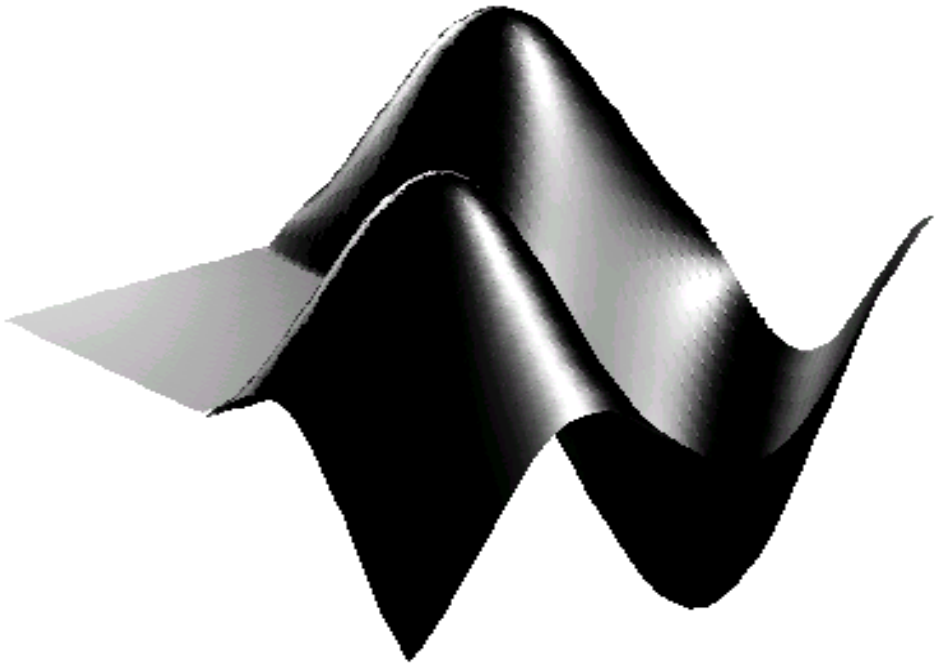
Л.А. Мироновский, К. Ю. Петрова

ВВЕДЕНИЕ В MATLAB

Учебное пособие



ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Л.А. Мироновский, К. Ю. Петрова

ВВЕДЕНИЕ В MATLAB

Учебное пособие

Санкт-Петербург
2005 г.

УДК518
ББК32

Мироновский Л.А., Петрова К.Ю.

Введение в MATLAB. Учебное пособие. СПбГУАП. СПб., 2005

Изложены структура, организация и язык программирования интегрированного математического пакета MATLAB. Описаны основные функции ядра пакета и некоторых тулбоксов (CONTROL, SYMBOLIC, SIGNAL, OPTIMIZATION). Рассмотрена методика решения задач линейной алгебры, оптимизации, линейных и нелинейных дифференциальных уравнений. Описано применение системы SIMULINK для структурного моделирования систем автоматического управления. Изложение сопровождается большим количеством примеров.

Учебное пособие предназначено студентам, обучающимся по направлениям 220100, 552800, а также родственным специальностям и направлениям.

Рецензенты:

Кафедра прикладной информатики Международного банковского института,
Доктор технических наук, профессор Слаев В.А.

Утверждено
редакционно-издательским советом университета
в качестве учебного пособия

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Учебное пособие написано на основе работы [6], вышедшей более 10 лет назад. За это время в структуре и командах пакета MATLAB произошли заметные изменения, а сам он прочно занял лидирующее положение среди математических пакетов, ориентированных на решение научно-технических и инженерных задач. В его составе появилась система визуального моделирования SIMULINK – эффективное и удобное средство для моделирования систем, заданных структурными схемами.

К настоящему времени выпущено много руководств и справочников по MATLAB ([1-6, 10-12, 14]), содержащих подробное описание его обширной системы команд (их общее число давно перевалило за тысячу), расширений и приложений. Однако, большинство из этих руководств неудобны для начинающего пользователя из-за обилия представленного материала и перегруженности деталями.

Настоящее пособие предназначено для первоначального знакомства с MATLAB и SIMULINK и содержит элементарные сведения о пакете. В первую очередь это относится к разделам 1-3, где даются минимальные сведения о MATLAB и SIMULINK, необходимые для выполнения лабораторных работ по курсу «Моделирование». Разделы 4-6 предназначены для более глубокого знакомства с пакетом, в них подробнее описываются его графические и математические возможности, файловая система, типы данных. Из многочисленных тулбоксов MATLAB в пособии затронуты лишь некоторые, в частности, CONTROL, SYMBOLIC и OPTIMIZATION.

1. ЭЛЕМЕНТАРНЫЕ СВЕДЕНИЯ О ПАКЕТЕ

1.1. Структура пакета

Пакет MATLAB широко используется во всем мире при решении задач, связанных с матричными вычислениями. Название пакета образовано путем сокращения от MATrix LABoratory (матричная лаборатория). Операции и команды в MATLAB достаточно естественны и аналогичны математической записи формул на бумаге. MATLAB создавался как пакет программ, реализующих наиболее эффективные вычислительные алгоритмы линейной алгебры. Он организован таким образом, чтобы пользователь имел возможность применять при работе обычный математический язык.

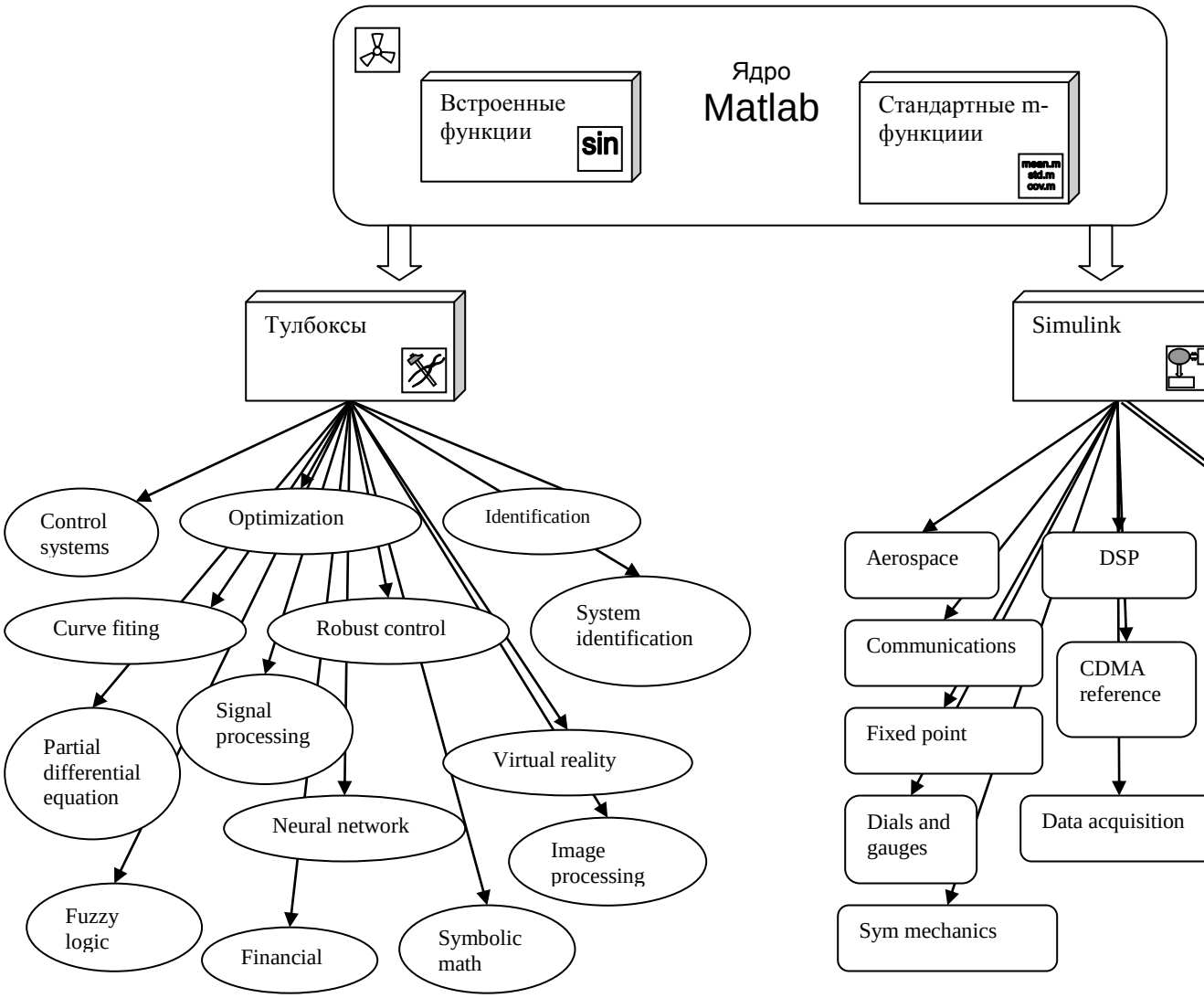
В настоящее время пакет MATLAB представляет собой развитую интегральную программную среду, включающую собственный язык программирования. Он дает пользователю возможность быстро выполнять различные операции над векторами и матрицами, такие как умножение и обращение матриц, вычисление определителей, нахождение собственных чисел и векторов. Кроме того, в MATLAB входят операции вычисления обычных функций (алгебраических, тригонометрических, логических), решения алгебраических и дифференциальных уравнений, операции построения графиков и ряд других.

MATLAB является языком высокого уровня. По отдельным его командам можно выполнять такие сложные операции, как нахождение корней полиномов, решение линейных и нелинейных алгебраических уравнений, моделирование линейных динамических систем. Указанные операции являются элементарными функциями MATLAB.

Помимо ядра, реализующего вычислительные алгоритмы общего назначения, в пакете MATLAB реализовано несколько десятков так называемых тулбоксов (библиотек специализированных подпрограмм), предназначенных для решения разнообразных практических

задач. Например, тулбокс SYMBOLIC предназначен для выполнения символьных вычислений, а тулбокс CONTROL – для расчета и моделирования систем автоматического управления.

Его надо повернуть!



Вместе с пакетом MATLAB, поставляется также среда для визуального моделирования структурных схем SIMULINK, технология работы в которой в значительной степени копирует технику моделирования на аналоговых вычислительных машинах.

Общая структура системы MATLAB поясняется рис. 1.1. Его верхняя часть соответствует ядру MATLAB, содержащему быстро выполняемые встроенные функции (сложение, умножение, тригонометрические и другие базовые функции) и так называемые *m*-функции, алгоритмы выполнения которых написаны на языке MATLAB. Слева внизу показано семейство тулбоксов, каждый из которых содержит несколько десятков *m*-функций, справа – среда SIMULINK и связанные с ней средства (расширения и библиотеки блоков для разных приложений).

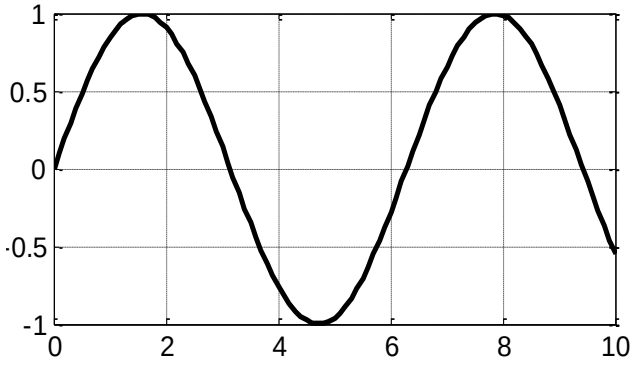
1.2. Начало работы

1.2.1. Запуск и элементарные операции

При запуске MATLAB, как правило, открывается интерфейс, который содержит меню, панель инструментов и два окна – командное окно (*Command Window*) и окно рабочего пространства (*Workspace*). В случае если командное окно или окно рабочего пространства отсутствует, их можно активизировать во вкладке меню "View".

Команды вводятся в диалоговом режиме непосредственно в командное окно. Например, для того чтобы вычислить значение $\sin 30^0$, надо в командном окне набрать текст sin(pi/6) и нажать клавишу Enter. На экране появится ответ ans=0.5.

Если мы хотим построить график функции y=sin*t* на интервале $0 \le t \le 10$, то сначала следует сформировать массив значений аргумента (для этого в командном окне набираем *t*=0:.1:10;), а затем набрать текст y=sin(t); plot(t, y). Разделительный знак «точка с запятой» ставится, чтобы не выводить на экран результаты промежуточных вычислений. После нажатия Enter в графическом окне появится график синусоиды. При желании его можно снабдить надписями и нанести координатную сетку (команда **grid**).



>>t=0:.1:10; y=sin(t); plot(t,y), grid

Рис.1.2

В дальнейшем обе переменные t, y (каждая из них представляет собой массив из 101 числа) сохраняются в рабочем пространстве MATLAB и доступны для использования.

Вместо того чтобы набирать команды в окне MATLAB, их можно записать в текстовый файл с расширением *.m* (он называется *m*-файл или файл-сценарий). Имя файла может быть любым, например, vova.m. Для того чтобы запустить файл сценарий, достаточно набрать его имя (без расширения) в командном окне: >> vova

Чтобы MATLAB «увидел» файл, либо помещайте его в стандартный рабочий каталог, например C:\MATLAB \work\, либо укажите путь к нему.

В разобранном примере использовалась функция **sin.** В состав MATLAB входят команды для вычисления более 50 элементарных функций. Обозначения наиболее употребительных из них приведены в табл. 1.

Таблица 1

X ± Y	**X^2**	**EXP**	**SIN**	**ASIN**	**SINH**
X * Y	**SQRT**	**LOG**	**COS**	**ACOS**	**COSH**
X / Y	**ABC**	**LOG10**	**TAN**	**ATAN**	**ASINH**

В первом столбце перечислены команды для выполнения четырех арифметических действий. Второй столбец содержит операции возведения в квадрат, извлечения квадратного корня и взятие абсолютной величины (для комплексных чисел – модуля). Для вычисления каждой из этих функций нужно в круглых скобках указать значение аргумента (например, sqrt(4) даст ответ 2).

В третьем столбце перечислены команды для получения экспоненты и логарифмов (натурального и десятичного), например exp(1) даст число e=2,71828… Четвертый и пятый столбцы содержат прямые и обратные тригонометрические функции, а в последнем, шестом столбце находятся команды для вычисления гиперболических функций, которые в русской литературе обозначаются символами sh, ch, arcsh. Заметим, что аргумент функций **sin**, **cos** должен указываться в радианах, если задавать его в градусах (*degree*), то надо использовать команды **sind**, **cosd** (впервые они появились в MATLAB 7). Например, sin(pi/6) и sind(30) дадут один и тот же результат ans=0.5000. Аргументом каждой из функций может быть число или вектор (набор чисел, массив). Например, набрав sind([0, 30, 90]) получим ans=0 0.5000 1.0000.

Для оперативного получения справок об этих и других командах используется команда **help**. Например, набрав в MATLAB 7

```
>> help sind,
```

получим справку:

SIND Sine of argument in degrees. SIND(X) is the sine of the elements of X, expressed in degrees.
For integers n, sind(n*180) is exactly zero, whereas sin(n*pi) reflects the accuracy of the floating point value of pi.
See also **asind, sin**.

1.2.2. Ввод числовых данных

Перечислим несколько простых команд для ввода числовых данных в виде векторов и матриц. Самый простой способ формирования векторов и матриц в MATLAB заключается в непосредственном вводе их элементов с клавиатуры. Например, набирая на клавиатуре данные X = [1 -2 3 8 5 6], получаем одномерный массив (вектор-строку) X из шести элементов.

Формирование вектора-строки из равноотстоящих значений аргумента выполняется с помощью команды x = x0:h:xn. По умолчанию шаг h принимается равным 1. Например, команда x=0:10 дает целые числа от 0 до 10, а x=0:0.1:10 задает набор значений аргумента от нуля до 10 с шагом 0.1.

Двумерные массивы задаются в виде матриц, при этом строки разделяются символом «точка с запятой». Элементы одной и той же строки могут разделяться как пробелами, так и запятыми:

```
>> a=[1 2 3; 4 5 6; 7 8 9
a =
   1   2   3
   4   5   6
   7   8   9
```

```
>> a=[1 ,2, 3; 4, 5, 6; 7, 8, 9]
a =
   1   2   3
   4   5   6
   7   8   9
```

Для доступа к элементам массива используются круглые скобки:

```
>> a(1,1)
ans =   1
```

```
>> a(3,3)
ans =   9
```

```
>> b=[1 2 3 4 5]; b(4)
ans =  4
```

Для получения строки или столбца матрицы используется символ «двоеточие»:

```
>> a(:,1)
ans =   1
        4
        7
```

```
>> a(2,:)
ans =
    4   5   6
```

```
>> b(1:3)
ans =
    1   2   3
```

```
>> b(3:end)
ans =
    3   4   5
```

В MATLAB имеется ряд команд, облегчающих формирование векторов и матриц (табл.2).

Таблица 2

linspace	**zeros**	**eye**
logspace	**ones**	**diag**

Команда **linspace** используется для получения набора равноотстоящих значений аргумента. Например, x=linspace(xmin,xmax) создаст массив из 100 значений аргумента между точками xmin и xmax. Если требуется иметь другое число точек, например, N точек на интервале от 0 до 10, следует записать x = linspace(0,10,N). Аналогичные модификации имеет команда **logspace**, обеспечивающая логарифмическое расположение точек массива.

Для создания некоторых распространенных матриц имеются команды **zeros, ones, eye** и **diag**. Команды **zeros** и **ones**, создают матрицы, заполненные нулями и единицами. Так, по командам A=zeros (3) и B=ones (3) будут сформированы матрицы

```
A=  0 0 0          B = 1 1 1
    0 0 0              1 1 1
    0 0 0              1 1 1 ,
```

а по командам C = zeros(1,4) и D = ones(1,4) – векторы-строки C = 0 0 0 0 и D = 1 1 1 1.

Команда **eye** предназначена для создания (она часто обозначается буквой I, название которой произносится как слово «*eye*») или ее фрагмента.

```
>> E=eye(3)                >> G=eye(2,7)
E =  1   0   0             G =
     0   1   0                1   0   0   0   0   0   0
     0   0   1                0   1   0   0   0   0   0
```

Команда **eye**(size(A)) даст единичную матрицу того же размера, что и матрица A.

Команда **diag** имеет два различных назначения. Если ее аргументом является матрица, то результатом будет вектор, содержащий элементы, стоящие на диагонали. Если же аргументом является вектор, то команда **diag** создает матрицу с заданной диагональю и остальными нулевыми элементами.

Например, применяя команду **diag** к приведенной выше матрице B, получим строку [1 1 1], а повторное применение команды **diag** даст единичную матрицу E.

1.3. Построение графиков

Основное средство для построения графиков в MATLAB – это команда **plot** и различные ее модификации. Она может вызываться с одним или несколькими входными аргументами.

Стандартный вариант ее вызова – это plot(x,y), где x и y – два массива чисел, содержащие абсциссы и ординаты точек графика функции $y = f(x)$. Выше был приведен пример построения графика синусоиды, аналогично строятся графики любых других функций. При этом вычерчивание осей и выбор масштабов по ним производится автоматически. В случае, если вызов команды **plot** производится с одним аргументом в формате plot(y), координатами x служат индексы массива y.

Для того чтобы снабдить рисунок координатной сеткой, используется команда **grid**. Вызов ее без параметров осуществляет переключение режимов «с сеткой»/«без сетки», а задание **grid on** и **grid off** явно указывает, следует включить сетку или отключить.

Иногда на одном графике требуется нарисовать несколько кривых. В этом случае в команде **plot** указывают несколько пар аргументов (по числу функций) plot(x1, y1, x2, y2, ..., xn, yn), где x1, y1;

x2, y2 и т.д. – пары векторов. Каждой паре x, y будет соответствовать свой график, при этом они могут быть заданы векторами разной длины.

Пример. Пусть требуется построить графики затухающих колебаний $x(t) = e^{-0,2t} \sin t$, $y(t) = e^{-0,2t} \cos t$, причем аргумент t изменяется от 0 до 10 с шагом 0,1. Это делается с помощью следующей группы команд:

```
>>t=0:.1:10; x=exp(–.2*t).*sin(t); y=exp(–.2*t).*cos(t); plot(t, x, t, y), grid.
```

Результат показан на рис. 1.3. Использование точки перед знаком * (умножение) при вычислении переменных x, y, указывает на поэлементное перемножение массивов чисел (каждая из функций sin t, cos t, $e^{-0,2t}$, представлена вектором из 101 точек).

Добавляя команду plot(x,y), grid, получим график логарифмической спирали, показанный на рис. 1.4.

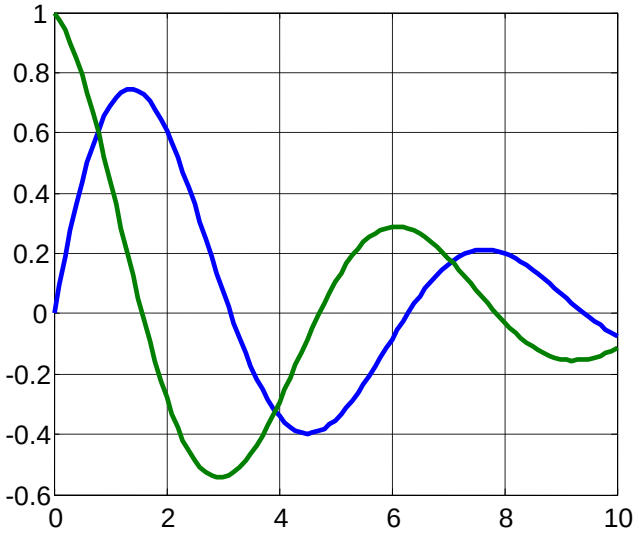


Рис. 1.3

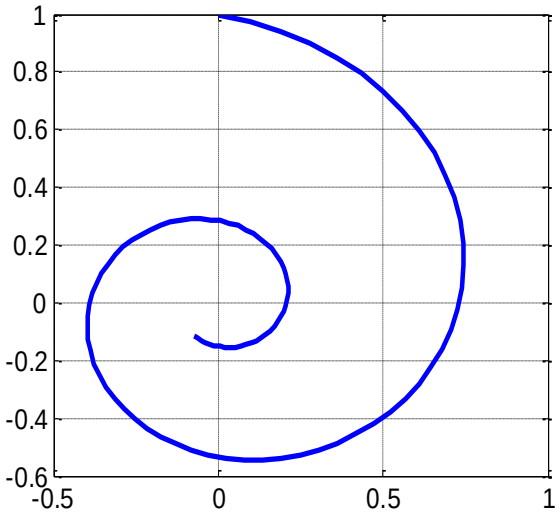


Рис.1.4

После того как график выведен на экран, его можно озаглавить, обозначить оси, сделать текстовую разметку, для чего используются команды **title, xlabel, ylabel, text**. Например, чтобы нанести обозначения осей на последний график, надо набрать xlabel('x'), ylabel('y').

В команде plot в одиночных кавычках можно использовать дополнительный аргумент, указывающий тип символов, используемых для построения графика. Так, plot(X,Y,'x') вычерчивает точечный график, используя символы x (крестики), тогда как plot(X1,Y1,':',X2,Y2,'+') использует символ двоеточия для первой кривой и символ + для второй. Цвет линий также может задаваться пользователем. Например, команды plot (X,Y,'r') и plot (X,Y,'+g') используют красную линию для получения первого графика и зеленые + метки для второго. Справку о возможных вариантах типов линий, точек и цветов можно получить, набрав help plot.

Команда **plot** строит графики на плоскости. MATLAB позволяет также наглядно изображать линии и поверхности в трехмерном пространстве. Для изображения линий в пространстве используется команда **plot3**. Получим, например, график винтовой линии, которая задается уравнениями x=sin$t$; y=cos$t$; z=t. Возьмем диапазон изменения параметра $0 \le t \le 10\pi$ с шагом $0,02\pi$:

```
>>t = 0:pi/50:10*pi;    plot3(sin(t),cos(t),t);
```

Результат показан на рис. 1.5.

Три стандартные поверхности – сфера, эллипсоид и цилиндр – строятся с помощью команд **sphere, ellipsoid, cylinder** соответственно. Результат выполнения первой из них показан на рис. 1.6.

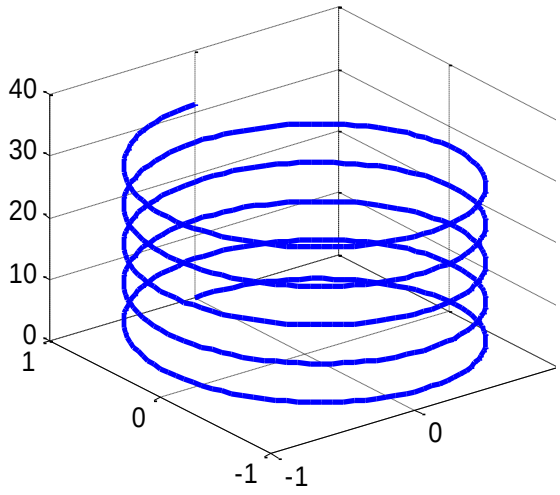


Рис. 1.5

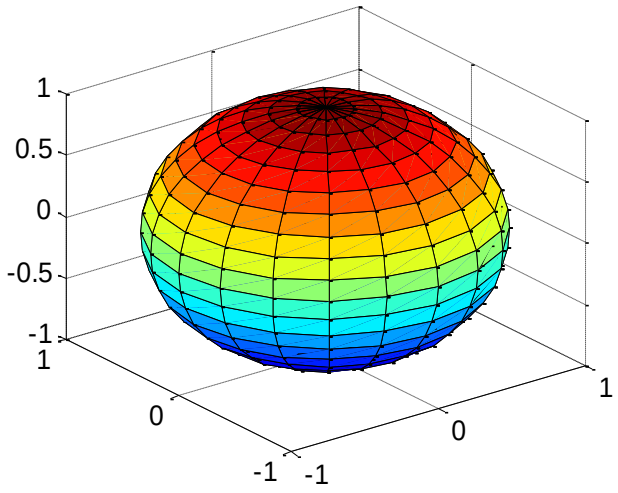


Рис. 1.6

Дополнительные сведения о графических возможностях MATLAB можно найти в разд. 1.7 и 4.1.

1.4. Матричные операции

MATLAB имеет обширный арсенал матричных операций. К простейшим из них относятся сложение и умножение, вычисление ранга и определителя, а также обращение матрицы.

Элементарные операции над матрицами перечислены в табл. 3

Таблица 3

A±B	**A'**	**trace**	**inv**
A∗B	**det**	**rank**	**pinv**

Для сложения и вычитания матриц одинакового размера используются знаки + и –, например, C = A + B. Умножение матриц обозначается звездочкой: C=A∗B. Оно допускается, если число строк матрицы A равно числу столбцов матрицы B. При этом в общем случае $A*B \neq B*A$.

Напомним, что матричное умножение вычисляется по известному правилу «строка на столбец». В частности, произведение матриц второго порядка находится по формуле

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{22}b_{12}+a_{22}b_{22} \end{bmatrix}.$$

Для MATLAB такое умножение – элементарная операция. Приведем простой пример:

```
>>A=[1 2; 3 4]      >>B=[5 6; 7 8]      >>C=A*B      >>CT=B'*A'
     1  2               5  6              19   22          19   43
     3  4               7  8              43   50          22   50
```

Простейшей матричной операцией является транспонирование (строки заменяются столбцами). Например, результатом транспонирования вектора-строки будет вектор-столбец. В MATLAB транспонирование обозначается штрихом. В последней колонке приведена транспонированная матрица C, она равна произведению транспонированных матриц A и B, взятых в обратном порядке.

По команде **trace(A)** вычисляется след матрицы A, т.е. сумма ее диагональных элементов. Полезно помнить, что он равняется также сумме ее собственных чисел. По команде **rank** находится ранг матрицы. Он определяется максимальным размером ее минора с ненулевым определителем и одновременно указывает на число линейно независимых строк (столбцов) матрицы.

Следующая матричная операция – вычисление определителя, осуществляется командой **det.** Как известно, определитель матрицы $A=\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ размера 2x2 равен $\det A = ad - bc$. Формула для определителя третьего порядка имеет вид:

$$\det\begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2 - a_3b_2c_1 - a_1b_3c_2 - a_2b_1c_3.$$

Обращение квадратной матрицы A производится по команде **inv(A).** Она является основной при решении системы линейных алгебраических уравнений.

Напомним формулу для вычисления обратной матрицы

$$A^{-1} = \frac{1}{\det A} A^*,$$

где A^* – присоединенная матрица, составленная из алгебраических дополнений A_{ij} матрицы A. В соответствии с этой формулой процедура вычисления обратной матрицы содержит 2 шага.

Шаг 1. Каждый элемент a_{ij} матрицы A заменяется его алгебраическим дополнением A_{ij}, т.е. определителем матрицы, получаемой вычеркиванием соответствующей строки и столбца. Если сумма индексов $i+j$ нечетна, определитель берется с минусом.

Шаг 2. Полученная матрица транспонируется и делится на определитель исходной матрицы.

В частности, для матрицы второго порядка получаем

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

Пример. Табл. 4 содержит примеры выполнения упомянутых операций.

Таблица 4

>> A=[1 2 3; 0 1 4; 0 0 1]	>> A'	>> det(A)	>> inv(A)	>> inv(A')
1 2 3	1 0 0	1	1 -2 5	1 0 0
0 1 4	2 1 0		0 1 -4	-2 1 0
0 0 1	3 4 1		0 0 1	5 -4 1

В первой строке приведены команды, набираемые в рабочем окне, во второй – ответы, которые даст MATLAB. В частности видим, что при транспонировании матрицы обратная матрица тоже транспонируется.

Типичная задача линейной алгебры – решение системы линейных уравнений. На матричном языке она сводится к тому, чтобы найти вектор X, удовлетворяющий равенству $AX = B$, где матрица A и вектор-столбец B заданы. Решение этого уравнения имеет вид $X = A^{-1}B$.

Для получения такого решения в пакете MATLAB достаточно набрать
>>X=inv(A)*B либо >>X=(A^-1)*B.

Разумеется, для выполнения этих операций матрица A должна быть невырожденной. В случае если система уравнений $AX = B$ недоопределена или переопределена, решение получают с помощью команды **pinv**, выполняющей псевдообращение матрицы A. Можно также использовать знак деления – косую черту (*slash* и *backslash*).

Пример. Требуется решить систему трех уравнений с тремя неизвестными

$$x + y + z = 2$$
$$2x + 2y + z = 2$$
$$x + 2y + 2z = 5.$$

В данном случае матрицы A и B имеют вид $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 1 & 2 & 2 \end{bmatrix}$, $B = \begin{bmatrix} 2 \\ 2 \\ 5 \end{bmatrix}$.

Вводя их в MATLAB и набирая >>X=inv(A)*B, получим:

```
>>A=[1 1 1; 2 2 1; 1 2 2]      >>B=[2; 2; 5]     >>X=inv(A)*B
        A=1  1  1                   B=2               X=-1
          2  2  1                     2                 1
          1  2  2                     5                 2
```

Следовательно, решение системы имеет вид x=-1, y=1, z=2.

1.5. Работа с полиномами

В состав MATLAB входит ряд команд, позволяющих выполнять различные операции с полиномами от одной переменной, включая поиск корней, умножение и деление полиномов, построение полинома, проходящего через заданные точки и др. Полином описывается строкой своих коэффициентов в порядке от старшего к младшему. Так, полином $x^3 - 2x + 5$ будет представлен вектором $[1 \quad 0 \quad -2 \quad 5]$.

Перечень основных команд MATLAB для работы с полиномами приведен в табл. 5.

Таблица 5

roots	**conv**	**polyval**
poly	**deconv**	**residue**

Дадим краткие пояснения к ней.

Команда **roots** предназначена для отыскания корней полинома. Например, чтобы решить квадратное уравнение $x^2 + 5x + 6 = 0$ следует набрать r=roots([1 5 6]), результатом будут значения корней r=[-2; -3].

Функция **poly** выполняет обратную операцию – строит полином по заданным корням. Так p=poly([-2 -3]) даст p=[1 5 6]. Если в качестве входного аргумента функции **poly** фигурирует квадратная матрица, то результатом будет ее характеристический полином.

Следует иметь в виду, что вычисление корней полиномов высокой степени с помощью команды **roots** может сопровождаться заметными погрешностями, поэтому пользоваться ей следует с осторожностью.

Пример (демонстрация ошибок при вычислении корней). Все корни полинома $(x+1)^8$ вещественны и равны -1. Сформируем этот полином командой **poly** и найдем его корни командой **roots.**

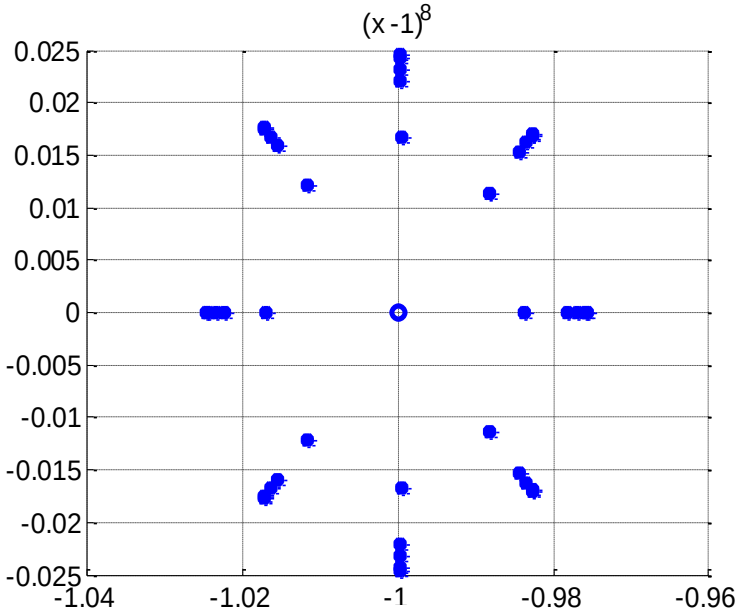


Рис. 1.7

```
>>p=poly(-ones(1, 8)), r=roots(p),
p =  1  8  28  56  70  56  28  8  1
r = -1.0203; -1.0142±0.0144i; -0.9998±0.0201i; -0.9858±0.0140i; -0.9801
```

В результате получили комплексные числа, причем их модуль отличается от единицы на 2%. Если учесть, что MATLAB работает с 32-разрядной сеткой, это очень большая погрешность.

В наглядной форме ошибки вычисления корней показаны на рис. 1.7. Он получен путем пятикратного применения пары команд **roots**/**poly,** когда по корням восстанавливался полином, снова искались его корни и т.д. Кружок в центре рис. 1.7 соответствует истинным корням полинома $(x+1)^8$, а звездочками помечены значения корней, вычисленные в процессе итераций.

```
% Корневой тест
n=8; r=-ones(1,n); clg, hold on
   for i=1:5,
      p=poly(r),        r=roots(p),
      plot(r,'*','LineWidth',2)
   end
plot(-1,0,'o','LineWidth',2),grid on;  hold off
```

Избежать подобных ошибок можно, переходя к символьным вычислениям (для этого необходимо наличие тулбокса SYMBOLIC).

```
>> r=-ones(1,8); p=poly(r)                    % численное представление полинома
p = 1   8   28  56   70  56   28   8   1
>> P=poly2sym(p),                             % символьное представление полинома
P =x^8+8*x^7+28*x^6+56*x^5+70*x^4+56*x^3+28*x^2+8*x+1
>>R=solve(P);                                 % решение уравнения  P(x) = 0.
R'=[ -1, -1, -1, -1, -1, -1, -1, -1]         % найденные корни
```

Как видим, символьный решатель уравнений **solve** дал точный ответ. К сожалению, его возможности ограничены только уравнениями, допускающими аналитическое решение.

Для того чтобы построить график полинома, надо предварительно вычислить его значения в точках заданного интервала. Для этой цели служит функция **polyval** (сокращение от *polynomial value*). Например, чтобы построить график полинома $y = x^2 + 5x + 6$ на интервале $-5 \le x \le 5$ следует набрать:

```
>> x= -5:0.1:5;   p=[1  5  6];  y=polyval(p, x);  plot(x, y), grid.
```

В результате будет получен график, который пересекает ось абсцисс в точках x_1= -3, x_2= -2 (это найденные выше корни полинома).

Для умножения полиномов предназначена функция **conv** (сокращение от *convolution*. Рассмотрим произведение двух квадратных трехчленов:

$$P1 = x^2 + 3x + 2; \quad P2 = 4x^2 + 5x + 1; \quad P = P1 \cdot P2 = 4x^4 + 17x^3 + 24x^2 + 13x + 2.$$

Вычислим коэффициенты результирующего полинома четвертой степени в MATLAB:

```
>> P1=[1 3 2];  P2=[4 5 1];  P= conv(P1,P2).
```

Получим ответ: `P=  4  17  24  13  2.`

Обратная операция – деление полиномов – выполняется по команде **deconv.** Результат операции деления полиномов представляет собой частное и остаток.

Пример. Найдем целую часть и остаток неправильной рациональной дроби

$$\frac{4x^4 + 17x^3 + 24x^2 + 14x + 4}{x^2 + 3x + 2} = 4x^2 + 5x + 1 + \frac{x + 2}{x^2 + 3x + 2}.$$

Выполним это деление в MATLAB:

```
>> num=[4 17 24 14 4]; den=[1 3 2];  [q,r]=deconv(num,den),
```

Результат будет иметь вид

```
q =  4  5  1,  r =  0  0  0  1  2.
```

Здесь вектор q характеризует целую часть деления, а вектор r – остаток.

Отношение двух полиномов относится к классу так называемых дробно-рациональных функций. Именно такой вид имеют, например, передаточные функции систем автоматического управления. С ними часто приходится делать две типовые операции – разложение передаточной функции в сумму элементарных дробей и обратное действие – сложение элементарных дробей, т.е. приведение их к общему знаменателю.

В MATLAB обе эти операции могут быть выполнены с помощью команды **residue.**

Пример. Пусть требуется разложить на элементарные дроби рациональную функцию

$$\frac{x + 5}{x^2 + 3x + 2} = \frac{k_1}{x - r_1} + \frac{k_2}{x - r_2}.$$

При «ручном» счете сначала находят числа r_1, r_2 – это корни знаменателя: r_1= -2, r_2= -1. Затем для определения неизвестных коэффициентов k_1, k_2 приравнивают числители правой и левой частей:

$$x + 5 = k_1(x + 1) + k_2(x + 2).$$

Отсюда $k_1 + k_2 = 1$, $k_1 + 2k_2 = 5$. Решая эту систему, находим $k_1 = -3$, $k_2 = 4$.

Выполним указанное разложение с помощью команды **residue.** Ее входными аргументами являются числитель num и знаменатель den исходной дроби, выходными аргументами – векторы R, K, содержащие коэффициенты знаменателей и числителей элементарных дробей (полюсы и вычеты исходной функции). Набрав

```
>>num=[1  5];  den=[1  3  2];  [K,  R]=residue(num,  den),
```

получим результат `K=[-3  4], R=[-2  -1],` . Следовательно, искомое разложение имеет вид

$$\frac{x + 5}{x^2 + 3x + 2} = \frac{4}{x + 1} - \frac{3}{x + 2}.$$

В общем случае корни знаменателя могут оказаться комплексными и кратными, кроме того, исходная дробь может быть неправильной (порядок числителя больше или равен порядку знаменателя). Об особенностях применения команды в этих случаях можно узнать с помощью справки **help residue.**

При вызове с синтаксисом [num, den]=residue(R, K, P) команда выполняет обратное действие – находит сумму элементарных дробей, характеризуемых параметрами R, K и полиномом P, заданным вектором своих коэффициентов. Одно из применений этого варианта команды – сложение комплексно-сопряженных пар дробей для получения вещественных элементарных дробей второго порядка.

1.6. Собственные числа и векторы

С необходимостью вычисления собственных чисел и векторов приходится сталкиваться при решении многих физических и технических задач, таких как определение осей эллипсоида инерции тяжелого тела, определение собственных частот колебаний электрических и механических систем, решение систем дифференциальных уравнений, приведение линейных систем к каноническому виду.

Напомним, что собственными числами или собственными значениями квадратной матрицы A называются корни ее характеристического полинома. Характеристический полином находим, раскрывая определитель

$$\det(A-\lambda E)=\lambda^n+a_{n-1}\lambda^{n-1}+\ldots+a_1\lambda+a_0,$$

где E – единичная матрица, n – размерность матрицы A.
Приравнивая его нулю, получим характеристическое уравнение матрицы A.

Найдем характеристический полином при n=2:

$$\det\begin{bmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{bmatrix}=(a_{11}-\lambda)(a_{22}-\lambda)-a_{12}a_{21}=\lambda^2-(a_{11}+a_{22})\lambda+(a_{11}a_{22}-a_{12}a_{21}).$$

Заметим, что коэффициент при λ равен следу матрицы A, взятому с минусом, а свободный член равен ее определителю. Поэтому характеристическое уравнение можно записать в виде

$$\lambda^2-\lambda\,\mathrm{tr}\,A+\det A=0.$$

Корни λ_1, λ_2 этого уравнения и будут собственными числами матрицы A.

В MATLAB для получения характеристического полинома матрицы A можно воспользоваться командой poly(A). Ее результатом будет вектор коэффициентов характеристического полинома. Корни этого полинома находим командой **roots.** Найдем, например, собственные числа матрицы $\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$:

```
>>A=[1 2; 0 3];  p=poly(A);  L=roots(p).
```

В результате выполнения этих команд на экран будут выведены собственные числа 1 и 3.

Более короткий путь получения собственных чисел состоит в применении команды **eig** (от немецкого «eigen» – собственный). Для нашего примера, вводя код L=eig(A), получаем вектор-столбец собственных чисел с элементами 1; 3. Отметим, что матрица A в примере была треугольной, поэтому собственные числа равны ее диагональным элементам.

Перейдем к определению собственных векторов квадратной матрицы. Вектор H называется собственным вектором матрицы A, если в результате его умножения на матрицу он не изменяет своего направления, а лишь удлиняется или укорачивается.

Алгебраическая запись этого условия имеет вид

$$AH = \lambda H \quad \text{или} \quad (A-\lambda E)H = 0, \qquad (*)$$

где коэффициент λ показывает, во сколько раз изменяется длина вектора. Для того чтобы однородная система (*) имела ненулевое решение H, необходимо, чтобы определитель системы равнялся нулю: $\det(A-\lambda E)=0.$ Последнее равенство представляет собой характеристическое уравнение матрицы A. Следовательно его корни $\lambda_1, \ldots, \lambda_n$, т.е. собственные числа, надо поочередно подставлять в уравнение (*), чтобы найти собственные векторы, причем каждому собственному числу λ_i будет отвечать свой собственный вектор H_i.

Замечание 1. Если все собственные числа $\lambda_1, \ldots, \lambda_n$ различны, то у матрицы A будет n линейно независимых собственных векторов $H_1, \ldots, H_n$.

Замечание 2. Поскольку определитель системы $(A - \lambda_i E)H_i = 0$ равен нулю, то одно из уравнений этой системы будет линейной комбинацией других, т.е. «лишним» и его следует отбросить. Решение оставшейся системы будет определено с точностью до произвольной константы. Геометрически это означает, что если H_1 – собственный вектор матрицы A, то и $k\,H_1$, где k – любое число, также собственный вектор. В пакете MATLAB при вычислении собственных векторов константа k обычно выбирается так, чтобы собственные векторы имели единичную длину (чтобы сумма квадратов их компонент равнялась единице).

Замечание 3. Если матрица A – симметрична, то ее собственные числа вещественны, а собственные векторы – ортогональны. У несимметричных матриц все или часть собственных чисел и векторов могут оказаться комплексными.

Чтобы найти собственные векторы матрицы в пакете MATLAB, надо использовать команду **eig** с двумя выходными параметрами [H,L]=eig(A). При этом столбцами матрицы H будут служить собственные векторы матрицы A, а диагональными элементами матрицы L – соответствующие им собственные числа.

Пример. Дана матрица второго порядка $A=\begin{bmatrix}1 & 2\\ 3 & 2\end{bmatrix}.$ Чтобы найти ее собственные числа, выписываем характеристический полином

$$\det(A-\lambda E)=\begin{bmatrix}1-\lambda & 2\\ 3 & 2-\lambda\end{bmatrix}=(1-\lambda)(2-\lambda)-6=\lambda^2-3\lambda-4.$$

Его корни вещественны $\lambda_1=-1, \quad \lambda_2=4.$

Матричное уравнение для определения первого собственного вектора имеет вид

$$AH_1=\lambda_1 H_1, \qquad \begin{bmatrix}1 & 2\\ 3 & 2\end{bmatrix}\begin{bmatrix}h_1\\ h_2\end{bmatrix}=-\begin{bmatrix}h_1\\ h_2\end{bmatrix}.$$

Ему соответствует система двух скалярных уравнений

$$\begin{cases}h_1+2h_2=-h_1\\ 3h_1+2h_2=-h_2\end{cases}\Rightarrow\begin{cases}2h_1+2h_2=0,\\ 3h_1+3h_2=0.\end{cases}$$

Они отличаются только постоянным множителем и эквивалентны уравнению $h_1 + h_2 = 0.$ Принимая, например, $h_1 = 1,$ получаем $h_2 = -1,$ т.е. первый собственный вектор равен $H_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$

Аналогичным образом получаем систему уравнений для определения второго собственного вектора:

$$\begin{cases} h_1 + 2h_2 = 4h_1 \\ 3h_1 + 2h_2 = 4h_2 \end{cases} \Rightarrow \begin{cases} -3h_1 + 2h_2 = 0, \\ 3h_1 - 2h_2 = 0. \end{cases}$$

Полагая $h_1 = 2,$ получаем $h_2 = 3,$ т.е. второй собственный вектор равен $H_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$ Заметим, что его можно записать также в виде $H_2 = \begin{bmatrix} 1 \\ 3/2 \end{bmatrix}$ или $H_2 = \begin{bmatrix} 2/3 \\ 1 \end{bmatrix}.$

Решим эту задачу в MATLAB с помощью команд:

```
>> A=[1 2;3 2]; [H,D]=eig(A)
```

В результате получаем матрицы:

```
   A              H                 D
1    2      -0.7071   -0.5547      -1    0
   3    2    0.7071   -0.8321       0    4
```

Заметим, что MATLAB выдал нормированные собственные векторы.

