Юрий ЛАЗАРЕВ

Начала программирования в среде MatLAB

Учебное пособие
для студентов высших учебных заведений

Киев – НТУУ "КПИ" - 2003

УДК 681.3.06(075.8)
ББК 32.973.26-018.2 Я73
Л17

Лазарев Юрий Федорович
Л17 Начала программирования в среде MatLAB: Учебное пособие. - К.: НТУУ "КПИ", 2003. - 424 с.

Изложены основные особенности проведения вычислений в среде MatLAB как в режиме калькулятора, так и в программном режиме. Ознакомление с системой рассчитано на начинающего. Приведены сведения об основных командах, операторах, функциях и процедурах MatLAB. Изложение ведется таким образом, чтобы пользователь мог сразу применить полученные знания для проведения вычислений. Пособие содержит много примеров, которые поясняют и иллюстрируют работу по использованию процедур. Рассмотрена работа с некоторыми наиболее важными для инженеров пакетами прикладных программ MatLAB. (Signal Toolbox, Control и SimuLink).

Для студентов высших технических учебных заведений. Может быть полезно научным работникам и инженерам для начального ознакомления с системой MatLAB и приобретения навыков работы с ней.

Табл. 8. Илл. 283. Библиогр. 18 назв.

© Ю. Ф. Лазарев, 2003

Содержание

Предисловие 6
Вступление 8
1. MatLAB как научный калькулятор ***11***
1.1. Командное окно ***11***
1.2. Операции с числами ***12***
1.2.1. Ввод действительных чисел 12
1.2.2. Простейшие арифметические действия 14
1.2.3. Ввод комплексных чисел 16
1.2.4. Элементарные математические функции 17
1.2.5. Специальные математические функции 18
1.2.6. Элементарные действия с комплексными числами 20
1.2.7. Функции комплексного аргумента 20
1.2.8. Задания 21
1.2.9. Вопросы 28
1.3. Простейшие операции с векторами и матрицами ***29***
1.3.1. Ввод векторов и матриц 29
1.3.2. Формирование векторов и матриц 30
1.3.3. Извлечение и вставка частей матриц 34
1.3.4. Действия над векторами 36
1.3.5. Поэлементное преобразование матриц 39
1.3.6. Матричные действия над матрицами 41
1.3.7. Матричные функции 43
1.3.8. Задания 44
1.3.9. Вопросы 46
1.4. Функции прикладной численной математики ***47***
1.4.1. Операции с полиномами 47
1.4.2. Обработка данных измерений 50
1.4.3. Функции линейной алгебры 54
1.4.4. Аппроксимация и интерполяция данных 63
1.4.5. Векторная фильтрация и спектральный анализ 66
1.4.6. Задания 71
1.4.7. Вопросы 74
1.5. Построение простейших графиков ***75***
1.5.1. Процедура plot 75
1.5.2. Специальные графики 78
1.5.3. Дополнительные функции графического окна 83
1.5.4. Вывод графиков в печать 85
1.5.5. Задания 85
1.5.6. Вопросы 85
1.6. Операторы управления вычислительным процессом ***86***
1.6.1. Оператор условного перехода 86
1.6.2. Оператор переключения 88
1.6.3. Операторы цикла 88
1.6.4. Задания 90
1.6.5. Вопросы 92
2. Программирование в среде MatLAB ***93***
2.1. Функции функций ***93***
2.2. Создание М-файлов ***96***
2.2.1. Особенности создания М-файлов 96
2.2.2. Основные особенности оформления М-файлов 97
2.3. Создание простейших файлов-функций (процедур) ***99***
2.3.1. Общие требования к построению 99
2.3.2. Типовое оформление процедуры-функции 101
2.3.3. Задания 102
2.3.4. Вопросы 103
2.4. Создание Script-файлов ***103***
2.4.1. Основные особенности Script-файлов 103
2.4.2. Ввод и вывод информации в диалоговом режиме 104

2.4.3. Организация повторения действий 106
2.4.4. Организация изменения данных в диалоговом режиме 107
2.4.5. Типовая структура и оформление Script-файла 110
2.5. Графическое оформление результатов ***111***
2.5.1. Общие требования к представлению графической информации 111
2.5.2. Разбивка графического окна на подокна 113
2.5.3. Вывод текста в графическое окно (подокно) 114
2.6. Создание функций от функций ***118***
2.6.1. Процедура feval 118
2.6.2. Примеры создания процедур от функций 119
2.6.3. Задания 124
2.7. Пример создания сложной программы ***130***
2.7.1. Программа моделирования движения маятника 131
2.7.2. Задания 140
3. Интерфейс MatLAB и команды общего назначения. М-книг ***141***
3.1. Функции меню командного окна ***141***
3.1.1. Меню "File" 141
3.1.2. Другие меню командного окна 151
3.1.3. Панель инструментов 155
3.2. Команды общего назначения ***155***
3.3. Создание М-книги ***157***
3.3.1. Начало новой М-книги 157
3.3.2. Написание М-книги 158
3.3.3. Редактирование М-книги 160
3.3.4. Преобразование документа WORD в М-книгу 160
3.3.5. Некоторые особенности использования системы MatLAB 160
3.3.6. Изменение параметров вывода результатов 161
4. Классы вычислительных объектов ***162***
4.1. Основные классы объектов ***162***
4.1.1. Класс символьных строк (char) 163
4.1.2. Класс записей (struct) 166
4.1.3. Класс ячеек (cell) 168
4.2. Производные классы MatLAB ***171***
4.2.1. Класс объектов Inline 171
4.2.2. Классы пакета CONTROL 174
4.3. Пример создания нового класса polynom ***178***
4.3.1. Создание подкаталога @polynom 178
4.3.2. Создание конструктора 178
4.3.3. Создание процедуры символьного представления polynom-объекта 180
4.4. Создание методов нового класса ***182***
5. Цифровая обработка сигналов (пакет Signal Processing Toolbox) ***188***
5.1. Формирование типовых процессов ***190***
5.1.1. Формирование одиночных импульсных процессов 190
5.1.2. Формирование колебаний 193
5.2. Общие средства фильтрации. Формирование случайных процессов ***199***
5.2.1. Основы линейной фильтрации 199
5.2.2. Формирование случайных процессов 205
5.3. Процедуры спектрального (частотного) и статистического анализа процессов ***208***
5.3.1. Основы спектрального и статистического анализа 208
5.3.2. Примеры спектрального анализа 212
5.3.3. Статистическийанализ 221
5.4. Проектирования фильтров ***223***
5.4.1. Формы представления фильтров и их преобразования 223
5.4.2. Разработка аналоговых фильтров 227
5.4.3. Проектирование БИХ-фильтров 233
5.4.4. Проектирование КИХ-фильтров 237
5.5. Графические и интерактивные средства ***245***
5.5.1. Графические средства 245
5.5.2. Интерактивная оболочка SPTOOL 257
6. Исследование линейных стационарных систем (пакет Control Toolbox) ***270***

6.1. Ввод и преобразование моделей ***272***
6.2. Получение информации о модели ***286***
6.3. Анализ системы ***288***
6.4. Интерактивный "обозреватель" ltiview ***295***
6.5. Синтез системы ***303***
7. Моделирование нелинейных систем (пакет SimuLINK) ***308***
7.1. Общая характеристика пакета SimuLink ***308***
7.1.1. Запуск SimuLink 308
7.1.2. Библиотека модулей (блоков) 310
7.1.3. Раздел Sinks (Приемники) 312
7.1.4. Раздел Sources (Источники) 321
7.1.5. Раздел Continuous 336
7.1.6. Раздел Discrete 339
7.1.7. Раздел Math 340
7.1.8. Раздел Functions & Tables 345
7.1.9. Раздел Nonlinear 346
7.1.10. Раздел Signals & Systems 349
7.2. Построение блок-схем ***352***
7.2.1. Выделение объектов 352
7.2.2. Оперирование с блоками 353
7.2.3. Проведение соединительных линий 357
7.2.4. Проставленние меток сигналов и комментариев 359
7.2.5. Создание подсистем 362
7.2.6. Запись и распечатка S-модели 363
7.3. Примеры моделирования ***363***
7.3.1. Моделирование поведения физического маятника 363
7.3.2. Моделирование поведения гироскопа в кардановом подвесе 368
7.4. Объединение S-моделей с программами MatLab ***373***
7.4.1. Принципы функционирования блоков системы SimuLink 374
7.4.2. Функции пересечения нуля 377
7.4.3. Передача данных между средой MatLab и S-моделью 380
7.4.4. Запуск процесса моделирования S-модели из среды MatLab 383
7.4.5. Образования S-блоков путем использования программ на языке MatLab. S-функции 384
7.4.6. Пример создания S-функции 388
7.5. Создание библиотек S-блоков пользователя ***396***
7.5.1. Создание библиотеки 396
7.5.2. Маскировка блоков 402
7.5.3. Моделирование процесса ориентации космического аппарата 405

Послесловие ***413***
Список литературы ***414***
Предметный указатель ***415***
Указатель операторов, команд, функций и функциональных блоков MatLAB ***420***

Предисловие

В последние годы в университетских и инженерно-технических кругах мира наблюдается интенсивное распространение новой компьютерной системы осуществления математических расчетов - системы MatLAB. В чем причина такой популярности этой системы?

Главные преимущества "языка технических вычислений" MatLAB, которые выгодно отличают его среди других существующих ныне математических систем и пакетов, состоят в следующем:

- система MatLAB специально создана для проведения именно инженерных расчетов: математический аппарат, который используется в ней, 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, которую требует именно инженерная документация;
- язык программирования системы MatLAB весьма прост, близок к языку BASIC, посилен любому начинающему; он содержит всего несколько десятков операторов; незначительное количество операторов здесь компенсируется большим числом процедур и функций, содержание которых легко понятно 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 корректировки и модифицирования; MatLAB - система, которая может расширяться пользователем по его желанию созданными им программами и процедурами (подпрограммами); ее легко приспособить к решению нужных классов задач;
- очень удобной является возможность использовать практически все вычислительные возможности системы в режиме чрезвычайно мощного научного калькулятора; в то же время можно составлять собственные отдельные программы с целью многоразового их использования для исследований; это делает MatLAB незаменимым средством проведения научных расчетных исследований;
- последние версии MatLAB позволяют легко интегрировать ее с текстовым редактором Word, что делает возможным использование при создании текстовых документов вычислительных и графических возможностей MatLAB, например, оформлять инженерные и научные отчеты и статьи с включением в них сложных расчетов и выводом графиков в текст.

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.

Эта книга является вторым, переработанным и существенно дополненным изданием учебного пособия "Початки програмування в середовищі MatLAB" [11], содержит, в основном, описание MatLAB версии 5.3 и в него добавлены материалы по практическому овладению процедурами пакетов CONTROL (анализа и синтеза линейных систем автоматического управления), SIGNAL (цифровой обработки сигналов), SIMULINK (интерактивного моделирования динамических систем) и некоторых новых важных возможностей MatLAB.

Пособие состоит из семи глав.

В первой главе читатель знакомится с возможностями системы в режиме научного калькулятора. Здесь помещены 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, которые помогают рационально организовать вычислительный процесс.

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

Важной частью MatLAB, которая позволяет приспосабливать систему к задачам пользователя, является возможность образования 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 интерактивного (визуального моделирования динамических систем во временной области).

В целях удобства пользования учебным пособием его снабжен двумя алфавитными указателями - предметным и указателем операторов, команд, функций и функциональных блоков.

В пособии использованы материалы из изданий, указанных в списке литературы.

Введение

Система MatLAB создана фирмой MathWork Inc. (США, г. Нейтик, штат Массачусетс). Хотя впервые эта система начала использоваться в конце 70-х лет, расцвет ее применения начался в конце 80-х, в особенности после появления на рынке версии 4.0. Последние версии MatLAB, - это чрезвычайно развитые системы, которые содержат огромную совокупность 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.

Система MatLAB (сокращение от MATrix LABoratory - матричная лаборатория) представляет собой интерактивную компьютерную систему для выполнения инженерных и научных расчетов, ориентированную на работу с массивами данных. Система предполагает возможность обращения к программам, которые написаны на языках FORTRAN, C и C++.

Привлекательной особенностью системы является то, что она содержит 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 функций в квадратурах, численное интегрирование дифференциальных и разностных уравнений, построение разнообразных видов графиков, трехмерных поверхностей и линий уровня. В ней реализована удобная операционная среда, которая позволяет формулировать проблемы и получать решения в обычной математической форме, не прибегая к рутинному программированию.

Основной объект системы MatLAB - прямоугольный числовой массив (матрица), который допускает комплексные элементы. Использование матриц не требует явного указания их размеров. Система позволяет решать многие вычислительные задачи за значительно меньшее время, чем 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 многое другое. Тем не менее, характерная основная особенность системы – ее "открытость", то есть легкость ее модификации и адаптации к конкретным задачам пользователя. Пользователь может ввести в систему любую новую команду, оператор или функцию и пользоваться потом ими так же просто, как и встроенными операторами и функциями.

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.п. То есть MatLAB предоставляет пользователю большой набор готовых средств (более половины из них - внешние расширения в виде m-файлов).

Система MatLAB имеет собственный язык программирования, который напоминает BASIC. Запись программ в системе является традиционной и потому обычной для большинства пользователей персональных компьютеров. И вдобавок система дает возможность редактировать программы при помощи любого привычного для пользователя текстового редактора.

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

С системой MatLAB поставляются свыше ста m-файлов, которые содержат демонстрационные примеры и определения новых операторов и функций. Эта библиотека, все файлы которой подробно прокомментированы, - подлинная 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.

Робота в среде MatLab может осуществляться в двух режимах:

- в режиме калькулятора, когда вычисления осуществляются сразу после набора очередного оператора или команды MatLab; при этом значение результатов вычисления могут присваиваться некоторым переменным, или результаты получаются непосредственно, без присваивания (как в обычных калькуляторах);
- путем вызова имени программы, написанной на языке MatLAB, предварительно составленной и записанной на диске, 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 входных данных.

MatLAB имеет черты разных известных языков программирования высокого уровня.

С языком BASIC ее роднит то, что она представляет собой интерпретатор (то есть компилирует и выполняет программу пооператорно, не образовывая отдельного исполняемого файла), незначительное количество операторов и отсутствие необходимости в объявлении типов и размеров переменных, то есть все то, что делает язык программирования весьма удобным в пользовании.

От языка Pascal система MatLAB позаимствовала объектно-ориентированную направленность, то есть такое построение языка, которое обеспечивает образование новых типов вычислительных объектов по желанию пользователя на основе типов объектов, уже существующих в языке. Эти новые типы объектов (в MatLAB они называются *классами*) могут иметь собственные процедуры их преобразования (они образуют *методы* этого класса). Весьма удобно то, что новые процедуры могут быть вызваны с помощью обычных знаков арифметических операций и некоторых специальных знаков, которые применяются в математике.

Принципы сохранения значений переменных в MatLAB более всего приближаются к тем, которые присущи языку FORTRAN, а именно: все переменные являются *локальными*, то есть действуют лишь в границах той программной единицы (процедуры, функции или главной, управляющей программы), где им присвоены некоторые конкретные значения. При переходе к выполнению другой программной единицы, значения переменных предыдущей программной единицы или совсем теряются (в случае, если выполненная программная единица является процедурой или функцией), или становится недосягаемыми (если выполненная программа - управляющая). В отличие от языков BASIC и Pascal, в языке MatLAB *нет глобальных переменных*, действие которых распространялась бы на все программные единицы. Но и здесь в MatLAB есть особенность, которая отличает ее от других языков. В отличие от интерпретатора BASIC интерпретатор MatLAB позволяет в одном и том же сеансе работы выполнять несколько самостоятельных программ, причем все переменные, используемые в этих программах, являются общими для этих программ и образуют общее *рабочее пространство* (Work Space). Это позволяет более рационально организовывать сложные (громоздкие) вычисления по принципу, который напоминает *оверлейные* структуры.

Эти особенности MatLAB делают ее весьма гибкой и удобной в пользовании вычислительной системой.

1. MatLab как научный калькулятор

1.1. Командное окно

После вызова MatLAB из среды Windows на экране возникает изображение так называемого "командного окна" среды MatLAB (рис. 1.1).

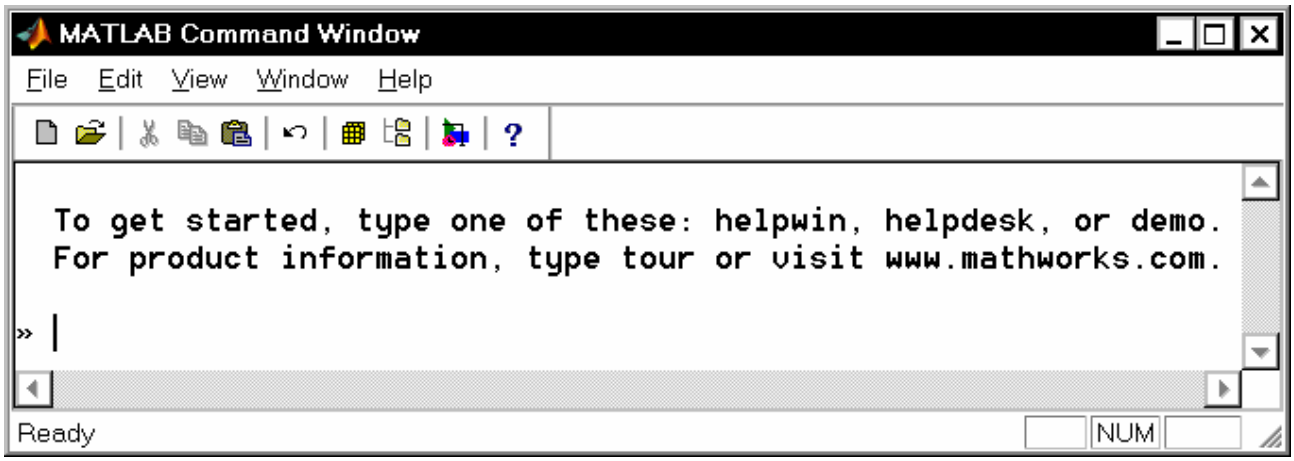

Рис. 1.1

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

Признаком того, что MatLAB готова к восприятию и выполнению очередной команды, является возникновение в последней строке текстового поля окна знака приглашения '»', после которого расположена мигающая вертикальная черта.

В верхней части окна (под заголовком) размещена строка меню, в которой находятся меню ***File***, ***Edit***, ***View, Windows***, ***Help***. Чтобы открыть какое-либо меню, следует установить на нем указатель мыши и нажать ее левую кнопку. Подробнее функции команд меню описаны далее, в главе 3 "Интерфейс MatLab и команды общего назначения. Написание М-книг".

Здесь отметим лишь, что *для выхода из среды MatLAB* достаточно открыть меню ***File*** и выбрать в нем команду ***Exit MATLAB*****,** или просто закрыть командное окно, нажав левую клавишу мыши, когда курсор мыши установлен на изображении верхней крайней правой кнопки этого окна (с обозначением косого крестика)**.**

1.2. Операции с числами

1.2.1. Ввод действительных чисел

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

для отделения дробной части мантиссы числа используется десятичная точка (вместо запятой при обычной записи);

десятичный показатель числа записывается в виде целого числа после предшествующей записи символа "e";

между записью мантиссы числа и символом "e" (который отделяет мантиссу от показателя) не должно быть никаких символов, включая и символ пропуска.

Если, например, ввести в командном окне MatLAB строку

1. 20357651e -17,

то после нажатия клавиши <Enter> в этом окне появится запись:

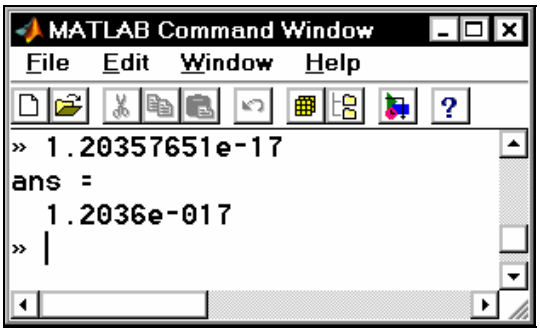

Рис. 1.2

Следует отметить, что результат выводится в виде (формате), который определяется предварительно установленным форматом представления чисел. Этот формат может быть установлен с помощью команды ***Preferences*** меню ***File*** (рис. 1.3). После ее вызова на экране появится одноименное окно (рис. 1.4). Один из участков этого окна имеет название ***Numeric Format***. Он предназначен для установки и изменения формата 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 шестнадцатиричного 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.

Избирая с помощью мыши нужный вид представления чисел, можно обеспечить в дальнейшем выведение чисел в командное окно именно в этой форме.

Рис. 1.3

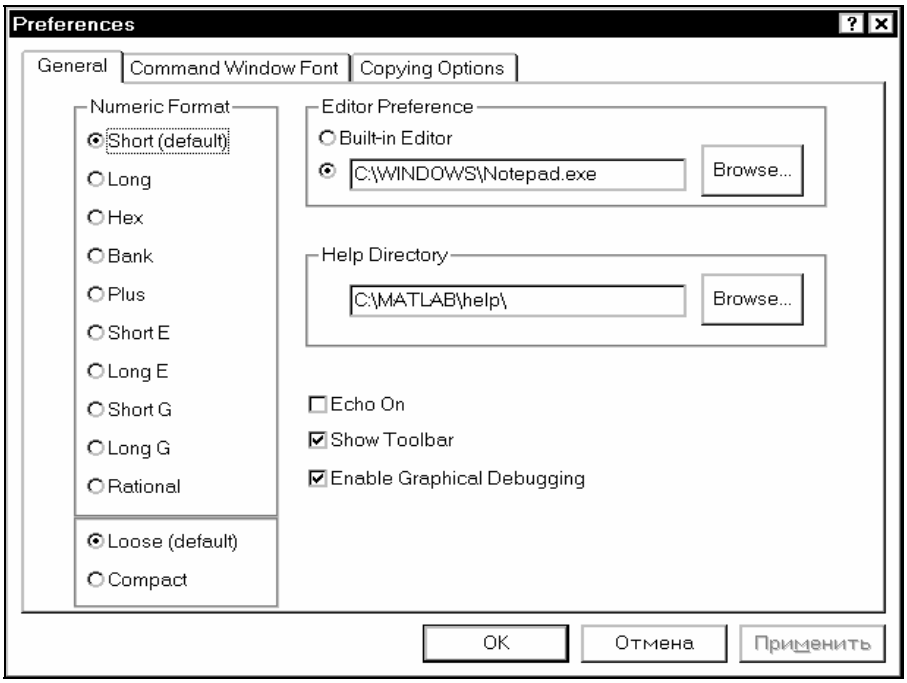

Рис. 1.4

Как видно из рис. 1.2, число, которое выведено на экран, не совпадает с введенным. Это обусловлено тем, что установленный по умолчанию формат представления чисел (Short) не позволяет вывести більше 6 значащих цифр. На самом деле введенное число сохраняется внутри MatLAB со всеми введенными его циф-

рами. Например, если избрать мышью селекторную кнопку **Long E** (т. е. установить указанный формат представления чисел), то, повторяя те же действия, получим:

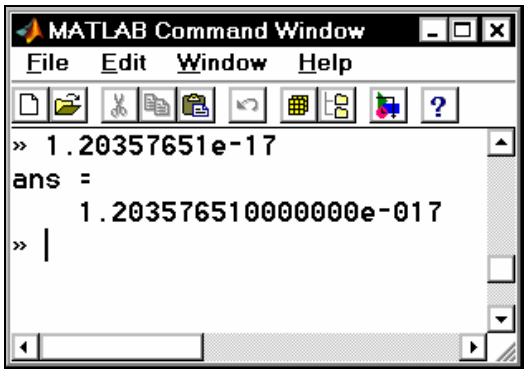

Рис. 1.5

где уже все цифры отображены верно (рис. 1.5).

Следует помнить:

- *введенное число и результаты всех вычислений в системе MatLAB сохраняются в памяти ПК с относительной погрешностью около* 2.10^{-16} *(т. е. с точными значениями в 15 десятичных разрядах);*

- *диапазон представления модуля действительных чисел лежит в диапазоне между* 10^{-308} *и* 10^{+308}.

1.2.2. Простейшие арифметические действия

В арифметических выражениях языка MatLAB используются следующие знаки арифметических операций:

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

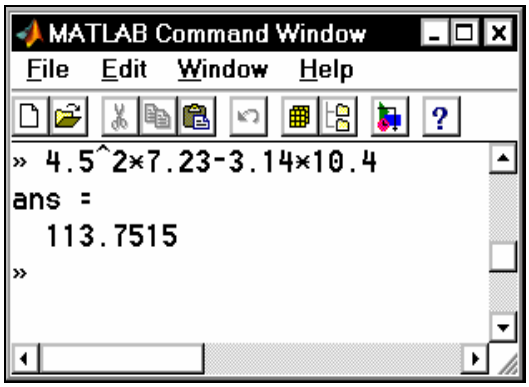

Рис. 1.6

Использование MatLAB в режиме калькулятора может происходить путем простой записи в командную строку последовательности арифметических действий с числами, то есть 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 возникнет результат выполнения в виде, представленном на рис. 1.6, т. е. на экран под именем системной переменной ***ans*** выводится результат действия последнего выполненного оператора.

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 його действия не отображается в командном окне;*
- *если оператор не содержит знака присваивания (=), т. е. является просто записью некоторой последовательности действий над числами и переменными, значение результата присваивается специальной системной переменной по имени* ***ans****;*
- *полученное значение переменной* ***ans*** *можно использовать в следующих операторах вычислений, применяя это имя* ***ans****; при этом следует помнить, что значение системной переменной* ***ans*** *изменяется после действия очередного оператора без знака присваивания;*
- *в общем случае форма представления результата в командном окне имеет вид:*

<Имя переменной> = <результат>.

Пример. Пусть нужно вычислить выражение (25+17)*7. Это можно сделать таким образом. Сначала набираем последовательность **25+17** и нажимаем <Enter>. Получаем на экране результат в виде ans = 42. Теперь записываем последовательность **ans*7** и нажимаем <Enter>. Получаем ans = 294 (рис. 1.7). Чтобы предотвратить выведение промежуточного результата действия 25+17, достаточно после записи этой последовательности добавить символ ' ; '. Тогда будем иметь результаты в виде, представленном на рис. 1.8.

Применяя MatLAB как калькулятор, можно использовать имена переменных для записи промежуточных результатов в память ПК. Для этого служит операция присваивания, которая вводится знаком равенства ' = ' в соответствия со схемой:

<Имя переменной> = <выражение>[;]

Имя переменной может содержать до 30 символов и должно не совпадать с именами функций, процедур системы и системных переменных. При этом система различает большие и маленькие буквы в переменных. 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) или символьным выражением. Если выражение не заканчивается

символом ' ; ', после нажатия клавиши <Enter> в командном окне возникнет результат выполнения в виде :

<Имя переменной> = <результат>.

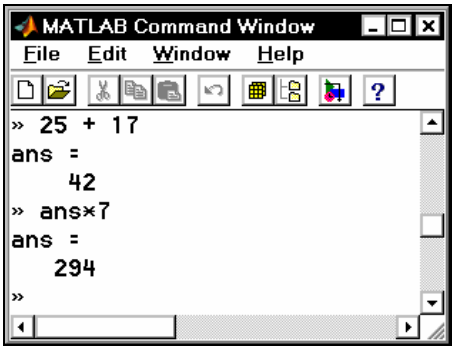

Рис. 1.7

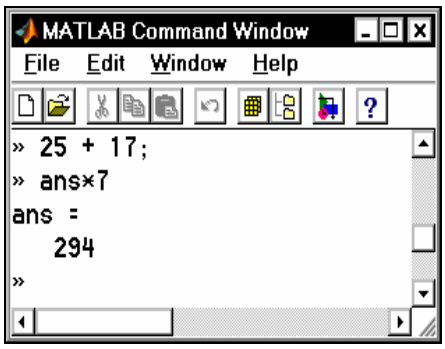

Рис. 1.8

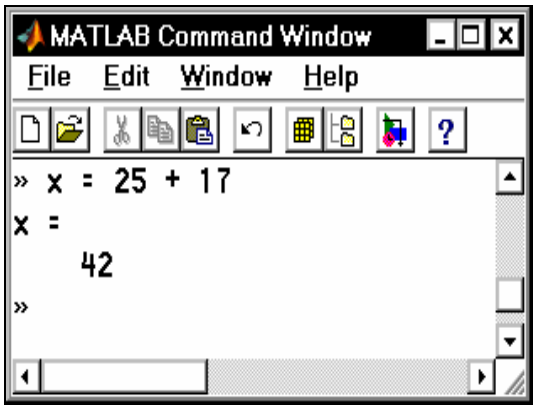

Рис. 1.9

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

Система MatLAB имеет несколько имен переменных, которые используются самой системой и входят в состав зарезервированных:

i, ***j*** - мнимая единица (корень квадратный из -1); ***pi*** - число π (сохраняется в виде 3.141592653589793); ***inf*** - обозначение машинной бесконечности; ***Na***- обозначение неопределенного результата (например, типа 0/0 или inf/inf); ***eps*** - погрешность операций над числами с плавающей запятой; ***ans***- результат последней операции без знака присваивания; ***realmax*** и ***realmin*** – максимально и минимально возможные величины числа, которое может быть использованы.

Эти переменные можно использовать в математических выражениях.

1.2.3. Ввод комплексных чисел

Язык системы MatLAB, в отличие от многих языков программирования высокого уровня, содержит в себе очень простую в пользовании встроенную арифметику комплексных чисел. Большинство элементарных математических функций допускают в качестве аргументов комплексные числа, а результаты формируются

как комплексные числа. Эта особенность языка делает его очень удобным и полезным для инженеров и научных работников.

Для обозначения мнимой единицы в языке MatLAB зарезервированы два имени ***i*** и ***j***. Ввод с клавиатуры значения комплексного числа осуществляется путем записи в командное окно строки вида:

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

где ДЧ - действительная часть комплексного числа, МЧ - мнимая часть. Например:

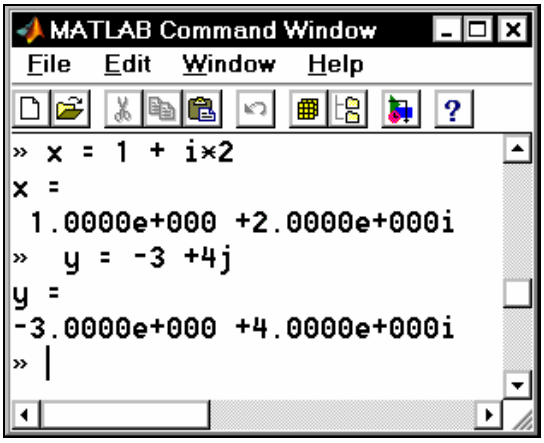

Рис. 1.10

Из приведенного примера видно, в каком виде система выводит комплексные числа на экран (и на печать).

1.2.4. Элементарные математические функции

Общая форма использования функции в MatLAB такова:

<имя результата> = <имя функции>(<перечень аргументов или их значений>).

В языке MatLAB предусмотрены следующие элементарные арифметические функции.

Тригонометрические и гиперболические функции

sin(Z) - синус числа Z;
sinh(Z) - гиперболический синус;
asin(Z) - арксинус (в радианах, в диапазоне от $-\pi/2$ к $+\pi/2$);
asinh(Z) - обратный гиперболический синус;
cos(Z) - косинус;
cosh(Z) - гиперболический косинус;
acos(Z) - арккосинус (в диапазоне от 0 к π);
acosh(Z) - обратный гиперболический косинус;
tan(Z) - тангенс;
tanh(Z) - гиперболический тангенс;
atan(Z) - арктангенс (в диапазоне от $-\pi/2$ к $+\pi/2$);

atan2(X,Y) - четырехквадрантный арктангенс (угол в диапазоне от $-\pi$ до $+\pi$ между горизонтальным правым лучом и лучом, который проходит 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 к ближайшему целому в сторону нуля;
floor(Z) - округление к ближайшему целому в сторону отрицательной бесконечности;
ceil(Z) - округление к ближайшему целому в сторону положительной бесконечности;
round(Z) - обычное округление числа Z к ближайшему целому;
mod(X,Y) - целочисленное деление X на Y;
rem(X,Y) - вычисление остатка от деления X на Y;
sign(Z) - вычисление сигнум-функції 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 содержание этих функций. Правила обращения к ним и использования пользователь может отыскать в описаниях этих функций, которые выводятся на экран, если набрать команду ***help*** и указать в той же строке имя функции.

Функции 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 Якобе;
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 при помощи обычных арифметических знаков +, -, *, /, \ и ^ соответственно.

Примеры использования приведены на рис. 1.11.

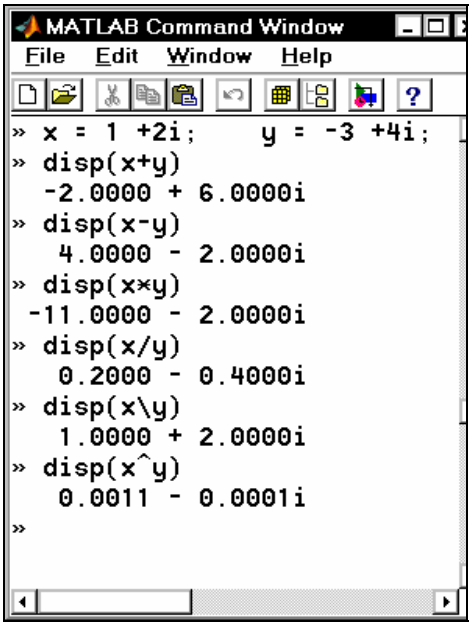

Рис. 1.11

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

1.2.7. Функции комплексного аргумента

Практически *все* элементарные *математические функции*, приведенные в п. 1.2.4, *вычисляются при комплексных значениях аргумента* и получают в результате этого комплексные значения результата.

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

В MatLAB есть несколько дополнительных функций, рассчитанных только на комплексный аргумент:

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 в диапазоне от $-\pi$ до $+\pi$);

conj(Z) - выдает число, комплексно сопряженное относительно Z.

Примеры приведены на рис. 1.13.

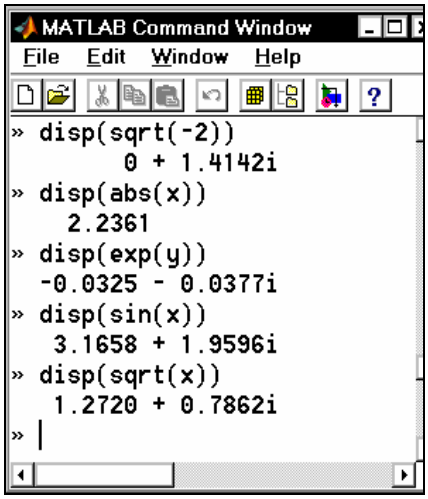

Рис. 1.12

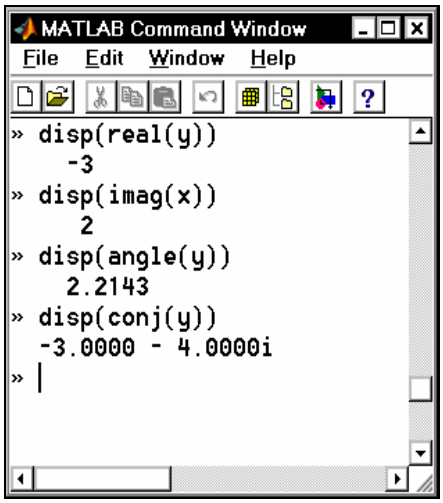

Рис. 1.13

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 векторе-результате в порядке возрастания их действительных частей, при этом элемент с отрицательной мнимой частью всегда располагается первым. Действительные элементы завершают комплексно-сопряженные пары. Например (*в дальнейшем в примерах команды, которые набираются с клавиатуры, будут написаны жирным шрифтом, а результат их выполнения - обычным шрифтом*):

```
» 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
```

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.

1.2.8. Задания

Задание 1.1. Вычислите указанное арифметическое выражение. Укажите последовательность нажатия клавиша. Сравните полученный результат с приведенным ответом.

Ответ

1. $\dfrac{\left(12\frac{1}{6}-6\frac{1}{27}-5{,}25\right)13{,}5+0{,}111}{0{,}02}$. 599,3

2. $\dfrac{\left(1\frac{1}{12}+2\frac{5}{32}+\frac{1}{24}\right):9{,}6+2{,}13}{0{,}0004}$. 6179,5

3. $\dfrac{\left(6{,}6-3\frac{3}{14}\right)5\frac{5}{6}}{(21-1{,}25):2{,}5}$. 2,5

4. $\dfrac{2{,}625-\frac{2}{3}\cdot 2\frac{5}{14}}{\left(3\frac{1}{12}+4{,}375\right):19\frac{8}{9}}$. 2,8095

5. $\dfrac{0{,}134+0{,}05}{18\frac{1}{6}-1\frac{11}{14}-\frac{2}{15}\cdot 2\frac{6}{7}}$. 0,0115

6. $\dfrac{\left(58\frac{4}{15}-56\frac{7}{24}\right):0{,}8+2\frac{1}{9}\cdot 0{,}225}{8{,}75\cdot 0{,}6}$. 0,56071

7. $\dfrac{\left(\frac{0{,}216}{0{,}15}+0{,}56\right):0{,}5}{\left(7{,}7:24{,}75+\frac{2}{15}\right)4{,}5}$. 2

8. $\dfrac{1\frac{4}{11}\cdot 0{,}22:0{,}3-0{,}96}{\left(0{,}2-\frac{3}{40}\right)1{,}6}$. 0,2

9. $\dfrac{\left(\frac{3}{5}+0{,}425-0{,}005\right):0{,}12}{30{,}5+\frac{1}{6}+3\frac{1}{3}}$. 0,25

10. $\dfrac{3\frac{1}{3}+2{,}5}{2{,}5-1\frac{1}{3}}\cdot\dfrac{4{,}6-2\frac{1}{3}}{4{,}6+2\frac{1}{3}}:\left(\dfrac{0{,}05}{\frac{1}{7}-0{,}125}+5{,}7\right)$. 0,19231

11. $\dfrac{0{,}725+0{,}42(6)}{0{,}128-6{,}25-(0{,}0345:0{,}12)}\cdot 0{,}25$. -0,04492

12. $\dfrac{\left(4{,}5\cdot 1\frac{2}{3}-6{,}75\right)\cdot 0{,}(6)}{\left(3{,}333\cdot 0{,}3+0{,}222\cdot\frac{4}{9}\right)\ 2\frac{2}{3}}$. 0,17068

13. $\dfrac{\left(5\frac{4}{45}-4\frac{1}{6}\right)\ :\ 5\frac{8}{15}}{\left(4\frac{2}{3}+0{,}75\right)3\frac{9}{13}}\cdot 34\frac{2}{7}$. 0,28571

14. $\dfrac{1\frac{4}{11}\cdot 0{,}22\ :\ 0{,}3-0{,}96}{\left(0{,}2-\frac{3}{40}\right)1{,}68}$. 0,19048

15. $\dfrac{\left(40\frac{7}{30}-38\frac{5}{12}\right)\ :\ 10{,}9+\left(0{,}875-\frac{7}{30}\right)\cdot\frac{20}{11}}{0{,}008}$. 166,67

16. $\dfrac{(68{,}023-66{,}028)\ :\ 6\frac{1}{9}+\frac{7}{40}\cdot 4{,}5}{0{,}042+0{,}086}$. 8,7028

17. $\dfrac{(2{,}1-1{,}965)\ :\ (1{,}2\cdot 0{,}045)}{0{,}00325\ :\ 0{,}013}-\dfrac{4}{0{,}2\cdot 073}$. -17,397

18. $\dfrac{(1{,}88+2{,}127)\cdot 0{,}01875}{0{,}625-\frac{13}{18}\ :\ 3{,}13}+8{,}29$. 8,2441

19. $\dfrac{3\ :\ 0{,}4-0{,}009\ :\ (0{,}15\ :\ 2{,}5)}{0{,}32\cdot 6+0{,}033-(5{,}3-3{,}88)}$. 13,79

20. $\dfrac{(34{,}06-33{,}81)\cdot 4}{6{,}84\ :\ (28{,}57-25{,}15)}+1{,}33\ :\ \dfrac{4}{21}$. 7,4825

21. $\dfrac{8{,}8077}{20-\left(28{,}2\ :\ (13{,}333\cdot 0{,}3+0{,}0125)\right)2{,}004}$. 1,4889

22. $\dfrac{\left(1{,}75\ :\ \frac{2}{3}-1{,}75\cdot 1{,}125\right)\ :\ \frac{7}{12}}{(0{,}2012-0{,}0325)\ :\ 400}$. 2667,5

23. $\dfrac{\left(26\frac{1}{3}-18{,}02\cdot 0{,}75\right)\cdot 2{,}4\ :\ 0{,}88}{1{,}37-23\frac{2}{3}\ :\ 1{,}82}$. -3,005

24. $26:\dfrac{3:(0{,}48-0{,}27)}{2{,}52(1{,}38+2{,}45)}+1{,}27.$ 18,836

25. $\left(16{,}5-13\dfrac{7}{9}\right)\dfrac{6}{11}+2{,}2:(0{,}241-0{,}91)$. -1,8036

Задание 1.2. Проведите вычисления по заданной формуле при заданных значениях параметров. Укажите необходимую последовательность действий. Сравните полученный результат с приведенным ответом.

Указание. *В системе MatLAB несколько последних команд запоминаются. Повторный вызов этих команд в командное окно осуществляется нажатием клавиш* <↓> и <↑>. Используйте эту возможность для повторного обращения к набранной функции.

1. $3m^2+\sqrt[3]{2n^2}:m$; а) $m=-\dfrac{14}{5},\ n=tg\dfrac{\pi}{8}$; б) $m=2{,}2\cdot10^{-2},\ n=\dfrac{1}{3{,}1}$.

ОТВЕТ: а) 23,27; б) 26,938.

2. $\dfrac{4}{3}l^3\sin^2\dfrac{\alpha}{2}\sqrt{\cos\alpha}$; а) $l=1{,}7\cdot10^3,\ \alpha=18^\circ$; б) $l=\dfrac{16}{21},\ \alpha=\dfrac{\pi}{5}$.

ОТВЕТ: а) 1. 5633e+008; б) 5. 0651e-002.

3. $\sqrt{\dfrac{a\sqrt{b}}{\sqrt[3]{tg\alpha}}}$; а) $a=1{,}5,\ b=0{,}8,\ \alpha=61^\circ$; б) $a=3\cdot10^{-2},\ b=0{,}71,\ \alpha=\dfrac{3}{7}\pi$.

ОТВЕТ: а) 1. 0498e+000; б) 1. 2429e-001.

4. $\dfrac{3a^2\sqrt{6{,}8\cdot(a-b)}}{4(a+b)^3}$; а) $a=4{,}13\cdot10^{-1},\ b=\dfrac{1}{261}$;

б) $a=\sin\dfrac{5\pi}{8},\ b=-tg12^\circ$.

ОТВЕТ: а) 2. 9464e+000; б) 4. 9445e+000.

5. $\dfrac{c^3}{6}\cos\dfrac{\alpha}{2}\sqrt{\sin\alpha}$; а) $c=\lg2{,}38,\ \alpha=\dfrac{\pi}{5}$; б) $c=e^{-0{,}3},\ \alpha=65^\circ$.

ОТВЕТ: а) 3. 4657e-004; б) 2. 2120e-002.

6. $\sqrt{\dfrac{n^3}{16{,}3\sin\alpha\sin2\alpha}}$; а) $n=3{,}1516\cdot10^{-2},\ \alpha=5^\circ$; б) $n=e^{3{,}5},\ \alpha=\dfrac{2\pi}{13}$.

ОТВЕТ: а) 1. 1265e-002; б) 7. 6324e+001.

7. $5\sin35^\circ\sqrt{\dfrac{S^3\cos36^\circ}{\pi^3tg\alpha}}$; а) $S=\ln3,\ \alpha=44^\circ$; б) $S=\dfrac{18}{25},\ \alpha=\dfrac{7}{12}\pi$.

ОТВЕТ: а) 5. 4283e-001; б) 8. 9703e-018+ 1. 4650e-001i.

8. $\left|\lg(1+\sin\alpha)+\ln(1-\sin\beta)\right|$; а) $\alpha=\dfrac{3\pi}{7},\ \beta=83^\circ$; б) $\alpha=\dfrac{2}{3}\pi,\ \beta=16^\circ$.

ОТВЕТ: а) 4. 6035e+000; б) 5. 1546e-002.

9. $\sqrt[3]{\sin^2(\alpha+\beta)-\sin^2(\alpha-\beta)}$; а) $\alpha=\frac{5}{7}\pi,\ \beta=0{,}3\pi$; б) $\alpha=12^\circ,\ \beta=220^\circ$

ОТВЕТ: а) 4. 8756e-001+ 8. 4448e-001i; б) 7. 3715e-001.

10. $\left(\log_a(b+1{,}4)\right)^{-3/4}$; а) $a=3{,}56,\ b=e^{0{,}316}$; б) $a=2,$ b = $2{,}1649\cdot10^{-2}$.

ОТВЕТ: а) 1. 1790e+000; б) 1. 6630e+000.

11. $3\left(p^{-2/3}+q^{-1/2}\right)\sqrt[3]{pq}$; а) $p=\ln 3,\ q=\lg 3$; б) $p=0{,}013,\ q=1{,}4\cdot10^2$.

ОТВЕТ: а) 5. 7737e+000; б) 6. 6559e+001.

12. $\frac{2}{3}m\sqrt{m\sqrt[3]{m\sqrt[4]{m}}}$; а) $m=3{,}6485\cdot10^2$; б) $m=\frac{24}{37}$.

ОТВЕТ: а) 1. 5880e+004; б) 5. 4516e-001.

13. $\frac{8}{3}S\sqrt{\frac{S}{\pi}}\sin^6\frac{\alpha}{2}$; а) $S=e^{1{,}11},\ \alpha=\frac{7}{11}\pi$; б) $S=5{,}403,\ \alpha=28^\circ$.

ОТВЕТ: а) 2. 8187e+000; б) 3. 7879e-003.

14. $2\sqrt{\frac{F}{\pi}}tg\alpha\sin^2\frac{\alpha}{2}$; а) $F=\frac{1}{0{,}03},\ \alpha=\frac{5}{7}\pi$; б) $F=\ln 7,\ \alpha=1{,}34^\circ$.

ОТВЕТ: а) -6. 6313e+000; б) 5. 0346e-006.

15. $\frac{1}{12}\cdot\frac{m^3\cos\alpha}{(\sin\alpha+\cos\alpha)^3}$; а) $m=-20{,}1,\ \alpha=20^\circ$; б) $m=\lg 13{,}6,\ \alpha=1{,}48$.

ОТВЕТ: а) -3. 0201e+002; б) 8. 5792e-003.

16. $\frac{\sqrt{3h^3}}{\cos^2\alpha}\sin(\alpha+30^\circ)\sin(\alpha-30^\circ)$;

а) $h=0{,}28,\ \alpha=41^\circ$; б) $h=e^{0{,}415},\ \alpha=237.^\circ$

ОТВЕТ: а) 8. 1284e-002; б) 4. 9334e+000.

17. $\frac{\alpha}{3}\left(\lg(d+2)-tg\alpha\right)^2$;

а) $d=6{,}178,\ \alpha=20^\circ$; б) $d=-2{,}2461\cdot10^{-2},\ \alpha=1{,}146.$

ОТВЕТ: а) 3. 5028e-002; б) 1. 4003e+000.

18. $d^3ctg\alpha\sqrt{\sin^4\alpha-\cos^4\alpha}$; а) $d=10{,}6,\ \alpha=50^\circ$; б) $d=e^{2,3},\ \alpha=1$.

ОТВЕТ: а) 4. 1645e+002; б) 4. 1101e+002.

19. $\frac{a^2\sqrt{3}}{4}(\sec\alpha+\operatorname{cosec}\alpha)^4$;

а) $a=5{,}08,\ \alpha=25^\circ$; б) $a=\ln 1{,}37,\ \alpha=\frac{12}{25}\pi$

ОТВЕТ: а) 1. 6193e+003; б) 3. 5238e+003.

20. $\frac{\sqrt{\pi}}{3}\cdot\frac{1}{\left(ctgA+ctgB\right)^2}$; а) $A=51^\circ$, $B=39^\circ$; б) $A=0{,}643$, $B=\frac{\pi}{7}$.

ОТВЕТ: а) 1. 4132e-001; б) 5. 0772e-002.

21. $\lg\left(3^{x^2-x-9}+\frac{8}{27}\right)$; а) $x=e^{1{,}648}$; б) $x=tg1{,}21$.

ОТВЕТ: а) 6. 1109e+000; б) -5. 1927e-001.

22. $\frac{\sqrt[5]{5e^{4a}\left(a+12{,}36\right)^2}}{\ln\left(a+7\right)}$; а) $a=2{,}1754\cdot10^2$; б) $a=\cos17^\circ$.

ОТВЕТ: а) 8. 5511e+075 ; б) 4. 0272e+000;

23. $\lg^2 x-\left(\frac{27}{8}\right)^{x-1}\sin\sqrt{x}$; а) $x=e^{2{,}145}$; б) $x=2{,}468\cdot10^{-1}$.

ОТВЕТ: а) -2. 0936e+003; б) 1. 7858e-001.

24. $\sqrt[5]{\left(x-y\right)^2\sqrt[3]{1/y-x}}$; а) $x=e^{-0{,}37}$, $y=\ln2{,}1517$; б) $x=37^\circ$, $y=\cos\frac{7}{24}\pi$.

ОТВЕТ: а) 3. 4445e-001; б) 2. 6745e-001.

25. $\frac{\sin A+tgB}{\sqrt[5]{\left(A-3B\right)^2}}$; а) $A=5{,}6$, $B=\lg25$; б) $A=\frac{8}{9}\pi$, $B=\frac{\pi}{10}$.

ОТВЕТ: а) 4. 4466e+000; б) 5. 2145e-001.

Задание 1.3. Выполните такие действия (см. таблицу 1.1):

а) число z_1, заданное в алгебраической (экспоненциальной) форме, переведите в экспоненциальную (алгебраическую), проверьте и запишите результат;

б) число z_2, заданное в экспоненциальной (алгебраической) форме, переведите в алгебраическую (экспоненциальную), проверьте и запишите результат;

в) вычислите заданное выражение; запишите результат экспоненциальной форме, причем аргумент результата обеспечьте в границах между (-π) и +π.

Таблица 1.1

Вариант	Комплексное число				Выражение
	z_1	z_2	z_3	z_4	
1	$4+3i$	$2{,}71e^{i\pi/12}$	$1{,}82e^{-1{,}2i}$	$\sqrt{3}-2i$	$z_1^2\cdot z_2:z_3+z_4$
2	$0{,}8-2i$	$3{,}08e^{i7\pi/12}$	$8{,}01e^{2i}$	$-5+\sqrt{2}i$	$z_1^2:z_2+z_3-z_4$
3	$-0{,}7+4i$	$1{,}74e^{i0{,}3\pi}$	$3+4i$	$2{,}1e^{-2{,}3i}$	$\sqrt{z_1:z_2}\cdot z_3+z_4$
4	$-3-2i$	$3{,}21e^{15^\circ i}$	$1{,}2+3i$	$2{,}71e^{-78^\circ i}$	$\sqrt{z_1\cdot z_2}:z_3+z_4$
5	$2{,}71e^{i\pi/12}$	$-0{,}7+4i$	$1{,}31e^{-i5\pi/12}$	$-8-3i$	$\sqrt{z_1}:z_2\cdot z_3-z_4$

6	$3{,}08e^{i7\pi/12}$	$-3-2i$	$2{,}03e^{i14/13}$	$\sqrt{3}+\sqrt{2}i$	$(z_1+z_2)^4\cdot z_3:z_4$
7	$1{,}74e^{0{,}3\pi i}$	$0{,}8-2i$	$3{,}28e^{-1{,}2i}$	$\sqrt{3}-\sqrt{2}i$	$(\sqrt{z_1}+z_2)\cdot z_3:z_4$
8	$3{,}21e^{15i}$	$4+3i$	$\sqrt{3}-4i$	$1{,}23e^{111^\circ i}$	$(z_1-z_2)^3\cdot z_3+z_4$
9	$1+i\pi/2$	$1{,}2e^{107^\circ i}$	$0{,}8-2i$	$2{,}5e^{14^\circ i}$	$(z_1:z_2+z_3)^3\cdot z_4$
10	$\sqrt{5}-i$	$0{,}7e^{1{,}7i}$	$1{,}2e^{0{,}9i}$	$-3-2i$	$(z_1:z_2+z_3)^2-z_4$
11	$0{,}187-3{,}94i$	$0{,}3e^{-107^\circ i}$	$-0{,}7+4i$	$1{,}5e^{23^\circ i}$	$\sqrt[3]{z_1+z_2-z_3}\cdot z_4$
12	$-1+\sqrt{5}i$	$2{,}1e^{211^\circ i}$	$0{,}4e^{32^\circ i}$	$4+3i$	$\sqrt[3]{z_1}\cdot z_2:z_3+z_4$
13	$-\sqrt{3}-4i$	$1{,}25e^{-0{,}8i}$	$-3-2i$	$0{,}75e^{0{,}7i}$	$(\sqrt[3]{z_1\cdot z_2}+z_3):z_4$
14	$1{,}2e^{1{,}7i}$	$0{,}18-3{,}9i$	$0{,}71e^{4i}$	$0{,}8-2i$	$(\sqrt[3]{z_1:z_2}+z_3)\cdot z_4$
15	$0{,}3e^{-97^\circ i}$	$-1+\sqrt{5}i$	$-0{,}7+4i$	$5{,}2e^{71^\circ i}$	$(\sqrt{z_1\cdot z_2}-z_3):z_4$
16	$1{,}25e^{0{,}6i}$	$-\sqrt{3}-4i$	$4+3i$	$2{,}5e^{3{,}8i}$	$(\sqrt{z_1:z_2}-z_3)\cdot z_4$
17	$1{,}05e^{-0{,}4i}$	$\sqrt{5}-i$	$2{,}7e^{0{,}8i}$	$-0{,}7+4i$	$\sqrt{(z_1:z_2+z_3)\cdot z_4}$
18	$2{,}1e^{73^\circ i}$	$1+i\pi/2$	$\sqrt{2}+\sqrt{3}i$	$1{,}93e^{192^\circ i}$	$\sqrt{(z_1\cdot z_2-z_3):z_4}$
19	$2{,}7+0{,}8i$	$2e^{-\sqrt{3}i}$	$0{,}81e^{i\pi/7}$	$-\sqrt{2}-\sqrt{3}i$	$\sqrt{(z_1+z_2):z_3\cdot z_4}$
20	$-0{,}8+2{,}7i$	$-2e^{\sqrt{3}i}$	$0{,}9e^{i5\pi/7}$	$3{,}1-2{,}1i$	$\sqrt{(z_1+z_2)\cdot z_3:z_4}$
21	$-1{,}1-3{,}2i$	$0{,}33e^{-1{,}9i}$	$2e^{\sqrt{2}i}$	$2{,}08+i$	$\sqrt{z_1-z_2}\cdot z_3:z_4$
22	$2{,}1-3{,}2i$	$0{,}68e^{148^\circ i}$	$\sqrt{5}+\sqrt{2}i$	$2{,}73e^{23^\circ i}$	$\sqrt{z_1-z_2}:z_3\cdot z_4$
23	$1{,}1e^{-0{,}8i}$	$\sqrt{5}-2i$	$-1{,}7+i$	$0{,}97e^{\sqrt{2}i}$	$((z_1+z_2)^2-z_3):z_4$
24	$2{,}1e^{0{,}8i}$	$-\sqrt{5}+2i$	$1{,}7e^{\sqrt{3}i}$	$0{,}8e^{2{,}5i}$	$((z_1-z_2)^2+z_3):z_4$
25	$1{,}1e^{-2{,}1i}$	$\pi/8-2{,}1i$	$2{,}71+0{,}4i$	$1{,}71e^{-\sqrt{3}i}$	$(z_1-z_2)^3:z_3+z_4$

Задание 1.4. Найдите корни квадратного уравнения

$$a\cdot x^2+b\cdot x+c=0$$

при заданных значениях коэффициентов *a*, *b* и *c* (см. таблицу 1.2).

Таблица 1.2

Вариант	*a*	*b*	*c*
1	0.56	1.2e-4	4.08
2	1	0.1	100

3	4. 2e-3	8. 03e-4	1.06
4	7. 1e3	9. 4e4	8. 3e10
5	5.09	4.32	256
6	8.3	5.34	693
7	27	27	1276
8	3.08	0.2	30
9	5.3	10.6	876
10	0.45	0. 034	121
11	4.3	10.7	3. 4e3
12	13	0.8	287
13	6. 035	5.2	875
14	2.3	7.9	324
15	1	0.02	16.57
16	1.3	0.56	18.8
17	0.13	0. 056	18.8
18	17	12	956
19	0. 085	1	1. 3e3
20	1.2	0.32	15
21	7.1	6.4	256
22	0.2	0. 002	2.9
23	1. 4e-3	3.9	2. 6e2
24	0.86	3.2	5. 4e2
25	7. 3e3	8. 2e2	3. 5e8

1.2.9. Вопросы

1. Как представляются действительные числа при вычислениях в системе MatLAB?

2. Как изменить формат представления действительных чисел в командном окне?

3. Каким образом объявляются переменные в языке MatLAB?

4. Как сделать так, чтобы результат действий, записанных в очередной строке а) выводился в командное окно; б) не выводился на экран?

5. Какую роль играет системная переменная ***ans***?

6. Как возвратить в командную строку ранее введенную команду?

7. Как ввести значения комплексного числа, и в каком виде оно выведется на экран?

8. Как на языке MatLAB обеспечить сложение, вычитание, умножение, деление и возведение в степень комплексных чисел?

9. Какие функции работы с комплексными числами предусмотрены в языке MatLAB?

1.3. Простейшие операции с векторами и матрицами

MatLAB - система, специально предназначенная 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 система воспринимает как матрицу размером (1*1), а вектор-строку из N элементов - как матрицу размером (1*N).

1.3.1. Ввод векторов и матриц

Начальные значения векторов можно задавать с клавиатуры путем поэлементного ввода. Для этого в строке следует сначала указать имя вектора, потом поставить знак присваивания ' = ', затем, - открывающую *квадратную* скобку, а за ней ввести заданные значения элементов вектора, *отделяя их пробелами или запятыми*. Заканчивается строка записью закрывающей квадратной скобки.

Например, запись строки V = [1.2 -0.3 1.2e-5] задает вектор V, который содержит три элемента со значениями 1.2, -0.3 и 1.2e-5 (рис. 1.14):

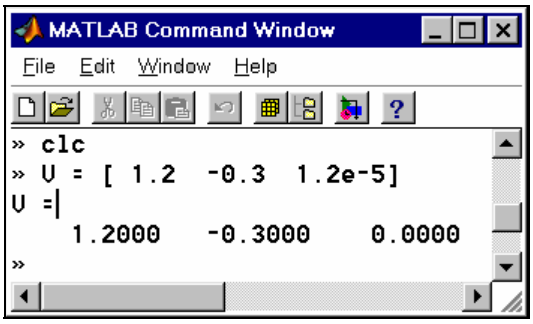

Рис. 1.14

После введения вектора система выводит его на экран. То, что в приведенном примере последний элемент выведен как 0, обусловлено установленным форматом short, в соответствии с которым выводятся данные на экран.

Длинный вектор можно вводить частями, которые потом объединять с помощью операции *объединения векторов в строку* : v = [v1 v2]. Например:

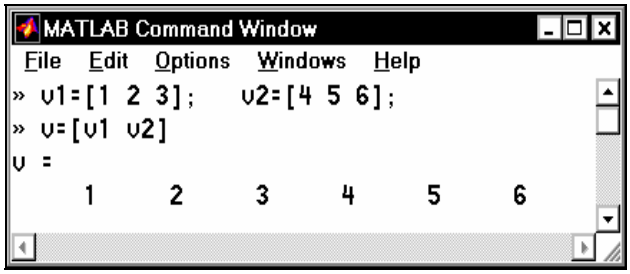

Рис. 1.15

Язык MatLAB дает пользователю возможность *сокращенного введения вектора, значения элементов которого составляют арифметическую прогрессию*. Если обозначить: nz - начальное значение этой прогрессии (значение первого эле-

мента 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, введение строки V = - 0.1 : 0.3 : 1.4 приведет к такому результату:

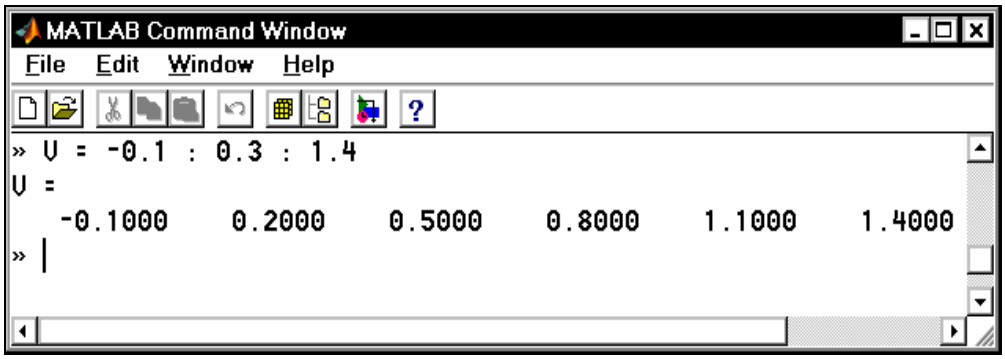

Рис. 1.16

Если средний параметр (разность прогрессии) не указан, то он по умолчанию принимается равным единице. Например, команда

```
>> -2.1 : 5
```

приводит к формированию такого вектора

```
ans =  -2.1000 -1.1000 -0.1000 0.9000 1.9000 2.9000 3.9000   4. 9000
```

Так вводятся векторы-строки. *Вектор-столбец* вводится аналогично, но *значения элементов отделяются знаком ";".*

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

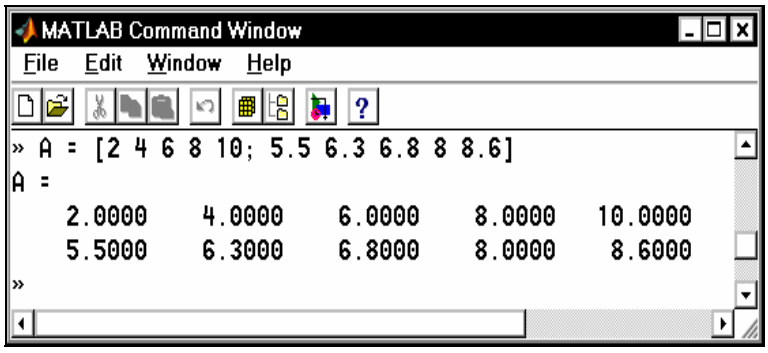

Рис. 1.17

1.3.2. Формирование векторов и матриц

MatLAB имеет несколько функций, которые позволяют формировать векторы и матрицы некоторого определенного вида. К таким функциям относятся:

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

```
» 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, например:

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

eye(M,N) - создает единичную матрицу размером (M*N), т. е. с единицами по главной диагонали и остальными нулевыми элементами, например:

```
» 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, например:

```
» 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 (гауссовому) закону с нулевым математическим ожиданием и стандартным (среднеквадратичным) отклонением, равным единице, например:

```
» 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), например:

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

hilb(N) - создает матрицу Гільберта размером (N*N), например:

```
» 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), например:

```
» 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), например:

```
» 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 принадлежат:

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) - переставляет строки заданной матрицы A относительно горизонтальной оси, например:

```
» 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
```

reshape(A,m,n) - образует матрицу размером (m*n) путем выборки элементов заданной матрицы A по столбцам и последующему распределению этих элементов по 'n' столбцам, каждый из которых содержит 'm' элементов; при этом число элементов матрицы A должно равняться m*n, например:

```
» 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, который состоит 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 указать дополнительно 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 таким 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 заданной матрицы в MatLAB осуществляется путем указания (в скобках, через запятую) после имени матрицы 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
```

Тогда получить значение элемента этой матрицы, расположенного на пересечении второй строки с третьим столбиком, можно следующим образом:

```
>> A(2,3)
ans =    7
```

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

Рассмотрим эти операции на примерах.

Пусть нужно создать вектор V1, состоящий из элементов третьего столбца последней матрицы A. Для этого произведем такие действия:

```
>> V1 = A(:, 3)
V1 =
    3.0000
    3. 1416
   11. 0000
```

Чтобы создать вектор V2, состоящий из элементов второй строки матрицы A, поступают так:

```
>> V2 = A(2, : )
V2 =   5. 0000   6. 0000   3. 1416   8. 0000
```

Допустим, что необходимо из матрицы A образовать матрицу B размером (2*2), которая состоит из элементов левого нижнего угла матрицы A. Тогда делают так:

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

Аналогично 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
```

Как видно, для этого *вместо указания номеров элементов матрицы можно указывать диапазон изменения этих номеров путем указания 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, за исключением отдельных элементов. Для примера рассмотрим формирование разреженной матрицы размером (5*7) с единичными элементами в ее 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 обычной записи *"V = A(:)".* При этом создается вектор-столбец с количеством элементов (m*n), в котором столбцы заданной 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, "расширить" матрицу, составляя ее из отдельных заданных матриц ("блоков") можно тоже довольно просто. Если заданы несколько матриц-блоков A1, A2,... AN с одинаковым количеством 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. *Вертикальная конкатенация* матриц реализуется (при условии, что все составные блоки-матрицы имеют одинаковое количество столбцов) аналогично, путем применения для отделения блоков вместо запятой точки с запятой:

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

Приведем примеры. 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

Будем различать две группы действий над векторами:

а) векторные действия - т. е. такие, которые предусмотрены векторным исчислением в математике;

б) *действия по преобразованию элементов* - это действия, которые преобразуют элементы вектора, но не являются операциями, разрешенными математикой.

