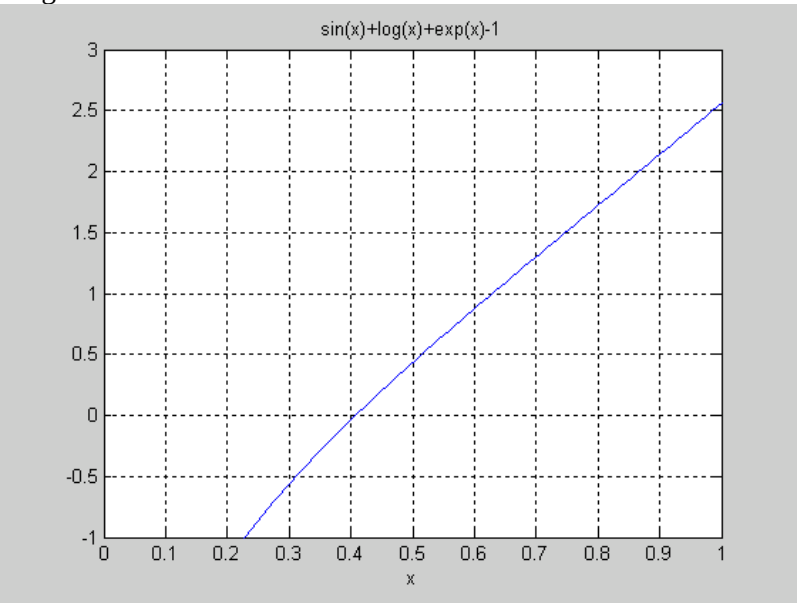


Рис.5.2

Вещественный корень со стартовым приближением *0,4* найдем с помощью фукциии **fzero** (см. раздел 6.2):

```
>> format long
>> [X,f]=fzero('sin(x)+log(x)+exp(x)-1',0.4)
X =
```

```
   0.40716029855672
f =
   -2.220446049250313e-016
```

Итак, вещественный корень $x_2=0,4072$.

Перейдем теперь к системам уравнений.

Решим систему уравнений $\begin{cases} x+y=3; \\ xy=4. \end{cases}$

Решение:

```
>> syms x y
>> Y1=x+y-3;
>>  Y2=x*y^2-4;
>> S=solve(Y1,Y2,x,y)
S =
    x: [3x1 sym]
    y: [3x1 sym]
```

```
disp([S.x S.y])
[  4, -1]
[  1,  2]
[  1,  2]
```

Получили три решения $(x_1\ y_1)=(4;-1)$ и $(x_2,\ y_2)=(1;2)$ (второе – кратности *2*), причем $(x_1;y_1)$ хранится в *[S.x(1) S.y(1)]*, а $(x_2;y_2)$ – в *[S.x(2) S.y(2)]*:

```
>> disp([S.x(1) S.y(1)])
[  4, -1]
>> disp([S.x(2) S.y(2)])
[ 1, 2]
```

Для проверки подставим в выражения $Y1=x+y-3$ *и* $Y2=xy^2-4$ вначале первое решение, а затем второе:

```
>> disp(subs([Y1 Y2],[x y],[S.x(1) S.y(1)])
[ 0, 0]
>> disp(subs([Y1 Y2],[x y],[S.x(2) S.y(2)])
[ 0, 0]
```

Как видим, найдены точные решения, т.к. выражения *Y1* и *Y2* при их подстановке обратились в *0*.

Функция **solve** позволяет решать системы уравнений, представленные в аналитическом виде.

Решим систему уравнений

$$\begin{cases} \frac{a+b}{x+y}+\frac{b+c}{y+z}-\frac{c+a}{z+x}=1; \\ \frac{a+b}{x+y}-\frac{b+c}{y+z}+\frac{c+a}{z+x}=1; \\ -\frac{a+b}{x+y}+\frac{b+c}{y+z}+\frac{c+a}{z+x}=1. \end{cases}$$

Решение:

```
>> syms a b c x y z
>> Y1=(a+b)/(x+y)+(b+c)/(y+z)-(c+a)/(z+x)-1;
>> Y2=(a+b)/(x+y)-(b+c)/(y+z)+(c+a)/(z+x)-1;
>> Y3=-(a+b)/(x+y)+(b+c)/(y+z)+(c+a)/(z+x)-1;
>> S=solve(Y1,Y2,Y3,x,y,z)
S =
   x: [1x1 sym]
   y: [1x1 sym]
```

```
   z: [1x1 sym]
>> disp([S.x S.y S.z])
[ a, b, c]
```

Проверим найденное решение *(x;y;z)=(a;b;c)* подстановкой:

```
>> subs([Y1 Y2 Y3],[x y z],[S.x S.y S.z])
ans =
[          0,          0, (-a-b)/(a+b)+1]
>> disp(simplify(ans))
[ 0, 0, 0]
```

Убеждаемся, что решение найдено верно.

Если **MATLAB** не может найти ни одного решения, то функция **solve** возвращает сообщение *[empty sym]*. Это означает, что либо решения не существует, либо **MATLAB** не удалось его найти. Иногда системе **MATLAB** можно помочь, преобразовав уравнение или систему к эквивалентному виду.

Например, уравнение $ln(4-2x)+x^2-2=0$ имеет эквивалентный вид $e^{2-x^2}+2x-4=0$. Легко убедиться, что для из них функция **solve** выдаст свой вещественный корень. Это будут разные корни, но каждый из них удовлетворяет исходному уравнению. Существует и третий вещественный корень, который можно найти с помощью функции **fzero**.

Решим систему трансцендентных уравнений.

$$\begin{cases} 3^y 9^x = 81; \\ \lg(y+x)^2 - \lg x = 2\lg 3. \end{cases}$$

Решение:

```
>> syms x y
>> Y1=3^y*9^x-81;
>> Y2=log10((y+x)^2)-log10(x)-2*log10(3);
>> S=solve(Y1,Y2,x,y)
S =
   x: [4x1 sym]
   y: [4x1 sym]S =
>> R=[S.x S.y];
>> disp(vpa(R,10))
[  16.00000002, -28.00000004]
[  16.00000002, -3.999999992]
```

[1.000000000, -3.999999996]
[1.000000000, 1.999999996]

В результате получены четыре решения. Однако, верными являются только

первое и последнее из них. Это легко проверить подстановкой:

```
>> disp(vpa(subs([Y1,Y2],[x y],[S.x(1) S.y(1)]),15))
[ .1e-12, .15e-13]
>> disp(vpa(subs([Y1,Y2],[x y],[S.x(2) S.y(2)]),15))
[ 22876792454891.6, .25e-13]
>> disp(vpa(subs([Y1,Y2],[x y],[S.x(3) S.y(3)]),15))
[ -80.8888888888889, .31e-13]
>> disp(vpa(subs([Y1,Y2],[x y],[S.x(4) S.y(4)]),15))
[ -.71e-11, .18e-13]
```

Лишь при подстановке первого и последнего решений *[Y1,Y2]=[0,0]*. В остальных случаях *[Y1,Y2]≠[0,0]*.

Исходную систему можно преобразовать и привести к более простой

$$\frac{(y+x)^2}{x}\begin{cases} y+2x=4; \\ =9. \end{cases}$$

Тогда ее решения будут точными.

Решение:

```
>> syms x y
>> S=solve('y+2*x=4','(y+x)^2/x=9',x,y);
>> [S.x S.y]
ans =
[ 1, 2]
[ 16, -28]
```

С помощью функции **fsolve** можно найти одно вещественное решение системы нелинейных уравнений

$$\begin{cases} x+x^2-2xz-0,1=0; \\ y-y^2+3xz+0,2=0; \\ z+z^2+2xy-0,3=0. \end{cases}$$

Решим эту систему с помощью функции **solve**:

```
>> syms x y z
```

```
>> Y1=x+x^2-2*y*z-.1;
>> Y2=y-y^2+3*x*z+.2;
>> Y3=z+z^2+2*x*y-.3;
>> S=solve(Y1,Y2,Y3,x,y,z);
>> R=[S.x S.y S.z];
>> disp(vpa(R,6))
[    -.541941+.626019e-1*i,          -.179057-.433417*i,
.148543-.344892*i]
[    -.541941-.626019e-1*i,          -.179057+.433417*i,
.148543+.344892*i]
[         .128241e-1,          -.177801,          .244688]
[          -1.08804,          -.130325,          .161425e-1]
[         .578802e-1,          .156279e-1,          -1.24040]
[        .374678+.356411*i,          .353227-.580416*i,          -
.281751+.419593*i]
[    .374678-.356411*i,    .353227+.580416*i,    -.281751-
.419593*i]
[         .121093,          1.17493,          .152166e-1]
```

Получено *8* решений, *4* из которых – вещественные (с *3*-го по *5*-е и *8*-е). В разделе 6.2 с помощью функции **fsolve** было найдено *4*-е решение.

Проверим все *8* решений подстановкой с помощью цикла **for**:

```
>> for i=1:8
T=subs([Y1 Y2 Y3],[x y z],[S.x(i) S.y(i) S.z(i)]);
disp(vpa(T,6))
end
[ -.5e-31-.1471e-30*i,  -.32e-30+.71e-30*i,  .25e-30+.103e-
29*i]
[ -.5e-31+.1471e-30*i,  -.32e-30-.71e-30*i,  .25e-30-.103e-
29*i]
[  .12e-30,   .9e-31, -.13e-30]
[  .8e-31, .14e-30, -.3e-31]
[   .2e-31,  -.6e-31, -.14e-30]
[   .637e-29+.829e-29*i,   -.754e-29+.15e-30*i,   .709e-29-
.346e-29*i]
```

[.637e-29-.829e-29*i, -.754e-29-.15e-30*i, .709e-29+.346e-29*i]

[.17422e-27, -.1012e-28, .12735e-27]

Видно, что *[Y1 Y2 Y3]≈[0 0 0]* при подстановке любого из решений. Отсюда можно сделать вывод, что все *8* решений найдены с высокой степенью точности.

5.14 Решение дифференциальных уравнений – функция dsolve

Для решения обыкновенных дифференциальных уравнений (систем уравнений) **MATLAB** имеет функцию

r = dsolve('eq1','eq2',...,'cond1','cond2',...,'v').