Пример. Найдем характеристический полином и собственные числа матрицы третьего порядка

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

Используя команды **poly** и **eig**, получаем:

```
>>B=[3 2 1; 2 1 1; 1 1 1]      >>v=poly(B)                  >>eig(B)
      B=3    2    1
        2    1    1          v=1   -5    1    1      ans= -0.3489    0.6041    4.7448
        1    1    1
```

Второй коэффициент характеристического полинома, взятый с обратным знаком, равен сумме собственных чисел и следу матрицы B.

1.7. Символьные вычисления в MATLAB

Для проведения символьных вычислений необходим тулбокс SYMBOLIC пакета MATLAB. Если он установлен, то имеется возможность выполнять аналитические преобразования формул с буквенными коэффициентами, производить численные расчеты без округлений, строить графики функций, заданных в неявной форме.

Формирование символьных переменных производится командами **sym** и **syms**, например, **sym(2)** или **syms x y z.** После этого можно вводить математические выражения, содержащие эти числа или переменные.

Перечень основных символьных алгебраических операций приведен в табл. 6.

Таблица 6

det	**rank**	**inv**	**expand**	**simple**	**solve**
diag	**poly**	**eig**	**factor**	**collect**	**numden**

Первые три столбца в этой таблице содержат перечень операций над матрицами, которые могут выполняться как в числовой, так и в символьной форме. Например, набрав в командном окне текст

```
>>syms a b c d;  A=[a b;c d],   D=det(A)
```

и нажав Enter, получим ответ

$$A=\begin{bmatrix} a, & b \\ c, & d \end{bmatrix}, \quad D = a*d - b*c.$$

По команде `P=poly(A)` получим символьную запись характеристического полинома матрицы:

$$P = x^2 - (a+d)x + ad - bc,$$

а команды **inv** и **eig** дают возможность найти в символьном виде обратную матрицу, собственные числа и собственные векторы.

Пример. Найдем символьные выражения собственных чисел, собственных векторов и характеристического полинома для матриц A, B из двух предыдущих примеров. Преобразуем эти матрицы к символьному виду:

```
>> A=sym(A);  [H,D]= eig(A),  B=sym(B), V=poly(B)
```

A=sym(A)	H	D	B=sym(B)	V=poly(B)
[1 2]	[-1, 1]	[-1, 0]	B= [3, 2, 1]	
[3 2]	[1, 3/2]	[0, 4]	[2, 1, 1]	V =x^3-5*x^2+x+1
			[1, 1, 1]	

Когда используются символьные вычисления собственных векторов, MATLAB берет одну из компонент собственного вектора равной единице. Заметим, что хотя собственные числа матрицы B вещественны, команда `eig(sym(B))` дает для них комплексные выражения, которые не удается упростить средствами MATLAB.

Три последние столбца табл. 6 содержат команды, применяемые для преобразования и упрощения символьных формул. Так, команда **expand** раскрывает скобки, команда **factor,** наоборот, пытается факторизовать выражение (разложить его на множители), команда **collect** приводит подобные члены. Например, команда `expand (a+b)^2` даст ответ `a^2+2*a*b+b^2`, а команда `factor(a^2-b^2)` даст ответ `(a+b)*(a-b)`, Естественно, надо предварительно объявить символьные переменные командой: `sym a b`.

Команды **simple** и **simplify** используются для упрощения формул, например, набрав

```
>>sym(x), y=simple(sin(x)^ 2+cos(x)^ 2)
```

получим ответ y=1.

Особо следует выделить команду **solve,** которая позволяет решать алгебраические уравнения, включая нахождение корней полиномов, решение линейных и нелинейных систем уравнений. Найдем, например, в символьном виде корни квадратного уравнения $x^2 + 2bx + c = 0$. Набрав

```
>>syms x b c;  solve('x^2+2*b*x+c=0', x)
```

получаем ответ: `ans= -b+(b^2-c) ^(1/2),  -b-(^2-c) ^(1/2),`

что соответствует школьной формуле $x_{1,2} = -b \pm \sqrt{b^2 - ac}$.

Команда **numden** служит для выделения числителя и знаменателя дробно-рациональных выражений. Найдем с ее помощью сумму элементарных дробей

$$\frac{1}{p+1}+\frac{2}{p+3}=\frac{3p+5}{(p+1)(p+3)}=\frac{3p+5}{p^2+4p+3}.$$

Вводя текст
>>syms p; [num,den]=numden(1/(p+1)+2/(p+3)),
получаем ответ: num=3*p+5, den=(p+1)*(p+3).

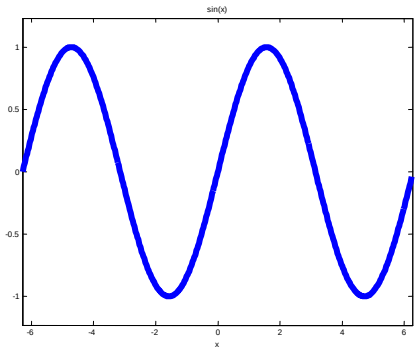
Раскрывая последнее выражение с помощью команды **expand**:
>>den=expand(den), получаем den=p^2+4*p+3, что совпадает со знаменателем суммы дробей.

Отметим еще команды преобразования полиномов из символьного вида в числовой и обратно **sym2poly** и **poly2sym,** а также команды **double** и **vpa,** служащие для перевода символьных данных в числовые. Например, P=sym2poly(den) даст P=[1 4 3], а результатом команд A=sym([1 2; 2 3]), L=eig(A), L1=double(L) будут символьная матрица A, символьный вектор собственных чисел L и его числовое значение L1:

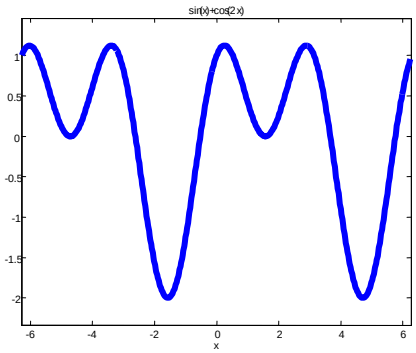
A=[1, 2]	L=2+5 ∧ (1/2)	L1=4.2361
[2, 3]	2-5 ∧ (1/2)	-0.2361

О применении команд **laplace** и **ilaplace** для выполнения прямого и обратного преобразования Лапласа будет сказано позже.

В тулбоксе Symbolic есть два удобных средства для построения графиков, они вызываются командами **ezplot** и **funtool.** Имя первой из них читается как ‘*easy plot*’**,** она предназначена для построения графиков функций, заданных аналитически. На рис. 1.8 приведен пример ее использования для построения графика синуса и функции $y=\sin x+\cos 2x$.



>>ezplot('sin')



>>ezplot('sin(x)+cos(2*x)')

Рис. 1.8

Функция **ezplot** позволяет также рисовать графики функций, заданных неявно, либо параметрически. Например, для того чтобы изобразить гиперболу, заданную уравнением $x^2-y^2=1$ достаточно набрать ezplot('x^2-y^2-1'). Можно также указать пределы, в которых будет изображена функция, например, команда ezplot('-x^3+2*x+1',[-2,3]) изобразит график полинома $-x^3+2x+1$ на интервале [-2, 3].

Для изображения функций, заданных параметрически $x=f(t),\ y=g(t),$ команда вызывается в формате ezplot('f(t)', 'g(t)', [t0, t1]). Пусть, например, требуется нарисовать на плоскости (*x*, *y*) кривую, заданную уравнениями:

$$x = 2t - 4t^3,\ y = t^2 - 3t^4.$$

Набирая в командной строке:

```
>>ezplot('2*t-4*t^3','t^2-3*t^4',[-1,1]),
```

получаем кривую, показанную на рис.1.9

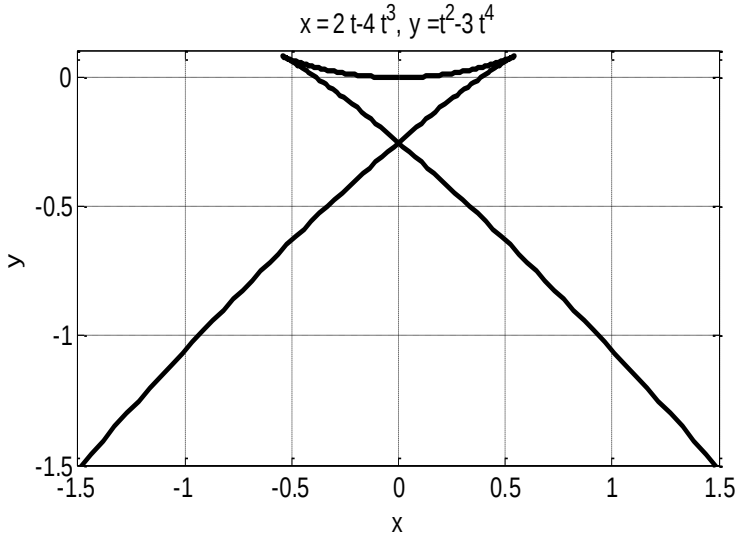


Рис. 1.9

В математической теории катастроф она известна как «ласточкин хвост».

Второе графическое средство тулбокса SYMBOLIC – это функциональный калькулятор **funtool.** Он позволяет выполнять различные манипуляции с двумя функциями $f(x)$ и $g(x)$ от одной переменной, например, их складывать, перемножать, интегрировать.

Задачи и упражнения

1. **Кривые на плоскости.** Используя команду plot, построить графики заданных плоских кривых:

а) кардиоида $x = 2\cos t - \cos 2t,\ y = 2\sin t - \sin 2t$

(траектория точки, лежащей на окружности, которая катится по кругу того же радиуса);

б) нефроида $x = 3cost - cos3t,\ y = 3sint - sin3t$

(траектория точки, лежащей на окружности, которая катится по кругу, радиус которого в два раза больше);

в) дельтоида $x = 2cost + cos2t,\ y = 2sint - sin2t$

(траектория точки, лежащей на окружности, которая катится по внутренней стороне другой окружности, радиус которой в три раза больше);

г) астроида $x = 3cost + cos3t,\ y = 3sint - sin3t$

(траектория точки, лежащей на окружности, которая катится по внутренней стороне другой окружности, радиус которой в четыре раза больше).

2. Собственные векторы. Дана матрица $A = \begin{bmatrix} a_1 & a_3 \\ 0 & a_2 \end{bmatrix}$. Требуется найти ее собственные векторы и написать MATLAB-программу, которая при заданных a_1, a_2, a_3 находит собственные векторы и изображает их на плоскости.

3. Три матрицы. Для заданных матриц найти характеристический полином, собственные числа и собственные векторы

а) $\begin{bmatrix} -1 & 24 \\ -3 & -18 \end{bmatrix}$; б) $\begin{bmatrix} -4 & 1 & 2 \\ -2 & -1 & 2 \\ -1 & 1 & -1 \end{bmatrix}$; в) $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & -1 \end{bmatrix}$.

Ответ:

а) $\lambda^2+19\lambda+90$
$[\lambda_1, \lambda_2] = [-9\ -10]$
$H = \begin{bmatrix} 3 & -8 \\ -1 & 3 \end{bmatrix}$.

б) $\lambda^3+6\lambda^2+11\lambda+6$
$[\lambda_1, \lambda_2, \lambda_3] = [-1\ -2\ -3]$
$H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$.

в) $\lambda^3+\lambda^2+\lambda+1$
$[\lambda_1, \lambda_2, \lambda_3] = [-1\ \ i\ -\ i]$
$H = \begin{bmatrix} 1 & 1 & 1 \\ -1 & i & -i \\ 1 & -1 & -1 \end{bmatrix}$.

4. Типы телосложения. Существуют три типа телосложения: астено-торакальный (худой, тощий), мышечный (атлетический) и дигестивный (толстый, тучный).

Астено-торакальные люди выносливы, способны к монотонным длительным нагрузкам. Они хороши для бега на длинные дистанции, лыж, гребли, велосипеда, спортивной ходьбы. По натуре это стайеры – и в спорте и в жизни. Дигестивные люди хорошо справляются с интенсивными импульсными нагрузками, у них широкие кости и сильные мышцы. Спорт – толкание ядра, штанга. По натуре это спринтеры. В жизни они энергичны, подвижны, легко переключаются с одного дела на другое.

Чтобы определить тип телосложения, используется формула $T=P/H^3$, где P – вес в килограммах, H – рост в метрах. Чем выше T, тем плотнее (упитаннее) человек. Для мужчин, если $T<11.5$, то тип астено-торакальный, если $11.5<T<13$, то тип мышечный, если $T>13$, то тип дигестивный. Для женщин границы сдвинуты на 1 вверх.

Требуется составить MATLAB-программу и построить кривые, которые делят плоскость (P, H) на области, соответствующие каждому типу и определяет тип телосложения по росту и весу (для мужчин и женщин).

5. Вложенные корни. Функция $y(n)$ определена равенством $y(n)=\sqrt{1+\sqrt{2+\sqrt{3+...+\sqrt{n}}}}$.
Требуется составить MATLAB-программу для ее вычисления, найти $y(9)$ и построить график $y(n)$.

Решение. Найти $y(9)$ можно, набрав в командном окне явную формулу

```
>>y=sqrt(1+sqrt(2+sqrt(3+sqrt(4+sqrt(5+sqrt(6+sqrt(7+sqrt(8+sqrt(9)))))))),
```

что даст ответ y =1.7579.

Для вычисления результата при любом n составим программу-функцию sqrt1234, используя оператор **for**.

```
function y=sqrt1234(n)
%задача про вложенные корни

y(n)=sqrt(n);
for i=n-1:-1:1,
  y(i)=sqrt(i+y(i+1));
end
y=y(1);
```

Вычисляя sqrt1234(9), вновь получим ans =1.7579.

Для построения графика $y(n)$ составим программу-сценарий **sqr**.

```
%Program sqr
% программа sqr.m  использует функцию sqrt1234(n)

for j=1:10,
 y(j)=sqrt1234(j);
end,
plot(y,'*'), grid
```

6. Световое табло. Сто светящихся кнопок расположены в виде квадрата 10x10. При нажатии любой кнопки она и все, находящиеся с ней в одном ряду и в одном столбце, меняют свое состояние на противоположное (светящиеся гаснут, а несветящиеся загораются). Какое наименьшее число кнопок нужно нажать, чтобы все кнопки оказались погашенными, если сначала они все светились?

Для проведения экспериментов в MATLAB составьте программу, позволяющую проверять разные стратегии нажатия кнопок.

Указание. Исходное состояние табло изобразим матрицей 10x10

```
>> a0=ones(10);
a0=
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1
```

После нажатия кнопки a(k,l) исходная матрица a0 преобразуется по правилу

```
>>a0(k,:)=~a0(k,:);a0(:,l)=~a0(:,l);a0(k,l)=~a0(k,l);a=a0;
```

где знак тильда означает логическое отрицание.

Рассмотрите стратегии: обход табло по спирали до центра; обход "змейкой" (строка за строкой).

2. Моделирование линейных систем в MATLAB

2.1. Способы описания линейных систем

Прежде чем рассматривать вопросы моделирования динамических систем в MATLAB, напомним основные определения. Рассмотрим систему S с входом u и выходом y (рис. 2.1). Если мгновенное значение выхода $y(t)$ зависит от предыстории, система называется динамической.

Системы с одним входом и одним обозначаются английской аббревиатурой SISO – Single Input Single Output. Системы с несколькими входами и выходами (рис. 2.2) обозначаются MIMO – Multiple Input Multiple Output. В этом случае входы и выходы системы часто записывают в виде векторов, например, $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$.

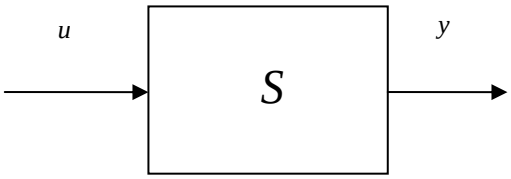


Рис. 2.1

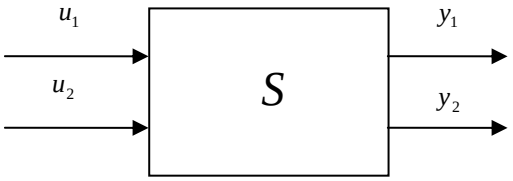


Рис. 2.2

Понятие динамической системы весьма широко. Наиболее хорошо изученными и удобными для анализа являются линейные стационарные динамические системы (LTI – Linear Time Invariant systems).

Для исследования и моделирования таких систем предназначен тулбокс CONTROL, входящий в семейство специальных библиотек пакета MATLAB. Он содержит около 60 команд, служащих для формирования линейных моделей, их анализа, преобразования и моделирования.

В данном разделе ограничимся рассмотрением линейных систем с одним входом и одним выходом (рис. 2.1). Они могут быть заданы передаточной функцией или матричным описанием в пространстве состояния.

Напомним, что передаточной функцией линейной системы называется отношение изображения по Лапласу выходного сигнала $y(p)$ к изображению по Лапласу входного сигнала $u(p)$ при нулевых начальных условиях: $Q(p) = \dfrac{y(p)}{u(p)}$. Например, передаточная функция интегратора равна $\dfrac{1}{p}$, передаточная функция апериодического звена $\dfrac{k}{Tp+1}$, передаточная функция идеального колебательного звена $\dfrac{k}{p^2+\omega^2}$.

Передаточная функция линейной системы n-го порядка представляет собой отношение двух полиномов:

$$Q(p) = \frac{b_m p^m + b_{m-1} p^{m-1} + \ldots + b_0}{a_n p^n + a_{n-1} p^{n-1} + \ldots + a_0}, \quad m \le n.$$

Корни $z_1, \ldots, z_m$ числителя B(p) называются нулями системы (*zeros*), корни $p_1, .., p_n$ знаменателя A(p) – полюсами системы (*poles*). Знание нулей и полюсов позволяет представить описание системы в виде

$$Q(p) = k\frac{(p-z_1)\cdot\ldots\cdot(p-z_m)}{(p-p_1)\cdot\ldots\cdot(p-p_n)},$$

где коэффициент $k = b_m / a_n$.

В MATLAB полиномы B(p) и A(p) представляются векторами-строками их коэффициентов: num=[b_m, ..., b_0] и den=[a_n, ..., a_0], а нули и полюсы – векторами-столбцами z и p: z=$[z_1, \ldots, z_m]^T$, p=$[p_1, \ldots, p_n]^T$. Например, для передаточной функции третьего порядка

$$Q(p) = \frac{2p^2 - 6p + 4}{p^3 + 3p^2 + 5p + 2}$$

имеем num=[2 -6 4], den=[1 3 5 2].

Для создания модели, заданной в виде передаточной функции, используется конструктор **tf** (от *Transfer Function*). Его входными параметрами являются массивы коэффициентов числителя и знаменателя, например, по команде sys=tf(2,[3 1]) будет сформирована система sys с передаточной функцией $\frac{2}{3p+1}$. Доступ к числителю и знаменателю объекта, заданного в виде передаточной функции, можно получить, обращаясь к полям *num* и *den*. Так, набрав sys.num{1}, получим ответ ans = 0 2, а набрав sys.den{1}, получим ans = 3 1.

Другой способ задания линейных систем – описание в пространстве состояний. Оно включает четыре матрицы A, B, C, D и имеет вид:

$$\dot{X} = AX + BU, \qquad Y = CX + DU,$$

где U, Y – векторы входных и выходных сигналов, $X = [x_1 \ \ldots \ x_n]^T$ – вектор состояния, A, B, C, D – постоянные матрицы. Если у системы один вход и один выход, то B – вектор-столбец, C – вектор-строка, D – число (часто равное нулю).

Для того чтобы создать в MATLAB объект, заданный описанием в пространстве состояний, используется конструктор **ss** (от *State Space* – пространство состояний). Его входными параметрами служат матрицы A, B, C, D системы.

Пример. Объект второго порядка описывается уравнениями

$$\begin{aligned}\dot{x}_1 &= -2x_1 + x_2,\\ \dot{x}_2 &= 3x_1 - 5x_2 + 4u,\\ y &= x_1 + 7x_2.\end{aligned}$$

Им соответствуют следующие матрицы описания в пространстве состояний:

$$A = \begin{bmatrix} -2 & 1 \\ 3 & -5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 4 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 7 \end{bmatrix}, \quad D = 0.$$

Чтобы ввести это описание в MATLAB, следует набрать текст:
>>A=[-2, 1; 3, -5]; B=[0; 4]; C=[1, 7]; D=0; sys=ss(A, B, C, D).

Доступ к полям a, b, c, d созданного ss-объекта можно получить так же, как к полям num и den tf-объекта, например, набрав sys.c, получим ответ ans=1 7.

Мы рассмотрели два способа представления линейных моделей в MATLAB – с помощью передаточной функции Q(p) (tf-модель) и в пространстве состояний $\dot{X} = AX + BU, \quad Y = CX + DU$ (ss-модель). Аналитическая связь между этими описаниями дается формулой

$$Q(p) = C(pE - A)^{-1}B + D.$$

В MATLAB переход от ss-модели к tf-модели можно осуществить, используя конструктор **tf** с аргументом в виде исходной ss-модели. Обратный переход выполняется при помощи команды **ss** с аргументом в виде tf-модели. Выполним переход к передаточной функции для нашего примера, набрав sys1=tf(sys). На экране появится текст: transfer function $\frac{28s+60}{s^2+7s+7}$, извещающий о том, что сформирована tf-модель sys1 с указанной передаточной функцией.

Если после этого ввести команду sys2=ss(sys1), то мы снова получим ss-модель, однако матрицы A, B, C, D в ней будут уже иными:

$$a = \begin{bmatrix} -7 & -3.5 \\ 2 & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} 8 \\ 0 \end{bmatrix}, \qquad c = \begin{bmatrix} 3.5 & 3.75 \end{bmatrix}, \qquad d = 0.$$

Тем не менее, системы sys и sys2 эквивалентны, поскольку у них одинаковые передаточные функции (в этом можно убедиться с помощью команды tf(sys2)). Они представляет собой две различные реализации одной и той же системы в пространстве состояний.

В MATLAB имеется еще один способ описания систем – нуль-полюсное или zpk-описание. Оно получается в результате разложения на множители числителя и знаменателя передаточной функции. Zpk-представление передаточной функции имеет вид $Q(p) = k\frac{(p-z_1)\cdot\ldots\cdot(p-z_m)}{(p-p_1)\cdot\ldots\cdot(p-p_n)}$, где $k = b_m/a_n$ – коэффициент усиления (gain). Такое описание создается конструктором **zpk**, который можно использовать также для перехода от ss-модели или tf-модели к zpk-описанию.

В частности, выполняя команду sys3=zpk(sys2), получим систему с описанием

$$\frac{28(s+2{,}143)}{(s+5{,}791)(s+1{,}209)}.$$

Доступ к отдельным элементам zpk-модели осуществляется так же, как и для tf-моделей, например, sys3.z даст ans=[-2.1429], т.е. нули системы.

Отметим возможность объединения нескольких моделей в более сложные системы. Например, произведение sys=sys1*sys2 означает последовательное соединение систем sys1 и sys2, а сумма sys=sys1+sys2 означает их параллельное соединение. Можно использовать также знаки вычитания и деления – , / (подумайте, какому соединению они соответствуют?).

2.2. Моделирование линейных систем

Главная цель моделирования – получение реакции системы на те или иные входные сигналы. Входные воздействия могут быть стандартными (единичный скачок, импульс, синусоида), либо произвольными, когда входной сигнал формируется как некоторая функция времени.

В теории линейных систем широко используются две временны*е* характеристики систем – импульсная весовая функция и импульсная переходная характеристика. Импульсной весовой функцией $q(t)$ называется реакция системы на входной сигнал в виде дельта-функции, т.е. на бесконечно короткий импульс единичной площади, действующий в момент времени t=0.

Импульсной переходной характеристикой (переходной функцией) $h(t)$ называется реакция системы на входной сигнал в виде единичной ступеньки (такой сигнал может рассматриваться, как интеграл от дельта-функции). Весовая и переходная функции связаны соотношением $q(t) = \frac{d}{dt} h(t).$

Изображения по Лапласу дельта-функции и единичного скачка равны 1 и 1/p соответственно. Изображение весовой функции по Лапласу равно передаточной функции системы, а изображение переходной характеристики равно передаточной функции, деленной на p.

Отсюда вытекает возможность получения весовой функции с помощью обратного преобразования Лапласа от передаточной функции. Например, передаточная функция интегратора $Q(p) = \frac{1}{p},$ поэтому его весовая функция $q(t)$=1; передаточная функция апериодического звена $Q(p) = \frac{b}{p+a},$ поэтому его весовая функция имеет вид $q(t) = be^{-at}.$ Аналогично, для колебательного звена с передаточной функцией $Q(p) = \frac{k}{p^2 + k^2}$ для весовой функции получаем $q(t) = \sin kt.$

Эта возможность отыскания весовой функции по известной передаточной функции реализуется в MATLAB командой **ilaplace** тулбокса SYMBOLIC, выполняющей обратное преобразование Лапласа (*inverse Laplace*). В частности, для получения весовой функции колебательного звена достаточно набрать syms k p; q=ilaplace(k/(p^2+k^2), чтобы получить результат q=sin(k*t). Команда **laplace** служит для выполнения обратной операции.

Например,чтобы получить изображение по Лапласу функции sin 10t, достаточно набрать:
>>sym p, t; y=laplace(sin(10*t), t,p).

Получаем ответ: y=10/(p^2+100), т.е. $y(p) = \frac{10}{p^2 + 100}.$

Другой путь аналитического получения весовой и переходной функций связан с символьным решением дифференциальных уравнений. Если система задана описанием в пространстве состояний (ss-модель)

$$\dot{X} = AX + bu, \qquad y = cX,$$

то дифференциальные уравнения для весовой функции q и переходной характеристики h имеют вид

$$\dot{X} = AX, \qquad q = cX, \quad X(0) = b;$$
$$\dot{X} = AX + b, \quad h = cX, \quad X(0) = 0.$$

Например, описание в пространстве состояний апериодического звена с передаточной функцией $Q(p) = \frac{b}{p+a}$ имеет вид

$$\dot{x} = -ax + bu, \qquad y = x.$$

Его весовая функция является решением дифференциального уравнения

$$\dot{x} + ax = 0, \qquad x(0) = b,$$

откуда $q(t) = be^{-at}$. Для получения переходной функции надо решить неоднородное дифференциальное уравнение

$$\dot{x} + ax = b, \qquad x(0) = 0.$$

Его решение получаем, складывая общее решение однородного уравнения $x_{одн} = Ce^{-at}$ и частное решение неоднородного уравнения $x_{част} = -a/b$:

$$h(t) = Ce^{-at} - a/b = \frac{a}{b}\left(1 - e^{-at}\right).$$

Постоянная C найдена из начального условия $h(0)=0$.

Этот способ отыскания весовой и переходной функций может быть реализован в MATLAB с помощью команды **dsolve** тулбокса SYMBOLIC. Чтобы получить функцию $h(t)$ для нашего примера, нужно набрать:

```
>>syms a b; h=dsolve('Dx+a*x=b, x(0)=0').
```

На дисплее появится ответ h=b/a-exp(-a*t)*b/a, совпадающий с полученным выше.

В библиотеке CONTROL весовая и переходная характеристики получаются путем прямого численного моделирования. Соответствующие команды называются **impulse** и **step**, это сокращения от *impulse function* (импульсная функция) и *step function* (переходная функция). Для выполнения этих операций во всех случаях необходимо предварительно ввести исходную информацию о системе в виде tf-модели или ss-модели, а также сформировать массив равноотстоящих моментов времени t, задающий временной интервал моделирования.

Существует несколько модификаций команды **impulse.** Простейшая из них имеет вид impulse(sys), ее результатом является график весовой функции. Если набрать impulse(sys,10), то график будет построен на интервале $0 \le t \le 10$ (в предыдущем случае MATLAB сам определял длительность интервала).

Более содержательные варианты этой команды получаются, если использовать выходные параметры (их число можно задавать от одного до трех). Наиболее полный вариант имеет вид [y,t,X]=impulse(sys,t), он предполагает, что sys – это ss-модель. Здесь t – массив точек времени, который нужно сформировать заранее (например, t=0:.1:10), y – выходной сигнал, X – вектор состояния. Если моделируется система второго порядка на указанном интервале времени, то массив X будет содержать два столбца по 101 числу в каждом. Первый столбец – это отсчеты функции $x_1(t)$, второй – отсчеты функции $x_2(t)$. Столбец y будет содержать 101 значение выходного сигнала $y(t)$.

Для построения графиков этих сигналов нужно использовать команды plot(t,y), plot(t,X), plot(t,X(:,1)). В первом случае будет выведен график функции $y(t)$, во втором – графики обеих переменных $x_1(t)$, $x_2(t)$, в третьем – график одной переменной $x_1(t)$. Команда plot(X(:,1), X(:,2)) построит траекторию на фазовой плоскости $x_2 = f(x_1)$..

Функция **step** обеспечивает получение переходной функции модели, т.е. реакции на входной сигнал в виде единичной ступеньки. Она имеет те же модификации, что и **impulse**:

```
>>step(sys), step(sys,T), y=step(sys,t), [y,t,X]=step(sys,t).
```

Здесь, как и раньше, в качестве второго входного аргумента можно указывать либо число T (последний момент времени), либо массив t (все точки временного интервала).

Аналогичный синтаксис имеет и команда **initial** (от *initial condition* – начальные условия). Она позволяет моделировать свободное движение системы, заданной своим матричным описанием, из начальных условий X_0 (входной сигнал при этом не подается $u=0$).

Просто initial(sys,X_0) строит график выходного сигнала, в более полном варианте используются три выходных параметра [y,t,X]=initial(sys,X0,t).

Основная команда, применяемая для моделирования линейных систем – это команда **lsim** (от *linear simulation* – линейное моделирование). Она обеспечивает получение реакции модели на произвольный входной сигнал $u(t)$, представленный массивом своих отсчетов. Простейшая модификация этой команды lsim(sys,u,t) выводит график выходного сигнала системы. Естественно, предварительно надо сформировать массив времени t, входной сигнал и ввести описание системы sys. Например, задав:

>>*t*=0:1:10; u=sin(t); sys=tf(1,[1 1]; lsim(sys,u,t),

получим реакцию апериодического звена с передаточной функцией $\frac{1}{p+1}$ на входной сигнал $u=\sin t$ при $0 \le t \le 10$. Другие модификации этой команды могут использовать выходные параметры и начальные условия:

>>y=lsim(sys,u,t), [y,t,X]=lsim(sys,u,t), [y,t,X}=lsim(sys,u,t,X0).

Последний вариант требует задания системы sys в виде ss-модели. Построение графиков производится с помощью команды **plot,** например plot(t,u,t,y).

В пакете MATLAB имеется еще ряд возможностей для расчета отклика линейных систем на известные входные воздействия. К ним относятся:

– использование матричной экспоненты для получения весовой и переходной функций (команда **expm**);
– вычисление интеграла свертки входного сигнала и весовой функции системы (команда **trapz**);
– аналитическое решение дифференциальных уравнений (команда **dsolve** тулбокса SYMBOLIC);
– использование прямого и обратного преобразований Лапласа (команды l**aplace** и **ilaplace** тулбокса SYMBOLIC);
– структурное моделирование в SIMULINK.

Часть из них будет рассмотрена в следующих разделах.

2.3. Частотные характеристики

В библиотеке CONTROL имеются функции для исследования систем в частотной области. К основным частотным характеристикам относятся амплитудно-частотная характеристика (АЧХ), фазо-частотная характеристика (ФЧХ) и амплитудно-фазовая характеристика (АФХ), называемая также диаграммой Найквиста. Все они могут быть получены из передаточной функции системы Q(p) после формальной подстановки $p = i\omega$, где $i = \sqrt{-1}$, ω – вещественная переменная, изменяющаяся в пределах от нуля до бесконечности. С физической точки зрения ω – это частота синусоидального сигнала, подаваемого на вход системы, а $Q(i\omega)$ – Фурье изображение соответствующего выходного сигнала.

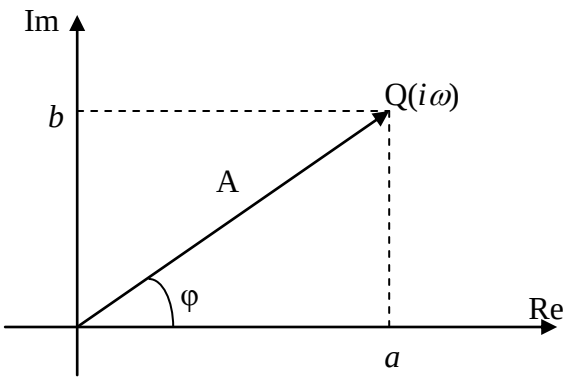


Рис. 2.3

Число $Q(i\omega) = a(\omega) + ib(\omega)$ – комплексное, оно может быть изображено на комплексной плоскости с помощью вектора с координатами (a, b) (рис. 2.3). Длина этого вектора определяется формулой $A = \sqrt{a^2 + b^2}$, а угол φ – соотношением $tg\varphi = \frac{b}{a}$.

Если изменять частоту ω от нуля до бесконечности, конец вектора опишет некоторую траекторию на комплексной плоскости. Она называется

амплитудно-фазовой характеристикой системы или годографом Найквиста (в последнем случае берут $-\infty < \omega < \infty$).

Зависимость длины вектора от частоты $A(\omega)$ называется амплитудно-частотной характеристикой, а зависимость $\varphi(\omega)$ – фазо-частотной характеристикой. Графики АЧХ и АФХ часто рисуют в логарифмическом или полулогарифмическом масштабе. Логарифмическая амплитудная характеристика (ЛАХ) описывается формулой $20\log_{10} A(\omega)$ и изображается в логарифмическом масштабе. ЛАХ и АФХ, построенные в логарифмическом масштабе, называются в зарубежной литературе диаграммами Боде.

Опишем процедуру получения этих характеристик в MATLAB. Будем считать, что система задана своей tf-моделью и сформирован вектор частот W, например:

```
>>sys=tf([1 1], [1 1 1]);  W=0: .1: 10;
```

Для получения частотного отклика (frequency response) $Q(i\omega)$ используется команда **freqresp.** В нашем примере передаточная функция и частотный отклик имеют вид

$$Q(p) = \frac{p+1}{p^2+p+1}, \quad Q(i\omega) = \frac{i\omega+1}{i\omega+1-\omega^2}.$$

По команде `H=freqresp(sys,W)` будет вычислен массив комплексных чисел, содержащих 101 значение функции $Q(i\omega)$.

Команда `plot(H(:))` построит зависимость мнимой части этих чисел от вещественной, т.е. график АФХ (рис. 2.4).

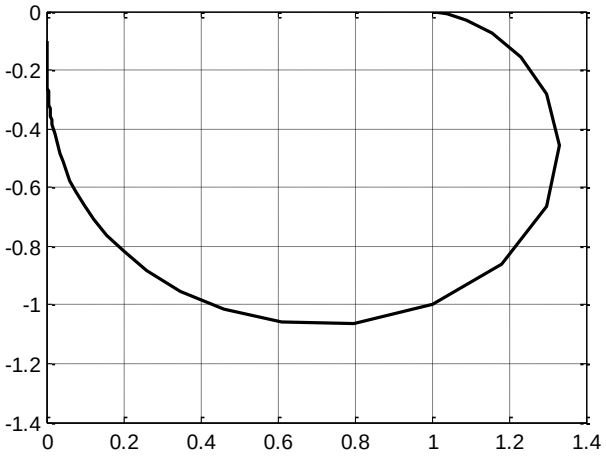


Рис. 2.4

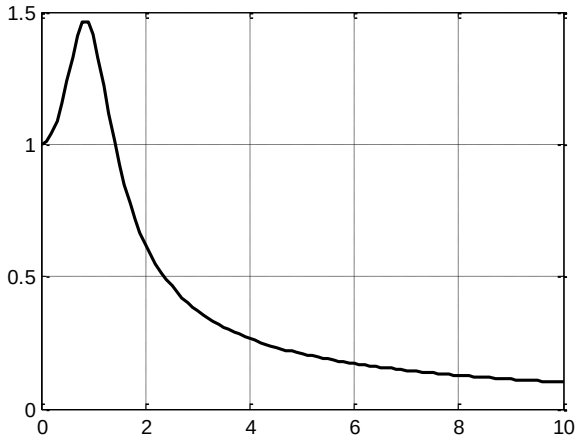


Рис. 2.5

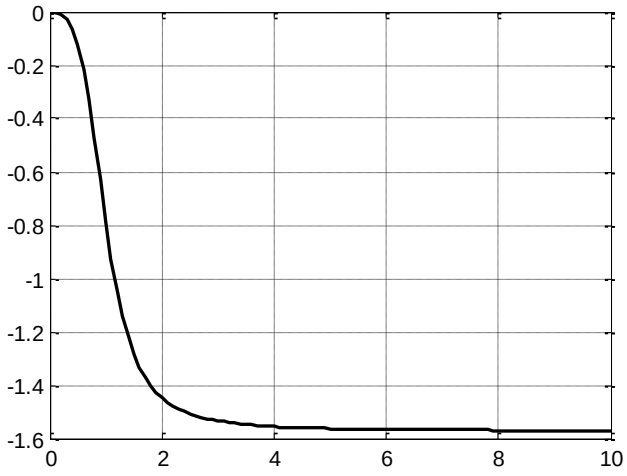


Рис. 2.6

Для построения графиков АЧХ и ФЧХ надо найти модуль и аргумент этих чисел, для чего служат команды **abs** и **angle:**

```
>>A=abs(H(:));  f=angle(H(:));    plot(W,A), plot((W,f).
```

Результат показан на рис. 2.5, 2.6, из них видно, что с ростом частоты АЧХ стремится к нулю, а ФЧХ – к величине $-\pi/2$.

Более короткий путь построения частотных характеристик – использование команд **nyquist** и **bode.** Команда nyquist(sys) построит диаграмму Найквиста, т.е. АФХ (при этом рассматриваются как положительные, так и отрицательные частоты). По команде bode(sys) будут построены графики АЧХ и ФЧХ в логарифмическом масштабе.

Еще один способ построения графиков частотных характеристик, а также импульсной и переходной функций – это использование специального графопостроителя тулбокса CONTROL, который вызывается командой **ltiview.**

2.4. Анализ линейных систем

К классическим задачам исследования линейных систем относятся отыскание нулей и полюсов передаточной функции, а также анализ устойчивости, ограниченности, управляемости и наблюдаемости. Коротко охарактеризуем команды библиотеки CONTROL, предназначенные для решения этих задач (табл. 1).

Таблица 1

pole	**eig**	**ctrb**	**gram**	**dcgain**
zero (tzero)	**pzmap**	**obsv**	**minreal**	**roots**

Наиболее известный способ проверки устойчивости связан с вычислением полюсов системы, т.е. корней ее характеристического уравнения.

Критерий устойчивости. Для того чтобы линейная система была устойчивой, необходимо и достаточно, чтобы все корни характеристического уравнения лежали в левой полуплоскости.

Пример (анализ устойчивости и ограниченности). Дана неоднородная система дифференциальных уравнений

$$\dot{x} = 2y - 4x + a,$$

$$\dot{y} = 2x - y + b.$$

Требуется проанализировать ее устойчивость и выяснить, при каких постоянных *a* и *b* все решения системы ограничены.

Решение. Запишем систему в матричной форме $\dot{X} = AX + B$, где $A = \begin{bmatrix} -4 & 2 \\ 2 & -1 \end{bmatrix}$, $B = \begin{bmatrix} a \\ b \end{bmatrix}$ и найдем корни ее характеристического уравнения, т.е. собственные числа матрицы A:

```
>> A=[-4 2;2 -1]; eig(A)
ans = -5    0
```

Одно из собственных чисел отрицательно, а другое лежит на мнимой оси, следовательно, однородная система находится на границе устойчивости.

Для ответа на вопрос об ограниченности неоднородной системы найдем ее аналитическое решение с помощью команды **dsolve**:

```
>> syms a b;s=dsolve('Dx=2*y-4*x+a,Dy=2*x-y+b')
s =     x: [1x1 sym]
        y: [1x1 sym]
>> x=s.x
x =2/5*exp(-5*t)*C1-1/10*b+1/5*a+2/5*t*b+1/5*t*a+1/2*C2
>> y= s.y
y =-1/5*exp(-5*t)*C1+4/5*t*b+2/5*t*a+C2
```

Приведем подобные члены и перейдем к обычной нотации:

$$\begin{aligned} x &= (a+2b)t + 2c_1e^{-5t} + a + c_2 \\ y &= 2(a+2b)t + c_1e^{-5t} + 2c_2 \end{aligned}.$$

Решение будет ограниченным, если $a=-2b$, тогда $x = 2c_1e^{-5t} + a + c_2$, $y = c_1e^{-5t} + 2c_2$.

Нули и полюсы системы, заданной передаточной функцией – это просто корни z_i и p_i полиномов, стоящих в числителе и знаменателе. Поэтому для вычисления вектора нулей z и вектора полюсов p передаточной функции Q(p) = num/den, могут использоваться команды z = roots(num); p = roots(den). Если система sys задана как tf-модель или ss-модель, то используются команды p=pole(sys), z=zero(sys). Для нахождения полюсов допустимо также использование команды p=eig(sys), что эквивалентно команде p=eig(sys.a), т.е. вычислению собственных чисел матрицы A. Функция **tzero** (от *transfer zeros* – передаточные нули) позволяет находить нули системы по матрицам описания в пространстве состояний z=tzero(A,B,C,D).

Функция **pzmap** предназначена для одновременного вычисления нулей и полюсов. Если набрать [p, z]=pzmap(sys), то будут выведены столбцы p и z полюсов и нулей, а просто pzmap(sys) показывает расположение нулей и полюсов на комплексной плоскости (на графике нули изображаются ноликами, а полюсы – крестиками).

При анализе управляемости и наблюдаемости линейных систем используются матрицы управляемости R и наблюдаемости D, построенные на основе матриц A, B , C описания в пространстве состояний:

$$\mathrm{R} = [\mathrm{B},\ \mathrm{AB},\ ...,\ \mathrm{A}^{n-1}\mathrm{B}],\quad \mathrm{D} = \begin{bmatrix} \mathrm{C} \\ \mathrm{CA} \\ ... \\ \mathrm{CA}^{n-1} \end{bmatrix}.$$

Приведем соответствующие критерии.

Критерий управляемости. Для того чтобы система была управляемой, необходимо и достаточно, чтобы матрица управляемости имела полный ранг: rankR = *n*.

Критерий наблюдаемости. Для того чтобы система была наблюдаемой необходимо и достаточно, чтобы матрица наблюдаемости имела полный ранг: rankD = *n*.

Критерий минимальности. Для того чтобы система была минимальной необходимо и достаточно, чтобы обе матрицы R и D имели полный ранг: rankR = *n*, rankD = *n*.

Формирование матриц управляемости и наблюдаемости производится с помощью команд **ctrb** и **obsv** (от *controlability* и *observability*). Их синтаксис одинаков: R=ctrb(sys), D=obsv(sys). В качестве аргументов можно использовать непосредственно матрицы A, B, C описания в пространстве состояний, например R=ctrb(A, B), D=obsv(A, C) или R=ctrb(sys.a, sys.b), D=obsv(sys.a, sys.c). Для вычисления ранга этих матриц используется команда **rank.**

Другой способ проверки управляемости и наблюдаемости опирается на вычисление рангов так называемых грамианов управляемости и наблюдаемости. Для их нахождения служит команда **gram,** дополнительные сведения о ней можно найти в разд. 4.4, 4.5.

Если система неуправляема или ненаблюдаема, то ее порядок может быть понижен путем удаления неуправляемых и ненаблюдаемых подсистем. У SISO-систем это эквивалентно сокращению совпадающих нулей и полюсов передаточной функции. Для этой цели используется команда **minreal** (от *minimal realization*).

Ее можно использовать с одним и двумя входными аргументами: sys1=minreal(sys) и sys1=minreal(sys, eps). Второй аргумент позволяет указывать допуск ε, задающий степень близости сокращаемых нулей и полюсов.