Векторные действия над векторами

Сложение векторов. Как известно, суммироваться могут только векторы одинакового типа (т. е. такие, которые являются или векторами-строками, или векторами-столбцами), имеющие одинаковую длину (т. е. одинаковое количество 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 вектора, который нужно транспонировать. Например:

```
» 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.

Пример:

```
» 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-множителей строка, а второй - столбец. Иначе говоря, если векторы X и Y являются строками, то математическое смысл имеют лишь две формы умножения этих векторов: U = X' * Y и V = X * Y'. При этом в первом случае результатом будет квадратная матрица, а во втором - число. В MatLAB умножение векторов осуществляется применением обычного знака умножения ' * ', который записывается между множителями-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***. Для этого, если заданы два трехкомпонентных вектора v1 и v2, 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.

Поэлементное преобразование векторов

В языке MatLAB предусмотрен ряд операций, которые преобразуют заданный вектор в другой того же размера и типа, но не являются операциями с вектором как математическим объектом. К таким операциям относятся, например, *все элементарные математические функции*, приведенные в разделе 1.2.4 и которые зависят от одного аргумента. В языке MatLAB запись, например, вида Y = sin(X), где X - некоторый известный вектор, приводит к формированию нового вектора Y, имеющего тот же тип и размер, но элементы которого равняются синусам соответствующих элементов вектора-аргумента X. Например:

```
» 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. 3679   1. 0000   2. 7183   7. 389
```

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

Добавление (отнимание) числа к (из) каждому элемента вектора. Осуществляется с помощью знака ' + ' (' - ').

Поэлементное умножение векторов. Проводится с помощью 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 с помощью 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 с помощью совокупности знаков '.^ '. Результат - вектор, каждый элемент которого является соответствующим элементом первого вектора, возведенным в степень, величина которой равняется значению соответствующего элемента второго вектора. Примеры:

```
» 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 (а затем - строить графики) сложные математические функции, не используя при этом операторы цикла, т. е. осуществлять построение графиков в режиме калькулятора. Для этого достаточно задать значение аргумента как арифметическую прогрессию так, как это было показано в п. 1.3.1, а потом записать нужную функцию, используя знаки поэлементного преобразования векторов.

Например, пусть нужно 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 с помощью лишь двух простых операторов :

```
» a = 3;   h = 0.5;     x = 0:10;
» y = a * exp(-h*x) . * sin(x)
y =
 Columns 1 through 7
       0 1.5311 1.0035 0.0945 -0.3073 -0.2361 -0.0417
 Columns 8 through 11
  0. 0595    0. 0544    0. 0137   -0. 0110
```

1.3.5. Поэлементное преобразование матриц

Для поэлементного преобразования матрицы пригодны все указанные ранее в п. 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 этого, в MatLAB определены операции ***поэлементного умножения*** матриц одинакового размера (совокупностью знаков ' .* ', записываемой между именами перемножаемых матриц), ***поэлементного деления*** (совокупности './ ' и '.\'), ***поэлементного возведения в степень*** (совокупность '.^'), 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 следующим образом: A + x, или x + A (A - матрица, а x - число). Такой операции нет в математике. В MatLAB она эквивалентна совокупности операций

$$A + x * E,$$

где E - обозначение матрицы, которая состоит только из единиц, тех же размеров, что и матрица A. 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 над матрицами

К матричным действиям над матрицами относят такие операции, которые используются в матричном исчислении в математике и не противоречат ему.

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 являются возможными:

при сложении или вычитании матрицы должны иметь одинаковые размеры;

при умножении матриц количество столбцов первой матрицы должно совпадать с количеством строк второй матрицы.

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, а ее определитель не должен равняться 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 С, что является признаком правильности выполнения обращения матрицы.

Возведение матрицы в целую степень осуществляется в MatLAB с помощью знака "^": A^***n***. При этом матрица должна быть квадратной, а ***n*** - целым (положительным или отрицательным) числом. Это матричное действие эквивалентно умножению матрицы А на себя ***n*** раз (если ***n*** - положительно) или умножению обратной матрицы на себя (при ***n*** отрицательно).

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
```

Оригинальными в языке MatLAB являются две новые, неопределяемые в математике функции ***деления матриц***. При этом вводятся понятие ***деления матриц слева направо*** и ***деление матриц справа налево***. Первая операция записывается с помощью знака ' / ' , а вторая - ' \ '.

Операция B / A эквивалентна последовательности действий B * ***inv***(A), где 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 представляет собой решение матричного уравнения:

$$A * X = B.$$

Для примера рассмотрим задачу отыскания корней системы линейных алгебраических уравнений:

$$x_1 + 2x_2 + 3x_3 = 14$$
$$2x_1 - x_2 - 5x_3 = -15$$
$$x_1 - x_2 - x_3 = -4$$

В среде 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) осуществляется с помощью функций ***expm, expm1, expm2, expm3.*** Эти функции следует отличать от прежде рассмотренной функции ***exp***(A), которая формирует матрицу, каждый элемент которой равняется ***e*** в степени, которая равняется соответствующему элементу матрицы A.

Функция ***expm*** является встроенной функцией MatLAB. Функция ***expm1***(A) реализована как M-файл, который вычисляет 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 на основе использования спектрального разложения 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
    1.2302 + 0. 0000i      -0. 0882 - 0. 0000i       0. 7548 - 0. 0000i
```

Функция ***logm***(A) осуществляет 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
```

1.3.8. Задания

Задание 1.5. Вычислите значения функции *f(x)* на отрезке [*a*; *b*] с шагом *h*.

Таблица 1.3

Вариант	$f(x)$	*a*	*b*	*h*
1	$\frac{x^2}{1+0{,}25\sqrt{x}}$	1,1	3,1	0,2
2	$\frac{x^3-0{,}3x}{\sqrt{1+2x}}$	2,05	3,05	0,1
3	$\frac{2e^{-x}}{2\pi+x^3}$	0	1,6	0,16
4	$\frac{\cos \pi x^2}{\sqrt{1-3x}}$	-1	0	0,1
5	$\sqrt{1+4x}\sin \pi x$	0,1	0,8	0,07
6	$\frac{e^{x/3}}{1+x^2}$	1,4	2,4	0,1
7	$e^{-2x}+x^2-1$	0,25	2,25	0,2
8	$(e+x)\sin\left(\pi\sqrt{x-1}\right)$	1,8	2,8	0,1
9	$\sqrt{3+2x}\cdot tg\frac{\pi x^3}{2}$	0,1	0,9	0,08

10	$\sqrt{2+3x}\cdot\ln\left(1+3x^2\right)$	-0,1	0,9	0,1
11	$\sqrt[3]{x^2+3}\cdot\cos\frac{\pi x}{2}$	1	2,5	0,15
12	$\left(4+7x\right)\sin\left(\pi\sqrt[3]{1+x}\right)$	0	7	0,7
13	$e^{-x^2}\left(1+3x-x^2\right)$	0	2	0,2
14	$x^3-3x+\frac{8}{\sqrt{1+x^2}}$	0	1,7	0,17
15	$\sqrt{sh\sqrt{2\pi x}}$, $\left(sh\,x=\frac{e^x-e^{-x}}{2}\right)$	0	1,2	0,12
16	$\sqrt{ch\frac{x}{\sqrt{2\pi}}}$, $\left(ch\,x=\frac{e^x+e^{-x}}{2}\right)$	0,5	1,5	0,1
17	$\frac{x^3+2x}{\sqrt{1+e^x}}$	-0,2	0,8	0,1
18	$\sqrt{1+2x^2}\cdot\sin\frac{3x}{2}$	2	4	0,2
19	$\sqrt{3x^2+5}\cdot\cos\frac{\pi x}{2}$	0,5	1,5	0,1
20	$\arccos e^{-\sqrt[3]{3x}}$	0,2	0,5	0,03
21	$\arcsin e^{-x^2/5}$	8	13	0,5
22	$x+\ln\left(x+\sqrt{1+x^2}\right)$	-0,5	0,5	0,1
23	$\frac{1+e^{-x/2}}{\sqrt{3x^2+1}}$	3	5	0,2
24	$3x^3+\frac{1}{x}+e^{-2x^2}$	1,2	2,2	0,1
25	$x^{2x+1}+x^3-2x$	1	5	0,4

1.3.9. Вопросы

1. Как вводятся векторы в языке MatLAB? Какими функциями можно формировать векторы в языке MatLAB?

2. Какие функции MatLAB разрешают преобразовывать вектор поэлементно?

3. С помощью каких средств в MatLAB осуществляются основные операции с векторами?

4. Как вводятся матрицы в системе MatLAB?

5. Какие функции имеются в MatLAB для формирования матриц определенного вида?

6. Как сформировать матрицу: а) по заданным векторам ее строк? б) по заданным векторам ее столбцов? в) по заданным векторам ее диагоналей?

7. Какие функции поэлементного преобразования матрицы есть в MatLAB?

8. Как осуществляются в MatLAB обычные матричные операции?

9. Как решить в MatLAB систему линейных алгебраических уравнений?

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 системе MatLAB полином задается и сохраняется в виде вектора, элементами которого являются коэффициенты полинома от a_n до a_0 в указанном порядке:

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

Введение полинома в MatLAB осуществляется так же, 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 этого 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 таким образом, что будет выполненное соотношение

B = conv(A,Q) + R.

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

Пусть нужно 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 коэффициентов 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 *получаемый вектор не показывает старшего коэффициента, который по умолчанию полагается равным единице*.

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

p = poly(A)

формирует вектор ***p*** коэффициентов характеристического полинома

$p(s) = \det(s*E - A) = p1*sn + ... + pn*s + pn_{+1}$,

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 осуществляется 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 в качестве аргумента полинома указана матрица X, то функция ***polyval(p,X)*** вычисляет матрицу Y, каждый элемент которой является значением

указанного полинома при значении аргумента, 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 значение полинома для каждого элемента матрицы X, и поэтому размеры исходной и конечной матриц одинаковы size(Y) = size(X).

Вычисление производной от полинома осуществляется 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***, элементы которого являются коэффициентами полинома-производной от произведения двух полиномов ***p1*** и ***p2***:

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

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

Допустим, что есть некоторая зависимость ***y(x),*** заданная 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 машины и использовать потом при дальнейших вычислениях данные о числе строк ***n*** и 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 =  2. 0000   5. 5000   4. 0000   6. 3000   6. 0000   6. 8000  8. 0000   8. 0000  10. 0000   8. 6000
»  size(v')
ans =    1   10
```

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

Функция сортировки ***sort***(V) формирует вектор, элементы которого расположены 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 предшествующих элементов вектора V (вектор кумулятивной суммы).

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

Функция ***diff***(V) создает вектор, который имеет размер на единицу меньший размера 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 об индексе первого 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 соседние точки, заданные векторами x і y, соединены отрезками прямых. Если первый вектор x не указан в обращении, по умолчанию допускается, что шаг интегрирования равняется единице (т. е. вектор x представляет собой вектор 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 ведутся не по отношению к строкам матриц, а к каждому из столбцов заданной матрицы. Т. е. каждый столбец матрицы A рассматривается как переменная, а каждая строка - как отдельное наблюдение. Так, в результате применения функций ***max, min, mean, std*** получаются векторы-строки с количеством элементов, которое равняется количеству столбцов заданной матрицы. Каждый элемент содержит, соответственно, максимальные, минимальное, среднее или среднеквадратичное значения элементов соответствующего столбца заданной матрицы.

Приведем примеры. Пусть имеем 3 величины y1, y2 и y3, измеренные при некоторых пяти значениях аргумента (они 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 и 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 измерений описанные 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 номерах строк, где находятся в соответствующем столбце первые элементы 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 любого из столбцов матрицы. Результатом является матрица того же размера.

Функция ***sum*** и ***prod*** формируют вектор-строку, каждый элемент которого 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 начальной матрицы, начиная с соответствующего элемента и выше.

Наконец, функция ***diff*** создает из заданной матрицы размером (m*n) матрицу размером ((m-1)*n), элементы которой являются разностью между элементами соседних строк начальной матрицы.

Применяя эти процедуры к принятой 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 получают квадратную симметричную матрицу с количеством строк и столбцов, рав-

ным количеству измеренных величин, т. е. количеству 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) таким 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 разложения матриц, вычисление собственных значений и векторов, сингулярных чисел матриц, вычисление функций от матриц. Ознакомимся с некоторыми основными функциями MatLAB в этой области.

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

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

k = ***sum***(***abs***(v) . p)(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, где p = 1,2, 'fro' или inf. Если аргумент p не указан, вычисляется 2-норма. При этом справедливы такие соотношения:

norm(A,1) = max(sum(abs(A)));

norm(A,inf) = max(sum(abs(A')));

*norm(A,'fro') = sqrt(sum(diag(A'*A)));*

norm(A) = norm(A,2) = $\sigma_{max}(A)$.

Функция ***rd = rcond***(A) вычисляет величину, обратную значению числа обусловленности матрицы A относительно 1-нормы. Если матрица A хорошо обу-

словлена, значение ***rd*** близко к единице. Если же она плохо обусловленная, ***rd*** приближается к нулю.

Функция ***r = rank***(A) вычисляет ранг матрицы, который определяется как количество сингулярных чисел матрицы, превышающие 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, равный сумме ее диагональных элементов.

Q = null(A) вычисляет ортонормированный *базис нуль-пространства* матрицы A.

Q = orth(A) выдает *ортонормированный базис матрицы* A.

Процедура R = ***rref***(A) осуществляет *приведение матрицы 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) осуществляет ***разложение Холецького*** для действительных симметричных и комплексных эрмитовых матриц. Например:

```
» 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 вытекает, что в первом, упрощенном варианте обращения 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; 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 равнозначна обычному обращению.

Процедура [Q, R, P] = ***qr***(A) осуществляет разложение матрицы A на три - унитарную матрицу Q, верхнюю треугольную R с диагональными элементами, уменьшающимися по модулю, 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 с помощью функции ***poly***(A). Обращение к ней вида *p=**poly**(A)* дает возможность найти вектор-строку *p* коэффициентов характеристического полинома

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

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, которые удовлетворяют условию

A*R = R * D.

Эти векторы являются нормированными так, что норма любого 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 осуществляет 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, которая состоит 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, которая содержит *неотрицательные значения квадратных корней из собственных значений матрицы* A^T*A (их *называют сингулярными числами*). Эти матрицы удовлетворяют соотношению:

A = U * S * VT.

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 процедурой ***hess***(A). Например:

```
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; P' * P = eye(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) предназначена для ***приведения матрицы 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'; U' * U = eye(size(A)).

Если начальная 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; Q * B * Z = BB.

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

A * V * diag(BB) = B * V * diag(AA).

Необходимость в одновременном приведении пара 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 обычных дифференциальных уравнений в неявной форме Коши с одним входом u и одним выходом y такого вида:

$$\boldsymbol{Q}\cdot\dot{\boldsymbol{x}}+\boldsymbol{R}\cdot\boldsymbol{x}=\boldsymbol{b}\cdot u;$$

$$y=\boldsymbol{c}\cdot x+d\cdot u$$

причем матрицы $\boldsymbol{Q}$, $\boldsymbol{R}$ и векторы $\boldsymbol{b}$, $\boldsymbol{c}$ и $\boldsymbol{d}$ равны соответственно

```
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, которые удовлетворяют 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, которая приводит 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

$$(A0 + \lambda * A1 + ... + \lambda^{p} * Ap) * r = 0.$$

Входными переменными этой процедуры являются p+1 квадратные матрицы A0, A1, ... Ap порядка n. Исходными 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 = [pn, pn-1, ... , p0] 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 значение полинома для любого 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

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) из коэффициентов аппроксимирующего полинома.

Пусть массив значений аргумента имеет вид:

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

а массив соответствующих значений измеренной величины - вид:

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, що заданную зависимость можно аппроксимировать или прямой

$$y(x) = 0{,}1143x - 0{,}2393 \ ,$$

или квадратной параболой

$$y(x) = -0{,}1024x^2 + 1{,}0357x - 1{,}775 \ ,$$

или кубической параболой

$$y(x) = 0{,}0177x^3 - 0{,}341x^2 + 1{,}9461x - 2{,}65 \ ,$$

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);
x1 = 0.5 : 0.05 : 8.5;
y1=polyval(P1,x1);
y2=polyval(P2,x1);
y3=polyval(P3,x1);
y4=polyval(P4,x1);
hold on
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.

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

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

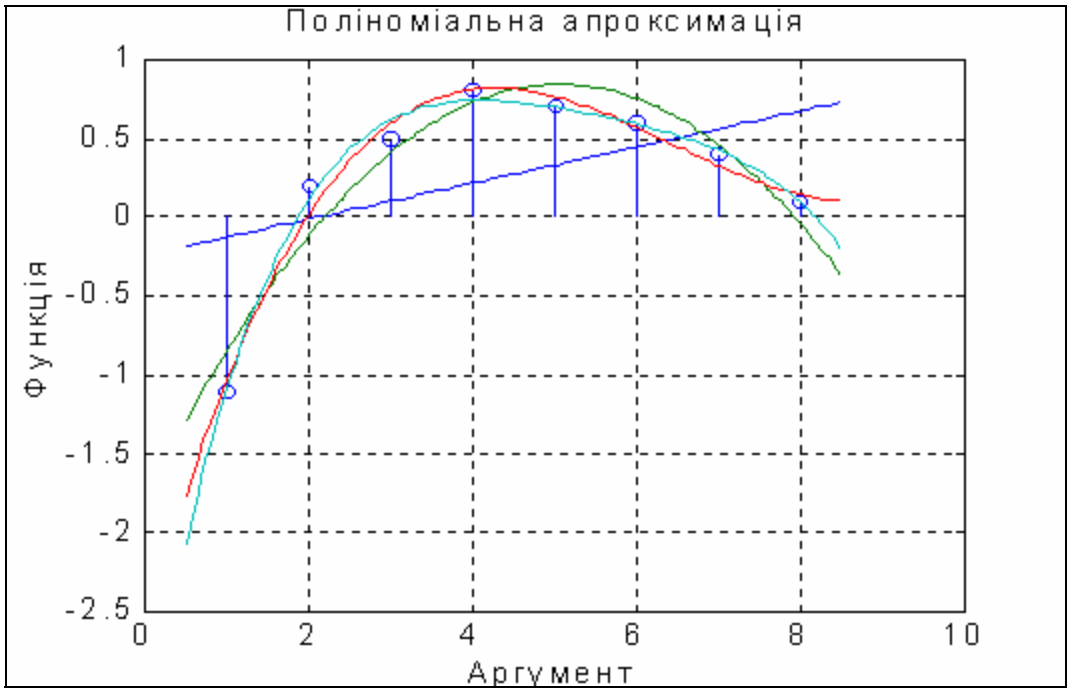

Рис. 1.18

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

В качестве примера 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.

Одномерную табличную интерполяцию осуществляет 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.

Рис. 1.19

Если метод не указан, осуществляется по умолчанию линейная интерполяция. Например, (для одного и 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
plot (x1,y1,x1,y2,'.',x1,y3,'--',x1,y4,':'), grid
set(gca,'FontName','Arial Cyr','FontSize',8),
legend('линейная','кубическая','сплайновая','ступенчатая',0)
set(gca,'FontName','Arial Cyr','FontSize',14),
set(gcf,'color','white')
title('Интерполяция процедурой INTERP1      ');
xlabel('Аргумент');
ylabel('Функция')
```

Результаты приведены на рис.1.20.

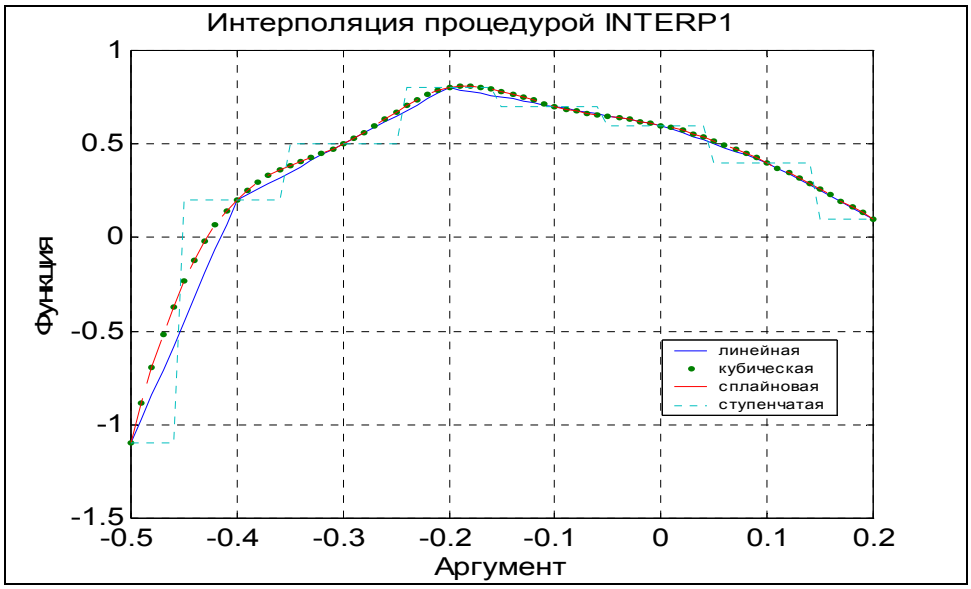

Рис. 1.20

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:

$$y(k) = b(1)*x(k) + b(2)*x(k-1) + \ldots + b(nb+1)*x(k-nb) - a(2)*y(k-1) - a(3)*y(k-3) - \ldots - a(na+1)*y(k-na), \quad (1.1)$$

где вектор ***b*** имеет такой состав

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

а вектор ***a***

$$\boldsymbol{a} = [1, a(2), a(3), \ldots , a(na+1)].$$

Соотношение (1) можно рассматривать как конечно-разностное уравнение фильтра с дискретной передаточной функцией вида рациональной дроби, коэффициенты числителя которого образовывают 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 значение исходного 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 7
        0    0.1000    0.3900    0.2610   -0.5861    0.3146    3.9129
  Columns 8 through 11
    0.2503  -13.4768    2.8466   56.4225
```

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

Обращение к этим функциям вида:

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

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

$$y(k) = \sum_{m=1}^{n} x(m) \cdot e^{-j \cdot 2\pi \cdot (m-1) \cdot (k-1)/n} \; ; \qquad (1.2)$$

$$x(m) = \frac{1}{n} \sum_{k=1}^{n} y(k) \cdot e^{j \cdot 2\pi \cdot (m-1) \cdot (k-1)/n} \; , \qquad (1.3)$$

где j - обозначение мнимой единицы; n - число элементов заданного вектора ***x*** (оно представляет также размер выходного вектора ***y***).

Приведем пример. Сформируем входной сигнал в виде вектора, элементы которого равняются значениям функции, являющейся суммой двух синусоид с частотами 5 и 12 Гц (рис. 1.21).

```
 t =0:0.001:2;
 x = sin(2*pi*5*t) + cos(2*pi*12*t);
plot(t, x); grid
set(gcf,'color','white')
set(gca,'FontName','Arial Cyr','FontSize',16),
title('Входной процесс ');
xlabel('Время (с)');
ylabel('X(t)')
```

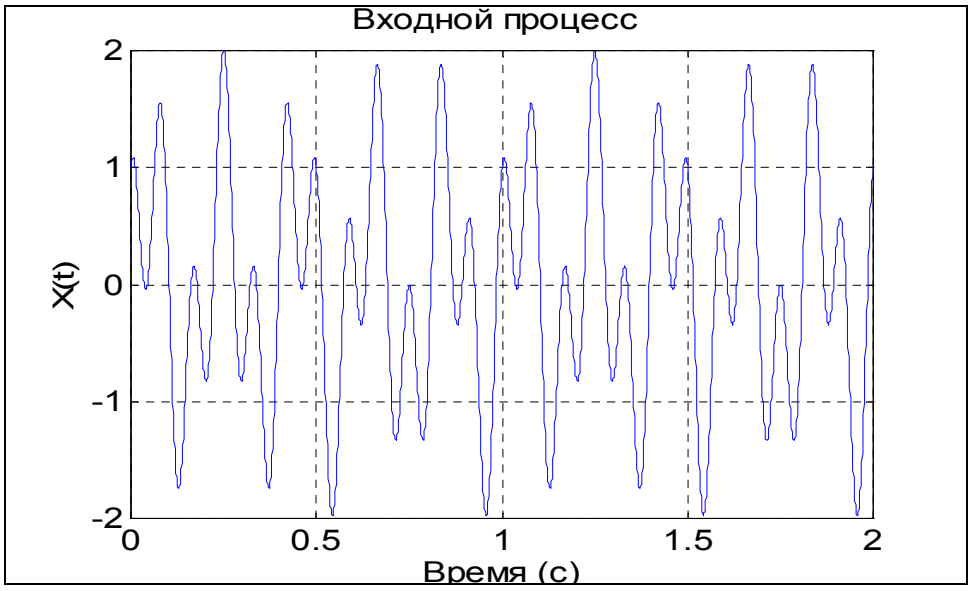

Рис. 1.21

Найдем Фурье-изображение этого сигнала и выведем графическое представление модуля його Фурье-изображения:

```
y = fft(x);
```

```
a =abs(y);
plot(a); grid
 set(gca,'FontName','Arial Cyr','FontSize',16),
title('Модуль Фурьє - изображения ');
xlabel('Номер элемента вектора');
ylabel('abs(F(X(t))')
```

Результат отображен на рис. 1.22.

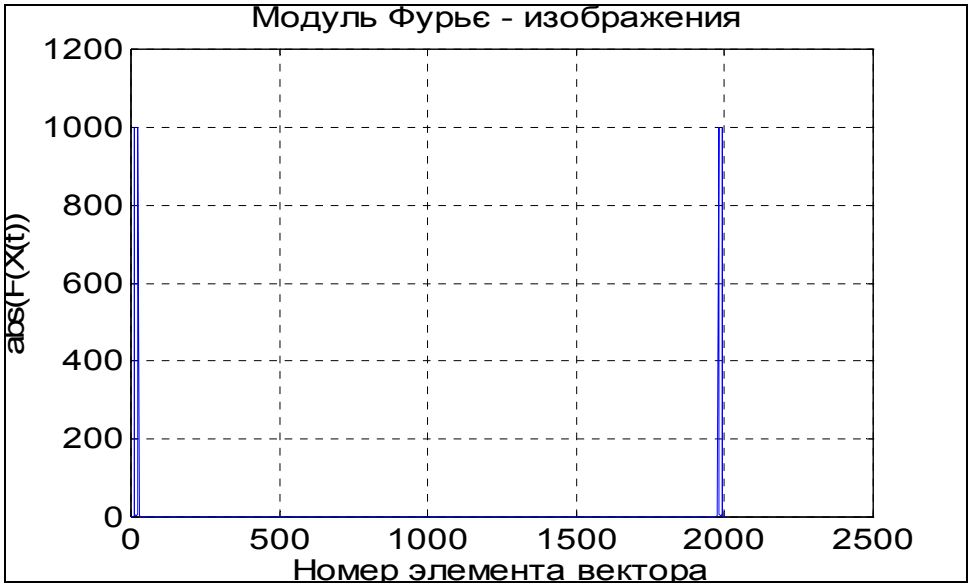

Рис. 1.22

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 точно совпадает с исходным.

Внимательно изучая формулу дискретного преобразования Фурье, можно заметить:

а) номер m отвечает 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 отвечает найденный элемент $y(k)$ дискретного преобразования Фурье;

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

$$Df = 1/T, \qquad (1.4)$$

а диапазон изменения частоты - формулой

$$F = 1/h; \qquad (1.5)$$

так, в анализируемом примере (h =0. 001, T = 2, n = 21):

Df = 0.5; F = 1000;

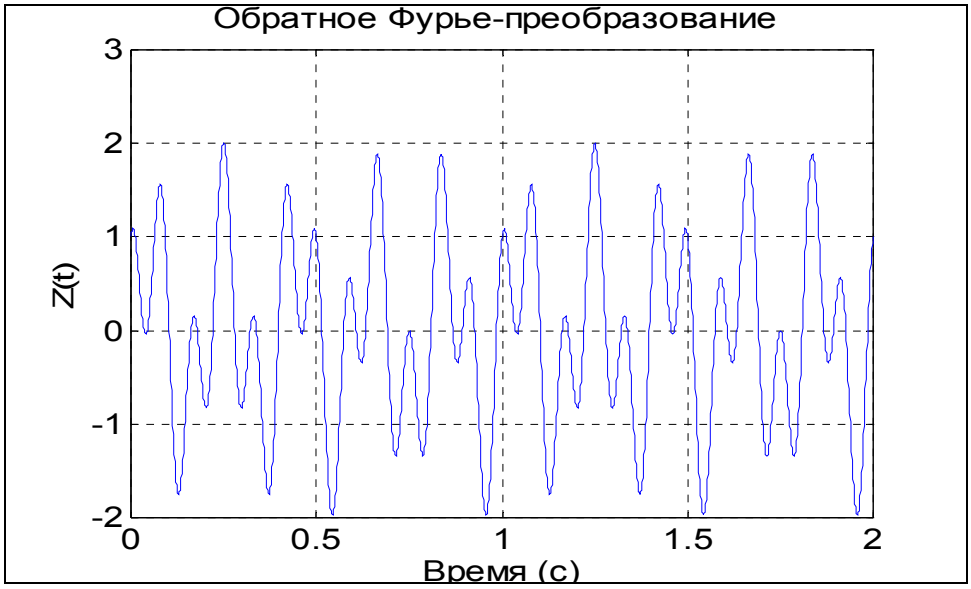

Рис. 1.23

г) из (2) следует, что индексу k = 1 отвечает 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 получаем, что вектор *y(k)* содержит значение Фурье-изображения, начиная из частоты $f_0 = 0$ (которой отвечает k = 1) до максимальной частоты *fmax* = F (которой отвечает $k = n$); таким образом, Фурье-изображение определяется функцией ***fft*** только для положительных частот в диапазоне от 0 к F; это неудобно для построения графиков Фурье-изображения от частоты; более удобным и привычным представляется переход к вектору Фурье-изображения, определенному в диапазоне частот от (-F/2) до F/2; частота $F_N = F / 2$ получила название *частоты Найквиста*;

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

Сформируем для анализируемого примера массив частот, исходя из выше-упомянутого:

```
f = 0 : 0.5 : 1000;
```

и выведем графік с аргументом-частотой (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))')
```

Как следует из рассмотрения рис. 1.24, по нему непросто распознать те частоты (5 и 12 Гц), с которыми изменяется входной сигнал. Это - следствие того обстоятельства, которое было отмечено в примечании г). Чтобы определить частотный спектр входного сигнала, нужно сначала преобразовать полученный вектор ***y*** Фурье-изображения с помощью процедуры ***fftshift.***

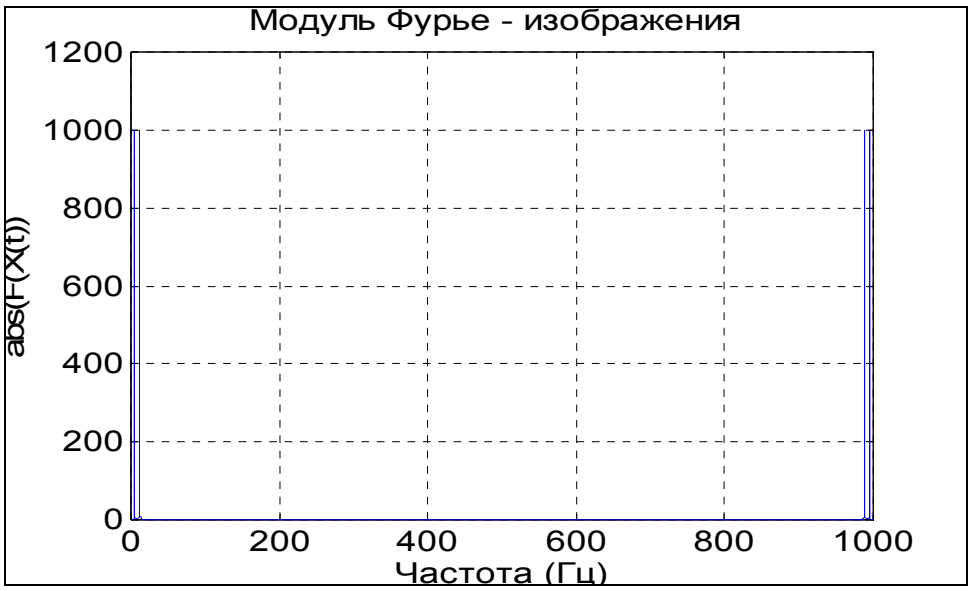

Рис. 1.24

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 задать соотношениями:

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

Примечание. Операцию ***fftshift*** удобно использовать для *преобразования массива Фурье-изображение с целью построения его графика в частотной области. Тем не менее этот массив не может быть использован для обратного преобразования Фурье*.

Проиллюстрируем применение этой функции к предыдущему примеру:

```
f1 = -500 : 0.5 : 500;    % Перестройка вектора частот
v = fftshift(y); % Перестройка вектора Фурье-изображения
a =abs(v); % Отыскание модуля
  % Вывод 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))')
```

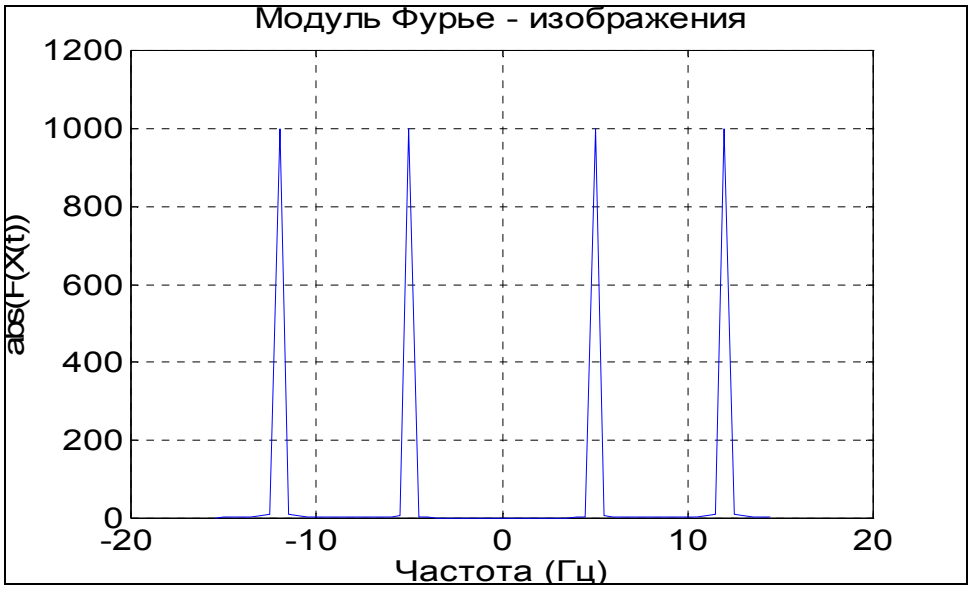

Рис. 1.25

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

Остается лишь то неудобство, что из графика спектра невозможно установить амплитуды этих гармоник. Во избежание этого, нужно весь вектор ***y*** Фурье-изображения разделить на число его элементов (***n***), чтобы получить вектор комплексного спектра сигнала:

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

Результат приведен на рис. 1.26. Из его рассмотрения вытекает, что "амплитуды" всех составляющих гармоник равны 0.5. При этом нужно принять во внимание, что "амплитуды" распределены между положительными и отрицательными частотами поровну, поэтому они вдвое меньшее действительной амплитуды соответствующей гармоники.

1.4.6. Задания

Задание 1.6.

1. Введите произвольную матрицу размером (4*6). Найдите сумму наибольших элементов ее строк.

2. Введите квадратную матрицу (5*5) с одним наименьшим элементом. Найдите сумму элементов строки, в которой размещен элемент с наименьшим значением.

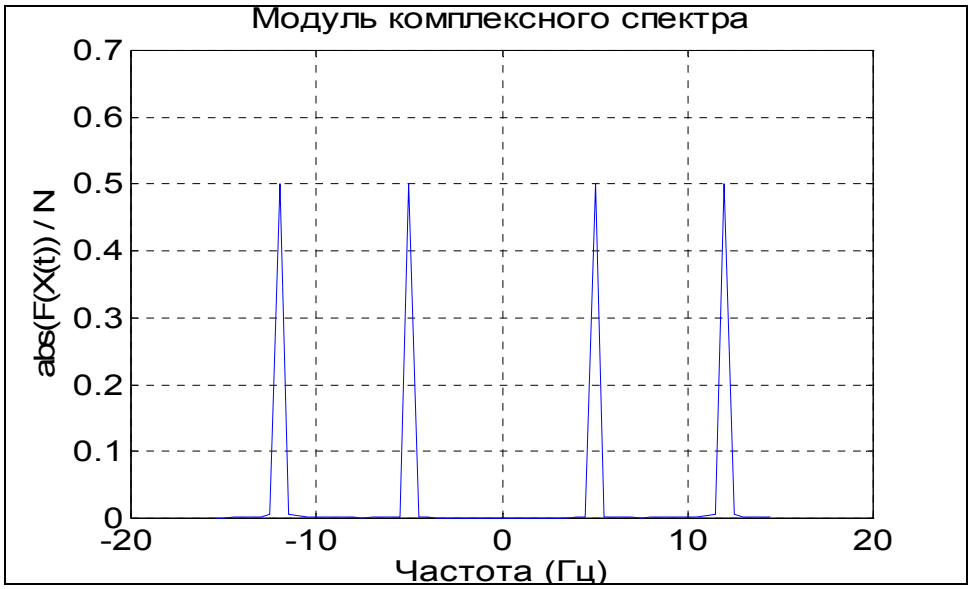

Рис. 1.26

3. Введите матрицу (6*9), в которой есть единственные наибольший и наименьшие элементы и они расположены в разных строках. Поменяйте местами строку с наибольшим элементом и строку с наименьшим элементом.

4. Введите матрицу (5*6) с разными значениями элементов. В каждой строке выберите элемент с наименьшим значением, из полученных чисел выберите наибольшее. Найдите индексы полученных элементов.

5. Введите матрицу (5*6). Найдите вектор, элементами которого являются наибольшие элементы соответствующей строки матрицы.

6. Введите матрицу (5*6). Постройте вектор, элементами которого являются суммы наибольшего и наименьшего элементов соответствующей строки матрицы.

7. Введите матрицу (5*6). Постройте вектор, элементами которого являются средние значения элементов соответствующей строки матрицы.

8. Введите матрицу (5*6). Постройте вектор, элементами которого являются среднеквадратичные отклонения элементов соответствующей строки матрицы от их среднего значения.

9. Введите матрицу (5*6). Постройте вектор, элементами которого являются средние арифметические наибольшего и наименьшего элементов соответствующей строки матрицы.

10. Введите матрицу (6*5). Постройте вектор, элементами которого являются суммы квадратов элементов соответствующего столбца матрицы.

11. Введите матрицу (5*5). Постройте векторы, элементами которых являются суммы элементов столбцов матрицы, произведения элементов столбцов и наименьшие значения элементов столбцов.

12. Введите матрицу (5*6). Найдите среднее арифметическое наибольших и наименьших ее элементов.

13. Введите матрицу (5*5). Постройте вектор, элементами которого являются элементы главной диагонали матрицы. Найдите след матрицы.

14. Введите две матрицы (4*4). Постройте новую матрицу размером (4*8), включая в первые 4 столбца строки первой матрицы, а в другие - столбцы второй матрицы.

15. Найдите сумму всех элементов матрицы размером (4*3).

Задание 1.7. Вычислите векторы:

а) модуля частотной передаточной функции (ЧПФ);

б) аргумента ЧПФ;

в) действительной части ЧПФ;

г) мнимой части ЧПФ

по заданным числителю и знаменателю передаточной функции (таблица 1.4).

Предварительно найдите корни знаменателя передаточной функции, определите наибольшую собственную частоту ω_{max} системы. Обеспечьте вычисление ЧПФ при 100 значениях частоты ω в диапазоне от 0 до $5\omega_{max}$.

Таблица 1.4

Вариант	Числитель	Знаменатель
1	1. 82p+67.56	p^4+2. 65p^3+3. 09p^2+7. 04p+34.05
2	4. 61p^2+1. 82p+67.56	p^4+3. 65p^3+45p^2+7. 04p+125
3	p^2+4p+23	p^4+2p^3+39p^2+2p+45
4	3p^2+1. 82p+67.56	p^2+7. 04p+34.05
5	p+6	p^2+0. 7p+48
6	p^3+4. 61p^2+1. 82p	2. 65p^3+3p^2+4p+87
7	p^3+4. 61p^2+1. 82p+67.56	p^4+2. 65p^3+68p^2+5p+34
8	4. 61p^2+68	p^4+2. 65p^3+3. 09p^2+7. 04p+34.05
9	7.56	p^4+2. 65p^3+3. 09p^2+7. 04p+34.05
10	p^3+1. 8p+7	p^4+6. 5p^3+39p^2+7p+45
11	p^3+4. 61p^2+1. 82p+67.56	p^3+3. 09p^2+70p+34
12	p^2+1. 8p+78	2. 65p^3+3. 09p^2+7. 04p+34.05
13	p^3+1. 82p+67.56	p^4+2. 6p^3+3p^2+4p+34
14	p^3+4. 61p^2+1. 82p+67.56	7p^2+7p+34
15	4. 61p^2+1. 82p+67.56	p^2+7. 04p+560
16	1. 82p+67.56	3. 09p^2+7. 04p+34.05
17	p^3	3. 09p^2+7. 8p+125
18	1. 82p	p^3+3. 09p^2+7. 04p+34.05
19	4. 61p^2	p^2+7. 04p+34.05
20	p^3+67.56	p^4+2. 65p^3+3. 09p^2+7. 04p+34.05
21	p^3	p^4+2p^3+3p^2+12p+100
22	p^3+4. 61p^{2+}1. 82p+67.56	p^4+5p^3+30p^2+7p+305
23	p^2+1. 82p+67.56	p^4+2p^3+9p^2+4p+35
24	p^{3+}61p^2+182p+67	p^4+3p^3+9p^2+0. 04p+39

25	$p^2+1.82p+67.56$	$p^4+5p^3+20p^2+7p+34$

Указание. *Частотной передаточной функцией называют передаточную функцию системы при мнимых значениях її аргумента ($p = j \cdot \omega$).*

Собственные частоты системы - это значения модулей мнимых частей корней характеристического уравнения системы (которое получается приравниванием нулю знаменателя передаточной функции).

Задача 1.8. Введите произвольную матрицу размером (5*5). Найдите:

1) определитель матрицы; в случае, если определитель окажется равным нулю, или слишком малым, измените некоторые элементы матрицы и повторите вычисления;
2) обратную матрицу; проверьте правильность путем обращения обратной матрицы;
3) характеристический полином матрицы;
4) корни характеристического полинома матрицы; рассортируйте корни по комплексно-спряженным парам и в порядке возрастания величин;
5) собственные значения матрицы; сравните с ранее найденными корнями характеристического полинома;
6) LU-разложение матрицы; проверьте его правильность;
7) QR-разложение матрицы; проверьте его правильность;
8) сингулярные числа матрицы; сравните их с получаемыми при ***svd***-разложении;
9) след матрицы;
10) число обусловленности матрицы;
11) экспоненту от матрицы;
12) логарифм от экспоненты матрицы; сравните с исходной матрицей.

1.4.7. Вопросы

1. Какой объект в MatLAB называется полиномом?
2. Как в MatLAB осуществляется перемножение и деление полиномов?
3. При помощи каких функций можно найти корни заданного полинома, значение полинома по известному значению аргумента?
4. Какие функции позволяют найти производную от полинома?
5. Как найти характеристический полином матрицы?

1.5. Построение простейших графиков

1.5.1. Процедура plot

Вывод графиков в системе MatLAB - настолько простая и удобная операция, что ее можно использовать даже в режиме калькулятора.

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

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

Здесь x1, y1 - заданные векторы, элементами которых являются массивы значений аргумента (x1) и функции (y1), отвечающие 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 является обязательным). Любая из них может содержать до трех специальных символов, определяющих соответственно: а) тип линии, которая соединяет отдельные точки графика; б) тип точки графика; в) цвет линии. Если переменные s не указаны, то тип линии по умолчанию - отрезок прямой, тип точки - пиксел, а цвет устанавливается (в версии 5.3) в такой очередности: - *синий, зеленый, красный, голубой, фиолетовый, желтый, черный и белый* - в зависимости от того, какая по очереди линия выводится на график. 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 отдельное графическое окно*, которое называют *фигурой*.

Приведем пример. Пусть нужно 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,
```

потом вычислить массив соответствующих значений функции:

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

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

В командном окне 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 окно с графиком (см. рис. 1.27).

*Если вектор аргумента при обращении к функции **plot** не указан явно, то система по умолчанию принимает в качестве аргумента номера элементов вектора функции.* Например, если ввести команду

» plot(y),

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

Графики, приведенные на рис. 1.27, 1.28, имеют несколько недостатков:

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

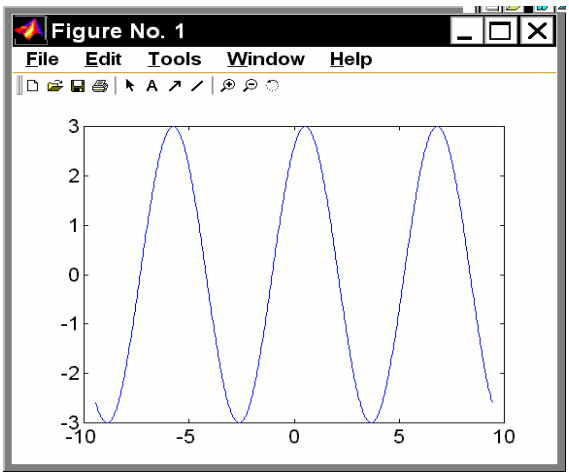

Рис. 1.27

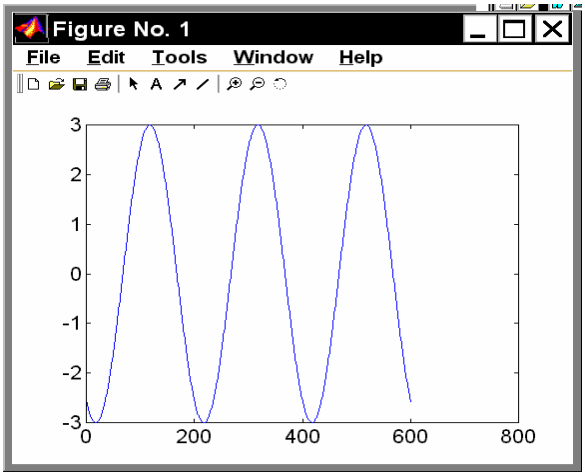

Рис. 1.28

Первый недостаток устраняется с помощью функции ***grid***. Если к этой функции обратиться 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 соответствует "целым" шагам изменения*, что делает графики "читабельными", т. е. такими, что по графику можно отсчитывать значение функции при любом заданном значении аргумента и наоборот.

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

title(‘<текст>’),

то над графиком появится текст, записанный между апострофами в скобках. При этом следует помнить, что *текст всегда должен помещаться в апострофы*.

Аналогично можно вывести объяснения 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');
```

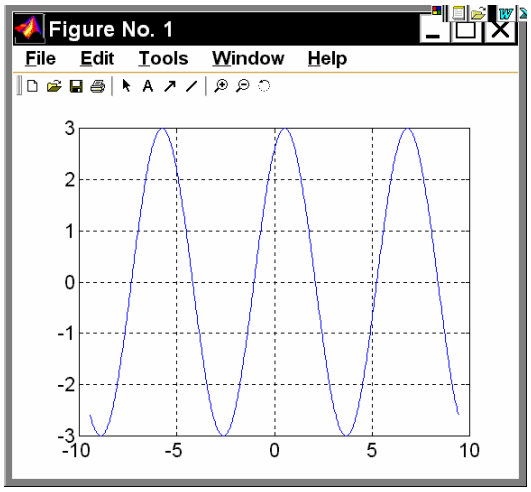

Рис. 1.29

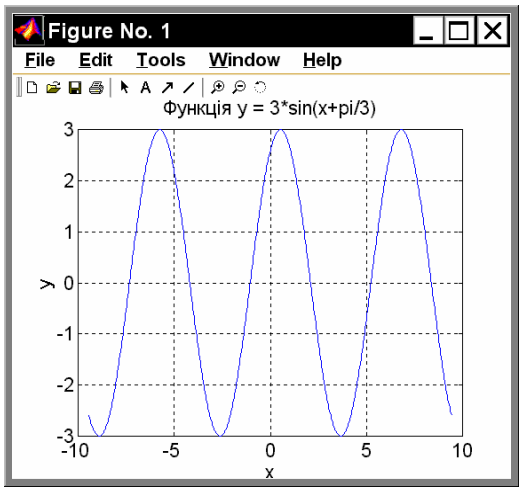

Рис. 1.30

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

Не сложнее вывод в среде MatLAB графиков функций, заданных *параметрически*. Пусть, например, необходимо построить график функции y(x), заданной формулами:

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

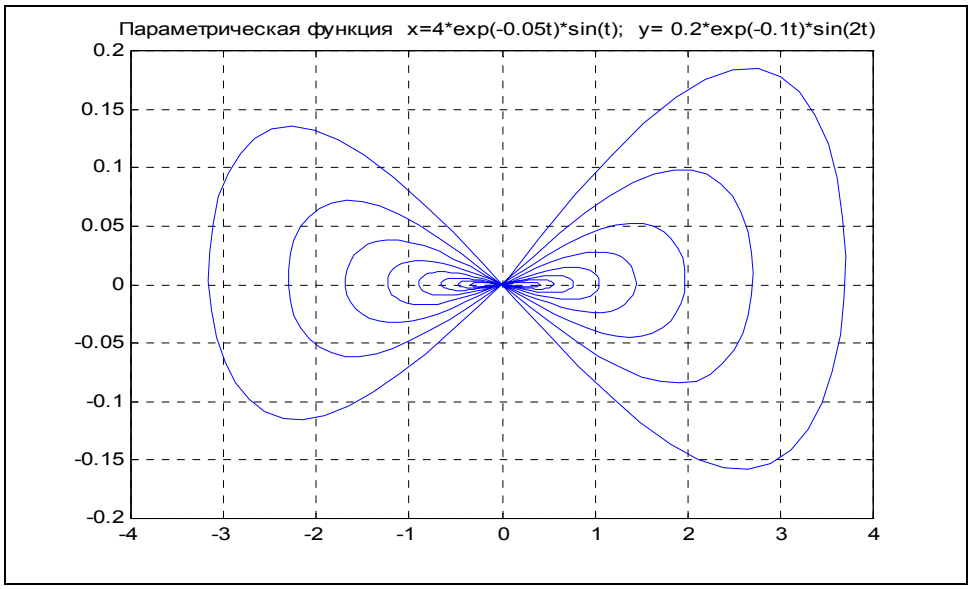

Рис. 1.31

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)
set(gcf,'color','white')
title('Параметрическая функция  x=4*exp(-0.05t)*sin(t);  y= 0.2*exp(-0.1t)*sin(2t) ')
```

grid,
получим график рис. 1.31.

1.5.2. Специальные графики

Большим удобством, предоставляемым системой MatLAB, является указанная 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, например, можно построить "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.

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

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

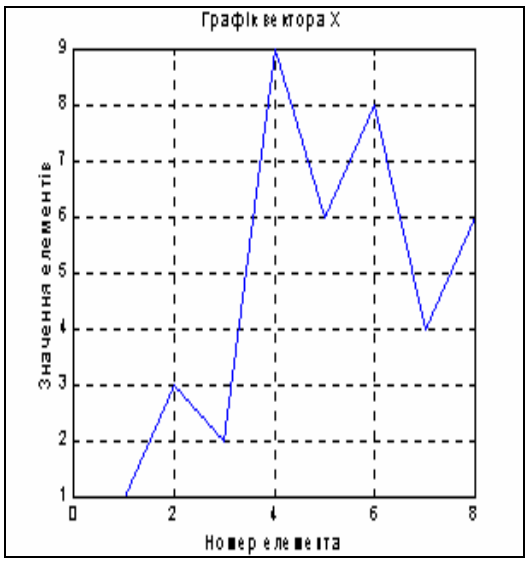

Рис. 1.32

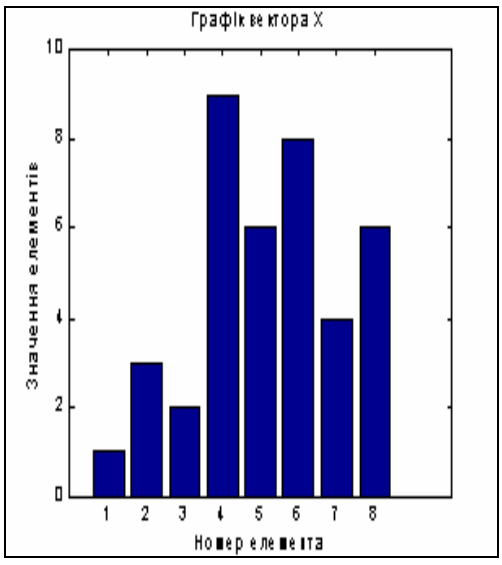

Рис 1.33

Если функция задана своими значениями при дискретных значениях аргумента, и неизвестно, как она может изменяться в промежутках между значениями аргумента, удобнее представлять график ее в виде отдельных вертикальных линий для любого из заданных значений аргумента. Это можно сделать, применяя процедуру ***stem***, обращение к которой целиком 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 векторы X')
ylabel('Значение элементов')
xlabel(' Номер элемента')
```

На рис. 1.34 изображен полученный при этом график.

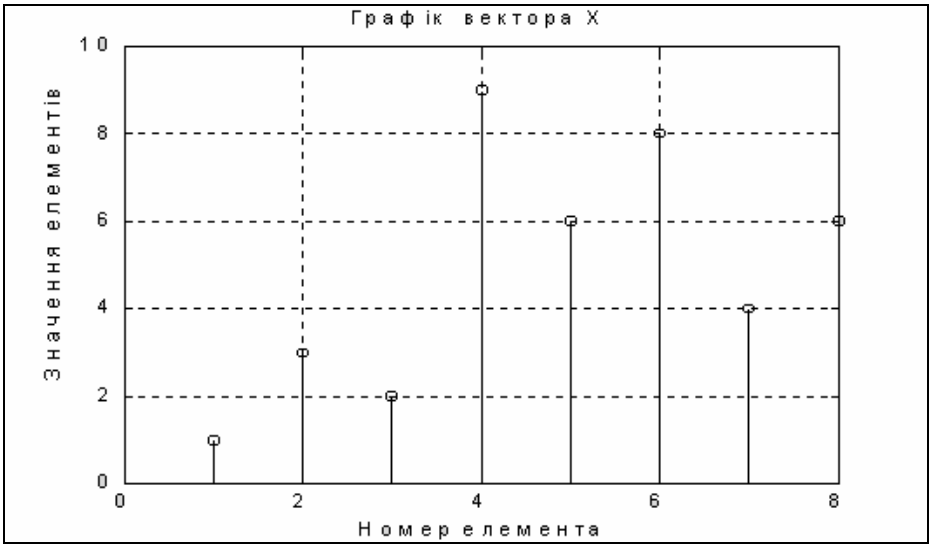

Рис. 1.34

Другой пример - построение графика функции $y = e^{-x^2}$ в виде столбцовой диаграммы (рис. 1.35):

```
» x = - 2.9 : 0.2 : 2.9;
» bar(x, exp(-x . * x))
» title('Стовпцева  диаграмма  функции  y = exp(-x2)')
» xlabel (' Аргумент  x')
» ylabel (' Значение функции  y')
```

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

hist(y, x),

где y - вектор, гистограмму которого нужно построить; x - вектор, элементы его определяют интервалы изменения первого вектора, внутри которых подсчитывается количество элементов вектора “y”.

Эта функция осуществляет две операции:

- подсчитывает количество элементов вектора “y”, значения которых попадают внутрь соответствующего диапазона, указанного вектором “x”;

- строит столбцовую диаграмму подсчитанных чисел элементов вектора “y” как функцию диапазонов, указанных вектором “x”.

В качестве примера рассмотрим построение гистограммы случайных величин, которые формируются встроенной функцией ***randn***. Примем общее количество элементов вектора этих случайных величин 10 000. Построим гистограмму для диапазона изменения этих величин от -2,9 до +2,9. Интервалы изменеия пусть

будут равны 0,1. Тогда график гистограммы можно построить с помощью совокупности таких операторов:

```
x = -2.9:0.1:2.9;
y = randn(10000,1);
hist(y,x)
set(gca,'fontsize',14)
ylabel('Количество из 10000')
xlabel('Аргумент')
title('Гистограмма нормального распределения')
```

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

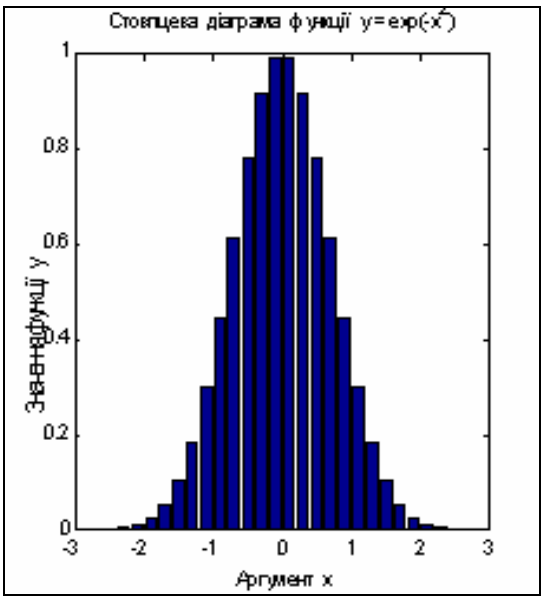

Рис. 1.35

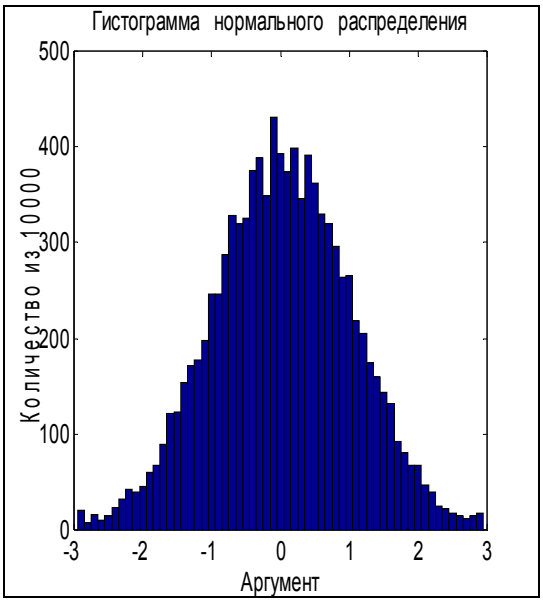

Рис. 1.36

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 с течением времени.

MatLAB имеет несколько функций, которые позволяют строить графики в логарифмическом масштабе. К примеру, функция ***logspace*** с обращением

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

формирует вектор-строку "x", содержащую "n" равноотстоящих в логарифмическом масштабе друг от друга значений, 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, которые используют 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 аналогично обращению к функции ***plot***.

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

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

Для этого нужно, во-первых, создать полином числителя Pc = [1 4] и знаменателя передаточной функции Pz = [1 4 100]. 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 измения частоты так, чтобы он охватывал все найденные корни:

om0 = 1e-2; omk = 1e2.

Теперь нужно задаться количеством точек будущего графика (например, n = 41), и сформировать массив точек по частоте в логарифмическом масштабе

OM = logspace(-2,2,41),

где значения -2 и +2 отвечают десятичным порядкам начального om0 и конечного omk значений частоты.

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

Вектор значений АЧХ (амплитудно-частотной характеристики) можно найти, расчитывая модули векторов числителя и знаменателя ЧПФ и поэлементно деля полученные векторы. Чтобы найти вектор значений ФЧХ (фазочастотной характеристики), нужно разделить поэлементно комплекснозначные векторы числителя и знаменателя ЧПФ и определить вектор аргументов элементов полученного

вектора. Для того чтобы фазу представить в градусах, полученные результаты следует домножить 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.***

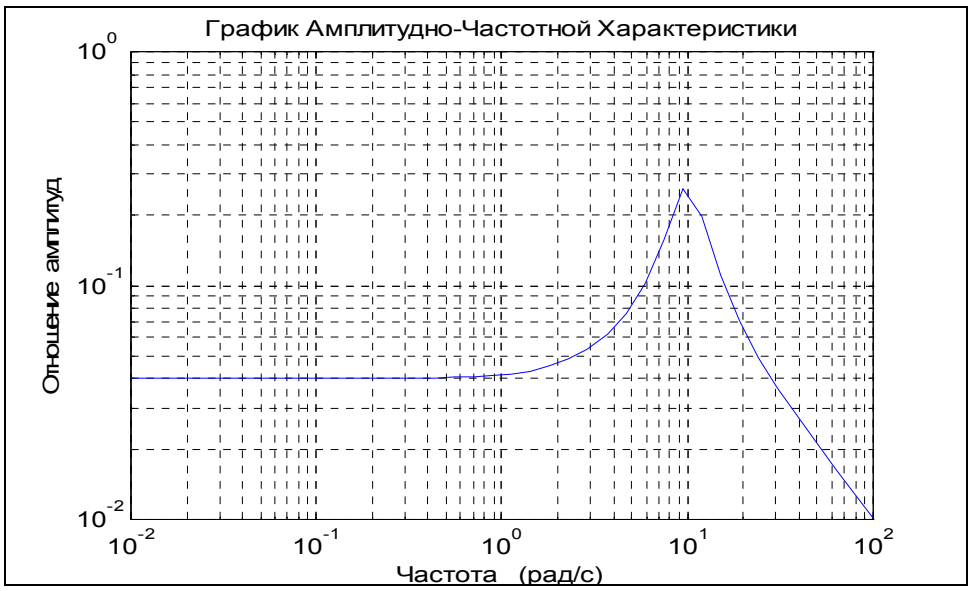

Рис. 1.37

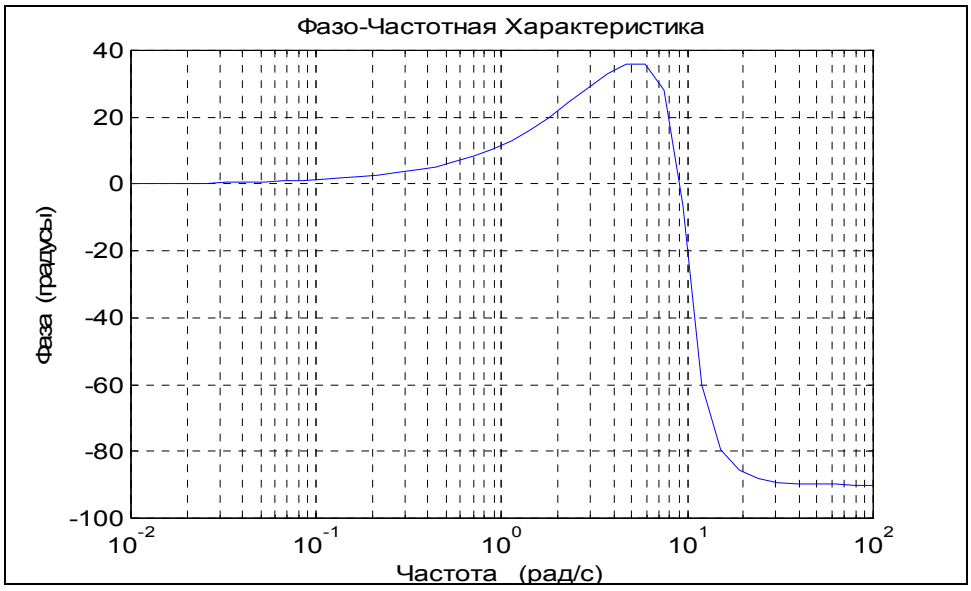

Рис. 1.38

В целом последовательность действий может быть такой:

```
P1 = [1 4];  P2 = [1 4 100];
OM = logspace(-2,2,40);
ch = polyval(P1,i*OM);   zn = polyval(P2,i*OM);
ACH = abs(ch)./abs(zn);
loglog(OM,ACH);grid;    set(gca,'FontSize',12)
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, изображенные на рис. 1.37 и 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 по осям, чтобы в поле графика поместились все вычисленные точки графика, включая максимальные и минимальные значения аргумента и функции. Тем не менее, MatLAB имеет возможности установления 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, которые откладываются по осям.

Команда ***axis***(‘auto') возвращает масштабы по осям к их штатному значению (принятому по умолчанию).