Она возвращает аналитическое решение дифференциальных уравнений *eq1*, *eq2*,..., использующих *v* как независимую переменную, с граничными и (или) начальными условиями *cond1*, *cond2*,... . По умолчанию независимой переменной считается переменная *t*, обычно обозначающая время. Если в выражениях *eqI (condI)* не используется знак равенства, то полагается, что *eqI (condI)=0.*

Символ *D* обозначает производную по независимой переменной, то есть *d/dt*, при этом *D2* означает d^2/dt^2 и т.д. Имя независимой переменной не должно начинаться с буквы *D*.

Начальные условия задаются в виде равенств *'y(a)=b'* или *'Dy(a)=b'* , где *y* – зависимая переменная, *a* и *b* – константы, которые могут быть и символьными. Могут быть символьными и константы в уравнениях. Если число начальных условий меньше числа зависимых переменных, то в решении будут присутствовать произвольные постоянные *C1*, *C2* и т.д. Формы вывода результата такие же, как и для функции **solve**. Справка по функции – **doc dsolve**.

Решим дифференциальные уравнения

1) x'' = -2x', 2) y'' = -ax+y', y(0)=b, 3) $y^{(4)} - y = 5e^x sinx + x^4$*, 4) y''+4y'+3y=cost, y(0)=1, y'(0)=0.*

Решения *3*-го и *4*-го уравнений проверим подстановкой.

Решение:

```
>> dsolve('D2x=-2*x')
```

```
ans =
C1*cos(2^(1/2)*t)+C2*sin(2^(1/2)*t)
>> dsolve('D2y=-a*x+y','y(0)=b','x')
ans =
a*x+C1*sinh(x)+b*cosh(x)
>> syms x
>> S=dsolve('D4y-y-5*exp(x)*sin(x)-x^4','x')
S =
149/208*cos(x)*exp(x)-24-x^4-57/104*exp(x)*sin(x)-
21/26*exp(x)*sin(x)*cos(x)^2-
1/4*sin(x)*exp(x)*sin(2*x)+1/2*sin(x)*exp(x)*cos(2*x)-
41/52*cos(x)^3*exp(x)+15/208*cos(3*x)*exp(x)-
5/104*sin(3*x)*exp(x)+C1*exp(x)+C2*sin(x)+C3*cos(x)+C4*exp
(-x)
>> [R,HOW]=simple(S)
R =
-24-x^4-
exp(x)*sin(x)+C1*exp(x)+C2*sin(x)+C3*cos(x)+C4*exp(-x)
```

Проверка *3*-го решения:

```
>> diff(R,x,4)-R-5*exp(x)*sin(x)-x^4
ans =
0
>> syms x
>> S=dsolve('D2y+4*Dy+3*y=cos(t)','y(0)=1','Dy(0)=0','t')
S =
1/10*cos(t)+1/5*sin(t)-7/20*exp(-3*t)+5/4*exp(-t)
```

Проверка *4*-го решения:

```
>> syms t
>> diff(S,t,2)+4*diff(S,t)+3*S
ans =
cos(t)
```

Проверка выполнения начальных условий *4*-го решения:

```
>> subs(S,t,0)
ans =
     1
>> subs(diff(S,t),t,0)
```

```
ans =
     0
```

Решим систему линейных дифференциальных уравнений

$$\begin{cases} \frac{dx}{dt} = y; \\ \frac{dy}{dt} = -x. \end{cases}$$

```
>> S=dsolve('Dx = y', 'Dy = -x')
S =
   x: [1x1 sym]
   y: [1x1 sym]
>> disp([S.x, S.y ])
[  cos(t)*C1+sin(t)*C2, -sin(t)*C1+cos(t)*C2]
```

Решим систему нелинейных дифференциальных уравнений с начальными условиями и проверим решение.

$$\begin{cases} \frac{dx}{dt} = y, \ x(0)=1; \\ \frac{dy}{dt} = -x, \ y(0)=2. \end{cases}$$

```
>> S=dsolve('Dx = y', 'Dy = -x','x(0)=1','y(0)=2')
S =
   x: [1x1 sym]
   y: [1x1 sym]
>> [S.x S.y]
ans =
[  cos(t)+2*sin(t), -sin(t)+2*cos(t)]
```

То есть $x=cost+2*sint$, $y=-sint+2*cost$.

Проверка решения:

```
syms t
>> diff(S.x,t)-S.y
ans =
0
>> diff(S.y,t)+S.x
ans =
0
```

Проверка выполнения начальных условий:

```
>> subs(S.x,t,0)
ans =
    1
>> subs(S.y,t,0)
ans =
    2
```

Функция **dsolve** не позволяет получить аналитическое решение дифференциального уравнения произвольного вида. Например, решая уравнения Ван-дер-Поля $y''-(1-y^2)y'+y=0$, $y(0)=2$, $y'(0)=0$, получим

```
>> dsolve('D2y-(1-y^2)*Dy+y=0','y(0)=2','Dy(0)=0')
ans =
[ empty sym ]
```

Решение не найдено.

В некоторых случаях **dsolve** возвращает решение через специальные функции, например, для уравнения Бесселя $x^2y''+xy'+(x^2-v^2)y=0$ получим

```
>> dsolve('x^2*D2y+x*Dy+(x^2-v^2)*y=0','x')
ans =
C1*besselj(v,x)+C2*bessely(v,x)
```

Решение выражается через функции Бесселя, информацию о которых можно получить по справке **doc besselj**.

5.15 Прямое и обратное преобразования Лапласа – функции laplace, ilaplace

Преобразование Лапласа любой комплексной функции *f(t)* действительной переменной *t* имеет вид:

$$L(s) = \int_0^\infty f(t)e^{-st}dt.$$

Функцию *f(t)* принято назывть оригиналом, а функцию *L(s)* – изображением. Функция *f(t)* должна удовлетворять следующим условиям:

a) f(t) является непрерывной для всех значений *t*, принадлежащих области определения. (Допускается наличие разрывов первого рода в конечном числе точек,

расположенных на интервалах конечной длины. Количество таких интервалов должно быть конечным числом);

б) $f(t)=0$ при $t<0$;

в) существуют числа $M>0$ и $p\geq 0$ такие, что для всех t $|f(t)|<Me^{pt}$ (p называется показателем роста $|f(t)|$).

Некоторые простейшие) преобразования Лапласа приведены в таблице 5.1.

Таблица 5.1. Некоторые преобразования Лапласа

$f(t)$	$L(s)=\int_0^\infty f(t)e^{-st}dt.$
1	s^{-1}
e^{-at}	$(s+a)^{-1}$
$\sin at$	$a(s^2+a^2)^{-1}$
t^n	$n!s^{-n-1}$
$e^{-at}\cos wt$	$\frac{s+a}{(s+a)^2+w^2}$
$t^n e^{-at}$	$\frac{n!}{(s+a)^{n+1}}$

В **MATLAB** преобразование Лапласа функции $f(t)$ осуществляется с помощью встроенной функции **laplace(F,t,s)**.

Найдем с помощью этой встроенной функции изображения заданных в таблице 5.1 оригиналов $f(t)$:

```
>> syms a t w s
>> n=sym('n','positive');
>> laplace(1,t,s)
ans =
1/s
>> laplace(exp(-a*t),t,s)
ans =
1/(s+a)
>> laplace(sin(a*t),t,s)
ans =
a/(s^2+a^2)
```

```
>> laplace(t^n,t,s)
ans =
s^(-n-1)*gamma(n+1)
>> laplace(exp(-a*t)*cos(w*t),t,s)
ans =
(s+a)/((s+a)^2+w^2)
>> laplace(t^n*exp(-a*t),t,s)
ans =
gamma(n+1)*(s+a)^(-n-1)
```

Полученные изображения совпадают с табличными, если учесть, что

gamma(n+1)=n! для целых $n \geq 0$.

Найдем изображение функции

$f(t)=e^{-2t}\sin 2t \cos 3t$,

которое нельзя найти непосредственно с помощью таблиц преобразования Лапласа.

```
>> syms t s
>> laplace(exp(-2*t)*sin(2*t)*cos(3*t),t,s)
ans =
5/2/((s+2)^2+25)-1/2/((s+2)^2+1)
>> pretty(ans)
                    1                 1
5/2 ---------------- - 1/2 ------------
                   2                 2
      (s + 2)  + 25     (s + 2)  + 1
>> factor(ans)
ans =
2*(s^2+4*s-1)/(s^2+4*s+29)/(s^2+4*s+5)
>> pretty(ans)
                        2
                   s  + 4 s - 1
2 -----------------------------
                 2                 2
  (s  + 4 s + 29) (s  + 4 s + 5)
```

Найдем изображение функции $f(t)=\frac{1}{t}$.

```
>> laplace(1/t,t,s)
```

```
ans =
laplace(1/t,t,s)
```

Эта функция не является оригиналом ввиду того, что не выполнено условие *а)*. Следовательно, изображения этой функции не существует, и **MATLAB** не смог его найти.

Существуют и другие модификации функции **laplace** (справка – **doc laplace**).

Обратное преобразование Лапласа имеет вид:

$$f(t) = \frac{1}{2\pi i}\int_{\delta-i\infty}^{\delta+i\infty} L(s)e^{st}ds.$$

В среде **MATLAB** обратное преобразование Лапласа функции *L(s)* vожно получить с помощью встроенной функции **ilaplace(L,s,t)**. Найдем с ее помощью оригиналы заданных в таблице 7.1 изображений *L(s)*:

```
>> syms a t w s
>> n=sym('n','positive');
>> ilaplace(1/s,s,t)
ans =
1
>> ilaplace(1/(s+a),s,t)
ans =
exp(-a*t)
>> ilaplace(a/(s^2+a^2),s,t)
ans =
a/(a^2)^(1/2)*sin((a^2)^(1/2)*t)
>> ilaplace(s^(-n-1)*gamma(n+1),s,t)
ans =
t^n
>> ilaplace((s+a)/((s+a)^2+w^2),s,t)
ans =
exp(-a*t)*cos(w*t)
>> ilaplace(gamma(n+1)*(s+a)^(-n-1),s,t)
ans =
exp(-a*t)*t^n
```

Как видно, полученные оригиналы совпадают с табличными.

Теперь найдем оригинал изображения, которое было полученно ранее: $2\frac{s^2+4s-1}{(s^2+4s+29)(s^2+4s+5)}$.

