МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«КУЗБАССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Т. Ф. ГОРБАЧЕВА»

КАФЕДРА ПРИКЛАДНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Система MATLAB

Справочное пособие по дисциплине
«ЧИСЛЕННЫЕ МЕТОДЫ АНАЛИЗА»
для студентов специальности 080801
«Прикладная информатика в экономике»

Составитель **М. А. Тынкевич**

Утверждено на заседании кафедры
Протокол № 4 от 04.11.2011
Рекомендовано к печати
учебно-методической комиссией
специальности 080801
Протокол № 6 от 02.11.2011
Электронная копия хранится
в библиотеке КузГТУ

Кемерово 2012

Введение в MatLab (происхождение и возможности)

Электронные вычислительные машины (ЭВМ) первого поколения создавались для расчета баллистических таблиц, оболочек ядерных реакторов, траекторий вывода на орбиту космических аппаратов и решения других, более скромных, задач прикладной математики и инженерии. Программист того времени должен был найти взаимопонимание с заказчиком в математической постановке задачи, быть специалистом в области вычислительной математики, способным найти или создать численный метод решения задачи и, зная систему команд ЭВМ и основные приемы программирования, составить машинную программу.

Система команд первых ЭВМ (для каждого типа машин своя) включала ограниченный набор элементарных операций (арифметических и логических, пересылки между ячейками памяти, перехода по условию) и лишь избранные ЭВМ (например, первая наша серийная машина “Стрела” с памятью в 2048 43-разрядных ячеек и быстродействием 2000-4000 операций/сек) имели в этом наборе более сложные операции (перевод числа из двоично-десятичной системы в двоичную и обратно, вычисления синуса, натурального логарифма, экспоненты, обратной величины и квадратного корня). Построение из этих “кирпичиков” программ для реальных задач в условиях ограниченной емкости памяти и невысокого (по современным меркам) быстродействия было достаточно трудоемким процессом и не случайно возникло понятие *искусства программирования*.

Естественно, что уже на первых этапах практического программирования появилась идея *подпрограммы* как программного блока, допускающего многократное обращение к себе из различных участков программы. Так появились сборники текстов подпрограмм для вычисления элементарных функций и основных численных методов (интегрирования, решения обыкновенных дифференциальных уравнений и пр.), позднее эти сборники стали хранить во внешней памяти машины (как правило, на магнитных лентах или барабанах). Наконец, были созданы системы, позволяющие по номеру подпрограммы вызывать ее из внешней памяти в оперативную с настройкой по месту вызова и при наличии определенных договоренностей передавать входную и выходную информацию. В отечественной практике наиболее совершенной была интерпретирующая система ИС-2 (позднее ИС-22) для семейства машин типа М-20 с уникальной по качеству и разнообразию

библиотекой стандартных подпрограмм.

Переход от программирования в кодах ЭВМ к универсальным языкам программирования – в первую очередь, к Алголу-60 - породил *библиотеки алгоритмов*, которые и составили 30 лет спустя базу для построения систем для решения математических задач, возникающих в разнообразных научных исследованиях и технических разработках – Maple, Mathematica, MathCad, MatLab и др. Для этих систем характерны простота подготовки данных, удобные формы вывода результатов вычислений, встроенные средства помощи, диагностики ошибок – т.н. *дружественный интерфейс*.

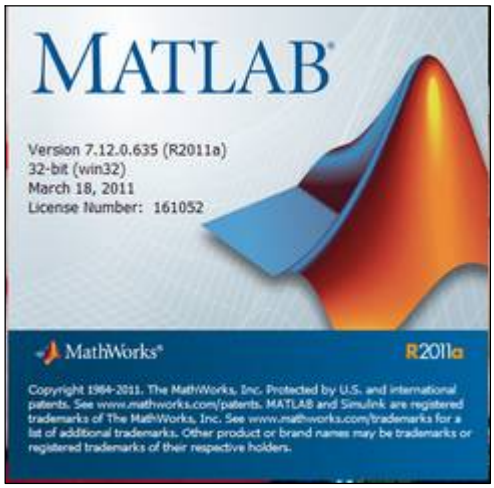
Предлагаемая вниманию читателя система MatLab (Matrix Laboratory) является интерактивной (от англ. *interaction* — «взаимодействие») системой для выполнения инженерных и научных расчетов, ориентированная на ***работу с массивами*** данных. Она поддерживает работу в программном и интерактивном режиме с векторами и матрицами, позволяет с легкостью решать системы уравнений, выполнять численное интегрирование, строить графики и пр. Система допускает использование пакетов прикладных программ (ППП) символьной математики, статистики, оптимизации, анализа и синтеза систем управления, обработки сигналов и изображений, финансов, картографии и др. Система позволяет обмениваться информацией с текстовым редактором Microsoft Word, в частности переносить любые тексты и рисунки в буфер или читать текстовые строки из буфера как исполняемые команды.

Существует несколько версий системы, но преемственность почти полностью сохраняется и речь идет лишь о расширении возможностей.

Опыт показывает, что студенты, владеющие программированием в Pascal’е или Visual Basic’е, улавливают технологию программирования и работы в MatLab’е в течение 2-3 часов (на полное знакомство с возможностями системы потребуется существенно бóльшее время).

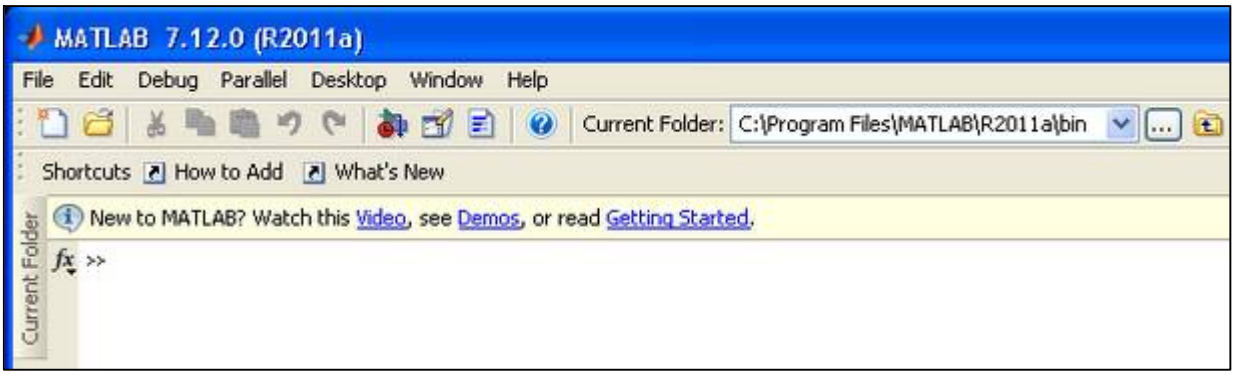
Ниже мы излагаем информацию лишь о небольшой части этой системы, полезную при изучении численных методов решения основных задач вычислительной математики студентами, интересы которых ограничены сферой экономико-математического моделирования.

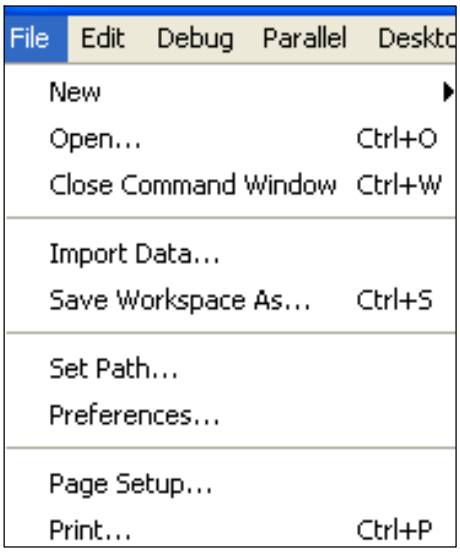
1. Режим командной строки. Форматы данных



При вызове MatLab на дисплей выводится заставка, которая сменяется *командным окном*, в верхней части которого размещено окно управления - меню с пунктами *Файл*(*File*), *Правка* (*Редактирование*), Debug (Отладка), Parallel Desktop, Window (*Окно*) и Help (*Помощь*).

Во второй строке перечислен ряд вспомогательных кнопок для быстрого копирования, удаления и др.





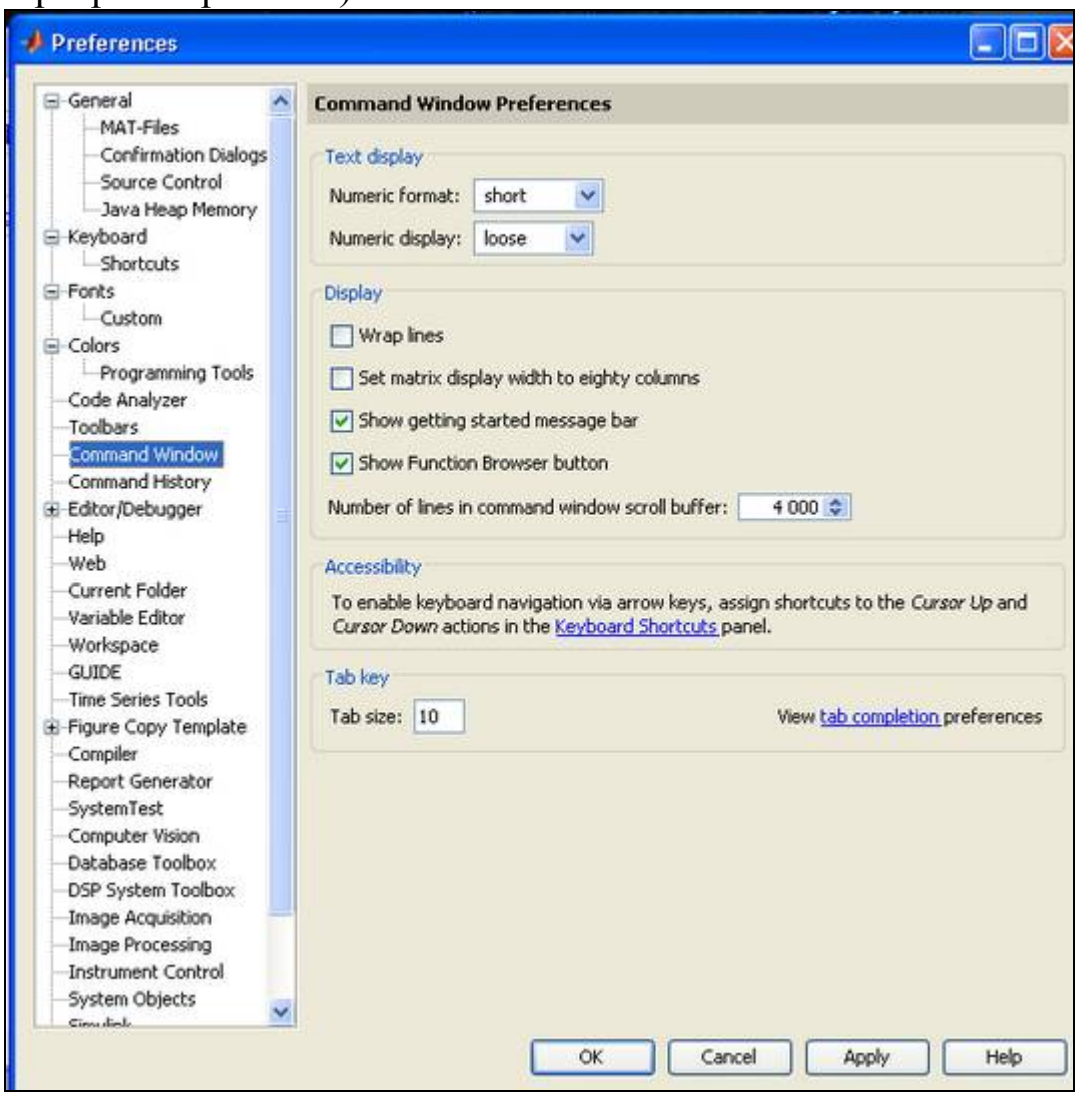
При выборе File (Файл) обнаруживается меню, где, в первую очередь, следует обратить внимание на New (создать новый файл), Open (открыть существующий файл), Set Path (установить путь к файлу) и Preferences (установки) .

При входе в Preferences в правом верхнем углу обнаруживаются установки для представления числовой информации на дисплее в окне команд:

Numeric display – loose (свободный, с пробелом между строками) или compact (компактный вывод) – рекомендуется последний вариант ;

Numeric format –**Short** (короткое, 5-значное) , **Long** (длинное, 15-значное), **Hex** (шестнадцатеричное), **Bank** (денежное, доллары и центы), **ShortE** и **LongE** (экспоненциальные, с разновидностми **G** и

Eng), + (с выводом знака), **Rational** – отношение целых чисел (обратите внимание на эту форму – такой нет ни в одной универсальной среде программирования).



Здесь же можно указать емкость буфера для сохранения последних из выполненных команд (с помощью клавиши ↑ можно возвращаться к ним), стиль и размер выбираемого шрифта (Fonts) для командной строки и множество других вспомогательных средств .

Ниже строк меню после предварительных предложений приступить к демонстрации или прогуляться по справочнику о возможностях системы, выводится ***командная строка*** (начинается символом "»") .

В командной строке (режим диалога) можно *набрать команду (оператор, выражение*) и, нажав Enter, получить ответ (*answer*). Например, после набора команды (всем известного оператора присваивания) **a=3.2** в последующих строках (в зависимости от установок) появится **a = 3.2000 или 3.2000e+000** (переменной **a** присвоено значение 3.2), после набора выражения **sin(a)/a** увидим его значение **ans = -0.01824191982112**.

Сразу же учтите, что в именах переменных (последовательностях латинских букв и цифр, начинающихся с буквы; знак _ относится к буквам) **строчные и заглавные буквы** отнюдь **не тождественны !**

Если вы хотите выполнить команду без вывода результата, в конце команды ставьте символ точки с запятой.

Кстати, если команда не помещается полностью в видимой части одной строки экрана, поставьте многоточие (хотя бы две точки), нажмите Enter и продолжайте в следующей строке.

Имейте в виду, что фрагмент окна команд (с размещенными в нем командами и сообщениями) можно выделить и копировать в буфер, например, для переноса в Word. Возможен перенос в командную строку текстовых фрагментов из других систем.

Как и в любой системе программирования, в MatLab'е присутствует понятие *переменной* величины, но в роли ее значения выступает *массив* (*array*).

В системе определены 6 встроенных типов данных (массивов):

- числовые величины удвоенной точности (**double**);
- массивы символов – строки (**char**), при задании строковой константы ее символы заключают в апострофы;
- двумерные разреженные матрицы (**sparse**), массивы ячеек (**cell**), массивы записей (**struct**) и специальные массивы целых чисел от 0 до 255 (uint8).

Здесь мы ограничимся рассмотрением лишь обычных числовых массивов и строк.

В отличие от систем, требующих предварительного описания типа переменных, в Matlab'е тип, как правило, определяется по умолчанию при первом же присваивании.

Так, командой **» a=[1 2 3; 4 5 6]** формируется матрица размерности 2×3 с соответствующими элементами; командой **» b=[1 2 3]** – вектор-строка; командой **» b=[1; 2; 3]** – вектор-столбец; **d=zeros(4,7)** – матрица размерности 4×7 с нулевыми элементами. Для ссылок на отдельные элементы массива можно пользоваться индексами (начало отсчета от 1), например, **a(k,3)** определяет третий элемент k-ой строки, **a(:,3)** – весь третий столбец.

Обратите внимание на требования к символике:

– при задании массива значениями использовать квадратные скобки;

– элементы в строке массива разделять пробелами или запятыми, а завершать строку точкой с запятой;

– при указании списка индексов использовать круглые скобки и

разделительные запятые (указание индекса символом двоеточия соответствует заданию всех значений по соответствующему индексу).

При работе с массивами можно использовать списки **i:k (**от **i** до **k** с шагом **1)** или **i:j:k** (то же с шагом **j)** ; например, операторы присваивания t=–pi:0.01:pi или p=0:8 задают массивы с соответствующими значениями (некоторые сочетания дают пустое множество, например, q=3:1). Оператор a=[0,1:7] порождает массив из 8 значений от 0 до 7.

В библиотеке системы предусмотрен ряд функций для формирования массивов простейшей структуры, например :

- нулей zeros(n), zeros(m,n), zeros(m,n,p,..), zeros(size(A)) (одномерный, двумерный, многомерный, соразмерный с массивом A);

- единиц ones(n), ones (m,n), ones (size(A)).

Естественно, что к числовым переменным применимы традиционные арифметические операции(+,–,*, / и ^ (возведение в степень). Однако при выполнении ряда операций приходится различать *поэлементные операции* с массивами (перед знаком операции ставят точку) и *операции над матрицами* (по правилам линейной алгебры).