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

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

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

Команда ***axis***(‘equal') обеспечивает одинаковый масштаб по обоих 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 иметь такой вид:

subplot(m,n,p).

Здесь m - указывает, на сколько частей разделяется графическое окно по вертикали, n - по горизонтали, а p - номер подокна, в котором будет строиться график. При этом подокна нумеруются слева направо построчно сверху вниз (так, как по строкам читается текст книги).

Например, два предшествующих графика можно поместить в одно графическое окно следующим образом:

```
subplot(2,1,1);  loglog(OM,ACH,'k'); grid;
set(gca,'FontName','Arial','FontSize',12),
title('Амплитудно-Частотная Характеристика'),  ylabel('Амплитуда'),
subplot(2,1,2);  semilogx(OM,FCH,'k'); grid
title('Фазо-Частотная Характеристика')
xlabel('Частота  (рад/с)'),  ylabel('Фаза (гр.)')
```

Результат представлен на рис. 1.39.

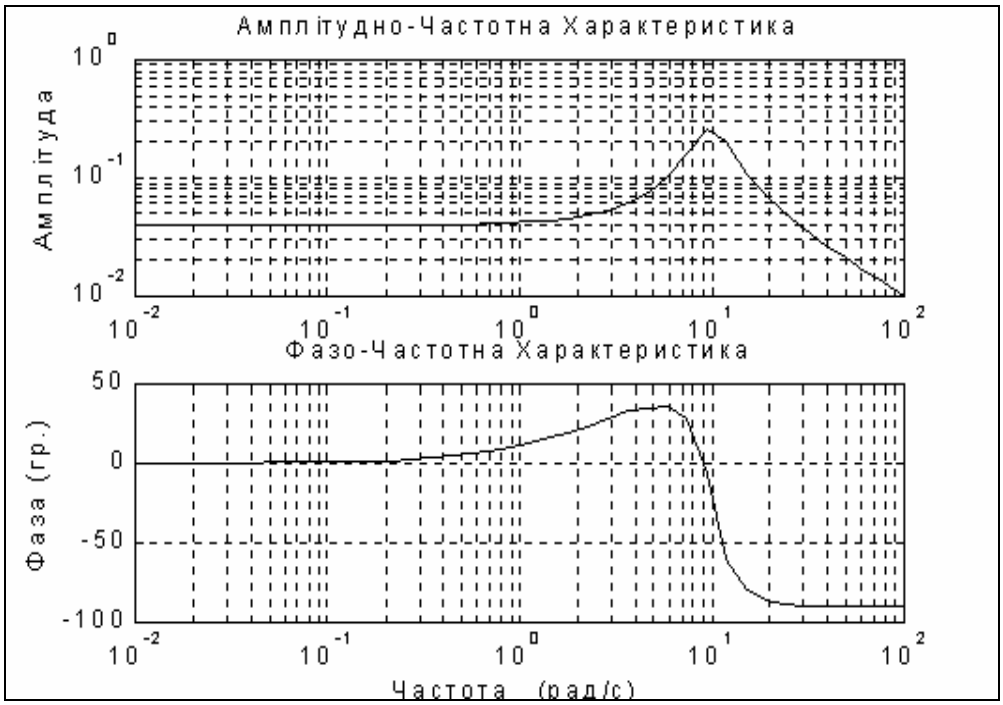

Рис. 1.39

Команда ***text***(x, y, ‘<текст>’) позволяет расположить указанный текст на 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 изменения этих величин. Часто это неудобно, так как требует предварительного знания этого диапазона, что не всегда возможно.

Более удобно для размещения текста внутри поля графика использовать команду ***gtext***(‘<текст>’), которая высвечивает в активном графическом окне перекрестие, перемещение которого с помощью мыши позволяет указать место начала вывода указанного текста. После этого нажатием левой клавиши мыши или любой клавиши текст вводится в указанное место:

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

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

Чтобы создать несколько графических окон, в любом из которых расположены соответствующие графики, можно воспользоваться командой ***figure***, которая создаст такое графическое окно, оставляя предшествующие.

Наконец, для того, чтобы несколько последовательно вычисленных графиков были изображены в одном графическом окне в одном стиле, можно использовать команду ***hold on***, тогда каждый такой график будет строиться в том же предварительно открытом графическом окне, т. е. каждая новая линия будет добавляться к прежде построенным. Команда ***hold off*** выключает режим сохранения графического окна, установленного предшествующей командой.

1.5.4. Вывод графиков на печать

Чтобы вывести график из графического окна (фигуры) на печать, т. е. на лист бумаги, следует воспользоваться командами меню, расположенного в верхней части окна фигуры. В меню ***File*** выберите команду ***Print***. Подготовьте принтер к работе и нажмите кнопку <Ок> в окошке печати, - принтер распечатает содержимое графического окна на отдельном листе бумаги.

Для предварительной настройки на определенный тип принтера и установления вида печати используйте в том же меню ***File*** команду ***Print Setup***.

1.5.5. Задания

Задание 1.9. Постройте в графическом окне MatLAB график функции из *задания 1.5.* Распечатайте этот график на листе бумаги.

Задача 1.10. Постройте в графическом окне MatLAB графики амплитудно-частотной (модуля ЧПФ) и фазочастотной (аргумента ЧПФ) характеристик функции из *задания 1.7.* Распечатайте полученный график на листе бумаги.

1.5.6. Вопросы

1. Какие функции MatLAB осуществляют вывод графиков на экран?
2. Какими функциями обеспечивается снабжение графика координатными линиями и надписями?
3. Что такое "график вектора" и как его построить?
4. Как вывести график в виде столбцовой диаграммы?
5. Как построить гистограмму?
6. Можно ли построить несколько графиков в одной системе координат и в одном графическом окне?
7. Как вывести несколько отдельных графиков в разных графических окнах?
8. Как построить несколько отдельных графиков в одном графическом окне в разных графических полях?

1.6. Операторы управления вычислительным процессом

Вообще говоря, 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. Тем не менее, система MatLAB построена таким образом, что эти операторы могут быть использованы при роботе MatLAB и в режиме калькулятора.

В языке MatLAB отсутствует оператор безусловного перехода и, в соответствии с этим, нет понятию метки. Это обстоятельство затрудняет организацию перехода вычислительного процесса к любому предшествующему или следующему оператору программы.

Все операторы цикла и условного перехода построены в MatLAB в виде составного оператора, который начинается одним из служебных слов ***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> при переходе к следующей строке не приводит в этом случае к выполнению этих операторов. Выполнение операторов начинается лишь тогда, если введена "завершающая скобка" сложного оператора в виде слова ***end***, а затем нажата клавиша <Enter>. Если несколько составных операторов такого типа вложены друг в друга, вычисления начинаются лишь тогда, когда записан конец ***end*** наиболее охватывающего (внешнего) составного оператора. Отсюда следует 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 оператор так. Сначала проверяется, выполняется ли указанное условие. Если оно выполнено, программа выполняет совокупность операторов, которая записанная в делении <операторы1>. Если условие не выполнено, выполняется последовательность операторов <операторы2>.

Сокращенная 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: *все части программы, которые выполняются в зависимости от условия, должны размещаться внутри операторных скобок* ***if*** *и* ***end***.

В качестве условия используются выражения 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. состоять из нескольких простых условий, объединенных знаками логических операций. Знаками логических операций в языке MatLAB являются:

& - логическая операция “И” (“AND”);
| - логическая операция “ИЛИ” (“OR”);
~ - логическая операция “НЕТ” (“NOT”).

Логическая операция “Исключительное ИЛИ” может быть реализована с помощью 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, <операторы2> игнорируются, и происходит переход к следующему оператору ***elseif***, т. е. к проверке выполнения <условия3>. Аналогично, при выполнении его выполняются <операторы3>, в противном случае происходит переход к следующему оператору ***elseif***. Если ни одно из условий в операторах ***elseif*** не выполнено выполняются <операторы>, размещенные за операто-

ром ***else***. Так может быть обеспечено разветвление 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 разветвление 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, которые следуют за словом ***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 выполняются лишь в 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 значений 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 указанном примере обеспечен вывод на экран значений нескольких переменных в одну строку. Для этого используется 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 с помощью обычной операции формирования вектора-строки из отдельных элементов

[x1, x2, ... , x].

Таким образом, с помощью оператора вида:

disp([x1, x2, ... , x])

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***. Когда программа сталкивается с этим оператором, выполнение цикла досрочно прекращается, и начинает выполняться оператор, следующий за словом ***end*** цикла.

Для примера используем предыдущую задачу:

```
» 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
```

В результате получаем

```
    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
```

Так можно обеспечить вывод информации в виде таблиц.

1.6.4. Задания

Задание 1.11.

1. В соответствии с таблицей 1.5 выполнить:

- вычисление точных (используя стандартные функции MatLAB) значений соответствующей функции в диапазоне изменения аргумента от x_1 до x_2 в m равноотстоящих точках этого диапазона, включая его границы;

- вычисление по указанным степенным рядам приближенных значений функции в тех же точках, ограничиваясь r первыми членами ряда;

- расчет погрешности приближенного определения функции в каждой точке, сравнивая приближенное значение с точным, и построение графика зависимости погрешности от аргумента;

- вычисление приближенных значений функции в тех же точках с относительной погрешностью не более ε=0.001; построение графика полученных относительных погрешностей.

Таблица 1.5.

Вар.	x1	x2	m	r	f(x)	Приближенная формула

1	0.2	5	20	4	sin(x)	$\sum(-1)^k \frac{x^{2k-1}}{(2k-1)!}$
2	1	10	30	5	cos(x)	$1-\sum(-1)^k \frac{x^{2k}}{(2k)!}$
3	0.3	3	40	5	exp(x)	$1+\sum\frac{x^k}{k!}$
4	0.4	4	50	4	ln(1+x)	$\sum(-1)^{k-1}\frac{x^k}{k}$
5	0.5	5	30	3	ln(x)	$\sum\frac{a^k}{k}$, где $a=\frac{x-1}{x}$
6	0.6	6	40	4	ln(x)	$\sum(-1)^k\frac{a^k}{k}$, где $a=x-1$
7	0.7	7	50	5	ln(x)	$2\sum\frac{a^{2k-1}}{2k-1}$, где $a=\frac{x-1}{x+1}$
8	0.8	8	45	6	ln(x+a)	$\ln(a)+2\sum\frac{b^{2k-1}}{2k-1}$, где $b=\frac{x}{2a+x}$
9	1.1	11	40	3	ctg(x)	$\frac{1}{x}+2x\sum\frac{1}{x^2-k^2\pi^2}$
10	1.2	12	50	4	cosec(x)	$\sum\frac{1}{(x-k\pi)^2}$
11	1.3	13	50	5	cosec(x)	$\frac{1}{x}+2x\sum\frac{(-1)^k}{x^2+k^2\pi^2}$
12	1.4	14	60	6	arctg(x)	$\sum(-1)^{k-1}\frac{x^{2k-1}}{2k-1}$
13	1.5	15	45	5	arctg(x)	$\frac{\pi}{2}+\sum(-1)^k\frac{1}{(2k-1)x^{2k-1}}$
14	1.6	16	40	4	ln(x)	$\sum(-1)^{k-1}\frac{a^k}{r}$, где $a=x-1$
15	0.9	9	50	6	sin(x)	$x\prod(1-\frac{x^2}{(k\pi)^2})$
16	1	10	50	4	cos(x)	$\prod(1-\frac{x^2}{(2k-1)^2\pi^2})$
17	0.6	5	50	3	ln(x)	$\sum\frac{a^k}{k}$, где $a=(x-1)/x$

18	-0. 9	0.9	45	4	arcctg(x)	$\frac{\pi}{2}-\sum(-1)^k\frac{x^{2k-1}}{(2k-1)}$
19	1	20	50	5	sh(x)	$\sum\frac{x^{2k-1}}{(2k-1)!}$
20	1	20	50	5	ch(x)	$1+\sum\frac{x^{2k}}{(2k)!}$
21	-0. 9	0.9	50	5	arcth(x)	$\sum\frac{x^{2k-1}}{2k-1}$
22	1	20	50	5	arcth(x)	$\sum\frac{1}{(2k-1)x^{2k-1}}$
23	-0. 8	0.8	50	4	arcsin(x0	$x+\sum\frac{1\cdot3\cdot5...(2k-1)\cdot x^{2k+1}}{2\cdot4\cdot6....(2k)\cdot(2k-1)}$
24	-0. 8	0.8	50	4	arccos(x 0	$\frac{\pi}{2}-\left\{\ x+\sum\frac{1\cdot3\cdot5...(2k-1)\cdot x^{2k+1}}{2\cdot4\cdot6....(2k)\cdot(2k-1)}\ \right\}$
25	-5	5	50	6	exp(x)	$1+\sum\frac{x^k}{k!}$

Задание 1.12. Вычислить значения функции из *задачи 1.5* при значениях аргумента в диапазоне от 0.1 до 100, образующих *геометрическую прогрессию* со знаменателем, равным квадратному корню из 10, и выведите в командное окно таблицу результатов вычислений.

Задание 1.13. Вычислить значения модуля ЧПФ и ее аргумента (в градусах) из *задачи 1.7* при значениях аргумента в диапазоне от 0.1 до 100, образующих *геометрическую прогрессию* со знаменателем, равным квадратному корню из 10, и выведите в командное окно таблицу результатов вычислений.

1.6.5. Вопросы

1. Какие средства управления ходом вычислительного процесса предусмотрены в языке MatLAB?
2. Как можно организовать вычисления по циклу в языке MatLAB?
3. Как организовать вывод таблицы результатов вычислений в командное окно MatLAB?
4. Как осуществить сложные (многооператорные) вычисления в режиме калькулятора?

2. Программирование в среде MatLAB

Работа в режиме калькулятора в среде MatLAB, несмотря на довольно значительные возможности, во многих отношениях неудобна. Невозможно повторить предшествующие вычисления и действия при новых значениях исходных данных без повторного набора предшествующих операторов. Нельзя возвратиться назад и повторить некоторые действия, или по некоторому условию перейти к выполнению другой последовательности операторов. И вообще, если количество операторов значительно, становится проблемой отладить правильную их работу из-за неминуемых ошибок при наборе команд. Поэтому сложные, с прерываниями, сложными переходами по определенным условиям, с часто повторяемыми однотипными действиями вычисления, которые, вдобавок, необходимо проводить неоднократно при измененных исходных данных, требуют их специального оформления в виде записанных на диске файлов, т. е. в виде программ. Преимущество программ в том, что, так как они зафиксированы в виде записанных файлов, становится возможным многократное обращение к одним и тем же операторам и к программе в целом. Это позволяет упростить процесс отладки программы, сделать процесс вычислений более наглядным и прозрачным, а благодаря этому резко уменьшить возможность появления ошибок при разработке программ. Кроме того, в программах возникает возможность автоматизировать также и процесс изменения значений первоначальных 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 поэтому требуют при обращении к ним указания имени М-файла, в котором записан текст 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 известному 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 граница изменения аргумента функции; I - полученное значение интеграла; cnt - количество 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 численного решения (интегрирования) простых дифференциальных уравнений, так и для моделирования сложных динамических систем, т. е. систем, поведение которых можно описать совокупностью обыкновенных дифференциальных уравнений.

Известно, что любая система обыкновенных дифференциальных уравнений (ОДУ) может быть представлена как система уравнений 1-го порядка в форме Коши:

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

где $\boldsymbol{y}$ - вектор переменных состояния (фазовых переменных системы); t - аргумент (обычно - время); $\boldsymbol{f}$ - нелинейная вектор-функция переменных состояния $\boldsymbol{y}$ 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 используемые параметры имеют такой смысл: <имя функции> - строка символов, представляющая собой имя М-файла, в котором вычисляется вектор-функция ***f***(*y,t*), т. е. *правые части системы ОДУ*; y0 - вектор начальных значений переменных состояния; t - массив рассчитанных значений аргумента, отвечающих шагам интегрирования; y - матрица проинтегрированных значений фазовых переменных, в которой каждый столбец соотвествует одной из переменных состояния, а строка содержит значения переменных состояния, отвечающие соответствующему шагу интегрирования; tspan - вектор-строка [t0 tfinal], содержащая два значения: t0 - начальное и tfinal - конечное значение аргумента; ortions - строка из параметров, которые определяют значения допустимой относительной и абсолютной погрешности интегрирования.

Параметр ortions можно не указывать. Тогда, по умолчанию, допустимая относительная погрешность интегрирования принимается равной $1\cdot10^{-3}$, абсолютная (по любой из переменных состояния) - $1\cdot10^{-6}$. Если же эти значения не устраивают пользователя, нужно 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 фазовым 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 погрешности интегрирования на каждом шаге и изменяет величину шага интегрирования так, чтобы обеспечить заданные границы погрешностей интегрирования.

Для функции ***ode45*** основным методом интегрирования является метод Рунге-Кутта 4-го порядка, а величина шага контролируется методом 5-го порядка.

Вычисление минимумов и корней функции осуществляется такими функциями MatLAB:

fmin - отыскание минимума функции одного аргумента;

fmins - отыскание минимума функции нескольких аргументов;

fzero - отыскание нулей (корней) функции одного аргумента.

Обращение к первой из них в общем случае имеет такой вид:

Xmin = ***fmin*** (‘<имя функции>’, X1, X2).

Результатом этого обращения будет значение Xmin аргумента функции, которое отвечает локальному минимуму в интервале X1<X<X2 функции, заданной М-файлом с указанным именем.

В качестве примера рассмотрим отыскание значения 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, отвечающий найденному локальному минимуму.

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

z = ***fzero*** (‘<имя функции>’, x0, tol, trace).

Здесь обозначен: x0 - начальное значение аргумента, в окрестности которого отыскивается действительный корень функции, значение которой вычисляется в М-файле с заданным именем; tol - заданная относительная погрешность вычисления корня; trace - знак необходимости выводить на экран промежуточные результаты; z - значение искомого корня.

Построение графиков функции одной переменной может быть осуществлена с помощью процедуры ***fplot***. Отличие ее от процедуры ***plot*** в том, что для построения графика функции нет необходимости в предшествующем вычислении значений функции и аргумента. Обращение к ней имеет вид:

fplot (‘<имя функции>’, [<интервал>], n),

где <интервал> - это вектор-строка из двух чисел, которые задают, 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 - желательное число частей разбиения указанного интервала. Если последнюю величину не задать, по умолчанию интервал разбивается на 25 частей. Несмотря на то, что количество частей "n" задано, 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 больее 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 этойпроцедуре так:

[x, Y] = ***fplot*** (‘<имя функции>’, [<интервал>], n),

то график указанной 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

2.2.1. Особенности создания М-файлов

Создание программы в среде MatLAB осуществляется с помощью либо собственного встроенного (начиная из версии MatLAB 5), либо стороннего текстового редактора, который автоматически вызовется, если его предварительно установить с помощью команды *Preferences* меню ***File*** командного окна MatLAB. Например, это может быть редактор Notepade среды Windows. Окно предварительно установленного редактора появляется на экране, если перед этим вызвана команда ***M-file*** из меню ***New*** или выбрано название одного из существующих М-файлов при вызове команды ***Open M-file*** из меню ***File*** командного окна. 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 или корректировки существующего.

Программы на языке MatLAB имеют две разновидности - так называемые *Script-файлы* (*файлы-сценарии*, или *управляющие программы*) и *файлы-функции (процедуры)*. Обе разновидности должны иметь расширение имени файла ***.m***, т. е. их нельзя различить по типу файла. С помощью Script-файлов оформляют 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 измененных значениях исходных 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 обозначен ПКВ - Перечень Конечных Величин, ПВВ - Перечень Входных Величин. *Script-файлы такой строки не имеют*.

Принципиальное же отличие состоит в совсем различном восприятии системой имен переменных в этих двух видах файлов.

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)*. Значение и содержание их сохраняются не только на протяжении времени работы программы, но и на протяжении всего сеанса работы с системой, а, значит, и при переходе от выполнения одного Script-файла к другому. Иначе говоря, рабочее пространство является единым для всех Script-файлов, вызываемых в текущем сеансе работы с системой. Благодаря этому любой 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. Этим обеспечивается компактное и наглядное представление даже довольно сложной программы.

За исключением указанных отличий, файл-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) языком MatLAB.

1. 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.

2. Можно размещать несколько операторов в одной строке. Тогда предыдущий оператор этой строки должен заканчиваться символом ' ; ' или ' , '.

3. Можно длинный оператор записывать в несколько строк. При этом предыдущая строка оператора должна заканчиваться тремя точками (' ... ').

4. Если очередной оператор не заканчивается символом ' ; ', результат его действия при выполнении программы будет выведен в командное окно. Чтобы предотвратить вывод на экран результатов действия оператора программы, запись этого оператора в тексте программы должна заканчиваться символом ' ; '.

5. 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 ' % '.

6. 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>

7. В программах на языке MatLAB *отсутствует символ окончания текста программы.*

8. В языке MatLAB переменные не описываются и не объявляются. Любое новое имя, появляющееся в тексте программы при ее выполнении, 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 предшествующих операторах 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".

9. Имена переменных могут содержать лишь буквы латинского алфавита или цифры и должны начинаться из буквы. Общее число символов в имени может достигать 30. В именах переменных могут использоваться как прописные, так и строчные буквы. Особенностью языка MatLAB есть то, что *строчные и прописные 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 процедурой (в языке Паскаль), или подпрограммой. Общий вид первой строки в этом случае становится таким:

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

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

В простейшем случае функции одной переменной заголовок приобретет вид:

function y = ***func***(x),

где ***func*** **-** имя функции (M-файла).

В качестве примера рассмотрим составление M-файла для функции

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

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

```
function y = F1(x,d)
  %       Процедура,  которая вычисляет значение функции
  %         y = (d3)*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);
```

После этого необходимо сохранить этот текст в файле под именем F1.m. Необходимый M-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 аналогично 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 в ту же процедуру

» my = F1(zet,1),

то получим:

```
Warning: Divide by zero
my =
  Columns 1 through 4
     Na + Infi 0 + 2. 9369i        0 + 0. 8799i   0. 3783
  Columns 5 through 7
   0. 3339          0. 0706          -0. 2209
```

Примечания.

1. *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.*

2. Во избежание вывода на экран нежелательных промежуточных результатов, *необходимо в тексте процедуры все вычислительные операторы завершать символом* " ; ".

3. 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. 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, которая вычисляет значение функции
y = (d3)*ctg(x)*sqrt(sin(x)4-cos(x)4).
Обращение y = F1(x,d).

Другой пример. Построим график двух функций:

y1 = 200 sin(x)/x; ***y2 = x2.***

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) = x2.
y(:,1) = 200*sin(x) . / x;
y(:,2) = x .  2;
```

Теперь построим графіки этих функций:

```
» fplot('myfun', [-20 20], 50, 2), grid
» set(gcf,'color','white'); title('Графік функции  "MYFUN"')
```

Результат изображен на рис. 2.1.

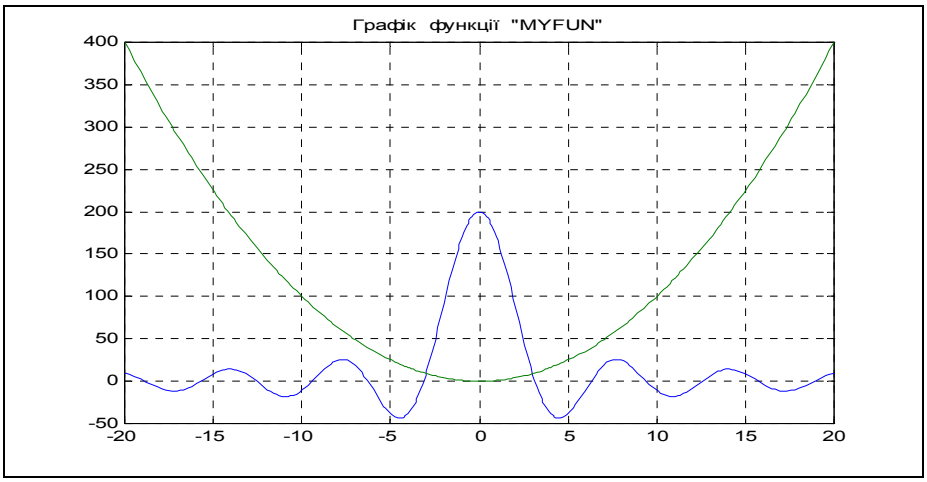

Рис. 2.1

Третий пример - создание файл-функции, вычисляющей значения функции

$$y(t) = k1+k2*t+k3*\sin(k4*t+k5).$$

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

К = [k1 k2 k3 k4 k5]

и создать такой М-файл:

```
function y = dvob(x, K)
  % Вычисление функции
  % y = K(1)+K(2)*x+K(3)*sin(K(4)*x+K(5)),
  % где К - вектор из пяты элементов
  % Используется 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 так

```
» 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

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

```
function [<Выход>] = <имя функции>(<Вход>)
  % <Краткое пояснение назначения процедуры>
  % Входные переменные
  %<Детальное пояснение о назначении, типе и размерах
  % каждой из переменных, перечисленных в перечне  <Вход>
  % Выходные переменные
  %<Детальное 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 обращение
  %    к другим процедурам, кроме встроенных>
```

```
< Пустая строка >
% Автор : <Указывается автор процедуры, дата создания конечного варианта
% процедуры и организация, в которой созданная программа>
<Текст исполняемой части процедуры>
```

Здесь обозначен: <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.*

2.3.3. Задания

Задание 2.1. Создайте М-файл, вычисляющий значение функции из *задания 1.5.* Постройте график этой функции с помощью процедуры ***fplot*** в границах, заданных в задании 1.5. Вычислите интеграл от функции в тех же пределах, используя процедуры ***quad*** и ***quad8***. Найдите точку локального минимума и локальный минимум функции и ближайший корень (нуль).

Задание 2.2. Найдите точку локального минимума и локальный минимум функции двух переменных, приняв за начальную точку с заданными координатами (таблица 2.1). Предварительно создайте соответствующую файл-функцию.

Таблица 2.1

Вариант	x_o	y_o	$f(x,y)$
1	0	1	$e^{x+y}+(x-y)^2-2x-2y$
2	0.7	-1. 2	$(x-y)^2-\cos(x-y-1)$
3	1.5	-0. 5	$e^{x+y}-2x-2y-\cos(x-y-1)$
4	0.5	1.5	$e^{x+y}+4x^2-3x-3y$
5	0	1	$4x^2+\ln(x+y)+\frac{1}{x+y}$
6	1.2	0.7	$2^{x+y}-2x-2y+2(x-y)^2$
7	0	-0. 9	$e^{x-y}+2x+2y+(x+y)^2$
8	0.8	1.3	$(x-y)^2-\cos(x+y-1)$
9	1.5	0.5	$e^{x-y}-2x+2y-\cos(x+y-1)$
10	0.5	-1. 5	$e^{x-y}-3x+3y+4x^2$
11	0	-1	$4x^2+\ln(x-y)+\frac{1}{x-y}$
12	1.2	-0. 8	$2^{x-y}-2x+2y+2(x+y)^2$

2.3.4. Вопросы

1. Для чего создаются программы в среде MatLAB?
2. Чем отличаются файл-функции от Script-файлов? Какова сфера применения каждого из этих видов файлов?
3. Что понимается под понятием "функция функций"? Какие наиболее употребительные стандартные функции функций есть в MatLAB?
4. Как создать М-файл процедуры или функции?
5. Какие основные правила написания текстов М-файлов в языке MatLAB?

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;

■ в них отсутствует заголовок, т. е. первая строка определенного вида и назначения;

■ обращение к ним 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 файлов-функций*, которые используются в нем. В файлах-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 могла сохранить 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 значение 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)
  % Вычисление 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 К из пяти элементов (заданы их значения).

Если в одной строке объявляются *несколько переменных* как глобальные, 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, в особенности, объединение текста с численными значениями некоторых переменных, что часто необходимо 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 вектор, пользуясь операцией создания вектора-строки:

x = [x1 x2 ... x].

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

disp ([x1 x2 ... x]).

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.
```

Аналогично 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 значение переменных, что часто бывает необходимо. Трудности возникают потому, что нельзя объединять текстовые и числовые переменные, так как они являются данными разных типов. Одним из путей преодоления этого препятствия есть перевод числового значения переменной в символьную (текстовую) форму. Это возможно, если вос-

пользоваться функцией ***num2str***, которая осуществляет такое преобразование. Запись:

y = ***num2str****(x)*

превратит 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 обращениям

disp ([T num2str(X)]).

Рассмотрим пример:

```
x =
 -9. 3088e-015
» T = 'Значение параметра равняется ';
» disp([T x])
Значение параметра равняется
» disp([T num2str(x)])
Значение параметра равняется -9. 309e-015
```

Как следует из этого примера, "механическое" объединение текстовой и числовой переменных не приводит к желаемому результату.

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

Y = ***sprintf*** (‘<текст1> **%g** <текст2>’, X).

В результате получается текстовая строка Y, состоящая из текста, указанного в <текст1>, и значения числовой переменной X в соответствия с форматом %g, причем текст из фрагмента <текст2> располагается 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','Альтернатива1','Альтернатива2',..., '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 меню с альтернативами. После правильного ответа исходному параметру "k" присваивается значение номера избранной альтернативы (1, 2 , …, n). В общем случае число альтернатив может быть до 32.

Теперь, в зависимости от полученного 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 М-файла встречается команда ***keyboard,*** то при выполнении программы выполнение М-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 осуществить любые действия, проверить или изменить данные. При этом ему доступны все команды и процедуры системы MatLAB. Для завершения работы в этом режиме необходимо ввести команду ***return.*** Тогда система продолжит роботу программы с оператора, следующего за командой ***keyboard.***

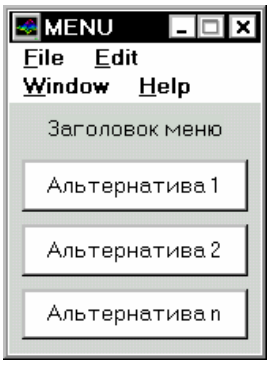

Рис. 2.2

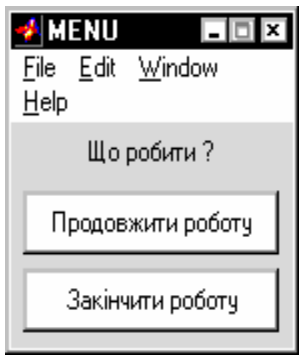

Рис. 2.3

2.4.3. Организация повторения действий

Одной из главных задач создания самостоятельной программы есть 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, например, 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 из 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 роботу ', ' Закончить работу ');
end
```

Тогда, после первого выполнения Script-файла "ScrFil_Yadro" на экране появится окно 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, цикл закончится и программа перейдет к окончанию работы.

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 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, если значение всех параметров установлены.

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

```
k = menu( ‘   Що змінити ?   ', ' Параметр1 ', ' Параметр2 ', ...
   ‘  Параметр3  ’, ‘  Параметр4  ’, ‘  Параметр5  ’,‘  Ничого не змінювати  ’),
```

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

Легко заметить недостаток такого оформления окна меню. Чтобы принять решение, 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( '   Що  змінювати ?   ', ...
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),...
 '  Нічого не змінювати  ')
```

Результат приведен на рис. 2.5.

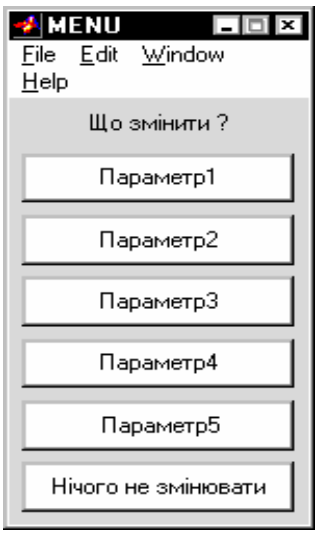

Рис. 2.4

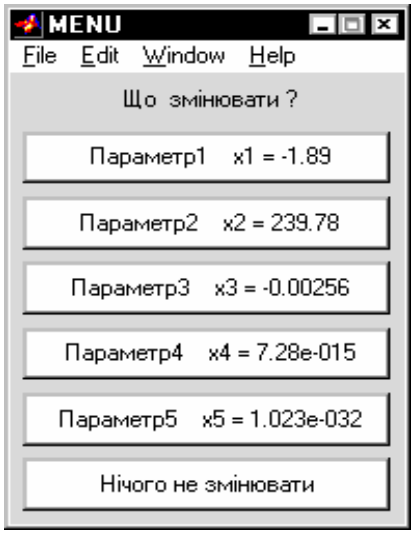

Рис. 2.5

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 надпись:

```
Текущее значение  x = 0. 0302   Новое значение  x =
```

Выполнение программы приостановится. ПК будет ожидать ввода информации с клавиатуры. Если теперь набрать на клавиатуре "0.073" и нажать клавишу <Enter>, то в командном окне появится запись:

```
Текущее значение  x = 0. 0302   Новое значение  x = 0. 073
x =    0. 0730
```

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

Теперь следует 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
```

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 возникало то же меню, но уже со скорректированными значениями. При этом конец работы с меню должен наступить только при условии выбора последней альтернативы меню " Нічого не змінювати ", соответствующей значению k, равному 6.

Поэтому предыдущие операторы следует заключить в цикл:

```
k=1;
while k<6
k = menu( ' Що  змінювати ?   ', ...
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), ' Нічого не змінювати ');
   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
```

Так организуется возможность достаточно удобного изменения значений параметров в диалоговом режиме.

Если входных параметров, значение 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 программы в виде Script-файла необходимо принимать во внимание следующее.

1. Удобно оформлять весь процесс диалогового изменения параметров в виде отдельного Script-файла, к примеру, с именем "ScrFil_Menu", где под сокращением "ScrFil" понимается имя основного (собирательного) Script-файла.

2. 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 поместить часть программы, которая задает первоначальные 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 содержания 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".

3. При завершении работы программы обычно возникает потребность несколько упорядочить рабочее пространство, например, очистить его от введенных глобальных переменных (оставаясь в рабочем пространстве, они препятствуют корректной работе другой программы, которая может иметь другие глобальные переменные, или переменные с теми же именами, но иными по типу, смыслу и значениею), закрыть открытые программой графические окна (фигуры) и т.д. Эту завершающую часть тоже можно оформить как отдельный Script-файл, например, назвав его "ScrFil_Kin".

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

```
%<Обозначение Script-файла (ScrFil.m)>
% <Текст комментария с описанием назначения программы>
```

```
            < Пустая строка >
 %Автор < Фамилия И. О., дата создания, организация>

 ScrFil_Zastavka
k = menu(' Що 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, предназначена, в большинстве случаев, 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 отвечает 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. На основе полученной графической информации пользователь-инженер должен иметь возможность принять какое-то решение о выборе значений некоторых конструктивных параметров, которые характеризуют исследуемый процесс или техническое устройство, с целью, чтобы прогнозируемое поведение технического устройства удовлетворяло определенным заданным условиям. Поэтому инженерные графики должны быть, как говорят, “читабельными”, т. е. иметь такой вид, чтобы из них легко было “отсчитывать” значения функции при любых значениях аргумента (и наоборот) с относительной погрешностью в несколько процентов. Это становится возможным, если координатная сетка графиков отвечает определенным целым числам какого-либо десятичного разряда. Как уже раньше отмечалось, графики, построенные системой MatLAB, полнлстью 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 ин-

формация, 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:

- если нужно сравнивать графики функций одного аргумента, диапазоны изменения которых не слишком отличаются один от другого (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. в общих координатных осях); в этом случае *следует выводить графики с помощью одной функции* ***plot***;
- если при тех же условиях диапазоны изменения функций значительно различаются, можно предложить два подхода:

  а) если все сравниваемые функции являются значениями величин одинаковой физической природы и эти значения все положительны, графики следует выводить также в одно графическое поле, но *в логарифмическом масштабе* (т. е. использовать процедуру ***semilogy***)*;*

  б) когда все функции имеют разную физическую природу, но аргумент в 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, которое обеспечивается применением процедуры ***subplot***); при этом удобно размещать отдельные графики один под одним таким образом, чтобы одинаковые значения аргумента во всех графиках располагались 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, исполнителя 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. Разбивка 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.

Рассмотрим, как обеспечить желательную разбивку 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 использованная верхняя из этих трех частей (рис. 2.6);

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 трех четвертях экрана по горизонтали, а текстовую информацию разместить в последней четверти по горизонтали, то это можно сделать (рис. 2.7) так:

- 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 вывода 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;
- 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]);

Рис. 2.7

- 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 выводу 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. Из него видно, что:

- после обращения к процедуре ***subplot*** в соответствующем подокне появляется изображение осей координат с обозначением делений по осям;
- начальный диапазон изменений координат по обеим осям подокна всегда устанавливается по умолчанию 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').

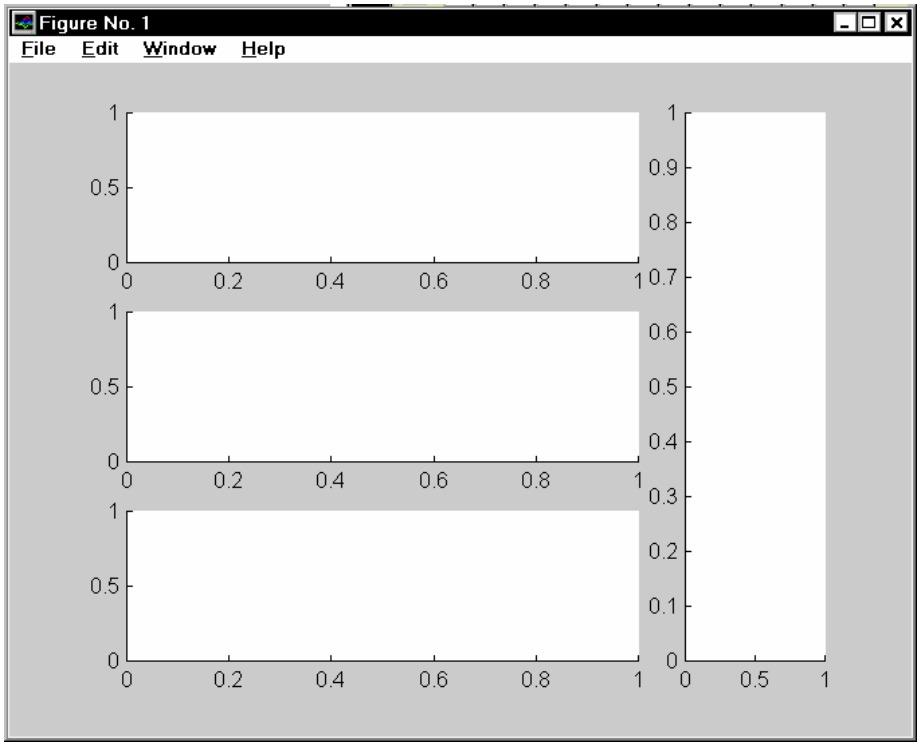

Рис. 2.8

Так, если эту команду ввести после предидущих команд, в окне фигуры исчезнет изображение координатных осей последнего подокна (рис. 2.9). Теперь можно начинать выведение 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 пикселах>).

Она осуществляет вывод указанного текста указанным шрифтом указанного размера, начиная из точки подокна с 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 равняется [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 несколько более широкий диапазон, учитывая то, что поле подокна больше поля его графика.

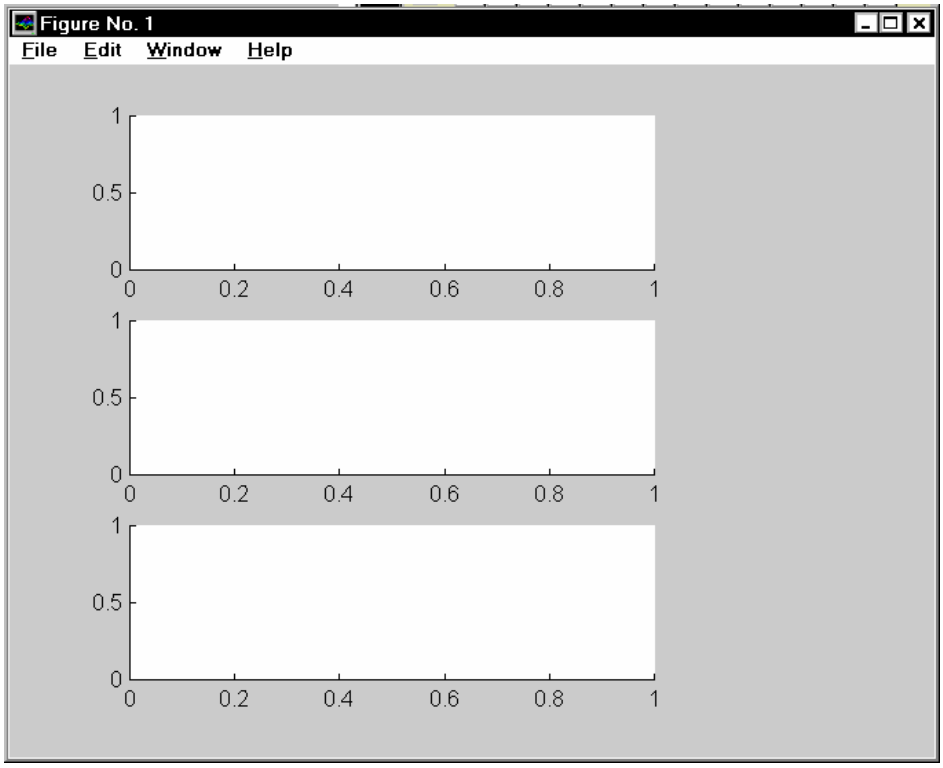

Рис. 2.9

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 = 'vsp1'; 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(' = %g ',D1(3)),'FontSize',10);
h1=text(-0.2,0.85,sprintf(' = %g ',D1(4)),'FontSize',10);
h1=text(0.6,0.85,sprintf(' = %g ',D1(5)),'FontSize',10);
h1=text(-0.2,0.8,sprintf(' = %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);
```

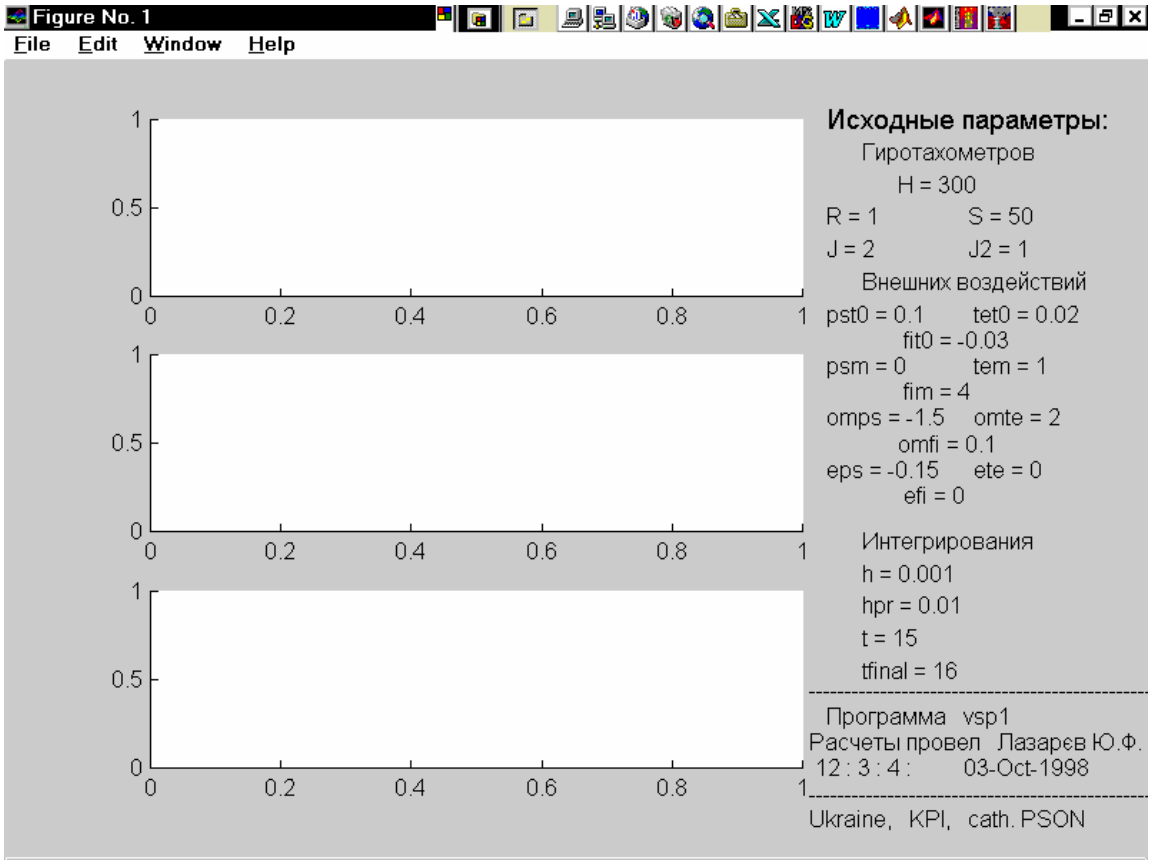

Рис. 2.10

```
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 его приводит к появлению в окне фигуры изображения, представленного на рис. 2.10.

2.6. Создание функций от функций

Некоторые алгоритмы являются общими для функций определенного типа. Поэтому их программная реализация едина для всех функций этого типа, 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 файла-функции) только при обращении к основному алгоритму.

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

- вычислений 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 нуждаются в указания файла-функции, нуль которого отыскивается;
- алгоритмов поиска минимума функции, которую, в свою очередь, надо задавать соответствующим М-файлом и т.п.

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

2.6.1. Процедура feval

В MatLAB любая функция (процедура), например, с именем 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 изменяет формы при изменении имени функции, например, на FUN2. Это позволяет унифицировать обращение ко всем функциям определенного вида, 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 обращения 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:

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

где $\boldsymbol{y}$ - вектор переменных состояния системы; t - аргумент (время); $\boldsymbol{Z}$ - вектор заданных (в общем случае – нелинейных) функций, которые, собственно, и определяют конкретную систему ОДУ.

Если значение вектора $\boldsymbol{y}$ 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:

$\boldsymbol{yout} = \boldsymbol{y} + h*\boldsymbol{F}(\boldsymbol{y},t)$.

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

$$\mathbf{F} = (\mathbf{k_1} + 3\cdot\mathbf{k_2} + 3\cdot\mathbf{k_3} + \mathbf{k_4})/8\ ,$$

где $\mathbf{k_1} = \mathbf{Z}(\mathbf{y}, t)$;

$\mathbf{k}_2 = \mathbf{Z}(\mathbf{y} + h\cdot\mathbf{k_1}/3,\quad t + h/3)$;

$\mathbf{k}_3 = \mathbf{Z}(\mathbf{y} + h\cdot\mathbf{k}_2 - h\cdot\mathbf{k_1}/3,\quad t + 2h/3)$;

$\mathbf{k}_4 = \mathbf{Z}(\mathbf{y} + h\cdot\mathbf{k}_3 - h\cdot\mathbf{k}_2 - h\cdot\mathbf{k_1},\quad t + h)$.

Создадим М-файл процедуры, которая осуществляет эти вычисления, назвав 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, который содержит имя процедуры
 %           вычисления правых частей ОДУ
 %  Обращение: z = fun(t,y), где Zpfun = 'fun'
 %                    где 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
 %      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
```

Обратите внимание на такие обстоятельства:

- обращение к процедуре вычисления правых частей не конкретизированно; имя этой процедуры входит в число входных переменных процедуры интегрирования и должно быть конкретизировано лишь при вызове последней;

- промежуточные 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 гармоничному 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:

$y1 = \varphi$; $y2 = \dot{\varphi}$.

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]
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 с помощью служебного слова ***global***, а потом определены все 8 его элементов.

Эту процедуру можно несколько усложнить, группируя вместе вычисление всех внешних моментов сил, кроме момента сил тяготения, и оформляя их как отдельную процедуру. Для этого сначала превратим начальное 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 количество 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 действий 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, которые действуют на маятник:

```
function m = MomFM1(t,y);
  % Вычисление Моментов сил, которые действуют 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]
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
  %  векторов "y" переменных состояния по формулам:
  %             z(1)=y(2);
  %             z(2)= - sin(y(1)) +S(t,y),
  % 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.
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 и 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 параметры:
 %          Zpfun - строка символов, который содержит имени процедуры
 %                    вычисление 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 - предшествующий момент времени
 %          y - предшествующее значение вектора переменных состояния.
 % Выходные параметры:
 %          tout - новый момент времени
 %          yout - новое значение вектора переменных состояния
 %                 через шаг интегрирования
 % Расчет промежуточных значений производных
  k1 = feval(Zpfun, Mpfun,t, y);
  k2 = feval(Zpfun, Mpfun, t+h/3, y+h/3*k1);
  k3 = feval(Zpfun, Mpf un, 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. 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). Во-вторых, такая форма вызовет необходимость создания новых М-файлов методов численного интегрирования.

Этого можно избегнуть, представив имя дополнительной функции Mpfun на 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 вектору "y" переменных состояния по формулам:
 %                 z(1)=y(2);
 %                 z(2)= - sin(y(1)) +S(t,y),
 % Входные параметры:
 %       mpfun - имя процедуры S(t,y) - глобальная переменная
  %                mpfun = 'S';
 %       t - текущий момент времени;
 %       y - текущее значение вектора переменных состояния;
```

```
 % Выходные параметры:
 %       z - вектор значений производных от переменных состояния.
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 через заголовок, и может быть использована любой процедурой численного метода интегрирования, в том числе - процедурами ***ode23*** и ***ode45***. Необходимо лишь 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';
```

2.6.3. Задания

Задания 2.1 - 2.13. Создайте М-файл метода численного интегрирования дифференциальных уравнений в соответствии с формулами, приведенными в таблицах 2.1 и 2.2.

Таблица 2.1. Методы Рунге-Кутта

$y_{m+1} = y_m + h\,F(t_m; y_m)$

N% вар	Формула метода	Вспомогательные величины	Название методу
1	F=k1	$k1=Z(t_m;y_m)$	Ейлера
2	F=(k1+k2)/2	$k1=Z(t_m;y_m)$; $k2=Z(t_m+h;y_m+hk1)$	модифицированный Ейлера
3	$F=Z(t_m+h/2; y_m+hk1/2)$	$k1=Z(t_m;y_m)$	
4	F=(k1+4k2+k3)/6	$k1=Z(t_m;y_m)$; $k2=Z(t_m+h/2;y_m+hk1/2)$; $k3=Z(t_m+h;y_m+h(2k2-k1))$	Хойне
5	F=(k1+3k3)/4	$k1=Z(t_m;y_m)$; $k2=Z(t_m+h/3;y_m+hk1/3)$; $k3=Z(t_m+2h/3;y_m+2hk2/3)$	
6	F=(k1+2k2+2k3+ +k4)/6	$k1=Z(t_m;y_m)$; $k2=Z(t_m+h/2;y_m+hk1/2)$; $k3=Z(t_m+h/2;y_m+hk2/2)$; $k4=Z(t_m+h;y_m+hk3)$	Рунге-Кутта

7	F=(k1+3k2+3k3+ +k4)/8	k1=Z(t_m;y_m); k2=Z(t_m+h/3;y_m+hk1/3); k3=Z(t_m+2h/3;y_m+h(k2-k1/3)); k4=Z(t_m+h;y_m+h(k1--k2+k3))	

Таблица 2.2. Многошаговые методы

N% вар	Формула прогнозу	Формула коррекции	Название методу
8	$y_{m+1}=y_{m-1}+2h(t_m;y_m)$	$y_{m+1}=y_m+h[Z(t_{m+1};y^*_{m+1})+Z(t_m;y_m)]/2$	
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	$y_{m+1}=y_{m-3}+4h[2Z(t_m;y_m)-Z(t_{m-1};y_{m-1})+2Z(t_{m-2};y_{m-2})]/3$	$y_{m+1}=y_{m-1}+h[Z(t_{m+1};y^*_{m+1})+4Z(t_m;y_m)+Z(t_{m-1};y_{m-1})]/3$	Мілна
13	$y_{m+1}=y_{m-3}+4h[2Z(t_m;y_m)-Z(t_{m-1};y_{m-1})+2Z(t_{m-2};y_{m-2})]/3$	$y_{m+1}=\{9y_m-y_{m-2}+3h[Z(t_{m+1};y^*_{m+1})+2Z(t_m;y_m)-Z(t_{m-1};y_{m-1})]\}/8$	Хеммінга

Задание 2.14. Создайте М-файл процедуры правых частей дифференциальных уравнений движения двухстепенного гироскопического компаса:

$$J_1\ddot{\beta}+[H-J_2(\omega_3\cos\varphi\cos\beta+u_N(t)\cos\beta-u_E(t)\sin\beta)]*$$

$$*(\omega_3\cos\varphi\sin\beta+u_E(t)\cos\beta+u_N(t)\sin\beta)=0,$$

где J_1, J_2 - моменты инерции гирокомпаса; H - его собственный кинетический момент; β - угол отклонения главной оси гирокомпаса от плоскости географического меридиана места ; φ - географическая широта места объекта, на котором установлен гирокомпас;

$$u_N(t)=u_{Nm}\sin(\omega t+\varepsilon_N), \qquad u_E(t)=u_{Em}\sin(\omega t+\varepsilon_E)$$

- соответственно северная и восточная составляющие угловой скорости поворота основания, на котором установлен гирокомпас.

Задание 2.15. Создайте процедуру правых частей дифференциальных уравнений, которые описывают динамику объемов популяций $x_1(t)$ хищников и $x_2(t)$ жертв и известны как модель Вольтерра:

$$\dot{x}_1 = -a_{11}x_1 + a_{12}x_1x_2;$$

$$\dot{x}_2 = a_{22}x_1 - a_{21}x_1x_2.$$

Задание 2.16. Создайте процедуру правых частей дифференциального уравнения углового движения торпеды в горизонтальной плоскости, которая управляется нелинейным исполнительным элементом:

$$J\frac{d^2\psi}{dt^2} + R\frac{d\psi}{dt} + kF(\psi) = 0,$$

Процедура должна предусматривать возможность использования нескольких существенно нелинейных законов управления $F(x)$, в частности, релейного с зонами нечувствительности и гистерезисом:

если $\dot{x} > 0$, то
$$F(x) = \begin{cases} -c & \text{при } x < -b_1, \\ 0 & \text{при } -b_1 < x < b_2, \\ +c & \text{при } x > b_2; \end{cases}$$

если же $\dot{x} < 0$, то
$$F(x) = \begin{cases} +c & \text{при } x > b_1, \\ 0 & \text{при } -b_2 < x < b_1, \\ -c & \text{при } x < b_2. \end{cases}$$

Задание 2.17. Создайте процедуру правых частей дифференциального уравнения углового движения искусственного спутника Земли, управляемого по логическому закону:

$$J\frac{d\omega}{dt} = k\Phi(\omega,\varphi); \qquad \frac{d\varphi}{dt} = \omega \ ,$$

где ω - угловая скорость движения спутника; J - его момент инерции; $\Phi(\omega,\varphi)$ - заданная логическая нелинейная функция, которую определим с помощью такой таблицы:

Значение функции $\Phi(\omega,\varphi)$

Знак ω	Знак φ		
	-	0	+
-	+1	0	0
0	+1	0	-1
+	0	0	-1

Задание 2.18. Создайте процедуру правых частей дифференциальных уравнений движения волчка со сферическим подпятником, установленным в конической лунке:

$$\begin{cases} \dfrac{dK_\xi}{dt} = M_{\text{□p}\xi}, \\ \dfrac{dK_\eta}{dt} = mgl\sin\delta_1\cos\delta_2 + M_{\text{□p}\eta}, \\ \dfrac{dK_\varsigma}{dt} = mgl\sin\delta_2 + M_{\text{□p}\varsigma}, \\ J_e\dot{\delta}_1\cos\delta_1 = K_\eta - K_\xi\cos\delta_1\sin\delta_2 + K_\varsigma\sin\delta_1\sin\delta_2, \\ J_e\dot{\delta}_2 = K_\xi\sin\delta_1 + K_\varsigma\cos\delta_1, \end{cases}$$

где J_e - экваториальный момент инерции волчка относительно точки опоры; H - кинетический момент волчка; δ_1, δ_2 - углы отклонения оси волчка от вертикали; mgl - опорный маятниковый момент волчка; $M_{mp\xi}, M_{mp\eta}, M_{mp\varsigma}$ - составляющиемомента сил трения в подпятнике волчка; $K_\xi, K_\eta, K_\varsigma$ - проекции кинетического момента волка на неподвижные оси. Моменты сил трения $M_{mp\xi}, M_{mp\eta}, M_{mp\varsigma}$ можно представить следующими зависимостями:

$$M_{\text{□p}\xi} = -C\cos^2\alpha\cos\delta_1\cos\delta_2;$$

$$M_{\text{□p}\eta} = -C\{\sin\delta_2[1-(\sin^2\alpha)/2] + \cos\delta_2\sin\delta_1(\sin^2\alpha)/2\};$$

$$M_{\text{□p}\varsigma} = C\cos\delta_2\sin\delta_1[1-(\sin^2\alpha)/2],$$

где

$$C = k\frac{R}{l}mgl\frac{1}{B\cos\alpha}sign(H),$$

а

$$B = \sqrt{1-\cos^2\alpha\cos^2\delta_1\cos^2\delta_2 - \sin^2\alpha(\sin^2\delta_2 + \sin^2\delta_1\cos^2\delta_2)/2}\ .$$

Выше использованы обозначения: k - коэффициент трения материала подпятника и материала опоры; R - радиус сферы подпятника; α - угол между образующей конуса лунки и плоскостью горизонта.

Взаимосвязь проекций $K_\xi, K_\eta, K_\varsigma$ с собственным кинетическим моментом H и другими кинематическими величинами определяется соотношениями:

$$K_\xi = H\cos\delta_2\cos\delta_1 - J_e\dot{\delta}_1\sin\delta_2\cos\delta_2\cos\delta_1 + J_e\dot{\delta}_2\sin\delta_1;$$

$$K_\eta = H\sin\delta_2 + J_e\dot{\delta}_1\cos^2\delta_2;$$

$$K_\varsigma = -H\cos\delta_2\sin\delta_1 + J_e\dot{\delta}_1\sin\delta_2\cos\delta_2\sin\delta_1 + J_e\dot{\delta}_2\cos\delta_1.$$

Задание 2.19. Создайте процедуру правых частей дифференциальных уравнений гироскопа в кардановом подвесе (ГКП), установленного на неподвижном основании:

$$\begin{cases} (J_1 + J_2\cos^2\beta)\ddot{\alpha} - 2J_2\dot{\alpha}\dot{\beta}\sin\beta\cos\beta + H\dot{\beta}\cos\beta = \\ = -f_2\dot{\alpha} + N_0 + N_m\sin(\omega t + \varepsilon_N) - [R_0 + R_m\sin(\omega t + \varepsilon_R)]\sin\beta, \\ J_3\ddot{\beta} + J_2\dot{\alpha}\sin\beta\cos\beta - H\dot{\alpha}\cos\beta = -f_2\dot{\beta} + L_0 + \\ + L_m\sin(\omega t + \varepsilon_L), \\ \frac{dH}{dt} = R_0 + R_m\sin(\omega t + \varepsilon_R), \end{cases}$$

где $J_1 = J_{2X} + J_{1Z}$; $J_2 = J_{1X} + J_e - J_{1Z}$; $J_3 = J_{1Y} + J_e$; J_{2X} - момент инерции внешней рамки карданового подвеса относительно наружной оси подвеса; J_{1X}, J_{1Y}, J_{1Z} - моменты инерции внутренней рамки относительно указанных осей; J_e - экваториальный момент инерции ротора гироскопа; α, β - углы поворота главной оси ГКП вокруг наружной и внутренней осей подвеса; H - собственный кинетический момент ГКП; f_1, f_2 - коэффициенты вязкого трения по внутренней и наружной осям подвеса; N_0, L_0, R_0 - постоянные составляющие моментов внешних сил, направленных по наружной, внутренней осям подвеса и главной оси гироскопа соответственно; N_m, L_m, R_m - амплитуды гармонических составляющих моментов сил, действующих по соответствующим осям; ω - частота изменения гармонических составляющих моментов сил; $\varepsilon_N, \varepsilon_L, \varepsilon_R$ - начальные фазы гармонических составляющих моментов сил.

Задание 2.20. Создайте процедуру правых частей дифференциального уравнения гироскопического тахометра (ГТ), установленного на вращающейся основе:

$$J_1\ddot{x} + f\dot{x} + cx = -c[u_{Z1} - (J_1/H)\dot{u}_{Y1} + (J_2/H)u_{X1}u_{Z1}] + M_0c/H,$$

где x - выходной сигнал ГТ; H - собственный кинетический момент ГТ; J_1, J_2 - моменты инерции ГТ; c - угловая жесткость упругой связи ГТ с основанием; f - коэффициент углового демпфирования; u_{X1}, u_{Y1}, u_{Z1} - проекции угловой скорости основания на оси, связанные с ГТ. Последние связаны с проекциями на оси, связанные с основанием, соотношениями:

$$u_{X1} = u_X\cos\beta - u_Z\sin\beta;$$

$$u_{Z1} = u_Z\cos\beta + u_X\sin\beta;$$

$$u_{Y1} = u_Y,$$

где $\beta = -\frac{H}{c}x.$

Проекции угловой скорости основания на оси, связанные с тем же основанием, полагать изменяющимися со времени по законам:

$$u_X = u_{X0} + u_{Xm}\sin(\omega t + \varepsilon_X);$$

$$u_Y = u_{Y0} + u_{Ym}\sin(\omega t + \varepsilon_Y);$$

$$u_Z = u_{Z0} + u_{Zm}\sin(\omega t + \varepsilon_Z).$$

Задание 2.21. Следящая система состоит из задающего элемента, который задает ϑ_1 угол, на который должен повернуться выходной вал следящей системы (вал электродвигателя), формирующего элемента (сельсина), который сравнивает этот угол с углом поворота ϑ_2 выходного вала электрического двигателя и формирует электрический сигнал, пропорциональный синусу разности этих двух углов:

$$u_1 = U_{1m}\sin(\vartheta_1 - \vartheta_2).$$

Этот сигнал суммируется с сигналом тахогенератора на валу двигателя:

$$u_2 = u_1 - u_k; u_k = k_Б\omega_Б.$$

Сигнал u_2 подается на усилительное устройство, которое представляет собой трехпозиционное реле с гистерезисом. Последнее формирует напряжение $u_Д$ в соответствия с зависимостью:

$$u_Д = f(u_2) = \begin{cases} z_b\ sign(u_2) & при \quad |u_2| > x_b; \\ 0 & при \quad |u_2| < x_a. \end{cases}$$

Вращательное движение вала двигателя описывается дифференциальными уравнениями:

$$\begin{cases} T_Б\dfrac{d\omega_Б}{dt} + \omega_Б = k_Б u_Б; \\ \dfrac{d\vartheta_2}{dt} = \omega_Б. \end{cases}$$

Создайте в форме М-файла процедуру вычисления правых частей дифференциальных уравнений следящей системы, считая выходными величинами угол $\vartheta = \vartheta_1 - \vartheta_2$ рассогласования и скорость его изменения.

Задание 2.22. Составьте процедуру отыскания точных решений системы линейных однородных дифференциальных уравнений по заданной матрице $\boldsymbol{A}$ системы ДУ в форме Коши:

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}\cdot\mathbf{y}$$

и заданному вектору **y0** начальных условий.

Задание 2.23. Создайте процедуру правых частей дифференциальных уравнений движения в пространстве трех гравитирующих материальных точек (задача трех тел в небесной механике)

$$\begin{cases} \dfrac{d^2\mathbf{R_2}}{dt^2} = -\gamma\left(\dfrac{m_1}{R_{21}^3}\mathbf{R_{21}} - \dfrac{m_3}{R_{32}^3}\mathbf{R_{32}}\right), \\ \dfrac{d^2\mathbf{R_3}}{dt^2} = -\gamma\left(\dfrac{m_2}{R_{32}^3}\mathbf{R_{32}} - \dfrac{m_1}{R_{13}^3}\mathbf{R_{13}}\right), \\ \mathbf{R_1} = -\dfrac{1}{m_1}(m_2\mathbf{R_2} + m_3\mathbf{R_3}), \end{cases}$$

где m_1, m_2, m_3 - массы гравитирующих материальных тел; $\mathbf{R_1}, \mathbf{R_2}, \mathbf{R_3}$ - радиусы-векторы материальных точек относительно их общего центра масс; $\mathbf{R_{21}} = \mathbf{R_2} - \mathbf{R_1}; \mathbf{R_{32}} = \mathbf{R_3} - \mathbf{R_2}; \mathbf{R_{13}} = \mathbf{R_1} - \mathbf{R_3}$ - радиусы-векторы точек друг относительно друга; R_{21}, R_{32}, R_{13} - текущие расстояния между точками; γ - гравитационная постоянная.

Задание 2.24. Составьте процедуру правых частей дифференциального уравнения лампового генератора:

$$\ddot{q} + \omega^2 q = (\chi - \lambda q^2)\dot{q} \ .$$

Это так называемое уравнение Ван-дер-Поля.

Задание 2.25. Составьте процедуру правых частей дифференциального уравнения движения маятника на подвижном основании с учетом момента сил сухого трения вдоль оси маятника:

$$J\ddot{\varphi} + R\dot{\psi} + mgl\sin\varphi = M_{mp} + M_0 + M_m \sin(\omega t),$$

где $\psi = \varphi - \vartheta(t)$, $\vartheta(t)$ - текущий угол поворота основания вокруг оси маятника; M_0 - постоянная составляющая момента внешних сил, которые действуют на маятник; M_m - амплитуда гармонической составляющей момента внешних сил; ω - частота изменения момента сил; M_{mp} - момент сил сухого трения по оси маятника.

Считать $M_{mp} = -M\,sign(\dot{\psi}),$ где

$$sign(x) = \begin{cases} +1, & \text{при } x > 0, \\ 0, & \text{при } x = 0, \\ -1, & \text{при } x < 0, \end{cases}$$

а также $\vartheta(t) = \vartheta_0 + \dot{\vartheta}_0 t + \vartheta_m \sin(\omega t + \varepsilon)$.

Задание 2.26. Составьте процедуру отыскания точных частных решений системы линейных неоднородных дифференциальных уравнений по заданной матрице $\boldsymbol{A}$ системы ДУ в форме Коши и матрице $\boldsymbol{B}$ коэффициентов входных воздействий:

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}\,\mathbf{y} + \mathbf{B}\,\mathbf{x} \ ,$$

где вектор $\boldsymbol{x}$ входных воздействий принять в виде

$$\mathbf{x} = \mathbf{B_o} + \mathbf{B_S}\sin(\omega t) + \mathbf{B_C}\cos(\omega t).$$

2.7. Пример создания 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, которая позволила бы 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.

Нужно 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. Для подготовки дифференциальных уравнений к численному интегрированию прежде всего необходимо *привести эти уравнения к нормальной форме Коши*. Желательно также представить их в безразмерной форме. Для представленного уравнения это было сделано в предідущем разделе.

Запись М-файла процедуры правых частей. Следующим шагом подготовки программы является написание и запись на диск текста процедуры вычисления правых частей полученной системы ДР в форме Коши.

Эта процедура была создана в предшествующем 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),
 % 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
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 процедуру, используемой в процедуре FM2, выберем ранее созданную процедуру MomFM1, которую запишем в файл MomFM1.

Файл MomFM1 . m

```
function m = MomFM1(t,y);
 % Вычисление Моментов Сил, которые действуют 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]
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 соответственно рекомендациям деления. 2.4.5 :

Файл FizMayatn2 . m

```
 % FizMayatn2
 % Управляющая программа исследования движения физического маятника,
 % установленного на поступательно вибрирующем основании
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 вызовет три дополнительных Script-файла - FizMayatn2_Zastavka, FizMayatn2_Menu и FizMayatn2_Yadro. Поэтому нужно создать еще эти три М-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 М-файла FizMayatn2_Zastavka.

Файл FizMayatn2_Zastavka . m

```
  % FizMayatn2_Zastavka
  % Часть (вывод заставки на экран) программы FizMayatn2
  % Ввод "вшитых" значений
sprogram = 'FizMaytn2.м';
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 виброперегрузки 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
  % Часть (осуществляющая диалоговое изменение данных)
```

```
  % программы FizMayatn2
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(' Перегрузка (вертикаль) = %g', KM1(2)),...
  sprintf(' Перегрузка (горизонталь) = %g', KM1(3)),...
  sprintf(' Относительная частота = %g', KM1(4)),...
  sprintf(' Фаза (вертикаль) = %g', KM1(5)),...
  sprintf(' Фаза (горизонталь) = %g', KM1(6)),...
  sprintf(' Начальный угол (градусы) = %g', fi0*180/pi),...
  sprintf(' Начальная скорость = %g', fit0),...
  sprintf(' Количество 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 количество периодов = %g', tfinal/2/pi),...
             ' Введите новое значение = ']);
     tfinal = tfinal*2*pi;
  end
end% FizMayatn2_Menu
```

Файл осуществляет организацию диалоговой ввода-изменения значений параметров физического маятника, движения основания и параметров численного интегрирования в соответствия 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
  % Часть (осуществляющая основные вычисления)
   % программы FizMayatn2
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 1. Подготовка начальных условий
 %---------------------------------
t = 0; tf = tfinal;        y0 =[fi0 fit0];
options = odeset('RelTol',1e-8,'AbsTol',[1e-10 1e-10]);
 %---------------------------------
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 2.  Организация цикла интегрирование
```

```
 %---------------------------------
  [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 маленьких 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);
 %_____________________
 % Вывод текстовой информации в графическое окно
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,'за уравнением','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,'за 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 понимал:','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
```

Как видим, основные операции включают три главные группы - ввод начальных условий, организацию цикла интегрирования и организацию оформления графического окна вывода.