Напомним, что оригиналом в этом случае является функция

$f(t)=e^{-2t}sin2tcos3t.$

```
>> syms t s
>> ilaplace(2*(s^2+4*s-1)/(s^2+4*s+29)/(s^2+4*s+5),s,t)
ans =
1/2*exp(-2*t)*sin(5*t)-1/2*exp(-2*t)*sin(t)
>> factor(ans)
ans =
1/2*exp(-2*t)*(sin(5*t)-sin(t))
```

Поскольку $\frac{1}{2}(sin5t-sint)=sin2tcos3t$, то оригинал найден правильно.

Найдем оригинал изображения

$L(s) = \frac{e^{-2s}}{s+a}$.

```
>> syms t s a
>> ilaplace(exp(-2*s)/(s+a),s,t)
ans =
Heaviside(t-2)*exp(-a*(t-2))
```

В результате найдена фунукция Хевисайда, которая определяется следующим образом

$$f(t) = \begin{cases} e^{-a(t-2)} & (t \geq 2); \\ 0 & (t < 2). \end{cases}$$

Найдем оригинал изображения

$L(s) = \frac{s^4-1}{s^5+s}$.

```
>> syms t s
>> ilaplace((s^4-1)/(s^5+s),s,t)
ans =
-1+2*sum(1/4*exp(_alpha*t),_alpha = RootOf(_Z^4+1))
>> vpa(ans,5)
ans =
```

-1.+1.0000*exp(-
.70711*t)*cos(.70711*t)+1.0000*exp(.70711*t)*cos(.70711*t)

Когда **MATLAB** не может найти решение, то функция **ilaplace** возвращается без результата. Это означает, что, либо решения не существует, либо **MATLAB** не удалось его найти.

```
>> syms t s
>> ilaplace(exp(2*s)/(s+3)^2,s,t)
ans =
ilaplace(exp(2*s)/(s+3)^2,s,t)
```

Существуют и другие модификации функции **ilaplace** (справка – **doc ilaplace**).

5.16 Графики символьных функций – команды ezplot, ezpolar

Чтобы избавить пользователя от хлопот, связанных с построением графиков функций с помощью стандартных средств (например, команды **plot**), в пакет **Symbolic** введены довольно удобные команды класса **ezplot**:

- **ezplot(f)** – строит график символьно заданной функции *f(x)* независимой переменной *x* в интервале *[-2*pi, 2*pi]*.
- **ezplot(f,xmin,xmax)** – делает то же, но позволяет задать диапазон изменения независимой переменной *x* в интервале от *xmin* до *xmax*.
- **ezplot(f, [xmin, xmax, ymin, ymax])** – строит график функции *f(x,y)=0* для *xmin<x<xmax, ymin<y<ymax*.

Рассмотрим пример построения графика функции *sin(t)/t* (рис. 5.3):

```
>> ezplot('sin(t)/t'),grid
```

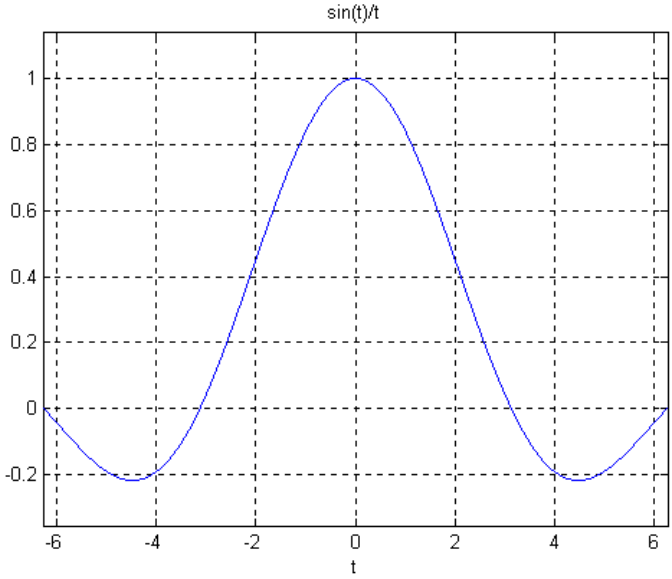


Рис.5.3

Следующая команда строит график гиперболы $u^2-v^2-1=0$ для $-3<u<3$, $3<v<-3$ (рис. 7.4):

```
>> ezplot('u^2-v^2-1',[-3, 3, 3, -3]),grid
```

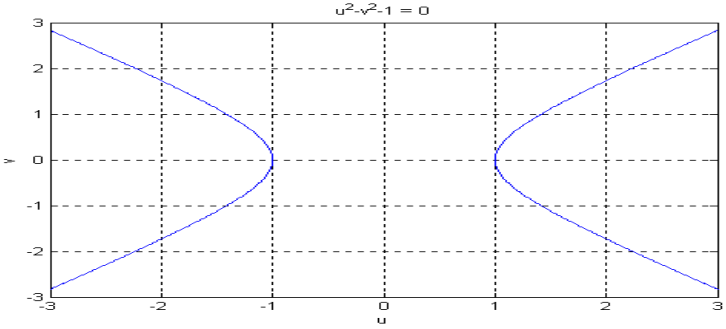


Рис. 5.4

График функции *f(t)* в полярной системе координат строит команда **ezpolar**:

ezpolar(f) – строит график функции *f(t)* при изменении угла *t* от *0* до *2π*;

ezpolar(f,[a b]) – строит график функции *f(t)* при изменении угла *t* от *a* до *b*.

Пример (рис.5.5):

```
>> ezpolar('cos(3*t)')
```

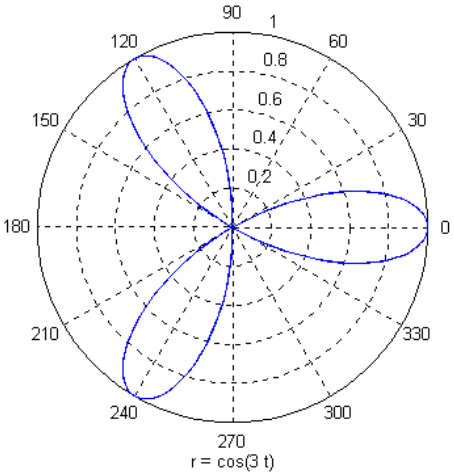


Рис.5.5

Помимо команд **ezplot** и **ezpolar** пакет **Symbolic** поддерживает построение графиков других типов. Так, команда **ezcontour** служит для построения контурных графиков функций вида *f(x,y)*. Похожая команда **ezcontourf** строит контурные графики с функциональной окраской областей между линиями равного уровня. Для построения трехмерных графиков параметрически заданных функций служит команда **ezplot3**. Команды **ezsurf**, **ezsurfc**, **ezmesh, ezmeshc** применяются для построения графиков поверхностей, заданных функциями двух переменных *f(x,y)*. Справка с примерами по применению любой из этих команд – **doc** <*имя команды*>.

5.17 Доступ к ресурсам ядра системы Maple

Применение возможностей системы **Maple** совместно с возможностями системы **MATLAB** придает последней особую гибкость и резко расширяет возможности решения сложных математических задач, где целесообразно объединять аналитические (символьные) методы с численными расчетами.

Пусть требуется найти аналитическое решение дифференциального уравнения

y''+2xy' + ny= 0.

Обращение к **dsolve** приводит к решению, выраженному через функции Уиттекера:

```
>> dsolve('D2y+2*x*Dy+n*y=0','x')
ans =
C1/x^(1/2)*WhittakerW(1/4*n-1/4,1/4,x^2)*exp(-
1/2*x^2)+C2/x^(1/2)*WhittakerM(1/4*n-1/4,1/4,x^2)*exp(-
1/2*x^2)
```

Для получения списка специальных математических функций **Maple**, доступных из **MATLAB**, служит команда **mfunlist**. В этом списке функции **WhittakerW** и **WhittakerM** отсутствуют, т.е непосредственно из **MATLAB** они недоступны.

Для доступа к информации о функциях **Maple** предназначена команда **mhelp**. Указание в качестве параметра имени функции

```
>> mhelp WhittakerW
```

выводит определение функций Уиттекера, варианты вызова и подробное описание с примерами использования.

Функция **maple** позволяет вычислить значение любой функции **Maple**, например

```
>> maple('WhittakerM(1,2,0.5)')
ans =
.1607
```

Приведем примеры использования некоторых стандартных функций системы **Maple** при решении в **MATLAB** следующих задач:

1) *Разложение рациональной дроби на сумму простейших дробей;*

2) *Нахождение интерполяционного полинома Лагранжа;*

3) *Решение неравенств и систем неравенств;*

4) *Разложение в ряд Тейлора функции нескольких переменных;*

5) *Решение дифференциальных уравнений с помощью степенных рядов;*

6) *Решение тригонометрических уравнений.*

1) *Разложение рациональной дроби на сумму простейших дробей*

При нахождении (вручную) неопределенного интеграла

$$I = \int \frac{(x^2+2)^2(x^3+3)dx}{(x+1)(x^2+1)^2}$$

возникает задача разложения подынтегральной дроби $\frac{(x^2+2)^2(x^3+3)}{(x+1)(x^2+1)^2}$ на сумму простейших дробей. Получить такое разложение непосредственно в **MATLAB** нельзя (функция **residue** дает несколько иные результаты). Следует использовать команду **convert** (с формой разложения **parfrac**) системы **Maple**:

```
>> maple('convert((x^2+2)^2*(x^3+3)/(x+1)/(x^2+1)^2,parfrac,x)')
ans =
x^2-x+3+9/2/(x+1)-1/2*(-7+9*x)/(x^2+1)-(-1+2*x)/(x^2+1)^2
>> pretty(sym(ans))
        2                  1          -7 + 9 x    -1 + 2 x
       x  - x + 3 + 9/2 ----- - 1/2 ---------- - -----------
                         1 + x            2               2 2
                                      1 + x        (1 + x )
```

т.е. $\frac{(x^2+2)^2(x^3+3)}{(x+1)(x^2+1)^2} = x^2 - x + 3 + \frac{9}{2}\frac{1}{x+1} - \frac{1}{2}\frac{-7+9x}{x^2+1} - \frac{-1+2x}{(x^2+1)^2}$.