Пример (анализ минимальности). Рассмотрим систему с передаточной функцией $Q(p)=\dfrac{p+1}{p^2+3p+2}$. Она имеет один нуль $z_1 = -1$ и два полюса $p_1 = -1$, $p_2 = -2$. Система неминимальна, поскольку $p_1 = z_1$. Сформируем tf-модель этой системы и найдем ее нули и полюсы:

```
>>s=tf([1 1],[1 3 2])
     s + 1
  ----------
  s^2 + 3 s + 2
```

```
>>zero(s)
   -1
```

```
>>pole(s)
   -2
   -1
```

```
>>eig(s)
   -2
   -1
```

```
>> [p, z]=pzmap(s)
   p = – 2    z =  – 1
                   – 1
```

Расположение нулей и полюсов на комплексной плоскости можно получить, набрав команду pzmap(s) без выходного аргумента (на рис. 2.7, полюсы помечены крестиками, нуль – ноликом).

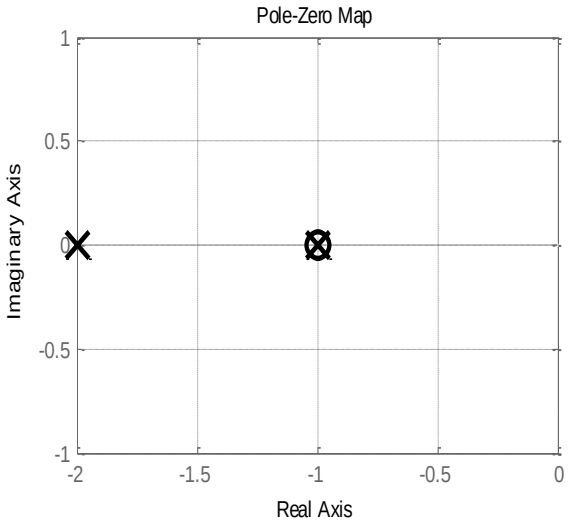


Рис. 2.7

Оба полюса отрицательны, следовательно, система устойчива. Наличие диполя (совпадающего нуля и полюса) в точке (-1, 0) говорит о неминимальности системы.

Перейдем к ss-модели и проанализируем ее управляемость и наблюдаемость, используя команды **ctrb**, **obsv** и **rank**:

```
>>s1 = ss(s)                >>R=ctrb(s1)      >>D=obsv(s1)      >>rank(R)   >>rankD
a =  -3  -2      b =  2     R =     2   -6     D = 0,5  0,5
      1   0           0     c =  0.5  0.5          0    2          -1   -1          2           1
```

Анализируя ранги матриц R и D, заключаем, что система s1 управляема, но ненаблюдаема, следовательно, ее порядок может быть понижен.

Найдем передаточную функцию минимальной реализации: q=minreal(s), получаем ответ $q = \dfrac{1}{s+2}$. К тому же результату приходим, сокращая числитель и знаменатель исходной передаточной функции $Q(p)$ на общий множитель p+1.

Заметим, что статический коэффициент усиления при переходе к минимальной реализации не изменился:

```
>>K=dcgain(s)        >>K=dcgain(q)
K=0.5                k=0.5
```

Весовая и переходная функции также остаются прежними. В этом можно убедиться с помощью команд impulse(s, q), step(s, q), по которым будут построены графики указанных функций для обеих систем.

Команды **ctrb** и **obsv** можно использовать при работе с символьными выражениями, например, когда часть элементов матриц A, b, c заданы в буквенном виде.

Пример (анализ управляемости и наблюдаемости системы третьего порядка). Объект управления задан описанием в пространстве состояний

$$\dot{X} = AX + bu, \quad y = cX, \qquad A = \begin{bmatrix} -a_1 & 1 & 0 \\ 0 & -a_2 & 0 \\ 0 & 1 & -a_1 \end{bmatrix}, \; b = \begin{bmatrix} a_2 \\ a_3 \\ 1 \end{bmatrix}, \; c = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix},$$

где $X \in R^3$ – вектор состояний, u, y – входной и выходной сигналы.

Требуется проанализировать его управляемость и наблюдаемость.

Решение. Вводим исходные символьные матрицы

```
>> syms a1 a2 a3 real
>> A=[-a1 1 0;0 -a2 0;0 1 -a1], b=[a2;a3;1]; c=[1 1 1];
```

Формируем матрицы управляемости и наблюдаемости R=[b, Ab, A^2b], D=[c^T, $(cA)^T$, $(cA^2)^T$]^T:

```
>> R=ctrb(A, b)
[ a2,     -a1*a2+a3,     -a1*(-a1*a2+a3)-a2*a3]
[a3,      -a2*a3,                 a2^2*a3   ]
[1,       a3-a1,          -a2*a3-a1*(a3-a1)]
```

```
>> D=obsv(A, c),
[  1,          1,          1 ]
[ -a1,         2-a2,       -a1]
[a1^2,    -2*a1-(2-a2)*a2,  a1^2]
```

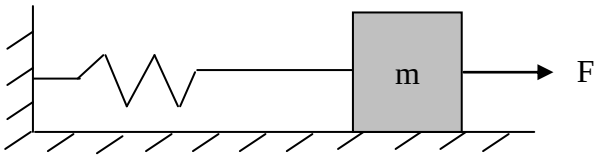
Вычисляем определители: det(R) =0, det(D) =0. Оба определителя равны нулю, следовательно, система неуправляема и ненаблюдаема.

Вырожденность матрицы наблюдаемости очевидна (ее первый и третий столбцы совпадают). Вырожденность матрицы управляемости «на глаз» обнаружить значительно сложнее.

Задачи и упражнения

1. Для приведенных ниже систем второго порядка найти дифференциальное уравнение, передаточную функцию, весовую функцию, описание в пространстве состояний и выполнить моделирование в MATLAB.

а) Механическая система с трением

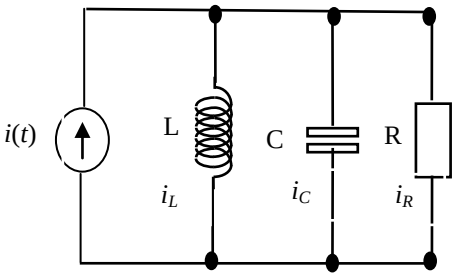


m – масса груза,
k – жесткость пружины,
ρ – коэффициент трения,
$F_{сопр} = \rho v$ – сила трения

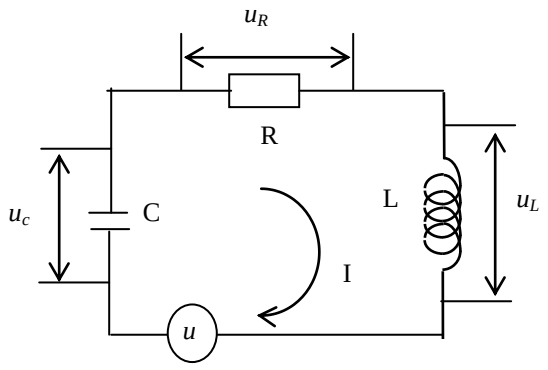
Входной сигнал – сила F, выход – координата x или скорость $v = \dot{x}$.

б) Электрическая схема (параллельный колебательный контур).

в) Электрическая схема (последовательный колебательный контур).



Входной сигнал i,
выходной сигнал i_L



Входной сигнал u;
выходной сигнал u_c или u_R.

Численные значения параметров L, m принять равными числу букв в Вашей фамилии; R, ρ – числу букв в имени/10; c, k – числу букв в отчестве.

Указание. При построении математической модели использовать уравнения:
второй закон Ньютона

$$ma + \rho v + kx = F, \quad \text{где} \quad v = \dot{x}, \qquad a = \ddot{x};$$

законы Кирхгофа и Ома

$$i_L + i_C + i_R = i(t), \quad u_L + u_c + u_R = u, \quad i = u / z(p), \quad z_R = R, \quad z_C = \frac{1}{Cp}, \quad z_L = Lp.$$

2. Используя команду **dsolve,** найти решение дифференциального уравнения

$$\dddot{x} + \ddot{x} + \dot{x} + x = 0,$$

с начальными условиями а) $x_0 = \dot{x}_0 = 1, \quad \ddot{x}_0 = -2, \quad$ б) $x_0 = 60, \quad \dot{x}_0 = \ddot{x}_0 = 0.$

Ответ: а) $x = e^{-t} - \sin t$, б) $x = 30(e^{-t} + \sin t + \cos t) = 30e^{-t} + 42{,}4\sin(t + 45^o)$.

3. Решить в символьном виде следующие дифференциальные уравнения:

а) $\dddot{x} - 2\ddot{x} + \dot{x} = 4,\ x(0) = 1,\ \dot{x}(0) = 2,\ \ddot{x}(0) = -2;$

б) $\ddot{x} + n^2 x = a\sin(nt + \alpha),\ x(0) = \dot{x}(0) = 0.$

Ответ: а) $x(t) = 4t + 3 - 2e^t$, б) $x(t) = \frac{a}{2n^2}\left[\sin nt\cos\alpha - nt\cos(nt + \alpha)\right].$

4. Найти реакцию апериодического звена с передаточной функцией $Q(p) = \frac{1}{p + a}$ на входной сигнал $u = e^{-\lambda t}$. Определить максимум выходного сигнала. Выполнить численное и символьное моделирование в MATLAB, приняв a=1; b=2; T=3.

Решение.

а) Программа моделирования в MATLAB (тулбокс CONTROL):

```
a = 1;    b = 2;    T = 3;              % Задание параметров
t = linspace(0, T); u = exp(-b*t);      % Входной сигнал
sys = tf(1,  [1  a]);                   % Описание системы
y = lsim(sys, u, t);                    % Моделирование
ym = max(y);  plot(t, y).               % Вывод результатов
```

б) Аналитическое решение (тулбокс SYMBOLIC); $y(t) = \frac{1}{\lambda - a}\left(e^{-at} - e^{-\lambda t}\right).$

При $\lambda = a$ (особый случай) решение принимает вид: $y(p) = \frac{1}{(p + a)^2}; \quad y(t) = te^{-at}.$

При этом $t_M = 1/a,\ y(t_M) = 1/ae.$

5. Дано дифференциальное уравнение $y^{(4)} - y = e^{-t}$, с начальными условиями: $y_0 = 3,\ \dot{y}_0 = -5.25,\ \ddot{y}_0 = 7.5,\ \dddot{y}_0 = -5.75.$ Найти аналитическое решение в MATLAB. Рассмотреть три варианта численного моделирования и сравнить их по точности.

Возможная техническая интерпретация задачи – управление неустойчивым объектом, считая y отклонением, а $u = e^{-t}$ – управлением.

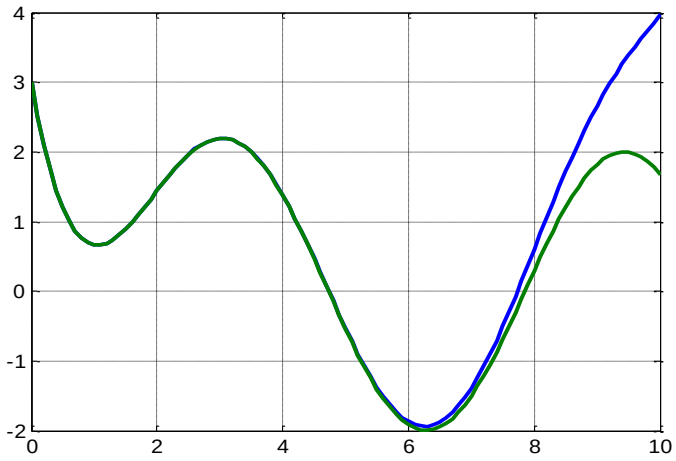
Ответ: Аналитическое решение исходного уравнения: $y = (5 - 0{,}25t)e^{-t} - 2\cos t$.

Отметим, что в решение не вошла неустойчивая мода C_1e^t. Это означает, что программное управление e^{-t} обеспечивает колебательное движение объекта около неустойчивого положения равновесия. На практике неизбежно появление малых отклонений, которые со временем будут лавинообразно нарастать.

Три варианта численного моделирования.

а) Раздельная реализация управления и системы.

```
sys=tf(1,[1 0 0 0 -1]); s=ss(sys);
T=10;t=0:.1:T;
u=exp(-t);
X0=[-5.75;7.5;-5.25;3];
y=lsim(s,u,t,X0);
 z=(5-t/4).*exp(-t)-2*cos(t);
plot(t,y,t,z)
```



Компьютер выдает правильное решение при T< 8 сек., далее машинное решение «разваливается».

б) Совместная реализация управления и системы. Входной сигнал $u = e^{-t}$ формируется с помощью апериодического звена, в качестве вектора b используется вектор X_0. Общий порядок системы равен пяти. Ранг матрицы управляемости $R = [b, Ab, ..., A^4b]$ равен 4, т.е. реализация неминимальна. Тем не менее, теперь правильное решение сохраняется втрое дольше (на интервале 25 сек.).

в) Дальнейшего увеличения точности решения можно достичь, если выполнить редукцию и перейти к минимальной реализации 4-го порядка. В пакете MATLAB такая редукция выполняется с помощью команды **minreal**. Аналитически редукцию можно выполнить, вычислив передаточную функцию

$$Q(p) = \frac{-5{,}75p^4 + 1{,}75p^3 + 2{,}25p^2 - 2{,}25p + 4}{(p+1)(p^4 - 1)}$$

и затем сократив числитель и знаменатель на общий множитель p–1.

3. МОДЕЛИРОВАНИЕ в SIMULINK

3.1. Запуск и начало работы в SIMULINK

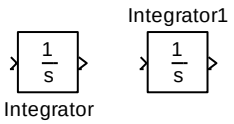
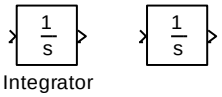
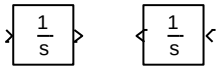
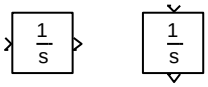
Наиболее простым и удобным средством для моделирования динамических систем в пакете MATLAB является SIMULINK. Он поставляется в виде отдельного приложения и содержит библиотеки различных блоков, из которых в рабочем окне строится структурная схема модели. Процесс программирования в явном виде отсутствует, поэтому иногда говорят о визуальном моделировании. Особенно удобен такой подход для моделирования систем автоматического управления, заданных структурными схемами, дифференциальных уравнений (схемы для них строятся по методу понижения производных), механических систем и электрических схем.

SIMULINK можно запустить, набрав команду **simulink** или нажимая на кнопку  в линейке меню. При этом открывается окно браузера библиотек. Для создания новой модели выбираем пункт меню “File/New/Model”. То же самое можно сделать при помощи кнопки на панели инструментов. После выделения одной из библиотек (например, непрерывные блоки – **Continuous**) на правой панели появится список доступных блоков. Выбранный блок следует

перетащить мышью в окно модели. Если щелкнуть мышью по блоку в модели, то откроется окно параметров блока.

При нажатии правой кнопки мыши на блоке открывается меню с перечнем операций, описанных в табл. 1.

Таблица 1

Название	Перевод	Пример
Cut, Paste, Clear	Вырезать, Вставить, Удалить	
Format/Flip name	Формат/Развернуть имя	
Format/Hide name	Формат/Спрятать имя	
Format/Flip block	Формат/Развернуть блок	
Format/Rotate block	Формат/Повернуть блок	

На рис.3.1 показана простейшая схема моделирования. Она содержит три блока: Sine wave (генератор синусоиды) из группы **Sources** (генераторы входных сигналов), Scope (осциллограф) из группы **Sinks** (регистрация выходных сигналов) и Integrator (интегратор) из группы **Continuous** (непрерывные модели). Блоки можно соединить, протянув линию от одного блока к другому при нажатой левой кнопке мыши. Чтобы удалить связь, достаточно выделить ее мышью, а затем нажать клавишу DEL. Блоки можно удалять точно таким же способом.

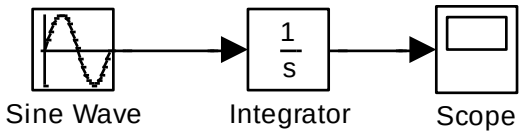


Рис.3.1

Для запуска модели следует нажать кнопку ▶ или выбрать пункт меню Simulation/Start. Если во время моделирования или после его окончания совершить двойной щелчок мышью по блоку Scope, откроется окно с графиком сигнала. Для разумного выбора масштаба графика нужно нажать мышкой на кнопку с изображением бинокля.

3.2. Генераторы входных сигналов и регистрация результатов

Для моделирования входных и управляющих воздействий (синусоидальных, ступенчатых, импульсных и других) используются генераторы сигналов. Все они сосредоточены в библиотеке (группе блоков) **Sources** (источники). Часть из них показана на рис.3.2.

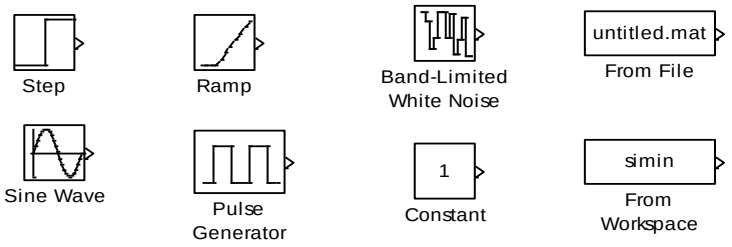


Рис. 3.2

Описание их параметров приведено в табл. 2.

Таблица 2

Название блока	Тип сигнала	Параметры	
Step	Ступенька	Step time Initial value Final value	Момент скачка Начальное значение Величина скачка
Sine wave	Гармонический сигнал $y = A \cdot \sin(\omega t + \varphi) + b$	Amplitude Bias Frequency Phase	Амплитуда A Смещение b Частота ω радиан/сек Сдвиг по фазе φ в радианах
Ramp	Линейно изменяющийся сигнал $y = \begin{cases} y_0 + k(t - t_0), t > t_0 \\ 0, t < t_0 \end{cases}$	Slope Start time Initial output	Коэффициент наклона k Начальное время t_0 Начальное значение y_0
Pulse generator	Последовательность прямоугольных импульсов	Amplitude Period Pulse width Phase delay	Амплитуда Период колебаний в секундах Ширина импульса Сдвиг по фазе в секундах
Band-limited white noise	Белый шум, ограниченный по полосе	Noise power Sample time Seed	Мощность Частота дискретизации Параметр инициализации
Constant	Константа	Constant value	Величина константы
From File	Загрузка из файла	File name Sample time	Имя файла Частота дискретизации
From Workspace	Загрузка из рабочего пространства MATLAB	Data Sample time	Имя переменной Частота дискретизации

В последних версиях SIMULINK в число генераторов входных сигналов включен блок Signalbuilder**,** позволяющий формировать кусочно-линейные сигналы произвольной формы.

Для представления результатов моделирования используются блоки, расположенные в группе **Sinks**. Наиболее важными из них являются блоки Scope, Graph (осциллографы), Display, To File и To Workspace. Их изображения приведены на рис. 3.3.

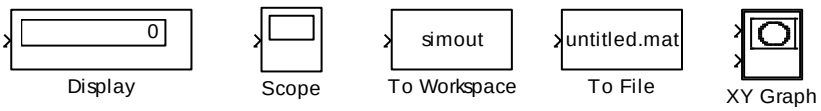


Рис. 3.3

Блок Scope предназначен для вывода графиков сигналов. Блок XY Graph необходим для изображения параметрических графиков, например, фазовых траекторий динамических систем.

Блок Display выводит в окне числовое значение сигнала. Блок To File предназначен для сохранения результатов моделирования в файле. Пользователь может выбрать имя файла и имя

переменной. После выполнение моделирования содержимое файла можно загрузить в рабочее пространство командой **load**.

Результаты моделирования можно передать в рабочее пространство MATLAB при помощи блока To Workspace. Пользователь должен ввести имя переменной (по умолчанию simout). Имеется три варианта представления результатов – массив (*array*), структура (*structure*), структура со временем (*structure with time*). В первом случае в рабочем пространстве MATLAB появляется массив значений переменной simout и соответствующий массив времени timeout. Во втором и третьем случаях результат возвращается в виде структуры и помещается в рабочее пространство MATLAB. Извлечение переменных из полей структуры производится с помощью команд типа simout.time и simout.signal.

3.3. Основные линейные и нелинейные блоки

На рис. 3.4 приведены обозначения основных непрерывных блоков из библиотеки **Continuous**.

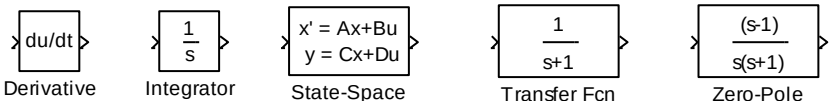


Рис. 3.4

Описание их назначения и смысл некоторых параметров указан в табл. 3

Таблица 1

Название блока	Тип блока	Параметры	
Derivative	Производная		нет
Integrator	Интегратор	External reset: • none • rising • falling • either • level Initial condition Limit output: • upper saturation limit • lower saturation limit	Внешний сброс: • отсутствует • по растущему фронту • по падающему фронту • по любому фронту • по уровню Начальные условия Ограничения по выходу: • верхний предел насыщения • нижний предел насыщения
State-space	Описание в пространстве состояний (ОПС)	A, B, C, D Initial conditions	Матрицы системы Вектор начальных условий
Transfer Fcn.	Передаточная функция	Numerator Denominator	Массив коэффициентов числителя Массив коэффициентов знаменателя
Zero-pole	Нуль-полюсное описание системы	Zeros Poles Gain	Массив нулей Массив полюсов Коэффициент усиления

На рис. 3.5 приведены аналогичные дискретные блоки (библиотека **Discrete**). Среди них есть два блока, не имеющих аналога среди непрерывных блоков. Это блок задержки Unit delay и

блок Discrete Filter – дискретный фильтр, который определяет рациональную функцию от оператора задержки $1/z$.

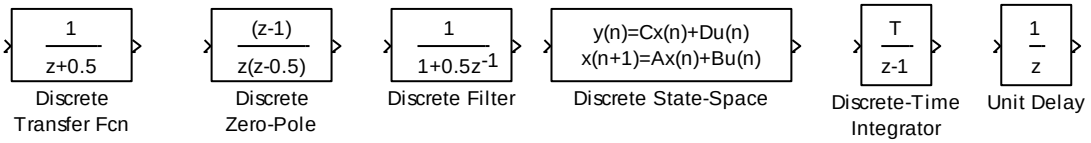


Рис. 3.5

На рис. 3.6 показаны изображения блоков для выполнения математических операций. Они входят в группу «математических функций» (**Math Operations**).

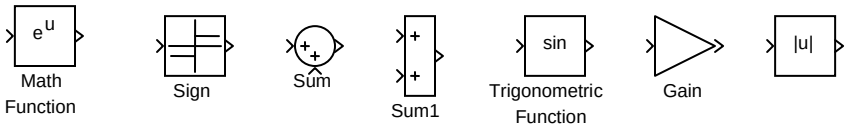


Рис. 3.6

Их описание и назначение параметров представлено в табл. 4.

Таблица 4

Название блока	Тип блока	Параметры	
Math function	Математическая функция	Function: exp, log, 10^u, log10, sqrt, …. Output signal type • Auto • Real • Complex	Вид функции Тип выходного сигнала • Не определен • Вещественный • Комплексный
Sign	Знак	нет	нет
Sum	Сумма	Icon shape: • Rectangular • Round List of signs	Изображение блока: • Прямоугольное • Круглое Знаки при входах
Trigonometric function	Тригонометрическая функция	Function: sin, cos, tan, asin, acos, sinh, cosh, …. Output signal type • Auto • Real • Complex	Вид функции Тип выходного сигнала • Не определен • Вещественный • Комплексный
Gain	Усилитель	Gain Multiplication: • Element-wise (K.*u) • Matrix (K*u) • Matrix (u*K)	Коэффициент усиления Способ умножения: • Поэлементное • Матричное (K*u) • Матричное (u*K)
Abs	Модуль	нет	нет

3.4. Пример моделирования в SIMULINK

Основной способ исследования систем в SIMULINK – составление структурной схемы на усилителях, сумматорах, интеграторах и других блоках, последующее моделирование и наблюдение результатов на имитаторах осциллографов и других регистрирующих приборов.

Рассмотрим процедуру моделирования на примере дифференциального уравнения

$$\ddot{y} + \sin y = 0, \quad \dot{y}(0) = 0,\ y(0) = 1/2.$$

Оно описывает свободное колебание математического маятника, который отклонили от положения равновесия на угол около 30^0 и отпустили без начальной скорости. Соответствующая схема моделирования приведена на рис.3.7, а. Она содержит два интегратора, тригонометрический блок для получения функции $\sin y$ и инвертирующий усилитель. Начальные условия на интеграторах устанавливаются при задании внутренних параметров блока.

Внесем в эту схему дополнительные блоки, позволяющие:

- Наблюдать фазовые траектории, т.е. графики зависимости $\dot{y}$ от y.
- Получать два графика $y(t)$ и $\dot{y}(t)$ на одном осциллографе.
- Передавать результаты моделирования в MATLAB.

Измененная схема показана на рис.3.7, б. Для наблюдения фазовых траекторий введен блок XY Graph. Для совмещения двух графиков на вход осциллографа Scope подан «сдвоенный» сигнал. Он получен с помощью блока «мультиплексор» (Mux) из группы блоков «маршрутизация сигналов» (**Signal Routing**).

Для передачи данных в MATLAB использован блок To Workspace с форматом вывода *structure with time.* Теперь после прогона модели графики $y(t)$ и $\dot{y}(t)$ можно будет построить из командного окна MATLAB, набрав:

```
>> plot(simout.time,simout.signals.values)
```

Такой способ построения полезен, если необходимо изменять вид графика, поскольку возможности осциллографа Scope значительно ниже, чем у команды **plot.**

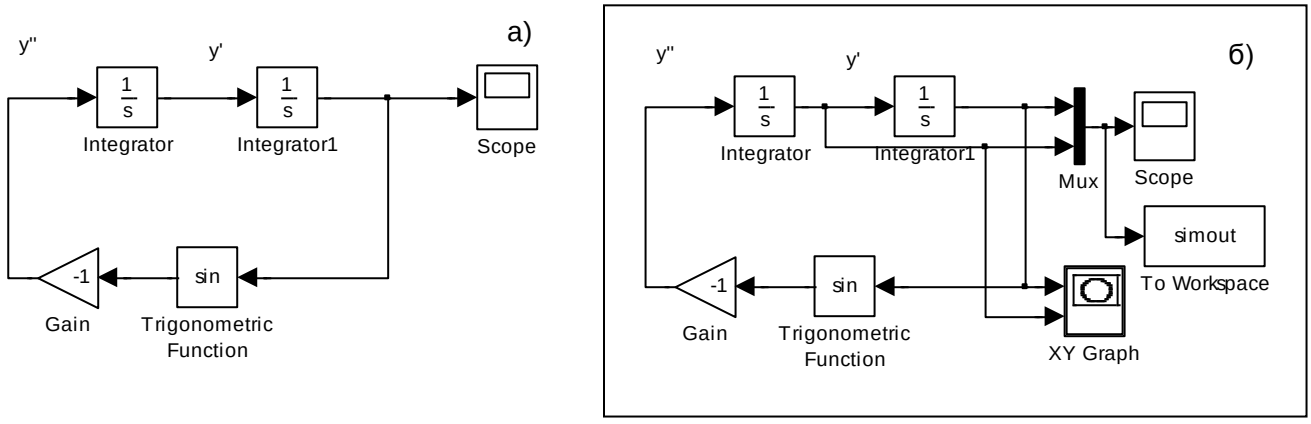


Рис. 3.7

В пункте меню «Simulation/Simulation parameters» можно задать ряд параметров моделирования. Часть из них расположены на панелях Solver (решатель) и Workspace I/O (интерфейс с рабочим пространством). На панели Solver в группе элементов управления Simulation Time (время моделирования) можно определить время начала (Start time) и конца моделирования (Stop time). В группе опций решателя (Solver) можно выбрать используемый алгоритм решения дифференциальных уравнений с фиксированным шагом (Fixed-step) или с изменяемым шагом (Variable step). Можно также задать минимальный и максимальный размер шага, а также относительную и абсолютную точность интегрирования. На панели Workspace I/O можно

определить имена переменных, в которых хранятся результаты моделирования и которые используются как источники сигналов.

Чтобы проиллюстрировать их применение, усложним нашу модель, введя в нее управляющий сигнал $u = e^{-t}$:

$$\ddot{y} + \sin y = u, \quad y(0) = 1, \quad \dot{y}(0) = 1.$$

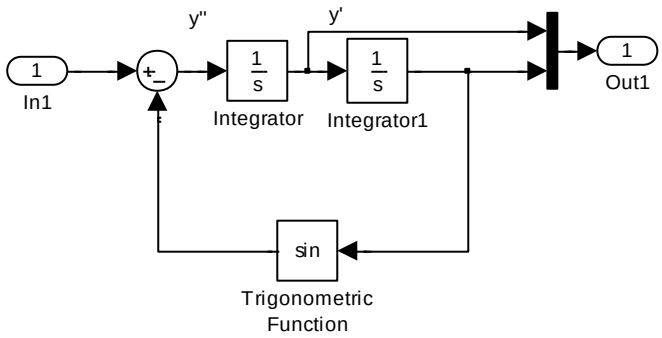
Теперь это уравнение описывает вынужденное движение маятника под действием управляющего воздействия $u(t)$. Управляющий сигнал можно получить либо непосредственно в SIMULINK, либо сформировать его в рабочем пространстве MATLAB и оттуда передать в SIMULINK. Воспользуемся второй возможностью. Для этого потребуется изменить схему, изображенную на рис 3.7, добавив блоки «Вход» (In из группы **Sources**) и «Выход» (Out из группы **Sinks**). Результат показан на рис 3.8, слева.

Зададим время моделирования Stop time=20. На панели Workspace I/O в графе Input запишем [t, u], в графе Initial State запишем xInitial, в графе Time – tout, в графе Output – yout, и установим соответствующие чек-боксы.

В рабочем пространстве MATLAB подготовим управляющий сигнал $u = e^{-t}$ и установим начальные условия $y(0) = 1$, $y(0) = 1$:

```
>> xInitial=[1; 1];        % столбец начальных условий
>> t=linspace(0,20)';      % время моделирования
>> u=exp(-t);              % управление u
```

Запустим процесс моделирования в SIMULINK. По окончании моделирования в рабочем пространстве появятся переменные tout и yout. Набрав plot(tout,yout,t,u,'r:'**),** можно просмотреть три графика – $y(t)$, $\dot{y}(t)$, $u(t)$ (рис 3.8, справа).



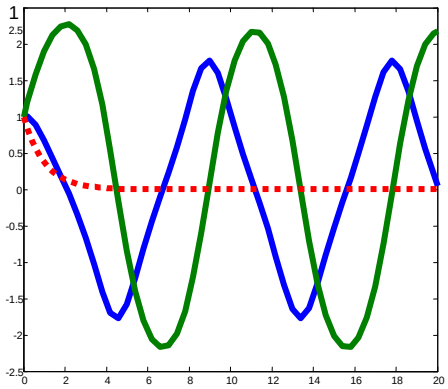


Рис. 3.8

Задачи и упражнения

1. Дана система дифференциальных уравнений с единичными начальными условиями

$$\dot{X} = AX, \quad X(0) = [1 \ \ldots \ 1]^T.$$

Для двух вариантов матрицы A

а) $A=-\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 6\\ 3 & 6 & 9\end{bmatrix}$, б) $A=-\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$

получить в MATLAB и SIMULINK графики выходных сигналов. Найти аналитическое решение с помощью команды **dsolve**.

Ответ.

а) $x_1 = x_2 = x_3 = e^{-14t}$

```
>> A=-[1 2 3;2 4 6;3 6 9]; s=ss(A,eye(3),eye(3),0);initial(s,[1;1;1])
```

б) $x_1 = x_2 = x_3 = x_4 = e^{-t}$

```
>> s=ss(-ones(4),eye(4),eye(4),0);initial(s,[1;1;1;1])
```

2. Дана функция $y = a_1 e^{-a_2 t} + \sin a_3 t$. Параметры a_1, a_2, a_3 – это число букв в Вашей фамилии, имени, отчестве. Требуется найти дифференциальное уравнение, решением которого она является, рассчитать начальные условия и нарисовать схему для моделирования в СИМУЛИНК.

3. Составить схему и выполнить моделирование в SIMULINK следующих линейных дифференциальных уравнений с переменными коэффициентами:

а) уравнения Матье $\ddot{y} + (1 - 0{,}5\cos t)\dot{y} + 4y = 0$;

б) уравнения Бесселя $t^2 \ddot{y} + t\dot{y} + (t^2 - n^2)y = 0$;

в) уравнения Лежандра $(1 - t^2)\ddot{y} - 2t\dot{y} + n(n+1)y = 0$;

г) уравнения Чебышева $(1 - t^2)\ddot{y} - t\dot{y} + n^2 y = 0$;

д) уравнения Лагерра $t\ddot{y} + (1 - t)\dot{y} + ny = 0$.

4. Свободные колебания маятника описываются дифференциальным уравнением

$$\ddot{y} + \frac{g}{l}\sin y = 0,$$

где y – угол отклонения, g – ускорение свободного падения, l – длина маятника.

а). Требуется исследовать в SIMULINK колебания маятника, когда его движение ограничивается вертикальной стенкой, расположенной на расстоянии b от положения равновесия (рис.3.9). Считать, что при ударе о стенку скорость маятника изменяет знак на противоположный. Для разных вариантов начальных условий получить графики $y(t)$, $\dot{y}(t)$, $\dot{y} = f(y)$.

б). Сделать то же для симметричного двустороннего ограничения (маятник помещают в стакан радиуса b)

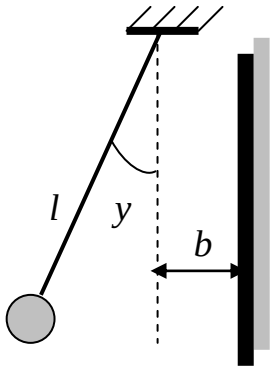


Рис.3.9

РЕШЕНИЕ АЛГЕБРАИЧЕСКИХ ЗАДАЧ

В предыдущих разделах были изложены начальные сведения о MATLAB и SIMULINK, необходимые для освоения этих пакетов. В этом разделе описываются графические средства MATLAB и его возможности по решению алгебраических задач и поиску экстремумов.

Графические средства MATLAB

Информация о построении графиков с помощью команды **plot** была приведена в разд. 1.3. Однако графические возможности MATLAB отнюдь не исчерпываются этой командой. В данном разделе описываются средства, предназначенные для управления графическими окнами и построения различных видов графиков.

Управление графическим экраном

В MATLAB существует ряд команд для управления графическими окнами, их перечень приводится ниже:

grid **axis** **hold** **figure** **shg** **clf** **subplot**

Кратко охарактеризуем каждую из них. О назначении команды **grid** (сетка) и ее вариантах **grid on** и **grid off** было сказано ранее.

Изменить масштабы осей текущего графика можно с помощью команды **axis**. Наиболее распространенные форматы ее вызова – это axis square, axis normal, axis auto и axis equal. Те же действия можно произвести, набрав axis('square'), axis('normal'), axis('auto') и axis('equal'), соответственно. Команда axis('square') делает область графического изображения квадратной. При этом наклон линии plot(x,x) должен быть 45 градусов, если она не скашивается неправильной формой экрана. Круг, задаваемый командой plot(sin(t),cos(t)) должен выглядеть как круг, а не как овал. Команде axis('normal') восстанавливает масштаб обратно в нормальное положение. Есть также возможность задать размеры осей явно, например, axis([10 10 -1 2]). При этом область вывода примет вид $10 \le x \le 10,\ -1 \le y \le 2,$ а plot (X,Y,'LineWidth',2) сделает толщину линии равной двум.

На один график можно поместить несколько кривых. Это можно сделать тремя способами, которые мы поясним на примере команды **plot**.

Первый способ заключается в том, что в команде **plot** указывают несколько пар аргументов, например, набрав plot (x1, y1, x2, y2, x3, y3), получим три графика $y_1 = f_1(x_1)$, $y_2 = f_2(x_2)$, $y_3 = f_3(x_3)$.

Второй способ изображения нескольких кривых на одном графике применяется для изображения нескольких функций одного и того же аргумента. Если x – вектор, Y – матрица, то по команде plot(x, Y) будет построен набор графиков, отражающих зависимость столбцов матрицы Y от аргумента x (длина вектора x должна совпадать с длиной столбцов матрицы Y).

Пример. Нарисуем, используя этот прием, синусоиду, косинусоиду и экспоненту на одном графике:

```
>>t=0:0.1:10; x=[sin(t);cos(t); exp(-t)]; plot(t,x)
```

Результат представлен на рис. 4.1.

Третий способ основан на использовании команды **hold,** которая удерживает текущий график на экране. Последующие команды **plot** будут добавлять к этому графику новые, сохраняя ранее вычерченные кривые. Режим удержания графиков будет действовать до тех пор, пока команда **hold** не будет введена повторно. Другая возможность включения и выключения этого режима – использовать команды **hold on** и **hold off**. Для рассматриваемого примера нужно набрать:

```
>> hold on, plot(t,sin(t)),  plot(t,cos(t)), plot(t,exp(t)), hold off
```

Результат работы двух последних способов идентичен (за исключением цветов), он показан на рис. 4.1 слева.

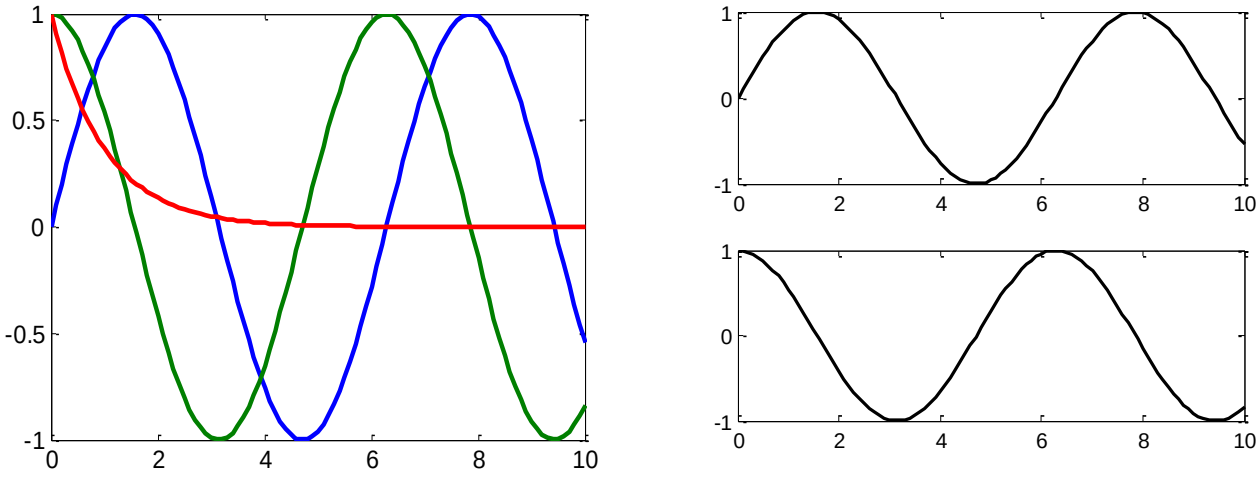


Рис. 4.1

Команда **figure** позволяет открывать нужное количество графических окон. Каждый график выводится в свое графическое окно, их количество может быть произвольным. Без указания аргумента **figure** открывает новое графическое окно, присваивая ему очередной порядковый номер. Команда figure(5) откроет графическое окно, присвоив ему номер 5, после этого в него командой **plot** или другой графической командой может быть помещен нужный график. Это дает возможность одновременно сохранять несколько графических окон с различными графиками. Вызвать нужное окно для просмотра можно с помощью клавиш Alt-Tab или мыши, для вызова последнего графика существует команда **shg** (от *show graphic*). Для очистки графического окна служит команда **clf** (от *clear figure*).

Заметим, что очистка командного окна производится командой **clc** (от *clear command*), а очистка рабочей области, т.е. удаление всех переменных – командой **clear.** Для удаления одной переменной *x* надо набрать clear(x), а команда clear x y z удалит из рабочей области три переменные x, y, z.

Каждое графическое окно можно разбить на таблицу из нескольких прямоугольных частей и поместить в них отдельные графики. Это делается с помощью команды **subplot.** Команда subplot(mnp) разбивает графическое окно на *mxn* частей (*m* строк и *n* столбцов) и помещает следующий график в ячейку номер *p*. Например, команды

```
>>subplot(211),  plot(t,sin(t)),  subplot(212),  plot(t,cos(t))
```

разбивают экран на две части, изображая график синуса в верхней половине, а график косинуса – в нижней половине (рис. 4.1, справа). Команда subplot(111) или просто subplot возвращает к положению, когда одно окно занимает целый экран.

Большая часть сказанного справедлива не только для команды **plot**, но и для других графических команд, описываемых ниже.

Двумерная графика

В состав MATLAB входит целый ряд команд**,** которые дают различные возможности по изображению двумерных графиков. Их перечень приведен в табл. 4.1.

Таблица 4.1

loglog	**polar**	**stairs**	**area**	**pcolor**	**line**	**pie**	**plotyy**
semilogx	**comet**	**bar**	**fill**	**colormap**	**ribbon**	**pie3**	**strips**
semilogy	**stem**	**barh**	**patch**	**rectangle**	**scatter**	**errorbar**	**imagesc**

Дадим их краткую характеристику.

Применение команд **loglog, semilogx, semilogy, polar** аналогично применению **plot.** Они дают возможность изображать данные графически на различных видах "миллиметровой бумаги"**,** т.е. в различных системах координат – логарифмической, полулогарифмической и полярной. Команда **loglog** строит график, используя логарифмические шкалы по обеим осям. **Semilogx** делает график, используя полулогарифмический масштаб, при этом ось X – логарифмическая, а ось Y – линейная. **Semilogy** строит график, используя полулогарифмический масштаб, при котором ось Y – логарифмическая, а ось X – линейная.

Пример. В телевизионной игре "*Кто хочет стать миллионером*" всего 14 туров, выигрыши в них составляют 100, 200, 300, 500, 1000, 2000, 4000, 8000, 16000, 32000, 64000, 125000, 500000, 1000000 рублей соответственно[1]. Требуется построить график выигрыша в зависимости от номера тура, используя команды **plot, subplot, semilogy, title, xlabel, ylabel, bar**.

Вводим исходные данные и строим графики в обычном и полулогарифмическом масштабах.

```
>> y=[100 200 300 500 1000 2000 4000 8000 16000 32000 64000 125000 500000 1000000];
>> subplot(121); plot(y),grid,  title('Rate of game'),  xlabel('Tours');ylabel('Rubles');
>> subplot(122); semilogy(y), grid, title('Rate of game'), xlabel('Tours');ylabel('Rubles');
```

Графики приведены на рис.4.2. Видим, что в полулогарифмическом масштабе зависимость выигрыша от номера тура близка к линейной.

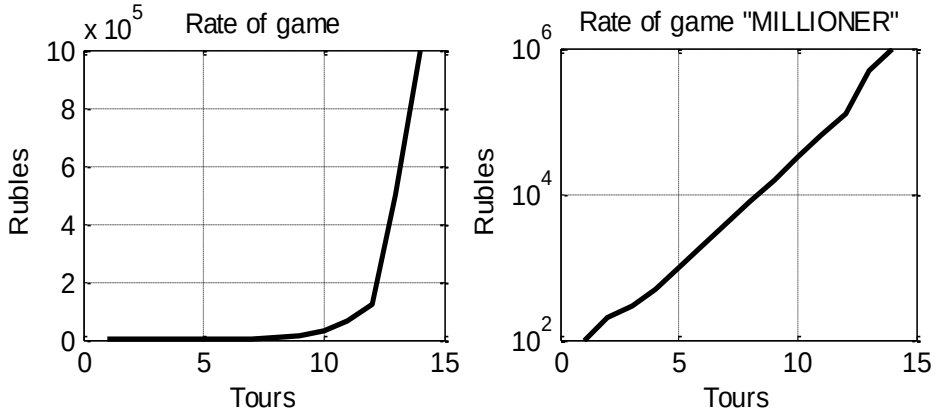
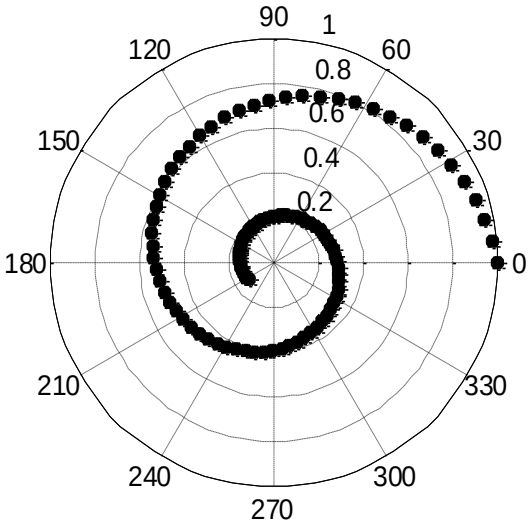


Рис.4.2

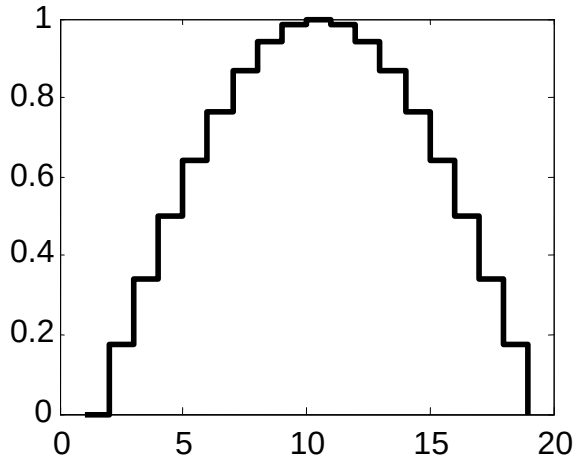
[1] Данные относятся к середине 2005 года

Для построения графиков в полярной системе координат используется команда **polar**. Она имеет те же модификации, что и команда **plot**, допуская использование как векторных, так и матричных аргументов. По команде polar(φ, r) строится график $r = f(\varphi)$, показывающий зависимость длины радиус-вектора r от угловой координаты φ, которая измеряется в радианах. Например, полярное уравнение логарифмической спирали имеет вид $r = be^{a\varphi}$. Ее построение в полярной системе координат для b=1, a= – 0,2 показано на рис. 4.3.



t = 0:.1:10; r = exp(-.2*t); polar(t,r,'*')

Рис. 4.3

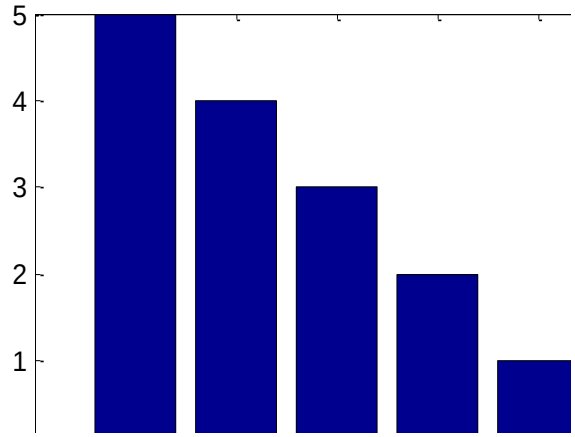


t=0:10:180; x=sind(t); stairs(x)

Рис. 4.4

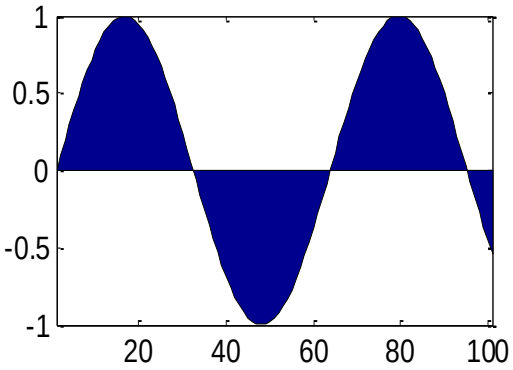
Команда **comet** удобна для вычерчивания траекторий движения, в ней «хвост» графика выделяется другим цветом, что дает возможность наблюдать динамику его построения. График, построенный командой **stem,** имеет вид вертикальных отрезков, заканчивающихся кружочками, наподобие стебельков цветов.