Отладка программы. Отладка программы состоит из запуска главного М-файла FizMayatn2, проверки правильности функционирования всех частей программы, внесение 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. соответствия получаемых программой результатов отдельным априорно известным случаям поведения исследуемой системы. Очевидно, для такой проверки нужно подобрать не-

сколько 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 отвечают 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 продемонстрированные 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 потом устанавливаются, причем частота постоянных колебаний вдвое меньше частоты вибрации основания и составляет примерно 1,15.

Выпрямительный эффект маятника проиллюстрирован 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 около -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 показан на рис. 2.15, и маятник "застывает" в верхнем положении.

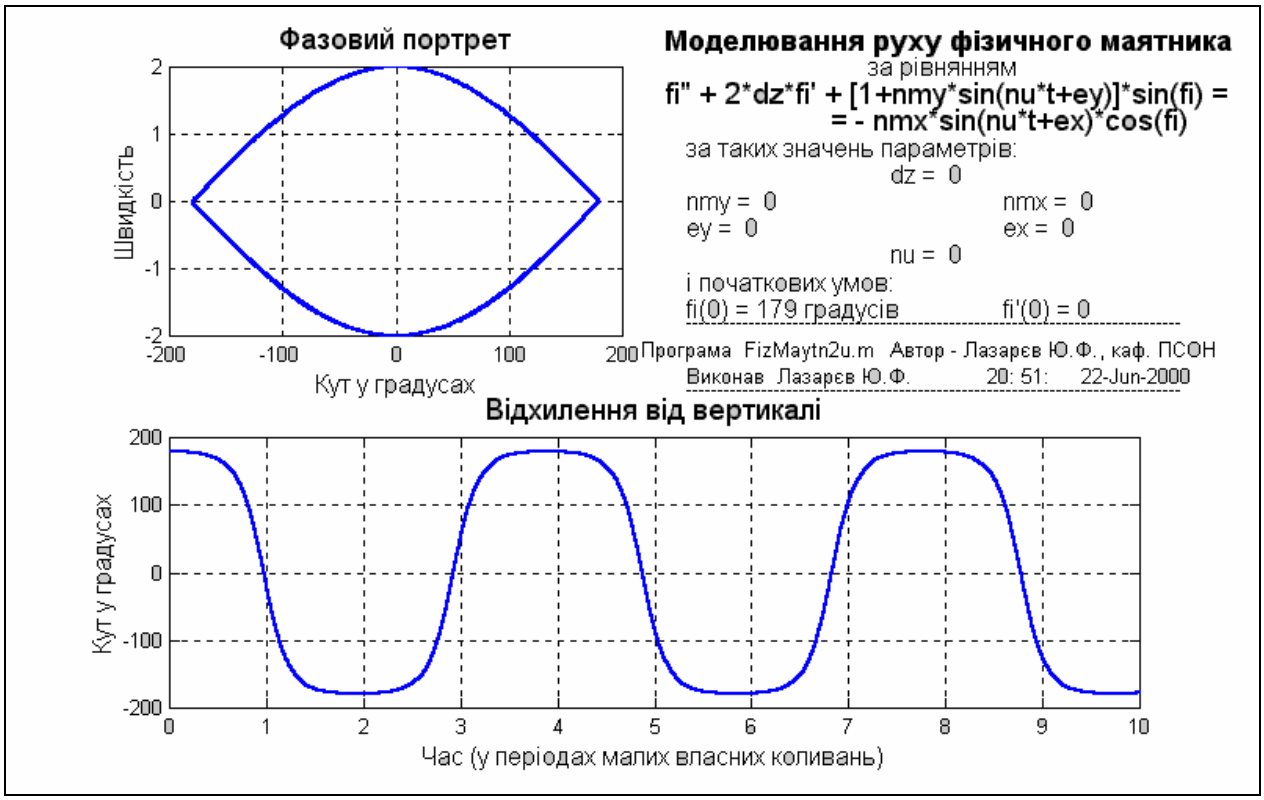

Рис. 2.11

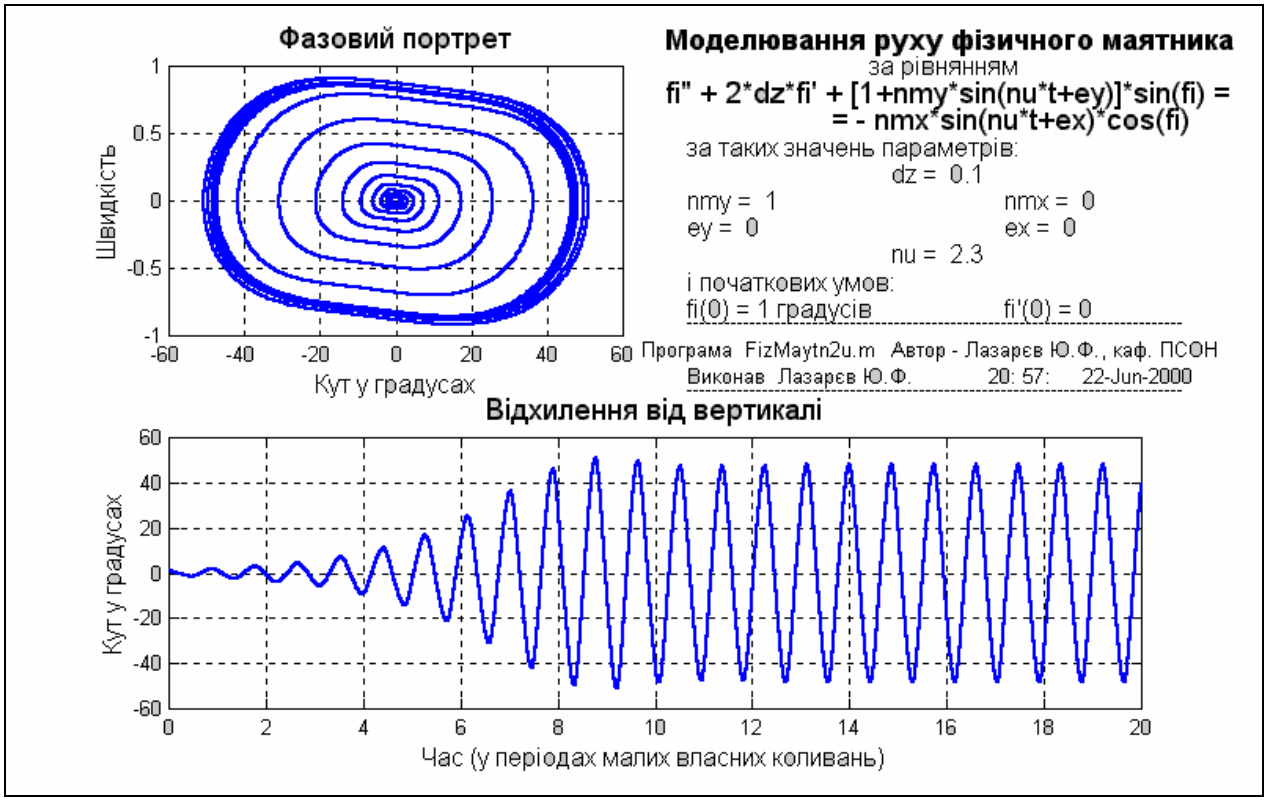

Рис. 2.12

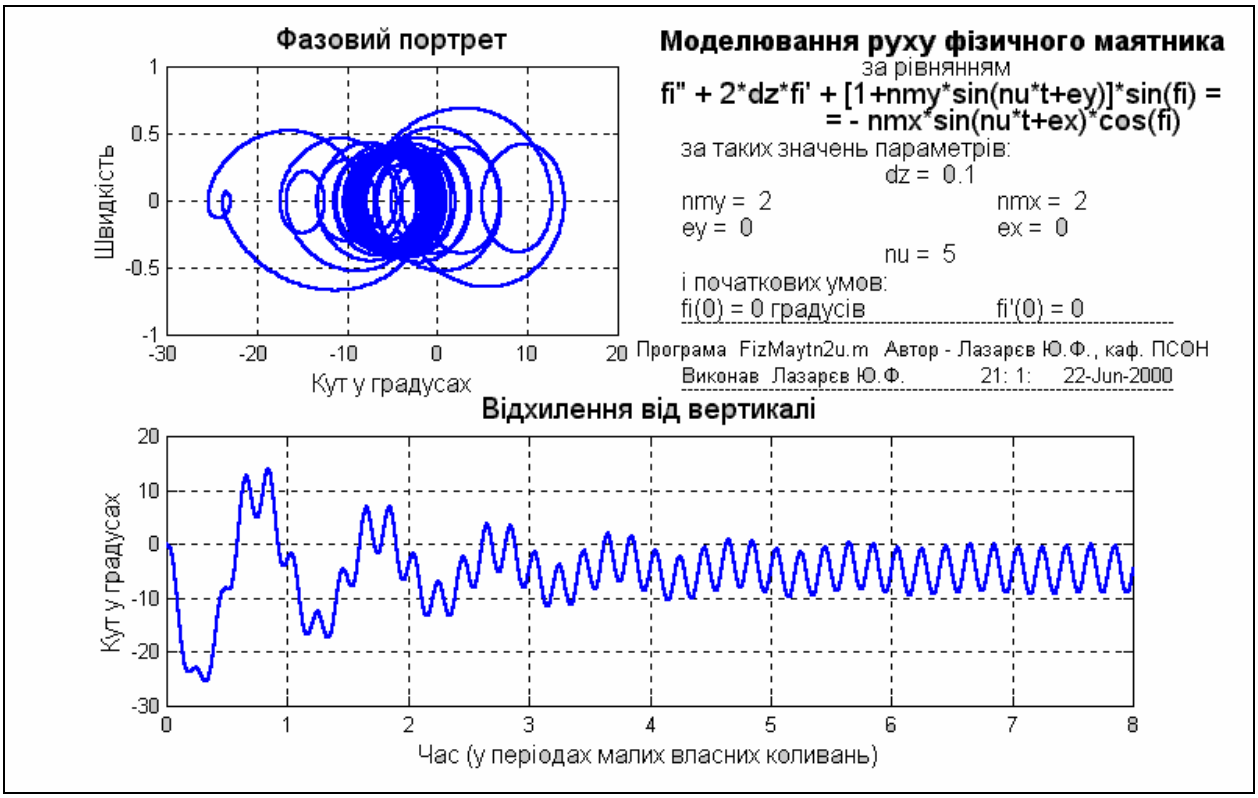

Рис. 2.13

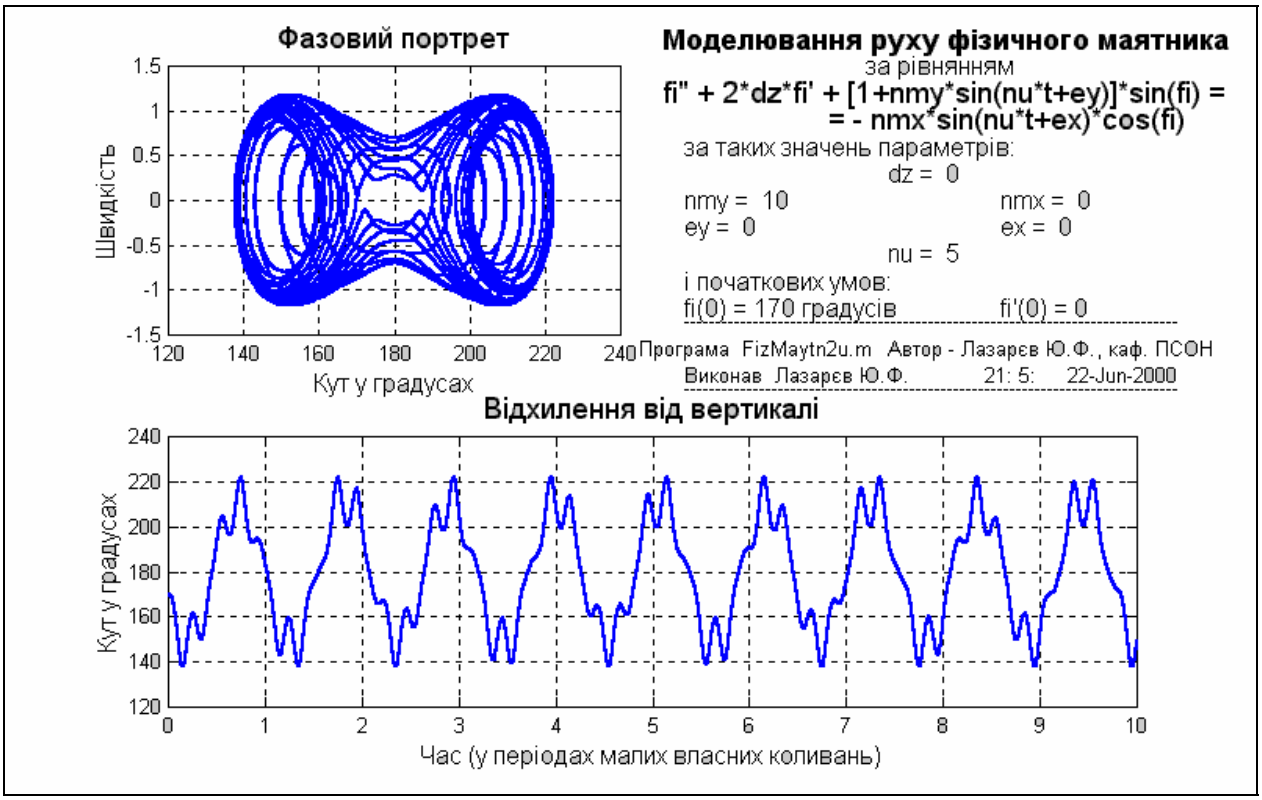

Рис. 2.14

Рис. 2.15

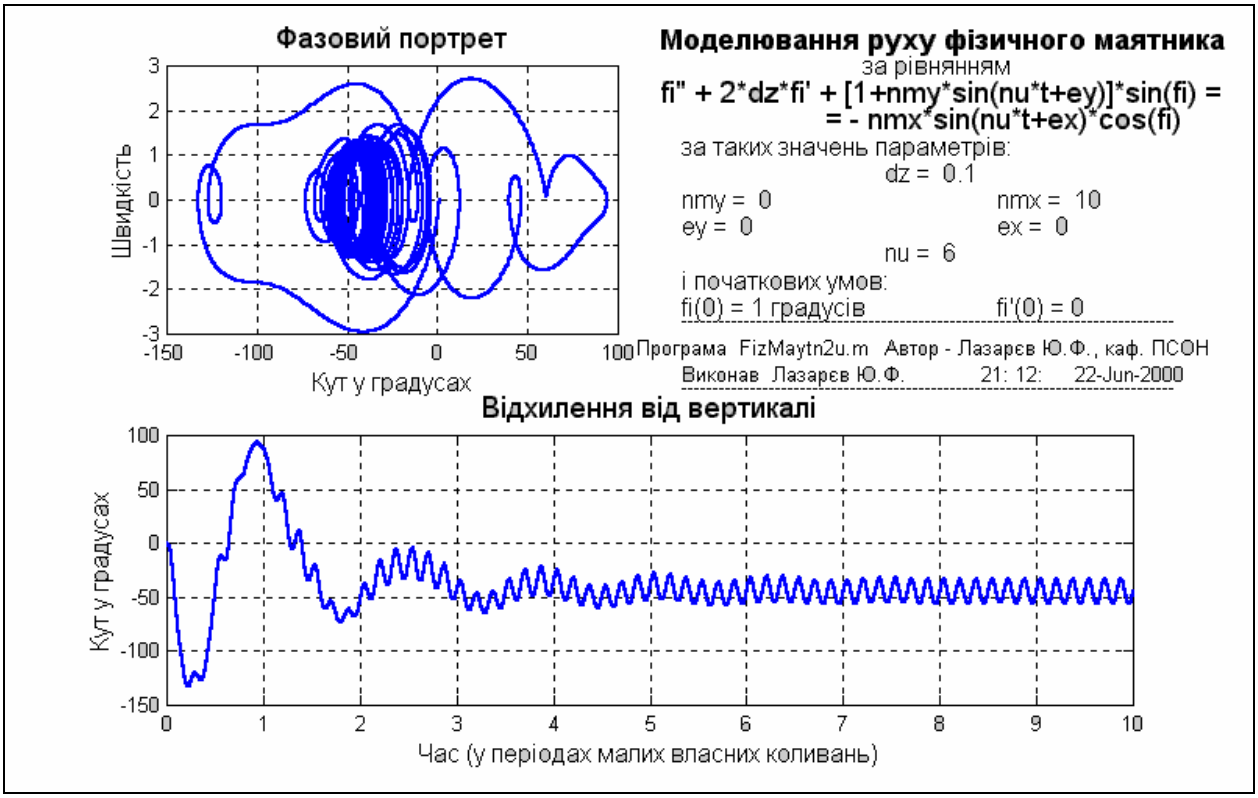

Рис. 2.16

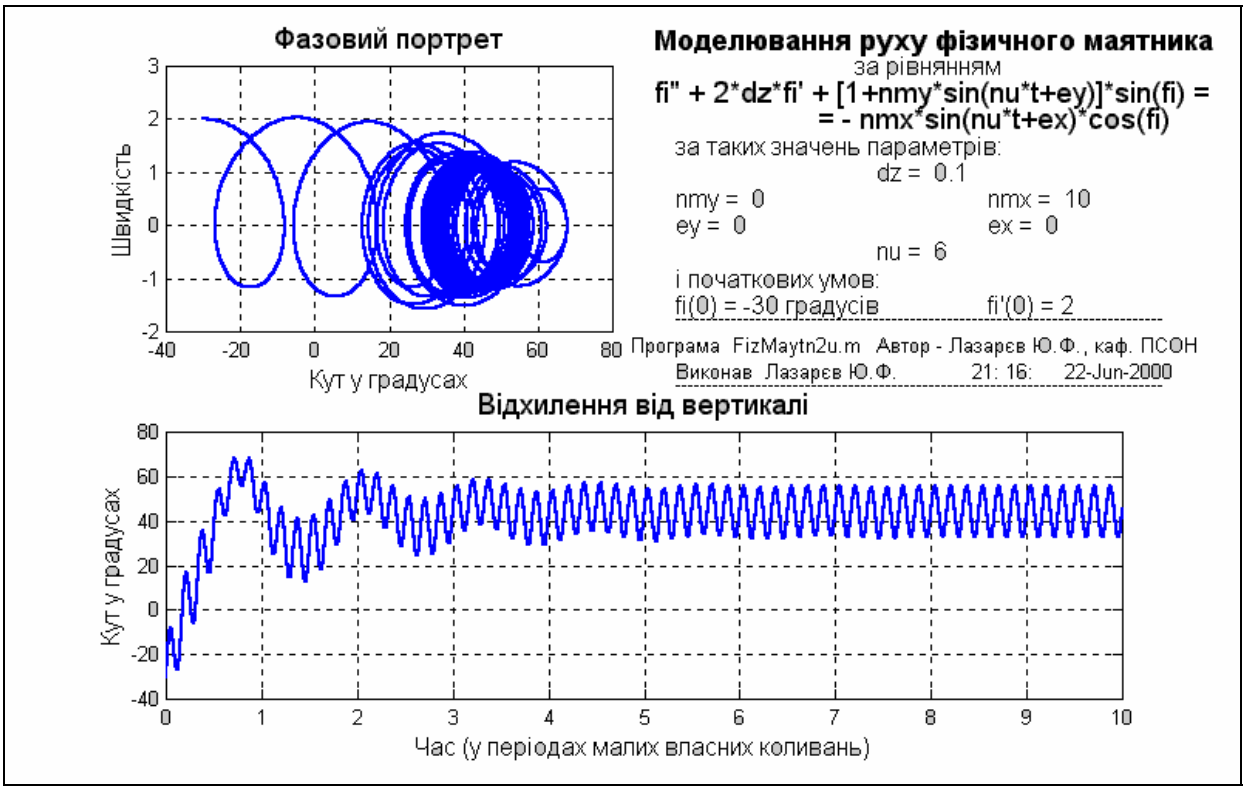

Рис. 2.17

Наконец, рис. 2.16 и 2.17 демонстрируют возможность существования значительных отклонений среднего положения маятника от вертикали и при чисто горизонтальной вибрации основания (явления, пока не описанного теоретически). 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.

2.7.2. Задания

При условиях задач 2.14 - 2.26 (разд. 2.6.3) составить программы численного интегрирования соответствующих дифференциальных уравнений, которые обеспечивали бы численное моделирование поведения заданной системы при заданных условиях и произвольных, устанавливаемых пользователем значениях начальных параметров. При этом обеспечить:

а) диалоговый ввод-изменение-вывод на экран значений всех начальных параметров;

б) вывод графиков изменений во времени важнейших выходных величин;

в) оформление графиков в соответствии с ранее указанными требованиями.

3. Интерфейс MatLAB и команды общего назначения. M-книги

Интерфейс – наиболее изменяемая от версии к версии часть системы MatLAB. Ниже приведено описание интерфейса версии 5.3.

3.1. Функции меню командного окна

Рассмотрим подробнее возможности, предоставляемые командным окном MatLAB. Напомним, что после запуска системы MatLAB на экране появляется командное окно в виде, представленном на рис. 3.1.

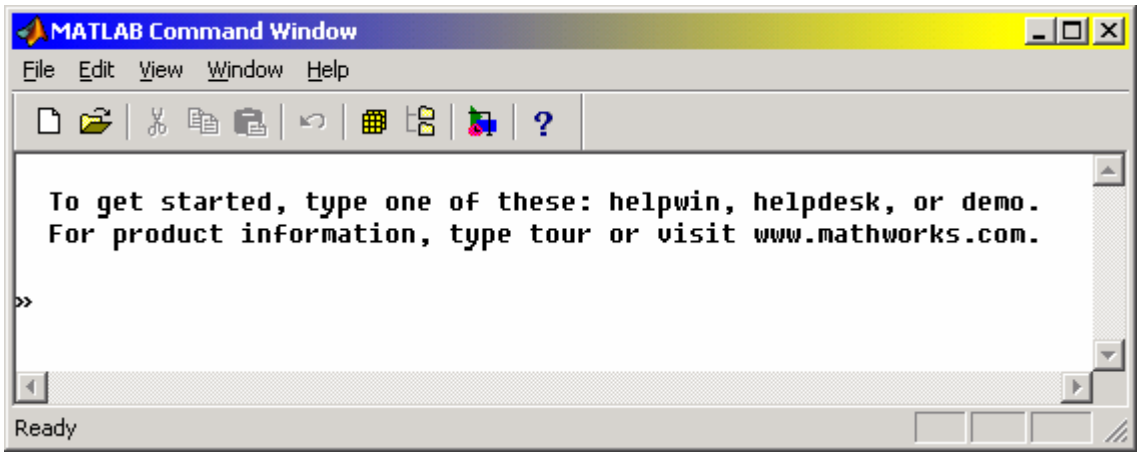

Рис. 3.1.

В верхней его части (непосредственно под заголовком **MATLAB Command Window**) расположена строка меню, которая состоит из четырех меню - **File**, **Edit**, **Window** и **Help**. Под главным меню помещена панель инструментов, которая позволяет выполнять некоторые наиболее употребляемые операции.

Открытие меню осуществляется "нажатием" клавиши мыши. Чтобы избрать какую-то команду меню, достаточно установить курсор мыши на имя команды и нажать левую клавишу мыши ("щелкнуть").

3.1.1. Меню File

Меню **File** (Файл) содержит команды, которые позволяют выполнять следующие задачи:

- создание, редактирование и запуск программ;
- управление рабочим пространством MatLAB;
- смена оформления графических и интерактивных (диалоговых) окон;
- управление путями доступа MatLAB и оформлением собственно командного окна;
- управление выводом на принтер;
- выход из системы MatLAB.

Команды разделены на группы в соответствия с назначением (рис. 3.2). Последняя группа содержит лишь одну **команду Exit MATLAB**. Она предназначена для завершения сеанса работы с системой MatLAB и выхода в среду Windows.

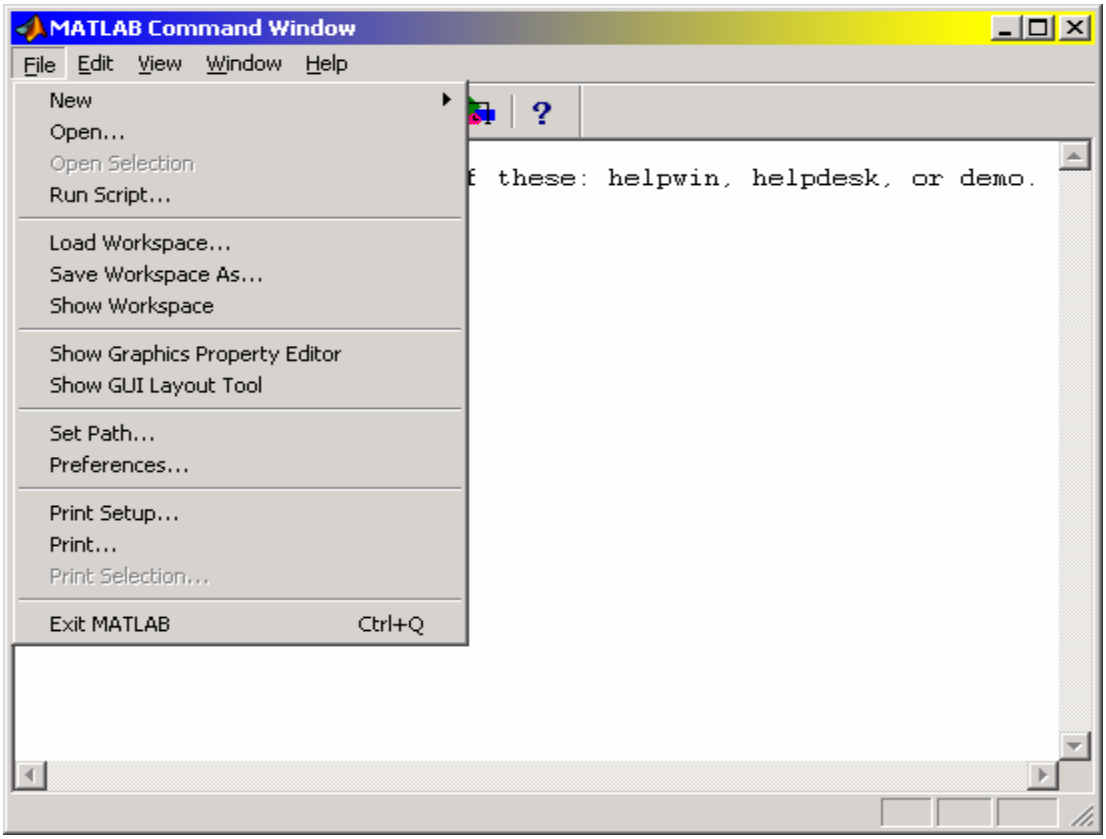

Рис. 3.2

Создание, редактирование и запуск программ

Первая группа команд меню **File** состоит из команд: **New**, **Open**, **Run Script**.

Команда **New** ("Новый ... ") позволяет перейти к созданию нового объекта среды MatLAB - нового М-файла (текстового файла на языке MatLAB), графического файла ("фигуры") или файла-модели (Model).

После избрания команды **New** открывается подменю, содержащее команды **M-file**, **Figure**, **Model**.

Вызов команды **M-file** приведет к появлению нового окна встроенного текстового редактора системы. В нем можно начать введение текста нового М-файла. После завершения ввода и редактирования текста следует записать его в виде файла с определенным именем на диск. Для завершения работы с редактором нужно избрать в меню ***File*** его окна команду ***Exit Editor/Debugger***.

Если в подменю команды **New** командного окна MatLAB избрать команду **Figure**, на экране возникнет графическое окно **Figure** и система готова к восприятию команд по оформлению этого графического окна.

Наконец, при выборе команды **Model**, система MatLAB переходит в интерактивный режим пакета SIMULINK (Моделирование связей) и на экране появятся два окна (рис. 3.3) - окно **Library Simulink**, которое позволяет избрать в инте-

рактивном режиме любую функцию из представленной библиотеки пакета SIMULINK, и окно редактора блок-схемы, в верхнем заголовке которого указанное имя будущего файла схемы связей (сейчас указано Untitled - Безымянный). Более подробно о создании MDL-файлов и их использовании будет изложено в седьмой главе пособия.

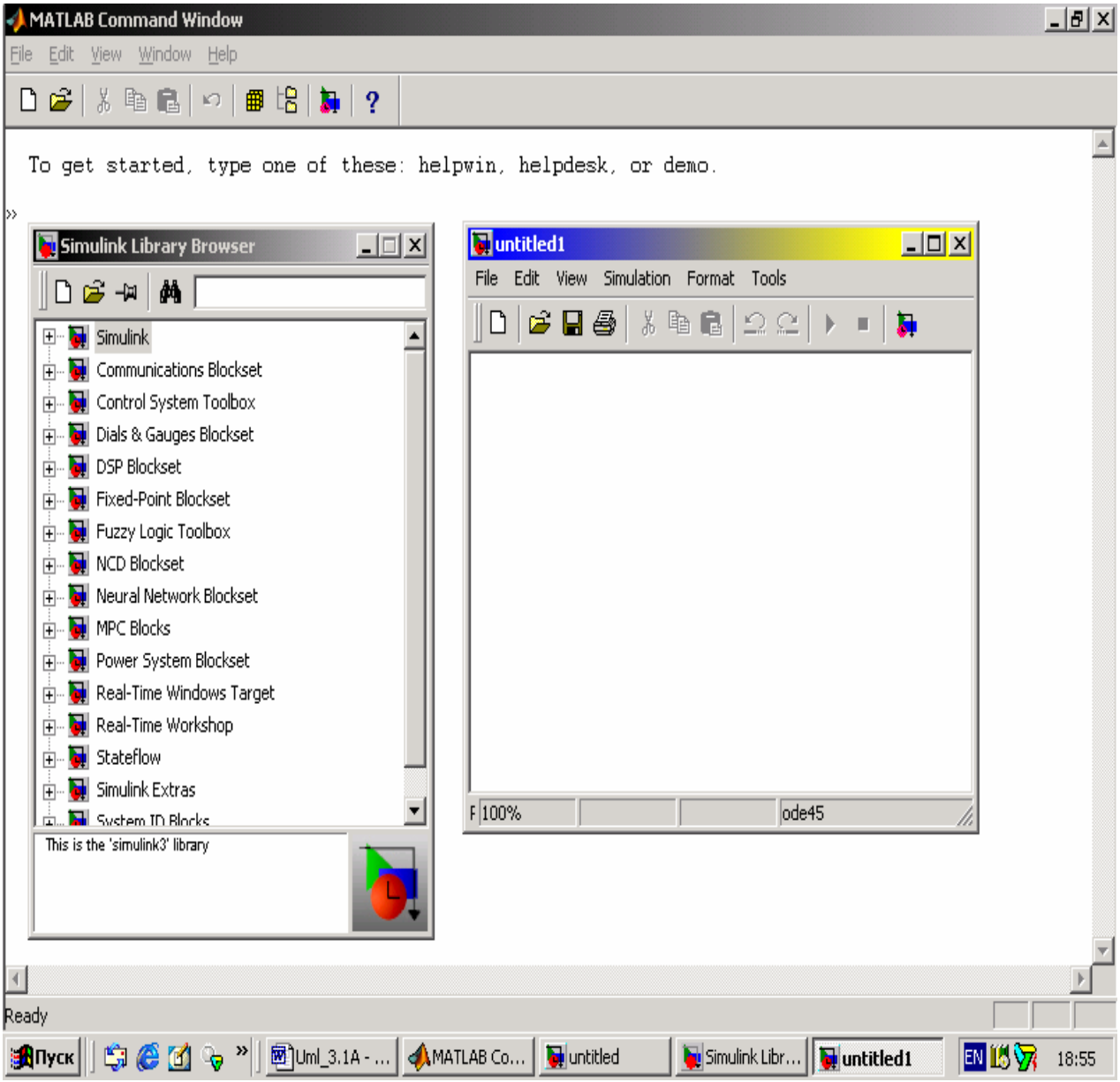

Рис. 3.3

Команда **Open** ("Открыть") открывает новое диалоговое окно с перечнем существующих в текущей директории М-файлов. Избрание из этого списка необходимого файла и последующее нажатие кнопки <ОК> приводит к появлению на экране окна предварительно установленного текстового редактора с текстом избранного М-файла. Теперь можно приступить к редактированию этого текста с последующей записью его на диск.

Вызов из меню **File** команды **Run Script** приводит к появлению на экране нового окна (рис. 3.4) с приглашением ввести имя М-файла с текстом программы, которую нужно запустить для выполнения. После введения в командном окне имени Script-файла и нажатия кнопки **ОК** эта программа будет запущена на выполнение. Команду удобно использовать в случае, когда вызывающий файл не содержится в директориях, указанных в путях, открытых для системы MatLAB (например, он находится в некоторой отдельной директории пользователя).

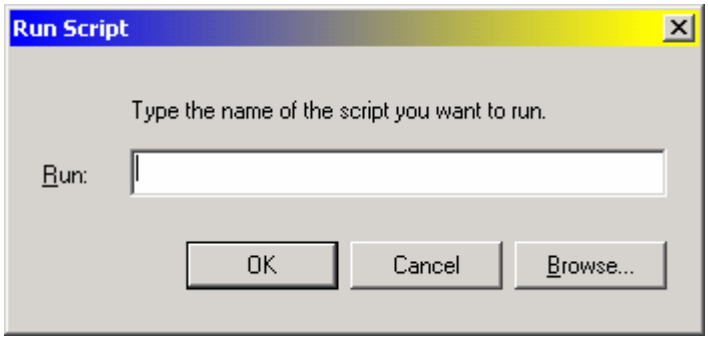

Рис. 3.4

Управление рабочим пространством MatLAB

Эта группа команд меню **File** состоит из трех - **Load Workspace**, **Save Workspace As** и **Show Workspace**.

Команда **Load Workspace** (Загрузить рабочее пространство) позволяет воспользоваться данными (загрузить в рабочее пространство системы), сохраненными на диске в виде так называемых MAT-файлов. Если вызвать эту команду, на экране возникает окно **Load .mat file** (рис. 3.5), в котором будет показано содержимое текущей папки с точки зрения наличия в ней MAT-файлов.

Избрав с помощью этого окна нужный MAT-файл и вызвав его с помощью мыши, осуществляют дополнение существующего рабочего пространства системы переменными и их значениями из указанного MAT-файла.

Команда **Save Workspace As** ("Записать рабочее пространство как") осуществляет запись существующего рабочего пространства системы на диск в виде MAT-файла. После избрания этой команды на экране возникает окно **Save Workspace As** (рис. 3.6), с помощью которого выбирается нужная папка, записывается имя MAT-файла и осуществляется запись в него существующего рабочего пространства.

Команда **Show Workspace** (Показать рабочее пространство) позволяет просмотреть перечень переменных, содержащихся в рабочем пространстве, и их параметры. После вызова этой команды на экране возникает окно **MATLAB Workspace** (рис. 3.7), в котором содержится полный перечень переменных рабочего пространства.

Примечание. *В начале нового сеанса работы с системой рабочее пространство является пустым*.

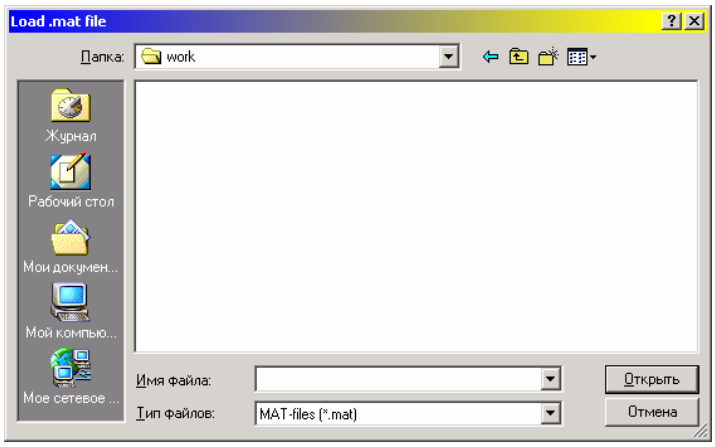

Рис. 3.5

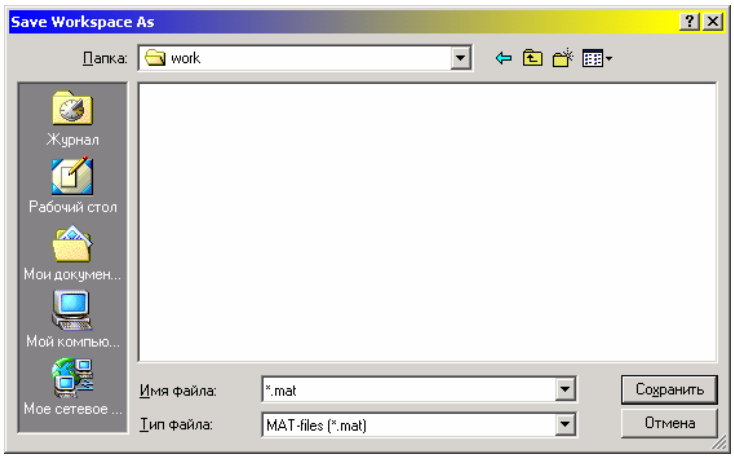

Рис. 3.6

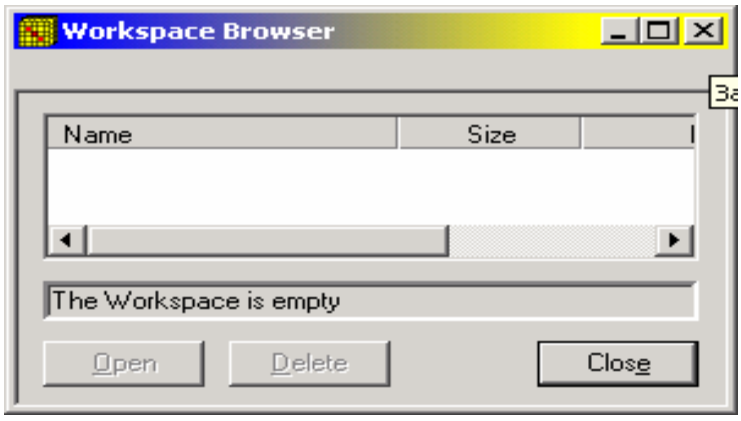

Рис. 3.7

Изменение оформления графических и интерактивных окон

Эта совокупность команд меню **File** вызывает специальные интерактивные редакторы, которые позволяют изменять параметры, определяющие стиль графического оформления графических и интерактивных (диалоговых) окон, используемых системой MatLAB.

Например, вызов команды **Show Graphics Property Editor** (Показать редактор графических свойств) приводит к появлению на экране окна **Graphics**

Property Editor (рис. 3.8), пользуясь которым, можно изменить некоторые установленные ранее свойства оформления графических окон.

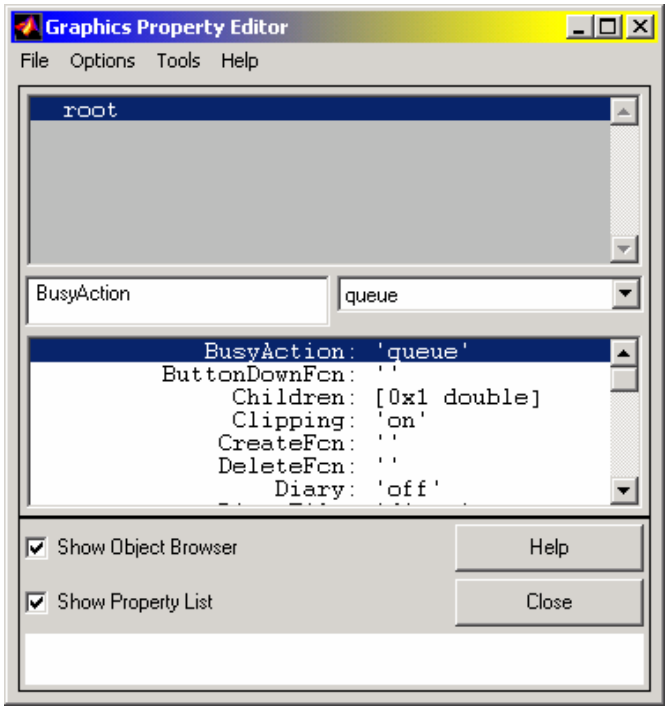

Рис. 3.8

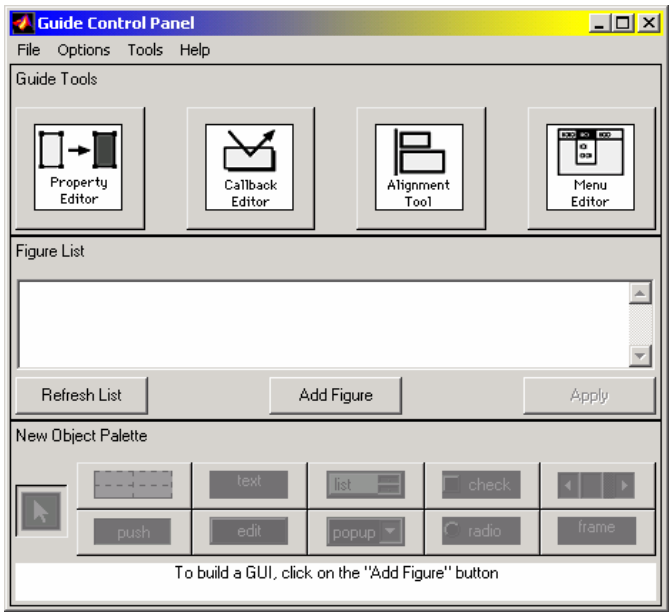

Рис. 3.9

Команда **Show GUI Layout Tool** (Показать средства оформления графического интерфейса пользователя) позволяет открыть окно **Guide Control Panel** (рис. 3.9), пользуясь которым можно создавать собственные диалоговые графические окна, которые содержат кнопки, нисходящие меню, графические и текстовые подокна, с помощью которых можно обеспечивать удобный диалог с машиной.

Управление путями доступа MatLAB и оформлением командного окна

Для этой цели предусмотрены две команды - **Set Path** (Установка путей) и **Preferences** (Свойства).

Первая команда **Set Path** предназначена для введения в перечень путей доступа системы MatLAB, автоматически проверяемых системой при поисках файлов, новых путей по желанию пользователя. При вызове этой команды на экране появляется окно **MATLAB Path** (рис. 3.10), с помощью которого пользователь осуществляет изменение путей доступа системы по собственному усмотрению.

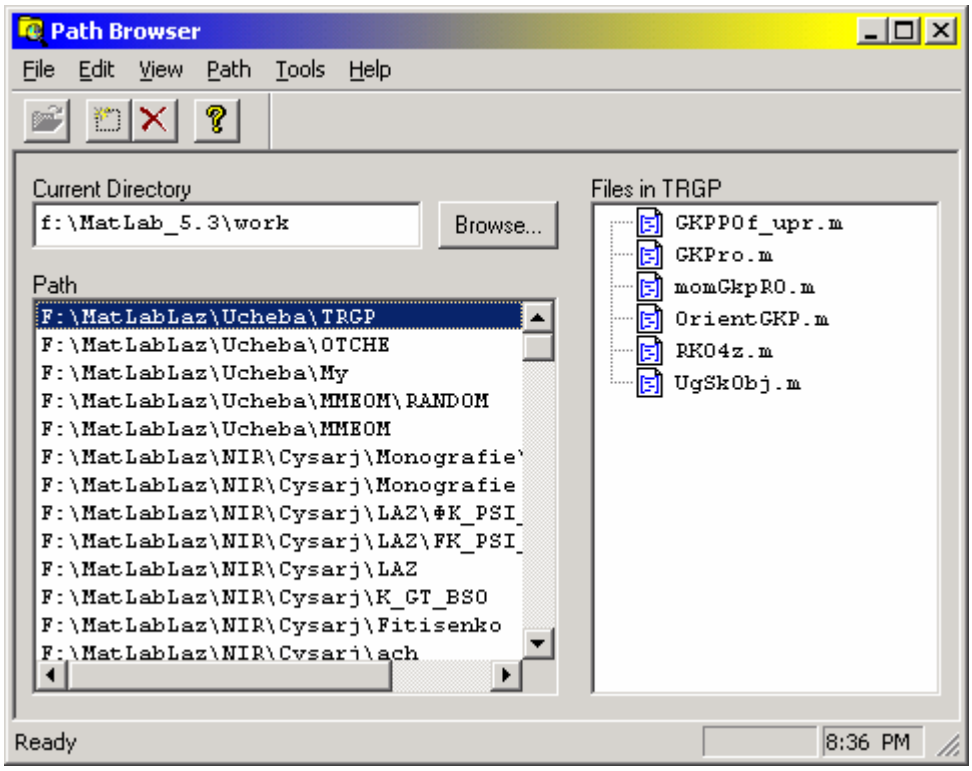

Рис. 3.10

Вызов команды **Preferences** ("Свойства") приводит к появлению окна с таким же названием (рис. 3.11). Как видно, окно состоит из трех вкладышей **General**, **Command Window Font** и **Copying Ohtions**.

Вкладка **General** (Общие) содержит несколько областей: **Numeric Format**, **Editor Preference**, **Help Directory**.

Область **Numeric Format** 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 шестнадцатиричого 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 плавающей запятой;

Rational - запись в виде рациональной дроби.

Выбирая нужный формат представления чисел, можно обеспечить в дальнейшем вывод чисел в командное окно именно в этой форме.

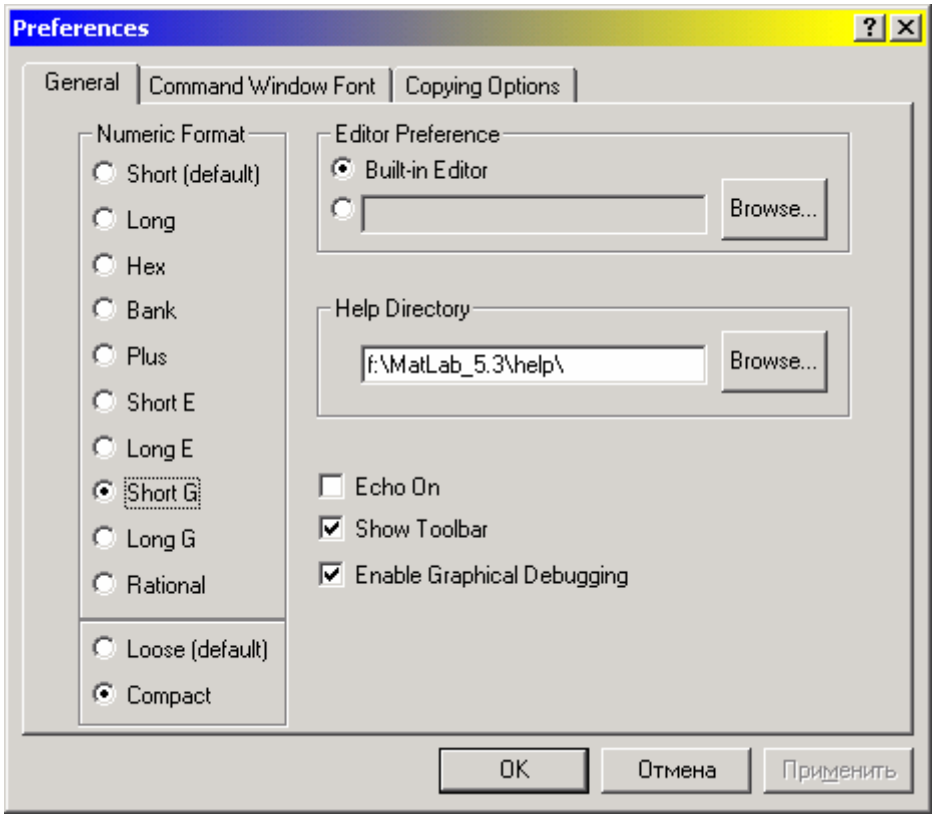

Рис. 3.11

Кроме отмеченных форматов представления чисел в этой области расположенные еще две опции **Loose (default)** и **Compact**, которые определяют форму вывода результатов в командное окно. По умолчанию применяется режим Loose, за которым отдельные строки выводятся на экран через одну пустую строку. Установление же режима Compact позволяет выводить информацию в командное окна более плотно, без пропуска строк.

Область **Editor Preference** (Свойства редактора) позволяет выбрать текстовый редактор, в котором будут представляться и редактироваться тексты всех M-файлов. Система MatLAB, начиная с версии 5.0, имеет собственный встроенный текстовый редактор MATLAB Editor/Debugger с отладчиком. В области **Editor Preference** можно поменять его на любой другой текстовый редактор. Например, удобным и простым в пользовании с кириллицей является редактор Notepad.

В области **Help Directory** можно изменить имя папки, в которой расположены файлы справок (Help-файлы).

Кроме перечисленных, в окне **Preferences** есть еще три опции **Echo On** (Включить эхопечать), **Show Toolbar** (Показать панель инструментов) и **Enable Graphical Debugging** (Включить графический отладчик). Чтобы активизировать опцию, нужно щелкнуть на ней мышью. После этого рядом с ее надписью появит-

ся пометка в виде галочки. Если галочка стоит рядом с командой **Echo On**, то при выполнении текстового М-файла одновременно с выполнением программы ее текст будет постепенно выводиться в командное окно (так называемая "эхопечать"). Аналогичная пометка против команды **Show Toolbar** приводит к тому, что под линейкой главного меню будет размещена еще одна, дополнительная панель инструментов (рис. 3.1). Пометка рядом с командой **Enable Graphical Debugging** означает, что выполнение графических операций будет сопровождаться их отладкой с помощью специальной программы-отладчика. Если же соответствующие пометки отсутствуют, то указанные действия не проводятся.

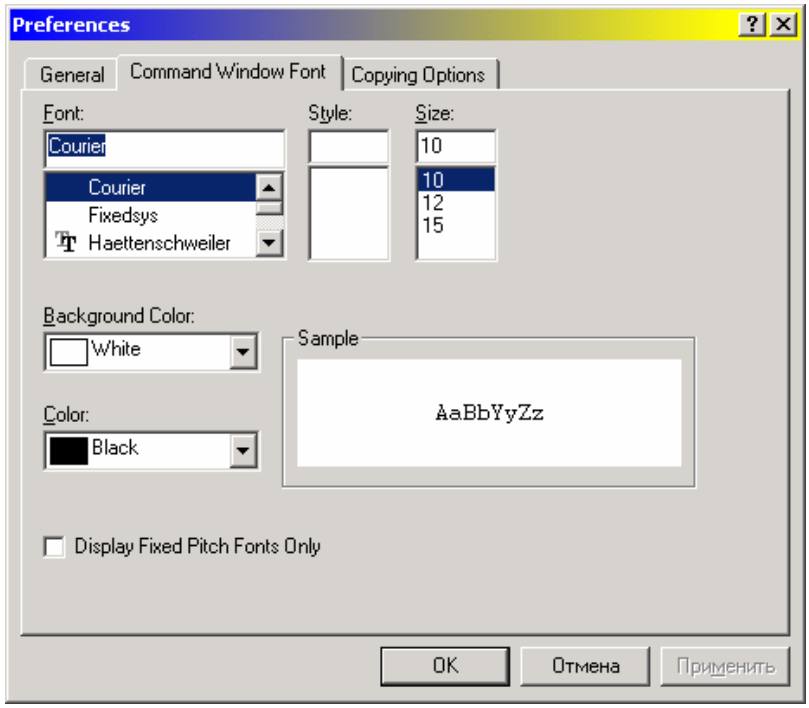

Рис. 3.12

Вкладка **Command Window Font** имеет вид, приведенный на рис. 3.12. В ней можно изменить

- тип шрифта, которым выводится текст в командное окно (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**);

Управление выводом на печать

Рассмотрим еще одну группу команд меню **File**, которая осуществляет управление выводом информации на принтер. Она состоит из двух команд **Print Setup** и **Print**.

Если выбрать команду **Print Setup**, на экране возникнет окно **Настройка печати** (рис. 3.13), которое позволяет выбрать тип принтера, размер листа бумаги,

вид подачи бумаги на печать, ориентацию печати относительно листа (книжная или альбомная). Кроме того, нажав кнопку **Свойства** в этом окне, можно перейти к окну **Свойства принтера** (рис. 3.14), с помощью которого можно также изменять точность и плотность, то есть качество печати. Вид этого окна определяется типом принтера.

Непосредственно печать осуществляется путем избрания команды **Print**. При этом на экран выводится изображение окна **Печать** (рис. 3.15). Это окно позволяет выбрать нужный вид принтера, установить по номерам страниц необходимый диапазон, количество копий и т.п.

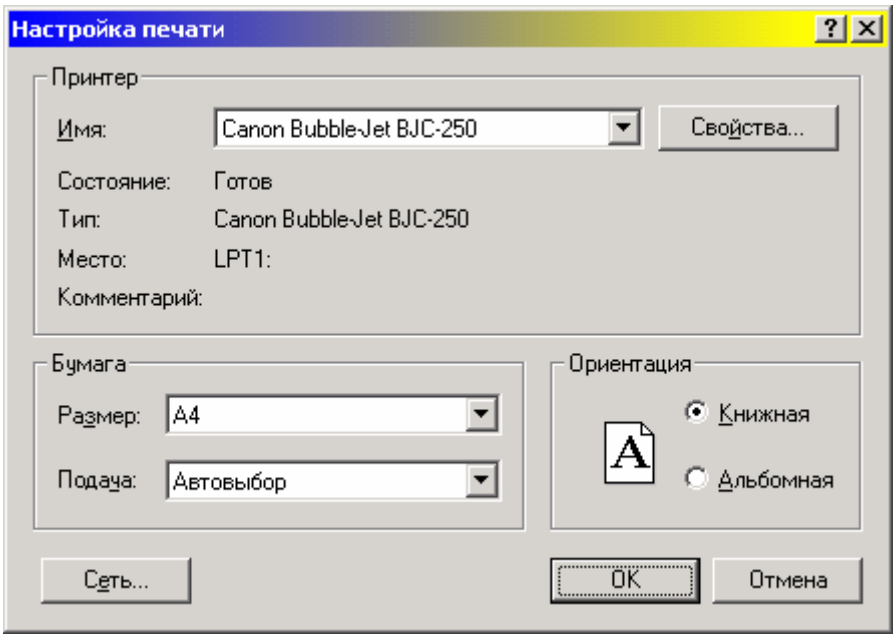

Рис. 3.13

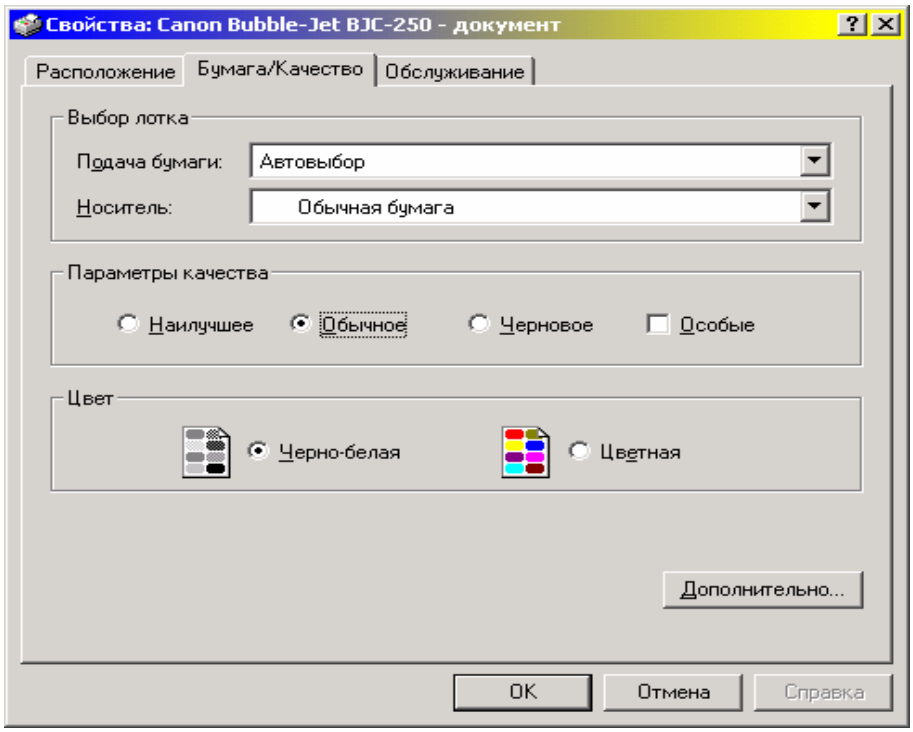

Рис. 3.14

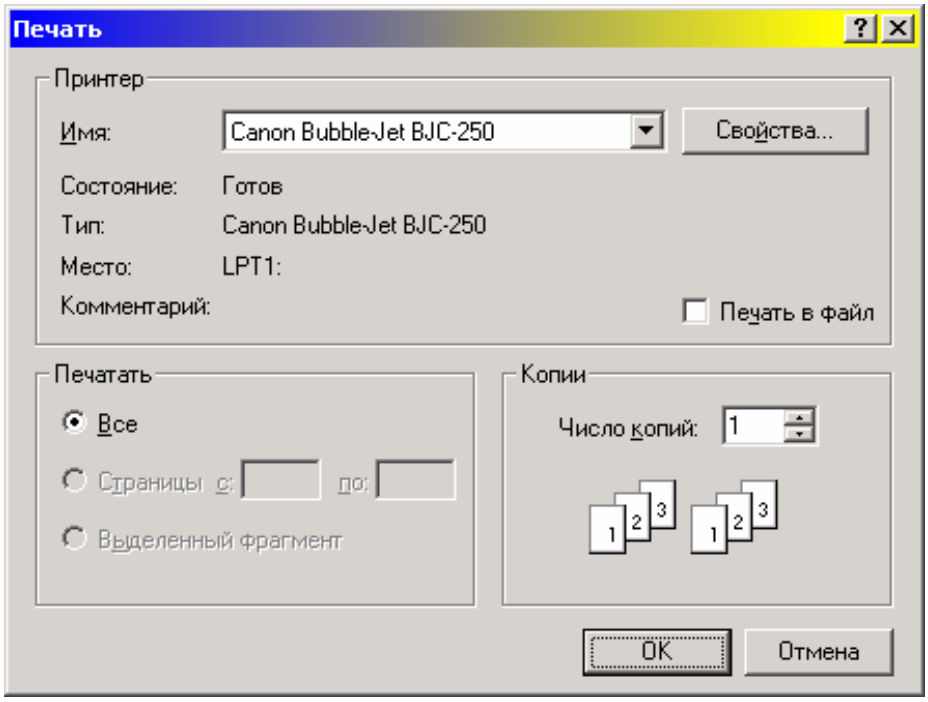

Рис. 3.15

3.1.2. Другие меню командного окна

*Меню **Edit***

Меню **Edit** (Правка, рис. 3.16) содержит семь команд:

- *Undo* (Отменить предыдущую команду);
- *Cut* (Вырезать);
- *Copy* (Скопировать);

- *Paste* (Вставить);
- *Clear* (Очистить);
- *Select All* (Отметить все);
- *Clear Session* (Очистить командное окно).

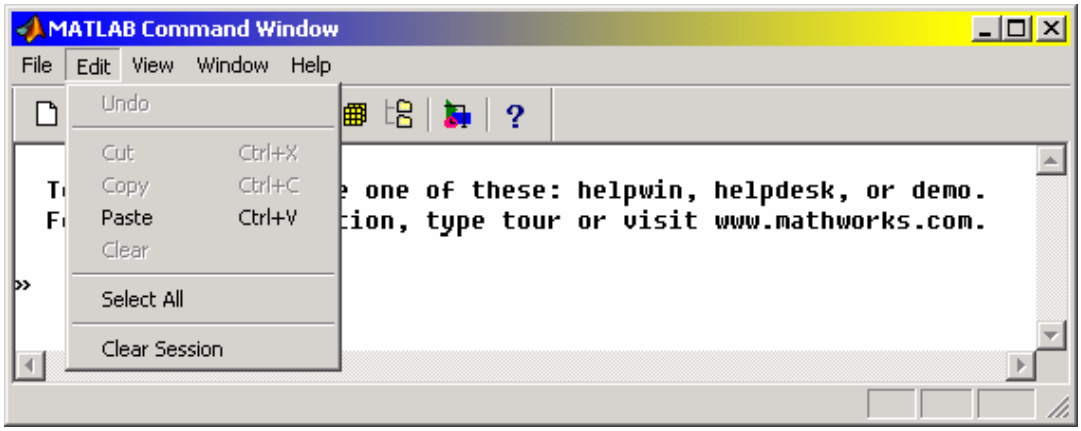

Рис. 3.16

В начале сеанса работы в MatLAB обычно можно воспользоваться только командой **Clear Session**, которая позволяет очистить командное окно MatLAB от всей информации в нем, оставляя в нем лишь знак ‘ » ’ готовности к восприятию новой команды. Тем не менее, когда в командном окне есть какой-то текст, то часть этого текста можно *запомнить* и перенести в любое другое окно Windows. Для этого достаточно *подвести курсор миши на начало текста, который нужно запомнить, нажать левую* клавишу *миши и, не отпуская клавиши, перевести курсор на конец текста*. При этом текст будет выделен другим цветом. Теперь, переходя к меню **Edit** командного окна, надо избрать в нем *команду* ***Copy*** - выделенный *текст будет скопирован в буфер памяти*. Если теперь перейти в любое окно Windows и нажать клавиши <Ctrl+V>, текст будет записан в это окно. Так можно переносить тексты программ MatLAB в документы Word (или другого редактора).