2) ***Нахождение интерполяционного полинома Лагранжа***

Предположим, что некоторая функция *f(x)* задана таблицей своих значений:

	0	1		n
	0	1		n

Требуется найти интерполяционный полином Лагранжа – многочлен $L_n(x)$ степени не выше n, значения которого в точках x_k совпадают со значениями данной функции в этих точках, т.е. $L_n(x_k) = y_k$, $k=0,\ldots,n$.

Для нахождения интерполяционного полинома Лагранжа в **Maple** служит функция **interp**.

В качестве примера рассмотрим таблично заданную функцию *f(x)*:

Решение:

```
>> syms x
>> maple('interp([0,1,3,7],[5,4,2,1],x)')
ans =
1/56*x^3-1/14*x^2-53/56*x+5
```

Построим на рис. 5.7 узлы интерполяции (команда **stem**) и график найденного интерполяционного полинома Лагранжа

$$L_3(x_k) = \frac{1}{56}x^3 - \frac{1}{14}x^2 - \frac{53}{56}x + 5:$$

```
>> stem([0 1 3 7],[5 4 2 1],'fill')
>> hold
Current plot held
>> ezplot(ans,-1, 8)
```

>>grid

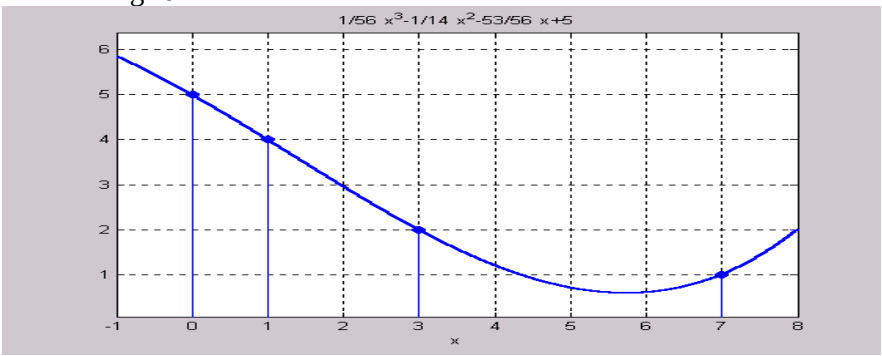


Рис. 5.7

Как видим из рис. 5.7, график найденного интерполяционного полинома Лагранжа проходит через узлы интерполирования

3) ***Решение неравенств и систем неравенств***

Для решения неравенств и систем неравенств в **Maple** служит функция **solve**. Непосредственно в **MATLAB** функция **solve** неравенства не решает.

Пример: Решить неравенство $\frac{x-2}{x+3} > 2$.

Решение:

>> maple('solve((x-2)/(x+3)>2,{x})')

ans =

{-8 < x, x < -3}

Значит, $-8<x<-3$ – решение неравенства.

Пример: Решить систему неравенств

$$\begin{cases} \frac{x-2}{x+3} \le 51; \\ \sqrt{x}(\sqrt{x}-1)<10; \\ 10x^2+4x \ge 69. \end{cases}$$

Решение:

>> maple('solve({(x-2)/(x+3)<=51,sqrt(x)*(sqrt(x)-1)<10,10*x^2+4*x>=69},x)')

ans =

{RootOf(-69+10*_Z^2+4*_Z,2.434387974463898132679677675012 1) <= x,

x < RootOf(-21*_Z+_Z^2+100,13.701562118716424343244108837311)}

>> vpa(ans,4)

ans =

{2.434 <= x, x < 13.70}

Таким образом, $2{,}434 \le x < 13{,}70$ – приближенное решение системы неравенств (с точностью до 4-х значащих цифр). Точное же решение имеет вид $\frac{-2+\sqrt{694}}{10} \le x < \frac{21+\sqrt{41}}{2}$.

4) *Разложение в ряд Тейлора функции нескольких переменных*

Для получения разложений аналитических функций нескольких переменных в ряд Тейлора (и Маклорена) в **Maple** служит функция **mtaylor**.

Разложение функции двух переменных $f(x,y)=\sin(x^2+y^2)$ в ряд Маклорена (по степеням x, y) до 8-й степени получим следующим образом:

>> maple('readlib(mtaylor):mtaylor(sin(x^2+y^2),[x,y],8)')

ans =

x^2+y^2-1/6*x^6-1/2*y^2*x^4-1/2*y^4*x^2-1/6*y^6

Разложение этой же функции в ряд Тейлора по степеням $x-1$, y) до 3-й степени получим следующим образом:

>>

maple('readlib(mtaylor):mtaylor(sin(x^2+y^2),[x=1,y],3)')

ans =

sin(1)+2*cos(1)*(x-1)+(-2*sin(1)+cos(1))*(x-1)^2+cos(1)*y^2

5) ***Решение дифференциальных уравнений с помощью степенных рядов***

Пусть требуется решить дифференциальное уравнение $y' = cosx + e^y$ с начальным условием $y(0)=1$.

```
>> dsolve('Dy=cos(x)+exp(y)','y(0)=1','x')
ans =
[ empty sym ]
```

Функция **dsolve** не нашла аналитического решения в **MATLAB**.

Известно, что решения этого дифференциального уравнения в аналитическом виде не существует. Найти разложение решения в степенной ряд (до 6-й степени по умолчанию) можно с помощью функции **dsolve** системы **Maple**.

```
>>
maple('dsolve({diff(y(x),x)=cos(x)+exp(y(x)),y(0)=1},y(x),series)')
ans =
y(x) =
series(1+(1+exp(1))*x+(1/2*exp(1)*(1+exp(1)))*x^2+(-
1/6+1/3*exp(1)*(3/2*exp(1)+exp(1)^2+1/2))*x^3+(1/2*exp(1)^3+
1/4*exp(1)^4+7/24*exp(1)^2)*x^4+(1/2*exp(1)^4+1/5*exp(1)^5+
5/12*exp(1)^3+1/120-
1/40*exp(1)+1/12*exp(1)^2)*x^5+O(x^6),x,6)
```

Имеется возможность управлять порядком разложения. Найдем разложение решения в степенной ряд до 3-й степени:

```
>>
maple('Order:=3;dsolve({diff(y(x),x)=cos(x)+exp(y(x)),y(0)=1},y(
x),series)')
ans =
Order := 3y(x) =
series(1+(1+exp(1))*x+(1/2*exp(1)*(1+exp(1)))*x^2+O(x^3),x,3)
```

6) ***Решение тригонометрических уравнений***

Пусть требуется решить уравнение $cos2x + sinx = 1$.

```
>> syms x
>> solve('cos(2*x)+sin(x)=1',x)
ans =
[     0]
```

```
[     pi]
[ 1/6*pi]
[ 5/6*pi]
```

Отметим, что непосредственно в **MATLAB** функция **solve** возвращает только значения корней, которые находятся в интервале $[-\pi;\pi]$. Для получения всех решений тригонометрических уравнений следует использовать следующие команды системы **Maple**:

```
>> maple('_EnvAllSolutions:=true');
>> maple('solve(cos(2*x)+sin(x)=1,x)')
ans =
2*pi*_Z, pi+2*pi*_Z, 1/6*pi+2*pi*_Z, 5/6*pi+2*pi*_Z
```

Здесь _Z – переменная целого типа.

ГЛАВА 6. АНАЛИЗ ДАННЫХ И СТАТИСТИКА

В данном разделе будут рассмотрены некоторые основные возможности системы MATLAB в области анализа данных и статистической обработки информации. Помимо базовых функ-ций, в системе MATLAB имеется также ряд специализированных пакетов, предназначенных для решения соответствующих задач в различных приложениях (на английском языке даны названия пакетов) :

- **Optimization** – Нелинейные методы обработки данных и оптимизация.
- **Signal Processing** – Обработка сигналов, фильтрация и частотный анализ.
- **Spline** – Аппроксимация сплайнами.
- **Statistics** – Углубленный статистический анализ, нелинейная аппроксимация и регрессия.
- **Wavelet** - Импульсная декомпозиция сигналов и изображений.

Внимание ! MATLAB выполняет обработку данных, записанных в виде двумерных массивов по столбцам ! Одномерные статистические данные обычно хранятся в отдельных векорах, причем n-мерные векторы могут иметь размерность 1x n или nx1. Для многомерных данных матрица является естественным представлением, но здесь имеются две возможности для ориентации данных. По принятому в системе MATLAB соглашению, различные переменные должны образовывать столбцы, а соответствующие наблюдения - строки. Поэтому, например, набор данных, состоящий из 24 выборок 3 переменных записывается в виде мат-рицы размера 24x3.

6. 1 Основные функции обработки данных

Перечень функций обработки данных, расположенных в директории MATLAB-а **datafun** приведен в **Приложении** 1.

Рассмотрим гипотетический числовой пример, который основан на ежечасном подсчете чис-ла машин, проходящих через три различные пункта в течении 24 часов. Допустим, результа-ты наблюдений дают следующую матрицу **count**

count =

```
11    11    9
 7    13   11
14    17   20
11    13    9
43    51   69
38    46   76
 61   132  186
75   135  180
38    88  115
28    36   55
12    12   14
 18    27   30
 18    19   29
17    15   18
 19    36   48
 32    47   10
 42    65   92
 57    66  151
 44    55   90
 114  145  257
 35    58   68
 11    12   15
 13     9   15
 10     9    7
```

Таким образом, мы имеем 24 наблюдения трех переменных. Создадим вектор времени, **t**, со-стоящий из

целых чисел от 1 до 24: **t = 1 : 24.** Построим теперь зависимости столбцов матри-цы **counts** от времени и надпишем график:

```
plot(t, count)
legend('Location 1','Location 2','Location 3',0)
xlabel('Time')
ylabel('Vehicle Count') grid on
```

где функция **plot(t, count)** строит зависимости трех векторов-столбцов от времени; функция **legend('Location 1','Location 2','Location 3',0)** показывает тип кривых; функции **xlabel** и **ylabel** надписывают координатные оси, а **grid on** выводит координатную сетку. Соответствующий график показан ниже.