В первом варианте предполагается одинаковая размерность или один из операндов – скаляр. Например, A+B, A-B, A .*B, A.\ B (левое деление B на A) , A ./B (правое деление A на B) , A .^B (поэлементное возведение в степень), A. ’ (транспонирование массива – преобразование строк в столбцы или обратно, для комплексных значений получаются комплексно-сопряженные)).

Во втором варианте оператор X= A\B определяет решение системы *m* уравнений AX=B с *k* правыми частями: B и X – матрицы $m \times k$ {можно X=B’/A’) , A’ задает транспонирование матрицы (для комплексных дополняется комплексным сопряжением), A^k определяет степень матрицы (при k=0 – единичная матрица, при целом k<0 – умножение обратной и при целом k>0 исходной матриц; для других k вычисляет $R^T D.\hat{}k/R$, где R – диагональная матрица собственных чисел и D – матрица собственных векторов)

Примечательно, что **система работает** не только с действительными, но и **с комплексными числами** (роль мнимой единицы играют символы **i, j):**

» a=1+2i a = 1.0000 + 2.0000i
» b=1-3i b = 1.0000 – 3.0000i
» a’ ans = 1.0000 –2.0000i
» a*b ans = 7.0000 – 1.0000i
» exp(ans) ans = 592.51 –922.78 i

Над массивами можно выполнять *операции поэлементного отношения* : A < B , A <= B , A > B , A>= B (только для действитель-

ных частей), A == B, A~=B (равно/не равно – для действительных и мнимых частей), которые порождают массив с единицами (*истина*) и нулями (*ложь*) той же размерности. Аналогично реализуются и *логические операции*: отрицания ~A, конъюнкции (логического умножения – И) A**&**B, дизъюнкции (логического сложения – ИЛИ) A|B.

Все переменные системы размещаются в т.н. **рабочей области** (**workspace**), содержимое которой (имена, размерность, тип) можно просмотреть командами **who** (перечень имен переменных) и **whos**:

Name	Size	Bytes	Class	Attributes
i	1x1	8	double	array
r	4x629	20128	double	array
t	1x629	5032	double	array
ans	3x3	144	double	complex
Grand total is 3146 elements using 25168 bytes				

Можно рабочую область очистить полностью командой **clear** или частично – **clear** <список имен>.

2. Элементарные математические функции

Цепочкой действий Help | Product Help | Functions By Category или Alphabetical можно получить информацию о многообразия т.н. встроенных функций (здесь мы отмечаем лишь самые известные или оригинальные математические функции. Аргументами большинства из них являются не только скаляры, но и массивы.

pi = 4*atan(1)=imag(log(–1))=3.1415926535897..;

abs(X) – абсолютная величина (модуль) (для комплексного числа $a+bi$ его модуль равен $\sqrt{a^2+b^2}$) : abs(3–4i)=5 , abs(–13)=13;

angle(X) – аргумент комплексного числа (из диапазона [–π,π]): комплексное $X=a+bi$ представимо как $r\cdot e^{i\varphi}$, где $a = r\cos\varphi$, $b = r\sin\varphi$, φ – круговой арктангенс отношения b к a : angle(1) =0 ; angle(3+4i) = 0.9273; angle(4+3i) = 0.6435 .

real(X), **imag**(X) – действительная и мнимая часть числа;

conj(X) – *комплексно-сопряженное*: conj(2+3i) = 2– 3i ;

ceil(X), **fix**(X), **floor**(X), **round**(X) – округления (до ближайшего целого, не меньшего X; отбрасывание дробной части; до ближайшего целого, не большего X; до ближайшего целого);

mod(X,Y) –приведение X по модулю Y (остаток от деления X на Y);

sign(X) – знак числа (для комплексных X / |X|): sign(–6) =–1 ;

gcd(m,n) –наибольший общий делитель (для целых чисел):

gcd(18,27) = 9 ;

lcm(m,n) – наименьшее общее кратное: lcm(34,51) = 102 ;

rat(X), **rat** (X,k) – представление **цепной дробью** с точностью $|X|\cdot 10^{-k/2}$ (по умолчанию $|X|\cdot 10^{-6}$): rat(12.546)=**13**+1/(**–2**+1/(**–5** +1/(**15**))) ; rat(12.5) =13 + 1/(-2) ;

rats(X), **rats**(X,k) – представление отношением целых чисел: rats(12.546) = 2045/163 ;

sqrt(X) – квадратный корень: sqrt(5) = 2.2361; sqrt(3+4i) = 2+ 1i;

exp(X) – экспонента e^X ($e^{x+iy}= e^x(cos\ y+i\ siny)$) : exp(1) = 2.7183*;* exp(2+i) = 3.9923 + 6.2177i ;

log(X) – натуральный логарифм;

log2(X), **log10**(X) –логарифм по основанию 2 и основанию 10;

sin(X), **cos**(X), **tan**(X), **cot**(X), **csc**(X), **sec**(X) – тригонометрические функции (синус, косинус, тангенс, котангенс, косеканс, секанс):

$sin(x+iy)=sin(x)\ ch(y)+i\ cos(x)\ sh(y)$; $cos(x+iy)=cos(x)\ ch(y)-i\ sin(x)\ sh(y)$,

$tg\ (X)= sin(X)/\ cos(X)$; $ctg(X)=cos(X)/sin(X)$;

$cosec(X)=1/sin(X)$; $sec(X)=1/\ cos(X)$:

sin(pi/2) = 1; sin(3+4i) = 3.8537 –27.0168i ;

asin(X), **acos**(X), **atan**(X), **acot**(X), **acsc**(X), **asec**(X) – обратные тригонометрические функции (арксинус, арккосинус и т.д.): asin(1/sqrt(2)) = 0.7854 ; asin(3+4i) = 0.6340 + 2.3055i ;

atan2(Y,X) – круговой арктангенс (только для действительных частей аргументов Y и X), берется в интервале $[-\pi,\pi]$;

sinh(X), **cosh**(X), **tanh**(X), **coth**(X), **csch**(X), **sech**(X) – гиперболические функции (синус, косинус, тангенс, котангенс, косеканс, секанс): $sh(X)=(e^X-e^{-X})/2$, $ch(X)=(e^X+e^{-X})/2$ и др.;

asinh(X), **acosh**(X), **atanh**(X), **acoth**(X), **acsch**(X), **asech**(X) – обратные гиперболические функции:

$$arsh(X)=ln(X+\sqrt{X^2+1}),\ arch(X)=ln(X+\sqrt{X^2-1}).\ arth(X)=\frac{1}{2}ln\frac{1+X}{1-X},$$

$$arcth(X)=\frac{1}{2}ln\frac{1-X}{1+X},\ arcsch(X)=arsh(1/X),\ arsech(X)=arch(1/X);$$

erf (X) – интеграл вероятностей (функция Гаусса, функция ошибок)

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

и родственные функции :

erfc(X)=1–erf(X) (дополнительный интеграл вероятностей) ;

erfcx(X)=exp(X^2)·erfc(X) (нормированный дополнительный интеграл вероятностей);

erfinv(X) (аргумент, для которого интеграл вероятностей равен X);

gamma(X) – гамма-функция $\Gamma(x)=\int_0^\infty e^{-t}t^{x-1}dt$ (при целочисленных x $\Gamma(1+x)=x!$): gamma(5)=4!=24, gamma(0.5)= 1.7725, gamma(-0.5)= – 3.5449, gamma(0.1) = 9.5135, gamma(0) выдает сообщение о делении на нуль и неопределенное значение Inf;

gammainc(X,a) – $P(x,a)=\frac{1}{\Gamma(a)}\int_0^x e^{-t}t^{a-1}dt$ (неполная гамма-функция);

gammaln(X) – логарифмическая гамма-функция $ln\ \Gamma(x)$;

beta(X,Y) – бета-функция $B(x,y)=\int_0^1 t^{x-1}(1-t)^{y-1}dt=\frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ и родственные ей неполная и логарифмическая бета-функции;

функции преобразования координат:

из декартовых (X,Y) в полярные (r,φ): $r=(X^2+Y^2)^{1/2}$, $\varphi=Arctg(Y/X)$ – **[φ,r]=cart2pol(X,Y)**;

из декартовой системы (X,Y,Z) в цилиндрическую (r,φ,Z) – **[φ,r,Z]=cart2pol(X,Y,Z)** ;

из декартовой системы в сферическую (r,φ,θ) : $r=(X^2+Y^2+Z^2)^{1/2}$, $\varphi=Arctg(Z/(X^2+Y^2)^{1/2})$, $\theta=Arctg(Y,X)$ – **[θ,φ,r]=cart2sph(X,Y,Z)**;

из полярной и цилиндрической в декартову (**pol2cart)**: $X=r\cdot cos(\varphi)$, $Y=r\cdot sin(\varphi)$; из сферической в декартову(**sph2cart)**: $Z=r\cdot sin(\varphi)$, $X=r\cdot cos(\varphi)\cdot\cos(\theta)$, $X=r\cdot cos(\varphi)\cdot sin(\theta)$ (эти функции лежат в основе средств графического отображения результатов анализа);

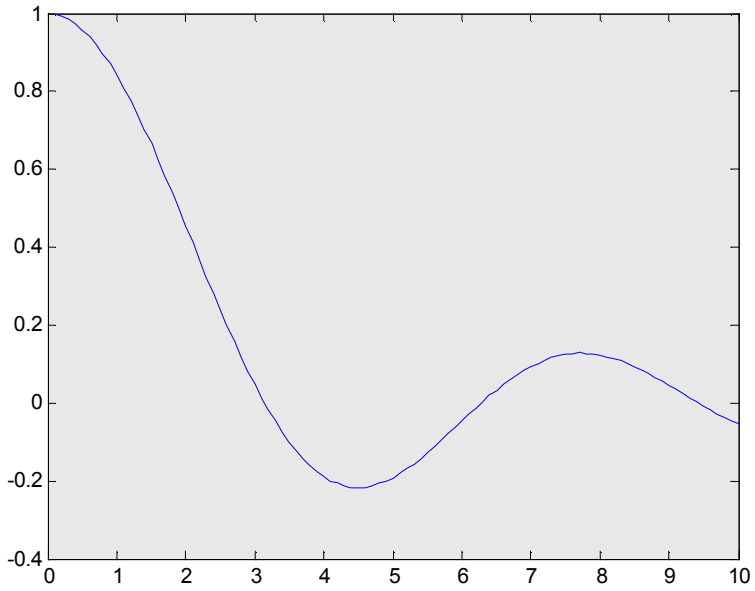
специальные функции (цилиндрические функции Бесселя, Неймана, Ханкеля; функции Эйри, эллиптические функции Якоби и эллиптические интегралы, интегральная показательная функция, присоединенные функции Лежандра и множество других функций, полезных при изучении физических процессов).

Наряду с упомянутыми, в библиотеке системы содержится колоссальное количество функций, предназначенных для решения задач линейной алгебры, аппроксимации данных, численного интегрирования, решения обыкновенных дифференциальных уравнений, поиска корней уравнений, экстремальных значений функций, обслуживания графики, обработки дат, множеств и др.

Например, если вы имеете намерение познакомиться с поведением какой-то из функций, вам достаточно поступить по аналогии с

ниже приведенным элементарным примером:

```
»  t=0.1:0.1:10;  % значения аргумента  (без вывода на экран);
» e=sin(t)..t;       %  массив значений     функции;
» plot(t,e)          % построение графика   функции .
```



3. Режим программирования

Для перехода в режим программирования в меню выбираем пункт **File** и входим в редактор MatLab'a: **New** (создать новый файл) или **Open** (открыть существующий файл с расширением **.m**). В дальнейшем может производиться обычный набор текста программы (подпрограммы) или его корректура и действия в соответствии с меню (сохранение под текущим или другим именем, запуск на исполнение в обычном и отладочном режимах и др.).

Различают два вида М-файлов: *М-сценарии* (*script*) и *М-функции* (*function*).

М-сценарий – это файл, содержащий последовательность команд и комментариев (строк, начинающихся символом %) и оперирующий данными из рабочей области. Заголовок его начинается командой **script** или может отсутствовать. Для входа в файл достаточно указать имя соответствующего файла.

Для М-функции характерны традиционные требования к подпрограммам-функциям в других развитых систеиах программирования – допускаются входные и выходные аргументы, локализация внутренних ее переменных и возможность обращения к ней из других программ. М-функции хранятся в библиотеке функций системы в виде текстовых файлов. Обращение к функции может производиться из

любого выражения в вызывающей программе указанием ее имени и заключенного в круглые скобки списка фактических параметров.

Заголовок М-функции имеет вид:

function [<список выходных переменных>**]=**
<имя функции> **(**<список входных переменных>**)**

Так функция вычисления факториала положительного целого числа и его обратной величины может быть описана файлом fact.m:

```
function [f, g]=fact(n)   %  факториал и обратная величина
f=prod (1:n);             % произведение элементов списка
g=1/f;
```

Кроме приведенных выше операторов присваивания, программирование в MatLab'е допускает и ряд других традиционных операторов.

Условный оператор выступает в одной из следующих форм:

```
if <условие>          if <условие>          if <условие>
    <команды>             <команды>             <команды>
end                   else                  elseif <условие>
                          <команды>             <команды>
                      end                     else
                                                <команды>
                                              end
```

В роли условия может использоваться любое логическое выражение, построенное на основе операций отношения и логических. **Если значение этого выражения является массивом,** то **условие считается истинным, если все его элементы истинны** (истина – 1, ложь – 0).

Оператор цикла с заданным числом повторений, в основном используемый в форме:

```
for V=A :H:B          for V=A:B
    <команды>             <команды>
end                   end
```

(V –переменная/параметр цикла, A,B – предельные значения; H – приращение, по умолчанию 1). Допускаются вложенные циклы, например:

```
for i=1:n                     for i=1:n-1
    for j=1:m                    for k=i+1:n
        a(i,j)=x(i)^j;             if a(i)<a(k)
    end                                m=a(i); a(i)=a(k);  a(k)=m;
end                                  end
                                   end
                               end
```

В заголовке цикла можно использовать одномерный массив. Так цикл

```
k=1;
for i=[0 5 7]
        x(k)=2^i;  k=k+1;
end
```

формирует массив X=[1 32 128].

Оператор цикла с предусловием имеет традиционную конструкцию:

```
while <условие>
        <команды>
end
```

и обеспечивает выполнение команд тела цикла, пока истинно проверяемое условие. Заметим, что работа цикла может быть прервана (выход из внутреннего цикла) оператором **break**:

```
while a<1
        … n=n+1;
         if   n>250
                break
end ...
```

Оператор переключения обобщает условный оператор на случай более двух условий и имеет конструкцию:

```
switch  <выражение>
    case <значение 1>
            <команды>
    case <значение 2>
            <команды>
        . . . . .
    otherwise                  % может отсутствовать
            <команды>
    end
```

Контрольные значения проверяются на равенство и могут задаваться и списком:

```
switch     k
        case 0
                t=1
        case (1, 2,5)
                t=2
        otherwise
                t=0
end
```

Выход из функции в вызывающую программу обеспечивается выполнением последнего ее оператора или командой **return**.

Кроме упомянутых основных операторов, традиционных для

любой системы программирования, остановимся на ряде операторов обеспечения пользовательского интерфейса.

Ввод с клавиатуры реализуется командой вида

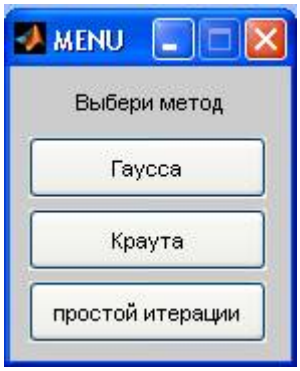
<переменная>= input (‘подсказка’)

Например, оператор x=input(' степень полинома='); выводит на дисплей текст подсказки и ожидает ввода арифметического выражения, имени встроенной функции или М-файла (в кавычках).

Приостановку работы программы можно предусмотреть включением в текст программы команды **pause** (приостановка до нажатия любой клавиши) или **pause (n)** (приостановка на *n* сек). Командой **pause off** режим приостановок можно отключить до выполнения указания **pause on** . Можно пользоватьсяи и командой **keyboard** (приостановка с возможностью выполнять практически любые команды и последующим возвратом в программу командой **return**).

Существует возможность создания **меню**:

<переменная>=**menu**(‘заголовок’,’выбор1’,’выбор2’,...)



Так, команда: k=**menu**(‘Выбери метод’, ’Гаусса’,’Краута’, ’простой итерации’) создаст в левом верхнем углу экрана всплывающее меню с указанными пунктами-клавишами и щелчок по клавише задаст значение переменной k, равное 1, 2 или 3.

Для вывода информации на дисплей здесь мы обратим внимание на команду **disp(**<переменная>) или **disp(**‘текст’), оставляя команду форматированного вывода **fprintf (**<формат>, A) на особое рассмотрение.