Чтобы, наоборот, перенести часть текста из окна Word в командное окно, надо запомнить в буфере текст Word, перейти к командному окну MatLAB и избрать команду **Paste** меню **Edit**.

Меню ***Windows***

Здесь содержится перечень открытых в среде MatLAB окон. Чтобы перейти к нужному окну, достаточно выбрать его в этом перечне.

Меню ***Help***

Выбор этого меню приводит к появлению на экране совокупности команд обращения к системе подсказок (справок) среды MatLAB (рис. 3.17).

Среди них такие команды:

Help Window (Окно помощи (справки));

Help Tips (Темы справок);

Help Desk (HTML) (Стол справок);
Examples and Demos (Примеры и демонстрационные программы);
About MATLAB (Про MATLAB);

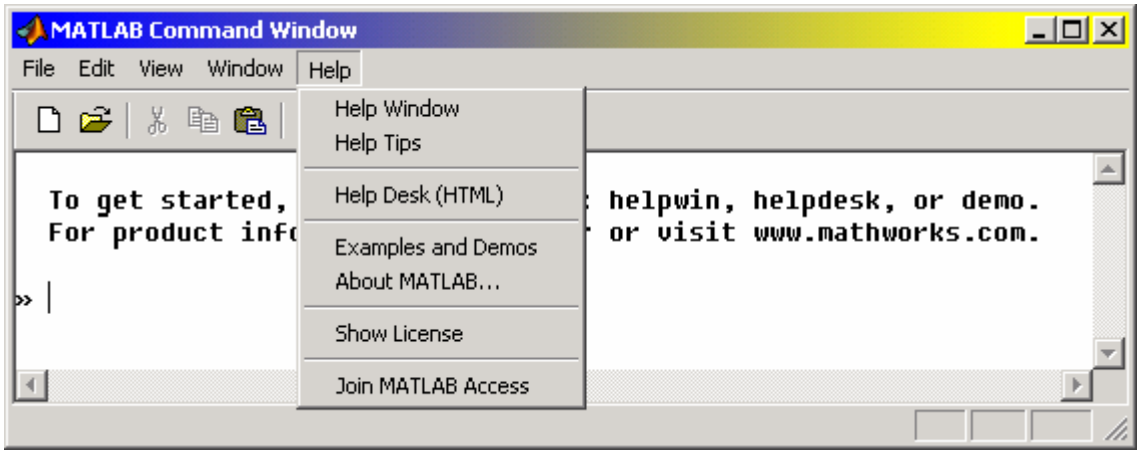

Рис. 3.17

Команда **Help Window** вызывает появление на экране окна **MATLAB Help Window**, которое содержит перечень файлов-содержаний, в которых расположены перечни названий процедур и функций. Выбор сначала одного из файлов-содержаний приводит к появлению в том же окне перечня процедур, которые входят в его состав, а последующий выбор названия одной из процедур - к появлению на экране окна с описанием этой процедуры (на английском языке).

Команда **Help Tips** вызовет возникновение на экране того же окна **MATLAB Help Window**, которое позволяет ознакомиться со всеми функциями и процедурами MatLAB общего назначения по темам, как в учебном пособии.

Выбор команды **Help Desk (HTML)** вызовет появление окна **MATLAB Help Desk**, которое привлекает для справок диалоговую систему, оформленную в формате HTML.

Команда **Examples and Demos** приводит к появлению окна **MATLAB Demo Window** (рис. 3.19), в котором в интерактивном режиме можно ознакомиться с примерами применения основных процедур системы MatLAB, ее вычислительными и графическими возможностями, а также с текстами программ, с помощью которых достигаются демонстрируемые результаты.

Команда **About MATLAB** вызовет появление на экране логотипа системы MatLAB.

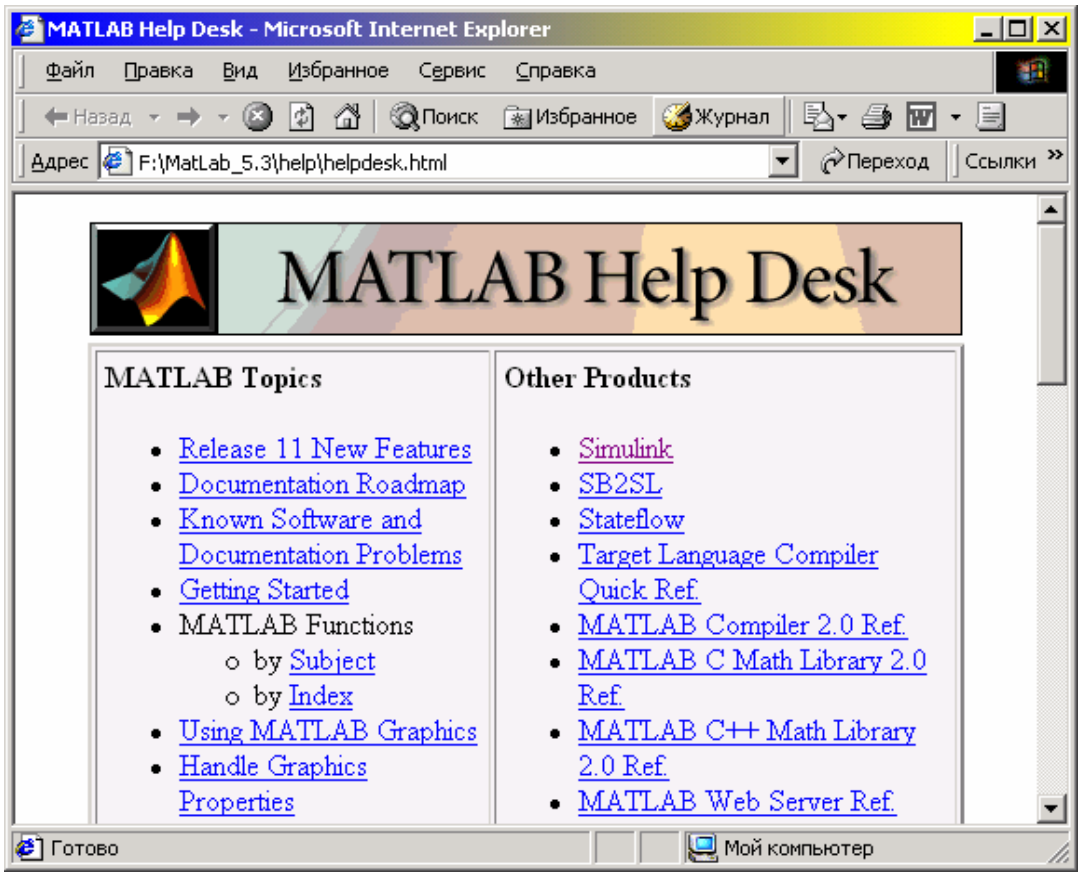

Рис. 3.18

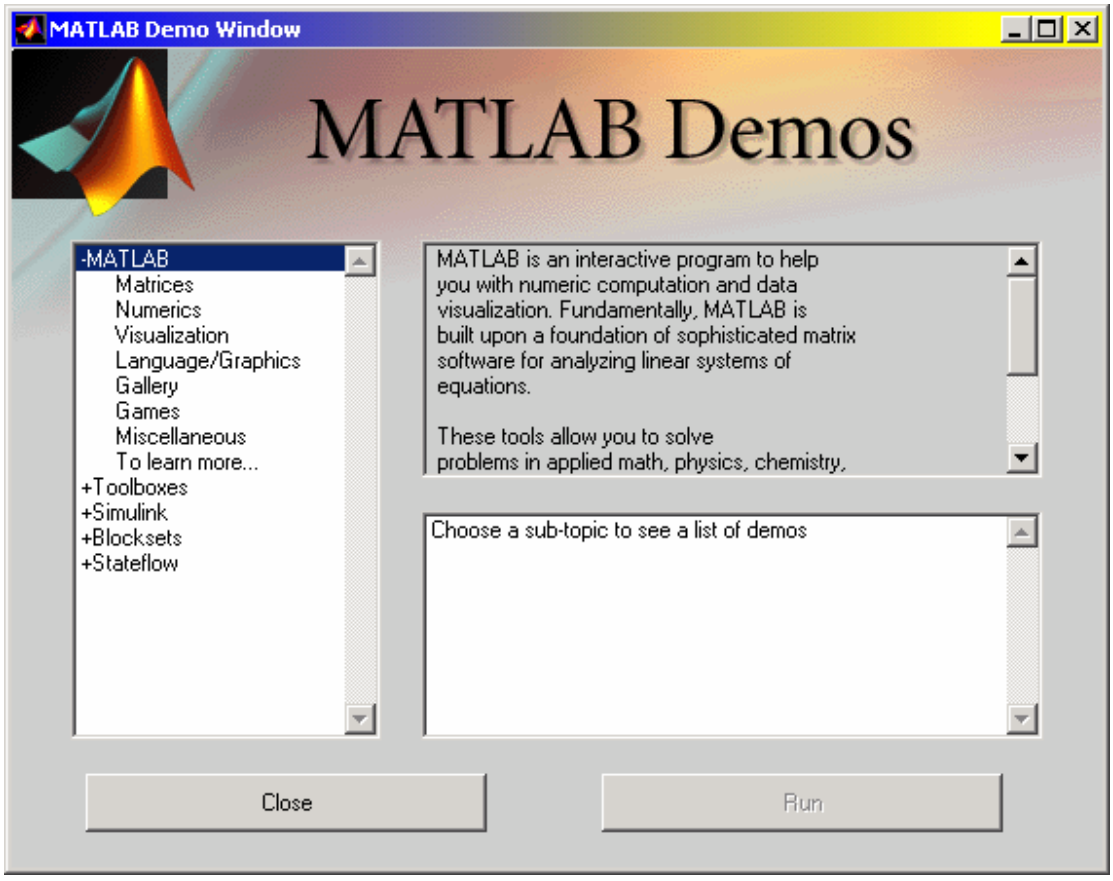

Рис. 3.19

3.1.3. Панель инструментов

Кроме линейки главного меню в командном окне MatLAB ниже ее может располагаться так называемая панель или линейка инструментов (см. рис. 3.1). Панель инструментов содержит десять пиктограмм, которые дублируют наиболее употребляемые команды главного меню командного окна.

Первая слева пиктограмма вызывает пустое окно установленного текстового редактора системы, то есть ее действие эквивалентно выбору команды **New M-file** (Создать новый М-файл). Выбор *второй* пиктограммы эквивалентно действию команды **Open M-file** (Открыть существующий М-файл). *Третья* пиктограмма выполняет ту же функцию, что и команда **Cut** (Вырезать). *Четвертая* – аналогична команде **Copy** (Скопировать). *Пятая* - команде **Paste** (Вставить). *Шестая* пиктограмма действует как команда **Undo** (Возвратиться к предыдущей команде). *Седьмая* выполняет команду **Show Workspace** (Показать рабочее пространство). *Восьмая* - команду **Set Path** (Установить путь доступа). Действие *девятой* пиктограммы такое же, как у команды **New Model**, то есть она подготавливает систему к созданию новой Simulink-модели. Наконец, *десятая* пиктограмма эквивалентна команде **Help Window**.

Если установить курсор мыши на соответствующую пиктограмму, рядом с ней со временем появится название той команды, которую эта пиктограмма выполняет. Чтобы активизировать пиктограмму, следует подвести к ней курсор и нажать левую клавишу мышки.

3.2. Команды общего назначения

Команды общего назначения набираются с клавиатуры. Текст их возникает в командном окне по мере набора рядом со знаком приглашения (>>). Выполняются они после нажатия клавиши <Enter>.

Эти команды удобно разделить на такие группы:

1) управляющие команды и функции;
2) команды управления переменными и рабочим пространством;
3) команды работы с файлами и операционной системой;
4) команды управления командным окном;
5) команды запуска и выхода с MatLAB;
6) команды получения общей информации.

Рассмотрим вкратце 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 М-, MAT- и MEX-файлов в текущей папке;

type - вывод на экран текста указанного М-файла;

lookfor - поиск программы (функции) по указанному ключевому слову;

which - вывод на экран полного пути расположения 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 представления 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 ПК 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 указанной папки;

delete - уничтожить (стереть) указанный файл;

getenv - вывести значение 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 - перевести курсор на начало страницы;

format - установить указанный формат вывода чисел на экран;

echo - установить или отменить режим эхопечати текста выполняемой программы;

more - установить режим постраничного вывода текста на экран командного окна.

Команды запуска и выхода с MatLAB:

quit - выйти с MatLAB;
startup - запуск MatLAB через М-файл "startup";
matlabrc - запуск главного стартового М-файла.

Команды получения общей информации:

info - получение информации про MatLAB и фирму MathWorks, Inc.;
subscribe - подписка по Internet как пользователя MatLAB;
whatsnew - информация о новых особенностях, которые не вошли в документацию;
version - информация о поставленной версии MatLAB;
ver - информация о версиях всех программных продуктов, которые входят в поставленный комплект системы MatLAB.

3.3. Создание М-книги

Очень полезным и привлекательным свойством системы MatLAB является возможность создания текстовых документов в среде редактора Word с одновременным проведением в нем вычислений с помощью системы MatLAB и фиксированием результатов вычислений (в том числе - графиков) в тексте документа Word. Благодаря этому можно создавать сложные научно-расчетные и инженерные текстовые документы непосредственно в редакторе Word.

Средством, которое позволяет это сделать, является пакет NoteBook, входящий в систему MatLAB. Этот пакет связывается с редактором Word с помощью специального Word-шаблона, который содержится в системе MatLAB. Для того чтобы можно было создавать М-книги, нужно, чтобы этот шаблон, носящий имя ***M-book. dot***, был предварительно подсоединен к редактору Word.

3.3.1. Начало новой М-книги

Чтобы приступить к написанию новой М-книги, нужно:

1) запустить редактор Word;
2) выбрать в диалоговом окне Word опцию **New** из меню **File**;
3) в окне, которое появится на экране, выбрать шаблон **M-book**.

В результате этих действий будет запущена система MatLAB, и вид главного меню редактора Word несколько изменится - в нем появится новое меню **Notebook**. Это и будет свидетельствовать, что к Word присоединена система MatLAB. Если теперь с помощью мыши активизировать меню **Notebook** окна Word, на экране появится дополнительное меню (рис. 3.20).

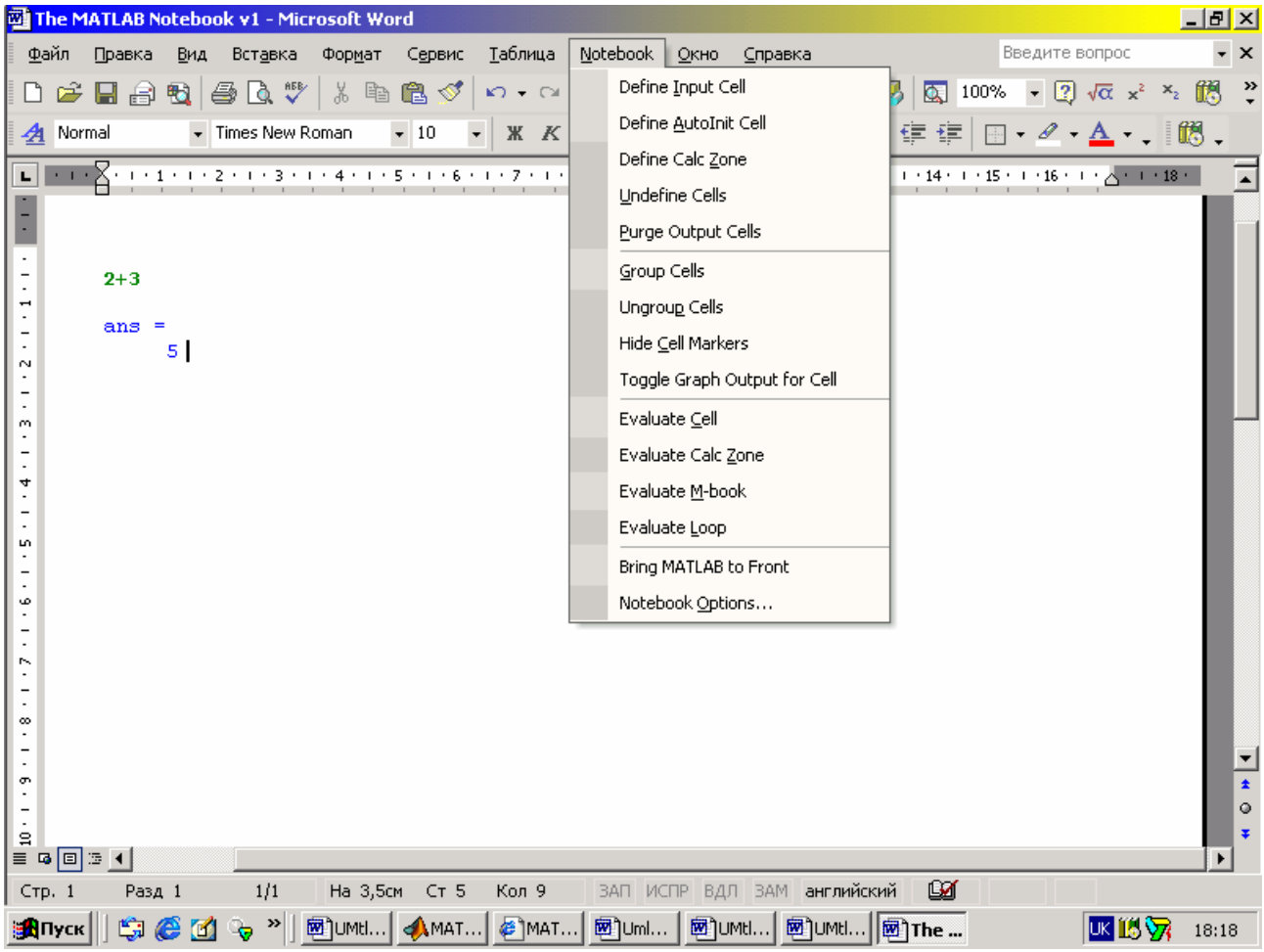

Рис. 3.20

3.3.2. Написание М-книги

Написание М-книги связано с набором текста, а также операторов и команд MatLAB. Введение текста осуществляется по обычным правилам редактора Word.

Чтобы ввести и выполнить команду MatLAB, необходимо:

1) написать текст команды в виде отдельной строки;

2) после набора строки с командой не нажимать клавишу <Enter> (курсор должен остаться в строке команды);

3) выбрать команду **Define Input Cell** (Определить Входную Ячейку) в меню **Notebook** (см. рис. 3.20), или нажать клавиши <Alt+D>; после этого вид строки команды должен измениться - символы команды приобретают темно-зеленый цвет, а команда становится отороченной квадратными скобками темно-серого цвета;

4) выбрать мышкой команду **Evaluate Cell** (Вычислить ячейку), или нажать комбинацию клавиш <Ctrl+Enter>; результатом этих действий должно стать появление сразу после текста команды результатов ее выполнения системой MatLAB.

Результаты выполнения команды выводятся, синим цветом и взяты в квадратные скобки.

Приведем пример. Пусть вы набрали в Word строку

A = [1 2 3; 4 5 6; 7 8 9]

Тогда после нажатия <Alt +D> эта строка изменит свой вид

```
[ A = [1 2 3; 4 5 6; 7 8 9] ]
```

а после нажатия <Ctrl+Enter> в следующих строках появится результат

```
[ A =
     1     2     3
     4     5     6
     7     8     9  ]
```

Если желательно выполнить несколько команд MatLAB одну за другой, наберите их несколькими строками по правилам написания текста программ, *выделите эти строки*, как это делается при копировании части текста в Word, и повторите вышеупомянутые 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. 3.21).

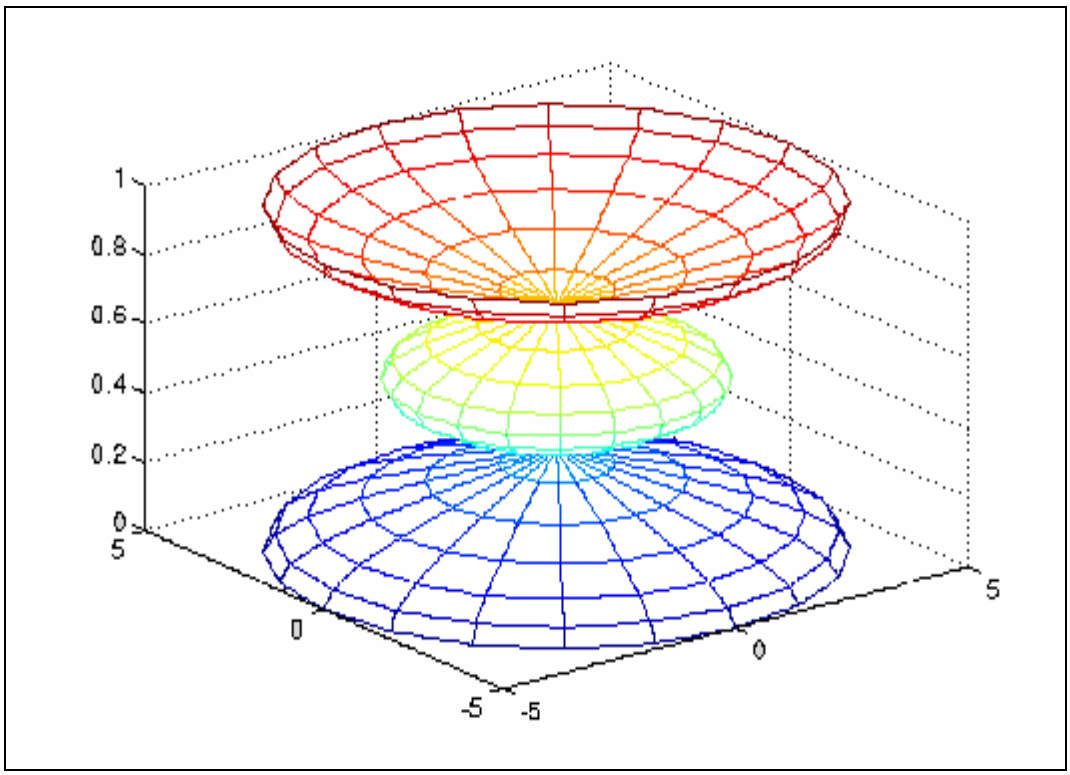

Рис. 3.21

Чтобы *оставить в тексте* документа *введенные команды и* выведенные *результаты*, нужно:

1) поместить курсор мышки в одну из строк выполненной команды;

*2) выбрать команду **Undefine Cells** из меню **Notebook** или нажать комбинацию клавиш <Alt+U>.*

В результате все символы, как введенных команд, так и результатов их выполнения приобретут обычный для текста Word стиль, цвет и размеры, и исчезнут квадратные скобки, которые их окаймляли.

3.3.3. Редактирование М-книги

Чтобы откорректировать существующую М-книгу или внести в ее какие-то дополнения, надо выполнить одно из следующих действий:

- войти в редактор Word и открыть, используя команду **Open** из меню **File** окна Word, файл М-книги, которую нужно корректировать;
- войти в редактор Word и выбрать нужный файл с М-книгой из перечня последних документов в нижней части нисходящего меню **File**;
- дважды "щелкнуть" мышью на документ М-книги.

Редактор Word откроет документ, используя шаблон **M-book**, запустит систему MatLAB, если она не была до этого активной, и добавит меню **Notebook** в окно Word.

3.3.4. Преобразование документа Word в М-книгу

Чтобы превратить ранее созданный документ Word в М-книгу, необходимо сделать следующее:

1) в редакторе Word создать новую (пока пустую) М-книгу;
2) в меню **Insert** ("Вставка") редактора Word выбрать команду **File**;
3) выбрать в появившемся окне **Вставка файла**, файл, который нужно превратить 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 ей в режиме калькулятора (непосредственных вычислений);*
- *нельзя пользоваться Script-файлами, то есть готовыми М-программами, а также процедурами и функциями, доступными лишь при работе с Script-файлами (например, процедурами создания меню и т.п.).*

Последнее ограничение не является важным. Его можно обойти, если воспользоваться командой **Bring MATLAB to Front** (Вывести MatLAB на передний план) из меню **Notebook**.

В этом случае командное окно MatLAB выйдет на экране на первый 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 возникать как обычно в соответствующих окнах MatLAB. Используя обычные операции перенесения текста и графических изображений из одного окна Windows 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 усмотре-

нию пользователя. Если эту команду активизировать с помощью мыши, на экране возникнет окно, представленное на рис. 3.22.

Как видно, это окно позволяет устанавливать в интерактивном режиме:

- формат вывода чисел в Word (область **Numeric Format**);
- более или менее плотный вывод строк (та же область, переключатели **Loose** и **Compact**);
- размеры выведенных в окно Word графических изображений (область **Figure Options**);
- выводить или нет графические изображения, получаемые при работе MatLAB, в текст М-книги (опция **Embed Figure in M-book**);
- использовать при выводе графических изображений в М-книгу 16 цветов (опция **Use 16-Color Figures**), или 256 цветов.

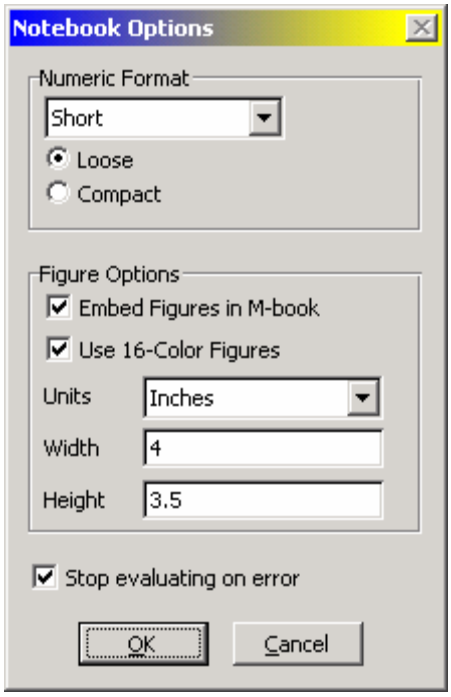

Рис. 3.22

Когда все установки сделаны, надо "нажать" кнопку "Ok" и в дальнейшем эти установки заработают.

В заключение заметим, что *это учебное пособие написано именно как М-книга.*

4. Классы вычислительных объектов

Классом в MatLAB принято называть определенную форму представления вычислительных объектов в памяти ЭВМ 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 записи переменной в память ЭВМ и структуру размещения данных в этом объеме. Операции и функции, которые могут быть применены к определенному типу переменных, образовывают *методы* этого класса.

4.1. Основные классы объектов

В системе MatLAB определены шесть встроенных классов вычислительных объектов:

- ***double*** - числовые массивы и матрицы действительных или комплексных чисел с плавающей запятой в формате двойной точности;
- ***sparse*** - двумерные действительные или комплексные разреженные матрицы;
- ***char*** - массивы символов;
- ***struct*** - массивы записей (структуры);
- ***cell*** - массивы ячеек;
- ***uint8*** - массивы 8-битовых целых чисел без знаков.

Класс ***double*** определяет наиболее распространенный тип переменных в системе MatLAB, с которыми оперирует большинство функций и процедур. Класс ***char*** определяет переменные, которые являются совокупностью символов (каждый символ занимает в памяти 16 битов). Эту совокупность часто называют *строкой*. Класс ***sparse*** определяет тип переменных, которые являются 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 носит 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 имени поля через точку после имени структуры. Наконец, класс ***uint8*** позволяет сохранять 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: определение 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*** - единый метод - операция сохранения (чаще всего применяется с пакетом Image Processing Toolbox).

4.1.1. Класс символьных строк (char)

Введение строк символов из 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 с помощью 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 количество элементов (символов) сцепляемых символьных строк может быть различным). Например

```
>> [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). Например:

```
>> 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, которые совпадают с 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. Например:

```
>> 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 пара-

метров, что связано с переводом числовой переменной в строку символов определенного вида. Это можно сделать с помощью 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 форматом **Numeric Format.** В качестве примера рассмотрим формирования текстовой строки с включением значения переменной:

```
>> x = pi;
>> disp(['Значение переменной "x" равно ', num2str(x)])
Значение переменной 'x' равно 3.1416
```

Аналогичним 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 составных 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. образует 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 разрешается 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 за именем ***field*** осуществляется так:

<имя переменной-записи> ***. field***

Например, команда

```
>> PG81. fam = 'Аврутова'
PG81 =
   fam: 'Аврутова'
```

приводит к автоматическому формированию переменной ***PG81*** класса ***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, но уже с тремя полями. Чтобы создать массив аналогичных переменных с теми же полями и с тем же именем ***PG81*** достаточно доибавлять при обращении к этому имени номер записи (в скобках, как к элементу массива):

```
>>  PG81(2). fam = 'Березнюк' ;
>>  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 о именах полей записи можно использовать функцию ***fieldnames***:

```
>> fieldnames(PG81)
ans =
   'fam'
   'imya'
   'bat'
```

Другой способ задания переменной-записи - применение функции ***struct*** по схеме

<имя 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, можно 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 (структуры) добавляется значение нового поля, то же поле автоматически появляется во всех остальных элементах, хотя значение этого поля у других элементов при этом остается пустым. Например:

```
>> PG81. fam = 'Аврутова' ;
>> PG81. imya = 'Марина'; PG81. bat = 'Степановна';
>> PG81(2). fam = 'Березнюк' ;
>> PG81(2). imya = 'Алексий'; PG81(2). bat = 'Иванович';
>> PG81(3). fam = 'Попіль' ;
>> 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 использовании 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 есть 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:

- использованием 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 каморкам - *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 C ячеек 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. 0000и]   [1x11 double]
```

Как видно из примеров, система MatLAB отображает массив ячеек в сокращенной форме.

Чтобы отобразить содержимое ячеек, нужно 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)
```

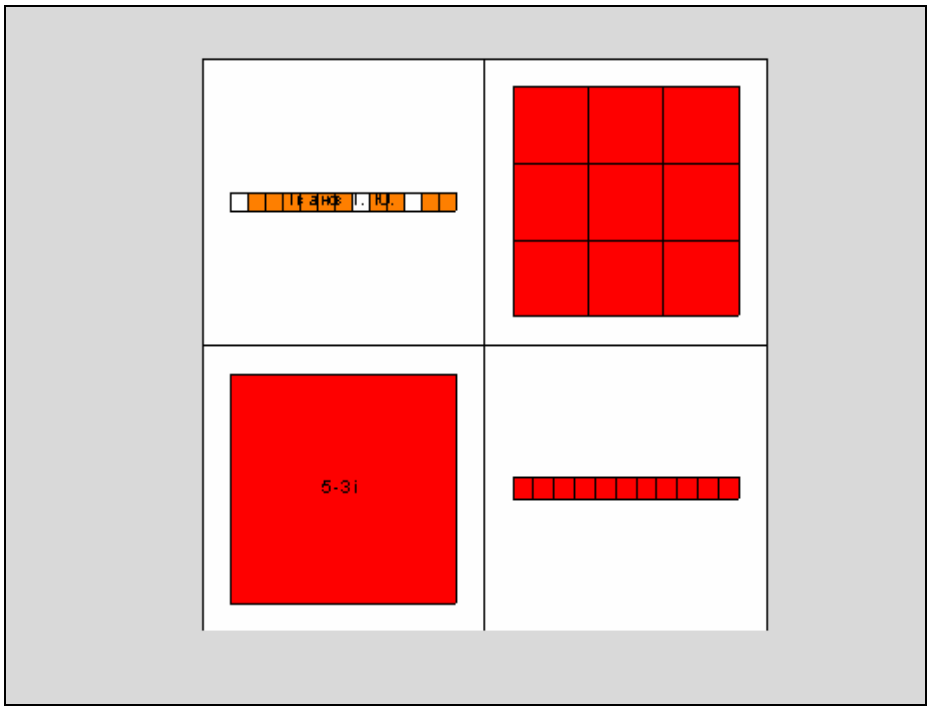

Рис. 3.1

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, предшествующий массив С ячеек может быть построен так:

```
>> C = { ' Иванов И. Ю.', [1 2 3; 4 5 6; 7 8 9]; 5-3и,-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 путями.

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 каморки производится дополнительным указанием в дужках индексов элемента массива, находящегося в нужной каморке:

```
>> 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. Массивы ячеек удобнее массивов записей (структур) в 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 другой вычислительный *объект*, к имени этого объекта следует *применить процедуру* ***class***:

```
>> x=pi;
>> class(x)
ans =double
>> st='Письмо';
>> class(st)
ans =char
>> s=class(num2str(x))
s =char
```

4.2. Производные классы MatLAB

Рассмотренные ранее классы вычислительных объектов построены таким образом, что на их основе пользователь имеет возможность 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 и синтеза 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. к потомкам (наследникам) класса ***struct***. *Поля структуры и операции с полями являются доступными только внутри методов данного класса.*

Все М-файлы, определяющие методы объектов данного класса, должны размещаться в специальном каталоге, который называется *каталогом класса* и обязательно имть имя, состоящее из знака @ (коммерческое 'эт') и имени класса, т. е. **@**<имя класса>. Каталог класса должен быть подкаталогом одного из каталогов, описанных в путях доступа системы MatLAB, но не самим таким каталогом. Каталог класса обязательно должен содержать М-файл с именем, совпадающим с именем класса. Этот файл называют *конструктором класса*. Назначение такого М-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 вычисления 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 такие М-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 - один из следующих за ним по алфавиту:

```
>> FUN1 = inline('am*sin(om*t+eps)')
FUN1 =
    Inline function:
    FUN1(t) = am*sin(om*t+eps)
```

Если обратиться в конструктора 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, должны содержаться только аргумент ***x*** и параметры P1, P2, ... , Pn**.** Например:

```
>> Fun3= inline('P1+P2*x+P3*x^2',3)
Fun3 =
    Inline function:
    Fun3(x,P1,P2,P3) = P1+P2*x+P3*x2
```

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 дисплея 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*x2
```

Процедуры незначительно различаются формой вывода. Основное их отличие в том, что *процедура* ***display*** *работает и* при неявном обращении - *если* 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*x2
```

Получение имен аргументов *inline*-объекта. Это 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, которая записана 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 '.' (точки) перед каждым знаком арифметической операции. Это делает процедура ***vectorize***. Если аргументом этой процедуры есть 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 значения 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 этого 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 алгоритма.

Таким образом, *процедура **feval** позволяет использовать внешние функции при программировании* в среде MatLAB. Общий вид обращения и примеры применения процедуры ***feval*** приведены в разд. 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 *линейных стационарных систем* (ЛСС);

- дочерние объекты (классы), т. е. подклассы класса ***LTI***, которые отвечают 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 и типа ЛСС (непрерывного или дискретного). Дочерние объекты определяются конкретной формой представления ЛСС, т. е. зависят от модели представления. Объект класса ***TF*** характеризуется векторами коэффициентов полиномов числителя и знаменателя рациональной передаточной функции. Объект класса ***ZPK*** характеризуется векторами, которые содержат значения нулей, полюсов передаточной функции системы и коэффициента передачи системы. Наконец, объект класса ***SS*** определяется четверкой матриц, описывающих динамическую систему в пространстве состояний. Ниже приведены основные атрибуты этих классов, их обозначения и содержание.

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)]
[(s2-5s+6)/(s2+s)]

Variable - Имя (тип) переменной (из перечня).
Возможные 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, NY-на-NU матрица для MM-систем.

Variable - Имя (тип) переменной (из перечня)
То же, что и для TF (см. выше)

Специфические атрибуты SS-объектов (моделей пространства состояния)

a,b,c,d - A,B,C,D матрицы, соответствующие уравнениям в пространстве состояния

$$\dot{E}x = Ax + Bu\ ,\quad y = Cx + Du\ ,$$

e - E матрица для систем пространства состояния.
По умолчанию E = eye(size(A)).

StateName - имена переменных состояния (не обязательное).
NX-на-1 массив ячеек из строк
(используйте '' для состояний без имени)
Пример: {'position' ; 'velocity'}.

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).
1-на-NU вектор промежутков времени задержек входов.
Установление 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. NU-на-1 массив ячеек из строк для систем с несколькими входами (используйте '' для переменных без имени). Примеры: 'torque' или {'thrust' ; 'aileron deflection'}.

OutputName - Имена выходов.
Строка для систем с одним выходом. NY-на-1 массив ячеек из строк для систем с несколькими выходами (используйте '' для переменных без имени). Пример: 'power' или {'speed' ; 'angle of attack'}.

Notes - Заметки.
Любая строка или массив ячеек из строк символов. Пример: 'Эта модель создана в течение 2000 года'.

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***	***lti***	***obsv***	***rlocus***	***tfdata***		

Конструктором ***LTI***-объектов является файл ***lti. m*** в поддиректории @LTI. Он создает только шаблон ***LTI-объекта*** по некоторым его параметрам. Ниже приведен его текст:

```
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 непрерывна, а имена входа/выхода являются
 % 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 количество входов и выходов ЛСС, а также величину дискрета времени. Другие атрибуты ***LTI***-объекта могут быть определены только употреблением других процедур. 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 созданный ***LTI***-объект имеет именно указанные параметры, воспользуемся процедурой ***get(sys)*** для получения значений его атрибутов:

```
>> get(sys)
    Ts = 0
    Td = 0
    InputName = {''}
    OutputName = {''}
    Notes = {}
    UserData = []
```

Как видно, большинство полей созданного ***LTI***-объекта пусты, только два из них равны нулю. Кроме того, из описания конструктора вытекает, что обращение к нему не предусматривает возможности установления значений таких полей ***LTI-***объекта, как ***InputName***, ***OutputName***, ***Notes*** и ***UserData*** . Последнее можно сделать, лишь используя специальную функцию ***set*** по схеме

set (<имя LTI-объекта>,'<имя поля>', <Значение>).

Рассмотрим это на примере установления значений некоторых из указанных полей в уже сформированном ***LTI-***объекте ***sys***:

```
>>set(sys,'InputName','Угол','OutputName','Напряжение','Notes','Гиротахометр').
```

Проконтролируем результат:

```
>> get(sys)
     Ts = 0
```

```
Td = 0
InputName = {'Угол'}
OutputName = {'Напряжение'}
Notes = {'Гиротахометр'}
UserData = []
```

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

$$p(x) = an*xn + ... + a2*x2 + a1*x + a0.$$

Очевидно, полином как функция целиком 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*** меню ***File*** командного окна, а затем в появившемся окне перейдите к папке Toolbox\Matlab**Polyfun.** Воспользуйтесь пиктограммой создания новой папки в этом окне, чтобы открыть новую папку с именем @POLYNOM**.**

Перейдите во вновь созданную папку. Теперь вы готовы к созданию М-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, которая состоит из двух полей

- ***n*** - целого числа**,** которое задает порядок полинома;
- ***c*** **-** вектора длиной ***n+1*** коэффициентов полинома.

Входным аргументом для образования ***polynom***-объекта должен быть, очевидно, заданный вектор его коэффициентов**.**

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*** по схеме:

$$p = class(p, '<имя\ класа>').$$

Ниже приведен возможный текст М-файла **polynom**. **m**.

```
function p=polynom(v,cs);
 % POLYNOM - конструктор полином-объектов
 % Под полином-объектом понимается объект языка MatLab,
 % который является записью с двумя полями:
 % .c - вектор-строка, содержащая коэффициенты
 %      полинома в порядке уменьшения степени аргумента;
 % . n - число, равное порядку полинома.
 %          p=POLYNOM(v)  формирует полином-объект "p" по заданному
 %      вектору "v", который состоит из значений коэффициентов
 %      будущего полинома в порядке уменьшения степени аргумента.
 %          p=POLYNOM(v,cs)  формирует полином-объект "p" по заданному
 %       вектору "v" корней полинома и значению "cs" его
 %       старшего коэффициента.
if nargin==0                        % Эта часть
    p.c=[];                         % создает пустой полином-объект,
    p.n=0;                          % если отсутствуют аргументы
    p=class(p,'polynom');
elseif isa(v,'polynom')             % Эта часть создает дубликат,
    p=v                             % если аргумент является полином-объектом
elseif nargin==2                    % Эта часть работает, если в обращении есть 2 аргумента,
                                    % то есть задан вектор корней полинома
    if cs==0                        % Если старший коэффициент равен нулю,
        cs=1;                       % его следует заменить на 1;
        end
        k=length(v);                % Определение длины заданного вектора
                                    % коэффициентов
      for i=1:k
          vs(i,:)=[1 -v(i)];
      end
      p.n=k;                        % Определение порядка полинома
      p.c=cs*vs(1,:);               % Формирование
      for и=2:k                     % вектора
        p.c=conv(p.c,vs(и,:));      % коэффициентов
      end                           % полинома
    p=class(p,'polynom');           % Присвоение метки полином-объекта
else                                % Эта часть работает, если аргумент один,
                                    % то есть задан вектор коэффициентов
    k=length(v);
    n=k; m=1;
   while v(m)==0                    % Этот цикл сокращает длину входного вектора
      n=n-1;                        % (уменьшает порядок полином) в случае,
      m=m+1;                        % если первые элементы вектора
    end                             % равны нулю
    p.n=n-1;                        % Тут присваиваются значения полям
    p.c=v(k-n+1:end);               % записи будущего полином-объекта
    p=class(p,'polynom');           % Присвоение метки полином-объекту
end                                 % Завершение конструктора POLYNOM
```

Система MatLAB позволяет вызывать конструктор без аргументов. В этом случае конструктор может образовать шаблон объекта с пустыми полями. Возможно также, что конструктор будет вызываться с входным аргументом, который уже является полином-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 полинома, а второй - значением старшего коэффициента полинома. Так как в этом случае порядок полинома обязательно может должен быть равен числу корней, старший коэффициент не может быть равным 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.

4.3.3. 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 М-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 М-файла. Представленный вариант формирует символьную строку вида

*<значение **an**>**x^ n + ... +<значение **a2**>**x^2+<значение **a1**>**x+...*
*+<значение **a0**>*

с изъятием 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 М-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 полином-объекта
disp('');
disp([' ',inputname(1),' = ',char(p),';']);
disp('');  % Завершение POLYNOM/DISPLAY
% Завершение POLYNOM/DISPLAY
```

Проверим эффективность работы созданных трех М-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*x8 + 2*x7 + 3*x6 + 4*x5-6*x2-5*x - 7;
```

Создадим теперь полином-объект по заданным его корням и значению старшего коэффициента:

```
>> Pol2=polynom([1 2 3 4 5],-5)
  Pol2 = -5*x5 + 75*x4 - 425*x3 + 1125*x2 - 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 свидетельствует результат, все созданные М-файлы работают нормально.

4.4. Создание методов нового класса

Весьма удобной в системе MatLAB является предоставляемая ею возможность создания процедур, которые могут быть выполнены не только стандартным путем обращения к ее имени, но и более простым путем использования знаков арифметических действий, операций сравнения, скобок и т.п. С этим мы уже столкнулись в предыдущем разделе, убедившись, что процедура ***display*** выполняется не только при явном обращении вида ***display(x)***, но и неявно, если некоторый оператор формирует величину ***x***, а после этого оператора отсутствует символ ' ; '. Поэтому, *если в любом новом классе объектов присутствует М-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 М-файлов, предусмотренных системой MatLAB, с указанием вида оператора их неявного вызова и краткого описания особенностей их использования.

Таблица 3.1. Операторные функции

Оператор вызова	Имя М-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 поелементное	Два аргумента. Результат - того же класса, что и аргументы

a . / b	rdivide(a,b)	Правое делени поелементное	Два аргумента. Результат - того же класса, что и аргументы
a . \ b	ldivide(a,b)	Левое деление поелементное	Два аргумента. Результат - того же класса, что и аргументы
a . ^ b	power(a,b)	Степень поелементная	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)	Транспониро-вание	Аргумент один. Результат - того же класса.
a . '	transpose(a)	Транспониро-вание	Аргумент один. Результат - того же класса.
a : d : b a : b	colon(a,d,b) colon(a,b)	Формирова-ние вектора	Два или три аргумента. Результат - вектор того же класса, что и аргументы
вывести в командное окно	display(a)	Вывод на терминал	Аргумент один. Результат - изобра-жение на терминале символьного представления аргумента .
[a b]	horzcat(a,b,...)	Объединение в строку	Два или больше аргумента. Резуль-тат - вектор-строка из аргументов
[a; b]	vertcat(a,b,...)	Объединение в столбец	Два или больше аргумента. Резуль-тат - 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 таблице 3.1, 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 превращает 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*** - полином-объект, а ***r*** - число. Затем процедура дополняет нулями векторы коэффициентов полиномов-слагаемых, если в этом возникает необходимость (при неодинаковых порядках полиномов). Фактически сложение сводится к сложению этих исправленных векторов коэффициентов. Заключительная операция - по полученному вектору полинома суммы создается новый полином-объект с помощью конструктора полиномов. Проиллюстрируем работу этого метода на примере. Введем вектор коэффициентов первого полинома:

```
>> V1 = [ 2-3 0 6];
```

и создадим из него полином-объект

```
>> P1 = polynom(V1)
   P1 = 2*x3-3*x2 + 6;
```

Аналогично 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*x7 + 9*x4 + 2*x3-3*x2-3*x + 11;
```

Аналогичные результаты получаются и в случае, если 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*x7 + 9*x4 + 2*x3-3*x2-3*x + 11;
```

Однако если оба аргумента представлены векторами, система сразу же отреагирует на это, обнаружив ошибку:

```
>> V1+V2
??? Error using ➔ +
Matrix dimensions must agree.
```

В этом случае система уже использует не М-файл ***plus*** из подкаталога @POLYNOM, а встроенную аналогичную процедуру для векторов. А эта процедура осуществляет сложение векторов лишь при условии их одинаковой длины. Поэтому и возникает ошибка.

Аналогичной является процедура вычитания полином-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*x7-9*x4 + 2*x3-3*x2 + 3*x + 1;
```

Создадим еще такие методы класса ***polynom***:

- метод ***double***, который осуществляет обратную относительно создания полином-объекта операцию, - по заданному полиному определяет вектор его коэффициентов и порядок:

```
function [v,n]=double(p)
  % POLYNOM/DOUBLE -  преобразование полином-объекта
   %  в вектор его коэффициентов
   % v=DOUBLE(p) превратит полином-объект "p" в вектор "v",
  %  содержащий коэффициенты полинома 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 полином-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 полином-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***; он создает два полином-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, который содержит 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. 8579и
>> plot(Pol)
```

Результат представлен на рис. 3.2.

Чтобы получить перечень всех созданных методов класса, следует воспользоваться 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
```

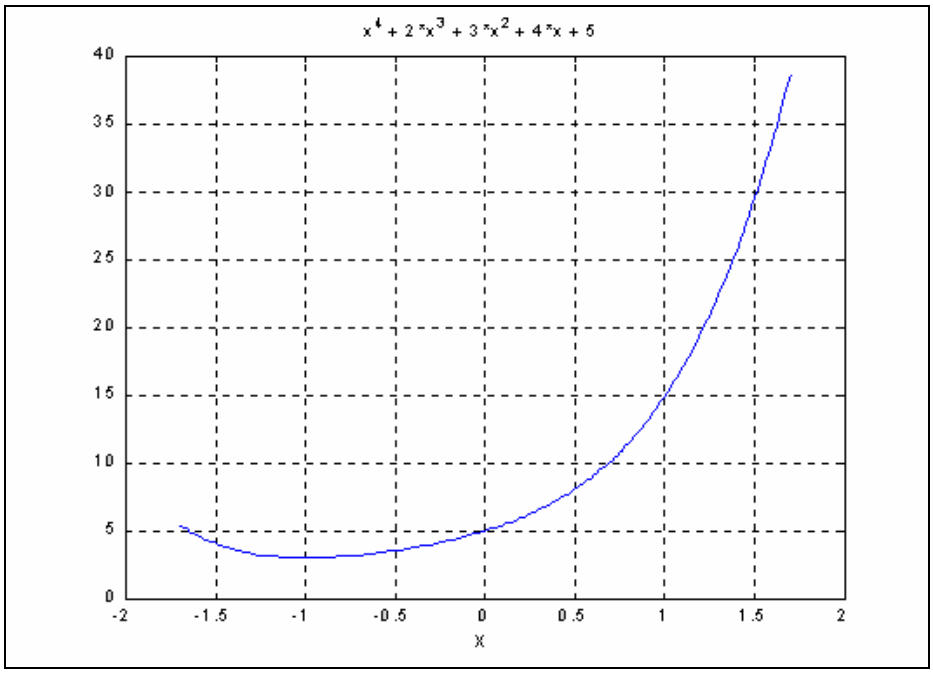

Рис. 3.2

Теперь мы имеем мощный вычислительный аппарат для работы с полиномами. Развивая его дальше, можно создать новый класс более сложных объектов - класс рациональных передаточных функций, которые являются конструкцией 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 звеньев) и многие другие, отвечающие определенным типам соединений звеньев. Примерно на такой основе создан, к примеру, класс ***tf*** (Transfer Function), используемый в пакете CONTROL.

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 преобразования в целях получения информаци о величине, которая должна быть измерена, представлена на рис. 5.1.

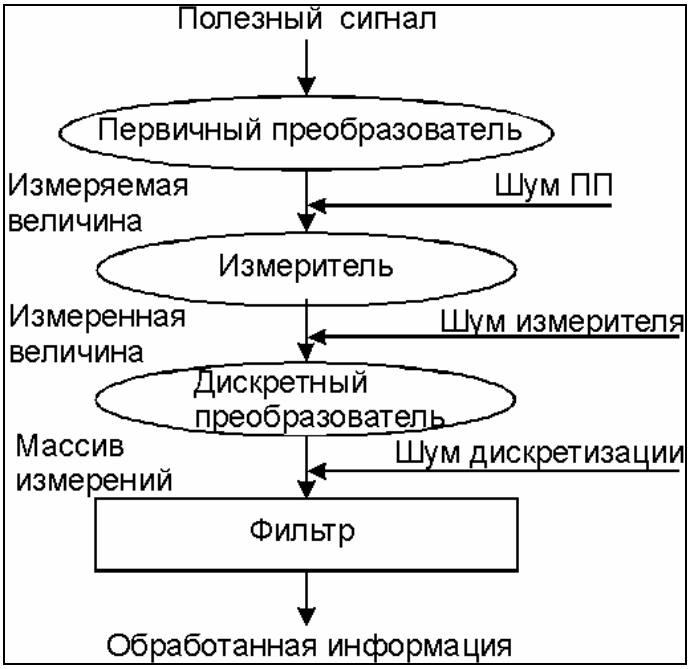

Рис. 5.1.

Физическая величина, являющаяся полезной (несущей в себе необходи-мую 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 вос-становить информационную составляющую в дальнейшем было бы невозможным. Первичный преобразователь вносит зависимость сигнала, который может быть измерен, от некоторых других физических величин. Вследствие этого выходная его величина содержит, кроме полезной информа-ционной составляющей, другие, вредные составляющие или черты, искажа-ющие 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 пребразовывать электрическими устройствами.

Для осуществления цифровой обработки измеренная величина должна быть преобразована в дискретную форму при помощи специального устрой-ства, которое содержит э*кстраполятор* и *аналого-цифровой преобразователь* (АЦП). Первый производит фиксацию отдельного текущего значения измерен-ной 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.

Чтобы на основе полученного 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 видов таких моделей, чаще всего - для линейных моделей.

В общем случае процесс создания фильтра распадается на такие 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, теория проектирования которых разработана;

- 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), не лишними становятся *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 фильтра-ми. В пакет входят процедуры, осуществляющие преобразования Фурье, Гиль-берта, а также статистический анализ. Пакет позволяет рассчитывать корреля-ционные функции, спектральную плотность мощности сигнала, оценивать параметры фильтров по измеренным отсчетам входной и выходной последова-тельностей.

5.1. Формирование типовых процессов

5.1.1. Формирование одиночных импульных процессов

В пакете Signal предусмотрено несколько процедур, образующих последовательности данных, представляющие некоторые одиночные импульс-ные 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 амплитуди, шириною ***w***, центрированного относительно t=0 по заданому 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 висоты 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)')
```

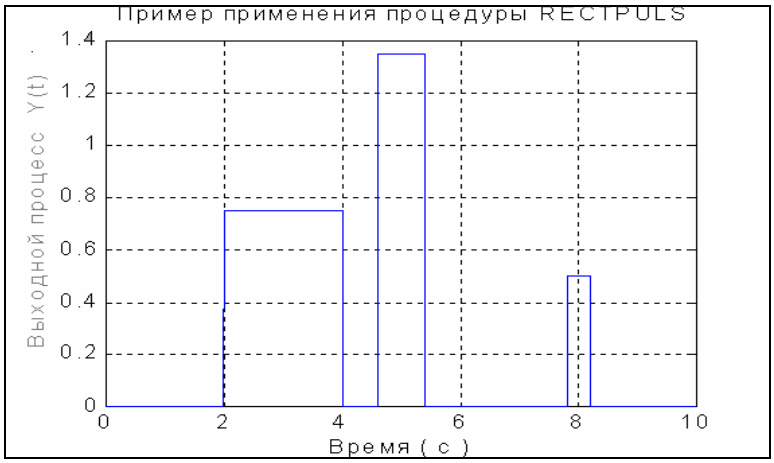

Рис. 5.2.

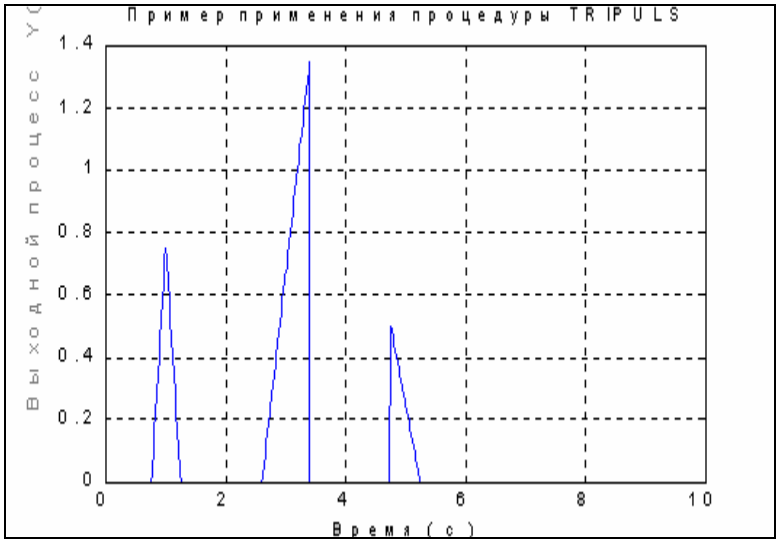

Рис. 5.3.

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.

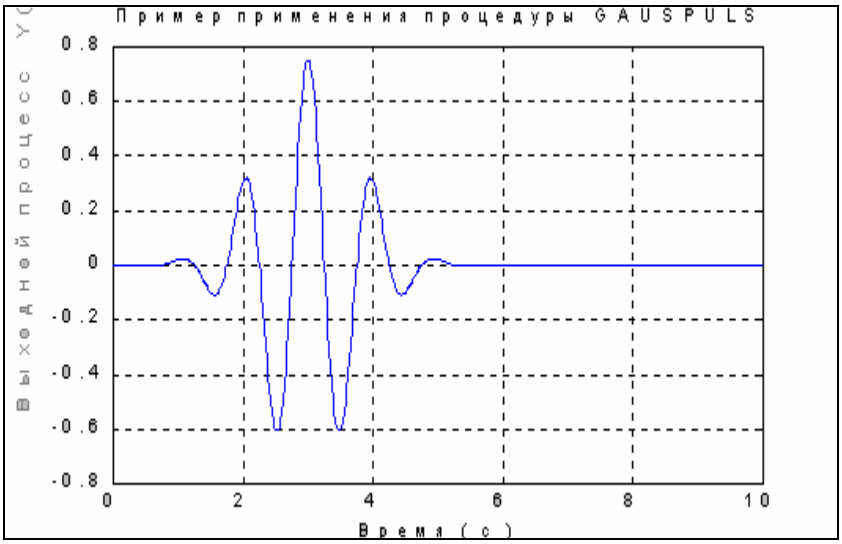

Рис. 5.4.

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)')
```

Наконец, рассмотрим процедуру ***sinc***, формирующую вектор значений функции sinc(t), которая определяется формулами:

$$\sin c(t) = \begin{cases} 1 & t = 0, \\ \dfrac{\sin(\pi t)}{\pi t} & t \neq 0. \end{cases}$$

Эта функция является обратным преобразованием Фурье прямоугольного импульса шириной 2π и высотою 1:

$$\sin c(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega t} d\omega .$$

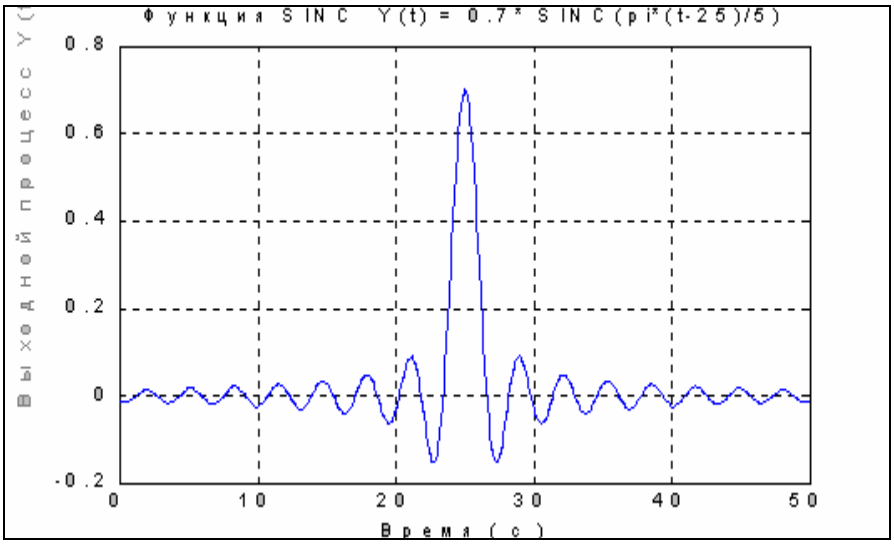

Рис. 5.5.

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.

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 (см. 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 ( с )')
ylabel('Выходной процесс Y(t)')
```

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 ( с )')
ylabel('Выходной процесс  Y(t)')
```

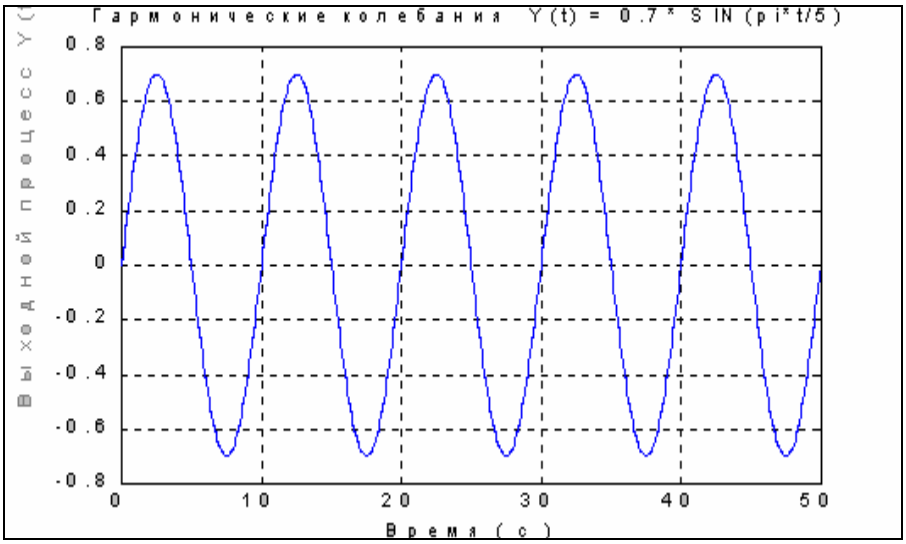

Рис. 5.6.

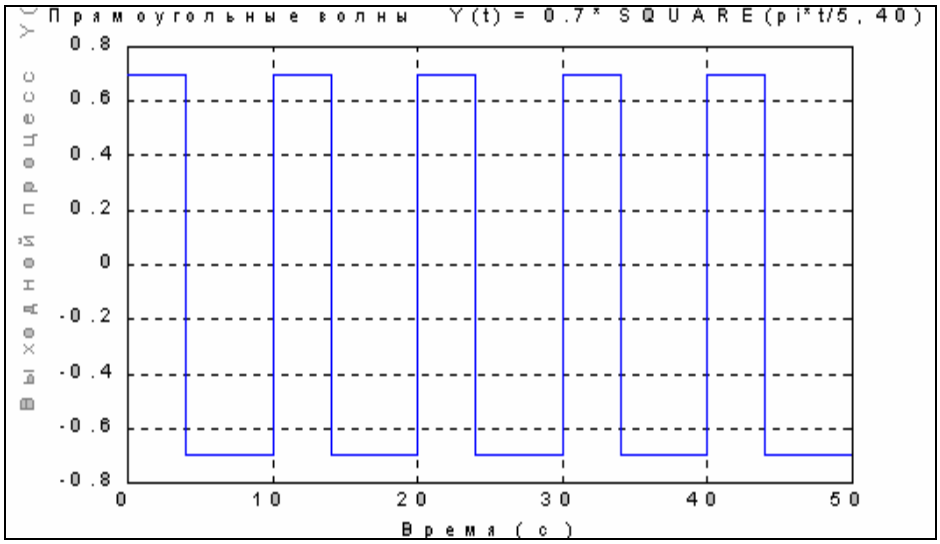

Рис. 5.7.

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 векто-ром ***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.

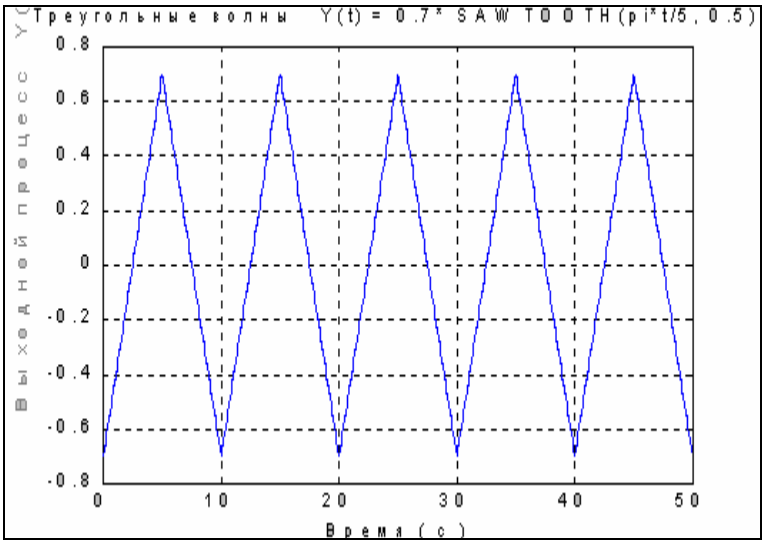

Рис. 5.8.

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,...)***,

где ***d*** определяет вектор значений тих 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 соответствиии 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;

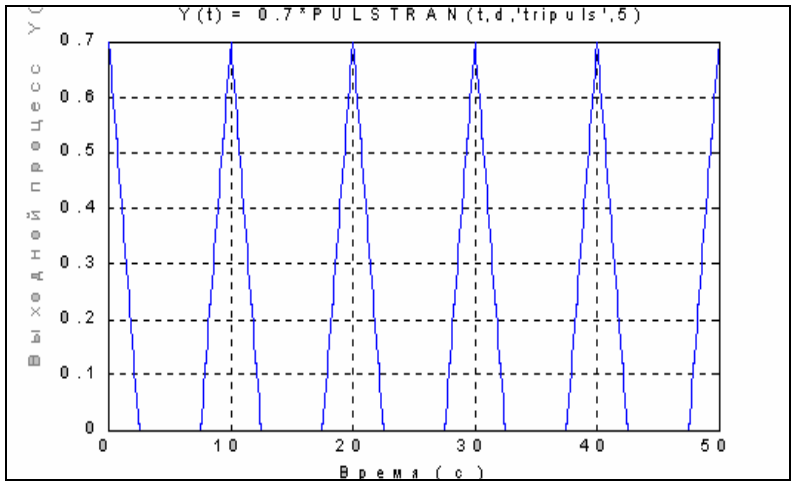

Рис. 5.9.

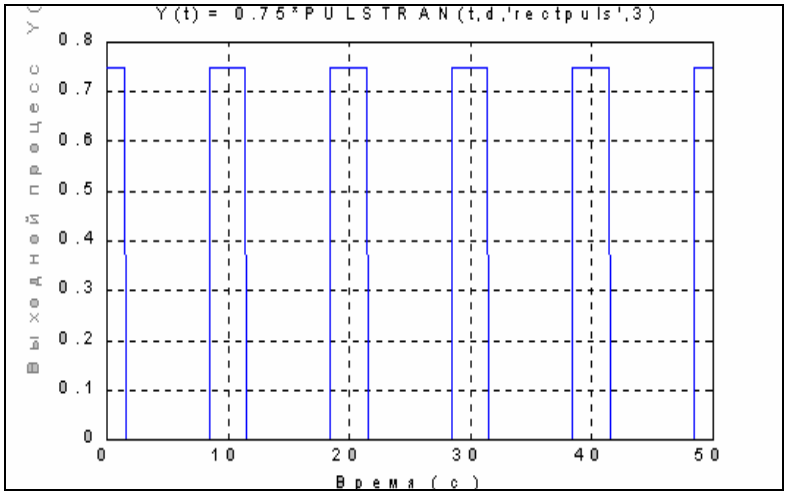

Рис. 5.10.

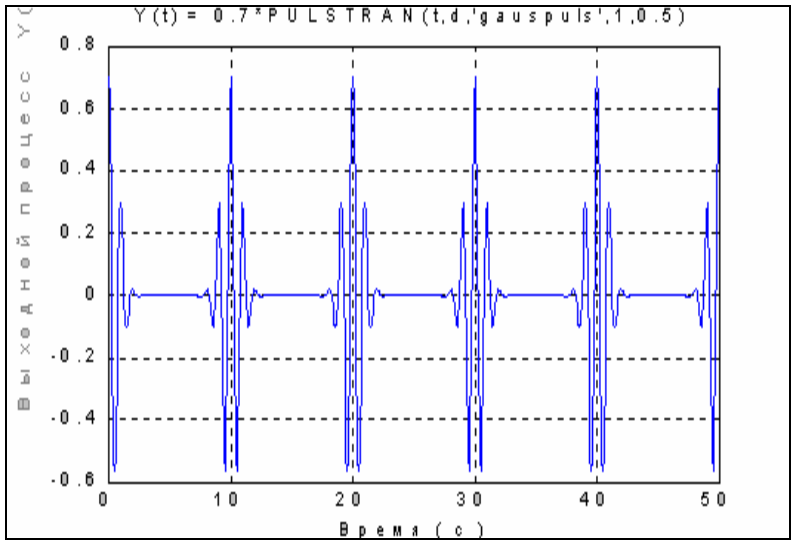

Рис. 5.11.

- 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 изменеия которой линейно изменяется со временем. Общая форма обращения к этой процедуре является такой:

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 процедуры CHIRP')
xlabel('Время ( с )'), ylabel('Выходной процесс  Y(t)')
```

Результат показан на рис. 5.12.