Команда **stairs** рисует график вектора x в виде ступенек (лестницы), как это показано на рис. 4.4 для полуволны синусоиды. Еще один способ отображения информации – использование столбцовых графиков, которые строит команда **bar.** Она дает изображение элементов вектора x в виде расположенных рядом столбцов, высота которых определяется значением элементов (рис. 4.5). Команда **barh** отличается горизонтальным расположением столбцов.



x=5:-1:0; bar(x)

Рис. 4.5



t=0:.1:10; x=sin(t); area(x)

Рис. 4.6

Команды **area, fill** и **patch** позволяют закрашивать области на графике. Пример действия первой из них приведен на рис. 4.6, где закрашена область между графиком синусоиды и осью абсцисс.

Цветовая палитра графика может уточняться с помощью функции **colormap.** В качестве аргумента у нее возможны опции white, gray, black, cool, hot, 'default'. Разные цвета можно задавать трехэлементной численной rgb-кодировкой, где первый элемент указывает интенсивность красного цвета, второй – зеленого, третий – синего. Например, [0 0 0] черный цвет, [1 1 1] – белый, [.5 .5 .5] – серый, [1 0 0] – красный, [127/255 1 212/255] – аквамариновый. Можно использовать также буквенные обозначения 'r','g','b','c','m','y', 'w', или 'k'. Функция **colormap** может использоваться и с другими командами графики, такими как **errorbar**, **imagesc, pcolor.**

Команда **rectangle** строит графический примитив в виде прямоугольника с прямыми или скругленными углами, а команда **line** проводит отрезок прямой линии, соединяющей две точки. Команда **ribbon** рисует двумерный график в виде ленты в трехмерном пространстве. Например, набрав x=0:.2:10; y=sin(x); ribbon(y) получим синусоидальную поверхность (рис. 4.7).

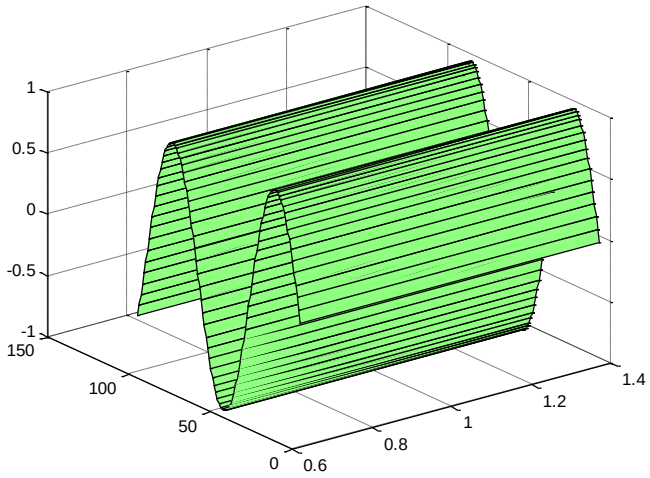


Рис. 4.7

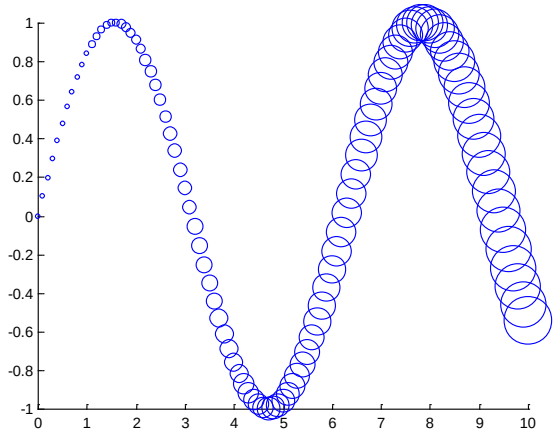
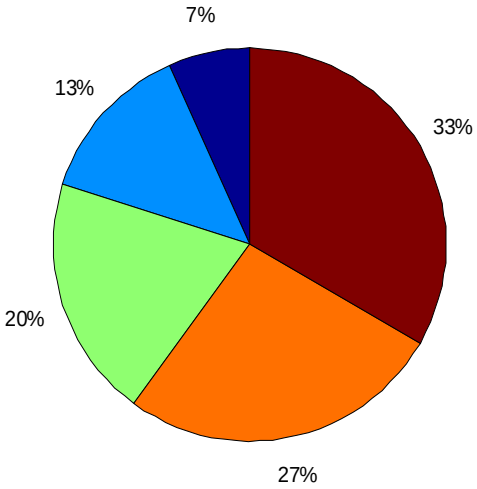


Рис. 4.8

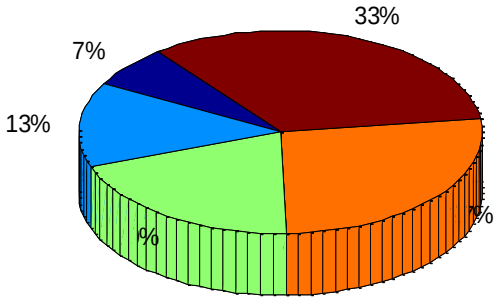
Команда **scatter** строит фигурный или «рассыпчатый» график (от scatter – рассыпать). Например, набрав x=0:.2:10; y=sin(x); scatter(x,y, 10*(x.^2+1)) получим синусоиду, нарисованную с помощью кружков возрастающего размера (рис. 4.8). В частности, таким образом можно изобразить динамическую трубку точности.

Примеры действия команд **pie** и **pie3,** которые строят круговые диаграммы, показаны на рис. 4.9 и рис. 4.10.



pie(1:5)

Рис. 4.9



pie3(1:5)

Рис. 4.10

Команда **errorbar** строит график кривой с учетом погрешности. В качестве примера на рис. 4.11 показан результат выполнения последовательности команд:

```
t=1:.05:20;y=exp(.2*t).*sin(t);x=exp(.2*t).*cos(t); errorbar(x,y,(t.^1.5)*.1)
```

Команда **plotyy** позволяет нарисовать две разномасштабные кривые на одном графике. Набрав plotyy(X1,Y1,X2,Y2) получим график Y1 от X1 с размеченной *y*-осью слева и график Y2 от X2 с размеченной *y*-осью справа. На рис. 4.12 показан результат выполнения команд

```
t=0:.1:10;   y=sin(t);   plotyy(t,y,t,t.^2),  grid
```

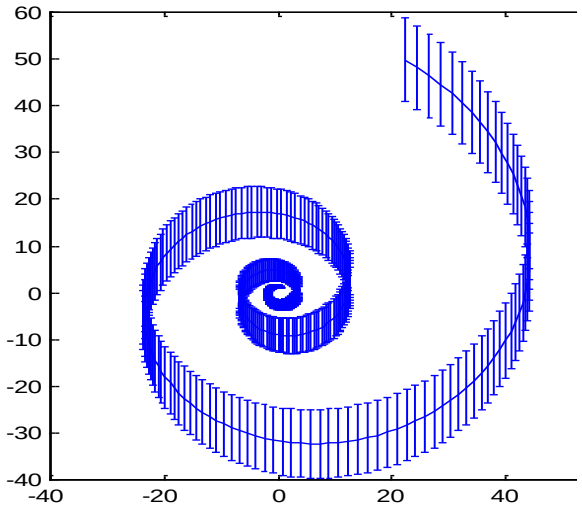


Рис. 4.11

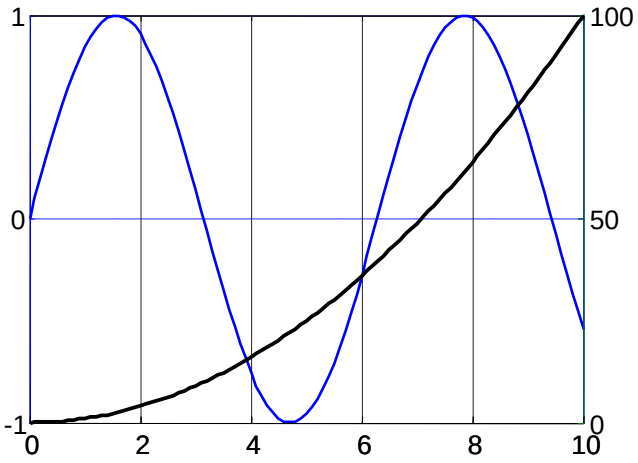
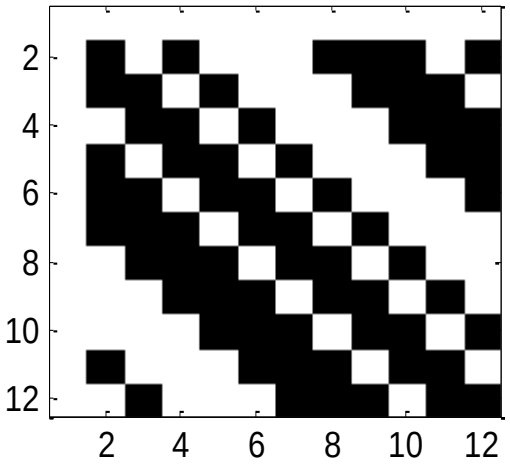


Рис. 4.12

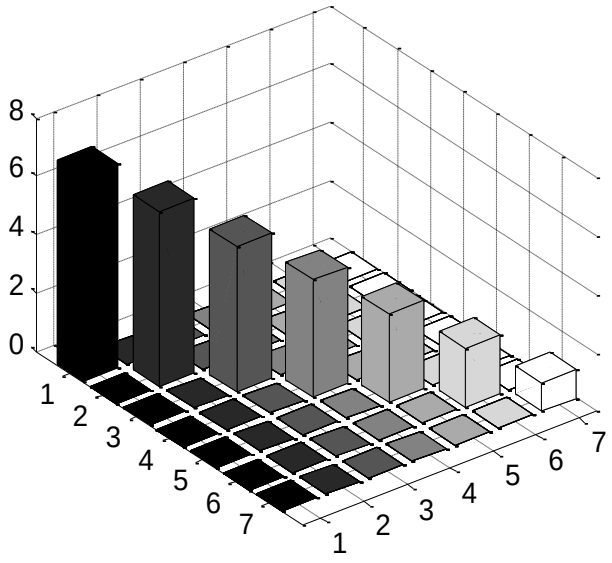
Функция **strips** (от *strip* – полоска) используется для просмотра «длинных» графиков. Она разрезает такой график на куски равной длины и размещает их друг под другом.

Imagesc служит для визуализации матриц (цвет элементов зависит от их величины). На рис. 4.7 приведен пример изображения с ее помощью матрицы Адамара двенадцатого порядка (элементы +1 выделены белым цветом, элементы -1 – черным).

Из других команд для визуализации матриц отметим **hintonw, spy, printmat, plotmatrix.**



imagesc(hadamard(12))
Рис. 4.13



bar3(diag(7:-1:1))
Рис. 4.14

Трехмерная графика

Выше речь шла о графиках на плоскости. MATLAB позволяет также изображать линии, поверхности и фигуры в трехмерном пространстве, вращать полученное изображение и смотреть на него с различных ракурсов. Соответствующие команды приведены в табл.4.2.

Таблица 4.2

bar3	**plot3**	**mesh**	**surf**	**sphere**	**cylinder**
bar3h	**contour**	**meshgrid**	**fill3**	**ellipsoid**	**logo**

Команда **bar3** предназначена для трехмерного изображения матрицы в виде столбиков, высота которых пропорциональна величине ее элементов. В качестве простого примера на рис. 4.14 приведено изображение диагональной матрицы. Команда **bar3h** отличается горизонтальным расположением столбиков.

Для изображения линий и кривых в трехмерном пространстве используется команда **plot3.** Пример построения с ее помощью графика винтовой линии был приведен в разд.1.3.

Для изображения поверхностей в трехмерном пространстве служат команды **contour, mesh, meshgrid** и **surf.** Команда contour(Z) строит горизонтальные сечения поверхности, задаваемой матрицей Z, считая, что ее элементы указывают высоту над плоскостью. Число контурных линий может указываться с помощью второго аргумента. Дополнительными возможностями обладают команды **contourf** и **contour3.**

Команда mesh(Z) создает график трехмерной перспективы элементов матрицы Z. Она может использоваться для визуализации больших матриц и графического изображения функций двух переменных.

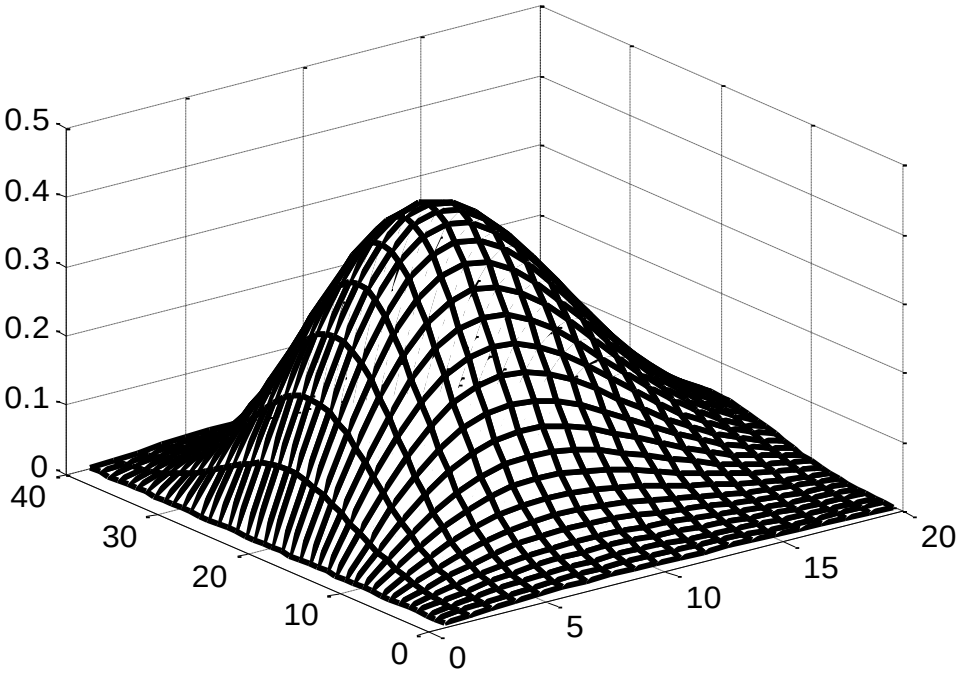


Рис. 4.15

В сочетании с **meshgrid** команда **mesh** позволяет строить графики поверхностей в трехмерном пространстве. При этом **meshgrid** формирует координатную сетку в заданной области плоскости (*x, y*). В качестве примера на рис. 4.15 показан график поверхности, задаваемой уравнением $z = xe^{-x^2-y^2}$ в области $0 < x < 1{,}9; \quad -2 < y < 1{,}9$, построенный с использованием указанных команд:

```
[X,Y] = meshgrid(0:.1:1.9, -2:.1:1.9); Z =X.*exp(-X.^2-Y.^2);   mesh(Z)
```

Команды **surf** и **fill3** служат для изображения раскрашенных поверхностей. Кроме того имеется ряд команд для изображения конкретных трехмерных фигур и поверхностей. К ним относятся уже упоминавшиеся в разд. 1.3 команды **cylinder, sphere, ellipsoid,** а также **logo, membrane, peaks** и некоторые другие. Специальные графические команды имеются в тулбоксе SYMBOLIC, к ним относятся **ezplot, ezcontour, ezmesh, ezmeshc, ezplot3, ezpolar, ezsurf, ezsurfc.**

В графической системе MATLAB реализован довольно большой спектр возможностей полигональной графики. В частности, есть возможность использовать как диффузные, так и точечные источники освещения, различные алгоритмы закраски и т.д. Это позволяет строить графики сложных поверхностей. Примером может служить логотип MATLAB, изображенный на рис. 4.16. Он строится при помощи команды **logo**. Изображенная зависимость является графиком первой собственной функции L-образной мембраны с защемленным внутренним краем, которая рассчитывается с помощью команды **membrane.** Чтобы посмотреть графики других собственных функций нужно заменить первый параметр вызова **membrane** на 2, 3, 4 и т.д. На рис. 4.17 приведен график третьей собственной функции мембраны. При расчете этих поверхностей используется функции Бесселя.

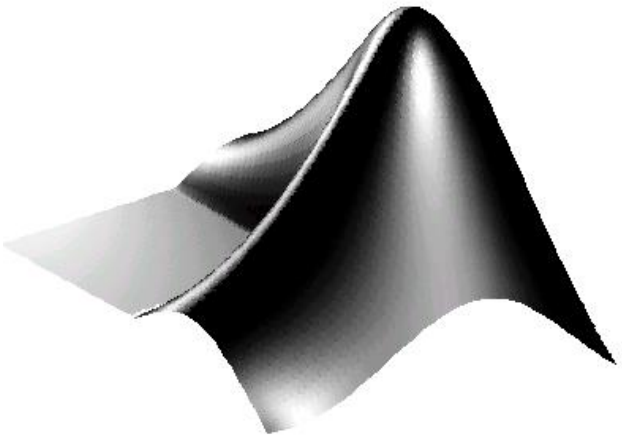


Рис. 4.16

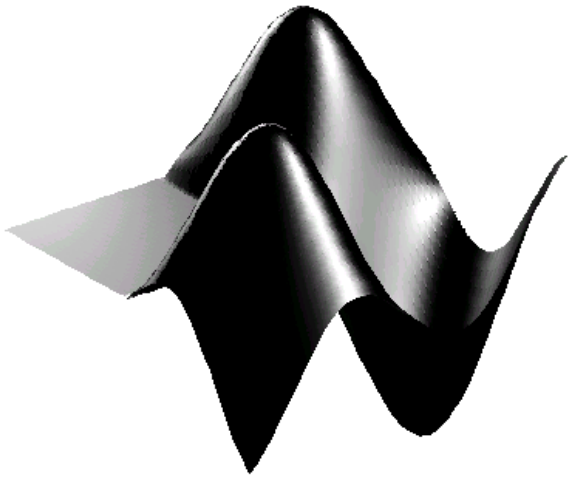


Рис. 4.17

Все графики, построенные в MATLAB, можно переводить в стандартные графические форматы (*bmp, metafile*) и переносить в другие приложения, например, в WORD. В свою очередь, в MATLAB есть средства для импорта графических файлов различных форматов и их последующей обработки. Соответствующие команды сосредоточены в тулбоксе IMAGE PROCESSING.

Решение алгебраических уравнений и поиск экстремумов функций

MATLAB в первую очередь ориентирован на решение задач линейной алгебры, однако в нем есть и средства для решения нелинейных алгебраических задач, таких как отыскание корней нелинейных уравнений, поиск экстремумов функций одной или нескольких переменных, решение задач аппроксимации и интерполяции. Опишем некоторые из них.

Решение нелинейных уравнений

Одна из распространенных алгебраических задач – поиск корней уравнения $f(x)=0$. Для численного отыскания корней проще всего построить график функции $y=f(x)$ и найти точки его пересечения с осью абсцисс (в MATLAB это можно сделать с помощью команд **plot, fplot, ezplot**). Аналогично можно поступить и в случае системы двух уравнений с двумя неизвестными $f(x, y)=0$, $g(x, y)=0$, построив на плоскости (x, y) графики этих функций и найдя точки их пересечения друг с другом. Сложнее обстоит дело с поиском комплексных корней, здесь требуется привлечение специальных методов.

Ранее уже было описано применение команды **roots** для нахождения корней полиномов. Для поиска корней более сложных уравнений с одной переменной, например, включающих логарифмические, тригонометрические, экспоненциальные зависимости, применяют команду **fzero**. Ее входными аргументами служат имя функции, вычисляющей левую часть уравнения $f(x)=0$, и начальное приближение x_0.

Пример. Возьмем полином $f(x)=x^2+3x+2$ с корнями -1 и -2. В MATLAB их можно найти двумя путями: посредством команды **roots** и с помощью команды **fzero**. В последнем случае нужно сформировать вспомогательную функцию **ff** в виде отдельного m-файла или же создать временную функцию (на период данного сеанса MATLAB) при помощи команды **inline**. Команда **fzero**, использующая численные методы, возвращает один из корней полинома в зависимости от начального приближения.

```
>> p=[1 3 2];        >>f(x)= inline('x^2+3*x+2';    function y=ff(x)
>> roots(p)          >> fzero f(x), -3))            y=x^2+3*x+2;
ans =   -2
        -1
```

```
ans =   -2.0000          >> fzero(@ff,0)
>> fzero(f(x),0)         ans =    -1
ans =    -1
```

В команде **fzero** и других командах, рассматриваемых в этом параграфе, имеется возможность задавать структуру опций решателя. Ее можно описать вручную, но удобнее использовать команды **optimget** и **optimset**. Для получения полного списка опций и значений по умолчанию применяют формат optimset('имя решателя'), например, opts=optimset('fzero'). Для изменения опций используется команда opts=optimset('имя параметра',значение_параметра).

Уменьшим точность вычислений в предыдущем примере, изменяя опции решателя:

```
>> o=optimset('fzero');          % опции команды fzero по умолчанию
>> optimget(o,'TolX')            % точность вычислений по умолчанию
ans =  2.2204e-016
>> fzero(@ff,0)                  % оптимизация с точностью 2.22e-16
ans =    -1
>> o=optimset(o,'TolX',1e-1);    % понизили точность до 0.1
>> fzero(@ff,0,o)                % оптимизация с точностью 0.1
ans =   -1.0240
```

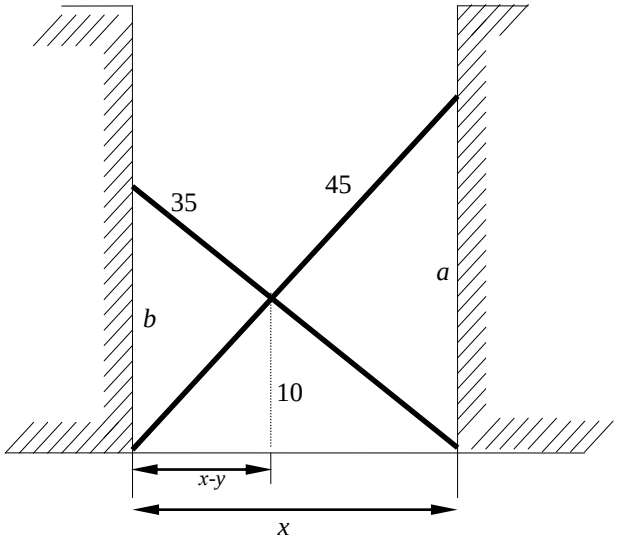
Теперь вместо точного ответа $x_1 = -1$ получено приближенное значение $x_1 = -1{,}024$.

Другая возможность решения нелинейных алгебраических уравнений (и систем таких уравнений) связана с использованием команды **solve** тулбокса SYMBOLIC. Проиллюстрируем ее на простом примере.

Пример. Требуется найти корни трансцендентного уравнения $x + e^{-x} = 0$.

Попытка его графического решения к успеху не приводит – график функции $y = x + e^{-x}$ ни разу не пересекает ось абсцисс. Это означает, что вещественных корней нет. Для поиска комплексных корней воспользуемся функцией **solve.** Набирая solve('x+exp(-x)'), получим ответ ans=lambert(-1).

Таким образом, решение нашего уравнения выражается через функцию Ламберта (это функция, обратная по отношению к функции xe^x). Чтобы вычислить ее значение, можно набрать double(ans) (результатом будет ans=-0.3181+1.3372i), либо использовать команду x=lambertw(a). В частности при $a = -1$ получим $x = -0{,}318+1{,}337i$, а при $a = -1{,}4$ получим $x = -0{,}0828+1{,}517i$ (это решение уравнения $x + 1{,}4e^{-x} = 0$).



$$x^2+a^2=45^2,$$
$$x^2+b^2=35^2,$$
$$\frac{y}{10}=\frac{x}{b},$$
$$\frac{x-y}{10}=\frac{x}{a}.$$

Рис. 4.18

Для численного решения систем нелинейных алгебраических уравнений с несколькими неизвестными вида F(X)=0, где F – вектор-функция, X – вектор переменных, предназначена команда **fsolve**. Ее входными аргументами служат имя функции, оформленной в виде отдельного *m*-файла, в которой описаны левые части нелинейных уравнений, и вектор начальных значений переменных. Проиллюстрируем применение этой команды на примере.

Задача о двух лестницах. В узком переулке крест-накрест стоят две лестницы длиной 45 и 35 футов (рис. 4.18). Расстояние от земли до точки их пересечения составляет 10 футов. Определить ширину переулка.

Решение. Обозначим расстояние от верхних точек лестниц до земли через *a* и *b*, ширину переулка через *x*, а расстояние от точки пересечения до левой стенки через *y*. Тогда можем записать систему 4 уравнений с 4 неизвестными, приведенную на рис. 4.18. В MATLAB ее можно решить двумя способами.

Способ 1 (символьный). Найдем аналитическое решение с помощью команды **solve**:

```
>>syms a b x y;s=solve('x^2 + a^2-45^2, x^2 + b^2-35^2, y/10-.x/b, (x-y)/10-x/a')
s =    a: [12x1 sym]    b: [12x1 sym]    x: [12x1 sym]    y: [12x1 sym]
```

В результате получаем структуру s содержащую 12 аналитических вариантов решения. Однако попытка вывода их на дисплей, обращаясь к полям >> s.x, s.y, показывает, что решение содержит многоэтажные корни, громоздко и неудобно для обозрения.

Его можно представить в численном виде, набрав:

```
>>[double(s.x) double(s.y) double(s.a) double(s.b)]
```

На дисплей будут выведены все 12 вариантов решения, но среди них только одно имеет физический смысл:

```
>>ans=31.8175   21.8189   31.8222   14.5825
```

Способ 2 (численный). Решим систему в численном виде с помощью команды **fsolve**. Предварительно составим вспомогательную функцию **ladder**, содержащую информацию об уравнениях.

```
function fn=ladder(p)
x=p(1); y=p(2); a=p(3); b=p(4);
f(1)=x^2 + a^2 - 45^2; f(2)=x^2 + b^2 - 35^2;
f(3)=y/10 - x/b; f(4)=(x-y)/10 - x/a;
fn=f(:);
```

Решаем систему, задав вектор начальных условий [10; 10; 20; 20]

```
>>x=fsolve('ladders',[10; 10; 20; 20])
```

Получаем ответ: x=31.8175; y=21.8189; a=31.8222; b=14.5825.

Видим, что оба способа дают один и тот же результат.

Поиск экстремумов

Распространенная группа задач связана с поиском экстремумов функций одного или нескольких аргументов. Широко известны аналитические методы решения конечномерных экстремальных задач – метод Ферма (рецепт "взять производную и приравнять нулю") и метод множителей Лагранжа.

Первый из них применяется для решения задач безусловной оптимизации, когда требуется найти экстремумы функции $y = f(x_1,\ldots,x_n)$. Необходимые условия экстремума имеют вид

$$\partial f / \partial x_1 = 0 \ , \ ..., \ \partial f / \partial x_n = 0,$$

их можно записать в компактном виде grad y = 0. После отыскания корней этой системы алгебраических уравнений проверяются достаточные условия экстремума.

Метод Лагранжа применяется для аналитического решения задач условной оптимизации, когда требуется найти экстремумы функции $y = f(x_1,\ldots,x_n)$ при наличии ограничений $g(x_1,\ldots,x_n) = 0.$ В этом случае строится составной критерий $L = f + \lambda g$ и его частные производные приравнивают нулю.

Для численного решения тех же задач применяют методы половинного деления и золотого сечения (одномерный поиск), методы градиента и наискорейшего спуска, методы целочисленного, линейного и нелинейного программирования. В пакете MATLAB эти методы реализованы в командах **fminsearch, fminunc, fmincon, fminbnd.** Для задач линейного программирования предназначена команда **linprog**.

Минимум одномерной функции отыскивают с помощью команды **fminsearch**. Для поиска максимума функции $f(x)$ достаточно найти минимум функции $-f(x)$, поэтому специальной функции для поиска максимумов в MATLAB не существует.

Пример. Найдем точку минимума полинома $f(x) = x^2 + 3x + 2$ из примера 1 и убедимся в правильности результата, приравнивая производную нулю. Производную берем с помощью функции **polyder**, ее корень находим командой **fzero**.

```
fminsearch(@ff,0)    polyder(p)          fzero(inline('2*x+3'),0)    ff(ans)
ans =   -1.5000      ans =    2     3     ans =   -1.5000             ans =   -0.25
```

В последнем столбце приведен расчет значения функции $f(x)$ в точке минимума.

Для безусловной минимизации функций от нескольких переменных используют функцию **fminunc** (от слова *unconstrained* – без ограничений). Ее первый входной аргумент – имя минимизируемой функции, второй – координаты начальной точки для поиска.

Пример. Найдем минимум функции двух переменных $y = x_1^2 + x_2^2$, который, очевидно, достигается в точке $x_1 = x_2 = 0$. Предварительно нужно в отдельном *m*-файле описать минимизируемую функцию, назовем ее **fff**:

```
function y=fff(x)      >> x=fminunc(@fff,[3  3])
y=x(1)^2+x(2)^2;       Warning: Gradient must be provided for trust-region method;
                         using line-search method instead.
                       x =  1.0e-008 *   [-0.9290   -0.9290]
```

Ответ получен с достаточно хорошей точностью. В то же время MATLAB рекомендует вместо линейного поиска применить градиентный метод. Для этого требуется, во-первых, переписать функцию **fff** так, чтобы она возвращала не только переменную *y*, но и ее частные производные (вектор градиента), и, во-вторых, подключить градиентный метод при помощи команды **optimset**.

```
function [y,dy]=fff1(x)                >> fminunc(@fff1,[3  3],optimset('Gradobj','on'))
y=x(1)^2+x(2)^2; % функци              Optimization terminated successfully:
dy=[2*x(1),2*x(2)]; % градиент          First-order optimality less than OPTIONS.TolFun, and no
                                       negative/zero curvature detected
                                       ans =     0     0
```

Теперь получено точное решение.

Пример. Рассмотрим задачу отыскания экстремума функции трех переменных

$$y = 2x_1^2 + 8x_2^2 + x_3^2 + 4x_1x_2 + 2x_1x_3 - 4x_3.$$

Вычисляя производные $\partial y / \partial x_i$ и приравнивая их нулю, получаем

$$4x_1 + 4x_2 + 2x_3 = 0; \quad 16x_2 + 4x_1 = 0; \quad 2x_3 + 2x_1 - 4 = 0.$$

Решение этой системы линейных уравнений имеет вид $x_1 = -4$, $x_2 = 1$, $x_3 = 6$.

Дополнительный анализ показывает, что найденное решение – точка минимума.

Упражнение. Найдите полученное решение с помощью функции **fminunc.**

Пример. Для проверки эффективности численных алгоритмов поиска минимума используют различные тестовые задачи. Одна из популярных тестовых функций была предложена Розенброком:

$$z = 100(y - x^2)^2 + (1 - x)^2.$$

Она всегда неотрицательна и обращается в нуль в единственной точке $x = y = 1$ (точка минимума). Сначала найдем этот минимум теоретически. Формальные выкладки приводят к двум уравнениям

$$z'_x = -400x(y - x) - 2(1 - x) = 0; \qquad z'_y = 200(y - x) = 0,$$

решая которые, получаем $x = y = 1$.

Коэффициенты функции Розенброка подобраны так, что экстремум оказывается очень слабо выраженным и представляет трудность для обнаружения поисковыми методами. Подробнее с процедурой его поиска можно познакомиться, запустив демонстрационную программу **bandem** тулбокса Optimization. Название программы связано с тем, что график функции Розенброка в окрестности точки минимума напоминает лежащий на боку банан.

Найдем минимум с помощью команды **fminunc**, используя градиентный поиск. Поиск начинаем с точки $x = -1{,}9$; $y = 2$.

```
>> f='100*(x(2)-x(1)^2)^2+(1-x(1))^2';                    % описание функции
>>  GRAD='[100*(4*x(1)^3-4*x(1)*x(2))+2*x(1)-2; 100*(2*x(2)-2*x(1)^2)]';      % описание градиента
>>OPTIONS = optimset(OPTIONS,'gradobj','on');          % подключение опции градиентного поиска
>> [x,fval] = fminunc({f,GRAD},[-1.9, 2],OPTIONS)
x =    1.0000    1.0000 fval =  1.2371e-015                % результат
```

Видим, что точка минимума успешно найдена.

Пример. Другой известный тестовый пример, представляющий трудность для компьютерных алгоритмов оптимизации – функция Пауэла. Эта функция четырех аргументов:

$$z = (a + 10b)^2 + 5(c - d)^4 + (b - 2c)^4 + 10(a - d)^4.$$

Она всегда положительна и достигает минимума при $a = b = c = d = 0$. Однако этот минимум очень плоский (в окрестности нуля функция растет крайне медленно), поэтому программы поиска экстремума могут остановиться (и выдать результат) довольно далеко от начала координат. Попытка аналитического отыскания экстремума функции Пауэла методом Ферма также наталкивается на трудности, поскольку приводит к необходимости решения системы из четырех кубических уравнений.

Упражнение. Найдите минимум функции Пауэла средствами MATLAB.

Для решения оптимизационных задач с ограничениями используются команды **fminbnd** и **fmincon**. Команда **fminbnd** (от *bound* – граница) предназначена для минимизации функции одной переменной на интервале $x_1 < x < x_2$. Например, x=fminbnd(inline('x^2+3*x+2'),0,1) даст ответ x = 0.

Для минимизации функций от нескольких переменных с ограничениями более сложного вида применяют команду **fmincon**. Список входных параметров функции довольно внушителен: x=fmincon(fun,x0,A,B,Aeq,Beq,Lb,Ub,nonlcon,options,p1,p2,...). Первый параметр, как обычно, имя функции fun, второй – начальное приближение x0. Остальные параметры вплоть до опций options,

определяют ограничения. При необходимости можно указать дополнительные параметры p1, p2, ... Если в задаче отсутствует какой-либо тип ограничений, соответствующие аргументы заменяются пустым массивом [ ].

Рассмотрим подробнее типы ограничений. Матрицы A и B определяют ограничения вида $AX \le B$, матрицы Aeq и Beq определяют ограничения вида $A_{eq}X = B_{eq}$. Векторы Ub и Lb определяют ограничения вида $Lb \le X \le Ub$. Нелинейные ограничения типа $G(X) = 0$ описываются функцией **nonlcon (**от *nonlinear constraints*).

Пример. Рассмотрим классическую задачу об экстремумах квадратичной формы $f(X)=X^TAX$ на сфере $X^TX=1$. Известно, что ее решения достигаются на собственных векторах симметричной матрицы A и равны ее собственным числам. Задав $A=\begin{bmatrix} 1 & 0{,}5 \\ 0{,}5 & 2 \end{bmatrix}$, получаем задачу об отыскании экстремума функции $f = x^2 + xy + 2y^2$ при ограничении $x^2 + y^2 = 1$. Получим решение двумя способами, используя команды **eig** и **fmincon.**

Спаособ 1.Найдем собственные векторы и собственные числа матрицы A, используя численный и символьный вариант ее задания:

```
>>A=[1 1/2;1/2 2]; [v,d]=eig(A), [V,D]=eig(sym(A)),
```

В результате получаем:

```
v =                    d =                    V =                        D =
  -0.9239    0.3827       0.7929         0     [       1,         1]      [ 3/2+1/2*2^(1/2),              0]
   0.3827    0.9239            0    2.2071     [ 1+2^(1/2), 1-2^(1/2)]    [         0,    3/2-1/2*2^(1/2)]
```

Следовательно, экстремумы функции f определяются формулой $f_э = \frac{3 \pm \sqrt{2}}{2}$ и равны 2,2 и 0,79. Максимум достигается в точке единичной окружности с координатами x=0,924, y=0,383; а минимум – в точке с координатами x=-0,924, y=0,383.

Геометрически они соответствуют точкам касания единичной окружности с описанным и вписанным эллипсами (рис. 4.19).

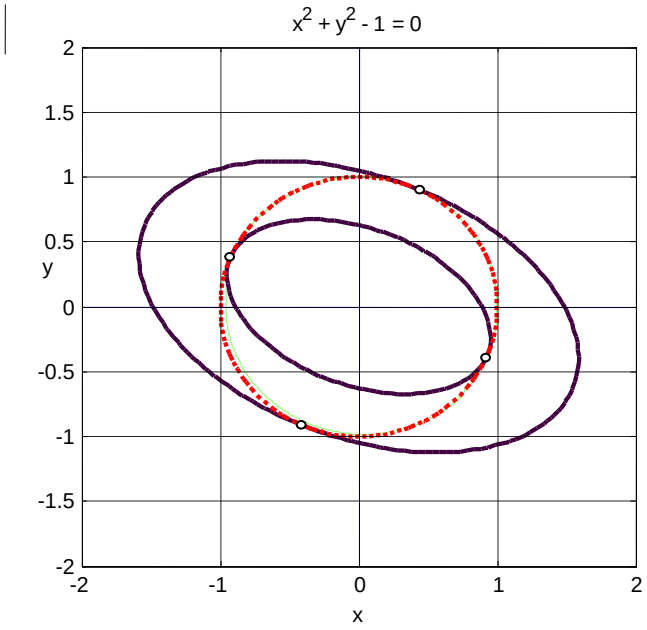


Рис. 4.19

Способ 2. Для применения команды **fmincon** нужно описать функцию $f = x^2 + xy + 2y^2$ и ограничение $g = x^2 + y^2 - 1 = 0$. Для описания первой из них воспользуемся командой **inline**, для второй создадим *m*-файл nlcon.m:

```
function [h,g] = nlcon(x)
    h = [];      g = x(1)^2+x(2)^2-1;
```

После этого набираем основную команду:

```
>> f=inline('x(1)^2+x(1)*x(2)+2*x(2)^2'); x=fmincon(@(x) f(x),[1;1],[],[],[],[],[],[],[[],@(x)nlcon(x)])
x =   -0.9239    0.3827
```

Мы получили координаты точки минимума. Чтобы получить координаты точки максимума, изменяем знак функции *f*:

```
f1=inline('-(x(1)^2+x(1)*x(2)+2*x(2)^2)'); x=fmincon(@(x) f1(x),[1;1],[],[],[],[],[],[],[[],@(x)nlcon(x)])
x =    0.3827    0.9239
```

Видим, что оба способа решения привели к одинаковым результатам.

Пример использования функции **fmincon** для подбора оптимальных параметров схемы моделирования, реализованной в SIMULINK, приводится в разд. 5.5.4.

Все перечисленные команды могут возвращать один, два и более выходных аргументов:

```
>> x=fminunc(@fff,[3  3]);
>> [x,fval]=fminunc(@fff,[3  3]);
>> [x,fval,flag]=fminunc(@fff,[3  3]);
>> [x,fval,flag]
ans =   -0.0000   -0.0000    0.0000    1.0000
```

Первый выходной аргумент – точка минимума, второй – значение функции в этой точке, а третий – флаг, равный единице при успешном завершении алгоритма. Если флаг меньше нуля, то алгоритм не сошелся, а если он равен нулю, превышено максимальное количество вычисления функций.

Отдельный класс оптимизационных задач образуют задачи линейного программирования, в которых и оптимизируемый критерий, и ограничения линейны. В них требуется найти экстремум критерия $J = c_1 x_1 + \ldots + c_n x_n$ при наличии ограничений в виде неравенств

$$a_{i1} x_1 + \ldots + a_{in} x_n \le b_i, \qquad i = 1, 2, \ldots, m.$$

Обычно эти условия записывают в матричной форме

$$c^T X \to extr, \qquad AX \le b.$$

Здесь b и c – векторы-столбцы, A – матрица размера $m \times n$.

Ограничения типа равенств можно отдельно не рассматривать, так как они легко сводятся к неравенствам, например вместо $x_1 + 3x_2 = 0$ можно записать два неравенства

$$x_1 + 3x_2 \le 0; \qquad -x_1 - 3x_2 \le 0.$$

Для решения задач линейного программирования разработаны различные численные методы, наиболее известным из которых является симплекс-метод.

В MATLAB задачи линейного программирования решают с помощью функции **linprog**. В простейшем случае у нее три входных параметра: linprog(c, A. b). В этом случае решается задача минимизации выражения c^T*x при условии A*x<=b (сравнение производится по всем строкам). Поясним ее применение на конкретном примере.

Пример. Задача о производстве стульев. Мебельная фабрика может выпускать стулья двух типов, стоимостью 6000 и 12000 рублей. Имеются следующие ресурсы: 440 погонных метров досок, 65 кв.м. обивочной ткани и 320 человеко-часов трудовых ресурсов. На изготовление одного стула требуются следующее количество ресурсов:

Стул	Расход досок	Расход ткани	Расход времени
Первый	2	0.5	2
Второй	4	0.25	2.5
Ресурс	440	65	320

Требуется так спланировать производство стульев, чтобы общая цена продукции была максимальной.

Решение. Перейдем к математической формулировке задачи. Обозначим через x количество стульев первого типа, через y – количество стульев второго типа. Тогда условия задачи сводятся к следующему:

$8x+12y \to \max$ – оптимизируемый критерий.

$2x+4y \le 440$ – ограничение по расходу досок

$0.5x+0.25y \le 65$ – ограничение по расходу ткани

$2x+2.5y \le 320$ – ограничение по расходу времени.

Матричная форма записи:

$$c^T X \to \max,\ c=\begin{bmatrix}8\\12\end{bmatrix},\ X=\begin{bmatrix}x\\y\end{bmatrix},\quad A=\begin{bmatrix}2 & 4\\0.5 & 0.25\\2 & 2.5\end{bmatrix},\quad b=\begin{bmatrix}440\\65\\320\end{bmatrix}.$$

Приведем два способа решения этой задачи в MATLAB – графический и численный.

Способ 1 (графический). Для графического решения построим на плоскости (*x*, *y*) три прямые, соответствующие ограничениям по трем ресурсам. По оси *x* будем откладывать количество стульев второго вида, по оси *y* количество стульев первого вида. Полученные прямые показаны на рис.4.20. Они, вместе с осями координат, задают область допустимых решений в виде неправильного пятиугольника. На том же рисунке показано семейство прямых $8x+12y=const$.

Решение задачи дает крайняя правая прямая этого семейства, касающаяся многоугольника допустимых решений в точке с координатами (80, 60).