За пределами предлагаемого руководства остается многообразие операторов, связанных с отладкой и сигнализацией об ошибках, анализом списка аргументов и др.

Приведем несколько простейших примеров использования программного режима.

Как известно, для вычисления значений многих математических функций используются их аппроксимации через разложение в ряд Тейлора (в частности, в ряд Маклорена)

$$f(x_0+h)=f(x_0)+h\cdot f'(x_0)+h^2\frac{f''(x_0)}{2!}+h^3\frac{f'''(x_0)}{3!}+\ldots+h^k\frac{f^{(k)'}(x_0)}{\mathrm{k!}}+\ldots$$

Так известная в ряде приложений фунция Бесселя вещественного аргумента и нулевого индекса в окрестности x_0=0 имеет вид

$$J_0(x)=1-\left(\frac{x}{2}\right)^2+\frac{1}{2!^2}\left(\frac{x}{2}\right)^4-\frac{1}{3!^2}\left(\frac{x}{2}\right)^6+\ldots+(-1)^k\frac{1}{k!^2}\left(\frac{x}{2}\right)^{2k}+\ldots$$

Поскольку этот ряд знакочередующийся, то для вычисления его

значения при некотором *x* с погрешностью, не превышающей ε, достаточно суммировать его слагаемые (члены ряда) до тех пор, пока какой-то из них не станет меньше ε.

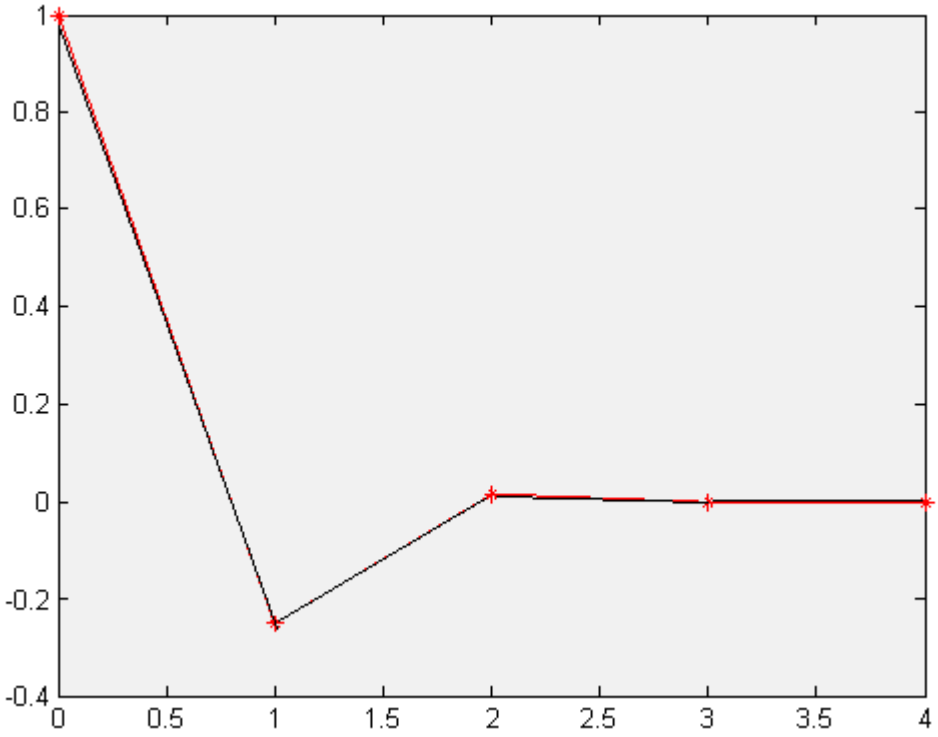
Если нам захочется оценить темп убывания значений членов ряда, достаточно описать функцию (файл *temp.m*):

```
function [f, k]=temp(x)
    k=0;   f(k+1)=1;
    while abs (f(k+1))>1e-4
        k=k+1;  f(k+1)= (-1)^k / (prod(1:k))^2*(x/2)^(2*k);
    end
```

и затем последовательно выполнить команды

```
X=1;      % задание аргумента
[F, n]=temp(X)  % с выводом оценок членов ряда и их количества
plot(0:n,F, 'r-*' )   %Линия - красная (r)сплошная(-), маркеры(*)
```

В итоге получаем F =[1.0000 -0.2500 0.0156 -0.0004 0.0000] , n =4 и в окне Figure1 график



Хотелось бы обратить внимание читателя на то, что в операторе f(k+1)= (-1)^k / (prod(1:k))^2*(x/2)^(2*k); присутствует три возведения в степень, что сказывается на затратах компьютерного времени, и поиск k!, что чревато потерей точности компьютерных значений (25!= 1.5511e+025 требует хотя бы 84-разрядной сетки). Этого можно избежать, если вычислять значение очередного члена ряда через пре-

дыдущий $f_k = -f_{k-1}\frac{x^2/4}{k^2}$.

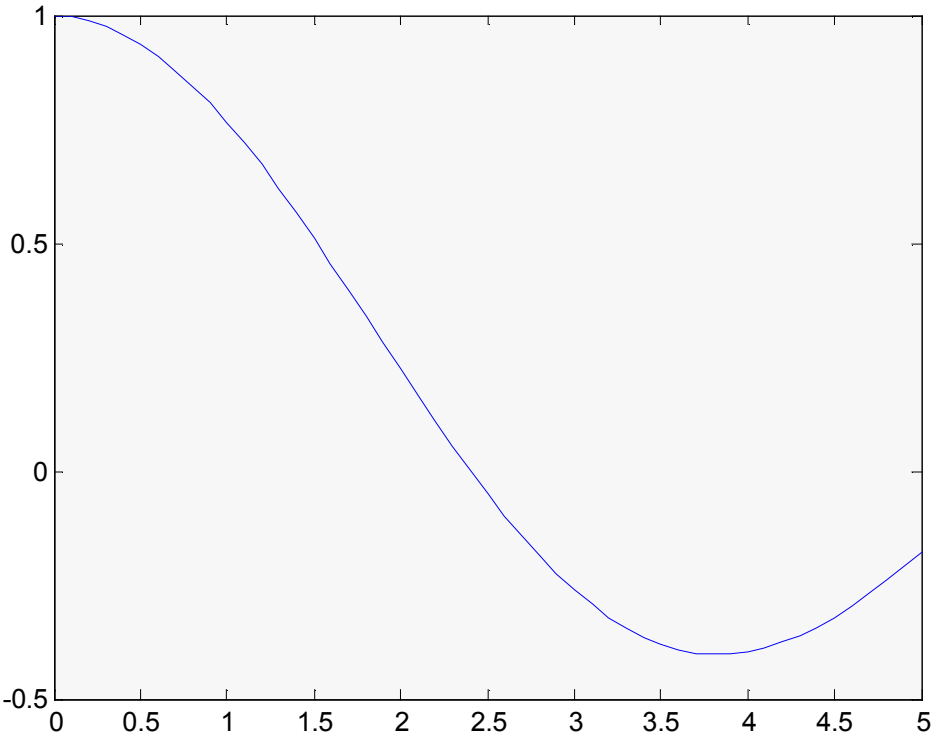
Если нас интересует поведение функции $J_0(x)$, то создаем

```
function  s=besselr0(x)
        k=0;   s=1;  f=1; d=-(x/2)^2;
        while abs (f) >1e-4
                k=k+1;  f= f*d / (k*k); s=s+f;
        end
```

и сценарий ***tbes.m***

```
m=0;
for x=0: 0.1:5
    m=m+1; R(m)=besselr0(x);
end
plot (0: 0.1:5, R)
```

обращение ***tbes*** к которому их командной строки дает картину



4. Операции над массивами

Как показано выше, массив можно создать прямым (построчным) перечислением его элементов подобно A=[1 3 5 7; 4 5 6 7] (матрица 2 на 4), B=[1; 3; 5; 7] или B=[1 3 5 7]‘ (столбец с 4 элементами) или заданием диапазона значений с заданным (или умалчиваемым единичным) шагом [1:2:7], [4:7], [[1:2:7]; [4:7]] и т.п.

Доступ к элементам или блокам элементов массива производится указанием индексов или массива индексов: A(2,k) – элемент второй

строки и k-го столбца; A(:,k) – k-й столбец; A(1:3; 1:4) – подматрица из первых 3 строк и 4 столбцов матрицы; C(:, :, 12) –12-я страница трехмерного массива.

Заметим, что хранение массивов в памяти ведется по столбцам. Поэтому можно работать с созданным многомерным массивом как с одномерным, например, A(:) – вектор-столбец из всех элементов массива A, A(13:17) – столбец из элементов с номерами от 13 до 17.

Имеется возможность **объединять массивы "по горизонтали"**- [A, B, C] или [A B C] (массивы с одинаковым числом строк) и **"по вертикали"** – [A; B; C] (массивы с одинаковым числом столбцов).

Существует возможность реализации логических операций: **объединения множеств** – **union**(X,Y), **пересечения** – **intersect**(X,Y), **разности** – **setdiff**(X,Y):

» a=[1 2 3 6]; » b=[1 3 7];	» union(a,b) ans = 1 2 3 6 7	»intersect(a,b) ans = 1 3	» setdiff(a,b) ans = 2 6

Для определение **длины вектора** используется функция **length** :

» k=length('Это строка') k = 10	» X=[1 0 -3 6 7 13]; » k=length(X) k = 6	» k=length([1 4 7 ; 2 0 -2]) k = 3

и для **размеров массива** – функцию **size :**

» X =[1 0 –3 ; 6 7 13] » k=size(X) k = 2 3	» [m, n]=size(X) m = 1 n = 6	» size([2 4 7]) ans = 1 3

Суммирование и умножение элементов массива можно реализовать функциями **sum**(A) и **prod**(A) (для двумерного массива выполняется поиск сумм и произведений по столбцам). С помощью функций **sum(A,dim)** и **prod(A,dim)** можно выполнить операции по конкретному измерению (dim=1– столбцы, 2 – строки). **Сортировку элементов массива по возрастанию** можно выполнить функцией **sort(A,dim)** , причем команда **[B,I]= sort(A)** выдает и список индексов. **Сортировку по убыванию** можно выполнить аналогичной функцией **sortrows.**

Среди других следует отметить и ряд функций комбинаторики:

perms(V) –перестановки всех элементов вектора V размерности *n* (массив размерности $n!\times n$) :

» perms (3:2:7)

ans =[7 5 3; 5 7 3; 7 3 5; 3 7 5; 5 3 7; 3 5 7]

» perms([3 2 7])

ans =[7 2 3; 2 7 3; 7 3 2; 3 7 2; 2 3 7; 3 2 7] ;

nchoosek (n,k) – число сочетаний из *n* по *k* $C_n^k = \dfrac{n!}{k!(n-k)!}$:

nchoosek (V,k) – массив всех сочетаний элементов вектора V:

» nchoosek([3 2 7],2) ans = [3 2; 3 7; 2 7];

Наряду с уже упоминавшимися функциями начального задания **zeros** и **ones**, стоит упомянуть функции:

rand(n), rand (m,n), rand (size(A)) - формирование массива чисел с равномерным законом распределения в (0,1);

randn(n), randn (m,n), randn (size(A)) -формирование массива чисел с нормальным законом распределения (Mx=0, Dx=1);

eye(n), eye(m,n), eye(size(A)) - формирование единичной матрицы (n×n, m×n, соразмерной с матрицей A).

Обратите внимание на полезные в приложениях конструкции.

Так команды **[**X,Y]=**meshgrid**(x,y), **[**X,Y]=**meshgrid**(x), [X,Y,Z]= **meshgrid**(x,y,z) выполняют преобразование векторов x, y, z в **двумерную (трехмерную) сетку - декартово пространство,** используемую при реализации графики). Например, [X,Y] = meshgrid(1:3,10:14) порождает

X	1	2	3	Y	10	10	10
	1	2	3		11	11	11
	1	2	3		12	12	12
	1	2	3		13	13	13
	1	2	3		14	14	14

Заметим,что вариант **[**X,Y]=**meshgrid**(x) тождественен **[**X,Y]= **meshgrid**(x,x).

Полезно иметь в виду существование функции текущего времени **clock** (6-элементный массив – дата и время). Так команда c = clock дает c =[2011 12 2 13 2 52]. В сочетании с командами **tic** (включить таймер) и **toc** (выключить таймер) можно проводить сравнительные оценки тех или иных алгоритмов по затратам времени.

5. Решение основных задач линейной алгебры

Ранее мы уже упоминали об арифметических операциях над матрицами, степенях и транспонировании. При решении задач матричной алгебры могут оказаться полезными общеизвестные функции:

det(A) – определитель квадратной матрицы;

rank(A) – ранг матрицы;

trace(A) – след матрицы (сумма элементов главной диагонали);

inv(A) –**обращение (инверсия**) **матрицы (**можно воспользоваться A^{-1}) .

При решении многих задач (например, при оценке сходимости вычислительных процедур) используется понятие **нормы** вектора

(матрицы). В Matlab'е для этого имеются функции **norm(A)** и **norm(A, k),** реализующие несколько видов норм для матриц и векторов.

Для вектора A т.н. k-ая норма определяется (по умолчанию k=2)

$$\|A\| = \sqrt[k]{\sum_{i=1}^{n} |A_i|^k};$$

при k=inf и k=-inf соответственно $\|A\| = \max(|A_i|)$ и $\|A\| = \min(|A_i|)$;

» v=[3 4 -10]	» norm(v) ans = 11.1803	» norm(v,2) ans = 11.1803	» norm(v,inf) ans = 10	» norm(v,-inf) ans = 3
	» norm(v,1) ans = 17	» norm(v,-1) ans = 1.4634	» norm(v,3) ans = 10.2946	» norm(v,'fro') ans = 11.1803

Если A – матрица, то норма определяется для k=1, 2, inf и fro (по умолчанию k=2):

$k=1$ – $\|A\| = \max_j \sum_{i=1}^{m} |A_{ij}|$; $k=2$ – $\|A\| = max(svd(A))$ – максимальное из сингулярных чисел матрицы (значений квадратных корней из собственных чисел матрицы A′A);

$k=inf$ – $\|A\| = \max_i \sum_{j=1}^{n} |A_{ij}|$; $k='fro'$ – $\|A\| = \sqrt{\sum_{i=1}^{n} B_{ii}}$, $B=A'*A$:

A = 1 2 3 5 4 3 3 4 3	» norm(A,1) ans = 10	» norm(A) ans = 9.6871	» norm(A,inf) ans = 12	» norm(A,'fro') ans = 9.8995

Для задачи ***решения системы линейных алгебраических уравнений,*** одной из популярнейших в вычислительной математике, предусмотрены даже "элементарные" операции.

Так для решения системы AX=B (A –матрица коэффициентов размерности n×n, B– матрица правых частей размерности n×k, X – матрица из k векторов-столбцов решений) можно использовать *команду обратного деления* "\". Например, для решения системы

$x_1+2\,x_2+3\,x_3 = 3$ (или 3)

$5\,x_1+4\,x_2+3\,x_3 = 9$ (или 9)

$3\,x_1+4\,x_2+3\,x_3 = 6$ (или 7)

задаем (построчно) матрицу коэффициентов и векторов правой части и, выполняем X=A\B, получая искомое решение.

» A=[1 2 3; 5 4 3; 3 4 3]	» B=[3 3; 9 9; 6 7]	» X=A\B
A = 1 2 3	B = 3 3	X =1.5000 1.0000
5 4 3	9 9	0.0000 1.0000

3 4 3	6 7	0.5000 0

При решении системы AX=B можно воспользоваться *операцией обычного деления*. Так решение той же системы

```
»X=B′/A′          X = 1.5000   -0.0000    0.5000
                      1.0000    1.0000         0
```

(обратите внимание на строчное представление решений).

Под кажущейся простотой решения скрывается достаточно серьезный анализ структуры матрицы и использование лучшего по точности и быстродействию алгоритма (метод Гаусса, разложение Холецкого и др.)

Для прямоугольной матрицы A (m≠n) решение строится по минимуму квадрата ошибки (используется QR-разложение на основе преобразований Хаусхолдера) и не сопровождается сообщениями о множественности решений или переопределенности системы:

одно уравнение с 3 неизвестными :	три уравнения с 2 неизвестными:
» a=[1 2 3]; » b=6; » x′=a\b x = 0 0 2	» c=[1 2; 3 7; 2 5]; » d=[3; 10 ; 9]; » x=c\d x = -5.00000000000002 3.66666666666667

Естественно, что квадратная матрица коэффициентов должна быть невырожденной (определитель отличен от нуля) и в противном случае выдается сообщение *Matrix is singular to working precision* и элементы решения принимают значения *inf* (не определено).