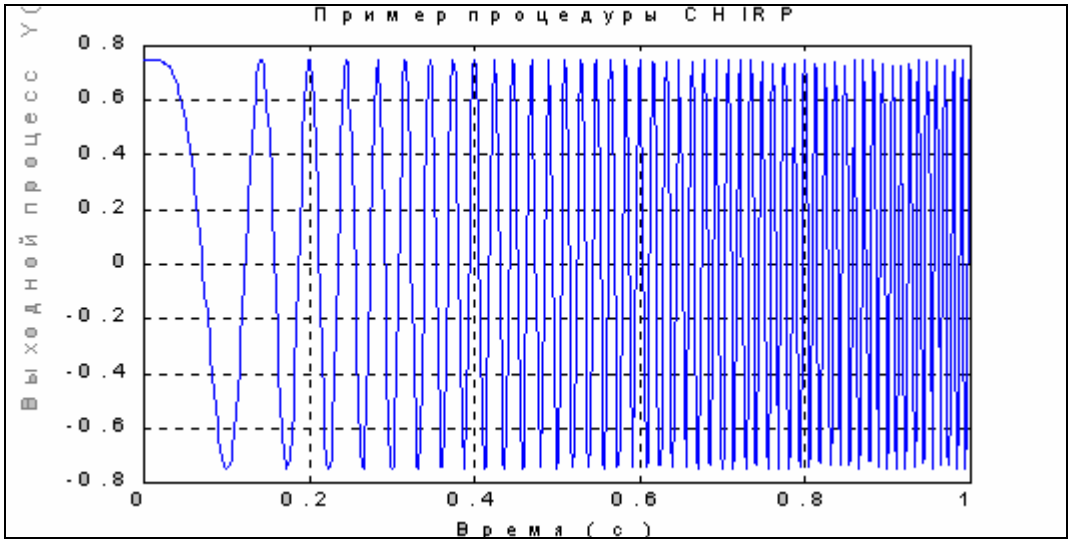

Рис. 5.12.

Еще одна процедура ***diric*** формирует массив значений так называемой *функции Дирихле*, определяемой соотношениями:

$$diric(t) = \begin{cases} -1^{k(n-1)} & t = 2\pi k, \quad k = 0, \pm 1, \pm 2, .. \\ \dfrac{\sin(nt/2)}{n \cdot \sin(t/2)} & при \quad других \quad t \end{cases}$$

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 графики функции Дирихле для ***n***=3, 4 и 5:

```
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 представлены на рис. 5.13...5.15.

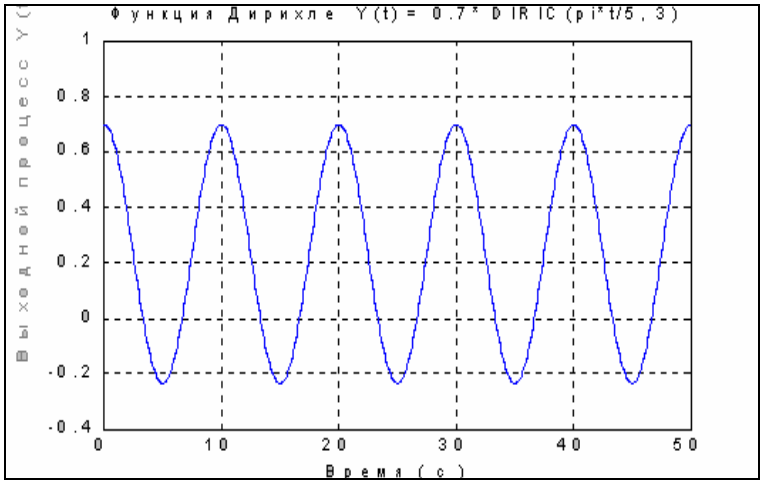

Рис. 5.13.

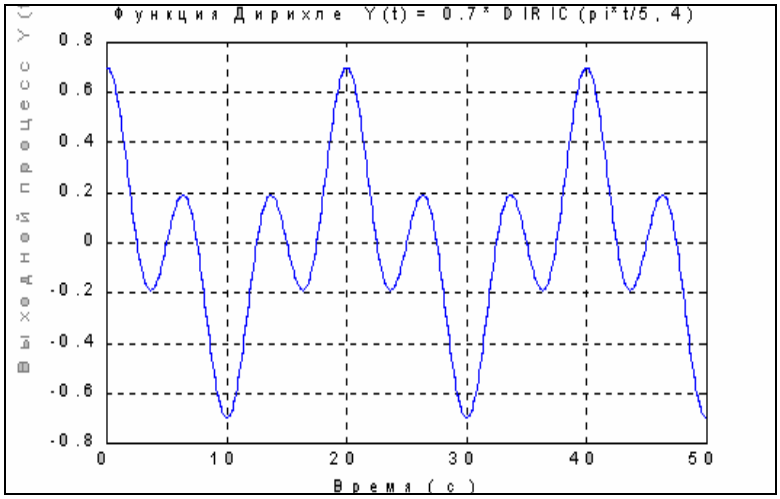

Рис. 5.14.

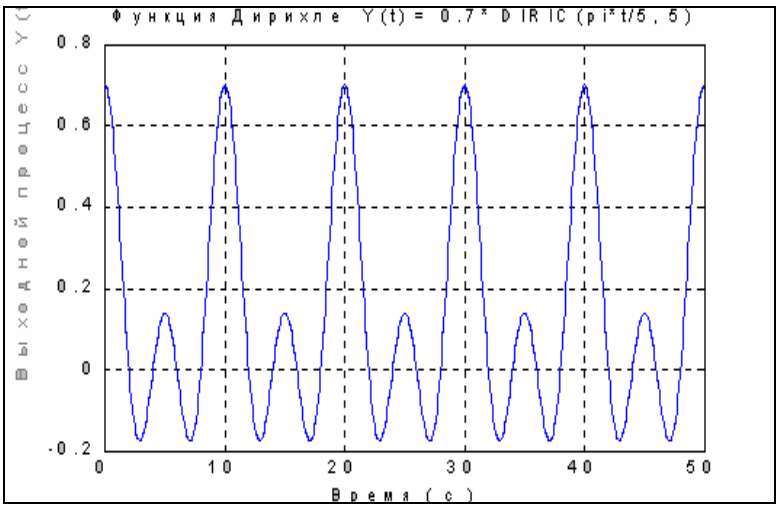

Рис. 5. 15.

5.2. Общие средства фильтрации. Формирование случайных процессов

5.2.1. Основы линейной фильтрации

Рассмотрим основы линейной фильтрации на примере линейного стационарного фильтра, который в непрерывном времени описывается дифференциальным уравнением второго порядка:

$$\ddot{y} + 2\zeta\omega_o \cdot \dot{y} + \omega_o^2 \cdot y = A \cdot x\,, \qquad (5.1)$$

где x - заданный процесс, подаваемый на вход этого фильтра второго порядка; y - процесс, получаемый на выходе фильтра; ω_o - частота собственных колебаний фильтра, а ζ - относительный коэффициент затухания этого фильтра.

Передаточная функция фильтра, очевидно, имеет вид:

$$W(s) = \frac{y(s)}{x(s)} = \frac{A}{s^2 + 2\zeta\omega_o \cdot s + \omega_o^2}. \qquad (5.2)$$

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 $W(j\omega)$ по передаточной функции $W(s)$ звена, заданной векторами коэффициентов ее числителя ***b*** и знаменателя ***a,*** а также по заданному век-тору ***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:

$A = 1; \qquad \zeta = 0.05; \qquad T_o = 2\pi / \omega_o = 1.$

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)
```

В результате получим рис. 5.16.

Допустим, что 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.

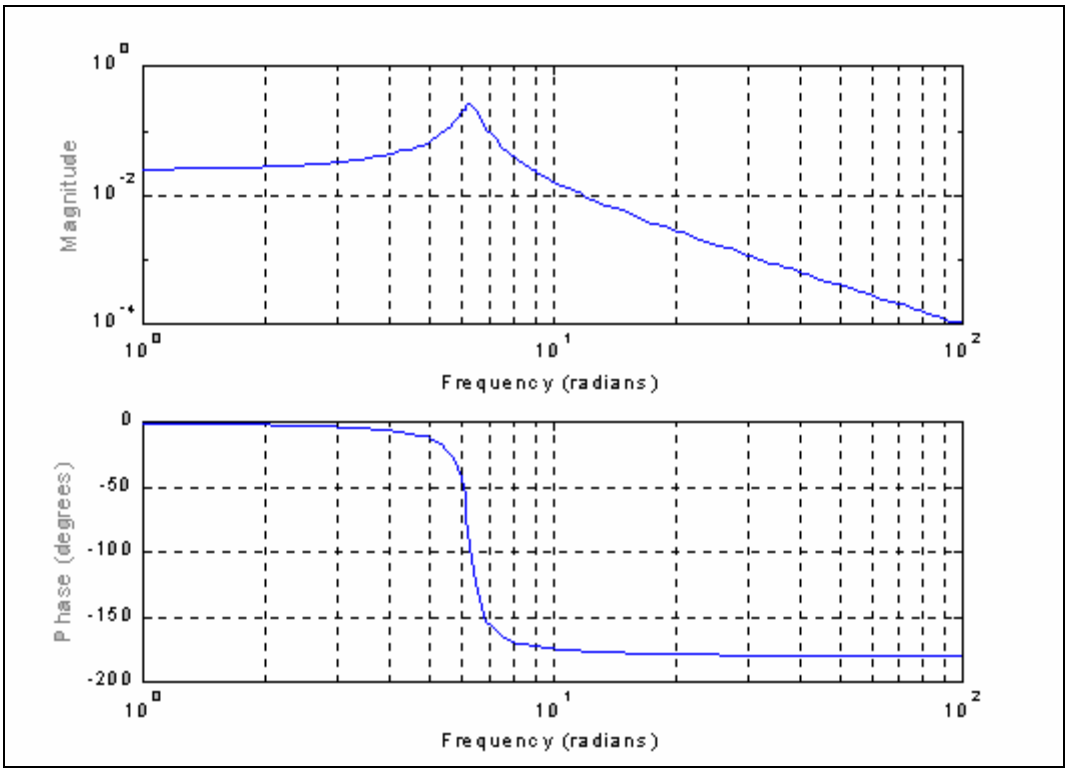

Рис. 5.16

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

$$(1 + 2\zeta\omega_o \cdot Ts + \omega_o^2 \cdot Ts^2) \cdot y(k) - 2(1 + \zeta\omega_o \cdot Ts) \cdot y(k-1) + y(k-2) = A \cdot Ts^2 \cdot x(k) \tag{5.3}$$

является дискретным аналогом дифференциального уравнения (5.1).

Применяя к полученному уравнению Z-преобразование, получим:

$$y(z) \cdot [a_o + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}] = A \cdot Ts^2 \cdot x(z), \tag{5.4}$$

где

$$a_o = 1 + 2\zeta\omega_o \cdot Ts + \omega_o^2 \cdot Ts^2;$$
$$a_1 = -2(1 + \zeta\omega_o \cdot Ts); \tag{5.5}$$
$$a_2 = 1.$$

Дискретная передаточная функция фильтра определяется из уравнения (5.4):

$$G(z)=\frac{y(z)}{x(z)}=\frac{A\cdot Ts^2}{a_o+a_1\cdot z^{-1}+a_2\cdot z^{-2}}, \qquad (5.6)$$

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 $G(z)$, которая задана векторами значений ее числителя ***b*** и знаменателя ***a,*** удобно использовать процедуру ***freqz***, обращение к которой аналогично обращению к процедуре ***freqs*** :

```
freqz( b,a)
```

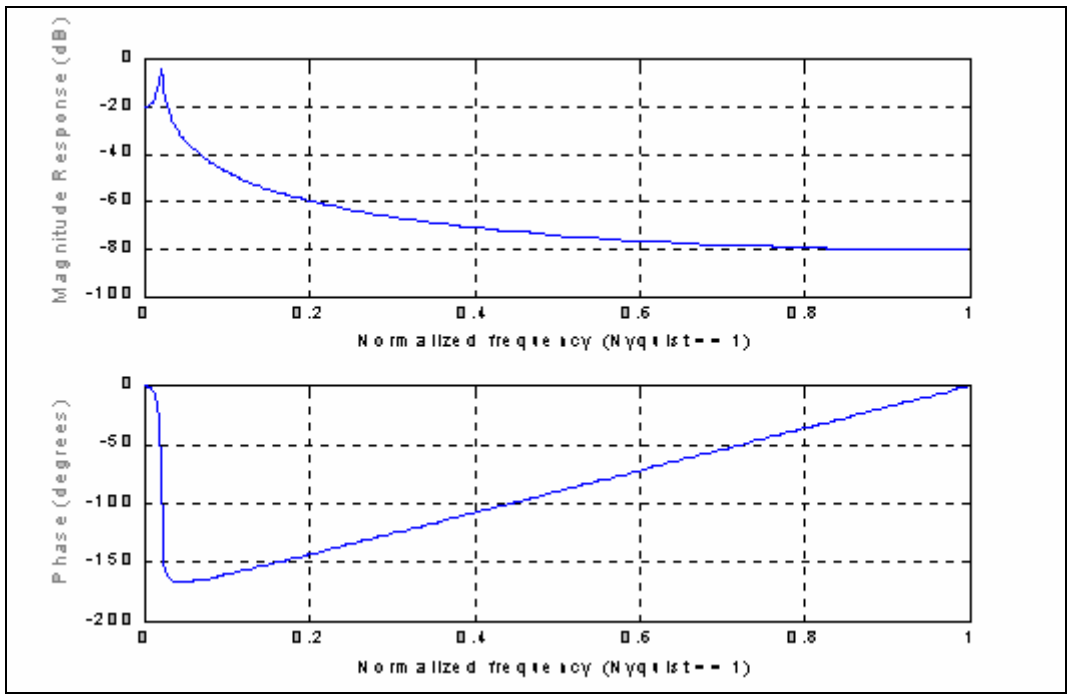

Рис. 5.17

В системе MatLAB *фильтрация*, т.е. преобразование заданного сигнала с помощью линейного фильтра, описываемого дискретной передаточной функцией вида

$$G(z)=\frac{y(z)}{x(z)}=\frac{b_o+b_1\cdot z^{-1}+\ldots+b_m\cdot z^{-m}}{a_o+a_1\cdot z^{-1}+\ldots+a_n\cdot z^{-n}} \qquad (5.7)$$

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 коэффициен-

тов 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 T_1 =1с и амплитудой A_1 =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 (с)');  ylabel('Yp(t)')
```

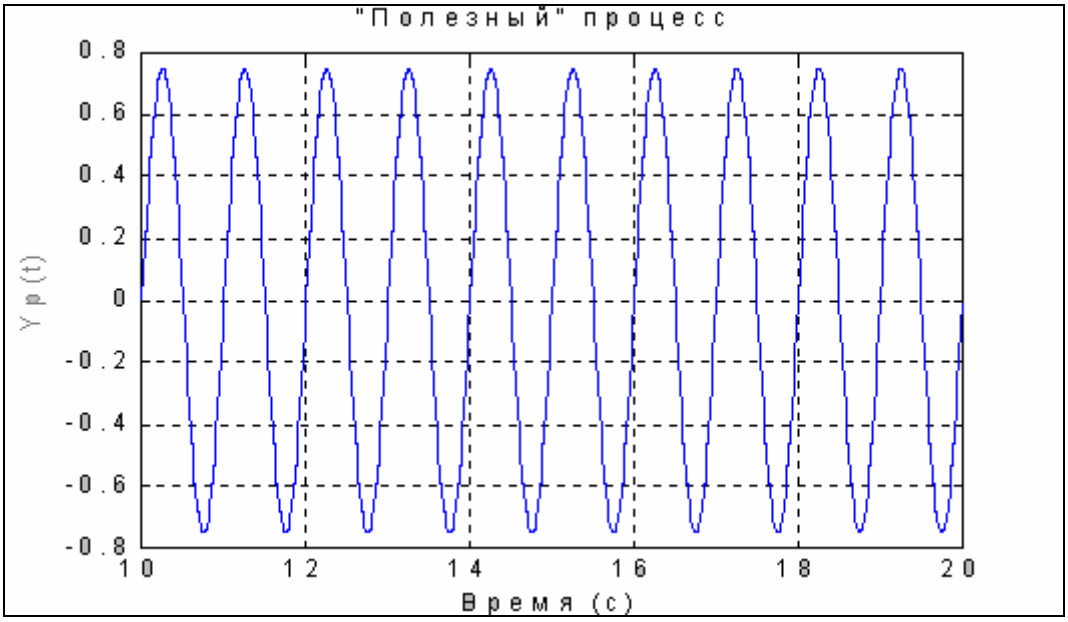

Рис. 5.18

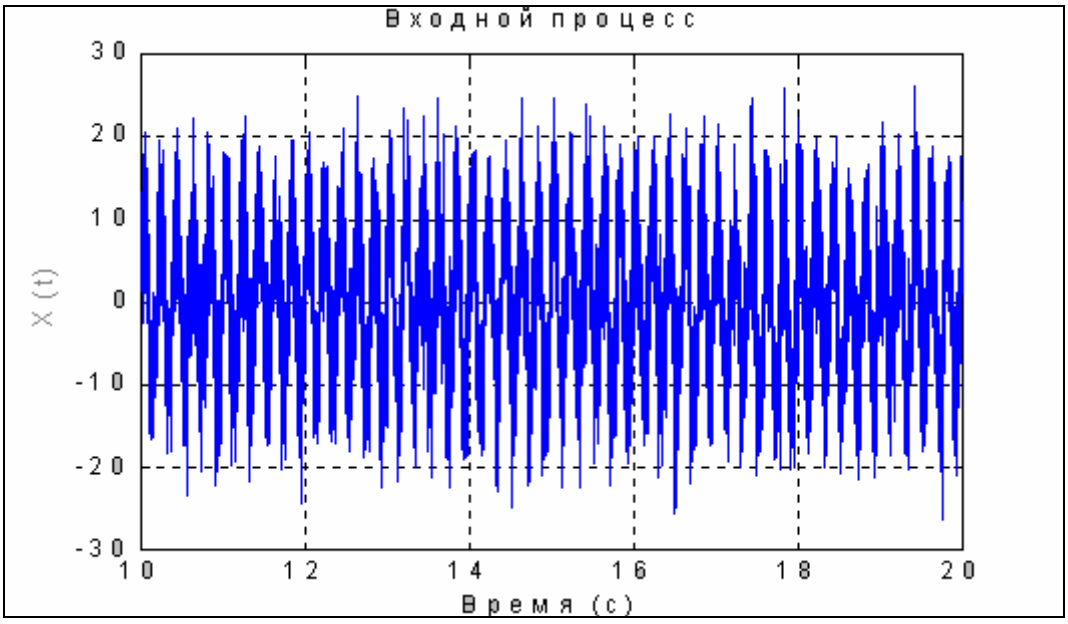

Рис. 5.19

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 $x(t)$(см. 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)')
```

Требуется так обработать измеренные данные $\boldsymbol{x}$, 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 ($T_ф = T_1$). Для того, чтобы после прохождения через такой фильтр амплитуда восстановленного сигнала совпадала с амплитудой полезного сигнала, нужно входной сигнал фильтра умножить на постоянную величину $2\varsigma\omega_o^2$ (ибо 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)')
```

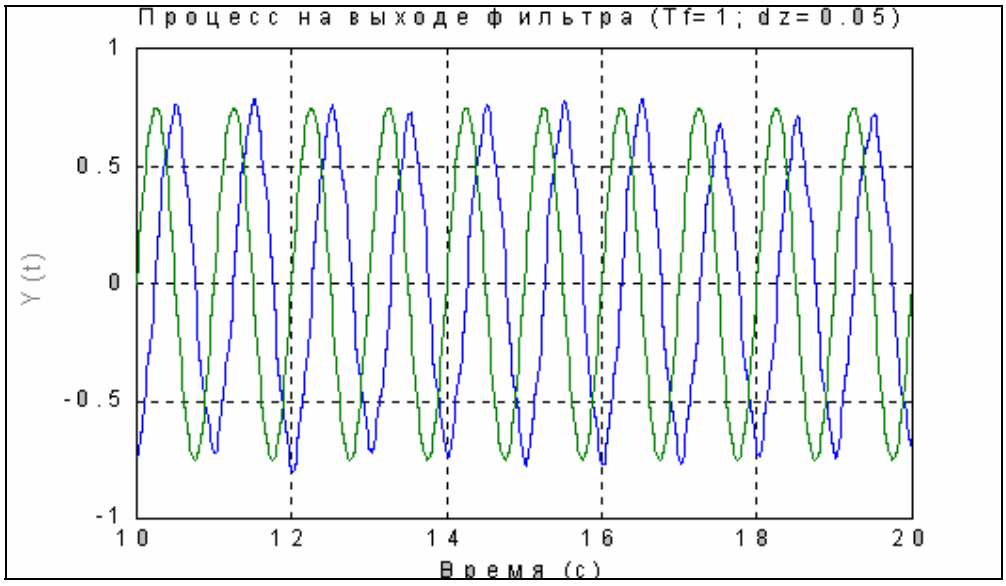

Рис. 5.20

В результате получаем восстановленный процесс (рис. 5.20). Для сравнения на этом же графике изображен восстанавливаемый процесс.

Как видим, созданный фильтр достаточно хорошо восстанавливает полезный сигнал. 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 иллюстрируется 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 (с)');      ylabel('Y(t)')
```

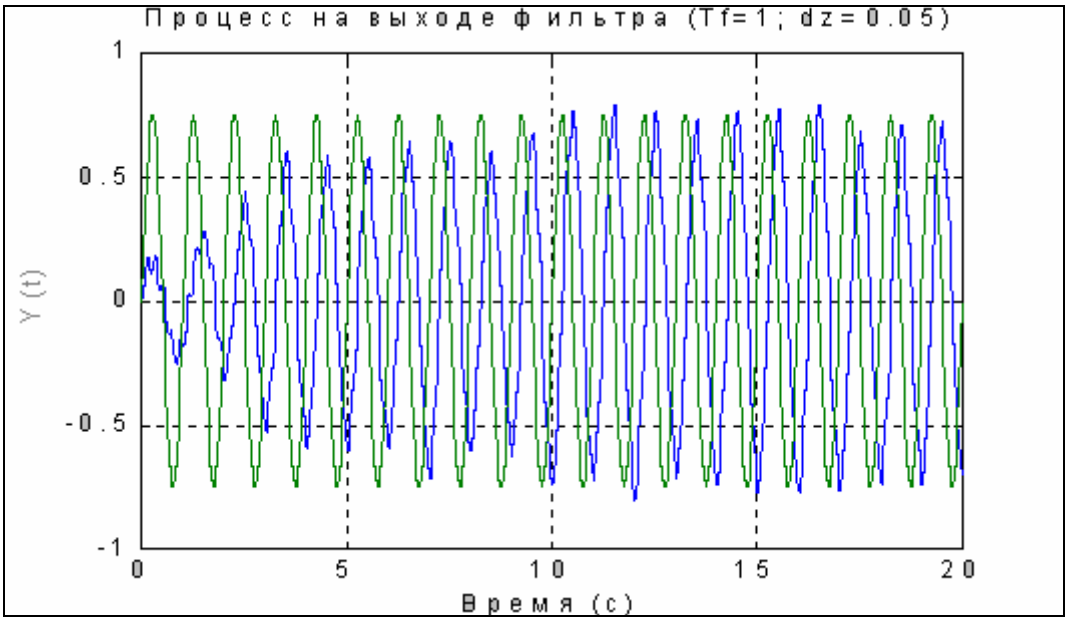

Рис. 5.21

2) в установившемся режиме наблюдается значительный сдвиг ($\pi/2$) фаз между восстанавливаемым и восстановленным процессами; это тоже 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 (с)');      ylabel('Y(t)')
```

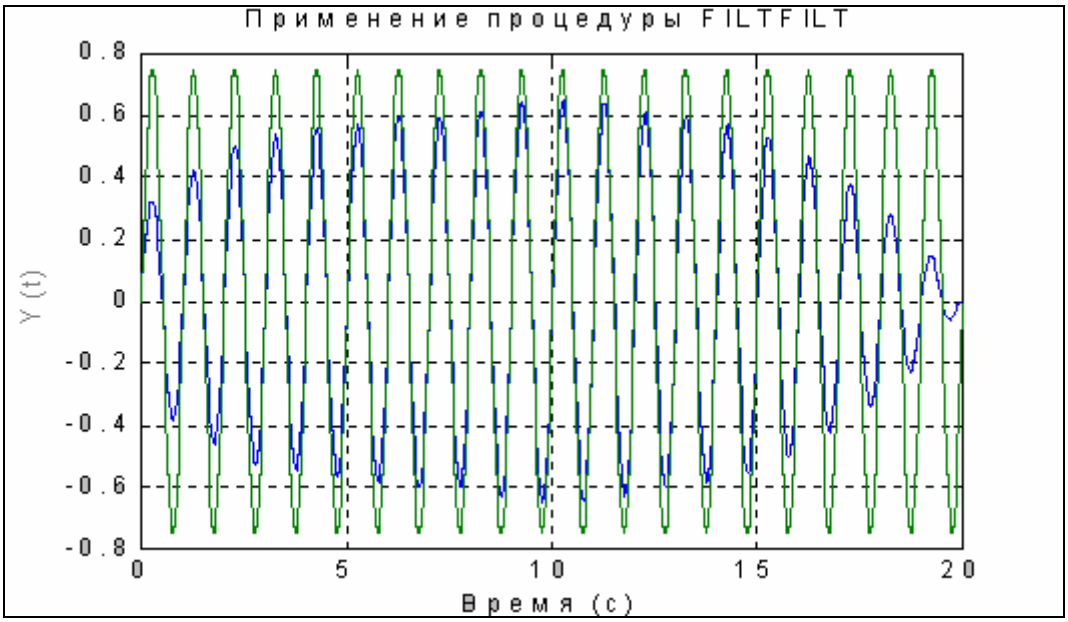

Рис. 5.22

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.

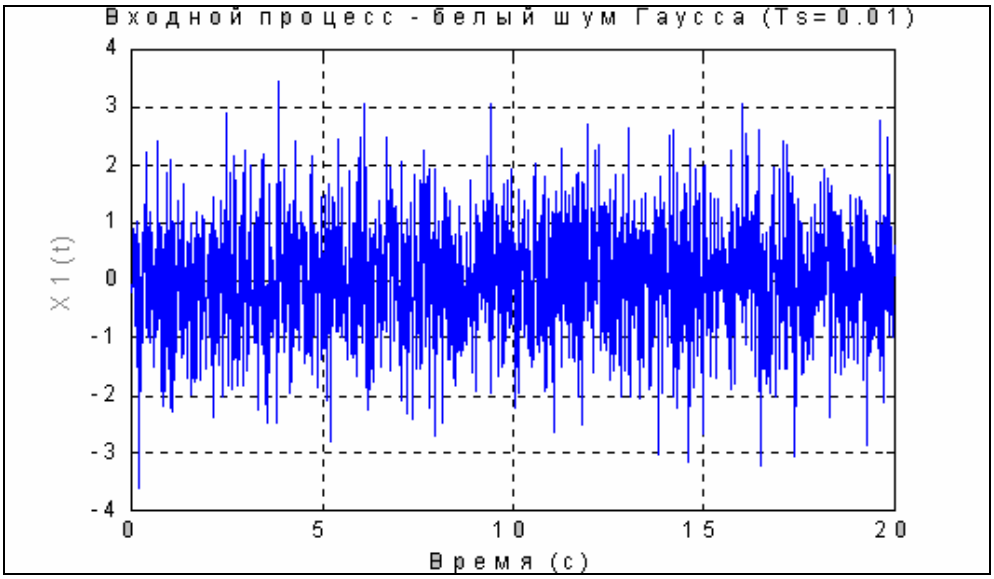

Рис. 5.23

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 длиною, равной длине вектора ***t,*** например

```
Ts=0.01;     t=0 : Ts : 20;     x1=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 $x_1(t)$ с дискретом времени Ts=0.01с представлен на рис. 5.23.

Для другого значения дискрета времени (Ts=0.001с), повторяя аналогичные операции, получим процесс $x_2(t)$, изображенный на рис. 5.24.

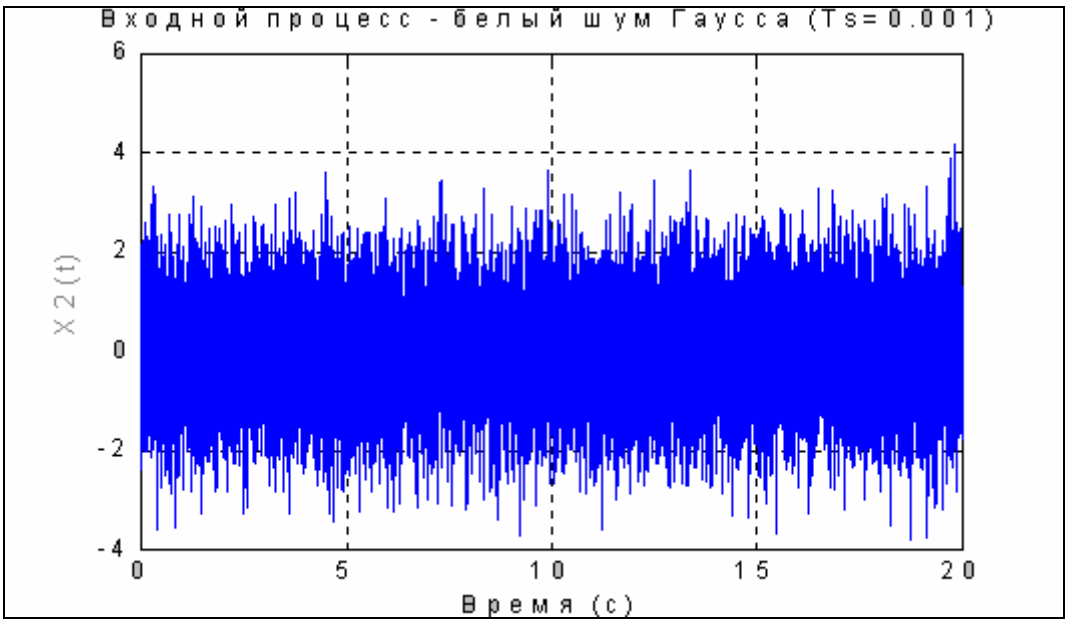

Рис. 5.24

Создадим дискретный фильтр второго порядка с частотой собственных колебаний $\omega_o = 2\pi$ рад./с = 1 Гц и относительным коэффициентом затухания $\zeta = 0.05$ 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);
```

и 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 (с)');     ylabel('Y1(t)')
```

Результат представлен на 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('Процес на виході фільтра (T0=1; dz=0.05, Ts= 0.001)');
xlabel('Час (с)');       ylabel('Y2(t)')
```

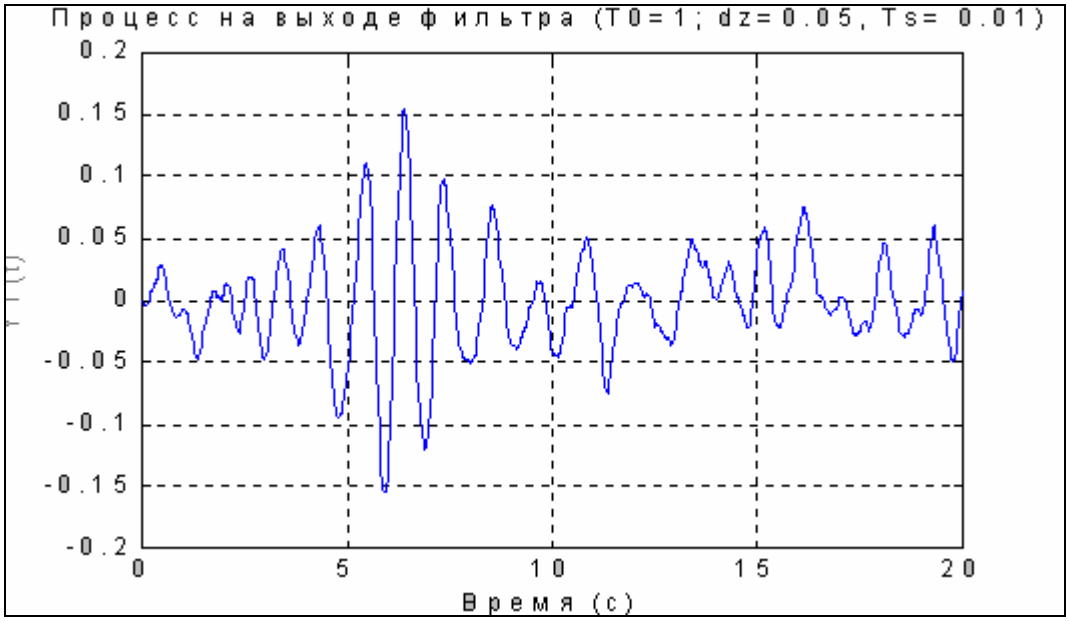

Рис. 5.25.

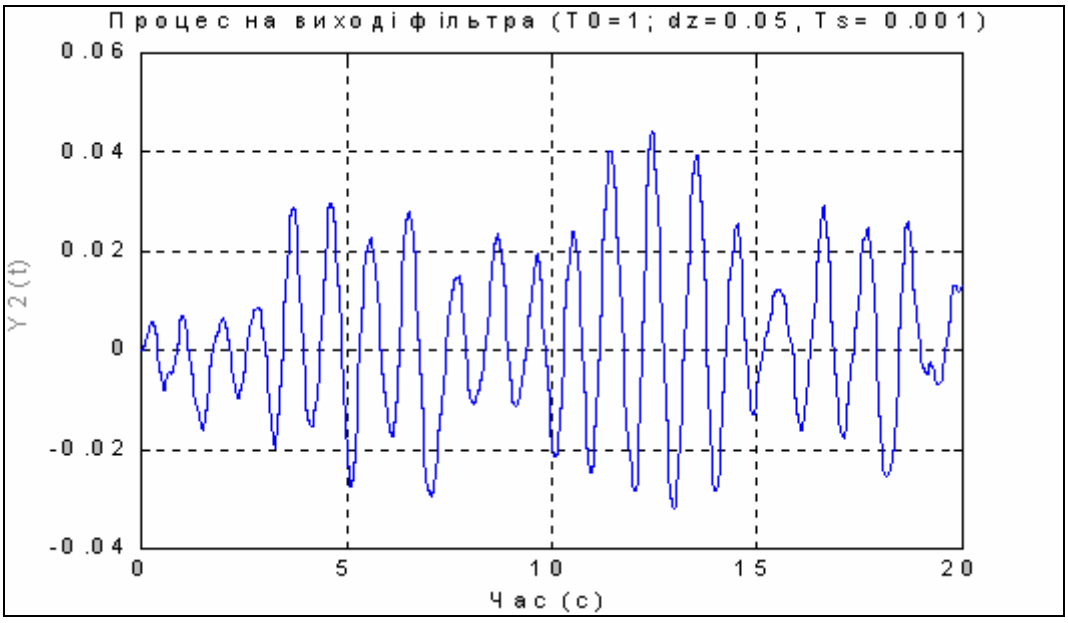

Рис. 5.26.

Как видим, на выходе формирующего фильтра действительно образуется случайный колебательный процесс с преобладающей частотой 1 Гц.

5.3. Процедуры спектрального (частотного) и статистического анализа процессов

5.3.1. Основы спектрального и статистического анализа

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

$$f_m = m \cdot f \qquad (m = 0,1,2,...) \qquad (5.10)$$

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

$$x(t) = a_o + \sum_{m=1}^{+\infty} a_m \cdot \cos(2\pi mft + \varphi_m), \qquad (5.12)$$

который, собственно, и называют *рядом Фурье*.

Для действительных процессов справедливы следующие соотношения

$$\mathrm{Re}\{X(-m)\} = \mathrm{Re}\{X(m)\}; \qquad \mathrm{Im}\{X(-m)\} = -\mathrm{Im}\{X(m)\}, \tag{5.13}$$

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, в том числе - непериодические процессы привело к введению понятия Фурье-изображения в соответствии со следующим выражением:

$$X(f) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-j(2\pi f)t} dt \tag{5.14}$$

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 $x(t)$, то 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 "двухсторонне затухающих" процессов (т.е. таких, которые стремятся к нулю как при $t \to +\infty$, так 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

$$x(t) = \int_{-\infty}^{+\infty} X(f) \cdot e^{j(2\pi f)t} df, \tag{5.15}$$

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

$$X[(k-1)\cdot\Delta f] = \Delta t \cdot \sum_{m=1}^{n} x[(m-1)\cdot\Delta t] \cdot e^{-j\cdot 2\pi\cdot(k-1)\cdot(m-1)\cdot\Delta f\cdot\Delta t}. \tag{5.16}$$

Тут, по сравнению с интегралом (5.14) осуществлены такие замены

- непрерывный интеграл приближенно заменен ограниченной суммой площадей прямоугольников, одна из сторон которых равна дискрету 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 T - промежуток времени, на котором задан процесс;
- дифференциал dt заменен ограниченным приращением времени Δt.

Если обозначить дискрет времени Δt через Ts, ввести обозначения

$$x(m) = x[(m-1)\cdot\Delta t]; \qquad X(k) = X[(k-1)\cdot\Delta f].$$

а также учесть то, что число n 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 разделе 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

$$X^*(k) = \frac{Ts}{T}\cdot\sum_{m=1}^{n} x(m)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} =$$

$$= \frac{1}{n}\cdot\sum_{m=1}^{n} x(m)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} = \frac{y(k)}{n} \tag{5.22}$$

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, просто 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:

$$R_{12}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{+T/2} x_1(t)\cdot x_2(t+\tau)\cdot dt\,, \tag{5.23}$$

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 так:

$$S_{12}(f) = \int_{-\infty}^{+\infty} R_{12}(\tau)\cdot e^{-j(2\pi f)\tau} d\tau \tag{5.24}$$

При числовых расчетах, когда оба процесса $x_1(t)$ и $x_2(t)$ заданы на определенном ограниченном промежутке T времени своими значениями в некоторых n точках, разделенных дискретом времени Ts, формулу (5.23) можно трансформировать в такую:

$$R_{12}(l) = \frac{1}{n-l}\sum_{m=1}^{n-l} x_1(m)\cdot x_2(m+l-1), \qquad (l = 1, 2, \ldots \quad n/2); \tag{5.25}$$

или в несколько более простое соотношение

$$R_{12}(l) = \frac{2}{n}\sum_{m=1}^{n/2} x_1(m)\cdot x_2(m+l-1), \qquad (l = 1, 2, \ldots \quad n/2); \tag{5.26}$$

а вместо (5.24) использовать

$$S_{12}(k) = Ts\cdot \sum_{l=1}^{n/2} R_{12}(l)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(l-1)}\,, \qquad (k = 1, 2, \ldots n/2). \tag{5.27}$$

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 :

$$S_{12}(k) = Ts\cdot\left\{\frac{2}{n} y_2(k)\right\}\cdot \overline{y}_1(k); \qquad (k = 1, 2, \ldots n/2), \tag{5.28}$$

где черта сверху означает комплексное сопряжение соответствующей величины.

С учетом (5.21) и (5.22) выражение (5.28) можно представить также в виде:

$$S_{12}(k) = X_2^*(k)\cdot \overline{X}_1(k)\,. \tag{5.29}$$

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, следует, 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*:

$$\boldsymbol{Fmax = 1/Ts}; \qquad (5.30)$$

- *по заданной длительности заданного процесса T рассчитать дискрет частоты df по формуле:*

$$\boldsymbol{df = 1/T}; \qquad (5.31)$$

- *по вычисленным данным сформировать вектор значений частот, в которых будет вычислено Фурье-изображение*.

Последнее проще (но не наиболее правильно) сделать таким образом:

$$\boldsymbol{f1=0 : df : Fmax.} \qquad (5.32)$$

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, надо сделать следующее (см. раздел 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*

$$\boldsymbol{f = -Fmax/2 : df : Fmax/2}. \qquad (5.33)$$

Приведем примеры.

Фурье-изображение прямоугольного импульса

Сформируем процесс, состоящий из одиночного прямоугольного импульса. Зададим дискрет времени Ts=0.01с, длительность процесса T=100с, амплитуду импульса A=0.75 и его ширину w=0.5с:

```
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.

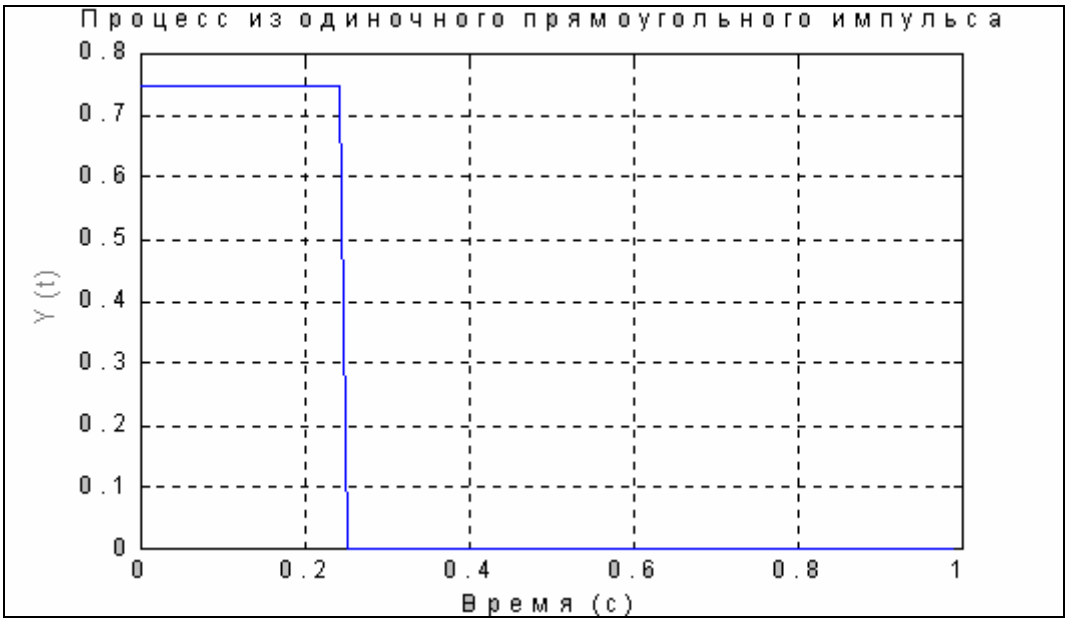

Рис. 5.27

Рис. 5.28

Применим к вектору ***y*** процедуру ***fft*** и построим график зависимости модуля результата от частоты. При этом *графики в частотной области удобнее выводить при помощи процедуры* ***stem*** (см. 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')
```

Теперь построим график модуля Фурье-изображения процесса:

```
xp=fftshift(x);      f1=-Fmax/2 : df : Fmax/2;     a=abs(xp);
stem(f1,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.

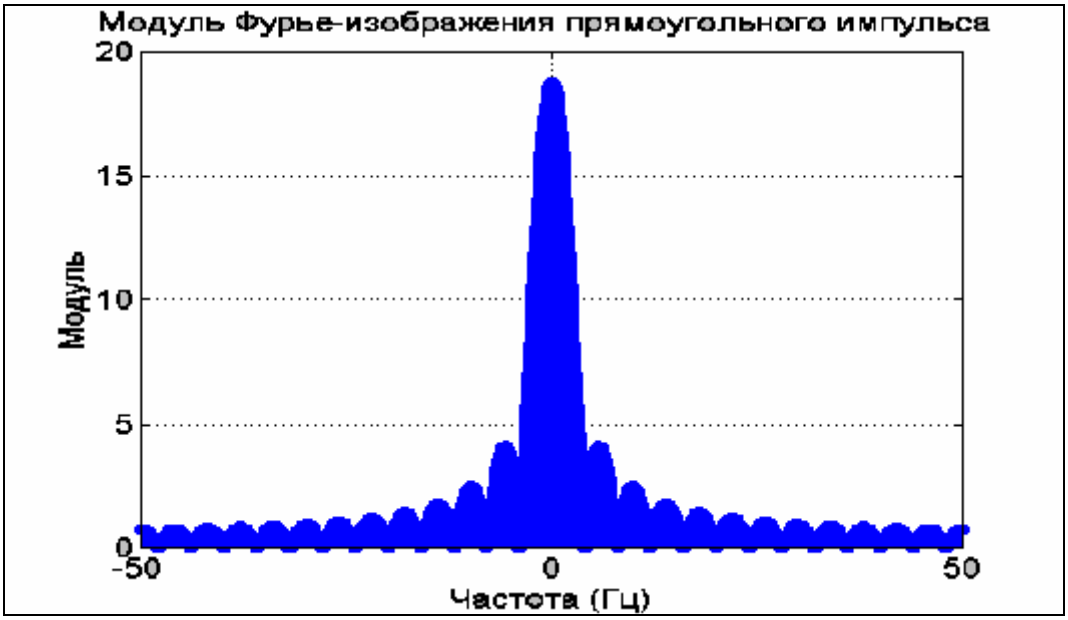

Рис. 5.29

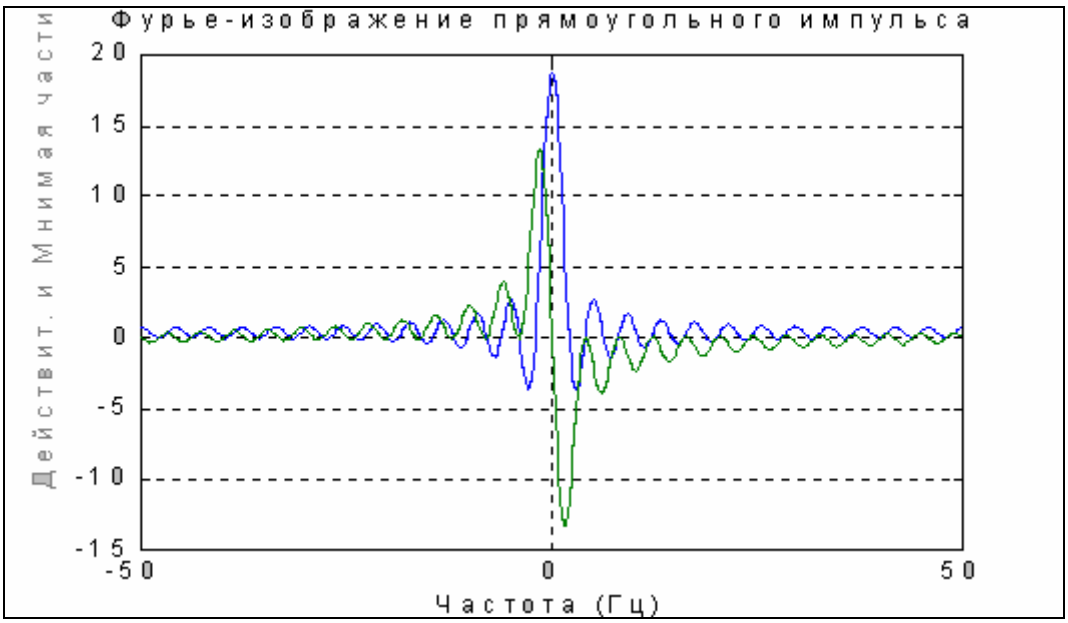

Рис. 5.30

В заключение построим графики действительной и мнимой частей Фурье-изображения прямоугольного импульса:

```
dch=real(xp);          mch=imag(xp);
plot(f1,dch,f1,mch), grid, set(gca,'FontName','Arial Cyr','FontSize',16),
title('Фурье-изображение прямоугольного импульса ');
ylabel('Действит. и Мнимая части'),     xlabel('Частота (Гц)');
```

Они представлены на рис. 5.30.

Фурье-изображение полигармонического процесса

Рассмотрим пример трехчастотных гармонических колебаний - с частотою $1/\pi$, 1 та 3 Гц и амплитудами соответственно 0.6, 0.3 та 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.

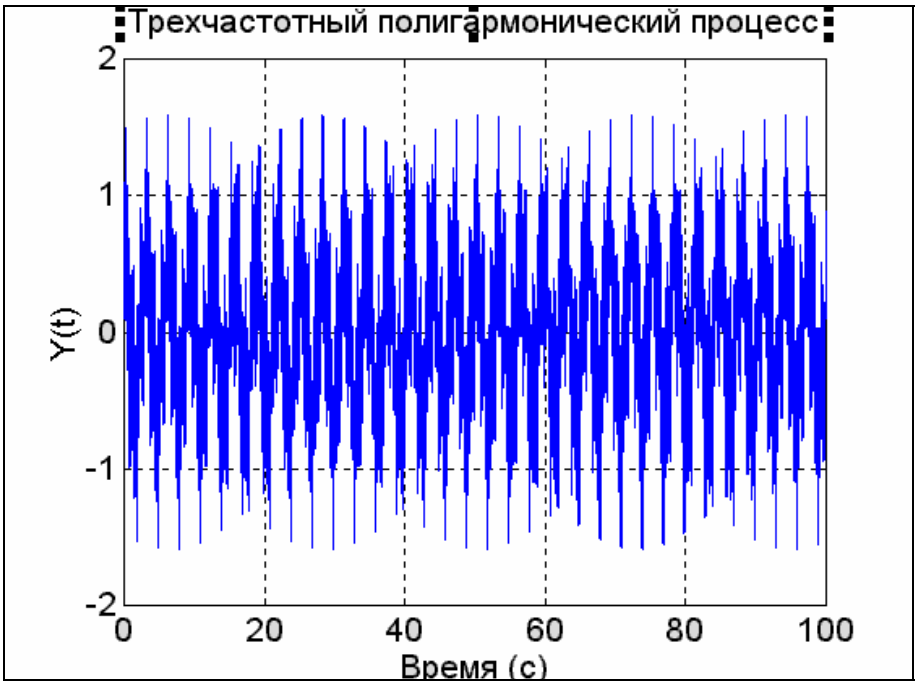

Рис. 5.31

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.

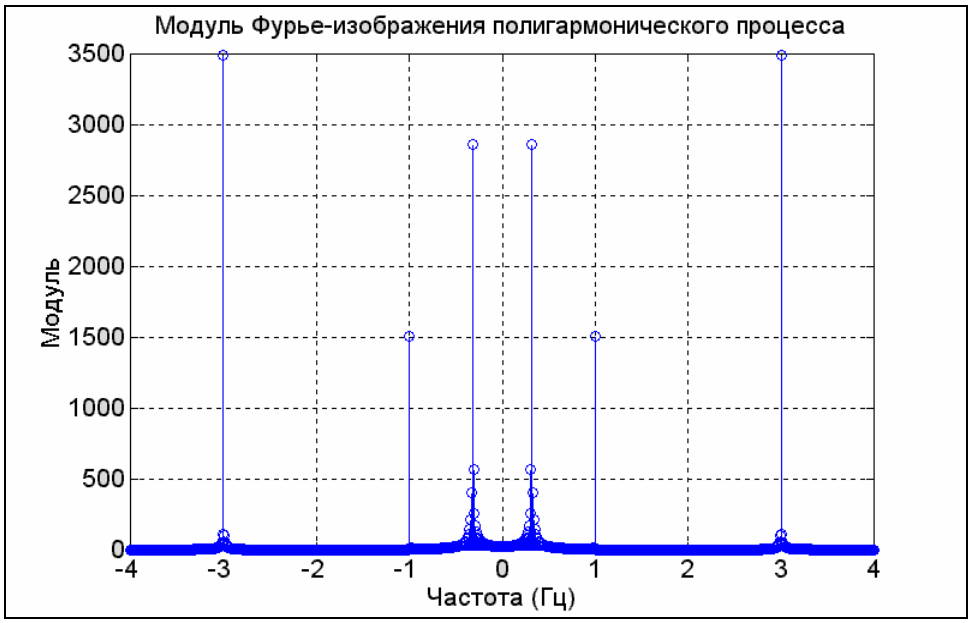

Рис. 5.32

Если изменить дискрет времени на Ts=0.02, получим результат, изображенный на рис. 5.33.

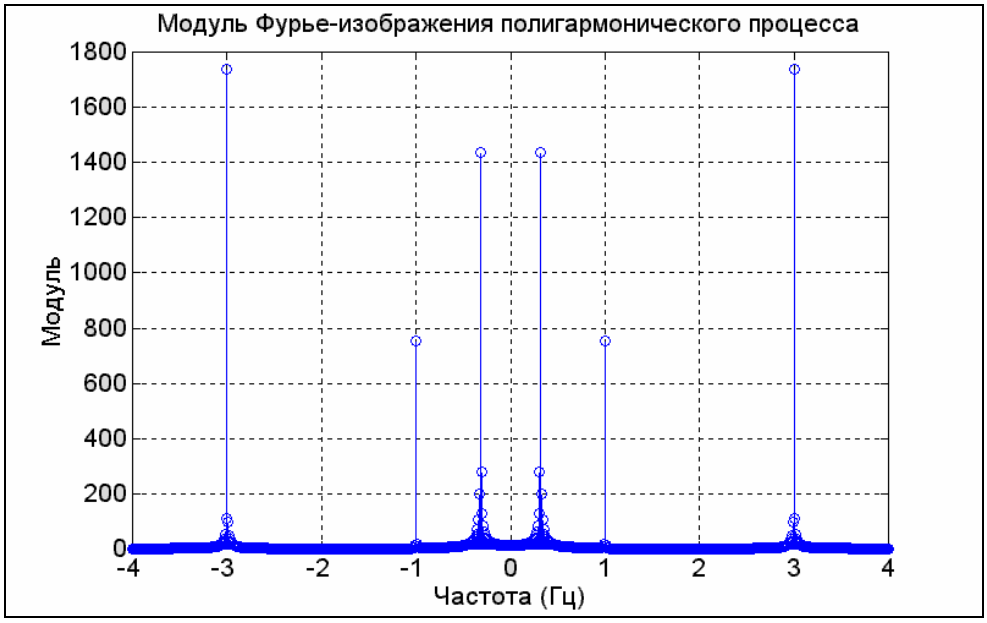

Рис. 5.33

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.

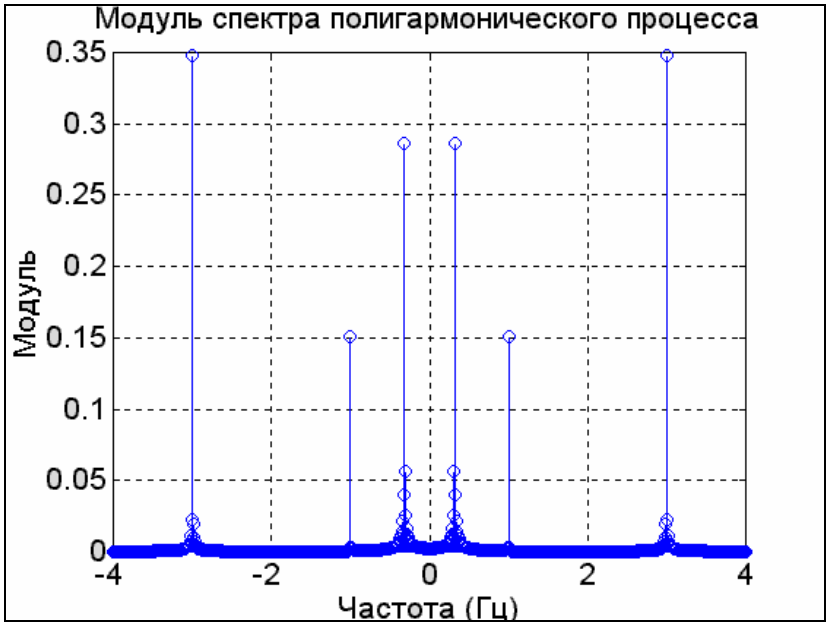

Рис. 5.34

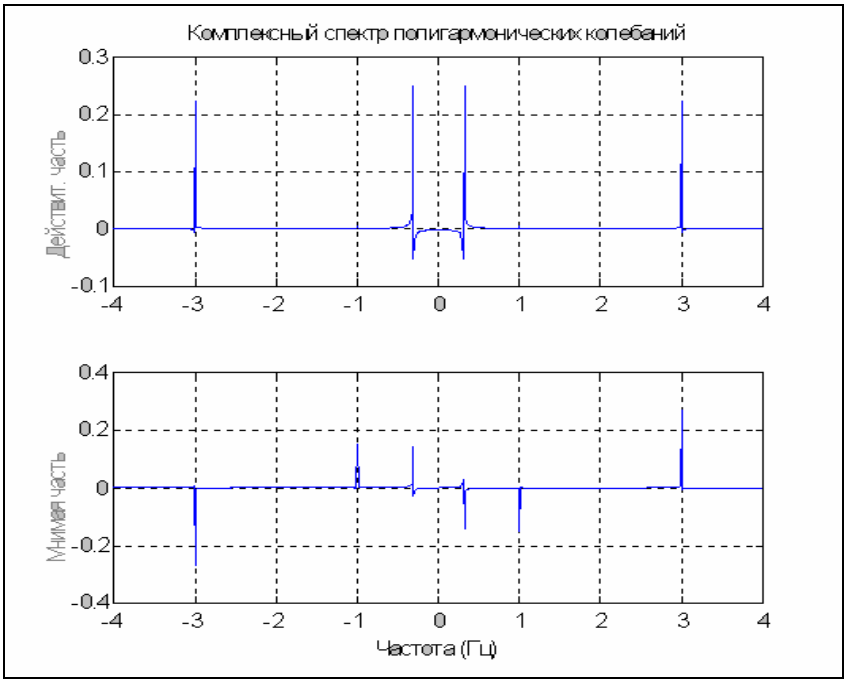

Рис. 5.35

В результате получается комплексный спектр (рис. 5.34), полностью соответствующий коэффициентам комплексного ряда Фурье.

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, а и о начальных фазах отдельных гармонических составляющих.

Фурье-изображение случайного процесса

В заключение рассмотрим Фурье-преобразование случайного стационарного процесса, сформированного ранее (см. рис. 5.25). Сначала сформируем процесс в виде белого гауссового шума (рис. 5.36) с шагом во времени 0.01 и длительностью в 100 с :

```
Ts=0.01;  T = 100;   t=0 : Ts : T;  % Задание параметров процесса
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)');
```

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)')
```

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):

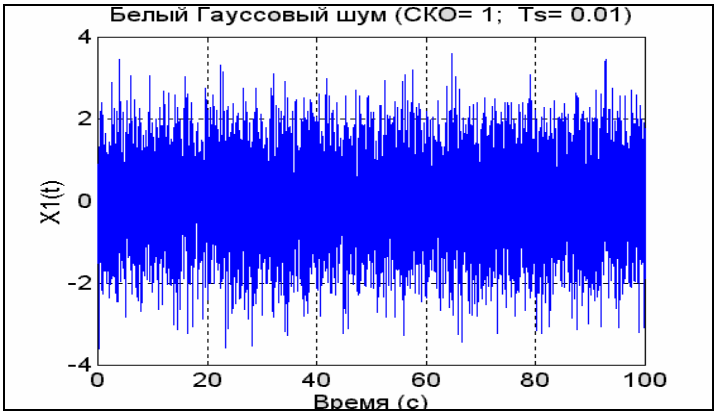

Рис. 5.36

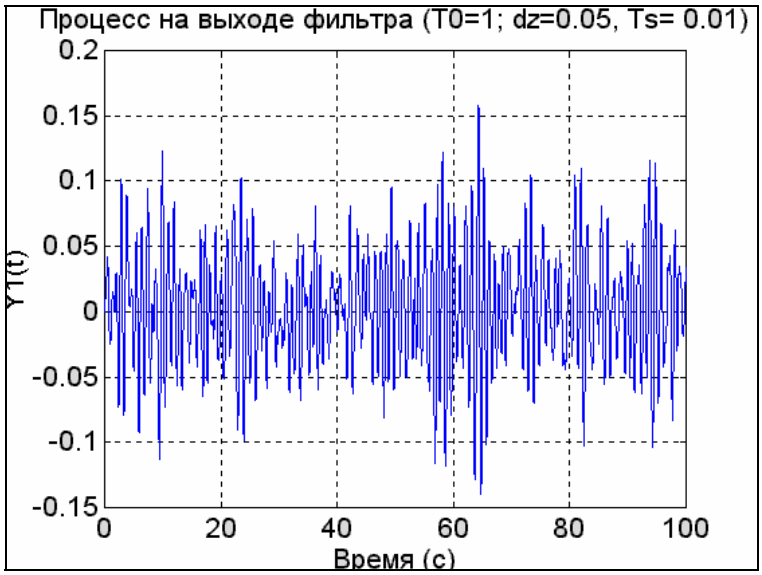

Рис. 5.37

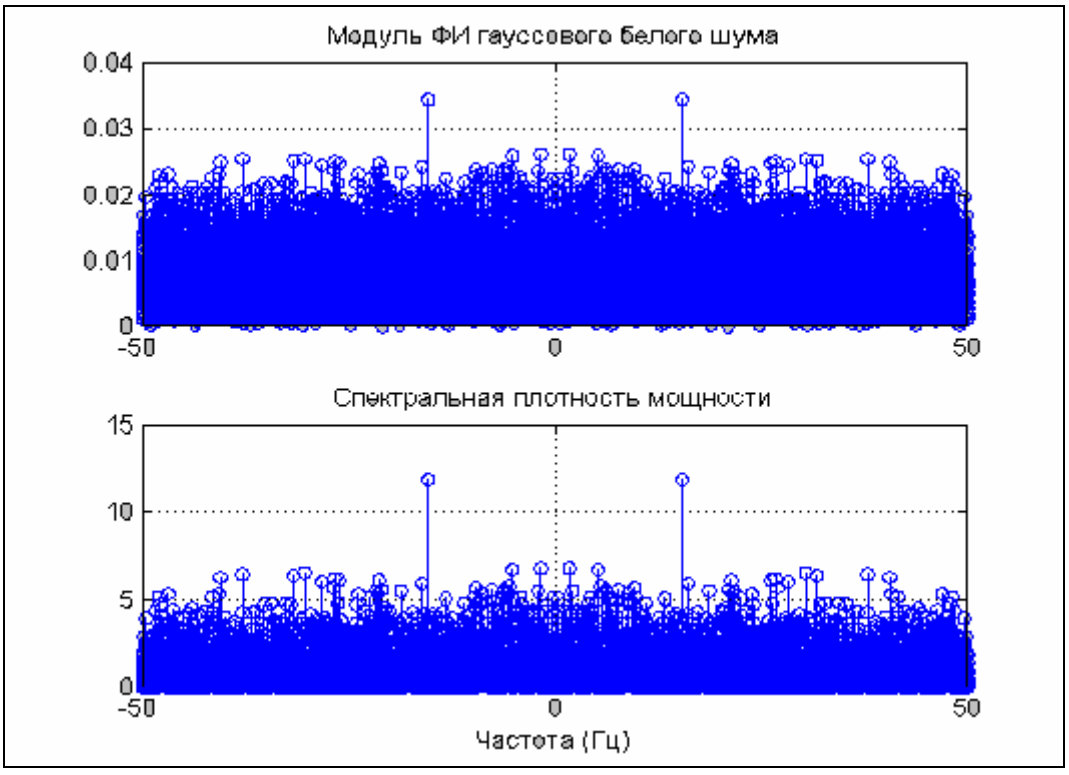

Рис.5.38

```
% 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)');
```

Рассматривая рис. 5.38, можно убедиться, что спектральная плотность практически одинакова по величине во всем диапазоне частот, чем и обусловлено название процесса - "белый шум".

Аналогичную процедуру проведем и с "профильтрованным" процессом (рис. 5.39):

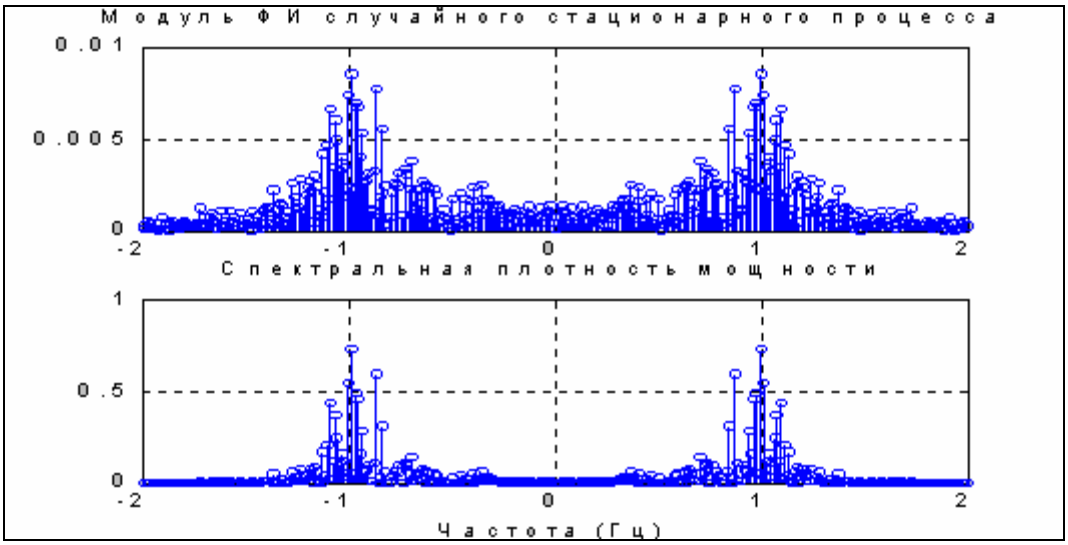

Рис. 5.39

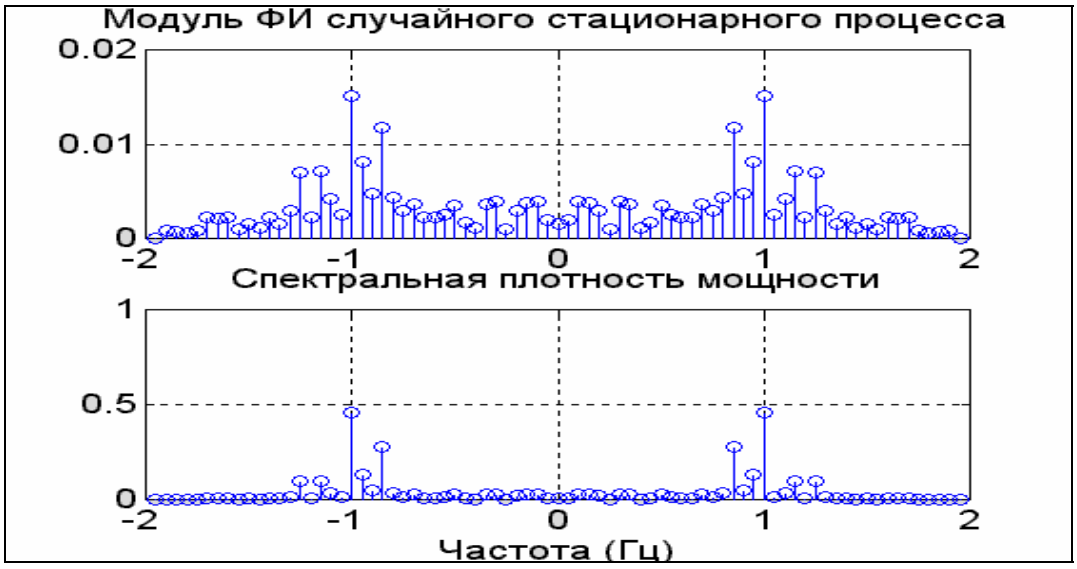

Рис. 5.40.

