Библиотека студента

Юрий Лазарев

MatLAB 5.x

- Операции с числами, векторами и матрицами
- Функции прикладной численной математики
- Построение графиков
- Программирование в среде MatLAB
- Классы объектов в MatLAB
- Цифровая обработка сигналов
- Исследование линейных стационарных систем
- Моделирование нелинейных систем

bhv®

УДК 681.3.06(075.8)
ББК 32.973.26-018.2 Я73
Л17

Рецензенты:

Л.Н. Блохин, доктор техн. наук, профессор (Киевский международный университет гражданской авиации), академик Международной академии навигации и управления, Транспортной академии Украины, Российской академии транспорта и Международной Славянской академии наук.
Ю.И. Калюх, доктор техн. наук, главн. научн. сотр. (Национальная академия управления при Президенте Украины).
Л.С. Глоба, доктор техн. наук, профессор (кафедра систем автоматизированного проектирования НТУУ "Киевский политехнический институт").

Ю.Ф. Лазарев

Л17 **MatLAB 5.x**. — К.: Издательская группа BHV, 2000. — 384 с.
(Серия "Библиотека студента")

ISBN 966-552-068-7
ISBN 5-7315-0096-7

Описаны приемы работы с системой MatLAB версий 5.x. Вы ознакомитесь с интерфейсом программы, основными ее командами и функциями, научитесь выполнять вычисления как в режиме калькулятора, так и в программном режиме. Книга содержит много полезных примеров и рисунков, иллюстрирующих применение процедур. Большое внимание уделено программированию в MatLAB и вопросам взаимодействия системы с другими приложениями. Рассмотрена работа с важными для инженеров пакетами MatLAB — Signal Toolbox, Control и SimuLink.
Для студентов высших технических учебных заведений, научных работников и инженеров.

Обложка А.А. Стеценко

ISBN 966-552-068-7
ISBN 5-7315-0096-7

Предисловие

В последние годы в университетских и инженерно-технических кругах мира (прежде всего — Европы) получает широкое распространение новая компьютерная система проведения математических расчетов — MatLAB (система для инженерных вычислений). В чем же причина такой популярности этой системы? Основные преимущества MatLAB, выгодно выделяющие ее среди существующих ныне математических систем и пакетов (MathCad, Mathematica и др.), заключаются в следующем:

- система MatLAB специально создана для проведения именно инженерных расчетов: математический аппарат, используемый ею, предельно приближен к современному математическому аппарату инженера и ученого и опирается на вычисления с матрицами, векторами и комплексными числами; графическое представление функциональных зависимостей здесь организовано в форме, требуемой именно инженерной документацией;
- язык программирования системы MatLAB очень прост, близок к языку Basic, посилен любому начинающему; он содержит всего несколько десятков операторов; незначительное количество операторов здесь компенсируется большим числом процедур и функций, смысл которых понятен пользователю с соответствующей математической и инженерной подготовкой;
- в отличие от большинства математических систем, MatLAB является открытой системой; это означает, что практически все процедуры и функции MatLAB доступны не только для использования, но и для коррекции и модификации; MatLAB — система, которую пользователь может расширять по своему усмотрению созданными им программами и процедурами (подпрограммами); ее легко приспособить к решению нужных классов задач;
- очень удобна возможность как составлять собственные отдельные программы с целью многократного их использования для исследований, так и применять практически все вычислительные возможности системы в режиме чрезвычайно мощного научного калькулятора; это делает MatLAB незаменимым средством проведения научных расчетных исследований;
- последние версии MatLAB позволяют легко интегрировать ее с текстовым редактором Word, что дает возможность использовать при составлении текстовых документов вычислительные и графические средства MatLAB.

Возможности системы огромны, а по скорости выполнения задач она опережает многие другие подобные системы. Все эти особенности делают систему MatLAB весьма привлекательной для использования в учебном процессе высших учебных заведений.

Система MatLAB создана фирмой MathWork Inc. (США, г. Нейтик, штат Массачусетс). Хотя впервые эту систему начали использовать в конце 70-х годов, настоящий расцвет ее применения наступил в конце 80-х, с появлением на рынке версии 4.0. Одна из последних версий — MatLAB 5.3 — это чрезвычайно развитая система, содержащая огромное количество процедур и функций, необходимых инженеру и научному работнику для осуществления сложных численных расчетов, моделирования поведения технических и физических систем, оформления результатов этих расчетов в наглядном виде.

Предлагаемое издание содержит в основном описание MatLAB версии 5.3. В нем содержатся материалы по практическому овладению процедурами пакетов Control (анализа и синтеза линейных систем автоматического управления), Signal (цифровой обработки сигналов), SimuLink (интерактивного моделирования динамических систем) и некоторыми новыми важными возможностями MatLAB.

Книга состоит из семи глав.

В первой главе читатель ознакомится с возможностями системы при работе в режиме научного калькулятора. Здесь содержатся сведения об основных операторах, командах, функциях и процедурах системы.

Во второй главе описаны правила и примеры составления программ на языке MatLAB. Кроме того, в ней представлены некоторые дополнительные процедуры, позволяющие рационально организовать вычислительный процесс.

Третья глава содержит перечень некоторых процедур и команд общего назначения, которые связывают систему MatLAB с операционной системой компьютера. Там же описано совместное использование редактора Word и системы MatLAB.

Важной частью MatLAB, позволяющей приспосабливать систему к задачам пользователя, является возможность создания новых классов вычислительных объектов. С понятием классов вычислительных объектов в MatLAB и правилами создания новых классов пользователь ознакомится в четвертой главе.

В пятой главе сосредоточены сведения об особенностях использования процедур цифровой обработки сигналов пакета Signal.

Содержание шестой главы — начальное ознакомление с особенностями работы с процедурами анализа и синтеза линейных стационарных систем автоматического управления пакета Control.

Седьмая глава знакомит с пакетом SimuLink, интерактивно моделируемом динамические системы во временной области.

MatLAB предназначена для довольно сложных расчетов. Это и определяет круг ее пользователей: инженеры-проектировщики и разработчики новых устройств, студенты и аспиранты, научные работники, физики и математики. Система приспособлена к любой области науки и техники, содержит средства, которые особенно удобны для электро- и радиотехнических расчетов (операции с комплексными числами, матрицами, векторами и полиномами, обработка данных, анализ сигналов и цифровая фильтрация).

Введение

Система MatLAB (сокращение от MATrix LABoratory — МАТричная ЛАБоратория) является интерактивной системой для выполнения инженерных и научных расчетов, ориентированной на работу с массивами данных. Система использует математический сопроцессор и допускает возможность обращения к программам, написанным на языках FORTRAN, C и C++.

Привлекательной особенностью системы является то, что она содержит развитую встроенную матричную и комплексную арифметику. Система поддерживает выполнение операций с векторами, матрицами и массивами данных, реализует сингулярное и спектральное разложения, расчет ранга и чисел обусловленности матриц, поддерживает работу с алгебраическими полиномами, решение нелинейных уравнений и задач оптимизации, интегрирование в квадратурах, решение дифференциальных и разностных уравнений, построение различных видов графиков, трехмерных поверхностей и линий уровня. В ней реализована удобная операционная среда, позволяющая формулировать проблемы и получать решения в привычной математической форме, не прибегая к рутинному программированию.

Основным объектом системы MatLAB является прямоугольный числовой массив, который допускает комплексные элементы и ввод матриц без явного указания их размеров. Система позволяет решать многие вычислительные задачи за значительно меньшее время, нежели то, которое необходимо для написания соответствующих программ на языках FORTRAN, Basic и C.

Система MatLAB выполняет операции с векторами и матрицами даже в режиме непосредственных вычислений без какого-либо программирования. Ею можно пользоваться как мощнейшим калькулятором, в котором наряду с обычными арифметическими и алгебраическими действиями могут использоваться такие сложные операции, как обращение матрицы, вычисление ее собственных значений и векторов, решение систем линейных алгебраических уравнений и много других. Однако основная отличительная черта системы — это легкость ее модификации и адаптации к конкретным задачам пользователя. Пользователь может ввести в систему любую новую команду, оператор или функцию и пользоваться затем ими так же просто, как и встроенными операторами и функциями. При этом, в отличие от языков программирования, таких как Basic, Pascal или C, нет необходимости в их предварительном описании. Новые программы, функции и процедуры в системе MatLAB сохраняются в виде файлов, имеющих расширение .m. Это делает набор операторов и функций практически неограниченным.

В базовый набор слов системы входят: спецзнаки; знаки арифметических и логических операций; арифметические, тригонометрические и некоторые специальные математические функции; функции быстрого преобразования Фурье и фильтрации; векторные и матричные функции; средства для работы с комплексными числами; операторы построения графиков в декартовой и полярной системах координат, трехмерных поверхностей и т.д. Таким образом, MatLAB предоставляет пользователю большой набор готовых средств (около половины из них — внешние расширения в виде М-файлов).

Система MatLAB имеет собственный язык программирования, напоминающий Basic. Запись программ в системе является традиционной и поэтому привычной для большинства пользователей персональных компьютеров. К тому же система дает возможность редактировать программы при помощи любого привычного для пользователя текстового редактора.

MatLAB имеет большие возможности для работы с сигналами, для расчета и проектирования аналоговых и цифровых фильтров, для построения их частотных, импульсных и переходных характеристик. Имеются в наличии и средства для спектрального анализа и синтеза, в частности, для реализации прямого и обратного преобразования Фурье. Благодаря этому система довольно привлекательна для проектирования электронных устройств.

С системой MatLAB поставляется свыше сотни М-файлов, которые содержат демонстрационные примеры и определения новых операторов и функций. Эта библиотека, все файлы которой подробно прокомментированы, — настоящая сокровищница прекрасных примеров программирования на языке системы. Изучение этих примеров и возможность работы в режиме непосредственных вычислений значительно облегчают знакомство с системой серьезных пользователей, заинтересованных в использовании математических расчетов.

Работа в среде MatLAB может осуществляться в двух режимах:

- в режиме калькулятора, когда вычисления производятся непосредственно после набора очередного оператора или команды MatLAB; при этом значения результатов вычисления могут присваиваться некоторым переменным, либо результаты получаются непосредственно, без присваивания (как в обычных калькуляторах);
- путем вызова программы, составленной и записанной на диске на языке MatLAB, которая содержит все необходимые команды, обеспечивающие ввод данных, организацию вычислений и вывод результатов на экран (программный режим).

В обоих режимах пользователю доступны практически все вычислительные возможности системы, в том числе по выводу информации в графической форме. Программный режим позволяет сохранять разработанные вычислительные алгоритмы и, таким образом, повторять вычисления при других исходных данных.

Глава 1

MatLAB как научный калькулятор

1.1. Командное окно

После вызова MatLAB из среды Windows на экране появляется командное окно среды MatLAB (рис. 1.1).

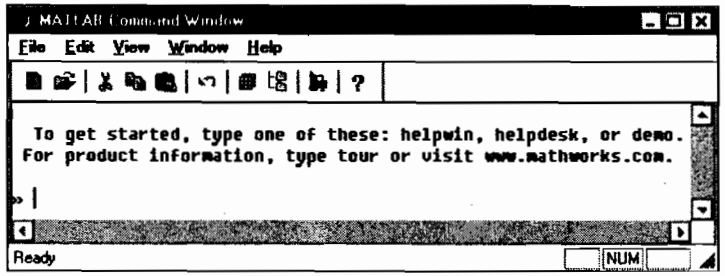

Рис. 1.1

Это окно является основным в MatLAB. В нем отображаются символы команд, которые набираются пользователем на клавиатуре, результаты выполнения этих команд, текст исполняемой программы, а также информация об ошибках выполнения программы, распознанных системой.

Признаком того, что программа MatLAB готова к восприятию и выполнению очередной команды, является наличие в последней строке текстового поля окна знака приглашения (»), после которого стоит мигающая вертикальная черта.

В верхней части окна (под заголовком) размещена строка меню, в которой находятся меню **File**, **Edit**, **Windows**, **Help**. Чтобы открыть какое-либо меню, следует установить на нем указатель мыши и нажать ее левую кнопку. Подробнее команды меню описаны далее, в главе 3 "Интерфейс MatLAB и команды общего назначения. М-книги". Здесь лишь отметим, что для выхода из MatLAB достаточно открыть меню **File** и выбрать в нем команду **Exit MATLAB**.

1.2. Операции с числами

1.2.1. Ввод действительных чисел

Ввод чисел с клавиатуры производится по общим правилам, принятым для языков программирования высокого уровня:

- для отделения дробной части мантиссы числа применяется десятичная точка (вместо запятой при обычной записи);
- десятичный показатель числа записывается в виде целого числа после предварительной записи символа *e*;
- между записью мантиссы числа и символом *e* (который отделяет ее от показателя) не должно быть никаких символов, включая и символ пробела.

Если, например, ввести в командном окне MatLAB строку

```
» 1.20357651e-17
```

то после нажатия клавиши [Enter] в этом окне появится следующая запись (рис. 1.2).

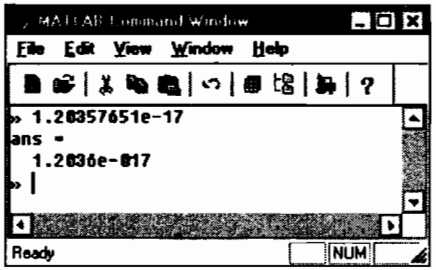

Рис. 1.2

Следует отметить, что результат выводится в виде (формате), который определяется предварительно установленным форматом представления чисел. Этот формат может быть установлен с помощью команды **Preferences** из меню **File** (рис. 1.3). После ее вызова появится одноименное диалоговое окно (рис. 1.4).

Одна из областей этого окна имеет название **Numeric Format**. Она предназначена для установки и изменения формата представления чисел, которые выводятся в командное окно в процессе расчетов. Вы можете выбрать такие форматы:

Short (default)	Краткая запись (применяется по умолчанию)
Long	Длинная запись
Hex	Запись в виде шестнадцатеричного числа
Bank	Запись до сотых долей
Plus	Записывается только знак числа
Short E	Краткая запись в формате с плавающей запятой
Long E	длинная запись в формате с плавающей запятой
Short G	Вторая форма краткой записи в формате с плавающей запятой
Long G	Вторая форма длинной записи в формате с плавающей запятой
Rational	Запись в виде рациональной дроби

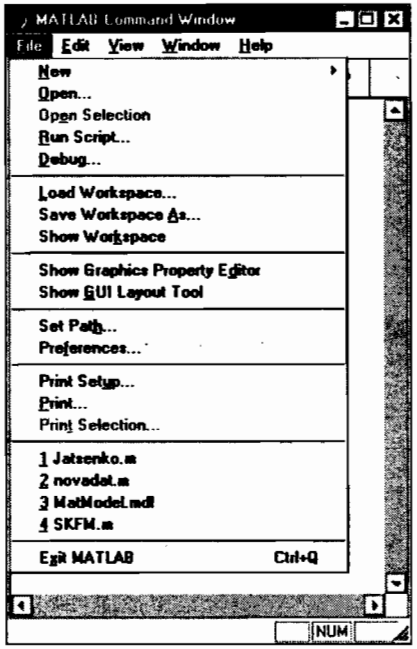

Рис. 1.3

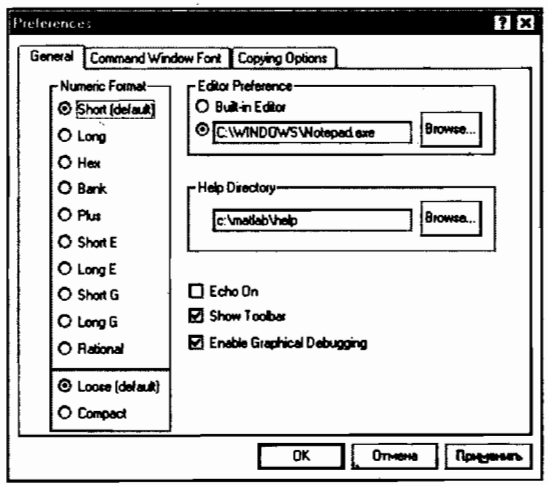

Рис. 1.4

Выбирая нужный вид представления чисел с помощью мыши, можно обеспечить в дальнейшем вывод чисел в командное окно именно в такой форме.

Как видно из рис. 1.2, число, которое выведено на экран, не совпадает с введенным. Это обусловлено только тем, что установленный по умолчанию формат представления чисел (Short) не позволяет вывести больше 6 значащих цифр числа. На самом деле введенное число внутри MatLAB сохраняется со всеми введенными цифрами. Например, если выбрать селекторную кнопку **Long E**, то, повторяя те же действия, получим результат, где все цифры отображены верно (рис. 1.5).

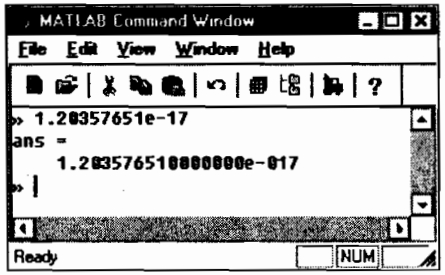

Рис. 1.5

Следует помнить, что:

- введенные числа и результаты всех вычислений в системе MatLAB сохраняются в памяти ПК с относительной погрешностью около 2.e-16 (т.е. с точными значениями в 15 десятичных разрядах);
- диапазон представления модуля действительных чисел лежит в промежутке между 10^{-308} и 10^{+308}.

1.2.2. Простейшие арифметические действия

В арифметических выражениях языка MatLAB применяются следующие знаки арифметических операций:

+	сложение
-	вычитание
*	умножение
/	деление слева направо
\	деление справа налево
^	возведение в степень

Использование MatLAB в режиме калькулятора может происходить путем простой записи в командную строку последовательности

арифметических действий с числами, т.е. обычного арифметического выражения, например:

```
» 4.5^2*7.23 - 3.14*10.4
```

Если после ввода с клавиатуры этой последовательности нажать клавишу [Enter], в командном окне появится результат выполнения в виде, представленном на рис. 1.6, т.е. результат действия последнего выполненного оператора выводится как значение системной переменной **ans**.

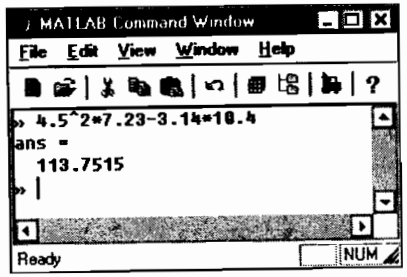

Рис. 1.6

Вообще вывод промежуточной информации в командное окно подчиняется таким правилам:

- если запись оператора не заканчивается символом ";", результат действия этого оператора сразу же выводится в командное окно;
- если оператор заканчивается символом ";", результат его действия не отображается в командном окне;
- если оператор не содержит знака присваивания (=), т.е. является просто записью некоторой последовательности действий над числами и переменными, то значение результата присваивается специальной системной переменной **ans**;
- полученное значение можно использовать в последующих операторах вычислений под именем **ans**; при этом следует помнить, что значение системной переменной **ans** изменяется после действия очередного оператора без знака присваивания;
- в общем случае форма вывода результата в командное окно имеет вид:

```
<имя переменной> = <результат>
```

Пример

Вычислим выражение (25+17)*7. Это можно сделать таким образом. Сначала введите последовательность 25+17 и нажмите клавишу [Enter]. Результат будет представлен в виде: ans = 42. Теперь

введите последовательность ans*7 и нажмите [Enter]. После этого вы получите следующий результат: ans = 294 (рис. 1.7). Чтобы исключить вывод промежуточного результата действия 25+17, достаточно после записи этой последовательности добавить символ ";". Тогда будем иметь результаты в виде, представленном на рис. 1.8.

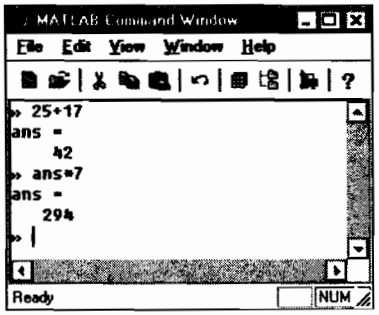

Рис. 1.7

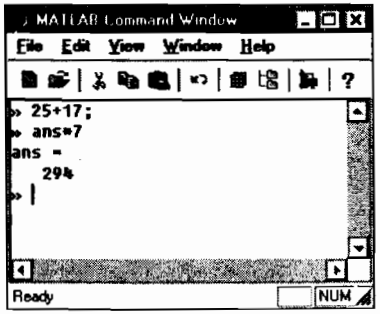

Рис. 1.8

Особенностью MatLAB как калькулятора является возможность использования имен переменных для записи промежуточных результатов в память ПК. Для этого применяется операция присваивания, которая вводится знаком равенства (=) в соответствии со схемой:

```
<имя переменной> = <выражение>[;]
```

Имя переменной может содержать до 19 символов и не должно совпадать с именами функций и процедур системы и системных переменных. При этом система различает в переменных большие и малые буквы. Так, имена amenu, Amenu, aMenu в MatLAB обозначают разные переменные.

Выражение справа от знака присваивания может быть просто числом, арифметическим выражением, строкой символов (тогда эти символы нужно заключить в апострофы) либо символьным выражением. Если выражение не заканчивается символом ";", то после нажатия клавиши [Enter] в командном окне появится результат выполнения в виде:

```
<имя переменной> = <результат>
```

Например, если ввести в командное окно строку x = 25 + 17, на экране появится запись, как на рис. 1.9.

В системе MatLAB имеется несколько имен переменных, которые используются системой и входят в состав зарезервированных. *Эти переменные можно использовать в математических выражениях.*

i, j мнимая единица (корень квадратный из -1)

pi число π (сохраняется в виде 3.141592653589793)
inf обозначение машинной бесконечности
NaN обозначение неопределенного результата (например, типа 0/0 или inf/inf)
ans результат последней операции без знака присваивания

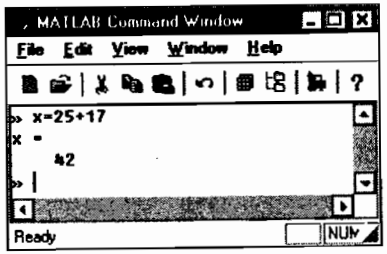

Рис. 1.9

1.2.3. Ввод комплексных чисел

Язык системы MatLAB, в отличие от многих языков программирования высокого уровня, содержит очень простую в пользовании встроенную арифметику комплексных чисел. Большинство элементарных математических функций построено таким образом, что аргументы предполагаются комплексными числами, а результаты также формируются как комплексные числа. Эта особенность языка делает его очень полезным для инженеров и научных работников.

Для обозначения мнимой единицы в MatLAB зарезервировано два имени — **i** и **j**. Ввод с клавиатуры значения комплексного числа производится путем записи в командном окне строки вида:

```
<имя переменной> = <значение ДЧ> + i [j]* <значение МЧ>
```

где *ДЧ* — действительная часть комплексной величины, *МЧ* — мнимая ее часть (рис. 1.10).

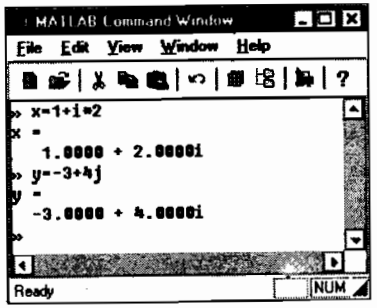

Рис. 1.10

1.2.4. Элементарные математические функции

Общая форма вызова функции в MatLAB имеет следующий вид:

```
<имя результата> = <имя функции>(<список имен аргументов
                                  или их значений>)
```

В языке MatLAB предусмотрены такие элементарные арифметические функции.

Тригонометрические и гиперболические функции

sin(Z)	синус числа Z
sinh(Z)	гиперболический синус
asin(Z)	арксинус (в радианах, в диапазоне от $-\pi/2$ до $+\pi/2$)
asinh(Z)	обратный гиперболический синус
cos(Z)	косинус
cosh(Z)	гиперболический косинус
acos(Z)	арккосинус (в диапазоне от 0 до π)
acosh(Z)	обратный гиперболический косинус
tan(Z)	тангенс
tanh(Z)	гиперболический тангенс
atan(Z)	арктангенс (в диапазоне от $-\pi/2$ до $+\pi/2$)
atan2(X,Y)	четырехквадрантный арктангенс (угол в диапазоне от $-\pi$ до $+\pi$ между горизонтальным правым лучом и лучом, проходящим через точку с координатами X и Y)
atanh(Z)	обратный гиперболический тангенс
sec(Z)	секанс
sech(Z)	гиперболический секанс
asec(Z)	арксеканс
asech(Z)	обратный гиперболический секанс
csc(Z)	косеканс
csch(Z)	гиперболический косеканс
acsc(Z)	арккосеканс
acsch(Z)	обратный гиперболический косеканс
cot(Z)	котангенс
coth(Z)	гиперболический котангенс
acot(Z)	арккотангенс
acoth(Z)	обратный гиперболический котангенс

Экспоненциальные функции

exp(Z)	экспонента числа Z
log(Z)	натуральный логарифм
log10(Z)	десятичный логарифм
sqrt(Z)	квадратный корень из числа Z
abs(Z)	модуль числа Z

Целочисленные функции

fix(Z)	округление до ближайшего целого в сторону нуля
floor(Z)	округление до ближайшего целого в сторону отрицательной бесконечности
ceil(Z)	округление до ближайшего целого в сторону положительной бесконечности
round(Z)	обычное округление числа Z до ближайшего целого
rem(X,Y)	вычисление остатка от деления X на Y
sign(Z)	вычисление сигнум-функции числа Z (0 при Z=0, -1 при Z<0, 1 при Z>0)

1.2.5. Специальные математические функции

Кроме элементарных в языке MatLAB предусмотрен целый ряд специальных математических функций. Ниже приведен перечень и краткое описание этих функций. Правила вызова и использования функций можно найти в их описаниях, которые выводятся на экран, если набрать команду **help** и указать в той же строке имя функции.

Фукции преобразования координат

cart2sph	преобразование декартовых координат в сферические
cart2pol	преобразование декартовых координат в полярные
pol2cart	преобразование полярных координат в декартовые
sph2cart	преобразование сферических координат в декартовые

Функции Бесселя

besselj	функция Бесселя первого рода
bessely	функция Бесселя второго рода
besseli	модифицированная функция Бесселя первого рода
besselk	модифицированная функция Бесселя второго рода

Бета-функции

beta	бета-функция
betainc	неполная бета-функция
betaln	логарифм бета-функции

Гамма-функции

gamma	гамма-функция
gammainc	неполная гамма-функция
gammaln	логарифм гамма-функции

Эллиптические функции и интегралы

ellipj	эллиптические функции Якоби
ellipke	полный эллиптический интеграл
expint	функция экспоненциального интеграла

Функции ошибок

erf	функция ошибок
erfc	дополнительная функция ошибок
erfcx	масштабированная дополнительная функция ошибок
erfinv	обратная функция ошибок

Другие функции

gcd	наибольший общий делитель
lcm	наименьшее общее кратное
legendre	обобщенная функция Лежандра
log2	логарифм по основанию 2
pow2	возведение 2 в указанную степень
rat	представление числа в виде рациональной дроби
rats	представление чисел в виде рациональной дроби

1.2.6. Элементарные действия с комплексными числами

Простейшие действия с комплексными числами — сложение, вычитание, умножение, деление и возведение в степень — осуществляются с помощью обычных арифметических знаков +, -, *, /, \ и ^ соответственно. Примеры использования приведены на рис. 1.11.

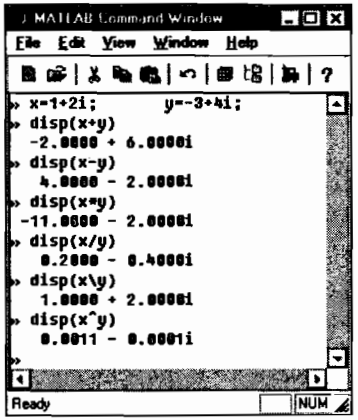

Рис. 1.11

Примечание

В приведенном фрагменте использована функция **disp** (от слова "дисплей"), которая также позволяет выводить в командное окно результаты вычислений или некоторый текст. При этом численный результат, как видно, выводится уже без указания имени переменной или имени **ans**.

1.2.7. Функции комплексного аргумента

Практически все элементарные математические функции, приведенные в разд. 1.2.4, вычисляются при комплексном значении аргумента и получают в результате этого комплексные значения результата.

Благодаря этому, например, функция **sqrt**, в отличие от других языков программирования, вычисляет квадратный корень из отрицательного аргумента, а функция **abs** при комплексном значении аргумента вычисляет модуль комплексного числа. Примеры приведены на рис. 1.12.

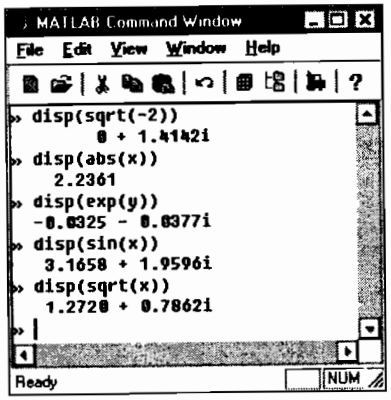

Рис. 1.12

В MatLAB есть несколько дополнительных функций, рассчитанных только на комплексный аргумент (примеры приведены на рис. 1.13):

real(Z)	выделяет действительную часть комплексного аргумента Z
imag(Z)	выделяет мнимую часть комплексного аргумента
angle(Z)	вычисляет значение аргумента комплексного числа Z (в радианах от $-\pi$ до $+\pi$)
conj(Z)	выдает число, комплексно сопряженное относительно Z

Кроме того, в MatLAB есть специальная функция **cplxpair(V)**, которая осуществляет сортировку заданного вектора V с комплексными элементами таким образом, что комплексно-сопряженные пары этих элементов располагаются в выходном векторе в порядке возрастания их действительных частей, при этом элемент с отрицательной мнимой частью всегда располагается первым. Действительные элементы завершают комплексно-сопряженные пары. Например:

```
» v = [-1,-1+2i,-5,4,5i,-1-2i,-5i]
v =
```

```
 Columns 1 through 4
 -1.0000      -1.0000 + 2.0000i      -5.0000       4.0000
 Columns 5 through 7
 0 + 5.0000i  -1.0000 - 2.0000i      0 - 5.0000i

» disp(cplxpair(v))
 Columns 1 through 4
 -1.0000 - 2.0000i  -1.0000 + 2.0000i  0 - 5.0000i  0 + 5.0000i
 Columns 5 through 7
 -5.0000       -1.0000          4.0000
```

Примечание

В этом и последующих примерах команды, которые набираются с клавиатуры, представлены жирным шрифтом, а результат их выполнения — обычным.

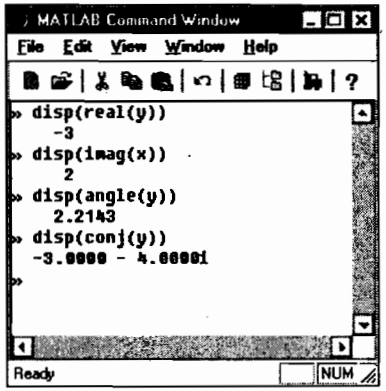

Рис. 1.13

Приспособленность большинства функций MatLAB к оперированию с комплексными числами позволяет значительно проще строить вычисления с действительными числами, результат которых является комплексным (например, находить комплексные корни квадратных уравнений).

1.3. Простейшие операции с векторами и матрицами

MatLAB является системой, которая специально предназначена для осуществления сложных вычислений с векторами, матрицами и полиномами.

Под вектором в MatLAB понимается одномерный массив чисел, а под матрицей — двумерный массив. При этом по умолчанию предполагается, что любая заданная переменная является вектором

или матрицей. Например, отдельное заданное число программа воспринимает как матрицу размером (1*1), а вектор-строку с N элементами — как матрицу размером (1*N).

1.3.1. Ввод векторов и матриц

Исходные значения векторов можно задавать с клавиатуры путем поэлементного ввода. Для этого в строке следует вначале указать имя вектора, потом поставить знак присваивания =, далее — открывающую *квадратную* скобку, а за ней ввести заданные значения элементов вектора, *отделяя их пробелами или запятыми*. Завершается строка закрывающей квадратной скобкой. Например, ввод строки V = [1.2 -0.3 1.2e-5] задает вектор V, содержащий три элемента со значениями 1.2, -0.3 и 1.2e-5 (рис. 1.14).

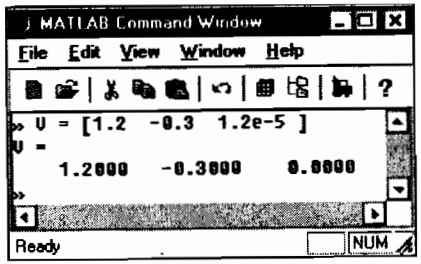

Рис. 1.14

После ввода вектора система выводит его на экран. То, что в приведенном примере последний элемент выведен как 0, обусловлено установленным форматом Short, в соответствии с которым данные выводятся на экран. Длинный вектор можно вводить частями, которые затем объединять с помощью операции *объединения векторов в строку:* v = [v1 v2] (рис. 1.15).

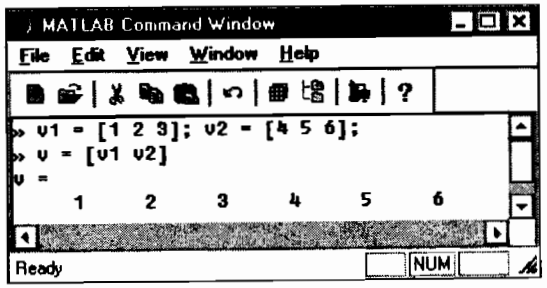

Рис. 1.15

Язык MatLAB дает пользователю возможность *сокращенного ввода вектора, значения элементов которого являются арифметической прогрессией.* Если обозначить: nz — начальное значение этой прогрессии (значение первого элемента вектора); kz — конечное значение прогрессии (значение последнего элемента вектора); h — разность прогрессии (шаг), то вектор можно ввести с помощью короткой записи V = nz : h : kz. Например, ввод строки V = - 0.1 : 0.3 : 1.4 приведет к следующему результату:

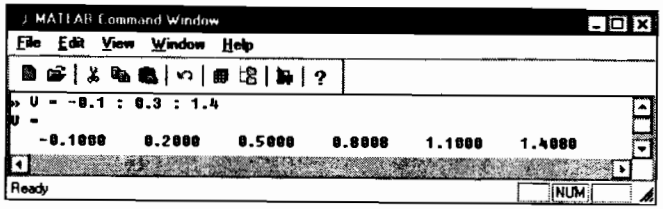

Рис. 1.16

Если средняя величина (разность прогрессии) не указана, то она по умолчанию принимается равной единице. Например, команда

```
» -2.1 : 5
```

приводит к формированию такого вектора

```
ans =
  Columns 1 through 7
   -2.1000 -1.1000 -0.1000  0.9000 1.9000  2.9000  3.9000
  Column 8
   4.9000
```

Таким образом вводятся векторы-строки. Вектор-столбец вводится аналогично, но значения элементов в перечне отделяются знаком ";".

Ввод значений элементов *матрицы* осуществляется в MatLAB в квадратных скобках *по строкам.* При этом элементы строки матрицы отделяются друг от друга пробелом или запятой, а *строки отделяются друг от друга знаком ";"* (рис. 1.17).

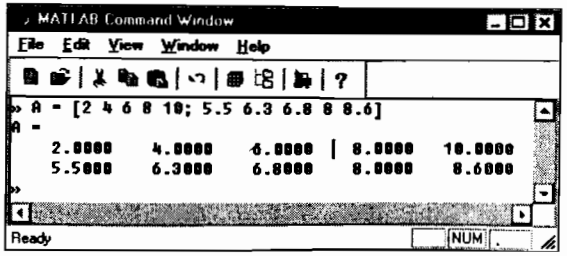

Рис. 1.17

1.3.2. Формирование векторов и матриц

MatLAB имеет несколько функций, которые позволяют формировать векторы и матрицы некоторого определенного вида. Перечислим и проиллюстрируем эти функции:

zeros(M,N) — создает матрицу размером (M*N) с нулевыми элементами:

```
» zeros(3,5)
ans =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

ones(M,N) — создает матрицу размером (M*N) с единичными элементами:

```
» ones(3,5)
ans =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

eye(M,N) — создает матрицу размером (M*N) с единицами по главной диагонали и остальными нулевыми элементами:

```
» eye(3,5)
ans =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
```

rand(M,N) — создает матрицу размером (M*N) из случайных чисел, равномерно распределенных в диапазоне от 0 до 1:

```
» rand(3,5)
ans =
  2.1896e-001  6.7930e-001  5.1942e-001  5.3462e-002  7.6982e-003
  4.7045e-002  9.3469e-001  8.3097e-001  5.2970e-001  3.8342e-001
  6.7886e-001  3.8350e-001  3.4572e-002  6.7115e-001  6.6842e-002
```

randn(M,N) — создает матрицу размером (M*N) из случайных чисел, распределенных по нормальному закону с нулевым математическим ожиданием и стандартным (среднеквадратическим) отклонением, равным единице:

```
» randn(3,5)
ans =
  1.1650e+000  3.5161e-001  5.9060e-002  8.7167e-001  1.2460e+000
  6.2684e-001 -6.9651e-001  1.7971e+000 -1.4462e+000 -6.3898e-001
  7.5080e-002  1.6961e+000  2.6407e-001 -7.0117e-001  5.7735e-001
```

hadamard(N) — создает матрицу Адамара размером (N*N):

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

hilb(N) — создает матрицу Гильберта размером (N*N):

```
» hilb(4)
ans =
  1.0000e+000  5.0000e-001  3.3333e-001  2.5000e-001
  5.0000e-001  3.3333e-001  2.5000e-001  2.0000e-001
  3.3333e-001  2.5000e-001  2.0000e-001  1.6667e-001
  2.5000e-001  2.0000e-001  1.6667e-001  1.4286e-001
```

invhilb(N) — создает обратную матрицу Гильберта размером (N*N):

```
» invhilb(4)
ans =
          16         -120          240         -140
        -120         1200        -2700         1680
         240        -2700         6480        -4200
        -140         1680        -4200         2800
```

pascal(N) — создает матрицу Паскаля размером (N*N):

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

В языке MatLAB предусмотрено несколько функций, которые позволяют формировать матрицу на основе другой (заданной) или используя некоторый заданный вектор. К таким функциям относятся:

fliplr(A) — формирует матрицу, переставляя столбцы известной матрицы A относительно вертикальной оси, например:

```
A =
     1     2     3     4     5     6
     7     8     9    10    11    12
    13    14    15    16    17    18

» fliplr(A)
ans =
     6     5     4     3     2     1
    12    11    10     9     8     7
    18    17    16    15    14    13
```


flipud(A) — формирует матрицу, переставляя строки заданной матрицы A относительно горизонтальной оси:

```
» flipud(A)
ans =
    13    14    15    16    17    18
     7     8     9    10    11    12
     1     2     3     4     5     6
```

rot90(A) — формирует матрицу путем "поворота" заданной матрицы A на 90 градусов против часовой стрелки:

```
» rot90(A)
ans =
     6    12    18
     5    11    17
     4    10    16
     3     9    15
     2     8    14
     1     7    13
```

rechape(A,m,n) — образует матрицу размером (M*N) путем выборки элементов заданной матрицы A по столбцам и последующего распределения этих элементов по n столбцам, каждый из которых содержит m элементов; при этом число элементов матрицы A должно быть равно M*N:

```
» reshape(A,2,9)
ans =
     1    13     8     3    15    10     5    17    12
     7     2    14     9     4    16    11     6    18
```

tril(A) — образует нижнюю треугольную матрицу на основе матрицы A путем обнуления ее элементов выше главной диагонали:

```
» tril(A)
ans =
     1     0     0     0     0     0
     7     8     0     0     0     0
    13    14    15     0     0     0
```

triu(A) — образует верхнюю треугольную матрицу на основе матрицы A путем обнуления ее элементов ниже главной диагонали:

```
» triu(A)
ans =
     1     2     3     4     5     6
     0     8     9    10    11    12
     0     0    15    16    17    18
```

hankel(V) — образует квадратную матрицу Ганкеля, первый столбец которой совпадает с заданным вектором V, например:

```
» V = [-5 6 7 4]
V =
    -5     6     7     4
```

```
» hankel(V)
ans =
    -5     6     7     4
     6     7     4     0
     7     4     0     0
     4     0     0     0
```

Процедура **diag(x)** — формирует или извлекает диагональ матрицы. Если x — вектор, то функция **diag(x)** создает квадратную матрицу с вектором x на главной диагонали:

```
» diag(V)
ans =
    -5     0     0     0
     0     6     0     0
     0     0     7     0
     0     0     0     4
```

Чтобы установить заданный вектор на другую диагональ, при обращении к функции необходимо указать еще один параметр (целое число) — номер диагонали (при этом диагонали отсчитываются от главной вверх), например:

```
» diag(V,-1)
ans =
     0     0     0     0     0
    -5     0     0     0     0
     0     6     0     0     0
     0     0     7     0     0
     0     0     0     4     0
```

Если x — матрица, то функция **diag** создает вектор-столбец, состоящий из элементов главной диагонали заданной матрицы x, например, для матрицы A, указанной перед примером применения процедуры **fliplr**:

```
» diag(A)
ans =
     1
     8
    15
```

Если при этом дополнительно указать номер диагонали, то можно получить вектор-столбец из элементов любой диагонали матрицы x, например:

```
» diag(A,3)
ans =
     4
    11
    18
```

Функция **zeros(1,N)** формирует (создает) вектор-строку из N нулевых элементов. Аналогично **zeros(N,1)** создает вектор-столбец из N нулей.

Векторы, значения элементов которых являются случайными, равномерно распределенными, формируются следующим образом: **rand(1,N)** — для вектора-строки и **rand(M,1)** — для вектора-столбца.

1.3.3. Извлечение и вставка частей матриц

Прежде всего отметим, что *обращение к любому элементу определенной матрицы в MatLAB осуществляется путем указания после имени матрицы (в скобках, через запятую) двух целых положительных чисел, которые определяют соответственно номера строки и столбца матрицы, на пересечении которых расположен этот элемент.*

Пусть имеем некоторую матрицу A:

```
» A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Получить значение элемента этой матрицы, расположенного на пересечении второй строки с третьим столбцом, можно таким образом:

```
» A(2,3)
ans =
     7
```

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

```
» A(2, 3) = pi;  A
A =
    1.0000    2.0000    3.0000    4.0000
    5.0000    6.0000    3.1416    8.0000
    9.0000   10.0000   11.0000   12.0000
```

Иногда требуется создать меньшую матрицу из большей, формируя ее путем извлечения из последней элементов ее нескольких строк и столбцов, или, наоборот, вставить меньшую матрицу таким образом, чтобы она стала определенной частью матрицы большего размера. Это делается в MatLAB с помощью знака двоеточия (:).

Рассмотрим эти операции на примерах.

Пусть требуется создать вектор V1, который состоит из элементов третьего столбца предыдущей матрицы A. Для этого поступим так:

```
» V1 = A(:, 3)
V1 =
    3.0000
    3.1416
   11.0000
```

Создать вектор V2, состоящий из элементов второй строки матрицы A, можно следующим образом:

```
» V2 = A(2, : )
V2 =
    5.0000    6.0000    3.1416    8.0000
```

Допустим, что необходимо из матрицы A создать матрицу B размером (2*2), состоящую из элементов левого нижнего угла матрицы A. Тогда делают следующее:

```
» B = A(2:3, 1:2)
B =
     5     6
     9    10
```

Аналогичным образом можно "вставить" матрицу B в верхнюю середину матрицы A:

```
» A(1:2,2:3)=B
A =
     1     5     6     4
     5     9    10     8
     9    10    11    12
```

Для этого *вместо указания номеров элементов матрицы можно указывать диапазон изменения этих номеров путем определения нижней и верхней границ, разделяя их двоеточием.*

Примечание

Если верхней границей изменения номеров элементов матрицы является ее размер в этом измерении, вместо него можно использовать служебное слово **end**.

Например:

```
» A(2:end,2:end)
ans =
     9    10     8
    10    11    12
```

Эти операции очень удобны для формирования матриц, большинство элементов которых одинаковы, в частности так называемых разреженных матриц, которые состоят в основном из нулей. Для примера рассмотрим формирование разреженной матрицы размером (5*7) с единичными элементами в центре:

```
» A = zeros(5,7);
» B = ones(3,3);
» A(2:4,3:5)=B
A =
     0     0     0     0     0     0     0
     0     0     1     1     1     0     0
     0     0     1     1     1     0     0
     0     0     1     1     1     0     0
     0     0     0     0     0     0     0
```


"Растянуть" матрицу (A) в единый вектор (V) можно с помощью простой записи V = A(:). При этом создается вектор-столбец с количеством элементов (M*N), в котором столбцы исходной матрицы размещены сверху вниз в порядке самих столбцов:

```
» A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
» v = A(:)
v =
     1
     4
     2
     5
     3
     6
```

Наконец, "расширять" матрицу, составляя ее из отдельных заданных матриц ("блоков") можно тоже довольно просто. Если заданы несколько матриц-блоков A1, A2,..., AN с одинаковым числом строк, то из них можно "слепить" единую матрицу A, объединяя блоки в одну "строку" таким образом:

```
A = [A1, A2,..., AN]
```

Эту операцию называют *горизонтальной конкатенацией* (сцеплением) матриц. Аналогично, *вертикальную конкатенацию* матриц можно реализовать при условии, что все составляющие блоки-матрицы имеют одинаковое число столбцов, применяя для отделения блоков вместо запятой — точку с запятой:

```
A = [A1; A2;...; AN]
```

Пример горизонтальной конкатенации:

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

Пример вертикальной конкатенации:

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

1.3.4. Действия над векторами

Выделим две существенно различающиеся группы действий над векторами: *векторные действия* — такие, которые предусмотрены векторным исчислением в математике; *действия по преобразованию элементов* — это действия, которые преобразуют элементы вектора, но не являются операциями, разрешенными математикой с векторами.

Векторные действия над векторами

Сложение векторов. Как известно, складываться (суммироваться) могут только векторы одинакового типа; т.е. оба вектора должны быть либо векторами-строками, либо векторами-столбцами и иметь одинаковую длину (одинаковое число элементов). Если X и Y именно такие векторы, то их сумму Z можно получить, введя команду Z = X + Y, например:

```
» x = [1 2 3];   y = [ 4 5 6];
» v = x + y
v =     5     7     9
```

Аналогично с помощью арифметического знака "-" осуществляется *вычитание векторов*, имеющих одинаковую структуру:

```
» Z = X - Y
```

Например:

```
» v = x - y
v =    -3    -3    -3
```

Транспонирование вектора осуществляется применением знака апострофа, который записывается сразу за записью имени исходного транспонируемого вектора:

```
» x'
ans =
     1
     2
     3
```

Умножение вектора на число осуществляется в MatLAB с помощью знака арифметического умножения (*) таким образом: Z = X*r или Z = r*X, где r — некоторое действительное число.

```
» v = 2*x
  v =     2     4     6
```

Умножение двух векторов определено в математике только для векторов одинакового размера (длины) и лишь тогда, когда один из векторов-сомножителей является строкой, а второй — столбцом. Иначе говоря, если векторы X и Y являются строками, то математический смысл имеют только две формы умножения этих векторов: U = X' * Y и V = X * Y'. Причем в первом случае результатом будет квадратная матрица (диада), а во втором — число. В MatLAB умножение векторов осуществляется с применением обычного знака умножения (*), который записывается между сомножителями-векторами.

Пример:

```
» x = [1 2 3];   y = [ 4 5 6];
» v = x' * y
v =
     4     5     6
     8    10    12
    12    15    18
» v = x * y'
v =    32
```

Для трехкомпонентных векторов в MatLAB существует функция cross, которая позволяет найти *векторное произведение двух векторов*. Если заданы два трехкомпонентных вектора v1 и v2, то для этого достаточно ввести оператор:

```
» cross(v1, v2)
```

Пример:

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

На этом перечень допустимых математических операций с векторами исчерпывается.

Действия по поэлементному преобразованию векторов

В языке MatLAB есть ряд операций, которые преобразуют заданный вектор в другой того же размера и типа, но в то же время не являются математическими операциями с вектором как математическим объектом. Все эти операции преобразуют элементы вектора как элементы обычного одномерного массива чисел. К таким операциям принадлежат, например, *все элементарные математические функции*, приведенные в разд. 1.2.4, которые зависят только от одного аргумента. В языке MatLAB запись вида Y = sin(X), где X — некоторый известный вектор, приводит к формированию нового вектора Y, имеющего тот же тип и размер, но элементы которого равны синусу соответствующих элементов вектора-аргумента X. Приведем примеры:

```
» x = [-2,-1,0,1,2];
» y = sin(x)
y =   -0.9093   -0.8415         0    0.8415    0.9093
» z = tan(x)
z =    2.1850   -1.5574         0    1.5574   -2.1850
» v = exp(x)
v =    0.1353    0.3679    1.0000    2.7183    7.389
```

Кроме этих операций в MatLAB предусмотрено несколько операций поэлементного преобразования, осуществляющихся с помощью знаков обычных арифметических действий. *Все эти операции применяются к векторам одинакового типа и размера. Результатом их является вектор того же типа и размера.*

Добавление (вычитание) числа к (из) каждому(го) элементу(а) вектора. Осуществляется с помощью знака "+" ("-").

Поэлементное умножение векторов. Производится при помощи совокупности знаков ".*", которая записывается между именами перемножаемых векторов. В результате получается вектор, каждый элемент которого является произведением соответствующих элементов векторов-"сомножителей".

Поэлементное деление векторов. Осуществляется при помощи совокупности знаков "./". Результат — вектор, каждый элемент которого является частным от деления соответствующего элемента первого вектора на соответствующий элемент второго вектора.

Поэлементное деление векторов в обратном направлении. Осуществляется с помощью совокупности знаков ".\". В результате получают вектор, каждый элемент которого является частным от деления соответствующего элемента второго вектора на соответствующий элемент первого вектора.

Поэлементное возведение в степень. Осуществляется при помощи совокупности знаков ".^". Результат — вектор, каждый элемент которого является соответствующим элементом первого вектора, возведенным в степень, величина которой равна значению соответствующего элемента второго вектора. Пример:

```
» x = [1,2,3,4,5];  y = [-2,1,4,0,5];
» disp(x + 2)
     3     4     5     6     7
» disp(y - 3)
    -5    -2     1    -3     2
» disp(x.*y)
    -2     2    12     0    25
» disp(x./y)
Warning: Divide by zero
   -0.5000    2.0000    0.7500       Inf    1.0000
» disp(x.\y)
   -2.0000    0.5000    1.3333         0    1.0000
» disp(x.^y)
    1           2          81           1        3125
```

Вышеуказанные операции позволяют очень просто вычислять значения (а затем — строить графики) сложных математических функций, не используя при этом операторы цикла, т.е. производить построение графиков в режиме калькулятора. Для этого достаточно задать значения аргумента как арифметическую прогрессию так, как это было показано в разд. 1.3.1, а затем записать нужную функцию, используя знаки поэлементного преобразования векторов.

Например, пусть требуется вычислить значения функции:

$$y = a \cdot e^{-hx} \cdot \sin x$$

при значениях аргумента x от 0 до 10 с шагом 1. Вычисление массива значений этой функции в указанных условиях можно осуществить при помощи только двух простых операторов:

```
» a = 3; h = 0.5;
» x = 0:1:10;
» y = a * exp(-h*x) .* sin(x)
y =    0    1.5311    1.0035    0.0945   -0.3073   -0.2361
-0.0417    0.0595    0.0544    0.0137   -0.0110
```

1.3.5. Поэлементное преобразование матриц

Для поэлементного преобразования матрицы пригодны все указанные ранее в разд. 1.2.4 алгебраические функции. Каждая такая *функция формирует матрицу того же размера, что и заданная, каждый элемент которой вычисляется как указанная функция от соответствующего элемента заданной матрицы.* Кроме того, в MatLAB определены операции *поэлементного умножения* матриц одинакового размера (с помощью символов ".*", помещаемых между именами "перемножаемых" матриц), *поэлементного деления* (символов "./" и ".\"), *поэлементного возведения в степень* (символов ".^"), когда каждый элемент первой матрицы возводится в степень, равную значению соответствующего элемента второй матрицы. Приведем несколько примеров:

```
» A = [1,2,3,4,5; -2, 3, 1, 4, 0]
A =
     1     2     3     4     5
    -2     3     1     4     0
» B = [-1,3,5,-2,1; 1,8,-3,-1,2]
B =
    -1     3     5    -2     1
     1     8    -3    -1     2
» sin(A)
ans =
    0.8415    0.9093    0.1411   -0.7568   -0.9589
   -0.9093    0.1411    0.8415   -0.7568         0
» A .* B
ans =
    -1     6    15    -8     5
    -2    24    -3    -4     0
» A ./ B
ans =
   -1.0000    0.6667    0.6000   -2.0000    5.0000
   -2.0000    0.3750   -0.3333   -4.0000         0
» A .\ B
Warning: Divide by zero
ans =
   -1.0000    1.5000    1.6667   -0.5000    0.2000
   -0.5000    2.6667   -3.0000   -0.2500       Inf
```

```
» A .^ B
ans =
  1.0e+003 *
    0.0010    0.0080    0.2430    0.0001    0.0050
   -0.0020    6.5610    0.0010    0.0002         0
```

Оригинальной в языке MatLAB является операция прибавления к матрице числа. Она записывается таким образом: A + x или x + A (где A — матрица, а x — число). Такой операции нет в математике. В MatLAB она эквивалентна совокупности операций:

$$A + x * E$$

где E — обозначение матрицы тех же размеров, что и матрица A, состоящей только из единиц. Например:

```
» A = [ 1 2 3 4 5; 6 7 8 9 11 ]
A =
     1     2     3     4     5
     6     7     8     9    11
» A + 2
ans =
     3     4     5     6     7
     8     9    10    11    13
»  2 + A
ans =
     3     4     5     6     7
     8     9    10    11    13
```

1.3.6. *Матричные действия с матрицами*

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

Базовые действия с матрицами — *сложение, вычитание, транспонирование, умножение матрицы на число, умножение матрицы на матрицу, возведение матрицы в целую степень* — осуществляются в языке MatLAB с помощью обычных знаков арифметических операций. При использовании этих операций важно помнить условия, при которых эти операции возможны:

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

Невыполнение этих условий приведет к появлению в командном окне сообщения об ошибке.

Приведем несколько примеров.

Пример сложения и вычитания

```
» A = [ 1 2 3 4 5; 6 7 8 9 11 ]
A =
     1     2     3     4     5
     6     7     8     9    11
» B = [ 0 -1 -2 -3 -4; 5 6 7 8 9 ]
B =
     0    -1    -2    -3    -4
     5     6     7     8     9
» A + B
ans =
     1     1     1     1     1
    11    13    15    17    20
» A - B
ans =
     1     3     5     7     9
     1     1     1     1     2
```

Пример умножения на число

```
» 5*A
ans =
     5    10    15    20    25
    30    35    40    45    55
» A*5
ans =
     5    10    15    20    25
    30    35    40    45    55
```

Пример транспонирования матрицы

```
» A'
ans =
     1     6
     2     7
     3     8
     4     9
     5    11
```

Пример умножения матрицы на матрицу

```
» A' * B
ans =
    30    35    40    45    50
    35    40    45    50    55
    40    45    50    55    60
    45    50    55    60    65
    55    61    67    73    79
» C = A * B'
C =
   -40   115
   -94   299
```

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

Приведем пример:

```
» inv(C)
ans =
 -2.6000e-001  1.0000e-001
 -8.1739e-002  3.4783e-002
```

Проверим правильность выполнения операции обращения, применяя ее еще раз к полученному результату:

```
» inv(ans)
ans =
 -4.0000e+001   1.1500e+002
 -9.4000e+001   2.9900e+002
```

Как видим, мы получили исходную матрицу C, что подтверждает правильность выполнения обращения матрицы.

Возведение матрицы в степень осуществляется в MatLAB при помощи знака "^" (например, A^n). При этом n должно быть целым (положительным или отрицательным) числом. Это матричное действие эквивалентно умножению матрицы A на себя n раз (если n — положительно) либо умножению обратной матрицы на себя (при n отрицательном).

Приведем пример:

```
» A^2
ans =
     8    -3   -10
    -5    10    16
    -2     4     9
» A^(-2)
ans =
  1.5385e-001 -7.6923e-002  3.0769e-001
  7.6923e-002  3.0769e-001 -4.6154e-001
  2.1328e-018 -1.5385e-001  3.8462e-001
```

Весьма оригинальными в языке MatLAB являются две новые, неизвестные в математике функции *деления матриц*. При этом вводятся понятия *деления матриц слева направо* и *деления матриц справа налево*. Первая операция записывается при помощи знака "/", а вторая — при помощи знака "\", которые помещаются между именами матриц, которые делятся друг на друга.

Операция B/A эквивалентна такой последовательности действий: **B * inv(A)**. Здесь функция **inv** осуществляет *обращение матрицы*. Ее удобно использовать для решения матричного уравнения:

$$X * A = B$$

Аналогично операция A\B равносильна совокупности операций **inv(A)*B**, которая является решением матричного уравнения:

$$A * X = B$$

Для примера рассмотрим задачу нахождения корней системы линейных алгебраических уравнений:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 14 \\ 2x_1 - x_2 - 5x_3 &= -15 \\ x_1 - x_2 - x_3 &= -4 \end{aligned}$$

В системе MatLAB это можно сделать таким образом:

```
» A = [ 1 2 3; 2 -1 -5; 1 -1 -1]
A =
     1     2     3
     2    -1    -5
     1    -1    -1
» B = [ 14;-15;-4]
B =
    14
   -15
    -4
» x = A \ B
x =
     1
     2
     3
```

1.3.7. Матричные функции

Вычисление *матричной экспоненты* (e^A) производится с помощью функций **expm**, **expm1**, **expm2** и **expm3**. Эти функции следует отличать от ранее упомянутой функции **exp(A)**, формирующей матрицу, каждый элемент которой равен *e* в степени, равной соответствующему элементу матрицы A.

Функция **expm(A)** является встроенной функцией MatLAB. Функция **expm1(A)** является M-файлом, вычисляющим матричную экспоненту путем использования разложения Паде матрицы A. Функция **expm2(A)** вычисляет матричную экспоненту, используя разложение Тейлора матрицы A. Функция **expm3(A)** вычисляет матричную экспоненту, используя спектральное разложение A.

Приведем примеры использования этих функций:

```
» A = [1,2,3; 0, -1,5;7, -4,1]
A =
     1     2     3
     0    -1     5
     7    -4     1
» expm(A)
ans =
  131.3648   -9.5601   80.6685
   97.8030   -7.1768   59.9309
  123.0245   -8.8236   75.4773
» expm1(A)
ans =
  131.3648   -9.5601   80.6685
   97.8030   -7.1768   59.9309
  123.0245   -8.8236   75.4773
» expm2(A)
ans =
  131.3648   -9.5601   80.6685
   97.8030   -7.1768   59.9309
  123.0245   -8.8236   75.4773
» expm3(A)
```

```
ans =
  1.0e+002 *
   1.3136 + 0.0000i  -0.0956 + 0.0000i   0.8067 - 0.0000i
   0.9780 + 0.0000i  -0.0718 - 0.0000i   0.5993 - 0.0000i
+ 0.0000i  -0.0882 - 0.0000i   0.7548 - 0.0000i
```

Функция **logm(A)** производит обратную операцию — логарифмирование матрицы по натуральному основанию, например:

```
A =
     1     2     3
     0     1     5
     7     4     1
» B = expm3(A)
B =
  1.0e+003 *
    0.9378    0.7987    0.9547
    1.0643    0.9074    1.0844
    1.5182    1.2932    1.5459
» logm(B)
ans =
    1.0000    2.0000    3.0000
    0.0000    1.0000    5.0000
    7.0000    4.0000    1.0000
```

Функция **sqrtm(A)** вычисляет такую матрицу Y, что Y*Y = A:

```
» Y = sqrtm(A)*
Y =
   0.7884 + 0.8806i   0.6717 - 0.1795i   0.8029 - 0.4180i
   0.8953 + 0.6508i   0.7628 + 0.8620i   0.9118 - 1.0066i
   1.2765 - 1.4092i   1.0875 - 0.5449i   1.3000 + 1.2525i
» Y * Y
ans =
   1.0000 + 0.0000i   2.0000 - 0.0000i   3.0000 + 0.0000i
   0.0000 - 0.0000i   1.0000 - 0.0000i   5.0000 - 0.0000i
   7.0000 + 0.0000i   4.0000 + 0.0000i   1.0000 + 0.0000i
```

1.4. Функции прикладной численной математики

1.4.1. Операции с полиномами

В системе MatLAB предусмотрены некоторые дополнительные возможности математического оперирования с полиномами.

Полином (многочлен) как функция определяется выражением:

$$P(x)=a_n \cdot x^n+...+a_2 \cdot x^2+a_1 \cdot x+a_0$$

В MatLAB полином задается и хранится в виде вектора, элементами которого являются коэффициенты полинома от a_n до a_0

$$P = [a_n \ ... \ a_2 \ a_1 \ a_0]$$

Ввод полиномов в MatLAB осуществляется таким же образом, как и ввод вектора длиной n+1, где n — порядок полинома.

Умножение полиномов. Произведением двух полиномов степеней n и m соответственно, как известно, называют полином степени n+m, коэффициенты которого определяют простым перемножением этих двух полиномов. Фактически *операция умножения двух полиномов сводится к построению расширенного вектора коэффициентов по заданным векторам коэффициентов полиномов-сомножителей.* Эту операцию в математике называют *сверткой векторов* (а сам вектор, получаемый в результате такой процедуры, — *вектором-сверткой двух векторов*). В MatLAB ее осуществляет функция **conv(P1, P2)**.

Аналогично, функция **deconv(P1, P2)** осуществляет *деление* полинома P1 на полином P2, т.е. *обратную свертку векторов* P1 и P2. Она определяет коэффициенты полинома, который является частным от деления P1 на P2.

Пример:

```
» p1 = [1,2,3];  p2 = [1,2,3,4,5,6];
» p = conv(p1,p2)
p =     1     4    10    16    22    28    27    18
» deconv(p,p1)
ans =     1     2     3     4     5     6
```

В общем случае деление двух полиномов приводит к получению двух полиномов — полинома-результата (частного) и полинома-остатка. Чтобы получить оба эти полинома, следует оформить обращение к функции таким образом:

```
[Q,R] = deconv(B,A)
```

Тогда результат будет выдан в виде вектора Q с остатком в виде вектора R таких, что будет выполнено соотношение:

```
B = conv(A,Q) + R
```

Система MatLAB имеет функцию **roots(P)**, которая *вычисляет вектор, элементы которого являются корнями заданного полинома* P.

Пусть требуется найти корни полинома:

$$P(x) = x^5 + 8x^4 + 31x^3 + 80x^2 + 94x + 20$$

Ниже показано, как просто это сделать.

```
» p = [1,8,31,80,94,20];
» disp(roots(p))
  -1.0000 + 3.0000i
  -1.0000 - 3.0000i
  -3.7321
  -2.0000
  -0.2679
```

Обратная операция — построение вектора p коэффициентов полинома по заданному вектору его корней — осуществляется функцией **poly**:

```
p = poly(r)
```

Здесь r — заданный вектор значений корней, p — вычисленный вектор полинома.

Приведем пример:

```
» p = [1,8,31,80,94,20]
p =     1     8    31    80    94    20
» r = roots(p)
r =
  -1.0000 + 3.0000i
  -1.0000 - 3.0000i
  -3.7321
  -2.0000
  -0.2679
» p1 = poly(r)
p1 =8.0000   31.0000   80.0000   94.0000   20.0000
```

Заметим, что полученный вектор не показывает старшего коэффициента, который по умолчанию полагается равным единице.

Эта же функция в случае, когда аргументом ее является некоторая квадратная матрица A размером (N*N), строит вектор характеристического полинома этой матрицы. Обращение

```
p = poly(A)
```

формирует вектор p коэффициентов полинома:

$$p(s) = \det(s*E - A) = p_1*s^n + \ldots + p_n*s + p_{n+1}$$

где E — обозначение единичной матрицы размером (N*N).

Рассмотрим пример:

```
» A = [1 2 3; 5 6 0; -1 2 3]
A =
     1     2     3
     5     6     0
    -1     2     3
» p = poly(A)
p =
    1.0000  -10.0000   20.0000  -36.0000
```

Для *вычисления значения полинома по заданному значению его аргумента* в MatLAB предусмотрена функция **polyval**. Обращение к ней происходит по схеме:

```
y = polyval(p,x)
```

где p — заданный вектор коэффициентов полинома, а x — заданное значение аргумента. Пример:

```
» y = polyval(p,2)
y = 936
```

Если как аргумент полинома указана матрица X, то функция **polyval(p,X)** вычисляет матрицу, элементы которой являются значениями указанного полинома при аргументе, равном соответствующему элементу матрицы X, например:


```
p =  1   8  31  80  94  20
» X = [1 2 3; 0 -1 3; 2 2 -1]
X =
     1     2     3
     0    -1     3
     2     2    -1
» disp(polyval(p,X))
         234         936        2750
          20         -18        2750
         936         936         -18
```

В этом случае функция вычисляет значения полинома для каждого элемента матрицы X, и поэтому размеры входной и выходной матриц одинаковы size(Y) = size(X).

Вычисление производной от полинома производится функцией **polyder**. Эта функция создает вектор коэффициентов полинома, представляющего собой производную от заданного полинома. Она имеет три вида обращений:

- **dp = polyder(p)** по заданному полиному p вычисляет вектор dp, элементы которого являются коэффициентами полинома-производной от заданного:

```
» dp = polyder(p)
dp =     5    32    93   160    94;
```

- **dp = polyder(p1,p2)** вычисляет вектор dp, элементами которого являются коэффициенты полинома-производной от произведения полиномов p1 и p2:

```
» p1 = [1,8,31,80,94,20];
» p2 = [1,2,16];
» p = conv(p1,p2)
p =
  Columns 1 through 6
    1    10    63   270   750  1488
  Columns 7 through 8
  1544   320
» dp = polyder(p)
dp =
  Columns 1 through 6
  7    60   315  1080  2250  2976
  Column 7
  1544
» dp1 = polyder(p1,p2)
dp1 =
  Columns 1 through 6
  7    60   315  1080  2250  2976  1544
```

- **[q,p] = polyder(p1,p2)** вычисляет производную от отношения (p1/p2) двух полиномов p1 и p2 и выдает результат в виде отношения (q/p) полиномов q и p:

```
» p1 = [1,8,31,80,94,20];
» p2 = [1,2,16];
» [q,p] = polyder(p1,p2)
```

```
q = 3    24    159    636    1554    2520    1464
p = 1    4     36     64     256
» z = deconv (q,p)
z =      3     12     3
» y = deconv(p1,p2)
y =      1     6      3    -22
» z1 = polyder(y)
z1 =     3     12     3
```

1.4.2. Обработка данных измерений (операции с векторами и матрицами как с массивами данных)

Система MatLAB дает пользователю дополнительные возможности для обработки данных, которые заданы в векторной или матричной форме.

Допустим, что есть некоторая зависимость y(x), которая задана рядом точек:

x	2	4	6	8	10
y	5.5	6.3	6.8	8	8.6

Ее можно задать в командном окне MatLAB как матрицу xydata, содержащую две строки — значения x и значения y:

```
» xydata =[2 4 6 8 10; 5.5 6.3 6.8 8 8.6]
xydata =
    2.0000    4.0000    6.0000    8.0000   10.0000
    5.5000    6.3000    6.8000    8.0000    8.6000
```

На примере этой зависимости рассмотрим основные средства для обработки данных.

Функция **size(xydata)** предназначена для определения числа строк и столбцов матрицы xydata. Она формирует вектор [n, p], содержащий эти величины:

```
» size(xydata)
ans =
     2     5
```

Обращение к ней вида:

```
[n, p] = size(xydata)
```

позволяет сохранить в памяти и использовать затем при дальнейших вычислениях данные о количестве строк (n) и числе столбцов (p) этой матрицы:

```
» n, p
n =
     2
p =
     5
```

С помощью этой функции можно установить длину и тип (строка или столбец) вектора:


```
» v = xydata(:)
v =
    2.0000
    5.5000
    4.0000
    6.3000
    6.0000
    6.8000
    8.0000
    8.0000
   10.0000
    8.6000
» n = size(v)
n =    10     1
» v1 = v'
v1 =
  Columns 1 through 7
2.0000  5.5000  4.0000  6.3000  6.0000  6.8000  8.0000
  Columns 8 through 10
    8.0000   10.0000    8.6000
»  size(v')
 ans =     1    10
```

Функция **max(V)**, где V — некоторый вектор, выдает значение максимального по значению элемента этого вектора. Аналогично, функция **min(V)** извлекает минимальный элемент вектора V. Функции **mean(V)** и **std(V)** определяют, соответственно, среднее значение и среднеквадратическое отклонение от него элементов вектора V.

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

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

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

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

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

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

Применение описанных функций проиллюстрировано ниже.

```
» v = [ 1, 0.1, 0.5, 0.1, 0.1,0.4 ];
» disp(size(v))
     1     6
» disp(max(v))
     1
» disp(min(v))
    0.1000
» disp(mean(v))
    0.3667
» disp(std(v))
    0.3559
```

```
» disp(sort(v))
  0.1000    0.1000    0.1000    0.4000    0.5000    1.0000
» disp(sum(v))
    2.2000
» disp(prod(v))
  2.0000e-004
» disp(cumsum(v))
  1.0000    1.1000    1.6000    1.7000    1.8000    2.2000
» disp(cumprod(v))
  1.0000    0.1000    0.0500    0.0050    0.0005    0.0002
» disp(diff(v))
   -0.9000    0.4000   -0.4000         0    0.3000
```

Если указать второй выходной параметр, то можно получить дополнительную информацию о первом индексе элемента, значение которого является максимальным или минимальным:

```
» [M,n]=max(v)
M =
     1
n =
     1
» [N,m]=min(v)
N =
    0.1000
m =
     2
```

Интегрирование методом трапеций осуществляет процедура **trapz**. Обращение к ней вида **trapz(x,y)** приводит к вычислению площади под графиком функции y(x), в котором все точки, заданные векторами x и y, соединены отрезками прямых. Если первый вектор x не указан в обращении, по умолчанию допускается, что шаг интегрирования является постоянным и равен единице (т.е. вектор x является вектором из номеров элементов вектора y).

Пример

Вычислим интеграл от функции y = sin(x) в диапазоне от 0 до π. Его точное значение равно 2. Возьмем равномерную сетку из 100 элементов. Тогда вычисления сведутся к совокупности операций:

```
» x = 0 : pi/100 : pi;
» y = sin(x);
» disp(trapz(x,y))
    1.9998
```

Те же функции **size**, **max**, **min**, **mean**, **std**, **sort**, **sum**, **prod**, **cumsum**, **cumprod**, **diff** могут быть применены и к матрицам. Основным отличием использования в качестве аргументов этих функций именно матриц является то, что соответствующие описанные выше операции производятся не по отношению к строкам матриц, а по отношению к каждому из столбцов заданной матрицы (за исключением, конечно, функции **size**), т.е. каждый столбец матрицы A рассматривается как переменная, а каждая строка — как наблюдение. Так, в результате

применения функций **max**, **min**, **mean**, **std** получаются векторы-строки с числом элементов, равным числу столбцов заданной матрицы. Каждый элемент содержит, соответственно, максимальное, минимальное, среднее или среднеквадратическое значение элементов соответствующего столбца заданной матрицы. Приведем примеры. Пусть имеем 3 величины y1, y2 и y3, измеренные при некоторых пяти одинаковых значениях аргумента (которые не указаны). Тогда данные измерений образуют 3 вектора по 5 элементов:

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

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

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

Применим к этой матрице описанные выше функции. Получим:

```
» size(A)
ans =     5     3
» max(A)
ans =    8.6000    1.0000   10.3000
» min(A)
ans =
   5.5000   -1.2000         0
» mean(A)
ans =
   7.0400   -0.0400    5.5400
» std(A)
ans =
   1.2582    0.8735    4.0655
```

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

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

Функция **sort** сортирует элементы каждого из столбцов исходной матрицы. Результатом является матрица того же размера.

Функции **sum** и **prod** формируют вектор-строку, каждый элемент которой является суммой или произведением элементов соответствующего столбца исходной матрицы. Функции **cumsum**, **cumprod** образуют матрицы того же размера, элементы каждого столбца которых являются суммой или произведением элементов этого же столбца исходной матрицы, лежащих выше соответствующего элемента. Наконец, функция **diff** создает из заданной матрицы размером (m*n) матрицу размером ((m-1)*n), элементы которой представляют собой разность между элементами соседних строк исходной матрицы.

Применяя эти процедуры к той же матрице измерений, получим:

```
» sort(A)
ans =
    5.5000   -1.2000         0
    6.3000   -0.6000    3.4000
    6.8000    0.1000    5.6000
    8.0000    0.5000    8.4000
    8.6000    1.0000   10.3000
» sum(A)
ans =   35.2000   -0.2000   27.7000
» prod(A)
ans =  1.0e+004 *
    1.6211    0.0000         0
» cumsum(A)
ans =
    5.5000   -1.2000    3.4000
   11.8000   -0.7000    9.0000
   18.6000   -1.3000    9.0000
   26.6000   -0.3000   17.4000
   35.2000   -0.2000   27.7000
» cumprod(A)
ans =  1.0e+004 *
    0.0006   -0.0001    0.0003
    0.0035   -0.0001    0.0019
    0.0236    0.0000         0
    0.1885    0.0000         0
    1.6211    0.0000         0
» diff(A)
ans =
    0.8000    1.7000    2.2000
    0.5000   -1.1000   -5.6000
    1.2000    1.6000    8.4000
    0.6000   -0.9000    1.9000
```

Рассмотрим некоторые другие функции, предоставляемые пользователю системой MatLAB.

Функция **cov(A)** вычисляет *матрицу ковариаций* измерений. При этом получается квадратная симметричная матрица с числом строк и столбцов, равным числу измеренных величин, т.е. числу столбцов матрицы измерений. Например, при применении к принятой матрице измерений она дает такой результат:

```
» cov(A)
ans =
    1.5830    0.6845    3.6880
    0.6845    0.7630    2.3145
    3.6880    2.3145   16.5280
```

На диагонали матрицы ковариаций размещены дисперсии измеренных величин, а вне ее — взаимные корреляционные моменты этих величин.

Функция **corrcoeff(A)** вычисляет *матрицу коэффициентов корреляции* при тех же условиях. Элементы матрицы S = corrcoef(A) связаны с элементами матрицы ковариаций C = cov(A) следующим соотношением:

$$S(k,l) = \frac{C(k,l)}{\sqrt{C(k,k) \cdot C(l,l)}}$$

Пример:

```
» corrcoef(A)
ans =
    1.0000    0.6228    0.7210
    0.6228    1.0000    0.6518
    0.7210    0.6518    1.0000
```

1.4.3. Функции линейной алгебры

Традиционно к линейной алгебре относят такие задачи, как обращение и псевдообращение матрицы, спектральное и сингулярное разложение матриц, вычисление собственных значений и векторов, сингулярных чисел матриц, вычисление функций от матриц. Кратко ознакомимся с некоторыми основными функциями MatLAB в этой области.

Функция **k = cond(A)** вычисляет и выдает число обусловленности матрицы по отношению к операции обращения, которое равно отношению максимального сингулярного числа матрицы к минимальному.

Функция **k = norm(v,p)** вычисляет p-норму вектора v по формуле:

$$k = \text{sum}(\text{abs}(v) \ .\hat{}\ p)\hat{}\ (1/p)$$

где p — целое положительное число. Если аргумент p при обращении к функции не указан, вычисляется 2-норма.

Функция **k = norm(A,p)** вычисляет p-норму матрицы A по формуле:

$$k = \max(\text{sum}(\text{abs}(A) \ .\hat{}\ p))\hat{}\ (1/p)$$

где p = 1,2, 'fro' или inf. Если аргумент p не указан, вычисляется 2-норма. При этом справедливы соотношения:

$$\text{norm}(A,1) = \max(\text{sum}(\text{abs}(A)))$$
$$\text{norm}(A,\text{inf}) = \max(\text{sum}(\text{abs}(A')))$$
$$\text{norm}(A,\text{'fro'}) = \text{sqrt}(\text{sum}(\text{diag}(A'*A)))$$
$$\text{norm}(A) = \text{norm}(A,2) = \sigma_{max}(A)$$

Функция **к = rcond(A)** вычисляет величину, обратную значению числа обусловленности матрицы A относительно 1-нормы. Если матрица A хорошо обусловлена, значение **к** близко к единице. Если же она плохо обусловлена, **к** приближается к нулю.

Функция **r = rank(A)** вычисляет ранг матрицы, который определяется как число сингулярных чисел матрицы, превышающих порог

```
max(size(A))*norm(A)*eps
```

Приведем примеры применения этих функций:

```
A =
     1     2     3
     0     1     5
     7     4     1
» disp(cond(A))
   13.8032
» disp(norm(A,1))
    9
» disp(norm(A))
    8.6950
» disp(rcond(A))
    0.0692
» disp(rank(A))
    3
```

Процедура **d = det(A)** вычисляет *определитель квадратной матрицы* на основе треугольного разложения методом исключения Гаусса.

Функция **t = trace(A)** вычисляет *след матрицы* A, который равен сумме ее диагональных элементов.

Q = null(A) вычисляет *ортонормальный базис нуль-пространства* матрицы A. **Q = orth(A)** выдает *ортонормальный базис* матрицы A.

Процедура **R = rref(A)** *приводит матрицу к треугольному виду на основе метода исключения Гаусса* с частичным выбором ведущего элемента.

Примеры:

```
» disp(det(A))
    30
» disp(trace(A))
     3
» disp(null(A))
» disp(orth(A))
    0.3395    0.4082   -0.8474
    0.2793    0.8165    0.5053
    0.8982   -0.4082    0.1632
```



```
» disp(rref(A))
     1     0     0
     0     1     0
     0     0     1
```

Функция **R = chol(A)** находит *разложение Холецкого* для действительных симметричных и комплексных эрмитовых матриц:

```
» A = [ 1 2 3; 2 15 8; 3 8 400]
A =
     1     2     3
     2    15     8
     3     8   400
» disp(chol(A))
    1.0000    2.0000    3.0000
         0    3.3166    0.6030
         0         0   19.7645
```

Функция **lu(A)** осуществляет *LU-разложение* матрицы A в виде произведения нижней треугольной матрицы L (возможно, с перестановками) и верхней треугольной матрицы U, так что A = L * U.

Обращение к этой функции вида:

```
[ L, U, P ] = lu(A)
```

позволяет получить три составляющие этого разложения — нижнюю треугольную матрицу L, верхнюю треугольную U и матрицу перестановок P такие, что:

$$P * A = L * U$$

Приведем пример:

```
A =
     1     2     3
     2    15     8
     3     8   400
» disp(lu(A))
    3.0000    8.0000  400.0000
   -0.6667    9.6667 -258.6667
   -0.3333    0.0690 -148.1724
» [ L, U, P] = lu(A);
» L
L =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333   -0.0690    1.0000
» U
U =
    3.0000    8.0000  400.0000
         0    9.6667 -258.6667
         0         0 -148.1724
» P
P =
     0     0     1
     0     1     0
     1     0     0
```

Из него следует, что в первом, упрощенном варианте вызова функция выдает комбинацию из матриц L и U.

Обращение матрицы осуществляется с помощью функции **inv(A)**:

```
» disp(inv(A))
    1.3814   -0.1806   -0.0067
   -0.1806    0.0910   -0.0005
   -0.0067   -0.0005    0.0026
```

Процедура **pinv(A)** находит матрицу, псевдообратную матрице A, которая имеет размеры матрицы A' и удовлетворяет условиям:

$$A * P * A = A; \quad P * A * P = P$$

Например:

```
A =
     1     2     3     4     5
     5    -1     4     6     0
» P = pinv(A)
P =
   -0.0423    0.0852
    0.0704   -0.0480
    0.0282    0.0372
    0.0282    0.0628
    0.1408   -0.0704
» A*P*A       % проверка 1
ans =
    1.0000    2.0000    3.0000    4.0000    5.0000
    5.0000   -1.0000    4.0000    6.0000    0.0000
» P*A*P       % проверка 2
ans =
   -0.0423    0.0852
    0.0704   -0.0480
    0.0282    0.0372
    0.0282    0.0628
    0.1408   -0.0704
```

Для квадратных матриц эта операция совпадает с обычным обращением.

Процедура **[Q, R, P] = qr(A)** осуществляет разложение матрицы A на три — унитарную матрицу Q, верхнюю треугольную R с уменьшающимися по модулю диагональными элементами и матрицу перестановок P, такие, что:

$$A * P = Q * R$$

Например:

```
A =
     1     2     3     4     5
     5    -1     4     6     0
» [Q,R,P] = qr(A)
Q =
   -0.5547   -0.8321
   -0.8321    0.5547
```



```
R =
   -7.2111   -2.7735   -4.9923   -4.7150   -0.2774
         0   -4.1603   -0.2774    1.9415   -2.2188
P =
     0     0     0     1     0
     0     0     0     0     1
     0     0     1     0     0
     1     0     0     0     0
     0     1     0     0     0
```

Определение характеристического полинома матрицы A можно осуществить при помощи функции **poly(A)**. Обращение к ней вида **p = poly(A)** дает возможность найти вектор-строку p коэффициентов характеристического полинома:

$$p(s) = \det(s*E - A) = p_1*s^n + \ldots + p_n*s + p_{n+1}$$

где E — обозначение единичной матрицы размером (n*n). Например:

```
» A = [1 2 3; 5 6 0; -1 2 3]
A =
     1     2     3
     5     6     0
    -1     2     3
» p = poly(A)
p =
    1.0000  -10.0000   20.0000  -36.0000
```

Вычисление собственных значений и собственных векторов матрицы осуществляет процедура **eig(A)**. Обычное обращение к ней позволяет получить вектор собственных значений матрицы A, т.е. корней характеристического полинома матрицы. Если же обращение имеет вид:

```
[ R, D ] = eig(A)
```

то в результате получают диагональную матрицу D собственных значений и матрицу R правых собственных векторов, удовлетворяющих условию:

$$A * R = R * D$$

Эти векторы пронормированы так, что норма каждого из них равна единице. Приведем пример:

```
A =
     1     2     3
    -1     8    16
    -5   100     3
» disp(eig(A))
    1.2234
   45.2658
  -34.4893
» [ R,D] = eig(A)
R =
    0.9979   -0.0798   -0.0590
```

```
    0.0492   -0.3915   -0.3530
    0.0416   -0.9167    0.9338
D =
    1.2234         0         0
         0   45.2658         0
         0         0  -34.4893
```

Сингулярное разложение матрицы производит процедура **svd(A)**. Упрощенное обращение к ней позволяет получить сингулярные числа матрицы A. Более сложное обращение вида:

```
[ U, S, V ] = svd(A)
```

позволяет получить три матрицы: U — сформированную из ортонормированных собственных векторов, отвечающих наибольшим собственным значениям матрицы $A*A^T$; V — из ортонормированных собственных векторов матрицы A^T*A и S — диагональную матрицу из *неотрицательных значений квадратных корней из собственных значений матрицы* A^T*A (их называют *сингулярными числами*). Эти матрицы удовлетворяют соотношению:

$$A = U * S * V^T$$

Рассмотрим пример:

```
» disp(svd(A))
  100.5617
   15.9665
    1.1896
» [U,S,V] = svd(A)
U =
   -0.0207    0.1806   -0.9833
   -0.0869    0.9795    0.1817
   -0.9960   -0.0892    0.0045
S =
  100.5617         0         0
         0   15.9665         0
         0         0    1.1896
V =
    0.0502   -0.0221   -0.9985
   -0.9978   -0.0453   -0.0491
   -0.0442    0.9987   -0.0243
```

Приведение матрицы к форме Гессенберга осуществляется с помощью процедуры **hess(A)**:

```
A =
     1     2     3
    -1     8    16
    -5   100     3
» disp(hess(A))
    1.0000   -3.3340   -1.3728
    5.0990   25.5000   96.5000
         0   12.5000  -14.5000
```


Более развернутое обращение **[P,H] = hess(A)** дает возможность получить, кроме матрицы H в верхней форме Гессенберга, также унитарную матрицу преобразований P, которая удовлетворяет условиям:

$$A = P * H * P; \quad P' * P = \text{eye}(\text{size}(A))$$

Пример:

```
» [P,H] = hess(A)
P =
    1.0000         0         0
         0   -0.1961   -0.9806
         0   -0.9806    0.1961
H =
    1.0000   -3.3340   -1.3728
    5.0990   25.5000   96.5000
         0   12.5000  -14.5000
```

Процедура **schur (A)** *преобразует матрицу к форме Шура*. Упрощенное обращение к ней приводит к получению матрицы в форме Шура.

Комплексная форма Шура — это верхняя треугольная матрица с собственными значениями на диагонали. *Действительная форма Шура* сохраняет на диагонали только действительные собственные значения, а комплексные изображаются в виде блоков (2*2), частично занимая нижнюю поддиагональ.

Обращение **[U,T] = schur(A)** позволяет, кроме матрицы T Шура, получить также унитарную матрицу U, удовлетворяющую условиям:

$$A = U * H * U'; \quad U' * U = \text{eye}(\text{size}(A))$$

Если исходная матрица A является действительной, то результатом будет *действительная форма Шура*, если же комплексной — то результат выдается в виде *комплексной формы Шура*. Приведем пример:

```
» disp(schur(A))
    1.2234   -6.0905   -4.4758
         0   45.2658   84.0944
         0    0.0000  -34.4893
» [U,T] = hess(A)
U =
    1.0000         0         0
         0   -0.1961   -0.9806
         0   -0.9806    0.1961
T =
    1.0000   -3.3340   -1.3728
    5.0990   25.5000   96.5000
         0   12.5000  -14.5000
```

Функция **[U,T] = rsf2csf(U,T)** преобразует действительную квазитреугольную форму Шура в комплексную треугольную:

```
» [U,T] = rsf2csf(U,T)
U =
   -0.9934   -0.1147         0
   -0.0449    0.3892   -0.9201
   -0.1055    0.9140    0.3917
T =
    1.4091   -8.6427   10.2938
         0   45.1689  -83.3695
         0         0  -34.5780
```

Процедура **[AA, BB, Q, Z, V] = qz(A,B)** приводит *пару матриц* A и B к *обобщенной форме Шура*. При этом AA и BB являются комплексными верхними треугольными матрицами, Q, Z — матрицами приведения, а V — вектором обобщенных собственных векторов такими, что:

$$Q * A * Z = AA; \quad Q * B * Z = BB$$

Обобщенные собственные значения могут быть найдены, исходя из следующего условия:

$$A * V * \mathrm{diag}(BB) = B * V * \mathrm{diag}(AA)$$

Необходимость в одновременном приведении пары матриц к форме Шура возникает во многих задачах линейной алгебры: решении матричных уравнений Сильвестра и Риккати, смешанных систем дифференциальных и линейных алгебраических уравнений.

Пример

Пусть задана система обыкновенных дифференциальных уравнений в неявной форме Коши с одним входом u и одним выходом y такого вида:

$$Q \cdot \dot{x} + R \cdot x = b \cdot u$$
$$y = c \cdot x + d \cdot u$$

причем матрицы Q, R и векторы b, c и d равны:

```
Q =
    1.0000         0
    0.1920    1.0000
R =
    1.1190   -1.0000
   36.4800    1.5380
b =
   31.0960
    0.1284
c =
    0.6299         0
d = -0.0723
```

Необходимо вычислить значения полюсов и нулей соответствующей передаточной функции.

Эта задача сводится к отысканию собственных значений λ, удовлетворяющих матричным уравнениям:

$$R \cdot r = -\lambda \cdot Q \cdot r$$
$$\begin{bmatrix} -R & b \\ c & d \end{bmatrix} \cdot r = \lambda \cdot \begin{bmatrix} Q & 0 \\ 0 & 0 \end{bmatrix} \cdot r$$

Решение первого уравнения позволяет *вычислить полюсы передаточной функции*, а второго — *нули*.

Ниже приведена совокупность операторов, приводящая к *расчету полюсов:*

```
» [AA, BB] = qz(R,-Q) % Приведение матриц к форме Шура
AA =
   5.5039 + 2.7975i  24.8121 -25.3646i
   0.0000 - 0.0000i   5.5158 - 2.8036i
BB =
  -0.6457 + 0.7622i  -0.1337 + 0.1378i
        0            -0.6471 - 0.7638i
» diag(AA) ./diag(BB) % Расчет полюсов
ans =
  -1.4245 - 6.0143i
  -1.4245 + 6.0143i
```

Расчет нулей осуществляется таким образом:

```
» A = [-R b                  % Формирование
       c  d]                 % первой матрицы
A =
   -1.1190    1.0000    0.1284
  -36.4800   -1.5380   31.0960
    0.6299    0        -0.0723
» B = [-Q                    zeros(size(b))      % Формирование
       zeros(size(c))  0                 ]      % второй матрицы
B =
   -1.0000         0         0
   -0.1920   -1.0000         0
         0         0         0
» [AA,BB] = qz(A,B)                  % Приведение матриц
                                     % к форме Шура
AA =
   31.0963   -0.7169  -36.5109
    0.0000    1.0647    0.9229
         0    0.0000    0.5119
BB =
         0    0.9860   -0.2574
         0    0.0657    0.9964
         0         0   -0.0354
» diag(AA) ./diag(BB)    % Вычисление нулей
ans =
       Inf
   16.2009
  -14.4706
```

Вычисление *собственных значений матричного полинома* осуществляет процедура **polyeig**.

Обращение

```
[R, d] = polyeig(A0, A1,..., Ap)
```

позволяет решить полную проблему собственных значений для матричного полинома степени p вида:

$$(A_0 + \lambda * A_1 + ... + \lambda^p * A_p) * r = 0$$

Входными переменными этой процедуры являются p+1 квадратные матрицы A_0, A_1, ... A_p порядка n. Выходными переменными — матрица собственных векторов R размером (n*(n*p)) и вектор d собственных значений размером (n*p).

Функция **polyvalm** предназначена для *вычисления матричного полинома* вида:

$$Y(X) = p_n \cdot X^n + p_{n-1} \cdot X^{n-1} + ... + p_2 \cdot X^2 + p_1 \cdot X + p_0$$

по заданному значению матрицы X и вектора p = [p_n, p_{n-1}, ..., p_0] коэффициентов полинома. Для этого достаточно обратиться к этой процедуре по схеме:

```
Y = polyvalm(p, X)
```

Пример:

```
p =     1     8    31    80    94    20
» X
X =
     1     2     3
     0    -1     3
     2     2    -1
» disp(polyvalm(p,X))
        2196        2214        2880
         882         864        1116
        1332        1332        1746
```

Примечание

Следует различать процедуры **polyval** и **polyvalm**. Первая вычисляет значения полинома для каждого из элементов матрицы аргумента, а вторая при вычислении полинома возводит в соответствующую степень всю матрицу аргумента.

Процедура **subspace(A,B)** вычисляет угол между двумя подпространствами, которые "натянуты на столбцы" матриц A и B. Если аргументами являются не матрицы, а векторы A и B, вычисляется угол между этими векторами.

1.4.4. Аппроксимация и интерполяция данных

Система MatLAB предоставляет удобные процедуры для аппроксимации и интерполяции данных измерений.

Полиномиальная аппроксимация данных измерений, которые сформированы как некоторый вектор Y, при некоторых значениях аргумента, которые образуют вектор X такой же длины, что и вектор Y, осуществляется процедурой **polyfit(X, Y, n)**. Здесь n — порядок аппроксимирующего полинома. Результатом действия этой процедуры является вектор длиной (n+1) коэффициентов аппроксимирующего полинома.

Пусть массив значений аргумента является таким:

```
x = [1 2 3 4 5 6 7 8]
```

а массив соответствующих значений измеренной величины — следующим:

```
y = [-1.1 0.2 0.5 0.8 0.7 0.6 0.4 0.1]
```

Тогда, применяя указанную функцию при разных значениях порядка аппроксимирующего полинома, получим:

```
» x = [1 2 3 4 5 6 7 8];
» y = [-1.1 0.2 0.5 0.8 0.7 0.6 0.4 0.1];
» polyfit(x,y,1)
ans =    0.1143   -0.2393
» polyfit(x,y,2)
ans =   -0.1024    1.0357   -1.7750
» polyfit(x,y,3)
ans =    0.0177   -0.3410    1.9461   -2.6500
» polyfit(x,y,4)
ans =   -0.0044    0.0961   -0.8146    3.0326   -3.3893.
```

Это означает, что заданную зависимость можно аппроксимировать прямой:

$$y(x) = 0.1143x - 0.2393$$

квадратной параболой:

$$y(x) = -0.1024x^2 + 1.0357x - 1.7750$$

кубической:

$$y(x) = 0.0177x^3 - 0.3410x^2 + 1.9461x - 2.6500$$

или параболой четвертой степени:

$$y(x) = -0.0044x^4 + 0.0961x^3 - 0.8146x^2 + 3.0326x - 3.3893$$

Построим в одном графическом поле графики заданной дискретной функции и графики всех полученных при аппроксимации полиномов:

```
» x = [1 2 3 4 5 6 7 8];
» y = [-1.1 0.2 0.5 0.8 0.7 0.6 0.4 0.1];
» P1=polyfit(x,y,1);
» P2=polyfit(x,y,2);
» P3=polyfit(x,y,3);
```

```
» P4=polyfit(x,y ,4);
» stem(x,y);
» hold
» x1 = 0.5 : 0.05 : 8.5;
» y1=polyval(P1,x1);
» y2=polyval(P2,x1);
» y3=polyval(P3,x1);
» y4=polyval(P4,x1);
» plot(x1,y1,x1,y2,x1,y3,x1,y4),
» grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16),
» title('Полиномиальная аппроксимация ');
» xlabel('Аргумент');
» ylabel('Функция')
```

Результат представлен на рис. 1.18.

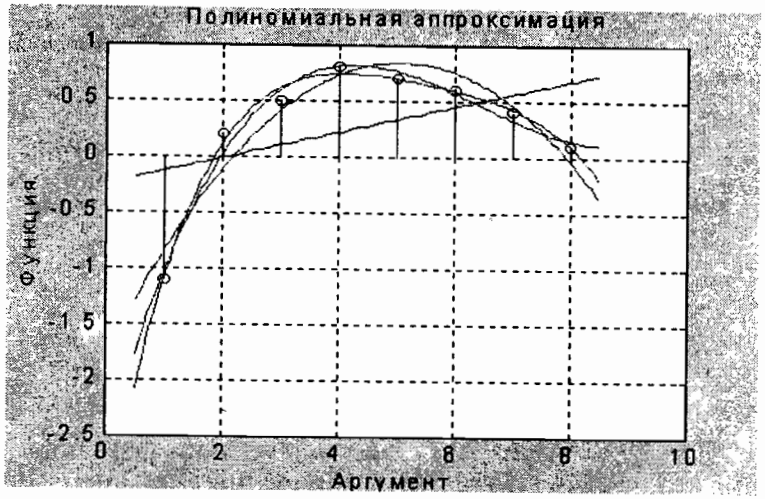

Рис. 1.18

Функция **spline(X,Y,Xi)** осуществляет *интерполяцию кубическими сплайнами*. При обращении:

```
Yi = spline(X,Y,Xi)
```

она интерполирует значения вектора Y, заданного при значениях аргумента, представленных в векторе X, и выдает значения интерполирующей функции в виде вектора Yi при значениях аргумента, заданных вектором Xi. В случае, когда вектор X не указан, по умолчанию принимается, что он имеет длину вектора Y и каждый его элемент равен номеру этого элемента.

Как пример рассмотрим интерполяцию вектора:

```
» x = -0.5 : 0.1 : 0.2;
» y = [-1.1 0.2 0.5 0.8 0.7 0.6 0.4 0.1];
» x1 = -0.5 : 0.01 : 0.2;
» y2 = spline(x,y,x1);
» plot (x,y,x1,y2), grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Интерполяция процедурой SPLINE ');
» xlabel('Аргумент');
» ylabel('Функция')
```

Результат приведен на рис. 1.19.

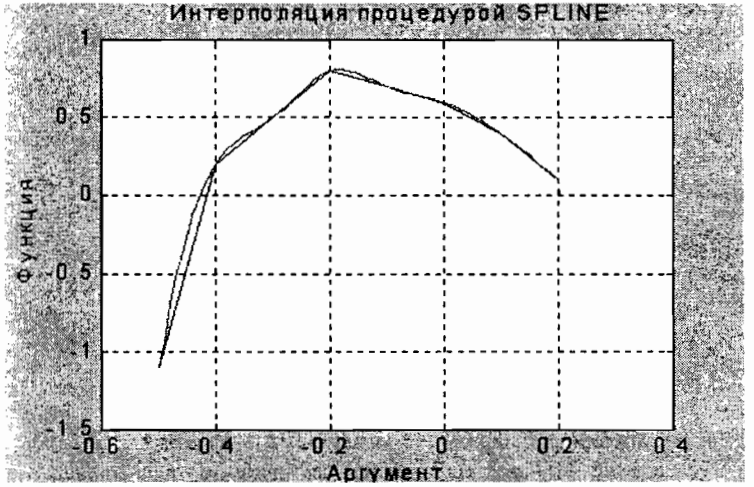

Рис. 1.19

Одномерную табличную интерполяцию производит процедура **interp1**. Обращение к ней в общем случае имеет вид:

```
Yi = interp1(X,Y,Xi,'<метод>')
```

и позволяет дополнительно указать метод интерполяции в четвертом входном аргументе: nearest — ступенчатая интерполяция, linear — линейная, cubic — кубическая, spline — кубическими сплайнами.

Если метод не указан, по умолчанию осуществляется линейная интерполяция. Например (для того же вектора):

```
» x = -0.5 : 0.1 : 0.2;
» y = [-1.1 0.2 0.5 0.8 0.7 0.6 0.4 0.1];
» x1 = -0.5 : 0.01 : 0.2;
» y1 = interp1(x,y,x1);
» y4 = interp1(x,y,x1,'nearest');
» y2 = interp1(x,y,x1,'cubic');
```

```
» y3 = interp1(x,y,x1,'spline');
» plot (x1,y1,x1,y2,x1,y3,x1,y4), grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Интерполяция процедурой INTERP1');
» xlabel('Аргумент');
» ylabel('Функция')
```

Результат приведен на рис. 1.20.

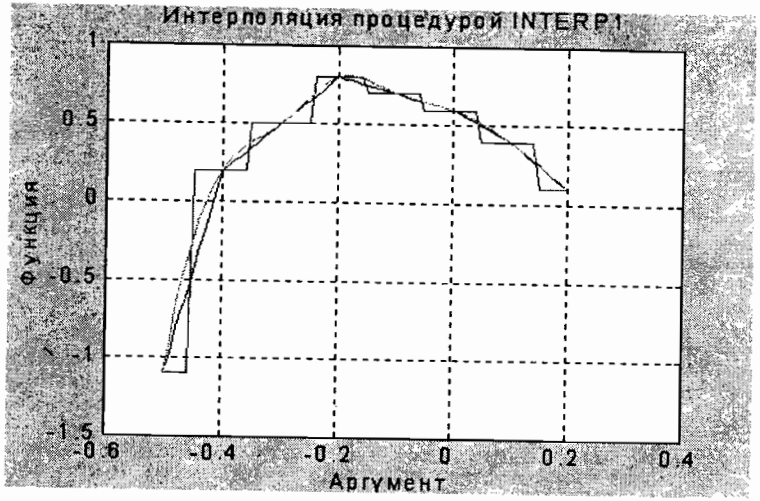

Рис. 1.20

1.4.5. Векторная фильтрация и спектральный анализ

В системе MatLAB есть несколько функций для проведения цифрового анализа данных наблюдений (измерений).

Так, функция **y = filter(b,a,x)** обеспечивает формирование вектора *y* по заданным векторам *b*, *a*, *x* в соответствии с соотношением:

$$y(k) = b(1)*x(k) + b(2)*x(k-1) + \ldots + b(nb+1)*x(k-nb) - a(2)*y(k-1) - a(3)*y(k-3) - \ldots - a(na+1)*y(k-na) \quad (1)$$

где вектор *b* имеет такой состав:

$$b = [b(1),b(2),\ldots,b(nb+1)]$$

а вектор *a*:

$$a = [1,a(2),a(3),\ldots,a(na+1)]$$

Соотношение (1) можно рассматривать как конечно-разностное уравнение фильтра с дискретной передаточной функцией вида рациональной дроби, коэффициенты числителя которой образуют

вектор *b*, а знаменателя — вектор *a*, на вход которого подается сигнал *x*(*t*), а на выходе формируется сигнал *y*(*t*).

Тогда вектор *y* будет представлять собой значения выходного сигнала этого фильтра в дискретные моменты времени, соответствующие заданным значениям входного сигнала *x*(*t*) (вектор *x*).

Ниже приведен пример применения функции **filter**.