Особого упоминания заслуживает **обращение (инверсия) матрицы** путем возведения в степень –1 или функцией **inv(A)**:

» A=[1 2 3; 5 4 3; 3 4 3]	» C=inv(A)	» D=A^(-1)
A =	C =	D =
1 2 3	-0.0000 0.5000 -0.5000	-0.0000 0.5000 -0.5000
5 4 3	-0.5000 -0.5000 1.0000	-0.5000 -0.5000 1.0000
3 4 3	0.6667 0.1667 -0.5000	0.6667 0.1667 -0.5000

Напомним, что обращение матрицы может оказаться полезным при решении системы AX=B в виде $X=A^{-1}B$:

» B=[3 3; 9 9; 6 7]	» X=inv(A)*B
B = 3 3	X = 1.5000 1.0000
9 9	0 1.0000
6 7	0.5000 0.0000

При решении линейных систем и других задач интересно представление матрицы разложением на матрицы упрощенной структуры.

[L,U]=lu(A) – дает т.н. **LU-разложение** произвольной квадратной матицы в виде *произведения нижней и верхней треугольных матриц* A=LU (в матрице L возможны перестановки; главная диагональ L

принята единичной); такое представление позволяет, в частности, решение системы AX=B свести к двум простым системам LZ=B, UX=Z .

[L,U,P]=lu(A) – дает LU-разложение с выводом матрицы перестановок P такой, что PA=LU.

```
A =                  » [L,U]=lu(A)
 1   2   3           L =                          U =
 5   4   3             0.2000  0.7500   1.0000      5.0000  4.0000  3.0000
 3   4   3             1.0000       0        0           0  1.6000  1.2000
                       0.6000  1.0000        0           0       0  1.5000

» [L,U,P]=lu(A)
L =                          U =                          P =
  1.0000       0        0      5.0000  4.0000  3.0000      0   1   0
  0.6000  1.0000        0           0  1.6000  1.2000      0   0   1
  0.2000  0.7500   1.0000           0       0  1.5000      1   0   0
```

R=chol(A), [R,p] =chol(A) - дает разложение Холецкого для положительно определенной симметрической матрицы A=R′R, где R – верхняя треугольная матрица (в численном анализе этот прием называют методом квадратных корней). Если матрица A не является положительно определенной, то в первом варианте возникает сообщение об ошибке (использование здесь комплексных значений не предусмотрено) и во втором R – матрица порядка q=p-1:

```
C =                  » R=chol(C)
   1   2   3         ??? Error using ==> chol
   2   5   5         Matrix must be positive definite
   3   5   7

» [R,p]=chol(C)
 R =    1    2
        0    1
 p =    3
```

В этом примере первые два главных минора положительны, но det(C) = -3; соответственно, q=2 и R′R дает второй главный минор матрицы.

[Q,R]=qr(A), [Q,R,P]=qr(A), [Q,R]=qr(A,0) находит **QR-разложение** для прямоугольной матрицы размерности m×n:

[Q,R]=qr(A) – в виде A=QR произведения унитарной матрицы Q (QQ'=E) и верхней треугольной матрицы R :

```
C =                  » [Q,R]=qr(C)
   1   2   3         Q = -0.2672 -0.0514  -0.9622   R =  -3.7416 -7.2160 -9.0868
   2   5   5            -0.5345 -0.8229   0.1924           0 -1.3887  -0.3086
   3   5   7            -0.8017  0.5657   0.1924           0       0  -0.5773

t =                  » [Q,R]=qr(t)                  R =
   1   3   5           Q = -0.1961   -0.9805          -5.0990  -3.5301  -1.9611
   5   3   1              -0.9805    0.1961                0  -2.3534  -4.7068
```

[Q,R,P]=qr(A) отличается от предыдущего варианта упорядочением по убыванию модулей диагональных элементов R и наличием соответствующей матрицы перестановок P (A*P'=Q*R);

[Q,R]=qr(A,0) при m>n отличается тем, что вычисляются лишь n столбцов матрицы Q.

Если после выполнения QR-разложения выполнить команду **[Q1,R1]=qrdelete(Q,R,k),** то будет выполнен пересчет матриц для варианта, когда в матрице A удален k-й столбец. Если после QR-разложения выполнить команду **[Q1,R1]=qrdelete(Q,R,k,X),** то будут пересчитаны матрицы для варианта, когда в матрице A перед столбцом k вставлен столбец X.

X=nnis(A,B), X=nnis(A,B,t) позволяют искать **решение системы** AX=B **методом наименьших квадратов**, где отыскиваются *неотрицательные решения* X, минимизирующие norm(A*X-B) или гарантирующие точность ε при задании t=max(m,n)*norm(A,1)* ε.

Особое место в библиотеке занимают средства для **вычисления собственных чисел и векторов**.

В простейшем варианте отыскиваются ненулевые решения системы AX=λX командами **d=eig(A)** или **[X,d]=eig(A)** (d- диагональная матрица собственных чисел, X –матрица из нормированных собственных векторов):

a =	» d=eig(a)	» [R,d]=eig(a)	
1 2 3	d =	R =	d =
1 4 9	0.2179	0.8484 0.7163 -0.1198	0.2179 0 0
1 8 27	1.8393	-0.5150 0.6563 -0.3295	0 1.8393 0
	29.9428	0.1222 -0.2371 -0.9365	0 0 29.9428

Для проверки качества поиска (при значительных размерностях и многочисленных особых случаях такая проверка весьма желательна) достаточно проверить на близость к нулю значений A*X=R*D:

```
» a*R-R*d
ans =   1.0e-014 *
0.0583   0.0666  -0.7549
0.0166  -0.0444  -0.5329
0.0902  -0.3886  -0.7105
```

Решение задачи осуществляется на основе QR-алгоритма и его модификаций и при числе итераций, превышающем *30n*, может быть прервано с сообщением *Solution will not converge* (решение не сходится).

Проблему собственных значений можно решать не только для матрицы, но и для матричного полинома $(A_0+\lambda A_1+\lambda^2 A_2+...+\lambda^p A_p)X=0$ командой **[R,d]=polyeig(A_0,A_1,.., A_p),** где R- матрица размера n×(n×p) собственных векторов. При p=0 эта функция тождественна eig(A_0) , при p=1 – eig(A_0,-A_1) и в случае матриц с n=1 (скаляров) – **roots**(A_p,...,A_1,A_0), то есть ищет корни уравнения $A_p\lambda^p+...+A_2\lambda^2+A_1\lambda$

$+A_0=0$.

6. Операции над полиномами

Рассмотрим полином вида $P_n(x)=p_1x^n+p_2x^{n-1}+...+p_nx+a_{n+1}$. Соответственно будем обозначать P – $n+1$-мерный вектор коэффициентов, X – массив значений аргумента.

При **вычислении значений полинома** для элементов массива можно использовать функцию **polyval(P,X):**

```
» polyval([1 2 5],[0 3 1])
ans =
     5    20     8
```

```
» polyval([1 2 5],[0 3 1; 1 1 1])
ans =     5    20     8
          8     8     8
```

С помощью функции **polyvalm(P,X)** можно вычислять **значения матричного полинома** для квадратной матрицы X :

```
» polyvalm([1 2 5],[0 3 1; 1 1 1; 0 0 2])
ans =      8     9     7
           3    11     6
           0     0    13
```

Умножение полиномов $C_{m+n}(x)=P_m(x) \times Q_n(x)$ выполняется командой **C=conv(P,Q) –**

$$C_k = \sum_{j=max(1,k+1-n)}^{min(k,m)} A_j \quad B_{k+1-j}.$$

Деление полиномов реализуют командой **[C,R]= deconv(A,B)** , где C –частное и R – остаток от деления A на B.

```
» conv([1 2 3],[5 6])
ans =
     5    16    27    18
```

```
» [c,r]=deconv([1 2 3],[5 6])
c =   0.2000   0.1600
r =        0         0    2.0400
```

Вычисление производных от полинома, произведения и отношения полиномов производится соответственно командами **dp=polyder(P), dc=polyder(A,B)** и **[f,g]=polyder(A,B)**:

```
» polyder([1 -2 3 4 5])
ans =
     4    -6     6     4
```

```
» polyder([1 2 3],[5 6])
ans =
    15    32    27
```

```
» [f,g]=polyder([1 2 3],[5 6])
f =     5    12    -3
g =    25    60    36
```

Вычисление корней полинома реализуется функцией **roots(P)**, а **построение полинома по его корням** – функцией **poly(R)**.

```
» r=roots([1 3 5 7])
r = -2.1795
    -0.4102  + 1.7445 i
    -0.4102  - 1.7445 i
```

```
» poly(r)
ans =
    1.0000   3.0000   5.0000   7.0000
```

Функция **poly(A)** обеспечивает **построение характеристического полинома** |λE-A|=0 (см. проблему собственных значений):

```
» A= magic(3)                » P=poly(A)
A =   8   1   6              P =
      3   5   7                1.0e+002 *
      4   9   2                 0.0100  -0.1500   -0.2400   3.6000
```

В приложениях, связанных с преобразованием Лапласа при решении дифференциальных уравнений, оперируют с отношениями полиномов и представлениями их в виде простых дробей:

$$\frac{P_m(s)}{Q_n(s)} = \sum_{k=1}^{n} \frac{r_k}{s - s_k} + f(s),$$

где s_k - простые корни полинома $Q_n(s)$; если некоторый корень s_j имеет кратность m, то соответствующее слагаемое представляется в виде

$$\sum_{i=1}^{m} \frac{r_{j+i-1}}{(s - s_j)^i}$$

Команда **[r,s,f] =residue(P,Q)** дает **разложение отношения полиномов на простые дроби** (в случае близких корней возможна значительная погрешность). В случае кратного корня пользуются функцией **rj=resi2(P,Q,sj,m,j)**, где j –номер вычисляемого коэффициента (по умолчанию j=m); по умолчанию m=1 (простой корень).

Команда **[P,Q] =residue(r,s,f)** выполняет обратное действие свертки разложения в отношение полиномов. Выполнив действия

```
[r,s,f]=residue([1 -6 11 -6],[1 -5 4 0])
r =   0.5000    0   -1.5000
s =   4         1    0
f =   1
```

```
» [A,B]=residue([0.5 0 -1.5],[4 1 0],[1])
A =    1   -6   11   -6
B =    1   -5    4    0
```

видим, что

$$\frac{s^3 - 6s^2 + 11s - 6}{s^3 - 5s^2 + 4s} = \frac{0.5}{s-4} + \frac{0}{s-1} - \frac{1.5}{s} + 1$$

В случае кратного корня

```
» P=poly([1 2 3 ])
P =    1   -6   11   -6
» Q=poly([0 0 0 6])
Q =    1   -6    0    0    0
» [r,s,f]=residue(P,Q)
r =   0.2778   0.7222  -1.6667   1.0000
s =    6    0    0    0
f =    []
```

```
» r1=resi2(P,Q,0,3,1)
r1 =   0.7222
» r2=resi2(P,Q,0,3,2)
r2 =  -1.6667
» r3=resi2(P,Q,0,3,3)
r3 =    1
```

видим разложение:

$$\frac{s^3-6s^2+11s-6}{s^4-6s^3}=\frac{0.2778}{s-6}+\frac{0.7222}{s}-\frac{1.667}{s^2}+\frac{1}{s^3}$$

7. Коллекция тестовых матриц

Предлагаемая ниже крошечная часть коллекции тестовых матриц интересна как с позиций хотя бы дилетантского знакомства с итогами многовекового математического творчества, так и тестирования элементов собственных программных разработок.

hadamard (n) – матрица Адамара (ортогональная матрица из 1 и –1); значения *n*, *n*/*12* или *n*/*20* должны быть степенью 2:

```
» hadamard (4)
ans =1    1    1    1
      1   -1    1   -1
      1    1   -1   -1
      1   -1   -1    1
```

magic(n) – магический квадрат (квадратная матрица с элементами от 1 до n^2 с равными суммами элементов по строкам и столбцам):

```
» magic(2)
ans =    1    3
         4    2
```

```
» magic(3)
ans =    8    1    6
         3    5    7
         4    9    2
```

pascal(n) – матрица Паскаля - симметрическая матрица из коэффициентов разложения бинома $(1+x)^j$ (треугольника Паскаля)

```
     1
   1   1
 1   2   1
1   3   3   1
1   4   6   4   1
```

```
» pascal(3)
ans =    1    1    1
         1    2    3
         1    3    6
```

```
» pascal(5)
ans =
     1    1    1    1    1    1
     1    2    3    4    5    6
     1    3    6   10   15   21
     1    4   10   20   35   56
     1    5   15   35   70  126
     1    6   21   56  126  252
```

wilkinson(n) – матрица Уилкинсона (трехдиагональная симметрическая матрица *n*-го порядка, служащая тестом для алгоритмов решения проблемы собственных значений ; обычно берут *n*=21, где возникают кратные и близкие значения);

vander(X) – матрица Вандермонда (размерность совпадает с числом *n* элементов вектора *X* $V_{ij}=x_i^{n-j}$):

```
» wilkinson(4)
ans =
  1.5000   1.0000        0        0
  1.0000   0.5000   1.0000        0
```

```
x= [1 2 3 5]
» vander(x)
ans =    1    1    1    1
         8    4    2    1
```

0	1.0000	0.5000	1.0000	27 9 3 1
0	0	1.0000	1.5000	125 25 5 1

8. Анализ и обработка данных

8.1. Обработка статистических данных

Никакой анализ статистических данных не может обойтись без предварительной их обработки:

max (A) , min(A) – поиск экстремальных элементов по столбцам массива A;

max(A,B) , min(A,B) – формирование массива с элементами, равными экстремальным из соответствующих элементов массивов;

max(A,[],dim) , min(A,[],dim) – вектор экстремальных элементов по измерению dim;

[C,I]=max(...) , [C,I]=min(...) – дополнительно выводится строка индексов экстремальных элементов;

median(X) , median (X,dim) – медианы массива;

mean(X) , mean (X,dim) – средние значения;

std(X), std(X,flag) , std(X,flag,dim) – стандартное отклонение (flag=0 –несмещенная оценка σ; flag=1 – смещенная оценка s):

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}\ ;\ s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}\ ;$$

cov(X,Y) , cov(X,Y, flag) – ковариация для для массивов X и Y (каждый столбец – переменная, строка – наблюдение)

$$cov_{ij}(X,Y) = \frac{1}{f} X_i^T Y_j \ , f=n \text{ или } n\text{-}1;$$

cov(X) , cov(X,flag) – ковариация для столбцов массива X ;

corrcoef (X) , corrcoef (X,Y) –коэффициенты корреляции:

$$R_{ij} = \frac{cov_{ij}}{\sqrt{cov_{ii}\, cov_{jj}}}.$$

8.2. Численное дифференцирование

Как известно, многие методы численного дифференцирования строятся на использовании аппарата конечных разностей, поиск которых обеспечивается функциями:

diff(X), diff(X, n) – вычисление конечных разностей (первых или n-го порядка):

```
» F=     [   0     0.0998    0.1987    0.2955    0.3894    0.4794]
» D=diff(F)
D =      0.0998   0.0988    0.0969    0.0939    0.0900
» D2=diff(F,2)
D2 =  -0.0010   -0.0020   -0.0030   -0.0039
```

Для задач оптимизации градиентными методами полезны функции **gradient(F), gradient(F,h), gradient(F,h1,h2,...)** –приближенная оценка градиента функции *n* переменных с автоматическим или указанным выбором шага (одинаковым или разным по переменным): Например,

```
» [x,y]=meshgrid(-2:0.2:2, -2:0.2:2);  %   выбор сетки узлов
» z=x.*exp(-x.^2-y.^2);                 %  значения на сетке
» [px,py]=gradient(z,.2,.2);            %  градиент  в узлах сетки
» contour(z), hold on,quiver(px,py), hold off          % см.рис.8.1
```

Случай одной переменной реализуется как для двух на сетке y=x:

```
» x=meshgrid(-2:0.2:2);
» z=x.*exp(-x.^2);
» px=gradient(z,.2);
»  px(1,:)
```

дает последовательность 21 значений производной

```
 -0.169318 -0.217641 -0.316767 -0.401564 -0.426694
 -0.343801 -0.126816  0.202441  0.566120  0.852144
  0.960789  0.852144  0.566120  0.202441 -0.126816
...
```

8.3. Аппроксимация и интерполяция

polifit(X,Y,n) – аппроксимация таблично заданной функции F=F(X) полиномом *n*-й степени:

```
» X=0:0.1:0.5;   F=X.*sin(X);
P=polyfit(X,F,1)   % коэффициенты полинома (по убыванию степеней)
» FF=polyval(P,X)  % значения полинома в узлах X
» plot(X,F,'ob', X,FF,'-g'),grid,axis([0 0.5 -0.05 0.25])   %   см.рис.8.2
```

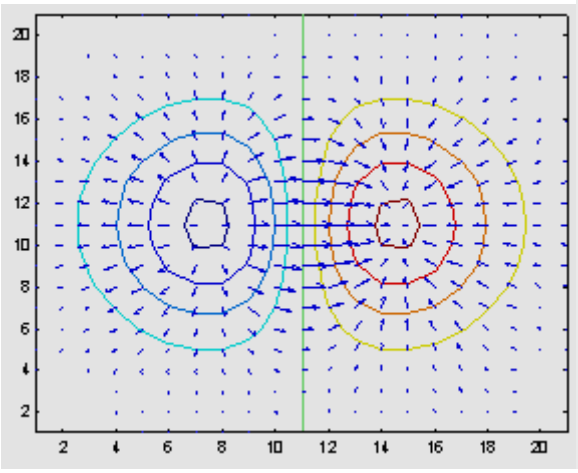


Рис. 8.1

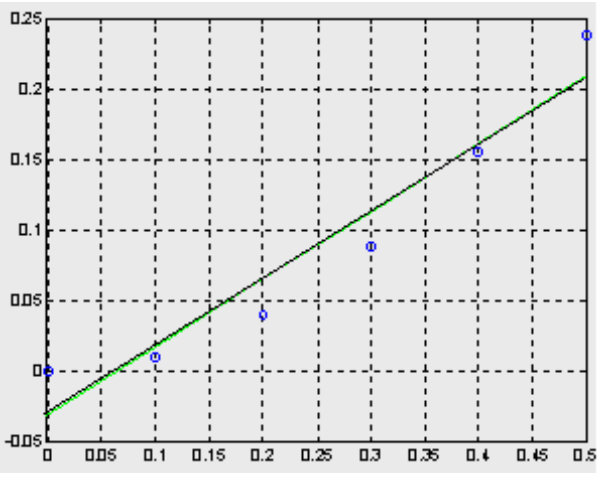


Рис. 8.2

interpft(Y,n,dim) – аппроксимация периодической функции на основе быстрого преобразования Фурье (Y – исходный массив значений функции; *n* – число узлов для итогового массива значений):

» X=0:10; Y=sin(X).^2.*exp(-0.1.*X);

» FX=interpft(Y,41); % Оценки в 41 узлах

» xp=0:0.25:10; plot(X,Y,'ob', xp,FX) % рис. 8.3

Заметим, что в библиотеке имеется богатый ассортимент средств аппроксимации на основе преобразования Фурье.