```
% 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 (см. рис. 5.40), можно наглядно убедиться, что величины ФИ и СПМ практически при этом не изменяются.

5.3.3. Статистический анализ

К задачам статистического анализа процессов относятся определение некоторых статистических характеристик процессов, таких, как корреляционные характеристики, спектральные плотности мощности и т. д.

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 x, которые обрабатываются процедурой ***fft***, $F\max = 1/Ts$ - значение частоты дискретизации сигналу, 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);
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

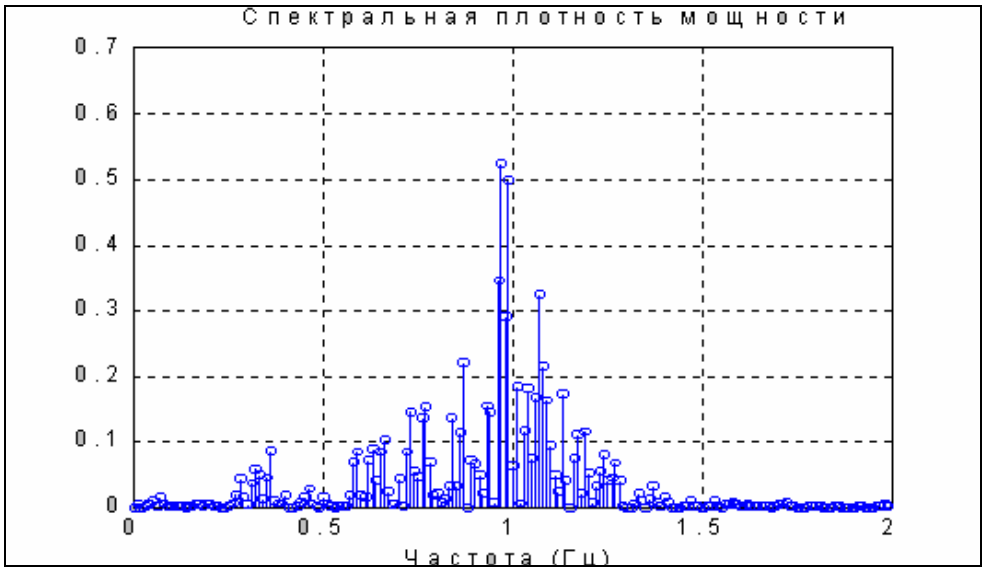

Рис. 5.41

Если ту же процедуру вызвать *без указания выходных величин*, то результатом ее выполнения станет *выведение графика СП от частоты*.

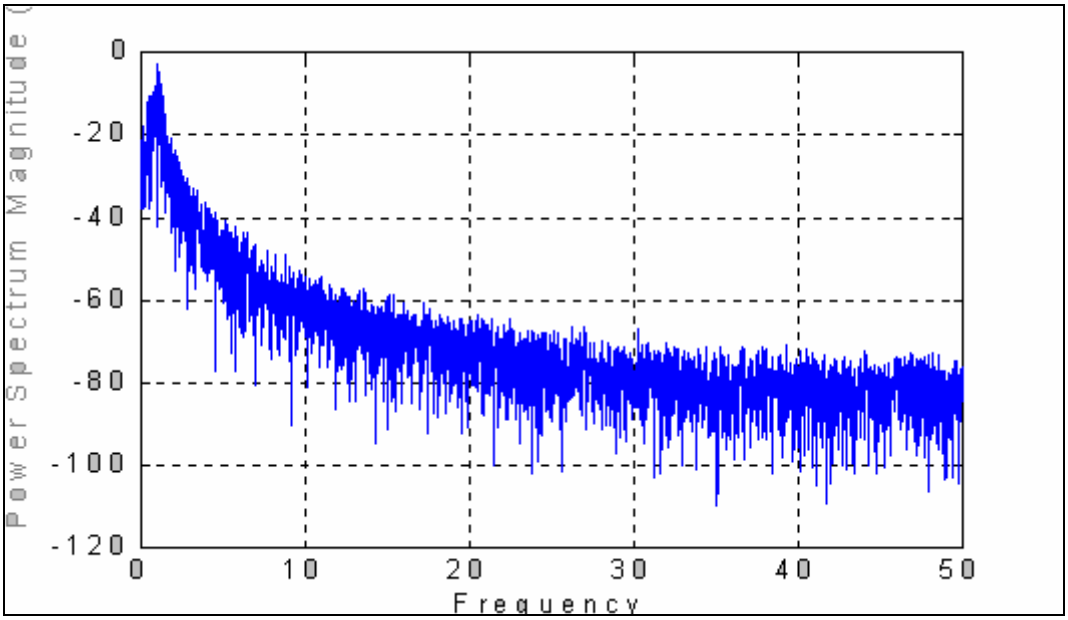

Рис. 5.42

Например, обращение

psd(y1, dovg, Fmax)

приведет к построению в графическом окне (фигуре) графика рис. 5.42. При этом значения СП будут откладываться в логарифмическом масштабе в децибелах.

Рис. 5.43

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. Обращение ***c = xcorr(x)*** 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***.

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

$$W(s)=\frac{b(s)}{a(s)}=\frac{b(1)\cdot s^{m}+b(2)\cdot s^{m-1}+...+b(m+1)}{a(1)\cdot s^{n}+a(2)\cdot s^{n-1}+...+a(n+1)}, \qquad (5.34)$$

  а в случае дискретного фильтра последний может быть представлен дискретной передаточной функцией вида

$$W(z)=\frac{b(z)}{a(z)}=\frac{b(1)+b(2)\cdot z^{-1}+...+b(m+1)\cdot z^{-m}}{a(1)+a(2)\cdot z^{-1}+...+a(n+1)\cdot z^{-n}}; \qquad (5.35)$$

  в обоих случаях для задания звена достаточно задать два вектора - вектор ***b*** 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):

$$\frac{b(z)}{a(z)}=\frac{r(1)}{1-p(1)\cdot z^{-1}}+...+\frac{r(n)}{1-p(n)\cdot z^{-1}}+k(1)+k(2)\cdot z^{-1}+... \qquad (5.36)$$
$$...+k(m-n+1)\cdot z^{-(m-n)};$$

  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:

$$H(z)=\prod_{k=1}^{L}H_k(z)=\prod_{k=1}^{L}\frac{b_{ok}+b_{1k}\cdot z^{-1}+b_{2k}\cdot z^{-2}}{a_{ok}+a_{1k}\cdot z^{-1}+a_{2k}\cdot z^{-2}}; \qquad (5.37)$$

параметры каскадного представления задаются в виде матрицы ***sos***, содержащей вещественные коэффициенты:

$$sos=\begin{bmatrix} b_{o1}b_{11}b_{21}a_{o1}a_{11}a_{21} \\ b_{o2}b_{12}b_{22}a_{o2}a_{12}a_{22} \\ \cdots\cdots\cdots\cdots \\ b_{oL}b_{1L}b_{2L}a_{oL}a_{1L}a_{2L} \end{bmatrix}; \qquad (5.38)$$

- в пространстве состояний (***ss*** *-представление)*, т. е. с помощью уравнений звена в форме

$$\begin{aligned} \dot{x} &= A\cdot x+B\cdot u \\ y &= C\cdot x+D\cdot u \end{aligned}; \qquad (5.39)$$

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)*:

$$W(s)=k\frac{[s-z(1)]\cdot[s-z(2)]\cdots[s-z(m)]}{[s-p(1)]\cdot[s-p(2)]\cdots[s-p(n)]}; \qquad (5.40)$$

- 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 ***b*** коэффициентов числителя и ***a*** знаменателя 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).

Вторая форма - если решетчатый БИХ-фильтр имеет только полюсы.

Третья и четвертая формы применяются 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 использованием 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 таковым:

[r, p, k] = ***residue***(b, a)
[r, p, k] = ***residuez***(b, a).

*Процедуры перехода от **tf** - формы к другим*

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 так:

[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 $a \neq 0$. 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 прове-

рить, 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 $W(s)$ и измеряется в децибелах:

$$A(\omega) = 20 \cdot \lg\{\ |W(j\omega)|^{-1}\ \} = 10 \cdot \lg L(\omega^2)\,, \tag{5.42}$$

"*фазовую характеристику*" $\vartheta(\omega)$:

$$\vartheta(\omega) = \arg(H(j\omega)) \tag{5.43}$$

и "*характеристику групповой задержки*" τ

$$\tau = -\frac{d\vartheta(\omega)}{d\omega}\ . \tag{5.44}$$

Функцию $$L(s^2) = \left[H(s) \cdot H(-s)\right]^{-1} \tag{5.45}$$

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 $\pm z_i$.

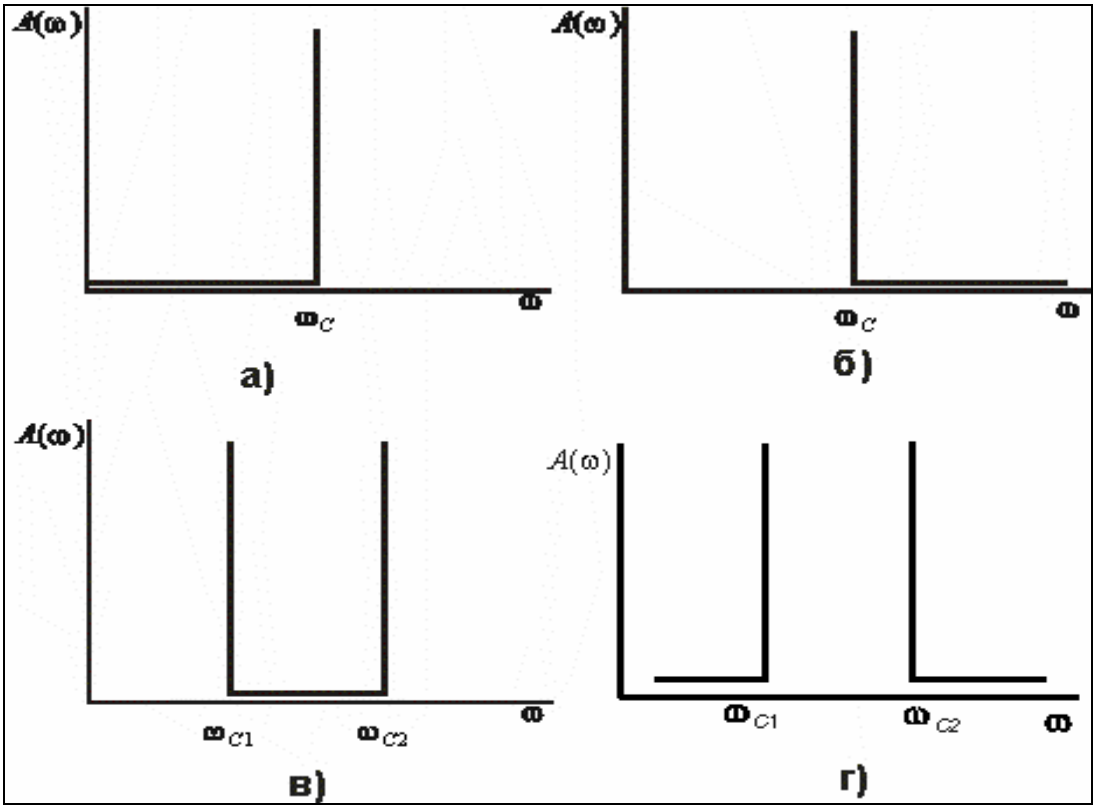

Рис. 5.44

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_{-} называется *полосой пропускания*, остальной частотный диапазон - *полосой задерживания*. *Граница между этими полосами*

(ω_-) *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*.

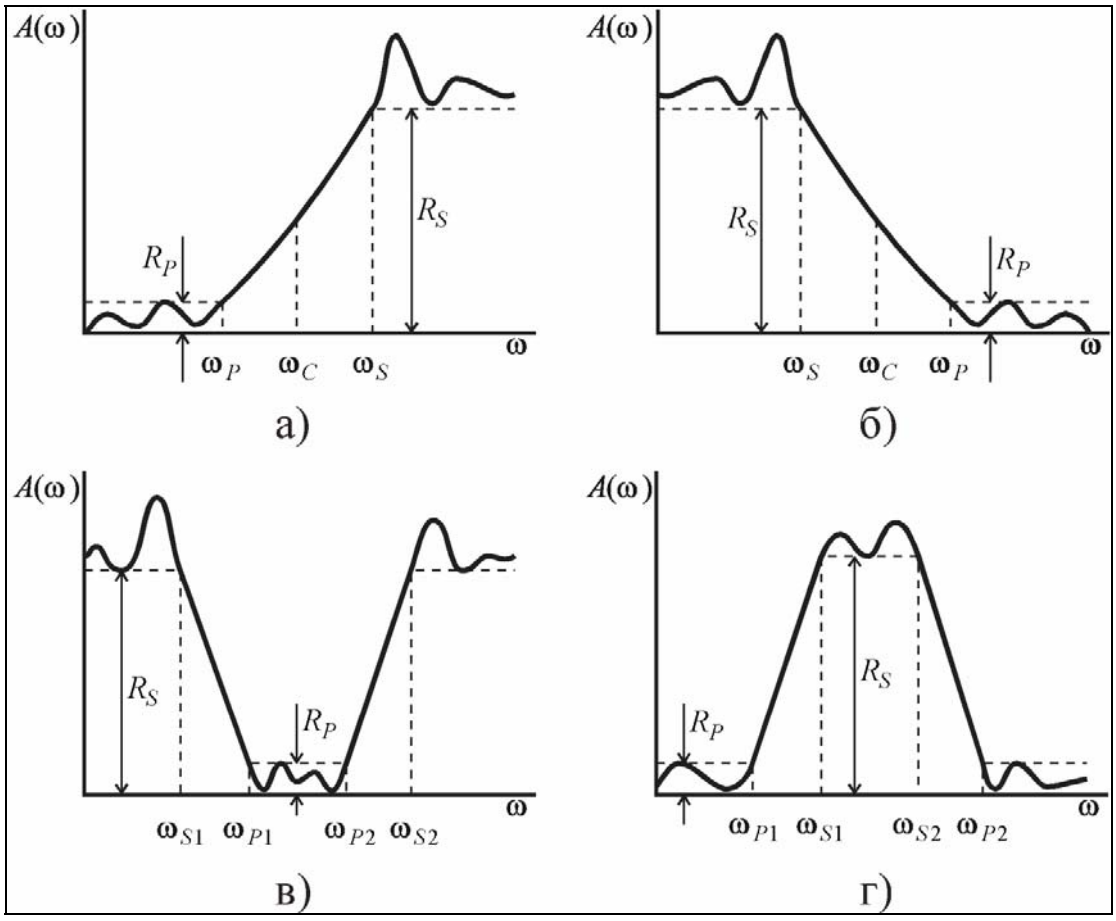

Рис. 5.45

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 ω_- 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.

Аппроксимацией фильтра называют реализуемую передаточную функцию, у которой график характеристики затухания $A(\omega)$ как функция от частоты приближается к одной из идеальных характеристик 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 требуемым характеристикам фильтра:

W_P - граничной частоте пропускания;

W_S - граничной частоте задерживания;

R_P - максимально допустимому подавлению в полосе пропускания, дБ;

R_S - 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 W_n 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, а 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.*** *Все они создают 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] = ***cheb1ap***(n, Rp).

ФНЧ Чебышева 2-го типа, имеющий величину подавления в полосе задерживания не менее Rs дБ, "создается" 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),

где смысл параметра Wo несколько иной - это центральная частота полосы пропускания, а Bw означает ширину полосы пропускания. В остальном особенности использования и смысл обозначений сохраняется прежним.

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.

Следующую 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 всех их является следующее:

- с их помощью можно проектировать ФНЧ, ФВЧ, полосовые и режекторный фильтры соответствующей аппроксимации; если параметр Wc является скаляром и не указан после него флажок 'high', 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 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)

$$G(z) = \frac{y(z)}{x(z)} = \frac{b_o + b_1 \cdot z^{-1} + ... + b_m \cdot z^{-m}}{a_o + a_1 \cdot z^{-1} + ... + a_n \cdot z^{-n}}.$$

Если эти два вектора известны, осуществление самой фильтрации, как было сказано ранее, происходит применением процедуры ***filter***, в которой аргументами выступают эти векторы.

Напомним, что представленная дискретная передаточная функция описывает в сжатой форме такое конечно-разностное уравнение фильтра

$$\begin{aligned} &a_o \cdot y(k) + a_1 \cdot y(k-1) + a_2 \cdot y(k-2) + ... + a_n \cdot y(k-n) = \\ &= b_o \cdot x(k) + b_1 \cdot x(k-1) + ... + b_m \cdot x(k-m) \end{aligned} \qquad (5.46)$$

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 последний называют также БИХ-фильтров. 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***, которая имеет три формы обращения к ней:

[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. Последний 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 фильт-

ра 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, т. е. 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
```

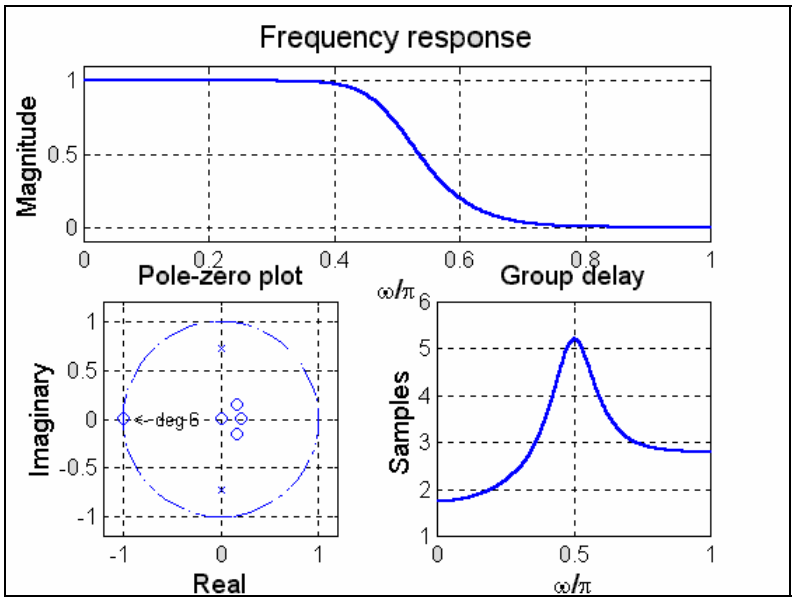

Рис. 5.46

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".

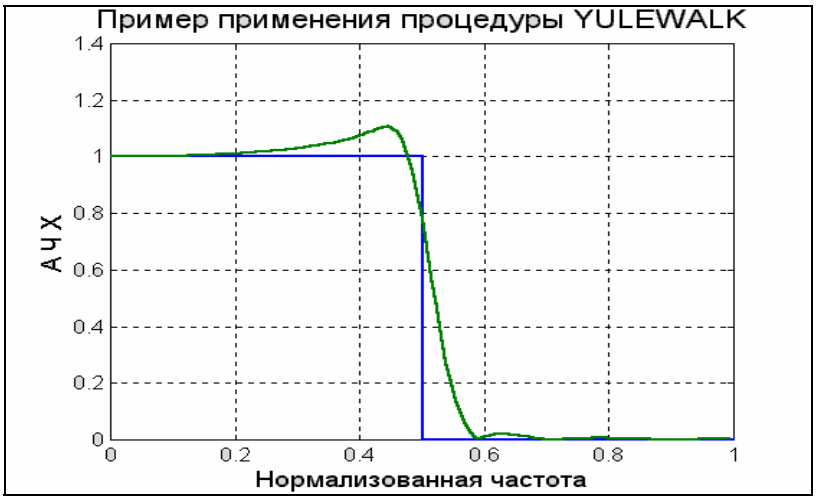

Рис. 5.47

Приведем пример расчета ФНЧ 8-го порядка и построим желаемую АЧХ и АЧХ полученного фильтра. Результат приведен на 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 применения процедуры YULEWALK')
xlabel('Нормализованная частота'), ylabel('A Ч X')
```

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 имеет вид

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 полосовой фильтр с полосой пропускания от W1 до W2, если 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 многополосового фильтра, первая полоса которого является полосой пропускания, а в четвертом, - тоже многополосовый 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 следующие 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 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.

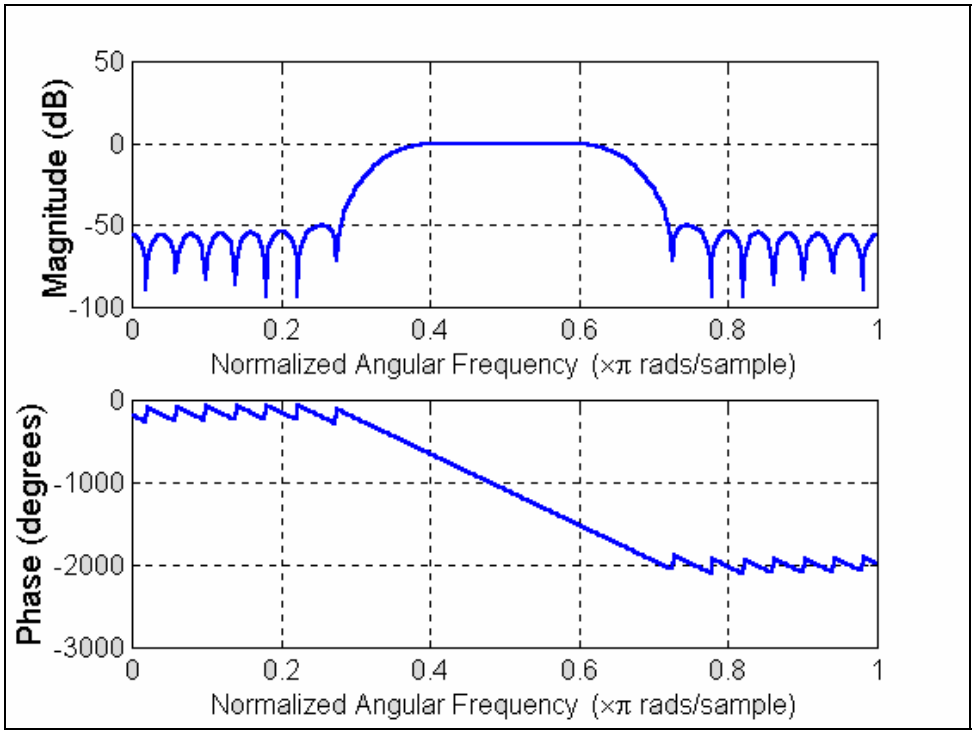

Рис. 5.48

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.

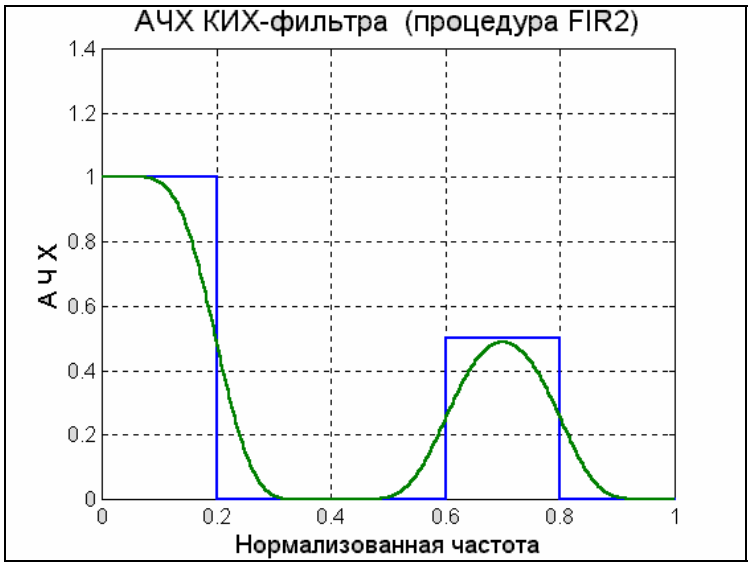

Рис. 5.49

Следующая процедура - ***fircls*** **-** также рассчитывает многополосный фильтр, но в несколько другой форме - путем задания кусочно-постоянной желаемой АЧХ.

Формат обращения к ней таков

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 желаемую 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 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
```

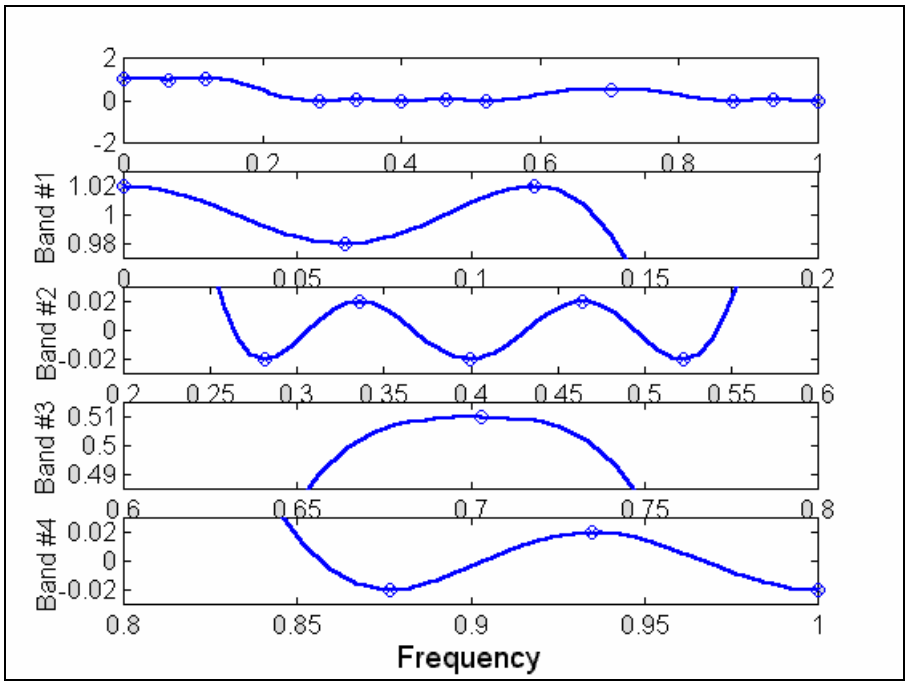

Рис. 5.50

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.

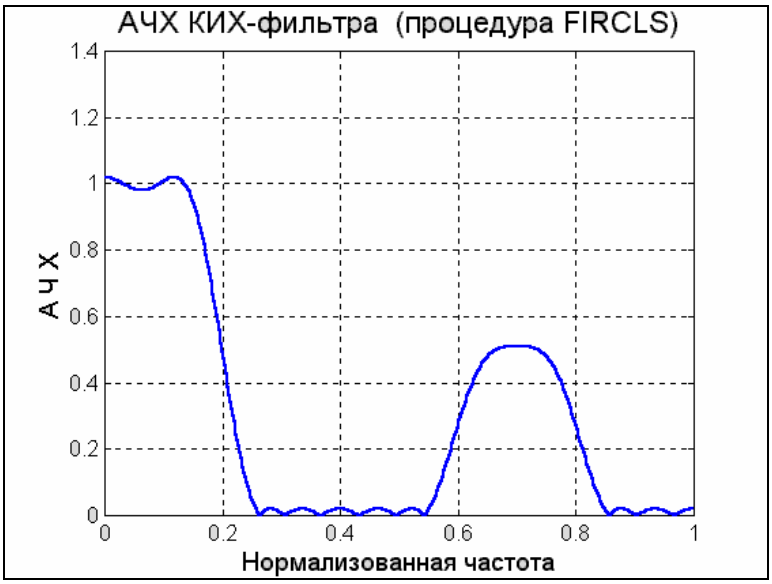

Рис. 5.51

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 Wo 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.

Указание параметра Wt позволяет задать частоту 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 k нечетном 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 k - четном значение желаемой АЧХ не определено (a, значит, при проектировании фильтра АЧХ в этих диапазонах может принимать любое значение). Следует заметить, что f(1) должно всегда быть равным 0. Векторы f и a должны быть одинаковой длины, причем общее количество элементов каждого вектора должно быть четным числом.

Вектор W задает значения коэффициентов веса каждого 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.

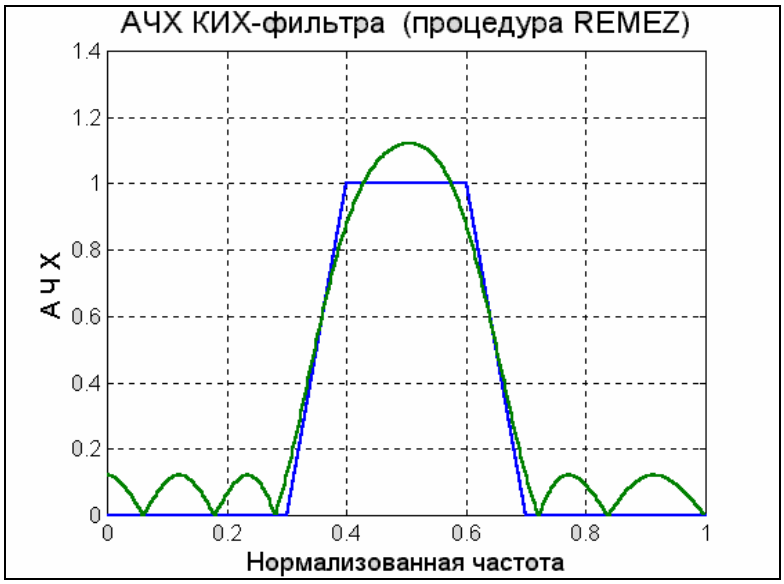

Рис. 5.52

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 требования к их представлению такие же, как и 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 (обращения к процедуре первого и второго видов), 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 (многополосовой 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.

Обращение к процедуре четвертого вида эквивалентно обращению

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' (см. 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 вид вызова процедуры) определяет вид выводимых на екран 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 использовать 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)
```

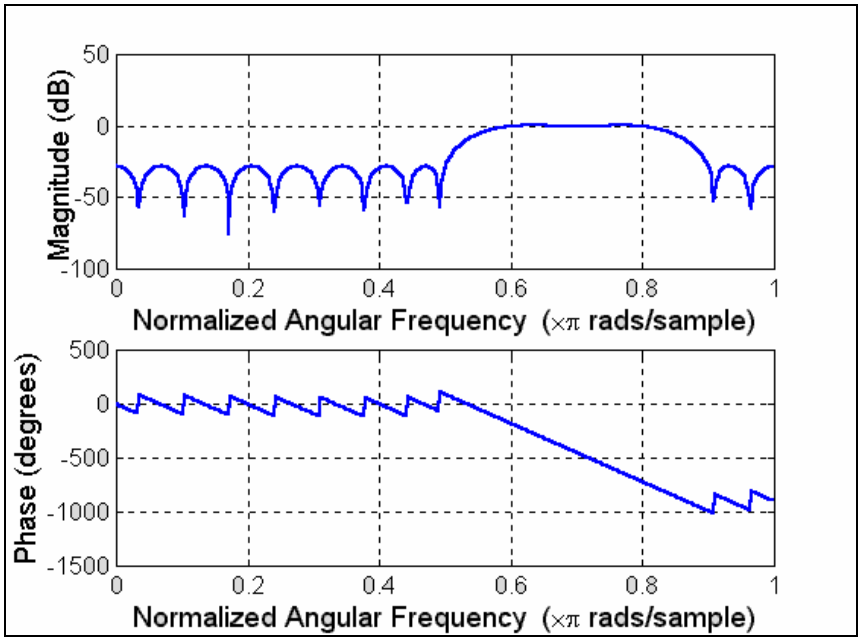

Рис. 5.53

5.5. Графические и интерактивные средства

5.5.1. Графические средства пакета SIGNAL

Некоторые графические средства пакета SIGNAL уже упоминались ранее. Сюда 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 (с. 247), а функции ***freqz*** - на рис. 5.17 (с.248). Из рассмотрения графиков следует:

- АЧХ первая процедура строит в логарифмическом масштабе, а вторая - в децибелах;
- частоты в первом случае откладываются в радианах в секунду и в лога-

рифмическом 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 'plot'.

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.

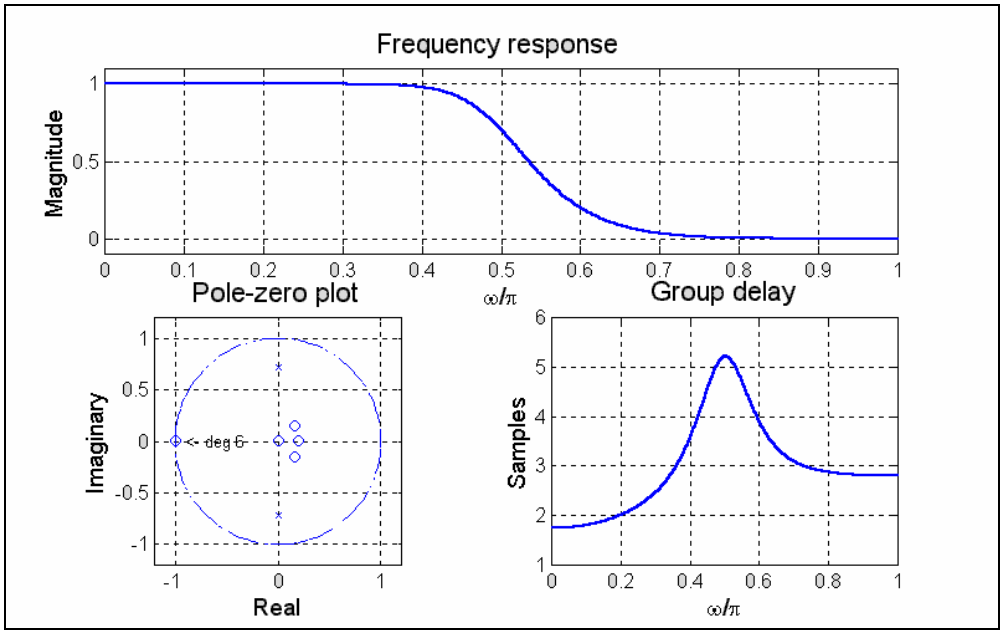

Рис. 5.54

При вызове функции ***fircls*** с этим флагом на график выводятся фрагменты АЧХ с максимальными отклонениями от требуемой АЧХ (см. 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');
```

Аналогичные графики строятся и при вызове функции ***fircls1***. Отличие в том, что теперь графики не орнаментированы никаким текстом (рис. 5.56):

```
fircls1(n,0.5,0.01,0.01,'plots');
```

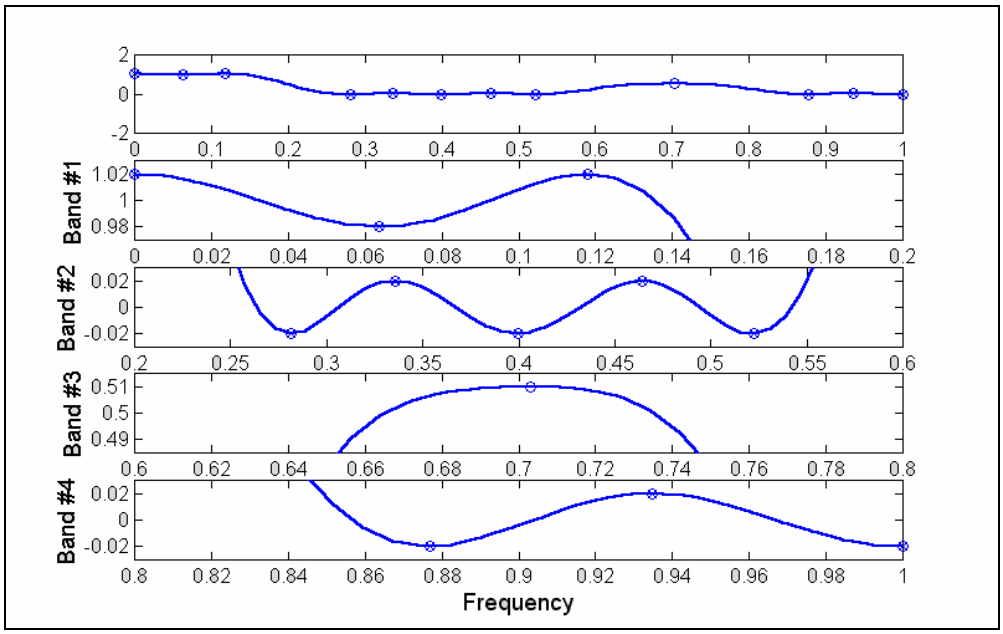

Рис. 5.55

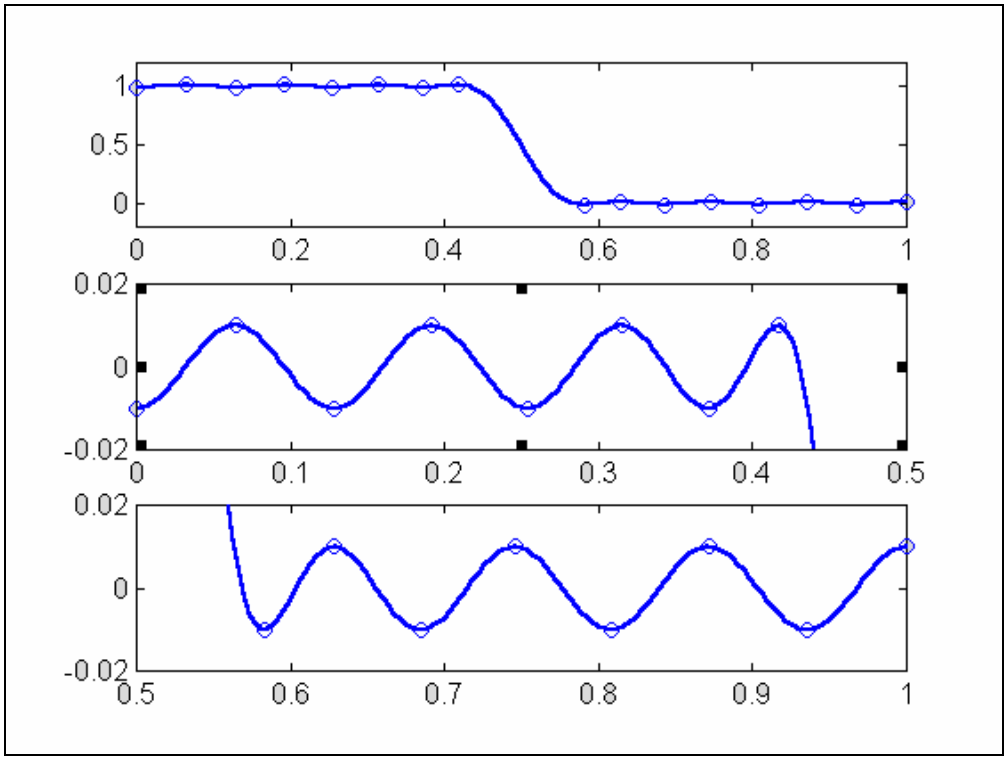

Рис. 5.56

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 проиллюстрировано на рис. 5.57:

```
cremez(30,[0 0.5 0.6 0.8 0.9 1],'bandpass','plots');
```

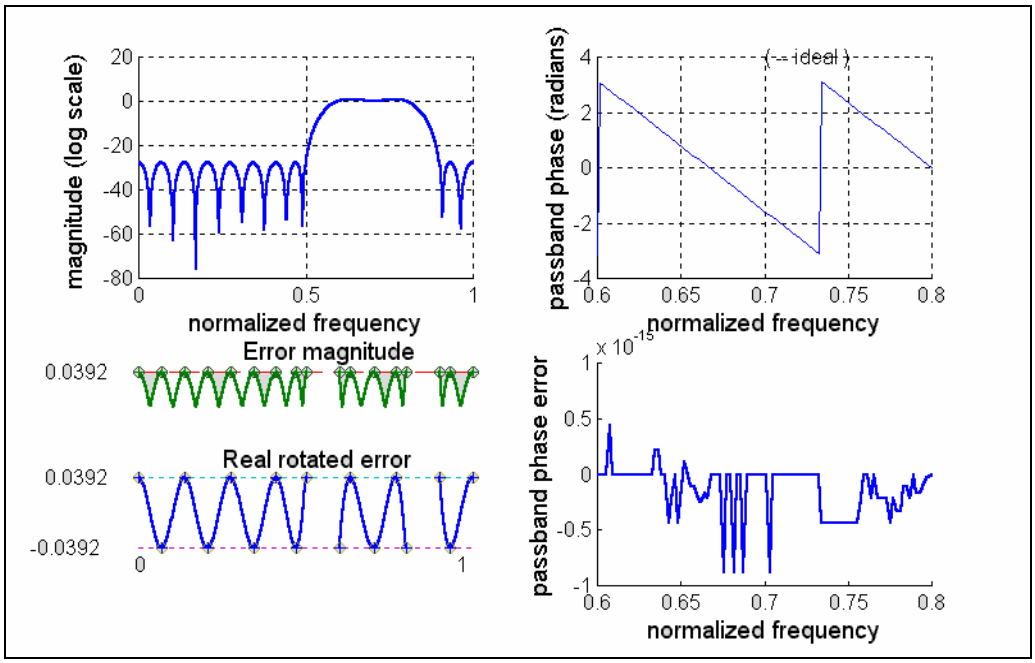

Рис. 5.57

В пакете SIGNAL имеются еще две важные для инженера графические процедуры ***grpdelay***, ***impz*** и ***zplane***. Первая строит график группового времени задержки (ГВЗ) от частоты, вторая - импульсную характеристику заданного фильтра, а третья отображает на комплексной Z-плоскости положение нулей и полюсов фильтра.

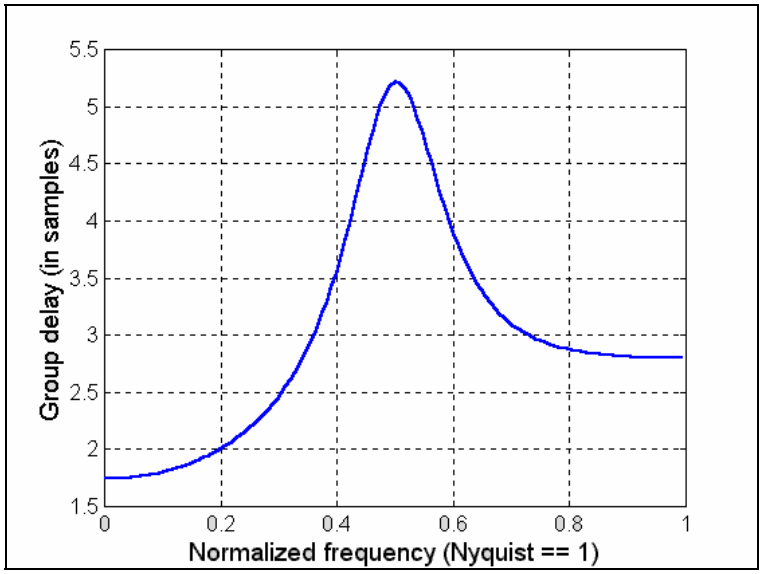

Рис. 5.58

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, получим график импульсной дискретной характеристики фильтра, изображенный на рис. 5.59:

impz(b,a) .

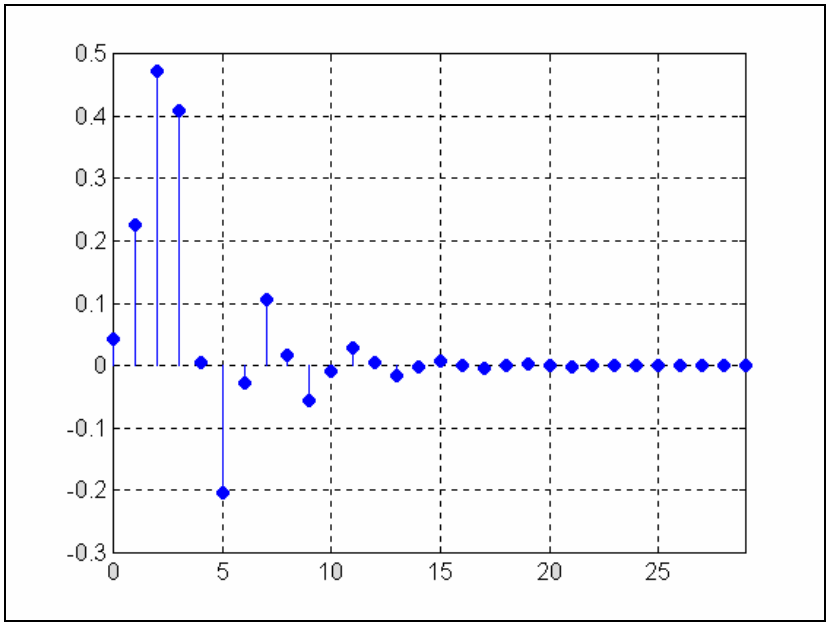

Рис. 5.59

Использование процедуры ***zplane*** для этого фильтра:

zplane(b,a)

приводит к построению графика рис. 5.60.

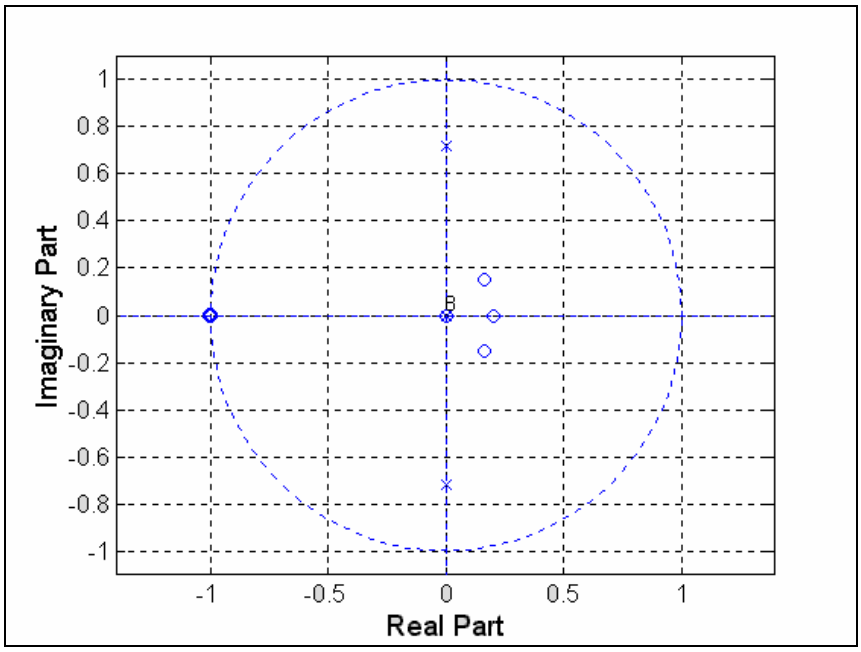

Рис. 5.60

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)');
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.

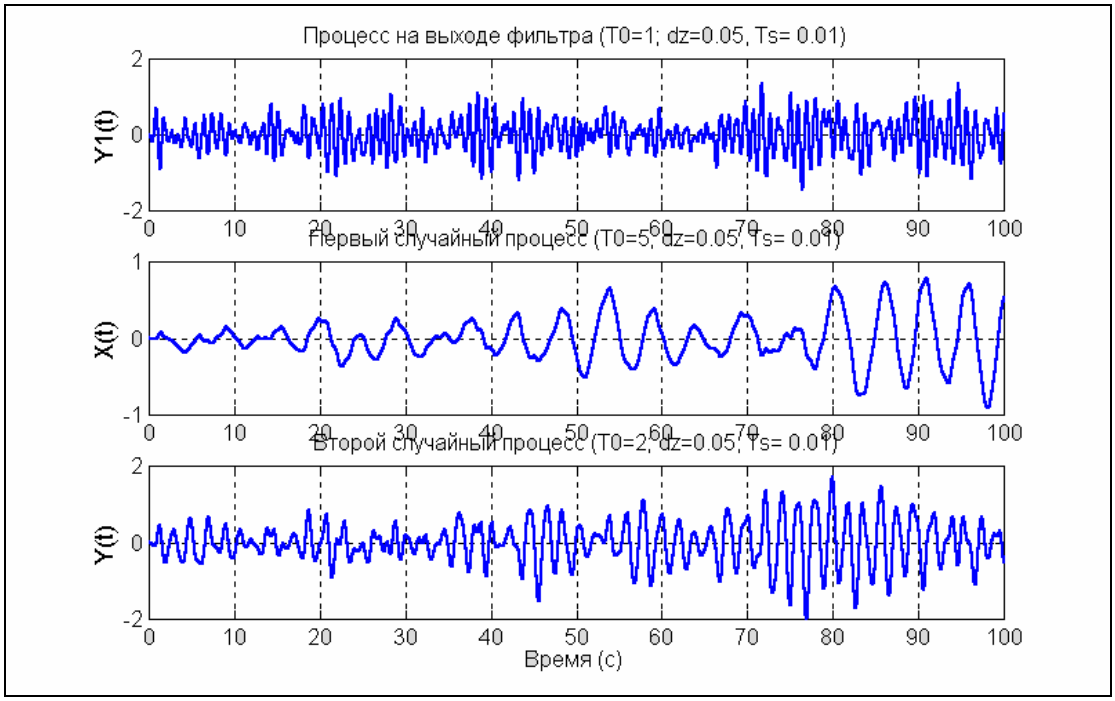

Рис. 5.61

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.

В качестве примера, выведем график порождающего случайного процесса, разбивая его на отдельные фрагменты по 20 секунд и задавая диапазон изменеия значения функции в каждом фрагменте от -2 до 2:

```
strips(y1,20,100, 2)
grid; title ('Застосування процедури STRIPS для виведення Y1(t)');
xlabel('Час, с')
```

На рис. 5.62 представлен результат.

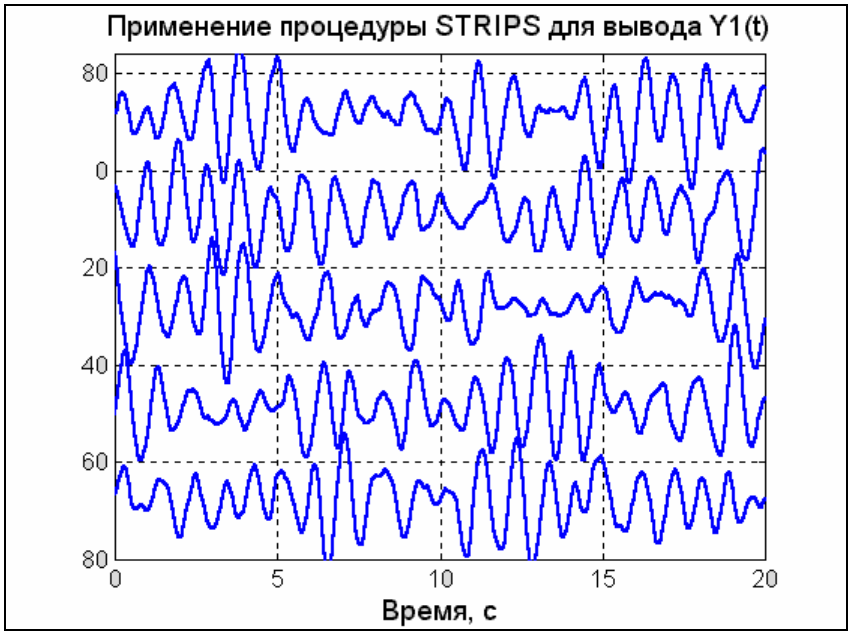

Рис. 5.62

Теперь познакомимся с графическими процедурами статистической обработки процессов. Ранее (разд. 5.3) мы познакомились с применением функции ***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 ')
csd(x,y,10000,100); title(' Применение процедуры CSD к процессам X(t)  и Y(t)');
ylabel('Взаимная    С  П'); xlabel('Частота, Гц ')
```