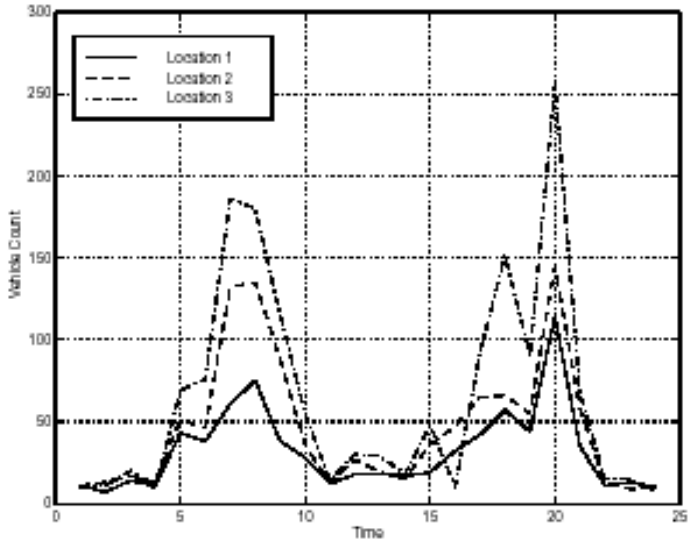


Рис. 6.1

Применим к матрице **count** функции **max** (максимальное значение), **mean** (среднее значение) и **std** (стандартное, или среднеквадратическое отклонение).

```
mx = max(count)
 mu = mean(count)
 sigma = std(count)
```

В результате получим

```
                                    mx =
114    145    257
                                   mu =
            32.00    46.5417    65.5833
                                 sigma =
             25.3703    41.4057   68.0281
```

где каждое число в строке ответов есть результат операции вдоль соответствующего столбца матрицы **count.** Для определения индекса максимального или минимального элемента нужно в соответствующей функции задать второй выходной параметр. Например, ввод

```
[mx,indx] = min(count)
```

```
                                    mx =
7      9      7
                                   indx =
  2     23     24
```

показывает, что наименьшее число машин за час было зарегестрировано в 2 часа для первого пункта наблюдения (первый столбец) и в 23 и 24 чч. для остальных пунктов наблюдения.

Вы можете вычесть среднее значение из каждого столбца данных, используя внешнее произ-ведение вектора, составленного из единиц и вектора **mu** (вектора средних значений)

```
e = ones(24, 1)
x = count - e*mu
```

Перегруппировка данных может помочь вам в оценке всего набора данных. Так, использование в системе MATLAB в качестве единственного индекса матрицы двоеточия, приводит к представлению этой матрицы как одного длинного вектора, составленного из ее столбцов. Поэтому, для нахождения минимального значения всего множества данных можно ввести

min(count(:))

что приводит к результату

ans =

7

Запись **count(:)** в данном случае привела к перегруппировке матрицы размера 24x3 в вектор-столбец размера 72x1.

Матрица ковариаций и коэффициенты корреляции

Для статистической обработки в MATLAB-е имеются две основные функции для вычисле-ния ковариации и коэффициентов корреляции:

- **cov** – В случае вектора данных эта функция выдает дисперсию, то есть меру распреде-
  ления (отклонения) наблюдаемой переменной от ее среднего значения. В случае
  матриц это также мера линейной зависимости между отдельными переменными,
  определяемая недиагональными элементами.
- **corrcoef** – Коэффициенты корреляции – нормализованная мера линейной вероятност-ной зависимости между перменными.

Применим функцию **cov** к первому столбцу матрицы **count**

cov(count(:,1))

Результатом будет дисперсия числа машин на первом пункте наблюдения

ans =

643.6522

Для массива данных, функция **cov** вычисляет матрицу ковариаций. Дисперсии столбцов мас-сива данных при этом расположены на главной диагонали матрицы ковариаций. Остальные элементы матрицы характеризуют ковариацию между столбцами исходного массива. Для матрицы размера mxn, матрица ковариаций имеет размер n-by-n и является симметричной, то есть совпадает с транспонированной.

Функция **corrcoef** вычисляет матрицу коэффициентов корреляции для массива данных, где каждая строка есть наблюдение, а каждый столбец – переменная. Коэффициент корреляции – это нормализованная мера линейной зависимости между двумя переменными. Для некор-релированных (линейно-независимых) данных коэффициент корреляции равен нулю; экива-лентные данные имеют единичный коэффициент корреляции. Для матрицы mxn, соответст-вующая матрица коэффициентов корреляции имеет размер nxn. Расположение элементов в матрице коэффициентов корреляции аналогично расположению элементов в рассмотренной выше матрице ковариаций. Для нашего примера подсчета количества машин, при вводе

corrcoef(count)

получим

ans =

1.0000 0.9331 0.9599
0.9331 1.0000 0.9553
0.9599 0.9553 1.0000

Очевидно, здесь имеется сильная линейная корреляция между наблюдениями числа машин в трех различных точках, так как результаты довольно близки к единице.

Конечные разности

MATLAB предоставляет три функции для вычисления конечных разностей.

Функция	Описание
diff	Разность между двумя последовательными элементами вектора. Приближенное дифференцирование.
gradient	Приближенное вычисление градиента функции.
del2	Пятиточечная аппроксимация Лапласиана.

Функция **diff** вычисляет разность между последовательными элементами числового вектора, то есть **diff(X)** есть **[X(2) -X(1) X(3) -X(2) ... X(n) -X(n-1)]**. Так, для вектора **A**,

```
A = [9   -2   3   0   1   5   4];
diff(A)
```

MATLAB возвращает

```
ans =
-11   5   -3   1   4   -1
```

Помимо вычисления первой разности, функция **diff** является полезной для определения определенных характеристик вектора. Например, вы можете использовать **diff** для определения, является ли вектор монотонным (значения элементов или всегда возрастают или убывают), или имеет ли он равные приращения и т.д. Следующая таблица описывает несколько различных путей использования функции **diff** с одномерным вектором x.

Применение (тест)	Описание
diff(x) == 0	**Тест на определение повторяющихся элементов**
all(diff(x) > 0)	**Тест на монотонность**
all(diff(diff(x)) == 0)	**Тест на опредедление равных приращений**

6.2 Обработка данных

В данном разделе рассматривается как поступать с:

- Отсутствующими значениями
- Выбросами значений или несовместимыми («неуместными») значениями

Отсутствующие значения

Специальное обозначение NaN, соответствует в MATLAB-е нечисловое значение. В соответ-ствие с принятыми соглашениями NaN является результатом неопределенных выражений та-ких как 0/0. Надлежащее обращение с отсутствующими данными является сложной пробле-мой и зачастую меняется в различных ситуациях. Для целей анализа данных, часто удобно использовать NaN для представления отсутствующих значений или данных которые недос-тупны. MATLAB обращается со значениями NaN единообразным и строгим образом. Эти значения сохраняются в процессе вычислений вплоть до конечных результатов. Любое мате-матическое действие, производимое над значением NaN, в результате также производит NaN. Например, рассмотрим матрицу, содержащую волшебный

квадрат размера 3x3, где це-нтральный элемент установлен равным NaN.

a = magic(3); a(2,2) = NaN;

a =

8 1 6
3 NaN 7
4 9 2

Вычислим сумму элементов всех столбцов матрицы:

sum(a)

ans =

15 NaN 15

Любые математические действия над NaN распространяют NaN вплоть до конечного резуль-тата. Перед проведением любых статистических вычислений вам следует удалить все NaN-ы из имеющихся данных. Вот некоторые возможные пути выполнения данной операции.

Программа	Описание
i = find(~isnan(x)); **x = x(i)**	Найти индексы всех эементов вектора, не равных NaN, и затем сохранить только эти элементы
x = x (find(~isnan(x)))	Удалить все NaN-ы из вектора
x = x (~isnan(x));	Удалить все NaN-ы из вектора (быстрее).
x (isnan(x)) = [ ];	Удалить все NaN-ы из вектора
X (any(isnan(X')), :) = [ ];	Удалить все

	строки матрицы X содержащие NaN-ы

Внимание. Для нахождения нечисловых значений NaN вам следует использовать специаль-ную функцию **isnan**, поскольку при принятом в MATLAB-е соглашении, логическое сравне-ние NaN == NaN всегда выдает 0. Вы не можете использовать запись **x(x==NaN) = [ ]** для удаления NaN-ов из ваших данных.

Если вам часто приходится удалять NaN-ы, воспользуйтесь короткой программой, записан-ной в виде M-файла.

function **X = excise(X)**
X(any(isnan(X')),:) = [ ];
Тогда. напечатав
X = excise(X);
вы выполните требуемое действие (**excise** по английски означает вырезать)

6.3 Удаление выбросов значений

Вы можете удалить выбросы значений или несовместимые данные при помощи процедур, весьма схожих с удалением NaN-ов. Для нашей транспортной задачи, с матрицей данных **count**, средние значения и стандартные (среднеквадратические) отклонения каждого столбца матрицы **count** равны

mu = mean(count)

sigma = std(count)

mu =
32.0000 46.5417 65.5833

sigma =

25.3703 41.4057 68.0281

Число строк с выбросами значений, превышающими утроенное среднеквадратическое откло-нение от среднего значения можно получить следующим образом:

```
[n, p] = size(count)

outliers = abs(count - mu(ones(n, 1),:)) > 3*sigma(ones(n, 1),:);

nout = sum(outliers)
```

nout =

1 0 0

Имеется только один выброс в первом столбце. Удалим все наблюдение при помощи выра-жения

```
count(any(outliers'),:) = [ ];
```

6.4 Регрессия и подгонка кривых

Часто бывает полезным или необходимым найти функцию, которая описывает взаимосвязь между некоторыми наблюдаемыми (или найденными экспериментально) переменными. Оп-ределение коэффициентов такой функции ведет к решению задачи переопределенной систе-мы линейных уравнений, то есть системы, у которой число уравнений превышает число не-известных. Указанные коэффициенты можно легко найти с использованием оператора обрат-ного деления \ (backslash). Допустим, вы производили измерения переменной **y** при разных значениях времени **t**.

```
t = [0  0.3  0.8  1.1  1.6  2.3]';
```

y = [0.5 0.82 1.14 1.25 1.35 1.40]';

plot(t,y,'o'); grid on

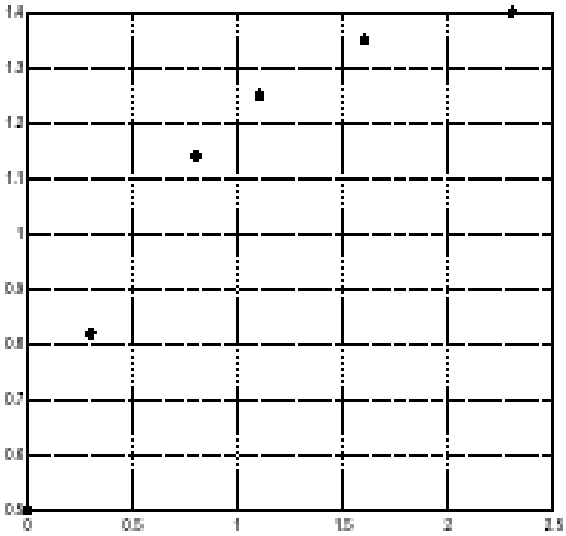


Рис. 6.2

В следующих разделах мы рассмотрим три способа моделирования (аппроксимации) этих данных:

- Методом полиномиальной регрессии
- Методом линейно-параметрической (linear-in-the-parameters) регрессии
- Методом множественной регрессии

Полиномиальная регрессия

Основываясь на виде графика, можно допустить, что данные могут быть аппроксимированы полиномиальной функцией второго порядка:

$$y = a_0 + a_1 t + a_2 t^2$$

Неизвестные коэффициенты $\mathbf{a_0}$, $\mathbf{a_1}$ и $\mathbf{a_2}$ могут быть найдены методом среднеквадратичес-кой подгонки (аппроксимации), которая основана на минимизации суммы квадратов отклоне-ний данных от модели. Мы имеем шесть уравнений относительно трех неизвестных,

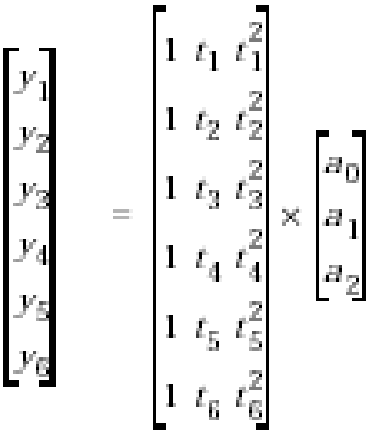


Рис. 6.3

представляемых следующей матрицей 6x3:

X = [ones(size(t)) t t.^2]

X = 1.0000 0 0

1.0000 0.3000 0.0900
1.0000 0.8000 0.6400
1.0000 1.1000 1.2100
1.0000 1.6000 2.5600
1.0000 2.3000 5.2900

Решение находится при помощи оператора \ :

a = X\y

a =

0.5318

0.9191

- 0.2387

Следовательно, полиномиальная модель второго порядка наших данных будет иметь вид

$$y = 0.5318 + 0.9191t - 0.2387\,t^2$$

Оценим теперь значения модели на равноотстоящих точках (с шагом 0.1) и нанесем кривую на график с исходными данными.

```
T = (0 : 0.1 : 2.5)';

Y = [ones(size(T))  T  T.^2]*a;

plot(T,Y,'-',t,y,'o'); grid on
```

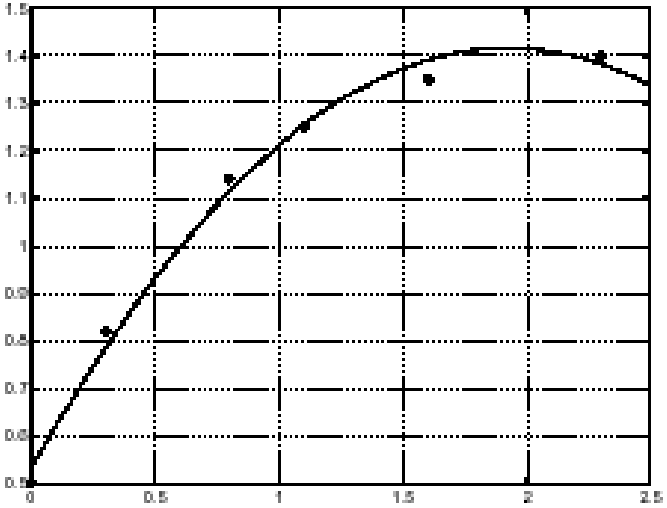


Рис. 6.4

Очевидно, полиномиальная аппроксимация оказалась не столь удачной. Здесь можно или по-высить порядок

аппроксимирующего полинома, или попытаться найти какую-либо другую функциональную зависимость для получения лучшей подгонки.

6.5 Линейно-параметрическая регрессия[1]

Вместо полиномиальной функции, можно было-бы попробовать так называемую линейно-параметрическую функцию. В данном случае, рассмотрим экспоненциальную функцию

$$y = a_0 + a_1 e^{-t} + a_2 t e^{-t}$$

Здесь также, неизвестные коэффициенты a_0 , a_1 и a_2 могут быть найдены методом наимень-ших квадратов. Составим и решим систему совместных уравнений, сформировав регресси-онную матрицу **X**, и применив для определения коэффициентов оператор \ :

```
X = [ones(size(t))    exp(- t)     t.*exp(- t)];

a = X\y

                                          a =

1.3974

                                              - 0.8988

0.4097
```

Значит, наша модель данных имеет вид

$$y = 1.3974 - 0.8988 e^{-t} + 0.4097 t e^{-t}$$

[1] Данная терминология не совсем соответствует принятой в русско-язычных изданиях.

Оценим теперь, как и раньше, значения модели на равноотстоящих точках (с шагом 0.1) и на-несем эту кривую на график с исходными данными.

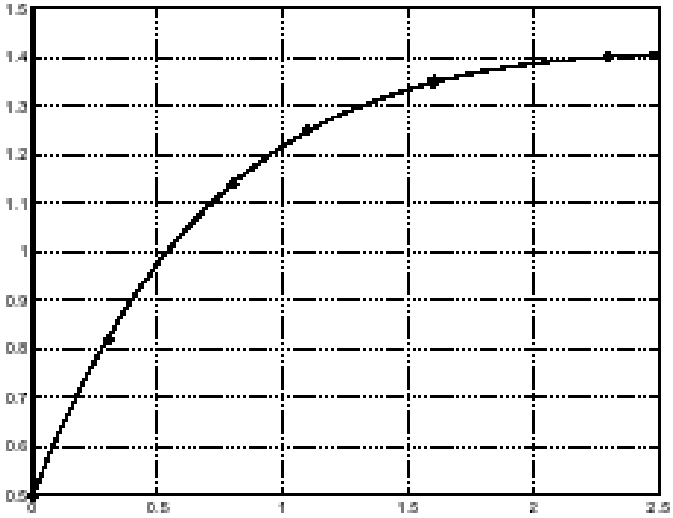


Рис. 6.6

Как видно из данного графика, подгонка здесь намного лучше чем в случае полиномиальной функции второго порядка.

6.6 Множественная регрессия

Рассмотренные выше методы аппроксимации данных можно распространить и на случай бо-лее чем одной независимой переменной, за счет перехода к расширенной форме записи. До-пустим, мы измерили величину **y** для некоторых значений двух параметров $\mathbf{x_1}$ и $\mathbf{x_2}$ и полу-чили следующие результаты

x1 = [0.2 0.5 0.6 0.8 1.0 1.1]' ;

x2 = [0.1 0.3 0.4 0.9 1.1 1.4]' ;

y = [0.17 0.26 0.28 0.23 0.27 0.24]' ;

Множественную модель данных будем искать в виде

$$y = a_0 + a_1x_1 + a_2x_2$$

Методы множественной регрессии решают задачу определения неизвестных коэффициентов a_0 , a_1 и a_2 путем минимизации среднеквадратической ошибки приближения. Составим сов-местную систему уравнений, сформировав матрицу регрессии **X** и решив уравнения отно-сительно неизвестных коэффициентов, применяя оператор \ .

X = [ones(size(x1)) x1 x2];

a = X\y

a =

0.1018
0.4844
-0.2847

Следовательно, модель дающая минимальную среднеквадратическую ошибку аппроксима-ции имеет вид

$$y = 0.1018 + 0.4844x_1 - 0.2847x_2$$

Для проверки точности подгонки найдем максимальное значение абсолютного значения от-клонений экспериментальных и расчетных данных.

Y = X*a;

MaxErr = max(abs(Y - y))

MaxErr =

0.0038

Эта ошибка дает основание утверждать, что наша модель достаточно адекватно отражает ре-зультаты наблюдений.

СПИСОК ЛИТЕРАТУРЫ:

1. **Дьяконов, Владимир Павлович**. **MATLAB** : Учебный курс / Владимир Павлович Дьяконов. - СПб. : Питер, 2001. - 553[7] с. : ил. - (Учебный курс). - Библиогр.: с. 535-537. - Алф. указ.: с. 538-553. - **ISBN** 5-272-00276-8 (в пер.) : 130.00 р.
2. **Потемкин,Валерий Георгиевич** Система инженерных и научных расчетов **MATLAB** 5.x : в 2 т. / Георгиевич Потемкин. - М. : Диалог-МИФИ, 1999 - . **Т. 1** / Валерий Георгиевич Потемкин. - М. : Диалог-МИФИ, 1999. - 366 с. : ил. - (в пер.) : Б. ц.
3. **Говорухин, В.** Компьютер в математическом исследовании : учебник / В. Говорухин, В. Цибулин. - СПб. : Питер, 2001. - 619[5] с. : ил. - (Учебный курс) (Программная поддержка исследовательских работ). - Библиогр.: с. 598-601. -Алф. указ.: с. 602-619. - **ISBN** 5-272-00220-2 : 137.00 р.
4. **Глушаков, Сергей Владимирович**. Математическое моделирование. MathCAD 2000. **MATLAB** 5.3 : Учебный курс / Сергей Владимирович Глушаков, Иван Анатольевич Жакин, Тимур Станиславович Хачиров. - Харьков : Фолио, 2001 ; М. : АСТ, 2001. - 525[3] с. : ил. - (Домашняя библиотека). - **ISBN** 966-03-1610-0 (в пер.)
5. **Лазарев,Юрий**. **MatLAB** 5.x : Учебное пособие / Юрий Лазарев. - Киев : BHV, 2000 ; Киев : Ирина, 2000. - 383[1] с. : ил. - (Библиотека студента). - Библиогр.: с. 381. - **ISBN** 966-552-068-7 (в пер.) : 79.00 р. - **ISBN** 5-7315-0096-7 : 79.00 р.