Графики построены в MATLAB с помощью следующей программы:

```
x=0:0.2:300; y1=-2*x+220; y2=(-1/2)*x+130; y3=(-5/4)*x+160;
plot(x,y1,x,y2,x,y3); grid; hold on
for c=0:60:1460
   y=-3/2*x+c/8;
   plot(x,y,'black');grid on;
end
```

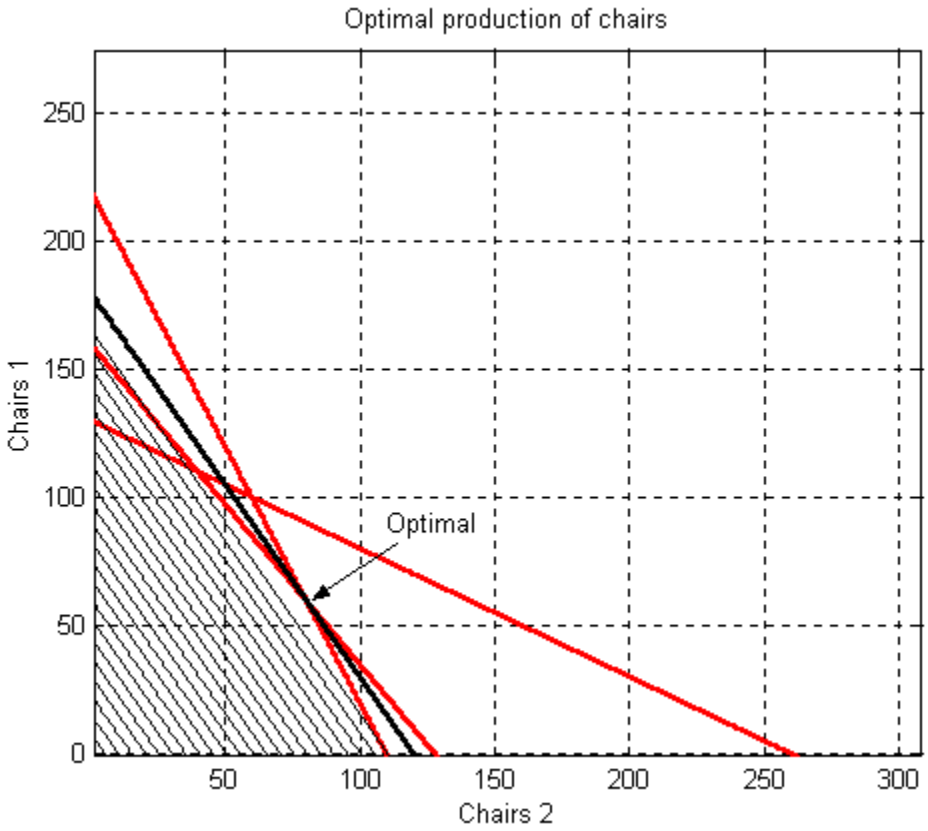


Рис. 4.20. Графическое решение.

Способ 2 (численный). Для численного решения той же задачи применяем функцию **linprog**.

```
>>X=linprog(-[8; 12],[2 4;0.5 0.25;2 2.5;],[440;65;320])
Optimization terminated successfully.
X= 60.0000
    80.0000
```

По условиям задачи требовалось найти максимум, поэтому, чтобы свести задачу к поиску минимума, первый параметр взят с коэффициентом -1. Мы получили то же решение, что и первым способом – максимальная прибыль будет получена, если выпустить 60 стульев первого типа и 80 стульев второго типа.

Функции от матриц

Нормы и числа обусловленности

В разделе 1 были рассмотрены команды MATLAB для определения таких числовых характеристик матриц, как ранг, определитель, след, собственные числа. Из других команд этого ряда отметим **svd, cond** и **norm**, служащие для определения сингулярных чисел, чисел обусловленности и норм (векторных и матричных).

По команде `S=svd(A)` вычисляются сингулярные числа матрицы A, т.е. положительные квадратные корни из собственных чисел матрицы A^TA. Они используются при определении ранга матрицы, при оценке ее обусловленности, при решении задач аппроксимации и редукции. Эта же команда с тремя выходными аргументами находит сингулярное разложение матрицы A. Заметим,

что для симметричных положительно определенных матриц сингулярные числа совпадают с собственными числами, в чем можно убедиться, сравнивая результаты команд **svd** и **eig.**

Обе эти команды могут работать с символьными аргументами. Приведем простой пример:

```
>>A =[2   4; 6   2];     >>S=svd(A)          >>S= svd(sym(A))
A= 2  4                  S=7.2361            S=5+5^(1/2)
   6  2                     2.7639              5-5^(1/2)
```

Максимальное из сингулярных чисел матрицы определяет ее спектральную норму, а отношение максимального сингулярного числа к минимальному дает число обусловленности по этой норме. Так, в рассматриваемом примере норма матрицы равна $\|A\|_2 = 5+\sqrt{5} = 7{,}2361,$ а число обусловленности $\mu_2 = \dfrac{5+\sqrt{5}}{5-\sqrt{5}} = \dfrac{3+\sqrt{5}}{2} = 2{,}618$ (любители математики узнают здесь золотое сечение, увеличенное на единицу).

В математике известен целый ряд векторных и матричных норм. Основные из них могут быть вычислены с помощью команды **norm** и ее модификаций. Три наиболее употребительные векторные нормы вычисляются с помощью следующих команд MATLAB:

norm (x) и **norm(x, 2)** – вторая евклидова норма вектора $\|X\|_2 = (x_1^2 + ... + x_n^2)^{1/2}$ (именно ей мы пользуемся в обыденной жизни для измерения длины);

norm(x, 1) - первая или модульная норма вектора $\|X\|_1 = |x_1| + ... + |x_n|$ (мы пользуемся ей, определяя расстояние в городе с прямоугольной планировкой улиц);

norm(x, inf) – бесконечная (чебышевская) норма вектора $\|X\|_\infty = \max_i(|x_i|)$ (это габаритный размер параллелепипеда с диагональю X).

Эта же команда позволяет вычислять матричные нормы:

norm(A) и **norm(A, 2)** – спектральная норма матрицы (равна ее наибольшему сингулярному числу) $\|A\|_2 = \max_i \lambda_i^{1/2}(A^T A),$ тот же результат можно получить как max(svd(A));

norm(A, 1) – первая (столбцовая) норма матрицы A (определяется столбцом с наибольшей суммой элементов) $\|A\|_1 = \max_i(|a_{ij}| + ... + |a_{mj}|)$;

norm(A, inf) – бесконечная (строчная) норма матрицы A (определяется строкой с наибольшей суммой элементов) $\|A\|_\infty = \max_i(|a_{i1}| + ... + |a_{in}|)$;

norm(A, 'fro') – фробениусова норма матрицы (равна евклидовой длине вектора, составленного из элементов матрицы) $\|A\|_F = (a_{11}^2 + a_{12}^2 + ... + a_{nm}^2)^{1/2} = tr^{1/2}(A^T A).$

Матричные нормы используются, в частности, при оценке степени обусловленности матриц. Как известно, числом обусловленности (*condition number*) называется величина $\mu = \mathrm{cond}(A) = \|A^{-1}\| \cdot \|A\|.$ Ее можно найти с помощью команды cond(A). Минимальное значение числа обусловленности μ=1 соответствует идеально обусловленной матрице. Чем больше μ, тем хуже обусловленность и тем большей погрешностью будет сопровождаться численное решение системы линейных алгебраических уравнений AX=b. Различным матричным нормам будут соответствовать различные числа обусловленности, требуемая модификация указывается вторым параметром команды **cond.**

При решении задач аппроксимации, оптимального управления и многих других возникает необходимость вычисления норм функций одной переменной $y = f(t),$ заданных на интервале $0 \le t \le T.$ В технических приложениях это чаще всего нормы входных и выходных сигналов некоторой системы.

Обычно рассматривают три классические нормы функций – модульную или первую, гильбертову или вторую (квадратичную) и чебышевскую (другие названия – равномерная, бесконечная):

$$\|y\|_1 = \int_0^T |y(t)|dt\,,\ \ \|y\|_2 = \left(\int_0^T y^2(t)dt\right)^{1/2},\qquad \|y\|_\infty = \max_{0\le t\le T}|y(t)|.$$

В MATLAB нет специальных команд для вычисления этих норм, однако они легко могут быть найдены с помощью функции численного интегрирования **trapz** и команды выбора максимального элемента массива **max**:

Вычисление модульной нормы функции $y(t)$: N=trapz(t, abs(y));

Вычисление гильбертовой нормы функции $y(t)$: N= sqrt(trapz(t, y.*y));

Вычисление чебышевской нормы функции $y(t)$: N=max(abs(y)).

При этом переменные t и y должны быть заданы массивами своих значений.

Возможны и другие варианты, например, для вычисления интеграла можно использовать команду **lsim** с интегратором в качестве первого аргумента.

Пример. Рассмотрим систему второго порядка с передаточной функцией $Q(p)=1/(p+\alpha)^2$, представляющую собой последовательное соединение двух одинаковых апериодических звеньев. Весовая функция системы $q(t)$ и ее энергия N^2 (квадрат гильбертовой нормы) на бесконечном интервале времени имеют вид

$$q = te^{-at};\ N^2 = \int_0^\infty q^2 dt = \frac{1}{4\alpha^3}.$$

Приведем два способа получения нормы N в MATLAB, полагая α=1.

а) Численный способ. Используем команды **impulse** и **trapz**:

```
>>sys=tf(1,[1 2 1]); [q,t]=impulse(sys);        % получение весовой функции системы и массива времени
>>Nq=trapz(t, (q.*q) ); nq = sqrt(Nq)           % Nq - интеграл на интервале [ 0,  T ], где T = max(t);
nq =  0.5000                                    % результат
```

б) Символьный способ. Используем обратное преобразования Лапласа и символьное интегрирование:

```
>>syms p; q=ilaplace(1/(p+1)^2)
q =t*exp(-t)                                    % весовая функция системы
>>Mq=int(q^2,0, inf); mq=sqrt(Mq)               % Mq – интеграл в пределах от нуля до бесконечности
mq =1/2                                         % результат
```

В обоих случаях результат совпадает с теоретическим.

Упражнение. Дано апериодическое звено с передаточной функцией $Q(p)=1/(p+\alpha)$. Его весовая функция $q(t)$ и квадрат ее гильбертовой нормы на интервале (0, T) определяются формулами

$$q = e^{-\alpha t};\ \ N^2 = \int_0^T q^2 dt = \frac{1}{2\alpha}(1-e^{-2\alpha T}).$$

Найдите средствами MATLAB норму N для значений $T = 1, 2, \infty$.

Матричная экспонента

В MATLAB имеется несколько команд для вычисления функций от квадратных матриц – это **polyvalm, sqrtm, expm, logm**. Команда **polyvalm** предназначена для вычисления значений матричного полинома $F(A) = a_n A^n + \ldots + a_1 A + a_0 E$. Ее первый аргумент – вектор коэффициентов полинома, второй – матрица A.

Функция **sqrtm (A)** вычисляет квадратный корень из матрицы A. Если матрица A имеет отрицательные собственные числа, результат будет комплексной матрицей.

Команда expm (A) обеспечивает вычисление матричной экспоненты e^A путем использования аппроксимации Паде, а операция logm (A) – вычисление матричного логарифма $\ln A$. Напомним определение матричной экспоненты. Как известно, обычная экспонента e^a задается разложением в ряд Тэйлора $e^a = 1 + a + \frac{a^2}{2!} + \frac{a^3}{3!} + \ldots \frac{a^n}{n!} + \ldots$.

Матричная экспонента для заданной квадратной матрицы A вводится как сумма аналогичного ряда

$$e^A = E + A + \frac{A^2}{2!} + \ldots + \frac{A^n}{n!} + \ldots \qquad (*)$$

Она представляет собой квадратную матрицу того же размера, что и матрица A, и обладает рядом свойств, характерных для обычной экспоненты.

Особенно легко находится матричная экспонента для диагональных матриц – для этого достаточно взять обычные экспоненты от диагональных элементов (заметим, что это единственный случай, когда expm(A)=exp(A)). В общем случае вычисление матричной экспоненты – трудоемкая процедура. Для контроля правильности результата можно использовать проверку равенства $\det\left(e^A\right) = e^{trA}$, связывающего определитель матричной экспоненты со следом исходной матрицы.

Пример. Найдем матричные экспоненты для следующих трех матриц:

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \; A_2 = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \; A_3 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

Возводя их в степени и выполняя поэлементное суммирование членов ряда (*), получаем:

$$e^{A_1} = \begin{bmatrix} e & 0 \\ 0 & e^2 \end{bmatrix}, \; e^{A_2} = \begin{bmatrix} e & 2e \\ 0 & e \end{bmatrix}, \quad e^{A_3} = \begin{bmatrix} \cos 1 & \sin 1 \\ -\sin 1 & \cos 1 \end{bmatrix}.$$

След матрицы A_1 равен трем, определитель матрицы e^{A_1} равен e^3, так что равенство $\det e^A = e^{trA}$ выполняется. Легко проверить его справедливость и для остальных матриц.

Вычислим те же матричные экспоненты в MATLAB с помощью команды **expm**:

```
>>A1=[1 0;0 2];
>>e1=expm(A1)
    2.7183    0
    0    7.3891
```

```
>>A2=[1 2;0 1];
>>e2=expm(A2)
2.7183    5.4366
0         2.7183
```

```
>>A3=[0 1;-1 0];
>>e3=expm(A3)
  0.5403    0.8415
 -0.8415    0.5403
```

Элементами полученных матриц служат величины $e \approx 2{,}7183$, $e^2 \approx 7{,}3891$, $2e \approx 5{,}4366$, а также $\sin 57{,}3^0$ и $\cos 57{,}3^0$.

У матрицы A_3 собственные числа были чисто мнимые $\pm i$. В общем случае матрица второго порядка имеет комплексные собственные числа $\alpha \pm i\beta$, преобразованием подобия она может быть приведена к виду $A = \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$. Матричная экспонента для таких матриц определяется формулой $e^A = e^a \begin{bmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{bmatrix}$.

Пример. Для матрицы $A=\begin{bmatrix}-1 & -0.5\\ 0.5 & -1\end{bmatrix}$ получаем $e^A=\begin{bmatrix}\cos 0.5 & -\sin 0.5\\ \sin 0.5 & \cos 0.5\end{bmatrix}/e$.

Применение функции **expm** дает тот же результат:

```
>>exp(-1)*[cos(-1/2), sin(-1/2);
          -sin(-1/2),cos(-1/2)]
    ans =
       0.3228   -0.1764
       0.1764    0.3228
```

```
>> A=[-1 -.5;.5 -1];
>>expm(A)
ans =
0.3228  -0.1764
 0.1764    0.3228
```

```
>>A=[-1 -.5;.5 -1];
>>expm(sym(A))
ans =
[exp(-1)*cos(1/2),-exp(-1)*sin(1/2)]
[exp(-1)*sin(1/2),exp(-1)*cos(1/2)]
```

Последний столбец и следующий пример демонстрируют возможность тулбокса SYMBOLIC вычислять матричную экспоненту в символьном виде.

Пример. Найти матричную экспоненту e^{tA}, если $A=\begin{bmatrix}-2 & 1 & -2\\ 1 & -2 & 2\\ 3 & -3 & 5\end{bmatrix}$.

Решение в MATLAB.

```
 >> A=[-2 1 -2;1 -2 2;3 -3 5]; eig(a)'
ans =   -1    3   -1
>> syms t, F=expm(t*A); 4*F
4F = [   -exp(3*t)+5*exp(-t),     -exp(-t)+exp(3*t),  2*exp(-t)-2*exp(3*t)]
[     -exp(-t)+exp(3*t),   -exp(3*t)+5*exp(-t), -2*exp(-t)+2*exp(3*t)  ]
[ -3*exp(-t)+3*exp(3*t),  3*exp(-t)-3*exp(3*t),  6*exp(3*t)-2*exp(-t)]
```

Это означает, что матричная экспонента имеет вид:

$$F=\frac{1}{4}\begin{bmatrix}-e^{3t}+5e^{-t} & e^{3t}-e^{-t} & -2e^{3t}+2e^{-t}\\ e^{3t}-e^{-t} & -e^{3t}+5e^{-t} & 2e^{3t}-2e^{-t}\\ 3e^{3t}-3e^{-t} & -3e^{3t}+3e^{-t} & 6e^{3t}-2e^{-t}\end{bmatrix}.$$

Структуру собственных чисел и векторы матрицы A можно найти с помощью команд **eig** и **jordan**:

```
>> [V,D]=eig(sym(A)),  [v,d]=jordan(sym(A))
```

```
V                D                v                d
[ -1,  1, -2]    [  3,  0,  0]    [ -3/4, -1/4,  -2]    [ -1,  0,  0]
[  1,  1,  0]    [  0, -1,  0]    [ -1/4,  1/4,   0]    [  0,  3,  0]
[  3,  0,  1]    [  0,  0, -1]    [  1/4,  3/4,   1]    [  0,  0, -1]
```

Хотя у матрицы A есть кратное собственное число -1, она диагонализируема, т.е. является матрицей простой структуры.

Одно из важных применений матричной экспоненты – аналитический расчет свободного движения линейных динамических объектов. Математически эта задача сводится к решению системы однородных дифференциальных уравнений

$$\dot{X}=AX,\quad X(0)=X_0,$$
$$y=CX,$$

где A – квадратная матрица, C – прямоугольная матрица, X – вектор переменных состояния.

Стандартное средство для решения этой задачи в MATLAB – команда **initial** тулбокса CONTROL. Другой способ – использование матричной экспоненты. Он опирается на запись решения в форме

$$X(t) = Ce^{At} X_0.$$

Такой способ может оказаться более удобным, если надо получить несколько решений одной и той же системы при различных начальных условиях. При этом матричная экспонента вычисляется только один раз, в то время как при использовании команды **initial** придется многократно повторять процесс моделирования.

Пример. Колебания маятника описываются дифференциальным уравнением

$$\ddot{y} + 0{,}2\dot{y} + 4y = 0.$$

Требуется получить графики $y(t)$ при четырех вариантах начальных условий: $y_0 = 0{,}25;\ 0{,}5;\ 0{,}75;\ 1;\quad \dot{y}_0 = 0.$

Решение. Преобразуем уравнение к форме Коши, обозначая $x_1 = y;\ x_2 = \dot{y}$:

$$\dot{x}_1 = x_2,$$
$$\dot{x}_2 = -4x_1 - 0{,}2x_2.$$

Выпишем матрицы A, C этой системы и матрицу X_0 четырех вариантов начальных условий:

$$A = \begin{bmatrix} 0 & 1 \\ -4 & -0{,}2 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad X_0 = \begin{bmatrix} 0{,}25 & 0{,}5 & 0{,}75 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Составим программу expmpend.m для получения решения согласно формуле $y = Ce^{At} X_0$:

```
% Program expmpend
A=[0 1; -4 -0.2]; c=[1 0];          % матрицы  A, C
X0=[1/4 1/2 3/4 1; 0 0 0 0];        % матрица начальных условий
h=0.1; t=0:h:10; N=length(t);       % массив времени
Eh=expm(A*h);
F=zeros(N,2); F(1,:)=c;
    for i=2:N
F(i,:)=F(i-1,:)*Eh;                 % цикл вычисления массива F = Ce^At размера Nx2
    end
Y=F*X0;                             % решение для четырех начальных условий
plot(t,Y), grid                     % вывод графиков
```

Графики четырех полученных решений показаны на рис. 4.21.

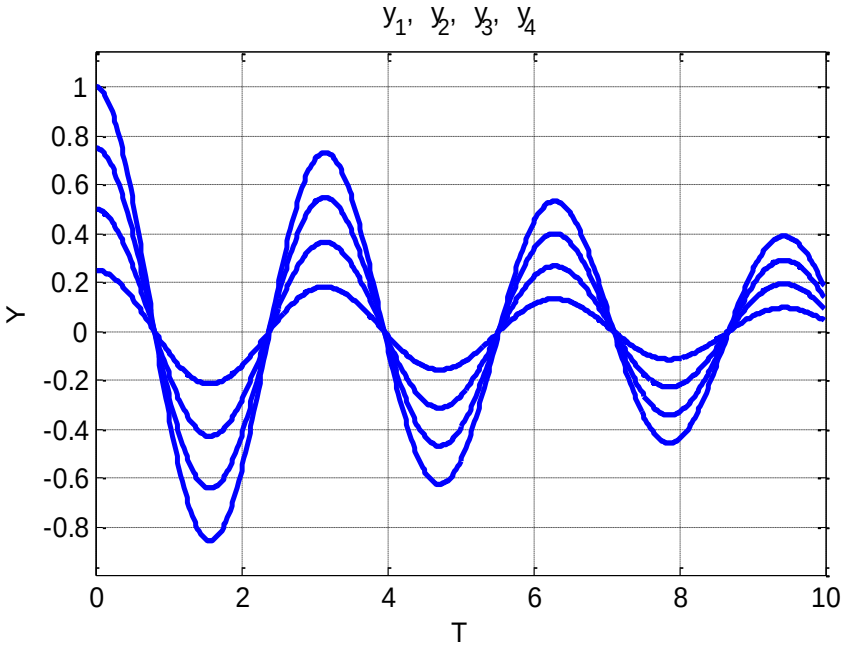


Рис. 4.21

Более короткий путь их получения, не требующий организации цикла, основан на использовании символьных вычислений и вывода графиков командой **ezplot.**

```
A=[0 1; -4 -0.2]; c=[1 0];                  % матрицы  A, C
X0=[1/4 1/2 3/4 1; 0 0 0 0];                % матрица начальных условий
syms  T
E=expm(A*T);Y=c*E*X0;
ezplot(Y(1),[0 10]),hold on,ezplot(Y(2),[0 10]),grid,ezplot(Y(3),[0 10]),
ezplot(Y(4),[0 10]),hold off, title('y_1,  y_2,  y_3,  y_4')
```

К сожалению, **ezplot** не позволяет выводить в одно графическое окно сразу несколько кривых, поэтому приходится дополнительно прибегать к команде **hold.**

Канонические формы матриц

Преобразование подобия

При решении многих задач линейной алгебры подходящей заменой переменных удается упростить вид матриц, входящих в эти задачи. Одно из наиболее распространенных преобразований описывается формулой $B = TAT^{-1}$, где A – исходная квадратная матрица, T – матрица невырожденной замены переменных, B – результат преобразования. Матрицы A и B, связанные таким преобразованием, называются подобными, а само преобразование – подобием.

Преобразование подобия часто возникает при решении прикладных задач. Наиболее распространенный пример – решение системы линейных уравнений A X = Y. При замене переменных матрица A этой системы подвергается преобразованию подобия, изменяющему как ее внешний вид, так и многие свойства (например, обусловленность, что необходимо учитывать при решении линейных систем уравнений на ЭВМ).

Другой важный пример – моделирование свободного движения линейной системы $\dot{X} = AX$. Переходя к новым переменным $X = T\,Y$, получаем систему $\dot{Y} = T^{-1}ATY$, матрица которой подобна матрице исходной системы.

Одна из команд MATLAB, использующая преобразование подобия, называется **balance.** Она подбирает диагональную матрицу преобразования T таким образом, чтобы выровнять нормы строк и столбцов матрицы A по величине. По существу, речь идет о разумном масштабировании переменных, улучшающем обусловленность матрицы (иногда говорят о балансировке или уравновешивании элементов).

Пример. Сформируем матрицу A с большим разбросом элементов и выполним ее балансировку:

```
>>A=[1  100  0.1;  1  0  0;   0  100  1],  [T, B]=balance(A)
```

В результате получаем:

A			T			B		
1	100	0.1	2	0	0	1	12.5	1.6
1	0	0	0	0.25	0	8	0	0
0	100	1	0	0	32	0	0.7813	1

Разброс элементов в матрице B по сравнению с исходной матрицей A заметно уменьшился. На диагонали матрицы T стоят масштабные коэффициенты, они представляют собой степени двойки (чтобы избежать погрешностей округления).

За счет преобразования подобия можно достичь значительного упрощения вида матрицы A, придав ей регулярную структуру и сделав нулевыми или единичными большинство ее элементов. Стандартные формы, к которым может быть приведена матрица A путем преобразования подобия, называются каноническими. Наибольшую известность получили сопровождающая каноническая форма (*companion form*), диагональная каноническая форма (форма Жордана), треугольная форма Шура, форма Хессенберга, форма Шварца.

В пакете MATLAB существуют команды для получения этих форм: **compan, jordan, schur, hess.** Остановимся на двух первых из них.

Фробениусова каноническая форма

Параметрами этой канонической формы служат коэффициенты характеристического полинома

$$\left|pE - A\right| = p^n + a_{n-1}p^{n-1} + ... + a_1 p + a_0.$$

Если степень минимального полинома матрицы A равна n, то ее фробениусова каноническая форма имеет вид

$$B = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix} = \begin{bmatrix} 0 & \vdots & E \\ \cdots & \cdots & \cdots \\ & -a & \end{bmatrix}.$$

где $a = (a_0, ..., a_{n-1})$.

Все варьируемые элементы матрицы расположены в последней строке и совпадают с точностью до знака с коэффициентами ее характеристического полинома. По этой причине ее называют также матрицей, сопровождающей характеристический полином, ее собственные числа равны его корням.

Вместо матрицы B можно использовать подобные ей матрицы

$$\begin{bmatrix} 0 & \vdots & \\ \dots & \vdots & -a \\ E & \vdots & \end{bmatrix}, \quad \begin{bmatrix} & -a & \\ \dots & \dots & \dots \\ E & \vdots & 0 \end{bmatrix}, \quad \begin{bmatrix} & \vdots & E \\ -a & \vdots & \dots \\ & \vdots & 0 \end{bmatrix},$$

получающиеся из нее транспонированием и перенумерацией переменных. Таким образом, имеем два строчных и два столбцовых варианта фробениусовых матриц.

Если степени минимального и характеристического полинома матрицы не совпадают, то фробениусова каноническая форма становится клеточно-диагональной с клетками вида *B*.

Процедура получения фробениусовой канонической формы довольно проста. Ранее отмечалось, что команда `P=poly(A)` дает вектор коэффициентов характеристического полинома матрицы *A*. Команда **compan** решает обратную задачу – она строит матрицу по характеристическому полиному.

Набрав `B=compan(P)`, получим матрицу *B*, первая строка которой образована коэффициентами полинома *P* (деленными на старший коэффициент и взятыми с минусом), а все остальные элементы равны нулю, за исключением первой поддиагонали, на которой стоят единицы.

Пример. Пусть задан полином P третьего порядка: $P = x^3 + 2x^2 + 3x + 4$. Для построения сопровождающей матрицы набираем команды: `P=[1 2 3 4]; B=compan(P)`. Результатом будет матрица

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

Находя ее характеристический полином (это можно сделать с помощью команды `poly(B)`), убеждаемся, что он совпадает с P.

В соответствии со сказанным выше, сопровождающими полином P будут также матрицы

$$\begin{bmatrix} -2 & 1 & 0 \\ -3 & 0 & 1 \\ -4 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -3 & -2 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 & -4 \\ 1 & 0 & -3 \\ 0 & 1 & -2 \end{bmatrix}.$$

Все они подобны и имеют одинаковые характеристические полиномы.

Если дана произвольная квадратная матрица A с разными собственными числами, то для преобразования ее в сопровождающую каноническую форму достаточно двух команд: `p=poly(A); B=compan(p)`. Матрицы A и B будут иметь одинаковые характеристические полиномы и будут подобны. В случае алгебраически кратных собственных чисел характеристические полиномы матриц A и B будут по-прежнему совпадать, однако подобие уже не будет иметь место. Простого способа получения сопровождающей канонической формы в MATLAB для этого случая не существует.

Упражнение 1. Выполните описанный выше переход для матрицы `A=eye(2)` и проверьте, будут ли подобными матрицы A и B.

Упражнение 2. Пусть матрицы A и `B=compan(poly(A))` подобны. Подумайте, как найти связывающую их матрицу преобразования T? Проверьте ваш способ в MATLAB, используя формулу $B = T^{-1}AT$.

Жорданова каноническая форма

Каноническая форма Жордана и разные ее модификации широко используются в приложениях, возможность ее нахождения предусмотрена в большинстве математических пакетов. Параметрами канонической формы Жордана служат корни характеристического уравнения матрицы, поэтому в общем случае это – комплексная матрица. Если все корни различны, то жорданова каноническая форма имеет диагональный вид, при этом диагональными элементами являются собственные числа исходной матрицы.

Жорданова форма полезна при решении многих задач. Один из характерных примеров – решение системы дифференциальных уравнений $\dot{X} = AX$. Если все собственные числа $d_1, \ldots, d_n$ матрицы A различны, то существует такая невырожденная матрица V, что заменой переменных $X = VY$ исходная система приводится к «диагональному» виду, который описывается уравнениями

$$\dot{y}_1 = d_1 y_1, \quad \dot{y}_2 = d_2 y_2, \ldots, \dot{y}_n = d_n y_n.$$

Решая их, получаем $y_1 = c_1 e^{d_1 t}$, $y_2 = c_2 e^{d_2 t}$, …, $y_n = c_n e^{d_n t}$, где постоянные c_i зависят от начальных условий.

Матричное описание полученной системы дифференциальных уравнений имеет вид

$$\dot{Y} = DY, \qquad D = \begin{bmatrix} d_1 & 0 & \ldots & 0 & 0 \\ 0 & d_2 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 0 & d_n \end{bmatrix}.$$

Матрицы A и D связаны соотношением $D = V^{-1}AV$, где V – матрица, составленная из собственных векторов матрицы A.

Матрица D называется диагональной или жордановой канонической формой матрицы A для матриц простой структуры. В MATLAB матрицы V и D можно найти с помощью команды [V, D]=eig(A).

Пример. Найдем жорданову каноническую форму матрицы $A = \begin{bmatrix} -3 & -6 & 6 \\ -2 & 6 & 2 \\ -12 & -6 & 15 \end{bmatrix}$. Ее собственные числа $d_1 = 3,\ d_2 = 6,\ d_3 = 9.$ Матрица собственных векторов имеет вид: $V = \begin{bmatrix} 1 & 2 & 1 \\ 0 & -1 & 2 \\ 1 & 2 & 4 \end{bmatrix}.$

Вычисления в MATLAB дают

```
>>A =[-3 -6  6;-2  6  2;-12 -6 15]; [V,D]=eig(A)
```

```
V =                                  D =
-0.7071   0.2182   0.6667            3.0000        0        0
-0.0000   0.4364  -0.3333                 0   9.0000        0
-0.7071   0.8729   0.6667                 0        0   6.0000
```

На диагонали матрицы D стоят собственные числа. Заметим, что MATLAB выдает матрицу собственных векторов в нормированном виде, так что сумма квадратов элементов в каждом столбце равна единице. Порядок расположения собственных векторов в матрице V соответствует

расположению собственных чисел в матрице D. В принципе, мы имеем право умножить каждый столбец матрицы V на любое число, поскольку собственные векторы определены с точностью до постоянного множителя.

Команду **eig** можно применять и к символьным матрицам, при этом MATLAB старается выдать ответ в целочисленном виде, например

```
>> A =sym([-3 -6  6;-2  6  2;-12 -6 15]); [V,D]=eig(A)
```

```
V =                          D =
[  -2,   1, 1/2]             [ 6, 0, 0]
[   1,   0,   1]             [ 0, 3, 0]
[  -2,   1,   2]             [ 0, 0, 9]
```

По сравнению с предыдущим ответом MATLAB изменил порядок собственных чисел и длину собственных векторов.

Если собственные числа матрицы комплексные, то ее собственные векторы также будут комплексными.

Пример. Два из трех собственных чисел матрицы $B=\begin{bmatrix} 0 & 1 & 1 \\ -2 & 1 & 1 \\ -2 & 1 & 3 \end{bmatrix}$ комплексные: $d_1=2,\ d_2=1+i,\ d_3=1-i,$ им соответствуют собственные векторы:

$$h_1=\begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix},\quad h_2=\begin{bmatrix} +i \\ -1 \\ i \end{bmatrix},\quad h_3=\begin{bmatrix} 1 \\ i \\ 1 \end{bmatrix}.$$

Найдем жорданову каноническую форму матрицы B:

```
>>B =[0 1  1;-2  1  1;-2 1 3]; [H,L]=eig(B)
```

```
H =                                                    L =
  -0.4472      0.0000 - 0.5774i   0.0000 + 0.5774i       2.0000            0                    0
   0.0000      0.5774             0.5774                    0      1.0000 + 1.0000i             0
  -0.8944      0.0000 - 0.5774i   0.0000 + 0.5774i          0                 0    1.0000 - 1.0000i
```

Повторим те же вычисления в символьной форме

```
>>B =sym([0 1  1;-2  1  1;-2 1 3]); [H,L]=eig(B)
```

```
H =                          L =
 [ 1,  1,  1]                [ 1+i,  0,   ]
[  i, -i,   0]               [ 0, 1-i,   0]
[  1,  1,  2]                [ 0,   0,   2]
```

Здесь матрицы H и L представлены в более удобном виде.

Полученную матрицу L можно преобразовать к вещественной форме, в которой каждой паре комплексных собственных чисел $\alpha\pm i\beta$ соответствует диагональная клетка $\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$.

В MATLAB такой переход осуществляется командой **cdf2rdf** (сокращение от *complex diagonal form to real diagonal form*). Ее входными аргументами являются комплексные матрицы

собственных чисел и векторов, возвращаемые командой **eig**, а выходными аргументами – матрицы вещественной формы.

Применим ее к результату последнего примера.

```
>> [H1,L1]=cdf2rdf(H,L)

H1 =                                    L1 =
  -0.4472   0.0000   -0.5774              2.0000        0        0
   0.0000   0.5774         0                 0   1.0000   1.0000
  -0.8944   0.0000   -0.5774                 0  -1.0000   1.0000
```

Матрица L1 – это вещественная форма Жордана матрицы B. Второй и третий столбцы матрицы H1 представляют собой вещественную и мнимую части соответствующих векторов матрицы H. Обратный переход к комплексной жордановой форме выполняется с помощью команды **rsf2csf**. Синтаксис вызова обеих команд одинаков.

Если среди корней минимального полинома матрицы A имеются кратные, то на диагонали ее жордановой канонической формы появляются клетки соответствующего размера вида

$$\begin{bmatrix} \lambda_1 & 1 \\ 0 & \lambda_1 \end{bmatrix}, \quad \begin{bmatrix} \lambda_2 & 1 & 0 \\ 0 & \lambda_2 & 1 \\ 0 & 0 & \lambda_2 \end{bmatrix} \quad \text{и т.д.}$$

В этом случае применять команду **eig** следует с осторожностью, поскольку неопытного пользователя ее результаты могут ввести в заблуждение.

Пример. Рассмотрим матрицу $C = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}$. Все ее собственные числа равны 2, характеристический полином имеет вид $(\lambda - 2)^3$, минимальный полином равен $(\lambda - 2)^2$. Следовательно, алгебраическая кратность собственного числа 2 равна трем, а геометрическая – двум. У матрицы C только два собственных вектора $V_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $V_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$. Вычислим их с помощью команды **eig.**

```
>>C=[2 0 0;0 2 1;0 0 2], [H,L]=eig(C)

H =                                     L =
   1.0000        0         0              2   0   0
        0   1.0000   -1.0000              0   2   0
        0         0    0.0000              0   0   2
```

MATLAB правильно нашел собственные числа матрицы C и оба собственных вектора, однако матрица L уже не является жордановой канонической формой матрицы C. При этом MATLAB не выдал сообщения о том, что имеются только два собственных вектора, а просто дважды поместил в матрицу H второй собственный вектор (во второй и третий столбцы).

Застраховаться от подобных недоразумений можно, вычисляя ранг матрицы собственных векторов (в нашем примере он равен двум, т.е. меньше размера матрицы), либо переходя к символьным вычислениям. При этом в команде **eig** можно задавать третий выходной параметр, который укажет количество собственных векторов, имеющихся у матрицы.

```
>> C=[2 0 0;0 2 1;0 0 2];  [H1,L1,p]= eig(sym(C))

C =                    H1 =          L1 =              p =
   2   0   0           [ 1, 0]       [ 2, 0, 0]        1   2
   0   2   1           [ 0, 1]       [ 0, 2, 0]
   0   0   2           [ 0, 0]       [ 0, 0, 2]
```

Теперь MATLAB выдал два собственных вектора, параметр p указывает на их число. Однако матрица L1 по-прежнему осталась диагональной.

Получить жорданову каноническую форму матрицы C можно, используя команду **jordan**. Ее синтаксис [V,J]=Jordan(A), где V – матрица перехода к жордановой канонической форме J: V/A*V=J.

Для нашего примера получаем:

```
>> C=[2 0 0;0 2 1;0 0 2]; [V,J] = jordan(C),

C =                    V =                J =
   2   0   0           0   1   -2         2   1   0
   0   2   1           1   0    1         0   2   0
   0   0   2           0   1    0         0   0   2
```

Столбцами V служат так называемые корневые или обобщенные собственные векторы матрицы C (подробнее о них можно прочитать в учебном пособии [9].

Надо иметь в виду, что команда **jordan** очень чувствительна к погрешностям. Малейшее изменение входных данных может изменить структуру канонической формы. Увеличим, например, на 0,01 последний элемент матрицы C:

```
>> C1=[2 0 0;0 2 1;0 0 2.01]; [V1,J1] = jordan(sym(C1)),

C1 =                   V1 =                          J1 =
   2   0    0          [     1,     0,  -201/100]        [ 2,       0,       0]
   0   2    1          [  -100,   100,         1 ]        [ 0,  201/100,    0]
   0   0 2.01          [     0,     1,         0 ]        [ 0,       0,       2]
```

В результате матрица J_1 – каноническая форма Жордана матрицы C_1 – стала диагональной, в то время как матрица J имела треугольную жорданову клетку.

Сопоставляя жорданову A_J и фробениусову (сопровождающую) A_F канонические формы матрицы A, отметим следующее.

При отсутствии кратных корней характеристического полинома связь жордановой и фробениусовой канонических форм осуществляется с помощью матрицы Вандермонда V, составленной из собственных чисел $\lambda_1, ..., \lambda_n$ матрицы A:

$$A_J = V^{-1} A_F V, \qquad V = \begin{bmatrix} \lambda_1^0 & \dots & \lambda_n^0 \\ \dots & \dots & \dots \\ \lambda_1^{n-1} & \dots & \lambda_n^{n-1} \end{bmatrix}.$$

В MATLAB матрица Вандермонда может быть получена с помощью команды **vander.** Непосредственной проверкой легко убедиться, что столбцы матрицы V являются собственными векторами матрицы Фробениуса. Сопровождающая каноническая форма имеет определенные преимущества перед формой Жордана: во-первых, она вещественная, во-вторых, алгоритм ее получения – рационален (использует только операции сложения и умножения). В то же время

применять обе формы в практических вычислениях нужно с осторожностью, поскольку они весьма чувствительны к погрешностям округлений.

Задачи и упражнения

1. В 2005 г. правила игры «Кто хочет стать миллионером» с учетом инфляции были несколько изменены по сравнению с описанными в разд. 4.2.1, и стоимость туров стала следующей: 500, 1000, 2000, 3000, **5000**, 10000, 15000, 25000, 50000, **100000**, 200000, 400000, 800000, 150000, 3000000 рублей. Постройте графики стоимости туров в обычном и полулогарифмическом масштабах, отметив на них крестиками «несгораемые» суммы 5000, 100000 и ноликами – остальные точки.

2. Эллипс задан уравнением $x^2 + xy + y^2 = 25.$ Предложите пять способов получения его графика в MATLAB.

Указание. Вот несколько возможных вариантов построения графика:

- по точкам (задавая *x* и рассчитывая *y* как корни квадратного уравнения);
- поворачивая на 45^o эллипс, задаваемый параметрическими уравнениями $x = a\cos t,\ y = b\sin t$, где a, b – длины полуосей эллипса (они находятся через собственные числа λ_1, λ_2 матрицы квадратичной формы A=[1 .5;.5 1]/25 по формулам $a = 1/\sqrt{\lambda_1}$, $b = 1/\sqrt{\lambda_2}$;
- находя с помощью команда **chol** разложение Холецкого A= R^TR матрицы A и выполняя аффинное преобразование единичной окружности с помощью матрицы R;
- находя матрицу H собственных векторов матрицы A и выполняя аффинное преобразование единичной окружности с ее помощью;
- применяя команду **contour** к поверхности $z = x^2 + xy + y^2$;
- для двоечников: наберите команду ezplot('x*x+x*y+y*y-25')

3. Требуется разделить число 8 на две части так, чтобы произведение их произведения на разность было максимальным. Рассмотрите три варианта решения этой задачи в MATLAB и сравните результаты.

Указание. Обозначив искомые части через *x, y,* получаем задачу на условный экстремум $xy(x-y) \rightarrow \max$ при ограничении $x+y=8$.

Ответ: $x = 4(1 + \frac{\sqrt{3}}{3}),\ \ y = 4(1 - \frac{\sqrt{3}}{3}).$

4. Дана матрица A=[a b; c 0], где *a, b, c* – число букв в Вашей фамилии, имени, отчестве. Найти три матричные нормы, соответствующие им числа обусловленности, а также векторы X_1, X_2, X_3, на которых достигаются матричные нормы.

Указание. Для определения векторов X_i использовать равенства $\|A\| = \|AX\| / \|X\|$.

5. Для приведенных ниже матриц найти матричную экспоненту e^{At}:

$$A_1 = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}.$$

Ответ:

$$e^{A_1 t} = \begin{bmatrix} 2e^{-t} - e^{-2t} & e^{-t} - e^{-2t} \\ -2e^{-t} + 2e^{-2t} & -e^{-t} + 2e^{-2t} \end{bmatrix}. \qquad e^{A_2 t} = \begin{bmatrix} e^{2t} & te^{2t} & 0 \\ 0 & e^{2t} & 0 \\ 0 & 0 & e^{2t} \end{bmatrix}. \qquad e^{A_3 t} = \begin{bmatrix} e^{2t} & te^{2t} & \frac{t^2}{2}e^{2t} \\ 0 & e^{2t} & te^{2t} \\ 0 & 0 & e^{2t} \end{bmatrix}.$$

6. Найти жорданову и фробениусову канонические формы следующих матриц: A1=[1 2 3;2 4 6;3 6 9]; A2=diag(1:4]); A3=ones(3); A4=ones(4); A5=triu(ones(4));

МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКИХ СИСТЕМ

Краткие сведения о методике моделирования линейных динамических систем в MATLAB были изложены в разделах 2, 3. В данном разделе рассматривается более широкий круг моделей, включая линейные системы с несколькими входами и выходами, дискретные модели и нелинейные дифференциальные уравнения.

Канонические формы линейных систем

Изменение базиса в пространстве состояний

В разд. 2.1. отмечалось, что одному и тому же объекту S с передаточной функцией Q(*p*) можно сопоставить несколько эквивалентных реализаций в пространстве состояний. Если известна одна из таких реализаций

$$\dot{X} = AX + bu, \quad y = cX,$$

то другие могут быть получены невырожденной заменой переменных типа X=TZ, что приводит к преобразованию матриц системы по формулам $\tilde{A} = T^{-1}AT$, $\tilde{b} = T^{-1}b$, $\tilde{c} = Tc$.

В MATLAB для этого служит команда **ss2ss** (читается *state space to state space*). Ее синтаксис имеет вид `sys1=ss2ss(sys,T)`, где `sys` – исходная система, `T` – матрица преобразования).

Такое преобразование, связанное с изменением базиса в пространстве состояний, называется преобразованием подобия, оно не меняет передаточную функцию системы. Варьируя матрицу преобразования T, можно получить бесчисленное множество реализаций одной и той же передаточной функции в пространстве состояний.

При компьютерном моделировании возникает вопрос, какой из реализаций отдать предпочтение. От ответа на него зависит сложность программы, точность моделирования и другие характеристики вычислительного процесса. На практике обычно предпочитают реализации, отличающиеся простотой математического описания и регулярной структурой. Последнее обеспечивается переходом от исходной ss-модели с помощью замены переменных X=TZ к такой системе координат, что большинство элементов матриц $\tilde{A}, \tilde{b}, \tilde{c}$ становятся равными нулю или единице.

Такие описания называются каноническими представлениями динамических систем или каноническими формами. С инженерной точки зрения канонические формы – это модели исходной системы, отличающиеся простой структурой и минимальным числом варьируемых параметров. Совокупность канонических форм можно рассматривать, как набор типовых моделей динамических систем. Знание свойств этих моделей и их характеристик позволяет выбирать реализации, удобные для решения конкретных задач моделирования, анализа или синтеза систем.

Рассмотрим три типа канонических форм линейных моделей, которые поддерживаются MATLAB – модальные, сопровождающие и сбалансированные канонические формы.

Модальная и сопровождающая канонические формы

В тулбоксе CONTROL имеется команда **canon**, которая строит модальную (жорданову) и сопровождающую канонические формы линейных систем. Ее первый входной аргумент – исходная система, второй – тип канонической формы (*'modal'* или *'companion'*). Выходной аргумент может быть один (результирующая система) или два (возвращается также матрица преобразования T).

У системы в *модальной канонической форме* матрица A представлена в жордановом вещественном виде. Представление системы в этом виде приводит к ее декомпозиции на несколько параллельно соединенных независимых схем, отвечающих различным собственным числам. Во многих случаях такое представление облегчает моделирование и анализ устойчивости, управляемости и наблюдаемости.

У системы в *сопровождающей канонической форме* матрицы A имеет фробениусов вид. В ее последнем столбце стоят коэффициенты характеристического полинома системы, на главной поддиагонали стоят единицы, а остальные элементы – нулевые. Если у системы один вход, то матрица B имеет вид единичного орта, а матрица управляемости – единичная.

Заметим, что в теории управления известны еще три сопровождающие формы, это каноническая форма фазовых переменных, идентификационная каноническая форма и каноническая форма с единичной матрицей наблюдаемости. Однако прямых команд для их получения в MATLAB нет.

Проиллюстрируем работу функции **canon** на примере системы с одним входом и одним выходом.