```
» x = 0:0.1:1;
» b = [1 2];
» a = [ 1 0.1 4];
» y = filter(b,a,x)
y =  Columns 1 through 6
  0  1.0000e-001 3.9000e-001 2.6100e-001 -5.8610e-001 3.1461e-001
  Columns 7 through 11
  3.9129e+000  2.5027e-001 -1.3477e+001  2.8466e+000  5.6422e+001
```

Функции **fft** (Fast Fourier Transformation) и **ifft** (Invers Fast Fourier Transformation) осуществляют преобразования заданного вектора, соответствующие дискретному прямому и обратному преобразованиям Фурье.

Обращение к этим функциям:

```
y = fft ( x, n );  x = ifft ( y, n )
```

приводит к формированию вектора *y* в первом случае и *x* — во втором по формулам:

$$y(k) = \sum_{m=1}^{n} x(m) \cdot e^{-j \cdot 2\pi \cdot (m-1) \cdot (k-1)/n} \quad (2)$$

$$x(m) = \frac{1}{n} \sum_{k=1}^{n} y(k) \cdot e^{j \cdot 2\pi \cdot (m-1) \cdot (k-1)/n} \quad (3)$$

где *j* — обозначение мнимой единицы; *n* — число элементов заданного вектора *x* (оно является также размером выходного вектора *y*).

Приведем пример. Сформируем входной сигнал в виде вектора, элементы которого равны значениям функции, являющейся суммой двух синусоид с частотами 5 и 12 Гц. Найдем Фурье-изображение этого сигнала и выведем графические представления входного процесса и модуля его Фурье-изображения:

```
» t =0 : 0.001 : 2;
» x = sin(2*pi*5*t) + cos(2*pi*12*t);
» plot(t, x); grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Входной процесс ');
» xlabel('Время (c)');
» ylabel('X(t)')
» y = fft(x);
» a =abs(y);
» plot(a); grid
```

```
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Модуль Фурье - изображения');
» xlabel('Номер элемента вектора');
» ylabel('abs(F(X(t))')
```

Результаты отображены соответственно на рис. 1.21 и 1.22.

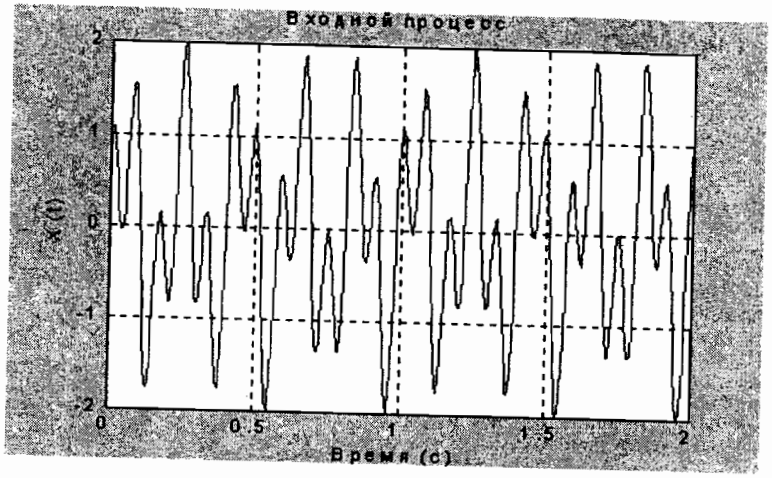

Рис. 1.21

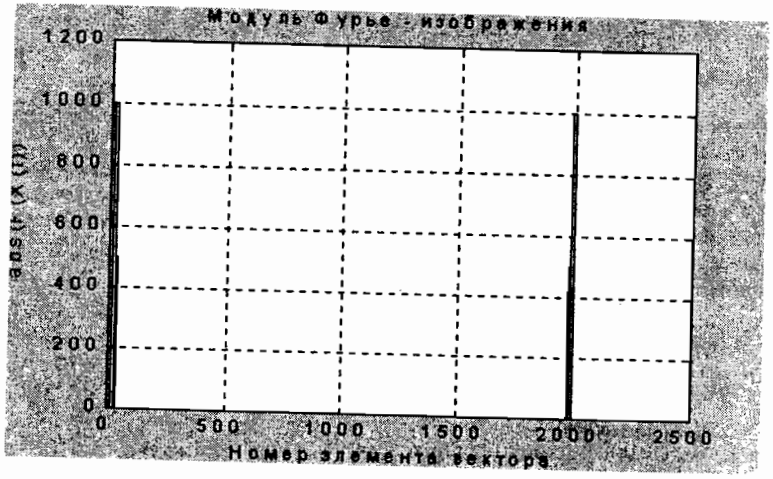

Рис. 1.22

Теперь осуществим обратное преобразование с помощью функции **ifft** и результат также выведем в форме графика:

```
» z = ifft(y);
» plot(t, z); grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Обратное Фурье-преобразование ');
» xlabel('Время (c)');
» ylabel('Z(t)')
```

На рис. 1.23 изображен результат. Рассматривая его, можно убедиться, что воспроизведенный процесс в точности совпадает с исходным.

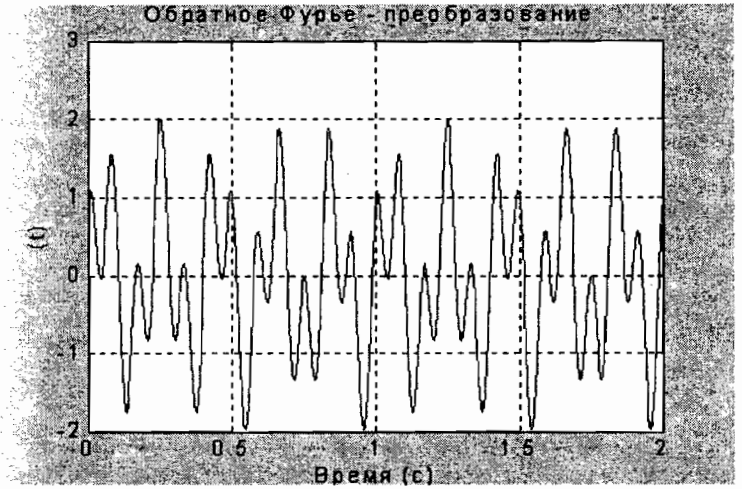

Рис. 1.23

Внимательно изучая формулу дискретного преобразования Фурье, можно заметить, что:

а) номер m соответствует моменту времени t_m, в который измерен входной сигнал $x(m)$; при этом $t_1 = 0$;

б) номер k — это индекс значения частоты f_k, которому соответствует найденный элемент $y(k)$ дискретного преобразования Фурье;

в) чтобы перейти от индексов к временной и частотной областям, надо знать значение h дискрета (шага) времени, через который измерен входной сигнал $x(t)$, и промежуток T времени, на протяжении которого он измерен; тогда шаг (дискрет) по частоте в изображении Фурье определится выражением:

$$Df = 1/T \qquad (4)$$

а диапазон изменения частоты — формулой:

$$F = 1/h \qquad (5)$$

Так, в рассматриваемом примере ($h = 0.001$, $T = 2$, $n = 21$):

$$Df = 0.5, \quad F = 1000$$

г) из (2) следует, что индексу $k = 1$ соответствует нулевое значение частоты ($f_0 = 0$); иначе говоря, первый элемент вектора $y(1)$ является значением Фурье-изображения при нулевой частоте, т.е. просто суммой всех заданных значений вектора x; отсюда следует, что вектор $y(k)$ содержит значения Фурье-изображения, начиная с частоты $f_0 = 0$ (которая соответствует $k = 1$) до максимальной частоты $f_{max} = F$ (соответствующей $k = n$); таким образом, Фурье-изображение определяется функцией **fft** только для положительных частот в диапазоне от 0 до F; это неудобно для построения графиков Фурье-изображения от частоты; более удобным и привычным является переход к вектору Фурье-изображения, определенному в диапазоне частот от $-F/2$ до $F/2$;

д) как известно, функция e^{jz} является периодической по z с периодом 2π; поэтому информация о Фурье-изображении при отрицательных частотах расположена во второй половине вектора $y(k)$.

Сформируем для рассматриваемого примера массив частот, исходя из вышеотмеченного:

```
» f = 0 : 0.5 : 1000;
```

и выведем график с аргументом-частотой (рис. 1.24).

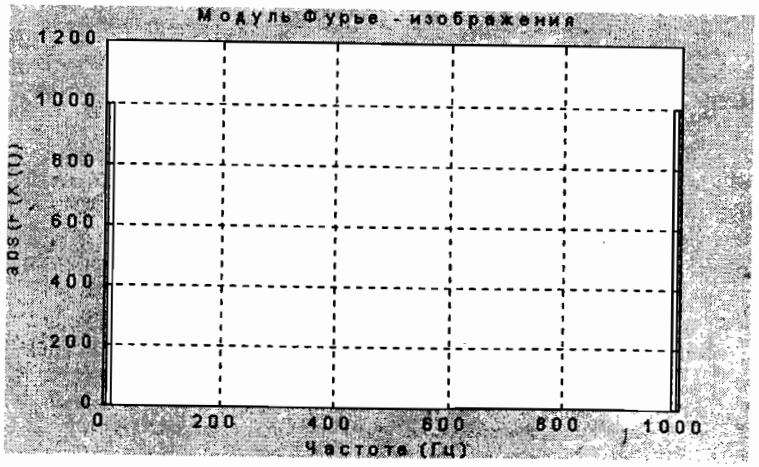

Рис. 1.24


```
» plot(f,a); grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Модуль Фурье - изображения ');
» xlabel('Частота (Гц)');
» ylabel('abs(F(X(t))')
```

На рис. 1.24 трудно различить частоты (5 и 12 Гц), с которыми колеблется входной сигнал. Это следствие того обстоятельства, которое было отмечено в пункте г. Чтобы установить истинный спектр входного сигнала, нужно вначале несколько преобразовать полученный вектор y Фурье-изображения с помощью процедуры **fftshift**.

Функция **fftshift** (обращение к ней осуществляется таким образом: **z = fftshift(y)**) предназначена для формирования нового вектора z из заданного вектора y путем перестановки второй половины вектора y в первую половину вектора z. При этом вторая половина вектора z состоит из элементов первой половины вектора y. Более точно эту операцию можно представить соотношениями:

$$z(1) = y(n/2+1);\ \ldots,\ z(k) = y(n/2+k);\ \ldots,\ z(n/2) = y(n);\ z(n/2+1) = y(1);\ \ldots,\ z(n/2+k) = y(k);\ \ldots\ z(n) = y(n/2)$$

Примечание

Операцию **fftshift** удобно использовать для определения массива Фурье-изображения с целью построения графика Фурье-изображения в частотной области. Однако этот массив не может быть использован для обратного преобразования Фурье.

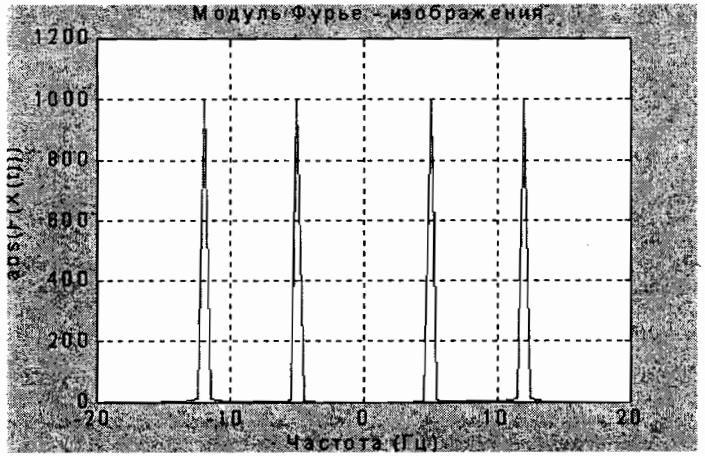

Рис. 1.25

Проиллюстрируем применение этой функции к предыдущему примеру:

```
» % Перестройка вектора частот
» f1 = -500 : 0.5 : 500;
» v = fftshift(y); % Перестройка вектора Фурье-изображения
» a =abs(v);       % Нахождение модуля
» % Построение графика
» plot(f1(970:1030),a(970:1030)); grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Модуль Фурье - изображения');
» xlabel('Частота (Гц)');
» ylabel('abs(F(X(t))')
```

Из графика (рис. 1.25) уже становится очевидным, что в спектре входного сигнала есть две гармоники — с частотами 5 и 12 Гц.

Неудобным является то, что по графику спектра невозможно определить амплитуды этих гармоник. Чтобы сделать это, нужно весь вектор *y* Фурье-изображения разделить на число его элементов (n):

```
» N=length(y);
» a =abs(v)/N;
» plot(f1(970:1030),a(970:1030)); grid
» set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Модуль Фурье - изображения / N');
» xlabel('Частота (Гц)');
» ylabel('abs(F(X(t)) / N')
```

Результат приведен на рис. 1.26. "Амплитуды" всех составляющих сигнал гармоник равны 0.5. При этом нужно принять во внимание, что "амплитуды" распределены между положительными и отрицательными частотами поровну, поэтому они вдвое меньше истинной амплитуды соответствующей гармоники.

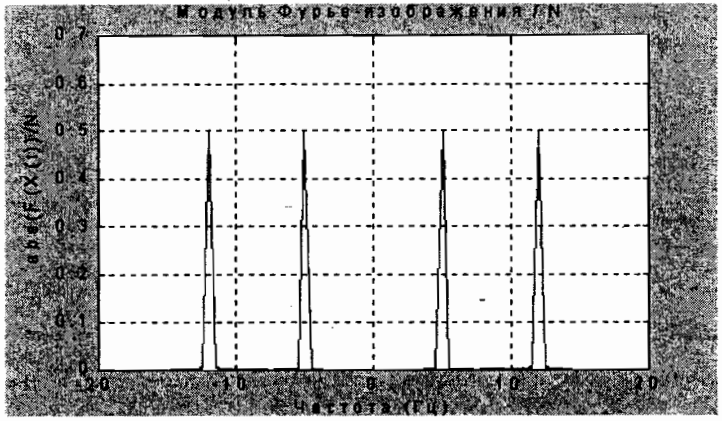

Рис. 1.26

1.5. Построение простейших графиков

1.5.1. Процедура plot

Вывод графиков в системе MatLAB является настолько простой и удобной процедурой, что ею можно пользоваться даже при вычислениях в режиме калькулятора.

Основной функцией, обеспечивающей построение графиков на экране дисплея, является функция **plot**. Общая форма обращения к этой процедуре такова:

```
plot(x1,y1,s1,x2,y2,s2,...)
```

Здесь x1, y1 — заданные векторы, элементами которых являются массивы значений аргумента (x1) и функции (y1), соответствующих первой кривой графика; x2, y2 — массивы значений аргумента и функции второй кривой и т.д. При этом предполагается, что значения аргумента откладываются вдоль горизонтальной оси графика, а значения функции — вдоль вертикальной оси. Переменные s1, s2,... являются символьными (их указание не обязательно). Каждая из них может содержать три специальных символа, которые определяют тип линии, соединяющей отдельные точки графика, тип точки графика и цвет линии. Если переменные s не указаны, то тип линии по умолчанию — отрезок прямой, тип точки — пиксель, а цвет устанавливается (в версии 5) в таком порядке: синий, зеленый, красный, голубой, фиолетовый, желтый, черный и белый, — в зависимости от того, какая по очереди линия отображается на графике. Например, обращение вида **plot(x1,y1,x2,y2,...)** приведет к построению графика, в котором первая кривая будет линией из отрезков прямых синего цвета, вторая кривая — такого же типа зеленой линией и т.д.

Графики в MatLAB всегда выводятся в отдельном (графическом) окне, которое называют *фигурой*.

Пусть требуется вывести график функции $y = 3\sin(x + \pi/3)$ на промежутке от -3π до $+3\pi$ с шагом $\pi/100$.

Сначала надо сформировать массив значений аргумента x:

```
x = -3*pi : pi/100 : 3*pi
```

затем вычислить массив соответствующих значений функции:

```
y = 3*sin(x+pi/3)
```

и, наконец, построить график зависимости y(x).

В целом, в командном окне эта последовательность операций будет выглядеть так:

```
» x = -3*pi:pi/100:3*pi;
» y = 3*sin(x+pi/3);
» plot(x,y)
```

В результате на экране появится дополнительное окно с графиком (рис. 1.27).

Если вектор аргумента при обращении к функции **plot** не указан явно, то система выбирает по умолчанию в качестве аргумента номер элемента вектора функции. Например, если ввести команду:

```
» plot(y)
```

то результатом будет появление графика в виде, приведенном на рис. 1.28.

Графики на рис. 1.27 и 1.28 имеют несколько недостатков:

- на них не нанесена сетка из координатных линий, что затрудняет "чтение" графиков;
- нет общей информации о кривых графика (заголовка);
- неизвестно, какие величины отложены по осям графика.

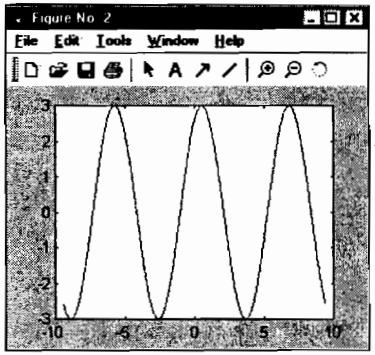

Рис. 1.27

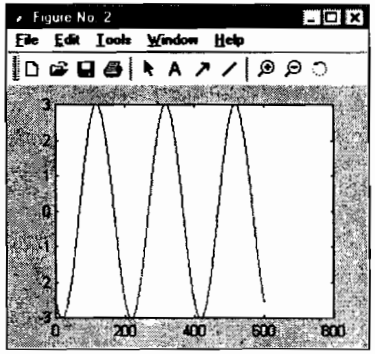

Рис. 1.28

Первый недостаток устраняется с помощью функции **grid**. Если эту функцию записать сразу после обращения к функции **plot**

```
» x = -3*pi:pi/100:3*pi;
» y = 3*sin(x+pi/3);
» plot(x,y),  grid
```

то график будет снабжен координатной сеткой (рис. 1.29).

Ценной особенностью графиков, построенных в системе MatLAB, является то, что *сетка координат всегда отвечает "целым" шагам изменения*, что делает графики "читабельными", т.е. по графику можно производить "отсчет" значения функции при любом заданном значении аргумента и наоборот. Таким свойством не обладает большинство графических пакетов-приложений к языкам программирования высокого уровня.

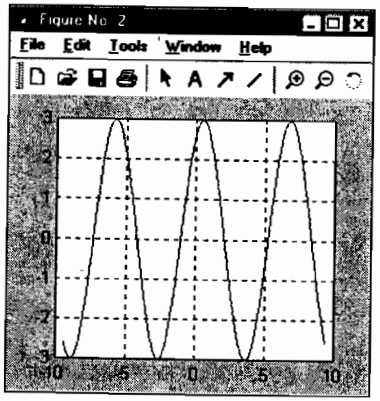

Рис. 1.29

Заголовок графика выводится с помощью процедуры **title**. Если после обращения к процедуре **plot** вызвать **title** таким образом:

```
title('<текст>')
```

то сверху в поле фигуры появится текст, записанный между скобками. При этом следует помнить, что *текст всегда должен помещаться в апострофы*.

Аналогично можно вывести пояснения к графику, которые размещаются вдоль горизонтальной оси (функция **xlabel**) и вдоль вертикальной оси (функция **ylabel**).

Например, совокупность операторов:

```
» x = -3*pi : pi/100 : 3*pi;
» y = 3*sin(x+pi/3);
» plot(x,y),  grid
» title('Функция    y = 3*sin(x+pi/3)');
» xlabel('x');   ylabel('y');
```

приведет к оформлению поля фигуры в виде, представленном на рис. 1.30. Очевидно, такая форма уже полностью удовлетворяет требованиям, которые предъявляются к инженерным графикам.

Не более сложным является вывод в среде MatLAB графиков функций, заданных *параметрически*. Пусть необходимо построить график функции y(x), которая задана параметрическими формулами:

$$x = 4\ e^{-0,05\ t} \sin t, \qquad y = 0,2\ e^{-0,1\ t} \sin 2t$$

Выберем диапазон изменения параметра t от 0 до 50 с шагом 0.1. Тогда, набирая совокупность операторов:

```
» t = 0 : 0.1 : 50;
» x = 4*exp(-0.05*t).*sin(t);
» y = 0.2*exp(-0.1*t).*sin(2*t);
```

```
» plot(x,y)
» title('Параметрическая функция  x=4*exp(-0.05t)*sin(t);...
       y= 0.2*exp(-0.1t)*sin(2t) ')
» grid
```

получим график, как на рис. 1.31.

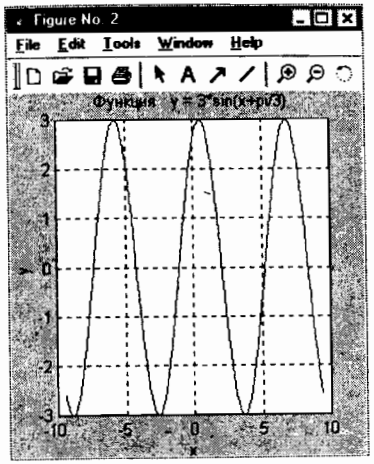

Рис. 1.30

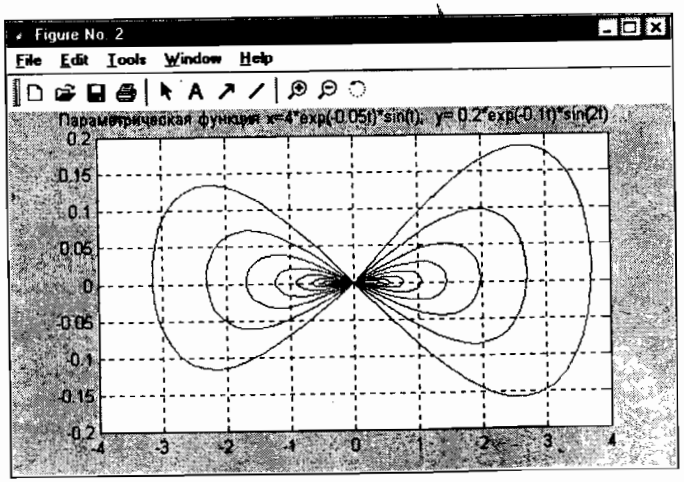

Рис. 1.31

1.5.2. Специальные графики

Большим удобством, предоставляемым системой MatLAB, является отмеченная ранее возможность не указывать аргумент функции при построении ее графика. В этом случае в качестве аргумента система принимает номер элемента вектора, график которого строится.

Пользуясь этим, можно построить, например, "график вектора":

```
» x = [ 1 3 2 9 6 8 4 6];
» plot (x)
» grid
» title('График вектора X')
» ylabel('Значение элементов')
» xlabel('Номер элемента')
```

Результат представлен на рис. 1.32.

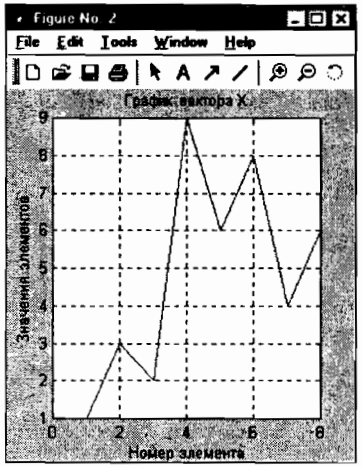

Рис. 1.32

Еще более наглядным является представление вектора в виде *столбчатой диаграммы* с помощью функции **bar** (рис. 1.33):

```
» bar(x)
» title('График вектора X')
» xlabel('Номер элемента')
» ylabel('Значение элементов')
```

Если функция задана своими значениями при дискретных значениях аргумента и неизвестно, как она может изменяться в промежутках между значениями аргумента, удобнее представлять график такой функции в виде отдельных вертикальных линий для каждого из заданных значений аргумента. Это можно сделать, применяя процедуру **stem**, обращение к которой аналогично обращению к процедуре **plot**.

```
» x = [ 1 3 2 9 6 8 4 6];
» stem(x,'k')
» grid
» set(gca,'FontName','Arial','FontSize',14),
» title('График вектора X')
» ylabel('Значение элементов')
» xlabel('Номер элемента')
```

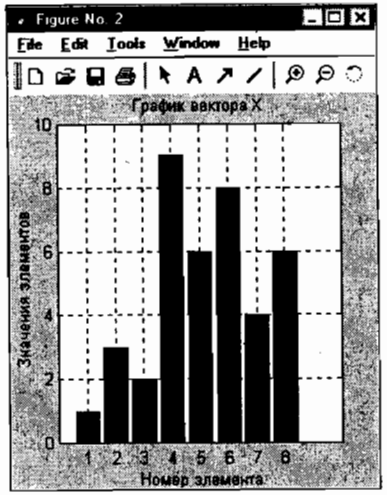

Рис 1.33

На рис. 1.34 изображен полученный при этом график.

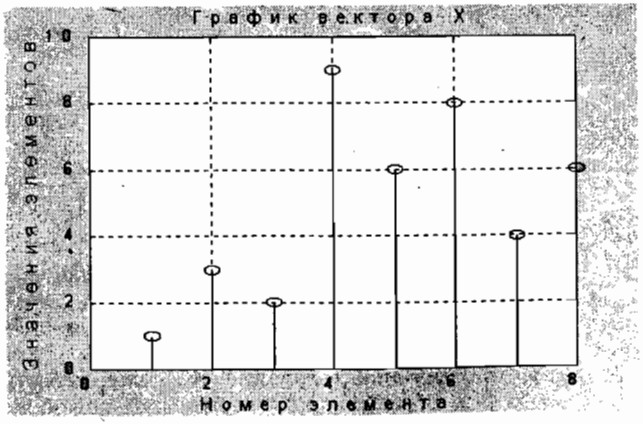

Рис. 1.34

Другой пример — построение графика функции $y = e^{-x*x}$ в виде столбчатой диаграммы (рис. 1.35):

```
» x = - 2.9 : 0.2 : 2.9;
» bar(x, exp(-x .* x))
» title('Стобчатая диаграмма функции y = exp(-x^2)')
» xlabel ('Аргумент x')
» ylabel ('Значение функции y')
```

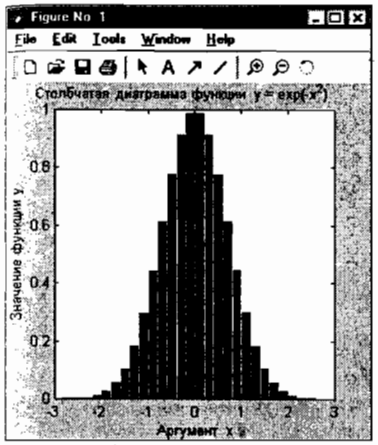

Рис. 1.35

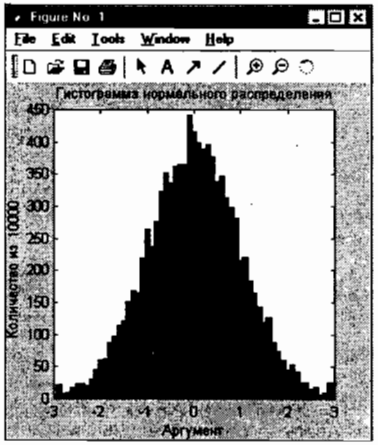

Рис. 1.36

Еще одна полезная инженеру функция — **hist** (построение графика *гистограммы* заданного вектора). Стандартное обращение к ней имеет вид:

```
hist(y,x)
```

где y — вектор, гистограмму которого нужно построить; x — вектор, определяющий интервалы изменения первого вектора, внутри которых подсчитывается число элементов вектора y. Эта функция производит две операции:

- подсчитывает число элементов вектора y, значения которых попадают внутрь диапазона, указанного вектором x;
- строит столбчатую диаграмму подсчитанных чисел элементов вектора y как функцию указанных вектором x диапазонов.

В качестве примера рассмотрим построение гистограммы случайных величин, которые формируются встроенной функцией **randn**. Возьмем общее число элементов вектора этих случайных величин 10000. Построим гистограмму для диапазона изменения этих величин от -2,9 до +2,9. Интервалы изменения пусть будут равны

0,1. Тогда график гистограммы можно построить при помощи совокупности таких операторов:

```
» x = -2.9 : 0.1 : 2.9;
» y = randn(10000,1);
» hist(y,x)
» ylabel('Число с 10000')
» xlabel('Аргумент')
» title('Гистограмма нормального распределения')
```

Результат представлен на рис. 1.36. Из него, в частности, следует, что встроенная функция **randn** достаточно верно отображает нормальный гауссовый закон распределения случайной величины.

Процедура **comet(x,y)** ("комета") строит график зависимости y(x) постепенно во времени в виде траектории кометы. При этом изображающая точка на графике имеет вид маленькой кометы (с головкой и хвостиком), которая плавно перемещается от одной точки к другой. Например, если ввести совокупность операторов:

```
» t = 0 : 0.1 : 50;
» x = 4 * exp(-0.05*t) .* sin(t);
» y = 0.2 * exp(-0.1*t) .* sin(2*t);
» comet(x,y)
```

то график, приведенный на рис. 1.31, будет построен как траектория последовательного движения кометы. Это обстоятельство может быть полезным при построении пространственных траекторий для выявления характера изменения траектории со временем.

MatLAB имеет несколько функций, позволяющих строить графики в логарифмическом масштабе.

Функция **logspace** с обращением:

```
x = logspace(d1, d2, n)
```

формирует вектор-строку x, содержащую n равноотдаленных в логарифмическом масштабе друг от друга точек, которые покрывают диапазон от 10^{d1} до 10^{d2}.

Функция **loglog** полностью аналогична функции **plot**, но графики по обеим осям строятся в логарифмическом масштабе. Для построения графиков, использующих логарифмический масштаб только по одной из координатных осей, пользуются процедурами **semilogx** и **semilogy**. Первая процедура строит графики с логарифмическим масштабом вдоль горизонтальной оси, вторая — вдоль вертикальной оси. Обращение к последним трем процедурам полностью аналогично обращению к функции **plot**.

В качестве примера рассмотрим построение графиков амплитудно-частотной и фазо-частотной характеристик звена, описываемого передаточной функцией:

$$W(p) = \frac{p+4}{p^2+4\cdot p+100}$$

Для этого следует, во-первых, создать векторы-полиномы числителя Pc = [1 4] и знаменателя Pz = [1 4 100] передаточной функции.

Во-вторых, определить корни этих двух полиномов:

```
» P1 = [1 4]; P2 = [1 4 100];
» roots(P1)
ans =    -4
» roots(P2)
ans =
-2.0000e+000 +9.7980e+000i
-2.0000e+000 -9.7980e+000i
```

В-третьих, задать диапазон изменения частоты таковым, чтобы он охватывал все найденные корни:

```
» om0 = 1e-2;  omk = 1e2
```

Теперь нужно задать количество точек будущего графика:

```
» n = 41
```

и сформировать массив точек по частоте OM = logspace(-2,2,41), где значения -2 и +2 соответствуют десятичным порядкам начального (om0) и конечного (omk) значений частоты.

Пользуясь функцией **polyval**, можно вычислить сначала вектор *ch* комплексных значений числителя частотной передаточной функции, соответствующей заданной передаточной функции, если в качестве аргумента функции **polyval** использовать сформированный вектор частот OM, элементы которого умножены на мнимую единицу (см. определение частотной передаточной функции). Аналогично вычисляется комплекснозначный вектор *zn* знаменателя ЧПФ.

Вектор значений АЧХ (амплитудно-частотной характеристики) можно найти, вычисляя модули векторов числителя и знаменателя ЧПФ и деля поэлементно полученные векторы. Чтобы найти вектор значений ФЧХ (фазо-частотной характеристики) надо разделить поэлементно комплекснозначные векторы числителя и знаменателя ЧПФ и определить вектор аргументов элементов полученного вектора. Чтобы фазы представить в градусах, полученные результаты следует умножить на 180 и разделить на π.

Наконец, для построения графика АЧХ в логарифмическом масштабе, достаточно применить функцию **loglog**, а для построения ФЧХ удобнее воспользоваться функцией **semilogx**.

В целом последовательность действий может быть такой:

```
» OM = logspace(-2,2,40)
» ch = polyval(P1,i*OM);
» zn = polyval(P2,i*OM);
» ACH = abs(ch)./abs(zn);
» loglog(OM,ACH);
» grid;
» title('График амплитудно-частотной характеристики')
» xlabel('Частота (рад/с)');
» ylabel('Отношение амплитуд')
```

```
» FCH = angle(ch./zn)*180/pi;
» semilogx(OM,FCH);
» grid,
» title('Фазо-частотная характеристика'),
» xlabel('Частота (рад/с)'),
» ylabel('Фаза (градусы)')
```

В результате получаются графики, как на рис. 1.37 и 1.38.

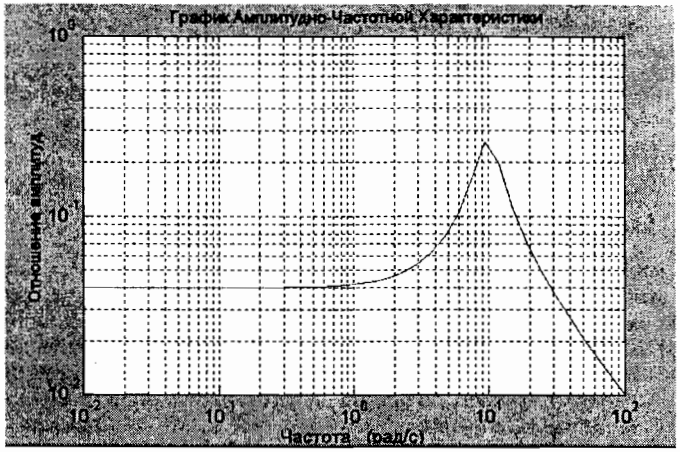

Рис. 1.37

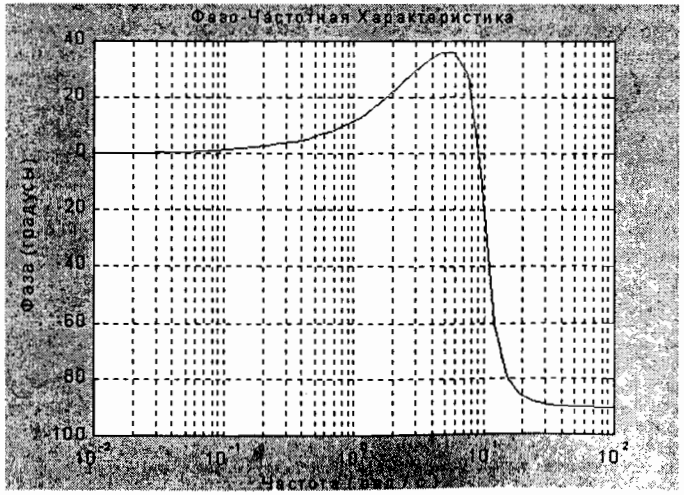

Рис. 1.38

1.5.3. Дополнительные функции графического окна

Обычно графики, получаемые с помощью процедур **plot**, **loglog**, **semilogx** и **semilogy**, автоматически строятся в таких масштабах, чтобы в поле графика поместились все вычисленные точки, включая максимальные и минимальные значения аргумента и функции. Но MatLAB имеет возможности определения и других режимов масштабирования. Это достигается за счет использования процедуры **axis**.

Команда **axis([xmin xmax ymin ymax])** устанавливает жесткие границы поля графика в единицах величин, откладываемых по осям.

Команда **axis('auto')** возвращает масштабы по осям к их штатным значениям (принятым по умолчанию).

Команда **axis('ij')** перемещает начало отсчета в левый верхний угол и реализует отсчет от верхнего левого угла (матричная система координат).

Команда **axis('xy')** возвращает декартову систему координат с началом отсчета в левом нижнем углу.

Команда **axis('square')** устанавливает одинаковый диапазон изменения переменных по осям графика.

Команда **axis('equal')** обеспечивает одинаковый масштаб по обеим осям графика.

В одном графическом окне, но на отдельных графических полях можно построить несколько графиков, используя процедуру **subplot**. Обращение к этой процедуре должно предшествовать обращению к процедурам **plot**, **loglog**, **semilogx** и **semilogy** и выглядеть так:

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

Здесь m указывает, на сколько частей делится графическое окно по вертикали, n — по горизонтали, а p является номером подокна, в котором будет строиться график. Подокна нумеруются слева направо, сверху вниз (так, как читается текст книги).

Например, два предыдущих графика можно поместить в одном графическом окне следующим образом:

```
» subplot(2,1,1);
» loglog(OM,ACH,'k'); grid;
» set(gca,'FontName','Arial','FontSize',14),
» title('Амплитудно-частотная характеристика')
» ylabel('Амплитуда')
» subplot(2,1,2);
» semilogx(OM,FCH,'k'); grid
» set(gca,'FontName','Arial','FontSize',14),
» title('Фазо-частотная характеристика')
» xlabel('Частота (рад/с)'), ylabel('Фаза (гр.)')
```

Результат представлен на рис. 1.39.

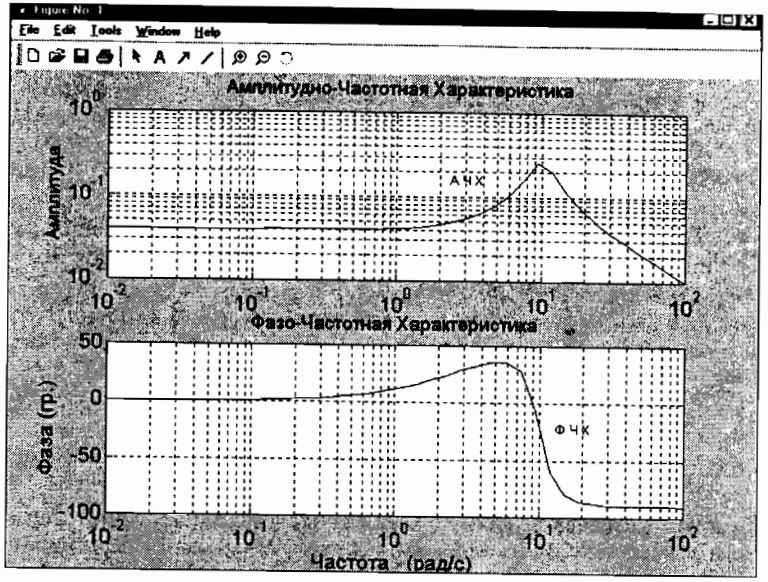

Рис. 1.39

Команда **text(x, y, '<текст>')** позволяет разместить указанный текст в поле графика, при этом начало текста помещается в точку с координатами x и y. Значения указанных координат должны быть представлены в единицах величин, откладываемых по осям графика, и находиться внутри диапазона изменений этих величин, что неудобно.

Более удобным для размещения текста внутри поля графика является использование команды **gtext('<текст>')**, с помощью которой в активном графическом окне высвечивается перекрестие. Перемещая перекрестие с помощью мыши, можно указать место начала вывода указанного текста. Нажатием левой кнопки мыши или любой клавиши текст вводится в указанное место:

```
» gtext('Ф Ч Х')
» subplot(2,1,1);
» gtext('А Ч Х')
```

Именно таким образом установлены соответствующие записи в поле графиков на рис. 1.39.

Чтобы создать несколько графических окон, в каждом из которых размещаются соответствующие графики, можно воспользоваться командой **figure**, которая создаст новое графическое окно, оставляя предыдущие.

Наконец, для того чтобы несколько последовательно вычисляемых графиков были отображены в одном графическом окне, можно использовать команду:

```
» hold on
```

Тогда каждый следующий график будет строиться в том же предварительно открытом графическом окне, т.е. новая линия будет добавляться к ранее построенным. Команда **hold off** выключает режим сохранения графического окна, установленный предыдущей командой.

1.5.4. Вывод графиков на печать

Чтобы вывести график из графического окна (фигуры) на печать, следует воспользоваться командами меню, расположенного в верхней части окна фигуры.

Выберите из меню **File** команду **Print**. Подготовьте принтер к печати, нажмите кнопку **OK** в диалоговом окне, и принтер распечатает содержимое графического окна на отдельном листе бумаги. Для настройки принтера используйте команду **Print Setup** из меню **File**.

1.6. Операторы управления вычислительным процессом

Вообще, операторы управления необходимы главным образом для организации вычислительного процесса, который записывается в виде некоторого текста программы на языке программирования высокого уровня. При этом к операторам управления вычислительным процессом обычно относят операторы безусловного перехода, условных переходов (разветвления вычислительного процесса) и операторы организации циклических процессов. Однако система MatLAB построена таким образом, что эти операторы могут быть использованы и при работе MatLAB в режиме калькулятора.

В языке MatLAB отсутствует оператор безусловного перехода, и поэтому нет понятия метки. Это является недостатком языка MatLAB и затрудняет организацию возвращения вычислительного процесса к любому предыдущему или последующему оператору программы.

Все операторы цикла и условного перехода построены в MatLAB в виде сложного оператора, который начинается служебным словом **if**, **while**, **switch** или **for** и заканчивается служебным словом **end**. Операторы между этими словами воспринимаются системой как части одного сложного оператора. Поэтому нажатие клавиши [Enter] для перехода к следующей строке не приводит в данном случае к выполнению этих операторов. Выполнение операторов начинается только тогда, когда введена "закрывающая скобка"

сложного оператора в виде слова **end**, а затем нажата клавиша [Enter]. Если несколько сложных операторов такого типа вложены один в другой, вычисления начинаются лишь тогда, когда записан конец (**end**) наиболее охватывающего (внешнего) сложного оператора. Из этого вытекает возможность осуществления даже в режиме калькулятора довольно сложных и объемных (состоящих из многих строк и операторов) вычислений, если они охвачены сложным оператором.

1.6.1. Оператор условного перехода

Конструкция оператора перехода по условию в общем виде такова:

```
if <условие>
 <операторы1>
else
 <операторы2>
end
```

Работает он следующим образом. Вначале проверяется, выполняется ли указанное условие. Если да, то программа выполняет совокупность операторов, которая записана в разделе <операторы1>. В противном случае выполняется последовательность операторов раздела <операторы2>.

Укороченная форма условного оператора имеет вид:

```
if <условие>
 <операторы>
end
```

Действие оператора в этом случае аналогично, за исключением того, что при невыполнении заданного условия выполняется оператор, следующий за оператором **end**.

Легко заметить недостатки этого оператора, вытекающие из отсутствия оператора безусловного перехода: вся часть программы, выполняющаяся в зависимости от условия, должна размещаться внутри операторных скобок **if** и **end**.

В качестве условия используется выражение типа:

```
<имя переменной1> <операция сравнения> <имя переменной2>
```

Операции сравнения в языке MatLAB могут быть такими:

<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
==	равно
~=	не равно

Условие может быть составным, т.е. складываться из нескольких простых условий, объединяемых знаками логических операций. Знаками логических операций в языке MatLAB являются:

&	логическая операция И (AND)
|	логическая операция ИЛИ (OR)
~	логическая операция НЕТ (NOT)

Логическая операция Исключающее ИЛИ может быть реализована при помощи функции **xor(A,B)**, где A и B — некоторые условия.

Допустима еще одна конструкция оператора условного перехода:

```
if <условие1>
 <операторы1>
   elseif <условие2>
       <операторы2>
   elseif <условие3>
       <операторы3>
   . . .
   else
 <операторы>
end
```

Оператор **elseif** выполняется тогда, когда <условие1> не выполнено. При этом сначала проверяется <условие2>. Если оно выполнено, выполняются <операторы2>, если же нет, то <операторы2> игнорируются и происходит переход к следующему оператору **elseif**, т.е. к проверке <условия3>. Аналогичным образом при его выполнении обрабатываются <операторы3>, в противном случае происходит переход к следующему оператору **elseif**. Если ни одно из условий в операторах **elseif** не выполнено, обрабатываются <операторы>, следующие за оператором **else**. Таким образом может быть обеспечено ветвление программы по нескольким направлениям.

1.6.2. Оператор переключения

Оператор переключения имеет такую структуру:

```
switch <выражение, скаляр или строка символов>
  case <значение1>
     <операторы1>
  case <значение2>
     <операторы2>
  . . .
  otherwise
     <операторы>
end
```

Он осуществляет ветвление вычислений в зависимости от значений некоторой переменной или выражения, сравнивая значение, полученное в результате вычисления выражения в строке **switch**, со значениями, указанными в строках со словом **case**. Соответствующая группа операторов **case** выполняется, если значение выражения совпадает со значением, указанным в соответствующей строке **case**. Если значение выражения не совпадает ни с одним из значений в группах **case**, выполняются операторы, следующие за **otherwise**.

1.6.3. Операторы цикла

В языке MatLAB есть две разновидности операторов цикла — *условный* и *арифметический*.

Оператор цикла с предусловием имеет вид:

```
while <условие>
      <операторы>
end
```

Операторы внутри цикла обрабатываются лишь в том случае, если выполнено условие, записанное после слова **while**. При этом среди операторов внутри цикла обязательно должны быть такие, которые изменяют значения одной из переменных, указанных в условии цикла.

Приведем пример вычисления значения синуса при 21 значении аргумента от 0.2 до 4 с шагом 0.2:

```
» i = 1;
» while i <= 20
   x = i/5;
   si = sin(x);
   disp([x,si])
   i = i+1;
end
    0.2000    0.1987
    0.4000    0.3894
    0.6000    0.5646
    0.8000    0.7174
    1.0000    0.8415
    1.2000    0.9320
    1.4000    0.9854
    1.6000    0.9996
    1.8000    0.9738
    2.0000    0.9093
    2.2000    0.8085
    2.4000    0.6755
    2.6000    0.5155
    2.8000    0.3350
    3.0000    0.1411
    3.2000   -0.0584
    3.4000   -0.2555
    3.6000   -0.4425
    3.8000   -0.6119
    4.0000   -0.7568
```

Примечание

Обратите внимание на то, какими средствами в приведенном примере обеспечен вывод на экран нескольких переменных в одну строку.

Для этого используется ранее применявшийся оператор **disp**. Но, в соответствии с правилами применения этого оператора, в нем должен быть только один аргумент (текст, переменная или матрица). Чтобы обойти это препятствие, нужно несколько числовых переменных объединить в единый объект — вектор-строку, а последнее легко выполняется при помощи обычной операции формирования вектора-строки из отдельных элементов

```
[x1, x2, ..., xN]
```

Таким образом, с помощью оператора вида:

```
disp([x1, x2, ..., xN])
```

можно обеспечить вывод результатов вычислений в виде таблицы данных.

Арифметический оператор цикла имеет вид:

```
for <имя> = <НЗ> : <Ш> : <КЗ>
   <операторы>
end
```

где <имя> — имя управляющей переменной цикла — счетчика цикла; <НЗ> — заданное начальное значение этой переменной; <Ш> — значение шага, с которым она должна изменяться; <КЗ> — конечное значение переменной цикла. В этом случае <операторы> внутри цикла выполняются несколько раз (каждый раз при новом значении управляющей переменной) до тех пор, пока значение управляющей переменной не выйдет за пределы интервала между <НЗ> и <КЗ>. Если параметр <Ш> не указан, по умолчанию его значение принимается равным единице.

Чтобы досрочно выйти из цикла (например, при выполнении некоторого условия), применяют оператор **break**. Если в программе встречается этот оператор, выполнение цикла досрочно прекращается и начинает выполняться следующий после слова **end** оператор.

Для примера используем предыдущее задание:

```
» a = ['    i    ','    x    ','  sin(x)   '];
» for i = 1:20
      x = i/5;
      si = sin(x);
      if i==1
         disp(a)
      end
      disp([i,x,si])
end
      i         x       sin(x)
    1.0000    0.2000    0.1987
    2.0000    0.4000    0.3894
    3.0000    0.6000    0.5646
    4.0000    0.8000    0.7174
    5.0000    1.0000    0,8415
    6.0000    1.2000    0.9320
    7.0000    1.4000    0.9854
    8.0000    1.6000    0.9996
    9.0000    1.8000    0.9738
   10.0000    2.0000    0.9093
   11.0000    2.2000    0.8085
   12.0000    2.4000    0.6755
   13.0000    2.6000    0.5155
   14.0000    2.8000    0.3350
   15.0000    3.0000    0.1411
   16.0000    3.2000   -0.0584
   17.0000    3.4000   -0.2555
   18.0000    3.6000   -0.4425
   19.0000    3.8000   -0.6119
   20.0000    4.0000   -0.7568
```


Глава 2

Программирование в среде MatLAB

Работа в режиме калькулятора в среде MatLAB, несмотря на довольно значительные возможности, имеет существенные недостатки. Невозможно повторить все предыдущие вычисления и действия при новых значениях исходных данных без повторного набора всех предыдущих операторов. Нельзя вернуться назад и повторить некоторые действия или по некоторому условию перейти к выполнению другой последовательности операторов. И вообще, если число операторов велико, становится проблемой отладить правильную их работу из-за неизбежных ошибок при наборе команд.

Поэтому сложные, с прерываниями, запутанными переходами по определенным условиям, с часто повторяемыми однотипными действиями вычисления, которые, к тому же, необходимо проводить неоднократно при измененных входных данных, требуют их специального оформления в виде записанных на диске файлов, т.е. в виде программ. Преимущество программ в том, что становится возможным неоднократное обращение к одним и тем же операторам и к программе в целом. Создание программ позволяет значительно упростить и сократить процесс подготовки повторяемых вычислений, сделать процесс вычислений более наглядным и прозрачным, а благодаря этому — резко уменьшить вероятность появления принципиальных ошибок при разработке программ. Кроме того, в программах появляется возможность автоматизировать и процесс изменения значений входных параметров в диалоговом режиме.

2.1. Функции функций

Некоторые важные универсальные процедуры в MatLAB используют в качестве переменного параметра имя функции, с которой они оперируют, и потому требуют при обращении к ним указания имени М-файла, в котором записан текст программы вычисления некоторой другой процедуры (функции). Такие процедуры называют *функциями функций*. Чтобы воспользоваться такой функцией от функции, необходимо, чтобы пользователь предварительно создал М-файл, в котором вычислялось бы значение нужной функции по заданному значению ее аргумента. Перечислим некоторые из стандартных функций от функций, предусмотренных в MatLAB.

Вычисление интеграла методом квадратур осуществляется процедурой:

```
[I, cnt] = quad('<имя функции>', a,b)
```

Здесь a и b — нижняя и верхняя границы изменения аргумента функции; I — полученное значение интеграла; cnt — число обращений к вычислению функции, представленной М-файлом с названием, указанным в <имя функции>. Функция **quad** использует квадратурные формулы Ньютона-Котеса четвертого порядка.

Аналогичная процедура **quad8** использует более точные формулы 8-го порядка.

Интегрирование обыкновенных дифференциальных уравнений осуществляют функции **ode23** и **ode45**. Они могут применяться как для решения простых дифференциальных уравнений, так и для моделирования сложных динамических систем.

Известно, что любая система обыкновенных дифференциальных уравнений (ОДУ) может быть представлена в так называемой форме Коши:

$$\frac{dy}{dt} = f(y,t)$$

где y — вектор переменных состояния системы; t — аргумент (обычно время); f — нелинейная вектор-функция от переменных состояния y и аргумента t.

Обращение к процедурам численного интегрирования ОДУ имеет вид:

```
[t, y] = ode23('<имя функции>', tspan, y0, options)
[t, y] = ode45('<имя функции>', tspan, y0, options)
```

где <имя функции> — строка символов, являющаяся именем М-файла, в котором вычисляется вектор-функция $f(y,t)$, т.е. *правые части системы ОДУ*; y0 — вектор начальных значений переменных состояния; t — массив значений аргумента, соответствующих шагам интегрирования; y — матрица проинтегрированных значений фазовых переменных, в которой каждый столбец соответствует одной из переменных состояния, а строка содержит значения переменных состояния, соответствующих определенному шагу интегрирования; tspan — вектор-строка [t0 tfinal], содержащая два значения: t0 — начальное значение аргумента t; tfinal — конечное значение аргумента; options — строка параметров, определяющих значения допустимой относительной и абсолютной погрешности интегрирования.

Параметр options можно не указывать. Тогда по умолчанию допустимая относительная погрешность интегрирования принимается равной 1.0e-3, а абсолютная (по каждой из переменных состояния) — 1.0e-6. Если же эти значения не устраивают пользователя, следует перед обращением к процедуре численного интегрирования установить новые значения допустимых погрешностей с помощью процедуры **odeset** таким образом:

```
options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-5])
```

Параметр RelTol определяет относительную погрешность численного интегрирования по всем переменным одновременно, а AbsTol является вектором-строкой, состоящим из абсолютных допустимых погрешностей численного интегрирования по каждой из фазовых переменных.

Функция **ode23** осуществляет интегрирование численным методом Рунге-Кутта 2-го порядка, а с помощью метода 3-го порядка контролирует относительные и абсолютные ошибки интегрирования на каждом шаге и изменяет величину шага интегрирования так, чтобы обеспечить заданные пределы ошибок интегрирования. Для функции **ode45** основным методом интегрирования является метод Рунге-Кутта 4-го порядка, а величина шага контролируется методом 5-го порядка.

Вычисления минимумов и корней функции осуществляется следующими функциями MatLAB:

fmin	нахождение минимума функции одного аргумента
fmins	нахождение минимума функции нескольких аргументов
fzero	нахождение нулей (корней) функции одного аргумента

Обращение к первой из них в общем случае имеет вид:

```
Xmin = fmin('<имя функции>', X1, X2)
```

Результатом этого обращения будет значение Xmin аргумента функции, соответствующее локальному минимуму в интервале X1<X<X2 функции, заданной М-файлом с указанным именем. В качестве примера рассмотрим нахождение числа π как значения локального минимума функции y = cos(x) на отрезке [3, 4]:

```
» Xmin = fmin('cos',3,4)
Xmin = 3.1416e+000
```

Обращение ко второй процедуре должно иметь форму:

```
Xmin = fmins('<имя функции>', X0)
```

При этом X является вектором аргументов, а X0 означает начальное (исходное) значение этого вектора, в окрестности которого отыскивается ближайший локальный минимум функции, заданной М-файлом с указанным именем. Функция **fmins** находит вектор аргументов Xmin, соответствующий найденному локальному минимуму.

Обращение к функции **fzero** должно иметь вид:

```
z = fzero('<имя функции>', x0, tol, trace)
```

Здесь x0 — начальное значение аргумента, в окрестности которого отыскивается действительный корень функции со значениями, вычисляемыми в М-файле с заданным именем; tol — заданная относительная погрешность вычисления корня; trace — обозначение необходимости выводить на экран промежуточные результаты; z — значение искомого корня.

Построение графиков функции одной переменной может быть осуществлено при помощи процедуры **fplot**. Отличие ее от процедуры **plot** состоит в том, что для построения графика функции нет необходимости в предварительном вычислении ее значения и аргумента. Обращение к ней имеет вид:

```
fplot('<имя функции>', [<интервал>], n)
```

где <интервал> — это вектор-строка из двух чисел, задающих, соответственно, нижнюю и верхнюю границы изменения аргумента; <имя функции> — имя М-файла с текстом процедуры вычисления значения желаемой функции по заданному значению ее аргумента; n — количество частей, на которые будет разбит указанный интервал. Если последнюю величину не указать, по умолчанию интервал разобьется на 25 частей. Несмотря на то, что количество частей n указано, число значений вектора x может быть значительно большим за счет того, что функция **fplot** проводит вычисления с дополнительным ограничением, чтобы приращение угла наклона графика функции на каждом шаге не превышало 10 градусов. Если же оно оказалось большим, осуществляется дробление шага изменения аргумента, но не более чем в 20 раз. Последние два числа (10 и 20) могут быть изменены пользователем, для этого при обращении следует добавить эти новые значения в заголовок процедуры в указанном порядке.

Если обратиться к этой процедуре так:

```
[x, Y] = fplot('<имя функции>', [<интервал>], n)
```

то график указанной в М-файле функции не отображается на экране (в графическом окне). Вместо этого вычисляется вектор x аргументов и вектор (или матрица) Y соответствующих значений указанной функции. Чтобы при обращении последнего вида построить график, необходимо сделать это в дальнейшем с помощью процедуры **plot(x, Y)**.

2.2. Создание М-файлов в среде MatLAB

2.2.1. Особенности создания М-файлов в MatLAB

Создание программы в среде MatLAB осуществляется при помощи либо собственного встроенного (начиная с версии MatLAB 5), либо стороннего текстового редактора, который вызывается автоматически, если он предварительно установлен с помощью команды **Preferences** меню **File**. Это может быть, например, редактор Notepade среды Windows. Окно предварительно установленного редактора появляется на экране, если перед этим вызвана команда **M-file** из подменю **New** или выбрано название одного из существующих М-файлов при вызове команды **Open M-file** из меню **File**. В первом случае окно текстового редактора будет пустым, во втором — в нем будет содержаться текст вызванного М-файла. В обоих случаях окно текстового редактора готово для ввода нового текста либо корректировки существующего.

В языке MatLAB имеются программы двух типов: так называемые *Script-файлы* (*файл-сценарии*, или *управляющие программы*) и *файл-функции* (*процедуры*). Все программы должны иметь расширение имен файлов .m, т.е. их нельзя различить по типу файла. При помощи Script-файлов оформляются основные программы, управляющие от начала до конца организацией всего вычислительного процесса, и отдельные части основных программ (они могут быть записаны в виде отдельных Script-файлов). Как файл-функции оформляются отдельные процедуры и функции (т.е. такие части программы, которые рассчитаны на неоднократное использование Script-файлами или другими процедурами при изменяемых значениях входных параметров и не могут быть выполнены без предварительного задания значений переменных, которые называют *входными*).

Главным внешним отличием текстов этих двух видов файлов является то, что *файл-функции имеют первую строку вида*:

```
function <ПКВ> = <имя процедуры >(<ПВВ>)
```

где ПКВ — Перечень Конечных Величин, ПВВ — Перечень Входных Величин. *Script-файлы такой строки не имеют.*

Принципиальное же отличие заключается в совершенно разном восприятии системой имен переменных в этих файлах.

В файл-функциях все имена переменных внутри файла, а также имена переменных, указанные в заголовке (ПКВ и ПВВ), воспринимаются как *локальные*, т.е. все значения этих переменных после завершения работы процедуры исчезают, и область оперативной памяти ПК, которая была отведена под запись значений этих переменных, освобождается для записи в нее значений других переменных.

В Script-файлах все используемые переменные образуют так называемое *рабочее пространство* (*work space*). Значения и смысл их сохраняются не только на протяжении работы программы, но и в течение всего сеанса работы с системой, а значит, и при переходе от выполнения одного Script-файла к выполнению другого. Таким образом, рабочее пространство является единым для всех Script-файлов, вызываемых в текущем сеансе работы с системой. Именно благодаря этому длинный Script-файл можно разбить на несколько фрагментов, оформить каждый из них в виде отдельного Script-файла, а в главном Script-файле вместо соответствующего фрагмента записать оператор вызова Script-файла, представляющего этот фрагмент. Таким образом обеспечивается компактное и наглядное представление даже довольно сложной программы.

За исключением указанных отличий, файлы-функции и Script-файлы оформляются одинаково.

2.2.2. Основные особенности оформления М-файлов

В дальнейшем под М-файлом будем понимать любой файл (файл-функцию или Script-файл), записанный на языке системы MatLAB.

Рассмотрим основные особенности записи текста программы (М-файла) на языке MatLAB.

- Обычно каждый оператор записывается в отдельной строке текста программы. Признаком конца оператора является символ (он не появляется в окне) возврата каретки и перехода на следующую строку, который вводится в программу при нажатии клавиши [Enter], т.е. при переходе на следующую строку.
- Можно размещать несколько операторов в одной строке. Тогда предыдущий оператор этой строки должен заканчиваться символом ";" или ",".
- Длинный оператор можно записывать в несколько строк. При этом предыдущая строка оператора должна заканчиваться тремя точками (...).
- Если очередной оператор не заканчивается символом ";", результат его действия при выполнении программы будет выведен в командное окно. Поэтому для предотвращения вывода на экран результатов действия оператора программы, запись этого оператора в тексте программы должна заканчиваться символом ";".
- Строка программы, начинающаяся с символа "%", не выполняется. Эта строка воспринимается системой MatLAB как *комментарий*. Таким образом, для ввода комментария в любое место текста программы достаточно начать соответствующую строку с символа "%".
- Строки комментария, предшествующие первому выполняемому оператору программы, т.е. такому, который не является комментарием, воспринимаются системой MatLAB как описание программы. Именно эти строки выводятся в командное окно, если в нем набрана команда:

```
help <имя файла>
```

- В программах на языке MatLAB отсутствует оператор окончания текста программы.
- В языке MatLAB переменные не описываются и не объявляются. Любое новое имя, появляющееся в тексте программы, воспринимается системой MatLAB как имя матрицы. Размер этой матрицы устанавливается при предварительном вводе значений ее элементов либо определяется действиями по установлению значений ее элементов, описанными в предыдущем операторе или процедуре. Эта особенность делает язык MatLAB очень простым в употреблении и привлекательным. В языке MatLAB невозможно использование матрицы или переменной, в которой предварительно не введены или не вычислены значения ее элементов (а значит, и не определены размеры этой матрицы). В этом случае при выполнении программы MatLAB появится сообщение об ошибке "Переменная не определена".
- Имена переменных могут содержать лишь буквы латинского алфавита или цифры и должны начинаться с буквы. Общее число символов в имени может достигать 19. В именах переменных могут использоваться как прописные, так и строчные буквы. Особенностью языка MatLAB является то, что *прописные и строчные буквы в именах различаются системой*. Например, символы "a" и "A" могут использоваться в одной программе для обозначения разных величин.

2.3. Создание простейших файл-функций (процедур)

2.3.1. Общие требования к построению

Как было отмечено ранее, файл-функция (процедура) должна начинаться со строки заголовка:

```
function [<ПКВ>] = <имя процедуры>(<ПВВ>)
```

Если перечень конечных (выходных) величин (ПКВ) содержит только один объект (в общем случае — матрицу), то файл-функция представляет собой обычную функцию (одной или нескольких переменных). Фактически даже в этом простейшем случае файл-функция является уже процедурой в обычном смысле других языков программирования, если выходная величина является вектором или матрицей. Первая строка в этом случае имеет вид:

```
function <имя переменной> = <имя процедуры>(<ПВВ>)
```

Если же в результате выполнения файл-функции должны быть определены (вычислены) несколько объектов (матриц), такая файл-функция представляет собой уже более сложный объект, который в программировании обычно называется либо процедурой (в языке Паскаль), либо подпрограммой. Общий вид первой строки в этом случае становится таким:

```
function [y1, y2, ..., yN] = <имя процедуры>(<ПВВ>)
```

т.е. перечень выходных величин y1, y2, ..., yN должен быть представлен как вектор-строка с элементами y1, y2, ..., yN (все они могут быть матрицами).

В простейшем случае функции одной переменной заголовок приобретет вид:

```
function y = func(x)
```

где func — имя функции (М-файла).

В качестве примера рассмотрим составление М-файла для функции:

$$y = f_1(x) = d^3 \cdot \mathrm{ctg}(x) \cdot \sqrt{\sin^4(x) - \cos^4(x)}$$

Для этого следует активизировать меню **File** командного окна MatLAB и выбрать в нем сначала команду **New**, а затем команду **M-file**. После этого на экране появится окно текстового редактора. В нем нужно набрать следующий текст:

```
function y = F1(x,d)
% Процедура, вычисляющая значение функции
% y = (d^3)*ctg(x)*sqrt(sin(x)^4-cos(x)^4).
% Обращение: y = F1(x,d).
y = (d^3)*cot(x).*sqrt(sin(x).^4-cos(x).^4);
```

Затем необходимо сохранить этот текст в файле под именем F1.m. Итак, М-файл создан. Теперь можно пользоваться этой функцией при расчетах. Так, если ввести команду:

```
» y = F1(1, 0.1)
```

то получим результат:

```
y = 4.1421e-004
```

Следует заметить, что аналогичным образом можно получить сразу вектор всех значений указанной функции при разных значениях аргумента, если последние собрать в некоторый вектор. Так, если сформировать вектор:

```
» zet= 0 : 0.3 : 1.8;
```

и обратиться к той же процедуре:

```
» my = F1(zet,1)
```


то получим:

```
Warning: Divide by zero
my =
     Columns 1 through 4
       NaN +   Infi    0 + 2.9369i    0 + 0.8799i    0.3783
     Columns 5 through 7
     0.3339             0.0706            -0.2209
```

Примечание

Возможность использования сформированной процедуры как для отдельных чисел, так и для векторов и матриц обусловлена применением в записи соответствующего М-файла вместо обычных знаков арифметических действий их аналогов с предшествующей точкой.

Чтобы избежать вывода на экран нежелательных промежуточных результатов, необходимо в тексте процедуры все вычислительные операторы завершать символом ";".

Как показывают приведенные примеры, имена переменных, указанные в заголовке файл-функции могут быть любыми, совпадать или нет с именами, используемыми при обращении к этой файл-функции, т.е. они носят формальный характер. Важно, чтобы структура обращения полностью соответствовала структуре заголовка в записи текста М-файла и чтобы переменные в этом обращении имели тот же тип и размер, как и в заголовке М-файла.

Чтобы получить информацию о созданной процедуре, достаточно набрать в командном окне команду:

```
» help f1
```

и в командном окне появится:

```
Процедура, вычисляющая значение функции
y = (d^3)*ctg(x)*sqrt(sin(x)^4-cos(x)^4).
Обращение: y = F1(x,d).
```

Другой пример. Построим график двух функций:

$$y_1 = 200\ \sin(x)/x, \quad y_2 = x^2$$

Для этого создадим М-файл, который вычисляет значения этих функций:

```
function y = myfun(x)
% Вычисление двух функций
% y(1) = 200 sin(x)/x, y(2) = x*x.
y(:,1) = 200*sin(x) ./ x;
y(:,2) = x .^ 2;
```

Теперь построим графики этих функций:

```
» fplot('myfun', [-20 20], 50, 2)
» grid
```

Результат показан на рис. 2.1.

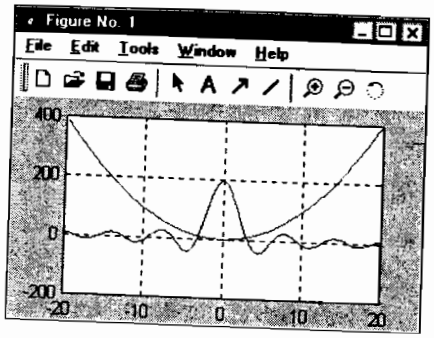

Рис. 2.1

Третий пример — создание файл-функции, вычисляющей значение функции:

$$y(t) = k_1 + k_2 * t + k_3 * \sin(k_4 * t + k_5)$$

В этом случае удобно объединить совокупность коэффициентов k в единый вектор K:

```
K = [k1 k2 k3 k4 k5]
```

и создать M-файл:

```
function y = dvob(x, K)
% Вычисление функции
% y = K(1)+K(2)*x+K(3)*sin(K(4)*x+K(5)),
% где K – вектор из пяти элементов.
% Применяется для определения текущих значений
% параметров движения подвижного объекта.
y = K(1)+K(2)*x+K(3)*sin(K(4)*x+K(5));
```

Тогда расчет, например, 11-ти значений этой функции можно осуществить следующим образом:

```
» K = ones(1,5);
» t = 0:1:10;
» fi = dvob(t, K)
fi = 1.8415  2.9093  3.1411  3.2432  4.0411  5.7206 7.6570
     8.9894  9.4560  10.0000
```


2.3.2. Типовое оформление процедуры-функции

М-файл процедуры-функции рекомендуется оформлять по такой схеме:

```
function [<Выход>] = <имя функции>(<Вход>)
% <Краткое пояснение назначения процедуры
% Входные переменные
% <Подробное пояснение о назначении, типе и размерах
% каждой из переменных, указанных в перечне <Вход>>
% Выходные переменные
% <Подробное пояснение о назначении, типе и размерах
% каждой из переменных перечня <Выход>
% и величин, используемых в процедуре как глобальные>
% Использование других функций и процедур
% <Раздел заполняется, если процедура содержит обращения
% к другим процедурам, кроме встроенных>
        < П у с т а я    с т р о к а >
% Автор: <Указывается автор процедуры, дата создания
% варианта процедуры и организация, в которой создана
% программа>
< Текст исполняемой части процедуры >
```

Здесь: <Выход> — перечень выходных переменных процедуры, <Вход> — перечень входных переменных, разделенных запятыми.

Примечание

При использовании команды **help <имя процедуры>** в командное окно выводятся строки комментария до первой пустой строки.

2.4. Создание Script-файлов

2.4.1. Основные особенности Script-файлов

Как уже было отмечено, основные особенности Script-файлов таковы:

- Script-файлы являются независимо (самостоятельно) исполняемыми блоками операторов и команд;
- все используемые переменные образуют так называемое *рабочее пространство*, которое является общим для всех исполняемых Script-файлов; из этого следует, что при выполнении нескольких Script-файлов имена переменных в них должны быть согласованы, так как одно имя означает в каждом из них один и тот же объект вычислений;
- отсутствует заголовок, т.е. первая строка определенного вида и назначения;
- обращение к Script-файлам не требует указания никаких имен переменных: все переменные формируются в результате выполнения программы либо сформированы ранее и существуют в рабочем пространстве.

Необходимо отметить, что рабочее пространство Script-файлов недоступно для файл-функций, которые в нем используются. В файл-функциях невозможно использовать значения, которые приобретают переменные в Script-файле, обходя заголовок файл-функции (так как все переменные файл-функции являются локальными). Единственной возможностью сделать так, чтобы внутри файл-функции некоторая переменная рабочего пространства сохранила свое значение и имя, является специальное объявление этой переменной в Script-файле как глобальной с помощью служебного слова **global**. Кроме того, аналогичная запись должна содержаться и в тексте М-файла той файл-функции, которая будет использовать значения соответствующей переменной Script-файла.

Например, можно перестроить файл-функции первого и третьего примеров из предыдущего раздела, вводя коэффициенты соответствующих функций как глобальные переменные:

```
function y = dvob1(x)
% Вычисления функции
% y = K(1)+K(2)*x+K(3)*sin(K(4)*x+K(5)),
% где K – глобальный вектор из пяти элементов.
% Применяется для определения текущих значений
% параметров движения подвижного объекта.
global K
y = K(1)+K(2)*x+K(3)*sin(K(4)*x+K(5));
```

Чтобы использовать новую файл-функцию **dvob1** в Script-файле, в последнем до обращения к этой функции должна быть записана строка:

```
global K
```

и определен вектор-строка K из пяти элементов (заданы их значения).

Если в одной строке объявляются как глобальные несколько переменных, они должны отделяться *пробелами* (не запятыми!).

2.4.2. Ввод и вывод информации в диалоговом режиме

Для обеспечения взаимодействия с пользователем в процессе выполнения М-файла в системе MatLAB используются такие команды:

```
disp, sprintf, input, menu, keyboard, pause
```

Команда **disp** осуществляет вывод значений указанной переменной или указанного текста в командное окно. Обращение к ней имеет вид:

```
disp(<переменная или текст в апострофах>)
```


Особенностью этой команды является то, что аргумент у нее может быть только один. Поэтому невозможно без специальных мер осуществить вывод нескольких переменных и, особенно, объединение текста с численными значениями некоторых переменных, что часто является необходимым для удобного представления информации.

Для устранения этого недостатка используют несколько способов.

Чтобы вывести значения нескольких переменных в одну строку (например, при создании таблиц данных), нужно создать единый объект, который содержал бы все эти значения. Это можно сделать, объединив соответствующие переменные в единый вектор, пользуясь операцией создания вектора-строки:

```
x = [x1 x2 ... xN]
```

Тогда вывод значений нескольких переменных в одну строку будет иметь вид:

```
disp([x1 x2 ... xN])
```

Приведем пример:

```
» x1=1.24; x2=-3.45; x3=5.76; x4=-8.07;
» disp([x1 x2 x3 x4])
    1.2400   -3.4500    5.7600   -8.0700
```

Аналогичным образом можно объединять несколько текстовых переменных, например:

```
» x1=' psi  '; x2='  fi   '; x3='  teta  '; x4='   w1  ';
» disp([x1 x2 x3 x4])
   psi    fi     teta     w1
```

Гораздо сложнее объединить в одну строку текст и значения переменных, что часто бывает необходимым. Трудности возникают потому, что нельзя объединять текстовые и числовые переменные, так как они являются данными различных типов. Одним из путей преодоления этого препятствия является перевод числового значения переменной в символьную (текстовую) форму. Это возможно, если воспользоваться функцией **num2str**, которая осуществляет такое преобразование. Запись:

```
y = num2str(x)
```

преобразует числовое значение переменной x в текстовое представление. При этом форма представления определяется установленным режимом вывода чисел на экран (Numeric Format). Например:

```
» x = -9.30876e-15
x =
-9.3088e-015
» y = num2str(x)
```

```
y =
-9.309e-015
```

Если T — текстовая переменная или некоторый текст, а X — числовая переменная, то вывод их в одной строке можно обеспечить обращением:

```
disp([T num2str(X)])
```

Рассмотрим пример:

```
x =
-9.3088e-015
» T = 'Значение параметра равно ';
» disp([T x])
Значение параметра равно
» disp([T num2str(x)])
Значение параметра равно -9.309e-015
```

Из этого примера следует, что "механическое" объединение текстовой и числовой переменных не приводит к тому результату, что использование функции **num2str**.

Другим способом достижения того же результата является использование функции **sprintf**. Обращение к ней имеет вид:

```
Y = sprintf('<текст1>   %g <текст2>', X)
```

В результате создается текстовая строка Y, состоящая из текста, указанного в <текст1>, и значения числовой переменной X в соответствии с форматом %g, причем текст из фрагмента <текст2> размещается после значения переменной X. Эту функцию можно использовать в команде **disp** в виде:

```
disp(sprintf('<текст>   %g', X))
```

Пример:

```
» disp(sprintf('Параметр1 =  %g ',x))
Параметр1 =  -9.30876e-015
```

Ввод информации с клавиатуры в диалоговом режиме можно осуществить с помощью функции **input**. Обращение к ней вида:

```
x = input('<приглашение>')
```

приводит к следующим действиям ПК. Выполнение операторов программы прекращается. ПК переходит в режим ожидания окончания ввода информации с клавиатуры. После окончания ввода с клавиатуры (которое определяется нажатием клавиши [Enter]) введенная информация запоминается в программе под именем x и выполнение программы продолжается.

Удобным инструментом выбора некоторой из альтернатив будущих вычислительных действий является функция **menu** MatLAB, которая создает текущее окно меню пользователя. Функция **menu** имеет такой формат обращения:

```
k = menu ('Заголовок меню', 'Альтернатива 1',
          'Альтернатива 2', ... , 'Альтернатива n')
```

Такое обращение приводит к появлению на экране меню, изображенного на рис. 2.2. Выполнение программы временно приостанавливается, и система ожидает выбора одной из кнопок с альтернативами. После правильного ответа выходному параметру k присваивается значение номера выбранной альтернативы (1, 2 или 3). В общем случае число альтернатив может быть до 32.

Теперь, в зависимости от значения этого параметра, можно построить процесс разветвления вычислений, например выбора параметра, значение которого нужно изменить.

Команда **pause** временно прекращает выполнение программы до тех пор, пока пользователь не нажмет любую клавишу клавиатуры. Если после названия команды указать в скобках некоторое положительное целое число n, то задержка выполнения программы будет осуществлена на протяжении n секунд.

Если в тексте M-файла встречается команда **keyboard**, то при выполнении программы выполнение M-файла прекращается и управление передается клавиатуре. Этот специальный режим работы сопровождается появлением в командном окне MatLAB нового вида приглашения к действиям:

```
k>>
```

В этом режиме пользователь может осуществлять любые действия, проверять или изменять данные. При этом ему доступны все команды и процедуры системы MatLAB. Для завершения работы в этом режиме необходимо ввести команду **return**. Тогда система продолжит работу программы с оператора, следующего за командой **keyboard**.

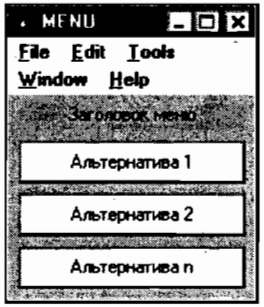

Рис. 2.2

2.4.3. Организация повторения действий

Одной из главных задач при самостоятельном создании программы является обеспечение возвращения к началу программы с целью продолжения ее выполнения при новых значениях исходных данных.

Пусть основные операторы созданной программы расположены в Script-файле с именем ScrFil_yadro.m. Тогда схема обеспечения возврата к началу выполнения этого Script-файла может быть, скажем, такой:

```
flag = 0;
while flag == 0
   ScrFil_yadro
   kon=0;
   kon=input('Закончить работу-<3>, продолжить-<Enter>');
   if kon==3,
      flag=3;
   end
end
```

В этом случае Script-файл ScrFil_yadro будет повторно выполняться до тех пор, пока на вопрос 'Закончить работу — <3>, продолжить — <Enter>' не будет введен с клавиатуры ответ "3". Если же ответ будет именно таким, цикл закончится и будут выполняться следующие за этим циклом операторы. Естественно, что переменная flag не должна изменять свое значение в Script-файле ScrFil_yadro.

Можно также с той же целью использовать механизм создания меню. В этом случае программу можно представить, к примеру, так:

```
k=1;
while k==1
  ScrFile_Yadro
  k = menu('Что делать?','Продолжить работу','Закончить ...
           работу');
end
```

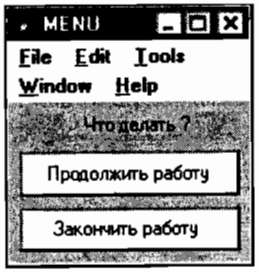

Рис. 2.3

Тогда после первого выполнения Script-файла ScrFil_Yadro на экране появится меню, изображенное на рис. 2.3, и при нажатии кнопки первой альтернативы значение k останется равным единице, цикл повторится, а при нажатии второй кнопки k станет равным 2, выполнится выход из цикла и программа завершится.

2.4.4. Организация изменения данных в диалоговом режиме

Повторение действий, содержащихся в ядре ScrFil_yadro, имеет смысл только в том случае, когда в начале этого ядра обеспечено выполнение действий по изменению некоторых из исходных величин. MatLAB содержит ряд удобных средств, позволяющих осуществлять изменение данных в диалоговом режиме с использованием стандартных меню-окон пользователя.

Организацию диалогового изменения данных рассмотрим на примере некоторых 5-ти параметров, которые назовем Параметр1, Параметр2, ..., Параметр5.

Пусть их обозначения как переменных в программе таковы: x1, x2, ..., x5. Тогда меню выбора параметра для изменения его значения должно содержать 6 альтернатив: 5 из них предназначены для выбора одного из указанных параметров, а последняя — должна предоставить возможность выхода из меню, когда значения всех параметров установлены.

Поэтому вариант оформления такого меню может быть, например, следующим:

```
k = menu('Что изменить?','Параметр1','Параметр2', ...
    'Параметр3','Параметр4','Параметр5','Ничего не менять')
```

что приведет к появлению окна, представленного на рис. 2.4.

Легко заметить недостатки такого оформления окна меню. Чтобы сделать правильный выбор, значение какого именно параметра следует изменить и как, пользователь должен иметь перед глазами не только перечень параметров, которые можно изменить, но и текущие значения этих параметров. Поэтому на каждой кнопке меню должна размещаться также информация о текущем значении соответствующего параметра. Это можно сделать, используя ранее упомянутую функцию **sprintf**, например, таким образом:

```
x1=-1.89; x2=239/78; x3=-2/56e-3;
x4=7.28e-15; x5=1/023e-32;
   k = menu('    Что  изменять?   ', ...
   sprintf (' Параметр1  x1 =   %g', x1),...
   sprintf (' Параметр2  x2 =   %g', x2),...
   sprintf (' Параметр3  x3 =   %g', x3),...
   sprintf (' Параметр4  x4 =   %g', x4),...
   sprintf (' Параметр5  x5 =   %g', x5),...
    '  Ничего не менять  ')
```

Результат приведен на рис. 2.5.

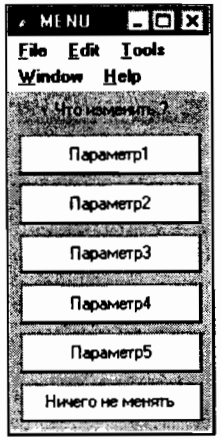

Рис. 2.4

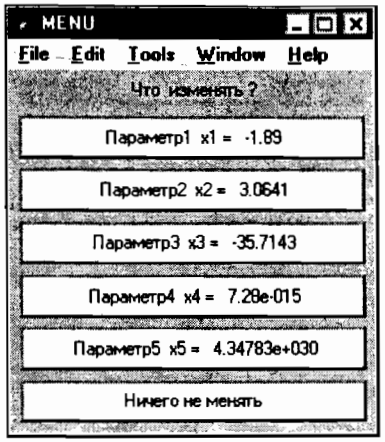

Рис. 2.5

Меню позволяет выбрать параметр, который нужно изменить, однако не обеспечивает самого изменения выбранного параметра. Это изменение должно быть осуществлено с помощью ввода нового значения с клавиатуры, скажем, так:

```
x = input([sprintf('Текущее значение x =%g',x) ...
       '   Новое значение x = '])
```

Если ввести команды:

```
» x = 3.02e-2;
» x=input([sprintf('Текущее значение x = %g',x), ...
       '   Новое значение x = '])
```

то в командном окне появится запись:

```
Текущее значение x = 0.0302  Новое значение x =
```

Выполнение программы приостановится, и программа будет ожидать ввода информации с клавиатуры. Если теперь набрать на клавиатуре число 0.073 и нажать клавишу [Enter], то в командном окне появится следующая запись:

```
Текущее значение x = 0.0302  Новое значение x = 0.073
   x = 0.0730
```

Чтобы предупредить повторный вывод на экран введенного значения, необходимо строку с функцией **input** завершить символом ";".

Теперь необходимо организовать выбор разных видов такого типа операторов в соответствии с отдельными выбранными параметрами. Для этого можно использовать оператор условного перехода, например, так:

```
if k==1,
    x1 = input([sprintf('Текущее значение x1 = %g', x1) ...
                '  Новое значение  x1= ']);
  elseif k==2,
    x2 = input([sprintf('Текущее значение x2 = %g', x2) ...
                '  Новое значение  x2= ']);
  elseif k==3,
    x3 = input([sprintf('Текущее значение x3 = %g', x3) ...
                '  Новое значение  x3= ']);
  elseif k==4,
    x4 = input([sprintf('Текущее значение x4 = %g', x4) ...
                '  Новое значение  x4= ']);
  elseif k==5,
    x5 = input( [sprintf('Текущее значение x5 = %g', x5) ...
                '  Новое значение  x5= ']);
  end
```

Для того чтобы можно было проконтролировать правильность ввода новых значений, обеспечить возможность их корректировки и последовательного изменения всех желаемых параметров, нужно, чтобы после ввода нового значения любого параметра на экране снова появлялось то же меню, но уже с откорректированными значениями. При этом конец работы с меню должен наступить только при условии выбора последней альтернативы меню "Ничего не менять", соответствующей значению k, равному 6.

Поэтому предыдущие операторы следует заключить в цикл:

```
k=1;
while k<6
   k = menu( '    Что  изменить ?   ', ...
          sprintf (' Параметр1  x1 = %g', x1),...
          sprintf (' Параметр2  x2 = %g', x2),...
          sprintf (' Параметр3  x3 = %g', x3),...
          sprintf (' Параметр4  x4 = %g', x4),...
          sprintf (' Параметр5  x5 = %g', x5),...
          '  Ничего не менять  ');
   if k==1,
     x1 = input([sprintf('Текущее значение x1 = %g', x1) ...
                 '  Новое значение x1= ']);
   elseif k==2,
     x2 = input([sprintf('Текущее значение x2 = %g', x2) ...
                 '  Новое значение x2= ']);
   elseif k==3,
     x3 = input([sprintf('Текущее значение x3 = %g', x3) ...
                 '  Новое значение x3= ']);
```

```
   elseif k==4,
    x4 = input([sprintf('Текущее значение x4 = %g', x4) ...
                '  Новое значение x4= ']);
   elseif k==5,
    x5 = input([sprintf('Текущее значение x5 = %g', x5) ...
                '  Новое значение x5= ']);
   end
end
```

Таким образом организуется возможность достаточно удобного изменения значений параметров в диалоговом режиме.

Если входных параметров, значения которых нужно изменять, довольно много, следует объединить их в компактные группы (желательно по какому-то общему свойству, отличающему определенную группу от других) и аналогичным образом обеспечить диалоговое изменение, используя отдельное меню для каждой группы. Очевидно, при этом необходимо предварительно обеспечить выбор одной из этих групп параметров через дополнительное меню.

2.4.5. Типовая структура и оформление Script-файла

При написании текста самостоятельно исполняемой программы в виде Script-файла необходимо принимать во внимание следующее.

- Удобно оформлять весь процесс диалогового изменения параметров в виде отдельного Script-файла, к примеру, с именем ScrFil_Menu, где под сокращением ScrFil подразумевается имя основного (собирательного) Script-файла.
- Так как уже в самом начале работы с программой в меню выбора изменяемого параметра должны сразу выводиться некоторые значения параметров, перед главным циклом программы, обеспечивающим возвращение к началу вычислений, необходимо разместить часть программы, которая задает начальные значения всех параметров. Кроме того, в начале работы программы очень удобно вывести на экран краткую информацию о назначении программы, более детальную информацию об исследуемой математической модели с указанием места в ней и смысла всех исходных параметров, а также исходные ("вшитые") значения всех параметров этой модели. Это желательно также оформить в виде отдельного Script-файла, например, с именем ScrFil_Zastavka.
- По завершении работы программы обычно возникает необходимость несколько упорядочить рабочее пространство, например очистить его от введенных глобальных переменных (оставаясь в рабочем пространстве, они препятствуют корректной работе другой программы, которая может иметь новые глобальные переменные или переменные с теми же именами, но иными по типу, смыслу и значению), закрыть открытые программой графические

окна (фигуры) и т.д. Эту завершающую часть тоже можно оформить как отдельный Script-файл, назвав его ScrFil_Kin.

В целом типовая схема оформления Script-файла отдельной программы может быть представлена в следующем виде:

```
% <Обозначение Script-файла  (ScrFil.m)>
% <Текст комментария с описанием назначения программы>
        < П у с т а я    с т р о к а >
% Автор <Фамилия И.О., дата создания, организация>
%
ScrFil_Zastavka
k = menu('Что делать? ','Продолжить работу ', ...
         ' Закончить работу   ');
if k==1
   while k==1
     ScrFil_Menu
     ScrFile_Yadro
     k = menu('Что делать? ','Продолжить работу  ', ...
              ' Закончить работу  ');
   end
end
ScrFil_Kin
```

2.5. Графическое оформление результатов

2.5.1. Общие требования к представлению графической информации

Вычислительная программа, создаваемая инженером-разработчиком, в большинстве случаев предназначена для исследования поведения разрабатываемого устройства при разных условиях его эксплуатации, при различных значениях его конструктивных параметров или для расчета определенных параметров его поведения. Информация, получаемая в результате выполнения вычислительной инженерной программы, как правило, имеет форму некоторого ряда чисел, каждое из которых соответствует определенному значению некоторого параметра (аргумента). Такую информацию удобнее всего обобщать и представлять в графической форме.

Требования к оформлению инженерной графической информации отличаются от требований к обычным графикам в математике. Руководствуясь полученной графической информацией, пользователь-инженер должен иметь возможность принять какое-то решение о выборе значений некоторых конструктивных параметров, характеризующих исследуемый процесс или техническое устройство, с целью, чтобы прогнозируемое поведение технического устройства отвечало некоторым заданным условиям. Поэтому инженерные графики должны быть, как говорят, читабельными, т.е. иметь такой вид, чтобы из них легко было "считать" значения функции при любых значениях аргумента и наоборот с относительной погрешностью в

несколько процентов. Это становится возможным, если координатная сетка графиков соответствует некоторым определенным целым числам какого-либо десятичного разряда. Как уже ранее отмечалось, графики, построенные системой MatLAB, полностью отвечают этим требованиям.

Кроме этого, инженерная графическая информация должна сопровождаться достаточно подробным описанием, поясняющим, какой объект и по какой математической модели исследован, должны быть приведены числовые значения параметров исследуемого объекта и математической модели. Не окажется лишним и указание имени программы, с помощью которой получена эта графическая информация, а также сведений об авторе программы и исследователе, чтобы пользователю было понятно, к кому и куда надо обращаться для наведения справок о полученной информации.

Задачей инженерной программы часто является сравнение нескольких функций, полученных при разных сочетаниях конструктивных параметров либо параметров внешних воздействий. Такое сравнение удобнее и нагляднее проводить, если упомянутые функции представлены в виде графиков.

При этом нужно обратить внимание на следующее.

- При необходимости сравнивать графики функций одного аргумента, диапазоны изменения которых не слишком отличаются друг от друга (не более чем на порядок, т.е. не более чем в десять раз), сравнение удобнее всего осуществлять по графикам этих функций, построенных в одном графическом поле (т.е. имеющих общие координатные оси); в этом случае следует выводить графики при помощи одной функции **plot**.
- Если при тех же условиях диапазоны изменения функций значительно различаются, можно предложить два подхода:
  - в случае, когда все сравниваемые функции являются значениями величин одинаковой физической природы и все эти значения положительны, графики следует выводить также в одно графическое поле, но *в логарифмическом масштабе* (т.е. использовать процедуру **semilogy**);
  - когда все функции имеют разную физическую природу, но аргумент у них общий и изменяется в одном диапазоне, графики нужно строить в одном графическом окне (фигуре), но в разных графических полях (пользуясь для этого несколькими отдельными обращениями к функции **plot** в разных подокнах графического окна, что обеспечивается применением процедуры **subplot**); при этом удобно размещать отдельные графики друг под другом таким образом, чтобы одинаковые значения аргумента во всех графиках размещались на одной вертикали.

- Кроме упомянутых ранее надписей в каждом графическом поле, законченное графическое оформление любого графического окна (фигуры) обязательно должно содержать *дополнительную текстовую информацию* такого содержания:
  - краткое сообщение об объекте исследования;
  - математическая модель, положенная в основу осуществленных в программе вычислений с указанием имен параметров и переменных;
  - информация об использованных значениях параметров;
  - информация о полученных значениях некоторых вычисленных интегральных параметров;
  - информация об имени М-файла использованной программы, исполнителе работы и дате проведения вычислительного эксперимента;
  - информация об авторе использованной программы и организации, где он работает.

Последнее требование ставит перед необходимостью выделять (с помощью той же процедуры **subplot**) в каждой фигуре место для вывода указанной текстовой информации.

2.5.2. Разбиение графического окна на подокна

Как следует из сказанного, при создании законченного графического инженерного документа в системе MatLAB необходимо использовать процедуру **subplot**. Общее назначение и применение функции **subplot** описаны в разд. 1.5.3.

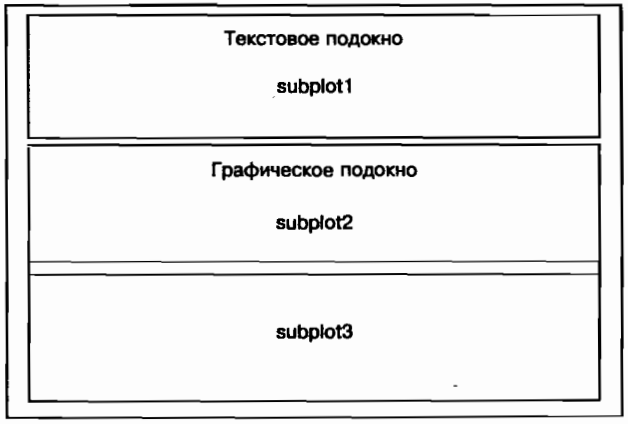

Рис. 2.6

Рассмотрим, как обеспечить желаемое разбиение всего графического окна на отдельные поля графиков и текстовое подокно.

Пусть требуется разбить все поле графического окна так, чтобы верхняя треть окна образовала поле вывода текста, а нижние две трети образовали единое поле вывода графиков. Это можно осуществить таким образом:

- перед выводом текстовой информации в графическое окно надо установить команду **subplot(3,1,1)**, которая показывает, что весь графический экран разделен на три одинаковые части по вертикали, а для последующего вывода будет использована верхняя из этих трех частей (рис. 2.6);
- выводу графиков в графическое окно должна предшествовать команда **subplot(3,1,[2 3])**, в соответствии с которой графическое окно разделяется, как и ранее, на три части по вертикали, но теперь для вывода графической информации будет использовано пространство, объединяющее второе и третье из созданных подокон (полей) (обратите внимание, что *подокна объединяются таким же образом, как элементы вектора в вектор-строку*).

Если требуется создать три отдельных поля графиков один под другим на три четверти экрана по горизонтали, а текстовую информацию разместить в последней четверти по горизонтали, то это можно сделать таким образом:

- разделить все пространство фигуры на 12 частей — на 3 части по вертикали и на 4 части по горизонтали; при этом подокна будут расположены так, как показано на рис. 2.7;
- чтобы организовать вывод графиков в первое графическое подокно надо предварительно ввести команду **subplot(3,4,[1 2 3])**, которая объединит подокна sp1, sp2 и sp3 в единое графическое подокно;

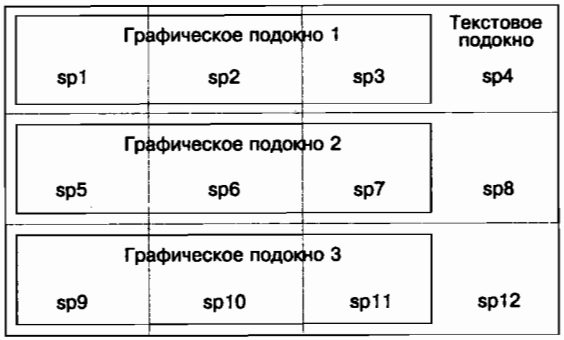

Рис. 2.7

- аналогично, выводу графиков во второе графическое подокно должно предшествовать обращение к команде **subplot(3,4,[5 6 7])**, а выводу графиков в третье графическое подокно — **subplot(3,4,[9 10 11])**;
- наконец, к оформлению текста можно приступить после обращения:

```
» subplot(3,4,[4 8 12])
```

2.5.3. Вывод текста в графическое окно (подокно)

Если поочередно сформировать подокна, к примеру, в соответствии с последней схемой, не осуществляя никаких операций по выведению графиков или текста:

```
» subplot(3,4,[5 6 7])
» subplot(3,4,1:3)
» subplot(3,4,9:11)
» subplot(3,4,[4 8 12])
```

в окне фигуры появится изображение, представленное на рис. 2.8.

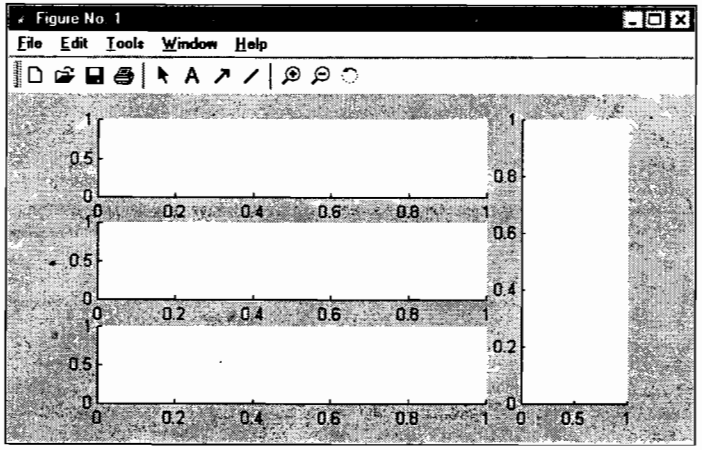

Рис. 2.8

Из рисунка видно, что:

- после обращения к процедуре **subplot** в соответствующем подокне отображаются оси координат с обозначением делений;
- начальный диапазон изменений координат по осям подокна — от 0 до 1;
- поле выведения графиков занимает не все пространство соответствующего подокна — остается некоторое место вокруг поля

графика для вывода заголовка графика, надписей по осям координат и др.

Поэтому для вывода текста в одно из подокон нужно сначала очистить это подокно от изображения осей координат и надписей на них. Это делается с помощью команды:

```
axis('off')
```

Так, если ввести эту команду после предыдущих команд, в окне фигуры исчезнет изображение координатных осей последнего подокна (рис. 2.9). Теперь можно начинать вывод текста в это подокно.

Основной функцией, обеспечивающей выведение текста в графическое окно, является функция **text**. Обобщенная форма обращения к ней имеет вид:

```
h = text(x, y, '<текст>','FontName','<название шрифта>',...
         'FontSize',<размер шрифта в пикселях>)
```

Данная функция осуществляет вывод указанного текста указанным шрифтом указанного размера, начиная с точки подокна, определяемой координатами x и y соответствующего поля графика подокна. При этом координаты x и y измеряются в единицах величин, откладываемых вдоль соответствующих осей графика подокна. Так как, как мы убедились, диапазон изменения этих координат равен [0 ... 1], то для того, чтобы поместить начало текста в точку внутри поля графика, необходимо, чтобы его координаты x и y были в этом диапазоне. Однако можно использовать и более широкий диапазон, учитывая то, что поле подокна больше поля его графика.

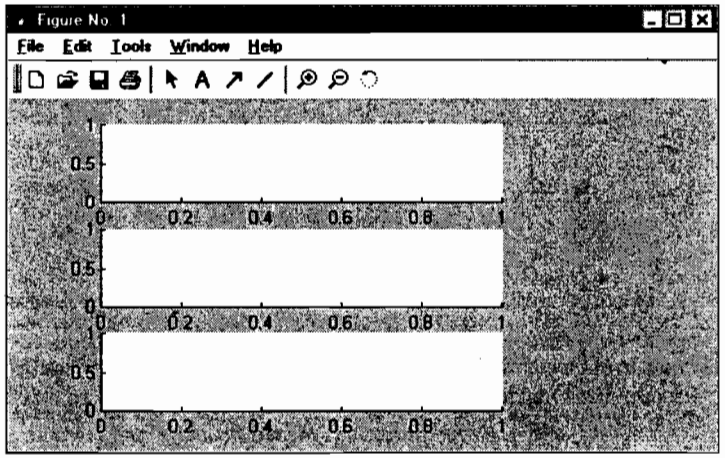

Рис. 2.9

Рассмотрим пример текстового оформления на следующем фрагменте программы:

```
subplot(3,4,1:3); subplot(3,4,5:7); subplot(3,4,9:11);
subplot(3,4,[4;8;12]);
axis('off');
% Процедура вывода данных в текстовое поле графического окна
D1 = [2 1 300 1 50];
D2 = [0.1 0.02 -0.03 0  1 4 -1.5 2 0.1 -0.15 0 0];
D5 = [0.001 0.01 15 16];
sprogram = 'vspl';  sname = 'Лазарев Ю.Ф.';
h1=text(-0.2,1,'Исходные параметры:','FontSize',12);
h1=text(0,0.95,'Гиротахометров','FontSize',10);
h1=text(0.2,0.9,sprintf('H = %g ',D1(3)),'FontSize',10);
h1=text(-0.2,0.85,sprintf('R = %g ',D1(4)),'FontSize',10);
h1=text(0.6,0.85,sprintf('S = %g ',D1(5)),'FontSize',10);
h1=text(-0.2,0.8,sprintf('J = %g ',D1(1)),'FontSize',10);
h1=text(0.6,0.8,sprintf('J2 = %g ',D1(2)),'FontSize',10);
h1=text(0,0.75,'Внешних воздействий','FontSize',10);
h1=text(-0.2,0.7,sprintf('pst0 = %g ',D2(1)),'FontSize',10);
h1=text(0.6,0.7,sprintf(' tet0 = %g ',D2(2)),'FontSize',10);
h1=text(0.2,0.66,sprintf(' fit0 = %g ',D2(3)),'FontSize',10);
h1=text(-0.2,0.62,sprintf('psm = %g ',D2(4)),'FontSize',10);
h1=text(0.6,0.62,sprintf(' tem = %g ',D2(5)),'FontSize',10);
h1=text(0.2,0.58,sprintf(' fim = %g ',D2(6)),'FontSize',10);
h1=text(-0.2,0.54,sprintf('omps = %g ',D2(7)),'FontSize',10);
h1=text(0.6,0.54,sprintf(' omte = %g ',D2(8)),'FontSize',10);
h1=text(0.2,0.5,sprintf('omfi = %g ',D2(9)),'FontSize',10);
h1=text(-0.2,0.46,sprintf('eps = %g ',D2(10)),'FontSize',10);
h1=text(0.6,0.46,sprintf(' ete = %g ',D2(11)),'FontSize',10);
h1=text(0.2,0.42,sprintf(' efi = %g ',D2(12)),'FontSize',10);
h1=text(0,0.35,'Интегрирования','FontSize',10,'FontUnderline','on');
h1=text(0,0.3,sprintf('h = %g ',D5(1)),'FontSize',10);
h1=text(0,0.25,sprintf('hpr = %g ',D5(2)),'FontSize',10);
h1=text(0,0.2,sprintf('t = %g ',D5(3)),'FontSize',10);
h1=text(0,0.15,sprintf('tfinal = %g ',D5(4)),'FontSize',10);
h1=text(-0.3,0.12,'----------------------','FontSize',10);
tm=fix(clock); Tv=tm(4:6);
h1=text(-0.2,0.08,['Программа    ' sprogram],'FontSize',10);
h1=text(-0.3,0.04,['Расчеты провел    ' sname],'FontSize',10);
h1=text(-0.3,0,[sprintf(' %g :',Tv) '         ' date],'FontSize',10);
h1=text(-0.3,-0.04,'----------------------','FontSize',10);
h1=text(-0.3,-0.08,'Ukraine,   KPI,   cath. PSON','FontSize',10);
```

Выполнение этой программы приводит к появлению в окне фигуры изображения, представленного на рис. 2.10.

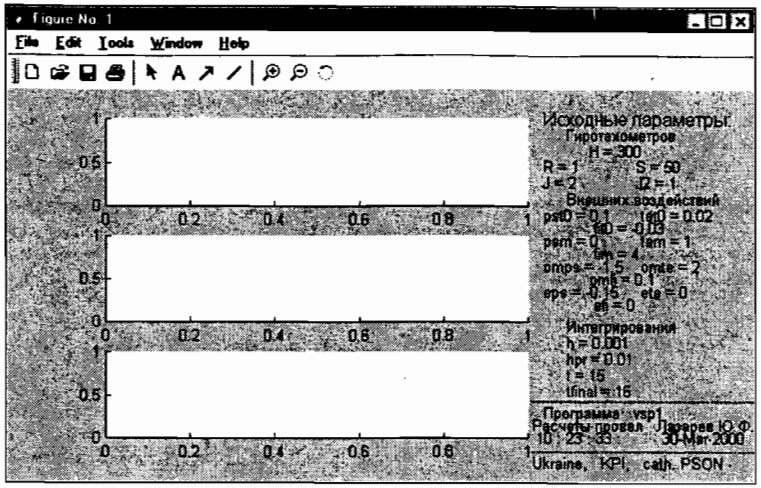

Рис. 2.10

2.6. Создание функций от функций

Некоторые алгоритмы являются общими для всех типов функций определенного вида. Поэтому их программная реализация едина для всех функций такого вида, но требует использования алгоритма вычисления конкретной функции. Последний алгоритм может быть зафиксирован в виде определенной файл-функции. Чтобы первый, более общий алгоритм был приспособлен для любой функции, нужно, чтобы имя этой функции было некоторой переменной, принимающей определенное значение (текстового имени файл-функции) только при обращении к основному алгоритму.

Такие функции от функций уже рассматривались в разд. 2.1. К ним принадлежат процедуры:

- вычисления интеграла от функции, которые требуют указания имени М-файла, содержащего вычисления значения подынтегральной функции;
- численного интегрирования дифференциальных уравнений, использование которых требует указания имени М-файла, в котором вычисляются правые части уравнений в форме Коши;
- алгоритмов численного вычисления корней нелинейных алгебраических уравнений (нулей функций), которые требуют указания файл-функции, нуль которой отыскивается;
- алгоритмов поиска минимума функции, которые, в свою очередь, надо задавать соответствующими М-файлами и т.д.

На практике довольно часто возникает необходимость создавать собственные такие процедуры. MatLAB предоставляет следующие возможности.

2.6.1. Процедура feval

В MatLAB любая функция (процедура), назовем ее FUN1, может быть выполнена не только с помощью обычного обращения:

```
[y1,y2,...,yk] = FUN1(x1,x2,...,xn)
```

а и при помощи специальной процедуры **feval**:

```
[y1,y2,...,yk] = feval('FUN1',x1,x2,...,xn)
```

где имя функции FUN1 является уже одной из входных переменных (текстовой) и поэтому помещается в апострофы.

Преимуществом вызова функции во второй форме является то, что этот вызов не меняет формы при изменении имени функции, например, на FUN2. Это позволяет унифицировать обращение ко всем функциям определенного типа, т.е. таким, которые имеют одинаковое число входных и выходных параметров определенного типа. При этом имя функции (а значит, и сама функция, которая используется) может быть произвольным и изменяться при повторных обращениях.

Так как при вызове функции с помощью процедуры **feval** имя функции рассматривается как один из входных параметров процедуры, то его (имя функции) можно использовать как переменную и оформлять в М-файле обращение к ней, не зная еще конкретного имени функции.

2.6.2. Примеры создания процедур от функций

Процедура метода Рунге-Кутта 4-го порядка численного интегрирования ОДУ

Пусть задана система обыкновенных дифференциальных уравнений (ОДУ) в форме Коши:

$$\frac{dy}{dt} = Z(y,t)$$

где y — вектор переменных состояния системы; t — аргумент (время); Z — вектор заданных нелинейных функций, который, собственно, и определяет конкретную систему ОДУ.

Если значение вектора y в момент времени t известно, то общая формула, по которой может быть найден вектор *yout* значений переменных состояния системы в момент времени tout = t + h (где h — шаг интегрирования), имеет вид:

$$yout = y + h^* F(y,t)$$

Функция $F(y,t)$ связана с вектором Z и может приобретать разный вид в зависимости от выбранного метода численного интегрирования. Для метода Рунге-Кутта 4-го порядка выберем такую ее форму:

$$F = (k_1 + 3 \cdot k_2 + 3 \cdot k_3 + k_4) / 8$$

где

$k_1 = Z(y,t)$

$k_2 = Z(y + h \cdot k_1 / 3, t + h / 3)$

$k_3 = Z(y + h \cdot k_2 - h \cdot k_1 / 3, t + 2 \cdot h / 3)$

$k_4 = Z(y + h \cdot k_3 - h \cdot k_2 - h \cdot k_1, t + h)$

Создадим М-файл процедуры, осуществляющей эти вычисления, и назовем его rko43:

```
function [tout,yout] = rko43(Zpfun,h,t,y)
% RKO43 Интегрирование ОДУ методом
% Рунге-Кутта 4-го порядка,
% правые части которых заданы процедурой Zpfun.
% Входные переменные:
% Zpfun - строка символов, содержащая имя процедуры
% вычисления правых частей ОДУ.
% Обращение: z = fun(t,y), где Zpfun = 'fun'
% t - текущий момент времени
% y - вектор текущих значений переменных состояния
% z - вычисленные значения производных z(i) = dy(i)/dt.
% h - шаг интегрирования
% t - предыдущий момент времени
% y - предыдущее значение вектора переменных состояния.
% Выходные переменные:
% tout - новый момент времени
% yout - вычисленное значение вектора y через шаг интегрирования
% Ю.Ф.Лазарев,  274-67-31.
% Copyright (c) 1998 by The PSON, NTU of Ukraine "KPI".
% Расчет промежуточных значений производных
k1 = feval(Zpfun, t, y);
k2 = feval(Zpfun, t+h/3, y+h/3*k1);
k3 = feval(Zpfun, t+2*h/3, y+h*k2-h/3*k1);
k4 = feval(Zpfun, t+h, y+h*(k3+k1-k2));
% Расчет новых значений вектора переменных состояния
tout = t + h;
yout = y + h*(k1 + 3*k2 + 3*k3 + k4)/8;
% Конец процедуры RKO43
```

Следует иметь ввиду, что:

- обращение к процедуре вычисления правых частей не конкретизировано; имя этой процедуры входит в число входных переменных процедуры интегрирования и должно быть конкретизировано только при вызове последней;
- переменные k являются векторами-строками (так же, как и переменные y и z, вычисляемые в процедуре правых частей).

Процедура правых частей ОДУ маятника

Рассмотрим процесс создания процедуры вычисления правых частей ОДУ на примере уравнения маятника, точка подвеса которого поступательно перемещается со временем по гармоническому закону:

$$J \cdot \ddot{\varphi} + R \cdot \dot{\varphi} + mgl \cdot (1 + n_{my} \cdot \sin(\omega \cdot t + \varepsilon_y)) \cdot \sin\varphi = \\ = -mgl \cdot n_{mx} \cdot \sin(\omega \cdot t + \varepsilon_x) \cdot \cos\varphi$$

где J — момент инерции маятника; R — коэффициент демпфирования; mgl — опорный маятниковый момент маятника; n_{my} — амплитуда виброперегрузки точки подвеса маятника в вертикальном направлении; n_{mx} — амплитуда виброперегрузки в горизонтальном направлении; φ — угол отклонения маятника от вертикали; ω — частота колебаний точки подвеса; ε_x, ε_y — начальные фазы колебаний (по ускорениям) точки подвеса в горизонтальном и вертикальном направлениях.

Чтобы составить М-файл процедуры вычисления правых частей заданной системы ОДУ, прежде всего надо привести исходную систему ОДУ к форме Коши. Для этого введем обозначения:

$$y1 = \varphi, \quad y2 = \dot{\varphi}$$

Тогда исходное уравнение маятника можно представить в виде совокупности двух дифференциальных уравнений 1-го порядка:

$$\frac{dy1}{dt} = y2$$

$$\frac{dy2}{dt} = \{-mgl \cdot n_{mx} \cdot \sin(\omega \cdot t + \varepsilon_x) \cdot \cos(y1) - R \cdot y2 - \\ -mgl \cdot [1 + n_{my} \cdot \sin(\omega \cdot t + \varepsilon_y)] \cdot \sin(y1)\} / J$$

Сравнивая полученную систему с общей формой уравнений Коши, можно сделать вывод, что

$$z1 = y2$$

$$z2 = \{-mgl \cdot n_{mx} \cdot \sin(\omega \cdot t + \varepsilon_x) \cdot \cos(y1) - R \cdot y2 - \\ -mgl \cdot [1 + n_{my} \cdot \sin(\omega \cdot t + \varepsilon_y)] \cdot \sin(y1)\} / J$$

Именно вычисление этих двух функций и должно происходить в процедуре правых частей. Назовем будущую процедуру **FM0**. Выходной переменной в ней будет вектор z = [z1 z2], а входными — момент времени t и вектор y = [y1 y2]. Некоторую сложность представляет то, что постоянные коэффициенты в правых частях нельзя передать в процедуру через ее заголовок. Поэтому объединим их в вектор коэффициентов k = [J,R,mgl, nmy,nmx, om,ey,ex] и отнесем этот вектор к категории глобальных:

```
global  K
```

Тогда M-файл будет иметь вид:

```
function z = FM0(t,y);
% Процедура правых частей уравнения Физического Маятника.
% Осуществляет расчет вектора z производных
% от вектора y переменных состояния по формулам:
% z(1)=y(2);
% z(2)=(-mgl*nmx*sin(om*t+ex)*cos(y(1))-R*y(2)-...
%  mgl*(1+nmy*sin(om*t+ey))*sin(y(1)))/J,
% Коэффициенты передаются в процедуру через глобальный
% вектор K=[J,R,mgl,nmy,nmx,om,ey,ex]
% Ю.Ф.Лазарев  5-10-1998 г.
global K
z(1) = y(2);
z(2) = (-K(3)*K(5)*sin(K(6)*t+K(8))*cos(y(1))-K(2)*y(2)...
        -K(3)*(1+K(4)*sin(K(6)*t+K(7)))*sin(y(1)))/K(1);
% Конец процедуры FM0
```

При использовании этой процедуры следует помнить, что в тексте программы предварительно должен быть объявлен глобальный вектор K при помощи служебного слова **global**, а затем определены все 8 его элементов.

Эту же процедуру можно несколько усложнить, группируя вместе вычисления всех внешних моментов сил, кроме момента сил тяжести, и оформляя их как отдельную процедуру. Для этого сначала преобразуем исходное уравнение, записывая его в виде:

$$\varphi'' + \sin\varphi = S(\tau,\varphi,\varphi')$$

где штрих — обозначение производной по безразмерному времени $\tau = \omega_0 \cdot t$

$$\omega_0 = \sqrt{\frac{mgl}{J}}$$

а через $S(\tau,\varphi,\varphi')$ обозначена некоторая заданная функция безразмерного времени, угла поворота маятника и его безразмерной скорости:

$$\varphi' = \frac{d\varphi}{d\tau}$$

В рассматриваемом случае эта функция приобретает такой вид:

$$S(t,\varphi,\varphi') = -2\cdot\zeta\cdot\varphi' - [n_{mx}\cdot\sin(\nu\cdot\tau+\varepsilon_x)\cdot\cos\varphi + n_{my}\cdot\sin(\nu\cdot\tau+\varepsilon_y)\cdot\sin\varphi]$$

причем безразмерные величины ζ и ν определяются выражениями:

$$\zeta = \frac{R}{2\cdot\sqrt{mgl\cdot J}}, \quad \nu = \frac{\omega}{\omega_0}$$

Такая безразмерная форма представления уравнений является предпочтительной и удобной, так как позволяет сократить число параметров (в нашем случае вместо трех размерных параметров J, R и mgl остался один — ζ), а также представлять решение уравнения в более общей форме.

Вынесение вычисления моментов внешних воздействий в отдельную вычислительную процедуру позволяет также сделать процедуру правых частей уравнения маятника более общей, если обращение к процедуре вычисления моментов осуществлять тоже через функцию **feval**.

Создадим процедуру **MomFm1**, которая будет вычислять моменты сил, действующих на маятник:

```
function m = MomFM1(t,y);
% Вычисление моментов сил, действующих на физический
% маятник.
% Осуществляет расчет момента m сил по формуле:
% m =-2*dz*y(2) - (nmx*sin(nu*t+ex)*cos(y(1)) +...
%     + nmy*sin(nu*t+ey)*sin(y(1)),
% Коэффициенты передаются в процедуру через глобальный
% вектор KM1=[dz,nmy,nmx,nu,ey,ex]
% Ю.Ф.Лазарев  5-10-1998 г.
global KM1
m = -2*KM1(1)*y(2)-(KM1(3)*sin(KM1(4)*t+KM1(6))...
     *cos(y(1))+ KM1(2)*sin(KM1(4)*t+KM1(5))*sin(y(1)));
% Конец процедуры MomFM1
```

Теперь следует перестроить процедуру правых частей. Назовем этот вариант **FM1**:

```
function z = FM1(mpfun,t,y);
% Процедура правых частей уравнения физического маятника.
% Осуществляет расчет вектора z производных
% вектора y переменных состояния по формулам:
% z(1)=y(2);
% z(2)= - sin(y(1)) +S(t,y), где
% входные параметры:
%   mpfun - имя процедуры S(t,y)
%   mpfun = 'S';
%   t - текущий момент времени;
%   y - текущее значение вектора переменных состояния;
% выходные параметры:
%   z - вектор значений производных от переменных состояния.
% Ю.Ф. Лазарев 5-10-1998 г.
%
z(1) = y(2);
z(2) = - sin(y(1)) + feval(mpfun,t,y);
% Конец процедуры FM1
```

Так как вид обращения к процедуре правых частей изменился (добавлена новая входная переменная — имя процедуры вычисления моментов), необходимо также перестроить процедуру численного метода. Назовем ее **RKO43m**:

```
function  [tout,yout] = rko43m(Zpfun,Mpfun,h,t,y)
% RKO43m Интегрирование ОДУ методом Рунге-Кутта
```

```
% 4-го порядка, правые части которых заданы
% процедурами Zpfun и Mpfun.
% Входные переменные:
%  Zpfun - строка символов, содержащая имя процедуры
%          вычисления правых частей ОДУ.
% Обращение: z = fun(Mpfun,t,y), где
% Zpfun = 'fun';
% Mpfun - строка с именем процедуры, к которой
%         обращается процедура fun;
% t     - текущий момент времени;
% y     - вектор текущих значений переменных состояния;
% z     - вычисленные значения производных z(i) = dy(i)/dt;
% h     - шаг интегрирования;
% t     - предыдущий момент времени;
% y     - предыдущее значение вектора переменных состояния.
% Выходные переменные:
% tout  - новый момент времени;
% yout  - вычисленное значение вектора переменных
%         состояния через шаг интегрирования
% Ю.Ф. Лазарев,  274-67-31.
% Copyright (c) 1998 by The PSON, NTU of Ukraine "KPI".
%
% Расчет промежуточных значений производных
k1 = feval(Zpfun, Mpfun, t, y);
k2 = feval(Zpfun, Mpfun, t+h/3, y+h/3*k1);
k3 = feval(Zpfun, Mpfun, t+2*h/3, y+h*k2-h/3*k1);
k4 = feval(Zpfun, Mpfun, t+h, y+h*(k3+k1-k2));
% Расчет новых значений вектора переменных состояния
tout = t + h;
yout = y + h*(k1 + 3*k2 + 3*k3 + k4)/8;
% Конец процедуры RKO43m
```

Такая форма представления процедуры вычисления правых частей дифференциальных уравнений неудобна, ибо, во-первых, процедуру вида **FM1** нельзя использовать при интегрировании процедурами MatLAB **ode23** и **ode45** (последние требуют, чтобы в процедуре правых частей было только два входных параметра, а в процедуре **FM1** их три), а во-вторых, такая форма вызывает необходимость создания новых M-файлов методов численного интегрирования, как это сделано только что.

Этого можно избежать, предствив имя дополнительной функции **Mpfun** как глобальную переменную. Тогда процедура правых частей может быть записана так:

```
function z = FM2(t,y);
% Процедура правых частей уравнения физического маятника.
% Осуществляет расчет вектора z производных
% вектора y переменных состояния по формулам:
% z(1)=y(2);
% (2)= - sin(y(1)) +S(t,y),где
% входные параметры:
% mpfun - имя процедуры S(t,y) - передается как глобальная
%         переменная;
% mpfun = 'S';
% t - текущий момент времени;
% y - текущее значение вектора переменных состояния;
% выходные параметры:
% z - вектор значений производных от переменных состояния.
% Ю.Ф. Лазарев  7-10-1998 г.
%
global MPFUN
z(1) = y(2);
z(2) = - sin(y(1)) + feval(MPFUN,t,y);
% Конец процедуры FM2
```

Теперь процедура **FM2** имеет только два входных параметра, передаваемых посредством заголовка, и может быть использована любой процедурой численного метода интегрирования, в том числе процедурами **ode23** и **ode45**. Следует помнить, что в основной программе переменной **MPFUN** надо присвоить некоторое символьное значение (имя функции, которая будет использована в процедуре правых частей), и она должна быть объявлена как глобальная. Например, если будет использована ранее созданная процедура **MomFun1**, в Script-файле должны присутствовать строки:

```
global MPFUN
MPFUN = 'MomFm1'
```

Ниже для справки приведены таблицы формул основных явных методов численного интегрирования дифференциальных уравнений.

Таблица 2.1. Методы Рунге-Кутта $y_{m+1} = y_m + h\,F(t_m;y_m)$

№ вар.	*Формула метода*	*Вспомогательные величины*	*Название метода*
1	$F=k_1$	$k_1=Z(t_m;y_m)$	Эйлера
2	$F=(k_1+k_2)/2$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h;y_m+hk_1)$	модифицированный Эйлера
3	$F=Z(t_m+h/2;$ $y_m+hk_1/2)$	$k_1=Z(t_m;y_m)$	
4	$F=(k_1+4k_2+k_3)/6$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/2;y_m+hk_1/2)$; $k_3=Z(t_m+h;y_m+h(2k_2-k_1))$	Хойне
5	$F=(k_1+3k_3)/4$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/3;y_m+hk_1/3)$; $k_3=Z(t_m+2h/3;y_m+2hk_2/3)$	
6	$F=(k_1+2k_2+2k_3+$ $+k_4)/6$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/2;y_m+hk_1/2)$; $k_3=Z(t_m+h/2;y_m+hk_2/2)$; $k_4=Z(t_m+h;y_m+hk_3)$	Рунге-Кутта

№ вар.	Формула метода	Вспомогательные величины	Название метода
7	$F=(k_1+3k_2+3k_3+$ $+k_4)/8$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/3;y_m+hk_1/3)$; $k_3=Z(t_m+2h/3;y_m+h(k_2-k_1/3))$; $k_4=Z(t_m+h;y_m+h(k_1-k_2+k_3))$	

Таблица 2.2. Многошаговые методы

№ вар.	Формула прогноза	Формула коррекции	Название метода
8	$y_{m+1}=y_{m-1}+2hZ(t_m;y_m)$	$y_{m+1}=y_m+h[Z(t_{m+1};y^*_{m+1})+$ $+Z(t_m;y_m)]/2$	
9	$y_{m+1}=y_m+h[Z(t_m;y_m)-$ $-Z(t_{m-1};y_{m-1})]/2$	$y_{m+1}=y_m+h[Z(t_{m+1};y^*_{m+1})+$ $+Z(t_m;y_m)]/2$	
10	$y_{m+1}=y_m+h[23Z(t_m;y_m)-$ $-16Z(t_{m-1};y_{m-1})+$ $+5Z(t_{m-2};y_{m-2})]/12$	$y_{m+1}=y_m+h[5Z(t_{m+1};y^*_{m+1})+$ $+8Z(t_m;y_m)-$ $-Z(t_{m-1};y_{m-1})]/12$	
11	$y_{m+1}=y_m+h[55Z(t_m;y_m)-$ $-59Z(t_{m-1};y_{m-1})+$ $+37Z(t_{m-2};y_{m-2})-$ $-9Z(t_{m-3};y_{m-3})]/24$	$y_{m+1}=y_m+h[9Z(t_{m+1};y^*_{m+1})+$ $+19Z(t_m;y_m)-$ $-5Z(t_{m-1};y_{m-1})+$ $+Z(t_{m-2};y_{m-2})]/24$	Адамса-Башфорта
12	$y_{m+1}=y_{m-3}+$ $+4h[2Z(t_m;y_m)-$ $-Z(t_{m-1};y_{m-1})+$ $+2Z(t_{m-2};y_{m-2})]/3$	$y_{m+1}=y_{m-1}+h[Z(t_{m+1};y^*_{m+1})+$ $+4Z(t_m;y_m)+$ $+Z(t_{m-1};y_{m-1})]/3$	Милна
13	$y_{m+1}=y_{m-3}+4h[2Z(t_m;y_m)-$ $-Z(t_{m-1};y_{m-1})+$ $+2Z(t_{m-2};y_{m-2})]/3$	$y_{m+1}=\{9y_m - y_{m-2} +$ $+3h[Z(t_{m+1};y^*_{m+1})+$ $+2Z(t_m;y_m)-$ $-Z(t_{m-1};y_{m-1})]\}/8$	Хемминга

2.7. Пример составления сложной программы

Ранее были рассмотрены основные препятствия, стоящие на пути создания сложных программ, и средства их преодоления. Теперь, учитывая это, попробуем составить и испытать в работе одну из довольно сложных комплексных программ.

2.7.1. Программа моделирования движения маятника

Постановка задачи. Пусть требуется создать программу, позволяющую моделировать движение физического маятника с вибрирующей точкой подвеса путем численного интегрирования дифференциального

уравнения этого движения. Дифференциальное уравнение движения маятника для этой задачи можно принять таким:

$$J\cdot\ddot{\varphi}+R\cdot\dot{\varphi}+mgl\cdot(1+n_{my}\cdot\sin(\omega\cdot t+\varepsilon_y))\cdot\sin\varphi=$$
$$=-mgl\cdot n_{mx}\cdot\sin(\omega\cdot t+\varepsilon_x)\cdot\cos\varphi$$

где J — момент инерции маятника; R — коэффициент демпфирования; mgl — опорный маятниковый момент маятника; n_{my} — амплитуда виброперегрузки точки подвеса маятника в вертикальном направлении; n_{mx} — амплитуда виброперегрузки в горизонтальном направлении; φ — угол отклонения маятника от вертикали; ω — частота колебаний точки подвеса; ε_x, ε_y — начальные фазы колебаний (по ускорениям) точки подвеса в горизонтальном и вертикальном направлениях.

Требуется создать такую программу, которая позволяла бы вычислять закон изменения угла отклонения маятника от вертикали во времени при произвольных, устанавливаемых пользователем значениях всех вышеуказанных параметров маятника и поступательного движения основания, а также при произвольных начальных условиях. Вычисления будем осуществлять путем численного интегрирования с помощью стандартной процедуры **ode45**.

Преобразование уравнения. Прежде всего для подготовки дифференциальных уравнений к численному интегрированию необходимо *привести эти уравнения к нормальной форме Коши*. Желательно также представить их в безразмерной форме. Для представленного уравнения это было проделано в предыдущем разделе.

Запись М-файла процедуры правых частей. Следующим шагом подготовки программы является создание и запись на диск процедуры вычисления правых частей полученной системы ДУ в форме Коши.

Эта процедура была создана в предыдущем разделе, и в окончательном варианте она имеет вид:

Файл FM2.m

```
function z = FM2(t,y);
% Процедура правых частей уравнения Физического Маятника.
% Осуществляет расчет вектора z производных вектора y
% переменных состояния по формулам:
%   z(1)=y(2);
%   z(2)=-sin(y(1)) +S(t,y),
% где
% входные параметры:
%   t - текущий момент времени;
%   y - текущее значение вектора переменных состояния;
%   MPFUN - имя процедуры S(t,y) — глобальная переменная
%           MPFUN = 'S';
% выходные параметры:
%   z - вектор значений производных от переменных состояния.
% Ю.Ф. Лазарев  7-10-1998 г.
%
global MPFUN
```

```
z(1) = y(2);
z(2) = - sin(y(1))+ feval(MPFUN,t,y);
z =z';
% Конец процедуры FM2
```

Примечание

Обратите внимание на незначительное, но существенное отличие приведенной процедуры от аналогичной процедуры предыдущего раздела — наличие в конце процедуры операции транспонирования вектора производных. Это обусловлено тем, что процедура **ode45** требует, чтобы вектор производных был обязательно столбцом.

В качестве дополнительной процедуры, используемой в **FM2**, выберем ранее созданную процедуру **MomFM1**, которую запишем в файл MomFM1.m.

Файл MomFM1.m

```
function m = MomFM1(t,y);
% Вычисление моментов сил, действующих на физический
% маятник.
% Осуществляет расчет момента m сил
% по формуле:
% m =-2*dz*y(2) - (nmx*sin(nu*t+ex)*cos(y(1)) +...
%    + nmy*sin(nu*t+ey)*sin(y(1)).
% Коэффициенты передаются в процедуру через глобальный
% вектор KM1=[dz,nmy,nmx,nu,ey,ex]
% Ю.Ф. Лазарев  5-10-1998 г.
global KM1
m = -2*KM1(1)*y(2)- KM1(3)*sin(KM1(4)*t+KM1(6))...
    *cos(y(1)) - KM1(2)*sin(KM1(4)*t+KM1(5))*sin(y(1));
% Конец процедуры MomFM1
```

Очевидно, что в вызывающем Script-файле надо предусмотреть объявление имени дополнительного файла MomFM1 как глобальной переменной MPFUN, а также обеспечить объявление глобальной переменной по имени KM1 и задание значений этого числового массива из пяти элементов.

Создание управляющего (главного) Script-файла. Главный файл создадим в соответствии с рекомендациями из разд. 2.4.5:

Файл FizMayatn2.m

```
% FizMayatn2
% Управляющая программа исследования движения
% физического маятника, установленного на поступательно
% вибрирующем основании
% Ю.Ф. Лазарев, кафедра ПСОН, КПИ, Украина, 7-10-1998 г.
%
FizMayatn2_Zastavka
k = menu('  Что делать ?  ',' Продолжить работу ','
          Закончить работу  ');
if k==1,
   while k==1
     FizMayatn2_Menu
     FizMayatn2_Yadro
     k =menu('  Что делать ?   ',' Продолжить работу  ',...
           '  Закончить работу   ');
   end
end
clear global
clear
  Конец FizMayatn2
```

Как видим, программа вызывает три дополнительных Script-файла — FizMayatn2_Zastavka, FizMayatn2_Menu и FizMayatn2_Yadro. Поэтому нужно создать еще эти три M-файла.

Создание Script-файла заставки. Как отмечалось, этот файл должен содержать операторы вывода на экран информации об основных особенностях математической модели, реализованной в программе, и ввода исходных значений параметров этой модели. Ниже приведен текст M-файла FizMayatn2_Zastavka.

Файл FizMayatn2_Zastavka.m

```
  FizMayatn2_Zastavka
  Часть (вывод заставки на экран)
  программы FizMayatn2
% Ю.Ф.Лазарев, кафедра ПСОН, КПИ, Украина, 24-09-1998 г.
%
  Ввод "вшитых" значений
sprogram = 'FizMaytn2.m';
sname ='Лазарев Ю.Ф.';
KM1 = [0 0 0 0 0, 0];
MPFUN = 'MomFm1';
global KM1 MPFUN
tfinal =2*pi*5;
fi0 =pi/180; fit0 = 0;
clc
disp(['Это программа, осуществляющая интегрирование уравнения ';...
' физического маятника при поступательной вибрации точки подвеса ';...
' в форме                                                        ';...
'      fi" + sin(fi) = - 2*dz*fi'' -                             ';...
'      - nmy*sin(nu*t+ey)*sin(fi) - nmx*sin(nu*t+ex)*cos(fi)     ';...
'где fi - угол отклонения маятника от вертикали,                 ';...
'     dz - относительный коэффициент затухания,                  ';...
'     nu - относительная частота вибрации точки подвеса,         ';...
'     nmy,nmx - амплитуды виброперегрузок в вертикальном         ';...
'               и горизонтальном направлениях соответственно,    ';...
'     ey,ex - начальные фазы колебаний в вертикальном            ';...
'             и горизонтальном направлениях соответственно,      ';...
'     KM1 = [dz,nmy,nmx,nu,ey,ex] - матрица коэффициентов        '])
  Конец FizMayatn2_Zastavka
```

В нем осуществляется присваивание исходных ("вшитых") значений всем параметрам заданного дифференциального уравнения, а также параметрам численного интегрирования — начальным условиям движения маятника и длительности процесса интегрирования. Часть этих параметров объединяется в единый глобальный вектор **KM1**. Одновременно переменной **MPFUN**, которая будет использоваться при интегрировании, присваивается значение 'MomFm1'.

Создание файла меню. Содержимое файла меню FizMayatn2_Menu приведено ниже.

Файл FizMayatn2_Menu.m

```
% FizMayatn2_Menu
% Фрагмент программы FizMayatn2 (диалоговое изменение данных)
% Ю.Ф. Лазарев, кафедра ПСОН, КПИ, Украина, 24-09-1998 г.
k=1;
while k<10
   disp(' ')
   disp(' Сейчас установлено ')
   disp([sprintf('Начальный угол (градусы) = %g',...
        fi0*180/pi),...
    sprintf('Начальная скорость              = %g', fit0)])
    disp(sprintf(' Число периодов               = %g', tfinal/2/pi))
   KM1   % KM1=[dz,nmy,nmx,nu,ey,ex]
   k = menu( 'Что изменять?   ', ...
   sprintf('Относительный к-нт затухания = %g', KM1(1)),...
   sprintf('Перегрузку (вертикаль)       = %g', KM1(2)),...
   sprintf('Перегрузку (горизонталь)     = %g', KM1(3)),...
   sprintf('Относительную частоту        = %g', KM1(4)),...
   sprintf('Фазу (вертикаль)             = %g', KM1(5)),...
   sprintf('Фазу (горизонталь)           = %g', KM1(6)),...
   sprintf('Начальный угол (градусы)     = %g', fi0*180/pi),...
   sprintf('Начальную скорость           = %g', fit0),...
   sprintf('Кол-во периодов              = %g', tfinal/2/pi),...
           'Ничего не изменять');
   disp(' ')
   if k<7,
  KM1(k)=input(['Сейчас KM1(',num2str(k),sprintf(') =  %g', KM1(k)),...
   '  Введите новое значение = ']);
   elseif k==7, fi0 = input([sprintf('Сейчас fi0 = %g градусов',...
  fi0*180/pi),...
  '  Введите новое значение = ']);
  fi0 = fi0*pi/180;
   elseif k==8, fit0 = input([sprintf('Сейчас fit0 = %g', fit0),...
   '   Введите новое значение = ']);
    elseif k==9,tfinal = input([sprintf('Сейчас кол-во периодов = %g',...
    tfinal/2/pi),...
  '   Введите новое значение = ']);
  tfinal = tfinal*2*pi;
 end
end    % FizMayatn2_Menu
```

Файл осуществляет организацию диалогового ввода-изменения значений параметров физического маятника, движения основания и параметров численного интегрирования в соответствии со схемой, описанной в разд. 2.4.4.

Создание файла-ядра программы. Основные действия по организации процесса численного интегрирования и выведению графиков сосредоточены в файле FizMayatn2_Yadro:

Файл FizMayatn2_Yadro.m

```
% FizMayatn2_Yadro
% Фрагмент программы FizMayatn2 (основные вычисления)
% Ю.Ф. Лазарев, кафедра ПСОН, КПИ, Украина,  7-10-1998 г.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. Подготовка начальных условий
%-----------------------------------
t = 0; tf = tfinal;
```



```
y0 =[fi0 fit0];
options = odeset('RelTol',1e-8,'AbsTol',[1e-10 1e-10]);
% ode45('rigidode',[0 12],[0 1 1],options);
%-----------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2. Организация цикла интегрирования
%-----------------------------------
[t,y] = ode45('FM2',[0 tf],y0,options);
%-----------------------------------
%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%
% 3. Вывод графиков
subplot(2,1,2);
plot(t/2/pi,y(:,1)*180/pi);grid;
title('Отклонение от вертикали','FontSize',14);
xlabel('Время (в периодах малых собственных колебаний)','FontSize',12);
ylabel('Угол в градусах','FontSize',12);
subplot(2,4,1:2);
plot(y(:,1)*180/pi,y(:,2));grid;
title('Фазовая траектория','FontSize',14);
xlabel('Угол в градусах','FontSize',12);
ylabel('Скорость','FontSize',12);
subplot(2,4,3:4);
axis('off');
h1=text(0,1.1,'Моделирование движения физического маятника',...
   'FontSize', 14, 'FontWeight', 'Bold');
h1=text(0.4, 1,'из уравнения','FontSize',12);
h1=text(0,0.9,'fi" + 2*dz*fi'' + [1+nmy*sin(nu*t+ey)]*sin(fi) =',...
   'FontSize',14);
h1=text(0.55,0.8,' = - nmx*sin(nu*t+ex)*cos(fi)','FontSize',14);
h1=text(0,0.7,'из таких значений параметров:','FontSize',12);
h1=text(0.45,0.6,sprintf('dz = %g',KM1(1)),'FontSize',12);
h1=text(0,0.5,sprintf('nmy = %g',KM1(2)),'FontSize',12);
h1=text(0.7,0.5,sprintf('nmx = %g',KM1(3)),'FontSize',12);
h1=text(0,0.4,sprintf('ey = %g',KM1(5)),'FontSize',12);
h1=text(0.7,0.4,sprintf('ex = %g',KM1(6)),'FontSize',12);
h1=text(0.45,0.3,sprintf('nu = %g',KM1(4)),'FontSize',12);
h1=text(0,0.2,'и начальных условий:','FontSize',12);
h1=text(0,0.1,[sprintf('fi(0) = %g',fi0*180/pi),' градусов'],'FontSize',12);
h1=text(0.7,0.1,sprintf('fi''(0) = %g',fit0),'FontSize',12);
h1=text(0,0.05,'----------------------------------------');
h1=text(0,-0.2,'----------------------------------------');
h1=text(-0.05,-0.05,['Программа  ',sprogram]);
h1=text(0.55,-0.05,'Автор - Лазарев Ю.Ф., каф. ПСОН');
h1=text(0,-0.15,['Выполнил  ',sname]);
tm=fix(clock);  Tv=tm(4:5);
h1=text(0.65,-0.15,[sprintf(' %g:',Tv),'     ',date]);
% Конец файла FizMayatn2_Yadro
```

Основные операции можно разделить на три главные группы: ввод начальных условий, организация цикла интегрирования и организация оформления графического окна вывода.

Отладка программы. Отладка программы состоит из запуска главного М-файла FizMayatn2, проверки правильности функционирования всех частей программы, внесения корректив в тексты используемых М-файлов до тех пор, пока все запрограммированные действия не будут удовлетворять заданным требованиям. Сюда же входят и действия по проверке "адекватности", т.е. соответствие получаемых программой результатов отдельным априорно известным случаям

поведения исследуемой системы. Очевидно, для такой проверки нужно подобрать несколько совокупностей значений параметров системы, при которых поведение системы является известным из предыдущих теоретических или экспериментальных исследований. Если полученные программой результаты полностью согласуются с известными, программа считается адекватной принятой математической модели.

В приведенном тексте программы "вшитые" начальные значения параметров соответствуют свободному движению маятника без влияния трения. При таких условиях движение маятника представляет собой незатухающие колебания относительно положения вертикали. Поэтому, если программа работает верно, на графиках должны наблюдаться именно такие колебания маятника.

Результат работы созданной программы при этих условиях представлен на рис. 2.11.

Как видно, в этом отношении программа является адекватной принятой математической модели.

Проведение исследований. Созданная программа теперь может быть использована для моделирования и исследования разнообразных нелинейных эффектов, которые наблюдаются у физического маятника при поступательной вибрации точки его подвеса. На рис. 2.12—2.17 продемонстрированы некоторые возможности созданной программы.

На рис. 2.12 приведены параметрические колебания маятника, которые могут возникать при вибрации точки подвеса в вертикальном направлении. Из рисунка видно, что в этом случае колебания маятника относительно вертикали сначала увеличиваются по амплитуде, а затем устанавливаются, причем частота установившихся колебаний вдвое меньше частоты вибрации основания и составляет примерно 1,15.

Выпрямительный эффект маятника приведен на рис. 2.13. В этом случае одновременная вибрация основания в вертикальном и горизонтальном направлениях приводит к отклонению среднего положения маятника от вертикали на угол примерно -5 градусов.

Рис. 2.14 иллюстрирует стационарные колебания маятника относительно верхнего положения равновесия, которые могут наблюдаться при интенсивной вертикальной вибрации. Эти колебания при наличии трения затухают, как показано на рис. 2.15, и маятник "застывает" в верхнем положении.

Наконец, рис. 2.16 и 2.17 демонстрируют возможность значительных отклонений среднего положения маятника от вертикали и при чисто горизонтальной вибрации основания. В соответствии с ними это отклонение достигает величины свыше 40 градусов при принятых значениях параметров вибрации, причем направление отклонения зависит от начальных условий движения маятника.

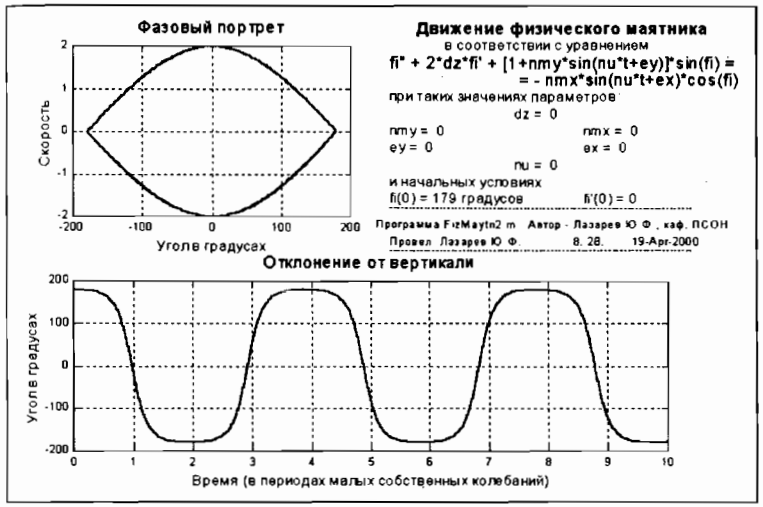

Рис. 2.11

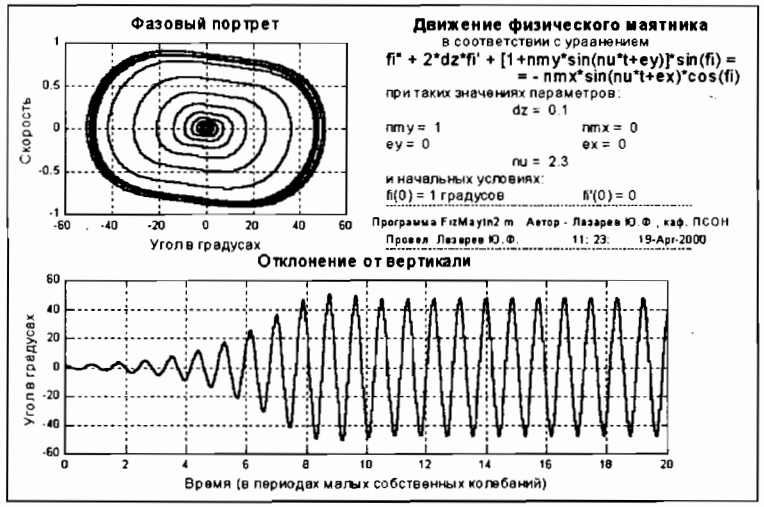

Рис. 2.12

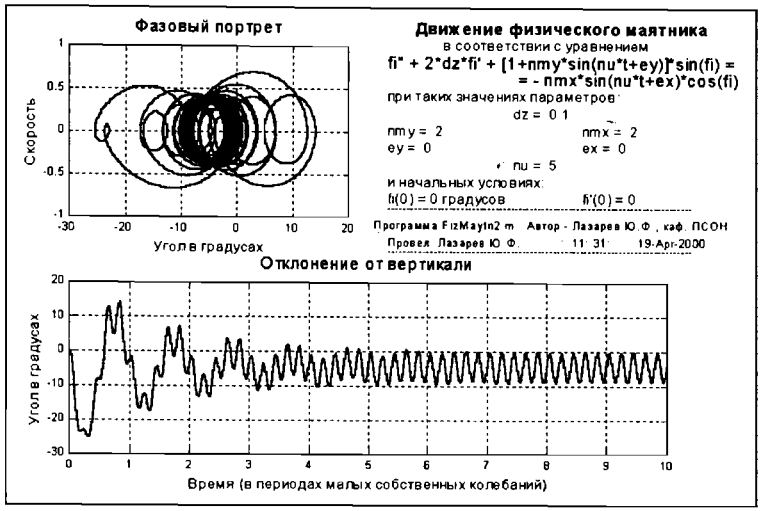

Рис. 2.13

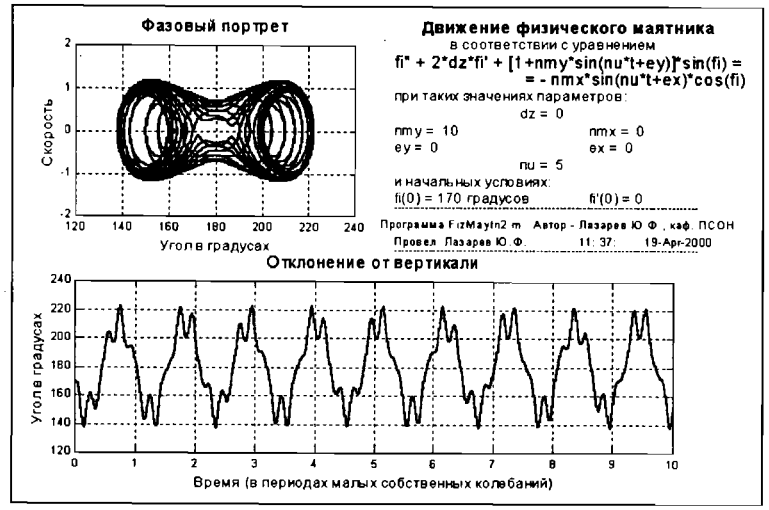

Рис. 2.14

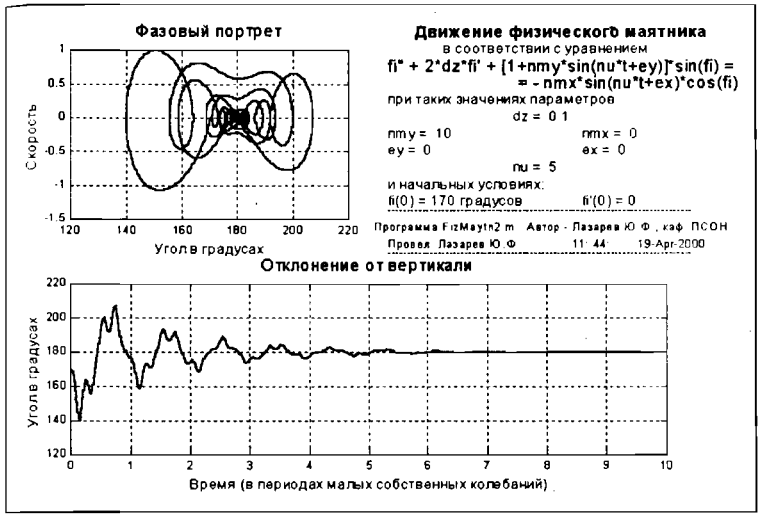

Рис. 2.15

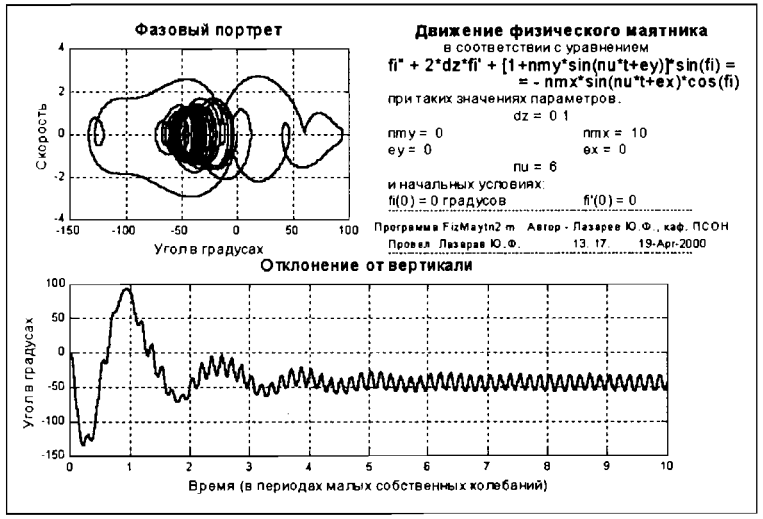

Рис. 2.16

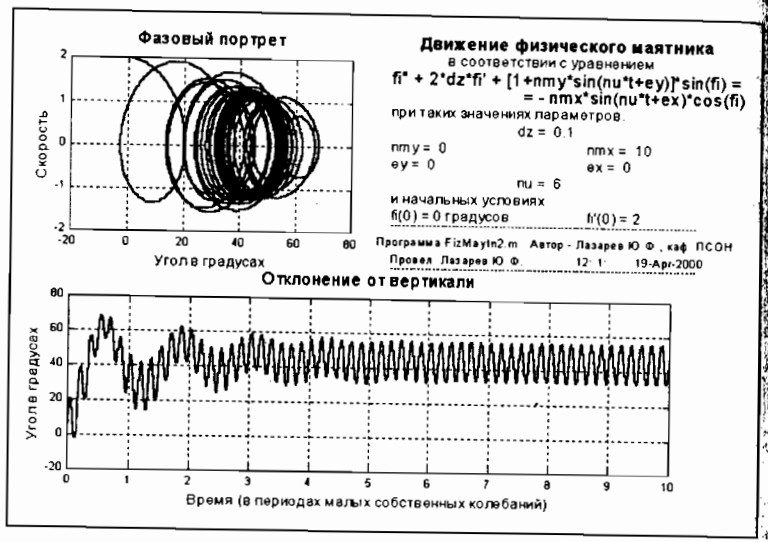

Рис. 2.17

Глава 3

Интерфейс MatLAB и команды общего назначения. М-книги

3.1. Командное окно MatLAB

Рассмотрим более подробно возможности, предоставляемые командным окном MatLAB. Напомним, что после запуска системы MatLAB на экране появляется командное окно, показанное на рис. 3.1.

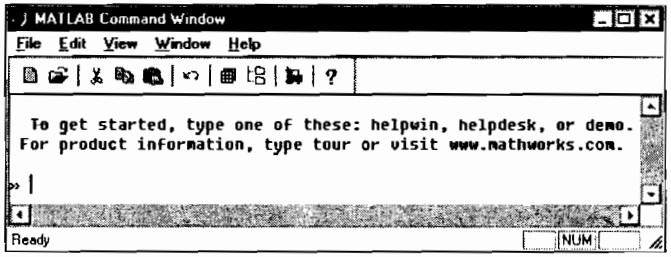

Рис. 3.1

В верхней его части (непосредственно под заголовком **MATLAB Command Window**) расположена строка меню, состоящая из четырех меню — **File**, **Edit**, **Window** и **Help**. Под главным меню размещена панель инструментов с пиктограммами, которые позволяют выполнять некоторые наиболее часто используемые операции.

Открытие меню осуществляется щелчком мыши. Чтобы выбрать какую-либо команду меню, достаточно установить курсор на имени команды и нажать левую кнопку мыши (выполнить щелчок).

3.1.1. Меню File

Меню **File** (Файл) включает команды, которые позволяют выполнять следующие задачи:

- создание, редактирование и запуск программ;
- управление рабочим пространством MatLAB;
- изменение оформления графических и диалоговых окон;
- управление путями доступа MatLAB и оформлением самого командного окна;
- управление выводом на печать;

- выход из системы MatLAB.

Команды разбиты на группы в соответствии со своим назначением (рис. 3.2). Последняя группа включает только одну команду — **Exit MATLAB** (Выход из MatLAB). Она предназначена для завершения работы MatLAB и выхода в среду Windows.

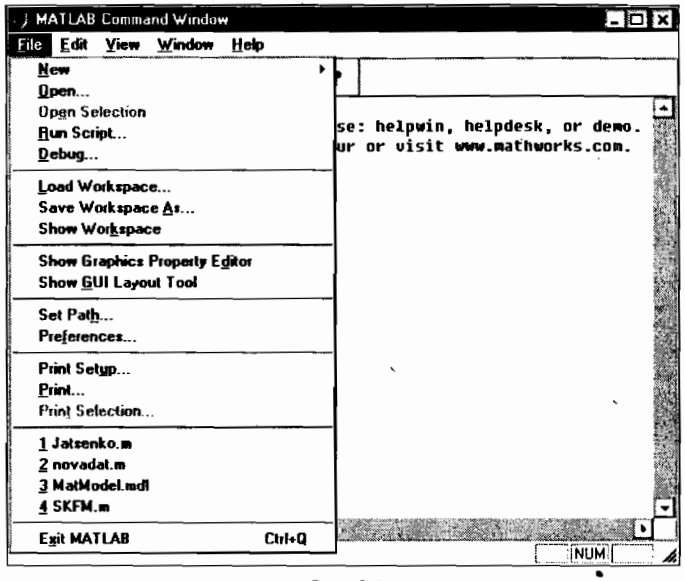

Рис. 3.2

В меню **File** также находится перечень последних вызванных М-файлов, который запоминается при завершении сеанса работы с MatLAB и автоматически отображается при новом запуске программы. Эта особенность удобна, так как позволяет быстро вызвать программу, с которой пользователь работал в последнее время.

Создание, редактирование и запуск программ

Первая группа команд меню **File** состоит из команд **New**, **Open**, **Open Selection**, **Run Script**, **Debug**.

Команда **New** (Новый) позволяет перейти к созданию нового объекта среды MatLAB — М-файла (текстового файла на языке MatLAB), графического файла ("фигуры") или файла-модели (Model).

После выбора команды **New** открывается подменю, включающее команды **M-file**, **Figure**, **Model** (рис. 3.3). Вызов команды **M-file** приведет к появлению нового активного окна — окна текстового редактора (рис. 3.4). Это окно предназначено для ввода текста нового М-файла. После завершения ввода и редактирования текста следует сохранить

его на диске, для чего в меню **Файл** (File) окна текстового редактора (рис. 3.4) следует выбрать команду **Сохранить** (Save).

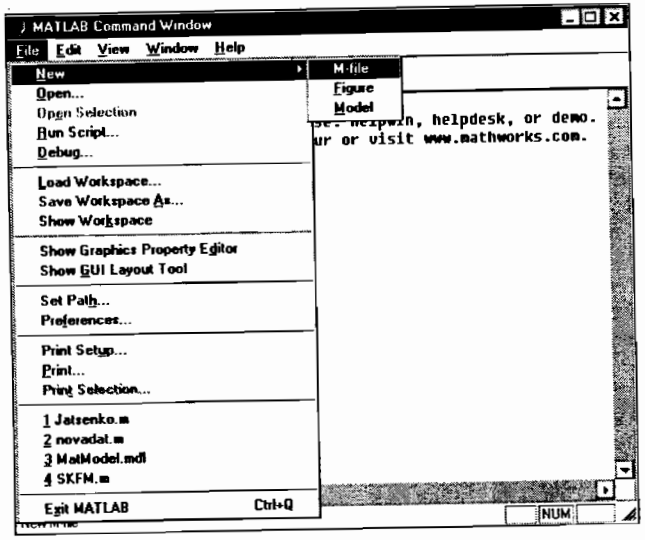

Рис. 3.3

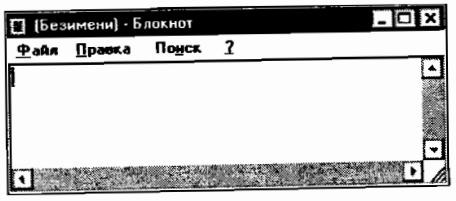

Рис. 3.4

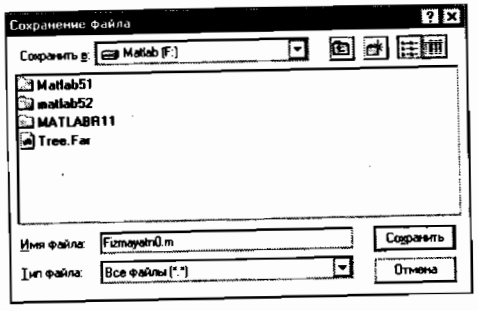

Рис. 3.5

После этого на экране появится диалоговое окно **Сохранение файла** (рис. 3.5), в котором представлена информация о текущей папке (каталоге) и содержащихся в ней файлах. Справа от поля, в котором указано имя текущей папки, размещены две кнопки. Первая кнопка позволяет перейти к более высокой по иерархии папке по сравнению с текущей, а вторая — создать новую папку в текущей. Чтобы открыть папку, которая содержится в текущей, следует выполнить на ее имени двойной щелчок мышью. Таким образом можно выбрать любую требуемую папку. В нижней части окна расположены поля, в которых необходимо задать имя файла и выбирать его тип. В поле для типа файла следует выбрать элемент **Все файлы (*.*)**. При вводе имени M-файла обязательно требуется указать расширение .m. После нажатия кнопки **Сохранить** (Save) файл под указанным именем будет записан в выбранную папку и диалоговое окно закроется. На экране снова появится окно текстового редактора (рис. 3.4). Для завершения работы с редактором нужно выбрать в меню **Файл** (File) его окна команду **Выход** (Exit).

Если в подменю команды **New** командного окна MatLAB выбрать команду **Figure**, на экране появится графическое окно **Figure** (рис. 3.6) и система будет готова к восприятию команд по оформлению этого графического окна.

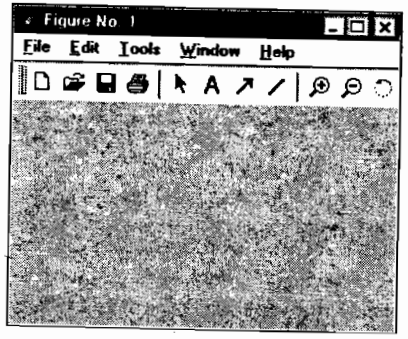

Рис. 3.6

Наконец, при выборе команды **Model** система MatLAB переходит в интерактивный режим пакета SIMULINK (Имитация связей), и на экране появляются два окна (рис. 3.7) — окно **Library: simulink**, позволяющее выбрать в интерактивном режиме любую функцию из представленной библиотеки пакета SIMULINK, и окно редактора блок-схемы связей, в верхнем заголовке которого указано имя файла схемы связей (сейчас указано Untitled — Без имени). Файл схемы связей должен иметь расширение .mdl. (Более подробно о создании и использовании MDL-файлов рассказано в главе 7.)

Команда **Open** (Открыть) открывает диалоговое окно с перечнем M-файлов текущей папки. Выбор необходимого файла из этого списка и последующее нажатие кнопки **OK** приводит к появлению окна предварительно установленного текстового редактора (например, Notepade — Блокнот) с текстом выбранного M-файла. Можно приступить к редактированию текста, затем записать его на диск.

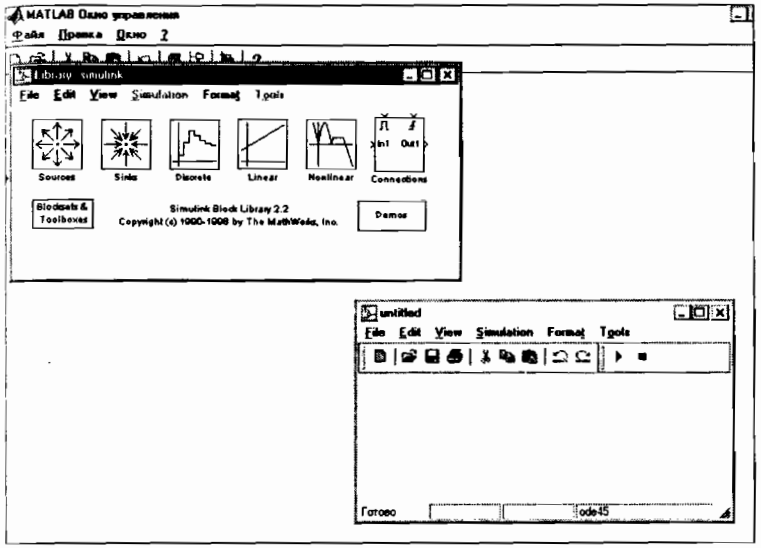

Рис. 3.7

Рассмотрим этот процесс подробнее. После выбора команды **Open** появляется окно **Открытие файла**, в котором представлено содержимое текущей папки (рис. 3.8). При первом обращении к программе текущей является папка MatLAB под названием work.

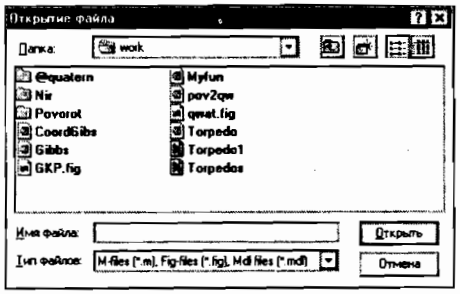

Рис. 3.8

Перейти к другой папке можно несколькими способами:

- нажать кнопку со стрелкой справа от списка **Папка** и в открывшемся списке выбрать (щелчком мыши) требуемые диск и папку;
- нажать первую кнопку справа от списка **Папка**, чтобы перейти к папке более высокого уровня.

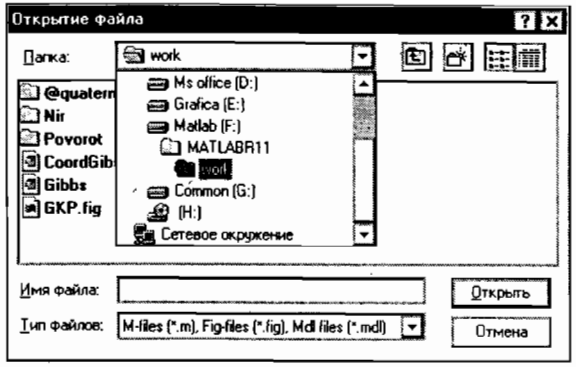

Рис. 3.9

Так, если, пользуясь первым способом, выбрать папку MATLABR11 (рис. 3.9), то получим результат, показанный на рис. 3.10. К тому же результату приведет нажатие первой кнопки в верхней части окна **Открытие файла** (в соответствии со вторым способом).

Вторая кнопка в верхней части этого окна позволяет создать новую папку внутри текущей.

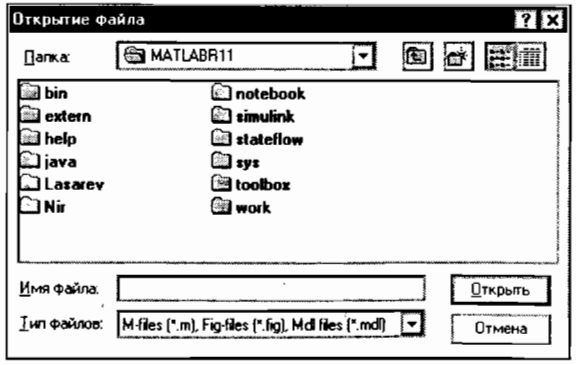

Рис. 3.10

Обратим теперь внимание на самую большую область этого окна, в которой представлен перечень файлов с расширениями .m и .mdl и папок, содержащихся в текущей папке (последние помечены пиктограммами в виде желтого прямоугольника).

Чтобы открыть какую-либо из этих папок, следует выполнить двойной щелчок мышью на ее пиктограмме. Например, если открыть папку toolbox, то диалоговое окно Открытие файла приобретет вид, изображенный на рис. 3.11. Теперь откроем папку matlab, находящуюся в папке toolbox. В результате окно **Открытие файла** станет таким, как на рис. 3.12.

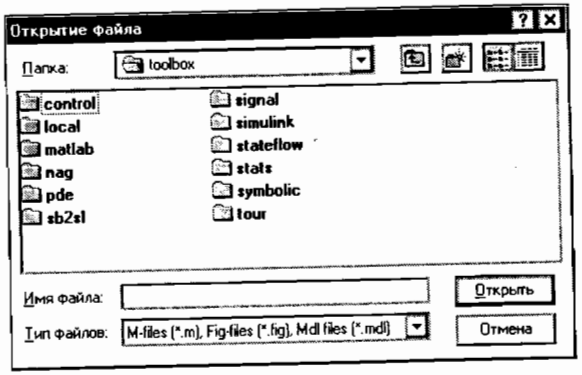

Рис. 3.11

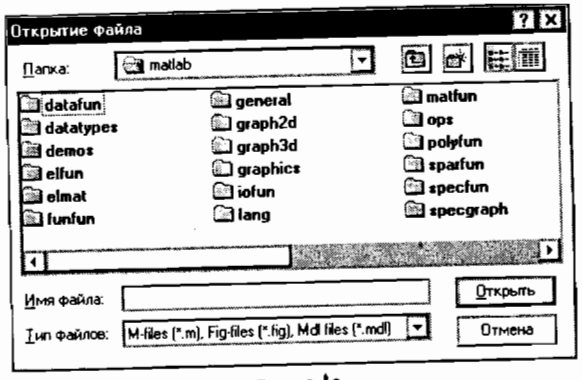

Рис. 3.12

И наконец, откроем папку datafun. Ее содержимое приведено на рис. 3.13. Как видно, эта папка содержит М-файлы.

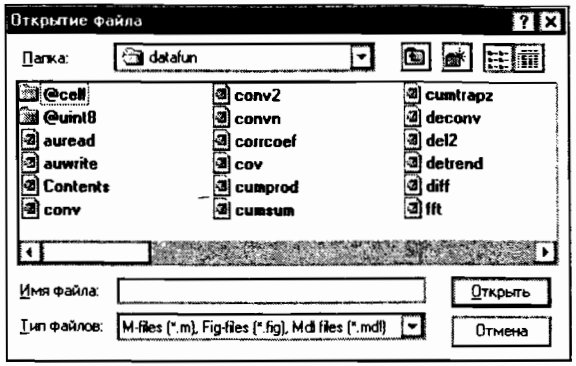

Рис. 3.13

Выберем файл max.m. После этого окно **Открытие файла** закроется, а вместо него появится окно текстового редактора (рис. 3.14) с текстом указанного M-файла.

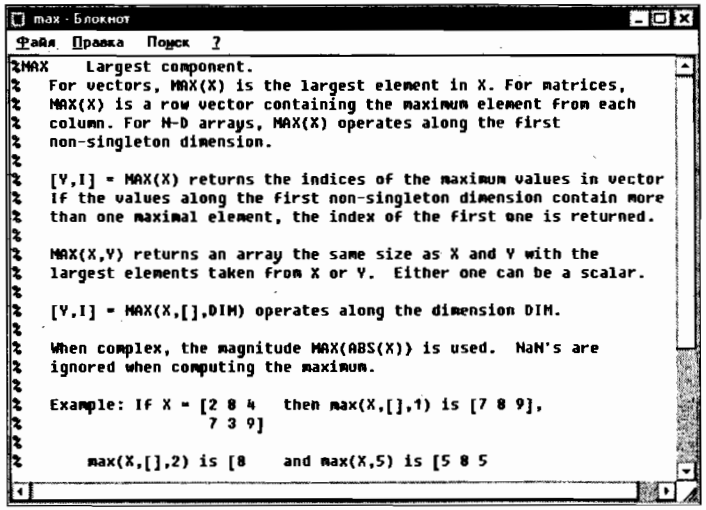

Рис. 3.14

Теперь этот файл можно откорректировать и сохранить на диске под тем же или другим именем.

Вызов из меню **File** команды **Run Script** приводит к появлению диалогового окна с приглашением ввести имя M-файла программы, которую нужно запустить на выполнение (рис. 3.15). После ввода имени Script-файла и нажатия кнопки **OK** эта программа будет запущена на выполнение.

Данную команду удобно использовать в случае, когда вызываемый файл не содержится в одном из каталогов, указанных в путях, открытых для системы MatLAB.

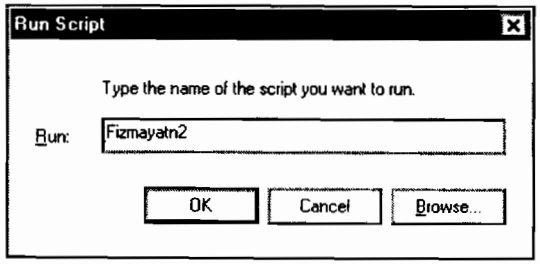

Рис. 3.15

Команда **Debug** из меню **File** позволяет вызвать для указанного M-файла окно редактора-отладчика. После выбора этой команды появляется окно **Debug**, которое по свойствам полностью аналогично ранее рассмотренному окну **Открытие файла** (рис. 3.16).

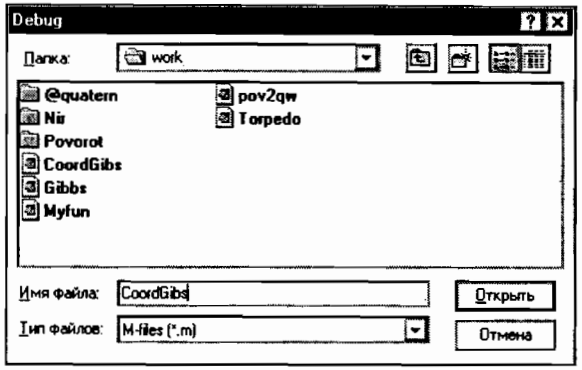

Рис. 3.16

Когда вы отыщете с помощью окна **Debug** нужные папку и файл и выберете этот файл при помощи мыши, откроется окно **MATLAB Editor/Debugger** (рис. 3.17). Это окно встроенного текстового редактора, позволяющее не только корректировать программу, но и проводить ее отладку. Заметим, что на рис. 3.17 вызван тот же файл max.m, что и ранее.

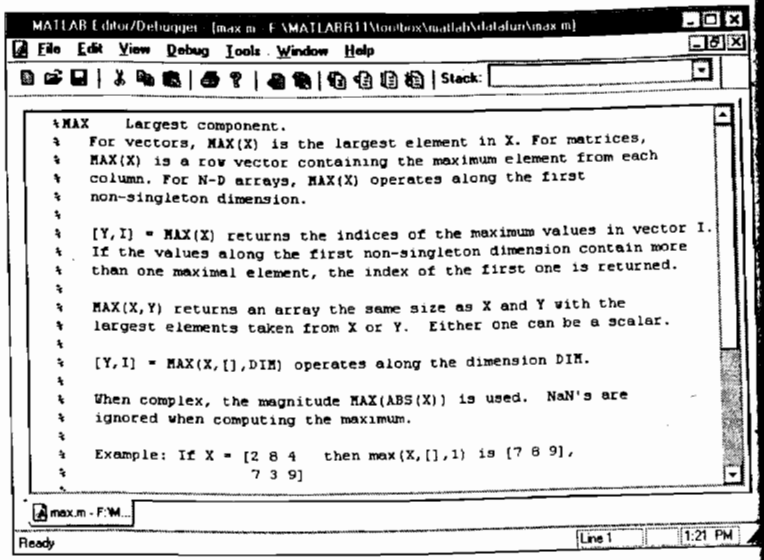

Рис. 3.17

Управление рабочим пространством MatLAB

Для управления рабочим пространством предназначена втора группа команд меню **File**, включающая команды **Load Workspace Save Workspace As** и **Show Workspace**.

Команда **Load Workspace** (Загрузить рабочее пространство) позво ляет использовать (загрузить в рабочее пространство системы) данны которые сохранены в виде так называемых MAT-файлов. В результат выбора команды появляется окно **Load .mat file** (рис. 3.18).

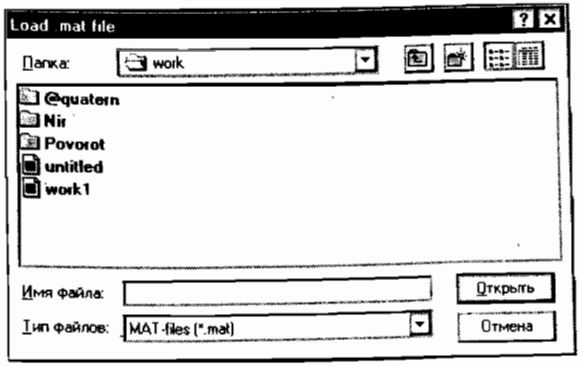

Рис. 3.18

После вызова MAT-файла из этого окна рабочее пространство MatLAB дополняется содержащимися в файле переменными и их значениями.

Команда **Save Workspace As** (Сохранить рабочее пространство как) производит операцию записи существующего рабочего пространства MatLAB на диск в виде MAT-файла. После выбора этой команды появляется окно **Save Workspace As** (рис. 3.19), посредством которого выбирается папка, задается имя MAT-файла и осуществляется запись существующего рабочего пространства в файл.

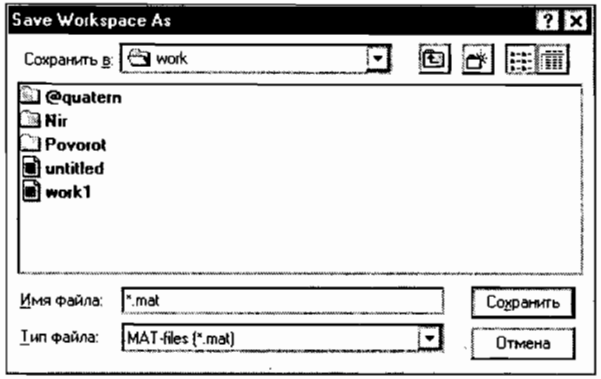

Рис. 3.19

Команда **Show Workspace** (Выбрать рабочее пространство) позволяет не указывать имя MAT-файла, загружаемого в рабочее пространство, а выбирать его в интерактивном (диалоговом) режиме. После вызова этой команды появляется окно **Workspace Browser** (рис. 3.20), в котором приведен полный перечень существующих MAT-файлов, созданных пользователем. Нужный MAT-файл выбирается и загружается при помощи мыши.

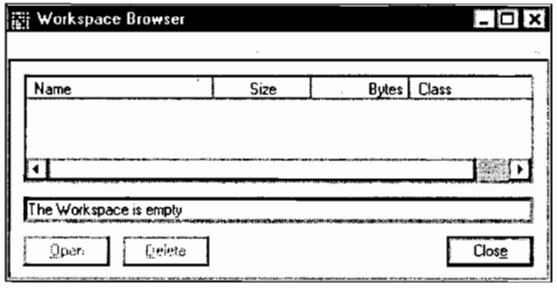

Рис. 3.20

Примечание

В поставляемой системе MatLAB MAT-файлы отсутствуют. Такие файлы создаются пользователем. В начале нового сеанса работы с системой рабочее пространство является пустым.

Изменение оформления графических и диалоговых окон

В меню **File** есть группа команд, вызывающих специальные интерактивные редакторы, которые позволяют изменять стиль оформления графических и диалоговых окон MatLAB.

Так, выбор команды **Show Graphics Property Editor** (Вызвать редактор графических свойств) приводит к появлению окна **Graphics Property Editor** (рис. 3.21), пользуясь которым можно изменить некоторые установленные ранее свойства, определяющие оформление графических окон.

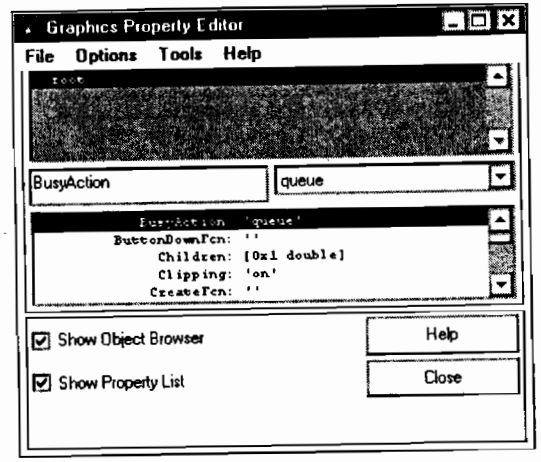

Рис. 3.21

Команда **Show GUI Layout Tool** (Вызвать средства оформления графического интерфейса пользователя) позволяет открыть окно **Guide Control Panel** (рис. 3.22), в котором можно изменять стиль графического оформления некоторых типовых элементов интерфейса пользователя системы MatLAB (в частности, меню пользователя, графических окон и некоторых других).

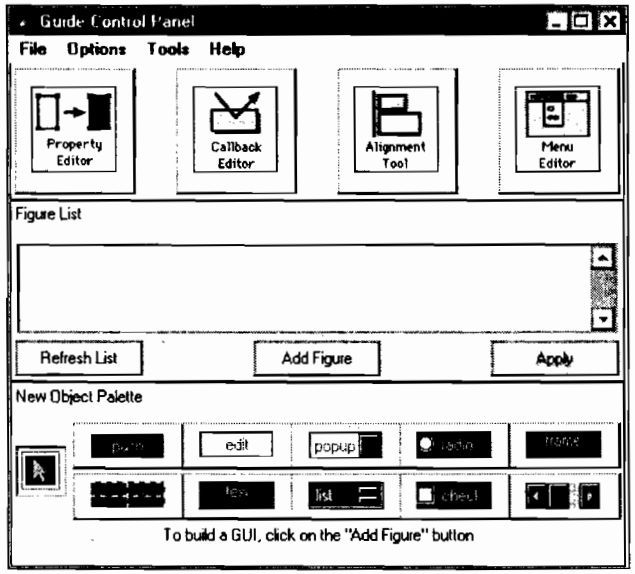

Рис. 3.22

Управление путями доступа MatLAB и оформлением командного окна

Для этой цели предусмотрено две команды — **Set Path** (Установить путь) и **Preferences** (Свойства).

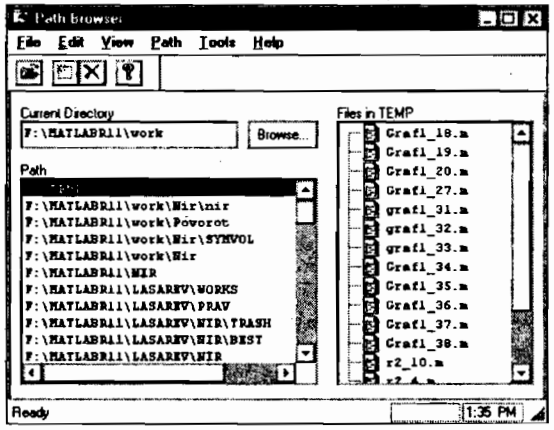

Рис. 3.23

Команда **Set Path** (Установить путь) предназначена для ввода в перечень путей доступа системы MatLAB, которые автоматически проверяются системой при поисках файлов, новых путей. При вызове этой команды появляется окно **Path Browser** (рис. 3.23), с помощью которого пользователь осуществляет изменение путей доступа системы по своему усмотрению.

Вызов команды **Preferences** (Свойства) приводит к открытию одноименного окна (рис. 3.24). Оно состоит из трех вкладок: **General**, **Command Window Font** и **Copying Options**.

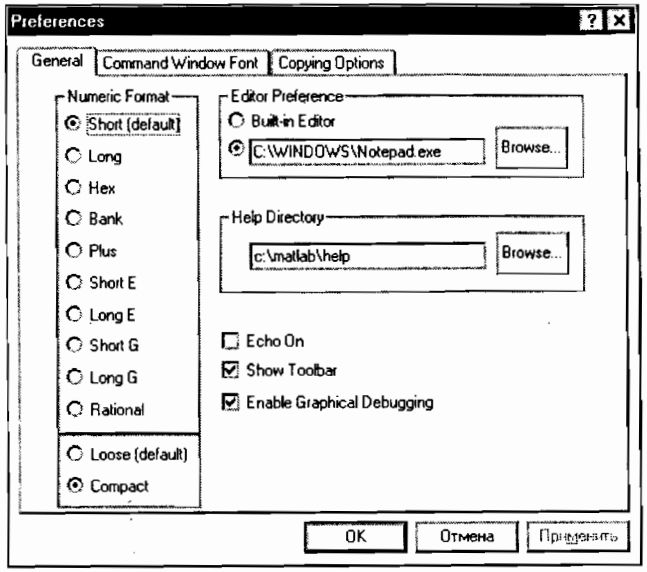

Рис. 3.24

Вкладка **General** (Общие) содержит несколько областей: **Numeric Format**, **Editor Preference**, **Help Directory**.

Область **Numeric Format** (Формат чисел) позволяет изменять формат представления чисел, которые выводятся в командное окно в процессе расчетов. Предусмотрены такие форматы:

Short (default)	Краткая запись (применяется по умолчанию)
Long	Длинная запись
Hex	Запись в виде шестнадцатеричного числа
Bank	Запись до сотых долей
Plus	Записывается лишь знак числа
Short E	Краткая запись в формате с плавающей запятой
Long E	Длинная запись в формате с плавающей запятой
Short G	Вторая форма краткой записи в формате с плавающей запятой
Long G	Вторая форма длинной записи в формате с плавающею запятой
Rational	Запись в виде рациональной дроби

Выбирая нужный формат представления чисел, можно обеспечить в последующем вывод чисел в командное окно именно в этой форме.

Кроме опций, соответствующих отмеченным форматам представления чисел, в этой области находятся еще опции **Loose (default)** и **Compact**, которые определяют форму вывода результатов в командное окно. По умолчанию применяется режим Loose, при котором отдельные строки выводятся на экран через одну пустую строку. Установка режима Compact позволяет выводить информацию в командное окно более плотно, без пропуска строк.

Область **Editor Preference** (Параметры редактора) позволяет выбрать текстовый редактор, используемый для представления и изменения М-файлов. Система MatLAB, начиная с версии 5.0, имеет собственный встроенный текстовый редактор MATLAB Editor/Debugger с отладчиком. Его окно показано на рис. 3.17. В области **Editor Preference** его можно заменить любым другим текстовым редактором. Удобным и простым при использовании кириллицы является редактор Notepad.

В области **Help Directory** можно изменить имя директории, в которой размещены файлы справок (Help-файлы).

Кроме перечисленных, в окне **Preferences** есть еще опции **Echo On** (Включить эхо-печать), **Show Toolbar** (Показать панель инструментов) и **Enable Graphical Debugging** (Включить графический отладчик). Чтобы активизировать опцию, следует выполнить на ней щелчок. После этого рядом с ее надписью появится пометка в виде галочки. Если галочка стоит рядом с опцией **Echo On**, то при выполнении текстового М-файла одновременно с выполнением программы ее текст будет постепенно выводиться в командное окно (так называемая эхо-печать). С помощью опции **Show Toolbar** можно управлять отображением панели инструментов (рис. 3.1). Пометка рядом с командой **Enable Graphical Debugging** означает, что выполнение графических операций будет сопровождаться их отладкой при помощи специального отладчика. Если же соответствующие пометки отсутствуют, то указанные действия не производятся.

На вкладке **Command Window Font** (Шрифт командного окна) (рис. 3.25) можно выбрать следующие параметры:

- шрифт, который будет использоваться для текста в командном окне (список **Font**);

- размер шрифта (список **Size**);
- стиль шрифта (для отдельных шрифтов) — список **Style**; возможны три вида стилей: Ligth (Тонкий), Regular (Обычный) и Bold (Жирный);
- цвет фона командного окна (список **Background Color**);
- цвет символов текста (список **Color**).

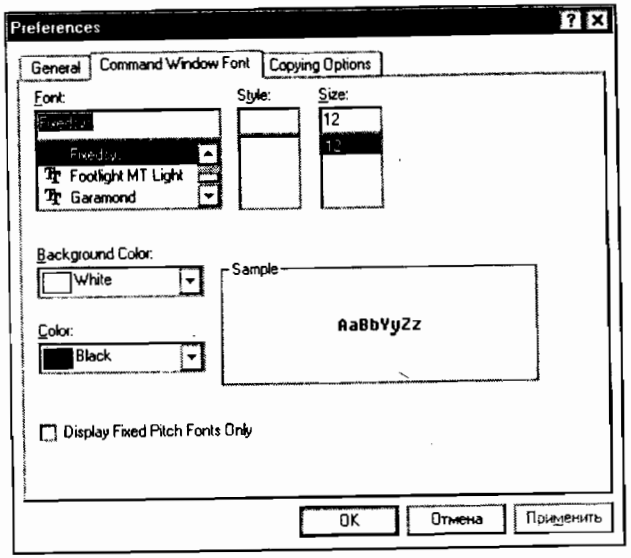

Рис. 3.25

Управление выводом на печать

Рассмотрим еще одну группу команд меню **File**. Это команды **Print Setup**, **Print** и **Print Selection**, управляющие выводом на печать.

После выбора команды **Print Setup** появляется окно **Настройка принтера** (рис. 3.26), в котором можно выбрать тип принтера, размер и способ подачи бумаги, а также ориентацию выводимой на печать страницы (книжная или альбомная).

Нажав кнопку **Свойства** в этом окне, можно перейти к окну **Свойства принтера** (рис. 3.27), в котором выполняются дополнительные настройки параметров принтера. Вид этого окна зависит от типа принтера.

Процесс печати активизируется при помощи команды **Print**. После ее вызова появляется диалоговое окно **Печать** (рис. 3.28). Это окно позволяет выбрать принтер, задать, какие страницы должны выводиться на печать, число копий и т.д.

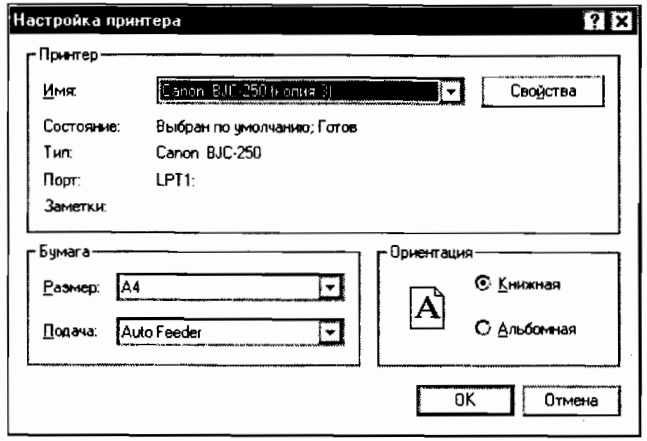

Рис. 3.26

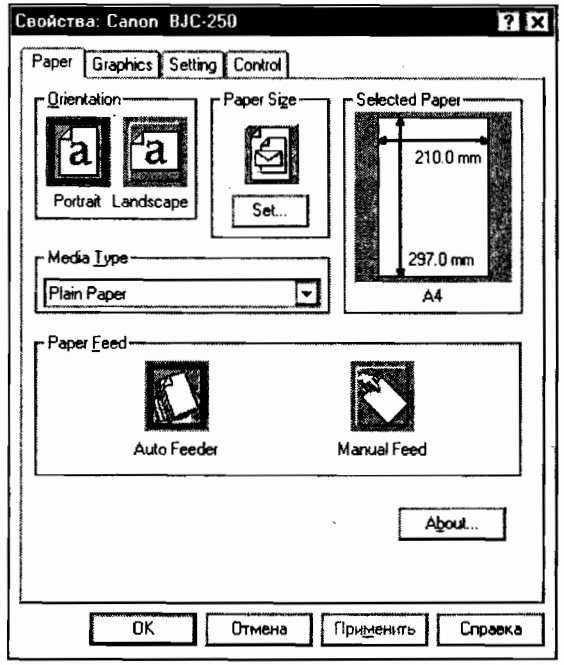

Рис. 3.27

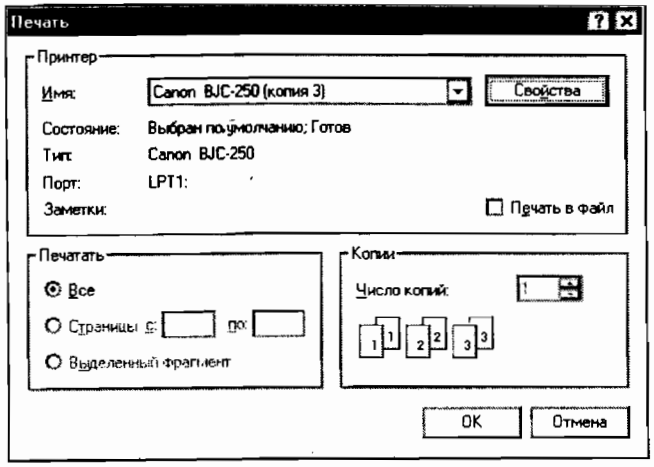

Рис. 3.28

3.1.2. Другие меню командного окна

Меню Edit

Меню **Edit** (Правка) содержит команды, позволяющие выполнять различные манипуляции с текстом — копирование, удаление, перемещение. Это меню (рис. 3.29) включает 7 команд:

- **Undo** (Отменить предыдущую команду);
- **Cut** (Вырезать);
- **Copy** (Скопировать);
- **Paste** (Вставить);
- **Clear** (Очистить);
- **Select All** (Отметить все);
- **Clear Session** (Очистить командное окно).

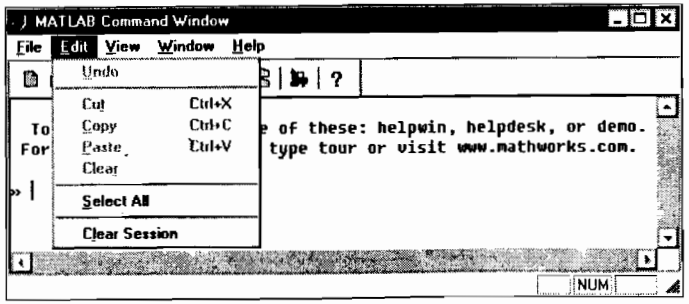

Рис. 3.29

В начале нового сеанса работы с MatLAB обычно можно воспользоваться только командой **Clear Session** из меню **Edit**, которая удаляет из командного окна MatLAB весь содержащийся в нем текст (очищает командное окно), оставляя в нем только знак готовности к восприятию новой команды (»).

Перед тем как производить какие-либо действия с текстом, его следует выделить. Чтобы выделить весь текст, можно воспользоваться командой **Select All** (Отметить все). Если вы хотите выделить только фрагмент текста, то установите указатель в начало этого фрагмента, нажмите левую кнопку мыши и, удерживая ее нажатой, переместите указатель в конец фрагмента. Выделенный текст отображается другим цветом.

Когда текст выделен, его можно переместить (команда **Cut** (Вырезать)) или скопировать (команда **Copy** (Скопировать)) в буфер. Отличие этих операций состоит в том, что при перемещении текст удаляется, а при копировании нет.

Текст из буфера можно скопировать в другое место этого же окна (команда **Paste** (Вставить)) или в другое Windows-приложение, воспользовавшись соответствующей командой. Таким образом можно копировать текст, например, из программы Word в окно MatLAB.

Меню Window

Здесь находится перечень открытых в среде MatLAB окон. Чтобы перейти к нужному окну, достаточно выбрать его из меню **Window**.

Меню Help

Данное меню (рис. 3.30) включает команды, которые позволяют обращаться к системе помощи MatLAB.

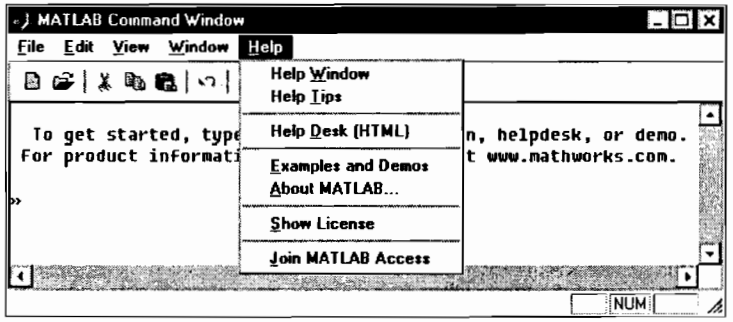

Рис. 3.30

Команда **Help Window** (Окно помощи) вызывает окно **MATLAB Help Window** (рис. 3.31), которое содержит список файлов-содержаний, включающих перечни названий процедур и функций. Выбор

одного из файлов-содержаний приводит к появлению в этом же окне перечня процедур, входящих в его состав, а выбор названия одной из процедур — к открытию окна с описанием этой процедуры (на английском языке).

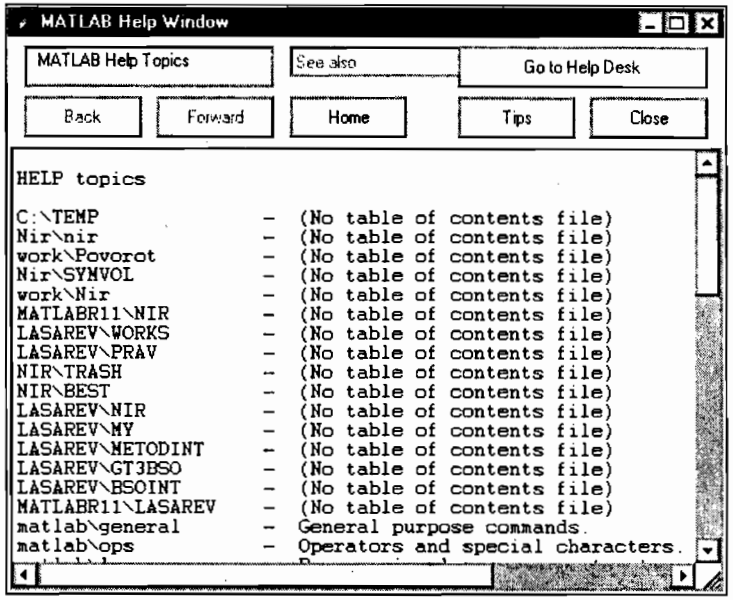

Рис. 3.31

Команда **Help Tips** (Темы помощи) также вызывает окно **MATLAB Help Window**, которое позволяет ознакомиться со всеми функциями и процедурами MatLAB общего назначения по темам, как в учебном пособии (рис. 3.32).

Выбор команды **Help Desk (HTML)** (Стол помощи) приводит к появлению окна **MATLAB Help Desk** (рис. 3.33), которое позволяет привлечь для справок сеть Internet.

Команда **Examples and Demos** (Примеры и демонстрационные программы) открывает окно **MATLAB Demo Window** (рис. 3.34), в котором можно ознакомиться с примерами применения основных процедур системы MatLAB, ее вычислительными и графическими возможностями, а также с текстами программ, при помощи которых достигаются демонстрируемые результаты.

Команда **About MATLAB** вызовет появление на экране логотипа системы MatLAB.

Наконец, благодаря команде **Join MATLAB Access** можно произвести подписку на пользование системой MatLAB в сети Internet. После выбора этой команды на экране появляется окно **Join MATLAB Access** (рис. 3.35).

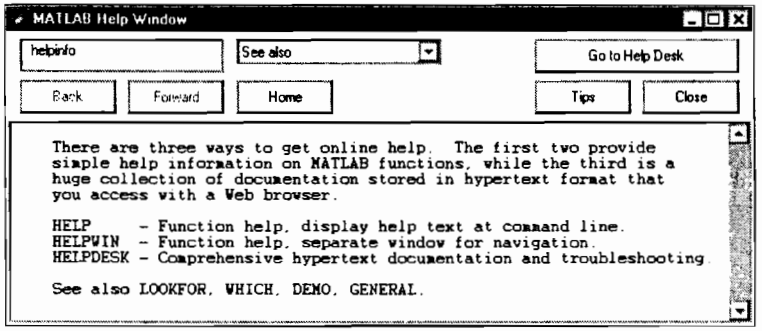

Рис. 3.32

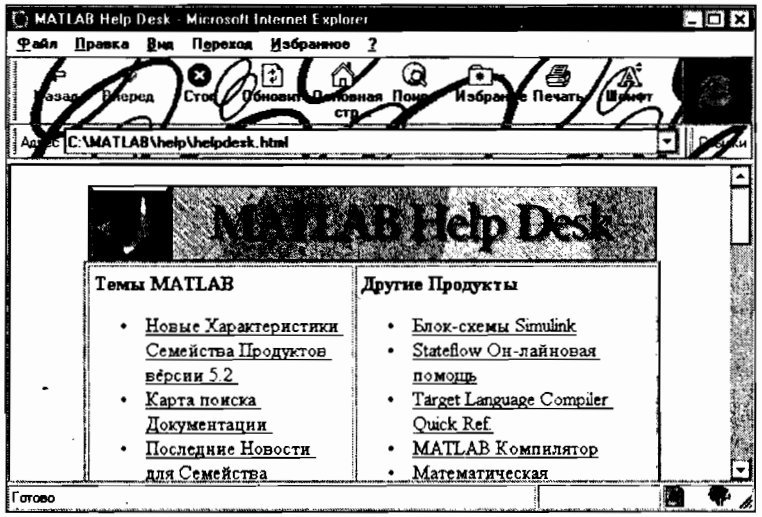

Рис. 3.33

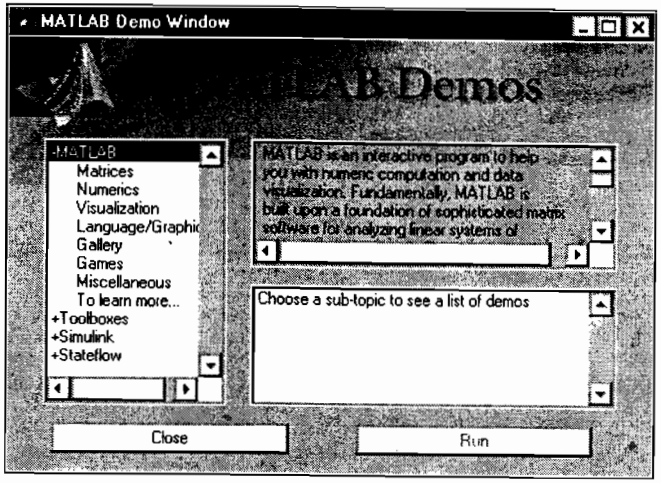

Рис. 3.34

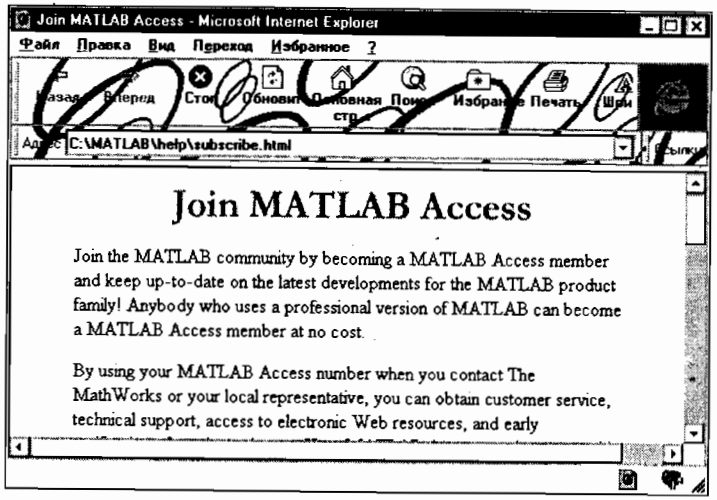

Рис. 3.35

3.1.3. Панель инструментов

Панель инструментов (рис. 3.1) содержит десять пиктограмм, дублирующих наиболее часто употребляемые команды главного меню командного окна. Назначение пиктограмм описано в приведенной ниже таблице.

Пиктограмма	*Назначение*
	Вызывает пустое окно установленного текстового редактора системы; ее действие эквивалентно выбору команды **New M-file** (Создать новый M-файл)
	Эквивалентна команде **Open M-file** (Открыть M-файл)
	Эквивалентна команде **Cut** (Вырезать)
	Эквивалентна команде **Copy** (Скопировать)
	Эквивалентна команде **Paste** (Вставить)
	Эквивалентна команде **Undo** (Отменить)
	Эквивалентна команде **Show Workspace** (Выбрать рабочее пространство)
	Эквивалентна команде **Set Path** (Установить путь)
	Эквивалентна команде **New Model**; подготавливает систему к созданию новой Simulink-модели
	Эквивалентна команде **Help Window**

Если установить курсор на соответствующей пиктограмме, рядом с ней через некоторое время появится название команды, которую эта пиктограмма выполняет. Чтобы активизировать пиктограмму, следует установить на ней курсор и нажать левую кнопку мыши.

3.2. Команды общего назначения

Команды общего назначения набираются с клавиатуры после знака приглашения (»). Выполняются они после нажатия клавиши [Enter].

Эти команды удобно разделить на такие группы:

- управляющие команды и функции;
- команды управления переменными и рабочим пространством;
- команды работы с файлами и операционной системой;
- команды управления командным окном;
- команды запуска и выхода из MatLAB;
- команды получения общей информации.

Рассмотрим некоторые из этих команд и функций.

Управляющие команды и функции

help	Вывод на экран первых строк описания указанной программы или функции
what	Вывод на экран перечня имен M-, MAT- и MEX-файлов в текущем каталоге
type	Вывод на экран текста указанного M-файла
lookfor	Поиск программы (функции) по указанному ключевому слову
which	Вывод на экран полного пути местонахождения указанной функции или файла
demo	Запуск программы демонстрации возможностей MatLAB
path	Вывод на экран полного перечня путей поиска файлов MatLAB по умолчанию

Команды управления переменными и рабочим пространством

who	Вывод на экран перечня текущих переменных
whos	Расширенная форма перечня текущих переменных
load	Загрузка в рабочее пространство значений переменных из указанного файла на диске
save	Запись значений переменных рабочего пространства в указанный файл на диске
clear	Очистка памяти от переменных и функций
pack	Уплотнение памяти рабочего пространства
size	Определение размеров двумерного массива
length	Определение длины одномерного массива
disp	Вывод на экран матрицы или текста

Команды работы с файлами и операционной системой

cd	Заменить текущий каталог на указанный
dir	Вывести на экран листинг указанного каталога
delete	Уничтожить (стереть) указанный файл
getenv	Вывести значения параметров окружения (среды)
!	Выполнить как команду операционной системы (применяется после указания команды операционной системы)
unix	Выполнить как команду операционной системы и вывести результат
diary	Записать текст командного окна в дневник MatLAB

Команды управления командным окном

cedit	Установить командную строку редактора клавиш
clc	Очистить командное окно
home	Поместить курсор в начало страницы
format	Установить указанный формат вывода чисел на экран
echo	Установка или упразднение режима эхо-печати текста выполняемой программы
more	Установка режима постраничного вывода текста в командное окно

Команды запуска и выхода из MatLAB

quit	Выйти из MatLAB
startup	Запуск MatLAB через M-файл startup
matlabrc	Запуск главного стартового M-файла

Команды получения общей информации

info	Получение информации о MatLAB и фирме MathWorks, Inc.
subscribe	Подписка по Internet как пользователя MatLAB
whatsnew	Информация о новых возможностях, которые не вошли в документацию
ver	Информация о поставленной версии MatLAB, SIMULINK и TOOLBOX

3.3. M-книги

Очень полезным и привлекательным свойством системы MatLAB является возможность создания текстовых документов в среде редактора Word с одновременным проведением в нем вычислений при помощи системы MatLAB и фиксированием результатов вычислений (в том числе графиков) в тексте Word-документа. Благодаря этому можно создавать сложные научно-расчетные и инженерные текстовые документы непосредственно в редакторе Word.

Средством, позволяющим это сделать, является пакет NoteBook, входящий в систему MatLAB. Пакет связывается с редактором Word при помощи специального шаблона Word, который входит в систему MatLAB. Для создания M-книг необходимо, чтобы шаблон, имеющий имя M-book.dot, был предварительно подключен к редактору Word.

3.3.1. Начало новой M-книги

Чтобы приступить к написанию новой M-книги, нужно:

- запустить Word;
- выбрать в окне Word команду **New** (Создать) из меню **File** (Файл);
- в появившемся окне выбрать шаблон M-book.

В результате этих действий будет запущена система MatLAB и вид главного меню Word изменится — в нем появится новый пункт **Notebook** (рис. 3.36). Это и будет свидетельствовать о том, что к Word присоединена система MatLAB.

3.3.2. Создание М-книги

Создание М-книги начинается с набора текста, а также операторов и команд MatLAB. Ввод текста осуществляется по обычным правилам редактора Word.

Чтобы ввести и выполнить команду MatLAB, необходимо:

- ввести текст команды в виде отдельной строки;
- после ввода строки с командой не нажимать клавишу [Enter] (курсор должен находиться в конце строки);
- выбрать из меню **Notebook** (рис. 3.36) команду **Define Input Cell** (Определить как входную ячейку) или нажать комбинацию клавиш [Alt+D]; после этого вид строки с командой должен измениться — символы команды должны приобрести темно-зеленый цвет, а команда будет заключена в квадратные скобки темно-серого цвета;
- выбрать из меню **Notebook** команду **Evaluate Cell** (Вычислить ячейку) или нажать комбинацию клавиш [Ctrl+Enter]; результатом этих действий должно стать появление в тексте сразу после текста команды результатов ее выполнения системой MatLAB.

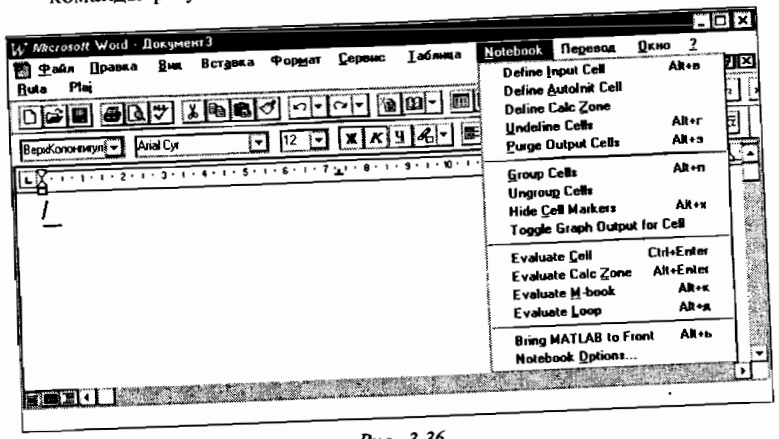

Рис. 3.36

Результаты выполнения команды выводятся синим цветом и заключаются в квадратные скобки.

Приведем пример. Пусть вы набрали в Word строку:

A = [1 2 3; 4 5 6; 7 8 9]

После нажатия комбинации клавиш [Alt+D] эта строка будет выглядеть так:

```
[ A = [1 2 3; 4 5 6; 7 8 9] ]
```


а после нажатия комбинации клавиш [Ctrl+Enter] в следующей строке появится результат:

```
[A =
     1     2     3
     4     5     6
     7     8     9  ]
```

Если требуется выполнить подряд несколько команд, введите их в отдельные строки по правилам написания текста программ, выделите эти строки, как это делается при копировании фрагмента текста в Word, и повторите описанные выше действия. Например:

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

Результатом будет появление трехмерного графика (рис. 3.37).

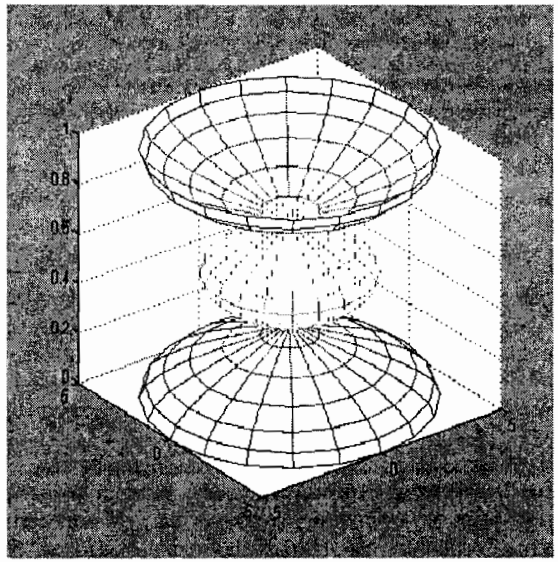

Рис. 3.37

Чтобы оставить в тексте документа введенные команды и выведенные результаты, нужно:

- поместить курсор мыши на одну из строк выполненной команды;
- выбрать из меню **Notebook** команду **Undefine Cells** или нажать комбинацию клавиш [Alt+U].

В результате все символы введенных команд и результатов их выполнения приобретут обычные для текста Word стиль, цвет и размеры, а квадратные скобки исчезнут.

3.3.3. Редактирование М-книги

Чтобы откорректировать существующую М-книгу или внести в нее какие-то дополнения, надо выполнить одно из следующих действий:

- вызвать редактор Word и открыть, используя команду **Open** (Открыть) из меню **File** (Файл), файл М-книги, который нужно корректировать;
- вызвать редактор Word и выбрать нужный файл с М-книгой из перечня последних документов в нижней части меню **File**;
- дважды щелкнуть мышью на документе М-книги в окне проводника Windows.

Редактор Word откроет документ, используя шаблон M-book, запустит систему MatLAB, если она не была перед этим активной, и добавит меню **Notebook** в главное меню Word.

3.3.4. Преобразование документа Word в М-книгу

Чтобы преобразовать Word-документ в М-книгу, необходимо сделать следующее:

- в редакторе Word создать новую (пока пустую) М-книгу;
- в меню **Insert** (Вставка) выбрать команду **File** (Файл);
- выбрать в появившемся окне **Вставка файла** файл, который нужно преобразовать в М-книгу, и нажать клавишу [Enter].

3.3.5. Некоторые особенности использования системы MatLAB

При написании М-книг следует учитывать некоторые особенности использования системы MatLab в среде редактора Word:

- можно пользоваться всеми возможностями системы MatLAB, доступными в режиме калькулятора (непосредственных вычислений);
- нельзя использовать Script-файлы, т.е. готовые М-программы, а также процедуры и функции, доступные только при работе со Script-файлами (например, процедуры создания меню и т.п.).

Последнее ограничение не является существенным. Его можно обойти, если воспользоваться командой **Bring MATLAB to Front** (Вывести MatLAB на передний план) меню **Notebook**.

В этом случае командное окно MatLAB переместится на передний план, и в нем уже можно осуществлять любые операции MatLAB. Естественно, результаты выполнения этих операций уже не будут автоматически записываться в текст М-книги. Они будут представлены, как обычно, в соответствующих окнах MatLAB. Однако, используя обычные операции переноса текста и графических изображений из одного окна в другое, можно их перенести в текст М-книги.

3.3.6. Изменение параметров вывода результатов

В меню **Notebook** есть команда **Notebook Options**, которая позволяет устанавливать некоторые параметры оформления результатов в М-книге по усмотрению пользователя. После активизации этой команды появляется окно, представленное на рис. 3.38. В данном окне можно устанавливать в интерактивном режиме:

- формат вывода чисел (область **Numeric Format**);
- более или менее плотный вывод строк (та же область, переключатели **Loose** и **Compact**);
- размеры выводимых в тест Word графических изображений (область **Figure Options**);
- выводить ли графические изображения, получаемые при работе MatLAB, в текст М-книги (опция **Embed Figure in M-book**);
- использовать ли при выводе графических изображений в М-книгу 16 цветов (опция **Use 16-Color Figures**); по умолчанию применяется 256 цветов.

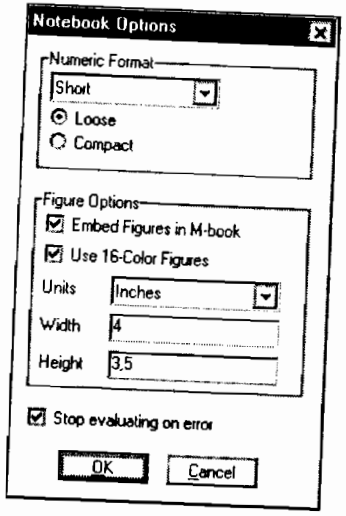

Рис. 3.38

Когда все установки произведены, надо нажать кнопку **OK** и в дальнейшем эти установки заработают.

В завершение отметим, что рукопись данной книги создана в виде М-книги.

Глава 4

Классы вычислительных объектов в MatLAB

Классом в MatLAB принято называть определенную форму представления вычислительных объектов в памяти компьютера в совокупности с правилами (процедурами) их преобразования. Класс определяет тип переменной, а правила — операции и функции, которые могут быть применены к этому типу. В свою очередь, тип определяет объем памяти, которая отводится на запись переменной в память, и структуру размещения данных в этом объеме. Операции и функции, которые могут быть применены к определенному типу переменных, образуют *методы* этого класса.

4.1. Основные классы объектов в MatLAB

В системе MatLAB определены 6 встроенных классов вычислительных объектов:

double	Числовые массивы и матрицы действительных или комплексных чисел с плавающей запятой в формате двойной точности
sparse	Двумерные комплексные разреженные матрицы
char	Массивы символов
struct	Массивы записей (структуры)
cell	Массивы ячеек
uint8	Массивы 8-разрядных целых чисел без знаков

Класс **double** определяет наиболее распространенный тип переменных в системе MatLAB, с которым оперирует большинство функций и процедур. Класс **char** определяет переменные, являющиеся совокупностью символов (каждый символ занимает 16 битов). Эту совокупность часто называют *строкой*. Класс **sparse** определяет тип переменных, являющихся разреженными матрицами двойной точности. Разреженная структура применяется для хранения матриц с незначительным количеством ненулевых элементов, что позволяет использовать лишь незначительную часть памяти, необходимой для хранения полной матрицы. Разреженные матрицы требуют применения специальных методов для решения задач. Переменные класса **cell** (ячейки) являются совокупностью некоторых других массивов. Массивы ячеек позволяют объединить связанные данные (возможно, разных типов и размеров) в единую структуру. Объекты класса **struct** состоят из нескольких составляющих, которые называются *полями*, каждое из которых имеет собственное имя. Поля сами могут содержать массивы. Подобно массивам ячеек, массивы записей объединяют связанные данные и информацию о них, однако способ обращения к элементам структуры (полям) принципиально иной — путем указывания имен полей через точку после имени структуры. Наконец, класс **uint8** позволяет хранить целые числа от 0 до 255 в 1/8 части памяти, необходимой для чисел двойной точности. Никакие математические операции для этого класса данных не определены.

Каждому типу данных соответствуют собственные функции и операторы обработки, т.е. *методы*. Приведем некоторые из них:

- класс **array** (обобщенный класс объектов-массивов, являющийся прародителем всех упомянутых встроенных классов) имеет такие методы: вычисление размеров (**size**), длины (**length**), размерности (**ndims**), объединение массивов (**[a b]**), транспонирование (**transpose**), многомерная индексация (**subindex**), переопределение (**reshape**) и перестановка (**permute**) измерений многомерного массива;
- методы класса **char** (строки символов) — строковые функции (**strcmp**, **lower**), автоматическое преобразование в тип **double**;
- методы класса **cell** — индексация с использованием фигурных скобок **{e1,...en}** и разделением элементов списка запятыми;
- методы класса **double** — поиск (**find**), обработка комплексных чисел (**real**, **imag**), формирование векторов, выделение строк, столбцов, подблоков массива, расширение скаляра, арифметические и логические операции, математические функции, функции от матриц;
- методы класса **struct** — доступ к содержимому поля (.field) (разделитель элементов списка — запятая);
- в классе **uint8** — единственный метод — операция сохранения (чаще всего применяется с пакетом Image Processing Toolbox).

4.1.1. Класс символьных строк (char)

Ввод строк символов с клавиатуры осуществляется в апострофах. Например, вводя совокупность символов:

```
» 'Это'
```

получим в командном окне:

```
ans =
Это
```

Аналогично, при помощи знака присваивания, производится определение переменных типа **char**:

```
» st1 = 'Это '; st2 = 'строка '; st3 = 'символов.';
» st1,st2,st3
```

```
st1 =
Это
st2 =
строка
st3 =
символов.
```

Объединение нескольких строк в единую строку (*сцепление*, или *конкатенацию*) можно осуществить при помощи обычной операции объединения векторов в строку:

```
» [st1 st2 st3]
ans =
Это строка символов.
```

Другая возможность достичь той же цели — использование процедуры **strcat(s1,s2,...sn)**, которая производит сцепление заданных строк s1, s2, ... sn в единую строку в порядке их указания в списке аргументов:

```
» st = strcat(st1,st2,st3)
st =
Это строка символов.
```

Объединить строки символов в несколько отдельных, но соединенных в единую конструкцию строк можно, используя другую процедуру — **strvcat** (вертикальной конкатенации):

```
» stv = strvcat(st1,st2,st3)
stv =
Это
строка
символов.
```

Примечание

Для такого сцепления символьных строк нельзя применять операцию вертикального сцепления (символ ";"), используемую для построения матрицы из отдельных строк, так как число элементов (символов) сцепляемых символьных строк может быть различным.

```
» [st1; st2; st3]
[]??? All rows in the bracketed expression
must have the same number of columns.
```

Символьная строка представляет собой массив (точнее, вектор-строку), элементами которого являются отдельные символы, из которых она состоит, включая символы пробелов. Поэтому информацию о любом символе в строке можно получить, указав номер этого символа от начала строки (при этом, конечно, надо учитывать и символы пробелов).


```
» st(3)
ans =
т
» st(3:12)
ans =
то строка
```

Совокупность вертикально сцепленных строк образует двумерный массив (матрицу) символов. Поэтому команда:

```
» ss =stv(2,3:end)
```

приводит к результату:

```
ss =
трока
```

Процедура **strrep(s1, s2, s3)** формирует строку из строки s1 путем замены всех ее фрагментов, совпадающих со строкой s2, на строку s3:

```
» st = [st1 st2 st3]
st =
Это строка символов.
» y = strrep(st,'o','a')
y =
Эта страка симвалав.
» x = strrep(st,'a','o')
x =
Это строко символов.
```

Функция **upper(st)** переводит все символы строки st в верхний регистр:

```
» x1 = upper(st)
x1 =
ЭТО СТРОКА СИМВОЛОВ.
```

Аналогично функция **lower(st)** переводит все символы в нижний регистр:

```
» x2 = lower(x1)
x2 =
это строка символов.
```

Процедура **findstr(st,st1)** выдает номер элемента строки st, с которого начинается первое вхождение строки st1, если она есть в строке st.

```
» findstr(st,'рок')
ans =    9
```

Как ранее отмечалось, довольно часто возникает необходимость вставить в строку символов числовое значение одного или нескольких

рассчитанных параметров, что связано с переводом числовой переменной в строку символов определенного вида. Это можно сделать при помощи процедуры **num2str**. Входным аргументом этой процедуры является числовая переменная (класса **double**). Процедура формирует представление значения этой числовой переменной в виде символьной строки. Формат представления определяется установленным в программе форматом. В качестве примера рассмотрим формирование текстовой строки с включением значения переменной:

```
» x = pi;
» disp(['Значение переменной X равно ', num2str(x)])
Значение переменной X равно 3.1416
```

Аналогичным образом при помощи процедуры **mat2str(A)** можно получить значение матрицы A в виде символьной строки:

```
» A = [1,2,3; 4 5 6; 7 8 9];
» disp(mat2str(A))
[1 2 3;4 5 6;7 8 9]
```

Обратный переход от символьного представления числа к численному осуществляется процедурой **str2num**.

```
» stx = num2str(x)
stx =
  3.1416
» y = str2num(stx)
y =
  3.1416
» y+5
ans =
  8.1416
» z = stx+5
z =
  56  51  54  57  54  59
```

Последний результат получен вследствие того, что строка символов stx при включении ее в арифметическую операцию автоматически перестраивается в класс **double**, т.е. все символы, составляющие ее, заменяются целыми числами, равными коду соответствующего символа. После этого сложение полученного числового вектора с числом 5 происходит по обычным правилам, т.е. 5 суммируется с каждым из кодов символов.

Проверим это, учитывая, что с помощью процедуры **double(str)** можно получить числовое представление строки символов str в виде кодов составляющих ее символов:

```
» double(stx)
ans =
  51  46  49  52  49  54
```


Сравнивая полученный результат с предыдущим, можно убедиться в справедливости сказанного.

Функция **str2mat(st1,st2,...,stn)** действует аналогично функции **strvcat(st1,st2,...,stn)**, т.е. создает символьную матрицу, располагая строки st1, st2, ..., stn одна под другой:

```
» Z = str2mat(st1,st2,st3)
Z =
 Это
 строка
 символов.
```

4.1.2. Класс записей (struct)

Массивы записей — это тип массивов в системе MatLAB, в котором позволяется сосредоточивать в виде записей разнородные данные (т.е. данные разных классов). Отличительной особенностью таких массивов является наличие именованных полей.

Как и вообще в MatLAB, массивы записей не объявляются. Отдельные экземпляры этого класса создаются автоматически при задании конкретных значений полям записи.

Обращение к любому полю по имени field осуществляется так:

```
<имя переменной-записи>.field
```

Например, команда

```
» PG81.fam = 'Аврутова'
PG81 =
  fam: 'Аврутова'
```

приводит к автоматическому формированию переменной **PG81** типа **struct** с единственным полем fam, значение которого — символьная строка "Аврутова". Таким же образом к этой переменной можно добавлять другие поля:

```
» PG81.imya = 'Марина'; PG81.bat = 'Степановна';
» PG81
PG81 =
  fam: 'Аврутова'
  imya: 'Марина'
  bat: 'Степановна'
```

В результате получим ту же переменную-запись с тремя полями. Чтобы создать массив аналогичных переменных с теми же полями и с тем же именем PG81 достаточно добавлять при обращении к этому имени номер записи (в скобках, как к элементу массива):

```
» PG81(2).fam = 'Березняк';
» PG81(2).imya = 'Алексей'; PG81(2).bat = 'Иванович';
» PG81(3).fam = 'Попель';
» PG81(3).imya = 'Богдан'; PG81(3).bat = 'Тимофеевич';
```

```
» PG81
PG81 =
1x3 struct array with fields:
  fam
  imya
  bat
```

Как видим, в случае массива записей, содержимое полей уже не выводится на экран. Выводится лишь информация о структуре массива, его размерах и именах полей.

Для получения информации об именах полей записи можно использовать функцию **fieldnames**:

```
» fieldnames(PG81)
ans =
  'fam'
  'imya'
  'bat'
```

Другой способ задания переменной-записи — применение функции **struct** по схеме

```
<имя переменной-записи> = struct('<имя поля1>',
            <значение1>, '<имя поля2>', <значение2>, ...)
```

Например, команда

```
» PG72=
struct('fam','Сергеев','imya','Сергей','bat','Сергеевич',
       'god', 1981)
```

приведет к формированию такой переменной-записи:

```
PG72 =
    fam: 'Сергеев'
    imya: 'Сергей'
    bat: 'Сергеевич'
    god: 1981
```

Используя индексацию, легко определить значение любого поля или элемента структуры. Таким же образом можно присвоить значение любому полю или элементу структуры.

Если к какому-либо из элементов массива записей добавляется значение нового поля, то же поле автоматически появится во всех остальных элементах, но его значение при этом остается пустым. Например:

```
» PG81.fam = 'Аврутова' ;
» PG81.imya = 'Марина'; PG81.bat = 'Степановна';
» PG81(2).fam = 'Березняк' ;
» PG81(2).imya = 'Алексей'; PG81(2).bat = 'Иванович';
» PG81(3).fam = 'Попель' ;
» PG81(3).imya = 'Богдан'; PG81(3).bat = 'Тимофеевич';
```



```
» PG81(3).god = 1982
PG81 =
1x3 struct array with fields:
  fam
  imya
  bat
  god
» PG81(2).god
ans =    []
```

Чтобы удалить некоторое поле из всех элементов массива записей, надо использовать процедуру **rmfield** по схеме **S = rmfield(S, 'имя поля')**, где S — имя массива записей, который корректируется. Рассмотрим пример:

```
» PG81 = rmfield(PG81, 'bat')
PG81 =
1x3 struct array with fields:
  fam
  imya
  god
```

Класс **struct**, как видим, имеет незначительное число методов, что делает его непосредственное использование при расчетах довольно проблематичным. Однако именно на применении объектов этого класса основана возможность создавать новые классы объектов (см. далее). Поэтому этот класс является очень важным для расширения возможностей системы MatLAB.

4.1.3. Класс ячеек (cell)

Массив ячеек — это массив, элементами которого являются ячейки, которые сами могут содержать любой тип массива, в том числе и массив ячеек. Массивы ячеек позволяют хранить массивы с элементами разных типов и разных измерений. Например, одна из ячеек может содержать матрицу действительных чисел, вторая — массив символьных строк, третья — вектор комплексных чисел. Можно строить массивы ячеек любых размеров и любой структуры, включая и многомерные.

Создать массив ячеек можно двумя способами:

- используя операторы присваивания;
- при помощи функции **cell** предварительно сформировать пустой массив, а потом присвоить значения отдельным ячейкам.

Применение операторов присваивания

Есть два способа присвоить значения отдельным ячейкам — индексация ячеек и индексация содержимого.

Индексация ячеек. При присваивании значений отдельным элементам массива ячеек индексы ячейки в левой от знака присваивания

части размещают в скобках, используя стандартные обозначения для массива, а в правой части присваиваемое значение ячейки помещают в фигурные скобки.

Для примера рассмотрим создание массива ячеек C размером (2*2). Для этого определим каждый элемент этого массива, т.е. каждую из ячеек, так:

```
» C(1,1) = {' Иванов И. Ю.'};
» C(1,2) = {[1 2 3; 4 5 6; 7 8 9]};
» C(2,1) = {5-3i};
» C(2,2) = {-pi : pi/5 : pi}
C =
   ' Иванов И. Ю.'    [3x3 double]
  [5.0000- 3.0000i]   [1x11 double]
```

Индексация содержимого. В этом случае в левой от знака присваивания части элемент массива ячеек указывается в фигурных скобках, а в правой части — содержимое соответствующей ячейки без скобок:

```
» C{1,1} = ' Иванов И. Ю.';
» C{1,2} = [1 2 3; 4 5 6; 7 8 9];
» C{2,1} = 5-3i;
» C{2,2} = -pi : pi/5 : pi
C =
   ' Иванов И. Ю.'    [3x3 double]
   [5.0000- 3.0000i]   [1x11 double]
```

Как видно из примеров, система MatLAB отображает массив ячеек в сокращенной форме.

Чтобы отобразить содержимое ячеек, следует применять функцию **celldisp**:

```
» celldisp(C)
C{1,1} =
  Иванов И. Ю.
C{2,1} =
   5.0000 - 3.0000i
C{1,2} =
     1     2     3
     4     5     6
     7     8     9
 C{2,2} =
 Columns 1 through 7
  -3.1416  -2.5133  -1.8850  -1.2566  -0.6283        0   0.6283
 Columns 8 through 11
   1.2566   1.8850   2.5133   3.1416
```

Для отображения структуры массива ячеек в виде графического изображения на экране предназначена функция **cellplot**:

```
» cellplot(C)
```


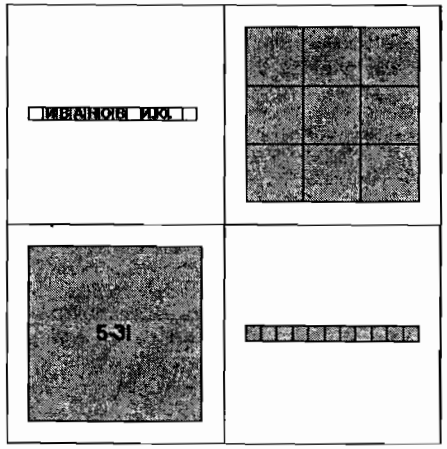

Рис. 4.1

Фигурные скобки являются конструктором массива ячеек так же, как квадратные скобки являются конструктором числового массива. Фигурные скобки аналогичны квадратным, за исключением того, что они могут быть еще и вложенными. Например, предыдущий массив ячеек C может быть построен так:

```
» C = {'Иванов И.Ю.',[1 2 3; 4 5 6; 7 8 9]; 5-3i,-pi : pi/5 : pi }
C =
   ' Иванов И.Ю.'    [3x3 double]
  [5.0000- 3.0000i]  [1x11 double]
```

Применение функции cell

Функция **cell** позволяет создать шаблон массива ячеек, заполняя его пустыми ячейками.

Пример

Создадим пустой массив ячеек размером (2*3):

```
» A = cell(2,3)
A =
    []    []    []
    []    []    []
```

Заполним одну из ячеек, используя оператор присваивания:

```
» A(2,2) = {0 : pi/10 : 2*pi}
A =
    []          []          []
    []  [1x21 double]       []
```

Извлечение данных из массива ячеек можно также осуществить двумя способами.

Первый способ рассмотрим на примерах.

Извлечение содержимого отдельных ячеек производится указанием индексов нужной ячейки в фигурных скобках:

```
» B =C{1,2}
B =
    1    2    3
    4    5    6
    7    8    9
» st = C{1,1}
st =
 Иванов И. Ю.
```

Извлечение содержимого отдельных элементов определенной ячейки производится дополнительным указанием в скобках индексов элемента массива, находящегося в нужной ячейке:

```
» x =C{1,2}(2,3)
x =
     6
» y= C{1,1}(1:5)
y =
 Иван
```

Второй способ позволяет *извлекать из массива ячеек другой массив ячеек*, составляющий часть первого:

```
» D = A(2,2:3)
D =
  [1x21 double]    []
```

В этом случае применяются обычные скобки.

Массивы ячеек используются для объединения массивов данных разных типов и размеров. Они удобнее массивов записей (структур) при следующих обстоятельствах:

- когда нужен доступ одновременно к нескольким полям;
- когда нужен доступ к подмножествам данных в виде списка переменных;
- когда число полей не определено;
- когда нужно извлекать поля из структуры.

В заключение заметим, что для того, чтобы установить, какому классу принадлежит тот или иной вычислительный объект, к имени этого объекта следует применить процедуру **class**:

```
» x=pi;
» class(x)
ans =double
» st='Лист';
» class(st)
ans =char
» s=class(num2str(x))
s =char
```

4.2. Производные классы MatLAB

Рассмотренные ранее классы объектов построены таким образом, что на их основе пользователь может создавать новые собственные классы объектов.

В самой системе MatLAB на этой основе создан и используется встроенный класс **inline**, который предоставляет простой способ определения встроенных функций для применения в программах вычисления интегралов, решения дифференциальных уравнений и вычисления минимумов и нулей функций. Пакет символьных вычислений Symbolic Math Toolbox, базируется на классе объектов **sym**, который позволяет выполнять вычисления с символьными переменными и матрицами. Пакет Control System Toolbox использует класс объектов **lti** и три его дочерних подкласса **tf**, **zpk**, **ss**, которые поддерживают алгоритмы анализа линейных стационарных систем автоматического управления.

В языке MatLAB отсутствует необходимость и возможность предварительного объявления типа или класса переменных, которые будут использованы. То же самое относится и к объектам любых вновь создаваемых классов.

Объекты класса создаются в виде структур (записей), т.е. относятся к потомкам (наследникам) класса **struct**. *Поля структуры и операции с полями являются доступными только внутри методов данного класса.*

Все M-файлы, определяющие методы для объектов данного класса, должны размещаться в специальном каталоге, который называется *каталогом класса* и обязательно имеет имя, состоящее из знака @ (коммерческое "эт") и имени класса, т.е. имеет вид @<имя класса>. Каталог класса должен быть подкаталогом одного из каталогов, описанных в путях доступа системы MatLAB, но не самим каталогом. Каталог класса обязательно должен содержать M-файл с именем, совпадающим с именем класса. Этот файл называют *конструктором класса*. Назначение такого M-файла — создавать объекты этого класса, используя данные в виде массива записей (структуры) и приписывая им метку класса.

4.2.1. Класс объектов inline

В MatLAB определен класс объектов **inline**. Он предназначен для описания функций в виде F(x, P1, P2,...), который соответствует их математическому описанию. При таком представлении вычисление функции при заданных значениях аргумента x и параметров P1, P2,... может осуществляться путем обращения к ней в естественной форме, например F(0.6, -0.5, 3).

Классу **inline** соответствует подкаталог @INLINE каталога TOOLBOX/MATLAB/FUNFUN. В нем содержатся следующие M-файлы:

- конструктор **inline**;
- методы класса: **argnames**, **disp**, **formula**, **nargin**, **vectorize**, **cat**, **display**, **horzcat**, **nargout**, **vertcat**, **char**, **feval**, **subsref**.

Конструктор inline. Эта процедура создает inline-объект, т.е. функцию, заданную в символьном виде, что позволяет обращаться к ней как к обычному математическому объекту. Процедура имеет несколько форм обращения. Обращение вида **F = inline('<математическое выражение>')** образует символьное представление заданного математического выражения как функции. Аргумент функции определяется автоматически путем поиска в составе выражения одноместного символа, отличного от **i** и **j**. Если символ отсутствует, в качестве аргумента используется символ x. Если в выражении есть несколько одноместных символов, в качестве аргумента выбирается символ, ближайший к x по алфавиту, прежде всего — следующие за ним.

```
» FUN1 = inline('am*sin(om*t+eps)')
FUN1 =
    Inline function:
    FUN1(t) = am*sin(om*t+eps)
```

Если обратиться к конструктору таким образом:

```
F = inline('<математическое выражение>', 'имя1', 'имя2', ...)
```

то формируется функция, имеющая заданные в 'имя1', 'имя2', ... обозначения аргументов:

```
» FUN2=inline('cos(alfa)*cos(beta)+...
sin(alfa)*sin(beta)*cos(gamma)','alfa','beta','gamma')
FUN2 =
    Inline function:
    FUN2(alfa,beta,gamma) =
    cos(alfa)*cos(beta)+sin(alfa)*sin(beta)*cos(gamma)
```

Наконец, при обращении вида

```
F = inline('<математическое выражение>', n)
```

создается функция, которая имеет один аргумент x и n параметров с заданными именами P1, P2, ..., Pn. Т.е. в выражении кроме некоторых заданных чисел должен содержаться только аргумент x и параметры P1, P2, ..., Pn.

```
» Fun3= inline('P1+P2*x+P3*x^2',3)
Fun3 =
    Inline function:
    Fun3(x,P1,P2,P3) = P1+P2*x+P3*x^2
```

Получение формулы функции. Это можно сделать при помощи любой из двух процедур класса **inline** — **char(F)** или **formula(F)**. Обе процедуры производят преобразование inline-объекта в символьный массив-строку, содержащую запись формулы функции:

```
» s1=char(FUN2)
s1 =
 cos(alfa)*cos(beta)+sin(alfa)*sin(beta)*cos(gamma)
» s2=formula(FUN2)
s2 =
 cos(alfa)*cos(beta)+sin(alfa)*sin(beta)*cos(gamma)
» s3= formula(Fun3)
s3 =
 P1+P2*x+P3*x^2
```

Вывод на экран. Процедуры **disp(F)** и **display(F)** осуществляют вывод на экран заданного inline-объекта **(F)**:

```
» disp(Fun3)
    Inline function:
    Fun3(x,P1,P2,P3) = P1+P2*x+P3*x^2
» display(Fun3)
Fun3 =
    Inline function:
    Fun3(x,P1,P2,P3) = P1+P2*x+P3*x^2
```

Процедуры незначительно различаются формой вывода. Основное их отличие в том, что процедура **display** работает и при неявном обращении — когда имя этой процедуры не указывается, а в командной строке записано лишь имя inline-объекта:

```
» Fun3
Fun3 =
    Inline function:
    Fun3(x,P1,P2,P3) = P1+P2*x+P3*x^2
```

Получение имен аргументов inline-объекта. Данное действие осуществляется процедурой **argnames(F)**:

```
» argnames(FUN1)
ans =    't'
» argnames(FUN2)
ans =
   'alfa'
   'beta'
   'gamma'
» argnames(Fun3)
ans =
   'x'
   'P1'
   'P2'
   'P3'
```

Векторизация функций. Часто желательно выражение для функции, которое записано для аргументов-чисел, преобразовать так, чтобы

вычисление можно было осуществлять и тогда, когда аргументами являются векторы. Для этого в исходном выражении функции надо вставить символ "." перед каждым знаком арифметической операции. Это делает процедура **vectorize**. Если аргументом этой процедуры является символьное выражение, она формирует другое символьное выражение с указанными изменениями. В случае, когда аргумент — inline-объект, она создает новый inline-объект, в формуле которого произведены эти изменения. Приведем примеры:

```
» s = char(Fun3)
s = P1+P2*x+P3*x^2
» sv = vectorize(s)
sv = P1+P2.*x+P3.*x.^2
» Fun3v = vectorize(Fun3)
Fun3v =
    Inline function:
    Fun3v(x,P1,P2,P3) = P1+P2.*x+P3.*x.^2
```

Вычисление inline-объекта. Чтобы вычислить значение функции, представленной как inline-объект, по заданным значениям аргументов и параметров, достаточно после указания имени inline-объекта указать в скобках значения аргументов и параметров функции:

```
» v = 0 : 0.2 :1
» F3 =Fun3v(v, 2, 3, 4)
v =     0  0.2000  0.4000  0.6000  0.8000  1.0000
F3 =    2.0000  2.7600  3.8400  5.2400  6.9600  9.0000
```

Вычисление значения функции, заданной М-файлом. Наиболее важной для практического программирования сложных вычислительных алгоритмов процедурой класса **inline** является функция **feval**. С ее помощью можно производить вычисления по алгоритмам, которые являются общими для любой функции определенной структуры. В этом случае алгоритмы удобно строить общими для всего класса таких функций, а конкретный вид функции будет определяться отдельной процедурой в виде М-функции. При этом, имя такого М-файла должно быть в структуре общего алгоритма одной из переменных, чтобы, изменяя его конкретное значение, можно было применять алгоритм для любых функций той же структуры. В таком случае говорят, что функция является внешней (external) по отношению к алгоритму.

Таким образом, процедура **feval** позволяет использовать внешние функции при программировании в среде MatLAB. Общий вид обращения и примеры применения процедуры **feval** приведены в разделе 2.6.1.

4.2.2. Классы пакета CONTROL

Пакет прикладных программ (ППП) Control System Toolbox (сокращенно — CONTROL) сосредоточен в подкаталоге CONTROL каталога TOOLBOX системы MatLAB.

Основными вычислительными объектами этого ППП являются:

- Родительский объект (класс) **LTI** (Linear Time-Invariant System — линейные, инвариантные во времени системы); в русскоязычной литературе за этими системами закрепилось название линейные стационарные системы (ЛСС).
- Дочерние объекты (классы), т.е. подклассы класса **LTI**, соответствующие трем разным представлениям ЛСС:
  - **TF**-объект (Transfer Function — передаточная функция);
  - **ZPK**-объект (Zero-Pole-Gain — нули-полюсы-коэффициент передачи);
  - **SS**-объект (State Space — пространство состояния).

Объект **LTI**, как наиболее общий, содержит информацию, не зависящую от конкретного представления ЛСС (непрерывного или дискретного), а также от имен входов и выходов. Дочерние объекты определяются конкретной формой представления ЛСС, т.е. зависят от модели представления. Объект класса **TF** характеризуется векторами коэффициентов числителя и знаменателя рациональной передаточной функции. Объект класса **ZPK** характеризуется векторами, содержащими значения нулей, полюсов передаточной функции системы и коэффициента передачи системы. Наконец, объект класса **SS** определяется четверкой матриц, описывающих динамическую систему в пространстве состояния. Ниже приведены основные атрибуты этих классов, их обозначения и смысл.

Атрибуты (поля) LTI-объектов

Ниже NU, NY и NX определяют число входов (вектор U), выходов (вектор Y) и переменных состояния (вектор X) ЛСС соответственно; OM (SISO) — одномерная система, т.е. система с одним входом и одним выходом; MM (MIMO) — многомерная система (с несколькими входами и выходами).

Специфические атрибуты передаточных функций (TF-объектов)

num — *Числитель*
Вектор-строка для OM-систем; для MM-систем — массив ячеек из векторов-строк размером NY на NU (например, {[1 0] 1 ; 3 [1 2 3]})

den — *Знаменатель*
Вектор-строка для OM-систем; для MM-систем — массив ячеек из векторов-строк размером NY на NU.
Например:
tf({-5 ; [1 -5 6]} , {[1 -1] ; [1 1 0]})
определяет систему с одним входом и двумя выходами
[-5 /(s-1)]
[(s^2-5s+6)/(s^2+s)]

Variable *Имя (тип) переменной (из перечня)*
Возможны варианты: s, p, z, z^-1 или q.
По умолчанию принимается s (для непрерывных переменных) и z (для дискретных).
Имя переменной влияет на отображение и создает дискретную ПФ для дискретных сигналов

Специфические атрибуты ZPK-объектов

z *Нули*
Вектор-строка для OM-систем; для MM-систем — массив ячеек из векторов-строк размером NY на NU

p *Полюсы*
Вектор-строка для OM-систем; для MM-систем — массив ячеек из векторов-строк размером NY на NU

k *Коэффициенты передачи*
Число — для OM-систем, матрица NY на NU для MM-систем

Variable *Имя (тип) переменной (из перечня)*
То же, что и для TF-объекта (см. выше)

Специфические атрибуты SS-объектов (моделей пространства состояния)

a,b,c,d A,B,C,D — матрицы, в соответствии с уравнениями в переменных состояния:
x = Ax + Bu, y = Cx + Du

e E — матрица для систем Descriptor'a (описателя).
По умолчанию E = eye(size(A))

StateName Имя переменной состояния (не обязательное). Массив ячеек NX на 1 из строк (используйте '' для состояний без имени). Пример: {'положение' ; 'скорость'}

Атрибуты, общие для всех LTI-моделей

Ts *Дискрет по времени (в секундах)*
Положительный скаляр (период дискретизации) для дискретных систем.
Ts = -1 для дискретных систем
с неустановленной частотой дискретизации.
Ts = 0 для непрерывных систем.

Td *Задержки входов (в секундах)*
Вектор 1 на NU промежутков времен задержек входов. Установка Td как скаляра определяет единую задержку по всем входам. Используется только для непрерывных систем. Используйте D2D для установки задержек в дискретных системах. Td = [] для дискретных систем

InputName *Имена входов*
Строка для систем с одним входом. Массив ячеек NU на 1 из строк для систем с несколькими входами (используйте '' для переменных без имени). Примеры: 'момент' или {'напор' ; 'отклонение элеронов'}

OutputName *Имена выходов*
Строка для систем с одним выходом. Массив ячеек NY на 1 из строк для систем с несколькими выходами (используйте '' для переменных без имени). Пример: 'мощность' или {'скорость' ; 'угол атаки'}

Notes *Заметки*
Любая строка или массив ячеек из строк символов. Пример: 'Эта модель создана в январе 2000'

Userdata Дополнительная информация или данные. Может быть любого типа MATLAB

Приведем перечень методов класса **LTI**:

augstate, damp, get, issiso, lticheck, pade, series, trange, balreal, display, gram, kalman, margin, parallel, set, tzero, bode, dssdata, impulse, kalmd, modred, pzmap, sigma, uplus, canon, eig, inherit, lqgreg, nichols, quickset, ss2ss, zpkdata, connect, estim, initial, lqry, norm, reg, ssdata, covar, evalfr, isct, lsim, nyquist, rlocfind, step, ctrb, fgrid, isempty, lti, obsv, rlocus, tfdata

Конструктором LTI-объектов является файл lti.m в поддиректории @LTI. Он создает только шаблон LTI-объекта по некоторым его параметрам.

```
function sys = lti(p,m,T)
% LTI - конструктор LTI-объекта
% SYS = LTI(P,M) создает LTI-объект размером P на M.
% SYS = LTI(P,M,T) создает LTI-объект размером P на M
% с дискретом времени T.
% По умолчанию система является непрерывной, а имена
% входа/выхода — векторами ячеек с пустыми строками.
ni = nargin;
error(nargchk(1,3,ni))
if isa(p,'lti')
% Дублирование LTI-объекта
  sys = p;
  return
elseif ni==3 & T~=0,
  sys.Ts = T;
  sys.Td = [];
else
  sys.Ts = 0;
```

```
  sys.Td = zeros(1,m);
end
estr = {''};
sys.InputName = estr(ones(m,1),1);
sys.OutputName = estr(ones(p,1),1);
sys.Notes = {};
sys.UserData = [];
sys.Version = 1.0;
sys = class(sys,'lti');          % Конец @lti/lti.m
```

Как видно из приведенного описания, непосредственное применение конструктора **lti** дает возможность задать только число входов и выходов ЛСС, а также величину дискрета времени. Остальные атрибуты LTI-объекта могут быть определены только употреблением других процедур. В дальнейшем будет показано, что такие атрибуты, как имена входов и выходов и некоторые вспомогательные данные можно задать процедурой **set**. Конкретные же числовые характеристики ЛСС возможно задать лишь с помощью применения одного из конструкторов дочерних классов.

Рассмотрим пример создания LTI-объекта для непрерывной ОМ-системы:

```
» sys=lti(1,1)
lti object
```

Чтобы убедиться, что LTI-объект создан именно с указанными параметрами, воспользуемся процедурой **get(sys)** для получения значений его атрибутов:

```
» get(sys)
    Ts = 0
    Td = 0
    InputName = {''}
    OutputName = {''}
    Notes = {}
    UserData = []
```

Большинство полей созданного LTI-объекта являются пустыми, только два из них равняются нулю. Кроме того, из описания конструктора вытекает, что обращение к нему не предусматривает возможности установки значений таких полей LTI-объекта, как **InputName**, **OutputName**, **Notes** и **UserData**. Последнее можно сделать, только используя специальную функцию **set** по схеме:

```
set(<имя LTI-объекта>,'<имя поля>', <Значение>)
```

Рассмотрим это на примере установки в уже сформированном LTI-объекте **sys** значений некоторых из отмеченных полей:

```
» set(sys,'InputName','Угол','OutputName',...
     'Напряжение','Notes','Гиротахометр')
```


Проконтролируем это:

```
» get(sys)
    Ts = 0
    Td = 0
    InputName = {'Угол'}
    OutputName = {'Напряжение'}
    Notes = {'Гиротахометр'}
    UserData = []
```

Ниже приведен сжатый перечень основных процедур пакета CONTROL, сгруппированных по функциональному назначению.

Создание LTI-моделей

ss	Создает модель пространства состояния
zpk	Создает модель нули/полюсы/к-ты передачи
tf	Создает модель передаточной функции
dss	Специфицирует описатель модели пространства состояния
filt	Специфицирует цифровой фильтр
set	Установка/модификация атрибутов LTI-модели
ltiprops	Детальная справка об атрибутах LTI-моделей

Извлечение данных

ssdata	Извлечение матриц пространства состояния
zpkdata	Извлечение данных о нулях/полюсах/КП
tfdata	Извлечение числителя(-лей) и знаменателя(-лей) ПФ
dssdata	Получение информации о версии описателя SSDATA
get	Получение информации о значениях свойств LTI-модели

Получение информации об отдельных характеристиках модели

class	Информация о типе модели (ss, zpk или tf)
size	Информация о размерах матриц входа и выхода
isempty	Проверка, является ли LTI-модель пустой
isct	Проверка, является ли модель непрерывной
isdt	Проверка, является ли модель дискретной
isproper	Проверка, является ли модель правильной
issiso	Проверка, имеет ли модель один вход и один выход
isa	Проверка, является ли LTI-модель моделью заданного типа

Преобразование системы

ss	Преобразование в пространство состояния
zpk	Преобразование в нули/полюсы/КП

tf	Преобразование в передаточные функции
c2d	Преобразование непрерывного времени в дискретное
d2c	Преобразование дискретного времени в непрерывное
d2d	Переопределение дискретной системы или добавление задержек входных воздействий

"Арифметические" операции

+ и -	Добавление и вычитание LTI-систем (параллельное соединение)
*****	Умножение LTI-систем (последовательное соединение)
****	Левое деление — sys1\sys2 равносильно inv(sys1)*sys2
/	Правое деление — sys1/sys2 равнозначно sys1*inv(sys2)
'	Перетранспонирование
.'	Транспонирование карты входа/выхода
[..]	Горизонтальное/вертикальное объединение LTI-систем
inv	Обращение LTI-системы

Модели динамики

pole, eig	Полюсы системы
tzero	Нули системы
pzmap	Карта нулей-полюсов
dcgain	Коэффициент передачи при нулевой (низкой) частоте
norm	Нормы LTI-систем
covar	Ковариация отклика на белый шум
damp	Частота собственных колебаний и демпфирование по полюсам системы
esort	Сортировка полюсов непрерывной системы по их действительным частям
dsort	Сортировка полюсов дискретной системы по их модулям
pade	Аппроксимация Паде задержек по времени

Модели пространства состояния

rss, drss	Генерирование случайных моделей пространства состояния
ss2ss	Преобразование переменных состояния
canon	Каноническая форма пространства состояния
ctrb, obsv	Матрицы управляемости и наблюдаемости
gram	Определители Грама управляемости и наблюдаемости
ssbal	Диагональная балансировка матриц пространства состояния
balreal	Балансировка входа-выхода на основе определителя Грама

modred	Редукция состояния модели
minreal	Минимальная реализация и сокращение нулей и полюсов
augstate	Увеличение выхода за счет присоединения состояний

Отклик во времени

step	Отклик на единичный скачок
impulse	Отклик на единичный импульс
initial	Отклик на заданные начальные условия состояния
lsim	Отклик на произвольные входы
ltiview	Анализ откликов с помощью графического интерфейса
gensig	Генерирует периодические сигналы для LSIM
stepfun	Генерирует единичный скачок

Частотный отклик

bode	Диаграмма Боде частотного отклика (АЧХ и ФЧХ)
sigma	Частотный график сингулярных значений
nyquist	Диаграмма Найквиста
nichols	Диаграмма Николса
ltiview	Анализ откликов с помощью графического интерфейса
evalfr	Расчет частотного отклика на заданной частоте
freqresp	Частотный отклик над сеткой частот
margin	Запасы по фазе и амплитуде

Объединение систем

append	Объединение LTI-систем путем объединения входов и выходов
parallel	Обобщенное параллельное соединение (см. также процедуру +)
series	Обобщенное последовательное соединение (см. также процедуру *)
feedback	Обратное соединение двух систем
star	Соединение звездой Редхеффера
connect	Получение ss-модели из описания блок-схемы

Процедуры классической графики

rlocus	Диаграмма Эванса размещения корней
rlocfind	Интерактивное определение звена заданием расположения корней
acker	Размещение полюсов ОМ-системы
place	Размещение полюсов ММ-системы
estim	Создает оцениватель по заданному КП оценивателя
reg	Создает регулятор по заданной матрице обратной связи и коэффициентам оценивателя

Инструменты проектирования LQG

lqr, dlqr	Линейно-квадратичный (LQ) регулятор обратной связи
lqry	LQ-регулятор с выходным взвешиванием
lqrd	Дискретный LQ-регулятор для непрерывной системы
kalman	Фильтр Калмана
kalmd	Дискретный фильтр Калмана для непрерывной системы
lqgreg	Формирователь LQG-регулятора по LQ-коэффициентам и фильтру Калмана

Решение матричных уравнений

lyap	Решение непрерывных уравнений Ляпунова
dlyap	Решение дискретных уравнений Ляпунова
care	Решение непрерывных алгебраических уравнений Риккати
dare	Решение дискретных алгебраических уравнений Риккати

Демонстрационные программы

ctrldemo	Введение в Control System Toolbox
jetdemo	Классическое проектирование САУ углом рыскания
diskdemo	Цифровое проектирование контроллера привода жесткого диска
milldemo	OM и MM LQG управление прокатного стана
kalmdemo	Проектирование и моделирование фильтра Калмана

LTI-модели можно создавать как SS-, TF- или ZPK-объекты. Для этого используются соответственно процедуры-конструкторы **ss**, **tf** и **zpk**.

Создание LTI-модели рассмотрим на примере модели трехстепенного астатического гироскопа. Уравнения движения такого гироскопа имеют вид:

$$\begin{cases} \ddot{\alpha} + \lambda\dot{\beta} = n(t) \\ \ddot{\beta} - \lambda\dot{\alpha} = l(t) \end{cases} \quad (1)$$

где $n(t)$ и $l(t)$ — моменты сил, действующие на гироскоп по осям подвеса; α и β — углы поворота гироскопа в пространстве; λ — частота собственных (нутационных) колебаний гироскопа.

Чтобы создать SS-модель, необходимо прежде всего привести дифференциальные уравнения движения динамической системы к стандартному виду типа:

$$\begin{cases} \frac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u \end{cases} \quad (2)$$

где u — вектор входных переменных; y — вектор выходных переменных, а x — вектор переменных состояния системы. Из этого следует, что перед формированием SS-модели необходимо:

- определить, какие величины будут задаваться как явные функции времени, т.е. какие величины составят вектор u входных переменных;
- определить, какие величины будут образовывать вектор y выходных переменных (т.е. будут находиться путем решения системы заданных дифференциальных уравнений);
- установить, какие величины будут составлять вектор x переменных состояния системы (их число должно совпадать с порядком системы заданных дифференциальных уравнений);
- с помощью введенных переменных состояния привести заданную систему дифференциальных уравнений к так называемой *нормальной форме Коши*, т.е. к системе дифференциальных уравнений первого порядка, разрешенных относительно производных.

Будем полагать моменты сил — "входами" гироскопа, а углы поворота гироскопа — "выходами". Тогда система гироскоп (будем обозначать ее GYRO) имеет 2 входа (n(t) и l(t)) и 2 выхода (α и β). В качестве переменных состояния примем выходные переменные и их первые производные по времени:

$$x_1 = \alpha; \quad x_2 = \beta; \quad x_3 = \dot{\alpha}; \quad x_4 = \dot{\beta} \quad (3)$$

Тогда уравнения гироскопа в форме Коши приобретут вид:

$$\begin{cases} \dot{x}_1 = x_3 \\ \dot{x}_2 = x_4 \\ \dot{x}_3 = -\lambda \cdot x_4 + n(t) \\ \dot{x}_4 = \lambda \cdot x_3 + l(t) \end{cases} \quad (4)$$

Теперь нужно образовать матрицы A, B, C и D в соответствии с формой (2) представления системы в пространстве состояния. В рассматриваемом случае в качестве выходного вектора y примем:

$$y = [\alpha, \beta]^T \quad (5)$$

в качестве входного — вектор моментов сил:

$$u = [n(t), l(t)]^T \quad (6)$$

Полагая

$$x = [x_1, x_2, x_3, x_4]^T \quad (7)$$

значения указанных матриц должны быть такими:

$$A=\begin{bmatrix}0&0&1&0\\0&0&0&1\\0&0&0&-\lambda\\0&0&\lambda&0\end{bmatrix};\ B=\begin{bmatrix}0&0\\0&0\\1&0\\0&1\end{bmatrix};\ C=\begin{bmatrix}1&0&0&0\\0&1&0&0\end{bmatrix};\ D=\begin{bmatrix}0&0\\0&0\end{bmatrix} \qquad (8)$$

Введем эти матрицы в командном окне MatLAB, принимая λ=10:

```
» lambda=10;
» A=zeros(4,4); A(1,3)=1;A(2,4)=1;A(3,4)= -lambda;
A(4,3)=lambda; A
A =
    0    0    1    0
    0    0    0    1
    0    0    0  -10
    0    0   10    0
» B=zeros(4,2); B(3,1)=1; B(4,2)=1
B =
    0    0
    0    0
    1    0
    0    1
» C=zeros(2.2); C=[diag([1 1]) C]
C =
    1    0    0    0
    0    1    0    0
```

Теперь можно приступить к созданию LTI-объекта с именем GYRO, используя модель в пространстве состояния:

```
» GYROss=ss(A,B,C,0)
a =
           x1     x2     x3     x4
      x1    0      0      1      0
      x2    0      0      0      1
      x3    0      0      0    -10
      x4    0      0     10      0
b =
           u1     u2
      x1    0      0
      x2    0      0
      x3    1      0
      x4    0      1
c =
           x1     x2     x3     x4
      y1    1      0      0      0
      y2    0      1      0      0
d =
           u1     u2
      y1    0      0
      y2    0      0
Continuous-time system.
```


Как видно, модель сформирована правильно. Можно начать некоторые ее преобразования.

Прежде всего интересно найти передаточные функции созданной системы. Очевидно, их должно быть 4 (ибо у нас 2 выхода и 2 входа). Для этого применим процедуру преобразования **tf**:

```
» GYROtf=tf(GYROss)
 Transfer function from input 1 to output...
          s^2
 #1: -------------
     s^4 + 100 s^2
         10 s
 #2: -------------
     s^4 + 100 s^2

 Transfer function from input 2 to output...
         -10 s
 #1: -------------
     s^4 + 100 s^2
          s^2
 #2: -------------
     s^4 + 100 s^2
```

Теперь преобразуем введенную SS-модель в ZPK-модель при помощи процедуры **zpk**:

```
» GYROzp=zpk(GYROss)

 Zero/pole/gain from input 1 to output...
           s^2
 #1: ---------------
     s^2 (s^2 + 100)
          10 s
 #2: ---------------
     s^2 (s^2 + 100)

 Zero/pole/gain from input 2 to output...
          -10 s
 #1: ---------------
     s^2 (s^2 + 100)
           s^2
 #2: ---------------
     s^2 (s^2 + 100)
```

Поскольку первая (SS) модель GYROss была создана непосредственно процедурой-конструктором по заданным числовым данным, а последующие (GYROtf и GYROzp) — путем преобразования уже созданной модели, будем называть модель, созданную конструктором, основной, а остальные — вспомогательными. Отметим, что SS-модель в MatLAB можно создать и по системе дифференциальных уравнений,

не разрешенных относительно производных, т.е. когда система описывается совокупностью уравнений вида:

$$\begin{cases} E \cdot \frac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u \end{cases} \quad (9)$$

где E — произвольная квадратная матрица размером (n*n), а n — порядок заданной системы дифференциальных уравнений. Для этого следует уже использовать не конструктор **ss**, а специальную процедуру **dss**, отличие которой от предыдущей лишь в том, что она требует задания не четырех, а пяти матриц, последней из которых должна быть матрица E.

В качестве примера рассмотрим уравнения того же гироскопа в виде:

$$\begin{cases} J_1 \cdot \ddot{\alpha} + H \cdot \dot{\beta} = N(t) \\ J_2 \cdot \ddot{\beta} - H \cdot \dot{\alpha} = L(t) \end{cases} \quad (10)$$

Вводя те же переменные (см. (3), (5)...(7)), получим систему уравнений в виде (9), где матрицы A и E будут иметь вид:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -H \\ 0 & 0 & H & 0 \end{bmatrix}; \quad E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & J_1 & 0 \\ 0 & 0 & 0 & J_2 \end{bmatrix} \quad (11)$$

Остальные матрицы — B, C и D будут прежними (8). Введем новые матрицы при таких значениях параметров — H=10, J_1=2, J_2=3:

```
» H= 10; J1=2; J2=3;
» A=zeros(4); A(1,3)=1; A(2,4)=1; A(3,4)=-H; A(4,3)=H
A =
    0    0    1    0
    0    0    0    1
    0    0    0  -10
    0    0   10    0
» E=eye(4); E(3,3)=J1; E(4,4)=J2
E =
    1    0    0    0
    0    1    0    0
    0    0    2    0
    0    0    0    3
```

Теперь зададим SS-модель, пользуясь процедурой **dss**:

```
» Gyross=dss(A,B,C,0,E)
a =
          x1     x2     x3     x4
   x1      0      0      1      0
   x2      0      0      0      1
   x3      0      0      0    -10
   x4      0      0     10      0
```



```
b =
          u1     u2
   x1      0      0
   x2      0      0
   x3      1      0
   x4      0      1
c =
          x1     x2     x3     x4
   y1      1      0      0      0
   y2      0      1      0      0
d =
          u1     u2
   y1      0      0
   y2      0      0
e =
          x1     x2     x3     x4
   x1      1      0      0      0
   x2      0      1      0      0
   x3      0      0      2      0
   x4      0      0      0      3
Continuous-time system.
```

Как и ранее, создадим на этой основе вспомогательные TF- и ZPK-модели:

```
» Gyrotf=tf(Gyross)
 Transfer function from input 1 to output...
         0.5 s^2
 #1: ---------------
     s^4 + 16.67 s^2
          1.667 s
 #2: ---------------
     s^4 + 16.67 s^2
 Transfer function from input 2 to output...
        -1.667 s
 #1: ---------------
     s^4 + 16.67 s^2

        0.3333 s^2
 #2: ---------------
     s^4 + 16.67 s^2

» Gyrozp=zpk(Gyross)
 Zero/pole/gain from input 1 to output...
          0.5 s^2
 #1: -----------------
     s^2 (s^2 + 16.67)
          1.6667 s
 #2: -----------------
     s^2 (s^2 + 16.67)
 Zero/pole/gain from input 2 to output...
```

```
        -1.6667 s
#1: -----------------
    s^2 (s^2 + 16.67)
       0.33333 s^2
#2: -----------------
    s^2 (s^2 + 16.67)
```

В предыдущих примерах за основу была принята SS-модель. Но в качестве основной можно выбрать и любую из двух других моделей. Возьмем, например, в качестве основной модель в передаточных функциях.

Система, описываемая уравнениями (10), если принять те же, что и ранее входные и выходные величины, имеет 4 передаточных функции, которые образуют матрицу передаточных функций размером (2*2). Каждый из столбцов этой матрицы содержит передаточные функции, соответствующие некоторой одной входной величине по всем выходным величинам. Определенная строка матрицы, наоборот, содержит передаточные функции какой-то одной выходной величины по всем входам системы. В целом матрица передаточных функций в рассматриваемом случае может быть представлена в виде:

$$W(s)=\begin{bmatrix} W_{11}(s) & W_{12}(s) \\ W_{21}(s) & W_{22}(s) \end{bmatrix}$$

В соответствии с уравнениями (10) значения элементов этой матрицы равны:

$$W_{11}(s)=\frac{J_2}{J_1\cdot J_2\cdot s^2+H^2};\quad W_{12}(s)=\frac{H}{s\cdot(J_1\cdot J_2\cdot s^2+H^2)};$$
$$W_{21}(s)=\frac{H}{s\cdot(J_1\cdot J_2\cdot s^2+H^2)};\quad W_{22}(s)=\frac{J_2}{J_1\cdot J_2\cdot s^2+H^2} \qquad (12)$$

Чтобы ввести эти передаточные функции и создать на их основе TF-модель, следует вначале создать два массива ячеек:

- массив ячеек размером (2*2) из векторов коэффициентов всех числителей передаточных функций;
- массив ячеек такого же размера из векторов коэффициентов знаменателей передаточных функций.

Для рассматриваемого случая это можно сделать так. Сначала создадим вектор коэффициентов общей части знаменателей:

$$V_{zn1}=\left[J_1\cdot J_2, 0, H^2\right]$$

затем — вектор дополнительного множителя в некоторых знаменателях:

$$V=[1,\ 0]$$

Теперь создадим вектор коэффициентов второго знаменателя путем свертки этих двух векторов (это соответствует перемножению полиномов):

$$V_{zn2}=\mathrm{conv}(V_{zn1},\ V)$$

Сформируем массив **den** ячеек знаменателей по схеме:

```
for k1=1:4
    for k2=1:2
        den(k1,k2)={Vzn1};
    end
end
den(1,2)={Vzn2};          den(2,1)={Vzn2}
```

Переходя к определению массива ячеек **nom** числителя, можно записать его таким образом: **nom** = {J_2, -H; H, J_1}.

Теперь можно сформировать TF-модель, используя установленные матрицы ячеек числителей и знаменателей.

```
» Vzn1=[J1*J2, 0, H^2]
Vzn1 =
   6   0  100
» V=[1, 0]
V =
   1   0
» Vzn2=conv(Vzn1,V)
Vzn2 =
   6   0  100   0
» for k1=1:2
      for k2=1:2
          den(k1,k2)={Vzn1};
      end
  end
» den(1,2)={Vzn2}; den(2,1)={Vzn2}
den =
  [1x3 double]  [1x4 double]
  [1x4 double]  [1x3 double]
» nom ={J2, -H; H, J1}
nom =
  [ 3]  [-10]
  [10]  [ 2]
» gyrotf=tf(nom,den)
 Transfer function from input 1 to output...
         3
 #1: ----------
     6 s^2 + 100
        10
 #2: ------------
     6 s^3 + 100 s
 Transfer function from input 2 to output...
```

```
          -10
  #1: -------------
      6 s^3 + 100 s
            2
  #2: -----------
      6 s^2 + 100
```

Предупреждение

При манипуляциях или преобразованиях LTI-модели учитывайте следующее:

- Три формы представления LTI-объектов не являются эквивалентными при численных расчетах; в частности, точность вычислений с передаточными функциями высокого порядка часто недостаточна; вы должны работать по преимуществу со сбалансированными моделями пространства состояния и использовать передаточные функции только для отображения на экране или для интерпретации (расшифровки) результатов.
- Преобразование в формат передаточных функций может сопровождаться потерями точности; в результате, полюсы передаточной функции могут заметно отличаться от полюсов заданной ZPK-модели или модели пространства состояния (для проверки наберите **help roots**).
- Преобразования в пространство состояния не являются однозначно определенными в случае одномерной системы и не гарантируют создания минимальной конфигурации системы в случае многомерной системы; так, заданная в пространстве состояния модель SYS, при преобразовании **ss(tf (sys))** может сформировать модель с другими матрицами пространства состояния или даже с другим числом переменных состояния в многомерном случае; таким образом, *следует по возможности избегать преобразований моделей из одной формы в другую.*

Проиллюстрируем это на примере. Преобразуем созданную основную TF-модель gyrotf в SS-модель gyross:

```
» gyross=ss(gyrotf)
a =          x1      x2         x3      x4      x5         x6
      x1      0       0          1       0       0          0
      x2      0       0    -4.1667       0       0          0
      x3      0       4          0       0       0          0
      x4      0       0          0       0       0          1
      x5      0       0          0       0       0    -4.1667
      x6      0       0          0       0       4          0
b =          u1      u2
      x1      0       0
      x2    0.5       0
      x3      0       0
      x4      0       0
      x5      0     0.5
      x6      0       0
c =          x1      x2         x3         x4      x5         x6
y1            0       0        .25   -0.83333       0          0
y2      0.83333       0          0          0       0    0.16667

d =          u1      u2
      y1      0       0
      y2      0       0
Continuous-time system.
```

Нетрудно убедиться, что мы получили систему, совсем не похожую на ранее введенную SS-модель (11), хотя обе они описывают одну и ту же ЛСС. Новая модель отличается от предыдущей не только другими значениями основных матриц, но и, что совсем необычно и непонятно, числом переменных состояния. Из теории известно, что число переменных состояния должно быть равно порядку выбранной системы дифференциальных уравнений. Поэтому в системе (10), имеющей четвертый порядок, должно быть четыре переменных состояния. В последнем случае число их возросло до шести.

4.3. Пример создания нового класса polynom

Создание нового класса рассмотрим на примере класса многочленов. Назовем этот класс **polynom**. В этом классе объектом будет полином, т.е. функция одной переменной (например, x) вида:

```
p(x) = an*x^n + ... + a2*x^2 + a1*x + a0
```

Очевидно, полином как функция полностью определяется указанием целого положительного числа n, которое задает наибольший показатель степени аргумента, коэффициент при котором не равен нулю (an не равно нулю), и вектора длиною n+1 из его коэффициентов:

```
c = [an ... a2 a1 a0]
```

4.3.1. Создание подкаталога @polynom

Для создания подкаталога нового класса вызовите команду **Open** из меню **File**, а затем в появившемся окне перейдите к папке Toolbox\Matlab\Polyfun. Воспользуйтесь пиктограммой создания новой папки в этом окне, чтобы открыть папку с именем @POLYNOM.

Перейдите во вновь созданную папку. Теперь вы готовы к созданию M-файлов нового класса.

4.3.2. Создание конструктора

Первым необходимым шагом в создании нового класса объектов является создание конструктора polynom-объекта, т.е. М-файла, который образовывал бы новый polynom-объект по некоторым заданным числовым данным.

Для этого прежде всего надо установить структуру polynom-объекта как записи. Из характеристики полинома как математического объекта следует, что можно выбрать представление polynom-объекта в виде записи, состоящей из двух полей:

- **n** — целого числа, задающего порядок полинома;
- **c** — вектора длиною n+1 коэффициентов полинома.

Входным аргументом для образования polynom-объекта должен быть, очевидно, заданный вектор его коэффициентов v.

В процедуре конструктора должны быть предусмотрены такие операции:

- создание структуры (записи) **p** с полями **p.n** и **p.c**;
- преобразование этой структуры в polynom-объект.

Последнее осуществляется применением специальной функции **class**:

```
p = class(p, '<имя класса>')
```

Ниже приведен возможный текст М-файла **polynom.m**.

```
function p=polynom(v,cs);
% POLYNOM — конструктор polynom-объектов.
% Под polynom-объектом понимается объект языка MatLab,
% который является записью с двумя полями:
% .c - вектор-строка, содержащий коэффициенты
%      полинома в порядке уменьшения степени аргумента;
% .n - число, равное порядку полинома.
% p=POLYNOM(v) формирует polynom-объект p по заданному
% вектору v, состоящему из значений коэффициентов
% будущего полинома в порядке уменьшения степени аргумента.
% p=POLYNOM(v,cs) формирует polynom-объект p по заданным
% вектору v корней полинома и значению cs его
% старшего коэффициента.

if nargin==0      % Эта часть
   p.c=[];        % создает пустой polynom-объект,
   p.n=0;         % если отсутствуют аргументы.
   p=class(p,'polynom');  %
elseif isa(v,'polynom')   % Этот фрагмент создает дубликат,
   p=v;           % если аргумент является polynom-объектом.
elseif nargin==2 % Эта часть работает, если в обращении
                 % 2 аргумента.
   if cs==0      % Если старший коэффициент равен нулю,
      cs=1;      % его надо заменить на 1.
   end
   k=length(v);
   for i=1:k
       vs(i,:)=[1 -v(i)];
   end
   p.n=k;        % Определение порядка полинома.
   p.c=cs*vs(1,:);              % Формирование
   for i=2:k                    % вектора
       p.c=conv(p.c,vs(i,:));   % коэффициентов
   end                          % полинома.
   p=class(p,'polynom'); % Присвоение метки polynom-объекту.
else          % Эта часть работает, если аргумент один.
   k=length(v);
   n=k;  m=1;
   while v(m)==0  % Этот цикл сокращает длину входного вектора
      n=n-1;      % (уменьшает порядок полинома) в случае,
      m=m+1;      % когда первые элементы вектора
   end            % равны нулю.
   p.n=n-1;               % Тут присваиваются значения полям
   p.c=v(k-n+1:end);      % записи будущего polynom-объекта.
   p=class(p,'polynom'); % Присвоение метки polynom-объекту.
end                      % Завершение конструктора POLYNOM.
```

Система MatLAB позволяет вызывать конструктор без аргументов. В этом случае конструктор должен создать шаблон объекта с пустыми полями. Возможно также, что конструктор будет вызываться с входным аргументом, который уже является polynom-объектом. Тогда конструктор должен создать дубликат входного аргумента. *Функция* ***isa*** *проверяет принадлежность входного аргумента указанному классу.*

Если аргумент существует и является единственным, он перестраивается так, чтобы стать вектором-строкой и присваивается полю **.c** результата. Если аргументов два, то первый из них полагается вектором корней, а второй — значением старшего коэффициента полинома. Так как в этом случае порядок полинома обязательно должен быть равен числу корней, старший коэффициент не может быть равен нулю. Поэтому, если ошибочно второй аргумент равен нулю, он исправляется на единицу.

Функция **class** используется для присвоения результату метки, которая определяет его как polynom-объект.

4.3.4. Создание процедуры символьного представления polynom-объекта

Следующим шагом в формировании класса **polynom** целесообразно сделать создание М-файла, который образовывал бы символьное представление заданного polynom-объекта. Такое представление необходимо для того, чтобы можно было убеждаться в правильности формирования polynom-объектов и контролировать правильность действий отдельных создаваемых методов класса

polynom, а также получать наглядные результаты преобразований полиномов в программах.

Создадим этот M-файл в подкаталоге @POLYNOM и назовем его **char**. Единственным аргументом процедуры **char** является заданный полином-объект p, а выходной величиной — массив s символов, являющийся символьным представлением полинома. Ниже приведен вариант такого M-файла. Представленный вариант формирует символьную строку вида:

```
<значение an>*x^n + ... + <значение a2>*x^2+
<значение a1>*x + <значение a0>
```

с исключением членов, коэффициенты при которых равны нулю.

```
function s = char(p)
% POLYNOM/CHAR - формирует символьное представление полинома.
c=p.c;
if all(c==0)
   s='0';
else
   d=p.n;
   n=d+1;
   s=[];
   for k=1:n
       a=c(k);
       if a~=0;
          if ~isempty(s)
             if a>0
                s=[s ' + '];
             else
                s=[s ' - '];
                a=-a;
             end
          end
          if a~=1|d==0
             s=[s num2str(a)];
             if d>0
                s=[s '*'];
             end
          end
       if d>=2
          s=[s 'x^' int2str(d)];
       elseif d==1
          s=[s 'x'];
       end
   end
   d=d-1;
   end
end        % Завершение POLYNOM/CHAR.
```


Чтобы эта символьная строка выводилась на экран, нужно создать еще один M-файл по имени **display** в том же подкаталоге @POLYNOM.

*Метод **display** автоматически вызывается всегда, когда оказывается, что исполняемый оператор не заканчивается точкой с запятой.* Для многих классов метод **display** просто выводит на экран имя переменной, а затем использует преобразователь **char** для вывода символьного изображения объекта. Для рассматриваемого случая он может быть такого вида:

```
function display(p)
% POLYNOM/DISPLAY - вывод на экран polynom-объекта.
disp('');
disp([' ',inputname(1),' = ',char(p),';']);
disp('');
% Завершение POLYNOM/DISPLAY.
```

Проверим эффективность работы созданных трех M-файлов на простом примере. Сформируем вектор коэффициентов полинома:

```
» V = [0 0 0 -1 2 3 4 0 0 -6 -5 -7]
V =   0   0   0  -1   2   3   4   0   0  -6  -5  -7
```

Создадим на его основе полином-объект и сразу выведем его символьное изображение на экран. Для этого достаточно не поставить символ ";" после обращения к функции **polynom**:

```
» Pol1=polynom(V)
Pol1 = -1*x^8 + 2*x^7 + 3*x^6 + 4*x^5 - 6*x^2 - 5*x - 7
```

Создадим теперь полином-объект по заданным его корням и значению его старшего коэффициента:

```
» Pol2=polynom([1 2 3 4 5],-5)
Pol2 = -5*x^5 + 75*x^4 - 425*x^3 + 1125*x^2 - 1370*x + 600
```

Проверим корни созданного полинома, используя процедуру **roots** (см. далее):

```
» roots(Pol2)
ans =
  5.0000
  4.0000
  3.0000
  2.0000
  1.0000
```

Как видим, все созданные M-файлы работают нормально.

4.4. Создание методов нового класса

Весьма удобной в системе MatLAB является предоставляемая ею возможность создания процедуры, которая может быть выполнены не только стандартным способом обращения к ее имени, но и более простым путем использования известных знаков арифметических действий, операций сравнения, скобок и т.п.

С этим мы уже столкнулись в предыдущем разделе, где процедура **display** выполняется не только при явном обращении вида **display(x)**, но и неявно, когда некоторый оператор вычисляет величину x, а после него отсутствует символ ";".

Поэтому, если в любом новом классе объектов присутствует M-файл с именем **display**, он будет выполняться во всех случаях, когда очередной оператор, создающий объект этого класса, не заканчивается точкой с запятой.

Приведем перечень имен таких M-файлов, предусмотренных системой MatLAB, с указанием вида оператора их неявного вызова и кратким описанием особенностей их использования.

Таблица 4.1. Операторные функции

Операторный вызов	*Имя M-файла*	*Условное название*	*Особенности применения*
+ a	uplus(a)	Добавление знака плюс	Аргумент один; результат того же класса
- a	uminus(a)	Добавление знака минус	Аргумент один; результат того же класса
a + b	plus(a,b)	Сложение	Два аргумента; результат того же класса, что и аргументы
a - b	minus(a,b)	Вычитание	Два аргумента; результат того же класса, что и аргументы
a * b	mtimes(a,b)	Умножение	Два аргумента; результат того же класса, что и аргументы
a / b	mrdivide(a,b)	Правое деление	Два аргумента; результат того же класса, что и аргументы
a \ b	mldivide(a,b)	Левое деление	Два аргумента; результат того же класса, что и аргументы
a ^ b	mpower(a,b)	Степень	Два аргумента; результат того же класса, что и аргументы
a .* b	times(a,b)	Умножение поэлементное	Два аргумента; результат того же класса, что и аргументы
a ./ b	rdivide(a,b)	Правое деление поэлементное	Два аргумента; результат того же класса, что и аргументы
a .\ b	ldivide(a,b)	Левое деление поэлементное	Два аргумента; результат того же класса, что и аргументы
a .^ b	power(a,b)	Степень поэлементная	Два аргумента; результат того же класса, что и аргументы
a < b	lt(a,b)	Меньше	Два аргумента; результат — логическая величина
a > b	gt(a,b)	Больше	Два аргумента; результат — логическая величина
a <= b	le(a,b)	Меньше или равно	Два аргумента; результат — логическая величина
a >= b	ge(a,b)	Больше или равно	Два аргумента; результат — логическая величина
a == b	eq(a,b)	Равно	Два аргумента; результат — логическая величина
a '	ctranspose(a)	Транспонирование	Аргумент один; результат того же класса
a .'	transpose(a)	Транспонирование	Аргумент один; результат того же класса

Операторный вызов	*Имя М-файла*	*Условное название*	*Особенности применения*
a : d : b a : b	colon(a,d,b) colon(a,b)	Формирование вектора	Два или три аргумента; результат — вектор того же класса, что и аргументы
Вывести в командном окне	display(a)	Вывод на терминал	Аргумент один; результат — изображение на терминале символьного представления аргумента
[a b]	horzcat(a,b,...)	Объединение в строку	Два или больше аргументов; результат — вектор-строка из аргументов
[a; b]	vertcat(a,b,...)	Объединение в столбец	Два или более аргументов; результат — вектор-столбец из аргументов
a(s1,...sn)	subsref(a,s)	Индексная ссылка	
a(s1,...sn)=b	subsasgn(a,s,b)	Индексное выражение	
b(a)	subsindex(a,b)	Индекс подмассива	

Перечисленные процедуры в MatLAB могут быть переопределены под теми же именами во всех новообразованных подкаталогах новых классов. После этого обычные операторы арифметических действий и операций сравнения могут применяться и при оперировании объектами новых классов. Смысл этих операций, конечно, может значительно отличаться от обычного и будет определяться содержимым соответствующих М-файлов в подкаталогах классов.

Учитывая это, можно сделать вывод, что М-файлов с названиями, указанными в табл. 4.1, может быть много. MatLAB различает их по типу аргументов, указанных в перечне входных параметров.

Создадим, например, операцию (метод) *сложения* полиномов, используя операторную процедуру **plus**. Текст соответствующего М-файла для подкаталога @POLYNOM приведен ниже.

```
function r=plus(p,q)
% POLYNOM/PLUS — сложение полиномов r = p + q.
p = polynom(p);
q = polynom(q);
k = q.n — p.n;
r = polynom([zeros(1,k) p.c]+[zeros(1,-k) q.c]);
% Завершение POLYNOM/PLUS.
```

Сначала процедура преобразует оба аргумента в класс **polynom**. Это нужно для того, чтобы метод работал и тогда, когда один из аргументов задан как вектор или когда в качестве аргумента используется выражение типа p + r, где p — polynom-объект, а r — число. Затем процедура дополняет нулями векторы коэффициентов полиномов-слагаемых, если в этом возникает необходимость (при неодинаковых порядках складываемых полиномов). Фактически сложение сводится к сложению этих исправленных векторов коэффициентов. Заключительная операция — по полученному вектору полинома-суммы создается новый полином-объект при помощи конструктора полиномов. Проиллюстрируем работу этого метода на примере. Введем вектор коэффициентов первого полинома:

```
» V1 = [ 2 -3 0 6];
```

и создадим из него polynom-объект:

```
» P1 = polynom(V1)
P1 = 2*x^3 — 3*x^2 + 6
```

Аналогичным образом создадим второй полином:

```
» V2=[2 0 0 9 0 0 -3 +5];
» P2=polynom(V2)
P2 = 2*x^7 + 9*x^4 — 3*x + 5
```

Сложим эти полиномы:

```
» P1+P2
```

В результате получим:

```
ans = 2*x^7 + 9*x^4 + 2*x^3 — 3*x^2 — 3*x + 11
```

Подобные результаты получатся и в том случае, когда один из аргументов ошибочно представлен вектором:

```
» P1+V2
ans = 2*x^7 + 9*x^4 + 2*x^3 — 3*x^2 — 3*x + 11
» V1+P2
ans = 2*x^7 + 9*x^4 + 2*x^3 — 3*x^2 — 3*x + 11
```

Однако если оба аргумента ошибочно представлены векторами, система сразу же отреагирует на это, обнаружив ошибку:

```
» V1+V2
_??? Error using ( +
Matrix dimensions must agree.
```

В этом случае система использует уже не M-файл **plus** из подкаталога @POLYNOM, а встроенную аналогичную процедуру для векторов. А такая процедура осуществляет сложение векторов лишь при условии их одинаковой длины. Поэтому и появляется ошибка.

Аналогичной является процедура вычитания polynom-объектов:

```
function r=minus(p,q)
% POLYNOM/MINUS - вычитание полиномов r = p - q
p = polynom(p);
q = polynom(q);
k = q.n - p.n;
r = polynom([zeros(1,k) p.c]-[zeros(1,-k) q.c]);
% Завершение POLYNOM/MINUS.
```

Проверим ее работу на примере:

```
» P1-P2
ans = -2*x^7 - 9*x^4 + 2*x^3 - 3*x^2 + 3*x + 1;
```

Создадим еще такие методы класса **polynom**:

- метод **double**, осуществляющий обратную по отношению к созданию polynom-объекта операцию, — по заданному polynom-объекту определяет вектор его коэффициентов и порядок:

```
function [v,n]=double(p)
% POLYNOM/DOUBLE - преобразование polynom-объекта
% в вектор его коэффициентов.
% v=DOUBLE(p) преобразует polynom-объект p в вектор v,
% содержащий коэффициенты полиному в порядке уменьшения
% степени аргумента.
% Обращение [v,n]=DOUBLE(p) позволяет получить также
% значение n порядка этого полинома.
v= p.c;
n= p.n;
% Завершение POLYNON/DOUBLE.
```

- метод **diff**, который создает polynom-объект, являющийся производной от заданного полинома:

```
function q = diff(p)
% POLYNOM/DIFF - формирует полином-производную q
% от заданного полинома p.
p = polynom(p);
d = p.n;
q = polynom(p.c(1:d).*(d:-1:1));
% Завершение POLYNOM/DIFF.
```

- метод **mtimes**; он создает polynom-объект, являющийся произведением двух заданных полиномов:

```
function r = mtimes(p,q)
% POLYNOM/MTIMES - произведение полиномов: r = p * q.
p = polynom(p);
q = polynom(q);
r = polynom(conv(p.c,q.c));
% Завершение POLYNOM/MTIMES.
```

- метод **mrdivide** создает два polynom-объекта, один из которых является частным от деления первого из указанных полиномов на второй, а второй — остатком такого деления:

```
function rez = mrdivide(p,q)
% POLYNOM/MRDIVIDE - деление полиномов: r = p / q.
p = polynom(p);
q = polynom(q);
[rr,ro]=deconv(p.c,q.c);
rez{1}=polynom(rr)
rez{2}=polynom(ro)
% Завершение POLYNOM/MRDIVIDE.
```

- метод **roots** создает вектор корней заданного полинома:

```
function r = roots(p)
% POLYNOM/ROOTS - вычисляет вектор корней полинома p.
p = polynom(p);
r = roots(p.c);
% Завершение POLYNOM/ROOTS.
```

- метод **polyval** вычисляет вектор значений заданного полинома по заданному вектору значений его аргумента:

```
function y = polyval(p,x)
% POLYNOM/POLYVAL - вычисляет значение полинома p
% по заданному значению аргумента x.
p = polynom(p);
y =0;
for a = p.c
  y = y.*x + a;
end
% Завершение POLYNOM/POLYVAL.
```

- метод **plot** строит график зависимости значений заданного полинома в диапазоне значений его аргумента, содержащем все его корни:

```
function plot(p)
% POLYNOM/PLOT - построение графика полинома p.
p=polynom(p);
r= max(abs(roots(p.c)));
x=(-1.1 : 0.01 : 1.1)*r;
y= polyval(p.c,x);
```

```
plot(x,y); grid;
title(char(p));
xlabel('X');
% Завершение POLYNOM/PLOT.
```

- метод **rdivide(p,xr)** осуществляет деление полинома p на число xr:

```
function r = rdivide(p,xr)
% POLYNOM/RDIVIDE — правое деление полинома
% на число: r = p / xr.
p = polynom(p);
r.c = p.c/xr;
r = polynom(r.c);
% Завершение POLYNOM/RDIVIDE.
```

Проверим действие некоторых из созданных методов:

```
» Pol = polynom([1 2 3 4 5])
  Pol = x^4 + 2*x^3 + 3*x^2 + 4*x + 5;
» v = roots(Pol)
v =
  0.2878 + 1.4161i
  0.2878 - 1.4161i
 -1.2878 + 0.8579i
 -1.2878 - 0.8579i
» plot(Pol)
```

Результат представлен на рис. 4.2.

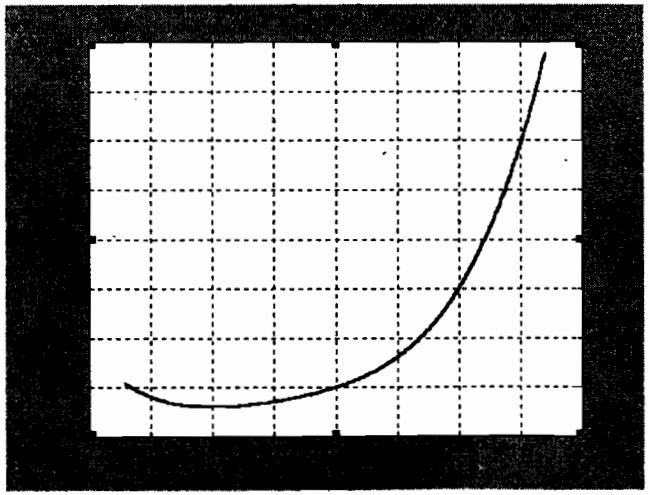

Рис. 4.2

Чтобы получить перечень всех созданных методов определенного класса, следует воспользоваться такой командой:

```
methods <имя класса>
```

Например:

```
» methods polynom
Methods for class polynom:
char    display minus  mtimes  plus    polynom0 rdivide times
diff    double  mrdivide plot   polynom polyval roots
```

Теперь мы имеем мощный вычислительный аппарат для работы с полиномами. Развивая его далее, можно создать новый класс более сложных объектов — класс рациональных передаточных функций, которые представляют собой конструкцию в виде дроби, числителем и знаменателем которой являются полиномы. Для этого класса также можно определить важные для инженера операции сложения передаточных функций (которое соответствует параллельному соединению звеньев с заданными передаточными функциями), умножения (ему соответствует последовательное соединение) и многие другие, отвечающие определенным типам соединений звеньев. Примерно на такой основе создан, например, класс TF (Transfer Function), используемый в пакете CONTROL.

Глава 5

Цифровая обработка сигналов (пакет Signal Processing Toolbox)

Цифровая обработка сигналов традиционно включает в себя создание средств численного преобразования массива заданного (измеренного в дискретные моменты времени) процесса изменения некоторой непрерывной физической величины с целью извлечения из него полезной информации о другой физической величине, содержащейся в измеренном сигнале.

Общая схема образования измеряемого сигнала и процесса его преобразований в целях получения информации о величине, которая должна быть измерена, представлена на рис. 5.1.

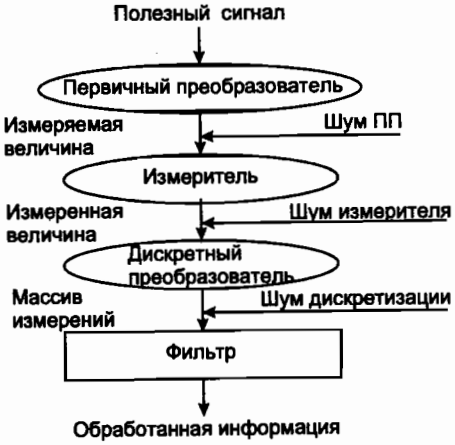

Рис. 5.1

Физическая величина, являющаяся полезной (несущей в себе необходимую информацию), редко имеет такую физическую форму, что может быть непосредственно измеренной. Обычно она представляет лишь некоторую составляющую (сторону, часть, черту) некоторой другой физической величины, которая может быть непосредственно измерена. Связь между этими двумя величинами обозначим введением звена, которое назовем *первичным преобразователем* (ПП). Обычно закон преобразования известен заранее, иначе восстановить информационную составляющую в дальнейшем было бы невозможно. Первичный преобразователь вносит зависимость сигнала, который может быть измерен, от некоторых других физических величин. Вследствие этого выходная его величина содержит, кроме полезной информационной составляющей, другие, вредные составляющие или черты, искажающие полезную информацию. И хотя зависимость выхода ПП от этих других величин также известна, однако вследствие неконтролируемого возможного изменения последних со временем, часто трудно спрогнозировать их влияние на искажение полезной составляющей. Назовем вносимую ПП вредную составляющую *шумом ПП*.

Пусть образованная таким образом непосредственно измеряемая величина измеряется некоторым измерителем. Любой реальный измеритель вносит собственные искажения в измеряемую величину и дополнительные зависимости от некоторых других физических величин, не являющихся объектом измерения. Назовем эти искажения *шумами измерителя*. Не ограничивая общности, будем полагать, что выходной величиной измерителя является электрический сигнал (*измеренная величина*), который можно в дальнейшем довольно просто преобразовывать электрическими устройствами.

Для осуществления цифровой обработки измеренная величина должна быть преобразована в дискретную форму при помощи специального устройства, которое содержит *экстраполятор* и *аналого-цифровой преобразователь* (АЦП). Первый производит фиксацию отдельного текущего значения измеренной величины в отдельные моменты времени через определенный постоянный промежуток времени, называемый *дискретом времени*. Второй переводит это значение в цифровую форму, которая позволяет в дальнейшем осуществлять преобразования с помощью цифровых ЭВМ. Хотя оба устройства могут вносить при таких преобразованиях собственные искажения в выходной (дискретный) сигнал, однако ими обычно пренебрегают, так как в большинстве случаев эти дополнительные искажения значительно меньше шумов ПП и измерителя.

Для того чтобы на основе имеющегося дискретизированного сигнала получить полезный сигнал, нужно рассчитать и создать устройство (программу для ЭВМ), которое осуществляло бы такие преобразования входного дискретного во времени сигнала, чтобы на его выходе искажения, внесенные шумами ПП и измерителя были минимизированы в некотором смысле. Это устройство называют фильтром.

В общем случае создание (проектирование) фильтра является задачей неопределенной, которая конкретизируется лишь на основе предварительно полученных знаний о закономерности образования измеряемой величины (модели ПП), о модели образования измеренной величины из измеряемой (модели измерителя), о характеристиках изменения во времени вредных физических величин, влияющих на образование измеряемой и измеренной величин, и закономерностей их влияния на искажение полезной информации.

Так как модели ПП и измерителя могут быть весьма разнообразными, традиционно задачу фильтрации решают только для некоторых наиболее распространенных на практике, чаще всего для линейных моделей.

В общем случае процесс создания фильтра включает следующие этапы:

- на основе априорной информации о моделях ПП и измерителя и о характеристиках шумов, а также о задачах, которые должен решать фильтр, выбирается некоторый тип фильтра из известных, с разработанной теорией проектирования;
- на основе конкретных числовых данных рассчитываются числовые характеристики выбранного типа фильтра (создается конкретный фильтр);
- проверяется эффективность выполнения разработанным фильтром поставленной перед ним задачи; для этого необходимо сымитировать на ЭВМ дискретный сигнал, содержащий полезную (информационную) составляющую с наложенными на нее предусмотренными шумами ПП и измерителя, пропустить его через построенный фильтр и сравнить полученный на выходе сигнал с известной (в данном случае) полезной его составляющей; разность между ними будет характеризовать погрешности измерения на выходе фильтра;
- так как в реальных условиях некоторые характеристики шумов могут отличаться от принятых при проектировании (создании фильтра), будут полезны *испытания эффективности работы фильтра* в условиях более приближенных к реальным, нежели принятые при проектировании.

Пакет Signal Processing Toolbox (в дальнейшем сокращенно Signal) предназначен для осуществления операций по трем последним из указанных этапов. Он позволяет проектировать (рассчитывать конкретные числовые характеристики) цифровые и аналоговые фильтры по требуемым амплитудно- и фазо-частотным их характеристикам, формировать последовательности типовых временных сигналов и обрабатывать их спроектированными фильтрами. В пакет входят процедуры, осуществляющие преобразования Фурье, Гильберта, а также статистический анализ. Пакет позволяет рассчитывать корреляционные функции, спектральную плотность мощности сигнала, оценивать параметры фильтров по измеренным отсчетам входной и выходной последовательностей.

5.1. Формирование типовых процессов

5.1.1. Формирование отдельных импульсных процессов

В пакете Signal предусмотрено несколько процедур, образующих последовательности данных, представляющие некоторые одиночные импульсные процессы типовых форм.

Процедура **rectpuls** обеспечивает формирование одиночного импульса прямоугольной формы. Обращение вида:

```
y = rectpuls(t,w)
```

позволяет образовать вектор *y* значений сигнала такого импульса единичной амплитуды, шириной *w*, центрированного относительно t=0 по заданному вектору *t* моментов времени. Если ширина импульса w не указана, ее значение по умолчанию принимается равным единице. На рис. 5.2 приведен результат образования процесса, состоящего из трех последовательных прямоугольных импульсов разной высоты и ширины, по такой последовательности команд:

```
» t = 0 : 0.01 : 10;
» y = 0.75*rectpuls(t-3, 2)+0.5*rectpuls(t-8, 0.4)...
  +1.35*rectpuls(t-5, 0.8);
» plot(t,y), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Пример применения процедуры RECTPULS')
» xlabel('Время ( с )')
» ylabel('Выходной процесс Y(t)')
```

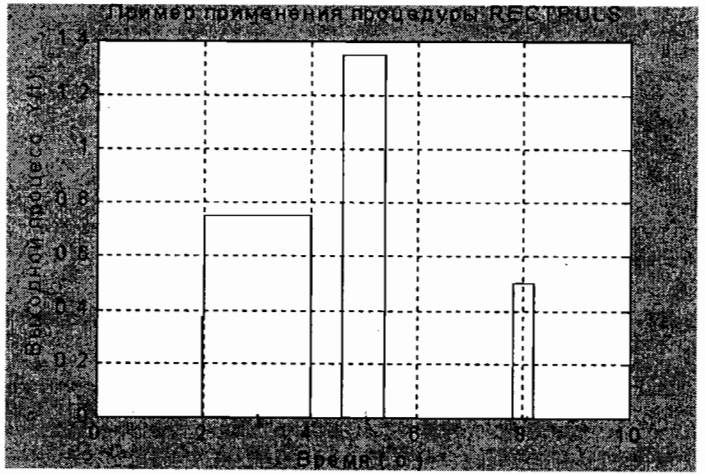

Рис. 5.2

Формирование импульса треугольной формы единичной амплитуды можно осуществить при помощи процедуры **tripuls**, обращение к которой имеет вид:

```
y = tripuls(t,w,s)
```

Аргументы y, t и w имеют тот же смысл. Аргумент s (-1<s<1) определяет наклон треугольника. Если s=0 или не указан, треугольный импульс имеет симметричную форму. Приведем пример (результат представлен на рис. 5.3):

```
» t = 0 : 0.01 : 10;
» y = 0.75*tripuls(t-1, 0.5)+0.5*tripuls(t-5, 0.5, -1)...
  +1.35*tripuls(t-3, 0.8,1);
» plot(t,y), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Пример применения процедуры TRIPULS')
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

Для формирования импульса, являющегося синусоидой, модулированной функцией Гаусса, используется процедура **gauspuls**. Если обратиться к ней по форме:

```
y = gauspuls(t,fc,bw)
```

то она создает вектор значений указанного сигнала с единичной амплитудой, с синусоидой, изменяющейся с частотой fc Гц, и с шириной bw полосы частот сигнала.