F=spline(X,Y,Z) – интерполяция табличной Y=Y(X) кубическим сплайном **F**(X) и вывод соответствующих значений для точек Z:

» x = 0:10; y = sin(x).*exp(-x./4); xx = 0:.25:10;

» fx = spline(x,y,xx); plot(x,y,'o',xx,fx); % рис. 8.4

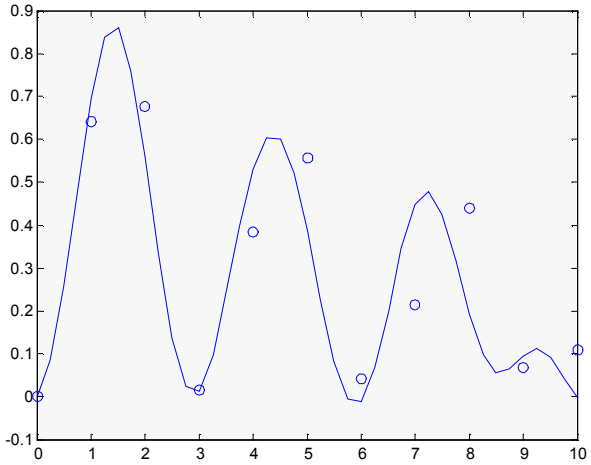


Рис. 8.3

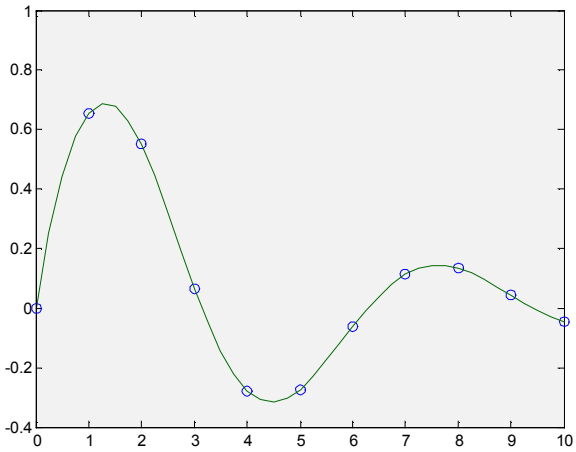


Рис. 8.4

Для получения большей информации используется конструкция **pp=spline(X,Y):** здесь командой **V=ppval(pp,Z)** можно найти значения в точках Z, а командой **[Xs, ABCD, m,L]=unmkpp(pp)** получить вектор узлов сплайна Xs, коэффициенты ABCD для подинтервалов, m=length(Xs), L=length(ABCD)/m :

```
>> x=0:10; y= sin(x).*exp(-x./4); pp=spline(X,Y);
>> xx = 0:.25:10; V=ppval(pp,xx);
 >> [Xs, ABCD, m,L]=unmkpp(pp)
Xs =    0    1    2    3    4    5    6    7    8    9   10
ABCD = -0.1328   0.0962   0.6773        0
       -0.1328  -0.3022   0.4713   0.6407
        0.5700  -0.7006  -0.5316   0.6769
       -0.4175   1.0094  -0.2228   0.0148
       -0.1269  -0.2430   0.5437   0.3839
        0.4317  -0.6236  -0.3229   0.5577
       -0.2248   0.6714  -0.2751   0.0428
       -0.1649  -0.0029   0.3933   0.2143
        0.2344  -0.4978  -0.1073   0.4398
        0.2344   0.2053  -0.3998   0.0691
m =    10
L =    4
```

interp1(X,Y,Z), interp1(X,Y,Z,'method') – одномерная табличная интерполяция (если Y двумерный массив, интерполяция ведется по каждому столбцу; значения Z должны входить в диапазон значений X). Можно указать метод интерполяции – кусочно-линейной (linear, по умолчанию), ступенчатой (*nearest*), кубической (*cubic*), кубическими сплайнами (*spline*). Функция **interp1q(X,Y,Z)** реализует быструю линейную интерполяцию на неравномерной сетке.

interp2(X1,X2,Y,Z1,Z2), interp1(X1,X2,Y,Z1,Z2,'method') – двумерная табличная интерполяция Y=Y(X1,X2), аргументы должны меняться монотонно и заданы в формате функции meshgrid.

```
» [X1,X2]=meshgrid(-1:0.1:1);  Y=exp(-X1.^2-X2.^2).*(1+X1+X2);
» [Z1,Z2]=meshgrid(-1:0.05:1); Y2=interp2(X1,X2,Y,Z1,Z2);
» mesh(X1,X2,Y),hold on,mesh(Z1,Z2,Y2+2),hold off      % Рис. 8.5
```

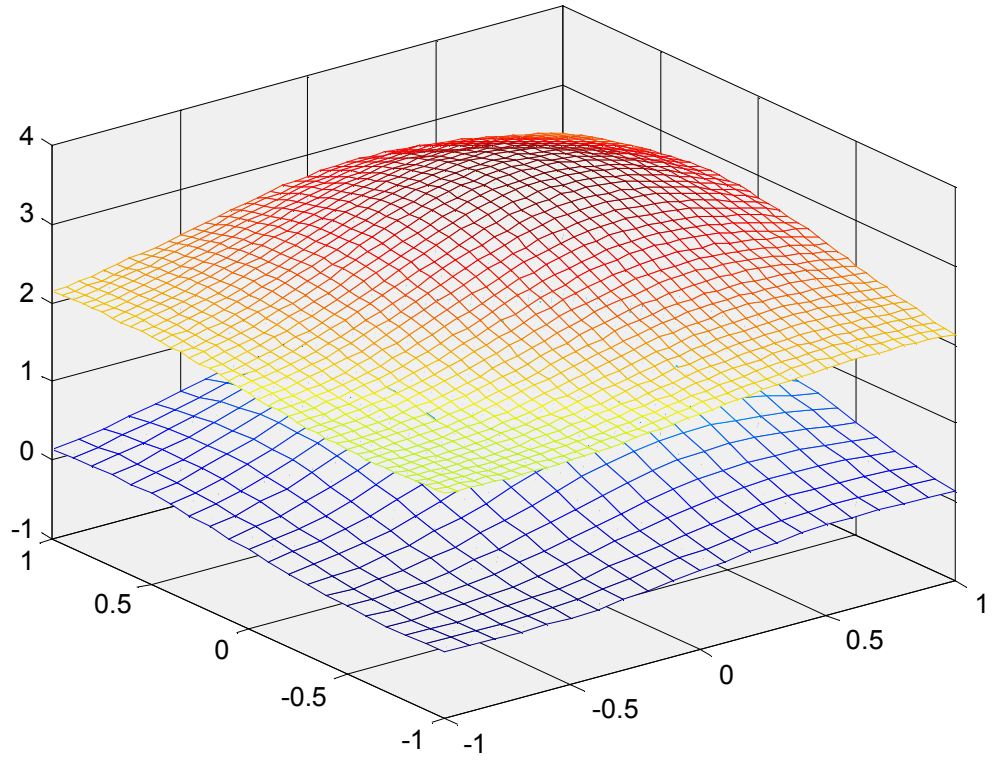


Рис. 8.5

interp3(X1,X2,X3,Y,Z1,Z2,Z3), interp3(..., 'method') – трехмерная табличная интерполяция Y=Y(X1,X2,X3);

interpn(X1,X2,...,Y,Z1,Z2,...), interp3(..., 'method') – многомерная табличная интерполяция Y=Y(X1,X2,...);

griddata(X1,X2,Y,Z1,Z2), griddata(X1,X2,Y,Z1,Z2, 'method') – двумерная табличная интерполяция на неравномерной сетке.

8.4. Численное интегрирование

polyarea (X,Y), polyarea (X,Y, dim) – площадь многоугольника с координатами вершин (X,Y):

```
» polyarea ([1 2 3 4 5],[0 3 6 3 0])
ans =    12
```

quad('имя', a,b), quad('имя', a,b, eps), quad('имя', a,b, eps, trace) - вычисление определенного интеграла: **a,b** – пределы интегрирования; **eps** – относительная погрешность (по умолчанию 10^{-6}); **trace** ≠0 – построение точечного графика функции; **'имя'** – имя подинтегральной функции (встроенной или М-файла); используется квадратура Симпсона. Допускается рекурсия до глубины 10. Может появиться сообщение – подозрение о сингулярности функции (наличии особенностей).

function f=integr(x)	function y = myfun(x)	»F = @(x)1./(x.^3-2*x-

f=x.*exp(-x);…	y = 1./(x.^3-2*x-5);	5);
» quad(‘integr’,0,2, 1e-5,1)	»Q =quad(@myfun,0,2)	»Q = quad(F,0,2)
ans = 0.5940	Q = -0.4605	Q = -0.4605

Ту же задачу решают функции **quadl**, базирующаяся на квадратуре Lobatto(лучше чем **quad в случае гладкихфункций)**, и **quadgk**, в основе которой лежит квадратура Гаусса-Кронрода, самая эффективная по точности для подинтегральных функций, опиисывающих колебательные процессы и быстро осциллирующих .

Для вычисления интегралов кратности 2 и 3 соответственно имеются функции

dblquad(‘имя’, a1,b1,a2,b2),
dblquad(‘имя’, a1,b1,a2,b2, eps),
dblquad(‘имя’, a1,b1,a2,b2, eps, <метод>)

и

triplequad(‘имя’, a1,b1,a2,b2,c1,c2)
triplequad (‘имя’, a1,b1,a2,b2,c1,c2,eps)
triplequad(‘имя’, a1,b1,a2,b2,c1,c2,eps,method)

8.5. Нули и экстремумы функций

Как известно, решение уравнений и поиск экстремумов функций – родственные задачи. Так решение системы $f_i(X)=0$, $i=1,...,n$ можно заменить поиском минимума $F(X)=\sum_{i=1}^{n} f_i^2(X)$, заведомо равного нулю, если система имеет решение. С другой стороны, поиск экстремумов функции можно свести к решению системы уравнений относительно нулевых значений ее производных.

Для некоторых классов функций имеются вполне универсальные методы решения: так для полиномов поиск корней реализуется без каких-то условий обобщенным методом Ньютона - функцией roots(...) .

В общем случае при великом многообразии методов решение подобных задач отнюдь не тривиально и требует определенного искусства. В Matlab’е реализованы функции лишь для простейших задач без гарантий получения решения.

Команды **x = fminbnd(fun,a,b), x = fminbnd(fun,a,b,options), [x,fx] = fminbnd(...), [x,fx,exitflag] = fminbnd(...), [x,fx,exitflag,output] =fminbnd(...)** реализуют поиск минимума функции одной переменной на интервале [**a,b**].

Например, [x,fx, flag] = fminbnd(@cos,3,4,optimset('TolX',1e-12, 'Display','off', 'MaxFunEvals', 200)) вычисляет точку минимума функции cos(x) в интервале [3,4] с точностью 10^{-12} и соответствующее значение функции fx . Параметр flag определяет условие завершения команды (1– соответствует условиям optimset, 0 – превышено максимальное число итераций 200 и др.). Если 'off' заменить на 'iter', происходит вывод на дисплей процесса итераций и использованного метода

Func-count	x	f(x)	Procedure
1	3.38197	-0.971249	initial
2	3.61803	-0.888633	golden
3	3.23607	-0.995541	golden
4	3.13571	-0.999983	parabolic
5	3.1413	-1	parabolic
6	3.14159	-1	parabolic
7	3.14159	-1	parabolic
8	3.14159	-1	parabolic
9	3.14159	-1	parabolic

Наличие параметра output позволяет получать информацию об использованном численном методе, количестве обращений к функции, числе итераций и условиях завершения.

Функция **fminsearch** ищет минимум функции нескольких переменных и реализуется в форматах, аналогичных функции **fminbnd x = fminsearch(fun,x0) , x = fminsearch(fun,x0,options), [x,fx] = fminsearch(...), [x,fx,exitflag] = fminsearch(...), [x,fx,exitflag,output] = fminsearch(...):** x0 –начальное приближение; как и в случае fminbnd решения могут быть только вещественными (в отличие от roots) и соответствовать локальному минимуму (гарантий глобальности нет).

Используется метод Нелдера-Мида (строится симплекс из n+1 вершины в n –мерном пространстве, берется новая точка внутри или вблизи симплекса и может заменить одну из вершин; процесс повторяется до малого диаметра симплекса).

Заметьте, что минимизируемая функция может зависеть от параметра и оформление команд аналогично примеру

```
function f = myfun(x,a)
        f = x(1)^2 + a*x(2)^2;
» a = 1.5;  X = fminsearch(@(x) myfun(x,a),[0,1])
```

или

```
a = sqrt(2);
 myfun = @(x)100*(x(2)-x(1)^2)^2+(a-x(1))^2;
 [x,fx] = fminsearch(myfun, [-1.2, 1], optimset('TolX',1e-8));
```

(выводится x =[1.4142 2.0000] и fx = 4.2065e-018

Функция **fzero** ищет корень функции одной переменной и реализуется в форматах **x = fzero(fun,x0), x = fzero(fun,x0,options), [x,fx] = fzero(...), [x,fx,exitflag] = fzero(...), [x,fx,exitflag,output] = fzero(...). Так команда** X=fzero(@(x)sin(x*x)-0.5,pi/4) дает X = 0.7236.

Параметр x0 определяет начальное приближение или интервал поиска с разнми знаками f(x) на его концах, options подобны опциям fminbnd. В случае неудачного поиска значение x =NaN.

Функция **sqnonneg (A,B)** ищет неотрицательные решения переопределенной системы линейных уравнений AX=B методом наименьших квадратов.

Не останавливаясь на других функциях, связанных с установкой режимов оптимизации, заметим, что в случае двух переменных существенную помощь для выбора начального приближения может оказать функция **gradient** (см. 8.2) и **contour**(...), которая будет рассмотрена при ознакомлении с функциями графики.

8.6. Обыкновенные дифференциальные уравнения

В системе MatLab предусмотрена многочисленная группа функций для решения задачи Коши для систем обыкновенных дифференциальных уравнений, заданных как в явной форме $\frac{dx}{dt} = F(t,x)$ так и в неявной $M(t,x)\frac{dx}{dt} = F(t,x)$ – т.н. *решатели ОДУ* (solver ODE), обеспечивающие пользователю возможность выбора метода, задания начальных условий и др.