Пример. Найдем модальную каноническую форму следующей системы третьего порядка:

$$\dot{X} = \begin{bmatrix} -4 & 1 & 2 \\ -2 & -1 & 2 \\ -1 & 1 & -1 \end{bmatrix} X + \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} u, \; y = [1 \; 0 \; 1] X.$$

Формируем ss-модель и применяем команду **canon** с опцией 'modal':

```
>> a=[-4 1 2;-2 -1 2;-1 1 -1]; b=[2;1;0]; c=[1 0 1]; sys=ss(a,b,c,0);
>> sysM=canon(sys,'modal'); sysM.a, sysM.b, sysM.c

                sysM.a                     sysM.b              sysM.c
   -1.0000        0          0            -1.7321        1.1547   0.7071   1.4142
      0        -3.0000       0             2.8284
      0           0       -2.0000          1.4142
```

Мы получили модальную реализацию, матрица A в ней диагональна.

Теперь построим сопровождающую каноническую форму той же системы:

```
>> [sysC,T]=canon(sys,'compan'); sysC.a, sysC.b, sysC.c, T, ctrb(sys)

      sysC.a          sysC.b     sysC.c              T                      ctrb(sys)
  0     0     -6        1      2  -8  24       0.5     0    -3.5          1     0     0
  1     0    -11        0                     -0.75   1.5  -3.25          0     1     0
  0     1     -6        0                     -0.25   0.5  -0.75          0     0     1
```

Теперь матрица A имеет фробениусов вид, вектор B – единичный орт. В качестве второго выходного аргумента фигурирует матрица T замены переменных.

Следует предупредить, что в случае кратных собственных чисел функция **canon** может выдавать некорректный результат, что видно из следующего примера:

```
>> sys=ss([2 0 0;0 2 1;0 0 2],[1;1;1],[1 1 1],0)

          a =              b =          c =
               2  0  0          1  1     1  1  1  1
               0  2  1          2  1
               0  0  2          3  1
```

```
>>sysm=canon(sys,'modal')

a =                b =                c =
   2   0   0             1              1   1  -1
   0   2   0       2.252e+015
   0   0   2       2.252e+015
```

Видим, что в результате преобразования матрица A стала диагональной (это грубая ошибка!), а два элемента матрицы B близки к бесконечности.

Сбалансированное представление

Перейдем к рассмотрению *сбалансированной канонической формы* устойчивых систем, которая строится командой **balreal.** Исторически эта каноническая форма заметно «моложе» других. Она появилась около 25 лет назад и сразу завоевала популярность. Сбалансированное представление (*balanced representation*) обладает рядом интересных свойств. Его прикладное значение определяется возможностью получения редуцированных моделей устойчивых системы путем простого отбрасывания части переменных состояния.

В определении сбалансированной формы системы, характеризуемой матрицами A, b, c, центральную роль играют грамианы ее управляемости и наблюдаемости. Грамианом управляемости W_c и грамианом наблюдаемости W_o устойчивой системы $\dot{X} = AX + bu;\quad y = cX$ называются постоянные симметричные квадратные матрицы, задаваемые формулами

$$W_c = \int_0^\infty e^{At} bb^T e^{A^T t} dt, \qquad W_o = \int_0^\infty e^{A^T t} c^T c e^{At} dt.$$

Невырожденность этих матриц является необходимым и достаточным условием управляемости и наблюдаемости системы. Для их вычисления служит команда **gram**. Ее первый входной аргмент – имя системы, второй – опция 'c' или 'o' (от *controlability* и *observability*).

Грамианы W_c и W_o удовлетворяют матричным уравнениям Ляпунова

$$W_cA^T+AW_c=-bb^T, \qquad W_oA+A^TW_o=-c^Tc.$$

Для решения уравнений Ляпунова в MATLAB существует команда **lyap.**

Представление системы в пространстве состояний $\dot{X} = \overline{A}X + \overline{b}u,\ y = \overline{c}X$ называется сбалансированным, если соответствующие ему грамианы управляемости и наблюдаемости равны и диагональны

$$\overline{W_c} = \overline{W_o} = \mathrm{diag}(\sigma_1,...,\sigma_n).$$

Таким образом, сбалансированное представление характеризуется простым видом матриц W_c, W_o, в отличие от предыдущих канонических форм, где обеспечивался простой вид матрицы A.

Диагональные элементы грамианов $\sigma_1,...,\sigma_n$ называют ганкелевыми сингулярными числами системы, они вещественны и положительны. Все диагональные элементы матрицы A сбалансированного представления отрицательны. Матрицы $\overline{A}, \overline{b}, \overline{c}$ этого представления удовлетворяют условию сигнатурной симметрии $\overline{b} = I\overline{c}^T,\ \overline{A} = I\overline{A}^T I,\ I = diag[i_1,...,i_n],$ где I – диагональная матрица с элементами ± 1 на главной диагонали. При этом количество ее отрицательных элементов равно числу отрицательных слагаемых в разложении передаточной функции системы на простейшие дроби.

Алгоритм вычисления матриц сбалансированного представления основан на одновременной диагонализации грамианов W_c, W_o исходной системы, которые при замене переменных с матрицей T изменяются по формулам конгруентного преобразования

$T^{-1}W_c(T^{-1})^T$, T^TW_oT. Его реализует команда **balreal.** Она может вызываться с одним или несколькими выходными параметрами: sysB=balreal(sys), [sysB, G, T]= balreal(sys). Во втором случае кроме сбалансированного представления системы sysB выводится столбец G ганкелевых сингулярных чисел системы и матрица T перехода к сбалансированному представлению.

Пример. Найдем сбалансированное представление системы sys из предыдущего примера.

```
>> [sysB,G,T]=balreal(sys); sysB.a, sysB.b, sysB.c, G, T

          sysB.a                 sysB.b             sysB.c                 G              T
-0.0356  -1.5276  -0.0329      0.1804     -0.1804  -1.4233  -0.0831     0.4569      0.5121  -0.8439  -1.5360
 1.5276  -3.5063  -0.4072     -1.4233                                   0.2889     -0.7893   0.1554  -0.5514
 0.0329  -0.4072  -2.4581     -0.0831                                   0.0014      0.3732  -0.8295   0.7452
```

Как и ожидалось, матрицы сбалансированного представления сигнатурно-симметричны. Вектор G содержит ганкелевы сингулярные числа системы. Одно из них значительно меньше остальных, следовательно, порядок системы может быть понижен на единицу без заметного изменения ее весовой функции. Косвенным подтверждением такой возможности служит наличие близкого нуля и полюса передаточной функции системы. Это видно из ее нуль-полюсного представления, которое можно найти по команде zpk(sys):

$$ans = \frac{2(s+2.414)(s-0.4142)}{(s+1)(s+2)(s+3)}.$$

Для получения модели пониженного порядка достаточно в системе sysB отбросить третью переменную состояния x_3.

Вычислbv грамианы управляемости и наблюдаемости системы sysB с помощью команд Wc=gram(sysB, 'c'), Wo= gram(sysB, 'o'):

```
Wc =                                   Wo =
   0.4569  -0.0000  -0.0000               0.4569   0.0000   0.0000
  -0.0000   0.2889  -0.0000               0.0000   0.2889   0.0000
  -0.0000  -0.0000   0.0014               0.0000   0.0000   0.0014
```

Оба грамиана равны и диагональны, причем на диагонали стоят элементы вектора G.

Линейные MIMO-модели

Описание MIMO-моделей

До сих пор в пособии рассматривались системы с одним входом и одним выходом, т.е. SISO-модели. Более общий случай – это системы с несколькими входами и выходами, т.е. MIMO-модели (*Multy Input Multy Output Models*). Для их описания по-прежнему можно использовать команды **ss**, **tf** и **zpk**. Если у системы *r* входов, то матрица B ss-модели состоит из *r* столбцов, а если у системы *m* выходов, то матрица C состоит из *m* строк.

Получение временных и частотных характеристик MIMO-систем производится так же, как и ранее, однако формат выходных данных оказывается более сложным. Поясним это на примере системы пятого порядка с тремя входами и двумя выходами, полученной при помощи функции **rss** **(**сокращение от *random space state system***).** Чтобы найти ее весовую функцию и частотные характеристики, набираем текст:

```
>> s=rss(5,2,3); impulse(s), figure, bode(s), figure, nyquist(s).
```

В результате будут построены 18 графиков: 6 весовых функций (от каждого входа до каждого выхода), 6 диаграмм Боде и 6 диаграмм Найквиста.

В общем случае модель системы с r входами и m выходами содержит $r \times m$ отображений $u_i \to y_j$, $i = 1, \ldots, r$, $j = 1, \ldots, m$. Каждое отображение i – го входа на j – й выход можно описать скалярной передаточной функцией $Q_{ij}(p)$. В совокупности они образуют матричную весовую функцию $Q(p)$ размера $m \times r$. Ее можно получить, используя команду tf(s). Для просмотра полученной передаточной функции воспользуемся функцией **tfdata:** [n,d]=tfdata(s). В ответ получаем информацию о структуре передаточной функции:

```
n =                                                  d =
[1x6 double]   [1x6 double]   [1x6 double]           [1x6 double]   [1x6 double]   [1x6 double]
  [1x6 double]   [1x6 double]   [1x6 double]         [1x6 double]   [1x6 double]   [1x6 double]
```

Массивы n и d содержат числители и знаменатели скалярных передаточных функций Q*ij*(*p*). Естественно, имеется возможность посмотреть любую из скалярных передаточных функций *Qij*. Например, для просмотра передаточной функции от третьего входа до второго выхода нужно набрать:

```
>>tf(n{2,3},d{2,3}).
```

При моделировании MIMO-систем с помощью команды **lsim**, управление U подается уже не виде вектора, а в виде матрицы, составленной из векторов u_i, подаваемых на различные входы. Выход Y также является матрицей, составленной из сигналов, снимаемых с различных выходов.

Пример. Рассмотрим систему, заданную матричной передаточной функциейой

$$Q(s) = \begin{bmatrix} \dfrac{1}{s+1} & \dfrac{3}{s^2+3} \\ \dfrac{1}{(s+1)^2} & \dfrac{s-1}{s+2} \end{bmatrix}.$$

Чтобы ввести ее в MATLAB, надо отдельно сформировать массивы числителей и знаменателей:

```
>> N={1,3; 1, [1, -1]};                      % формирование числителей
>> D={[1 1],[1 0 3]; poly([-1,-1]), [1, 2]}; % формирование знаменателей
>> sys=tf(N,D)                               % формирование MIMO-системы с матричной передаточной
                                               функцией Q(s)
```

В ответ получаем сообщение:

```
Transfer function from input 1 to output...      Transfer function from input 2 to output...
      1                       1                          3            s - 1
 #1:  -----              #2:  -------------         #1:  -------   #2:  -----
      s + 1                  s^2 + 2 s + 1              s^2 + 3         s + 2
```

Теперь с этой системой можно проводить компьютерные эксперименты. Подадим на ее первый вход синусоиду, а на второй – косинусоиду и найдем реакцию системы на эти сигналы.

```
>> t=linspace(0,10); u1=sin(t); u2=cos(t); U=[u1;u2], Y=lsim(sys,[U],t); plot(t,Y) ,grid.
```

Результатом будут графики двух выходных сигналов (рис.5.1):

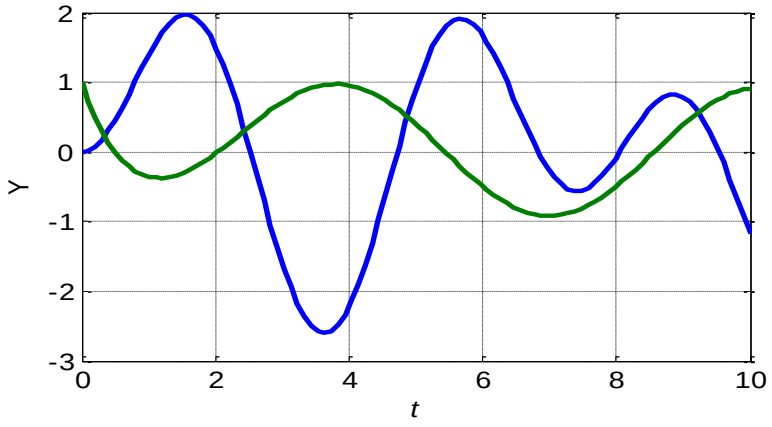


Рис.5.1

При построении частотных характеристик MIMO-систем с использованием команды **freqresp** результат является трехмерным массивом размерностью $m \times r \times W$, где m - число выходов, r - число входов, а W – число отсчетов частоты. Выходные аргументы команд **bode** и **nyquist** имеют такую же размерность, а выходные аргументы команд **impulse** и **step** имеют размерность $T \times m \times r$, где T – количество временных отсчетов.

Анализ минимальности

Вычисление матриц управляемости и наблюдаемости, а также грамианов управляемости и наблюдаемости MIMO-систем по-прежнему производится с помощью команд **ctrb, obsv, gram.** Для анализа минимальности моделей и понижения их порядка используются ранги этих матриц и команды **balreal, minreal, sminreal, modred.** Проиллюстрируем процедуру такого анализа на примере системы девятого порядка.

Пример. Нужно найти передаточную функцию и минимальную реализацию системы, схема которой приведена на рис. 5.2.

Требуется рассмотреть два случая:

1) когда система имеет единственный выход $y = y_1 + y_2 + y_3$;
2) когда система имеет три выхода y_1, y_2, y_3.

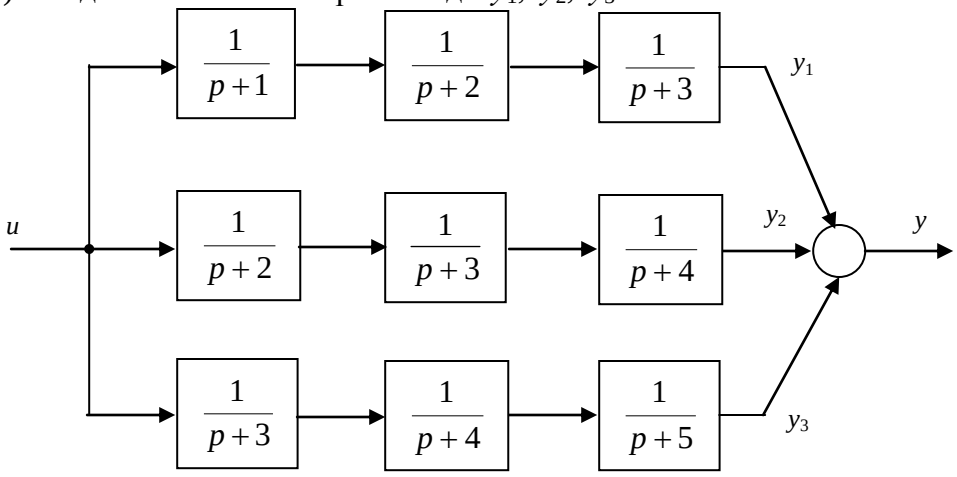


Рис. 5.2

Теоретический анализ. Обозначим передаточную функцию трех параллельных ветвей схемы через $Q_1(p),\ Q_2(p),\ Q_3(p)$:

$$Q_1(p)=\frac{1}{(p+1)(p+2)(p+3)},\quad Q_2(p)=\frac{1}{(p+2)(p+3)(p+4)},\quad Q_3(p)=\frac{1}{(p+3)(p+4)(p+5)},$$

а общую передаточную функцию системы в первом и втором случаях через $q(p)$ и $Q(p)$:

$$q(p)=Q_1(p)+Q_2(p)+Q_3(p),\qquad Q(p)=\begin{bmatrix}Q_1(p)\\Q_2(p)\\Q_3(p)\end{bmatrix}.$$

Поскольку порядок каждой из ветвей равен трем, то общий порядок системы равен 9. Очевидно, что эта реализация неминимальна. Вынесем, например, за скобки передаточной функции $q(p)$ общий множитель $\frac{1}{p+3}$:

$$q(p)=\left(\frac{1}{p+1}\cdot\frac{1}{p+2}+\frac{1}{p+2}\cdot\frac{1}{p+4}+\frac{1}{p+4}\cdot\frac{1}{p+5}\right)\left(\frac{1}{p+3}\right).$$

Такой записи передаточной функции отвечает эквивалентная схема, которая содержит 7 блоков первого порядка, т.е. общий порядок этой реализации равен 7 (на два меньше, чем раньше). Дальнейшего понижения порядка можно добиться, вынося общий множитель $\frac{1}{p+2}$ в двух первых слагаемых либо множитель $\frac{1}{p+4}$ в двух последних. Однако и в этом случае будет неясным, минимальны ли получаемые реализации.

Исчерпывающий ответ на вопрос о порядке минимальной реализации дает классическая декомпозиция Калмана. В соответствии с ней любую систему можно разбить на четыре подсистемы: управляемую и наблюдаемую, управляемую и ненаблюдаемую, неуправляемую и наблюдаемую, неуправляемую и ненаблюдаемую. Обозначим порядки этих подсистем через n_0, n_1, n_2, n_3 соответственно. Для их вычисления надо выписать матрицы управляемости и наблюдаемости

$$R=[B,\ AB,\ \ldots,\ A^{n-1}B],\qquad D=[C^T,\ (CA)^T,\ \ldots,\ (CA^{n-1})^T]^T,$$

а также их произведение H = DR и найти их ранги. Имеют место следующие простые соотношения $rankR=n_0+n_1,\ rankD=n_0+n_2,\ rankH=n_0$. Из них и равенства $n = n_0 + n_1 + n_2 + n_3$ получаем формулы для порядков каждой из подсистем:

$$n_0=rankH,\ \ n_1=rankR-rankH,\ \ n_2=rankD-rankH,\ \ n_3=n+rankH-rankR-rankD.\quad (*)$$

Анализ в MATLAB. Чтобы ввести описание системы, сформируем сначала модели пяти апериодических звеньев, а потом воспользуемся формулами последовательного и параллельного соединения:

```
>>s1=ss(-1, 1, 1, 0);  s2=ss(-2, 1, 1, 0); s3=ss(-3, 1, 1, 0);
>>s4=ss(-4, 1, 1, 0); s5=ss(-5, 1, 1, 0);
>> sys=s1*s2*s3+ s2*s3*s4 +s3*s4*s5;          % система с одним выходом (случай 1)
>> SYS=[s1*s2*s3; s2*s3*s4; s3*s4*s5]         % система с тремя выходами (случай 2)
```

При формировании систем мы использовали знаки умножения для последовательного соединения блоков (при этом передаточные функции перемножаются), знаки сложения для параллельного соединения (при этом передаточные функции складываются) и квадратные скобки для объединения подсистем по входам.

sys.a, SYS.a

```
-1  1  0  0  0  0  0  0  0
 0 -2  1  0  0  0  0  0  0
 0  0 -3  0  0  0  0  0  0
 0  0  0 -2  1  0  0  0  0
 0  0  0  0 -3  1  0  0  0
 0  0  0  0  0 -4  0  0  0
 0  0  0  0  0  0 -3  1  0
 0  0  0  0  0  0  0 -4  1
 0  0  0  0  0  0  0  0 -5
```

sys.b, SYS.b

```
 0
 0
 1
 0
 0
 1
 0
 0
 1
```

SYS.c

```
1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 0
```

sys.c

```
1  0  0  1  0  0  1  0  0
```

Матрицы A, B систем SYS и sys получаются одинаковыми, а матрицы C различаются (число строк равно числу выходов).

Случай 1. Проанализируем сначала систему sys с одним выходом.

Находим ее матрицы управляемости и наблюдаемости и вычисляем их ранги:

```
>>R=ctrb(sys); D=obsv(sys);  H=D*R;
>>r1=rank(R), r2=rank(D), r3= rank(H),
        r1=5, r2=5, r3=4
```

Отсюда по формулам (*) находим размерности подсистем:

$$n_0 = r_3 = 4; \quad n_1 = r_1 - r_3 = 1; \quad n_2 = r_2 - r_3 = 1; \quad n_3 = 9 + 4 - 5 - 5 = 3.$$

Таким образом, каноническая декомпозиция Калмана для системы с одним выходом y содержит управляемую и ненаблюдаемую подсистему порядка n_1= 1, неуправляемую и наблюдаемую подсистему порядка n_2=1, а также неуправляемую и ненаблюдаемую подсистему порядка n_3=3. Порядок минимальной реализации равен n_0 = 4.

Вычисление рангов грамианов управляемости и наблюдаемости дает тот же результат:

```
>> wc=gram(sys,'c'), wo=gram(sys,'o'), r1=rank(wc), r2=rank(wo), r3= rank(wc*wo),
r1=5, r2=5, r3=4
```

Для построения минимальной реализации воспользуемся командой **minreal**:

```
>>msys = minreal(sys);   zpk(msys),
```

Получаем сообщение:

```
 5 states removed                3 (s+3)
Zero/pole/gain:          ----------------------------
                         (s+5) (s+4) (s+2) (s+1)
```

Тот же результат можно получить, сокращая общие множители числителя и знаменателя в нуль-полюсном описания исходной модели:

```
>> zpk(sys)
                3 (s+2) (s+3)^4 (s+4)
          ------------------------------------
          (s+3)^3 (s+4)^2 (s+5) (s+2)^2 (s+1)
```

минимальная реализация

```
          3 (s+3)
     ----------------------------
     (s+4) (s+5) (s+2) (s+1)
```

Случай 2. Перейдем к анализу системы с тремя выходами y_1, y_2, y_3. Находим грамианы управляемости и наблюдаемости и вычисляем их ранги:

```
>> Wc=gram(SYS,'c'), Wo=gram(SYS,'o'),
>>r1=rank(Wc), r2=rank(Wo), r3= rank(Wc*Wo),
```

r1=5, r2=9, r3=5

Отсюда по формулам (*) находим размерности подсистем:

$$n_0 = r_3 = 5; \quad n_1 = r_1 - r_3 = 0; \quad n_2 = r_2 - r_3 = 4; \quad n_3 = 9 + 5 - 5 - 9 = 0.$$

Таким образом, в данном случае каноническая декомпозиция Калмана содержит две подсистемы: неуправляемую и наблюдаемую подсистему четвертого порядка и управляемую и наблюдаемую подсистему пятого порядка. Порядок минимальной реализации равен пяти.

Для ее получения воспользуемся командой **minreal:**

```
>>mSYS=minreal(SYS)
  4states removed.
```

MATLAB сообщает, что порядок системы понижен на четыре и возвращает систему mSYS – минимальную реализацию пятого порядка.

Заметим, что в данном случае из нуль-полюсного описания модели не удается усмотреть ни минимальной реализации, ни ее размерности. Более того, **zpk**-описания исходной и минимальной реализации совпадают:

zpk(SYS), zpk(mSYS)

```
 Zero/pole/gain from input to output 1          Zero/pole/gain from input to output 2          Zero/pole/gain from input to output 3
          1                                              1                                              1
 #1:  ------------------                        #2:  ------------------                        #3:  ------------------
     (s+1) (s+2) (s+3)                              (s+2) (s+3) (s+4)                              (s+3) (s+4) (s+5)
```

Команда **minreal** с двумя выходными параметрами [msys,U] = minreal(sys) дополнительно возвращает ортогональную матрицу преобразования U, что дает возможность найти каноническую декомпозицию Калмана исходной системы, используя формулы U*A*U',U*B,C*U'.

Дискретные модели

Описание дискретных моделей

В MATLAB имеется возможность моделировать линейные системы с дискретным временем. Для описания таких систем используются разностные уравнения различных порядков, модели в пространстве состояний и дискретные передаточные функции.

Линейное разностное уравнение *n*-го порядка имеет вид

$$y_{n+k} + a_{n-1}y_{n+k-1} + \ldots + a_1 y_{k+1} + a_0 y_k = b_{n-1}u_{n+k-1} + \ldots + b_1 u_{k+1} + b_0 u_k.$$

Здесь k – номер такта дискретного времени, u_k – входной сигнал, y_k – выходной сигнал, a_i, b_i – постоянные коэффициенты.

Такими уравнениями, в частности, описываются цифровые фильтры. В ядре MATLAB имеется команда **filter,** позволяющая рассчитать выходной сигнал фильтра по известному входному сигналу, заданному массивом своих значений. Ее входными аргументами являются векторы a и b коэффициентов разностного уравнения, а также массив значений сигнала u: y=filter(b,a,u). С помощью четвертого входного аргумента можно задавать начальные условия фильтра.

Гораздо больше возможностей по синтезу и моделированию цифровых фильтров предоставляет тулбокс SIGNAL, подробнее об этом говорится в конце данного раздела.

Описание дискретной системы в пространстве состояний, как и в случае непрерывных систем, задается четверкой матриц:

$$X_{k+1} = AX_k + Bu_k, \quad y_k = CX_k + Du_k.$$

Здесь $X_k \in R^n$ – вектор состояния; u_k, y_k – входной и выходной сигналы, A, B, C, D – постоянные матрицы.

При исследовании линейных систем с дискретным временем и при решении разностных уравнений широко применяется математический аппарат **z**-преобразования, которое представляет собой дискретный аналог преобразования Лапласа.

Дискретной передаточной функцией линейной системы с входом $x(t)$ и выходом $y(t)$ называется отношение **z**-преобразования выходного сигнала к **z**-преобразованию входного сигнала при нулевых начальных условиях $Q(z) = Y(z)/U(z)$.

Для систем с одним входом и одним выходом она представляет собой отношение двух полиномов

$$Q(z) = \frac{b_{n-1}z^{n-1} + \ldots + b_1 z + b_0}{z^n + a_{n-1}z^{n-1} + \ldots + a_1 z + a_0}.$$

Дискретная передаточная функция системы, заданной описанием в пространстве состояний, может быть найдена по формуле

$$Q(z) = C(zE - A)^{-1}B + D.$$

Дискретные модели во многом похожи на модели с непрерывным временем. Подробнее с ними можно ознакомиться по учебному пособию [4]. В табл. 4.3 приводится сравнение непрерывных и дискретных моделей.

Таблица 4.3

	Непрерывные	Дискретные
Вид уравнений	Дифференциальные уравнения	Разностные уравнения
Описание в пространстве состояний	$\dot{X} = AX + Bu$ $y = CX + Du$	$X_{k+1} = AX_k + Bu_k$, $y_k = CX_k + Du_k$
Описание в виде передаточной функции	$Q(z) = \frac{b_m z^m + b_{m-1}z^{m-1} + \ldots + b_1 z + b_0}{a_n z^n + a_{n-1}z^{n-1} + \ldots + a_1 z + a_0}$, где $1/z$ – оператор задержки	$Q(p) = \frac{b_m p^m + b_{m-1}p^{m-1} + \ldots + b_1 p + b_0}{a_n p^n + a_{n-1}p^{n-1} + \ldots + a_1 p + a_0}$, где $1/p$ – оператор интегрирования
Определение ПФ	$Q(z) = \frac{Y(z)}{U(z)}$	$Q(p) = \frac{Y(p)}{U(p)} = C(pE - A)^{-1}B + D$
Вид интегрального преобразования	Преобразование Лапласа $F(p) = \int_0^\infty f(t)e^{-pt}dt$	z-преобразование $F(z) = \sum_{n=0}^{\infty} \frac{f(n)}{z^n}$
Свободное движение	$X(t) = e^{At}X_0$	$X_k = A^k X_0$
Компоненты свободного движения	$e^{\lambda_i t}$, где λ_i – простой корень характеристического уравнения; $(k_j^0 + k_j^1 t + \ldots + k_j^m t^{m-1}) \cdot e^{\lambda_j t}$, где λ_i – корень кратности m.	z_i^n, где z_i – простой корень характеристического уравнения; $(k_j^0 + k_j^1 n + \ldots + k_j^m n^{m-1}) \cdot z_j^n$, где z_j – корень кратности m.

Тулбокс CONTROL позволяет моделировать линейные системы с непрерывным и дискретным временем. Дискретные модели в нем задаются при помощи тех же конструкторов, что

и непрерывные с той разницей, что последним параметром задается частота дискретизации ts (*sampling time*).

Пример. Рассмотрим дискретную систему, описываемую уравнениями

$$X_{k+1} = \begin{bmatrix} -1 & -1/2 \\ 1/2 & -1 \end{bmatrix} X_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k, \quad y_k = \begin{bmatrix} 0 & 1 \end{bmatrix} X,$$

при шаге дискретизации, равном 0,7.

Создадим соответствующую ss-модель с дискретным временем:

```
>> a=[-1 -1/2 ;1/2 -1];   b=[1;0]; c=[0 1]; d=0;  ts=0.7;  s1=ss(a,b,c,d, ts).
```

На дисплей будут выведены матрицы:

```
a =              b =        c =        d =
   -1    -0.5       1          0   1        0
   0.5   -1         0
```

и сообщение: Sampling time: 0.7 Discrete-time model.

Аналогично создаются tf и zpk-модели с дискретным временем, например:

```
>> num=1; den=[1 1];  s2=tf(num,den, ts);
```

$$s2 = \frac{1}{z+1}$$

```
>> z=[ ]; p=[-1  -2]; k=3;  s3=zpk(z,p,k, ts)
```

$$s3 = \frac{3}{z^2 - z - 2}$$

Обращение ко всем полям дискретной модели осуществляется так же, как и у непрерывной. Время дискретизации хранится в поле ts:

```
>> s3.ts
ans =    0.7000
```

Моделирование дискретных систем

Для моделирования дискретных систем используются те же команды (**step, impulse, initial, lsim**), что и в непрерывном случае. При получении реакции дискретной системы на входное воздействие с помощью команды **lsim** нужно помнить, что вектор временных отсчетов следует задавать, согласуясь с частотой дискретизации или же вовсе опускать.

```
>> u=1./1:10;                          % входной сигнал вида 1  ½  1/3  ….. 1/10.
>> % Правильно:
>> y=lsim(s1,u); y'
ans =       0       0    0.5000       0    0.8750    0.2500    0.9063    0.8750    0.6172    1.6719
>> % Неправильно:
>> t=0:0.1:0.9; y1=lsim(s1,u,t)
??? Error using ==> rfinputs
Time sample spacing must match sample time of discrete-time models.
>> % Правильно:
>> t=0:0.7:(0.7*9); y1=lsim(s1,u,t);
```

Как было сказано, для получения реакции линейной дискретной системы на входной сигнал наряду с командой **lsim** тулбокса CONTROL можно использовать команду **filter** ядра MATLAB**.** Повторим предыдущий пример, используя эту команду:

```
>> [n,d]=tfdata(s1);
```

```
>> yy=filter(n{1},d{1},u)
yy =      0      0   0.5000      0   0.8750   0.2500   0.9063   0.8750   0.6172   1.6719
```

Видим, что результаты совпадают.

При получении весовой функции дискретных систем вместо реакции на дельта-функцию рассматривают реакцию на единичный импульс вида ⎍⎍⎍. Для этого используется та же команда **impulse**. Реакцию на единичный скачок (переходную функцию) получают также при помощи команды **step**. Вместо функции **freqs** для дискретных систем используют функцию **freqz.**

Например, для построения амплитудно-частотной характеристики дискретного звена с передаточной функцией $Q(z) = 1/(z+2)$ нужно набрать:

```
>> [h,w]=freqz(1,[1 2], 200);  plot(w,abs(h)), grid on
```

Любую непрерывную систему путем дискретизации времени можно приближенно преобразовать в дискретную. В Control Toolbox для преобразования непрерывной модели в дискретную и обратно существуют команды **c2d** (читается *continuous to discrete*) и **d2c** (*discrete to continuous*). Входным аргументом является исходная система. В команде **c2d** используется второй входной аргумент – время дискретизации. Допускается использование третьего входного аргумента, позволяющего выбрать один из шести возможных методов преобразования. Для этих же целей служат команды **impinvar** и **bilinear** (Signal Processing Toolbox).

Ряд возможностей по работе с дискретными моделями и решению разностных уравнений содержит тулбокс SYMBOLIC. В нем имеются команды **ztrans** и **iztrans** для выполнения прямого и обратного **z**–преобразований. Например, в результате выполнения команд

```
>>syms t, ztrans(t^2)
```

получим ответ: ans =z*(z+1)/(z–1)^3, а в результате выполнения

```
>>syms a t, ztrans(sin(a*t)),
```

получим ответ: ans =z*sin(a)/(z^2-2*z*cos(a)+1).

Тем самым мы нашли два табличных **z**-преобразования:

$$t^2 \Rightarrow \frac{z(z+1)}{(z-1)^3}, \qquad \sin at \Rightarrow \frac{z\sin a}{z^2 - 2z\cos a + 1}.$$

Рассмотрим еще два примера дискретных моделей.

Пример 1. Задача о банковском вкладе. Клиент открывает счет в банке, внося B рублей. Каждый год он добавляет по b рублей, годовой банковский процент равен k. Какая сумма окажется на его счете через n лет?

Решение. Обозначая текущую величину вклада через y_n, можно записать следующее разностное уравнение:

$$y_{n+1}=y_n(1+k)+b; \quad y_0=\mathrm{B}.$$

Применяя к нему **z**-преобразование $y_n \to \mathrm{Y}(z)$; $y_{n+1} \to z(\mathrm{Y}(z)-\mathrm{B})$, получаем алгебраическое уравнение:

$$z(\mathrm{Y}(z)-\mathrm{B}) = (k+1)\mathrm{Y}(z) + b\frac{z}{z-1}.$$

Выражаем из него Y(z):

$$\mathrm{Y}(z) = z\frac{Bz - B + b}{(z-k-1)(z-1)}.$$

Выполним обратное **z**-преобразование с помощью команды **iztrans:**

```
>> syms z b B k ,  Y=z*(B*z-B+b)/(z-k -1)/( z-1);   y=iztrans(Y)
y =(-b+(k+1)^n*B*k+(k+1)^n*b)/k
```

Таким образом, через n лет на банковском счете окажется сумма $y_n = (\mathrm{B} + \frac{b}{k})(k+1)^n - \frac{b}{k}$.

Возможности тулбоксов CONTROL и SYMBOLIC при работе с дискретными системами дополняют друг друга. Рассмотрим пример на их совместное применение.

Пример 2. Дискретная система задана уравнениями

$$y_{k+1} = x_k$$
$$x_{k+1} = 5x_k - 6y_k + 7u_k.$$

Требуется найти двумя способами (символьным и численным) передаточную функцию от входа u до выхода y и импульсную весовую функцию.

Решение. В данном случае имеем следующие матрицы описания в пространстве состояний

$$X = \begin{bmatrix} y \\ x \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 \\ -6 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 7 \end{bmatrix}, \quad C = [0 \quad 1].$$

Способ 1 (символьный). Дискретную передаточную функцию получаем по формуле $Q = C(zE - A)^{-1}B$, импульсную весовую функцию находим с помощью обратного z-преобразования.

Проделаем эти выкладки в тулбоксе SYMBOLIC. Соответствующая программа имеет вид:

```
>>syms z;
>>A=[0 1;-6 5];B=[0; 7];C=[0 1];          %ввод матриц
>>Q=C*inv(z*eye(2)-A))*B;                 %передаточная функция
>>q=iztrans(Q); Q,q                       %весовая функция
Q=7*z/(z^2-5*z+6),q=-7*2^n+7*3^n          %результат
```

Способ 2 (смешанный). Сначала используем команды тулбокса CONTROL:

```
>> A=[0 1;-6 5];B=[0; 7];C=[0 1];
>>sd=ss(A,B,C,0,1);              % Discrete-time model    Sampling time: 1,       .
>> s=tf(sd)
                      7 z
Transfer function:    ----------
                      z^2 - 5 z + 6
```

Переходим к символьному представлению:

```
>> syms z; q=iztrans(7*z/(z^2-5*z+6))
q = -7*2^n+7*3^n
```

В обоих случаях получаем одинаковый результат: $Q(z) = \frac{7z}{z^2 - 5z + 6}$, $q_n = 7(3^n - 2^n)$.

Расчет аналоговых и цифровых фильтров

В пакете MATLAB предусмотрены средства для цифровой обработки сигналов и синтеза классических аналоговых и цифровых фильтров, таких как фильтры Баттерворта, Чебышёва и эллиптические фильтры.

Аналоговые фильтры. В тулбоксе SIGNAL имеются специальные команды, позволяющие синтезировать аналоговые фильтры и осуществлять обработку сигналов с их помощью. В первую очередь здесь следует назвать команды **buttap, cheb1ap, cheb2ap, ellipap**, предназначенные для построения аналоговых фильтров−прототипов Баттерворта, двух разновидностей фильтров Чебышева и эллиптических фильтров.

Аналоговыми фильтрами-прототипами называют фильтры низких частот с единичной частотой среза. В дальнейшем они могут использоваться для построения других типов фильтров, например, высокочастотных, режекторных и т.д.

Аналоговые низкочастотные прототипы фильтров реализуются с помощью следующих команд:

- **buttap** – прототип фильтра Баттерворта;
- **cheb1ap** – прототип фильтра Чебышева первого типа (равноволновые колебания в полосе пропускания);
- **cheb2ap** – прототип фильтра Чебышева второго типа (равноволновые колебания в полосе подавления);
- **ellipap** – прототип эллиптического фильтра.

Передаточные функции фильтров Баттерворта и Чебышёва первого рода не имеют нулей: $Q(p) = \frac{1}{A_n(p)}$, где $A_n(p)$ – некоторый полином *n*-го порядка. Квадраты их АЧХ $Q(j\omega)Q^*(j\omega)$ имеют вид $\frac{1}{1+\omega^{2n}}$ и $\frac{1}{1+\varepsilon^2 T_n^2(\omega)}$ соответственно, где T_n – полином Чебышева порядка *n*.

Полюсы фильтра Чебышева первого рода расположены в левой половине эллипса на s-плоскости. В пакете MATLAB аналоговый фильтр-прототип Чебышева первого рода рассчитывается с помощью команды [z, p, k] = cheb1ap(n, Rp), где n – порядок фильтра, Rp – уровень пульсаций в полосе пропускания (в децибелах).

Пример. Синтезируем низкочастотный фильтр Чебышева восьмого порядка с единичной частотой среза и единичной величиной пульсаций в полосе пропускания:

```
>> [z,p,k]=cheb1ap(8,1);
```

Построим графики его амплитудно-частотной и фазочастотной характеристик:

```
>>w=0:0.01:4;
>>h=freqs(k*poly(z), poly(p),w);
>>plot(w,abs(h));
>>sys=zpk(z,p,k); nyquist(sys)
```

Оба графика показаны на рис. 5.3

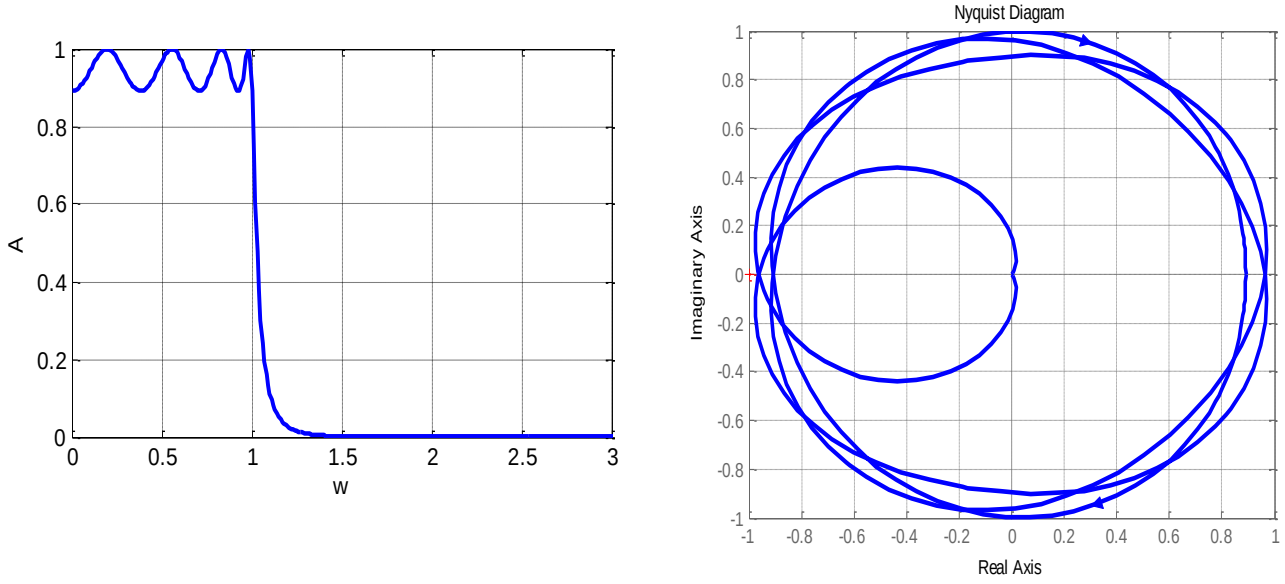


Рис. 5.3

Цифровые фильтры. Если имеется аналоговый фильтр-прототип, то его можно преобразовать в цифровой фильтр с помощью билинейной замены переменных $p = \frac{2}{T}\frac{z-1}{z+1}$. В MATLAB такое преобразование выполняется функцией **bilinear**. Ее синтаксис имеет вид:

[bz, az]= bilinear(b, a, F), где F – частота дискретизации в герцах.

Цифровые фильтры можно строить непосредственно, не опираясь на аналоговый прототип. Для этого имеются следующие команды:

butter – синтез цифрового фильтра Баттерворта.
cheby1 – синтез цифрового фильтра Чебышева первого типа.
cheby2 – синтез цифрового фильтра Чебышева второго типа.
ellip – синтез цифрового эллиптического фильтра.

Число входных аргументов этих команд колеблется от трех (у фильтра Баттерворта) до пяти (у эллиптического фильтра).

Перечисленные команды позволяют рассчитывать не только цифровые, но и аналоговые фильтры, для них в качестве последнего входного параметра надо указать опцию 's'. Например, по команде butter(n, w0, type, ' s') будет построен аналоговый фильтр Баттерворта порядка n с частотой среза ω_0.

Широкий набор средств для моделирования линейных и нелинейных дискретных систем предоставляет SIMULINK. Он содержит специальную библиотеку дискретных элементов, которая поддерживает все виды описания дискретных систем, а также допускает возможность построения и моделирования гибридных систем, содержащих непрерывные и дискретные элементы.

Решение дифференциальных уравнений

Решение задачи Коши

Задачей Коши называется задача о решении обыкновенного дифференциального уравнения с известными начальными условиями. В MATLAB имеются три возможности для решения задачи Коши, не считая моделирования в SIMULINK.

Первая из них касается численного решения линейных дифференциальных уравнений с известной правой частью или систем таких уравнений. Оно может быть выполнено с помощью команды **lsim** (для решения однородных уравнений достаточно команды **initial**).

Вторая возможность – аналитическое решение линейных и простых нелинейных дифференциальных уравнений с помощью решателя **dsolve** тулбокса SYMBOLIC.

Пример. Пусть требуется решить линейное уравнение второго порядка

$$\ddot{y} + a^2 y = 0.$$

В командной строке набираем:

```
>>y=dsolve('D2y+a^2*y=0'),
```

MATLAB выдает ответ: y=C1*sin(a*t)+C2*cos(a*t).

Это означает, что общее решение данного дифференциального уравнения имеет вид $y = C_1 \sin at + C_2 \cos at.$

Задав начальные условия $y_0 = 2,\ \dot{y}_0 = a,$ получаем задачу Коши. Для ее решения набираем:

```
>>y=dsolve('D2y+a^2*y=0, y(0)=2, Dy(0)=a').
```

Получаем ответ: y = sin(a*t)+2*cos(a*t).

Аналогично решатель **dsolve** применяют для систем дифференциальных уравнений.

Пример. Требуется решить систему линейных дифференциальных уравнений

$$\dot{u} = 2v, \quad \dot{v} = 2w, \quad \dot{w} = -2u$$

с начальными условиями $u_0 = v_0 = 0, \quad w_0 = 3.$

В командной строке набираем:

```
>> S=dsolve('Du=2*v, Dv=2*w, Dw=-2*u','u(0)=0, v(0)=0, w(0)=3').
```

MATLAB выдает сообщение о структуре решения:

```
S =    u: [1x1 sym]    v: [1x1 sym]    w: [1x1 sym]
```

Для доступа к полям структуры S набираем:

```
>>u = S.u, v = S.v,  w = S.w
```

MATLAB выдает ответ:

```
u=exp(-2*t)+3^(1/2)*exp(t)*sin(3^(1/2)*t)-exp(t)*cos(3^(1/2)*t)
v =-exp(-2*t)+3^(1/2)*exp(t)*sin(3^(1/2)*t)+exp(t)*cos(3^(1/2)*t)
w =exp(-2*t)+2*exp(t)*cos(3^(1/2)*t).
```

Для записи этих формул в более удобном виде можно воспользоваться командой **pretty**. Еще лучше перенести их через клипборд в пакет MAPLE, где они принимают вид:

$$u = \mathbf{e}^{(-2\,t)} + \sqrt{3}\ \mathbf{e}^{t} \sin(\sqrt{3}\ t) - \mathbf{e}^{t} \cos(\sqrt{3}\ t)$$
$$v = -\mathbf{e}^{(-2\,t)} + \sqrt{3}\ \mathbf{e}^{t} \sin(\sqrt{3}\ t) + \mathbf{e}^{t} \cos(\sqrt{3}\ t)$$
$$w = \mathbf{e}^{(-2\,t)} + 2\ \mathbf{e}^{t} \cos(\sqrt{3}\ t)\ .$$

Решим ту же систему, заменив начальные условия краевыми $u(0) = 1, \ v(0) = -1, \quad w(1) = e^{-2}$:

```
>> S=dsolve('Du=2*v, Dv=2*w, Dw=-2*u', 'u(0)=1, v(0)=-1, w(1)=exp(-2)');
>>u = S.u, v = S.v,  w = S.w
```

Получаем ответ: u =exp(-2*t), v =-exp(-2*t), w =exp(-2*t).