ПРИЛОЖЕНИЕ 1. АНАЛИЗ ДАННЫХ И ПРЕОБРАЗОВАНИЕ ФУРЬЕ (DATA ANALYSIS AND FOURIER TRANSFORMS)

Директория **matlab\datafun**

Основные операции (Basic operations)

1. **max** - Определение максимальных элементов массива.
2. **min** - Определение минимальных элементов массива.
3. **mean** - Определение средних значений элементов массива.
4. **median** - Определение медиан (срединных значений).
5. **std** - Определение стандартных отклонений элементов массива.
6. **var** - Определение дисперсий элементов массива.
7. **sort** - Сортировка элементов массива.
8. **sortrows** - Сортировка строк матриц.
9. **sum** - Суммирование элементов массива.
10. **prod** - Произведение элементов массива.
11. **hist** - Построение гистограммы.
12. **histc** - Подсчет элементов гистограммы.
13. **trapz** - Численное интегрирование методом трапеций.
14. **cumsum** - Куммулятивная сумма элементов массива.
15. **cumprod** - Куммулятивное произведение элементов массива.
16. **cumtrapz** - Куммулятивное численное интегрирование методом трапеций.

Конечные разности (Finite differences)

17. diff - Вычисление конечных разностей и приближенное дифференцирование.

18. gradient - Приближенное вычисление градиента функций.

19. del2 - Дискретная аппроксимация дифференциального оператора Лапласа.

Корреляционные соотношения (Correlation)

20. corrcoef - Вычисление коэффициентов корреляции.

21. cov - Вычисление ковариационной матрица.

22. subspace - Вычисление угла между двумя подпространствами.

Фильтрация и свертка (Filtering and convolution)

23. filter - Одномерная цифровая фильтрация.

24. filter2 - Двумерная цифровая фильтрация.

25. conv - Свертка и умножение полиномов.

26. conv2 - Двумерная свертка.

27. convn - N-мерная (многомерная) свертка.

28. deconv - Обращение свертки и деление полиномов.

29. detrend - Удаление линейного тренда.

Преобразование Фурье (Fourier transforms)

30. fft - Дискретное преобразование Фурье.

31. fft2 - Двумерное дискретное преобразование Фурье.

32. fftn - Многомерное дискретное преобразование Фурье.

33. ifft - Обратное дискретное преобразование Фурье.

34. ifft2 - Двумерное обратное дискретное преобразование Фурье.

35. ifftn - Многомерное обратное дискретное преобразование Фурье.

36. fftshift - Перенос нулевой частоты в середину спектра.

37. ifftshift - Аннулирование переноса нулевой частоты в середину спектра.

ПРИЛОЖЕНИЕ 2. СПРАВОЧНАЯ СИСТЕМА MATLAB

Среда **MATLAB** обладает многоуровневой справочной системой, которая позволяет получить информацию как по самой **MATLAB**, так и по всем ее приложениям.

Получить доступ к справочной информации **MATLAB** можно различными способами: введя в командную строку соответствующую справочную команду или функцию; с помощью команд меню **Help**; щелкнув на кнопке со знаком вопроса на панели инструментов основного окна **MATLAB**.

MATLAB имеет интерактивную справочную систему, которая реализуется в командном режиме с помощью ряда команд. Одной из них является команда

>> help

которая выводит весь список папок (каталогов), содержащих m-файлы с определениями операторов, функций и иных объектов (рис.П.1). Этот список дает представление о пакетах прикладных программ, расширяющих возможности системы **MATLAB** и содержащих примеры применения системы.

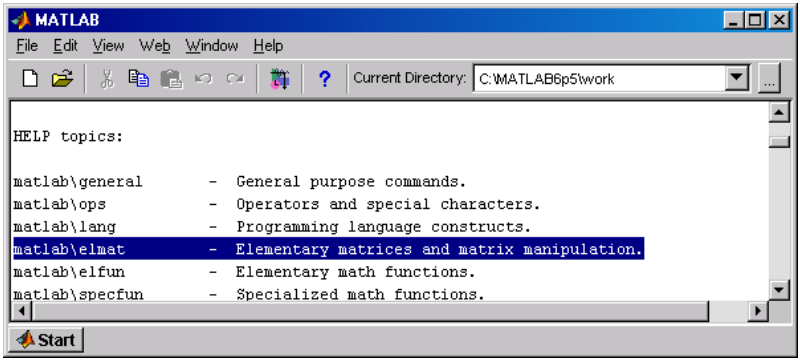


Рис.П.1

При вводе команды

>> help elfun

в командное окно выводится список встроенных элементарных функций (рис.П.2).

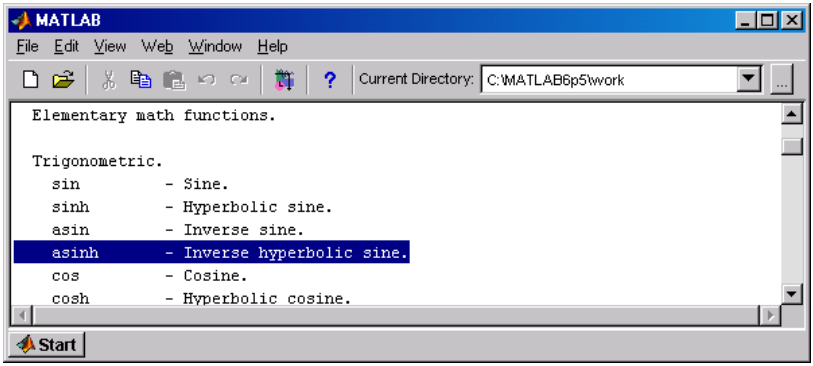


Рис.П.2

Для получения справочной информации по любой из них требуется выполнить команду **help** <имя функции>, например,

```
>> help asinh
 ASINH  Inverse hyperbolic sine.
    ASINH(X) is the inverse hyperbolic sine of the elements of X.
 Overloaded methods
    help sym/asinh.m
```

Теперь полученное сообщение содержит информацию о функции **asinh**. Хотя имя функции в **MATLAB** задается малыми (строчными) буквами, в сообщениях справочной системы имена функций и команд выделяются большими (прописными) буквами.

Если в командной строке набрать команду **doc elfun**, то при этом перейдем в главное окно **Help** внутренней справочной системы **MATLAB**, в правой половине которого будет открыта первая страница документа справочной информации с указанным заголовком.

Для получения справки по конкретной функции необходимо выбрать вкладку с одноименным названием и щелкнуть на ней левой кнопкой мыши. Для получения более полной информации, снабженной графиками и лучше отформатированной, надо выполнить команду **doc** <имя функции>, например, (рис.П.3)

>> doc asinh

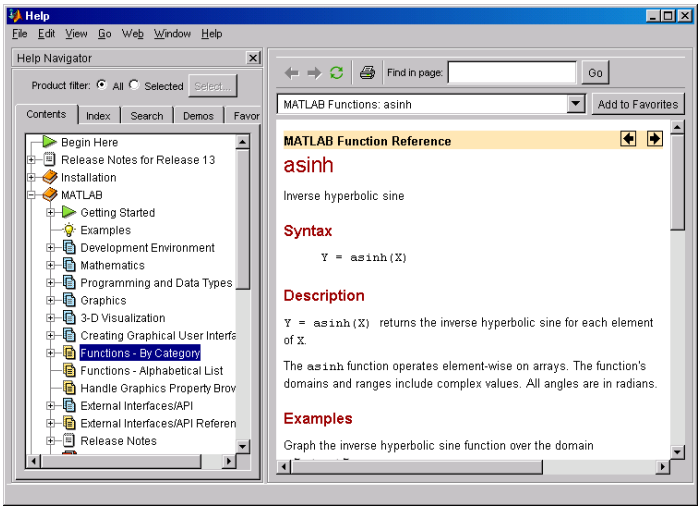


Рис.П.3

В главном окне **Help** внутренней внутренней справочной системы **MATLAB** слева находится панель **Help Navigator** – так называемый Help – навигатор – с пятью вкладками, на которых реализованы различные способы поиска информации:

- **Contents** (Содержание) – поиск информации по оглавлению доступных разделов;
- **Index** (Индекс) – поиск информации по алфавитному) каталогу;
- **Search** (Поиск) – поиск информации по ключевому слову или фразе, набираемых в поле ввода;
- **Demos** (Примеры) – переход к странице, с которой можно получить доступ к демонстрационным примерам по любым темам;
- **Favorites** (Фавориты) – список разделов, выделенных пользователем в качестве наиболее посещаемых.
- При вызове окна справки **Help** по умолчанию в нем отображается вкладка **Contents** с оглавлением, представленным в виде дерева справочных каталогов (разделов). Чтобы раскрыть в оглавлении нужный раздел,

щелкните по кнопке со знаком <+> (слева от названия раздела) либо дважды щелкните по кнопке с его названием.

Каждый раздел имеет подраздел **Getting Started** (Начало), который кратко знакомит с содержимым этого раздела, а также подраздел **Examples** (Примеры), помеченный пиктограммой с изображением лампочки – здесь можно найти соответствующие данной теме примеры.

Далее можно перейти к требуемому подразделу.

В раделе 6.6 рассмотрен пример решения с помощью солвера **ode45** задачи Коши для обыкновенного дифференциального уравнения. Там же упоминалось уравнение Ван-дер-Поля $y''-\mu(1-y^2)y'+y=0$, пример решения которого с помощью солвера **ode15s** можно найти во встроенной в пакет **MATLAB** справочной системе **Help**. Это решение приведено внутри раздела **Differential Equations**, являющегося подразделом раздела **Mathematics**. Ознакомиться с ним с помощью **Help Navigator** можно следующим образом (рис.П.4).