В простейшем варианте достаточно воспользоваться командой [T,X]=solver('fun', [t_o , t_k], X0, ...), где значения t_o и t_k определяют диапазон интегрирования, X0 – вектор начальных значений, fun – имя функции вычисления правых частей системы, solver – имя используемой функции (**ode45** – метод Рунге-Кутта 4 и 5-го порядков, **ode23** – тот же метод 2 и 3-го порядков, **ode113** – метод Адамса для нежестких систем, **ode23s, ode15s** – для жестких систем и др.). Версии решателя различаются используемыми методами (по умолчанию относительная погрешность $\mathbf{10^{-3}}$ и абсолютная $\mathbf{10^{-6}}$) и соответственно временем и успешностью решения. Под жесткостью здесь понимается повышенное требование к точности – использование минимального шага во всей области интегрирования. При отсутствии информации о жесткости рекомендуется попытаться получить решение посредством ode45 и затем ode15s. При задании диапазона в виде [t_o , t_k] число элементов в выходных массивах T и X определяется необходимым для обеспечения точности шагом; при задании его в виде [t_o , t_1, t_2, ... , t_k] или [t_o :Δt: t_k] – указанными здесь значениями.

Например, в простейшем варианте решение уравнения $\frac{dx}{dt} = t \cdot e^{-t}$ в интервале t ∈[0, 0.5] с начальным условием x(t=0)=1 :

```
function  f  =texp(t,x)
          f=t*exp(-t);
» [T,X]=ode23 (‘texp’, [0, 0.5], 1)
T =    0         0.0500   0.1000   0.1500   0.2000   0.2500   0.3000
       0.3500   0.4000   0.4500   0.5000
X =    1.0000   1.0012   1.0047   1.0102   1.0175   1.0265   1.0369
       1.0487   1.0616   1.0754   1.0902
```

При решении задачи $\frac{d^2x}{dt^2} = x + 2e^t, x(0) = 0, \frac{dx}{dt}|_{t=0} = 1$, сводимой к пешению системы $\frac{dx_1}{dt} = x_2; \frac{dx_2}{dt} = x_1 + 2e^t$ с начальными условиями $x_1(0)=0$, $x_2(0)=1$ при $t \in [0, 2]$ задаем

```
function f = odu2(t,x)
f = zeros(2,1);
f(1) = x(2);   f(2) =   x(1)+2 *exp(t);
```

и оператором [T,X] = ode15s(@odu2,[0:0.5: 2.5],[0 1]) формируем

```
T’   0   0.5000    1.0000   1.5000    2.0000   2.5000
X'   0   0.8246    2.7195   6.7262   14.7931  30.5020
  1.000  2.4734    5.4379  11.2081   22.1834  42.6873
```

Для иллюстрации решения систем и ряда нестандартных возможностей рассмотрим *задачу выравнивания цен* по уровню актива в следующей постановке.

Предположим, что изменение уровня актива y пропорционально разности между предложением s и спросом p, т.е. $y'=k(s-d)$, $k>0$, и что изменение цены z пропорционально отклонению актива y от некоторого уровня y_0, т.е. $z'=-m(y-y_0)$, $m>0$. Естественно, что предложение и спрос зависят от цены, например, $s(z)=az+s_0$, $d(z)= d_0 - cz$. Соответственно возникает система дифференциальных уравнений

$$y' = k\cdot(s(z)-d(z))$$
$$z' = - m\cdot (y-y_0) .$$

Вычисление правых частей оформляем функцией:

```
function f=odu2(t,X)
    y=X(1);  z=X(2); a=20; c=10;    s0=10; d0=50; k=0.3; m=0.1;
    s=a*z+s0; d= d0-c*z;    y0=19;
    f(1)=k*(s-d) ; f(2)=-m*(y-y0);
    f=f';                          % вектор-столбец
```

Если выполнить решение при $y_0= 19$, $z_0=2$

```
» [T,Y]=ode45('odu2', [0:0.3:9],[19  2]);
» [T Y]
» plot(T,Y)
```

будет выведена таблица значений искомых функций:

```
ans =      0         19.0000   2.0000
```

```
0.3000  20.7757   1.9731
0.6000  22.4101   1.8949
0.9000  23.7686   1.7714
1.2000  24.7427   1.6126  ...
```

и их графики (рис. 8.6).

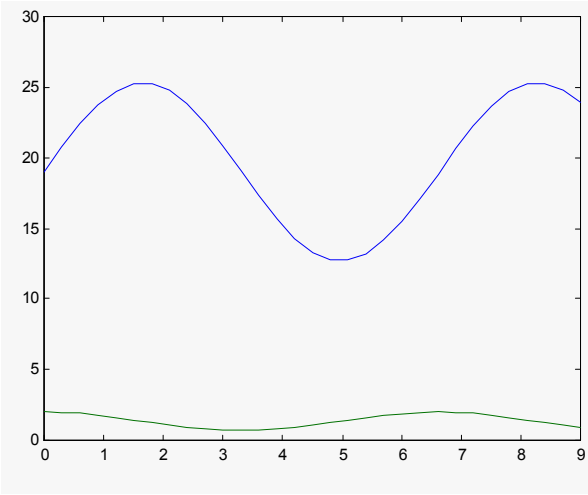


Рис.8.6

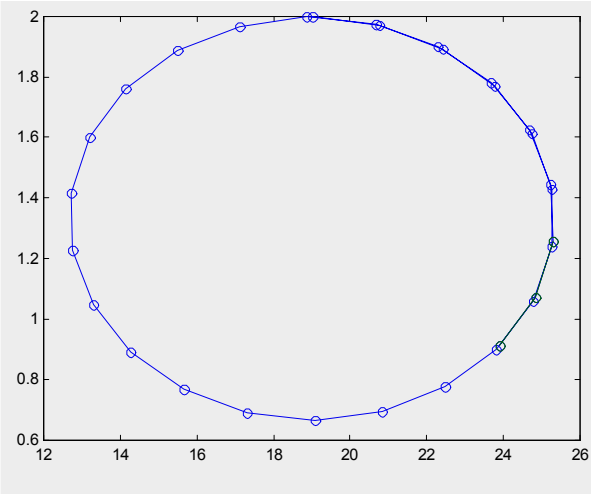


Рис.8.7

Эта система уравнений относится к числу т.н. *автономных* (или *динамических*), ибо независимая переменная t в нее явно не входит; соответственно можно установить связь между найденными решениями: в параметрическом задании линия $y=y(t)$, $z=z(t)$ определяет *фазовую кривую* (*траекторию*) системы − гладкую кривую без самопересечений, замкнутую кривую или точку, которая позволяет судить об устойчивости системы.

Так, установив опции к построению двумерного фазового портрета (функция **odephas2**) и номера переменных состояния :

```
» opt=odeset('OutputSel',[1 2], 'OutputFcn','odephas2');
» [T,Y]=ode45('odu2', [0:0.3:9], [19  2], opt);
```

получаем фазовый портрет системы, свидетельствующий о ее устойчивости – гармонии между активом и ценами (рис.8.7).

Другим примером подобных задач служит известная *задача динамики популяций* − модель взаимодействия “жертв” и “хищников”, в которой учитывается уменьшение численности представителей одной стороны с ростом численности другой. Модель была создана для биологических систем, но с определенными корректурами применима к конкуренции фирм, строительству финансовых пирамид, росту народонаселения, экологической проблематике и др.

Эта модель Вольтерра-Лотка с логистической поправкой описывается системой уравнений

$$\frac{dx_1}{dt} = (a - bx_2)x_1 - \alpha x_1^2$$

$$\frac{dx_2}{dt} = (-c + dx_1)x_2 - \alpha x_2^2$$

с условиями заданной численности “жертв” и “хищников” в начальный момент *t=0*.

Решая эту задачу при различных значениях α, получаем различные фазовые портреты (рис.8.8 и 8.9) − обычный колебательный процесс и постепенная гибель популяций .

```
function f=VolterraLog(t,x)
a=4;  b=2.5;   c=2;  d=1;  alpha=0.1;
f(1)= (a-b*x(2))*x(1)-alpha*x(1)^2;
f(2)= (-c+d*x(1))*x(2)-alpha*x(2)^2;      f=f';
» opt=odeset('OutputSel',[1 2], 'OutputFcn', 'odephas2');
» [T,X]=ode45('VolterraLog', [0  10],[3  1],opt );
```

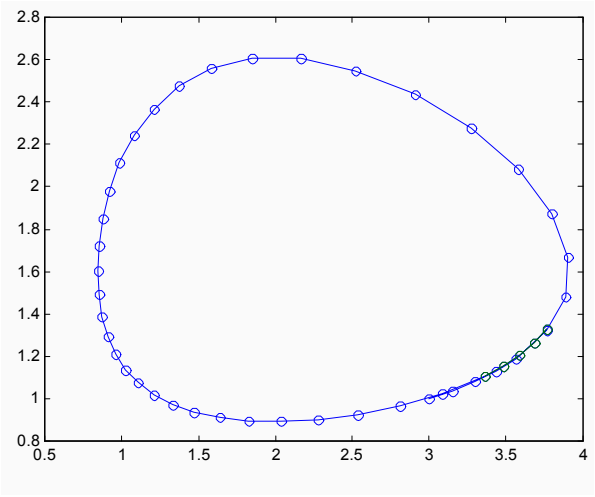


Рис.8.8 (α=0)

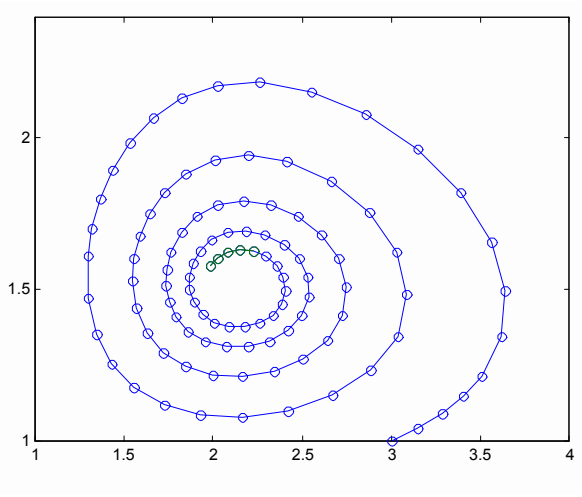


Рис.8.9 (α=0.1)

Имеется возможность построения и трехмерного фазового портрета с помощью функции odephas3. Например, решение задачи Эйлера свободного движения твердого тела:

$$\frac{dx_1}{dt} = x_2 x_3, \quad \frac{dx_2}{dt} = -x_1 x_3, \quad \frac{dx_3}{dt} = -0.51 \cdot x_1 x_2;$$

$$x_1(0) = x_2(0) = x_3(0) = 0$$

выступает в виде:

```
function f=Eiler(t,x)
    f(1)=  x(2)*x(3);   f(2)=  -x(1)*x(3) ;   f(3)=  -0.51*x(1)*x(2) ; f=f';
» opt=odeset('OutputSel',[1 2 3], 'OutputFcn', 'odephas3');
» [T,X]=ode45('Eiler', [0  7.25], [0  0 1], opt );          % рис.8.10
» [T,X]=ode45('Eiler', [0:0.25:7.25],[0 1  1]);
» plot(T,X)                                                  % рис.8.11
```

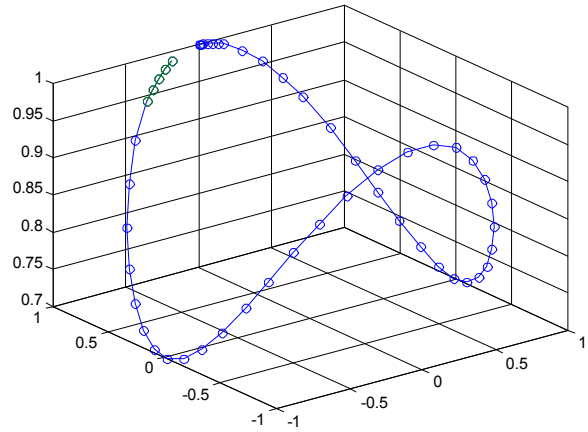


Рис.8.10

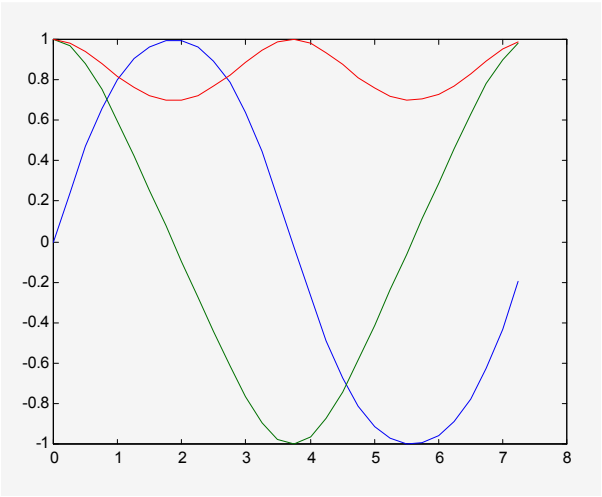


Рис.8.11

9. Элементарная графика

В систему MatLab входит графическая подсистема Handle Graphics, которая поддерживает своими установками (дескрипторами) двухмерную и трехмерную пользовательскую графику более низкого уровня. В ее составе такие графические объекты как Roots, Figure, Axes, Line, Patch, Surface, Text, знакомство с которыми потребовало бы существенного времени. Потому мы ограничимся здесь средствами простейшей пользовательской графики со ссылками на избранные дескрипторы.

9.1. Двумерная графика

line(X,Y), line(X,Y,Z) line(X,Y,Z, 'PropertyName', property-value,...) – построение прямой линии. Так последовательность line(x,y); line(x1,y1); создает в окне Figure два прямолинейных отрезка.

plot (y) – построение сглаженного графика одномерного массива в зависимости от номера элемента (для двумерного массива строятся графики для столбцов);

plot (x,y) – построение сглаженного графика значений функции $y=y(x)$; при двумерном x строятся графики $x=x(y)$; если оба массива двумерные, строятся зависимости для соответствующих столбцов;

plot (x,y, LineSpec) – построение сглаженного графика с заданием строки LineSpec (до 3 символов), определяющей стиль линий, форму маркера точек и цвет линий и маркера:

Символ стиля линии		Цвет		Цвет	
Непрерывная	-	Желтый	y	Зеленый	g

Штриховая	--	Фиолетовый	m	Синий	b
Двойной пунктир	:	Голубой	c	Белый	w
Штрихпунктирная	-.	Красный	r	Черный	k

Маркер может определяться символами : **.** + * ° × **s** (квадрат) **d** (ромб) **p** (пятиугольник) **h**(шестиугольник) **v** ^ < > (стрелки). По умолчанию выбирается непрерывная линия с точечным маркером и чередованием цветов с желтого по синий.

plot (x1,y1,LineSpec1,x1,y1, LineSpec2,...) – строит на одном графике несколько линий (диапазон по аргументу – объединение x1 и x2;

plot (...,'PropertyName',PropertyValue,...) – задает свойства графического объекта Line (толщину линий LineWidth, размер маркера MarcerSize, цвет маркера MarcerFaceColor и и др.) .

```
» x=0:0.3:6;   y=besselj(0,x);                % функция J0(x)
» x1=0:0.4:8;  y1=besselj(1,x1);              % функция J1(x)
» plot(x,y,'-sk', x1,y1,'-pk','LineWidth',1 )   % рис.9.1
```

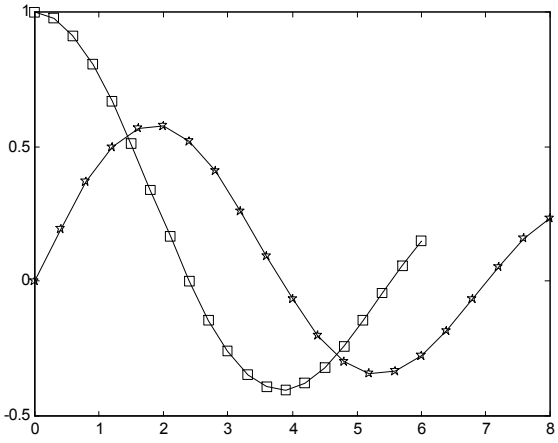


Рис.9.1

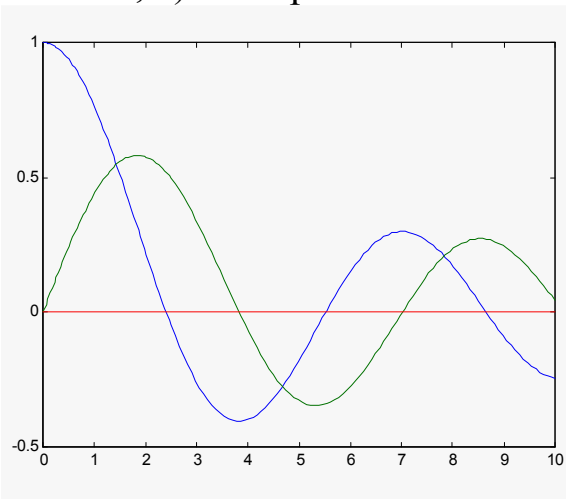


Рис.9.2

fplot(<имя функции>,limits) строит график функции (функций) в интервале limits=[xmin,xmax]. В качестве имени функции может использоваться М-файл или строка типа ′sin(x)′, ′[sin(x) cos(x)] ′, ′ [sin(x), myfun1(x), myfun2(x)] ′. Можно установить размеры графика по оси значений функции limits=[xmin,xmax ymin ymax].