Третья возможность – численное решение нелинейных дифференциальных уравнений с помощью команд типа **ode23** и **ode45.** Буквенная часть названия этих команд – сокращение от *Ordinary Differential Equation*, цифры указывают порядок используемой версии метода Рунге-Кутта. Команда **ode45** дает более точное решение, но требует больше времени. Основная модификация команды **ode23** имеет вид `[t, X] = ode23('fun', [T0 T1], X0)`. Она обеспечивает решение системы дифференциальных уравнений, записанных в форме Коши

$$\dot{X} = F(X, t), \ X(T_0) = X_0$$

на интервале времени $T_0 \le t \le T_1$.

Результатом ее выполнения является массив отсчетов времени t и соответствующий им массив значений X. Для того чтобы команда была выполнена, надо предварительно составить программу для вычисления вектор-функции $F(X,t)$, стоящей в правой части дифференциального уравнения. Эта программа должна быть оформлена в виде *m*-файла, которому присваивается любое имя, например, 'fun'.

Пример. Воспользуемся командой **ode23** для моделирования нелинейного уравнения

$$\ddot{y} + \sin y = 0{,}8\, y^3, \quad y(0) = 1, \quad \dot{y}(0) = 0.$$

Перепишем его в виде системы двух уравнений, обозначив $x_1 = y, \quad x_2 = \dot{y}$:

$$\dot{x}_1 = x_2,$$
$$\dot{x}_2 = 0{,}8\, x_1^{\,3} - \sin x_1, \qquad X_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Функцию для вычисления правых частей оформляем в виде *m*-файла с именем **fun**:

```
function F = fun (t, X)
F = [X(2); .8*X(1)^3-sin(X(1))];
```

Далее задаем численные значения параметров

```
>>T0 = 0, T1 = 20, X0 = [1; 0];
```

после чего уже может быть выполнена основная команда

```
>> [t, X] = ode23('fun', [T0 T1], X0).
```

Ее результатом будут одномерный массив времени t на интервале от 0 до 20 секунд и двумерный массив X, содержащий значения $x_1(t)$, $x_2(t)$. Как правило, шаг времени – переменный. График результата может быть получен командой plot(t, X). Для последующего перехода к равномерной сетке времени (если это необходимо), можно использовать команду **interp1**:

```
>>tt = 0:0.1:20;  XX  = interp1(t, X, tt, 'spline');
```

результатом которой будет массив переменных XX для равноотстоящих моментов времени tt.

Другая возможность получения равномерного шага связана с использованием команды **deval** . Для этого слегка изменим синтаксис команды **ode23**:

```
>> SOL = ode23(@fun,[T0 T1], X0)
```

Теперь результатом будет структура SOL, о чем свидетельствует сообщение:

```
SOL =    solver: 'ode23'   extdata: [1x1 struct]      x: [1x90 double]      y: [2x90 double]
```

Структура SOL имеет поля x и y. Поле x содержит набор отсчетов времени, а поле y – массив значений вектора $X(t)$. Доступ к полям производится командами SOL.x и SOL.y

Первый аргумент команды **deval** – структура SOL, а второй – вектор точек, в которых нужно вычислить аппроксимацию решения.

Задаем равномерную сетку по времени и строим график:

```
>>t=0:.1:20; y = deval(SOL,t); figure(2), plot(t,y,'o')
```

Результат приведен на рис. 5.4.

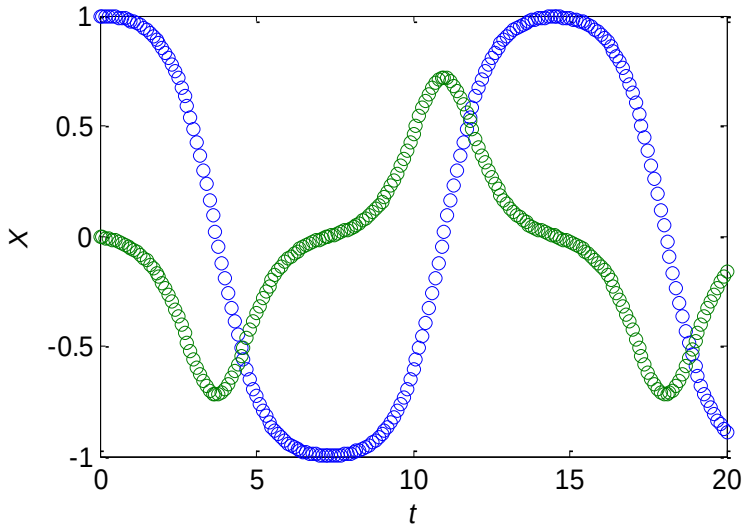


Рис. 5.4

В качестве дополнительного аргумента в команде **ode23** можно указать требуемую точность решения *eps* (по умолчанию *eps* = 0.001). Команда **ode45** выполняется аналогично и имеет такие же модификации. Обе команды можно использовать для моделирования нелинейных систем

автоматического управления, если в правой части дифференциальных уравнений учесть управляющее воздействие, задавая его как явную функцию времени.

В MATLAB существуют и другие решатели дифференциальных уравнений, например, **ode15s, ode23s, ode23t**. Команды с буквой *S* предназначены для моделирования жестких (*Stiff*) дифференциальных уравнений, буква *T* указывает на использование метода трапеций. Большинство из них способны решать и дифференциально-алгебраические системы уравнения вида $M(X,t)\dot{X} = F(X,\ t)$, где M – матрица, F – вектор-функция.

Решение краевых задач

В предыдущем пункте речь шла о решении задачи Коши, когда заданы дифференциальные уравнения и начальные условия. Значительно большую трудность представляют краевые задачи, когда задаются начальные и конечные условия или некоторые их комбинации.

Решение краевых нелинейных дифференциальных уравнений в MATLAB выполняется с помощью команды **bvp4c** (читается *boundary value problem for continous*). Она решает двухточечную краевую задачу для обыкновенных дифференциальных уравнений методом коллокаций.

Вызов в формате sol=bvp4c (F, G, solinit) интегрирует систему обыкновенных дифференциальных уравнений вида $Y' = F(x,Y)$ на интервале $[a,b]$, с краевыми условиями вида $G(Y(a),Y(b)) = 0$. В качестве входных параметров команда **bvp4c** принимает три аргумента – имя функции F для формирования правых частей дифференциального уравнения $F(x,Y)$, имя функции G для вычисления вектора краевых условий $G(Y(a),\ Y(b))$ и структуру solinit, содержащую начальное приближение решения.

Пример. Продемонстрируем смысл входных параметров решателя **bvp4c** и вид возвращаемого результата на стандартной демо-функции **twobvp.** В ней требуется найти решения дифференциального уравнения $y'' + |y| = 0$, которые удовлетворяют краевым условиям $y(0) = 0,\ y(4) = -2$.

Представим исходное уравнение в виде $Y' = F(x,Y)$. Введем обозначения $y_1 = y,\ y_2 = \dot{y},\ Y = [y_1 \quad y_2]^T$. Тогда уравнение можно записать как

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = -|y_1| \end{cases}, \quad \text{т.е.} \quad \dot{Y} = \begin{bmatrix} y_2 \\ -|y_1| \end{bmatrix}.$$

Правая часть этого уравнения формируется в функции **twoode**, она не зависит явно от x.

На каждом шаге итерационного алгоритма, решающего краевую задачу, проверяется выполнение краевых условий (*boundary conditions*). Для этого значения переменных в начальной и конечной точках $Y(a)$ и $Y(b)$ подаются на вход специальной функции, вычисляющей вектор $G(Y(a),Y(b))$. В нашем случае это вектор $[y(0) \quad y(4)+2]^T$. Он зависит только от первых компонент векторов $Y(a)$ и $Y(b)$, так как в краевые условия не входят производные, и вычисляется в функции **twobc**. Тексты функций **twoode** и **twobc** приведены ниже:

```
function dydx = twoode(x,y)
dydx = [ y(2); -abs(y(1)) ];
```

```
function res = twobc(ya,yb)
res = [ ya(1); yb(1) + 2 ];
```

Структура solinit имеет поля x и y. Поле x содержит набор отсчетов независимой переменной, а поле y – массив значений вектора $Y(x)$. Их начальные значения можно задавать вручную, или при помощи специальной команды **bvpinit**. Первым аргументом этой команды является набор

отчетов x. Второй аргумент – либо строка констант, которыми заполняется весь массив y, либо имя функции fun, возвращающей строку значений в точке x. Проиллюстрируем обе эти возможности:

```
>> solinit = bvpinit([0 1 2 3 4],[1 0])
solinit =     x: [0 1 2 3 4]
              y: [2x5 double]
>> solinit.y
ans =    1    1    1    1    1
         0    0    0    0    0
```

```
function y= fun (x)
   y=[x^2, x-1];
>> solinit = bvpinit([0 1 2 3 4],@ fun)
solinit =     x: [0 1 2 3 4]
              y: [2x5 double]
>> solinit.y
ans =    0    1    4    9   16
        -1    0    1    2    3
```

Теперь сформированы все вспомогательные функции и решатель **bvp4c** готов к работе. Последовательность команд

```
>>solinit = bvpinit([0 1 2 3 4],[1 0]); sol = bvp4c(@twoode,@twobc,solinit);
```

решает двухточечную краевую задачу на интервале $[0, 4]$.

В результате выполнения команды **bvp4c** возвращается структура sol, содержащая поля x, y, yp (вектор производных) и solver (имя решателя, в данном случае – bvp4c). График решения можно построить по команде plot(sol.x,sol.y), он показан на рис. 5.5 (верхняя кривая).

Для формирования начальной сетки по x мы использовали целочисленные точки [0 1 2 3 4], а начальный вид функции y и ее производной задается вектором [1 0], т.е. перед началом итераций функция y была равна единице, а ее производная – нулю: $y(x) = 1,\ y'(x) = 0$.

Изменим знак начальных значений функции на противоположный, приняв $y(x) = -1$, $y'(x) = 0$, и вновь найдем решение:

```
>>solinit = bvpinit([0 1 2 3 4],[1 0]);
>>sol = bvp4c(@twoode,@twobc,solinit);
```

Оно будет существенно отличаться от первого (нижняя кривая на рис. 5.5). Этот пример показывает, что разные начальные приближения могут привести к различным решениям.

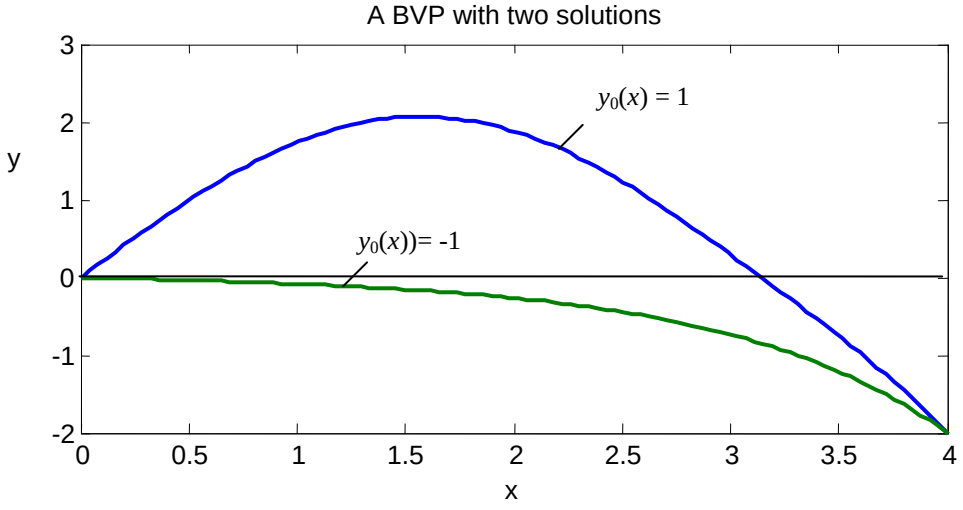


Рис. 5.5

Заметим, что команда **bvp4c** при решении использует переменный шаг. В этом можно убедиться, набрав команду plot(sol.x). Для получения аппроксимации решения в конкретных точках можно использовать уже упоминавшуюся команду **deval.** Первый ее аргумент – структура sol, а второй – вектор точек, в которых нужно вычислить аппроксимацию. Так, по командам xint = linspace(0,4); yint = deval(sol,xint); вычисляется решение в 100 точках, равномерно расположенных на

интервале [0 4]. График первой компоненты решения строится с помощью команды plot(xint,yint(1,:)).

Оба эти решения можно получить аналитически, заменяя нелинейное уравнение $y'' + |y| = 0$ двумя линейными уравнениями

$$\begin{cases} y'' + y = 0 & \text{при} \quad y > 0 \\ y'' - y = 0 & \text{при} \quad y < 0 \end{cases}.$$

Их общие решения имеют вид $y(x) = C_1 \sin x + C_2 \cos x$ и $y(x) = C_3 e^t + C_4 e^{-t}$, соответственно. Для получения решения при краевых условиях $y(0) = 0,\ y(4) = -2$ необходимо рассмотреть два случая, в зависимости от знака производной $y'(0)$.

Случай 1. $y'(0) < 0$, тогда $y(x) = C_3 e^x + C_4 e^{-x}$. Коэффициенты C_3, C_4 находим из краевых условий $y(0) = 0$, $y(4) = -2$: $C_3 e^0 + C_4 e^{-0} = 0,\ C_3 e^4 + C_4 e^{-4} = -2$. Имеем $C_3 = -\dfrac{2e^4}{e^8 - 1}, C_4 = \dfrac{2e^4}{e^8 - 1}$. Отсюда $y(x) = -2\dfrac{\operatorname{sh} x}{\operatorname{sh} 4}$. При любых $x > 0$ это решение отрицательно. Таким образом, нижняя кривая на рис. 5.5 – перевернутый гиперболический синус.

Случай 2. Пусть $y'(0) > 0$, тогда $y(x) = C_1 \sin x + C_2 \cos x$, и из условия $y(0) = 0$ имеем $C_2 = 0,\ y(x) = C_1 \sin x$. Производная этого решения $y'(x) = C_1 \cos x$. Решение обращается в ноль при $x = \pi$. После этого оно становится отрицательным и принимает вид $\tilde{y}(x) = C_3 e^x + C_4 e^{-x}$. Из условий $\tilde{y}(4 - \pi) = -2$ и $y(0) = 0$ находим константы $C_3 = -\dfrac{1}{\operatorname{sh}(4 - \pi)}, C_4 = -C_3$. Производная этой функции имеет вид $\tilde{y}'(x) = -\dfrac{e^x + e^{-x}}{\operatorname{sh}(4 - \pi)}$. Из равенства $\tilde{y}'(0) = y'(\pi)$ находим $C_1 = \dfrac{2}{\operatorname{sh}(4 - \pi)}$. Таким образом, верхняя кривая на рис. 5.5 состоит из двух частей:

$$y(x) = 2\frac{\sin x}{\operatorname{sh}(4 - \pi)} \quad \text{при} \quad 0 \le t \le \pi; \quad y(x) = -2\frac{\operatorname{sh}(x - \pi)}{\operatorname{sh}(4 - \pi)} \quad \text{при} \quad \pi < 0 \le 4.$$

Обе части гладко сопрягаются в точке π на оси абсцисс, где у них совпадают не только значения, но и производные.

Моделирование в SIMULINK

Необходимые начальные сведения о работе в SIMULINK были приведены в разд. 3. В этом параграфе описываются некоторые дополнительные возможности SIMULINK по составлению и анализу схем моделирования, а также его взаимодействию с MATLAB.

Редактор дифференциальных уравнений DEE

Эффективным средством для решения нелинейных дифференциальных уравнений с известными начальными условиями является моделирование в SIMULINK. Оно требует построения эквивалентной структурной схемы и ее реализации на стандартных линейных и нелинейных блоках. Определенную помощь в этом процессе может оказать редактор дифференциальных уравнений DEE (*Differential Equation Editor*)**.** Этот редактор сам строит схему моделирования для SIMULINK по системе дифференциальных уравнений, записанной в форме

Коши. Это немного эффективней, чем "ручное" построение схемы на интеграторах, и несколько проще, чем написание блока «с нуля».

Редактор вызывается командой **dee**. Появляется окно с несколькими блоками, один из них называется **Differential Equation Editor**, его надо скопировать (перетащить мышкой) на рабочую страницу SIMULINK, войти в него двойным щелчком мыши и ввести параметры дифференциального уравнения (левые части, начальные условия и др.).

Приведем пример заполнения параметров для моделирования дифференциального уравнения математического маятника

$$\ddot{y} + \sin y = 0, \quad y(0) = \dot{y}(0) = 0,$$

записанного в форме Коши:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\sin x_1$$

Name:		pendulum
# of inputs		0
First order equations, f(x,u):		x0
dx/dt=	x(2) -sin(x(1))	1 0
Output equations, f(x,u)		
y=	x(1) x(2)	

В дальнейшем входить в блок для изменения параметров можно с помощью команды **diffeqed**.

Схема моделирования приведена слева на рис. 5.6. Чтобы одновременно наблюдать графики двух сигналов в блоке **Scope** в параметрах осциллографа устанавливаем поле ”Number of axes” равным двум. Параметры осциллографа можно редактировать, нажав кнопку  . Если есть желание взглянуть непосредственно на схему моделирования, соответствующую заданному уравнению, надо, выделив блок, нажать правую кнопку мыши и выбрать пункт меню *look under mask*, при этом появится схема, показанная справа на рис. 5.6.

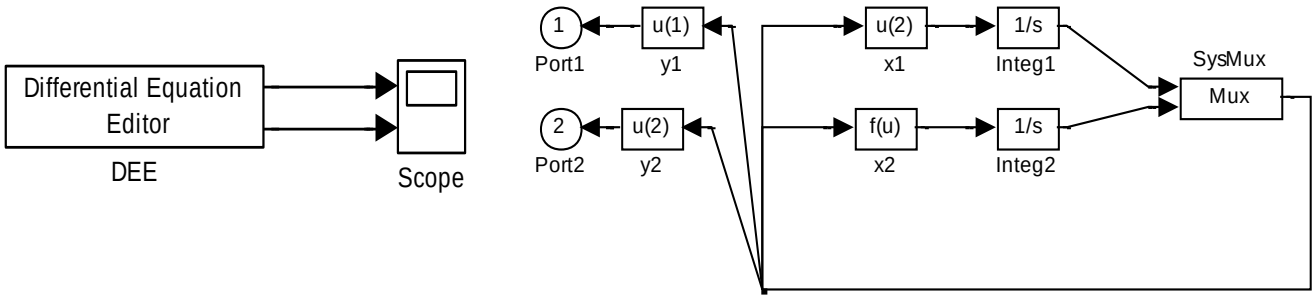


Рис. 5.6

Анализ Simulink-моделей

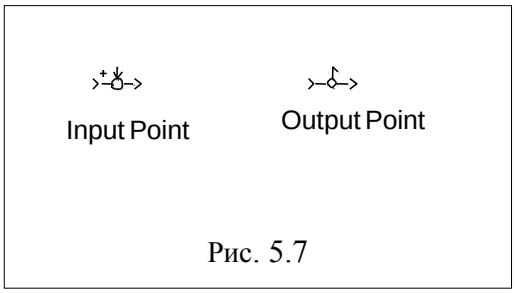


Рис. 5.7

Для получения графиков временных или частотных характеристик схемы, нарисованной в SIMULINK, и построения диаграммы нулей и полюсов ее передаточной функции, используется команда **Linear Analysis** из пункта Tools меню, расположенного вверху рабочей страницы SIMULINK. Ответ выдается в виде диаграмм средства LTIview. Предварительно на схеме надо пометить вход и выход, пользуясь метками входных и выходных точек **Input Point, Output Point** из библиотечки Control_System_Toolbox (в ранних версиях – Model_Inputs_and_Outputs) (см. рис. 5.7).

Для модели (в общем случае – нелинейной), реализованной в SIMULINK, можно получить описание в пространстве состояний. Это делается при помощи команды **linmod.** Поясним ее применение на примере схемы следящей системы, показанной на рис. 5.8, которой мы присвоим имя test_sys.mdl.

Предварительно на схеме надо пометить вход и выход, используя блоки **In** и **Out**. Найдем

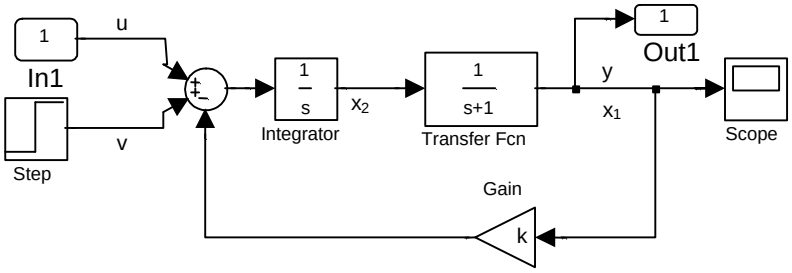


Рис. 5.8

описание схемы в пространстве состояний при k=1. Набирая в командной строке код [A,B,C,D]=linmod('test_ sys'), получим ответ:

```
A =  -1   1   B =    0        C =   1   0       D =   0
     -1   0          1
```

Число выходных параметров команды **linmod** можно брать любым от 0 до 4. Применяя эту команду для нелинейных систем, получим описание в пространстве состояний линеаризованной системы.

В теории управления часто встает задача определения стационарных точек системы, т.е. положений равновесия, при которых все производные равны нулю $\dot{X} = 0$. Для Simulink-моделей эту задачу можно решить при помощи команды **trim** (от *trim* – приводить в порядок; уравновешивать). Входным параметром функции является имя *mdl*-файла, а выходными – координаты точки равновесия в фазовом пространстве X, а также установившиеся значения входного и выходного сигналов u и y. Возвращаемое значение dX в случае успеха должно быть равно нулю.

Применительно к нашей схеме результатом команды [X,u,y,dX]=trim('test_ sys') будет:

```
X =  0.5       u =  –0.5          y =  0.5000        dX =  0
     0.5                                                   0
```

Действительно, если входной сигнал взять равным u= –0,5, то при y=0,5 и v=1 на выходе сумматора получим нуль, и выходной сигнал интегратора x_2=0,5 не будет изменяться. Производные от переменных x_1, x_2 будут равны нулю.

Изменим значение k, взяв k=2, и вновь выполним команду [X,u,y,dX]=trim('test_cx'). Результат будет иным:

```
X =   0.3333      u =  –0.3333      y =   0.3333      dX =  1.0e-015 *  0
      0.3333                                                          0.1110
```

Теперь стационарное состояние схемы достигается при u= –1/3, x_1= x_2 =1/3.

Для проверки получим те же результаты теоретически. При $k = 1$ схема описывается уравнением $\dot{x}_1 = -x_1 + x_2$, $\dot{x}_2 = -x_1 + u + v$, $y = x_1$, $v = 1$.

Полагая $\dot{x}_2 = 0$, получаем уравнение $x_1 - u = 1$. Это одно уравнение с двумя неизвестными x_1 и u, оно имеет бесчисленное множество решений. Команда trim выдает решение с минимальной нормой x_1 = 0,5; u = 0,5 (оно соответствует псевдообращению матрицы [1 –1]). Остальные переменные получаются из равенств $–x_1 + x_2 = 0$, $y = x_1$, что дает $x_2 = y$ = 0,5.

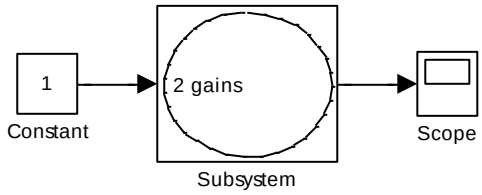
При $k = 2$ схема описывается уравнениями $\dot{x}_1 = -x_1 + x_2$, $\dot{x}_2 = -2x_1 + u + 1$, $y = x_1$.

Полагая $\dot{x}_2 = 0$, получаем уравнение $2x_1 – u = 1$. По команде **trim** получим одно из его решений x_1 = 1/3, u = –1/3 (заметим, что оно не совпадает с решением x_1 = 2/5, u = –1/5, полученным псевдообращением матрицы [2 1]).

Маскирование подсистем в SIMULINK

При моделировании сложных систем, когда Simulink-схема становится слишком большой, бывает полезно перейти к иерархической модели, разбив систему на подсистемы. Выделив мышью группу блоков, можно изобразить их в виде одного блока подсистемы, используя пункт меню *Edit/Create* subsystem. В дальнейшем пользователь может, щелкнув мышью по блоку подсистемы, войти внутрь и редактировать эту подсистему. Иногда такую возможность следует запретить, произведя так называемое «маскирование» системы (*Edit/Mask subsystem*). Содержимое маскированной системы можно просмотреть по команде меню *Edit/Look under mask*. По команде *Edit/Edit mask* открывается редактор масок. Он содержит кнопку «*Unmas*k», позволяющую удалить маску, а также управлять параметрами маскированной подсистемы и процедурой отрисовки блока в редакторе SIMULINK.

Пример. Создадим маскированную подсистему, состоящую из двух последовательно соединенных усилителей. В качестве коэффициентов усиления зададим не конкретные числа, а переменные $g1$ и $g2$. Отредактируем маску так, чтобы, во-первых, изображение блока содержало окружность и надпись «2 Gains» (рис. 5.9, слева) и, во-вторых, чтобы при щелчке мышью по маскированной подсистеме открывалось диалоговое окно, запрашивающее оба коэффициента усиления (рис. 5.9, справа).



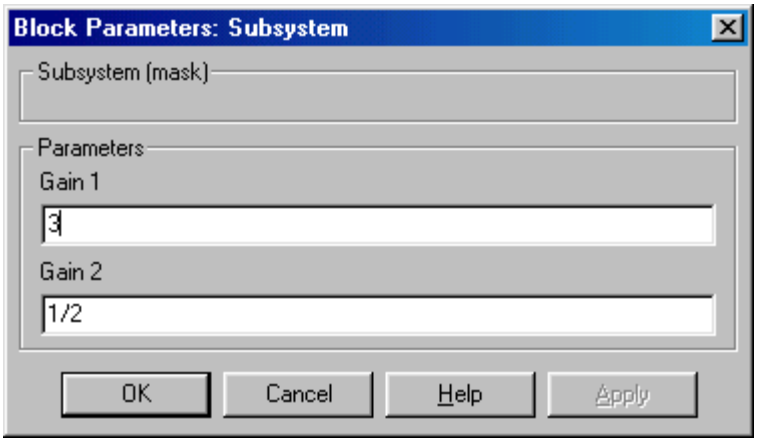


Рис. 5.9

Для решения первой задачи в редакторе масок на панели *Icon* в поле *Draw* commands следует ввести

```
plot( (1+sin(0:0.2:6.2))/2,(1+cos(0:0.2:6.2))/2)
text(0.1,0.5,'2 gains')
```

В списке *Units* выберем *Normalized*, это означает, что при отрисовке размер блока считается равным 1x1.

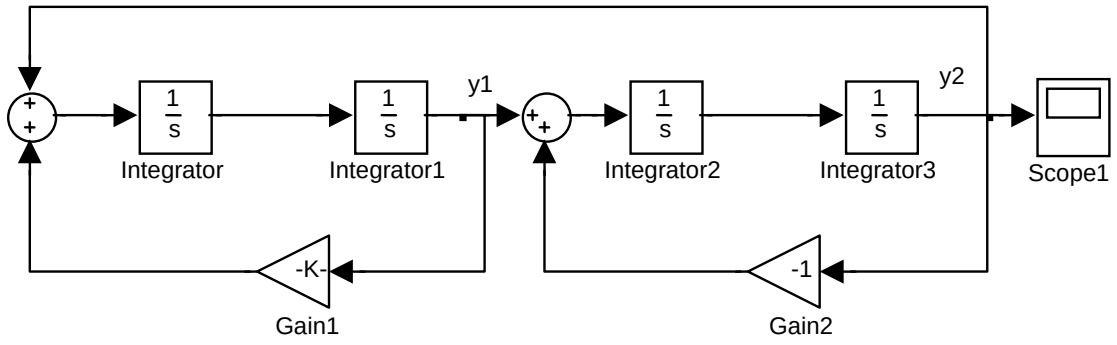
Для решения второй задачи на панели *Parameters* добавим два параметра – g1 и g2 (это имена параметров усилителей) с приглашениями “Gain 1” и “Gain 2”, которые будут отображены на форме.

Пример. Моделирование двойного маятника. Консервативная колебательная система из двух взаимосвязанных маятников описывается уравнениями

$$\ddot{y}_1 + k y_1 + y_2 = 0,$$
$$\ddot{y}_2 + y_1 + y_2 = 0.$$

Схема моделирования этих уравнений в SIMULINK приведена на рис. 5.10, а. Ее левая часть отвечает первому маятнику, правая – второму. Выделяя поочередно эти части и выполняя их маскирование, получаем схему, изображенную на рис. 5.10, б. Она занимает меньше места и наглядно показывает структуру взаимосвязи маятников. Чтобы войти внутрь подсистемы и изменить ее параметры, нужен двойной щелчок мышью.

а)



б)

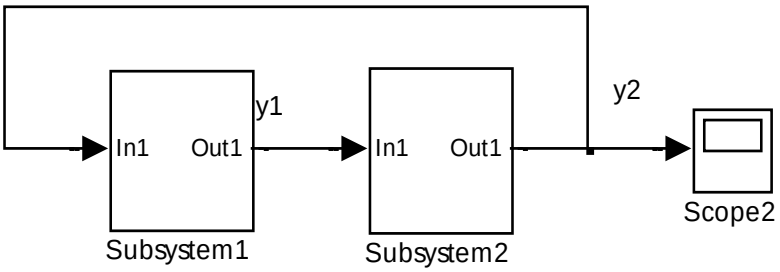


Рис. 5.10

На рис. 5.11 приведен результат моделирования для начальных условий $y_1(0)=0,\quad y_2(0)=1$ и значения k=1.2.

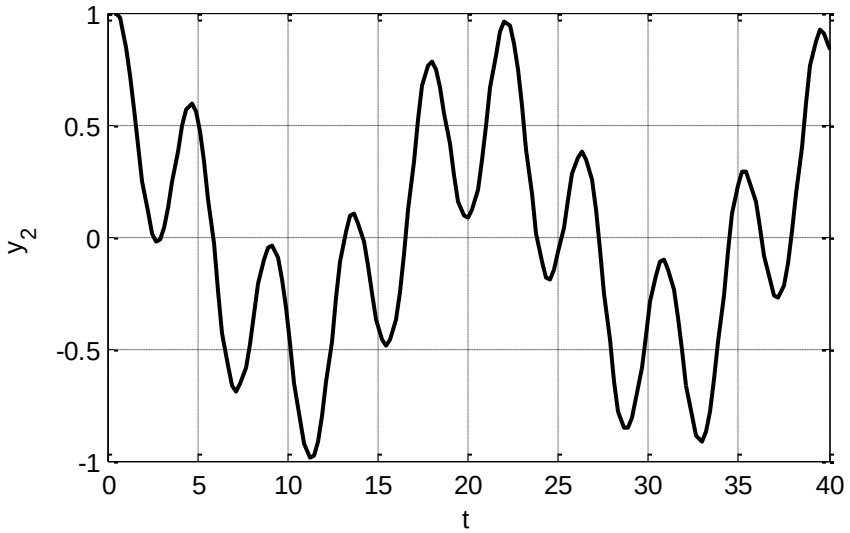


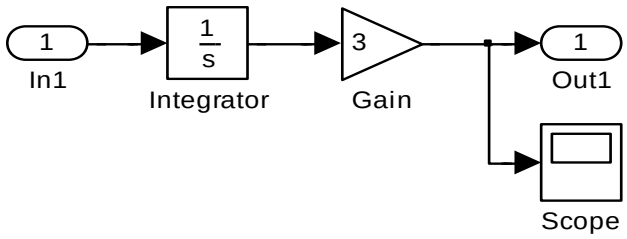
Рис. 5.11

Управление Simulink-моделью из MATLAB

При организации численных экспериментов в SIMULINK может возникнуть необходимость многократно запускать одну и ту же модель с различными параметрами и входными сигналами. Это можно обеспечить, управляя процессом моделирования из командного окна.

Прежде всего, необходимо открыть Simulink-модель вручную или при помощи команды **open**. Так, команда open('vdp') открывает модель vdp.mdl (схема моделирования уравнения Вандерполя). Чтобы запустить процесс моделирования из командной строки используется команда **sim**, например, sim('vdp'). В качестве параметров этой команде можно передать время моделирования, входной сигнал (при наличии входных портов у модели), начальные условия и опции решателя. В этом случае синтаксис ее вызова будет иметь вид y=sim('model',timespan,options,ut), где timespan – интервал моделирования, ut – массив, состоящий из отсчетов времени и значений входного сигнала.

Пример. Создадим модель, показанную на рис. 5.12 и сохраним ее под именем simple_model. Найдем ее реакцию на синусоиду на интервале 30с. Опций моделирования использовать не будем – в качестве аргумента возьмем пустой массив.




```
>>open('simple_model');
>>t=0:0.1:30;
>>sim('simple_model',[0 30],[],[t' sin(t')])

>> [t1,y1]=sim('simple_model',[0 30],[],[t' sin(t')]);

>>plot(t1,y1)
```

Рис. 5.12

После выполнения этих команд можно открыть блок **Scope** в схеме и увидеть результаты моделирования. Поместить результаты моделирования в рабочее пространство и затем посмотреть график с помощью команды **plot** можно, используя вызов **sim** с выходными параметрами.

При вызове команды **sim** с опциями решателя структура опций задается командой **simset.** Для примера изменим тип решателя на ode23:

```
>> [t2,y2]=sim('simple_model',[0 30],simset ('Solver','ode23'),[t' sin(t')]);
```

С полным списком опций моделирования можно ознакомиться, используя команду **simget,** например, o=simget ('simple_model').

Помимо задания опций моделирования, существует возможность программно изменять параметры блоков. Для их чтения и записи используют команды **get_param** и **set_param.** Поскольку названия параметров блоков не всегда совпадают с текстами пояснений в окне параметров, бывает полезна конструкция get_param(gcb, 'objectparameters'). Мышью в окне модели выбирается нужный блок. После этого команда **gcb** возвращает имя текущего блока. Так, если будет выделен блок **Gain**, команда **gcb** вернет строку 'simple_model/Gain'. При помощи той же функции **get_param** можно получить любой конкретный параметр, например,

```
>> p=get_param('simple_model/Gain', 'Gain'), class(p)

p =3                    ans =char
```

Обратите внимание на то, что тип параметра – строка. Это значит, что если, например, захочется сменить коэффициент усиления с 3 на 4 при помощи функции set_param, то придется набрать не set_param('simple_model/Gain', 'Gain',4), а set_param('simple_model/Gain', 'Gain',num2str(4))**!**

Использование этих команд позволяет, в частности, организовать циклическое изменение параметров схемы моделирования (коэффициентов усиления, начальных условий) с помощью оператора MATLAB **for.**

Пример (задача об оптимальных начальных условиях). Движение маятника описывается нелинейным дифференциальным уравнением

$$\ddot{y} + 0{,}3\dot{y} + \sin y = 0.$$

Его начальные условия удовлетворяют условию $y_0^2 + \dot{y}_0^2 = R^2$ (лежат на окружности радиуса R в фазовой плоскости). Требуется найти значения y_0, $\dot{y}_0$, при которых энергия выходного сигнала будет максимальной либо минимальной $E = \int_0^\infty y^2 dt \to \text{extremum}$.

Решение. Составим схему моделирования в SIMULINK и назовем ее sys.mdl (рис. 5.13).

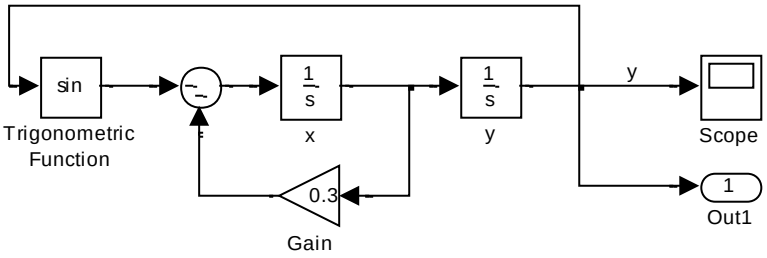


Рис. 5.13

Рассмотрим два способа поиска оптимальных начальных условий.

Способ 1 (одномерный линейный поиск).

Будем запускать схему из различных начальных условий, вычисляя каждый раз энергию E и запоминая результат, а затем выберем экстремальные значения.

Начальные условия будем формировать в MATLAB по формулам $y = R\sin\varphi,\quad \dot{y}_0 = R\cos\varphi,\quad 0 \le \varphi \le \pi.$

Программу для управления процессом моделирования оформим в виде отдельного *m*-файла optic.m.

```
Program optic                                  %Optimal Initial Condition for sys.mdl
open('sys');                                   % вызов модели sys.mdl
E=zeros(1,200);   R=1;                         % массив для записи энергий
for i=1:200
   x0=R*cosd(i);   y0=R*sind(i);               % очередные начальные условия
   set_param('sys/x','initial',num2str(x0));   % установка начальных условий в модель
   set_param('sys/y','initial',num2str(y0));
   sim('sys',[0,40],[],[]);                    % запуск моделирования
   E(i)=trapz(tout,yout.^2);                   % вычисление энергии
   disp(i);                                    % вывод текущего  i  на экран
end
plot(E,'LineWidth',2);grid                      % график зависимости энергии
title('ENERGY vs. IC angle'),                   % от угла на единичной окружности
```

Результат моделирования для R=1 представлен слева на рис. 5.14. Из графика видно, что максимальная энергия выходного сигнала примерно равна 2,4.

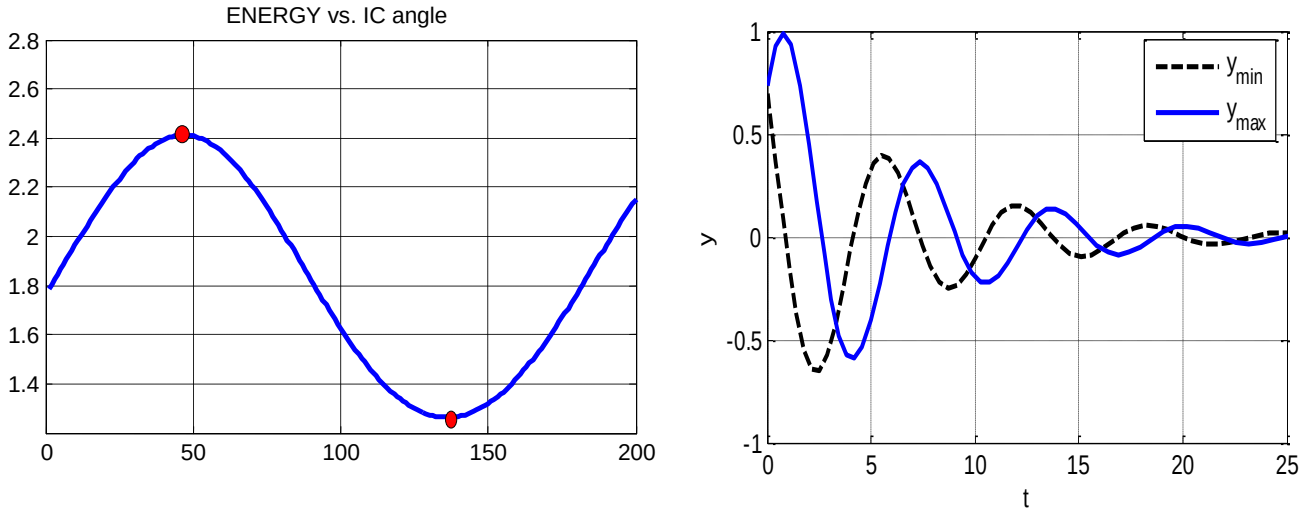


Рис. 5.14

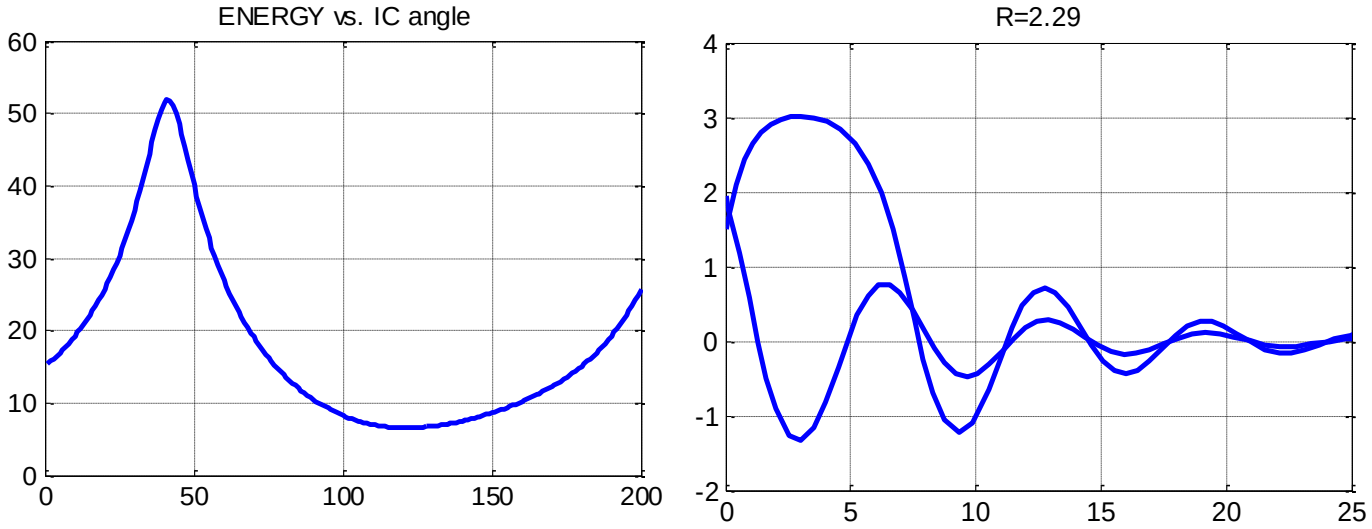
Более точные данные получаем, добавляя в программу следующие команды:
[EM,iM]=max(E);x0M=cosd(iM);y0M=sind(iM); [Em,im]=min(E);x0m=cosd(im);y0m=sind(im);
disp('iM,x0M y0M EM='),[iM,x0M y0M EM], disp('im,x0m y0m Em='),[im,x0m y0m Em],

Результат их выполнения:

```
iM,x0M y0M EM =   47   0.6820   0.7314   2.4118
im,x0m y0m Em =  136  -0.7193   0.6947   1.2637
```

Таким образом, максимальная энергия выходного сигнала, равная E=2,412, достигается при $\varphi=47^{o}$, т.е. при начальных условиях $y_0 = 0{,}682$, $\dot{y}_0 = 0{,}731$ (маятник отклоняем вправо и толкаем от положения равновесия). Минимальная энергия выходного сигнала, равная E=1,264, достигается при $\varphi=136^{o}$, т.е. при начальных условиях $y_0 = -0{,}719$, $\dot{y}_0 = 0{,}695$ (маятник отклоняем влево и толкаем к положению равновесия). Соответствующие графики приведены на рис. 5.14, справа.

Изменим радиус R окружности начальных условий, взяв его близким к критическому значению, при котором угол y отклонения маятника приближается к π. В этом случае маятник начинает «зависать» в верхнем положении, что приводит к резкому увеличению энергии выходного сигнала (при $y \to \pi$ энергия $E \to \infty$). На рис. 5.15 показаны графики энергии E в зависимости от угла φ (слева) и максимального и минимального выходного сигнала y в зависимости от времени (справа) для R=2,29, там же приведены численные значения экстремальных параметров.



R=2,29 iM,x0M y0M EM= 41 1.7283 1.5024 51.8763 im,x0m y0m Em= 122 -1.2135 1.9420 6.5953

Рис. 5.15

Способ 2 (поиск экстремума с помощью функции **fmincon**). В разд. 4.2.1 были описаны функции MATLAB для решения задач на условный и безусловный экстремум. В данном случае надо найти экстремумы функции E (энергии выходного сигнала), которая зависит от двух аргументов (начальных условий маятника) при наличии ограничения (сумма квадратов начальных условий фиксирована):

$$E = f(x_0, y_0) \to extr, \quad g(x_0, y_0) = x_0^2 + y_0^2 - R^2 = 0.$$

Для решения таких задач предназначена команда **fmincon**. Для ее применения нужно создать две вспомогательные функции, содержащие описание минимизируемого критерия и ограничений. Оформим эти функции в виде *m*-файлов с именами **energy** и **nlcon**:

function **energy**

```
function E = energy (x)
%open('sys');
x0=x(1);   y0=x(2);
set_param('sys_old/x','initial',num2str(x0,15));
set_param('sys_old/y','initial',num2str(y0,15));
[tout,xx,yout]=sim('sys_old',[0,40],[],[]);
E=trapz(tout,yout.^2);
```

function **nlcon**

```
function [h,g] = nlcon(x)

 h = [];
R=1;
 %R=2.29;
 g = x(1)^2+x(2)^2-R^2;
```

Запускаем процесс оптимизации, набирая в рабочем окне команду **fmincon**:

```
>> [X,E]=fmincon('energy',[1;1],[],[],[],[],[],[],'nlcon')
```

По окончании итерационного процесса получаем сообщение об его успешном завершении и значения выходных параметров (вектор начальных условий и минимальную энергию для R=1).