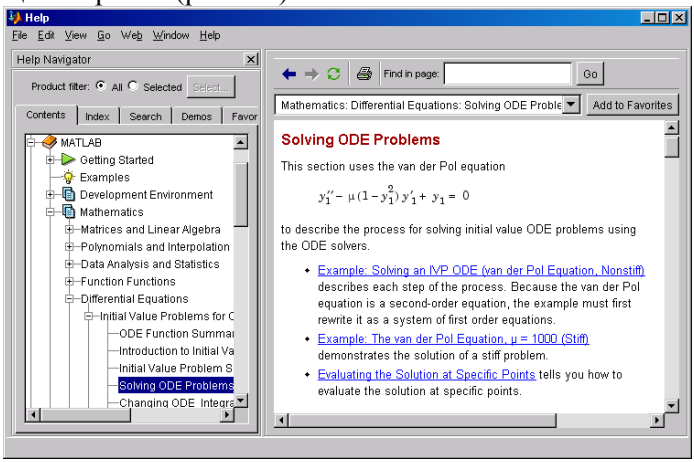


Рис.П.4

Система **MATLAB** предлагает и решатели для граничных задач, а также решатели для дифференциальных уравнений с частными производными. Об этих решателях можно узнать с помощью **Help Navigator** следующим образом (рис.П.5).

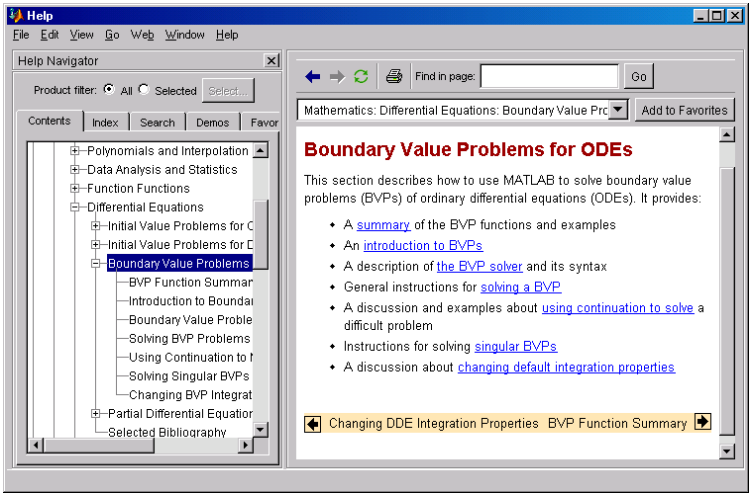


Рис.П.5

Из рис.П.5 видно, что нужно ознакомится с разделами **Boundary Value Problems for ODEs** (Граничные задачи для обыкновенных дифференциальных уравнений) и **Partial Differential Equations** (Дифференциальные уравнения в частных производных). Они находятся внутри раздела **Differential Equations**, являющегося подразделом раздела **Mathematics**. Решению дифференциальных уравнений в частных производных посвящен отдельный пакет расширенной системы **MATLAB – Partial Differentional ToolBox**.

Чтобы выделить больше места для отображения справочной информации, панель **Help Navigator** можно закрыть щелчком по кнопке с крестиком в правом верхнем углу. Открыть панель после этого можно посредством команды **View => Help Navigator**.● Вкладка **Index** (рис.П.6) позволяет

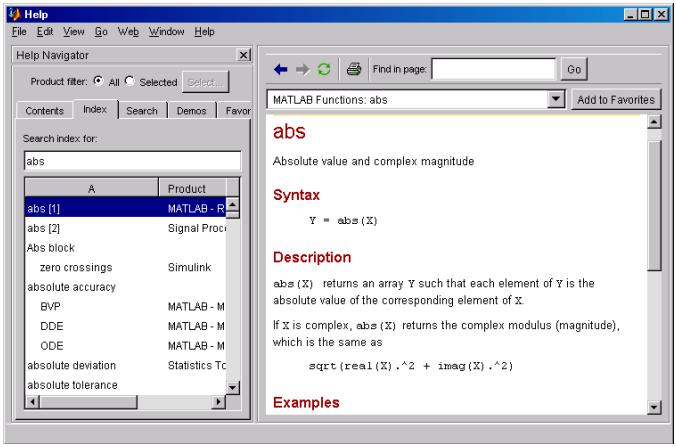


Рис.П.6

организовать поиск по ключевому слову, набираему в поле ввода под надписью **Search index for** (Поиск по индексу). Имеется возможность перемещаться по упорядоченному списку либо с помощью линейки прокрутки, либо набрав в поле ввода одну или несколько начальных букв.

- Вкладка **Search** (рис.П.7) позволяет организовать несколько вариантов поиска в зависимости от того, что выбрано среди строк раскрывающегося списка **Search type** (Тип поиска). Вариантов всего четыре – поиск по всем документам (**Full text** (Полный текст)), поиск по заголовкам документов (**Docment Titles** (Название документа)), поиск по именам функций (**Function name** (Имя функции)), поиск в базе данных по каналам Интернета (**Online Knowledge Base** (Информационная база в Интернете)).

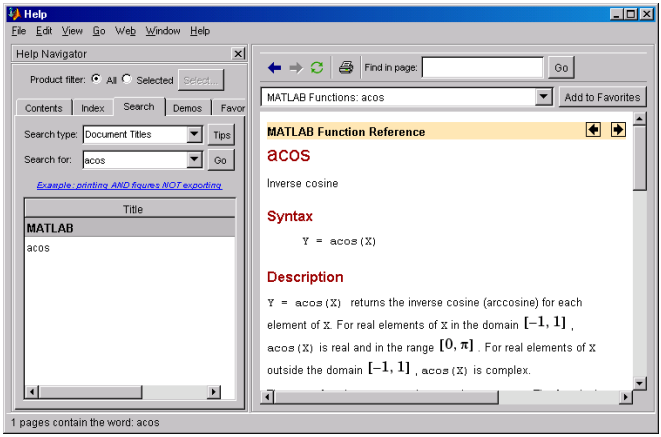


Рис.П.7

● Вкладка **Demos** (рис.П.8) содержит на левой панели дерево каталогов с

демонстрационными примерами. Открыв нужный каталог (щелчком на кнопке со знаком <+>), попадаем на страницу на страницу со списком примеров по выбранной теме.

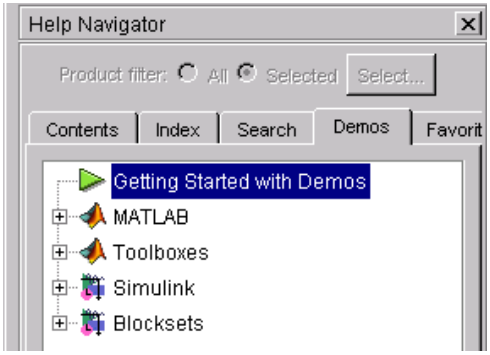


Рис.П.8

Примеры могут содержать листинги программ **MATLAB**, которые можно просмотреть и запустить на выполнение. Для запуска примера нужно щелкнуть на кнопке **Run this demo** (Запустить этот пример) внизу страницы с примером.

Примеры могут представлять собой видеоролики или видеоуроки, демонстрирующие, как пользоваться тем или иным инструментом.

● Страницы, которые просматриваются чаще всего, можно добавить в список избранных с помощью команды **Add to Favorites** (Добавить в избранное). После этого их названия появятся в меню **Favorites**, отсюда их можно будет быстро открыть.

Использование команд справки является быстрым способом получения информации, если вы точно знаете, **что** ищете и **как** это называется.

Например: **doc ops** – выводит информацию по разделу операторов и специальных символов; **doc function** – выводит информацию о назначении и создании файл-функций.

Для ускорения поиска сведений о функциях в файлах помощи предусмотрены два систематизированных каталога – список функций, упорядоченных по алфавиту и классифицированных по категориям (**MATLAB Function Listed by Category** (Список функций **MATLAB** по категориям)) и общий список всех функций, упорядоченных по алфавиту (**MATLAB Function Listed Alphabetically** (Алфавитный список функций **MATLAB**)).

Справочная система **MATLAB** позволяет получить информацию по пакету **Symbolic Math Toolbox** как по одному из приложений **MATLAB**. Для этого после вызова главного окна справки **Help** в оглавлении вкладки **Contents** надо раскрыть раздел **Symbolic Math Toolbox**. Способы получения информации по пакету в окне справки **Help** аналогичны тем, что были показаны выше при поиске информации по самой **MATLAB**.

В интерактивной справочной системе **MATLAB** с помощью команды

>>help symbolic

можно получить перечень входящих в пакет **Symbolic Math Toolbox** команд и функций. Для получения справки по любой команде или функции можно использовать команду

>>help sym/name.m,

где **name** – это имя соответствующей команды или функции, а **name.m** – имя m – файла, задающего данную команду или функцию.

В справках вместо **help** можно использовать **doc**. При этом перейдем в главное окно **Help** внутренней справочной системы **MATLAB**, в правой половине которого будет открыта первая страница документа справочной информации с указанным заголовком. Например, (рис. П.9)

>> doc dsolve

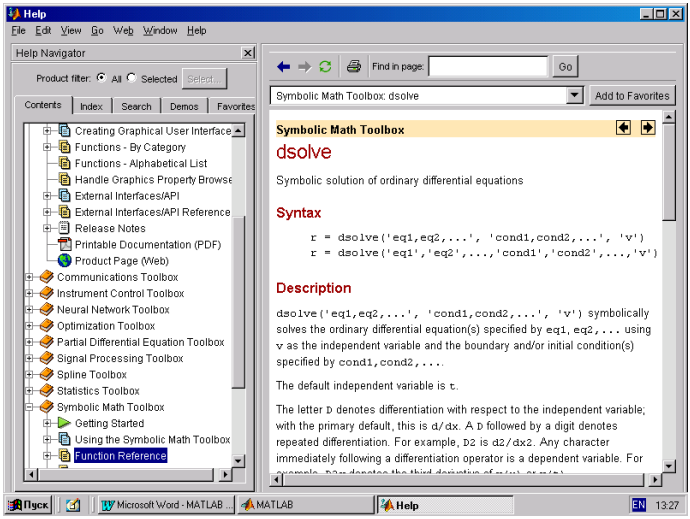


Рис.П.9

Для получения списка специальных математических функций системы **Maple**, доступных из **MATLAB**, служит команда **mfunlist**. Для доступа к информации о функциях **Maple**, в том числе и не входящих в список команды **mfunlist**, предназначена команда **mhelp**<*имя функции*>.

Ноздреватых Д.О. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ НАЧАЛЬНЫЕ СВЕДЕНИЯ О MATLAB Учебное пособие для студентов технических вузов по дисциплинам: «Информационные технологии», «Информатика», «Учебная практика» и др. – Томск: ТУСУР, 2016. – 174 с.