fplot(<имя функции>,limits, eps) строит график с относительной погрешностью eps (по умолчанию 0.002) и максимальное число шагов (1/eps)+1. Эту конструкцию можно дополнить четвертым параметром **n** (n+1 – минимальное число точек) и параметром LineSpec:

```
» fplot( '[besselj(0,x) besselj(1,x) 0]',[0 10],[],20)  & рис.9.2
```

ezplot('f(x)') строит график f(x), заданной символьным выражением (например, ezplot('x^2-2*x+1')), на интервале [-2π 2π] с выводом выражения в качестве заголовка графика.

ezplot('f(x)', limits) и **ezplot('f(x)', limits, fig)** строят график f(x) на указанном интервале и в заданном окне.

polar(t,r) и **polar(t,r, LineSpec)** строят график **в полярных координатах** x=r·cos(t), y=r·sin(t), где **r =r(t)** и **t**– массив значений угла:

```
» f=0:0.01:2*pi;
» r=sin(2.*f).*cos(2.*f);
» hp=polar(f,r),hold on
» set(hp,'LineWidth',4) % рис.9.3
```

```
» f=0:0.01:12*pi;
» r=exp(-0.1*f);
» hp=polar(f,r) ,hold on
» set(hp,'LineWidth',2)  % рис.9.4
```

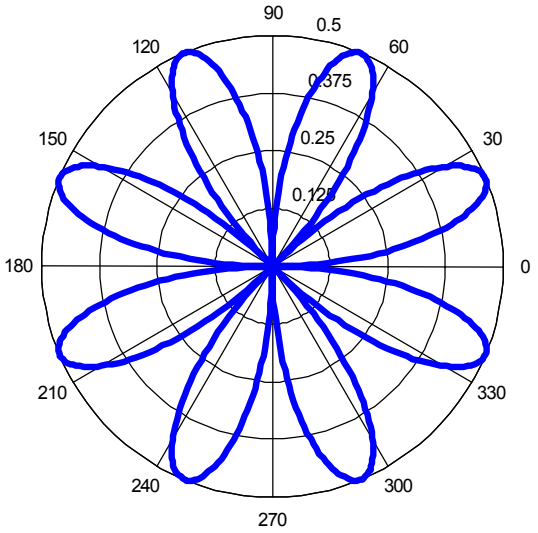


Рис.9.3

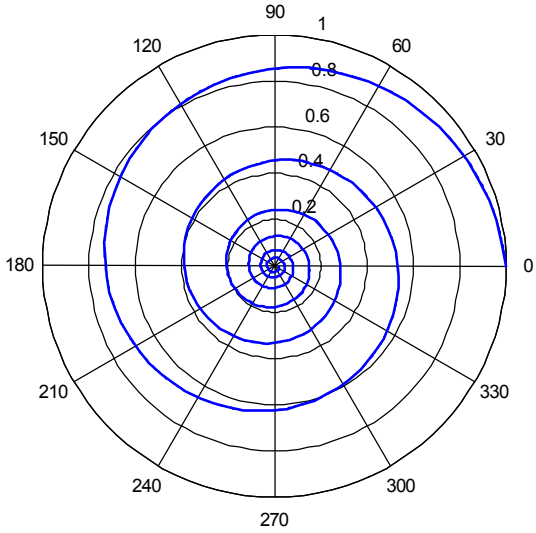


Рис.9.4

График в логарифмическом масштабе задается функцией **loglog** с тем же набором параметров, что и **plot**, с той лишь разницей, что по обеим координатам проводится масштабирование десятичным логарифмированием. **График в полулогарифмическом масштабе** задается функциями **semilogx** и **semilogy** с тем же набором параметров, что и **plot** (масштабирование по одной из координат).

График с двумя осями ординат (одна слева, другая справа) реализуется функцией **plotyy(x1,y1, x2,y2)** и той же функцией с добавлением параметров масштабирования ‘f1’ или ‘f1’,’f2’, в роли которых могут выступать plot, semilogx, semilogx, loglog:

```
» x=0:0.01:12*pi;
» plotyy(x, sin(x).*exp(-0.1.*x), x, 10*exp(-0.1.*x))   %  Рис.9.5
```

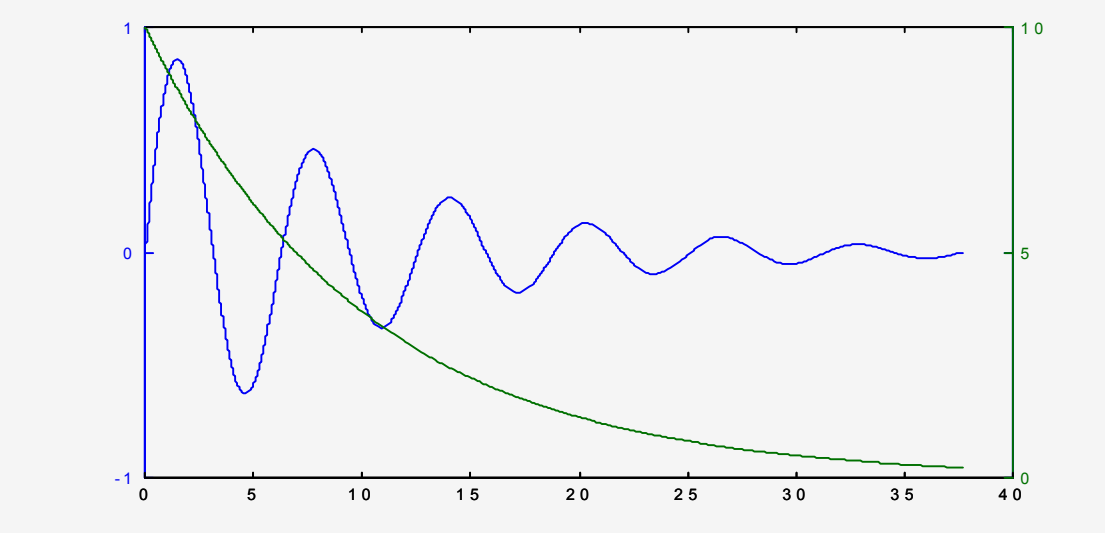


Рис.9.5

9.2. Трехмерная графика

В трехмерной графике выполняются представления функции z=z(x,y), отличающиеся способом соединения точек: линия, сечения, сетчатая или сплошная поверхность.

plot3(x,y,z) в тех же вариациях, что и plot, предполагает задание одномерных и двумерных массивов (строятся точки с координатами x(i,:),y(i,:),z(i,:) для каждого столбца и соединяются прямыми линиями. Если используется **[x,y]=meshgrid(...)**, то строятся сечения.

```
» t=0:pi/50:10*pi;
» plot3(sin(t),cos(t),t) % Рис.9.6
```

```
» [x,y]=meshgrid([-2:0.1:2],[-2:0.01:2]);
» z=exp(-x.^2-y.^2);
» plot3(x,y,z)        %  Рис.9.7
```

mesh(x,y,z,c), mesh(z,c), mesh(z) определяют задание сетчатой поверхности (массив **c** определяет цвета узлов поверхности; если **x,y** не указаны, то x=1:n, y=1:m, где [m,n]=size(z).

```
» [x,y]=meshgrid(-8:0.5:8);  t=sqrt(x.^2+y.^2)+0.001;
» z=sin(t)./t;   mesh(x,y,z)                              %  Рис.9.8
```

Аналогичная функция **meshc** в дополнение к поверхности строит проекции линий уровня, а **meshz** делает срез поверхности до нулевого уровня (своеобразный пьедестал).

```
» meshc(x,y,z)   &Рис.9.9          » meshz(x,y,z)     &Рис.9.10
```

surf(x,y,z,c), surf(z,c), surf(z) определяют задание сплошной поверхности, отличаясь от mesh системой окраски; аналогичная функция **surfc(...)** задает проекции линий уровня.

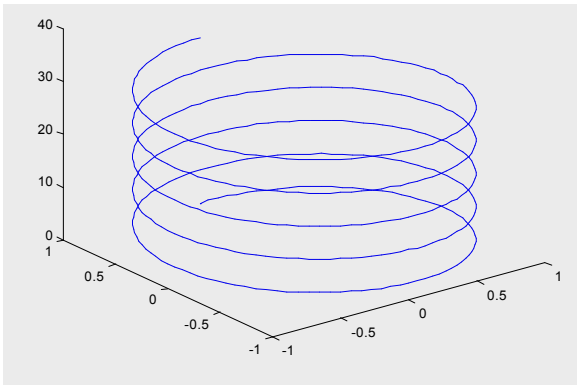


Рис.9.6

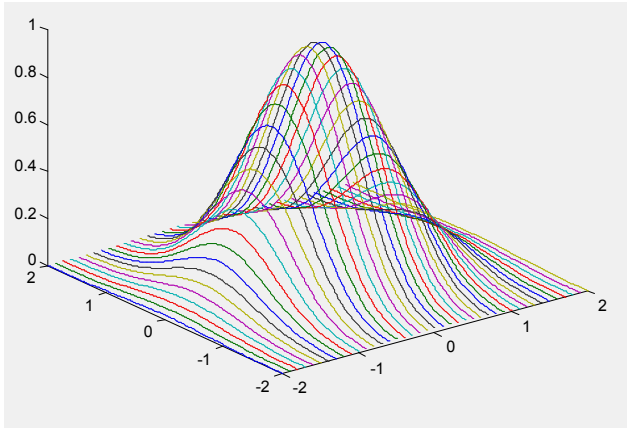


Рис. 9.7

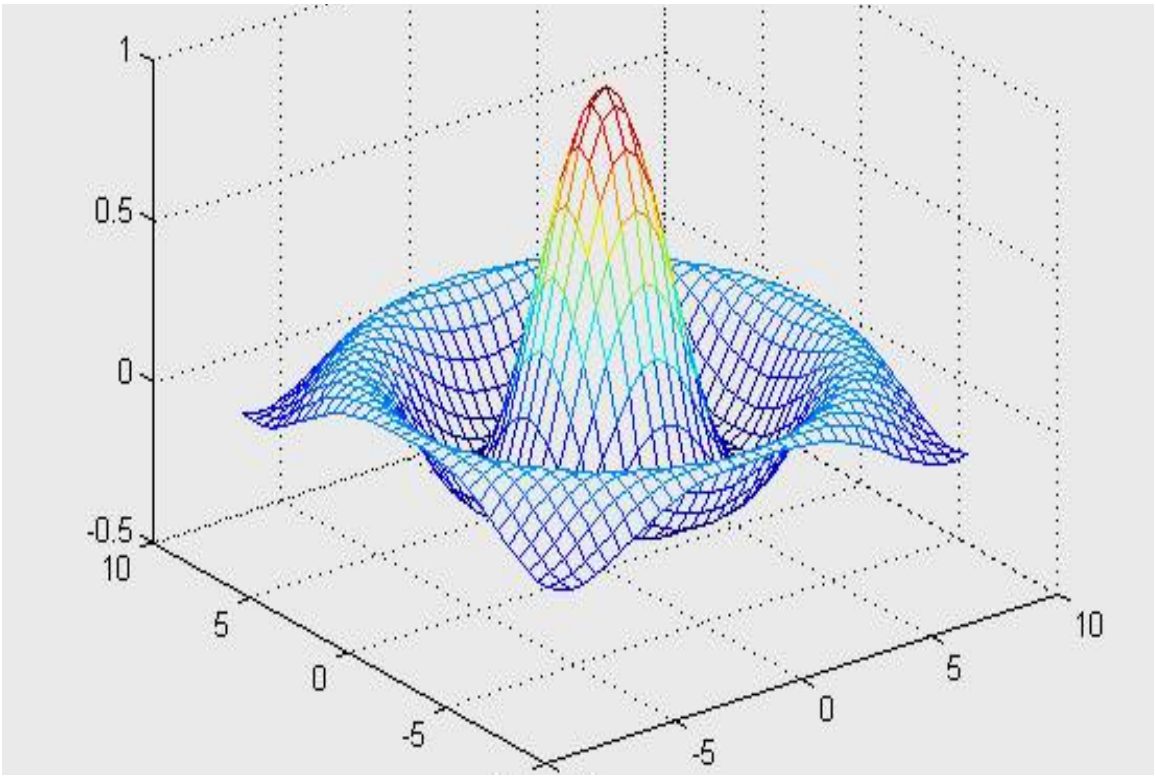


Рис.9.8

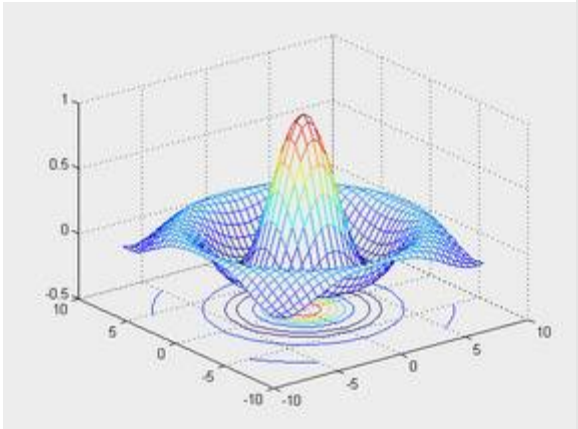


Рис.9.9

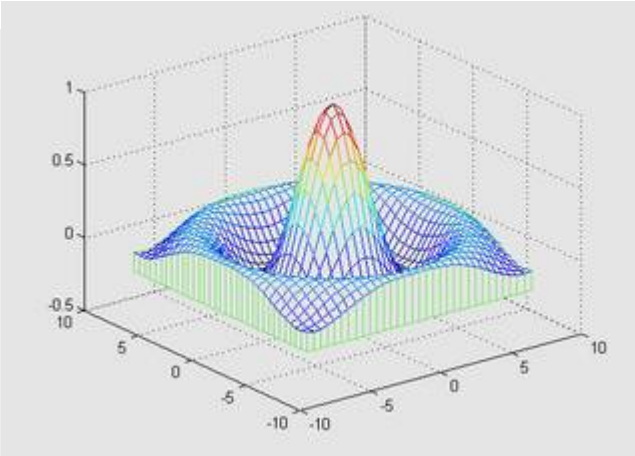


Рис.9.10

Реализация трехмерной графики может сопровождаться множеством вспомогательных команд, например:

hidden on/off включает или выключает режим удаления невидимых линий (по умолчанию on);

shading faceted / **flat** / **interp** устанавливает затенение поверхностей (по умолчанию faceted дает равномерную окраску ячеек с черными гранями, flat – цветами узлов сетки, interp – интерполяцией цветов.

9.3. Задание осей координат

axis([xmin xmax ymin ymax]), axis([xmin xmax ymin ymax zmin zmax]) устанавливает масштаб по осям;

axes off / on выключает (включает) вывод на координатные оси обозначений и маркеров;

grid on/off , grid включает (выключает) или переключает режим нанесения координатной сетки на осях;

box on/off, box включает (выключает) или переключает режим рисования контура параллелепипеда, трехмерный объект;

zoom on/off включает (выключает) **режим интерактивного масштабирования графиков** (левая мышь около точки увеличивает масштаб вдвое, правая – уменьшает; удержанием левой мыши можно выделить прямоугольную область для детального просмотра; **zoom out** восстанавливает исходный график.

9.4. Линии уровня

В отличие от **meshc (...)** и **surfc(...)** функция **contour** рисует только линии уровня соответствующих поверхностей и выступает в многообразии синтаксических форм: **contour(X,Y,Z) -** для массива Z =Z(X,Y), **contour(X,Y,Z,n) –** то же с указанием числа линий уровня (по умолчанию 10), **contour(X,Y,Z,v) –** то же для массива указанных значений ; **contour(Z), contour(Z,n), contour(Z,v)** – аналогичные функции без указания диапазонов для аргументов и **contour(...,LineSpec) -** аналогичные функции с указанием типа и цвета линий (см. plot); **[C,h]=contour (...)** возвращает массив C и вектор дескрипторов, позволяя тем самым продолжить работу с рисунком (давать оцифровку линий, заголовки и др.).