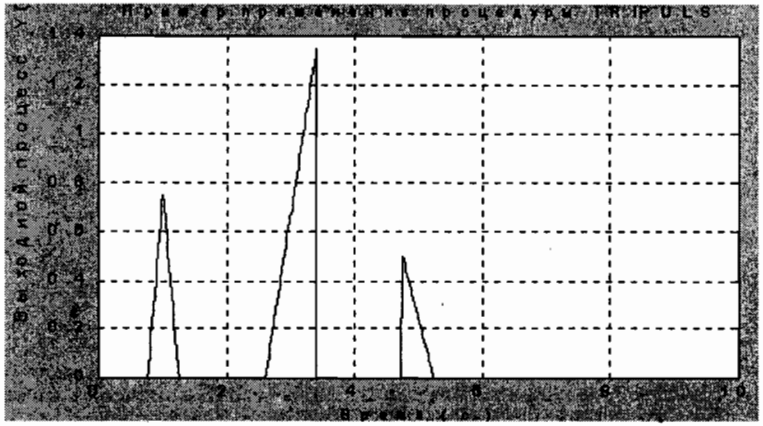

Рис. 5.3

В случае, когда последние два аргумента не указаны, они по умолчанию приобретают значения 1000 Гц и 0.5 соответственно. Приведем пример создания одиночного гауссового импульса (результат приведен на рис. 5.4):

```
» t = 0 : 0.01 : 10;
» y = 0.75*gauspuls(t-3, 1,0.5);
» plot(t,y), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
```



```
» title('Пример применения процедуры GAUSPULS')
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

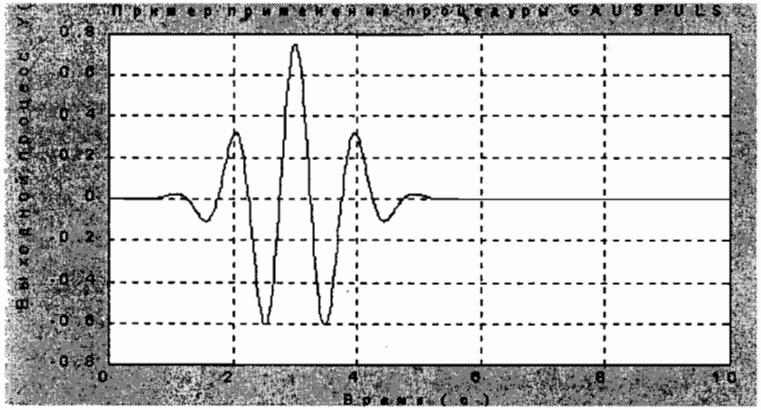

Рис. 5.4

Наконец, рассмотрим процедуру **sinc**, формирующую вектор значений функции sinc(t), которая определяется формулами:

$$\mathrm{sinc}(t) = \begin{cases} 1 & t = 0 \\ \dfrac{\sin(\pi t)}{\pi t} & t \neq 0 \end{cases}$$

Эта функция является обратным преобразованием Фурье прямоугольного импульса шириной 2π и высотой 1:

$$\mathrm{sinc}(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega t} d\omega$$

Приведем пример ее применения:

```
» t=0 : 0.01 : 50;
» y1=0.7*sinc(pi*(t-25)/5);
» plot(t,y1), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Функция SINC  Y(t) = 0.7* SINC(pi*(t-25)/5)')
» xlabel('Время ( с )')
» ylabel('Выходной процесс Y(t)')
```

Результат изображен на рис. 5.5.

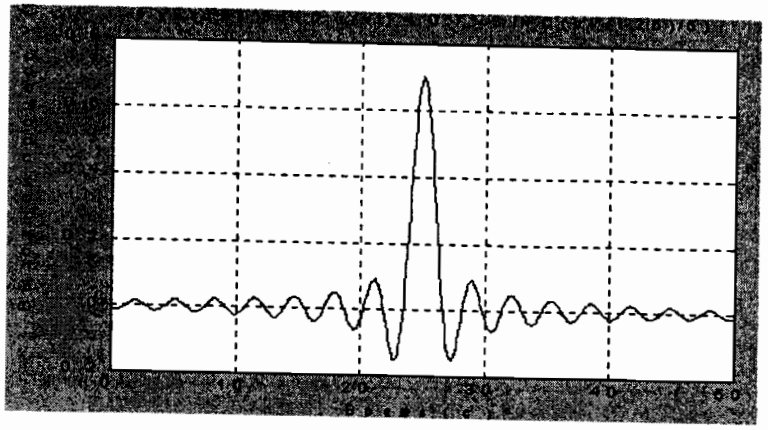

Рис. 5.5

5.1.2. Формирование колебаний

Формирование колебаний, состоящих из конечного числа гармонических составляющих (т.е. так называемых *полигармонических колебаний*), можно осуществить при помощи обычных процедур **sin(x)** и **cos(x)**. Рассмотрим пример (рис. 5.6):

```
» t=0 : 0.01 : 50;
» y1=0.7*sin(pi*t/5);
» plot(t,y1), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Гармонические колебания  Y(t) = 0.7* SIN(pi*t/5)')
» xlabel('Время ( c )')
» ylabel('Выходной процесс Y(t)')
```

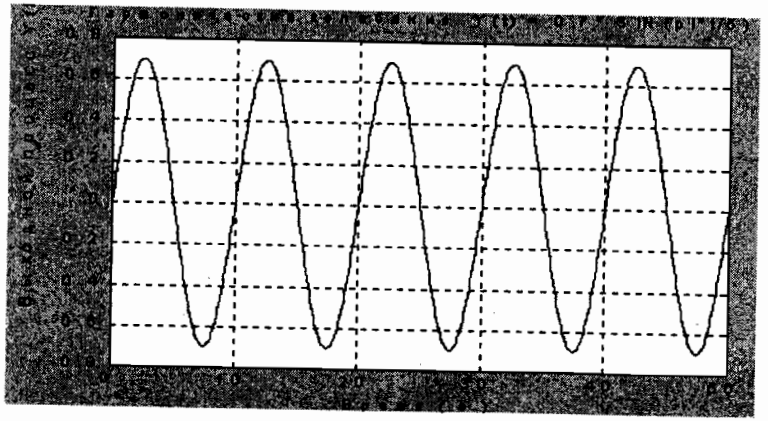

Рис. 5.6

Процесс, являющийся последовательностью прямоугольных импульсов с периодом 2π для заданной в векторе *t* последовательности отсчетов времени, генерируется при помощи процедуры **square**. Обращение к ней происходит по форме:

```
y = square(t,duty)
```

где аргумент duty определяет длительность положительной полуволны в процентах от периода волны. Например (результат приведен на рис. 5.7):

```
» y=0.7*square(pi*t/5, 40);
» plot(t,y), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Прямоугольные волны  Y(t) = 0.7* SQUARE(pi*t/5, 40)')
» xlabel('Время ( c )')
» ylabel('Выходной процесс  Y(t)')
```

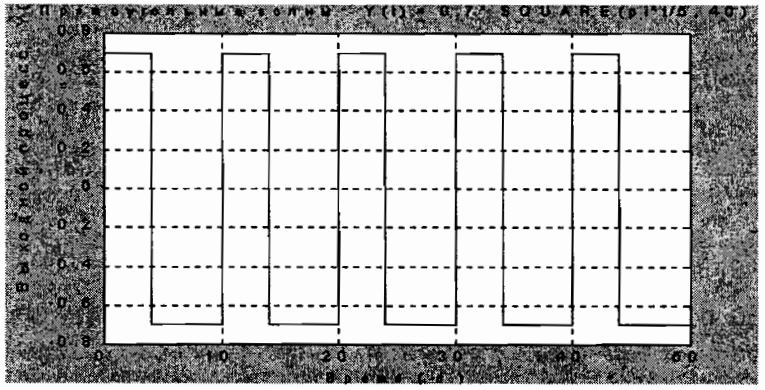

Рис. 5.7

Аналогично, генерирование пилообразных и треугольных колебаний можно осуществлять процедурой **sawtooth**. Если обратиться к ней так:

```
y = sawtooth(t,width)
```

то в векторе *y* формируются значения сигнала, представляющего собой пилообразные волны с периодом 2π в моменты времени, которые задаются вектором *t*. При этом параметр width определяет часть периода, в которой сигнал увеличивается. Ниже приведен пример применения этой процедуры:

```
» y=0.7*sawtooth(pi*t/5, 0.5);
» plot(t,y), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Треугольные волны  Y(t) = 0.7* SAWTOOTH(pi*t/5, 0.5)')
```

```
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

В результате получаем процесс, изображенный на рис. 5.8.

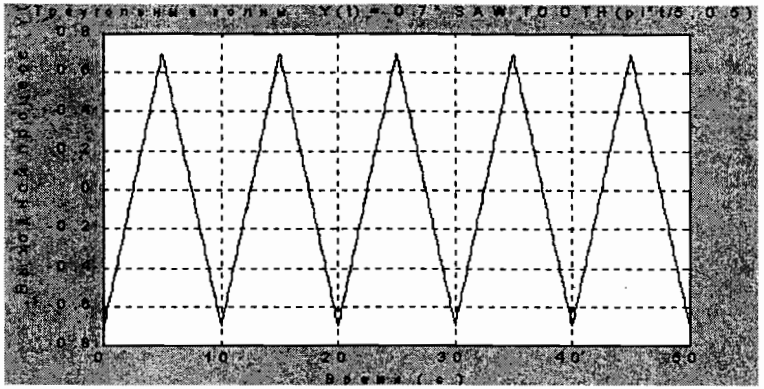

Рис. 5.8

Процедура **pulstran** позволяет формировать колебания, являющиеся последовательностью либо прямоугольных, либо треугольных, либо гауссовых импульсов. Обращение к ней имеет вид:

```
y = pulstran(t,d,'func',p1,p2,...)
```

Здесь d определяет вектор значений тех моментов времени, где должны быть центры соответствующих импульсов; параметр func определяет форму импульсов и может иметь одно из следующих значений: rectpuls (для прямоугольного импульса), tripuls (для треугольного импульса) и gauspuls (для гауссового импульса); параметры p1, p2,... определяют необходимые параметры импульса в соответствии с формой обращения к процедуре, определяющей этот импульс.

Ниже приведены три примера применения процедуры **pulstran** для разных форм импульсов-составляющих.

- Для последовательности треугольных импульсов:

```
» t=0 : 0.01 : 50
» d=[0 : 50/5 : 50]';
» y=0.7*pulstran(t, d,'tripuls',5);
» plot(t,y), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Y(t) = 0.7*PULSTRAN(t,d,''tripuls'',5)' )
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

Результат представлен на рис. 5.9.

- Для последовательности прямоугольных импульсов:

```
» t=0 : 0.01 : 50
» d=[0 : 50/5 : 50]';
» y=0.75*pulstran(t, d,'rectpuls',3);
» plot(t,y), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Y(t) = 0.75*PULSTRAN(t,d,''rectpuls'',3)' )
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

Результат приведен на рис. 5.10.

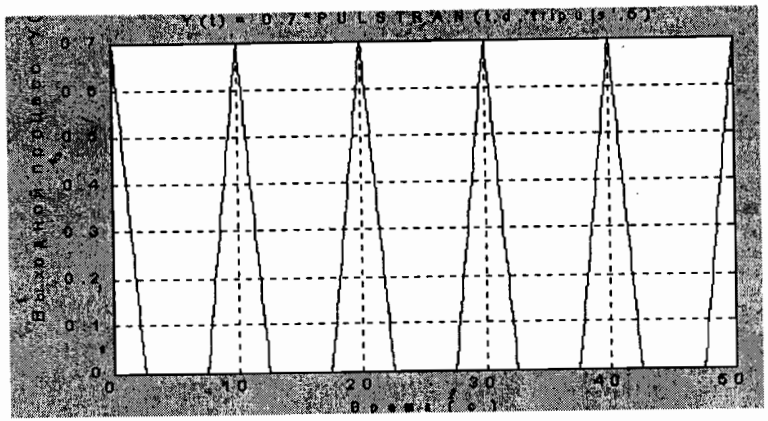

Рис. 5.9

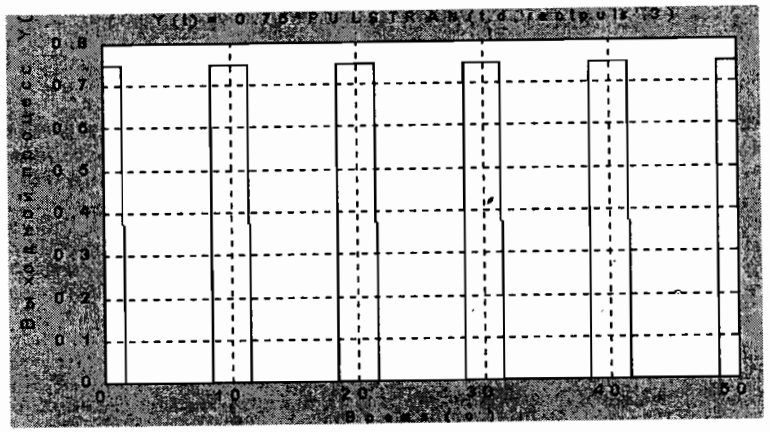

Рис. 5.10

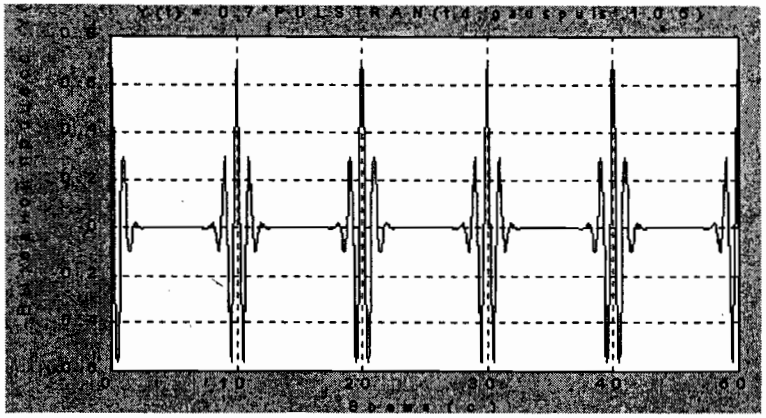

Рис. 5.11

- Для последовательности гауссовых импульсов:

```
» t=0 : 0.01 : 50
» d=[0 : 50/5 : 50]';
» y=0.7*pulstran(t, d,'gauspuls',1,0.5);
» plot(t,y), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Y(t) = 0.7*PULSTRAN(t,d,''gauspuls'',1,0.5)' )
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

Результат приведен на рис. 5.11.

Рассмотрим теперь процедуру **chirp**, формирующую косинусоиду, частота изменения которой линейно изменяется со временем. Общая форма обращения к этой процедуре такова:

```
y = chirp(t,F0,t1,F1)
```

где F0 — значение частоты в герцах при t=0, t1 — некоторое заданное значение момента времени; F1 — значение частоты (в герцах) изменения косинусоиды в момент времени t1. Если три последних аргумента не указаны, то по умолчанию им придаются такие значения: F0=0, t1=1, F1=100. Пример:

```
» t = 0 : 0.001 : 1;
» y = 0.75*chirp(t);
» plot(t,y), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Пример применения процедуры CHIRP')
» xlabel('Время ( с )')
» ylabel('Выходной процесс  Y(t)')
```

Результат показан на рис. 5.12.

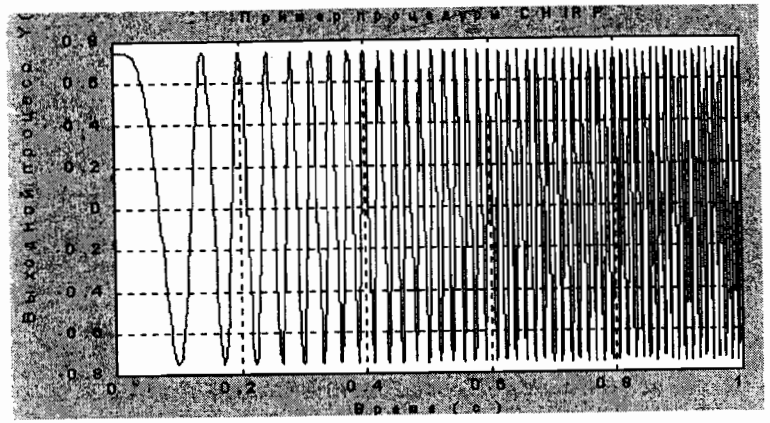

Рис. 5.12

Еще одна процедура — **diric** — формирует массив значений так называемой *функции Дирихле*, определяемой соотношениями:

$$diric(t) = \begin{cases} -1^{k(n-1)} & t = 2\pi k, \quad k = 0,\pm1,\pm2,\ldots \\ \dfrac{\sin(nt/2)}{n\cdot\sin(t/2)} & \text{при других } t \end{cases}$$

Функция Дирихле является периодической. При нечетных n период равен 2π, при четных — 4π. Максимальное значение ее равно 1, минимальное — -1. Параметр n должен быть целым положительным числом. Обращение к функции имеет вид:

```
y = diric(t,n)
```

Ниже приведены операторы, которые иллюстрируют использование процедуры **diric** и выводят график функции Дирихле:

```
» t=0 : 0.01 : 50;
» y1=0.7*diric(pi*t/5, 3);
» plot(t,y1), grid,set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Функция Дирихле  Y(t) = 0.7* DIRIC(pi*t/5, 3)')
» xlabel('Время ( с )')
» ylabel('Выходной процесс Y(t)')
```

Результаты для n=3, 4 и 5 представлены на рис. 5.13 — 5.15.

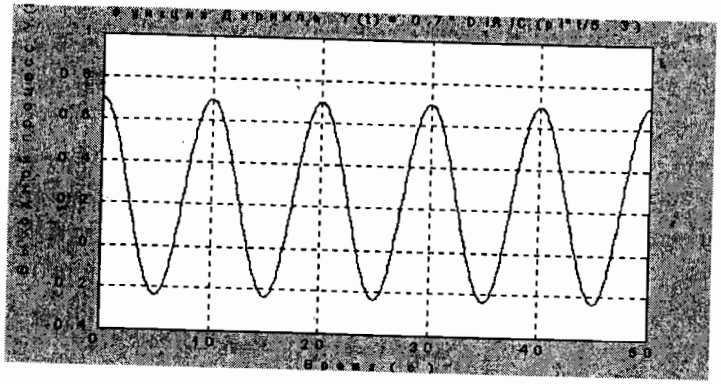

Рис. 5.13

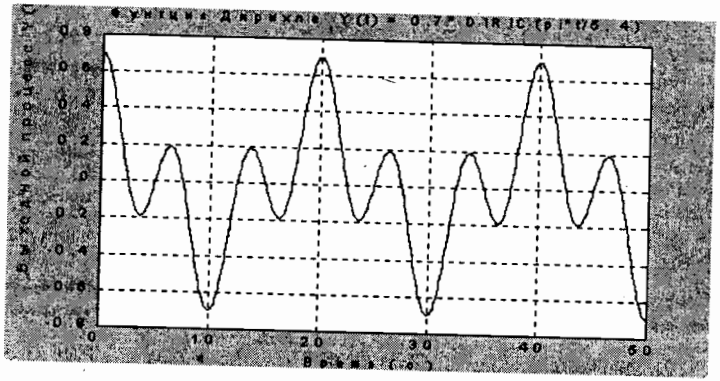

Рис. 5.14

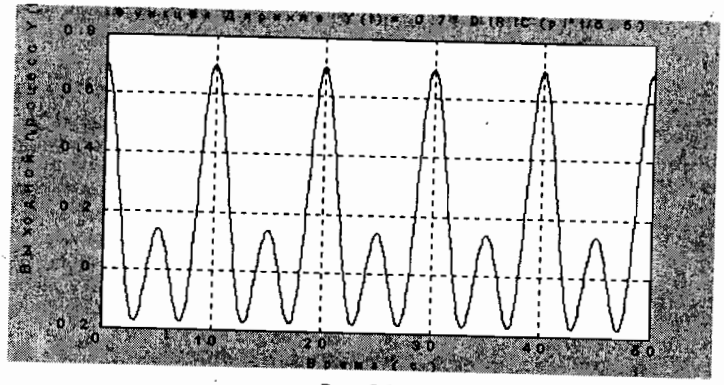

Рис. 5.15

5.2. Общие средства фильтрации. Формирование случайных процессов

5.2.1. Общие основы линейной фильтрации

Рассмотрим основы линейной фильтрации на примере линейного стационарного фильтра, который в непрерывном времени описывается дифференциальным уравнением второго порядка:

$$\ddot{y} + 2\zeta\omega_0 \cdot \dot{y} + \omega_0^2 \cdot y = A \cdot x \qquad (5.1)$$

где x — заданный процесс, подаваемый на вход этого фильтра второго порядка; y — процесс, получаемый на выходе фильтра; ω0 — частота собственных колебаний фильтра, а ζ — относительный коэффициент затухания этого фильтра.

Передаточная функция фильтра, очевидно, имеет вид:

$$W(s) = \frac{y(s)}{x(s)} = \frac{A}{s^2 + 2\zeta\omega_0 \cdot s + \omega_0^2} \qquad (5.2)$$

Для контроля и графического представления передаточной функции любого линейного динамического звена удобно использовать процедуру **freqs**. В общем случае обращение к ней имеет вид:

```
h = freqs(b,a,w)
```

При этом процедура создает вектор *h* комплексных значений частотной характеристики *W(jω)* по передаточной функции *W(s)* звена, заданной векторами коэффициентов ее числителя (*b*) и знаменателя (*a*), а также по заданному вектору *w* частоты ω. Если аргумент *w* не указан, процедура автоматически выбирает 200 отсчетов частоты, для которых вычисляется частотная характеристика.

Если не указана выходная величина, т.е. обращение имеет вид:

```
freqs(b,a,w)
```

процедура выводит в текущее графическое окно два графика — АЧХ и ФЧХ.

Приведем пример. Пусть для передаточной функции (5.2) выбраны такие значения параметров:

$$A=1; \quad \zeta=0.05; \quad T_0=2\pi/\omega_0=1$$

Вычислим значения коэффициентов числителя и знаменателя и выведем графики АЧХ и ФЧХ:

```
» T0=1;  dz=0.05;
» om0=2*pi/T0;  A=1;
» a1(1)=1;  a1(2)=2*dz*om0;  a1(3)= om0^2;  b1(1)=A;
» freqs(b1,a1)
```

Результат показан на рис. 5.16.

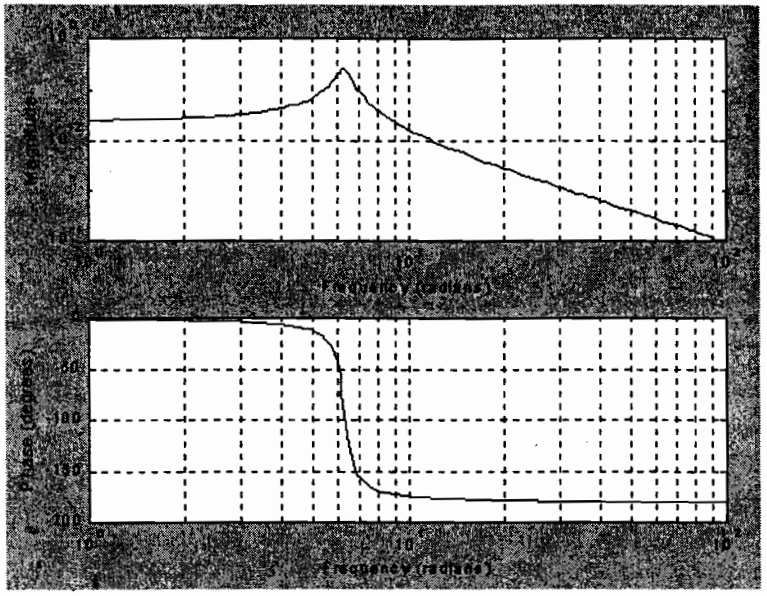

Рис. 5.16

Допустим, заданный процесс $x(t)$ представлен в виде отдельных его значений в дискретные моменты времени, которые разделены одинаковыми промежутками Ts времени (дискретом времени). Обозначим через $x(k)$ значение процесса в момент времени $t = k \cdot Ts$, где k — номер измерения с начала процесса.

Запишем уравнение (5.1) через конечные разности процессов x и y, учитывая, что конечно-разностным эквивалентом производной $\dot{y}$ является конечная разность:

$$\frac{\Delta y(k)}{Ts} = \frac{y(k) - y(k-1)}{Ts}$$

а эквивалентом производной второго порядка $\ddot{y}$ является конечная разность второго порядка:

$$\frac{\Delta^2 y(k)}{Ts^2} = \frac{\Delta y(k) - \Delta y(k-1)}{Ts^2} = \frac{y(k) - 2y(k-1) + y(k-2)}{Ts^2}$$

Тогда разностное уравнение:

$$(1 + 2\varsigma\omega_0 \cdot Ts + \omega_0^2 \cdot Ts^2) \cdot y(k) - 2(1 + \varsigma\omega_0 \cdot Ts) \cdot y(k-1) + y(k-2) = \\ = A \cdot Ts^2 \cdot x(k) \qquad (5.3)$$

является дискретным аналогом дифференциального уравнения (5.1).

Применяя к полученному уравнению Z-преобразование, получим:

$$y(z) \cdot [a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}] = A \cdot Ts^2 \cdot x(z), \text{ где} \qquad (5.4)$$

$$a_0 = 1 + 2\varsigma\omega_0 \cdot Ts + \omega_0^2 \cdot Ts^2; \\ a_1 = -2(1 + \varsigma\omega_0 \cdot Ts); \qquad (5.5) \\ a_2 = 1$$

Дискретная передаточная функция фильтра определяется из уравнения (5.4):

$$G(z) = \frac{y(z)}{x(z)} = \frac{A \cdot Ts^2}{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}} \qquad (5.6)$$

Таким образом, цифровым аналогом ранее введенного колебательного звена является цифровой фильтр с коэффициентами числителя и знаменателя, рассчитанными по формулам (5.4) и (5.5):

```
» T0=1;  dz=0.05;  Ts=0.01;
» om0=2*pi/T0;  A=1;  oms=om0*Ts;
» a(1)= 1+2*dz*oms+oms^2;
» a(2)= -2*(1+dz*oms);
» a(3)=1;
» b(1)=A*Ts*Ts*(2*dz*om0^2);
```

Чтобы получить частотную дискретную характеристику $G(e^{j\omega})$ по дискретной передаточной функции $G(z)$, заданной векторами значений ее числителя b и знаменателя a, удобно использовать процедуру **freqz**, обращение к которой аналогично обращению к процедуре **freqs**:

```
freqz(b,a)
```

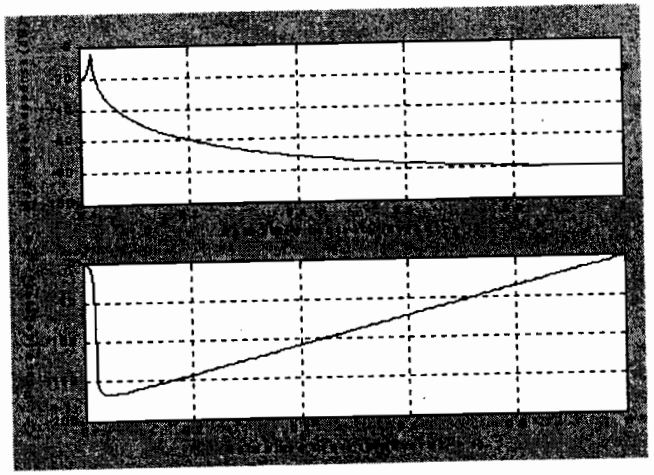

Рис. 5.17

В системе MatLAB *фильтрация*, т.е. преобразование заданного сигнала с помощью линейного фильтра, описываемого дискретной передаточной функцией вида:

$$G(z) = \frac{y(z)}{x(z)} = \frac{b_0 + b_1 \cdot z^{-1} + \ldots + b_m \cdot z^{-m}}{a_0 + a_1 \cdot z^{-1} + \ldots + a_n \cdot z^{-n}} \qquad (5.7)$$

осуществляется процедурой **filter** следующим образом:

```
y = filter(b,a,x)
```

где x — заданный вектор значений входного сигнала; y — вектор значений выходного сигнала фильтра, получаемого вследствие фильтрации; b — вектор коэффициентов числителя дискретной передаточной функции (5.7) фильтра; a — вектор коэффициентов знаменателя этой функции.

В качестве примера рассмотрим такую задачу. Пусть требуется получить достаточно верную информацию о некотором полезном сигнале, имеющем синусоидальную форму с известным периодом T1=1 с и амплитудой A1=0.75. Сформируем этот сигнал как вектор его значений в дискретные моменты времени с дискретом Ts=0.001 с:

```
» Ts=0.001;
» t=0 : Ts : 20;
» A1=0.75;     T1=1;
» Yp=A1*sin(2*pi*t/T1);
» plot(t(10002:end),Yp(10002:end)),grid,...
  set(gca, 'FontName','ArialCyr','FontSize',16)
» title('"Полезный" процесс');
» xlabel('Время (c)');
» ylabel('Yp(t)')
```

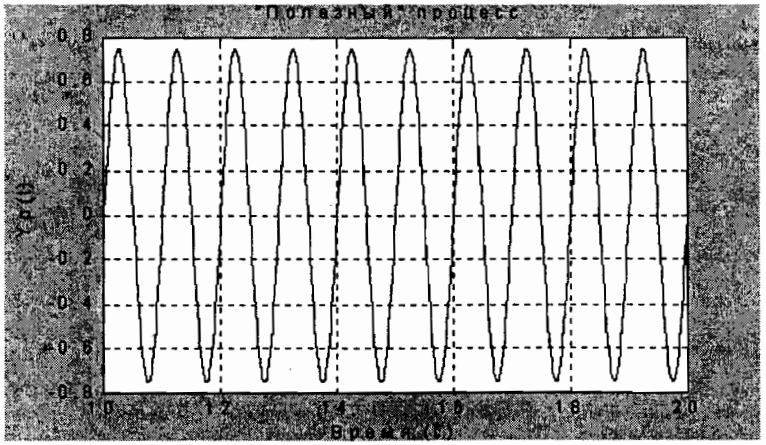

Рис. 5.18

Допустим, что вследствие прохождения через ПП (первичный преобразователь) к полезному сигналу добавился шум ПП в виде более высокочастотной синусоиды с периодом T_2=0.2 с и амплитудой A_2=5, а в результате измерения к нему еще добавился белый гауссовый шум измерителя с интенсивностью $A_ш$=5. В результате создался такой измеренный сигнал x(t) (рис. 5.19):

```
» T2=0.2;  A2=10;  eps=pi/4;
» Ash=5;
» x=A1.*sin(2*pi*t./T1)+A2.*sin(2*pi.*t./T2+eps)+...
  Ash*randn(1,length(t));
» plot(t(10002:end),x(10002:end)),grid,set(gca,'FontName',...
  'ArialCyr','FontSize',16),
» title('Входной процесс ');
» xlabel('Время (c)');
» ylabel('X(t)')
```

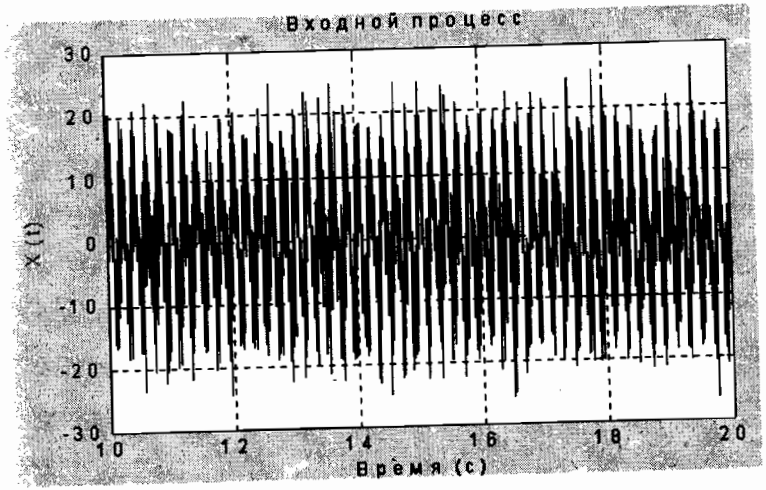

Рис. 5.19

Требуется так обработать измеренные данные x, чтобы восстановить по ним полезный процесс как можно точнее.

Так как частота полезного сигнала заранее известна, восстановление его можно осуществить при помощи резонансного фильтра отмеченного выше вида. При этом необходимо создать такой фильтр, чтобы период его собственных колебаний $T_ф$ был равен периоду колебаний полезного сигнала ($T_ф = T_1$). Чтобы после прохождения через такой фильтр амплитуда восстановленного сигнала совпадала с амплитудой полезного сигнала, нужно входной сигнал фильтра домножить на постоянную величину $2\varsigma\omega_0^2$ (поскольку при резонансе

амплитуда выходного сигнала "уменьшается" именно во столько раз по сравнению с амплитудой входного сигнала). Сформируем фильтр, описанный выше:

```
» T1=1;  Tf=T1;  dz=0.05;
» om0=2*pi/Tf;  A=1;  oms=om0*Ts;
» a(1)= 1+2*dz*oms+oms^2;
» a(2)= -2*(1+dz*oms);
» a(3)=1;
» b(1)=A*Ts*Ts*(2*dz*om0^2);
```

и пропустим сформированный процесс через него:

```
» y=filter(b,a,x)
» plot(t(10002:end),y(10002:end),t(10002:end), Yp(10002:end)),grid,
» set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Процесс на выходе фильтра (Tf=1; dz=0.05)');
» xlabel('Время (c)');
» ylabel('Y(t)')
```

В результате получим восстановленный процесс, изображенный на рис. 5.20. Для сравнения на этом же графике изображен восстанавливаемый процесс. Как видим, созданный фильтр достаточно хорошо восстанавливает полезный сигнал (сравните с рис. 5.19).

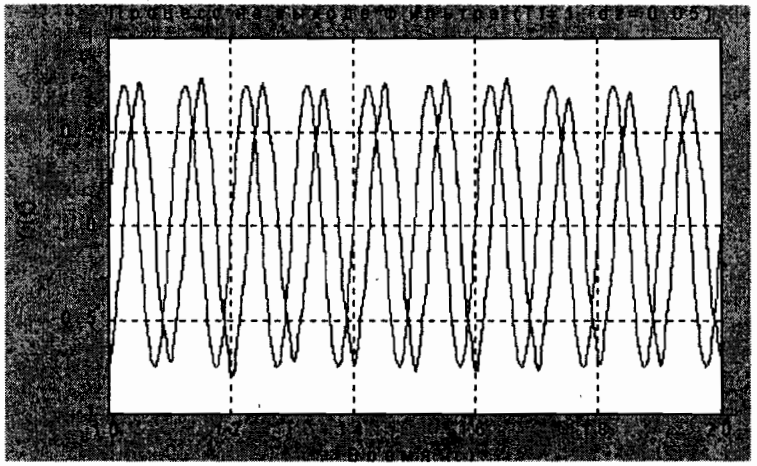

Рис. 5.20

Однако более точному восстановлению препятствуют два обстоятельства:

1. Восстановленный процесс устанавливается на выходе фильтра только спустя некоторое время вследствие нулевых начальных условий самого фильтра как динамического звена. Это продемонстрировано ниже, на рис. 5.21.

```
» y=filter(b,a,x)
» plot(t,y,t,Yp),grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Процесс на выходе фильтра (Tf=1; dz=0.05)');
» xlabel('Время (c)');
» ylabel('Y(t)')
```

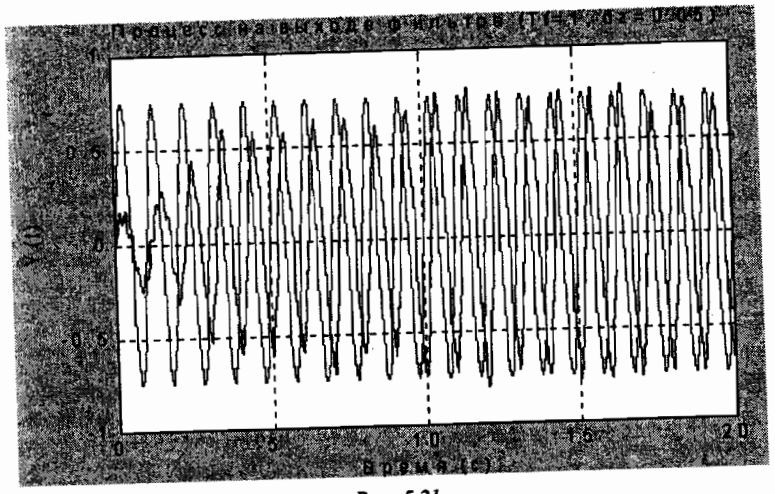

Рис. 5.21

2. В установившемся режиме наблюдается значительный сдвиг ($\pi/2$) фаз между восстанавливаемым и восстановленным процессами; это понятно, так как при резонансе сдвиг фаз между входным и выходным процессами достигает именно такой величины.

Чтобы избежать фазовых искажений полезного сигнала при его восстановлении, можно воспользоваться процедурой двойной фильтрации — **filtfilt**. Обращение к ней имеет такую же форму, что и к процедуре **filter**. В отличие от последней процедура **filtfilt** осуществляет обработку вектора *x* в два приема: сначала в прямом, а затем в обратном направлении.

Результат применения этой процедуры в рассматриваемом случае приведен на рис. 5.22.

```
» y=filtfilt(b,a,x)
» plot(t,y,t,Yp),grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Применение процедуры FILTFILT');
» xlabel('Время (c)');
» ylabel('Y(t)')
```

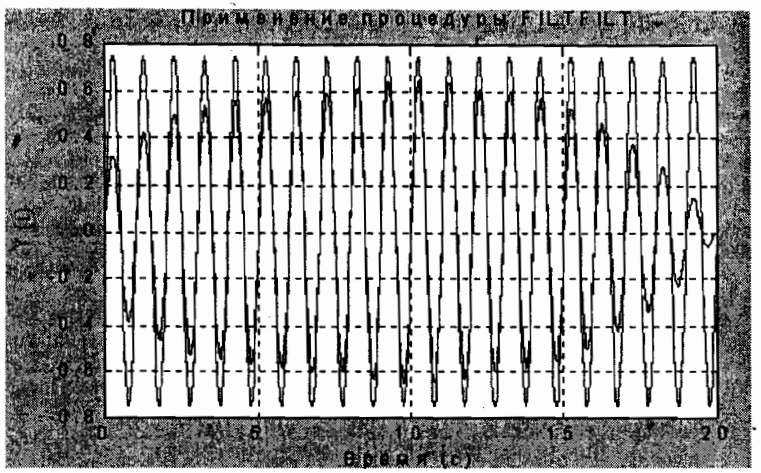

Рис. 5.22

5.2.2. Формирование случайных процессов

В соответствии с теорией сформировать случайный процесс с заданной корреляционной функцией можно, если сначала сформировать случайный процесс, являющийся нормально (по гауссовому закону) распределенным белым шумом, а затем "пропустить" его через некоторое динамическое звено (формирующий фильтр). На выходе получается нормально распределенный случайный процесс с корреляционной функцией, вид которой определяется типом формирующего фильтра как динамического звена.

Белый гауссовый шум в MatLAB образуется при помощи процедуры **randn**. Для этого достаточно задать дискрет времени Ts, образовать с этим шагом массив (вектор) t моментов времени в нужном диапазоне, а затем сформировать по указанной процедуре вектор-столбец длиной, равной длине вектора t, например:

```
» Ts=0.01;
» t=0 : Ts : 20;
» xl=randn(1,length(t));
```

Построим график полученного процесса:

```
» plot(t,x1),grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Входной процесс – белый шум Гаусса (Ts=0.01)');
» xlabel('Время (с)');
» ylabel('X1(t)')
```

Процесс x1(t) с дискретом времени Ts=0.01с представлен на рис. 5.23.

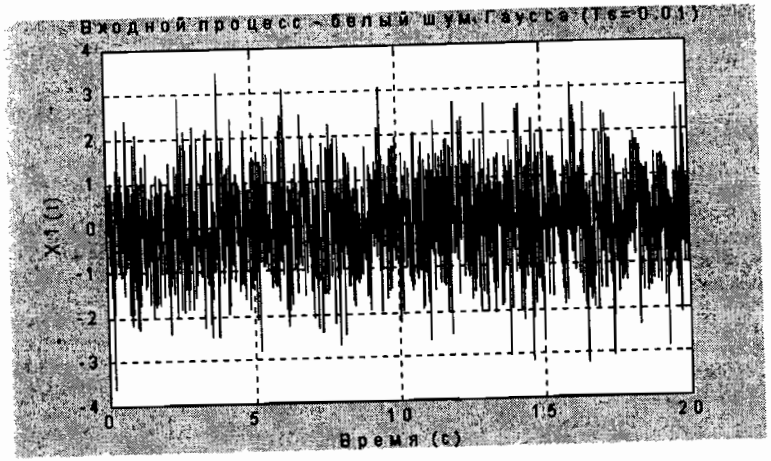

Рис. 5.23

Для другого значения дискрета времени (Ts=0.001 с), повторяя аналогичные операции, получим процесс $x_2(t)$ (рис. 5.24).

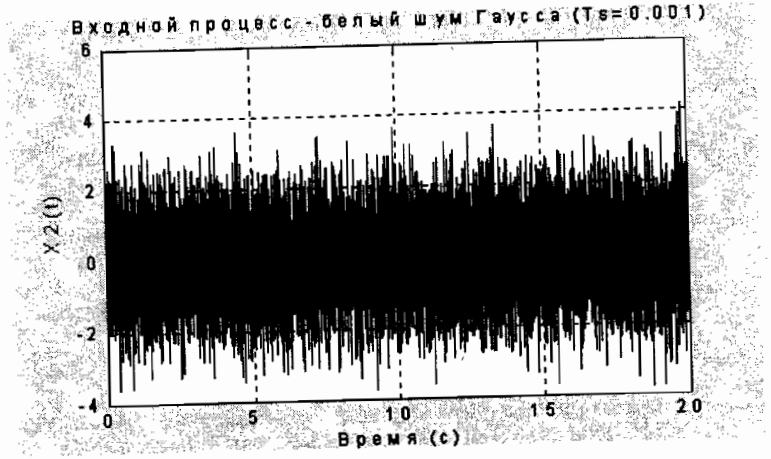

Рис. 5.24

Создадим дискретный фильтр второго порядка с частотой собственных колебаний $\omega_0 = 2\pi$ рад/с = 1 Гц и относительным коэффициентом затухания ς=0.05 по формулам (5.5) коэффициентов:

```
» om0=2*pi;  dz=0.05;  A=1;  oms=om0*Ts;
» a(1)= 1+2*dz*oms+oms^2;
```

```
» a(2)= -2*(1+dz*oms);
» a(3)=1;
» b(1)=A*2*dz*oms^2;
```

Пропустим образованный процесс $x_1(t)$ через созданный фильтр:

```
» y1=filter(b,a,x1);
```

Построим график процесса $y_1(t)$ на выходе фильтра:

```
» plot(t,y1),grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Процесс на выходе фильтра (T0=1; dz=0.05, Ts=0.01)');
» xlabel('Время (c)');
» ylabel('Y1(t)')
```

Результат представлен на рис. 5.25.

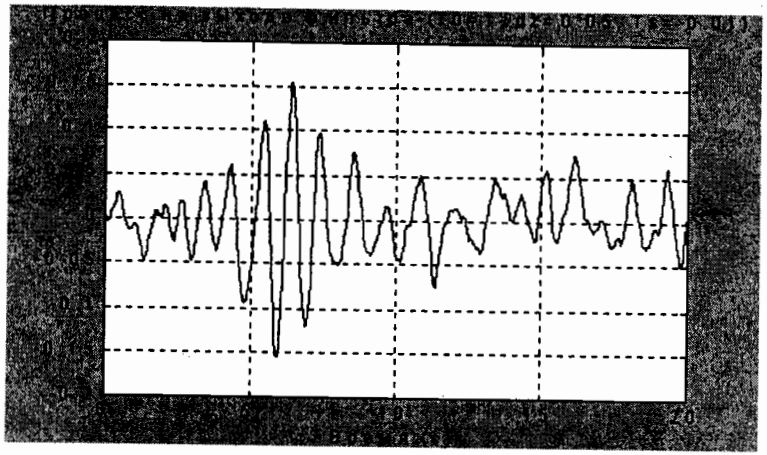

Рис. 5.25

Аналогичные операции произведем с процессом $x_2(t)$. В результате получим процесс $y_2(t)$, приведенный на рис. 5.26.

```
» Ts=0.001;
» om0=2*pi;  dz=0.05;  A=1;  oms=om0*Ts;
» a(1)= 1+2*dz*oms+oms^2;
» a(2)= -2*(1+dz*oms);
» a(3)=1;
» b(1)=A*2*dz*oms^2;
» y1=filter(b,a,x2); t=0 : Ts : 20;
» plot(t,y1),grid,
» set(gca,'FontName','Arial','FontSize',14)
» title('Процесс на выходе фильтра (T0=1; dz=0.05, Ts=0.001)');
» xlabel('Час (c)');
» ylabel('Y2(t)')
```


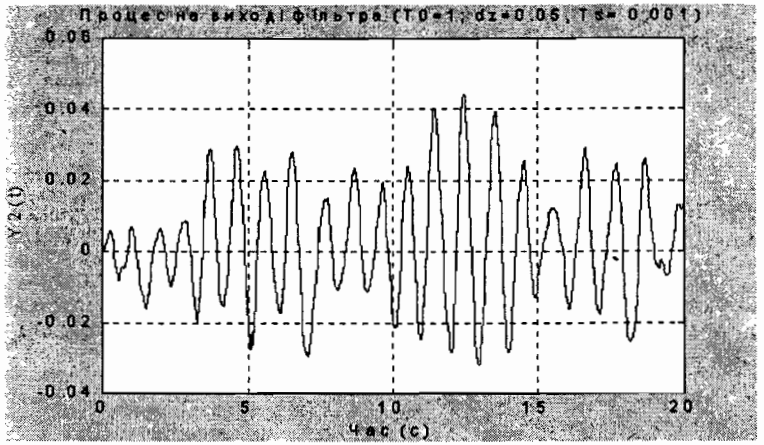

Рис. 5.26

Как видим, на выходе формирующего фильтра действительно образуется случайный колебательный процесс с преобладающей частотой 1 Гц.

5.3. Спектральный и статистический анализ

5.3.1. Основы спектрального (частотного) и статистического анализа процессов

Основная задача спектрального анализа сигналов — выявление гармонического спектра этих сигналов, т.е. определение частот гармонических составляющих сигнала (*выявление частотного спектра*), амплитуд этих гармонических составляющих (*амплитудного спектра*) и их начальных фаз (*фазового спектра*).

В основе спектрального анализа лежит теория Фурье о возможности разложения любого периодического процесса с периодом $T = \frac{2\pi}{\omega} = \frac{1}{f}$ (где ω — круговая частота периодического процесса, а f — его частота в герцах) в бесконечную, но счетную сумму отдельных гармонических составляющих.

Напомним некоторые положения спектрального анализа.

Прежде всего, любой периодический процесс с периодом T может быть представлен в виде так называемого комплексного ряда Фурье:

$$x(t) = \sum_{m=-\infty}^{+\infty} X^*(m) \cdot e^{j(2\pi mf)\cdot t} = \sum_{m=-\infty}^{+\infty} X^*(m) \cdot e^{j(m\omega)\cdot t} \qquad (5.8)$$

причем комплексные числа $X^*(m)$, которые называют *комплексными амплитудами* гармонических составляющих, вычисляются по формулам:

$$X^*(m) = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)\cdot e^{-j(2\pi mf)t}dt = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)\cdot e^{-j(m\omega)t}dt \qquad (5.9)$$

Таким образом, частотный спектр периодического колебания состоит из частот, кратных основной (базовой) частоте f, т.е. частот:

$$fm = m\cdot f \quad (m=0,1,2,...) \qquad (5.10)$$

Действительные и мнимые части комплексных амплитуд $X^*(m)$ образуют соответственно *действительный* и *мнимый спектры* периодического колебания. Если комплексную амплитуду (5.9) представить в экспоненциальной форме:

$$X^*(m) = \frac{a_m}{2}\cdot e^{j\cdot\varphi_m} \qquad (5.11)$$

то величина a_m будет представлять собой амплитуду гармонической составляющей с частотой $f_m=m\cdot f$, а φ_m — начальную фазу этой гармоники, имеющей форму косинусоиды, т.е. исходный процесс можно также записать в виде:

$$x(t) = a_0 + \sum_{m=1}^{+\infty} a_m\cdot\cos(2\pi mft + \varphi_m) \qquad (5.12)$$

который, собственно, и называют *рядом Фурье*.

Для действительных процессов справедливы следующие соотношения:

$$\mathrm{Re}\{X(-m)\} = \mathrm{Re}\{X(m)\}; \quad \mathrm{Im}\{X(-m)\} = -\mathrm{Im}\{X(m)\} \qquad (5.13)$$

т.е. *действительная часть спектра является четной функцией частоты*, а *мнимая часть спектра — нечетной функцией частоты*.

Разложения (5.12) и (5.8) позволяют рассматривать совокупность комплексных амплитуд (5.9) как изображение периодического процесса в частотной области. Желание распространить такой подход на произвольные процессы, в том числе и непериодические, привело к необходимости ввода понятия Фурье-изображения в соответствии со следующим выражением:

$$X(f) = \int_{-\infty}^{+\infty} x(t)\cdot e^{-j(2\pi f)t}dt \qquad (5.14)$$

Этот интеграл, несмотря на его внешнее сходство с выражением (5.9) для комплексных коэффициентов ряда Фурье, довольно существенно отличается от них.

Во-первых, в то время как физическая размерность комплексной амплитуды совпадает с размерностью самой физической величины $x(t)$, размерность Фурье-изображения равна размерности $x(t)$, умноженной на размерность времени.

Во-вторых, интеграл (5.14) существует (является сходящимся к конечной величине) только для так называемых "двусторонне затухающих" процессов (т.е. таких, которые уменьшаются до нуля как при $t\to+\infty$, так и при $t\to-\infty$). Иначе говоря, его нельзя применять к так называемым "стационарным" колебаниям.

Обратное преобразование Фурье-изображения в исходный процесс $x(t)$ в этом случае определяется интегралом:

$$x(t) = \int_{-\infty}^{+\infty} X(f)\cdot e^{j(2\pi f)t}df \qquad (5.15)$$

который представляет собой некоторый аналог комплексного ряда Фурье (5.1).

Указанное серьезное противоречие несколько сглаживается при численных расчетах, так как в этом случае можно иметь дело только с процессами ограниченной длительности, причем сам процесс в заданном диапазоне времени должен быть задан своими значениями в ограниченном числе точек.

В этом случае интегрирование заменяется суммированием, и вместо вычисления интеграла (5.14) ограничиваются вычислением суммы:

$$X[(k-1)\cdot\Delta f] = \Delta t\cdot\sum_{m=1}^{n} x[(m-1)\cdot\Delta t]\cdot e^{-j\cdot 2\pi\cdot(k-1)\cdot(m-1)\cdot\Delta f\cdot\Delta t} \qquad (5.16)$$

Тут по сравнению с интегралом (5.14) осуществлены такие замены:

- непрерывный интеграл приближенно заменен ограниченной суммой площадей прямоугольников, одна из сторон которых равна дискрету по времени Δt, с которым представлены значения процесса, а вторая — мгновенному значению процесса в соответствующий момент времени;
- непрерывное время t заменено дискретными его значениями $(m-1)\cdot\Delta t$, где m — номер точки от начала процесса;
- непрерывные значения частоты f заменены дискретными ее значениями $(k-1)\cdot\Delta f$, где k — номер значения частоты, а дискрет частоты равен $\Delta f = \frac{1}{T}$, где, в свою очередь, T — промежуток времени, на котором задан процесс;
- дифференциал dt заменен ограниченным приращением времени Δt.

Если обозначить дискрет времени Δt через Ts, ввести обозначения:

$$x(m) = x[(m\text{-}1)\cdot\Delta t];\quad X(k) = X[(k\text{-}1)\cdot\Delta f]$$

а также учесть то, что число точек, в которых задан процесс, равно:

$$n = \frac{T}{\Delta t} = \frac{T}{Ts} = \frac{1}{\Delta f \cdot \Delta t} \tag{5.17}$$

то соотношение (5.15) можно представить в более удобной форме:

$$X(k) = Ts \cdot \sum_{m=1}^{n} x(m) \cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} \tag{5.18}$$

Как было отмечено в разд. 1.4.5 (формулы (2) и (3)), процедуры MatLAB **fft** и **ifft** осуществляют вычисления в соответствии с формулами:

$$y(k) = \sum_{m=1}^{n} x(m) \cdot e^{-j\cdot 2\pi\cdot(m-1)\cdot(k-1)/n} \tag{5.19}$$

$$x(m) = \frac{1}{n}\sum_{k=1}^{n} y(k) \cdot e^{j\cdot 2\pi\cdot(m-1)\cdot(k-1)/n} \tag{5.20}$$

соответственно. Сравнивая (5.18) с (5.19), можно сделать вывод, что *процедура* **fft** *находит дискретное Фурье-изображение заданного дискретного во времени процесса* $x(t)$, *поделенное на дискрет времени:*

$$y(k) = \frac{X(k)}{Ts} \tag{5.21}$$

Осуществляя аналогичную операцию дискретизации соотношения (5.9) для комплексной амплитуды $X^*(k)$, получим:

$$\begin{aligned} X^*(k) &= \frac{Ts}{T}\cdot\sum_{m=1}^{n} x(m)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} = \\ &= \frac{1}{n}\cdot\sum_{m=1}^{n} x(m)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} = \frac{y(k)}{n} \end{aligned} \tag{5.22}$$

Из этого следует, что *комплексный спектр разложения стационарного процесса равен поделенному на число измерений результату применения процедуры* **fft** *к заданному вектору измеренного процесса.*

Если же принять во внимание, что для большинства стационарных колебательных процессов именно частотный, амплитудный и фазовый спектры не зависят от длительности T конкретной реализации и выбранного дискрета времени Ts, то надо также сделать вывод, что *для спектрального анализа стационарных процессов наиболее целесообразно применять процедуру* **fft**, *результат которой делить затем на число точек измерений.*

Перейдем к определению спектральной плотности мощности (СПМ), или, сокращенно, спектральной плотности (СП). Это понятие в теории определяется как Фурье-изображение так называемой корреляционной функции $R_{12}(\tau)$ и применяется, в основном, для двух одновременно протекающих *стационарных* процессов $x_1(t)$ и $x_2(t)$. Взаимная корреляционная функция (ВКФ) двух таких процессов определяется соотношением:

$$R_{12}(\tau) = \lim_{T\to\infty}\frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} x_1(t)\cdot x_2(t+\tau)\cdot dt \tag{5.23}$$

т.е. *ВКФ является средним во времени значением произведения первой функции на сдвинутую относительно нее на время задержки* τ *вторую функцию.*

Итак, взаимная спектральная плотность (ВСП) двух стационарных процессов может быть определена следующим образом:

$$S_{12}(f) = \int_{-\infty}^{+\infty} R_{12}(\tau)\cdot e^{-j(2\pi f)\tau} d\tau \tag{5.24}$$

При числовых расчетах, когда оба процесса $x_1(t)$ и $x_2(t)$ заданы на определенном ограниченном промежутке T времени своими значениями в некоторых точках, разделенных дискретом времени Ts, формулу (5.23) можно трансформировать в такую:

$$R_{12}(l) = \frac{1}{n-l}\sum_{m=1}^{n-l} x_1(m)\cdot x_2(m+l-1), \quad (l = 1,2,\ldots,n/2) \tag{5.25}$$

или в несколько более простое соотношение:

$$R_{12}(l) = \frac{2}{n}\sum_{m=1}^{n/2} x_1(m)\cdot x_2(m+l-1), \quad (l = 1,2,\ldots,n/2) \tag{5.26}$$

а вместо (5.24) использовать

$$S_{12}(k) = Ts\cdot\sum_{l=1}^{n/2} R_{12}(l)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(l-1)}, \quad (k = 1,2,\ldots,n/2) \tag{5.27}$$

Если теперь подставить выражение (5.26) в (5.27) и изменить в нем порядок суммирования, то можно прийти к такому соотношению между ВСП и результатами преобразований процедурой **fft** заданных измеренных значений процессов:

$$S_{12}(k) = Ts\cdot\{\frac{2}{n}y_2(k)\}\cdot\bar{y}_1(k), \quad (k = 1,2,\ldots,n/2) \tag{5.28}$$

где черта сверху означает комплексное сопряжение соответствующей величины.

С учетом (5.21) и (5.22) выражение (5.28) можно представить также в виде:

$$S_{12}(k) = X_2^*(k) \cdot \bar{X}_1(k) \qquad (5.29)$$

Из этого следует, что *взаимная спектральная плотность двух процессов при любом значении частоты равна произведению значения комплексного спектра второго процесса на комплексно-сопряженное значение Фурье-изображения первого процесса на той же частоте.*

Формулы (5.21), (5.22) и (5.28) являются основой для вычислений в системе MatLAB соответственно Фурье-изображения процесса, его комплексного спектра и взаимной спектральной плотности двух процессов.

5.3.2. Примеры спектрального анализа

Чтобы применить процедуру **fft** как преобразование процесса, представленного во временной области, в его представление в частотной области, необходимо, как было отмечено в разделе 1.4.5, сделать следующее:

- по заданному значению дискрета времени Ts рассчитать величину F_{max} диапазона частот (в герцах) по формуле:

$$Fmax = 1/Ts \qquad (5.30)$$

- по заданной длительности процесса T рассчитать дискрет частоты df по формуле:

$$df = 1/T \qquad (5.31)$$

- по вычисленным данным сформировать вектор значений частот, в которых будет вычислено Фурье-изображение.

Последнее проще (но не наиболее правильно) сделать таким образом:

$$f1=0 : df : Fmax \qquad (5.32)$$

В результате применения процедуры **fft** будет получено представление процесса в частотной области. Обратная процедура **ifft**, если ее применить к результатам первого преобразования, дает возможность восстановить исходный процесс во временной области.

Однако процедура **fft** не дает непосредственно Фурье-изображения процесса. Чтобы получить Фурье-изображение, необходимо выполнить следующие действия (разд. 1.4.5):

- к результатам действия процедуры **fft** применить процедуру **fftshift**, которая переставляет местами первую и вторую половины полученного вектора;
- перестроить вектор частот по алгоритму:

$$f = -Fmax/2 : df : Fmax/2 \qquad (5.33)$$

Приведем примеры.

Фурье-изображение прямоугольного импульса

Сформируем процесс, состоящий из одиночного прямоугольного импульса. Зададим дискрет времени Ts=0.01 с, длительность процесса T=100 с, амплитуду импульса A=0.75 и его ширину w=0.5 с:

```
» Ts=0.01;   T=100;   A=0.75;   w=0.5;
» t=0 : Ts : T;
» y = A*rectpuls(t, w);
» plot(t(1:100),y(1:100)), grid, set(gca,'FontName',...
  'Arial Cyr','FontSize',16),
» title('Процесс из одиночного прямоугольного импульса ');
» xlabel('Время (с)');
» ylabel('Y(t)')
```

Результат показан на рис. 5.27.

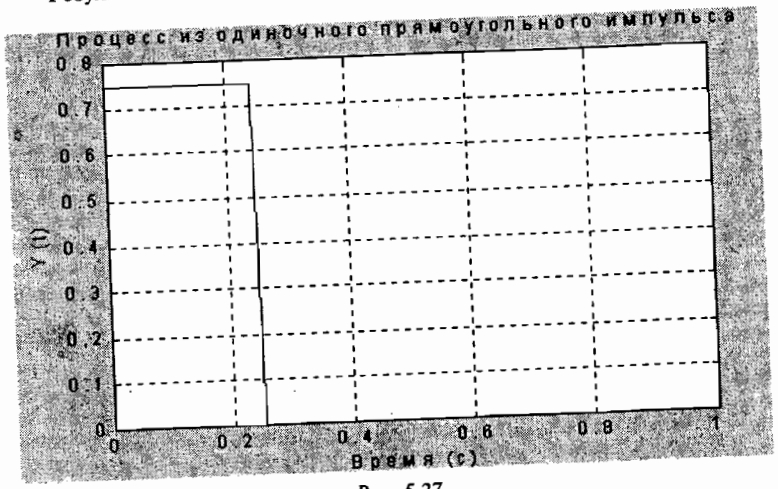

Рис. 5.27

Применим к вектору *y* процедуру **fft** и построим график зависимости модуля результата от частоты. *Графики в частотной области удобнее выводить при помощи процедуры stem* (рис. 5.28):

```
» x=fft(y);
» df=1/T;   Fmax=1/Ts;
» f=0 : df : Fmax;
» a=abs(x);
» stem(f,a), grid, set(gca,'FontName','Arial Cyr','FontSize',14),
» title('Модуль FFT-преобразования прямоугольного импульса');
» xlabel('Частота (Гц)');
» ylabel('Модуль')
```

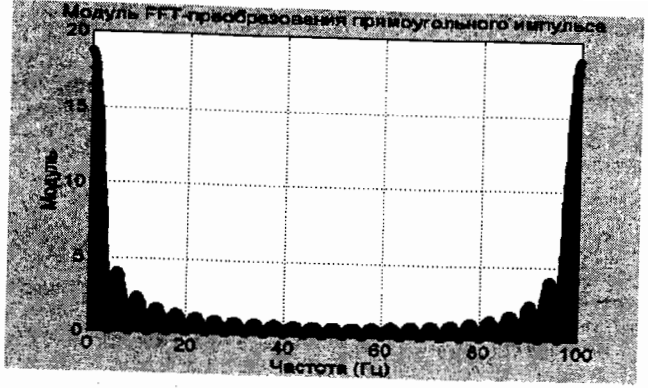

Рис. 5.28

Теперь построим график модуля Фурье-изображения процесса:

```
» xp=fftshift(x);
» fl=-Fmax/2 : df : Fmax/2;
» a=abs(xp);
» stem(fl,a), grid, set(gca,'FontName','Arial Cyr','FontSize',14),
» title('Модуль Фурье-изображения прямоугольного импульса');
» xlabel('Частота (Гц)');
» ylabel('Модуль')
```

Получим результат, приведенный на рис. 5.29.

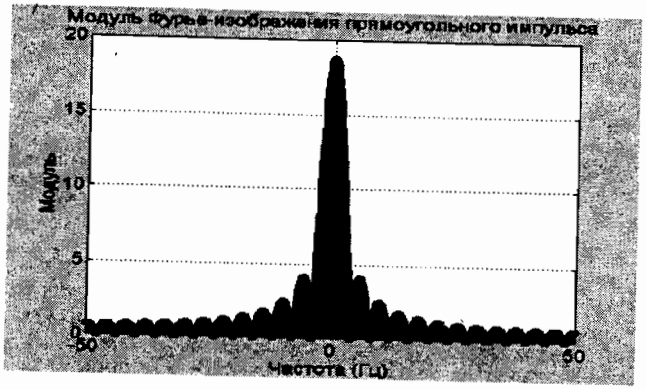

Рис. 5.29

В заключение построим графики действительной и мнимой частей Фурье-изображения прямоугольного импульса:

```
» dch=real(xp);     mch=imag(xp);
» plot(fl,dch,fl,mch), grid, set(gca,'FontName',...
  'Arial Cyr','FontSize',16),
» title('Фурье-изображение прямоугольного импульса ');
» ylabel('Действит. и Мнимая части')
» xlabel('Частота (Гц)');
```

Полученные графики представлены на рис. 5.30.

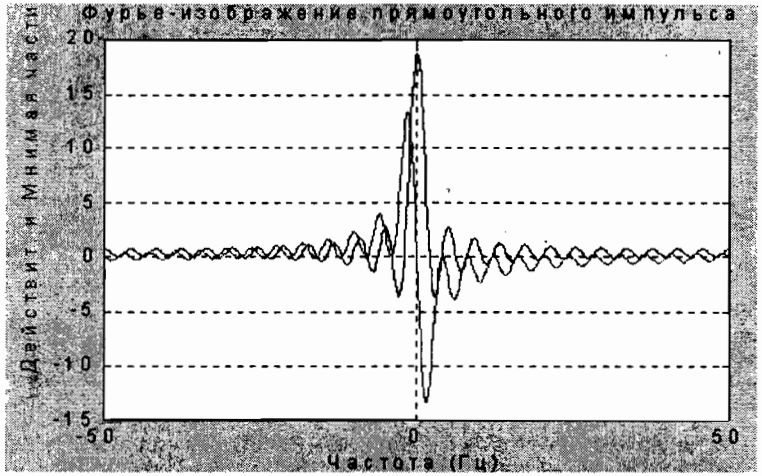

Рис. 5.30

Фурье-изображение полигармонического процесса

Рассмотрим пример трехчастотных гармонических колебаний — с частотой 1/π, 1 и 3 Гц и амплитудами соответственно 0.6, 0.3 и 0.7:

$$y(t)=0.6\cdot\cos(2t)+0.3\cdot\sin(2\pi\cdot t)+0.7\cdot\cos(6\pi\cdot t+\pi/4)$$

Найдем Фурье-изображение этого процесса и выведем графики самого процесса, модуля его Фурье-изображения, а также действительную и мнимую части:

```
» Ts = 0.01;    T = 100;
» t = 0 : Ts : T;
» Y = 0.6*cos(2*t)+0.3*sin(2*pi*t)+0.7*cos(6*pi*t+pi/4);
» plot(t,Y), grid, set(gca,'FontName','Arial Cyr','FontSize',16),
» title('Трехчастотный полигармонический процесс');
» xlabel('Время (с)');
» ylabel('Y(t)')
```

График процесса показан на рис. 5.31.

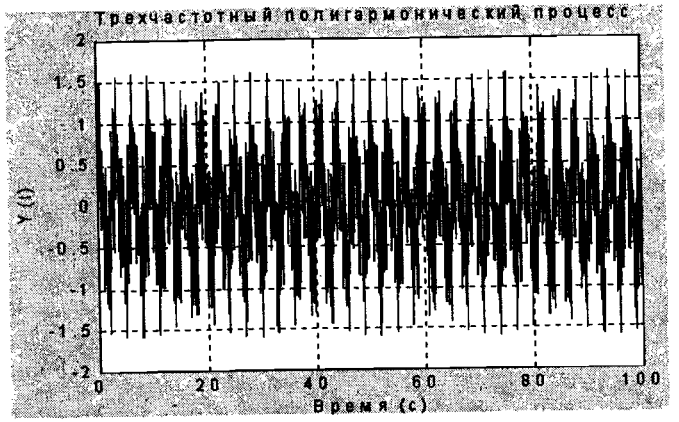

Рис. 5.31

Находим модуль Фурье-изображения этого процесса:

```
» df = 1/T;  Fmax = 1/Ts;  dovg=length(t);
» f = -Fmax/2 : df  : Fmax/2;
» X = fft(Y);  Xp = fftshift(X);
» A = abs(Xp);
» s1 = dovg/2 - 400;  s2 = dovg/2 + 400;
» stem(f(s1:s2),A(s1:s2)); grid,
» set(gca,'FontName','Arial Cyr','FontSize',14),
» title('Модуль Фурье-изображения полигармонического процесса');
» xlabel('Частота (Гц)');
» ylabel('Модуль')
```

Результат представлен на рис. 5.32.

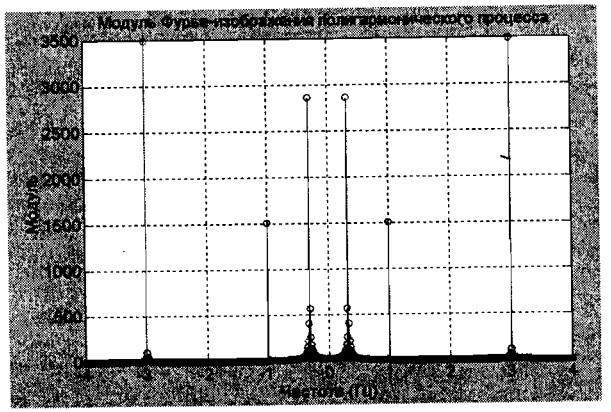

Рис. 5.32

Если изменить дискрет времени на Ts=0.02, получим результат, изображенный на рис. 5.33.

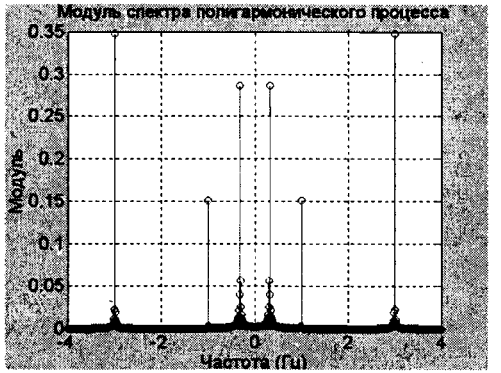

Рис. 5.33

Как видно, результат Фурье-преобразования в значительной степени зависит от величины дискрета времени и мало что говорит об амплитудах гармонических составляющих. Это обусловлено различием между определениями Фурье-изображения и комплексного спектра. Поэтому для незатухающих (установившихся, стационарных) колебаний любого вида намного удобнее находить не Фурье-изображение, а его величину, деленную на число точек в реализации. В предыдущей части программы это эквивалентно замене оператора X=fft(Y) на X=fft(Y)/dovg, где dovg — длина вектора t.

В результате получается комплексный спектр (рис. 5.34), полностью соответствующий коэффициентам комплексного ряда Фурье.

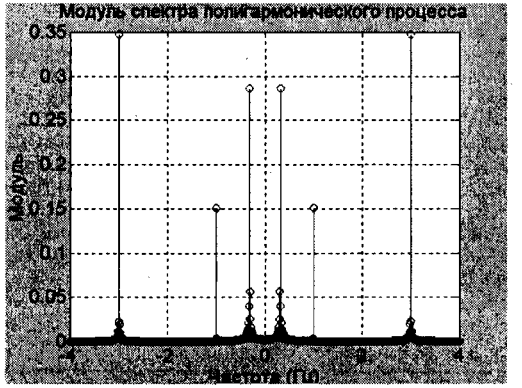

Рис. 5.34

Выделим действительную и мнимую части комплексного спектра:

```
» dch = real(Xp);    mch = imag(Xp);
» s1 = dovg/2 - 400;    s2 = dovg/2 + 400;
» subplot(2,1,1)
» plot(f(s1:s2),dch(s1:s2)), grid,
» set(gca,'FontName','Arial Cyr','FontSize',10),
» title('Комплексный спектр полигармонических колебаний');
» ylabel('Действит. часть')
» subplot(2,1,2)
» plot(f(s1:s2),mch(s1:s2)), grid,
» set(gca,'FontName','Arial Cyr','FontSize',10),
» xlabel('Частота (Гц)');
» ylabel('Мнимая часть')
```

По полученным графикам (рис. 5.35) можно судить не только о частотах и амплитудах, но и о начальных фазах отдельных гармонических составляющих.

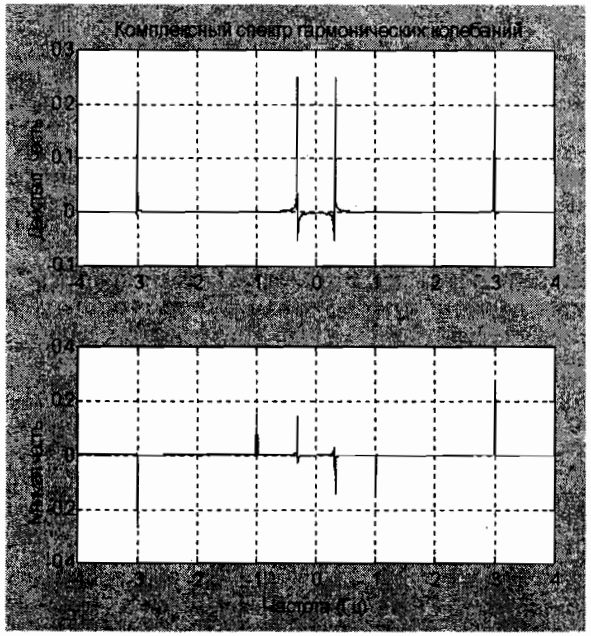

Рис. 5.35

Фурье-изображение случайного процесса

В заключение рассмотрим Фурье-преобразование случайного стационарного процесса, сформированного ранее (рис. 5.25). Сначала

сформируем процесс в виде белого гауссового шума (рис. 5.36) с шагом во времени 0.01 и длительностью 100 с:

```
» Ts=0.01; T = 100;    % Задание параметров процесса
» t=0 : Ts : T;
» x1=randn(1,length(t));  % Формирование белого шума
» % Построение графика белого шума
» plot(t,x1), grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Белый Гауссовый шум (СКО= 1;  Ts= 0.01)');
» xlabel('Время (с)');
» ylabel('X1(t)');
```

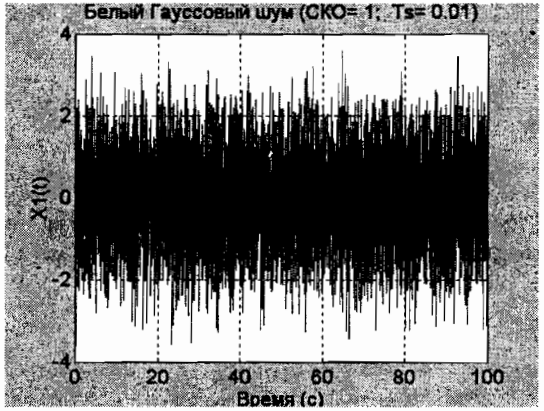

Рис. 5.36

Теперь создадим формирующий фильтр, "пропустим" через него белый шум и результат выведем на рис. 5.37:

```
» % Расчет параметров формирующего фильтра
» om0=2*pi;  dz=0.05; A=1; oms=om0*Ts;
» a(1)= 1+2*dz*oms+oms^2;
» a(2)= -2*(1+dz*oms);
» a(3)=1;
» b(1)=A*2*dz*oms^2;
» % Формирование "профильтрованного" процесса
» y1=filter(b,a,x1);
» % Построение графика процесса
» plot(t,y1),grid, set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Процесс на выходе фильтра (T0=1; dz=0.05, Ts=0.01)');
» xlabel('Время (с)');
» ylabel('Y1(t)')
```

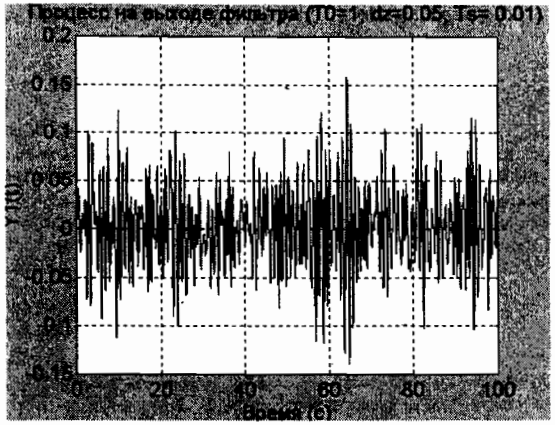

Рис. 5.37

Вычислим Фурье-изображение (ФИ) для процесса-шума с учетом замечания, сделанного для установившихся процессов, и построим на рис. 5.38 графики модуля ФИ и спектральной плотности мощности (СПМ):

```
» % Формирование массива частот
» df = 1/T;    Fmax = 1/Ts;
» f = - Fmax/2 : df : Fmax/2;
» dovg = length(f);
» % Расчет скорректированных массивов Фурье-изображений
» Fu1 = fft(x1)/dovg;    Fu2 = fft(y1)/dovg;
» Fu1p = fftshift(Fu1);    Fu2p = fftshift(Fu2);
» % Формирование массивов модулей ФИ
» A1 = abs(Fu1p);    A2 = abs(Fu2p);
» % Вычисление Спектральных Плотностей Мощности
» S1 = Fu1p.*conj(Fu1p)*dovg;  S2 = Fu2p.*conj(Fu2p)*dovg;
» % Вывод графиков белого шума
» subplot(2,1,1);
» stem(f,A1),grid,
» set(gca,'FontName','Arial Cyr','FontSize',10)
» title('Модуль ФИ гауссового белого шума');
» subplot(2,1,2);
» stem(f,S1),grid,
» set(gca,'FontName','Arial Cyr','FontSize',10)
» title('Спектральная плотность мощности');
» xlabel('Частота (Гц)');
```


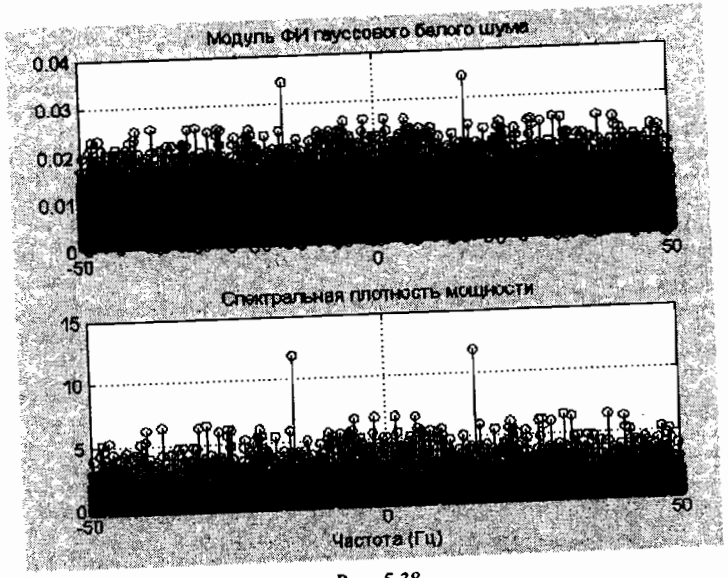

Рис. 5.38

Из рис. 5.38 видно, что спектральная плотность практически одинакова по величине во всем диапазоне частот, чем и обусловлено название процесса — белый шум.

Аналогичную процедуру выполним и для "профильтрованного" процесса (рис. 5.39):

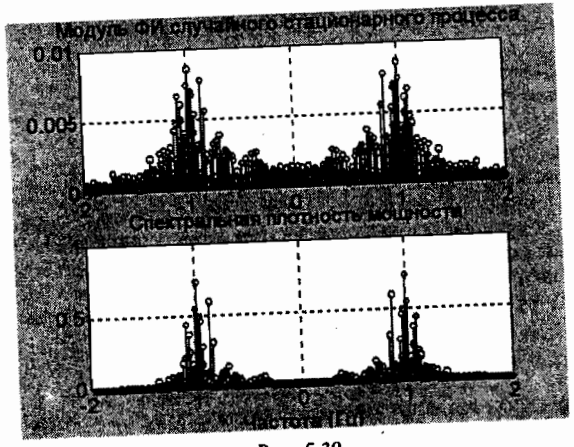

Рис. 5.39

```
% Вывод графиков профильтрованного процесса
c1 = fix(dovg/2)-200,   c2 = fix(dovg/2)+200,  length(f)
subplot(2,1,1);
stem(f(c1:c2),A2(c1:c2)),grid,
set(gca,'FontName','Arial Cyr','FontSize',16)
title('Модуль ФИ случайного стационарного процесса');
subplot(2,1,2);
stem(f(c1:c2),S2(c1:c2)),grid,
set(gca,'FontName','Arial Cyr','FontSize',16)
title('Спектральная плотность мощности');
xlabel('Частота (Гц)');
```

Проводя эти вычисления еще раз с новой длительностью процесса T=20 с (рис. 5.40), можно наглядно убедиться, что величины ФИ и СПМ при этом практически не изменяются.

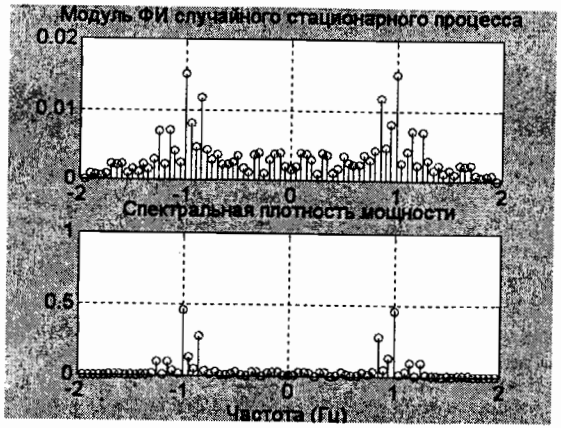

Рис. 5.40

5.3.3. Статистический анализ

К задачам статистического анализа процессов относится определение некоторых статистических характеристик процессов, а именно: корреляционных характеристик, спектральных плотностей мощности и т.п.

В предыдущем разделе уже были определены СП случайного процесса на основе установленной связи СП с Фурье-изображением. Однако в Signal Processing Toolbox предусмотрена специальная процедура **psd**, позволяющая сразу находить СП сигнала. Обращение к ней имеет вид:

```
[S,f] = psd(x,nfft,Fmax)
```

где x — вектор заданных значений процесса, *nfft*— число элементов вектора, которые обрабатываются процедурой **fft**, *Fmax*=1/*Ts* — значение частоты дискретизации сигнала, S — вектор значений СП сигнала, f — вектор значений частот, которым соответствуют найденные значения СП. В общем случае длина последних двух векторов равна *nfft*/2.

Приведем пример применения процедуры **psd** для нахождения СП предыдущего случайного процесса:

```
» [C,f] = psd(y1,dovg,Fmax);
» subplot
» stem(f(1:200),C(1:200)),grid,
» set(gca,'FontName','Arial Cyr','FontSize',16)
» title('Спектральная плотность мощности');
» xlabel('Частота (Гц)');
```

В результате получим картину, представленную на рис. 5.41.

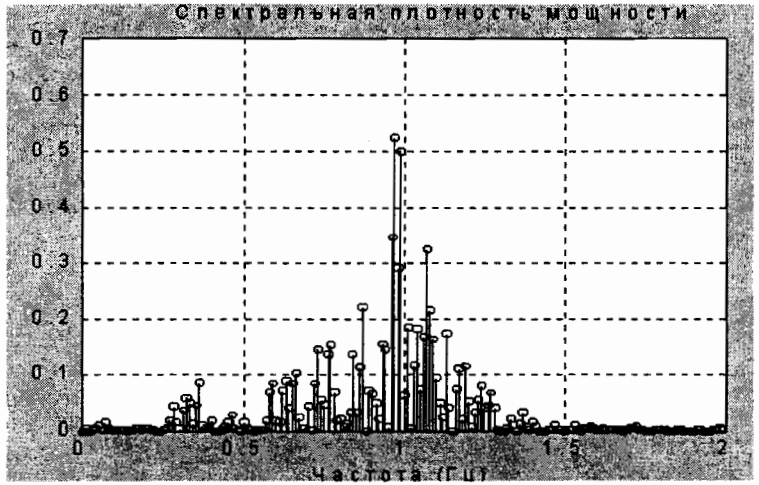

Рис. 5.41

Если ту же процедуру вызвать без указания выходных величин, то результатом ее выполнения станет выведение графика зависимости СП от частоты.

Например, обращение:

```
psd(y1,dovg,Fmax)
```

приведет к построению в графическом окне (фигуре) графика, как на рис. 5.42. При этом значения СП будут откладываться в логарифмическом масштабе в децибелах.

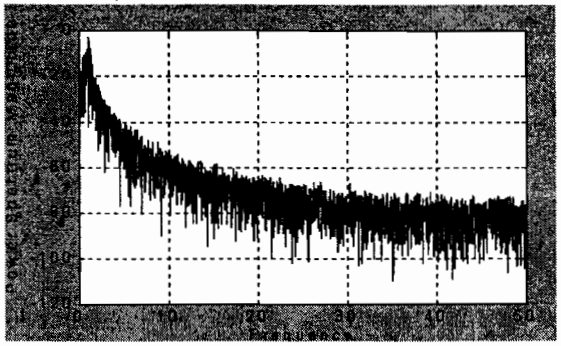

Рис. 5.42

Группа функций **xcorr** вычисляет оценку взаимной корреляционной функции (ВКФ) двух последовательностей *x* и *y*. Обращение **c = xcorr(x,y)** вычисляет и выдает вектор *c* длины 2N-1 значений ВКФ векторов *x* и *y* длины N. Также позволяет вычислить АКФ (автокорреляционную функцию) последовательности, заданной в векторе *x*. Вычислим АКФ для случайного процесса, сформированного ранее:

```
» R=xcorr(y1);
» tau= -10+Ts : Ts : 10;
» lt=length(tau );
» s1r=round(length(R )/2)-lt/2;
» s2r=round(length(R )/2)+lt/2-1;
» plot(tau,R(s1r:s2r)),grid
» set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16)
» title('АКФ случайного процесса')
» xlabel('Запаздывание (c)')
```

На рис. 5.43 представлен результат применения процедуры **xcorr**.

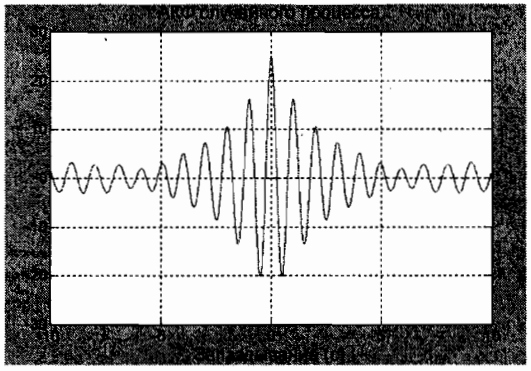

Рис. 5.43

5.4. Проектирование фильтров

5.4.1. Формы представления фильтров и их преобразования

Фильтр как звено системы автоматического управления может быть представлен в нескольких эквивалентных формах, каждая из которых полностью описывает его:

- В форме рациональной *передаточной функции* (**tf**-*представление*); если звено является непрерывным (аналоговым), то оно описывается непрерывной передаточной функцией:

$$W(s) = \frac{b(s)}{a(s)} = \frac{b(1)\cdot s^m + b(2)\cdot s^{m-1} + \ldots + b(m+1)}{a(1)\cdot s^n + a(2)\cdot s^{n-1} + \ldots + a(n+1)} \qquad (5.34)$$

  а в случае дискретного фильтра последний может быть представлен дискретной передаточной функцией вида:

$$W(z) = \frac{b(z)}{a(z)} = \frac{b(1) + b(2)\cdot z^{-1} + \ldots + b(m+1)\cdot z^{-m}}{a(1) + a(2)\cdot z^{-1} + \ldots + a(n+1)\cdot z^{-n}} \qquad (5.35)$$

  В обоих случаях для задания звена достаточно задать два вектора: *b* — вектор коэффициентов числителя и *a* — знаменателя передаточной функции.

- В виде разложения передаточной функции на простые дроби; в случае простых корней такое разложение имеет вид (для дискретной передаточной функции):

$$\frac{b(z)}{a(z)} = \frac{r(1)}{1 - p(1)\cdot z^{-1}} + \ldots + \frac{r(n)}{1 - p(n)\cdot z^{-1}} + k(1) + k(2)\cdot z^{-1} + \ldots + k(m-n+1)\cdot z^{-(m-n)} \qquad (5.36)$$

  В этой форме звено описывается тремя векторами: вектором-столбцом *r* вычетов передаточной функции, вектором-столбцом *p* полюсов и вектором-строкой *k* коэффициентов целой части дробно-рациональной функции.

- В каскадной форме (**sos**-*представление*), когда передаточная функция звена представлена в виде произведения передаточных функций не выше второго порядка:

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}\cdot z^{-1} + b_{2k}\cdot z^{-2}}{a_{0k} + a_{1k}\cdot z^{-1} + a_{2k}\cdot z^{-2}} \qquad (5.37)$$

  Параметры каскадного представления задаются в виде матрицы sos, содержащей вещественные коэффициенты:

$$sos = \begin{bmatrix} b_{01} & b_{11} & b_{21} & a_{01} & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & a_{02} & a_{12} & a_{22} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ b_{0L} & b_{1L} & b_{2L} & a_{0L} & a_{1L} & a_{2L} \end{bmatrix} \qquad (5.38)$$

- В пространстве состояний (**ss**-*представление*), т.е. с помощью уравнений звена в форме:

$$\dot{x} = A \cdot x + B \cdot u$$
$$y = C \cdot x + D \cdot u \qquad (5.39)$$

  В этой форме звено задается совокупностью четырех матриц A, B, C и D;
- Путем задания векторов z нулей передаточной функции, p — ее полюсов и k — коэффициента передачи звена (**zp**-*представление*):

$$W(s) = k\frac{[s - z(1)] \cdot [s - z(2)] \ldots [s - z(m)]}{[s - p(1)] \cdot [s - p(2)] \ldots [s - p(n)]} \qquad (5.40)$$

- Решетчатое **latc**-*представление*; в этом случае решетчатый фильтр задается векторами k коэффициентов знаменателя решетчатого дискретного фильтра и v — коэффициентов его числителя; коэффициенты k решетчатого представления некоторого полинома с коэффициентами, представленными вектором a, определяются по этому вектору с помощью рекурсивного алгоритма Левинсона.

Пакет Signal предоставляет пользователю ряд процедур, позволяющих преобразовать звено (фильтр) из одной формы в другую.

Процедуры преобразования к tf-форме

1. Процедура **zp2tf** осуществляет вычисления векторов коэффициентов числителя (b) и знаменателя (a) передаточной функции в форме (5.34) по известным векторам z ее нулей, p — ее полюсов и k — коэффициенту усиления звена. Обращение к процедуре имеет вид:

```
[b,a] = zp2tf(z,p,k)
```

   В общем случае многомерного звена величина z является матрицей, число столбцов которой должно быть равно числу выходов. Вектор-столбец k содержит коэффициенты усиления по всем выходам звена. В векторе a выдаются вычисленные коэффициенты знаменателя, а матрица b содержит коэффициенты числителей. При этом каждая строка матрицы соответствует коэффициентам числителя для отдельной выходной величины.
2. Процедура **ss2tf** преобразовывает описание звена (системы) из пространства состояний в форму передаточной функции. Обращение к ней вида:

```
[b,a] = ss2tf(A,B,C,D,iu)
```

   позволяет найти коэффициенты числителей (b) и знаменателя (a) передаточных функций системы по всем выходным величинам и по входу с номером iu, если заданы матрицы A, B, C, D описания системы в виде (5.39).

3. Процедура **sos2tf** позволяет найти передаточную функцию звена по заданным параметрам каскадной формы. Для этого надо обратиться к этой процедуре таким образом:

```
[b,a] = sos2tf(sos)
```

   где sos — заданная матрица каскадной формы (5.38).
4. С помощью процедуры **latc2tf** можно вычислить коэффициенты числителя и знаменателя передаточной функции (5.35) по коэффициентам знаменателя и числителя решетчатого фильтра. При этом обращение к ней должно иметь один из видов:

```
[b,a] = latc2tf(k,v);
[b,a] = latc2tf(k,'iir');
b = latc2tf(k,'fir');
b = latc2tf(k)
```

   Первый вид используется, если заданы коэффициенты решетчатого представления и числителя v, и знаменателя k БИХ-фильтра (фильтра с бесконечной импульсной характеристикой).

   Второй вид применяется, если решетчатый БИХ-фильтр имеет только полюсы.

   Третий и четвертый виды служат для вычисления коэффициентов передаточной функции решетчатого КИХ-фильтра (с конечной импульсной характеристикой).
5. Нахождение коэффициентов передаточной функции по коэффициентам разложения ее на простые дроби (5.36) осуществляется путем использования функций **residue** и **residuez**. Первая применяется для непрерывной передаточной функции вида (5.34), вторая — для дискретной передаточной функции (5.35). При обращении вида:

```
[b,a] = residue(r,p,k);
[b,a] = residuez(r,p,k)
```

   вычисляются коэффициенты числителя и знаменателя передаточной функции по заданным векторам ее разложения — вычетов r, полюсов p и коэффициентам целой части k.

   С помощью тех же процедур осуществляется разложение заданной передаточной функции на простые дроби.

   При этом обращение к ним должно быть таким:

```
[r,p,k] = residue(b,a);
[r,p,k] = residuez(b,a)
```

1. *Вычисление нулей, полюсов и коэффициентов усиления* звена с заданной передаточной функцией можно осуществить, применяя процедуру **tf2zp** в такой форме:

```
[z,p,k] = tf2zp(b,a)
```

При этом вектор z будет содержать значения нулей передаточной функции с коэффициентами числителя (b) и знаменателя (a), вектор p — значения полюсов, а k будет равен коэффициенту усиления звена.

2. Нахождение матриц A, B, C и D, описывающих звено с заданной передаточной функцией в виде совокупности дифференциальных уравнений в форме Коши (5.39), осуществимо с помощью процедуры **tf2ss**. Если обратиться к ней в форме:

```
[A,B,C,D] = tf2ss(b,a)
```

где b и a — соответственно векторы коэффициентов числителя и знаменателя передаточной функции, то в результате получим искомые матрицы в указанном порядке.

3. *Вычисление коэффициентов решетчатого фильтра* по заданной дискретной передаточной функции можно осуществить при помощи процедуры **tf2latc**, обращаясь к ней следующим образом:

```
[k,v] = tf2latc(b,a);
[k,v] = tf2latc(1,a);
k = tf2latc(1,a);
k = tf2latc(b)
```

Первое обращение позволяет вычислить коэффициенты k знаменателя и v — числителя решетчатого БИХ-фильтра (модель авторегрессии скользящего среднего). Обращение во второй форме дает возможность определить вектор коэффициентов знаменателя k и скалярный коэффициент v, когда БИХ-фильтр имеет только полюсы (не имеет нулей). В третьей форме определяются только коэффициенты знаменателя решетчатого фильтра. Наконец, четвертая форма предназначена для нахождения вектора k коэффициентов решетчатого КИХ-фильтра (задаваемого только вектором b коэффициентов числителя передаточной функции).

Другие преобразования

1. Вычисление коэффициентов решетчатого представления по коэффициентам полинома можно осуществить, используя функцию **poly2rc**. Обращение:

```
k = poly2rc(a)
```


позволяет найти коэффициенты решетчатого представления k по коэффициентам a заданного полинома. Вектор a должен содержать только вещественные элементы, и должно выполняться условие a≠0. Размер вектора k на единицу меньше размера вектора a коэффициентов полинома.
По коэффициентам решетчатого представления очень просто определить, находятся ли все полюсы внутри единичного круга. Для этого достаточно проверить, что все элементы вектора k по абсолютной величине не превышают единицы.
Обратная задача вычисления коэффициентов полинома по коэффициентам решетчатого представления решается путем применения функции **rc2poly**:

```
a = rc2poly(k)
```

2. Процедура **sos2ss** определяет матрицы (5.39) A, B, C и D, описывающие звено в пространстве состояний, по заданной матрице SOS каскадной формы (5.38):

```
[A,B,C,D] = sos2ss(SOS)
```

Элементы матрицы SOS должны быть вещественными.
Обратный переход осуществляется при помощи функции **ss2sos**:

```
SOS = ss2sos(A,B,C,D)
```

3. Функция **sos2zp** дает возможность определить (5.40) векторы z нулей, p — полюсов и коэффициент усиления k звена, заданного каскадной формой передаточной функции (т.е. матрицей SOS (5.38)):

```
[z,p,k] = sos2zp(SOS)
```

Обратный переход осуществляется при помощи функции **zp2sos**:

```
SOS = zp2sos(z,p,k)
```

4. Нахождение нулей z, полюсов p и коэффициента усиления k звена по его описанию в пространстве состояний можно произвести путем обращения к процедуре **ss2zp** по форме:

```
[z,p,k] = ss2zp(A,B,C,D,iu)
```

где iu — номер входа, по которому ищется передаточная функция.
Обратное преобразование осуществляется процедурой **zp2ss**:

```
[A,B,C,D] = zp2ss(z,p,k)
```

5.4.2. Разработка аналоговых фильтров

Цель разработки фильтров заключается в обеспечении частотно-зависимого изменения заданной последовательности данных (сигнала). В простейшем случае разработки фильтра низких частот целью является построение такого звена, которое обеспечило бы отсутствие амплитудных искажений входного сигнала в области частот от 0 до некоторой заданной и эффективное подавление гармонических компонент с более высокими частотами.

Аналоговый фильтр может быть представлен непрерывной передаточной функцией:

$$W(s) = \frac{Y(s)}{X(s)} = \frac{M(s)}{N(s)} \qquad (5.41)$$

где $Y(s)$ и $X(s)$ — изображения по Лапласу соответственно выходного и входного сигналов фильтра, а $M(s)$ и $N(s)$ — полиномы от s соответственно в числителе и знаменателе передаточной функции.

В качестве основных характеристик фильтра обычно принимают так называемую *характеристику затухания* $A(\omega)$, которая является величиной, обратной модулю частотной передаточной функции, и измеряется в децибелах:

$$A(\omega) = 20 \cdot \lg\{|W(j\omega)|^{-1}\} = 10 \cdot \lg L(\omega^2) \qquad (5.42)$$

фазовую характеристику $\vartheta(\omega)$:

$$\vartheta(\omega) = \arg(H(j\omega)) \qquad (5.43)$$

и *характеристику групповой задержки* τ:

$$\tau = \frac{d\vartheta(\omega)}{d\omega} \qquad (5.44)$$

Функцию:

$$L(s^2)=[H(s)\cdot H(-s)]^{-1} \qquad (5.45)$$

называют *функцией затухания*.

Нетрудно понять, что если z_i являются нулями передаточной функции $W(s)$, а p_i — ее полюсами, то нулями функции затухания будут $\pm p_i$, а полюсами $\pm z_i$.

Идеальный фильтр низких частот (ФНЧ) пропускает только низкочастотные составляющие. Его характеристика затухания имеет вид, показанный на рис. 5.44 а. Диапазон частот от 0 до ω_c называется *полосой пропускания*, остальной частотный диапазон — *полосой задерживания. Граница между этими полосами* (ω_c) *называется частотой среза.* Аналогично, идеальные фильтры высоких частот (ФВЧ), полосовой и режекторный можно определить как фильтры, имеющие характеристики затухания, показанные на рис. 5.44 б, в и г.

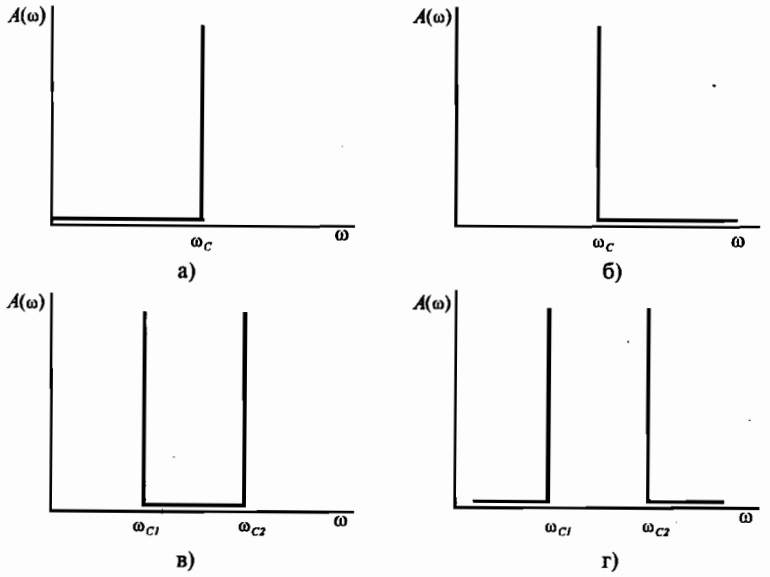

Рис. 5.44

Реальный ФНЧ отличается от идеального тем, что:

- затухание в полосе пропускания не равно нулю (децибел);
- затухание в полосе задерживания не равно бесконечности;
- переход от полосы пропускания к полосе задерживания происходит постепенно (не скачкообразно).

Возможный вид характеристики затухания реального фильтра приведен на рис. 5.45 а. На нем обозначено:

- ω_P — граничная частота полосы пропускания;
- ω_S — граничная частота полосы задерживания;
- R_P — максимальное подавление в полосе пропускания;
- R_S — минимальное подавление в полосе задерживания.

Частота среза в этом случае является условной границей между полосами пропускания и задерживания, которая определяется либо по уровню подавления в 3 дБ, либо как $\sqrt{\omega_P \omega_S}$ в эллиптических фильтрах.

Типичные характеристики затухания реальных фильтров высоких частот, полосовых и режекторных представлены на рис. 5.45 б, в и г соответственно.

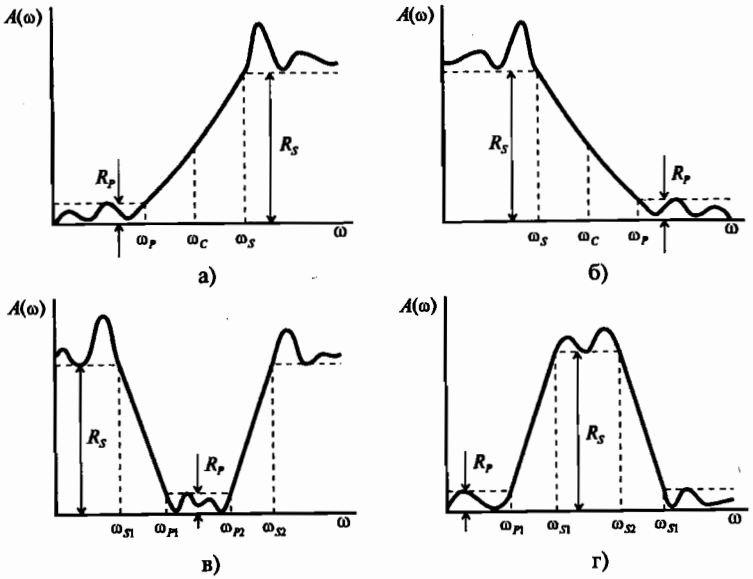

Рис. 5.45

Аппроксимацией фильтра называют реализуемую передаточную функцию, у которой график характеристики затухания $A(\omega)$ как функция от частоты приближается к одной из идеальных характеристик на рис. 5.44. Такая передаточная функция характеризует устойчивое физически реализуемое звено и должна удовлетворять следующим условиям:

- она должна быть рациональной функцией от s с вещественными коэффициентами;
- ее полюсы должны лежать в левой полуплоскости s-плоскости;
- степень полинома числителя должна быть меньшей или равной степени полинома знаменателя.

В пакете SIGNAL предусмотрен ряд процедур, осуществляющих расчет аналоговых аппроксимаций фильтров низких частот.

Группа процедур **buttord**, **cheb1ord**, **cheb2ord**, **ellipord** используется для определения минимального порядка и частоты среза аналогового или цифрового фильтра по заданным характеристикам фильтра:

- Wp — граничной частоте пропускания;
- Ws — граничной частоте задерживания;
- Rp — максимально допустимому подавлению в полосе пропускания, дБ;
- Rs— минимально допустимому подавлению в полосе задерживания, дБ.

Все эти процедуры предназначены для вычисления соответствующей аппроксимации ФНЧ, ФВЧ, полосовых и режекторных фильтров минимального порядка.

Функция **buttord** определяет порядок n и частоту среза Wn для аппроксимации в виде аналогового фильтра Баттерворта при обращении вида:

```
[n,Wn] = buttord(Wp,Ws,Rp,Rs,'s')
```

При этом значения частот Wp, Ws должны быть заданы в радианах в секунду, значение частоты среза Wn также получается в радианах в секунду. Для ФВЧ величина Wp должна превышать Ws. Для полосовых и режекторных фильтров Wp и Ws должны быть двухэлементными векторами, определяющими граничные частоты полос, причем первой должна стоять меньшая частота. В этом случае параметр Wn, вычисляемый процедурой, представляет собой двухэлементный вектор-строку.

Аналогично используются процедуры **cheb1ord**, **cheb2ord**, **ellipord**, которые определяют порядок аналоговых фильтров Чебышева 1-го и 2-го типов и эллиптического фильтра соответственно.

Вторая группа процедур позволяет определить векторы z нулей, p — полюсов и коэффициент усиления k основных аппроксимаций линейных фильтров заданного порядка n. К таким процедурам относятся **besselap**, **buttap**, **cheb1ap**, **cheb2ap** и **ellipap**, *создающие фильтр низких частот (ФНЧ) с частотой среза, равной 1 радиан в секунду.*

Процедура **besselap** путем обращения к ней:

```
[z,p,k] = besselap(n)
```

вычисляет нули и полюсы аналогового фильтра Бесселя, процедура **buttap** при помощи аналогичного обращения создает аналоговый фильтр Баттерворта.

Аналоговый прототип фильтра Чебышева нижних частот 1-го типа, имеющий пульсации в полосе пропускания не более Rp дБ, создается процедурой **cheb1ap** таким образом:

```
[z,p,k] = cheblap(n,Rp)
```

ФНЧ Чебышева 2-го типа, имеющий величину подавления в полосе задерживания не менее Rs дБ, создается с использованием процедуры **cheb2ap** так:

```
[z,p,k] = cheb2ap(n,Rs)
```

Процедура **ellipap** путем обращения к ней:

```
[z,p,k] = ellipap(n,Rp,Rs)
```

дает возможность найти нули и полюсы эллиптического ФНЧ, обеспечивающего пульсации в полосе пропускания не более Rp дБ и подавление в полосе задерживания не менее Rs дБ.

Третью группу образуют процедуры, позволяющие пересчитать параметры рассчитанного ФНЧ известной аппроксимации с частотой среза, равной 1, в ФНЧ, ФВЧ, полосовой или режекторный фильтр с заданной частотой среза. Эта группа состоит из процедур **lp2lp**, **lp2hp**, **lp2bp** и **lp2bs**. Первая образует фильтр нижних частот, вторая — ФВЧ, третья — полосовой фильтр и четвертая — режекторный фильтр. Обращение к процедуре **lp2lp** может иметь две формы:

```
[bt,at] = lp2lp(b,a,Wo)
[At,Bt,Ct,Dt] = lp2lp(A,B,C,D,Wo)
```

Первая форма применяется при задании исходного ФНЧ (с частотой среза 1 рад/с) в виде коэффициентов числителя (a) и знаменателя (b). Wo в этом случае означает желаемую частоту среза получаемого ФНЧ. Результатом являются вычисленные значения векторов коэффициентов числителя — at и знаменателя — bt полученного ФНЧ.

Вторая форма применяется при задании исходного ФНЧ в пространстве состояний. Результат получается также в форме матриц пространства состояний.

Применение процедуры **lp2hp** формирования ФВЧ полностью аналогично.

Процедура **lp2bp** формирования полосового фильтра тоже имеет два вида вызова:

```
[bt,at] = lp2bp(b,a,Wo,Bw)
[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw)
```

Здесь параметр Wo — это центральная частота полосы пропускания, а Bw означает ширину полосы пропускания. В остальном особенности использования и смысл обозначений сохраняются прежними.

Использование функции **lp2bs** проектирования режекторного типа полностью аналогично, за исключением того, что параметры Wo и Bw в этом случае имеют смысл центра полосы задерживания и ее ширины.

Четвертую группу образуют процедуры полной разработки фильтров указанных аппроксимаций по заданным порядку и значению частоты среза Wc. В нее входят процедуры **besself**, **butter**, **cheby1**, **cheby2** и **ellip**. Общим для них всех является следующее:

- С их помощью можно проектировать ФНЧ, ФВЧ, полосовые и режекторные фильтры соответствующей аппроксимации; если параметр Wc является скаляром и флажок high после него не указан, то проектируется ФНЧ с частотой среза Wc; если же указанный флажок в обращении есть, то в результате проектируется ФВЧ; если параметр Wc задан как вектор из двух величин, то результатом вычислений являются параметры полосового фильтра с полосой пропускания $W1 \le \omega \le W2$, где $W1$ — первый элемент этого вектора, а $W2$ — второй элемент; наконец, если дополнительно к этому в конце списка входных параметров указан флажок stop, то рассчитываются параметры режекторного фильтра с полосой задерживания, указанной элементами вектора *Wc*.
- Результаты расчета фильтра могут иметь три формы в зависимости от того, какое количество параметров указано при обращении к процедуре в качестве выходных, например:

```
[b,a] = besself(n,Wc,'ftype')
[z,p,k] = besself(n,Wc,'ftype')
[A,B,C,D] = besself(n,Wc,'ftype')
```

Если указано два выходных параметра, то им будут присвоены значения коэффициентов числителя и знаменателя передаточной функции фильтра; при указании трех параметров на выходе, они примут значения векторов нулей, полюсов и коэффициента усиления фильтра; если же выходов указано четыре, то ими становятся значения матриц пространства состояний проектируемого фильтра.

- Почти все они могут применяться для проектирования как аналоговых, так и цифровых фильтров; чтобы с их помощью создать аналоговый фильтр, необходимо в число входных параметров процедуры последним включить специальный флажок (s); исключение составляет фильтр Бесселя, аналога которому в цифровой форме не существует.

5.4.3. Проектирование цифровых БИХ-фильтров

Конечной задачей проектирования линейного цифрового фильтра будем считать расчет значений элементов векторов b числителя и a знаменателя его дискретной передаточной функции G(z), записанной в виде (5.7):

$$G(z) = \frac{y(z)}{x(z)} = \frac{b_0 + b_1 \cdot z^{-1} + \ldots + b_m \cdot z^{-m}}{a_0 + a_1 \cdot z^{-1} + \ldots + a_n \cdot z^{-n}}$$

Если эти два вектора известны, осуществление самой фильтрации, как было сказано ранее, происходит путем применения процедуры **filter**, в которой аргументами выступают эти векторы.

Напомним, что представленная дискретная передаточная функция описывает в сжатой форме такое конечно-разностное уравнение фильтра:

$$\begin{aligned} &a_0 \cdot y(k) + a_1 \cdot y(k-1) + a_2 \cdot y(k-2) + \ldots + a_n \cdot y(k-n) = \\ &= b_0 \cdot x(k) + b_1 \cdot x(k-1) + \ldots + b_m \cdot x(k-m) \end{aligned} \qquad (5.46)$$

Если n=0, фильтр называют *нерекурсивным*, а число m — *порядком* фильтра. Такой фильтр имеет конечную импульсную характеристику, поэтому его также называют КИХ-фильтром.

В случае n>0 фильтр называется *рекурсивным*. Порядком фильтра при этом называют наибольшее из чисел m и n. В этом случае импульсная характеристика фильтра является бесконечной, и его называют также БИХ-фильтром. БИХ-фильтры представляют собой некоторые аналоги динамических звеньев.

Одним из средств проектирования БИХ-фильтров, предусмотренных в пакете Signal, является разработка соответствующего аналогового прототипа, т.е. нахождение передаточной функции по Лапласу непрерывного фильтра, и последующий переход к цифровому фильтру путем нахождения цифрового аналога непрерывного звена. Последнее можно осуществить с помощью билинейного преобразования s-плоскости в z-плоскость. Билинейное преобразование выполняется в соответствии с выражением:

$$H(z) = H(s)\Big|_{s = 2f_s \frac{z-1}{z+1}} \qquad (5.47)$$

где f_s — частота дискретизации сигнала. При этом ось $j\omega$ преобразуется в единичную окружность на z-плоскости.

В пакете Signal билинейное преобразование осуществляется с помощью процедуры **bilinear**, к которой можно обратится тремя способами:

```
[bd,ad] = bilinear(b,a,Fs,Fp);
[zd,pd,kd] = bilinear(z,p,k,Fs,Fp);
[Ad,Bd,Cd,Dd] = bilinear(A,B,C,D,Fs,Fp)
```

Все они преобразуют параметры, характеризующие аналоговый прототип фильтра, в аналогичные параметры, описывающие дискретный БИХ-фильтр. Вид и количество входных параметров определяют вид и число выходных. Параметр Fs задает частоту дискретизации в герцах. Параметр Fp не обязателен. Он определяет частоту в герцах, для которой значения АЧХ до и после выполнения преобразования должны совпадать, т.е. задает так называемые предыскажения.

Обращение в первой форме позволяет определить коэффициенты полиномов числителя и знаменателя дискретной передаточной функции фильтра вида (5.35) по заданным коэффициентам полиномов числителя и знаменателя непрерывной передаточной функции вида (5.34). Обращение во второй форме дает возможность вычислить нули, полюсы и коэффициент усиления дискретного фильтра по заданным аналогичным параметрам аналогового прототипа. И наконец, третья форма определяет матрицы дискретного пространства состояний фильтра по известным матрицам непрерывного пространства состояний.

Второй способ построения цифрового фильтра по его аналоговому прототипу заключается в таком преобразовании параметров аналогового фильтра в параметры дискретного фильтра, при котором импульсная характеристика последнего совпадала бы с импульсной характеристикой аналогового фильтра в точках через дискрет по времени. Это в MatLAB осуществляется применением процедуры **impinvar**:

```
[bz,az] = impinvar(b,a,Fs)
```

Здесь b и a — заданные векторы коэффициентов числителя и знаменателя передаточной функции аналогового прототипа фильтра, bz и az — вычисляемые коэффициенты числителя и знаменателя дискретной передаточной функции дискретного фильтра, Fs — заданная частота дискретизации сигнала в герцах. Если параметр Fs при обращении не указан, то по умолчанию он принимается равным 1 Гц.

Третий способ формирования дискретных фильтров — использование ранее рассмотренных процедур формирования фильтров **butter**, **cheby1**, **cheby2** и **ellip**. Если при обращении к этим процедурам не указывать в конце списка входных параметров флажка s, то результатом работы этих процедур будут параметры именно цифровых фильтров.

Основное отличие применения этих функций для разработки цифровых фильтров заключается в другом представлении задаваемых частот в векторе Wc. *Все частоты должны задаваться по отношению к так называемой частоте Найквиста. Частотой Найквиста называют половину частоты дискретизации сигнала.*

Так как диапазон частот изменения дискретного сигнала всегда меньше частоты дискретизации, то все частотные характеристики дискретных фильтров определяются только внутри диапазона от 0 до частоты Найквиста. Поэтому все задаваемые в векторе Wc граничные частоты должны быть меньше единицы.

Существуют еще две процедуры расчета БИХ-фильтров.

Процедура **maxflat** производит расчет обобщенного цифрового фильтра Баттерворта. Формы обращения к ней таковы:

```
[b,a] = maxflat(nb,na,Wc)
[b,a] = maxflat(nb,'sym',Wc)
[b,a,b1,b2] = maxflat(nb,na,Wc)
[b,a] = maxflat(nb,na,Wc,'design_flag')
```

Первое обращение позволяет вычислить коэффициенты b — числителя и a — знаменателя дискретной передаточной функции H(z) цифрового ФНЧ Баттерворта с частотой среза Wc, порядок числителя которой равен nb, а знаменателя — na.

При обращении второго вида вычисляются коэффициенты цифрового симметричного КИХ-фильтра Баттерворта. В этом случае na принимается равным 0. Параметр nb должен быть четным.

Если обратиться к процедуре так, как указано в третьем обращении, указать в качестве выходных четыре величины, то дополнительные параметры b1 и b2 дадут коэффициенты двух полиномов, произведение которых является полиномом числителя b искомой дискретной передаточной функции, причем все нули полинома b1 равны -1, а полином b2 содержит все остальные нули полинома b.

Добавление в список входных параметров процедуры параметра 'design_flag' позволяет изменять характер выводимой на экран информации. Если значение этого параметра равно trace, на экране отображаются параметры, используемые в процессе проектирования:

```
» nb=10;  na = 2;  w = 0.5;
» [b,a,b1,b2] = maxflat(nb,na,w,'trace')
Table:
       L          M          N     wo_min/pi    wo_max/pi
    10.0000       0       2.0000        0         0.2394
     9.0000    1.0000     2.0000     0.2394       0.3259
     8.0000    2.0000     2.0000     0.3259       0.3991
     7.0000    3.0000     2.0000     0.3991       0.4669
     6.0000    4.0000     2.0000     0.4669       0.5331
     5.0000    5.0000     2.0000     0.5331       0.6009
     4.0000    6.0000     2.0000     0.6009       0.6741
     3.0000    7.0000     2.0000     0.6741       0.7606
     2.0000    8.0000     2.0000     0.7606       1.0000
b =
  Columns 1 through 7
  0.0414  0.2263  0.4932  0.5252  0.2494  0.0079  -0.0266
  Columns 8 through 11
  0.0008  0.0023  -0.0004 -0.0000
a =
    1.0000          0    0.5195
b1 =
     1     6    15    20    15     6     1
b2 =
    0.0414   -0.0221    0.0049   -0.0004   -0.0000
```

Если же этому параметру задать значение plots, то на экран будут выведены графики амплитудной характеристики, группового времени замедления, а также графическое изображение нулей и полюсов:

```
» [b,a,b1,b2] = maxflat(nb,na,w,'plots')
b =
  Columns 1 through 7
  0.0414  0.2263  0.4932  0.5252  0.2494  0.0079  -0.0266
  Columns 8 through 11
  0.0008  0.0023 -0.0004 -0.0000
a =    1.0000    0    0.5195
b1 =    1    6    15    20    15    6    1
b2 =-0.0221    0.0049   -0.0004   -0.0000
```


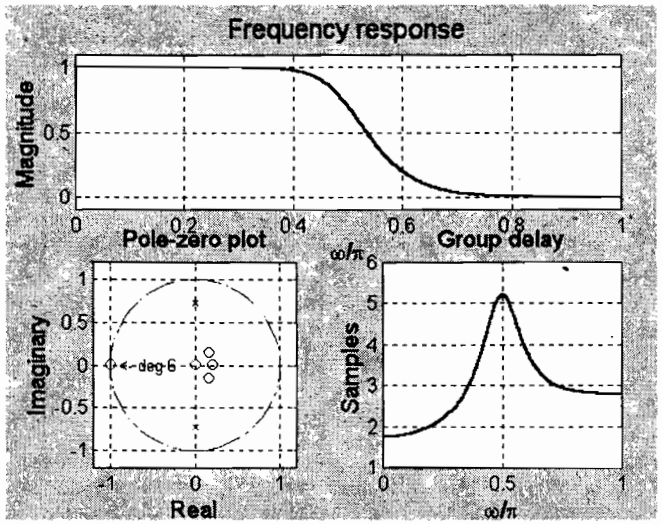

Рис. 5.46

Расчет БИХ-фильтра по заданной амплитудно-частотной характеристике производится процедурой **yulewalk**. Если набрать в командном окне MatLAB строку:

```
»  [b,a] = yulewalk(n,f,m)
```

то будут вычислены коэффициенты b числителя и a знаменателя дискретной передаточной функции БИХ-фильтра порядка n, АЧХ которого задана векторами f (частоты в нормированных значениях) и m — соответствующих значений отношений амплитуд выхода и входа. Первый элемент вектора f должен быть равен 0, а последний — 1. Все остальные элементы должны быть расположены в неубывающем порядке. Частоты, при которых происходит скачок АЧХ, указываются два раза с разными значениями соответствующих им амплитуд.

Приведем пример расчета ФНЧ 8-го порядка и построим желаемую АЧХ и АЧХ полученного фильтра (рис. 5.47).

```
» f = [0 0.5 0.5 1];
» m=[1 1 0 0];
» [b,a] = yulewalk(8,f,m);
» [h,w] = freqz(b,a,128);
» plot(f,m, w/pi,abs(h))
» grid, title('Пример использования процедуры YULEWALK')
» xlabel('Нормализованная частота')
» ylabel('А Ч Х')
```

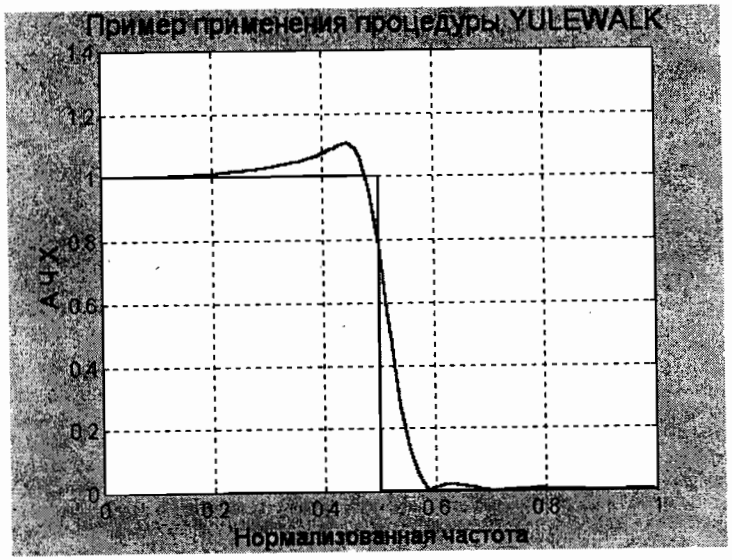

Рис. 5.47

5.4.4. Проектирование КИХ-фильтров

В отличие от БИХ-фильтров, которые характеризуются двумя векторами — коэффициентов b числителя и a знаменателя своей дискретной передаточной функции, КИХ-фильтры описываются только одним вектором b. Знаменатель их дискретной передаточной функции тождественно равен 1.

Группа функций **fir1** предназначена для расчета коэффициентов b цифрового КИХ-фильтра с линейной фазой методом взвешивания с использованием окна. Общий вид обращения к этой процедуре:

```
b = fir1(n,Wn,'ftype',window)
```

Процедура вычисляет вектор n+1 коэффициентов b КИХ-фильтра с нормализованной частотой среза Wn.

Параметр 'ftype' задает желаемый тип фильтра (ФНЧ, ФВЧ, полосовой или режекторный). Он может отсутствовать (и тогда по умолчанию рассчитываются параметры ФНЧ с частотой среза Wn, если последняя задана как скаляр, или полосового фильтра с полосой пропускания от W1 до W2, если параметр Wn задан в виде вектора из двух элементов [W1 W2]) или принимать одно из четырех значений: high, stop, DC-1 и DC-0. В первом случае синтезируется ФВЧ с частотой среза Wn. Во втором — режекторный фильтр (при этом Wn должен быть вектором из двух элементов, значения которых определяют границы полосы задерживания по отношению к частоте Найквиста). В третьем случае рассчитываются параметры многополосного фильтра, первая полоса которого является полосой пропускания, а в четвертом — тоже многополосный фильтр, первая полоса которого является полосой задерживания.

При расчете режекторных фильтров и ФВЧ порядок фильтра следует назначать четным числом.

Параметр window позволяет задавать отсчеты окна в векторе-столбце window длины n+1. Если этот параметр не указан, то, по умолчанию, будет использовано окно Хемминга.

Для вычисления окон различного типа в MatLAB предусмотрены функции:

bartlett(n)	Создает вектор-столбец из n элементов окна Бартлетта
blackman(n)	Создает вектор-столбец из n элементов окна Блэкмана
boxcar(n)	Создает вектор-столбец из n элементов прямоугольного окна
chebwin(n,r)	Создает вектор-столбец из n элементов окна Чебышева, где r — желаемый уровень допустимых пульсаций в полосе задерживания в децибелах
hamming(n)	Создает вектор-столбец из n элементов окна Хэмминга
hanning(n)	Создает вектор-столбец из n элементов окна Хэннинга
kaizer(n,beta)	Создает вектор-столбец из n элементов окна Кайзера, где параметр beta определяет затухание боковых лепестков преобразования Фурье-окна
triang(n)	Создает вектор-столбец из n элементов треугольного окна

Приведем пример. Произведем расчет полосового КИХ-фильтра 24-го порядка с полосой пропускания $0.35 \le \omega/\omega_N \le 0.65$:

```
» b = fir1(48,[0.35 0.65]);
» freqz(b,1,512)
```

Результат показан на рис. 5.48.

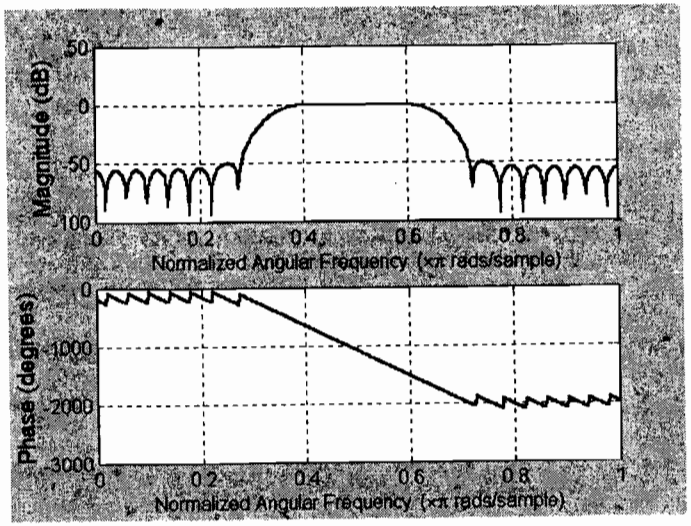

Рис. 5.48

Группа процедур **fir2** служит для расчета коэффициентов цифрового КИХ-фильтра с произвольной амплитудно-частотной характеристикой, задаваемой векторами f частот и m — соответствующих желаемых значений АЧХ. Общий вид обращения к процедуре таков:

```
b = fir2(n,f,m,npt,lap,window)
```

Вектор f должен содержать значения нормализованной частоты в неубывающем порядке от 0 до 1. Вектор m должен быть той же длины, что и вектор f, и содержать желаемые значения АЧХ на соответствующих частотах.

Параметр npt позволяет задать число точек, по которым выполняется интерполяция АЧХ. Параметр lap определяет размер (число точек) области около точек скачкообразного изменения АЧХ, в которой выполняется сглаживание. Если эти параметры не указаны, то по умолчанию принимается npt = 512 и lap = 25.

Рассчитаем двухполосный фильтр 30-го порядка:

```
» f = [0 0.2 0.2 0.6 0.6 0.8 0.8 1];
» m = [ 1 1 0 0 0.5 0.5 0 0];
» b = fir2(30,f,m);
» [h,w] = freqz(b,1,512);
» plot(f,m,w/pi,abs(h))
» grid
» title('АЧХ КИХ-фильтра (процедура FIR2)')
» xlabel('Нормализованная частота')
» ylabel('А Ч Х')
```


На рис. 5.49 приведены желаемая АЧХ и АЧХ, полученная в результате расчета.

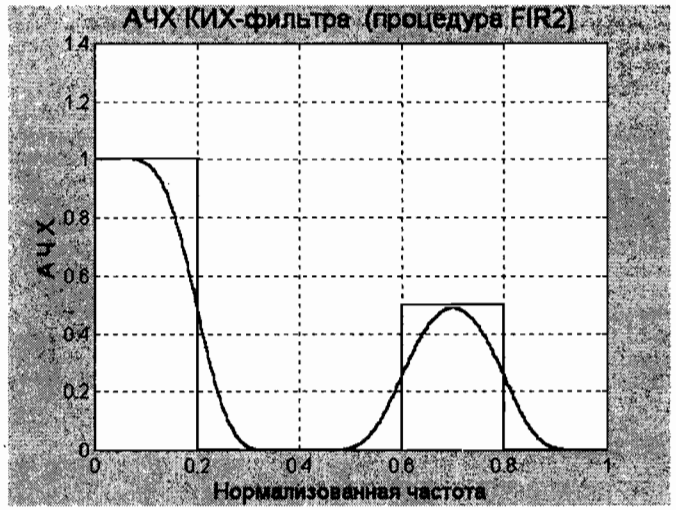

Рис. 5.49

Следующая процедура — **fircls** — также рассчитывает многополосный фильтр, но в несколько другой форме — путем задания кусочно-постоянной желаемой АЧХ.

Формат обращения к ней имеет вид:

```
b = fircls(n,f,amp,up,lo,'design_flag')
```

Здесь f, как и ранее, вектор значений нормализованных частот (от 0 до 1), определяющих границы полос фильтра. Вектор amp определяет кусочно-постоянную требуемую АЧХ фильтра, количество его элементов равно числу полос фильтра и, следовательно, на 1 меньше числа элементов вектора f. Векторы up и lo определяют соответственно верхние и нижние допустимые отклонения АЧХ спроектированного фильтра от желаемой АЧХ для каждой из полос. Размер их совпадает с размером вектора amp.

Параметр 'design_flag' может принимать три значения:

- trace — для обеспечения вывода результатов в виде текстовой таблицы;
- plots — для графического отображения АЧХ, групповой задержки, нулей и полюсов;
- both — для отображения результатов как в текстовой, так и в графической форме.

Приведем пример разработки прежнего двухполосного фильтра:

```
» n = 30;
» f = [0 0.2 0.6 0.8 1];
» amp = [1 0 0.5 0];
» up = [1.02 0.02 0.51 0.02];
» lo = [0.98 -0.02 0.49 -0.02 ];
» b = fircls(n,f,amp,up,lo,'both')
```

Результат приведен ниже и на рис. 5.50.

```
   Bound Violation = 0.0755112846369
   Bound Violation = 0.0116144793011
   Bound Violation = 0.0004154355279
   Bound Violation = 0.0000905996658
   Bound Violation = 0.0000214272508
   Bound Violation = 0.0000009624286
   Bound Violation = 0.0000002393147
   Bound Violation = 0.0000000596813
   Bound Violation = 0.0000000146532
   Bound Violation = 0.0000000036610
b =
   Columns 1 through 7
    -0.0001   -0.0031    0.0226    0.0101    0.0060    0.0011   -0.0105
   Columns 8 through 14
    -0.0231   -0.0626    0.0090   -0.0001   -0.0145    0.1775    0.1194
   Columns 15 through 21
     0.1272    0.3023    0.1272    0.1194    0.1775   -0.0145   -0.0001
   Columns 22 through 28
     0.0090   -0.0626   -0.0231   -0.0105    0.0011    0.0060    0.0101
   Columns 29 through 31
     0.226    -0.0031   -0.0001
```

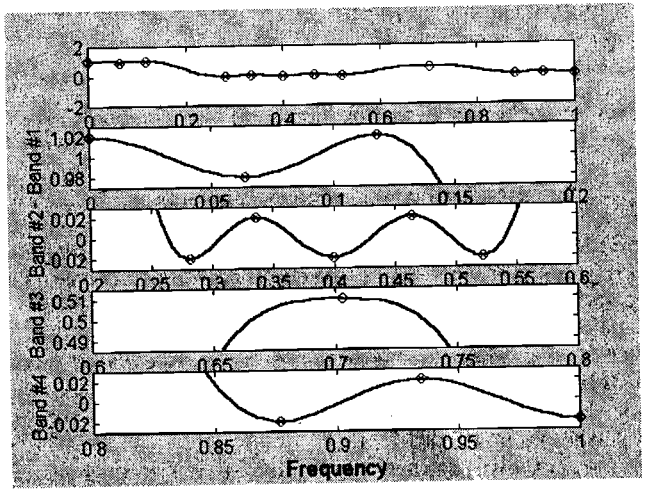

Рис. 5.50

Для сравнения с результатами работы процедуры **fir2** построим график полученной АЧХ, аналогичный приведенному на рис. 5.49:

```
» [h,w] = freqz(b,1,512);
» plot(w/pi,abs(h))
» grid
» title('АЧХ КИХ-фильтра (процедура FIRCLS)')
» xlabel('Нормализованная частота')
» ylabel('А Ч Х')
```

Результат представлен на рис. 5.51.

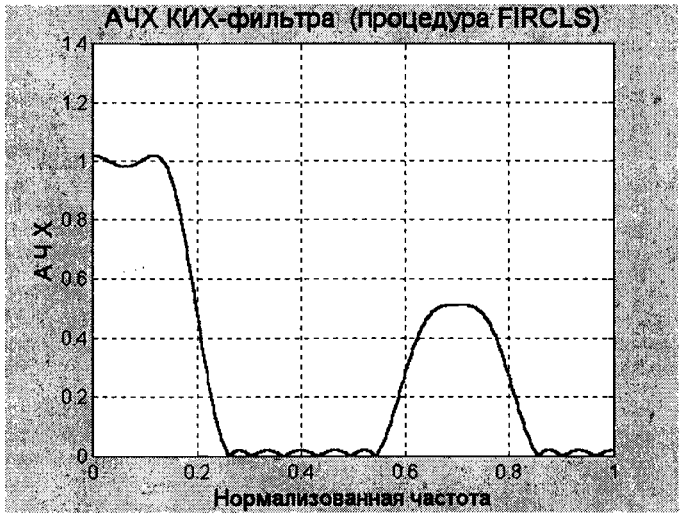

Рис. 5.51

Процедура **fircls1** предназначается для расчета параметров ФНЧ и ФВЧ с КИХ методом наименьших квадратов с учетом допусков на отклонения АЧХ. Предусмотрены следующие виды обращения к этой процедуре:

```
b = fircls1(n,Wo,dp,ds)
b = fircls1(n,Wo,dp,ds,'high')
b = fircls1(n,Wo,dp,ds,Wt)
b = fircls1(n,Wo,dp,ds,Wt,'high')
b = fircls1(n,Wo,dp,ds,Wp,Ws,k)
b = fircls1(n,Wo,dp,ds,Wp,Ws,k,'high')
b = fircls1(n,Wo,dp,ds,...,'design_flag')
```

Параметр W_o представляет собой нормализованную частоту среза; dp определяет максимально допустимое отклонение АЧХ рассчитанного фильтра от 1 в полосе пропускания, а ds — максимальное отклонение АЧХ рассчитанного фильтра от 0 в полосе задерживания.

Наличие флажка 'high' определяет, что рассчитываются параметры ФВЧ. Если этот флажок отсутствует, рассчитывается ФНЧ.

С помощью параметра Wt задается частота Wt, выше которой при Wt > Wo или ниже которой при Wt < Wo гарантируется выполнение требований к АЧХ синтезируемого фильтра.

Параметры Wp, Ws и k позволяют соответственно задать граничную частоту пропускания, граничную частоту задерживания и отношение ошибки в полосе пропускания к ошибке в полосе задерживания.

Флаг 'design_flag' имеет тот же смысл и принимает те же значения, что и у предыдущей процедуры.

Группа процедур **remez** осуществляет расчет коэффициентов цифрового КИХ-фильтра с линейной ФЧХ по алгоритму Паркса-МакКлелла, в котором использован обменный алгоритм Ремеза и метод аппроксимации Чебышева. При этом минимизируется максимальное отклонение АЧХ спроектированного фильтра от желаемой АЧХ. Приведем наиболее полный вид обращения к процедуре:

```
b = remez(n,f,a,W,'ftype')
```

Вектор f должен состоять из последовательных, в возрастающем порядке записанных пар нормализованных (от 0 до 1) частот, определяющих соответственно нижнюю и верхнюю границы диапазона полосы пропускания или задерживания. Вектор a должен содержать желаемые значения АЧХ на частотах, определяемых соответствующими элементами вектора f. Желаемая АЧХ в полосе частот от f(k) до f(k+1) при нечетном k представляет собой отрезок прямой от точки f(k), a(k) до точки f(k+1), a(k+1). В диапазонах от f(k) до f(k+1) при четном k значение желаемой АЧХ не определено (а значит, при проектировании фильтра АЧХ в этих диапазонах может принимать любое значение). Следует заметить, что значение f(1) всегда должно быть равным 0. Кроме того, необходимо, чтобы векторы f и a были одинаковой длины, причем общее количество элементов каждого вектора должно быть четным числом.

Вектор W задает значения коэффициентов веса каждой из полос АЧХ, заданных парами частот вектора f. Эти коэффициенты используются при аппроксимации АЧХ и определяют достигаемое при аппроксимации соотношение между реальным и желаемым значением АЧХ в каждом из диапазонов. Число элементов вектора W равно половине числа элементов вектора f.

Флаг 'ftype' может принимать одно из двух значений:

- hilbert — в этом случае процедура проектирует фильтры с нечетной симметрией и линейной фазой;

- differentiator — синтезируется фильтр с использованием специальных методов взвешивания; при этом для ошибок задаются веса, пропорциональные 1/f; поэтому ошибки аппроксимации на низких частотах меньше, чем на высоких; для дифференциаторов, АЧХ которых пропорциональна частоте, минимизируется максимальная относительная ошибка.

Ниже приводится пример проектирования полосового фильтра 17-го порядка:

```
» f = [0.0.3 0.4 0.6 0.7 1];
» a = [0 0 1 1 0 0];
» b = remez(17,f,a);
» [h, w] = freqz(b, 1, 512);
» plot(f,a,w/pi,abs(h)), grid
» title('АЧХ КИХ-фильтра (процедура REMEZ)')
» xlabel('Нормализованная частота')
» ylabel('А Ч Х')
```

Результат приведен на рис. 5.52.

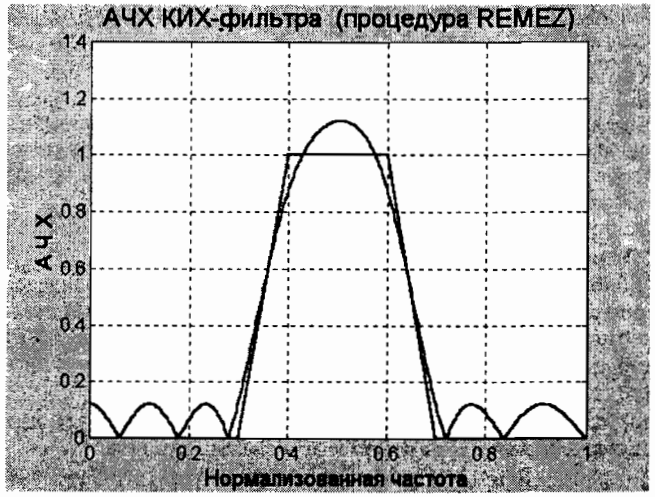

Рис. 5.52

Особенностью следующей процедуры, **cremez**, является то, что исходные данные по желаемой форме АЧХ фильтра задаются в виде функции, условно обозначенной fresp. Формы обращения к этой процедуре приведены ниже:

```
b = cremez(n,f,'fresp')
b = cremez(n,f,'fresp',w)
```

```
b = cremez(n,f,{'fresp',p1,p2,...},w)
b = cremez(n,f,a,w)
b = cremez(...,'sym')
b = cremez(...,'debug')
b = cremez(...,'skip_stage2')
[b,delta,opt] = cremez(...)
```

Параметры n, f имеют тот же смысл и такие же требования к их представлению, как при применении процедуры **remez**. В отличие от последней, вектор значений желаемой АЧХ, соответствующих заданным значениям вектора f, определяется путем обращения к функции fresp.

Функция fresp может принимать одно из следующих значений:

- lowpass, highpass, bandpass, bandstop (ФНЧ, ФВЧ, полосовой и режекторный фильтры) — при этом рассчитываются параметры указанного типа фильтра; если к функции fresp не указаны дополнительные параметры (первые два вида обращений к процедуре), то групповое время замедления (ГВЗ) принимается равным n/2; в случае же обращения к процедуре в третьей форме, где в качестве дополнительного параметра функции fresp указан один — d, ГВЗ = n/2+d;
- multiband (многополосный фильтр) — синтезируется фильтр, заданный вектором a желаемой АЧХ при значениях частот, определенных вектором f; при этом вектор a указывается в качестве первого дополнительного параметра к функции multiband (третья форма обращения); если кроме этого вектора не указаны другие дополнительные параметры, то ГВЗ принимается равным n/2, если же указан еще один дополнительный параметр — d, то ГВЗ = n/2+d;
- differentiator (дифференциатор) — эта функция позволяет рассчитывать коэффициенты дифференцирующего фильтра с линейной фазой; при обращении к этой функции в качестве дополнительного параметра необходимо указать частоту дискретизации Fs; по умолчанию Fs=1;
- hilbfilt (фильтр Гильберта) — в этом случае находятся коэффициенты фильтра Гильберта с линейной фазой.

Четвертая форма обращения к процедуре эквивалентна следующей:

```
b = cremez(n,f,{'multiband',a},w)
```


Параметр sym позволяет задать тип симметрии *импульсной характеристики* (ИХ) фильтра. Он может принимать следующие значения:

- none — в этом случае ИХ может быть произвольной; это значение параметра используется по умолчанию, если при определении желаемой АЧХ задаются отрицательные значения частот;
- even — АЧХ должна быть вещественной с четным типом симметрии; такое значение параметра используется по умолчанию при проектировании ФНЧ, ФВЧ, полосовых и режекторных фильтров;
- odd — АЧХ должна быть вещественной с нечетным типом симметрии; такое значение по умолчанию используется при проектировании фильтров Гильберта и дифференциаторов;
- real — АЧХ должна иметь сопряженный тип симметрии.

Использование флага skip_stage2 (седьмой вид обращения к процедуре) позволяет не выполнять второй этап алгоритма оптимизации, который рассчитывает коэффициенты фильтра в тех случаях, когда этого нельзя сделать с помощью алгоритма Ремеза. Исключение второго этапа сокращает время расчетов, но может повлечь снижение точности. По умолчанию выполняются оба этапа оптимизации. Параметр debug (см. шестой тип вызова процедуры) определяет вид выводимых на экран результатов расчета фильтра и может принимать следующие значения: trace, plots, both и off. По умолчанию используется off (т.е. на экран не выводится информация).

Использование дополнительного *выходного параметра* delta (см. восьмой вид обращения к процедуре) дает возможность применять в дальнейших операциях значение *максимальной амплитуды пульсаций АЧХ*.

Выходной параметр opt содержит набор дополнительных характеристик:

- opt.grid — вектор отсчетов частоты, использованных при оптимизации;
- opt.H — вектор значений АЧХ, соответствующих значениям элементов в векторе opt.grid;
- opt.error — вектор значений ошибок на частотах вектора opt.grid;
- opt.fextr — вектор, содержащий частоты с экстремальными ошибками АЧХ.

На рис. 5.53 изображен результат применения процедуры **cremez** для расчета параметров полосового КИХ-фильтра 30-го порядка.

```
» b = cremez(30,[0 0.5 0.6 0.8 0.9 1],'bandpass');
» freqz(b,1,512)
```

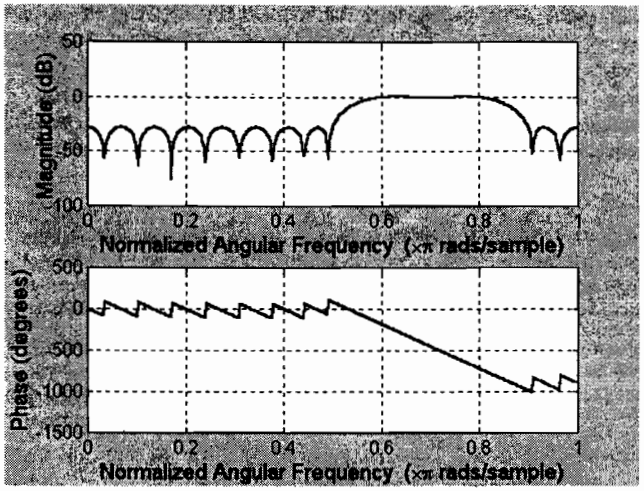

Рис. 5.53

5.5. Графические и интерактивные средства

5.5.1. Графические средства пакета Signal

Некоторые графические средства пакета Signal уже упоминались ранее. К ним относятся, прежде всего, процедуры **freqs** и **freqz**, применение которых без выходных параметров приводит к построению в графическом окне (фигуре) графиков АЧХ и ФЧХ аналогового звена по заданным векторам коэффициентов числителя и знаменателя передаточной функции по Лапласу (для первой из них) либо цифрового фильтра (звена) по коэффициентам его дискретной передаточной функции (для второй процедуры). Напомним, что общая форма вызова этих функций при выведении графиков такова:

```
freqs(b,a,n)   или   freqz(b,a)
```

При этом b и a представляют собой векторы коэффициентов числителя и знаменателя передаточной функции, а n задает число отсчетов в строящихся АЧХ и ФЧХ.

Пример применения функции **freqs** приведен на рис. 5.16, а функции **freqz** — на рис. 5.17. Из графиков видно, что:

- первая процедура строит АЧХ в логарифмическом масштабе, а вторая — в децибелах;
- частоты в первом случае откладываются в радианах в секунду и в логарифмическом масштабе, а во втором — в виде отношения к частоте Найквиста, в равномерном масштабе и в диапазоне от 0 до 1;

- форма оформления графиков достаточно жесткая и не предусматривает возможности изменения размеров графиков, надписей по осям и вывода заголовка.

Некоторые процедуры расчета фильтров, такие как **fircls**, **fircls1**, **cremez** и **maxflat** предусматривают выведение соответствующих графических изображений некоторых параметров спроектированного фильтра, если в качестве последнего входного параметра при обращении к процедуре указан флаг 'plots'.

Так, функция **maxflat** в этом случае выводит три графические зависимости:

- АЧХ в пределах до частоты Найквиста в равномерном масштабе;
- карту расположения нулей и полюсов в комплексной Z-плоскости;
- частотный график групповой задержки фильтра.

Например:

```
» [b,a,b1,b2] = maxflat(10,2,0.5,'plots')
```

приводит к появлению в графическом окне изображения, показанного на рис. 5.54.

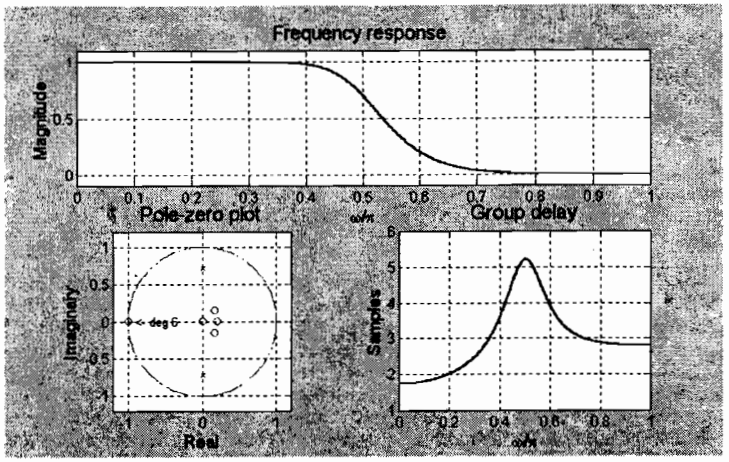

Рис. 5.54

При вызове функции **fircls** с этим флагом на график выводятся фрагменты АЧХ с максимальными отклонениями от требуемой АЧХ (рис. 5.55):

```
» n = 30;
» f = [0 0.2 0.6 0.8 1];
» amp = [1 0 0.5 0];
» up = [1.02 0.02 0.51 0.02];
```

```
» lo = [0.98 -0.02 0.49 -0.02];
» fircls(n,f,amp,up,lo,'plots')
```

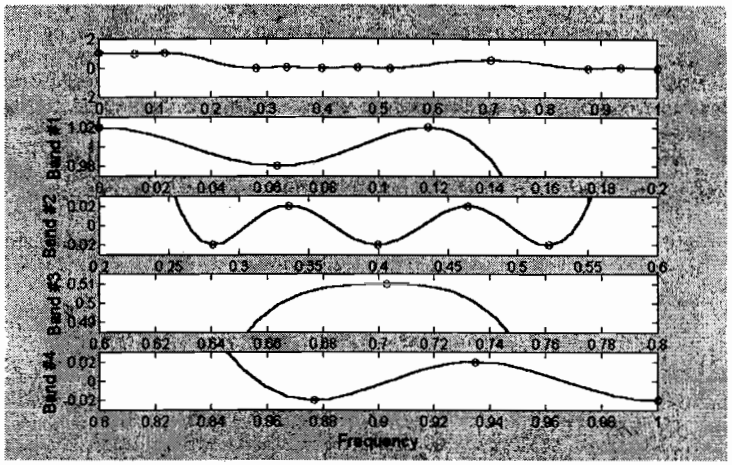

Рис. 5.55

Аналогичные графики строятся и при вызове функции **fircls1**. Отличие в том, что теперь графики не снабжены никаким текстом (рис. 5.56):

```
» fircls1(n,0.5,0.01,0.01,'plots')
```

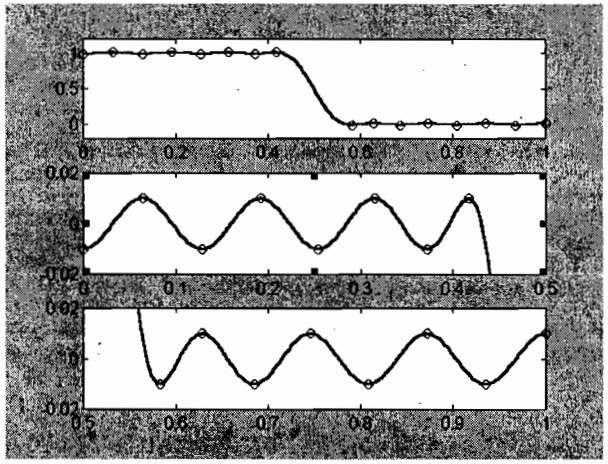

Рис. 5.56

Процедура **cremez** при таком обращении выводит следующие графики (в одном графическом окне): АЧХ, ФЧХ, зависимость погрешности по амплитуде от частоты и зависимость погрешности по фазе от частоты. Это продемонстрировано на рис. 5.57:

```
» cremez(30,[0 0.5 0.6 0.8 0.9 1],'bandpass','plots')
```

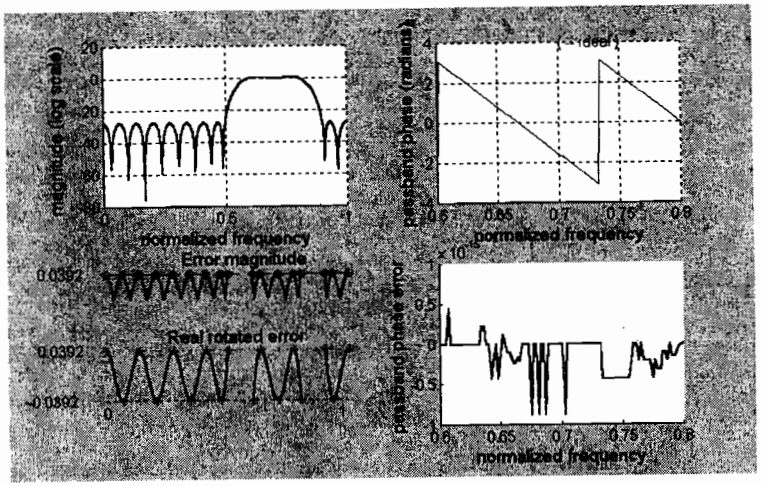

Рис. 5.57

В пакете Signal имеются еще три важные для инженера графические процедуры: **grpdelay**, **impz** и **zplane**. Первая строит график группового времени замедления (ГВЗ) от частоты, вторая — импульсную характеристику заданного фильтра, а третья отображает на комплексной Z-плоскости положение нулей и полюсов фильтра.

Рассмотрим в качестве примера применение этих процедур к БИХ-фильтру, созданному процедурой **maxflat**:

```
» [b,a] = maxflat(10,2,0.5) ;
» grpdelay(b,a,128)
```

Результат применения функции **grpdelay** приведен на рис. 5.58. Применяя процедуру **impz** к тому же фильтру:

```
» impz(b,a)
```

получим график импульсной дискретной характеристики фильтра, изображенный на рис. 5.59.

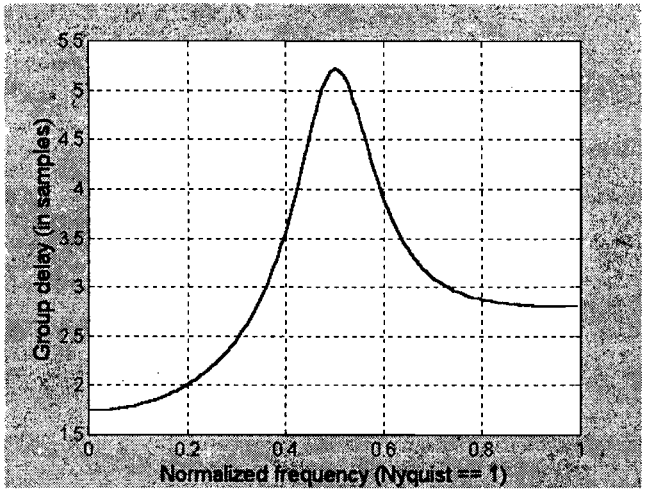

Рис. 5.58

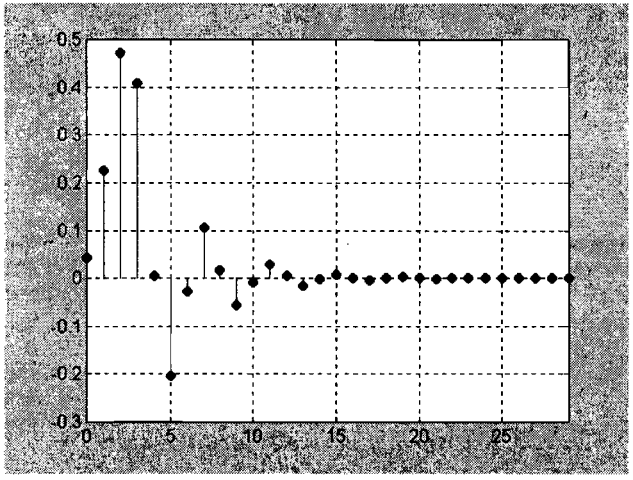

Рис. 5.59

Использование процедуры **zplane** для этого фильтра:

```
» zplane(b,a)
```

приводит к построению графика, показанного на рис. 5.60.

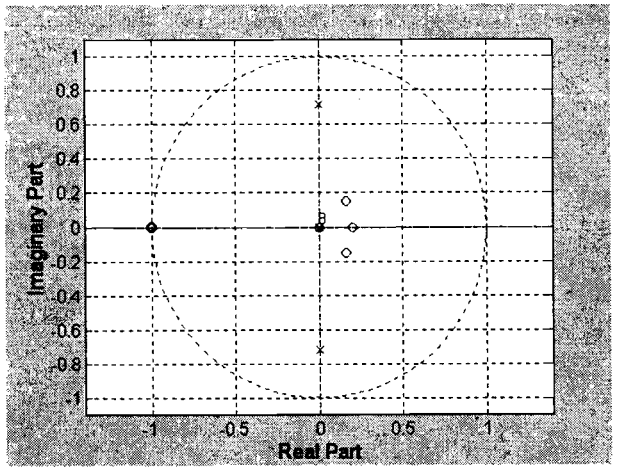

Рис. 5.60

Рассмотрим применение некоторых графических функций на примере двух коррелированных случайных процессов. Для этого вначале сформируем эти процессы:

```
Ts=0.01;   T = 100;   % Задание параметров процесса
t=0 : Ts : T;
x1=randn(1,length(t));   % Формирование белого шума
% Расчет параметров формирующего фильтра
om0=2*pi;   dz=0.05;   A=1;   oms=om0*Ts;
a(1)= 1+2*dz*oms+oms^2;
a(2)= -2*(1+dz*oms);
a(3)=1;
b(1)=A*oms^2;
% Формирование "профильтрованного" процесса
y1 =filter(b,a,x1);
% Построение графика процесса
subplot(3,1,1)
plot(t,y1),grid, set(gca,'FontName','Arial
Cyr','FontSize',8)
title('Процесс на выходе фильтра (T0=1; dz=0.05,
Ts=0.01)');
ylabel('Y1(t)')
% Расчет параметров первого звена
om0=2*pi*0.20;   dz=0.05;   A=1;   oms=om0*Ts;
a1(1)= 1+2*dz*oms+oms^2;
a1(2)= -2*(1+dz*oms);
a1(3)=1;
b1(1)=A*oms^2;
% Формирование "первого" процесса
x =filter(b1,a1,y1);
% Построение графика первого процесса
subplot(3,1,2)
```

```
plot(t,x),grid, set(gca,'FontName','Arial Cyr','FontSize',8)
title('Первый случайный процесс (T0=5; dz=0.05, Ts=0.01)');
%xlabel('Время (c)');
ylabel('X(t)')
% Расчет параметров второго звена
om0=2*pi*0.5;    dz=0.05;    A=1;    oms=om0*Ts;
a2(1)= 1+2*dz*oms+oms^2;
a2(2)= -2*(1+dz*oms);
a2(3)=1;
b2(1)=A*oms^2;
% Формирование второго процесса
y =filter(b2,a2,y1);
% Построение графика второго процесса
subplot(3,1,3)
plot(t,y),grid, set(gca,'FontName','Arial Cyr','FontSize',8)
title('Второй случайный процесс (T0=2; dz=0.05, Ts=0.01)');
xlabel('Время (c)');
ylabel('Y(t)')
```

Графики порождающего процесса и двух процессов, производных от него, приведены на рис. 5.61.

Представление графика длинного процесса в виде совокупности нескольких фрагментов меньшей длины по аргументу можно осуществить при помощи процедуры **strips** путем такого обращения к ней:

```
strips(x,sd,Fs,scale)
```

где x — вектор значений выводимой на график функции, sd — параметр, задающий в секундах длину одного фрагмента по аргументу, Fs — значение частоты дискретизации, scale — масштаб по вертикальной оси.

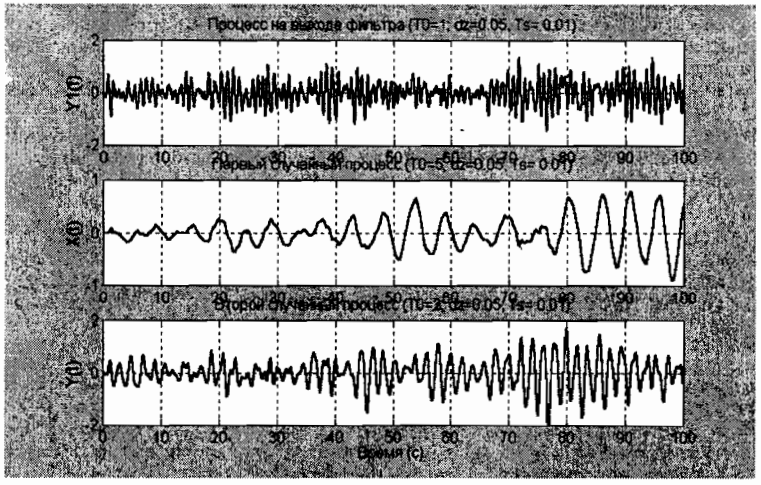

Рис. 5.61

В качестве примера выведем график порождающего случайного процесса, разбивая его на отдельные фрагменты по 20 секунд и задавая диапазон изменения значения функции в каждом фрагменте от -2 до 2:

```
» strips(y1,20,100,2)
» grid; title (Применение процедуры STRIPS для вывода Y1(t)');
» xlabel('Время, c')
```

На рис. 5.62 представлен результат.

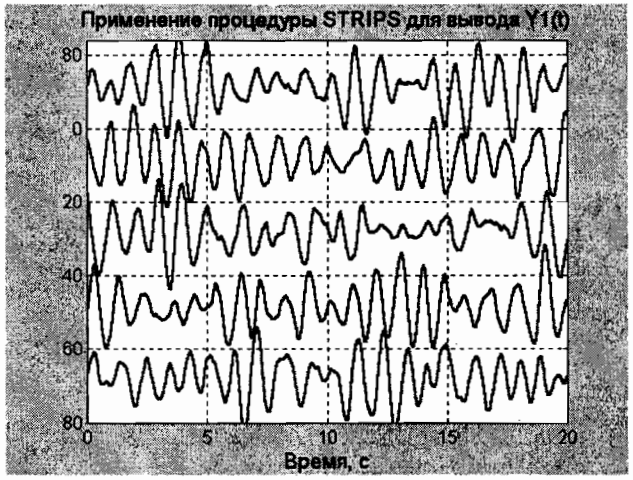

Рис. 5.62

Теперь познакомимся с графическими процедурами статистической обработки процессов. Ранее (разд. 5.3) была описана функция **psd**, которая, если не указывать выходных параметров, выводит в графическое окно график спектральной плотности мощности (рис. 5.42). Аналогичный график зависимости модуля взаимной спектральной плотности двух сигналов от частоты строит процедура **csd**, если обратиться к ней таким образом:

```
csd(x,y,nfft,Fs)
```

Здесь x и y — заданные последовательности отсчетов двух сигналов, nfft — число отсчетов, по которым вычисляется взаимная спектральная плотность, Fs — частота дискретизации этих сигналов.

Применим функцию **psd** к случайному сигналу X(t), а процедуру **csd** — для нахождения взаимной спектральной плотности сигналов X(t) и Y(t). Результаты приведены соответственно на рис. 5.63 и 5.64.

```
» psd(x,10000,100); title ('Применение процедуры PSD...
  к процессу X(t)');
» ylabel('Спектральная плотность); xlabel('Частота, Гц')
```

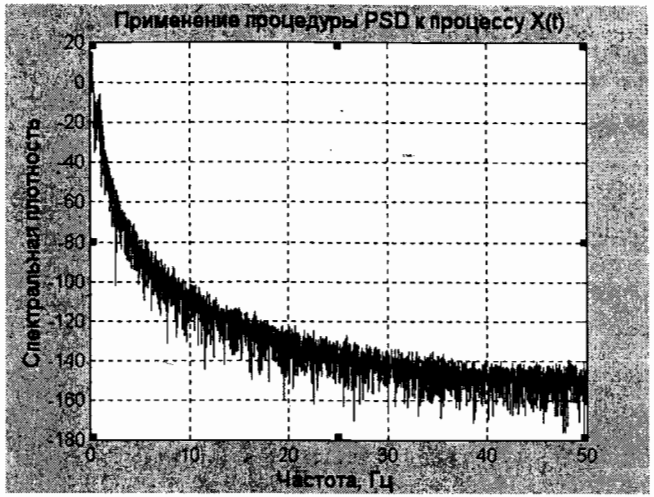

Рис. 5.63

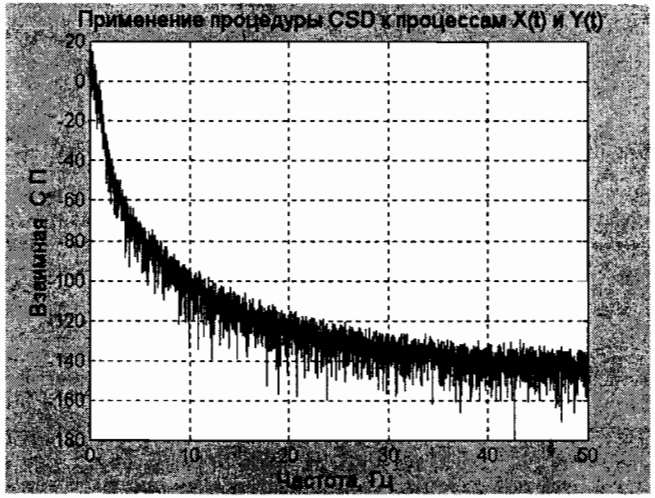

Рис. 5. 64


```
» csd(x,y,10000,100); title(' Применение процедуры CSD...
  к процессам X(t) и Y(t)');
» ylabel('Взаимная СП'); xlabel('Частота, Гц ')
```

Процедура **cohere** при обращении:

```
cohere(x,y,nfft,Fs)
```

вычисляет и выводит график от частоты квадрата модуля функции когерентности сигналов X(t) и Y(t), вычисленного по nfft точкам, заданным с частотой дискретизации Fs. Применяя эту процедуру к сформированным случайным процессам, получим картину, представленную на рис. 5.65:

```
» cohere(x,y,10000,100)
```

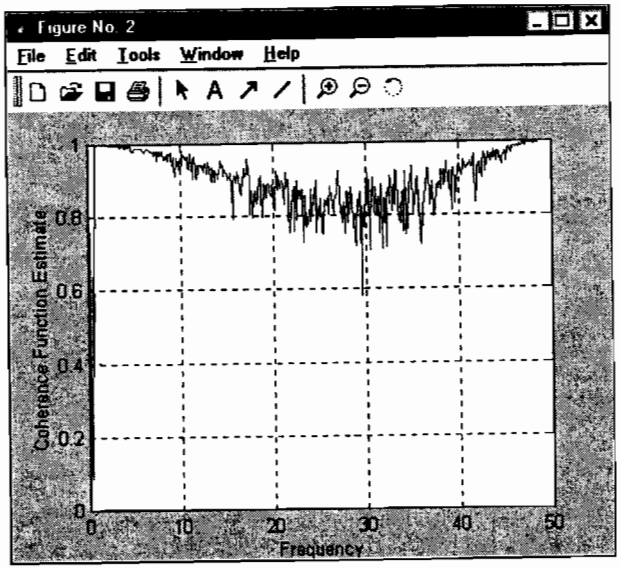

Рис. 5.65

Ознакомимся с процедурой **spectrum**, которая выполняет спектральный анализ двух процессов X(t) и Y(t). Обращение:

```
P = spectrum(x,y)
```

приводит к вычислению матрицы P, состоящей из восьми столбцов:

```
P = [Pxx,Pyy,Pxy,Txy,Cxy,Pxxc,Pyyc,Pxyc]
```

где Pxx — вектор-столбец, содержащий оценку СПМ процесса X; Pyy — вектор-столбец, содержащий оценку СПМ процесса Y; Pxy — вектор взаимной спектральной плотности процессов X и Y; Txy — комплексная передаточная функция: Txy = Pxy./Pxx; Cxy — функция когерентности, Cxy =((abs(Pxy)).^2)./(Pxx.*Pyy); Pxxc, Pyyc, Pxyc — векторы, содержащие доверительные интервалы для оценок Pxx, Pyy и Pxy.

Если эту функцию вызвать без выходных параметров:

```
spectrum(x,y)
```

то результатом ее работы будет поочередный вывод в одно графическое окно таких графиков:

1. Зависимости СПМ первого сигнала от нормализованной частоты (рис. 5.66); на графике появляются три кривые — кривая оценки усредненного значения СПМ на фиксированной частоте и две кривые с добавлением и вычитанием доверительного интервала на этой частоте.

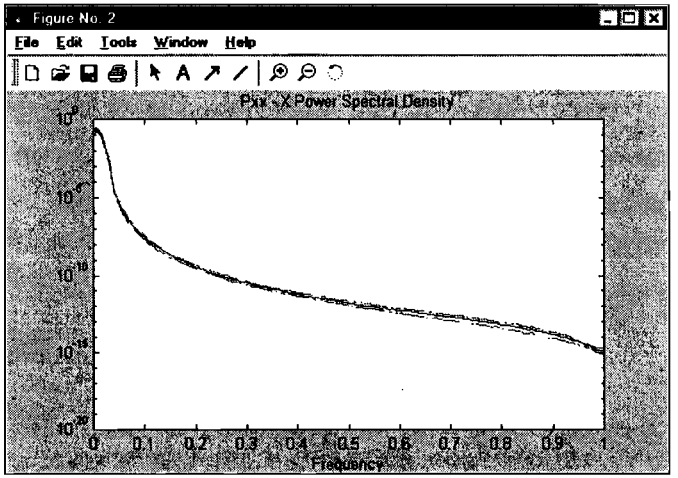

Рис. 5.66

2. После нажатия клавиши [Enter] прежние кривые исчезнут и на том же поле появятся три аналогичные кривые (рис. 5.67) для второго процесса Y(t).

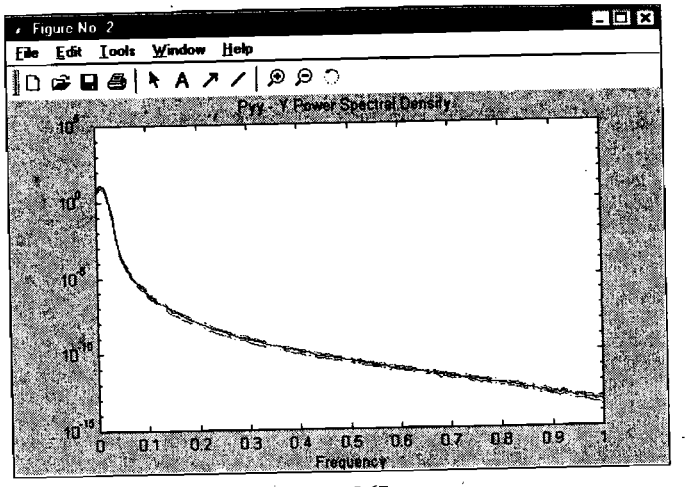

Рис. 5.67

3. Следующее нажатие [Enter] приведет (рис. 5.68) к появлению кривой зависимости модуля передаточной функции взаимной спектральной плотности указанных процессов от частоты.

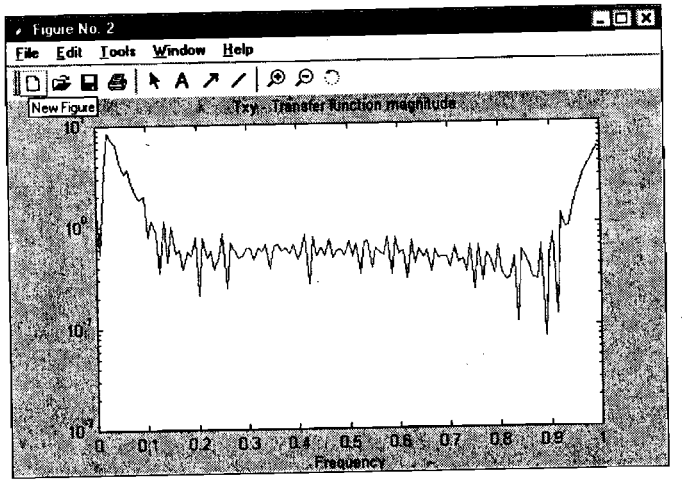

Рис. 5.68

4. Еще одно нажатие [Enter] приводит к появлению графика зависимости аргумента передаточной функции ВСП от частоты (рис. 5.69).

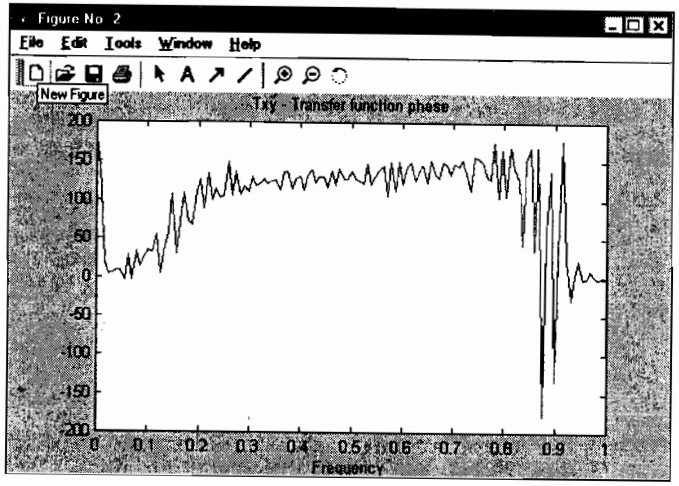

Рис. 5.69

5. Последнее нажатие [Enter] вызовет появление в поле графика графика функции когерентности (рис. 5.70).

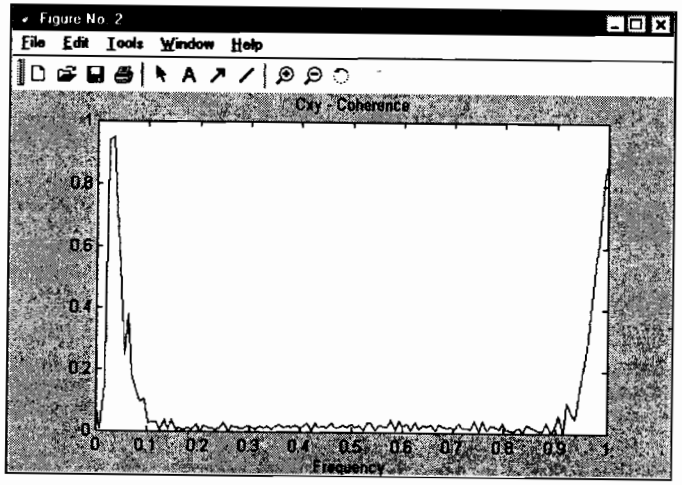

Рис. 5.70

Для построения спектрограммы процесса в MatLAB предусмотрена процедура **specgram**.

Примечание

Спектрограммой называется зависимость амплитуды вычисленного в окне ДПФ (дискретного преобразования Фурье) от момента времени, определяющего положение этого окна.

Для примера применим эту процедуру к сформированному выше процессу X(t):

```
» specgram(x,10000,100)
```

В результате получаем в графическом окне картину, изображенную на рис. 5.71.

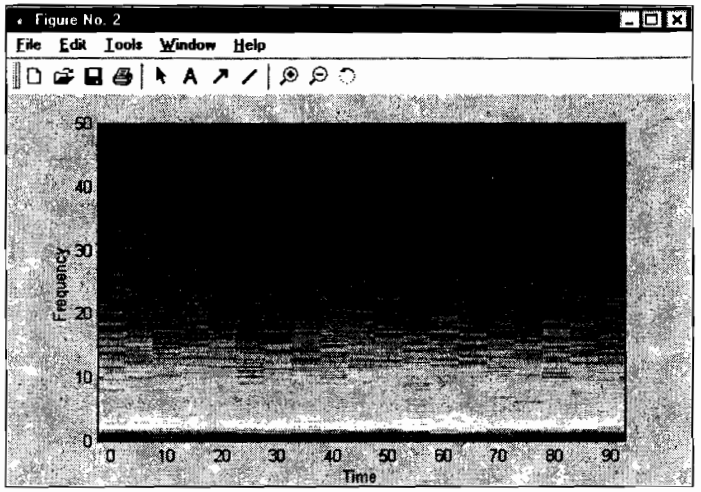

Рис. 5.71

Общий вид обращения к процедуре **specgram** напоминает обращение к процедуре **psd**:

```
specgram(x,nfft,Fs)
```

где x — вектор процесса, спектрограмма которого вычисляется, nfft — количество точек этого процесса, участвующих в вычислениях и Fs — частота дискретизации процесса.

Наконец, графическое представление имеет и процедура **tfe**, которая оценивает параметры и строит график АЧХ передаточной функции звена, на вход которого подан процесс, представленный первым вектором в обращении к процедуре, а на выходе получен

процесс, представленный вторым вектором. В целом, чтобы получить график АЧХ, нужно обратиться к процедуре таким образом:

```
tfe(x,y,nfft,Fs)
```

где x — вектор значений входного процесса, y — вектор выходного процесса, nfft — количество обрабатываемых точек (элементов указанных векторов), Fs — частота дискретизации.

Применяя процедуру к ранее сформированным процессам X(t) и Y(t):

```
» tfe(x,y,10000,100)
```

получим график, изображенный на рис. 5.72.

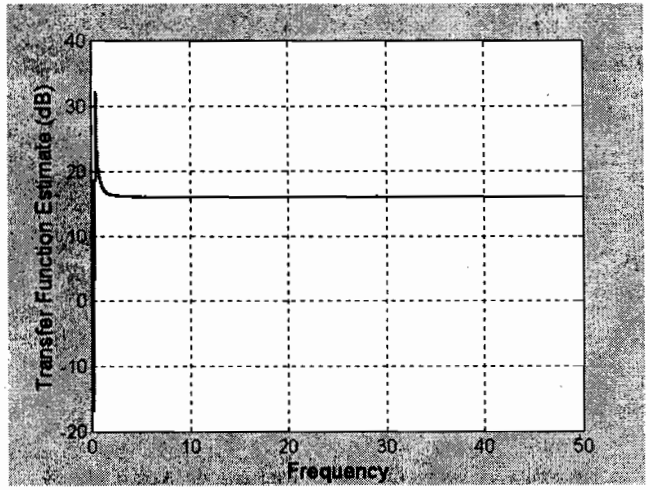

Рис. 5.72

5.5.2. Интерактивная оболочка SPTool

Процедура SPTool активизирует графическую интерактивную оболочку пакета Signal, включающую:

- средство поиска и просмотра сигналов — Signal Brouser;
- проектировщик фильтров — Filter Designer;
- средство просмотра характеристик фильтров — Filter Viewer;
- средство просмотра спектра — Spectrum Viewer.

Оболочка активизируется путем ввода в командном окне MatLAB команды **sptool**.

В результате на экране появляется окно, представленное на рис. 5.73.

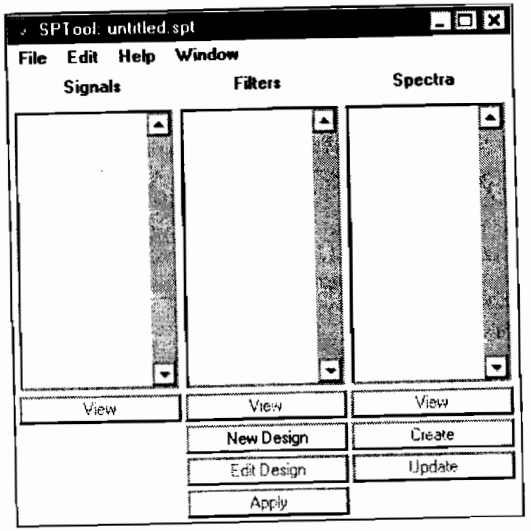

Рис. 5.73

Как видим, окно **SPTool** состоит из трех областей — **Signals** (Сигналы), **Filters** (Фильтры) и **Spectra** (Спектры), под каждой из которых имеются кнопки, указывающие на то, что можно сделать с объектами, расположенными в этих областях. Так, под областью **Signals** находится лишь кнопка **View**. Это означает, что объекты (сигналы), имена которых расположены в этой области, могут быть только просмотрены. Под областью **Filters** находятся четыре кнопки, которые указывают на то, что объекты (фильтры), имена которых размещаются внутри него, могут быть:

- созданы (кнопка **New Design**);
- отредактированы (кнопка **Edit Design**);
- просмотрены (кнопка **View**);
- применены к одному или нескольким объектам, выделенным в области **Signals** (**Apply**).

Аналогично, с объектами области **Spectra** (спектрами) можно производить такие действия:

- создавать (кнопка **Create**);
- просматривать (кнопка **View**);
- обновлять, т.е. создавать заново под тем же именем (кнопка **Update**).

Внутри областей обычно размещаются имена (идентификаторы) соответствующих переменных или процедур, входящих в открытый в SPTool файл с расширением .spt (имя этого файла указывается в заголовке окна **SPTool**).

При первом обращении в заголовке окна находится имя untitled.spt, все три области — пустые, а из кнопок, расположенных ниже, активной является только одна — **New Design**. Таким образом, после вхождения в оболочку SPTool доступной является только операция создания нового фильтра. Чтобы активизировать остальные кнопки, необходимо откуда-то импортировать данные о каком-то (или каких-то) сигнале (сигналах). Такие данные должны быть сформированы другими средствами, нежели сама оболочка SPTool, (например, они могут являться результатом выполнения некоторой программы MatLAB или результатом моделирования в среде SimuLink) и записаны как некоторые переменные либо в рабочем пространстве (Workspace), либо на диске в файле с расширением .mat.

Импорт сигналов

Чтобы обрабатывать какие-либо сигналы с помощью SPTool, прежде всего необходимо сформировать эти сигналы с помощью некоторой программы MatLAB, а затем импортировать полученные векторы значений этих сигналов в среду SPTool.

Допустим, что мы сгенерировали случайные процессы X(t), Y(t) и Y1(t) в соответствии с программой, приведенной в разд. 5.5.1. В результате в рабочем пространстве MatLAB появились векторы x, y и y1, каждый из которых содержит по 10000 элементов. Импортируем их в среду SPTool.

Войдя в среду SPTool (рис. 5.73), выберем из меню **File** (Файл) команду **Import** (Импорт). После этого откроется окно **Import to SPTool**, представленное на рис. 5.75.

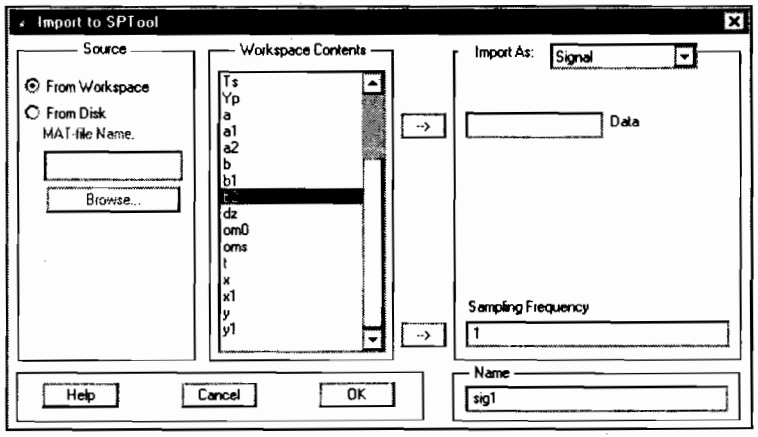

Рис. 5.75

В области **Source** (Источник) этого окна выбран переключатель **From Workspace** (Из рабочего пространства). Поэтому все имена переменных рабочего пространства представлены во второй области — **Workspace Contents** (Содержимое рабочего пространства). Выбрав при помощи мыши необходимую переменную, следует нажать кнопку со стрелкой, указывающей на поле ввода **Data**. После этого в поле ввода **Data** должно появиться имя выбранной переменной.

Затем в поле **Sampling Frequency** (Частота дискретизации) нужно ввести желаемое значение частоты дискретизации. Фактически этим параметром задается временной промежуток Ts между отдельными значениями выбранного вектора процесса.

В поле ввода **Name** (Имя) необходимо указать имя, под которым введенный вектор будет записан в среде SPTool.

После этого следует нажать кнопку **OK**, и импорт сигнала в среду SPTool будет произведен. Окно **Import to SPTool** исчезнет, а окно **SPTool** изменит свой вид (рис. 5.77): в области **Signals** появится запись имени вектора сигнала, и кнопка **View** под этой областью станет доступной. Кроме того, станет доступной кнопка **Create** под областью **Spectra**. Это означает, что можно находить спектральные характеристики импортированного сигнала.

Повторяя эти действия, можно перенести в SPTool и другие сигналы (X и Y).

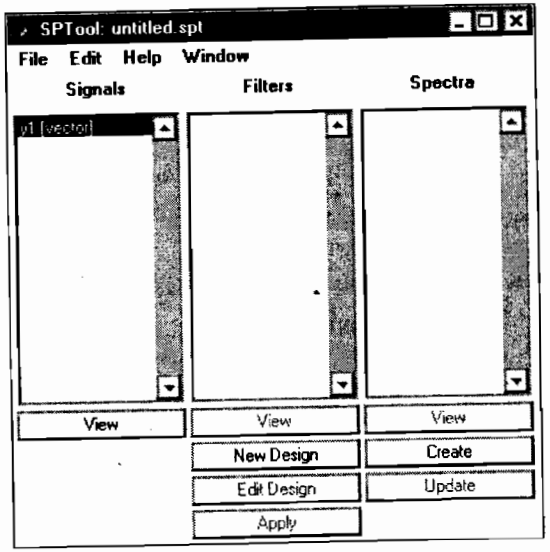

Рис. 5.77

Если векторы процессов записаны в MAT-файл, то для их импорта нужно после вызова окна **Import to SPTool** выбрать в нем переключатель **From disk**. В результате станут доступными поле ввода **MAT-file Name** и кнопка **Browse** (рис 5.75). Введя имя необходимого MAT-файла или отыскав MAT-файл при помощи кнопки **Browse**, вновь вызываем в окно **SPTool** его содержимое. Последующие действия аналогичны ранее рассмотренным.

Просмотр сигналов

После импорта вектора сигнала можно воспользоваться средствами его просмотра. Для этого достаточно выделить в области **Signals** нужные сигналы и нажать на кнопку **View** под областью. В результате должно появиться окно **Signal Browser**.

В нашем случае, выбрав сигнал y1, получим окно, изображенное на рис. 5.78.

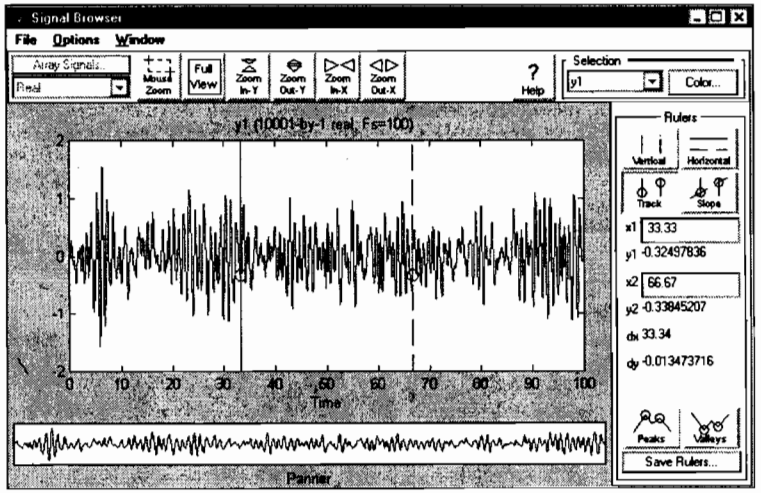

Рис. 5.78

Как видим, центральную часть окна занимает изображение кривых зависимости выделенных процессов от времени. В заголовке графика указаны имена сигналов, изображенных на графике, размерность соответствующих векторов и частота дискретизации.

В верхнем правом углу окна расположена область **Selection**, с помощью которой можно изменить цвета кривых, отображаемых в окне графиков.

Справа от графического поля окна (в области **Rulers**) размещаются инструменты, обеспечивающие точный отсчет показаний значений аргумента и процесса в двух точках графика.

Эти точки графика определяются пересечением графика процесса, имя которого стоит в списке **Selection**, с двумя вертикальными линиями розового цвета, расположенными в поле графиков. Изменение положения этих линий по шкале времени выполняется с помощью мыши. Подведите курсор к одной из вертикальных линий, и он примет вид руки. Нажмите левую кнопку мыши и, не отпуская ее, переместите курсор вправо или влево. При этом в области **Rulers** будут отображаться координаты X и Y точек пересечения обеих вертикальных прямых с графиком процесса, а также разности между ними. Координата X соответствует времени, а Y — значению процесса.

На рис. 5.79 отображены все три процесса. Если нажать кнопку **Slope**, в поле графика появится еще одна прямая, соединяющая указанные ранее точки пересечения выбранной кривой с вертикальными линиями, а в области **Rulers** отобразится новое число m, которое представляет собой тангенс угла наклона этой прямой к оси времени.

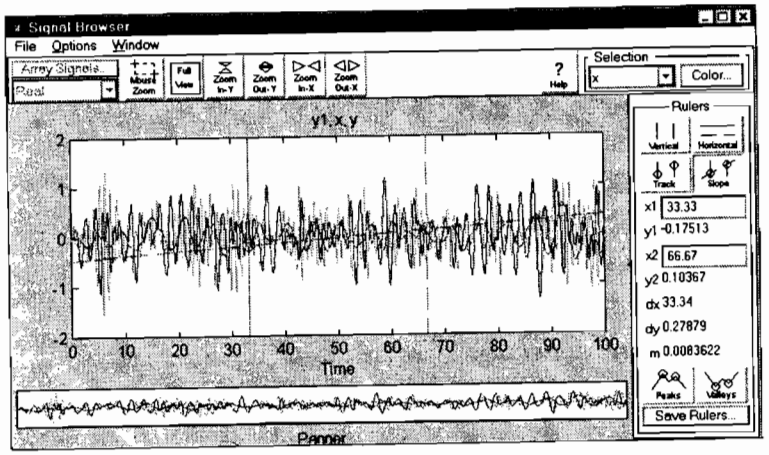

Рис. 5.79

В верхней части окна располагаются средства управления окном и масштабами внутри графического поля.

Создание спектров сигналов

После ввода сигналов в SPTool можно найти оценки спектральных свойств этих сигналов. Для этого достаточно в области сигналов окна **SPTool** отметить сигнал, оценку спектральной плотности которого вы хотите получить, и нажать на кнопку **Create** в нижней части окна. После этого на экране появится окно **Spectrum Viewer** (рис. 5.80).

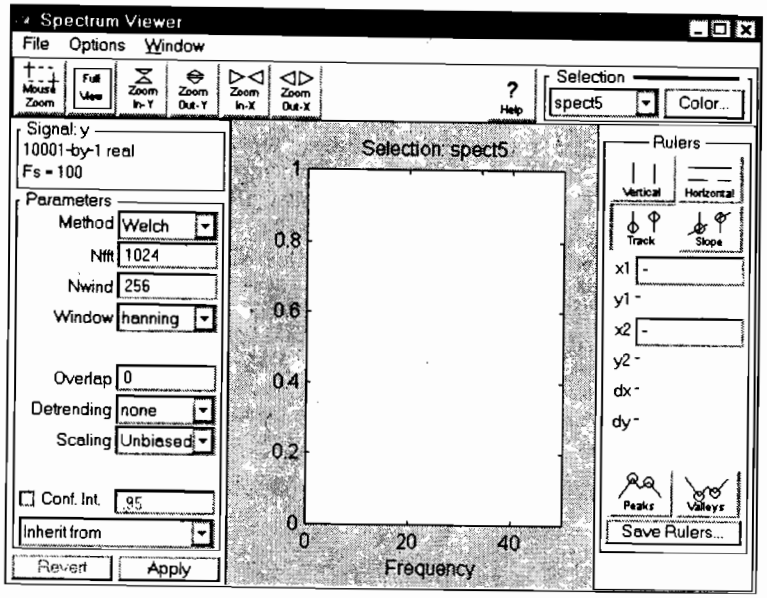

Рис. 5.80

Оно напоминает окно **Signal Browser**. Верхняя и правая части этих окон практически одинаковы. Однако графическая область окна **Spectrum Viewer** является пустой, а слева от нее располагается область, элементы которой позволяют:

- выбрать метод нахождения спектральной характеристики сигнала;
- установить количество обрабатываемых точек сигнала;
- установить количество точек сглаживающего окна;
- выбрать тип окна сглаживания;
- установить размер перекрытия окон;
- установить метод исключения тренда;
- установить метод масштабирования графика.

Метод вычисления спектра выбирается при помощи списка **Method**, который содержит следующие элементы: Burg, FFT, MEM, MTM, MUSIC, Welch, YuleAR. Каждому из них соответствует метод (процедура) вычисления спектра сигнала.

Для проведения вычислений после выбора метода следует нажать кнопку **Apply** ниже левого поля. Например, выделим для обработки процесс Y1, нажмем кнопку **Create** и выберем метод FFT. После нажатия кнопки **Apply** окно **Spectrum Viewer** примет вид, представленный на рис. 5.82.

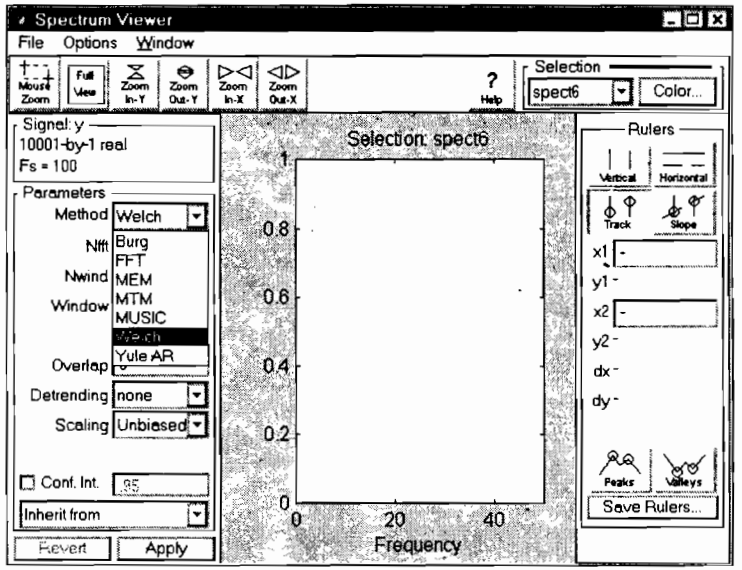

Рис. 5.81

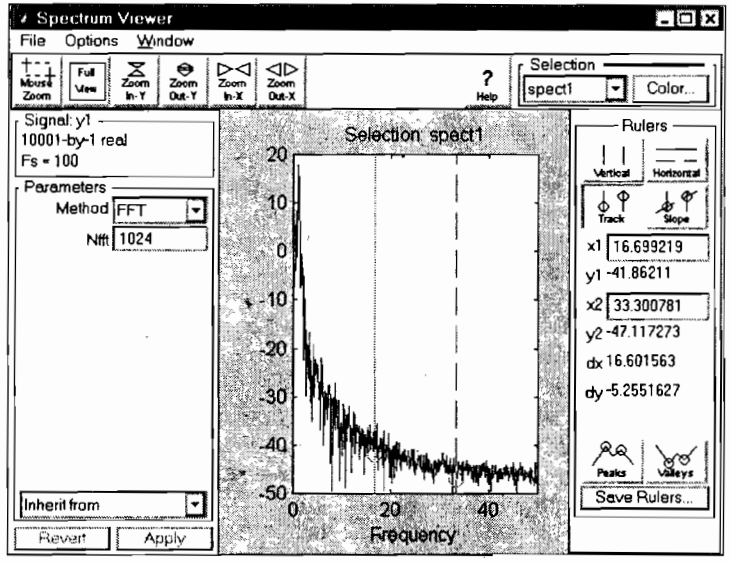

Рис. 5.82

Если в окне **SPTool** нажать кнопку **New Design**, то на экране появится окно **Filter Designer**, показанное на рис. 5.83.

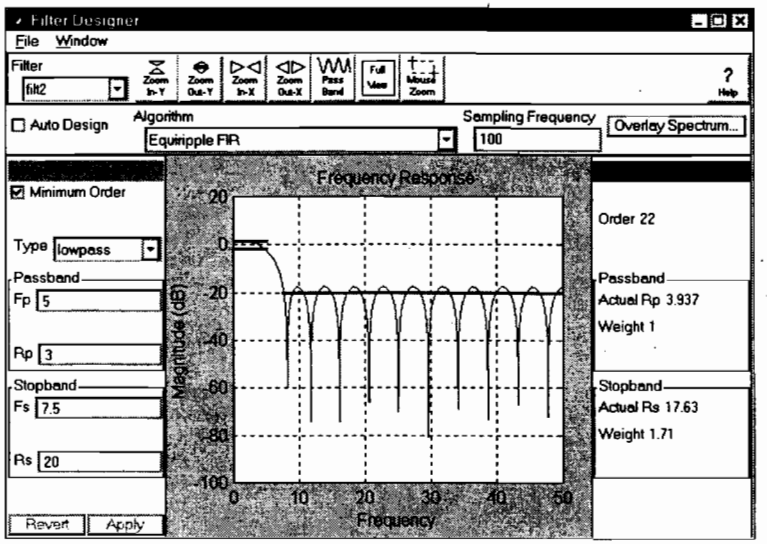

Рис. 5.83

Это окно позволяет произвести расчет коэффициентов нового фильтра и затем записать эти коэффициенты в объект-фильтр. При этом оно предоставляет возможность устанавливать и изменять следующие параметры будущего фильтра:

- прототип рассчитываемого фильтра (список **Algorithm**); при этом предоставляются такие варианты:
  - Equiripple FIR (КИХ-фильтр с равноотстоящими разрывами);
  - Least Square FIR (КИХ-фильтр по методу наименьших квадратов);
  - Kaizer Window FIR (КИХ-фильтр с окном Кайзера);
  - Butterwhorth IIR (БИХ-фильтр Баттерворта);
  - Chebyschev Type 1 IIR (БИХ-фильтр Чебышева 1-го типа);
  - Chebyschev Type 2 IIR (БИХ-фильтр Чебышева 2-го типа);
  - Elliptic IIR (Эллиптический БИХ-фильтр).
- тип фильтра (список **Type**); предоставляется возможность выбора следующих типов:
  - Lowpass — фильтр нижних частот;
  - Highpass — фильтр верхних частот;
  - Bandpass — полосовой фильтр;
  - Bandstop — режекторный фильтр.
- параметры полосы пропускания (область **Passband**); здесь можно установить, например, (для фильтра нижних частот) граничную частоту Fp полосы пропускания и максимально допустимое значение Rp подавления амплитуд внутри полосы пропускания (в децибелах);
- параметры полосы задерживания (область **Stopband**); здесь можно установить, например, (для фильтра нижних частот) граничную частоту Fs полосы задерживания и минимально допустимое значение Rs подавления амплитуд внутри полосы задерживания (в децибелах).

Количество устанавливаемых параметров и их смысл автоматически изменяются при переходе к другому типу фильтра.

Например, установив алгоритм фильтра Баттерворта нижних частот с граничными частотами полос пропускания в 0.5 Гц и задерживания в 0.7 Гц (рис. 5.84) и нажав кнопку **Apply**, получим параметры такого фильтра и запишем их в объект filt4.

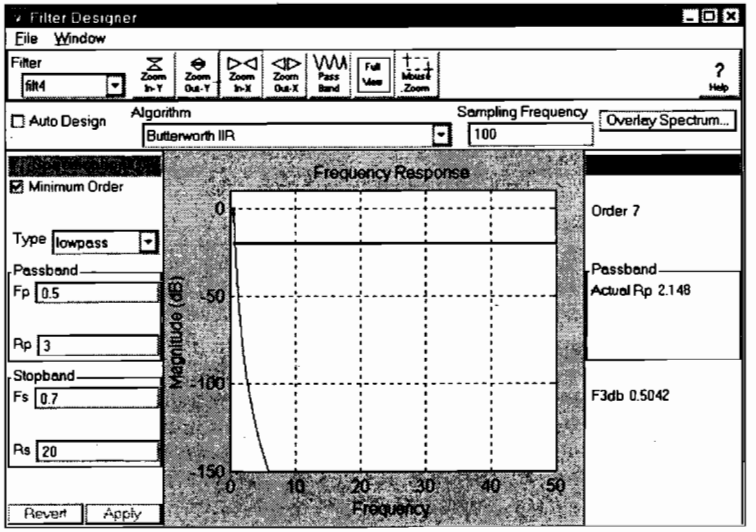

Рис. 5.84

Просмотр свойств фильтра

После создания фильтра можно просмотреть графики различных характеристик спроектированного и записанного фильтра. Для этого достаточно выделить имя фильтра, свойства которого нужно

посмотреть, в области Filters окна **SPTool**, а затем нажать кнопку **View** под этой областью.

Например, для только что созданного фильтра filt4 мы получим окно **Filter Viewer**, показанное на рис. 5.85.

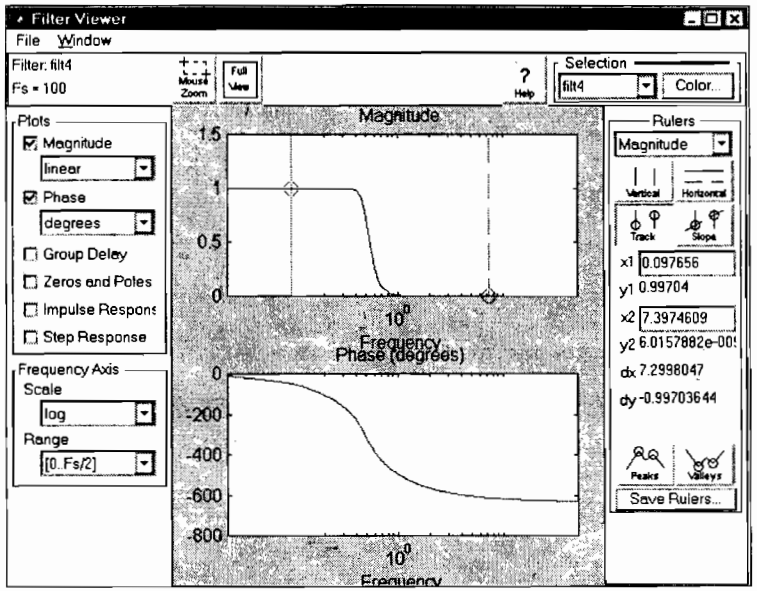

Рис. 5.85

Как видим, в окне выведены графики АЧХ и ФЧХ фильтра.

В число средств просмотра фильтров входят (см. левую область окна **Filter Viewer**):

- возможность вывода на экран одновременно любого сочетания таких графиков: АЧХ, ФЧХ, частотной зависимости группового времени замедления, графического представления расположения нулей и полюсов дискретной передаточной функции в Z-плоскости, графика временного отклика фильтра на импульсное единичное воздействие и графика отклика на ступенчатое единичное воздействие; для этого надо отметить галочкой с помощью мыши нужные виды графиков в области **Plots** (Графики) окна;
- возможность изменить вид шкалы как по оси частот, так и по оси амплитуд, установить диапазон представления графиков по частоте и изменить единицы представления фазового сдвига (области **Plots** и **Frequency Axis**).

Пример вывода всех доступных графиков одновременно показан на рис. 5.86.

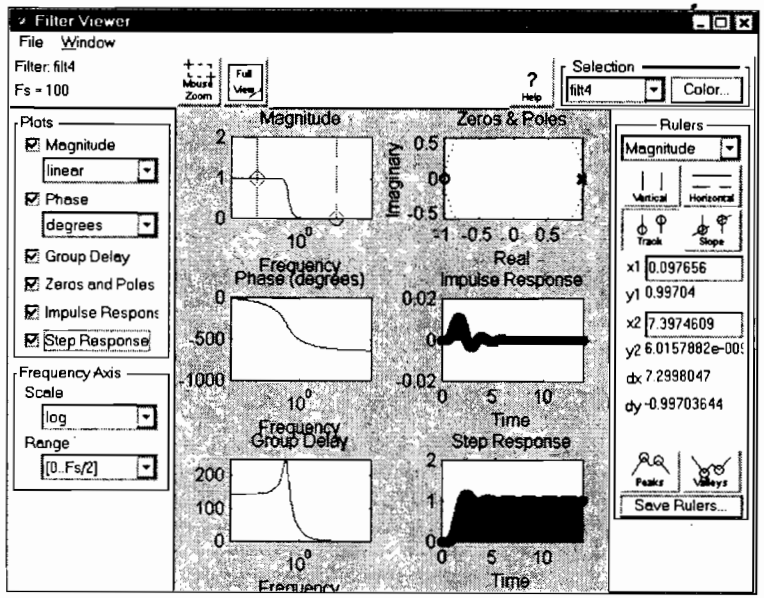

Рис. 5.86

Применение разработанного фильтра для фильтрации

Использовать в среде SPTool разработанный фильтр чрезвычайно просто. Для этого в области **Signals** окна **SPTool** нужно выделить имя сигнала, который нужно преобразовать с помощью фильтра, в области **Filters** — имя фильтра, с помощью которого надо преобразовать этот сигнал, и нажать кнопку **Apply**. В результате в первой области (**Signals**) появится имя нового сигнала, начинающееся с сочетания sig с последующим порядковым номером.

Полученный сигнал можно просмотреть, как это было описано ранее, используя команду **View**.

Например, применяя только что разработанный фильтр filt4 к процессу Y1(t), получим процесс, изображенный на рис. 5.87.

Спектральные характеристики полученного процесса можно изучить, применяя область **Spectra**, как это было описано.

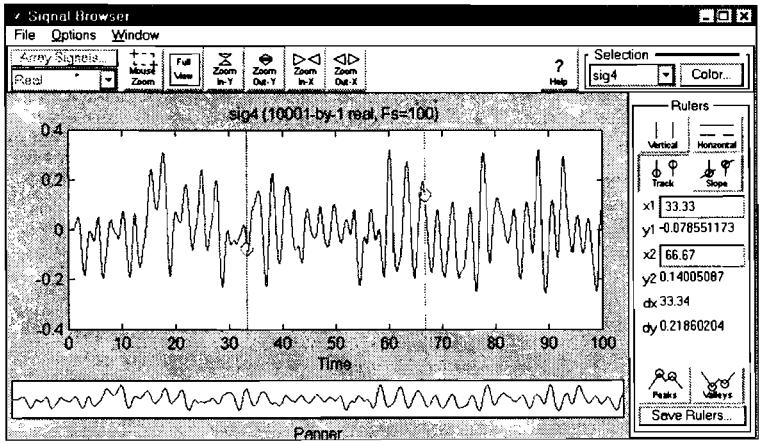

Рис. 5.87

Вторичное использование результатов SPTool

При завершении сеанса работы с SPTool система запрашивает, нужно ли записать полученные результаты на диск. В случае положительного ответа она сохраняет все данные в файле с расширением .spt. Кроме того, в меню **File** окна **SPTool** предусмотрены команды записи в файл (рис. 5.74) — **Save Session** и **Save Session As**.

При повторном запуске SPTool можно воспользоваться результатами такого сохранения, используя команду **Open Session** и выбирая один из записанных SPT-файлов.

Глава 6

Исследование линейных стационарных систем (пакет Control Toolbox)

Первое представление о пакете Control можно получить, изучая разд. 4.2.2 этой книги. Далее процедуры пакета рассмотрены подробнее.

6.1. Ввод и преобразования моделей

Ввод модели линейной стационарной системы (ЛСС) в среду пакета CONTROL возможен в трех формах — в форме матриц пространства состояния, в виде коэффициентов числителей и знаменателей передаточных функций и в форме задания нулей, полюсов и коэффициента передачи системы (разд. 4.2.2).

К процедурам создания LTI-моделей относятся:

ss Создает модель пространства состояния по заданным матрицам A, B, C, D уравнений состояния системы

dss Создает аналогичную модель по описанию пространства состояния более общего вида, когда уравнения переменных состояния не разрешены относительно производных

tf Создает модель по заданным передаточным функциям системы

zpk Создает модель по заданным нулям, полюсам и коэффициентам передачи системы

filt Создает модель по дискретным передаточным функциям, записанным в форме полиномов от z^{-1}

set Присваивает значения некоторым другим полям LTI-объекта (названиям входов и выходов, названиям системы и т.п.)

Указанные процедуры позволяют создавать как непрерывные модели (примеры приведены в разд. 4.2.2.), так и дискретные. В последнем случае к числу входных параметров процедуры следует добавить в конце значение параметра Ts — шага дискретизации, а вводимые значения коэффициентов уже должны задавать параметры дискретных передаточных функций (для функций **tf** и **zpk**) либо матрицы конечно-разностных уравнений пространства состояния — при использовании процедур **ss** и **dss**. При использовании процедуры **filt** должны задаваться векторы коэффициентов числителя и знаменателя дискретной передаточной функции, представленной в виде отношения полиномов от z^{-1}. Ниже приведено несколько примеров.