Чтобы решить задачу о максимальной энергии, нужно изменить знак перед командой trapz в последней строке функции **energy** и повторить вычисления. В результате получаем:

```
R=1          norm(X)    E=trapz(tout,yout.^2)        R=1          norm(X)    E=-trapz(tout,yout.^2)
X =  0.7198     1              E =1.2637             X =  0.6844     1              E =   -2.4118
    -0.6942                                             0.7291
```

Найденные значения близки к полученным первым способом, но более точные.

Выполняя аналогичные вычисления для R=2.29, также получаем уточненные значения оптимальных начальных условий:

```
R=2.29        norm(X)    E=-trapz(tout,yout.^2)      R=2.29        norm(X)    E=trapz(tout,yout.^2)
X =  1.7201    2.2900          E =  -51.8823          X =  1.2100    2.2900          E = 6.5955
     1.5118                                               -1.9442
```

Обратим внимание на два нюанса в оформлении функции **energy**. Если во второй строке убрать знак комментария (%), то вызов модели sys.mdl и перерисовка схемы моделирования будет происходить на каждом шаге итерационного процесса, что сильно увеличит время решения. Проще открыть эту модель вручную до начала решения, тогда команда open('sys') не потребуется. Второй нюанс касается синтаксиса команды num2str(x0,15). Ее второй аргумент задает количество разрядов числа x0, преобразуемых в строку. Если его опустить, то по умолчанию оно может оказаться равным 4, что полностью исказит результаты оптимизации (авторы столкнулись с этим явлением и долго не могли понять, в чем дело).

С подобными тонкостями нередко приходится встречаться при моделировании в MATLAB и других пакетах, они требуют от пользователя хорошего понимания сущности решаемой задачи и применяемого математического аппарата с одной стороны, и знания языка программирования и понимания процесса компьютерной реализации используемого алгоритма, с другой стороны.

Задачи и упражнения

1. Нелинейные осцилляторы. Выполнить моделирование в MATLAB приводимых ниже нелинейных дифференциальных уравнений и определить области их устойчивости в зависимости от начальных условий и значений параметров:

а) Уравнение математического маятника $\ddot{x} + a\sin x = 0$;

б) Уравнение Ван-дер-Поля $\ddot{x} + \mu(x^2 - 1)\dot{x} + x = 0$;

в) Уравнение Дюффинга $\ddot{x} + \omega^2 x + \mu x^3 = 0$.

Указание: В случае уравнения Дюффинга при $\mu > 0$ – решения колебательные. При $\mu < 0$ и при $x_0 = \pm\dfrac{\omega}{\sqrt{\mu}}$ происходит потеря устойчивости.

2. Цифровая модель. Построить теоретически и в пакете MATLAB цифровую модель непрерывной системы второго порядка $\ddot{y} + 3\dot{y} + 2y = 4u$; начальные условия – нулевые, шаг h = 0,1. Получить и сравнить графики переходных функций исходной системы и цифровой модели.

Теоретическое решение.

Корни характеристического уравнения непрерывной системы $p_1 = -1$; $p_2 = -2$; $h = 0{,}1$.

Корни характеристического уравнения цифровой модели находим по формуле $z_i = e^{p_i h}$: $z_1 = e^{-0,1}$; $z_2 = e^{-0,2}$.

Характеристическое уравнение цифровой модели

$(z - z_1)(z - z_2) = z^2 - (e^{-0,1} + e^{-0,2})z + e^{-0,3} \approx z^2 - 1{,}724z + 0{,}741 = 0.$

Ему соответствует разностное уравнение $y_n - 1{,}724y_{n-1} + 0{,}741y_{n-2} = \beta x_{n-2}$.

Для определения коэффициента β приравниваем установившиеся реакции непрерывной системы и цифровой модели на единичный входной сигнал x = 1. Для непрерывной системы $y_{уст.} = 2$; для цифровой модели $y_{n\,уст} = \dfrac{\beta}{1 - 1{,}724 + 0{,}741} = \dfrac{\beta}{0{,}017}$, откуда $\beta = 0{,}034$.

Итак, цифровая модель имеет вид $y_n - 1{,}724y_{n-1} + 0{,}741y_{n-2} = 0{,}034x_{n-2}$.

Переходные функции исходной системы и цифровой модели:

$y(t) = -4e^{-t} + 2e^{-2t} + 2$, $\quad y_n = -4(0{,}905)^n + 2(0{,}819)^n + 2$.

Решение в MATLAB. Приведем варианты решения с помощью команд **c2d**, **impinvar** и **bilinear:**

```
>> sys=
tf(4,[1 3 2])
     4
-------------
s^2 + 3 s + 2
```

```
>> sysd =
c2d(sys,0.1,'imp')
        0.3444 z
----------------------
z^2 - 1.724 z + 0.7408
```

```
>> [bz,az] =
impinvar(4,[1 3 2],10)
bz =     0    0.0344
az =1.0000   -1.7236    0.7408
```

```
>> [numd,dend] =
bilinear(4,[1 3 2],10)
numd =    0.0087    0.0173    0.0087
dend =    1.0000   -1.7229    0.7403
```

Видно, что с теоретическим решением совпадает результат команды **impinvar**.

3. Газопровод. Модель газопровода, условно разбитого на шесть участков, описывается системой дифференциальных уравнений

$$\dot{X} = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} X + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u, \qquad Y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} X.$$

Здесь u – давление газа на входе, x_i – давление на отдельных участках газопровода. Вектор Y описывает места установки измерительных датчиков.

Требуется: а) выполнить моделирование системы в SIMULINK при скачкообразном и синусоидальном изменении входного сигнала;

б) найти модальное, сопровождающее и сбалансированное представления это системы и сравнить графики их весовых функций.

4. Ограниченная задача двух тел. Движение спутника вокруг Земли в плоскости орбиты характеризуется дифференциальными уравнениями:

$$\ddot{x} = -\frac{kx}{\sqrt{(x^2+y^2)^3}}, \quad \ddot{y} = -\frac{ky}{\sqrt{(x^2+y^2)^3}}.$$

Гравитационная постоянная $k=3{,}9870 \cdot 10^5 \frac{km^3}{c^2}$.

Требуется составить схему и выполнить моделирование в SIMULINK при начальных условиях $x_0 = 0; y_0 = 6478{,}388$ км, $\dot{x}_0 = 7{,}89$ км/с, $\dot{y}_0 = 0$. Получить графики координат спутника, его траектории и высоты относительно поверхности Земли $h = \sqrt{x^2+y^2+z^2} - R_3$, где радиус Земли R_3=6378,388 км.

5. Управление луноходом. Луноход-1 (1970-71г.), имел вес 700кг, 2 скорости 1км/ч и 2км/ч, прошел 11 км за 10 месяцев. Управление им осуществлял экипаж, находящийся на Земле, задержка сигнала в контуре управления составляла порядка трех секунд. Упрощенная математическая модель системы управления характеризуется уравнением:

$$\dot{y}(t) + ay(t-T) = a\,y_{зад}.$$

Этому уравнению при a=1 соответствует структурная схема на интеграторе и элементе задержки ЭЗ, показанная на рис.5.16. Требуется выполнить ее моделирование в SIMULINK при $y_{зад} = 1$, получить графики $y(t)$ и $\dot{y} = f(y)$, определить период колебаний.

Найти значение a, при котором решение носит периодический характер. Построить область устойчивости на плоскости (a, T).

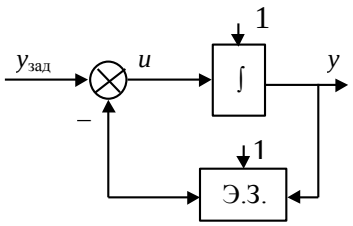


Рис. **5.16.**

ПРОГРАММИРОВАНИЕ В MATLAB

Типы данных

MATLAB поддерживает большое количество типов данных, которые, по сути, являются классами. Для того чтобы иметь общее представление о возможностях пакета, на рис. 5.1 приведено полное дерево классов пакета. Однако более чем в 90% приложений пользователь имеет дело лишь с классами *double, cell* и пользовательскими классами. Все классы MATLAB являются наследниками массива (*array*). Таким образом, MATLAB является по своей структуре типичным векторным процессором.

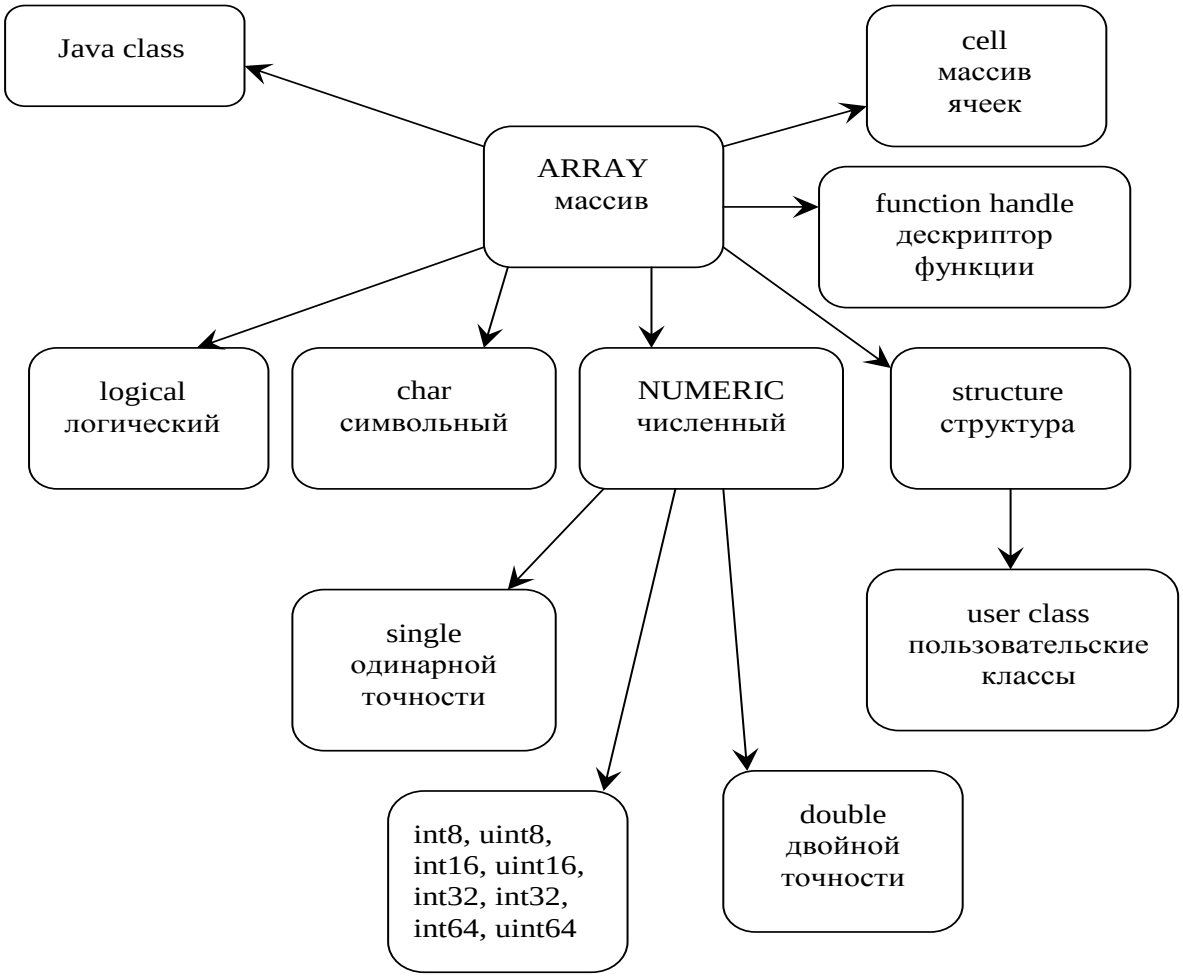


Рис. 6.1

Класс *double array* является, в некотором смысле, «классом по умолчанию». Даже если ввести в командном окне MATLAB скалярную переменную, то она будет восприниматься как массив размером 1x1:

```
>> c=3; size (c)
ans =1  1
```

Еще один важный тип – «массив ячеек» (*cell array*). Элементами подобного массива может быть что угодно: числа, строки, массивы, другие массивы ячеек, экземпляры класса и т.п. Задается он при помощи фигурных скобок { }:

```
>> c={1,2 3; 'scrambled eggs',[12, 13, 14], -0.3}
c =

    [           1]          [        2]        [      3]
    'scrambled eggs'    [1x3 double]    [-0.3000]
```

Употребление точки с запятой и запятой такое же, как и при задании массивов. Чтение и запись отдельных элементов производится так же, как и в случае с массивами типа *double*, однако вместо круглых используются фигурные скобки:

```
c{1,2}          c{2,1}                  c{2,2}=18
ans =           ans =                   c =
    2           scrambled eggs              [           1]          [ 2]    [      3]
                                            'scrambled eggs'    [18]    [-0.3000]
```

Команды **size** и **length** в случае с массивами ячеек ведут себя так же, как и с обычными:

```
>> size(c)              >> c1={1,2 ,3}          >> length(c1)          >> c2={3,; 4 ; 7}       >> size(c2)
ans =                   c1 =                    ans =                  c2 =                    ans =
    2     3                 [1]    [2]    [3]        3                      [3]                     3     1
>> length(c)            >> size(c1)                                       [4]                  >> length(c2)
ans =     3             ans =     1     3                                 [7]                  ans =     3
```

Операция «вырезка» из массива ячеек ведет себя довольно неожиданно, поэтому на начальном этапе знакомства с пакетом лучше ее избегать.

Для взаимного преобразования между массивами *double* и *cell array,* помимо присваивания в цикле, существуют стандартные средства – команды **num2cell, mat2cell** и **cell2mat**. Команда **num2cell**, как явствует из названия, предназначена для преобразования численных массивов в массивы ячеек. Цифру 2 (two) в обозначении операции следует читать как предлог to (к).

```
>> a=[1 2; 3 4]          >> num2cell(a)          >>n1=num2cell(a,1)     >> n1{1}              >>n2=num2cell(a,2)
a =                      ans =                   n1 =                   ans =                 n2 =
    1     2                  [1]    [2]              [2x1         double]      1                     [1x2 double]
    3     4                  [3]    [4]              [2x1 double]              3                     [1x2 double]
```

Простейший вариант вызова команды **num2cell** – с одним аргументом. Результат в этом случае очевиден. При вызове с двумя аргументами первый содержит исходный массив, а второй – номер измерения, по которому матрицу предстоит «разрезать» на полосы. Так num2cell(a,1) «режет» матрицу по вертикали, а num2cell(a,2) – по горизонтали.

Действие команды **cell2mat** обратно действию **num2cell**, причем вызов осуществляется только с одним входным аргументом:

```
>> cell2mat(num2cell(a))      >> cell2mat(num2cell(a))-a
ans =                         ans =
    1     2                       0     0
    3     4                       0     0
```

Команда **mat2cell** наиболее интересна из всех перечисленных выше. На входе она принимает три аргумента – исходный массив и два массива, содержащих размеры вертикальных и горизонтальных блоков. Линии, по которым следует разбить массив, показаны на рисунке. Видно,

что размеры вертикальных блоков – 1 и 2, а горизонтальных – 2, 3 и 1. Эти значения – [1 2] и [2 3 1] как раз и являются вторым и третьим аргументами команды **mat2cell**. Листинг программы и значения массивов в ячейках после применения команды **mat2cell** показаны ниже.

```
>> b=[ 1 2 3 4 5 6; 7 8 9 10 11 12; 13,14,15, 16, 17, 18 ]
b =
    1    2    3    4    5    6
    7    8    9   10   11   12
   13   14   15   16   17   18
```

```
>> b1=mat2cell(b,[1 2],[2 3 1])
b1 =
    [1x2 double]    [1x3 double]    [       6]
    [2x2 double]    [2x3 double]    [2x1 double]
```

```
>> b1{1,1}
ans =
     1     2
>> b1{1,2}
ans =
     3     4     5
```

```
>> b1{1,3}
ans =
     6
>> b1{2,1}
ans =
     7     8
    13    14
```

```
>> size(b)
ans =
     3     6
>> b1{2,2}
ans =
     9    10    11
    15    16    17
```

```
>> b1{2,3}
ans =
    12
    18
```

Использование структур и пользовательских классов

Скорее всего, Вам не придется писать собственные классы средствами MATLAB . Тем не менее, многие стандартные тулбоксы реализованы с их применением. Поэтому полезно владеть как минимум основами использования классов в MATLAB .

Первый вопрос, который возникает – как определить, к какому типу или классу принадлежит объект. Для этого существуют функции **class** и **isa** (*is a* ….). Первая из них возвращает имя класса, которому принадлежит объект, а вторая возвращает 1 или 0, в зависимости от того, принадлежит ли объект указанному в качестве второго параметра классу. Покажем их работу на примере уже рассмотренных типов данных *double* и *cell array*:

```
>> a=[1 2 3]
a =     1     2     3
>> b={4,5,6}
b =     [4]    [5]    [6]
```

```
>> class(a)
ans = double
>> class(b)
ans = cell
```

```
>> isa(a,'double')
ans =     1
>> isa(a,'cell')
ans =     0
```

```
>> isa(b,'double')
ans =     0
>> isa(b,'cell')
ans =     1
```

```
>> class(class(a))
ans =char
```

Напомним, что все пользовательские классы являются наследниками структуры (*struct*). Структура в пакете MATLAB создается командой **struct**, которая в качестве входных параметров принимает набор пар имени и значения поля: struct (имя1, значение1, имя2, значение2, …..). Обращение к полям структуры осуществляется при помощи оператора «точка». Значением поля структуры может быть объект любого класса, в том числе массив чисел или массив ячеек:

```
>> d = struct('strings',{{'hello','yes'}},'lengths',[5 3])
d =
    strings: {'hello'  'yes'}
    lengths: [5 3]
```

```
>> d.strings
ans =
    'hello'    'yes'
```

```
>> d.strings{1}='no'
d =
    strings: {'no'  'yes'}
    lengths: [5 3]
```

Обратите внимание, что при объявлении структуры, у которой одно из полей является массивом ячеек, используются одни «лишние» скобки. Если их не поставить, то будет создан целый массив структур. При этом функция **struct** демонстрирует следующее поведение. Если лишь одно поле f1 в качестве значения имеет массив ячеек, то создается массив структур, в котором поле f1 каждого из элементов принимает поочередно значения из массива ячеек, а остальные поля

фиксированы. Если два или более полей заданы массивами ячеек, то эти массивы должны быть одинаковой длины и команда **struct** создает массив структур, в котором элементы поочередно принимают соответствующие значения. Если же количество ячеек разное, то выдается сообщение об ошибке:

```
>> d = struct('strings',{'hello','yes'},'lengths',[5 3])
d =
1x2 struct array with fields:
   strings
   lengths

>> u=struct('f1',{1,2},'f2',{3,4})
u = 1x2 struct array with fields:
   f1
   f2
>> u=struct('f1',{1,2},'f2',{3,4,5})
??? Error using ==> struct
Array dimensions of input 4 must match those of
input 2 or be scalar.
>> u=struct('f1',{1,2},'f2',{3,4},'f3',{5,6})
u = 1x2 struct array with fields:
   f1
   f2
   f3
```

```
>> d.strings
ans =
hello
ans =
yes

>> u(1)
ans =
   f1: 1
   f2: 3
>> u(1)
ans =
   f1: 1
   f2: 3
   f3: 5
```

```
>> d.lengths
ans =  5     3
ans =  5     3

>> u(2)
ans =
   f1: 2
   f2: 4
>> u(2)
ans =
   f1: 2
   f2: 4
   f3: 6
```

```
>> size(d)
ans =  1     2
>> d(1)
ans =
   strings: 'hello'
   lengths: [5 3]
>> d(2)
ans =
   strings: 'yes'
   lengths: [5 3]
```

Для работы со структурами имеются следующие функции: **isfield, getfield, setfield**, **rmfield, fieldnames**. Команда isfield (s,fldname) возвращает 1 если у структуры s есть поле fldname и 0 в противном случае. Команда getfield (s,fldname) возвращает значение поля fldname структуры s, т.е. является синонимом для s.fldname. Функция setfield (s,fldname,fldvalue) возвращает копию объекта s, у которой полю fldname присвоено значение fldvalue. Сам объект s при этом не изменяется, т.е. это неравносильно присваиванию s.fldname=fldvalue. Функция rmfield(s,fldname) предназначена для удаления поля fldname у объекта s. Для того, чтобы добавить новое поле достаточно просто присвоить ему значение: s.new_field_name=some_value. Функция fieldnames(s) возвращает массив ячеек, содержащий имена полей структуры s.

Ниже приведены примеры использования этих команд.

```
>>u=struct('f1',1,'f2',0.1)
u =
   f1: 1
   f2: 0.1000

>> fieldnames(u)
ans =
   'f1'
   'f2'
```

```
>> isfield(u,'f1')
ans =     1
>> isfield(u,'f3')
ans =     0

>>rmfield(u1,'f2')
ans =     f1: -1
```

```
>> getfield(u,'f1')
ans =     1
>>setfield(u,'f1',-1)
ans =
   f1: -1
   f2: 0.1000
>> u1
u1 =
   f1: -1
   f2: 0.1000
```

```
>> u
u =
   f1: 1
   f2: 0.1000

>>u2=rmfield(u1,'f2')
u2 =
   f1: -1
```

```
>>u1=setfield(u,'f1',-1)
u1 =
   f1: -1
   f2: 0.1000

>> fieldnames(u2)
ans =     'f1'
>> u2.f1=19
u2 =     f1: 19
```

Поскольку пользовательские классы являются наследниками **struct**, все сказанное относится и к ним.

Для того чтобы успешно использовать существующие классы в пакете MATLAB, такие как ss, tf, zpk, lti нужно учитывать, что функции над объектами не изменяют исходного объекта, но возвращают его копию. В языке MATLAB методы класса, с точки зрения пользователя, являются обыкновенными функциями, которые «знают», как себя вести, когда в качестве аргумента задан объект этого класса.

Сервисные функции

В системе MATLAB существует ряд служебных функций. Здесь будет описаны функции, относящиеся к трем категориям – управление рабочим пространством, управление *m*-функциями и системные взаимодействия.

К первой категории можно отнести команды **who, whos, clear, save, load**. По команде **who** выводится список переменных в рабочем пространстве или *mat*-файле. Команда **whos** является расширенным вариантом **who**. Покажем разницу между информацией, выдаваемой этими двумя командами:

```
>> a=17;
>>b={'ratata','gagaga'};
>> who
Your variables are:
a  b
```

```
>> whos
Name      Size            Bytes  Class
  a       1x1                 8  double array
  b       1x2               144  cell array
Grand total is 15 elements using 152 bytes
```

```
>> v=who
v =     'a'
        'b'
>> w=whos
w = 3x1 struct array with fields:
        name
        size
        bytes
        class
```

```
>> w(1)
ans =
    name: 'a'
    size: [1 1]
   bytes: 8
   class: 'double'
>> w(1).bytes
ans =
     8
```

Как следует из листинга, обе перечисленные команды могут не только выводить информацию о переменных рабочего пространства на экран, но и помещать результаты в выходную переменную.

Для того чтобы очистить переменную в рабочем пространстве, используется команда **clear.** В зависимости от формата вызова, она может очистить все переменные рабочего пространства, указанную переменную, глобальные переменные, классы, загруженные модули Java или скомпилированные функции. В рамках настоящего пособия интерес представляют первые три способа использования:

```
>> a=1; b=2; c=3; global d; d=8;
>> who
Your variables are:
a   ans  b   c   d   v   w
```

```
>> clear c;
>> who
Your variables are:
a   ans  b   d   v   w
>> clear globals;
```

```
>> who
Your variables are:
a   ans  b   d   v   w
>> clear
>> who
```

Графический интерфейс браузера рабочего пространства можно запустить программно при помощи команды **workspace**. Для программного вызова GUI, предназначенного для редактирования массивов, пользуйтесь командой **openvar**. Чтобы понять, о чем идет речь, выполните следующий код:

```
>> m=eye(5);  openvar('m').
```

Для того чтобы сохранить в файле и считывать из файла переменные рабочего пространства, используются команды **save** и **load**. Формат вызова этих команд следующий:

```
save имя_файла переменная1 переменная2 ….
load имя_файла переменная1 переменная2 ….
load имя_файла
```

Приведем пример работы:

```
>> a=1; b=2; c=3; global d; d=8;
>> save file1 a b d
>> clear
```

```
>> who
>> load file1
>> who
```

```
Your variables are:
a  b  d
```

Стандартный формат хранения MATLAB-переменных – *.mat. Если Вы хотите читать и писать *mat*-файлы из собственных программ, воспользуйтесь имеющимся C или FORTRAN API. Для переноса результатов работы между различными версиями MATLAB воспользуйтесь дополнительными ключами –v4 или –v6. Тогда данные будут сохранены в *mat*-формате, поддерживаемом в 4-й и 6-й версиях, соответственно.

Свои данные можно сохранять и в ASCII-файлах, пользуясь ключом –ascii:

```
>> m=[23 67; 90 890];
>> save file2 m -ascii
>> clear m
>> load file2 m -ascii
>> m
??? Undefined function or variable 'm'.

>> m=[23 67; 90 890];
>> save file2 m n -ascii
>> clear
```

```
>> who
Your variables are:
file2
>> file2
file2 =
   23   67
   90  890
>> load file2   -ascii
??? Error using ==> load
Number of columns on line 3 of ASCII file
C:\MATLAB\work\file2
must be the same as previous lines.
```

```
>> n=eye(5)
n =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Обратите внимание на то, что в ASCII-формате поддерживаются далеко не все возможные типы данных MATLAB. Помимо этого, теряется вся информация об именах. Более того, теряется значительная часть информации о типах данных, что видно из последнего примера. Тем не менее, возможность импорта и экспорта в ASCII может быть весьма полезна, если необходимо работать с несколькими различными приложениями одновременно.

MATLAB обладает функциями для чтения/записи бинарных файлов, а также поддерживает импорт/экспорт большого количества различных форматов данных и графических форматов.

В MATLAB имеется несколько команд, которые делают более удобной работу с файлами функций и сценариев. Это команды **dir, type, pwd, cd, what, matlabroot, path, addpath, rmpath, genpath, rehash, which.**

Команда **type** выводит в командном окне содержимое *m*-файла, например, type misha печатает листинг текстового файла с именем misha.m, хранящегося в текущем каталоге. Если расширение (.m, .mat, .bat, ...) не указано, то по умолчанию вызывается *m*-файл.

Функции **pwd** и **cd** делают то же самое, что и соответствующие команды ОС – выводят название текущего каталога и меняют текущий каталог:

```
>> pwd
ans = C:\MATLAB\work
```

```
>> cd ..
>> pwd
ans = C:\MATLAB
```

Команда **dir** (*directory*) отображает в командном окне совокупность всех файлов, хранящихся в текущем каталоге. При вызове с выходным параметром в него помещается массив структур с полями name, date, bytes и isdir. Команда **what** делает примерно то же самое, но несколько в другом формате, отображая при этом только файлы с расширениями, используемыми в MATLAB. При вызове с выходным параметром она возвращает структуры с полями path, m, mat, mex, mdl, p, classes. Поле path содержит имя каталога, m – список *m*-файлов в виде массива ячеек, mat – список *mat*-файлов и т.д. Расширения .mex и .p соответствуют скомпилированным исполняемым MATLAB-файлам и файлам, скомпилированным в псевдокоде. Поле classes содержит список классов в текущем каталоге. Сравним работу команд **what** и **dir**:

```
>> d=dir(pwd)
d =
194x1 struct array with fields:
   name
   date
```

```
>> d(1)
ans =
    name: '.'
    date: '05-Jan-2003 16:19:16'
    bytes: 0
```

```
>> d(2)
ans =
    name: '..'
    date: '05-Jan-2003 16:19:16'
    bytes: 0
```

```
    bytes
    isdir
>> d(3)
ans =
    name: '1.TXT'
    date: '16-Feb-2005 00:19:52'
    bytes: 2104
    isdir: 0
```

```
    isdir: 1

>> w=what(pwd)
w =
    path: 'C:\MATLAB\work'
    m: {105x1 cell}
    mat: {2x1 cell}
    mex: {0x1 cell}
    mdl: {18x1 cell}
    p: {0x1 cell}
    classes: {3x1 cell}
```

```
    isdir: 1

>> w.mat
ans =
    'hanke14.mat'
    'hanke15.mat'
```

Команда **cd** (*change directory*) выдает сообщение о том, в каком каталоге работает в настоящее время пользователь и позволяет также переходить из текущего каталога в другой. Например, для перехода из каталога D:\MATLAB\BIN в каталог D:\MATLAB \CONTROL нужно ввести команду cd \MATLAB \control

Команда **matlabroot** возвращает имя каталога, в котором установлен MATLAB. Команда **path** выводит список путей, включая все установленные тулбоксы. Команды **addpath** и **rmpath** добавляют и удаляют каталоги в путь. При этом команда **addpath** поддерживает добавление в начало или конец пути при помощи дополнительных параметров –BEGIN и –END. Команда **genpath** составляет список всех подкаталогов указанного каталога, что упрощает процедуру установки новых тулбоксов. Командой **rehash** имеет смысл пользоваться тогда, когда Вы уже внесли изменения в пути MATLAB, но не уверены, что система уже «видит» эти изменения.

В связи с тем, что разные каталоги могут содержать *m*-файлы с одинаковыми названиями, может возникнуть сомнение, что выполняется нужный файл. В таких случаях полезна команда **which**, выдающая полный путь *m*-файла. Например,

```
>> which tmp08
C:\MATLAB\work\tmp08.m
```

Взаимодействие с системой

Самые простые «системные» команды MATLAB – это команды завершения сессии **quit** или **exit**. Прекращение сеанса MATLAB приводит к потере переменных в рабочей области.

В MATLAB предусмотрено 4 возможности организации интерфейса с другими программами:

- непосредственный вызов исполняемых файлов и команд операционной системы;
- вызов PERL-скриптов;
- использование классов Java;
- взаимодействие через COM-интерфейс.

Для взаимодействия с ОС используется символ восклицательного знака **!**. Он указывает, что остаток строки должен быть передан как команда в операционную систему. Если строка завершается символом &, то для работы внешней программы открывается отдельное окно. Так, сравните результаты выполнения следующих трех команд:

```
>>path          >> ! path          >> ! path &
```

Если просто набрать &, то будет открыто окно сеанса MS DOS. Если результат выполнения внешней программы следует поместить в переменную, то используются команды **system** (для всех платформ), **dos** или **unix**. Формат вызова всех этих команд одинаков: [status, result] = DOS('command', '-echo'), где status – это код завершения программы, а result – вывод программы. Аргумент '-echo' не является обязательным. В операционных системах Windows 98 и Windows ME встроенные команды DOS и bat-файлы в качестве кода завершения всегда возвращают 0.

В комплект поставки MATLAB входит интерпретатор языка PERL (см. каталог, например, C:\MATLAB\sys\perlsys\perl), позволяющий решать широкий круг программистских задач. Для

выполнения PERL-скриптов в MATLAB имеется команда **perl**. Синтаксис вызова этой команды result=perl(perlfile,arg1,arg2,...), где perlfile – имя скрипта с указанием пути, а arg1, arg2 и т.д. – аргументы.

Управляющие структуры языка MATLAB

MATLAB изначально сконструирован как векторный процессор, что резко сокращает необходимость использования большинства управляющих конструкций.

В MATLAB есть два условных оператора – **if** и **switch**. Синтаксис использования **if** в развернутой форме имеет вид:

```
if условие
  операторы
elseif еще одно условие
  операторы
else
  операторы
end
```

Ниже дан пример, выводящий на экран надпись ‘a is even’ для четных чисел *a*.

```
>> if rem(a,2) == 0
disp('a is even')
end
```

Если число *a* было четно, то получим ответ: a is even

Для организации циклов служат операторы **for** и **while**. Цикл **for** позволяет повторять группы команд заданное число раз. Например, цикл

```
>> for  i=1:5
        x(i)=0
      end
```

присваивает значение 0 первым пяти элементам вектора *x*.

Разумеется, вместо этого естественней было бы сказать x(1:5)=0.

Цикл **while** дает возможность повторять группу операторов нефиксированное число раз до выполнения логического условия. Вот простая задача, иллюстрирующая цикл **while**: найти наименьшее число n, факториал которого n! является 100-значным числом.

```
>> n = 1;  f=1;
while log10(f) < 100
 f=f*n; n = n+1;
end
disp(n-1)
   70
```

Для досрочного завершения цикла используется команда **break**, а для досрочного перехода к следующей итерации – команда **continue**. Команда **input**, фигурирующая в тексте, предназначена для ввода пользователем данных из командного окна. Для ввода строки эта команда вызывается с дополнительным параметром 's'.

Описание m-функций

Процедура создания файлов-сценариев кратко рассмотрена в п. 1.2. Большая часть системы MATLAB написана на языке сценариев MATLAB. Покажем, как можно расширить систему, добавляя собственные функции.

Создадим простейшую функцию, которая не принимает никаких входных параметров и не возвращает выходных.

```
function hello_world;
disp('Get lost!!!');
```

В отличие от файла сценария, который может называться как угодно, имя файла, содержащего функцию, должно совпадать с названием, стоящим при ключевом слове function, т.е. приведенная выше функция должна храниться в файле hello_world.m

Создадим функцию с несколькими входными параметрами и одним выходным.

```
function d=sum3(a,b,c)
d=a+b+c;
```

Если выходной параметр был определен, но функцию вызвали без него, ответ помещается в служебную переменную **ans:**

```
>> a=sum3(1,2,3)
a =  6
>> sum3(1,2,3)
ans =  6
```

Для того чтобы описать функцию с несколькими выходными параметрами, достаточно перечислить их в квадратных скобках, например:

```
function [s,d]=sum_and_difference(a,b);
% function [s,d]=sum_and_difference(a,b);
% This function returns sum and difference of two values.
s=a+b;
d=a-b;
```

Эта функция возвращает сумму и разность своих аргументов:

```
>> [x1,x2]=sum_and_difference(3,5)
x1 =  8
x2 = -2
```

Строчки с комментариями, расположенные сразу же после объявления функции, выводятся при запросе **help** на вновь созданное имя функции:

```
>> help sum_and_difference
  function [s,d]=sum_and_difference(a,b);
  This function returns sum and difference of two values.
```

Обработка входных и выходных аргументов функций MATLAB

Большинство функций MATLAB могут поддерживать различный синтаксис вызова. Для этого имеются четыре специальные переменные:

varargin – массив входных аргументов,

varargout – массив выходных аргументов,

nargin – количество входных аргументов,

nargout – количество выходных аргументов.

Поскольку входные и выходные аргументы могут быть разного типа, для их хранения используются массивы ячеек. Таким образом, первый входной аргумент будет адресоваться как **varargin{1},** второй – **varargin{2}** и т.д. Напишем функцию sum_and_difference так, чтобы предусмотреть разное количество аргументов.

```
function varargout=sum_and_difference2(varargin);
if nargin>=1
   a=varargin{1};
else
   a=0;
end
if nargin>=2
   b=varargin{2};
else
   b=0;
end
s=a+b;
d=a-b;
if nargout==2
   varargout{1}=s;
   varargout{2}=d;
else
   varargout{1}=[s, d];
end
```

Теперь можно будет вызывать функцию с разными вариантами синтаксиса:

```
>> [a,b]=sum_and_difference2(3,7)
a = 10
b = -4
>> a=sum_and_difference2(3,7)
a = 10    -4
```

```
>> a=sum_and_difference2(3)
a =  3     3
>> [a,b]=sum_and_difference2(3)
a =  3
b =  3
>> sum_and_difference2
ans =  0     0
```

Рассматривая исходные тексты большинства стандартных функций MATLAB, можно убедиться, что их большую часть занимает именно анализ входных и выходных аргументов.

Глобальные переменные. Доступ к переменным из различных рабочих пространств

Все переменные в функции являются локальными. Есть возможность объявления глобальных переменных при помощи команды **global**:

```
function a2=test_global
global a;
a2=a^2;
```

Прежде чем использовать глобальную переменную, ее необходимо объявить таковой в корневом рабочем пространстве:

```
>> a=5
a =  5
```

```
>> a
a =  [ ]
```

```
>> % переменная a еще не объявлена глобальной
>> test_global
ans =  [ ]
>> global a;
Warning: The value of local variables may have been changed to
match the globals.  Future versions of MATLAB will require that
you declare a variable to be global before you use that variable.
```

```
>> a=5
a =   5
>> test_global
ans = 25
```

Осуществить проверку того, является переменная глобальной или нет, можно при помощи команды **isglobal**:

```
>> a=9;
>> isglobal(a)
ans =  0
>> global a;
Warning: The value of local variables may have been changed to match the
      globals.  Future versions of MATLAB will require that you declare
      a variable to be global before you use that variable.
>> isglobal(a)
ans = 1
```

Существует еще один способ доступа к переменным родительского рабочего пространства – при помощи функций **assignin** и **evalin**. Синтаксис вызова первой из этих команд assignin(ws,'name',v), где ws – имя рабочего пространства, 'name' – имя присваиваемой переменной, а v – ее значения. Параметр ws может принимать два значения – 'base' для обозначения корневого рабочего пространства или 'caller' для обозначения рабочего пространства вызывающей функции. Команда **assignin**, как следует из названия, заносит значение v в переменную по имени 'name' рабочего пространства ws. Для того чтобы получить значение переменной (или вычислить выражение) в другом пространстве, используется команда evalin(ws,'expression'). Обращаясь к рабочему пространству 'caller' можно получить доступ лишь к тем локальным переменным, которые упомянуты в списке входных или выходных аргументов.

Приведем небольшой пример доступа к пространству 'caller'. Опишем две вложенные функции assignin_demo1 и assignin_demo2:

```
function m=assignin_demo1
disp('m in assignin_demo1 workspace:')
m=[1 2 ; 3 4]
assignin_demo2;
disp('After calling assignin_demo2 m had changed:')
disp(m)
```

```
function assignin_demo2
m=evalin('caller','m');
disp('m in caller(assignin_demo1) workspace:')
disp('Let''s change m and pass it back to assignin_demo1 workspace:')
m=[5 6; 7 8]
assignin('caller','m',m);
```

При вызове assignin_demo1 в командном окне появится следующая реакция:

```
>> k=assignin_demo1
m in assignin_demo1 workspace:
m =     1     2
        3     4
m in caller(assignin_demo1) workspace:
```

```
Let's change m and pass it back to assignin_demo1 workspace:
m =    5    6
       7    8
After calling assignin_demo2 m had changed:
   5    6
   7    8
k =
   5    6
   7    8
```

Проиллюстрируем возможность использования пространства имен 'base' из функции k=assignin_demo3:

```
function k=assignin_demo3
k=evalin('base','a+8');
```

Вызовем ее из командного окна:

```
>> a=19;
>> k=assignin_demo3
k =   27
```

В MATLAB-функциях существует также возможность использования статических переменных при помощи ключевого слова persistent.

Задачи и упражнения

1. Напишите m-функцию, возвращающую список файлов в текущем каталоге, отсортированных по размеру.

Указание. Вызвать каталог командой **dir**; сформировать массив, содержащий размеры файлов; выполнить его сортировку с помощью команды **sort**, например [a,b]=sort({'asasa','aaaaa','akkkk'}), после этого упорядочить файлы в соответствии с результатами сортировки.

2. Пусть функция описана как function myfunc(p1,p1,varargin).

а) При ее вызове с параметрами myfunc(17,64,[24 -5 0],{78,90,[8 9]}) чему будет равняться переменная nargin? *б*) Какой тип будет иметь параметр varargin{2}? varargin{3}?

Ответ: *а*) nargin=4, *б*) varargin{2}={78,90,[8 9]}, varargin{3} выдаст ошибку

3*а*. Пусть в рабочем пространстве была описана переменная global a. Ниже приведен текст функции: function f=mufunc a=1:10; f=sum(a); После ее выполнения чему будет равна глобальная переменная a?

Ответ: Она не изменит своего значения, поскольку в функции переменная a локальная (слово «global» не использовалось).

3*б*. Что произойдет при запуске на выполнение функции **mufunc1**?

```
function f=mufunc1
a=a+1:10;
f=fum(a);
```

Ответ: Будет выдано сообщение об ошибке (см. предыдущий ответ).

4. Попробуйте запустить следующий фрагмент кода

```
s=(1:3)'; k=0;
for i=s
k=k+i;
end
k
```

Поясните результат.

Ответ: Поскольку s – столбец, будет получен результат

```
k =
```

1
2
3

Цикл вызовется всего один раз со значением i=[1 2 3]'

5. а) Что нужно изменить, в условиях предыдущей задачи, чтобы получить k=6? *б*) Как добиться того же результата, если s={1,2,3}?

Ответ: Надо задать s=1:3, а не s=(1:3)'.

б) Организовать цикл по индексам.

6. Предположим, имеются две одноименные функции myfunc, расположенные в каталогах dir1 и dir2. Как определить, какая именно из них была запущена? Придумайте хотя бы 3 способа.

Ответ:

Способ 1: вставить в оба файла метки типа display(1) и display(2), запустить, функцию и посмотреть, какое сообщение появится на экране (1 или 2).

Способ 2: посмотреть в путях, какой каталог находится выше.

Способ 3: использовать команду **which**.

7. Напишите функции, вычисляющие числа Фибоначчи при помощи циклов **for**, **while** и вообще без циклов.

Указание. Воспользуйтесь функцией **sum/cumsum**. Другие варианты: использовать формулу Бине, команду **iztrans**, команду **initial** для соответствующей дискретной системы

8. Предположим, в вашей функции **myfunc1** при некотором условии выводится сообщение об ошибке error('Something wrong');

Вызывая эту функцию из другой функции **myfunc2**, Вы хотите, чтобы ошибка при выполнении функции **myfunc1** не прекращала выполнения вызывающей функции. Как этого добиться?

Указание. Воспользуйтесь командами **try/catch**.

9. Предположим, в рабочем пространстве есть переменная *a*. Как при этом вызвать программу с именем a.m?

Ответ: Следует переименовать программу либо изменить имя переменной.

10. Найдите ошибку в коде

```
k=7
if k=8
disp('k=8!')
end
```

Ответ: Следует написать if k= =8 (не присвоить =, а сравнить = =).

Заключение

Пакет MATLAB – это удобное и быстро осваиваемое программное средство, которое позволяет эффективно решать широкий круг задач линейной алгебры, численного анализа, обработки сигналов, моделирования систем управления и многих других. Пакет представляет собой обширный, хорошо развитый программный комплекс, содержащий около 1000 команд, 30 тулбоксов, а также систему визуального моделирования SIMULINK с расширениями. Он может обмениваться данными с другими популярными пакетами и приложениями, такими как EXCEL, WORD, MAPLE.

В настоящем пособии описана лишь небольшая часть его возможностей. Для желающих продолжить освоение пакета и применять его для решения учебно-методических задач и научных исследований можно рекомендовать более полные руководства, например, книги [1-5, 10, 11, 13]. Для того чтобы в полной мере использовать его возможности, нужна определенная математическая квалификация, соответствующая высокому интеллектуальному уровню команд MATLAB. Поэтому начинающим пользователям пакета можно пожелать наряду с освоением системы команд MATLAB совершенствовать свою математическую подготовку – обе эти компоненты в равной степени необходимы для эффективного применения MATLAB.

Библиографический список

1. *Ануфриев И. Е.* Самоучитель MatLab 5.3/6.x. -СПб.: БХВ-Петербург, 2003.
2. *Дьяконов В., Круглов В.* Математические пакеты расширения MATLAB. -СПб.: Питер, 2001.
3. *Дьяконов В.П.* MATLAB 6/6.1/6.5 + SIMULINK 4/5 в математике и моделировании. -М.: Солон-Пресс, 2003.
4. *Дэбни Дж., Хароган Т.* SIMULINK 4. Секреты мастерства. -М.: Бином, Лаборатория знаний. 2003.
5. *Кетков Ю., Кетков А., Шульц М.* MATLAB 6.x: программирование численных методов. -СПб.: БХВ-Петербург, 2004.
6. *Конев В.Ю., Мироновский Л.А.* Основные функции пакета MATLAB. Уч. пособие. -СПб., ГААП, 1994.
7. *Мальцев А.И.* Основы линейной алгебры. -М.: Наука, 1970.
8. *Мироновский Л. А.* Моделирование динамических систем. Уч. пособие. СПб., ГААП, 1992.
9. *Мироновский Л. А.* Моделирование разностных уравнений. Уч. пособие. -СПб., ГУАП, 2004.
10. *Половко А.М., Бутусов П.Н.* Matlab для студентов.-СПб: БХВ-Петербург, 2005.
11. *Потемкин В. Г.* MATLAB 6: среда проектирования инженерных приложений. М.: Диалог-МИФИ, 2003.
12. *Сергиенко А.* Цифровая обработка сигналов. -СПб.: Питер, 2002.
13. *Фаддеев Д. К.* Лекции по алгебре: Учебное пособие для вузов. 2-е изд. -СПб.: Издательство "Лань", 2002.
14. *Чен К., Джибмю П., Ирвинг А.* MATLAB в математических исследованиях. -М., Мир. 2001.