Функция **contourf(...)** закрашивает области между линиями уровня, аналогична contourf(...) с разницей в формате**[C,h, cf]=contour (...),** где cf определяет матрицу раскраски.

```
» [x,y]=meshgrid(-8:0.5:8);
» t=sqrt(x.^2+y.^2)+0.001;
» z=sin(t).^3./t;
» [c,h]=contour(x,y,z,20);
```

```
» [x,y]=meshgrid(-2:0.25:2);
» t=sqrt(x.^2+y.^2)+0.001;
» z=sin(t).^3./t;
» [c,h,cf]=contourf(x,y,z,4);
```

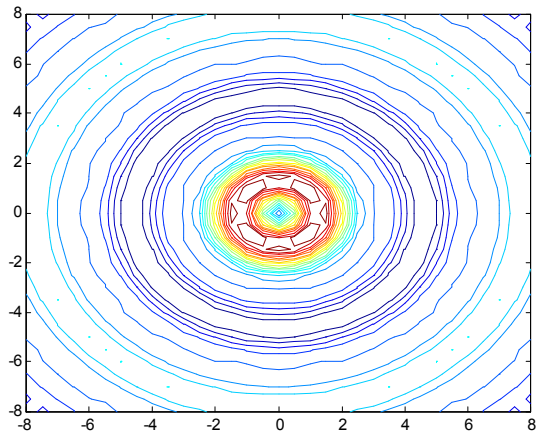


Рис.9.11

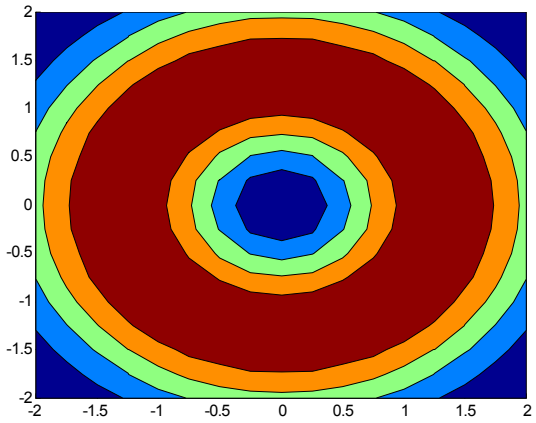


Рис.9.12

Функция **contour3(...)** по синтаксису полностью аналогична contour(...), но изображает не проекции линий уровня, а рисует их в пространственной интерпретации; так команда [c,h]=contour3(x,y,z,20); дает фигуру (рис. 9.13).

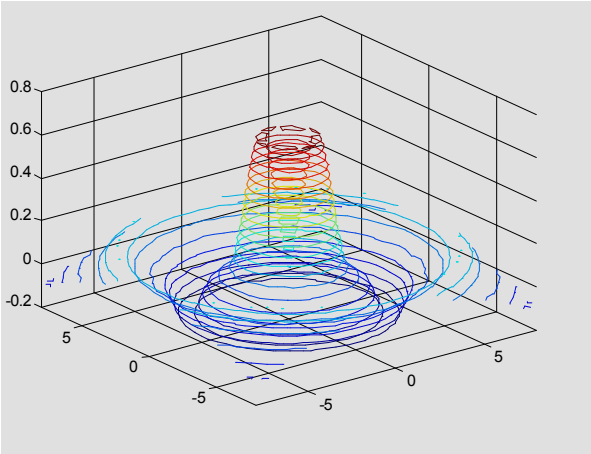


Рис.9.13

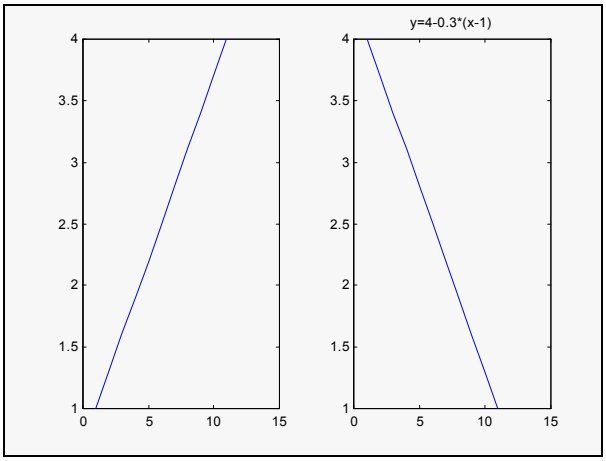


Рис.9.14

9.5. Дополнительные возможности

figure, figure('PropertyName',Propertyvalue,...) −создание нового (очередного) графического окна.

figure(n) − выбор *n*-го из созданных окон в качестве текущего.

hold on/off, hold −включение /выключение режима сохранения текущего графика.

title('текст'), title(<имя функции-строки>'), title(..., 'PropertyName','PropertyValue',...), h=title(...) − вывод заголовков для графиков (в текущем окне):

subplot(m,n,k), subplot(mnk) – вывод графиков в нескольких окнах рисунка: *m* – число окон по горизонтали, *n* – по вертикали, *k* –

номер окна:

» subplot(121) ; plot([1:0.3:4]) ; subplot(122); plot([4:-0.3:1])

» title ('y=4-0.3(x-1)') %Рис.9.14

xlabel(...), ylabel(...), zlabel(...) – вывод текста для обозначения координатной оси ; синтаксис аналогичен title(...).

text(x,y,'текст'), text(x,y,z,'текст'), text(...'PropertyName', 'PropertyValue',...), h=text(...) – вывод текста в указанной позиции графика (x,y,z –координаты начала текста).

gtext('текст'), h= gtext('текст') – вывод текста с возможностью последующего его перемещения мышью.

legend('текст1','текст2',...), legend(M), legend(h,M), legend off, legend(...,pos), h= legend(...) – вывод легенды: здесь M – строковый массив (длина строк одинакова), off удаляет пояснения к графику, pos определяет позицию легенды (-1 – справа от графика, 0 – в одном из 4 углов с минимумом потерь точек графика, 1-4 – в указанном углу, [x y] – в указанном месте); можно перетаскивать легенду мышью.

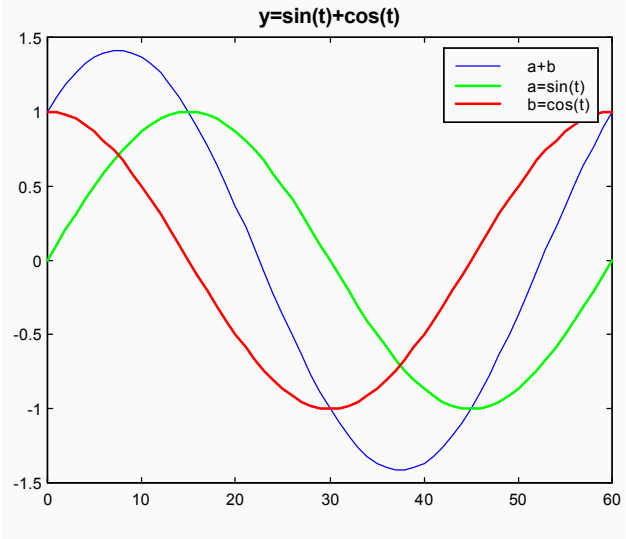


Рис.9.15

» subplot(111)

» t=[0:pi/30:2*pi]; » a=sin(t); b=cos(t); x=0:60;

» plot(x,a+b),hold on

» hp=plot(x,a', 'g', x,b','r'); set(hp,'LineWidth',2)

» legend('a+b','a=sin(t)','b=cos(t)')

» title('y=sin(t)+cos(t)','FontSize',12,'FontWeight','bold')

clabel(C,h), clabel(C,h,v), clabel(C,h,'manual'), clabel(C), clabel(C,v), clabel(C,'manual') – маркировка линий уровня, создаваемых функциями contour, contour3, contourf **;** при наличии **h** маркировка на линиях, при наличии **'manual'** – принудительная маркировка нажатием левой мыши или пробела (правая мышь или Return завершает маркировку).

9.6. Специальная графика

Раздел специальной графики включает команды для построения диаграмм, гистограмм и прочих дискретных графиков.

Столбцовые диаграммы реализуются функциями **bar** и **barh**:

bar(y), bar(x,y) , h=bar(...) – здесь **y** –массив (одно- или двумерный), **x** – одномерный, упорядоченный по возрастанию массив (число смежных по горизонтали столбцов диаграммы равно числу столбцов массива **y**); можно указать параметры относительной ширины столбцов (1 – касание, >1 – перекрытие, <1 – с промежутками), или стиля (‘group’,’stack’) :

```
» x=0:0.1:6;
» y1=sin(x); y2=cos(x); y3=exp(-x./2);
» y=[y1;y2;y3];
» bar(x,y')
```

```
» x=0:0.1:6;
» y1=sin(x); y2=cos(x); y3=exp(-x./2);
» y=[y1;y2;y3];
» bar(x,y','stack')
```

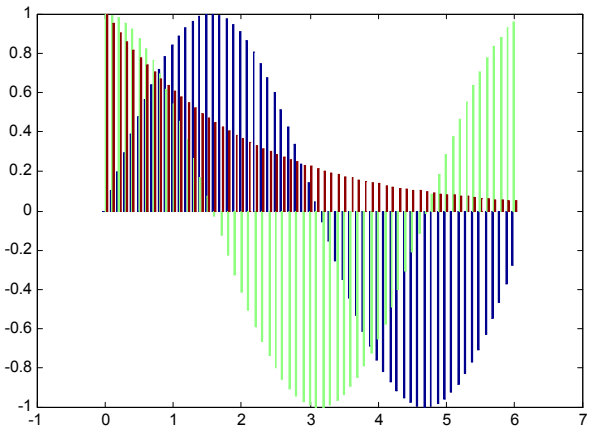


Рис.9.16

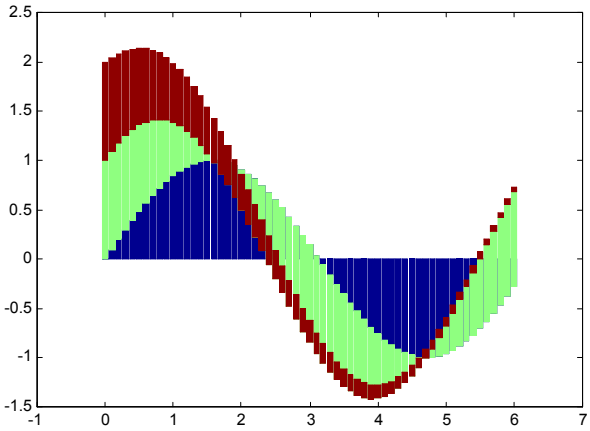


Рис.9.17

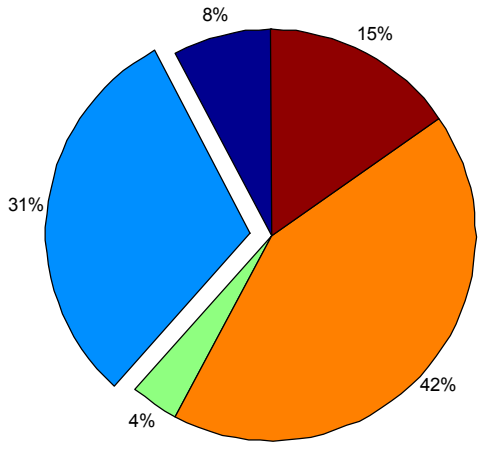


Рис.9.18

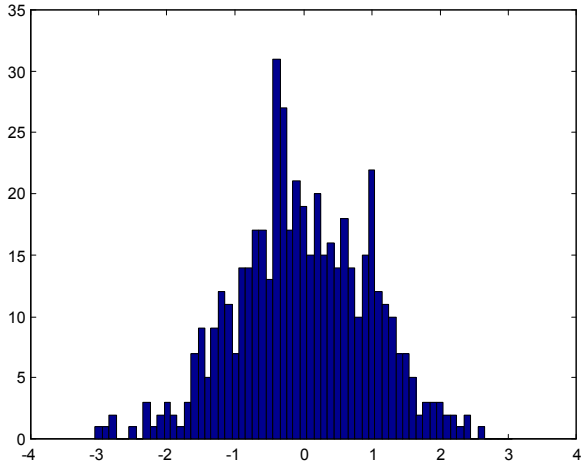


Рис.9.19

barh(...) отличается лишь размещением столбцов не по вертикали, а по горизонтали.

Секторная диаграмма реализуется функцией **pie(x), pi(x,v), h=pie(...) –** здесь v – вектор из 0 и 1 для отделения от диаграммы отдельных секторов:

» x=[1 4 0.5 5.5 2]; pie(x,[0 1 0 0 0]) % Рис.9.18

Построение гистограммы hist(y), hist(y,x), hist(y,n), [p,x]=hist(y,...) реализует подсчет числа элементов по столбцам массива **y** в **n** (по умолчанию 10) интервалах:

» x=-3:0.1:3; t=randn(500,1); hist(t,x) % Рис.9.19

Дискретный график stem(y), stem(x,y), stem(...,'fill'), stem (...,LineSpec), h=stem(...) аналогичен столбцовой диаграмме и выводит значения в виде отрезков с маркером (**'fill'** –закраска маркера):

» x=-3:0.1:3; » f=exp(-x.^2/2); stem(x,f) % Рис.9.20

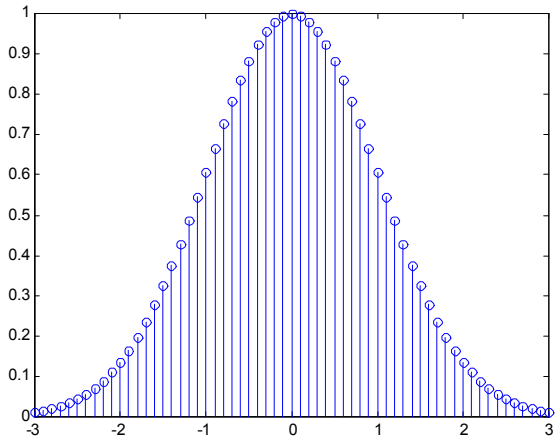


Рис.9.20

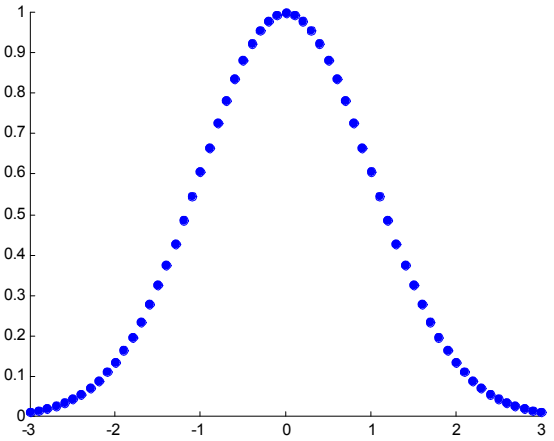


Рис.9.21

Вывод поля точек выполняется функцией **scatter(x,y,...)** с возможностью указывать размер, цвет и заполненность маркера:

» x=-3:0.1:3; f=exp(-x.^2/2); scatter(x,f,'filled') % Рис.9.21

Существенный интерес представляют функции **поворота графического объекта rotate** : например,

» h=surf(...); rotate (h,[1 0 0],90) % поворот по оси **x** на 90°

и функции **поворота графического объекта с помощью мыши rotate3d on|ON|off** (on – режим включен, off-выключен, ON – подавляет информацию о текущих углах).

ЛИТЕРАТУРА

1. *В.Г.Потемкин*. Система инженерных и научных расчетов MATLAB 5.x. В 2-х т. –М.: ДИАЛОГ-МИФИ. 1999. – 670 с.

2. *С.П.Иглин*. Математические расчеты на базе MATLAB. –СПб: БХВ-Петербург. 2005. –640 с.

3. *Д.Г.Метьюз, К.Д.Финк*. Численные методы (Использование MATLAB). –М.-СПб-Киев, Издат.дом «Вильямс», 2001. – 714 с

4. *А.И. Плис, Н.А. Сливина*. MATHCAD 2000. Практикум для экономистов и инженеров. -М.: Финансы и статистика. 2000. – 656 с.

СОДЕРЖАНИЕ

Составитель

Моисей Аронович Тынкевич

Система MATLAB

Справочное пособие по дисциплине
«ЧИСЛЕННЫЕ МЕТОДЫ АНАЛИЗА»
для студентов специальности 080801
«Прикладная информатика в экономике»

Печатается в авторской редакции

Подписано в печать 10.01.2012. Формат 60×84/16.
Бумага офсетная. Отпечатано на ризографе
Уч.-изд. л. 2,5. Тираж 50 экз. Заказ .
КузГТУ. 650000, Кемерово, ул. Весенняя, 28.
Типография КузГТУ. 650000, Кемерово, ул. Д. Бедного, 4 А.