```
» kzv1 = tf([1 4], [1 2 100])
Transfer function:
     s + 4
---------------
s^2 + 2 s + 100
» kzv2 = tf([1 4], [1 2 100],0.01)
Transfer function:
     z + 4
---------------
z^2 + 2 z + 100
Sampling time: 0.01
» kzv3 = tf([1 4], [1 2 100],'Variable','z^-1')
Transfer function:
     1 + 4 z^-1
--------------------
1 + 2 z^-1 + 100 z^-2
Sampling time: unspecified
» kzv4 =filt([1 4], [1 2 100])
Transfer function:
     1 + 4 z^-1
--------------------
1 + 2 z^-1 + 100 z^-2
Sampling time: unspecified
```

Как следует из примеров, процедура **filt** полностью аналогична процедуре **tf** с добавлением в конец списка входных параметров записи 'Variable','z^-1'.

Процедуры **ss**, **dss**, **tf** и **zpk** применяются также для преобразования моделей из одной из указанных выше форм в другую. Первая и вторая применяются для преобразования модели в пространство состояния, третья — в передаточную функцию, а четвертая — в нули-полюсы-коэффициент передачи.

Модель, заданную как непрерывная, можно преобразовать в дискретную форму, воспользовавшись процедурой **c2d** в соответствии со схемой:

```
sysd = c2d(sys, Ts, method)
```

Здесь sys — исходная непрерывная заданная модель, sysd — получаемый в результате работы процедуры дискретный аналог исходной системы, Ts — задаваемое значение шага дискретизации, method — параметр, определяющий метод дискретизации. Последний параметр может принимать одно из значений:

'zoh'	Соответствует применению экстраполятора нулевого порядка: внутри интервала дискретизации сигналы аппроксимируются постоянной величиной, равной значению сигнала в начале интервала дискретизации
'foh'	Соответствует применению экстраполятора первого порядка: внутри интервала дискретизации сигналы аппроксимируются отрезками прямых, проходящих через концы кривой сигнала в интервале дискретизации
'tustin'	Билинейная аппроксимация Тастина внутри интервала дискретизации
'prevarp'	Та же аппроксимация Тастина с заданной частотой предыскривления
'matched'	Метод согласования нуля и полюса

Далее приведены примеры перевода введенного ранее непрерывного колебательного звена kzv1 в дискретные звенья разными методами:

```
» KZVd1= c2d(kzv1,0.01)
Transfer function:
 0.01008 z - 0.009687
----------------------
z^2 - 1.97 z + 0.9802
Sampling time: 0.01
» KZVd2= c2d(kzv1,0.01,'zoh')
 Transfer function:
 0.01008 z - 0.009687
----------------------
z^2 - 1.97 z + 0.9802
Sampling time: 0.01
» KZVd3= c2d(kzv1,0.01,'foh')
Transfer function:
0.005029 z^2 + 0.0002308 z - 0.004864
-------------------------------------
        z^2 - 1.97 z + 0.9802
Sampling time: 0.01
» KZVd4= c2d(kzv1,0.01,'tustin')
Transfer function:
0.005037 z^2 + 0.0001975 z - 0.00484
-----------------------------------
       z^2 - 1.97 z + 0.9802
Sampling time: 0.01
» KZVd5= c2d(kzv1,0.01,'prewarp',50)
Transfer function:
0.005145 z^2 + 0.000206 z - 0.004939
-----------------------------------
       z^2 - 1.97 z + 0.9798
Sampling time: 0.01
» KZVd6= c2d(kzv1,0.01,'matched')
Transfer function:
0.01009 z - 0.009696
--------------------
z^2 - 1.97 z + 0.9802
Sampling time: 0.01
```

Процедура **d2c** осуществляет обратную операцию — переводит дискретную систему в непрерывную:

```
» k1 = d2c(KZVd1)
Transfer function:
     s + 4
---------------
s^2 + 2 s + 100
» k2 = d2c(KZVd4,'tustin')
Transfer function:
     s + 4
---------------
s^2 + 2 s + 100
```

Как можно убедиться, указанные операции являются взаимно-обратными.

Процедура **d2d** позволяет *переопределить дискретную систему*, либо меняя шаг дискретизации:

```
sys1 = d2d(sys,Ts)
```

либо вводя групповые задержки Nd (целое, в количестве шагов дискретизации):

```
sys1 = d2d(sys,[],Nd)
```

Приведем примеры. Вначале изменим шаг дискретизации на Ts=0.1 для системы KZVd1:

```
» kd1=d2d(KZVd1,0.1)
Transfer function:
  0.09352 z - 0.06018
----------------------
z^2 - 0.9854 z + 0.8187
Sampling time: 0.1
```

Затем введем задержку по входу, равную 3Ts. Получим:

```
» kd2=d2d(k d1,[],3)
Transfer function:
     0.09352 z - 0.06018
-----------------------------
z^5 - 0.9854 z^4 + 0.8187 z^3
Sampling time: 0.1
```

Для создания модели нужно предварительно либо привести уравнения всей системы к форме уравнений пространства состояний, либо найти передаточные функции системы. В общем случае это довольно сложная и громоздкая задача. В то же время реальные системы автоматического управления (САУ) состоят из соединенных между собой отдельных блоков (динамических звеньев), уравнения поведения которых обычно достаточно просты. Поэтому

в практике проектирования САУ принято использовать структурные методы, когда САУ задается как определенная схема соединения отдельных элементарных динамических звеньев, и фактически проектируется одно или несколько из этих звеньев таким образом, чтобы обеспечить заданное качество всей системы. В соответствии с этим в MatLAB предусмотрена возможность "набирать" программно "схему" САУ путем предварительного ввода моделей звеньев, составляющих САУ, и последующего "соединения" этих звеньев в единую структуру. К процедурам, осуществляющим расчет характеристик соединений отдельных звеньев, относятся:

- **plus (minus)** — выполняет "параллельное соединение" указанных при обращении звеньев, т.е. определяет характеристики модели системы из параллельно соединенных звеньев; особенностью является то, что вызов этих процедур может быть осуществлен не только обычным способом: путем указания имени процедуры и перечисления (в скобках после имени) идентификаторов соединяемых звеньев, но и простым указанием идентификаторов звеньев, которые должны быть объединены, с простановкой между ними знаков "+" (при суммировании выходных сигналов звеньев) или "-" (при вычитании выходных сигналов);
- **parallel** — осуществляет ту же процедуру параллельного соединения звеньев; в отличие от предыдущей процедуры может использоваться для многомерных систем и осуществления параллельного соединения лишь по некоторым входам и выходам;
- **mtimes** (или знак "*" между именами звеньев) — осуществляет последовательное соединение звеньев, имена которых указаны; применяется для одномерных систем;
- **series** — последовательное частичное соединение многомерных систем;
- **feedback** — такое соединение двух звеньев, когда второе указанное звено составляет цепь отрицательной обратной связи для первого звена;
- **append** — формальное объединение не связанных между собой систем (добавление выходов и входов второй системы к выходам и входам первой);
- **connect** — установление соединений выходов и входов многомерной системы, созданной предварительно формальным объединением процедурой **append**; схема соединений задается матрицей Q соединений, указываемой как один из входных параметров процедуры;
- **inv** — рассчитывает САУ, обратную указанной, т.е. такую, у которой выходы и входы поменяли местами;
- **vertcat** — производит так называемую вертикальную конкатенацию (сцепление) систем (звеньев), т.е. такое их объединение,

когда входы этих систем становятся общими, а выходы остаются независимыми; для такого объединения необходимо, чтобы число входов объединяемых систем было одинаковым; тогда число входов в результирующей системе останется таким же, как и в каждой из объединяемых систем, а число выходов будет равно сумме выходов объединяемых систем;

- **horzcat** — осуществляет "горизонтальное сцепление" указанных систем, при котором выходы становятся общими, а входы добавляются.

Проиллюстрируем применение некоторых из этих процедур. Создадим модель углового движения торпеды вокруг вертикали в виде двух последовательно соединенных звеньев: апериодического звена, характеризующего влияние момента внешних сил относительно вертикали на угловую скорость торпеды:

```
» Torsk = tf(25,[100 50])
Transfer function:
    25
----------
100 s + 50
```

и интегрирующего звена, описывающего переход от угловой скорости к углу поворота торпеды вокруг вертикали:

```
» SkUg = tf(1,[1 0])
Transfer function:
1
-
s
```

Последовательное соединение этих звеньев можно осуществить двумя способами — применением процедуры **series**:

```
» Tor1= series(Torsk,SkUg)
Transfer function:
      25
--------------
100 s^2 + 50 s
```

либо просто операцией "перемножения" моделей:

```
» Tor = Torsk*SkUg
Transfer function:
      25
--------------
100 s^2 + 50 s
```

Теперь сформируем цепь управления, входом которой является угол рыскания торпеды, а выходом — момент, накладываемый на торпеду со стороны ее рулей направления. Ее будем предполагать состоящей из двух параллельно соединенных частей — части, управляемой гироскопом направления и представляющей собой обычное усилительное (статическое) звено:

```
» GN = tf(2,1)
Transfer function:
2
Static gain.
```

и части, управляемой гиротахометром, которую можно представить как дифференцирующе-колебательное звено:

```
» GT = tf([100 0],[1 10 100])
Transfer function:
     100 s
----------------
s^2 + 10 s + 100
```

Параллельное соединение этих двух контуров управления можно осуществить тоже двумя способами: либо используя процедуру **parallel**:

```
» Izm1=parallel(GN,GT)
 Transfer function:
2 s^2 + 120 s + 200
-------------------
 s^2 + 10 s + 100
```

либо применяя операцию "сложения" моделей:

```
» Izm = GN+GT
Transfer function:
2 s^2 + 120 s + 200
-------------------
 s^2 + 10 s + 100
```

Теперь найдем модель всей системы автоматического управления угловым движением торпеды, рассматривая цепь управления как цепь отрицательной обратной связи для торпеды и пользуясь для объединения прямой и обратной цепи процедурой **feedback**:

```
» sys=feedback(Tor,Izm)
Transfer function:
            25 s^2 + 250 s + 2500
-----------------------------------------------
100 s^4 + 1050 s^3 + 10550 s^2 + 8000 s + 5000
```

Конечно, более простым и удобным средством "создания" (точнее, "набора") сложных систем из отдельных блоков является рассмотренная в главе 7 интерактивная система SIMULINK.

После того как система сформирована, можно ввести при помощи процедуры **set** некоторые ее символьные описания. В частности, присвоить названия входам и выходам системы, а также дать краткий комментарий к самой системе.

```
» set(sys,'Input Name',' Момент сил','OutputName',
        'Угол рыскания')
» set(sys,'Notes','Угловое движение торпеды')
» get(sys)
num = {[0 0 25 250 2.5e+003]}
den = {[100 1.05e+003 1.06e+004 8e+003 5e+003]}
Variable = 's'
Ts = 0
InputName = {'Момент сил'}
OutputName = {'Угол рыскания'}
Notes = {'Угловое движение торпеды'}
UserData = []
```

В заключение приведем примеры использования процедур конкатенации:

```
» sysvsp1=horzcat(Torsk,SkUg)
Transfer function from input 1 to output:
    25
----------
100 s + 50
Transfer function from input 2 to output:
1
-
s
» sysvsp2=vertcat(Torsk,SkUg)
Transfer function from input to output...
          25
 #1:  ----------
      100 s + 50
      1
 #2:  -
      s
```

6.2. Получение информации о модели

Чтобы получить отдельные характеристики (матрицы и векторы, описывающие пространство состояния, коэффициенты числителя и знаменателя передаточной функции и т.п.) полученной модели, можно использовать одну из следующих процедур: **tfdata** (для получения векторов числителя и знаменателя передаточной функции системы), **ssdata** (для получения значений матриц уравнений пространства состояния) и **zpkdata** (для получения векторов значений полюсов и нулей системы).


```
» [nom,den]=tfdata(sys,'v')
nom =
    0          0          25         250        2500
den =
   100        1050       10550       8000       5000
» sssys=ss(sys)
» [A,B,C,D] = ssdata(sssys)
A =
  -10.5000   -6.5938   -1.2500   -0.7813
   16.0000         0         0         0
         0    4.0000         0         0
         0         0    1.0000         0
B =
    0.5000
         0
         0
         0
C =
         0    0.0313    0.0781    0.7813
D =      0
» [z,p,k] = zpkdata(sys,'v')
z =
  -5.0000 + 8.6603i
  -5.0000 - 8.6603i
p =
  -4.8653 + 8.5924i
  -4.8653 - 8.5924i
  -0.3847 + 0.6040i
  -0.3847 - 0.6040i
k =    0.2500
```

Процедура **get** дает возможность получить полную характеристику модели, включая имена входов и выходов, примечания, значения шага дискретизации и т.п. Например:

```
» get(sys)
num = {[0 0 25 250 2.5e+003]}
den = {[100 1.05e+003 1.06e+004 8e+003 5e+003]}
Variable = 's'
Ts = 0
Td = 0
InputName = {'Момент сил'}
OutputName = {'Угол рыскания'}
Notes = {'Угловое движение торпеды'}
UserData = []
» get(ssys)
Continuous-time system.
        a = [4x4 double]
        b = [4x1 double]
        c = [0 0.0313 0.0781 0.781]
        d = 0
        e = []
        StateName = {4x1 cell}
```

```
        Ts = 0
        Td = 0
        InputName = {'Момент сил'}
        OutputName = {'Угол рыскания'}
        Notes = {'Угловое движение торпеды'}
        UserData = []
```

О числе входов и выходов системы можно узнать, обратившись к процедуре **size**:

```
» size(sys)
Transfer function with 1 input(s) and 1 output(s).
» size(ssys)
State-space model with 1 input(s), 1 output(s),
and 4 state(s).
```

6.3. Анализ системы

Пакет CONTROL предоставляет широкий набор процедур, осуществляющих анализ САУ с самых различных точек зрения и, прежде всего, определение откликов системы на внешние воздействия как во временной, так и в частотной областях.

Для нахождения временных откликов системы на внешние воздействия некоторых видов предусмотрены функции:

impulse	Нахождение отклика системы на единичное импульсное входное воздействие
step	Нахождение реакции системы на единичный скачок входного воздействия
initial	Определение собственного движения системы при произвольных начальных условиях
lsim	Определение реакции системы на входное воздействие произвольной формы, задаваемое в виде вектора его значений во времени

Рассмотрим применение этих процедур на примере движения торпеды, параметры которой как САУ приведены ранее.

Применяя процедуру **step** к созданной модели:

```
» step(sys)
```

можно получить график, представленный на рис. 6.1.

Аналогично, использование процедуры

```
» impulse(sys)
```

приведет к появлению в графическом окне графика, приведенного на рис. 6.2.

Чтобы применить процедуру **initial**, необходимо в число входных параметров включить, во-первых, полный вектор всех начальных условий по переменным состояния, а во-вторых, момент времени окончания процесса интегрирования.


```
» initial(ssys,[0 0 0 1],20)
```

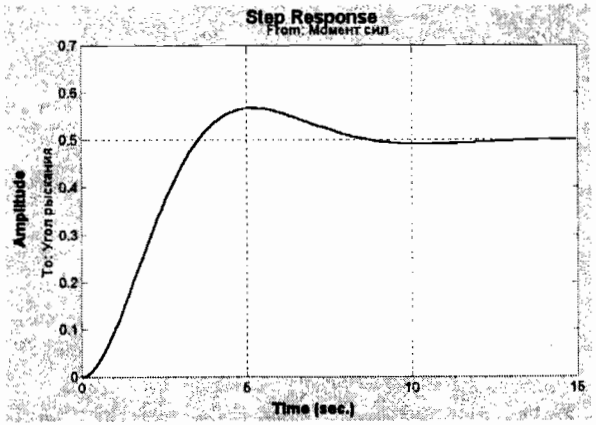

Рис. 6.1

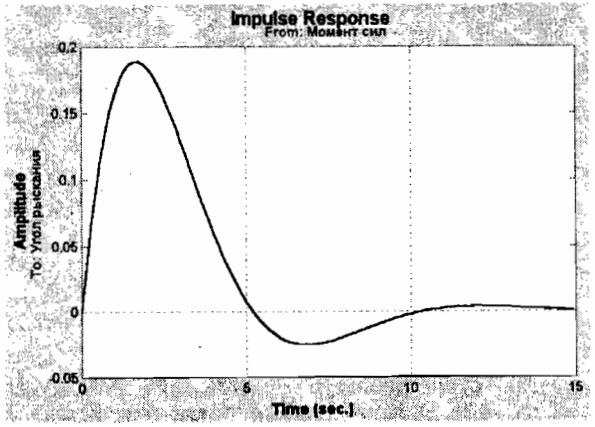

Рис. 6.2

Получим в графическом окне картину, показанную на рис. 6.3.

Для применения процедуры **lsim** необходимо предварительно задать вектор **t** значений времени, в которых будут заданы значения входного воздействия, а затем задать соответствующий вектор *u* значений входной величины в указанные моменты времени:

```
» t = 0:0.01:40; u = sin(t); lsim(ssys,u,t); grid
```

Результат изображен на рис. 6.4.

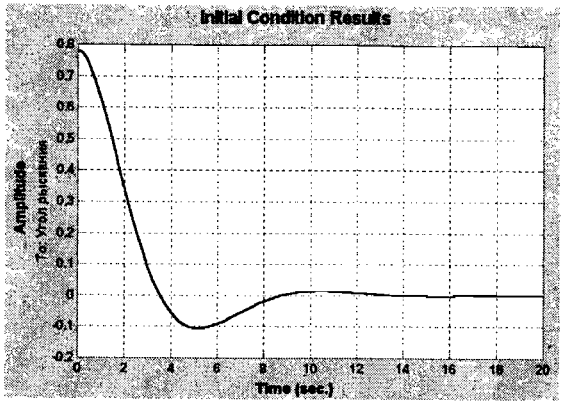

Рис. 6.3

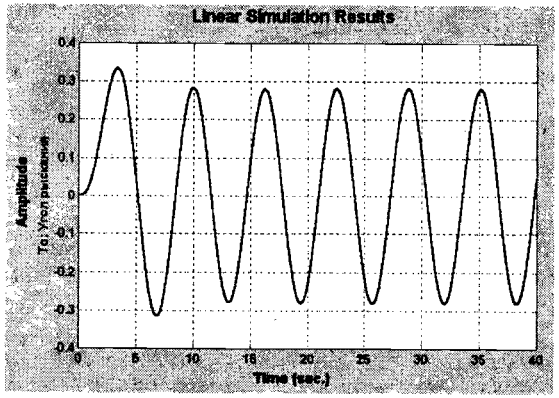

Рис. 6.4

Следующая группа процедур представляет в частотной области реакцию системы на внешние гармонические воздействия. К таким процедурам относятся:

bode	Строит графики АЧХ и ФЧХ (диаграмму Боде) указанной системы
nyquist	Строит в комплексной плоскости график амплитудно-фазовой характеристики (АФХ) системы в полярных координатах
nichols	Строит карту Николса системы, т.е. график АФХ разомкнутой системы в декартовых координатах
sigma	Строит графики зависимости от частоты сингулярных значений системы; обычно совпадает с АЧХ системы

margin	Строит диаграмму Боде с указанием запасов по амплитуде и по фазе

Приведем примеры: **bode(sys)** — рис. 6.5, **nyquist(sys); grid** — рис. 6.6, **nichols(sys); grid** — рис. 6.7, **sigma(sys)** — рис. 6.8, **margin(ssys); grid** — рис. 6.9.

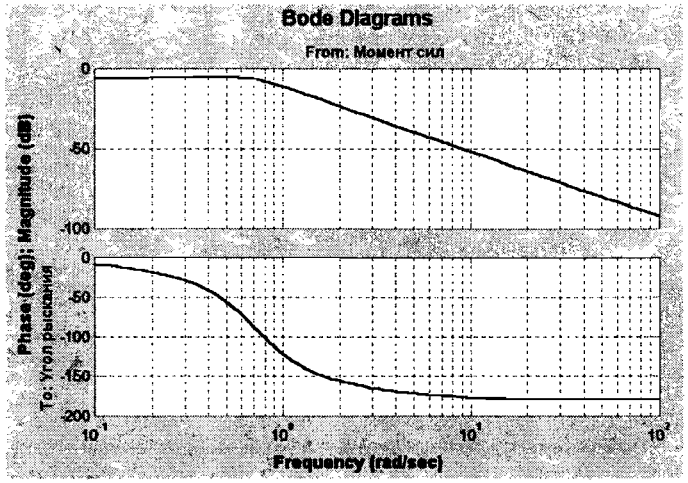

Рис. 6.5

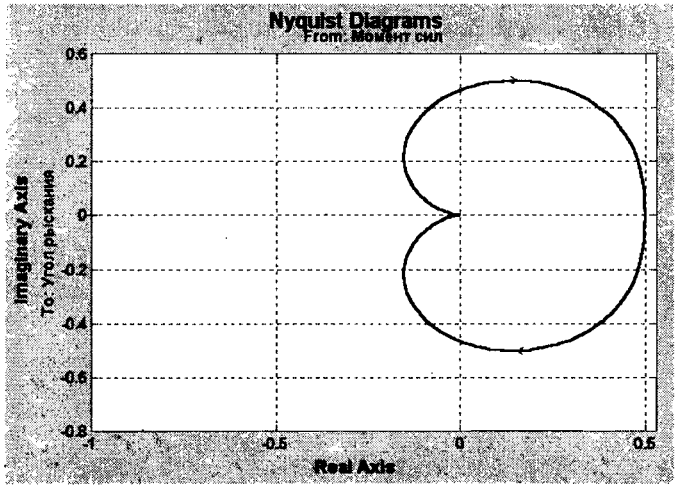

Рис. 6.6

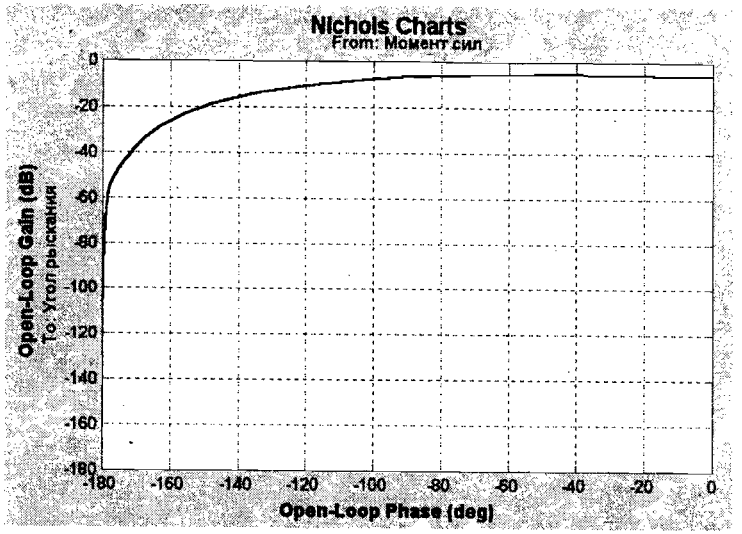

Рис. 6.7

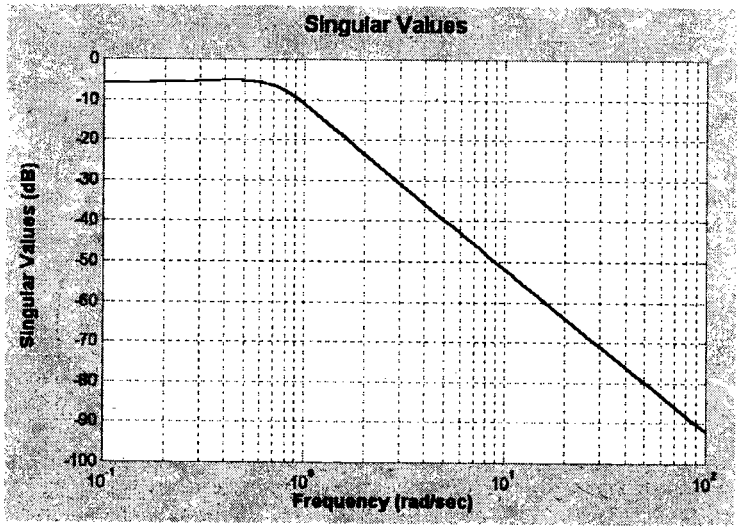

Рис. 6.8

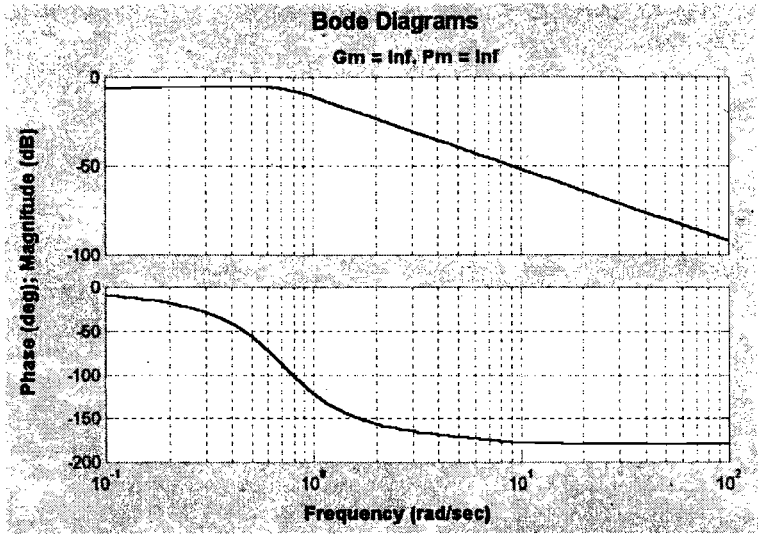

Рис. 6.9

Теперь рассмотрим процедуры, вычисляющие отдельные характеристики и графически показывающие расположение полюсов и нулей системы. К ним можно отнести следующие:

pole Расчет полюсов системы

zpkdata Расчет полюсов, нулей и коэффициента передачи системы

gram Вычисление Грамианов системы — матрицы управляемости (при указании в качестве последнего входного параметра процедуры флага 'c') и матрицы наблюдаемости системы (при указании флага 'o')

damp Вычисление собственных значений матрицы состояния системы и на этой основе — значений собственных частот (Frequency) незатухающих колебаний системы и относительных коэффициентов демпфирования (Damping)

pzmap Построение на комплексной плоскости карты расположения нулей и полюсов системы

rlocus Расчет и вывод в виде графиков в графическое окно траектории движения на комплексной плоскости корней полинома:

$$H(s) = D(s) + k * N(s) = 0$$

где $D(s)$ — знаменатель передаточной функции, $N(s)$ — ее числитель, при изменении положительного вещественного числа k от 0 до бесконечности

Далее приводятся примеры применения этих функций и результаты.

```
» pole(sys)
ans =
  -4.8653 + 8.5924i
  -4.8653 - 8.5924i
  -0.3847 + 0.6040i
  -0.3847 - 0.6040i
» sysz=zpk(sys)
Zero/pole/gain from input "Момент сил" to
output "Угол рыскания":
            0.25 (s^2 + 10s + 100)
-----------------------------------------------
(s^2 + 0.7693s + 0.5128) (s^2 + 9.731s + 97.5)
» [z,p,k]=zpkdata(sysz,'v')
z =
  -5.0000 + 8.6603i
  -5.0000 - 8.6603i
p =
  -4.8653 + 8.5924i
  -4.8653 - 8.5924i
  -0.3847 + 0.6040i
  -0.3847 - 0.6040i
k =    0.2500
» Wc= gram(sssys,'c')
Wc =
    0.0129    0.0000   -0.0083   -0.0000
    0.0000    0.0334   -0.0000   -0.0175
   -0.0083   -0.0000    0.0701   -0.0000
   -0.0000   -0.0175   -0.0000    0.1372
» Wo=gram(sssys,'o')
Wo =
    0.3334    0.2188    0.5469    0.3906
    0.2188    0.1442    0.3605    0.2726
    0.5469    0.3605    0.9011    0.6805
    0.3906    0.2726    0.6805    0.8545
```

pzmap(sys) — результат приведен на рис. 6.10.

```
» damp(sys)
    Eigenvalue                 Damping      Freq. (rad/s)
-3.85e-001 + 6.04e-001i     5.37e-001       7.16e-001
-3.85e-001 - 6.04e-001i     5.37e-001       7.16e-001
-4.87e+000 + 8.59e+000i     4.93e-001       9.87e+000
-4.87e+000 - 8.59e+000i     4.93e-001       9.87e+000
```

rlocus(sys) — результат приведен на рис. 6.11.

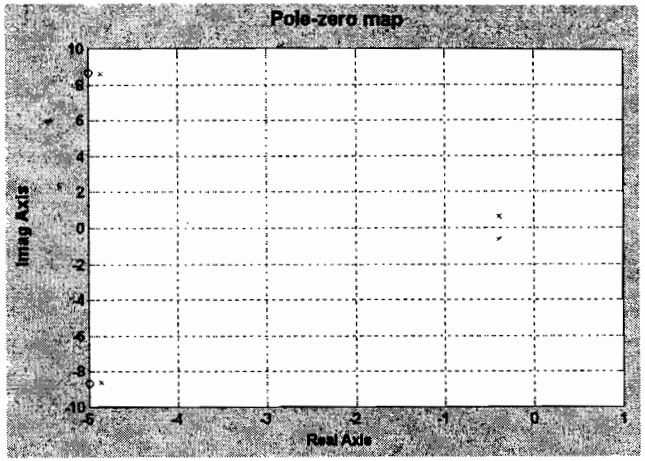

Рис. 6.10

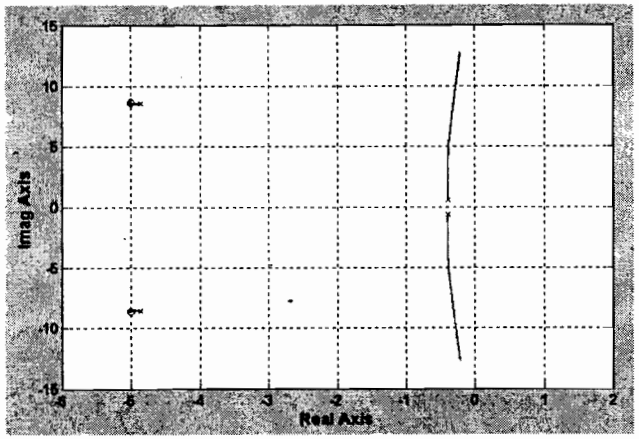

Рис. 6.11

6.4. Интерактивный обозреватель ltiview

Если в командном окне MatLAB ввести команду **ltiview**, то можно вызывать окно так называемого обозревателя LTI-объектов и в интерактивном режиме строить в этом окне практически все вышеуказанные графики, причем для нескольких систем синхронно.

После выполнения команды на экране появляется окно **LTI Viewer** (рис. 6.12).

Это окно состоит из нескольких частей. Главное место в нем занимает графическое поле, в котором строятся графики. При первом обращении к обозревателю оно пусто. В нижней левой части расположены два списка — **Workspace** и **Selected**. В первом из них отображены LTI-объекты, которые находятся в это время в рабочем пространстве системы MatLAB (на рис. 6.12 — это система sys бокового движения торпеды, созданная ранее). Во втором отображаются LTI-объекты, которые загружены в среду обозревателя. В начале работы с обозревателем этот список также пуст.

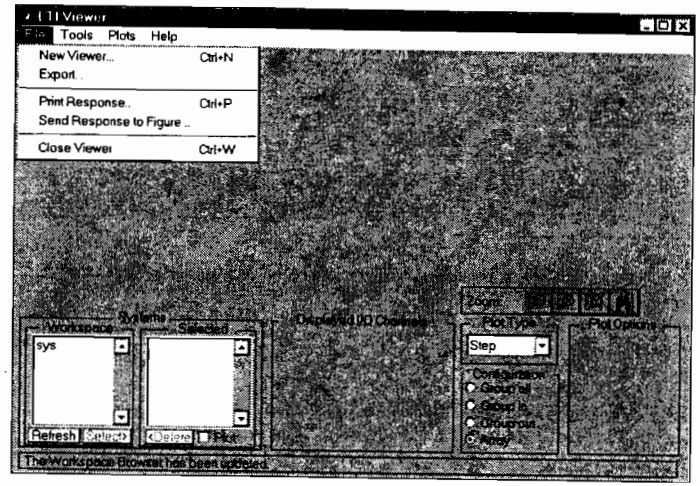

Рис. 6.12

По правой стороне внизу расположен открывающийся список **Plot Type**, содержащий перечень графических процедур, которые можно выполнить. Еще правее расположена группа опций **Plot Options**, позволяющих определить некоторые числовые характеристики соответствующего графика.

Работу с обозревателем необходимо начинать с загрузки в его среду LTI-объектов, которые нужно проанализировать. Это делается таким образом.

С помощью мыши следует выделить в списке **Workspace** имена исследуемых LTI-объектов и нажать кнопку **Select**. Соответствующее имя должно появиться в списке **Selected** рядом (рис. 6.13). Причем перед именем будет указан знак "+", а после него в скобках — цвет линии графика, отвечающего LTI-объекту с этим именем. Синхронно в графическом поле появляется график реакции избранной системы на входное действие в виде единичной ступеньки (рис. 6.13), так как по умолчанию в списке **Plot Type** выбран элемент **Step**.

Список **Plot Type** включает следующие элементы: **Step**, **Impulse**, **Bode**, **Nyquist**, **Nichols**, **Sigma**, **PoleZero**, **Lsim**, **Initial**. Как видим, они полностью совпадают с ранее рассмотренными одноименными процедурами, и потому следует ожидать, что их выбор приведет к построению тех же графиков. Например, при выборе элемента **Impulse** получим картину, изображенную на рис. 6.14.

В области **Plot Option** расположены опции, которые позволяют указать, какие параметры процесса можно будет отсчитать по графику. Так, на рис. 6.13 и 6.14 отмечены все возможные параметры, и поэтому на графиках появились точки, которые отвечают искомым параметрам.

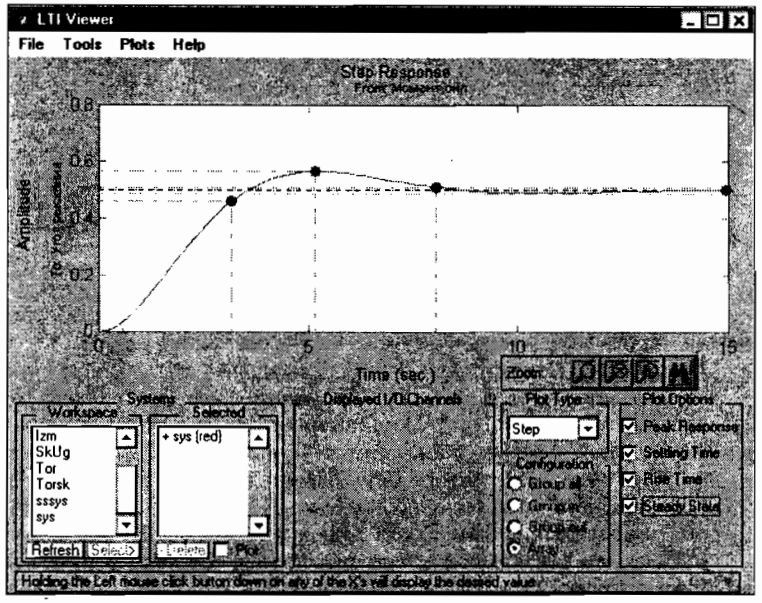

Рис. 6.13

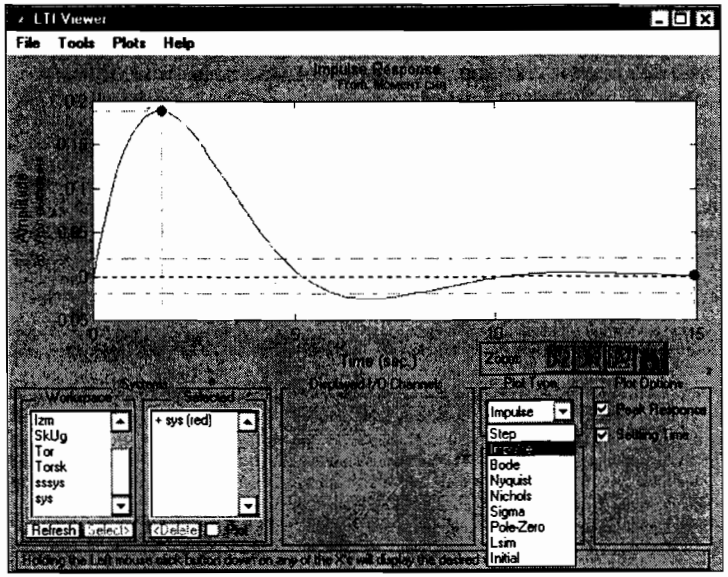

Рис. 6.14

Вид графиков можно изменять, используя меню окна обозревателя. К примеру, меню **File** (рис. 6.12) содержит команды:

- **New Viewer** (Открыть новый обозреватель) — при выборе этой команды появляется еще одно, дополнительное, окно такого же обозревателя, в котором можно построить иные нужные графики;
- **Export** (Экспортировать LTI-объект) — выбор команды приводит к появлению окна **Export LTI Models/Compensators** (рис. 6.15), позволяющего записать используемые LTI-объекты (имена которых находятся в списке **Export List**) в MAT-файл (кнопка **Export to Disk**) или в рабочее пространство (кнопка **Export to Workspace**);
- **Print Response** (Распечатать графики);
- **Send Response to Figure** (Расположить график в отдельном графическом окне (фигуре)) — после вызова этой команды появляется окно **LTI Viewer Responses**, в котором располагается тот же график, что и в графическом поле окна **LTI Viewer**. Однако в данном случае график занимает всю область окна. Это удобно по двум причинам: во-первых, в этом окне можно увидеть, как будет выглядеть график при печати на принтере, а во-вторых, что более важно, новое окно позволяет создать копию графика. Для этого в окне **LTI Viewer Responses** необходимо вызвать из меню **Edit** команду **Copy Figure**. В результате содержимое окна

будет помещено в буфер. После этого можно перейти в окно какого-нибудь редактора (графического или текстового) и при помощи команды вставки скопировать график в текстовый или графический документ. Таким образом получен рис. 6.16, содержащий годограф Найквиста для исследуемой системы (рис. 6.17);

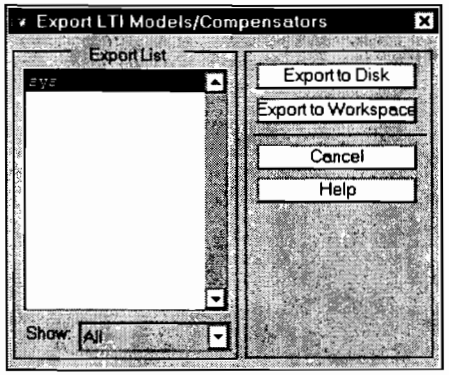

Рис. 6.15

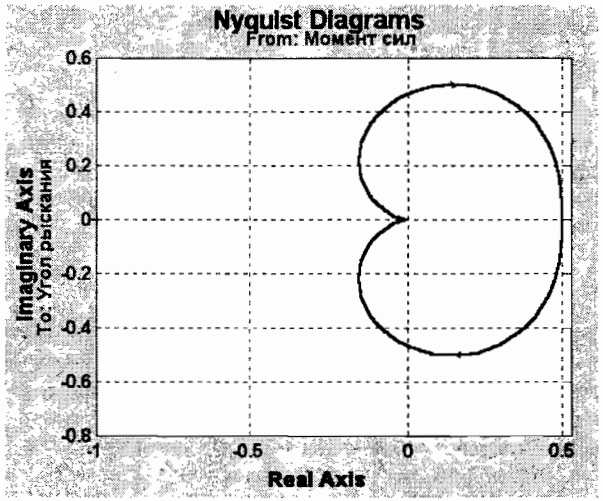

Рис. 6.16

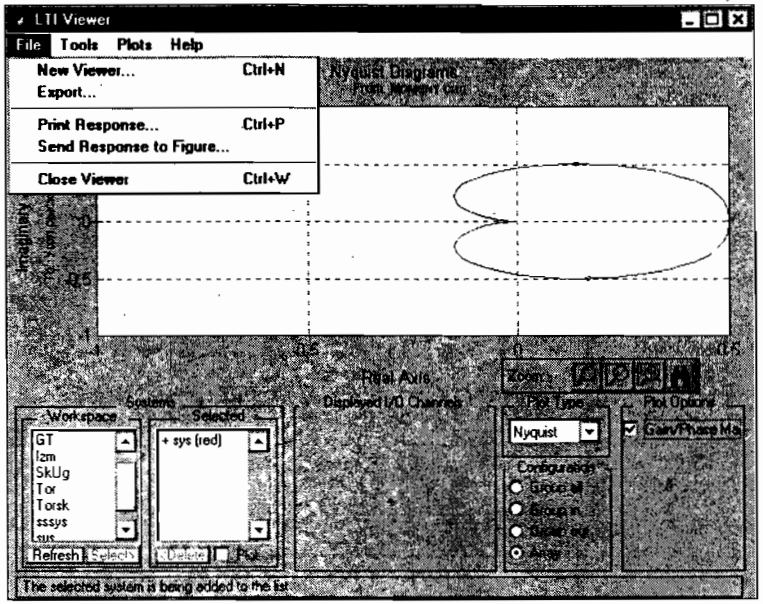

Рис. 6.17

- **Close Viewer** (Закрыть окно обозревателя).

Перейдем к следующему меню окна **LTI Viewer** под названием **Tools**. Данное меню (рис. 6.18) включает три элемента:

- **Viewer Controls** (Управление обозревателем) — эта опция позволяет управлять отображением управляющих элементов, расположенных в нижней части окна **LTI Viewer**; по умолчанию опция активна (отмечена "галочкой"). Если опция не установлена, окно **LTI Viewer** выглядит так, как на рис. 6.19;
- **Response Preferences** (Свойства графиков) — выбор этой команды приводит к появлению одноименного окна (рис. 6.20). В нем можно установить диапазоны изменений координат, изменить единицы измерения амплитуды и частоты, установить собственные определения времени установления и времени возрастания переходного процесса и т.п.;
- **Linestyle Preferences** (Свойства и стили линий) — после вызова этой команды появляется диалоговое окно (рис. 6.21), которое позволяет установить цвет, толщину и стили линий для графического изображения. Для этого достаточно выбрать нужный объект и установить требуемый параметр.

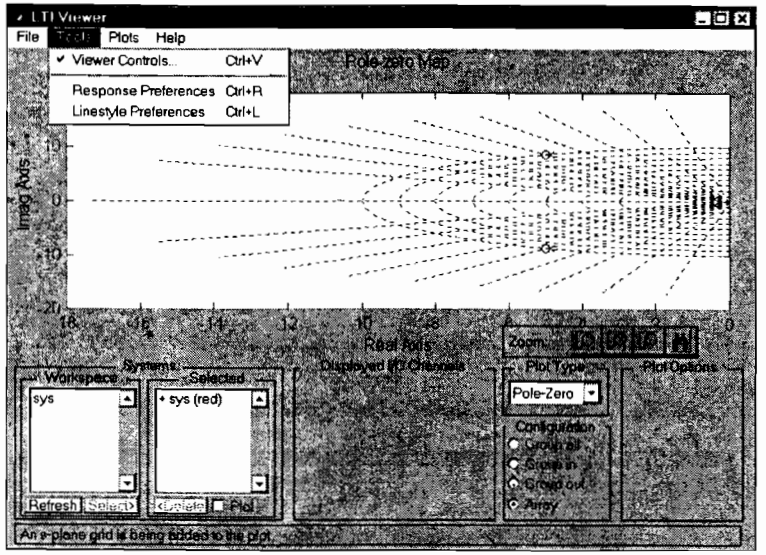

Рис. 6.18

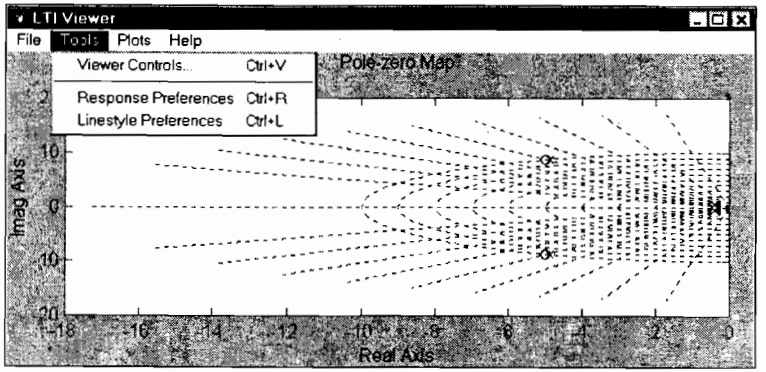

Рис. 6.19

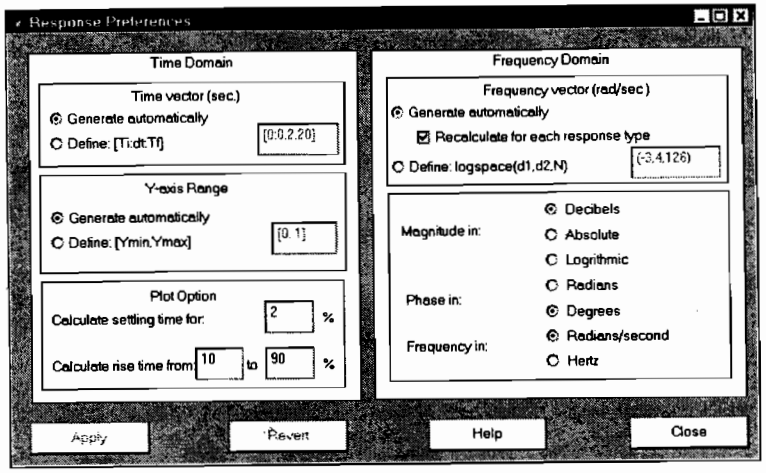

Рис. 6.20

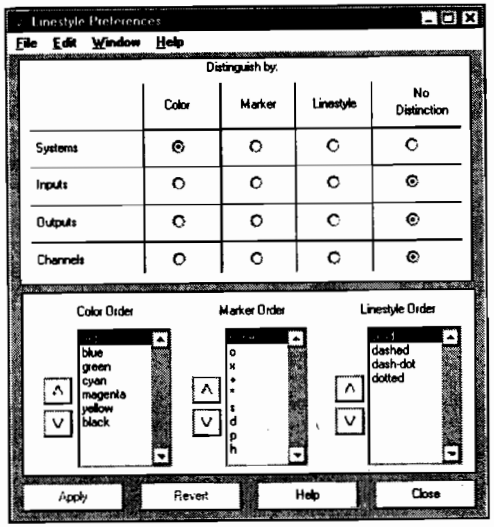

Рис. 6.21

Следующее меню обозревателя, **Plots** (Графики), содержит только одну опцию — **Grid On** (Сетка включена). Если опция установлена, то на графики накладывается сетка из координатных линий, например, как на рис. 6.22.

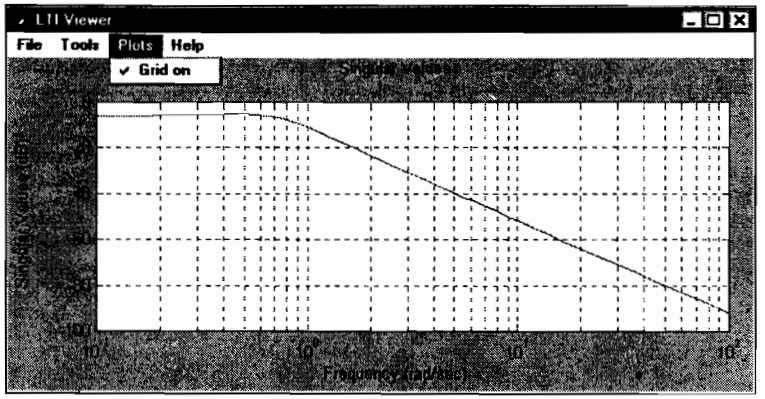

Рис. 6.22

В противном случае сетка координатных линий на графики не наносится (рис. 6.23). Установка или отключение опции осуществляется при помощи мыши.

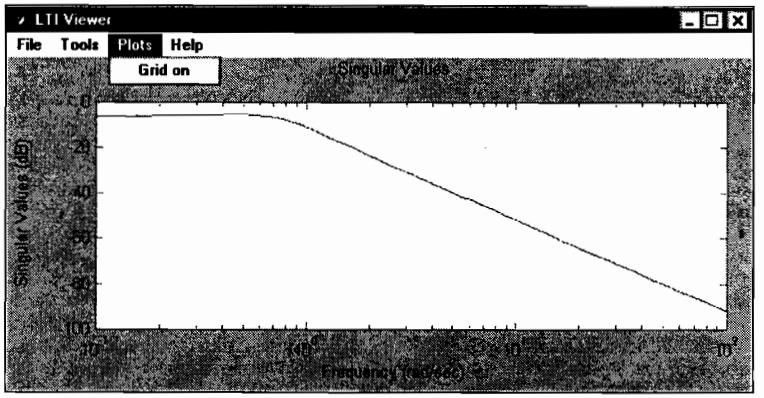

Рис. 6.23

6.5. Синтез системы

Под синтезом САУ обычно понимают процесс разработки (проектирования, расчета параметров) одного из звеньев САУ, обеспечивающего заданное ее качество. Пакет Control содержит несколько процедур, осуществляющих проектирование звеньев, использование которых в контуре системы управления делает САУ оптимальной в некотором, вполне определенном смысле.

Процедура **lqr** осуществляет проектирование линейно-квадратичного оптимального регулятора для систем непрерывного времени. При обращении вида **[K,S,E] = lqr(A,B,Q,R,N)** процедура рассчитывает оптимальное статическое матричное звено K такое, что использование его в цепи отрицательной обратной связи в пространстве состояния:

$$u = -Kx \tag{6.1}$$

минимизирует функционал

$$J = \int \{x'Qx + u'Ru + 2*x'Nu\}\, dt \tag{6.2}$$

если объект регулирования описывается уравнениями состояния:

$$\frac{dx}{dt} = A \cdot x + B \cdot u \tag{6.3}$$

Если последняя матрица N при обращении к процедуре не указана, то она принимается по умолчанию нулевой. Одновременно вычисляется решение S алгебраических уравнений Риккати:

$$S \cdot A + A' \cdot S - (S \cdot B + N) \cdot R^{-1} \cdot (B'S + N') + Q = 0 \tag{6.4}$$

и находятся собственные значения E замкнутой системы:

$$E = \mathrm{EIG}(A - B*K) \tag{6.5}$$

Применяя эту процедуру к ранее введенной САУ движением торпеды, получим:

```
» [A,B,C,D]=ssdata(sssys)
» Q=eye(4)
» R=1
» [K,S,E] = lqr(A,B,Q,R)
K =
   0.4417    0.2773    0.5719    0.2926
S =
   0.8834    0.5546    1.1438    0.5852
   0.5546    0.4497    0.7989    0.4353
   1.1438    0.7989    1.9896    1.0933
   0.5852    0.4353    1.0933    1.7924
E =
  -4.8886 + 8.6016i
  -4.8886 - 8.6016i
  -0.4718 + 0.6195i
  -0.4718 - 0.6195i
```

Следующая процедура **lqry** также применяется для систем "непрерывного времени". Она отличается тем, что, во-первых, проектируемая обратная связь по состоянию рассчитывается как дополнительная по отношению к существующим (а не как заменяющая уже существующие) и охватывающая только регулируемый объект. Во-вторых, функционал минимизируется не по вектору состояния, а по выходной величине (величинам) системы:

$$J = \int \{y'Qy + u'Ru + 2*y'Nu\}\, dt \tag{6.6}$$

В этом случае входным параметром процедуры является сама SS-модель системы в форме:

$$\frac{dx}{dt} = A \cdot x + B \cdot u, \quad y = C \cdot x + D \cdot u \tag{6.7}$$

а вызываться процедура должна таким образом:

```
[K,S,E] = lqry(sys,Q,R,N)
```

где sys — имя LTI-модели оптимизируемой САУ. Та же процедура может быть применена для дискретной системы (модели), уравнения состояния которой заданы в виде конечно-разностных уравнений вида:

$$x[n+1] = Ax[n] + Bu[n], \quad y[n] = Cx[n] + Du[n] \tag{6.8}$$

При этом минимизируется функционал:

$$J = \mathrm{Sum}\ \{y'Qy + u'Ru + 2*y'Nu\} \tag{6.9}$$

Применим процедуру к рассматриваемой системе. Получаем:

```
» Q=1; R=1;
» [K,S,E] = LQRY(sssys,Q,R)
K =
   0.0988    0.2471    0.1844

S =
   0.3002    0.1977    0.4941    0.3689
   0.1977    0.1308    0.3270    0.2584
   0.4941    0.3270    0.8172    0.6451
   0.3689    0.2584    0.6451    0.8316
E =
  -4.8653 + 8.5924i
  -4.8653 - 8.5924i
  -0.4222 + 0.6286i
  -0.4222 - 0.6286i
```

Процедура **lqrd** позволяет спроектировать *дискретный* оптимальный линейно-квадратичный регулятор, минимизирующий непрерывный функционал (6.2). Обращение к процедуре:

```
[K,S,E] = lqrd(A,B,Q,R,N,Ts)
```

где Ts — заданный период дискретизации, приводит к расчету матрицы K статического звена (6.1) обратной связи по вектору состояния системы. При этом модель системы должна быть задана в конечно-разностной форме (6.8).

Проектирование оптимального линейного дискретного регулятора для *дискретной* системы с использованием *дискретного* функционала (6.9) можно осуществить, используя процедуру **dlqr**, например, таким образом:

```
[K,S,E] = dlqr(A,B,Q,R,N,Ts)
```

Уравнения состояния системы должны быть предварительно приведены к конечно-разностной форме (6.8). Матрица S в этом случае представляет собой решение уравнения Риккати в виде:

$$A'SA - S - (A'SB+N)(R+B'SB)(B'SA+N') + Q = 0 \quad (6.10)$$

Процедура **kalman** осуществляет расчет фильтра Калмана для непрерывных или дискретных систем автоматического управления. Обращение к процедуре имеет вид **[KEST,L,P] = kalman(SYS,Qn, Rn,Nn)**, где SYS — имя модели системы. Для непрерывной системы:

$$\frac{dx}{dt} = A \cdot x + B \cdot u + G \cdot w \text{ (уравнение состояния)} \quad (6.11)$$

$$y = C \cdot x + D \cdot u + H \cdot w + v \text{ (уравнение измерения)} \quad (6.12)$$

с известными входами u, шумовым процессом w, шумом измерения v и шумами ковариаций:

$$E\{ww'\} = Qn, \quad E\{vv'\} = Rn, \quad E\{wv'\} = Nn \quad (6.13)$$

фильтр KEST имеет вход [u;y] и генерирует оптимальные оценки y_e и x_e соответственно величин y и x путем решения уравнений:

$$\frac{dx_e}{dt} = A \cdot x_e + B \cdot u + L \cdot (y - C \cdot x_e - D \cdot u) \quad (6.14)$$

$$y_e = C \cdot x_e + D \cdot u \quad (6.15)$$

При этом LTI-модель SYS системы должна содержать данные в виде (A,[B G],C,[D H]). Фильтр Калмана KEST является непрерывным, если SYS представлена как непрерывная система, и дискретным в противном случае. Процедура вычисляет также матрицу L коэффициентов усиления фильтра и матрицу P ковариаций ошибок оценивания состояния. Для непрерывной системы и H=0 матрица P рассчитывается как решение уравнения Риккати:

$$AP + PA' - (PC'+G*N)R^{-1}(CP+N'*G') + G*Q*G' = 0 \quad (6.16)$$

Если система SYS задана конечно-разностными уравнениями:

$$x[n+1] = Ax[n]+Bu[n]+Gw[n] \text{ (уравнение состояния)} \quad (6.17)$$

$$y[n] = Cx[n]+Du[n]+Hw[n]+v[n] \text{ (уравнение измерения)} \quad (6.18)$$

то обращение **[KEST,L,P,M,Z] = kalman(SYS,Qn,Rn,Nn)** позволяет спроектировать дискретный фильтр Калмана для заданной дискретной системы по заданным матрицам ковариаций E{ww'} = Qn, E{vv'}= = Rn, E{wv'} = Nn.

Фильтр Калмана в соответствии с разностными уравнениями:

$$\begin{aligned} x[n+1|n] &= Ax[n|n-1]+Bu[n]+L(y[n]-Cx[n|n-1]-Du[n]) \\ y[n|n] &= Cx[n|n]+Du[n] \\ x[n|n] &= x[n|n-1]+M(y[n]-Cx[n|n-1]-Du[n]) \end{aligned} \quad (6.19)$$

генерирует оптимальные оценки y[n|n] выхода и x[n|n] —переменных состояния, используя значения u[n] входа системы и y[n] — измеренного выхода.

Помимо KEST программа выдает матрицы L оптимальных коэффициентов усиления фильтра и M обновителя, а также матрицы ковариаций ошибок оценивания вектора состояния:

$$P = E\{(x-x[n|n-1])(x-x[n|n-1])'\} \text{ (решение уравнений Риккати)}$$

$$Z = E\{(x-x[n|n])(x-x[n|n])'\} \text{ (апостериорная оценка)}$$

Процедура **[KEST,L,P,M,Z] = kalmd(SYS,Qn,Rn,Ts)** создает дискретный фильтр (оцениватель) Калмана KEST для непрерывной системы, описываемой уравнениями (6.11) и (6.12) при H = 0 и таких параметрах шумов: E{w} = E{v} = 0, E{ww'} = Qn, E{vv'} = = Rn, E{wv'} = 0. Кроме параметров оценивателя процедура вычисляет и выдает ранее описанные матрицы L, M, P и Z.

Задача построения (формирования) оптимального регулятора решается в MatLAB при помощи процедуры **lqgreg**. Если обратиться к процедуре **RLQG = lqgreg(KEST,K)**, то в матрице RLQG создается регулятор, соединяя предварительно спроектированный фильтр Калмана KEST со статическим звеном оптимальной обратной связи по вектору состояния, спроектированным процедурами (D)LQR или LQRY. Регулятор RLQG, входом которого является выход y системы, генерирует команды u = -K x_e (x_e является оценкой Калмана вектора состояния, основанной на измерениях y) и должен быть подсоединен к исходной системе как положительная обратная связь.

Предыдущие процедуры опираются на некоторые вспомогательные процедуры, которые, однако, имеют и самостоятельное значение и могут использоваться при синтезе САУ. К таким процедурам можно отнести:

estim	Формирует оцениватель по заданной матрице коэффициентов передачи оценивателя по выходам и вектору состояния
care	Находит решение непрерывных алгебраических уравнений Риккати
dare	Находит решение дискретных алгебраических уравнений Риккати
lyap	Находит решение непрерывных уравнений Ляпунова
dlyap	Находит решение дискретных уравнений Ляпунова

Так, обращение **EST = estim(SYS,L)** формирует оцениватель EST по заданной матрице L для выходов и вектора состояния системы, заданной SS-моделью SYS, в предположении, что все входы системы SYS являются стохастическими, а все выходы — измеряемыми. Для непрерывной системы вида (6.7), где u — стохастические величины, создаваемый оцениватель:

$$\frac{dx_e}{dt} = (A_e - L \cdot C) \cdot x_e + L \cdot y$$

$$y_e = C \cdot x_e$$

генерирует оценки y_e и x_e соответственно выходов и вектора состояния. Подобным образом процедура применяется и для дискретных систем.

К процедуре **care** следует обращаться по такому образцу: **[X,L,G,RR] = care(A,B,Q,R,S,E)**. В этом случае она выдает решение X алгебраического уравнения Риккати:

$$A'XE + E'XA - (E'XB + S)R^{-1}(B'XE + S') + Q = 0$$

что эквивалентно:

$$F'XE + E'XF^{-1} - E'XBR^{-1}B'XE + Q - SR^{-1}S' = 0$$

где F:=A - $BR^{-1}S'$.

Если при обращении к процедуре пропущены входные параметры R, S и E, то по умолчанию им присваиваются такие значения R=I, S=0 и E=I (I — единичная матрица). Кроме того, процедура вычисляет:

- матрицу коэффициентов усиления G = R^{-1}(B'XE + S');
- вектор L собственных значений замкнутой системы (т.е. EIG(A--B*G,E));
- норму RR Фробениуса матрицы относительных остатков.

Процедура **[X,L,G,RR] = dare(A,B,Q,R,S,E)** вычисляет решение уравнения Риккати для дискретного времени:

$$E'XE = A'XA - (A'XB + S)(B'XB + R)^{-1}(A'XB + S)' + Q$$

что эквивалентно (если R не вырождена):

$$E'XE = F'XF - F'XB(B'XB + R)^{-1}B'XF + Q - SR^{-1}S'$$

где F:=A-$BR^{-1}S'$

В этом случае

$$G = (B'XB + R)^{-1}(B'XA + S')$$

Рассмотрим теперь процедуру **lyap**. Обращение **X = lyap(A,C)** позволяет найти решение X матричного уравнения Ляпунова:

$$A*X + X*A' = -C$$

а обращение **X = LYAP(A,B,C)** — решение общей формы матричного уравнения Ляпунова (называемого также уравнением Сильвестра):

$$A*X + X*B = -C$$

Аналогично, процедура **X = DLYAP(A,Q)** находит решение дискретного уравнения Ляпунова:

$$A*X*A' - X + Q = 0$$

Глава 7

Моделирование нелинейных систем (пакет SimuLink)

7.1. Общая характеристика пакета SimuLink

Пакет SimuLink позволяет осуществлять исследование (моделирование) поведения динамических нелинейных систем. Ввод характеристик исследуемых систем производится в диалоговом режиме, путем графической сборки схемы соединений элементарных стандартных звеньев. В результате такой сборки образуется модель исследуемой системы, которую в дальнейшем будем называть S-моделью. Модель хранится в файле с расширением .mdl.

Создание моделей в пакете SimuLink основано на использовании технологии Drag-and-Drop. В качестве "кирпичиков" для построения S-модели применяются модули (или блоки), хранящиеся в библиотеке SimuLink. Любая S-модель может иметь иерархическую структуру, т.е. состоять из моделей более низкого уровня, причем число уровней иерархии практически не ограничено. В ходе моделирования имеется возможность наблюдать за процессами, происходящими в системе. Для этого используются специальные смотровые окна, входящие в библиотеку SimuLink.

Состав библиотеки SimuLink может быть пополнен пользователем за счет разработки собственных блоков.

7.1.1. Запуск пакета SimuLink

Запуск пакета SimuLink можно произвести из командного окна MatLAB, выбрав команду **New Model** (Новая модель) из меню **File** (Файл) или нажав соответствующую пиктограмму в панели инструментов.

При запуске SimuLink открываются два окна (рис. 3.7):

- пустое окно **untitled** (заготовка для создания новой S-модели, MDL-файла или схемного изображения моделируемой системы);
- окно **Library: simulink** с перечнем основных разделов библиотеки SimuLink (рис. 7.1).

Оба окна имеют сходную структуру и содержат строку меню, панель инструментов (версии MatLAB 5.2 и 5.3) и рабочее поле.

Меню **File** (Файл) включает команды работы с MDL-файлами, меню **Edit** (Правка) — команды редактирования блок-схемы и работы с библиотекой, а меню **View** (Вид) (начиная с версии 5.2) — команды

изменения внешнего вида окна (показать/убрать панель инструментов и строку состояния). В меню **Simulation** (Моделирование) находятся команды управления моделированием, а в меню **Format** (Формат) — команды редактирования формата (внешнего вида) блоков схемы и блок-схемы в целом.

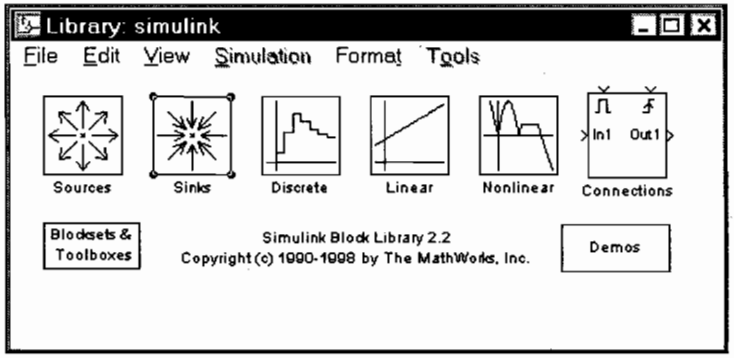

Рис. 7.1

Если в состав конфигурации MatLAB включено приложение Real-Time-Workshop, то меню дополняется разделом **Tools** (Сервис), содержащим средства работы с ним.

7.1.2. Библиотека модулей (блоков)

Библиотека блоков SimuLink представляет собой набор визуальных объектов, при помощи которых можно, соединяя модули линиями функциональной связи, составлять блок-схему любого устройства.

Библиотека блоков разбита на семь разделов. Шесть из них являются основными и не могут изменяться пользователем:

- **Sources** (Источники);
- **Sinks** (Приемники);
- **Discrete** (Дискретные элементы);
- **Linear** (Линейные элементы);
- **Nonlinear** (Нелинейные элементы);
- **Connections** (Связи (соединения)).

Седьмой раздел — **Blocksets & Toolboxes** (Наборы блоков и инструменты) — содержит блоки, созданные пользователем и включенные в рабочую конфигурацию пакета.

Блоки, входящие в раздел **Sources** (Источники), предназначены для формирования сигналов, обеспечивающих управление работой S-модели в целом или отдельных ее частей. *Все блоки-источники имеют по одному выходу и не имеют входов.*

Блоки, собранные в разделе **Sinks** (Приемники), имеют *только входы и не имеют выходов.* Условно их можно разделить на три вида:

- блоки, используемые как смотровые окна при моделировании;
- блоки, обеспечивающие сохранение промежуточных и выходных результатов моделирования;
- блок управления моделированием, который позволяет прервать моделирование при выполнении тех или иных условий.

В раздел **Discrete** (Дискретные элементы) входят блоки, с помощью которых в модели может быть описано поведение дискретных систем. Различают два основных типа таких систем: *системы с дискретным временем* и *системы с дискретными состояниями.* Блоки, входящие в раздел **Discrete**, обеспечивают моделирование как тех, так и других.

Раздел **Linear** (Линейные элементы) содержит блоки, которые можно условно разделить на две группы:

- блоки общего назначения (сумматоры, интеграторы и т.п.);
- блоки описания линейных стационарных звеньев.

Раздел **Nonlinear** (Нелинейные элементы) самый большой по составу. Он включает 30 блоков, которые условно можно разбить на следующие группы:

- блоки, реализующие элементарные математические функции;
- блоки, обеспечивающие логическую обработку входного сигнала;
- блоки, аппроксимирующие входной дискретный сигнал тем или иным способом;
- блоки, реализующие функцию задержки входного сигнала;
- блоки-переключатели;
- блоки, используемые при моделировании систем автоматического регулирования и управления.

Блоки раздела **Connections** (Связи) предназначены для разработки сложных S-моделей, содержащих модели более низкого уровня (подсистемы), и обеспечивают установление нужных связей между несколькими S-моделями.

Чтобы перейти в окно соответствующего раздела библиотеки, которое содержит графические изображения блоков, следует выполнить двойной щелчок мышью на пиктограмме раздела.

Работа по сборке S-модели заключается в том, что изображения выбранных блоков мышью перетаскиваются из окна раздела библиотеки в окно сборки модели, а затем выходы одних блоков в окне сборки соединяются со входами других блоков.

Технология перетаскивания блока такова: следует установить курсор на изображении блока в окне раздела библиотеки, нажать левую кнопку мыши и, не отпуская кнопку, переместить курсор в поле сборки схемы. Аналогично производятся соединения линиями выходов одних блоков со входами других блоков: необходимо

подвести курсор к нужному выходу некоторого блока, нажать левую кнопку мыши и, не отпуская кнопку, переместить курсор ко входу другого блока и затем отпустить кнопку мыши.

7.1.3. Раздел Sinks

После перехода к разделу **Sinks** на экране появляется окно, изображенное на рис. 7.2.

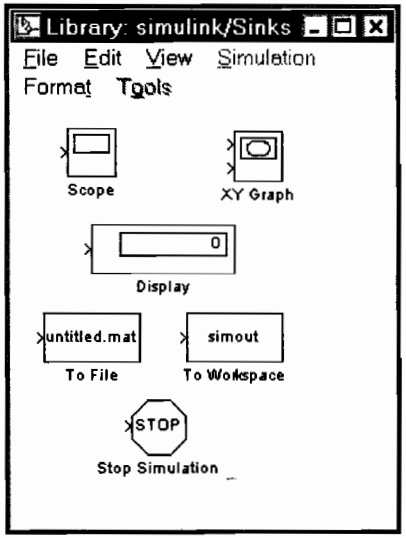

Рис. 7.2

В этом разделе размещены три группы блоков, не имеющих выходов (только входы):

- блоки, которые при моделировании играют роль смотровых окон; к ним относятся:
  - блок **Scope** с одним входом, который выводит график зависимости величины, подаваемой на его вход, от модельного времени;
  - блок **XY Graph** с двумя входами, обеспечивающий построение графика зависимости одной моделируемой величины (один вход) от другой (второй вход);
  - блок **Display** с одним входом, предназначенный для отображения численных значений входной величины;
- блоки для сохранения результатов:
  - блок **To File**, обеспечивающий сохранение результатов моделирования в MAT-файле;

  - блок **To Workspace**, сохраняющий результаты в рабочем пространстве;
- блок управления моделированием **Stop Simulation**, который позволяет прервать моделирование при выполнении некоторых условий; блок срабатывает в случае, когда на его вход поступает ненулевой сигнал.

Блок Scope

Блок **Scope** (Индикатор) позволяет в процессе моделирования наблюдать интересующие исследователя процессы. Для настройки параметров блока следует после установки блока в поле сборки схемы дважды щелкнуть мышью на его изображении. В результате появится окно **Scope** (рис. 7.3).

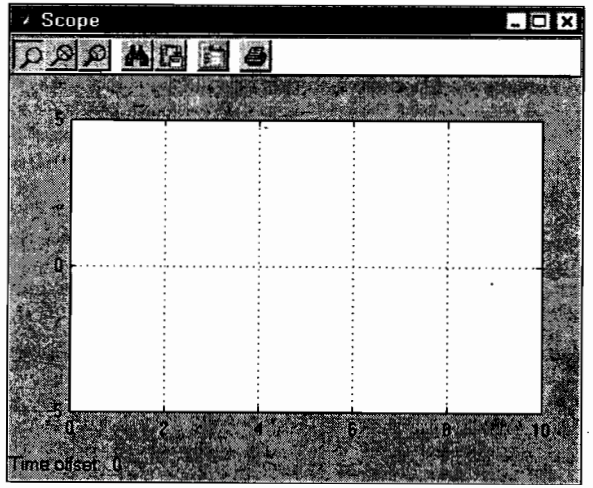

Рис. 7.3

Размер и пропорции этого окна можно изменять произвольно, используя мышь. По горизонтальной оси откладываются значения модельного времени, а по вертикальной — значения входной величины, соответствующие этим моментам времени. Если входная величина блока **Scope** представляет собой вектор, в окне строятся графики изменения всех элементов этого вектора, т.е. столько кривых, сколько элементов во входном векторе, причем каждая — своего цвета. Одновременно в окне может отображаться до 30 кривых.

Для управления параметрами графиков и выполнения над ними различных действий в окне имеется панель инструментов, содержащая 7 пиктограмм (рис. 7.3) со следующим назначением:

Пиктограмма	Назначение
	Изменение масштаба осей графика
	Изменение масштаба по горизонтальной оси
	Изменение масштаба по вертикальной оси
	Автоматическая установка оптимального масштаба осей (автошкалирование)
	Сохранение установленного масштаба осей
	Вызов окна настройки параметров блока **Scope**
	Печать содержимого окна **Scope**

Первые три пиктограммы являются альтернативными, т.е. в каждый момент времени может быть активна лишь одна из них. Первые пять пиктограмм не активны до тех пор, пока в окне **Scope** нет графика. Активными с самого начала являются лишь последние две пиктограммы.

Нажатие шестой пиктограммы приводит к появлению окна настройки параметров (свойств) **Properties: Scope** (рис.7.4).

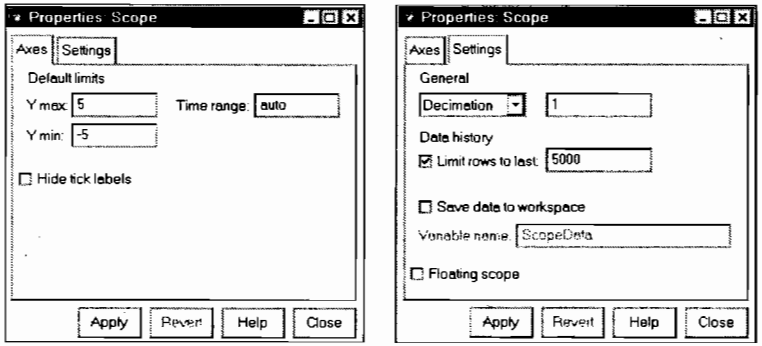

Рис. 7.4

Это окно имеет две вкладки: **Axes** (Оси), которая позволяет установить параметры осей, и **Setting** (Установки), которая предназначена для ввода значений дополнительных параметров блока **Scope**.

В нижней части окна расположены кнопки **Apply** (Применить), **Revert** (Вернуть исходные значения), **Help** (Справка) и **Close** (Закрыть).

На вкладке **Axes** имеются область **Default limits** (Пределы по умолчанию) и опция **Hide tick label** (Скрыть обозначения осей). В области **Default limits** устанавливаются верхняя (поле **Y max**) и нижняя (поле **Y min**) границы оси ординат, а также верхний предел модельного времени, отображаемого на оси абсцисс (поле **Time range**). При этом следует принимать во внимание следующее.

Если величина заданного интервала моделирования ($T_м$) не превышает установленное в поле **Time range** значение (т.е. весь процесс вмещается в окне **Scope**), то под графиком в строке **Time offset** выводится значение 0. В случае, когда интервал моделирования превышает значение в поле **Time range**, в окне **Scope** отображается только последний отрезок времени, меньший чем Time range и равный $T_м$ - n*Time range, где n — целое число. При этом в строке **Time offset** выводится величина скрытого интервала времени, т.е. n*Time range.

Например, если значение в поле **Time range** равно 3, а длительность интервала моделирования установлена равной 17, то в окне **Scope** будет выведен график моделируемого процесса за последние 2 единицы времени, а строка под графиком будет содержать число 15.

Примечание

Любые произведенные в окне **Properties** изменения оказывают влияние на окно **Scope** лишь в случае, если после их ввода нажата кнопка **Apply** в нижней части окна **Properties**.

Опция **Hide tick label** позволяет изменить форму вывода графика в окне **Scope**. Если она установлена, то график занимает все поле окна и на него не наносятся никакие надписи по осям и заголовок.

На вкладке **Setting** (рис. 7.4) имеются следующие элементы. Список **General** обеспечивает выбор дискретности измерения отображаемых величин. При этом выбор пункта **Decimation** позволяет задать дискрет измерения характеристик системы в виде целого числа шагов (дискретов) времени, а выбор пункта **Sample Time** — дискрет изменения модельного времени. Для ввода требуемых значений используется строка ввода, расположенная правее списка. По умолчанию дискрет (шаг) по времени равен нулю, т.е. модельное время предполагается непрерывным, а дискретность измерения равна единице, т.е. измерение осуществляется на каждом шаге времени.

Область **Data history** позволяет задавать максимальный объем и способ хранения отображаемых в окне **Scope** данных. Объем сохраняемых данных (опция **Limit rows to last**) вводится в поле ввода. Способ хранения задается при помощи опции **Save data to workspace**. Если она установлена, то отображаемые в окне **Scope** данные

сохраняются в рабочем пространстве MatLAB в виде матрицы, аналогичной по структуре MAT-файлу. Имя матрицы указывается в строке ввода **Variable name** (по умолчанию — ScopeData).

Опция **Floating scope** предназначена для изменения способа использования блока **Scope** в блок-схеме. Когда она установлена, то **Scope** отображается как блок *без входа* (если он был связан по входу с другими блоками, то эти связи обрываются).

Блок XY Graph

Этот блок также представляет собой смотровое окно. В отличие от **Scope**, он имеет два входа: на первый подается сигнал, значения которого откладываются по горизонтальной оси графика, а на второй — по вертикальной оси. Если перетащить этот блок в поле блок-схемы, а затем выполнить на его изображении двойной щелчок, то появится окно настройки блока (рис. 7.5). Оно позволяет установить пределы изменений входных величин, внутри которых будет выводиться график зависимости второй величины от первой, а также задать дискрет по времени.

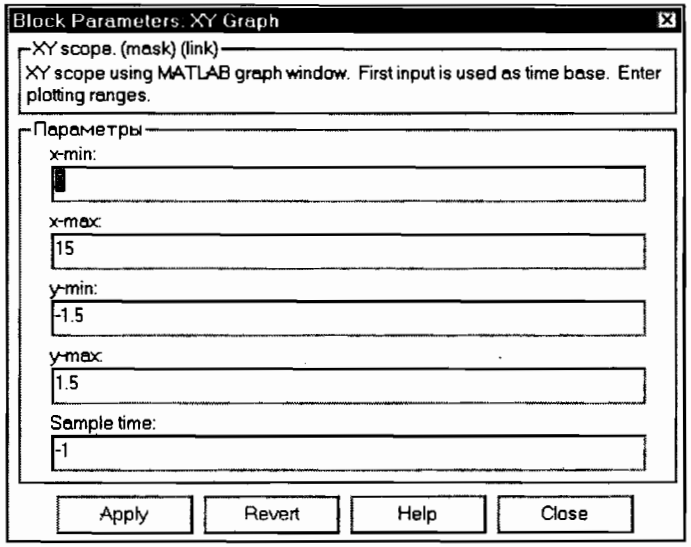

Рис. 7.5

Приведем пример использования блока **XY Graph**. Для этого в окно блок-схемы из окна **Library simulink/Sinks** перетащим изображение этого блока, а из окна **Library simulink/Sources** — два блока-источника: **Clock** и **Sine Wave**. Соединим выходы блоков-источников со входами блока **XY Graph**. Получим блок-схему, приведенную на рис. 7.6.

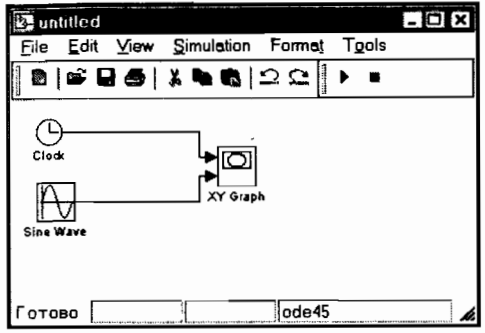

Рис. 7.6

Если теперь в меню **Simulation** (Моделирование) окна блок-схемы выбрать команду **Start** (Запуск), то по окончании расчетов в окне блока **XY Graph** появится изображение, приведенное на рис. 7.7.

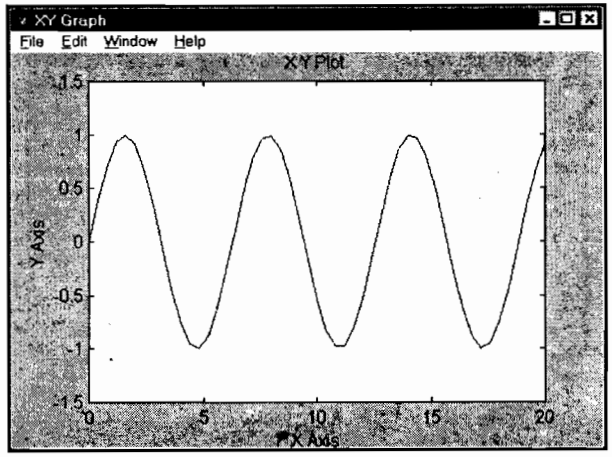

Рис. 7.7

Блок Display

Блок предназначен для вывода на экран численных значений величин, фигурирующих в блок-схеме. Он имеет 4 параметра настройки. Список **Format** задает формат вывода чисел и содержит 5 пунктов: short, long, short_e, long_e, bank. Поле ввода **Decimation** позволяет задать периодичность (через сколько дискретов времени) вывода значений в окне **Display**. Опция **Floating display** позволяет определять блок **Display** как блок без входа, обрывая его связи.

Блок **Display** может использоваться для вывода как скалярных, так и векторных величин. Если отображаемая величина является вектором, то исходное вид блока изменяется автоматически, о чем свидетельствует появление маленького черного треугольника в правом нижнем углу блока. Для каждого элемента вектора создается свое мини-окно, но чтобы они стали видимы, необходимо растянуть изображение блока. Для этого следует выделить блок, подвести курсор к одному из его углов, нажать левую кнопку мыши и, не отпуская ее, растянуть изображение блока так, чтобы черный треугольник исчез.

Для примера создадим блок-схему (рис. 7.8) из двух элементов: блока-источника **Constant** и блока-приемника **Display**. Откроем окно настройки блока **Constant** (рис. 7.9) и установим в нем значение константы-вектора, состоящее из четырех элементов [1e-17 pi 1757 -0.087]. Затем вызовем окно настройки блока **Display** и установим с его помощью формат вывода чисел Short_e. После активизации команды **Start** из меню **Simulation** блок-схема будет иметь вид, показанный на рис. 7.8. Растянув изображение блока **Display** на блок-схеме, получим картину, представленную на рис. 7.10.

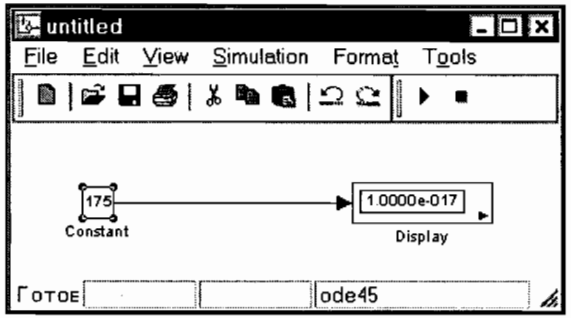

Рис. 7.8

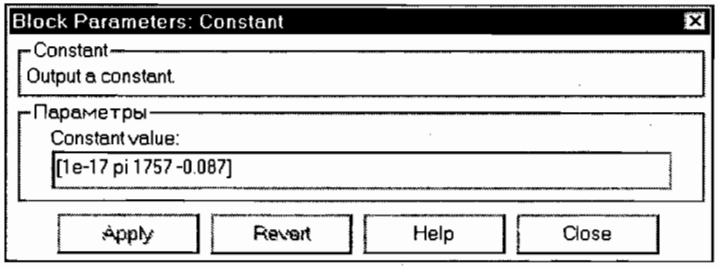

Рис. 7.9

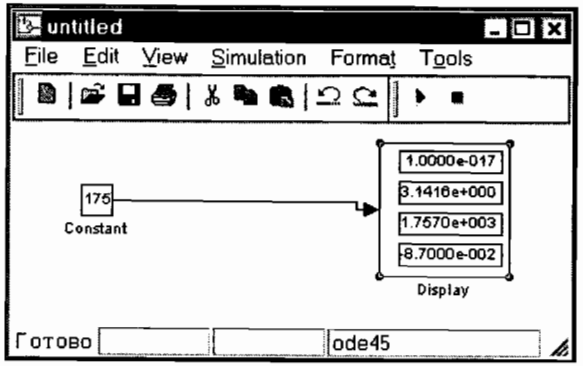

Рис. 7.10

Блок To File

Этот блок обеспечивает запись значений величины, поданной на его вход, в MAT-файл для использования их в других S-моделях.

Блок имеет следующие параметры настройки:

- **File name** — имя MAT-файла, в который будут записываться значения входной величины; по умолчанию — untitled.mat; имя файла выводится на изображении блока в блок-схеме;
- **Variable name** — имя переменной, по которому можно будет обращаться к данным, записанным в файле (для того чтобы просмотреть или изменить их в командном окне MatLAB); по умолчанию используется системное имя **ans**;
- **Decimation** — дискретность (в количестве дискретов времени) записи данных в файл;
- **Sample Time** — величина дискрета времени для данного блока.

Блок To Workspace

Этот блок предназначен для сохранения данных в рабочем пространстве системы MatLAB. Данные записываются в виде матрицы, структура которой отличается от структуры данных в MAT-файле тем, что:

- значения сохраняемых величин расположены по строкам, а не по столбцам;
- не записываются значения модельного времени.

Блок имеет 4 параметра настройки:

- **Variable name** — имя, под которым данные сохраняются в рабочем пространстве (по умолчанию — **simout**);
- **Maximum number of rows** — максимально допустимое число строк, т.е. значений записываемых данных; по умолчанию задается константой **inf** (данные регистрируются на всем интервале моделирования);

- **Decimation** и **Sample Time** имеют тот же смысл, что и ранее.

7.1.4. Раздел Sources

После выбора раздела **Sources** библиотеки SimuLink появляется окно, показанное на рис. 7.11. Как видим, в этом разделе библиотеки в качестве источников сигналов предусмотрены следующие блоки:

- **Constant** — формирует постоянную величину (скаляр, вектор или матрицу);
- **Signal Generator** — создает по выбору непрерывный колебательный сигнал одной из волновых форм: синусоидальная, прямоугольная, треугольная и случайная;
- **Step** — генерирует сигнал в виде одиночной ступеньки (скачкообразный сигнал) с заданными параметрами (начала ступеньки и ее высоты);
- **Ramp** — создает линейно нарастающий/убывающий сигнал (ступеньку по скорости или импульсный сигнал);

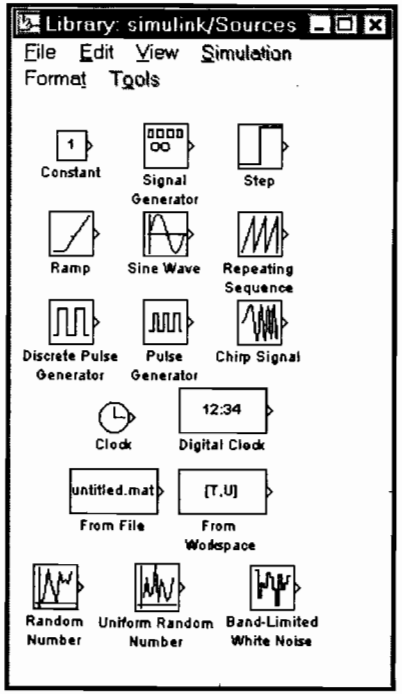

Рис. 7.11

- **Sine Wave** — генерирует гармонический сигнал;
- **Discrete Pulse Generator** — генератор дискретных импульсных сигналов;
- **Chirp Signal** — генератор гармонических колебаний с частотой, линейно изменяющейся во времени;
- **Clock** — источник непрерывного временного сигнала;
- **Digital clock** — формирует дискретный временной сигнал;
- **Random Number** — источник дискретного сигнала, амплитуда которого является случайной величиной, распределенной по нормальному закону;
- **Uniform Random Number** — источник дискретного сигнала, амплитуда которого является случайной равномерно распределенной величиной;
- **Band-Limited White Noise** — генератор белого шума с ограниченной полосой.

Два блока из раздела **Sources** обеспечивают использование в модели данных, полученных ранее. Первый из них — **From File** — предназначен для ввода в S-модель данных, хранящихся в MAT-файле. Второй — **From Workspace** — обеспечивает ввод в модель данных непосредственно из рабочего пространства MatLAB.

Структура данных в MAT-файле представляет собой многомерный массив с переменным числом строк, которое определяется числом регистрируемых переменных. Элементы первой строки содержат последовательные значения модельного времени, элементы в других строках — соответствующие им значения переменных.

Как и другие блоки библиотеки SimuLink, блоки раздела **Sources** могут настраиваться пользователем. Исключением является блок **Clock**, работа которого основана на использовании аппаратного таймера компьютера.

Блок Constant

Блок предназначен для установки констант, применяемых при моделировании. Он имеет один параметр настройки — **Constant value**, который может быть введен и как вектор-строка из нескольких элементов по общим правилам MatLAB. Пример его использования приведен ранее при рассмотрении блока **Display**.

Блок Signal Generator

Окно настройки этого блока выглядит так, как показано на рис. 7.12. Как видно, в параметры настройки входят:

- **Wave form** — позволяет выбрать одну из следующих форм периодического процесса: sine — синусоидальные волны; square — прямоугольные волны; sawtooth — треугольные волны; random — случайные колебания;

- **Amplitude** — определяет значение амплитуды генерируемых колебаний;
- **Frequency** — задает частоту колебаний;
- **Units** — позволяет выбрать одну из единиц измерения частоты: Hertz (Герцы) или Rad/Sec (радианы в секунду).

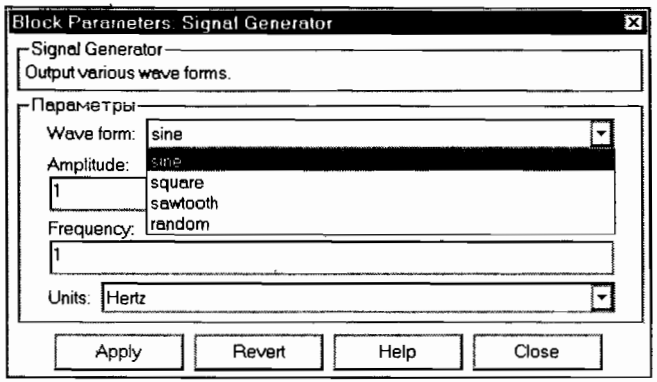

Рис. 7.12

На рис. 7.13 показана простейшая S-модель, состоящая из блока **Signal Generator** и блока отображения **XY Graph**. На следующем рисунке (7.14) представлено содержимое блока отображения после проведения моделирования при таких параметрах настройки: вид колебаний — **Sine**; амплитуда — 4,5; частота — 1 Гц.

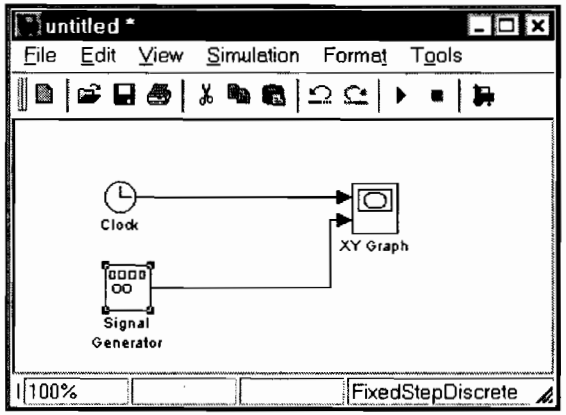

Рис. 7.13

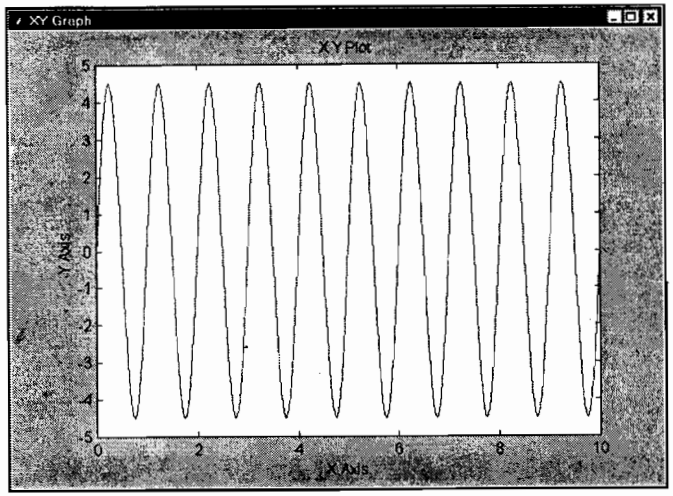

Рис. 7.14

Блок Step

Блок обеспечивает формирование управляющего сигнала в форме ступеньки (или, как говорят, скачкообразного постоянного сигнала). Блок имеет 3 параметра настройки:

- **Step time** (время начала ступеньки, т.е. время скачка сигнала) — определяет момент времени, в который происходит скачкообразное изменение сигнала; по умолчанию принимается равным 1;
- **Initial value** (начальное значение) — задает уровень сигнала до скачка; исходное значение — 0;
- **Final value** (конечное значение) — задает уровень сигнала после скачка; значение его по умолчанию — 1.

Рассмотрим пример использования блока. Перетащим посредством мыши из окна библиотеки источников в окно S-модели блоки **Step** и **Clock**, а из окна библиотеки приемников — блок **XY Graph** и соединим их (рис. 7.15)

Установим следующие параметры настройки блока: **Step time** — 3,5, **Initial value** — -2, **Final value** — 3. После активизации моделирования (команда **Start** из меню **Simulation**) получим в окне **Scope** картину, представленную на рис. 7.16.

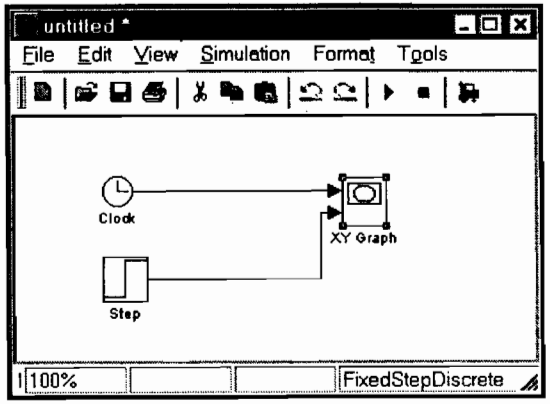

Рис. 7.15

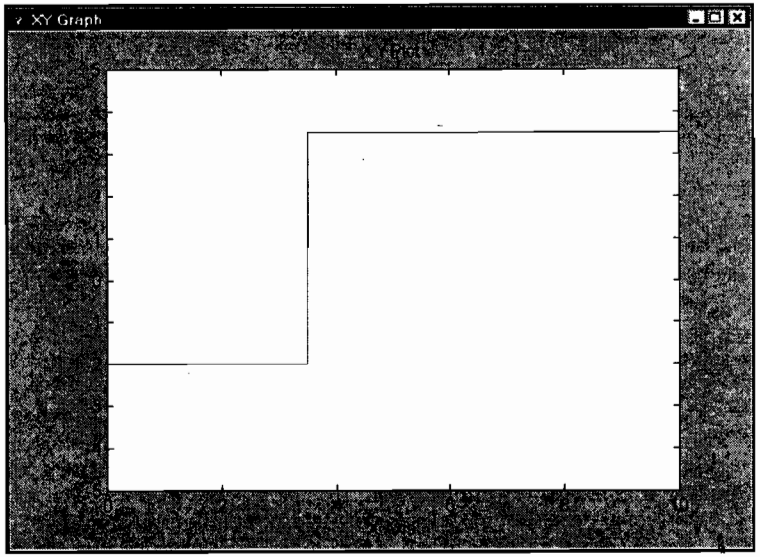

Рис. 7.16

Блок Ramp

Блок формирует постоянно нарастающий сигнал и имеет следующие параметры настройки:

- **Slope** — значение крутизны наклона прямой к оси времени;
- **Start time** — время начала действия нарастающего сигнала;

- **Initial output** — значение сигнала в момент начала его действия.

На рис. 7.17 приведен результат применения блока **Ramp** при следующих значениях параметров: **Slope** — 1, **Start time** — 3, **Initial output** — -3.

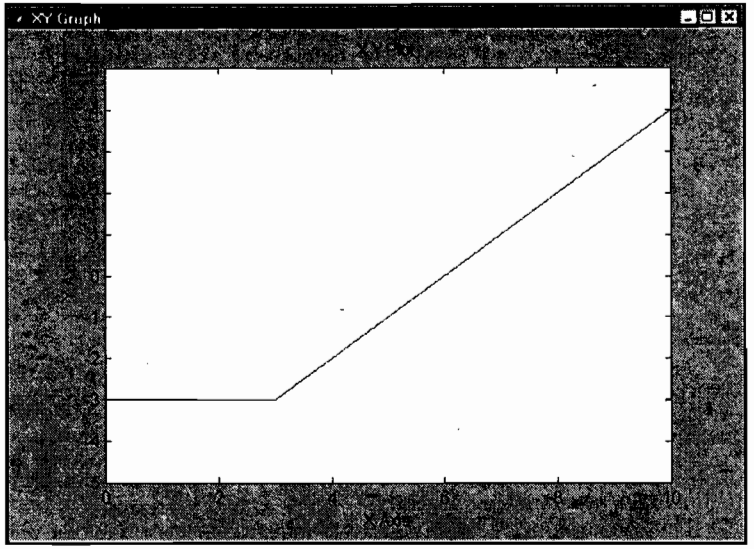

Рис. 7.17

Блок Sine Wave

Блок **Sine Wave** имеет такие настройки:

- **Amplitude** — определяет амплитуду синусоидального сигнала;
- **Frequensy (rad/sec)** — задает частоту колебаний в радианах в секунду;
- **Phase (rad)** — позволяет установить начальную фазу в радианах;
- **Sample time** — определяет величину дискретности по времени задания значений синусоидального сигнала.

На рис. 7.18 представлен результат применения блока при следующих значениях параметров: **Amplitude** — 4,5, **Frequensy** — 1 радиан в секунду и **Phase** — $\pi/2$ радиан.

Отличия этого блока от генератора синусоидальных колебаний в блоке **Signal Generator** состоят в следующем: 1) в рассматриваемом блоке можно устанавливать произвольную начальную фазу; 2) в нем нельзя задать частоту в Герцах.

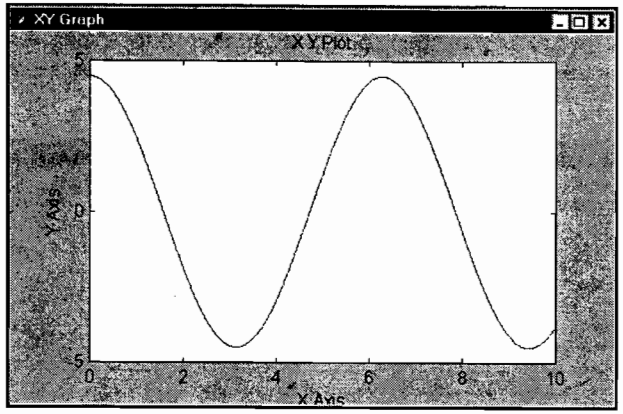

Рис. 7.18

Блок Repeating Sequence

Этот блок содержит две настройки:

- **Time values** — вектор значений времени, в которых заданы значения выходной величины;
- **Output values** — вектор значений выходной величины, которые она должна принять в указанные в первом векторе соответствующие моменты времени.

Блок обеспечивает генерирование колебаний с периодом, равным разности между последним значением вектора **Time values** и значением первого его элемента. Форма волны внутри периода представляет собой ломаную, проходящую через точки с указанными в векторах **Time values** и **Output values** координатами.

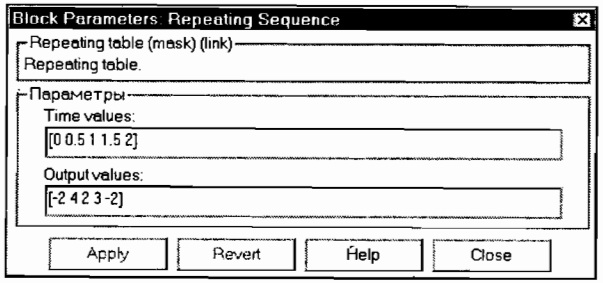

Рис. 7.19

В качестве примера на рис. 7.20 приведено изображение процесса, сгенерированного блоком **Repeating Sequence** при параметрах настройки, указанных на рис. 7.19.

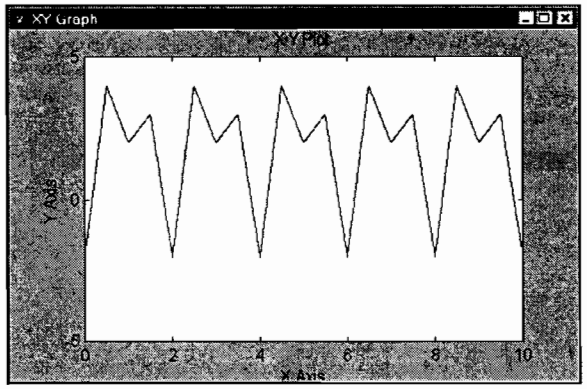

Рис. 7.20

Блок Discrete Pulse Generator

Блок генерирует последовательности прямоугольных импульсов. В число настраиваемых параметров этого блока входят (рис. 7.21):

- **Amplitude** — амплитуда сигнала (высота прямоугольного импульса);
- **Period** — период сигнала, отсчитываемый в целых числах (в дискретах времени);
- **Pulse width** — ширина импульса (также в дискретах времени);
- **Phase delay** — величина задержки импульса относительно t = 0 (в дискретах времени);
- **Sample time** — величина дискрета времени.

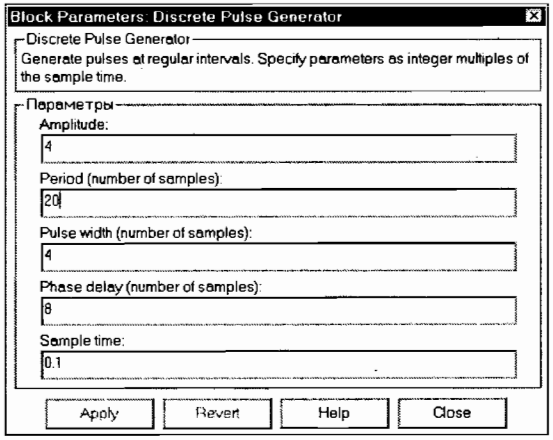

Рис. 7.21

Установку параметров этого блока целесообразно начинать с установки дискрета времени. Размер шага можно указать как в форме константы, так и в форме вычисляемого выражения. Пример применения этого блока при значениях параметров, указанных на рис. 7.21, приведен на рис. 7.22.

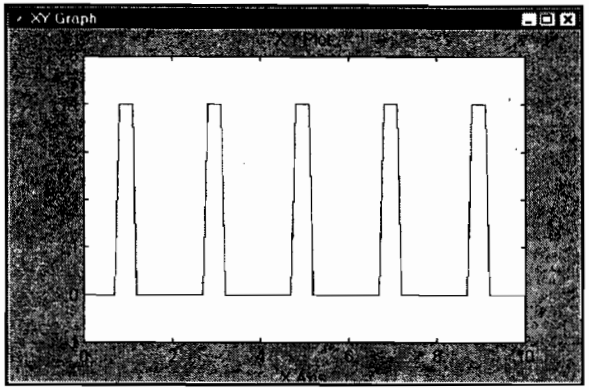

Рис. 7.22

Блок Pulse Generator

Блок **Pulse Generator** осуществляет практически ту же функцию, что и блок **Discrete Pulse Generator**, за исключением того, что параметры прямоугольных импульсов задаются иначе (рис. 7.23):

- **Period** — период следования импульсов;
- **Duty cycle (% of period)** — длительность прямоугольного импульса (в процентах от периода);
- **Amplitude** — амплитуда (высота) импульса;
- **Start time** — начальный момент времени одного из импульсов.

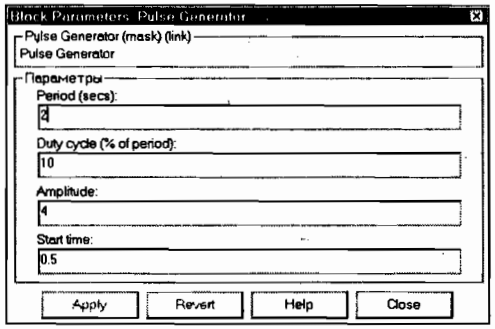

Рис. 7.23

Результат применения блока при значениях параметров, указанных на рис. 7.23, можно увидеть на рис. 7.24.

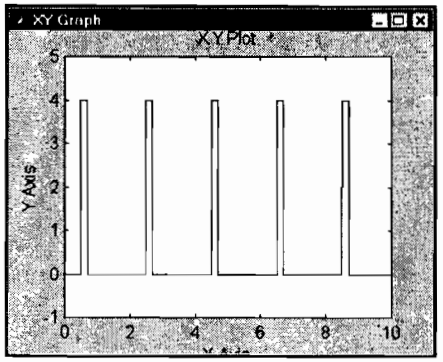

Рис. 7.24

Блок Chirp Signal

Этот блок генерирует синусоидальный сигнал единичной амплитуды и переменной частоты, причем частота колебаний изменяется во времени по линейному закону. В соответствии с этим в нем предусмотрены следующие параметры настройки (рис. 7.25):

- **Initial frequency (Hz)** — начальное значение (при t = 0) частоты (в Герцах);
- **Target time (secs)** — другой (больше нуля) момент времени (в секундах);
- **Frequency at target time (Hz)** — значение частоты в этот второй момент времени.

На рис. 7.26 продемонстрирован результат использования блока при параметрах, указанных на рис. 7.25.

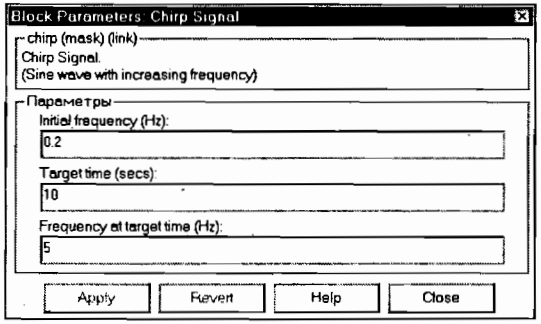

Рис. 7.25

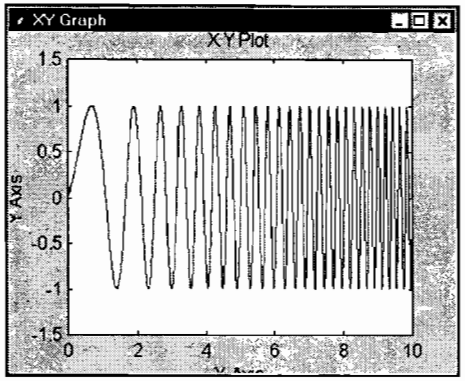

Рис. 7.26

Блок Random Number

Блок **Random Number** обеспечивает формирование сигналов, амплитуда которых является случайной величиной, распределенной по нормальному закону с заданными параметрами. Блок имеет четыре параметра настройки (рис. 7.27).

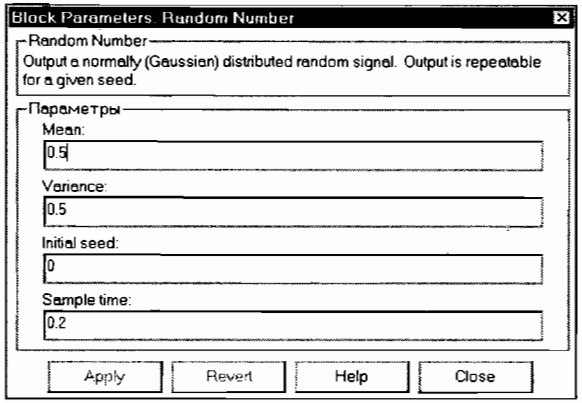

Рис. 7.27

Первые два — **Mean** и **Variance** — являются параметрами нормального закона (среднее и среднее квадратичное отклонение от этого среднего), третий — **Initial seed** — задает начальное значение базы для инициализации генератора последовательности случайных чисел. При фиксированном значении этого параметра генератор всегда вырабатывает одну и ту же последовательность. Четвертый параметр (**Sample time**) задает величину дискрета времени.

На рис. 7.28 показан результат использования блока при значениях параметров, приведенных на рис. 7.27.

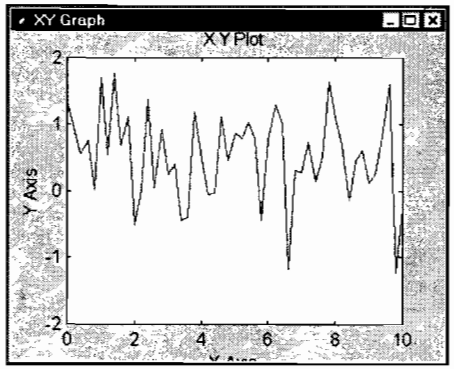

Рис. 7.28

Блок Uniform Random Number

Этот блок формирует сигналы, амплитуда которых является случайной величиной, равномерно распределенной в заданном интервале. В число параметров настройки блока (рис. 7.29) входят:

- **Minimum** — нижняя граница случайных чисел;
- **Maximum** — верхняя граница;
- **Initial seed** — начальное значение базы генератора случайных чисел;
- **Sample time** — дискрет по времени.

Пример процесса, сгенерированного блоком по параметрам, указанным на рис. 7.29, приведен на рис. 7.30

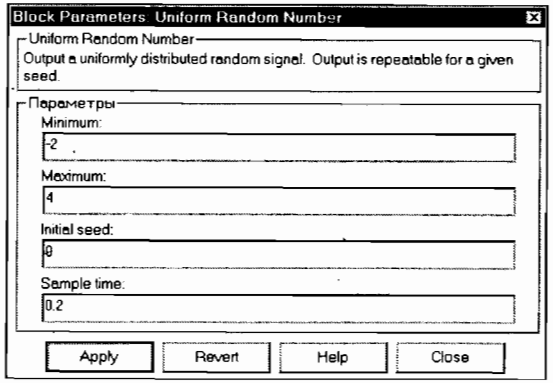

Рис. 7.29

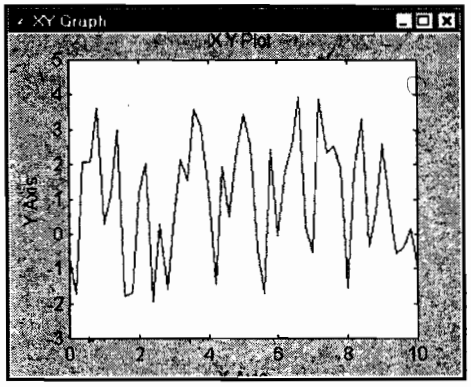

Рис. 7.30

Блок Band-Limited White Noise

Этот блок формирует процесс в виде частотно-ограниченного белого шума. Параметры настройки у него следующие (рис. 7.31):

- **Noise power** — значение мощности белого шума;
- **Sample time** — значение дискрета времени (определяет верхнее значение частоты процесса);
- **Seed** — начальное значение базы генератора случайной величины.

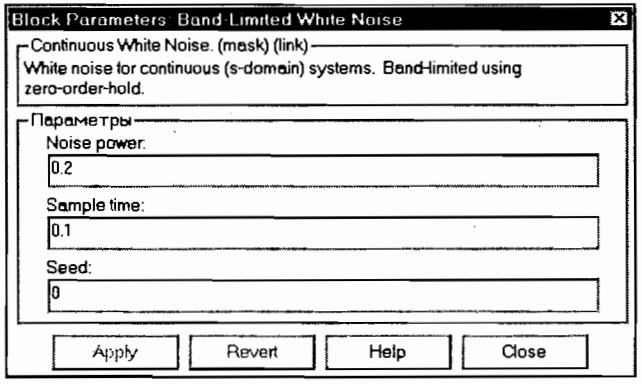

Рис. 7.31

Ниже (рис. 7.32) приведен пример реализации процесса с помощью блока **Band-Limited White Noise** при параметрах, указанных на рис. 7.31.

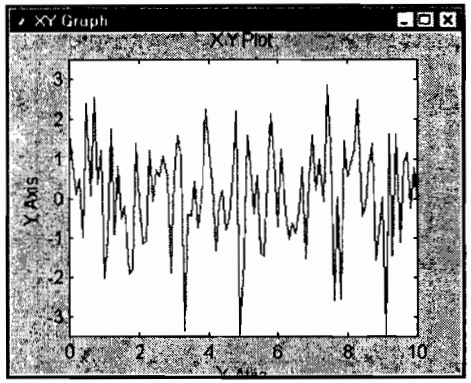

Рис. 7.32

7.1.5. Раздел Linear

Этот раздел библиотеки (рис. 7.33) содержит следующие блоки:

- **Gain** — линейное передаточное звено;
- **Transfer Fcn** — определение линейного звена через задание его передаточной функции;
- **State-Space** — определение звена через задание четырех матриц его пространства состояний;
- **Zero-Pole** — задание звена через указание векторов значений его полюсов и нулей, а также значения коэффициента передачи;
- **Derivative** — идеальное дифференцирующее звено;
- **Dot Product** — звено, осуществляющее перемножение двух входных величин, если они являются скалярами, или определяющее сумму поэлементных произведений элементов двух входных векторов (одинаковой длины);
- **Matrix Gain** — матричное усилительное звено для многомерной системы;
- **Integrator** — идеальное интегрирующее звено (интегратор);
- **Sum** — звено-сумматор;
- **Slider Gain** — звено интерактивного изменения коэффициента усиления.

Те, кто знает основы теории автоматического управления, без труда разберутся в том, как использовать большинство блоков-звеньев. Поэтому мы остановимся более подробно лишь на блоках интегратора (**Integrator**) и сумматора (**Sum**).

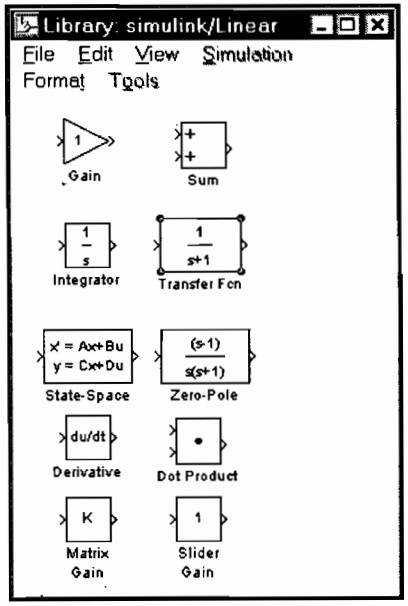

Рис. 7.33

Блок Integrator

Блок осуществляет интегрирование в непрерывном времени входной величины. Он имеет следующие параметры настройки (рис. 7.34):

- **External reset** — подключение дополнительного управляющего сигнала;
- **Initial condition source** — использование внешней установки начального значения входного сигнала;
- **Initial condition** — начальное значение выходной величины; значение вводится в строке редактирования либо как числовая константа, либо в виде вычисляемого выражения;
- **Limit output** (Ограничение выходного значения) — опция, которая определяет, будут ли использоваться следующие ниже 4 параметра;
- **Upper saturation limit** — верхнее предельное значение времени; по умолчанию не ограничено (inf);
- **Lower saturation limit** — нижнее предельное значение времени; по умолчанию параметр имеет значение -inf;
- **Show saturation port** — опция, которая задает отображение порта насыщения;

- **Show state port** — опция, которая задает отображение порта состояния;
- **Absolute tolerance** — допустимая предельная величина абсолютной погрешности.

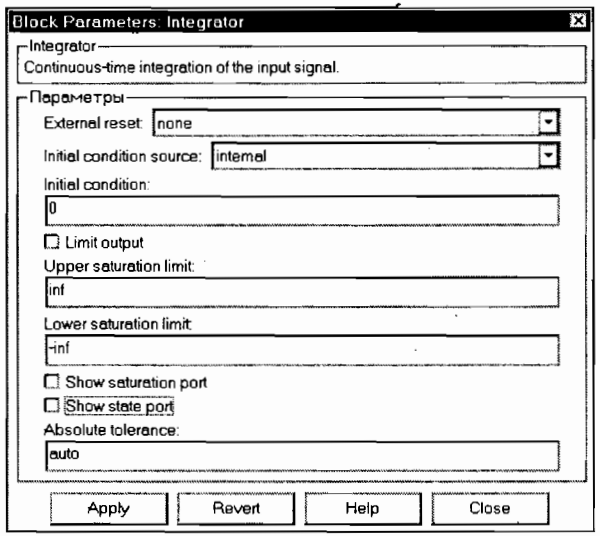

Рис. 7.34

Параметр **External reset** может принимать следующие значения:

none	Дополнительный управляющий сигнал не используется
rising	Для управления используется возрастающий сигнал
falling	Для управления используется убывающий сигнал
either	На работу блока влияет любое изменение амплитуды управляющего сигнала

Параметр **Initial condition source** принимает одно из двух значений:

internal	Используется внутренняя установка начального значения выходной величины
external	Установка начальных условий производится извне

Если выбранные пользователем значения этих двух параметров предполагают наличие дополнительных входных сигналов, то на изображении блока появляются дополнительные входные порты (после нажатия кнопки **Apply** в окне настроек блока). Предпоследние два параметра используются следующим образом. Если опция **Limit output** установлена, то при переходе выходного значения интегратора через верхний или нижний предел на дополнительных

выходах блока (saturation port и state port) формируется единичный сигнал. Чтобы этот сигнал можно было использовать для управления работой S-модели, опции **Show saturation port** и **Show state port** должны быть включены. При этом на изображении блока появляются обозначения портов. Особенность порта state port состоит в том, что снимаемый с него сигнал может быть использован только для прерывания алгебраического цикла или для согласования состояния подсистем модели.

Блок Sum

Блок **Sum** может использоваться в двух режимах:

- сложения входных сигналов (в том числе с разными знаками);
- суммирования элементов вектора, поступающего на вход блока.

Для управления режимами работы блока применяется единственный параметр настройки **List of signs** (Список знаков) (рис. 7.35).

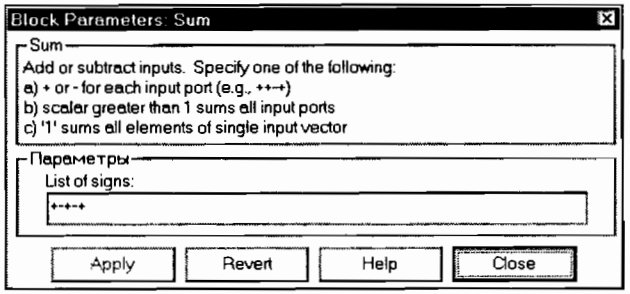

Рис. 7.35

Значения этого параметра можно задать одним из трех способов:

- в виде последовательности знаков "+"или "-"; при этом число знаков определяет число входов блока, а сам знак — полярность соответствующего входного сигнала;
- в виде целой положительной константы большей 1; значение этой константы определяет число входов блока, а все входы считаются положительными;
- в виде символа "1", который указывает, что блок используется во втором режиме.

Блок Slider Gain

Блок **Slider Gain** является одним из элементов взаимодействия пользователя с моделью. Он позволяет в удобной форме изменять в процессе моделирования значение некоторого параметра. Блок становится активным после того, как будет помещен в окно блок-схемы. Чтобы открыть окно с ползунковым регулятором (рис. 7.36), необходимо дважды щелкнуть мышью на изображении данного блока.

Окно **Slider Gain** имеет три поля ввода, которые задают:

- нижний предел изменения параметра (Low);
- верхний предел изменения параметра (High);
- текущее значение.

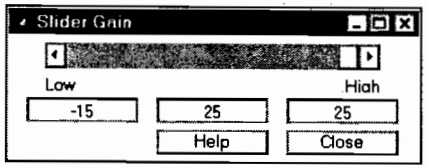

Рис. 7.36

Текущее значение должно лежать внутри диапазона [Low, High]. При выборе нового диапазона необходимо вначале указать новое этого значение параметра, а затем изменить границы диапазона.

7.1.6. Раздел Nonlinear

По составу элементов это самый большой и наиболее полезный раздел библиотеки SimuLink, включающий 30 блоков (рис. 7.37).

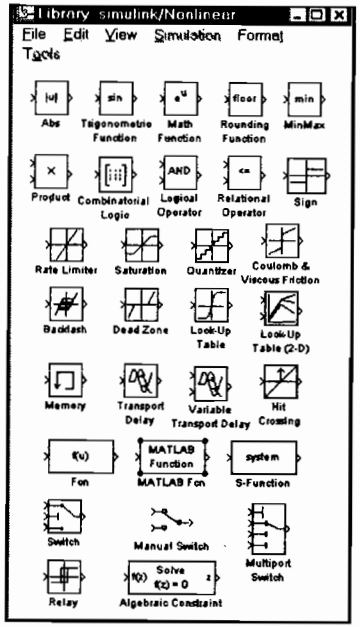

Рис. 7.37

Часть блоков реализуют математические функции.

Блок **Abs** формирует абсолютное значение входного сигнала. Он не имеет параметров настройки. Блок **Trigonometric Function** обеспечивает преобразование входного сигнала с помощью одной из следующих функций MatLAB: **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, **atan2**, **sinh**, **cosh**, **tanh**. Выбор функции производится в окне настройки блока с помощью списка.

Блок **Math Function** позволяет задать для преобразования входного сигнала элементарные не тригонометрические и не гиперболические функции, такие как вычисление экспоненты, натурального и десятичного логарифмов, возведение в степень, извлечение квадратного корня и т.д. Функция выбирается с помощью списка в окне настройки.

Блок **Rounding Function** содержит различные функции округления, предусмотренные в MatLAB. Он осуществляет округление значения входного сигнала. Выбор метода округления производится также с помощью списка в окне настройки.

Блок **MinMax** осуществляет поиск минимального или максимального элемента входного вектора. Если входом является скалярная величина, то выходная величина совпадает со входной. Если входов несколько, ищется минимум или максимум среди входов. В число настроек входит выбор метода и количество входов блока.

Для вышеперечисленных блоков имя выбранной функции выводится на графическом изображении блока.

Блок **Fcn** позволяет пользователю ввести любую *скалярную* функцию от одного (скалярного или векторного) аргумента, выражающуюся через стандартные функции MatLAB. Выражение функции вводится в окне настройки блока. Для обозначения входного сигнала (аргумента функции) используется символ "u".

Блок **MATLAB Fcn** позволяет применить к входному сигналу любую подпрограмму обработки, реализованную в виде M-файла. В отличие от предыдущего блока, здесь к числу параметров настройки добавлен параметр **Output width** (Ширина выходного сигнала), который определяет число элементов выходного вектора.

Блок **Product** выполняет умножение или деление нескольких входных сигналов. В параметры настройки входят число входов блока и вид выполняемой операции. Задание значений этих параметров аналогично настройке блока **Sum** из раздела **Linear**. Если в качестве значения параметра настройки блока ввести 1, то будет вычисляться произведение элементов входного вектора. При этом на изображении блока выводится символ "P".

Вторую группу образуют блоки, обеспечивающие логическую обработку входного сигнала. Выходная величина во всех них является булевой величиной, т.е. может принимать только два значения: 1 ("истина") или 0 ("ложь"). Во многих из них булевыми должны быть и все входные величины.

Блок **Relational Operator** реализует операции отношения над двумя входными сигналами >, <, ≤, ≥, ==, != (больше, меньше, меньше или равно, больше или равно, тождественно, не равно). Конкретная операция выбирается при настройке параметров блока посредством открывающегося списка. Знак операции выводится на изображении блока.

Блок **Logical Operator** содержит набор основных логических операций. Входные величины должны быть булевыми. Выбор логической операции осуществляется в окне настройки блока с помощью списка. Вторым параметром настройки является число входных величин (портов) блока (**Number of input ports**), т.е. число аргументов логической операции.

Блок **Combinatorial Logic** обеспечивает преобразование входных булевских величин в выходную в соответствии с задаваемой таблицей истинности. Блок имеет единственный параметр настройки — **Truth table** (Таблица истинности).

Следующая группа блоков обеспечивает формирование звеньев с некоторыми типовыми нелинейностями. Сюда относятся такие блоки.

Блок **Sign** реализует нелинейность типа сигнум-функции. В нем нет параметров настройки. Блок формирует выходной сигнал, принимающий только три возможных значения: +1 — в случае, когда входной сигнал положителен, -1 — при отрицательном входном сигнале и 0 — при входном сигнале равном нулю.

Блок **Saturation** (Насыщение) реализует линейную зависимость с ограничением. Выходная величина этого блока совпадает со входной, если последняя находится внутри указанного диапазона. Если же входная величина выходит за рамки диапазона, то выходной сигнал принимает значение ближайшей из границ. Значения границ диапазона устанавливаются в окне настройки блока.

Блок **Quantizer** (Квантователь) осуществляет дискретизацию входного сигнала по его величине. Параметр настройки блока один — **Quantization interval** (Интервал квантования) — величина дискрета по уровню входного сигнала.

Блок **Coulomb & Viscous Friction** (Сухое и вязкое трение) реализует нелинейную зависимость типа линейная с предварительным натягом. Если вход положителен, то выход пропорционален входу и увеличен на величину натяга (кулонова трения). Если вход отрицателен, то выход также пропорционален входу (с тем же коэффициентом пропорциональности — коэффициентом вязкого трения) за вычетом величины натяга. При входе, равном нулю, выход также равен нулю. В параметры настройки блока входят величины кулонова трения (натяга) и коэффициента вязкого трения.

Блок **BackLash** (Люфт) реализует нелинейность типа люфта (зазора). В нем предусмотрено два параметра настройки: **Deadband width** (Величина люфта) и **Initial output** (Начальное значение выходной величины).

Блок **Dead Zone** (Мертвая зона) реализует нелинейность типа зоны нечувствительности. Параметров настройки здесь два — начало и конец зоны нечувствительности.

Блок **Look-up Table** выполняет линейную интерполяцию входного сигнала в соответствии с задаваемой табличной функцией.

Блок **Look-up Table(2D)** производит линейную интерполяцию двумерного входного сигнала.

Блок **Hit Crossing** (Обнаружить пересечение) позволяет зафиксировать состояние, когда входной сигнал пересекает некоторое значение. При возникновении такой ситуации на выходе блока формируется единичный сигнал. Блок имеет 3 параметра настройки:

- **Hit crossing offset** — определяет значение, пересечение с которым необходимо идентифицировать;
- **Hit crossing direction** — позволяет указать направление, при котором это пересечение должно обнаруживаться; значение этого параметра выбирается с помощью списка, который содержит три альтернативы: rising (возрастание), falling (убывание), either (в любом направлении);
- **Show output port** (Указать порт выхода) — опция, с помощью которой выбирается формат использования блока.

При одновременном выполнении условий, задаваемых параметрами **Hit crossing offset** и **Hit crossing direction**, на выходе блока формируется единичный сигнал. Его длительность определяется значением дискрета времени (параметр **Sample time**) блока, предшествующего в модели блоку **Hit crossing**. Если этот параметр отсутствует, то единичный сигнал на выходе блока существует до его следующего срабатывания.

Блок **Relay** (Реле) работает по аналогии с обычным реле: если входной сигнал превышает некоторое пороговое значение, то на выходе блока формируется разрешающий сигнал. Блок имеет 4 параметра настройки:

- **Switch on point** (Точка включения) — задает пороговое значение, при превышении которого происходит включение реле;
- **Switch off point** (Точка выключения) — определяет уровень входного сигнала, при котором реле выключается;
- **Output when on** (Выход при включенном состоянии) — устанавливает уровень выходной величины при включенном реле;
- **Output when off** (Выход при выключенном состоянии) — определяет уровень выходного сигнала при выключенном реле.

В следующую группу можно объединить блоки, реализующие функцию задержки сигнала. Таких блоков три.

Блок **Memory** (Память) выполняет задержку сигнала только на один шаг модельного времени. Блок имеет два параметра настройки: **Initial condition** (Начальное условие) — задает значение входного сигнала в начальный момент времени; опция **Inherit sample time** (Наследование шага времени) позволяет выбрать величину шага,

на который будет производиться задержка сигнала. Если опция не задана, то используется минимальная задержка, равная 0,1 единицы модельного времени. Если опция установлена, то величина задержки равна значению дискрета времени блока, предшествующего блоку **Memory**.

Блок **Transport Delay** обеспечивает задержку сигнала на заданное количество шагов модельного времени, причем не обязательно целое. Настройка блока происходит по трем параметрам:

- **Time delay** (Время задержки) — количество шагов модельного времени, на которое задерживается сигнал; может вводиться либо в числовой форме, либо в форме вычисляемого выражения;
- **Initial input** (Начальное значение входа) — по умолчанию равно 0;
- **Initial buffer size** (Начальный размер буфера) — объем памяти (в байтах), выделяемой в рабочем пространстве MatLAB для хранения параметров задержанного сигнала; должно быть кратно 8 (по умолчанию — 1024).

Блок **Variable Transport Delay** позволяет задавать управляемую извне величину задержки. С этой целью блок имеет дополнительный вход. Подаваемый на него сигнал определяет длительность задержки.

В заключение рассмотрим группу блоков-переключателей, управляющих направлением передачи сигнала. Таких блока три: **Switch** (Переключатель), **Manual Switch** (Ручной переключатель) и **Multiport Switch** (Многовходовый переключатель).

Блок **Switch** имеет три входа: два (1-й и 3-й) информационных и один (2-й) — управляющий и один выход. Если величина управляющего сигнала, поступающего на 2-й вход, не меньше некоторого заданного порогового значения (параметр **Threshold** — порог), то на выход блока передается сигнал с 1-го входа, в противном случае — сигнал с 3-го входа.

Блок **Manual Switch** не имеет параметров настройки. У него два входа и один выход. На изображении блока показано перемычкой, какой именно из двух входов подключен к выходу. Блок позволяет вручную переключить входы. Для этого необходимо дважды щелкнуть мышью на изображении блока. При этом изменится изображение блока — на нем выход уже будет соединен перемычкой с другим входом.

Блок **Multiport Switch** имеет не менее трех входов. Первый сверху является управляющим, остальные — информационными. Блок имеет один параметр настройки **Number of inputs** (Число входов), который устанавливает число *информационных* входов. Номер входа, который соединяется с выходом, равен значению управляющего сигнала, поступающего на верхний вход. Если это значение является дробным числом, то округляется до целого по обычным правилам, если меньше единицы, считается равным 1, а если больше числа информационных входов, то принимается равным наибольшему номеру (входы нумеруются сверху вниз, исключая самый верхний — управляющий).

7.1.7. *Раздел Discrete*

Ранее рассмотренные разделы библиотеки позволяют формировать непрерывную динамическую систему. Раздел **Discrete** (рис. 7.38) содержит элементы (блоки), присущие только дискретным системам, а также превращающие непрерывную систему в дискретную:

- **Unit Delay** — блок задержки сигнала;
- **Discrete-Time Integrator** — дискретный интегратор;
- **Zero-Order Hold** — экстраполятор нулевого порядка;
- **First-Order Hold** — экстраполятор первого порядка;
- **Discrete State-Space** — блок задания дискретного звена матрицами его состояния;
- **Discrete Filter** — блок задания дискретного звена через дискретную передаточную дробно-рациональную функцию относительно 1/z;
- **Discrete Transfer Fcn** — блок задания дискретного звена через дискретную передаточную дробно-рациональную функцию относительно z;
- **Discrete Zero-Pole** — блок задания дискретного звена через указание значений нулей и полюсов дискретной передаточной функции относительно 1/z.

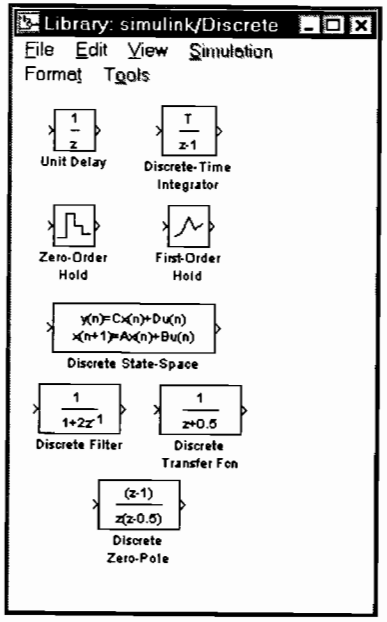

Рис. 7.38

Блок **Unit Delay** обеспечивает задержку входного сигнала на заданное число шагов модельного времени. Параметрами настройки блока являются: **Initial condition** (Начальное значение сигнала) и **Sample time** (Время задержки), задаваемое числом шагов модельного времени.

Блок **Discrete-Time Integrator** выполняет численное интегрирование входного сигнала. Большинство параметров настройки этого блока совпадают с параметрами блока **Integrator** раздела **Linear**. Отличия состоят в следующем. В блоке дискретного интегратора есть дополнительный параметр — метод численного интегрирования (**Integrator method**). С помощью списка можно выбрать один из трех методов:

- прямой метод Эйлера (левых прямоугольников);
- обратный метод Эйлера (правых прямоугольников);
- метод трапеций.

Второе отличие — вместо параметра **Absolute tolerance** введен параметр **Sample time**, который задает шаг интегрирования в единицах шагов модельного времени.

7.1.8. *Раздел Connections*

Раздел **Connections** (Соединения) предназначен для построения сложных S-моделей, состоящих из других моделей более низкого уровня. Состав блоков приведен на рис. 7.39.

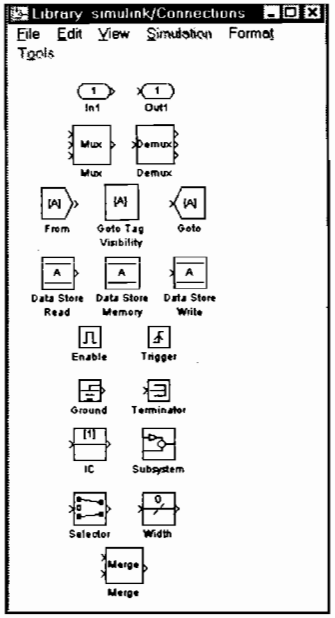

Рис. 7.39

Блоки **In** (Входной порт) и **Out** (Выходной порт) обеспечивают связь между подсистемами модели по информации.

Блок **Mux** (Концентратор) выполняет объединение входных величин в один выходной вектор. При этом входные величины могут быть как скалярными, так и векторными. Длина результирующего вектора равна сумме длин всех суммируемых векторов. Порядок элементов в векторе выхода определяется порядком входов (сверху вниз) и порядком расположения элементов внутри каждого входа. Блок имеет один параметр настройки — **Number of inputs** (Число входов).

Блок **Demux** (Разделитель) выполняет обратную функцию — *разделяет входной вектор на заданное число компонентов*. Он также имеет единственный параметр настройки **Number of outputs** (Число выходов). В случае, когда указанное число выходов (N) оказывается меньшим длины входного вектора (M), блок формирует выходные векторы следующим образом. Первые N-1 выходов будут векторами одинаковой длины, равной целой части отношения M/(N-1). Последний выход будет иметь длину, равную остатку от деления.

Блоки **From** (Принять), **Goto Tag Visibility** (Признак видимости) и **Goto** (Передать) используются совместно и предназначены для обмена данными между различными частями S-модели с учетом доступности (видимости) этих данных.

Блоки **Data Store Read** (Чтение данных), **Data Store Memory** (Запоминание данных) и **Data Store Write** (Запись данных) также используются совместно и обеспечивают не только передачу данных, но и их хранение на интервале моделирования.

Блоки **Enable** (Разрешить) и **Trigger** (Защелка) предназначены для логического управления работой подсистем S-модели.

Блоки **Ground** (Земля) и **Terminator** (Ограничитель) могут использоваться в качестве заглушек для тех портов, которые по какой-либо причине оказались не подключенными к другим блокам S-модели. При этом блок **Ground** используется как заглушка для входных портов, а **Terminator** — для выходных портов.

Блок **IC** (Initial Condition — начальное условие) позволяет установить произвольное начальное значение входного сигнала.

Блок **Subsystem** (Подсистема) представляет собой заготовку для создания подсистемы.

Примечание

Подсистема — это достаточно самостоятельная S-модель более низкого уровня, которая, в свою очередь, может содержать подсистемы с произвольным уровнем вложенности.

Блок **Selector** (Селектор) выбирает во входном векторе и передает на выход только те элементы, номера которых указаны в одном из параметров настройки блока. Существенным достоинством блока является то, что значения параметров его настройки отображаются в графической форме на изображении блока.

Блок **Width** (Размер) определяет размерность сигнала, поступающего на его вход. Значение размерности выводится непосредственно на изображении блока. Параметров настройки блок не имеет.

Блок **Merge** (Слияние) выполняет объединение поступающих на его входы сигналов в один.

7.2. Построение блок-схем

Рассмотрим, какие операции выполняют в процессе построения блок-схем сложных динамических систем.

7.2.1. Выделение объектов

При создании и редактировании S-модели нужно выполнять такие операции, как копирование или удаление блоков и линий. Для этого предварительно необходимо выделить один или несколько блоков и линий (объектов).

Чтобы *выделить отдельный объект*, нужно щелкнуть на нем мышью один раз. В результате по углам выделенного блока или в начале и конце линии появятся маленькие черные метки. При этом все другие ранее выделенные объекты станут невыделенными. Если щелкнуть на объекте второй раз, он станет невыделенным.

Выделение нескольких объектов по одному осуществляется следующим образом. Необходимо нажать клавишу [Shift] и, удерживая ее нажатой, выполнить щелчки на каждом выделяемом объекте. Затем клавишу [Shift] отпустить.

Группу объектов можно выделить также с помощью рамки. Делается это так. Необходимо установить курсор мыши в точку, которая будет являться начальной точкой рамки, нажать левую кнопку мыши и, не отпуская ее, переместить мышь в направлении диагонали прямоугольника. В результате на экране появится прямоугольная рамка и, когда вы отпустите кнопку мыши, все объекты и линии, охваченные рамкой, будут выделены.

Если требуется *выделить всю модель*, т.е. все объекты в активном окне блок-схемы, то для этого следует воспользоваться командой **Select All** (Выбрать все) из меню **Edit** (Правка) или комбинацией клавиш [Ctrl+A].

7.2.2. Операции с блоками

Копирование блоков из одного окна в другое

Можно копировать блоки из библиотеки или другой модели в текущую модель. Для этого достаточно открыть нужную библиотеку или окно модели-прототипа и перетащить мышью нужный блок в окно создаваемой (редактируемой) модели.

Блоки можно копировать и при помощи команд меню. Последовательность действий при этом такова:

1. В окне библиотеки или модели выделите блок/блоки, подлежащие копированию.
2. Выберите в меню **Edit** (Правка) активного окна команду **Copy** (Копировать).
3. Сделайте активным окно, в которое нужно скопировать блок, и выберите в нем команду **Paste** (Вставить) из меню **Edit** (Правка).

Каждому скопированному блоку SimuLink присваивает имя. Первый скопированный блок будет иметь то же имя, что и блок в библиотеке. Каждый следующий блок того же типа будет иметь такое же имя с добавлением порядкового номера. Пользователь может переименовать блок (см. далее). При копировании блок получает те же значения настраиваемых параметров, что и блок-оригинал.

Перестановка блоков в модели

Перестановка блока внутри модели осуществляется путем перетаскивания его мышью. При этом SimuLink автоматически перерисовывает линии, связывающие этот блок с другими блоками.

Чтобы переставить несколько блоков вместе с соединительными линиями и с сохранением относительных расстояний, необходимо их выделить и перетащить мышью один из блоков. Все другие выделенные блоки также займут новые места.

Копирование блоков одной модели

Копирование блоков одной модели можно выполнить двумя способами:

1. Перетащить блок в нужное положение, удерживая при этом клавишу [Ctrl].
2. Перетащить блок, удерживая нажатой правую кнопку мыши.

На рис. 7.40 представлен результат копирования блоков **Scope** и **XY Graph**.

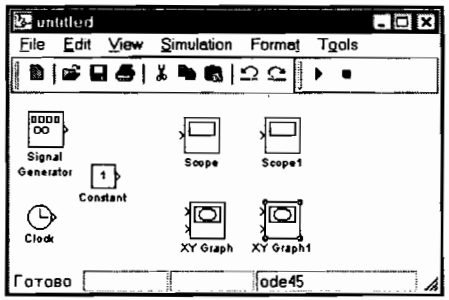

Рис. 7.40

Установка параметров блока

Функции, которые выполняет блок, зависят от значений параметров блока. Установка этих значений осуществляется в окне настройки блока, которое вызывается после двойного щелчка на изображении блока в блок-схеме.

Удаление блоков

Для удаления ненужных блоков из блок-схемы достаточно выделить эти блоки так, как было указано ранее, и нажать клавишу [Del] или [Backspace]. Можно также вызвать команду **Clear** (Очистить) или **Cut** (Вырезать) из меню **Edit** (Правка) окна блок-схемы. Если использована команда **Cut**, то в дальнейшем удаленные блоки можно скопировать в модель при помощи команды **Paste** (Вставить) того же меню.

Отсоединение блока

Чтобы отсоединить блок от линий, достаточно нажать клавишу [Shift] и, не отпуская ее, перетащить блок в другое место.

Изменение угловой ориентации блока

В начальном состоянии сигнал проходит через блок слева направо (по левую сторону располагаются входы блока, а по правую сторону — выходы). Чтобы изменить угловую ориентацию блока следует:

1. Выделить блок, который нужно повернуть.
2. Выбрать в меню **Format** (Формат) окна блок-схемы одну из следующих команд — **Flip Block** (Поворот блока на 180 градусов) или **Rotate Block** (Поворот блока по часовой стрелке на 90 градусов).

Изменение размеров блока

Изменение размеров блока выполняется следующим образом. Выделите блок и установите указатель мыши на одну из угловых меток блока. Форма указателя при этом изменится — он примет вид двунаправленной стрелки. "Захватите" мышью эту метку и перетяните ее в новое положение.

Изменение и перемещение имени блока

Все имена блоков в модели должны быть уникальными и состоять хотя бы из одного символа. Чтобы изменить имя блока, нужно выполнить щелчок на имени, а затем, используя обычные приемы редактирования, внести необходимые изменения.

Для изменения шрифта следует выделить блок, вызвать команду **Font** (Шрифт) из меню **Format** (Формат) окна модели и затем выбрать шрифт в открывшемся диалоговом окне.

По умолчанию имя блока располагается следующим образом. Если блок ориентирован слева направо, то имя находится под блоком, если справа налево — над блоком, если же сверху вниз или снизу вверх — по правой стороне блока. Изменить местоположение имени выделенного блока можно двумя способами:

- перетащить имя мышью на противоположную сторону блока;
- воспользоваться командой **Flip Name** из меню **Format** окна модели — она также переносит имя на противоположную сторону блока.

Скрыть имя блока можно, используя команду **Hide Name** (Скрыть имя) меню **Format** (Формат) окна модели. Чтобы восстановить отображение имени, следует воспользоваться командой **Show Name** (Показать имя) того же меню.

7.2.3. Создание соединительных линий

Сигналы в модели передаются по линиям. Каждая линия может передавать или скалярный, или векторный сигнал. Линия соединяет выходной порт одного блока с входным портом другого блока. Линия может также разветвляться и соединять выходной порт одного блока с входными портами нескольких блоков.

Создание линии между блоками

Чтобы соединить выходной порт одного блока с входным портом другого, нужно выполнить следующие действия:

1. Установить указатель мыши на выходной порт первого блока (при этом курсор должен принять форму перекрестия).
2. Нажать левую кнопку мыши и, удерживая ее в этом положении, передвинуть указатель ко входному порту второго блока;
3. Отпустить кнопку мыши.

SimuLink заменит символы портов соединительной линией с указанием направления передачи сигнала. Именно таким образом на рис. 7.41 выход блока **Sine Wave** соединен с входом блока **XY Graph**.

Линии можно рисовать как от входного порта к выходному, так и наоборот.

Примечание

По умолчанию SimuLink рисует соединительные линии, состоящие из горизонтальных и вертикальных сегментов. Чтобы построить линию под углом 45 градусов, необходимо в процессе рисования удерживать нажатой клавишу [Shift].

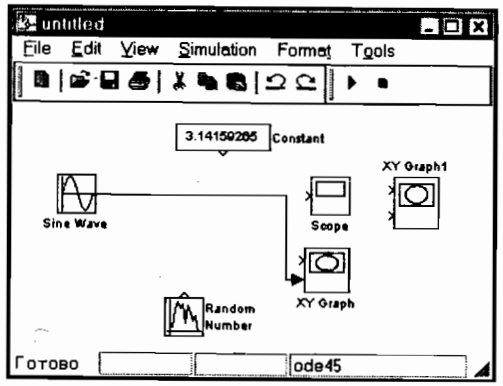

Рис. 7.41

Создание разветвления линии

Линия, которая ответвляется, начинается с существующей и передает ее сигнал к входному порту другого блока. Как существующая, так и ответвленная линии передают один сигнал. Разветвленная линия дает возможность передать один и тот же сигнал к нескольким блокам.

Чтобы образовать ответвление от существующей линии, необходимо выполнить следующие действия:

1. Установить курсор в точку ответвления.
2. Нажать клавишу [Ctrl] и левую кнопку мыши и удерживать их нажатыми.
3. Провести линию ко входному порту нужного блока; отпустить клавишу [Ctrl] и левую кнопку мыши (рис. 7.42).

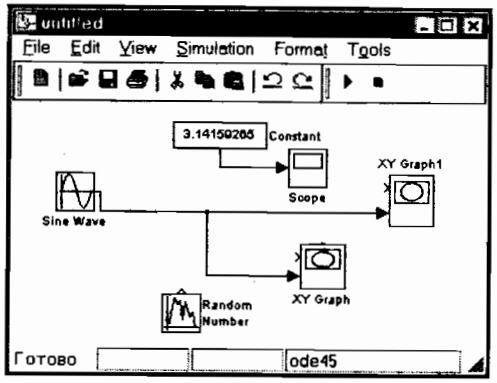

Рис. 7.42

Создание сегмента линии

Блоки можно соединять ломаными линиями, состоящими из нескольких сегментов. Для создания следующего сегмента необходимо установить курсор в конец предыдущего сегмента и нарисовать (удерживая левую клавишу мыши) следующий сегмент. Таким образом, например, соединены на рис. 7.43 блоки **Clock** и **XY Graph** и блоки **Random Number** и **XY Graph1**.

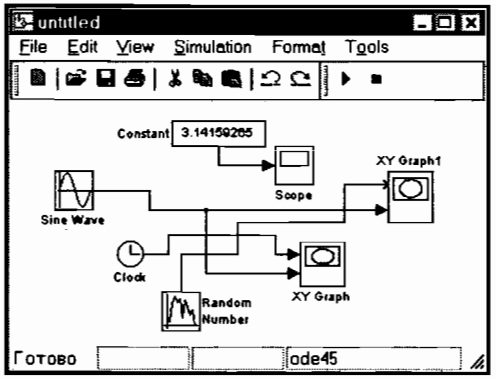

Рис. 7.43

Перемещение сегмента линии

Чтобы переместить отдельный сегмент линии, необходимо выполнить следующие действия:

1. Установить указатель на перемещаемом сегменте.
2. Нажать и удерживать левую кнопку мыши. Курсор при этом должен принять форму креста.
3. Переместить указатель в новое положение сегмента и отпустить кнопку мыши.

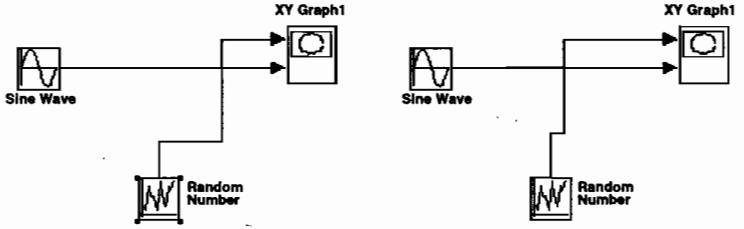

Рис. 7.44

На рис. 7.44 показан результат перемещения вертикального сегмента линии, соединяющей блоки **Random Number** и **XY Graph1**.

Примечание

Сегмент, непосредственно прилегающий к порту блока, переместить невозможно.

Деление линии на сегменты

При необходимости линию можно разделить на два сегмента. Для этого следует выполнить следующие действия:

1. Выделить линию и установить курсор в точку излома линии.
2. Нажать клавишу [Shift] и левую кнопку мыши. Курсор при этом примет форму окружности, а на линии образуется излом.
3. Переместить курсор (излом) в новое положение.
4. Отпустить клавишу [Shift] и кнопку мыши.

Результат этих действий представлен на рис. 7.45, где линия, соединяющая блоки **Sine Wave** и **XY Graph1**, разделена на два сегмента.

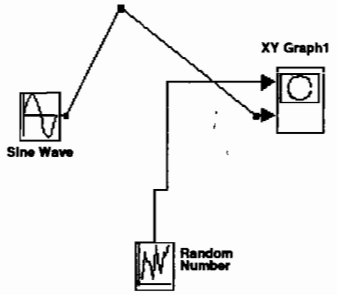

Рис. 7.45

Перемещение излома линии

Чтобы переместить излом линии, достаточно перетянуть мышью точку излома в новое положение.

7.2.4. Проставление меток сигналов и комментариев

Для наглядности и удобства блок-схемы линии можно снабдить метками, указывающими, какие сигналы по ним проходят. Метки размещают под или над горизонтальной линией, по левую или по правую сторону от вертикальной линии. Метка может быть расположена в начале, в конце или посередине линии.

Создание и манипулирование метками сигналов

Чтобы создать метку сигнала, нужно дважды щелкнуть на сегменте линии и затем ввести текст метки (рис. 7.46). Причем

двойной щелчок следует выполнять точно на линии, так как в противном случае будет создан комментарий к модели.

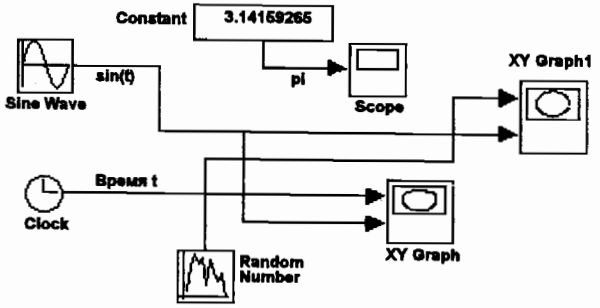

Рис. 7.46

Перемещение метки осуществляется путем перетаскивания ее мышью на новое место. Если при этом удерживать нажатой клавишу [Ctrl], то метка будет скопирована в новое место. Скопировать метку можно также, выполнив двойной щелчок на другом сегменте линии.

Чтобы отредактировать метку, следует щелкнуть на ней и затем внести соответствующие изменения в ее текст.

Чтобы удалить метку, выделите ее, удерживая клавишу [Shift], и нажмите клавишу [Del] или [Backspace]. При этом будут уделены все метки этой линии.

Распространение меток линии

Распространение меток линии — это процесс автоматического переноса метки/меток к сегментам линии, разорванной блоками **From/Goto**, **Mux** (рис. 7.47).

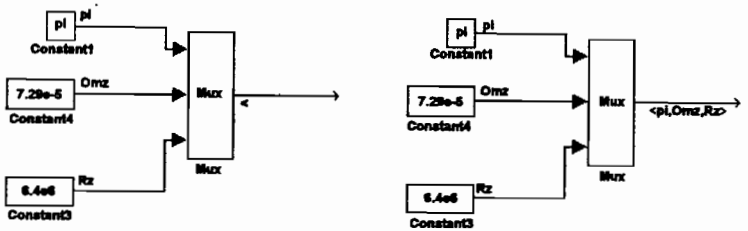

Рис. 7.47

Чтобы распространить метку, создайте во втором и последующих сегментах линии метки с именем "<" и вызовите команду

Update Diagram (Обновить схему) из меню **Edit** (Правка) или нажмите комбинацию клавиш [Ctrl+D]. В результате метки в этих сегментах будут проставлены автоматически (рис. 7.47).

Создание и манипулирование комментарием

Комментарии дают возможность сопровождать блок-схемы текстовой информацией о модели и отдельных ее составляющих. Комментарии можно проставлять в любом свободном месте блок-схемы.

После двойного щелчка в любом свободном месте блок-схемы появляется прямоугольная рамка, в которую можно ввести текст комментария.

Перемещение комментария осуществляется путем перетаскивания его с помощью мыши.

Если при этом удерживать нажатой клавишу [Ctrl], комментарий будет скопирован в новое место.

Созданный комментарий можно отредактировать. Для этого нужно выполнить на нем щелчок, а потом внести соответствующие изменения.

Чтобы изменить при этом параметры шрифта комментария, необходимо выделить текст комментария и выбрать команду **Font** (Шрифт) из меню **Format** (Формат) окна блок-схемы. После этого появится диалоговое окно, в котором следует выбрать название шрифта, его размер, атрибуты и стиль и нажать кнопку **OK**.

Комментарий можно удалить. Для этого выделите комментарий, удерживая клавишу [Shift], и нажмите клавишу [Del] или [Backspace].

7.2.5. Создание подсистем

Если блок-схемы модели слишком сложная и имеет большие размеры, ее можно упростить, группируя блоки в подсистемы. Использование подсистем дает следующие преимущества:

- сокращается количество блоков, которые выводятся в окне модели;
- появляется возможность объединить в одну группу (подсистему) функционально связанные блоки;
- появляется возможность создания иерархических блок-схем.

Подсистему можно создать двумя способами:

- добавить блок **Subsystem** в модель, потом войти в этот блок и создать подсистему в появившемся окне подсистемы;
- выделить часть блок-схемы модели и объединить ее в подсистему.

Создание подсистемы путем добавления блока Subsystem

В этом случае нужно поступить так:

1. Скопировать блок **Subsystem** в окно модели, перетянув его из раздела **Connections**.

2. Открыть окно блока **Subsystem**, дважды щелкнув на изображении блока в блок-схеме.
3. В пустом окне модели создать подсистему, используя блоки **In** и **Out** для создания входов и выходов подсистемы.

Создание подсистемы путем группировки существующих блоков

Если блок-схема уже содержит блоки, которые нужно объединить в подсистему, то последнюю можно создать так:

1. Выделить при помощи рамки блоки и соединяющие их линии, которые нужно включить в состав подсистемы (рис. 7.48).
2. Выбрать команду **Create Subsystem** (Создать подсистему) из меню **Edit** (Правка). В результате SimuLink заменит выделенные блоки одним блоком **Subsystem** (рис. 7.48).

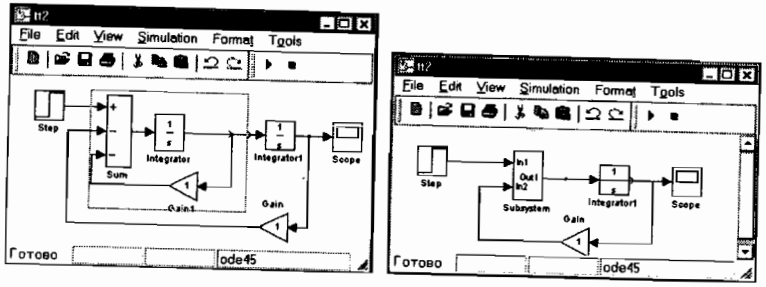

Рис. 7.48

Чтобы увидеть блок-схему созданной подсистемы (рис. 7.49), следует дважды щелкнуть на блоке **Subsystem**. Как видно из рисунка, SimuLink добавил в блок-схему блоки **In** и **Out**, представляющие входы и выходы в систему высшего уровня.

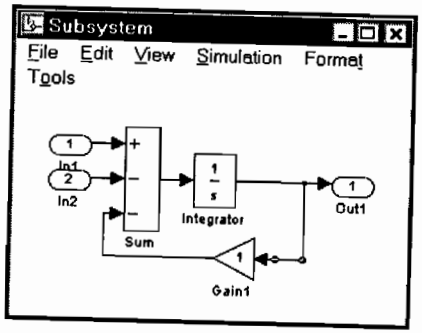

Рис. 7.49

7.2.6. Запись и печать S-модели

Для записи модели (блок-схемы) на диск нужно вызвать команду **Save** (Сохранить) или **Save As** (Сохранить как) из меню **File** (Файл) окна модели. При этом SimuLink записывает в указанную вами папку файл с заданным (введенным с клавиатуры) именем, присваивая ему расширение .mdl.

Чтобы распечатать модель (блок-схему), следует воспользоваться командой **Print** (Печать) из меню **File** (Файл) окна модели.

Блок-схему можно вставить в документ любого текстового редактора, например Word. Для этого следует сначала вызвать команду **Copy Model** (Копировать модель) из меню **Edit** (Правка) окна модели, а затем перейти в окно текстового редактора и нажать комбинацию клавиш [Shift+Ins]. Именно таким способом были получены рис. 7.44—7.47.

7.3. Примеры моделирования

7.3.1. Моделирование поведения физического маятника

Рассмотрим прежнюю задачу моделирования поведения физического маятника при гармонической вибрации точки его опоры.

Пользуясь результатами ранее проведенных преобразований (разд. 2.6.2), исходное уравнение движения маятника примем в такой безразмерной форме:

$$\varphi'' + \sin\varphi = S(\tau,\varphi,\varphi') \qquad (7.1)$$

где обозначено

$$S(\tau,\varphi,\varphi') = -2\cdot\varsigma\cdot\varphi' - \\ -[n_{mx}\cdot\sin(\nu\cdot\tau+\varepsilon_x)\cdot\cos\varphi + n_{my}\cdot\sin(\nu\cdot\tau+\varepsilon_y)\cdot\sin\varphi] \qquad (7.2)$$

причем безразмерные величины ζ и ν определяются выражениями:

$$\varsigma = \frac{R}{2\cdot\sqrt{mgl\cdot J}};\quad \nu = \frac{\omega}{\omega_0};\quad \omega_0 = \sqrt{\frac{mgl}{J}}$$

Задаваемыми (исходными) параметрами для моделирования будем полагать:

1. Параметры самого маятника; к ним в рассматриваемом случае относится только относительный коэффициент затухания ζ.
2. Параметры, характеризующие внешнее воздействие; сюда входят:
   — амплитуды виброперегрузок в вертикальном n_{my} и горизонтальном n_{mx} направлениях;
   — относительная (по отношению к частоте собственных колебаний маятника) частота вибрации точки опоры ν;
   — начальные фазы ε_y и ε_x вибрации точки опоры.

3. Начальные условия движения маятника:
   - начальное отклонение φ_0 от вертикали;
   - начальная безразмерная угловая скорость маятника $\varphi'_0 = \dot{\varphi}_0 / \omega_0$.

К выходным (моделируемым) величинам будем относить текущий угол отклонения маятника от вертикали $\varphi(\tau)$ и его безразмерную угловую скорость $\varphi'(\tau)$.

Запишем уравнение (7.1) несколько в иной форме:

$$\varphi'' = S(\tau, \varphi, \varphi') - \sin\varphi \qquad (7.3)$$

В основу воплощения этого уравнения в блок-схему положим следующую идею. Если сформировать правую часть уравнения по процессам $\varphi(\tau)$ и $S(\tau,\varphi,\varphi')$, считая их известными, то нам станет известно угловое ускорение $\varphi''(\tau)$. Проинтегрировав ускорение, можно получить угловую скорость. Наконец, проинтегрировав и ее, мы получим закон изменения угла от времени $\varphi(\tau)$. Последние полученные две величины (процессы) можно теперь использовать для формирования правой части уравнения (7.3).

Итак, для формирования блок-схемы, осуществляющей численное интегрирование уравнения (7.1), можно сделать следующее:

1. В основу блок-схемы положить два последовательно соединенных интегратора (блоки **Integrator**) с внешне (external) задаваемыми начальными условиями; на вход первого интегратора подать угловое ускорение, а в качестве начального условия использовать начальное значение угловой скорости $\varphi'(0)$; выходом этого блока будет текущая угловая скорость $\varphi'(\tau)$; эту величину следует подать на вход второго интегратора с начальным условием в виде начального значения угла $\varphi(0)$; выход этого блока будет представлять собой искомый процесс $\varphi(\tau)$.
2. Сформировать отдельным блоком в виде подмодели (блок **Subsystem**) функцию $S(\tau,\varphi,\varphi')$, используя в качестве входных процессов полученные процессы $\varphi(\tau)$ и $\varphi'(\tau)$, а в качестве входных величин — параметры $2\cdot\varsigma$, n_{mx}, ε_x, n_{my}, ε_y; один из вариантов воплощения такой подмодели в соответствии с выражением (7.2) представлен на рис. 7.50.
3. Используя созданную подмодель, в основной модели связать сформированные процессы $\varphi(\tau)$ и $\varphi'(\tau)$ с соответствующими входами подмодели, а выход подмодели связать с сумматором, на который подать также предварительно сформированный сигнал $\sin(\varphi)$; сигнал с выхода сумматора подать на вход первого интегратора, замыкая цепь интегрирования.

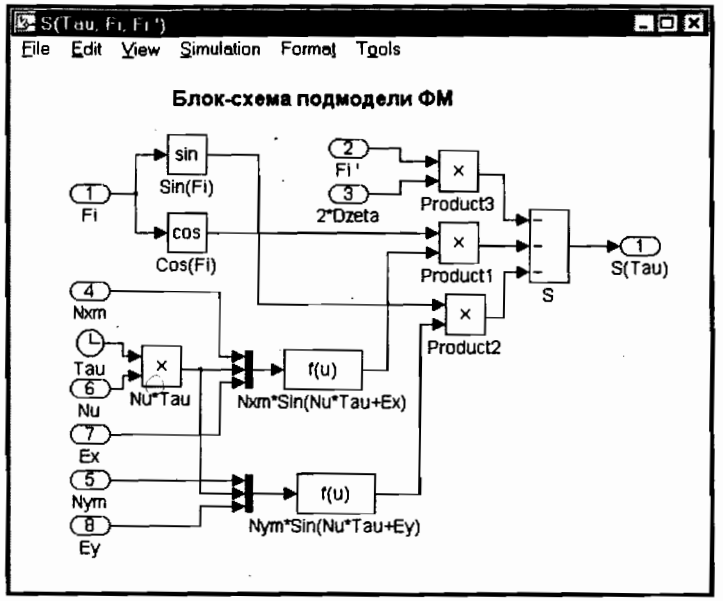

Рис. 7.50

4. Для отображения результатов интегрирования в графической форме подсоединить блок **Scope** к выходу системы — сформированному сигналу $\varphi(\tau)$; для отображения фазового портрета маятника использовать блок **XY Graph**, на входы которого направить сигналы $\varphi(\tau)$ и $\varphi'(\tau)$.

Блок-схема, реализующая указанные идеи приведена на рис. 7.51.

Следует обратить внимание на то, как схемно реализуется заданные колебания точки подвеса. Для этого используется генератор времени — блок **Clock** (рис. 7.50). Выход этого блока перемножается с заданным значением частоты ν. Полученная величина $\nu\tau$ с постоянными величинами n_{mx}, ε_x поступает на блок типа **Mux** (на схеме он имеет вид вертикальной жирной черты), в результате чего на выходе его образуется вектор из трех элементов n_{mx}, $\nu\tau$ и ε_x. Этот вектор подается на вход блока **f(u)** (рис. 7.50), который создан на основе стандартного блока **Fcn** и реализует следующую зависимость:

$$f(u) = u(1) * \sin(u(2) + u(3))$$

где $u(1)$, $u(2)$, $u(3)$ — элементы вектора u, подаваемого на вход блока. Таких блоков в подсистеме на рис. 7.50 два. В первом из

них в качестве первого элемента входного вектора используется величина n_{mx}, второго — $\nu\tau$ и третьего — ε_x. Во втором блоке первым элементом является n_{my}, вторым — $\nu\tau$ и третьим — ε_y.

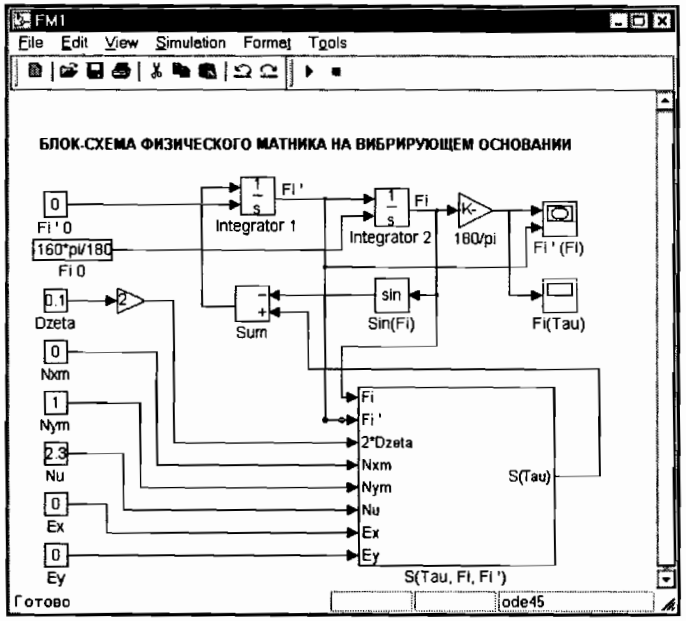

Рис. 7.51

Отметим также, что при построении блок-схемы подсистемы (рис. 7.50) связь подсистемы с основной системой осуществляется путем ввода в подсистему стандартных блоков типа **In** (Вход) и **Out** (Выход). Все величины, которые формируются в основной модели, а затем должны быть использованы в подмодели, должны "проникать" в подсистему через блоки **In**, а величины, сформированные в подсистеме и затем используемые в основной системе, должны "выходить" из подсистемы через блоки **Out**.

При этом на изображении блока подсистемы в блок-схеме основной модели автоматически появляется такое количество входов, которое совпадает с числом введенных в подсистеме блоков **In**, и выходов, равных числу блоков **Out**, использованных в подсистеме.

Вызвав команду **Start** из меню **Simulation** окна основной блок-схемы, можно активизировать процесс моделирования созданной S-модели маятника. По окончании этого процесса появится дополнительное графическое окно, созданное блоком **XY Graph** (рис. 7.52),

на котором изображен фазовый портрет маятника при выбранных параметрах маятника и возмущений.

Если теперь дважды щелкнуть мышью на блоке **Scope** в блок-схеме маятника, то появится еще одно графическое окно (рис. 7.53) с графиком зависимости угла от времени.

Графики, показанные на рис. 7.52 и 7.53, соответствуют значениям входных данных, указанным на схеме рис. 7.51, и иллюстрируют возникновение параметрических колебаний при вертикальной вибрации точки подвеса маятника.

Изменяя данные настройки входных блоков **Constant**, можно проводить исследования поведения маятника при произвольных значениях входных параметров.

Из приведенного видны преимущества и недостатки моделирования поведения динамических систем с помощью пакета SimuLink по сравнению с аналогичными исследованиями при помощи программ:

- сборка блок-схемы уравнений движения вместо набора текста процедуры правых частей более наглядна, позволяет контролировать правильность набора путем осознания физического смысла отдельных блоков и их взаимосвязей;
- при проведении моделирования в среде SimuLink отпадает необходимость в организации самого процесса численного интегрирования дифференциальных уравнений и даже вывода результатов в графической форме;
- однако форма вывода результатов в графической форме в SimuLink слишком жестка и мало изменяема: нельзя добавить собственные надписи в заголовок графика и по осям подвеса, нельзя установить сетку координатных линий; особенно неудобно то, что здесь не предусмотрены средства вывода текстовой дополнительной информации на поле графика, что делает графическое представление безадресным, мало что говорящим постороннему специалисту.

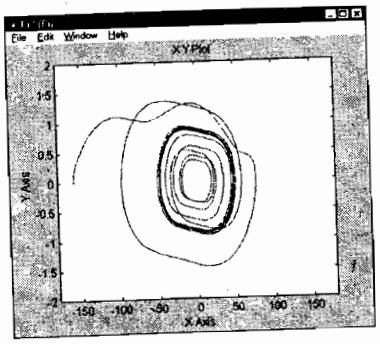

Рис. 7.52

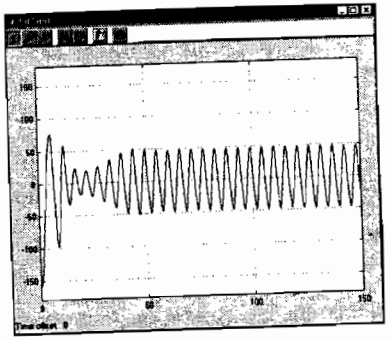

Рис. 7.53

Последний недостаток весьма существенен. Однако он может быть устранен имеющимися в пакете SimuLink средствами. Например, можно записать полученные значения выходных величин в MAT-файл (посылая их на блок **To File**), а затем создать и использовать *программу*, которая бы осуществляла считывание данных, записанных в MAT-файле, и формирование на этой основе графического изображения в окне фигуры по образцу, приведенному в разд. 2.5 и 2.7.

7.3.2. Моделирование поведения гироскопа в карданновом подвесе

Уравнения движения гироскопа в карданновом подвесе на неподвижном основании представим в виде:

$$\begin{cases} (J_1 + J_2\cos^2\beta)\ddot{\alpha} - 2J_2\dot{\alpha}\dot{\beta}\sin\beta\cos\beta + H\dot{\beta}\cos\beta = \\ = -f_2\dot{\alpha} + N_0 + N_m\sin(\omega t + \varepsilon_N) - [R_0 + R_m\sin(\omega t + \varepsilon_R)]\sin\beta, \\ J_3\ddot{\beta} + J_2\dot{\alpha}\sin\beta\cos\beta - H\dot{\alpha}\cos\beta = -f_2\dot{\beta} + L_0 + \\ + L_m\sin(\omega t + \varepsilon_L), \\ \dfrac{dH}{dt} = R_0 + R_m\sin(\omega t + \varepsilon_R) \end{cases}$$

Воплощение этой системы уравнений в блок-схему S-модели показано на рис. 7.54—7.57.

Рис. 7.54 представляет основную модель. Основная S-модель включает в себя следующие основные компоненты:

- блоки задания исходных данных; это совокупность блоков типа **Constant**, которые определяют значения постоянных величин, входящих в уравнения гироскопа;
- две основные подсистемы (построенные на основе стандартных блоков **Subsystem**): **ГКП** и **Моменты**;
- ряд блоков построения графических изображений выходных процессов; среди них 4 блока типа **Scope** и два блока типа **XY Graph**; назначение каждого из них становится ясным из рассмотрения блок-схемы;
- блок типа **To File**, который записывает полученные вычисленные значения углов α и β, а также угловых скоростей гироскопа $\dot{\alpha}$ и $\dot{\beta}$ и модельного времени t в файл GKP.mat.

Схема подсистемы **ГКП** представлена на рис. 7.55. Подсистема осуществляет основную функцию — представляет модель интегрирования исходной системы дифференциальных уравнений. Здесь размещены 5 интеграторов (ввиду того что принятая система дифференциальных уравнений имеет пятый порядок). Входы с 1 по 8, 12 и 13 представляют некоторые величины, не меняющиеся в процессе моделирования. Входы с 9 по 11 представляют собой моменты сил, действующие по осям подвеса гироскопа, значения которых формируются в подсистеме **Моменты**. Число выходов в

этой подсистеме — 4. Сюда входят два угла поворота гироскопа, две их производные по времени и текущее значение собственного кинетического момента гироскопа.

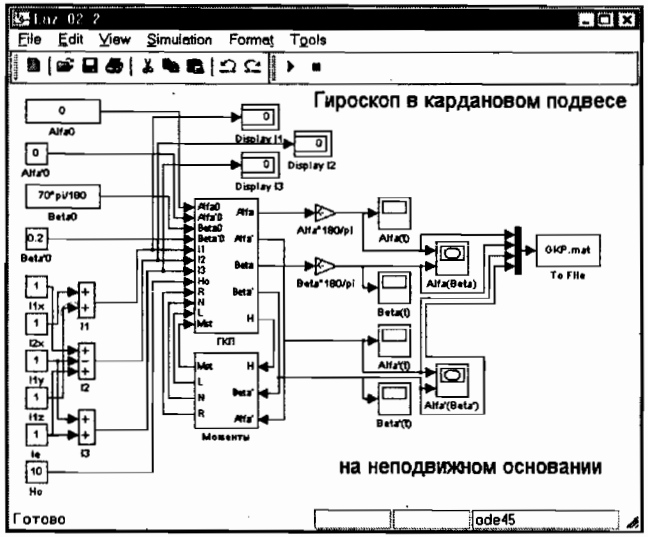

Рис. 7.54

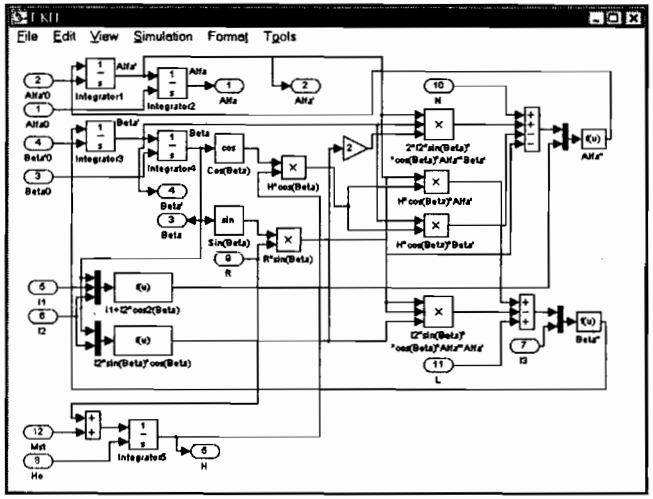

Рис. 7.55

Подсистема **Моменты** (рис. 7.56) использует данные об угловых скоростях поворота гироскопа и текущем значении кинетического момента для формирования моментов сил сопротивления (трения) по осям подвеса. В соответствии с этим блок **Моменты** имеет 3 входа. Кроме этого, внутри подсистемы производится установка 13 постоянных параметров, определяющих коэффициенты трения по осям и параметры гармонических колебаний моментов. Выходами подсистемы являются текущие значения моментов внешних сил по каждой из трех осей карданового подвеса. Формирование гармонически изменяющихся составляющих моментов сил происходит по однотипному правилу, воплощенному в трех полностью аналогичных подподсистемах, одна из которых представлена на рис. 7.57.

Как следует из схемы рис. 7.54, на вход оконечного файла GKP.mat подаются 4 входные величины: первая сверху — α, вторая — β, третья — угловая скорость гироскопа $\dot{\alpha}$ и четвертая — $\dot{\beta}$. Это значит, что в файл с указанным именем будет записываться матрица, состоящая из 5 строк. В первую строку будет записан массив значений модельного времени, для которых вычислены значения выходных (по отношению к S-модели) величин. В остальные строки будут записаны значения величин, являющихся входами блока **To File**, причем сверху вниз в порядке расположения самих входов этого блока (также сверху вниз).

Если файл GKP.mat сформирован (а это произойдет автоматически при запуске основной S-модели после вызова команды **Start** из меню **Simulation**), то теперь появляется возможность использовать записанные данные в специальном Script-файле для оформления графического окна с необходимым текстовым оформлением.

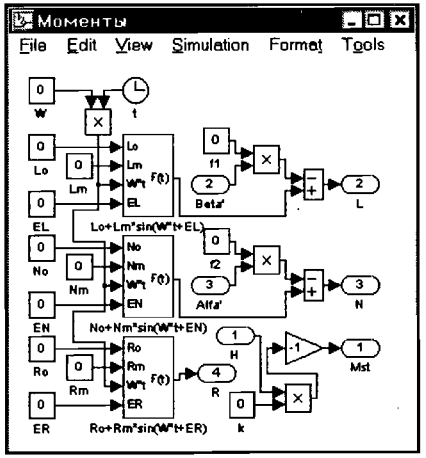

Рис. 7.56

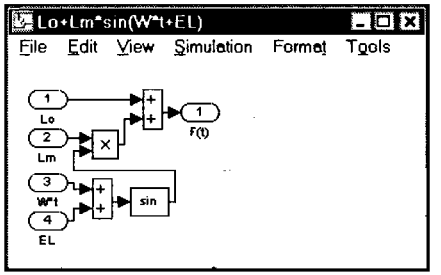

Рис. 7.57

Далее приведен вариант такого Script-файла. Запуская этот Script-файл, получим графическое окно, представленное на рис. 7.58.

Script-файл GKP_graf

```
load GKP; % Загрузка (чтение) файла GKP.mat
% (Происходит присвоение системной переменной ANS значений
% всей матрицы, содержащейся в файле GKP.mat.
% При этом модельное время занимает первую строку этой матрицы,
% а массивы входных величин - последующие строки матрицы
% в порядке указания входов в блок To File сверху вниз.)

% Теперь выделяем отдельные строки матрицы ANS и присваиваем им
% имена соответствующих переменных исходных уравнений.
tout=ans(1,:);
     Alfa=ans(2,:); Beta=ans(3,:);
     Alfat=ans(4,:); Betat=ans(5,:);

% Оформление графического подокна "alfa(t)"
subplot(4,4,[12 16]);
plot(tout,Alfa);grid;
title('Изменение угла альфа','FontSize',12);
xlabel('Время (сек)','FontSize',8);
ylabel('Альфа (градусы)','FontSize',8);

% Оформление графического подокна "Картинная плоскость"
subplot(4,6,[7 22]);
plot(Alfa,Beta);grid;
title('Картинная плоскость','FontSize',12);
xlabel('Альфа (градусы)','FontSize',8);
ylabel('Бета (градусы)','FontSize',8);

% Оформление текстового подокна "Модель"
subplot(4,4,1:4);
axis('off');
h1=text(0.25,1.1,'Свободное движение ГКП','FontSize',12);
h1=text(0.35,0.85,'по системе уравнений:','Fontsize',8);
h1=text(-0.1,0.6,'(J1+J2*cos^2(Beta))*Alfa''''-...
2*J2*Alfa''*Beta''*sin(Beta)*cos(Beta)+H0*Beta''...
*cos(Beta)= 0','Fontsize',10);
h1=text(-0.05,0.3,'J3*Beta''''+2*J2*(Alfa'')^2*sin(Beta)...
```

```
*cos(Beta)-H0*Alfa''*cos(Beta)= 0','Fontsize',10);

% Оформление текстового подокна "Параметры"
subplot(4,4,8);
axis('off');
h1=text(-0.1,1.4,'Параметры:','Fontsize',12);
h1=text(-0.45,1.2,'Гироскопа:','Fontsize',8);
h1=text(0.35,1.2,'Начальные условия:','Fontsize',8);
h1=text(-0.3,0.9,'H0=10','Fontsize',8);
h1=text(-0.3,0.7,'I1=2','Fontsize',8);
h1=text(-0.3,0.5,'I2=1','Fontsize',8);
h1=text(-0.3,0.3,'I3=2','Fontsize',8);
h1=text(0.6,0.9,'Alfa0=0','Fontsize',8);
h1=text(0.6,0.7,'Beta0=70','Fontsize',8);
h1=text(0.6,0.5,'Alfa0''=0','Fontsize',8);
h1=text(0.6,0.3,'Beta0''=0.2','Fontsize',8);
```

Результаты, представленные на рис. 7.58, иллюстрируют нелинейное свойство гироскопа в карданном подвесе — наличие даже в отсутствие моментов сил, действующих на гироскоп по осям его подвеса (гироскоп является "свободным"), систематического нарастания (с постоянной угловой скоростью) угла поворота оси гироскопа вокруг наружной оси карданового подвеса. Это так называемый уход Магнуса. Созданная S-модель гироскопа в карданном подвесе позволяет исследовать этот и другие нелинейные эффекты, возникающие в гироскопе на неподвижном основании.

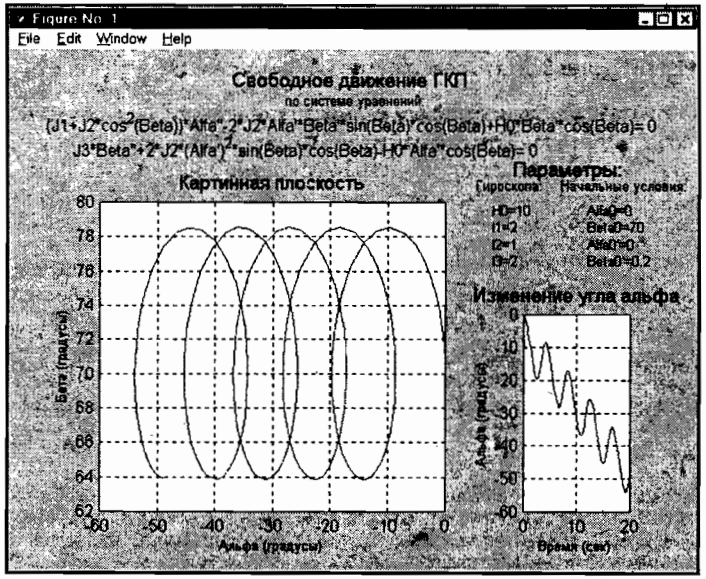

Рис. 7.58

Список литературы

1. Потемкин В.Г. Система MatLAB: Справ. пособие. — М.: ДИАЛОГ-МИФИ, 1997. — 350 с.
2. Потемкин В.Г. MatLAB 5 для студентов: Справ. пособие. — М.: ДИАЛОГ-МИФИ, 1998. — 314 с.
3. Потемкин В.Г., Рудаков П.И. MatLAB 5 для студентов. — 2-е изд., испр. и дополн. — М.: ДИАЛОГ-МИФИ, 1999. — 448 с.
4. Потемкин В.Г. Система инженерных и научных расчетов MatLAB 5.x: — В 2-х т. Том 1. — М.: ДИАЛОГ-МИФИ, 1999. — 366 с.
5. Потемкин В.Г. Система инженерных и научных расчетов MatLAB 5.x: — В 2-х т. Том 2. — М.: ДИАЛОГ-МИФИ, 1999. — 304 с.
6. Гультяев А.К. MatLAB 5.2. Имитационное моделирование в среде Windows: Практическое пособие. — СПб.: КОРОНА Принт, 1999. — 288 с.
7. Дьяконов В.П. Справочник по применению системы PC MatLAB. — М.: Физматлит, 1993. — 113 с.
8. Барановская Г.Г., Любченко И.Н. Микрокалькуляторы в курсе высшей математики: Практикум. — К.: Вища шк., 1987. — 288 с.
9. Сулима И.М., Гавриленко С.И., Радчик И.А., Юдицкий Я.А. Основные численные методы и их реализация на микрокалькуляторах. — К.: Вища шк., 1987. — 312 с.
10. Антонью А. Цифровые фильтры: анализ и проектирование. — М.: Радио и связь, 1983. — 320 с.
11. Лазарєв Ю.Ф. Початки програмування у середовищі MatLAB: Навч. посібник. — К.: "Корнійчук", 1999. — 160 с.
12. Краснопрошина А.А., Репникова Н.Б., Ильченко А.А. Современный анализ систем управления с применением MATLAB, Simulink, Control System: Учебное пособие. — К.: "Корнійчук", 1999. — 144 с.
13. Дьяконов В.П., Абраменкова И.В. MATLAB 5.0/5.3. Система символьной математики. — М.: Нолидж, 1999. — 640 с.
14. Медведев В.С., Потемкин В.Г. Control System Toolbox. MatLAB 5 для студентов. — М.: ДИАЛОГ-МИФИ, 1999. — 287 с.

Содержание