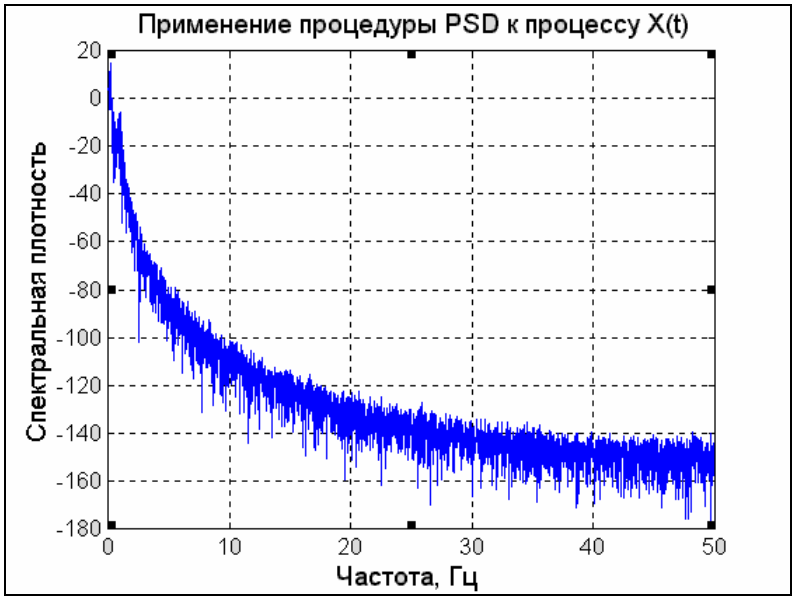

Рис. 5.63

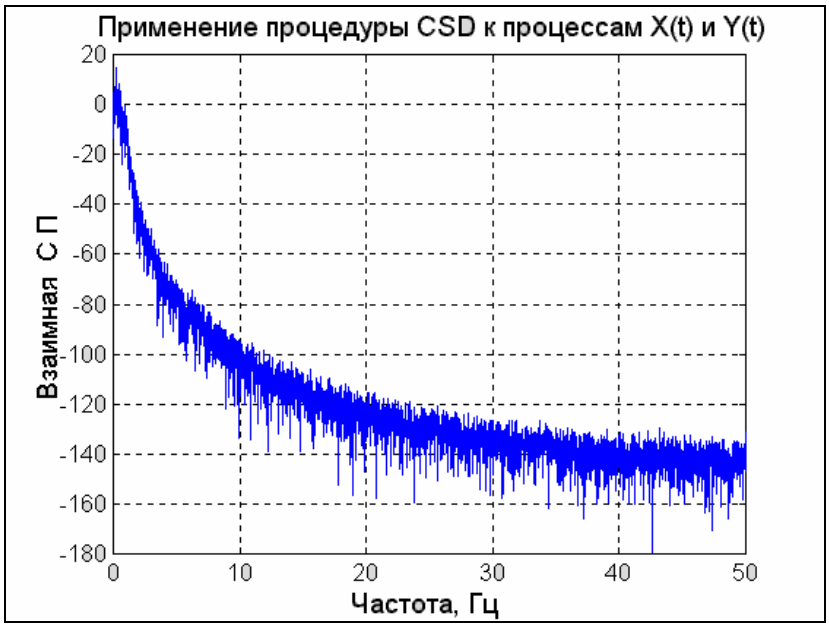

Рис. 5. 64

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)
```

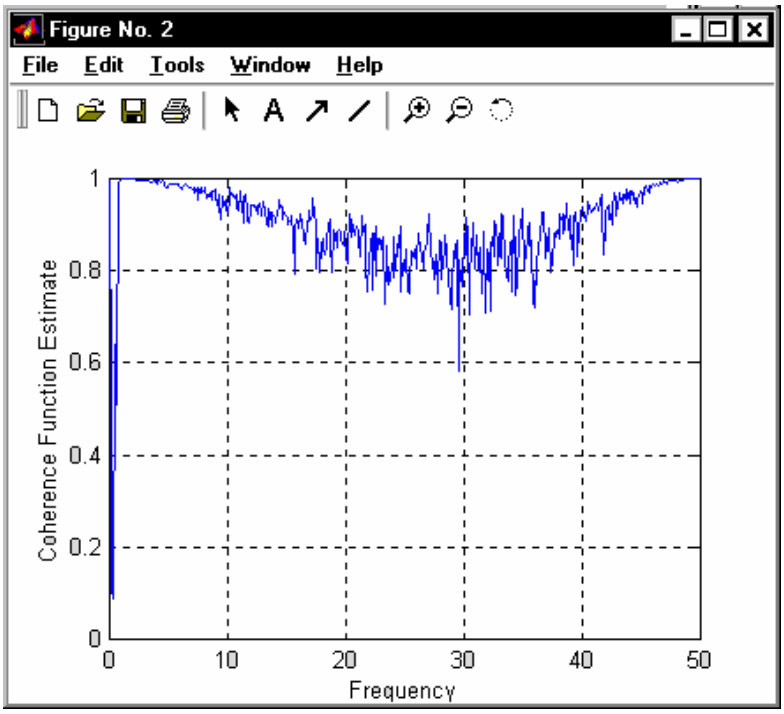

Рис. 5.65

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 представляются три кривые - кривая оценки осредненного значения СПМ на фиксированной частоте и две кривые с добавлением и вычитанием доверительного интервала на этой частоте;
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);
3) следующее нажатие <Enter> приведет (рис. 5.68) к появлению кривой зависимости модуля "передаточной функции" взаимной спектральной

плотности указанных процессов от частоты;

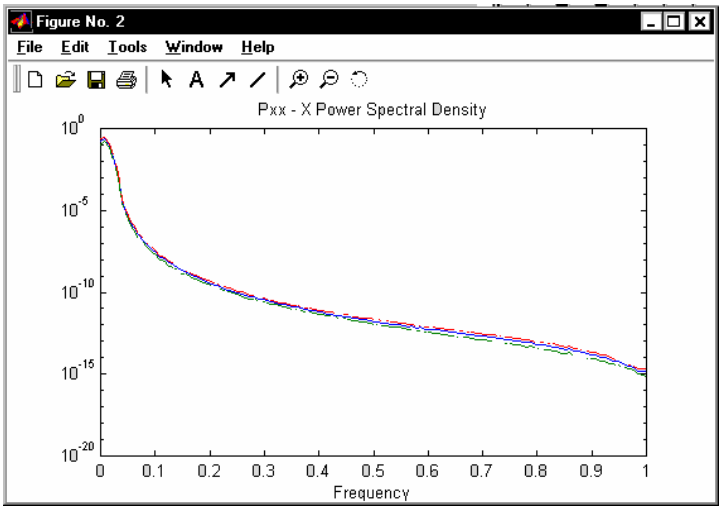

Рис. 5.66

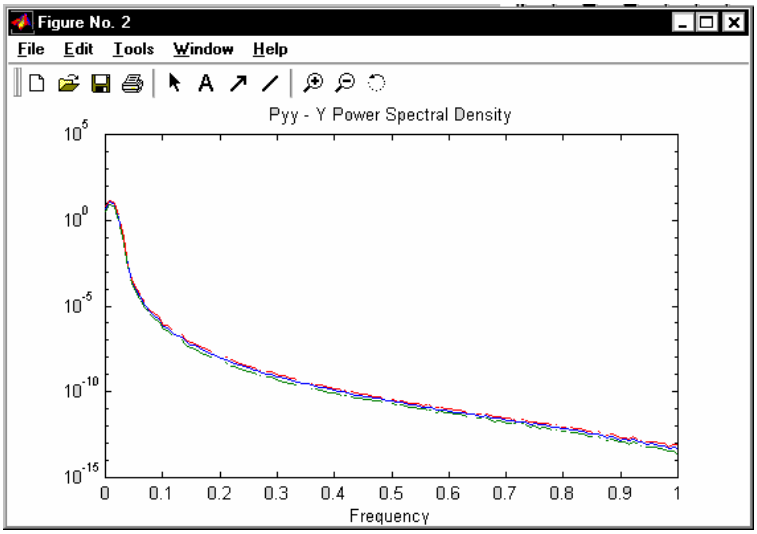

Рис. 5.67

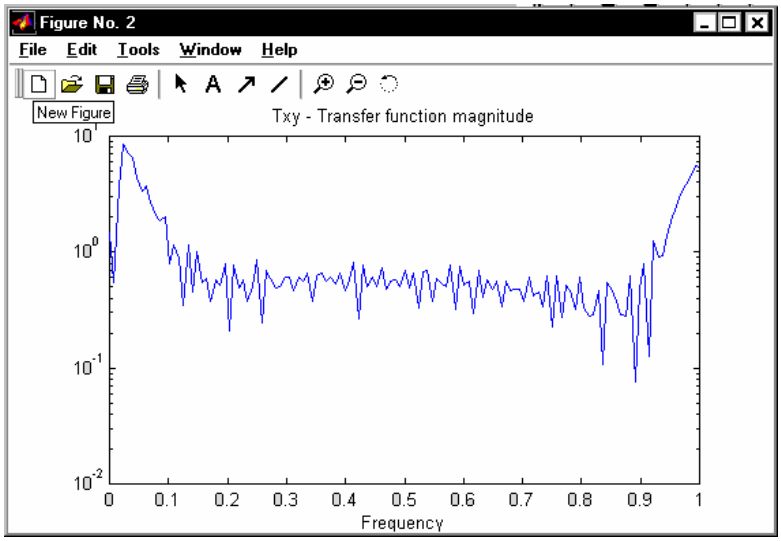

Рис. 5.68

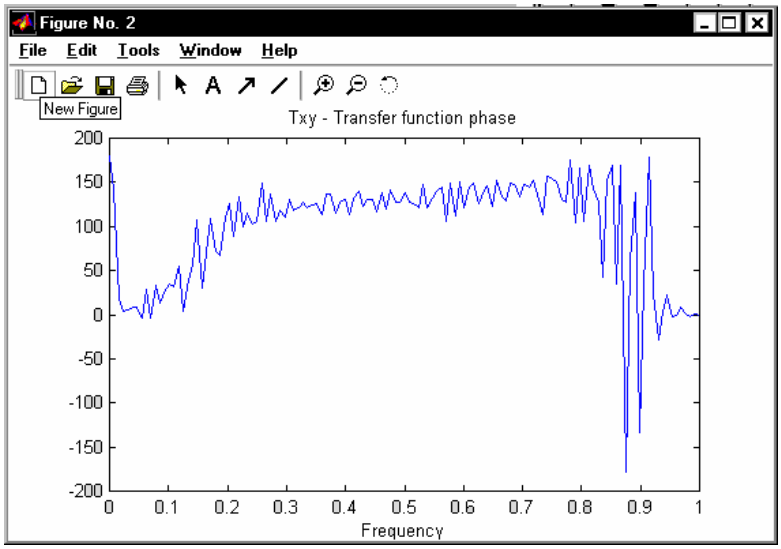

Рис. 5.69

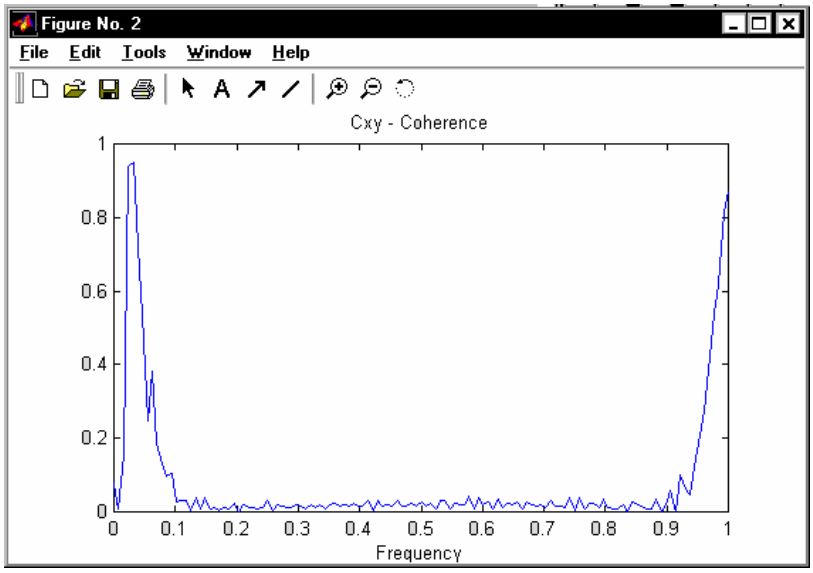

Рис. 5.70

4) дальнейшее нажатие <Enter> приводит к появлению графика зависимости аргумента " передаточной функции" ВСП от частоты (рис. 5.69);

5) последнее нажатие <Enter> вызовет появление в поле графика функции когерентности (рис. 5.70).

Для построения спектрограммы процесса в MatLAB предусмотрена процедура ***specgram***. Спектрограммой называется зависимость амплитуды вычисленного в окне ДПФ (дискретного преобразования Фурье) от момента времени, определяющего положение этого окна. Для примера используем эту процедуру применительно к ранее сформированному процессу X(t):

specgram(x, 10000,100)

В результате получаем в графическом окне картину, изображенную на 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.

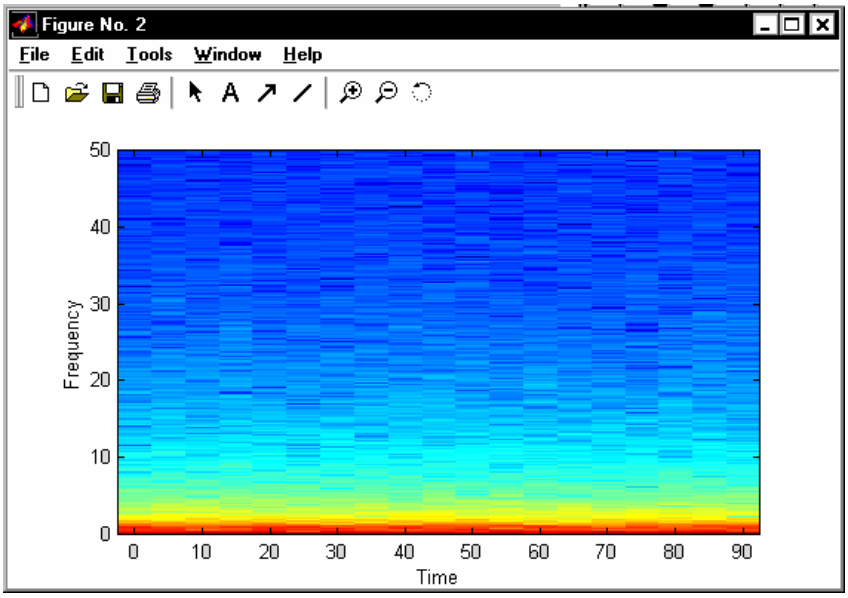

Рис. 5.71

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 обращение к процедуре с целью получить график АЧХ имеет вид:

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.

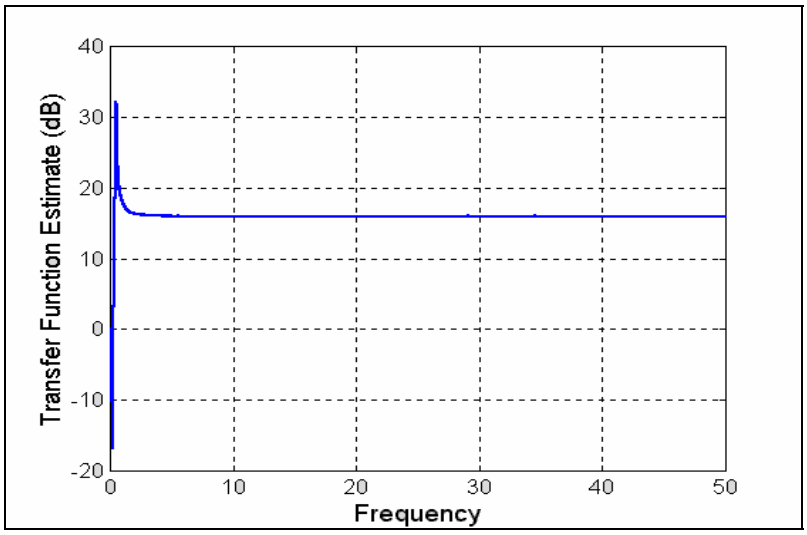

Рис. 5.72

Применяя процедуру к ранее сформированным процессам X(t) и Y(t):

tfe(x, y, 10000,100)

получим график 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 набора в командном окне MatLAB команды

sptool.

В результате на экране появляется окно, представленное на рис. 5.73.

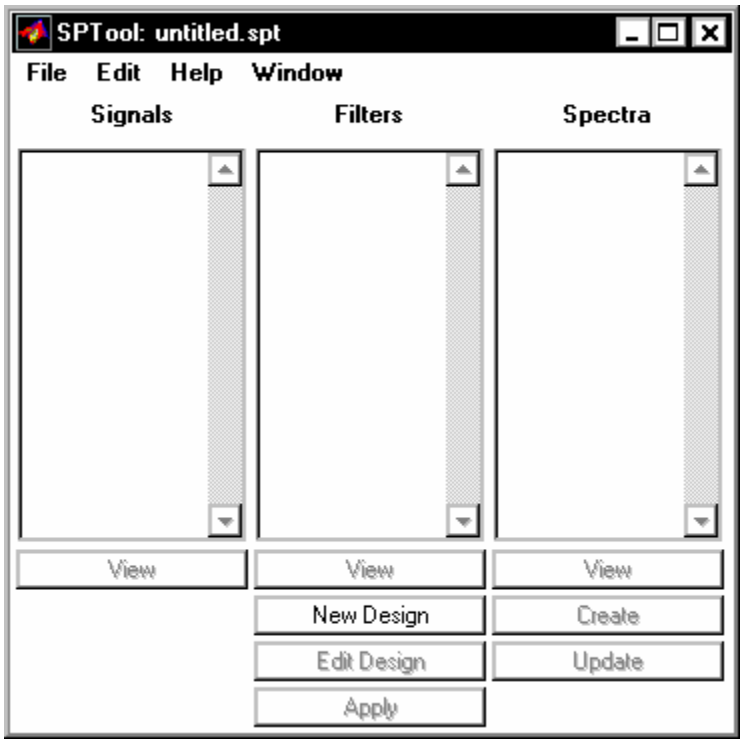

Рис. 5.73

Как видим, окно ***SPTool*** состоит из трех частей-окошек - *Signals (Сигналы), Filters (Фильтры)* и *Spectra (Спектры)*, под каждым из которых имеются надписи-команды, говорящие о том, что можно сделать с объектами, расположенными над ними.

Так, под окошком *Signals* находится лишь надпись *View*. Это означает, что объекты (сигналы), имена которых расположены в этом окошке, могут быть только просмотрены. Под окошком *Filters* находятся четыре надписи, которые означают, что объекты (фильтры), имена которых размещаются внутри него, могут быть:

- созданы (надпись *New Design - Спроектировать Новый*);
- отредактированы (надпись *Edit Design - Отредактировать Проект*);
- просмотрены (надпись *View*);
- применены к одному или нескольким объектам, выделенным в окошке *Signals* (надпись *Apply - Применить*).

Аналогично, с объектами окошка *Spectra* - (спектрами) можно произво-дить такие действия:

- создавать (команда *Create - Создать*);
- просматривать (команда *View*);
- обновить (создать заново под тем же именем) - команда *Update*.

Внутри окошек обычно размещаются имена (идентификаторы) соответствующих переменных или процедур, входящих в открытый в ***sptool*** файл с расширением .SPT (имя этого файла находится в заголовке окна ***SPTool***).

При первом обращении в заголовке окна находится имя *untitled.spt*, все три окошка - пустые, а из команд, расположенных ниже их, активной является только одна *New Design*. Таким образом непосредственно после вхождения в оболочку ***sptool*** непосредственно исполнимой является только операция разработки нового фильтра. Чтобы активизировать остальные команды, необходимо откуда-то импортировать данные о каком-то (или каких-то) сигналах. Такие данные должны быть сформированы другими средствами, нежели сама оболочка ***sptool,*** (например, являться результатом выполнения какой-то 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

Для того, чтобы обрабатывать какие-либо сигналы с помощью ***sptool***, необходимо, прежде всего, 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 разделе 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 (рис. 5.73), выберем "мышкой" раздел *File* меню окна ***sptool***. В результате появляется дополнительное подменю (рис. 5.74), одним из разделов которого является *Import*. "Нажатие" на него "мышкой" приведет к появлению нового окна ***Import to SPTool***, представленного на рис. 5.75.

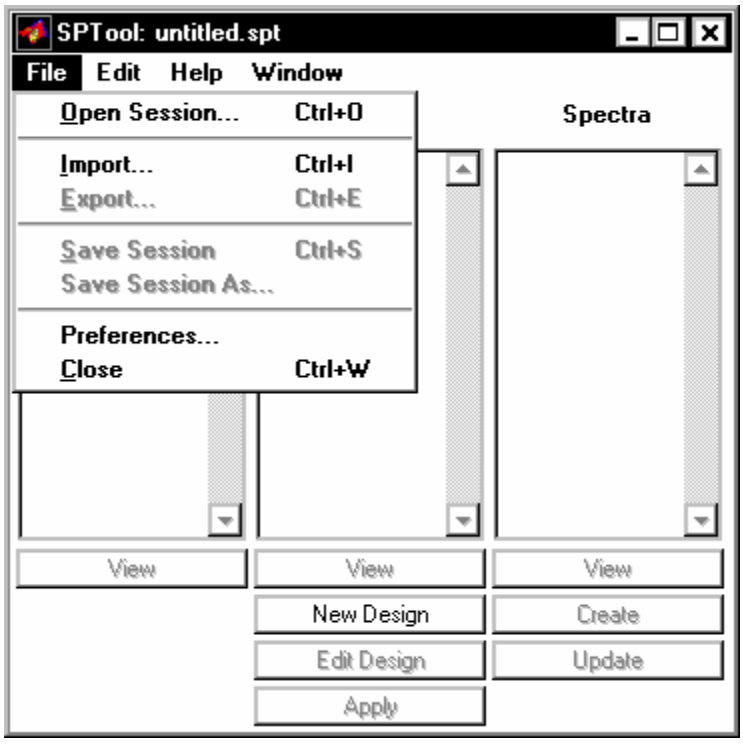

Рис. 5.74

В разделе *Source (Источник)* этого окна отмечен точкой внутри кружка раздел *From Workspace (Из Рабочего Пространства)*. Поэтому все имена переменных рабочего пространства представлены во втором окошке *Workspace Contents (Содержимое Рабочего Пространства)*. Выбрав при помощи "мышки" необходимую переменную, необходимо затем "нажать" кнопку со стрелкой, указывающей на окошко с надписью *Data*. После этого в окошке *Data* должно появиться имя выбранной переменной.

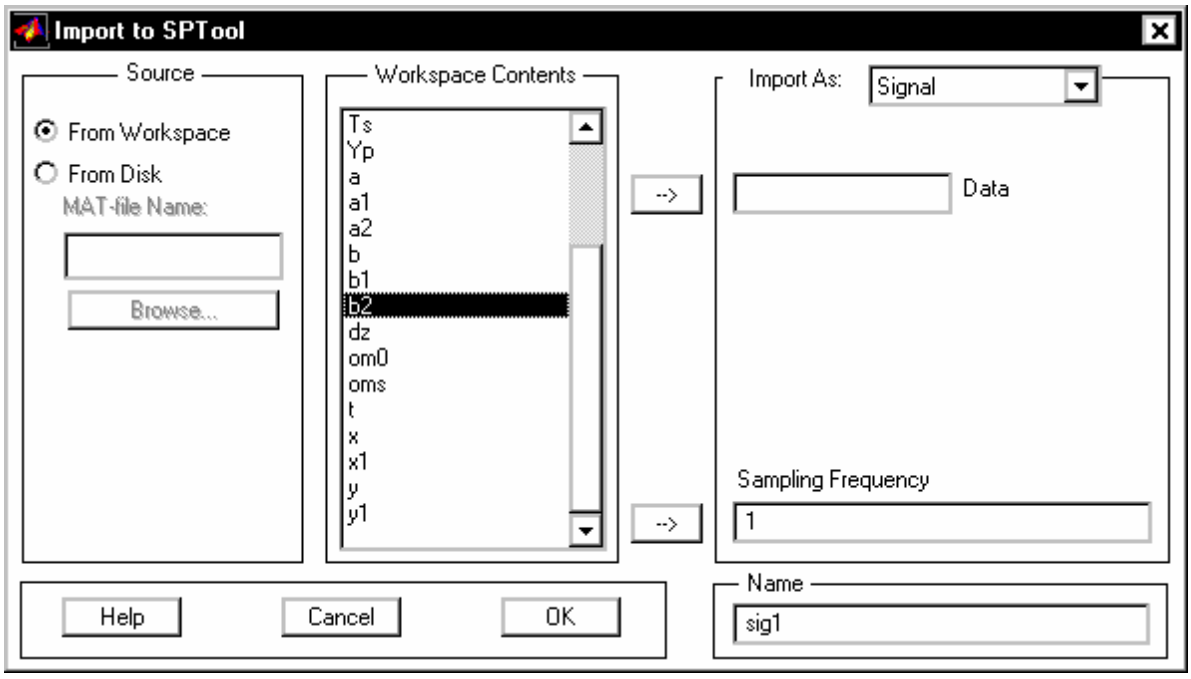

Рис. 5.75

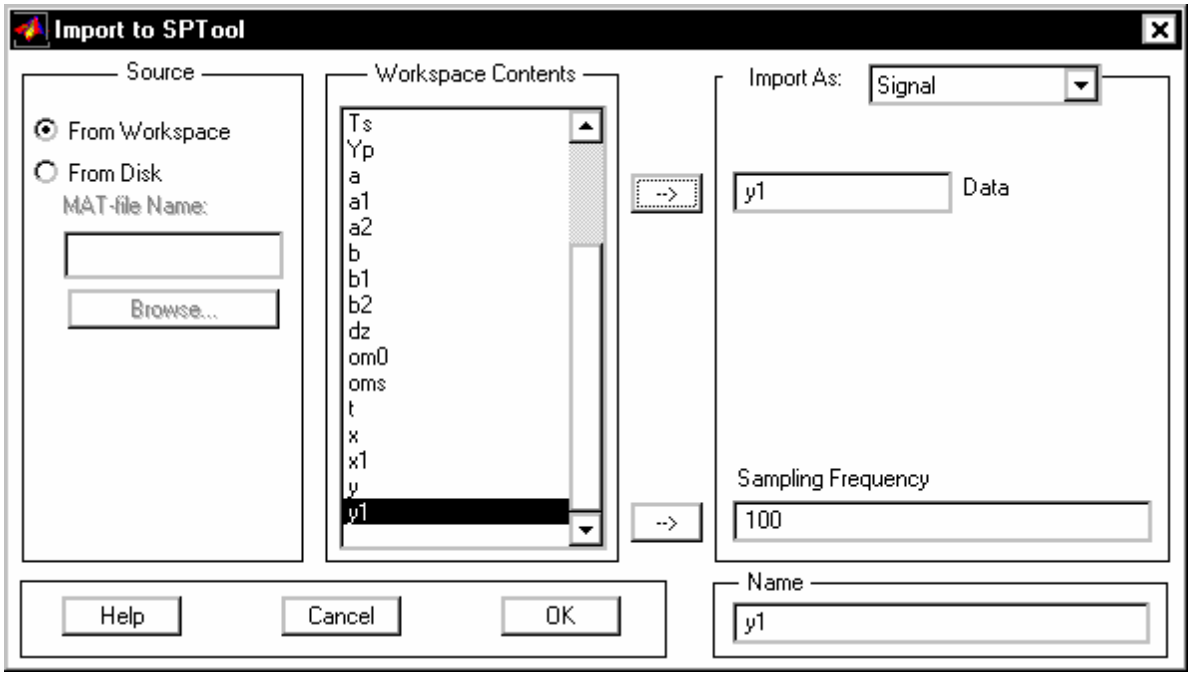

Рис. 5.76

Затем в окошке *Sampling Frequency (Частота Дискретизации)* следует записать желаемое значение частоты дискретизации. Фактически этим параметром задается временной промежуток ***Ts*** между отдельными значениями выбранного вектора процесса.

В окошке *Name (Имя)* следует вписать то имя, под которым введенный вектор будет записан в среде ***sptool.***

На рис. 5.76 виден результат выбора переменной ***y1***, которая будет записана в ***sptool*** под тем же именем с частотой дискретизации 100 Гц (т.е. с дискретом по времени в 0.01 с)

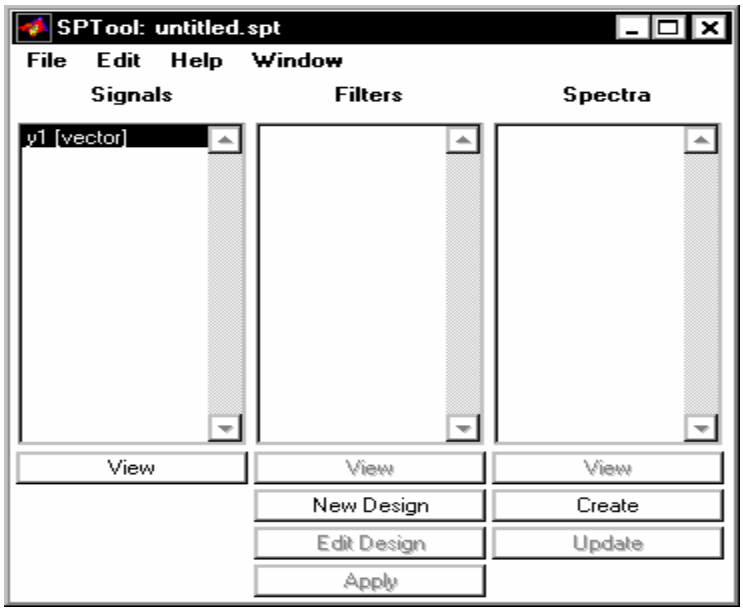

Рис. 5. 77

После такой подготовительной работы следует нажать мышью на кнопку <OK> внизу окна, и импорт сигнала в среду ***sptool*** будет произведен. Окно *IMPORT Sptool* исчезнет и в окно ***sptool*** изменит свой вид (рис. 5.77): в окошке *Signals* появится запись имени вектора сигнала и активизируется (станет более яркой) подпись *View* под этим окошком. Кроме того, станет активной и команда *Create* под окошком *Спектры*. Это означает, что можно находить спектральные характеристики импортированного сигнала.

Повторяя операцию, можно перенести в ***sptool*** и другие сигналы (***x*** и ***y***).

Если векторы процессов записаны в MAT-файл, то для их импорта необходимо, после вызова окна *IMPORT Sptool,* активизировать в нем "радиокнопку" с надписью *From disk*, нажав "мышкой" внутри кружка этой радиокнопки. В результате будут активизированы два нижерасположенных раздела - окошко *MAT-file Name* и *Browse* (рис 5.74) . Записывая в первое имя необходимого MAT-файла с записью процесса или отыскивая MAT-файл при помощи *Browse,* вновь вызываем в окошко *File Contents* его содержимое. Последующие действия аналогичны ранее рассмотренным.

Просмотр сигналов

После импорта вектора сигнала можно воспользоваться средствами его просмотра. Для этого достаточно выделить в окошке *Signals* нужные сигналы и "нажать мышкой" на надпись *View* под окошком. В результате должно появиться новое окно *Signal Browser*.

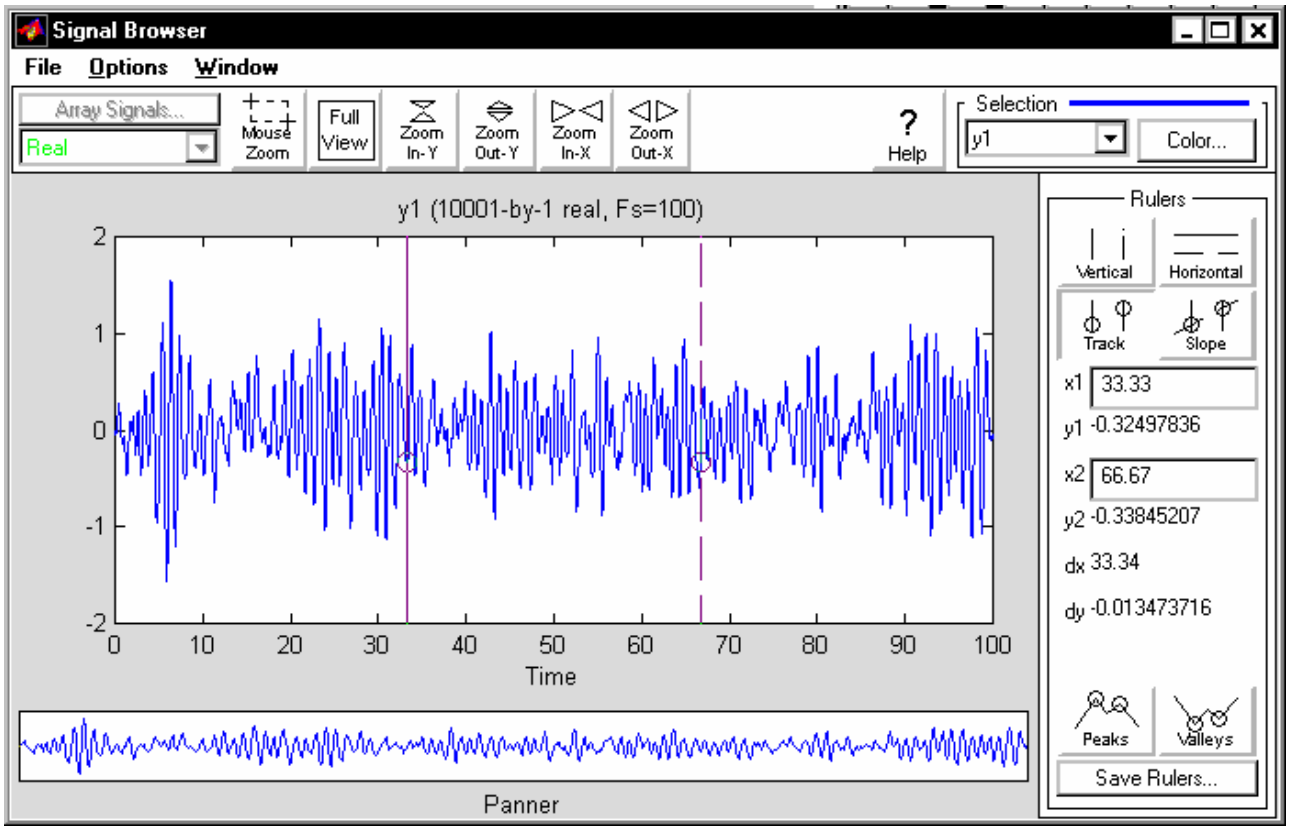

Рис. 5.78

В нашем случае, выбирая сигнал ***y1***, получим окно, изображенное на рис.

5.78. Как видим, центральную часть окна занимает изображение кривых зависимости выделенных процессов от времени. В заголовке графика указываются имена сигналов, изображенных на графике, размерность соответствующих векторов и частота дискретизации.

В верхнем правом углу окна расположена область *Selection*, с помощью средств которой можно изменить цвета кривых, изображаемых в окне графиков.

Справа от графического поля окна (область *Rulers*) помещены инструменты, обеспечивающие точный отсчет показаний значений аргумента и процесса в двух точках графика.

Эти точки графика определяются пересечением с графиком процесса, имя которого стоит в области *Selection*, с двумя вертикальными линиями розового цвета, расположенными в поле графиков. Изменение положения этих линий по шкале времени происходит с помощью "мышки". Если курсор "мышки" подвести к одной из этих вертикальных линий, курсор примет вид кисти руки. Нажав в этот момент на левую клавишу "мышки" и, не отпуская ее, переместив курсор вправо или влево, можно изменить положение соответствующей линии на графике. При этом в области *Rulers* будут непрерывно указываться координаты 'x' и 'y' точек пересечения обеих вертикальных прямых с графиком процесса, а также разности между ними. При этом координата 'x' соответствует времени, а 'y' - значению процесса.

На рис. 5.79 отображены все три процесса. Если нажать "мышкой" кнопку Slope, в поле графика появится еще одна прямая, соединяющая указанные ранее точки пересечения выбранной кривой с вертикальными линиями, а в области *Rulers* появится новое число 'm', значением которого является тангенс угла наклона этой прямой к оси времени.

В верхней части окна располагаются средства управления окном и масштабами внутри графического поля.

Создание спектров сигналов

После введения в ***sptool*** сигналов можно найти оценки спектральных свойств этих сигналов. Для этого достаточно в окне ***SPTool*** в окошке сигналов отметить (выделить) тот сигнал, оценку спектральной плотности которого вы хотите получить, и "нажать мышкой" на команду *Create* в нижней части окна. При этом на экране появится новое окно - ***Spectrum Viewer*** *(Обозреватель спектра)* - рис. 5.80.

Новое окно напоминает окно *Signal Browser*. Верхняя и правая части этих окон практически совпадают. Однако графическое окошко в окне *Spectrum Viewer* является пустым, а слева от него располагается область, инструменты которой позволяют:

- выбрать метод нахождения спектральной характеристики сигнала;
- установить количество обрабатываемых точек сигнала;
- установить количество точек сглаживающего окна;

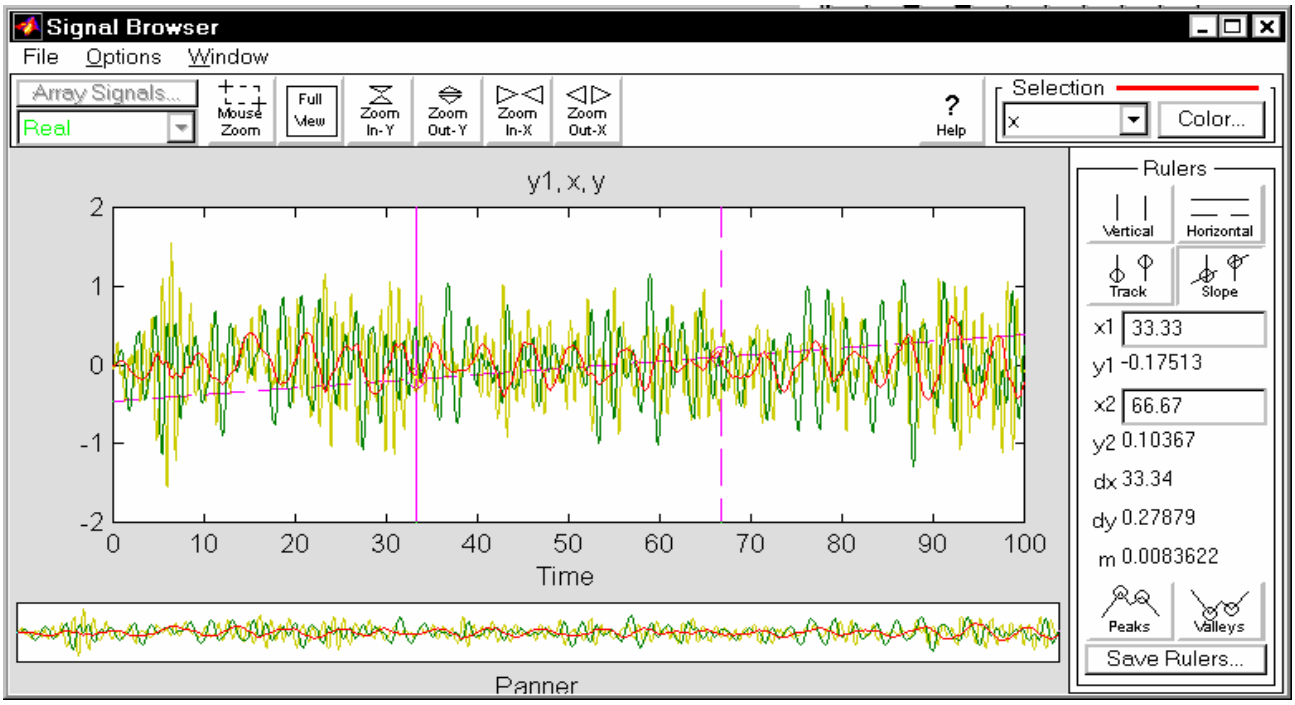

Рис. 5.79

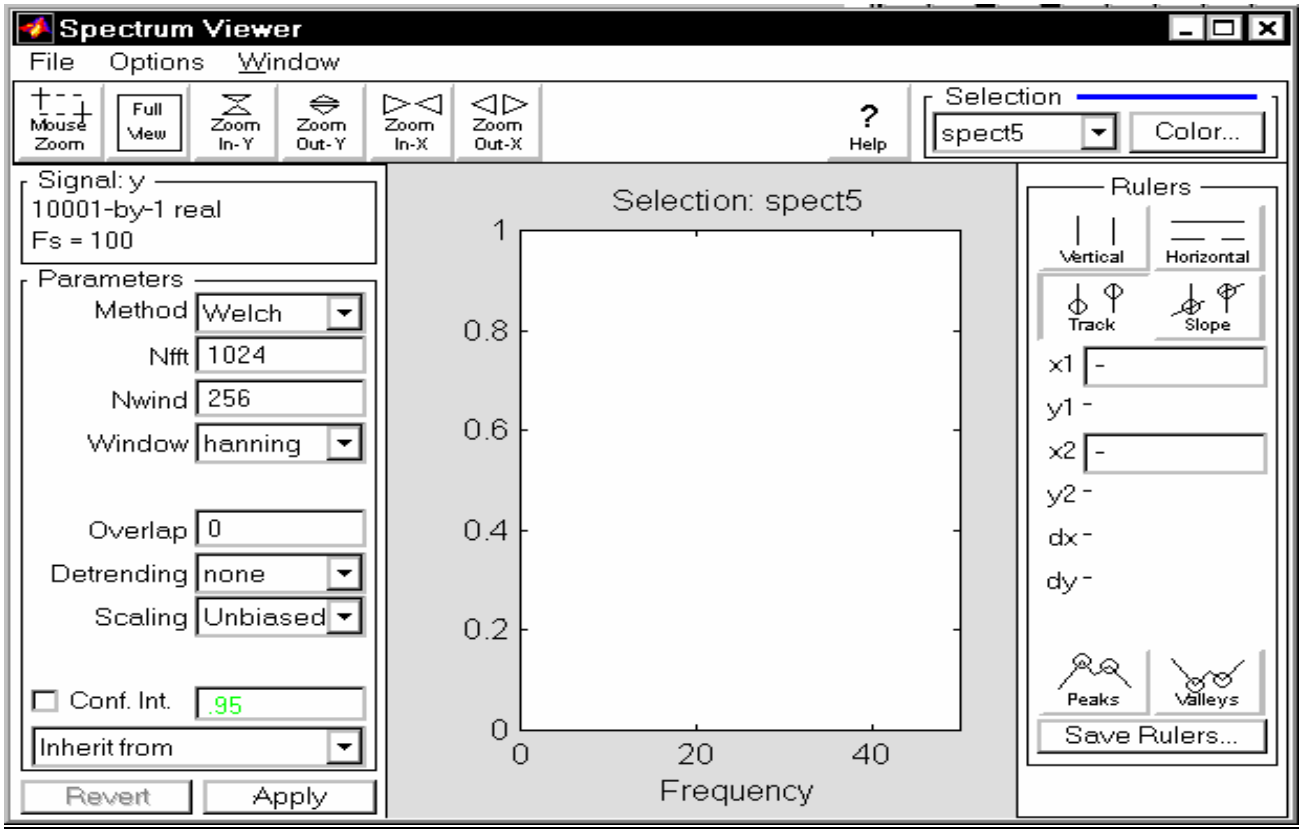

Рис. 5.80

- выбрать тип окна сглаживания;
- установить размер перекрытия окон;
- установить метод исключения тренда;
- установить метод масштабирования графика.

Метод вычисления спектра выбирается при помощи выпадающего меню в

окошке под названием *Method*. Выпадающее меню содержит такие альтернативы (см. рис. 5.81):

- Burg;
- FFT;
- MEM;
- MTM;
- MUSIC;
- Welch;
- YuleAR.

Каждому из названий соответствует аналогичный метод (процедура) вычисления спектра сигнала.

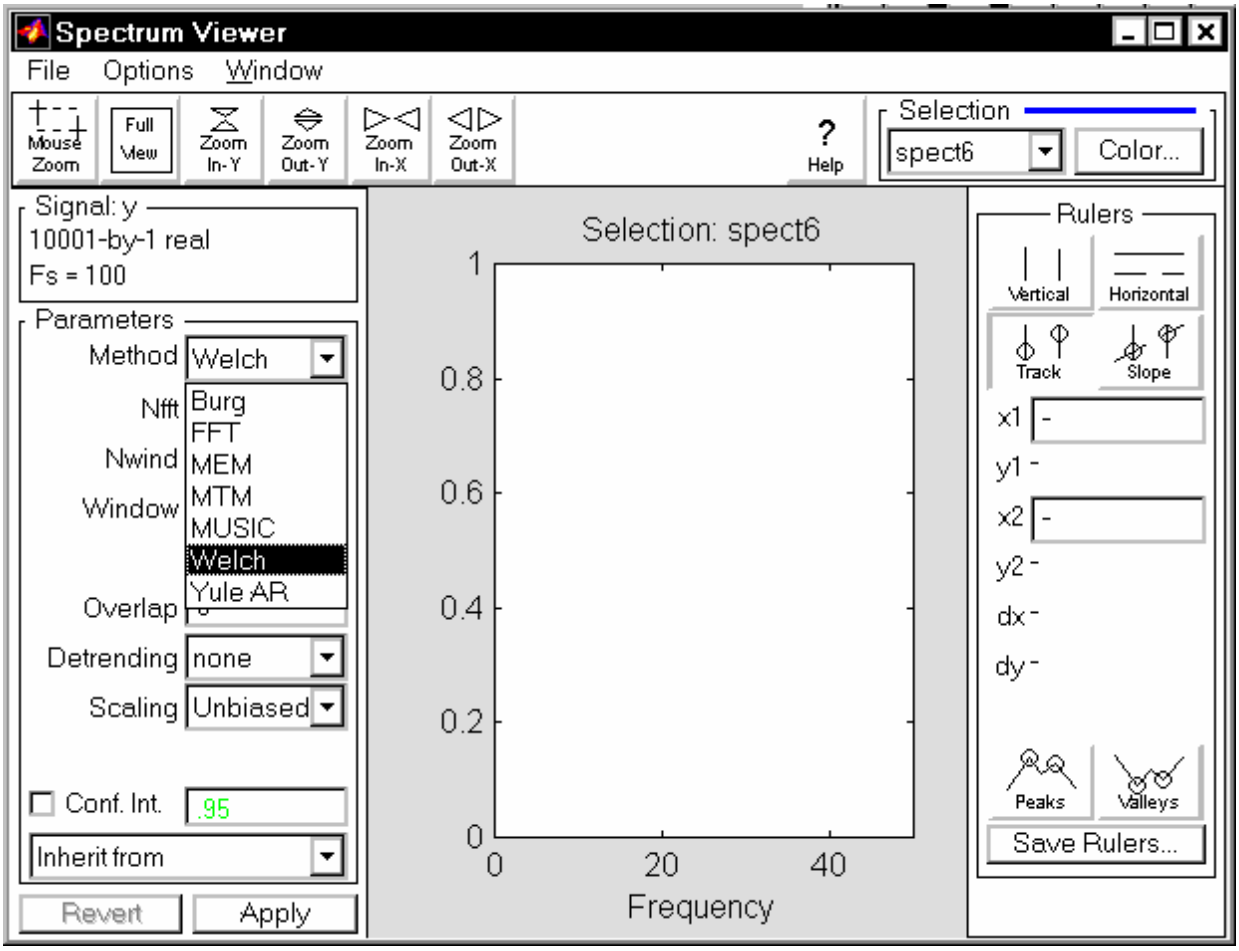

Рис. 5.81

Для проведения вычислений после выбора метода следует нажать кнопку *Apply* внизу левого поля. Например, выделим для обработки процесс Y1, "нажмем" команду *Create* и выберем метод FFT. После нажатия кнопки *Apply* в окне *Spectrum Viewer* появится картина, представленная на рис. 5.82.

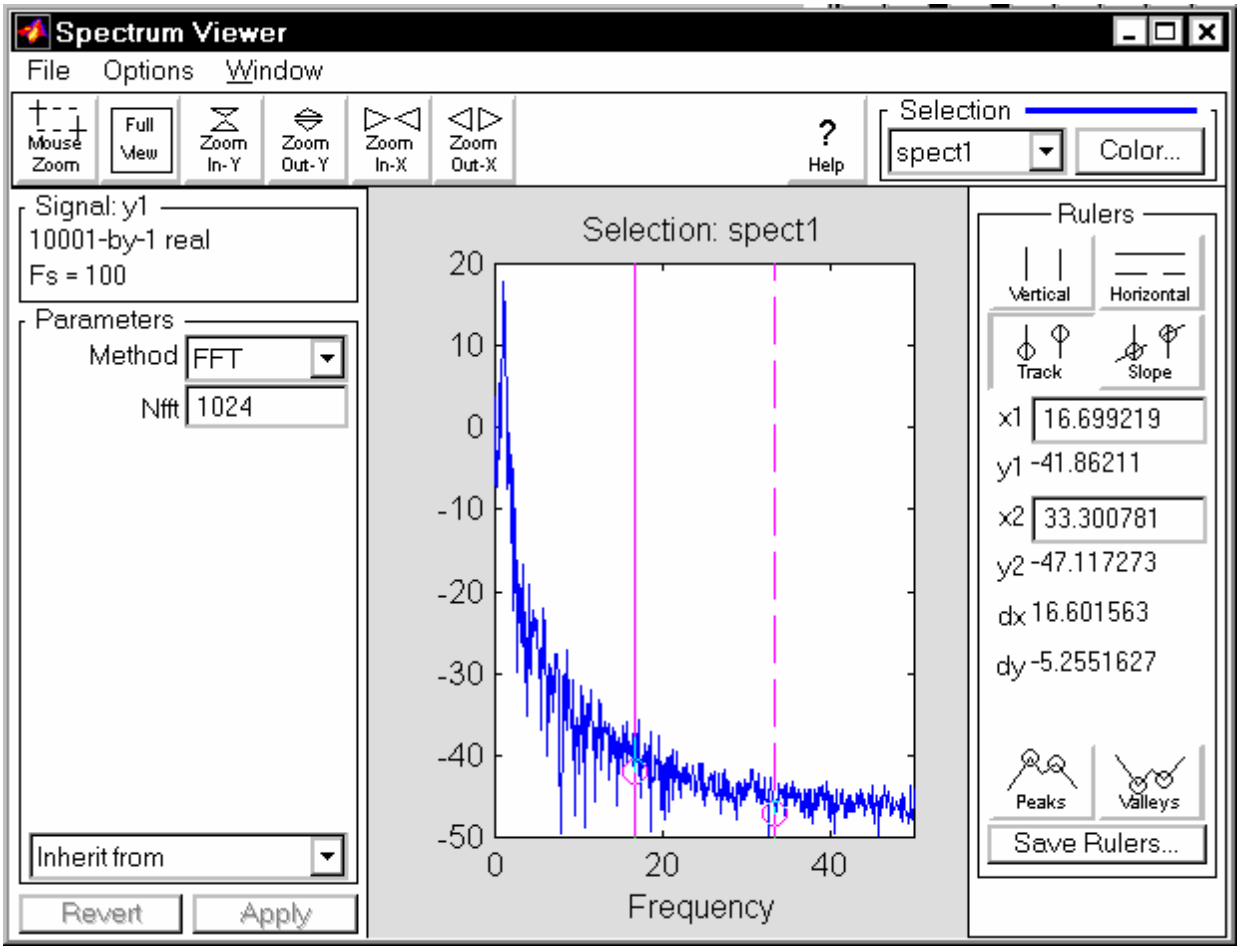

Рис. 5.82

Проектирование фильтра

Если в окне ***SPTool*** выбрать команду *New design*, то на экране возникнет окно ***Filter Designer***, показанное на рис. 5.83.

Новое 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 (окошко *Algorithm*); при этом предоставляются такие альтернативы:
  - *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 (окошко *Type*); предоставляется возможность выбора следующих типов:
  - *Lowpass* - фильтр нижних частот;
  - *Highpass* - фильтр верхних частот;
  - *Bandpass* - полосовой фильтр;

- *Bandstop* - режекторный фильтр.

- параметры полосы пропускания (раздел *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 (раздел *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, устанавливая алгоритм фильтра Баттерворта нижних частот с граничными частотами полос пропускания в 0.5 Гц и задерживания в 0.7 Гц (см. рис. 5.84) и нажимая кнопку *Apply*, получим параметры такого фильтра и запишем их в объект 'filt4' .

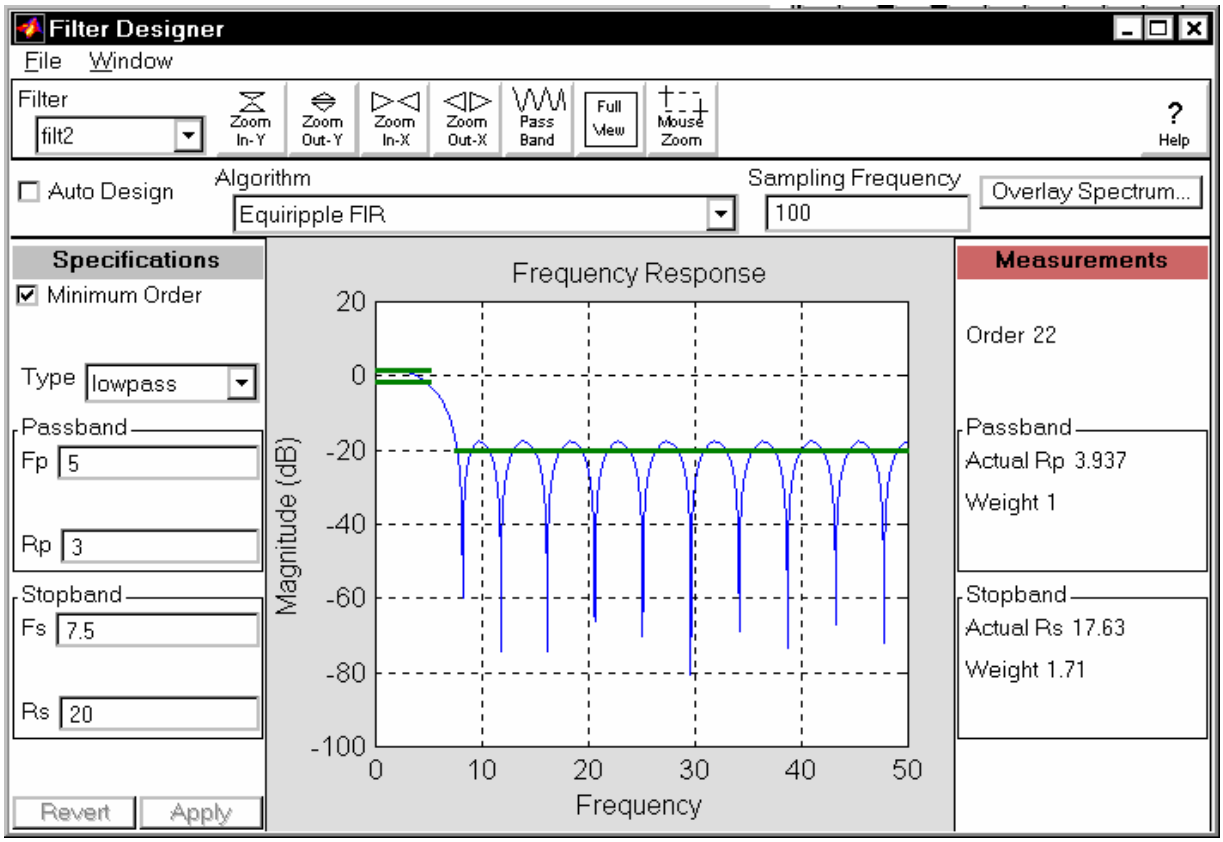

Рис. 5.83

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 окошке *Filters* окна ***SPTool***, а затем нажать кнопку *View* под этим окошком.

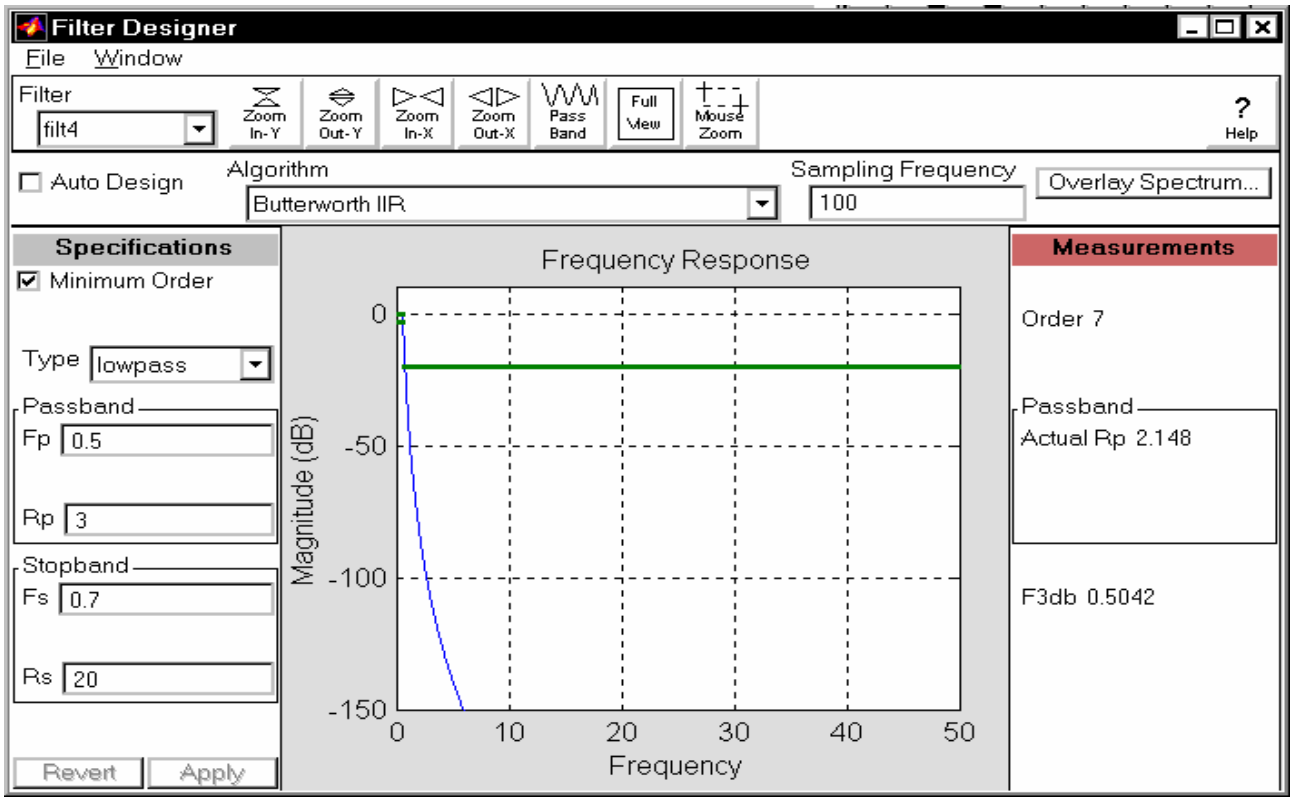

Рис. 5.84

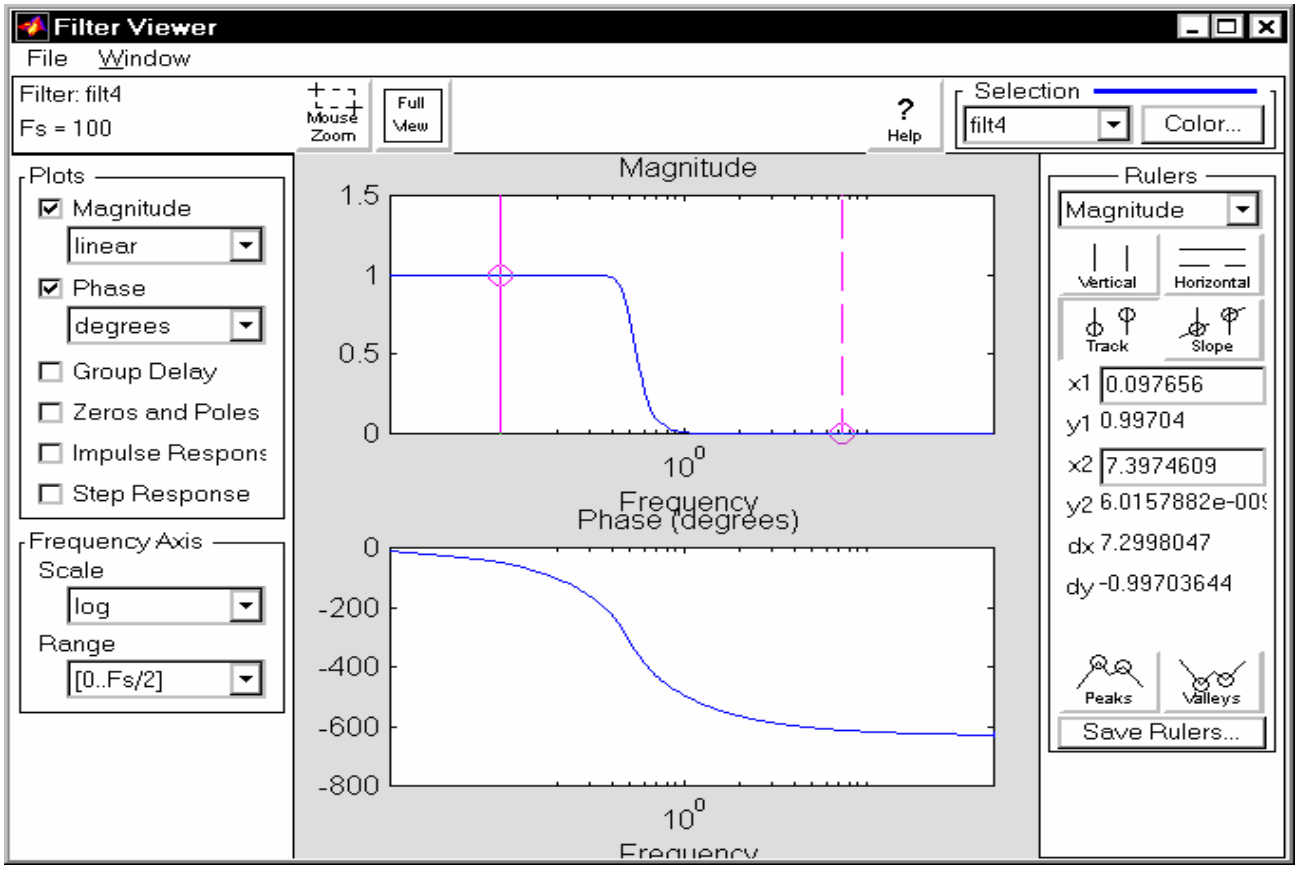

Рис. 5.85

Например, для только что созданного фильтра 'filt4' мы получим в результате на экране новое окно ***Filter Viewer*** с картиной, показанной на рис. 5.85.

Как видим, в окне выведены графики АЧХ и ФЧХ фильтра.

В число средств просмотра фильтров входят (см. левую сторону окна *Filter Viewer*):

- возможность вывода на экран одновременно любого сочетания из 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 "мышки" 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 *FrequencyAxis*).

Пример вывода всех доступных графиков одновременно показан на рис. 5. 86.

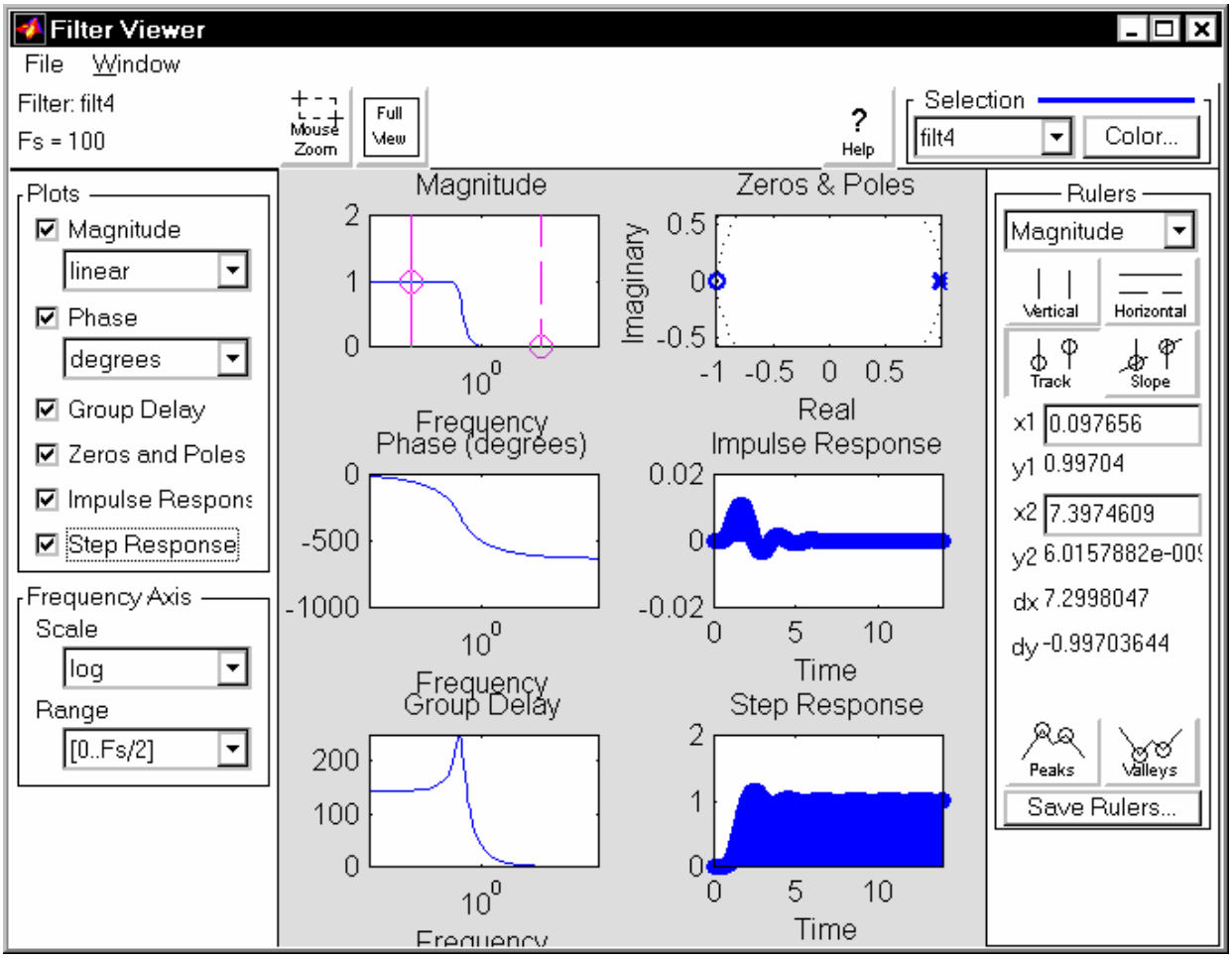

Рис. 5.86

Применение разработанного фильтра для фильтрации

Использование в среде ***sptool*** разработанного фильтра чрезвычайно просто. Для этого нужно в окне ***SPTool*** в окошке *Signals* выделить имя сигнала, который

нужно преобразовать с помощью фильтра, в окошке *Filters* - имя фильтра, с помощью которого надо преобразовать этот сигнал и нажать команду *Apply*. в результате в первом окошке (*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 раздел *Spectra*, как это было описано.

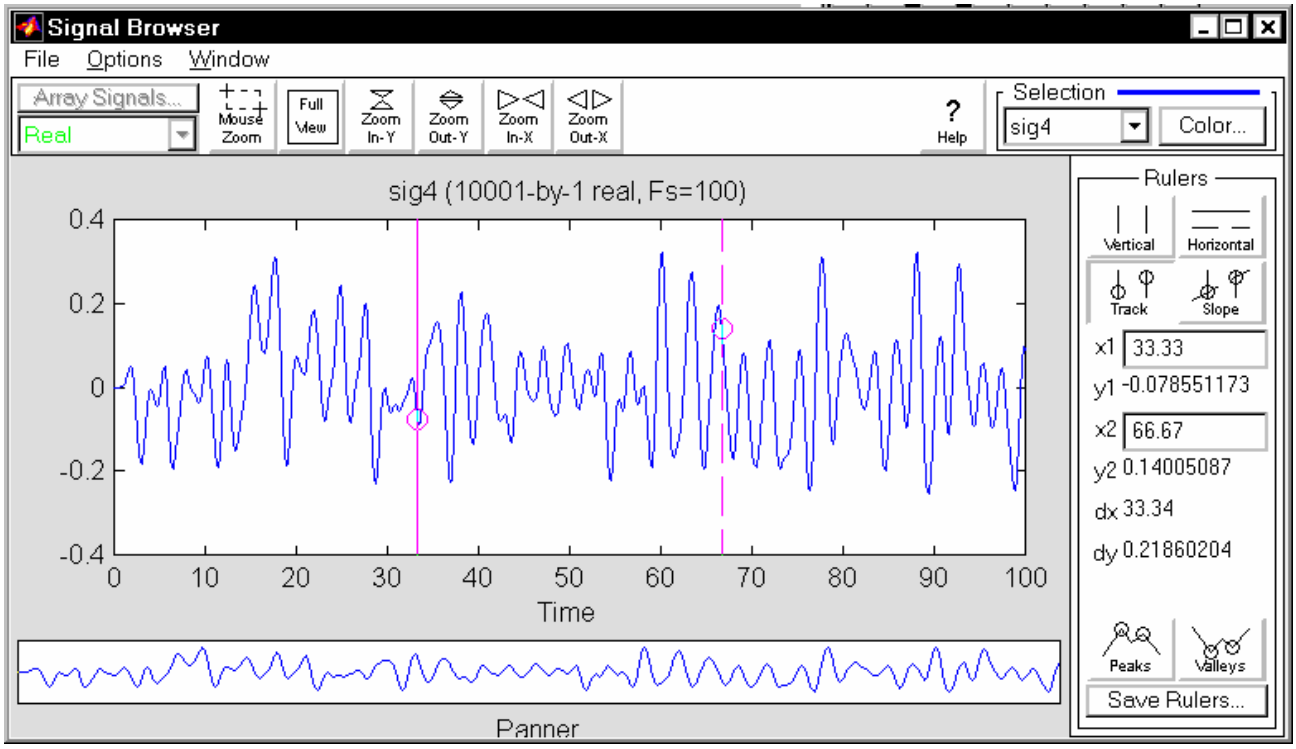

Рис. 5.87

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 разделе *File* меню окна ***SPTool*** предусмотрены команды записи в файл (см. рис. 5.74) *Save Session* и *Save Session As*.

При повторном запуске ***sptool*** можно воспользоваться результатами такого сохранения результатов, используя команду *Open Session* и выбирая один из записанных SPT-файлов.

6. Исследование линейных стационарных систем (пакет CONTROL Toolbox)

Первое представление о пакете CONTROL можно получить, изучая раздел 4.2.2. этого пособия. Ниже приведен сжатый перечень основных процедур пакета CONTROL, сгруппированных по общности функционального назначения.

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 - о типе модели ('ss', 'zpk' или 'tf').
size - 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 из непрерывного времени в дискретное
d2c - Преобразование из дискретного времени в непрерывное
d2d - Переопределение дискретной системы или добавление задержек входных воздействий

" Арифметические" операции

+ и - - Добавление и отнимание 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 - К-нт 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 состо-яния.
ss2ss - Преобразование переменных состояния
canon - Каноническая форма пространства состояния
ctrb, obsv - Матрицы управляемости и наблюдаемости
gram - Определители Грамма управляемости и наблюдаемости
ssbal - Диагональная балансировка матриц пространства состояния
balreal - Балансировка входа-выхода на основе определителя Грамма
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 +).
series - Обобщенное последовательное соединение (см. также *)
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 - ОМ и ММ LQG управление прокатного стана
kalmdemo - Проектирование и моделирование фильтра Калмана

Далее процедуры пакета изучаются более подробно.

6.1. Ввод и преобразование моделей

LTI- модели можно создавать в трех видах - ***SS***, ***TF*** и ***ZPK***-объектов. 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:

$$\begin{cases} \dfrac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u \end{cases} \tag{2}$$

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:

$$x_1 = \alpha; \quad x_2 = \beta; \quad x_3 = \dot{\alpha}; \quad x_4 = \dot{\beta}. \tag{3}$$

Тогда уравнения гироскопа в форме Коши приобретут вид:

$$\begin{cases} \dot{x}_1 = x_3; \\ \dot{x}_2 = x_4; \\ \dot{x}_3 = -\lambda \cdot x_4 + n(t); \\ \dot{x}_4 = \lambda \cdot x_3 + l(t); \end{cases} \tag{4}$$

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:

$$y = \left[\alpha, \beta\right]^T; \tag{5}$$

в качестве входного - вектор моментов сил:

$$u = \left[n(t), l(t)\right]^T \ . \tag{6}$$

Полагая

$$x = \left[x_1, x_2, x_3, x_4\right]^T, \tag{7}$$

значения указанных матриц должны быть такими:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\lambda \\ 0 & 0 & \lambda & 0 \end{bmatrix};\ B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix};\quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix};\ D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{8}$$

Введем эти матрицы в командном окне MatLAB, принимая $\lambda = 10$:

```
>> 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 по имени 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)
```

Ввиду того, что 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 первого порядка, не разрешенных относительно производных, т.е. когда система описывается совокупностью уравнений вида:

$$\begin{cases} E \cdot \dfrac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u, \end{cases} \qquad (9)$$

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

$$\begin{cases} J_1 \cdot \ddot{\alpha} + H \cdot \dot{\beta} = N(t) \\ J_2 \cdot \ddot{\beta} - H \cdot \dot{\alpha} = L(t). \end{cases} \qquad (10)$$

Вводя те же переменные (см. (3), (5)...(7)), получим систему уравнений в виде (9), где матрицы A и E будут иметь вид:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -H \\ 0 & 0 & H & 0 \end{bmatrix}; \qquad E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & J_1 & 0 \\ 0 & 0 & 0 & J_2 \end{bmatrix}. \qquad (11)$$

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:

$$W_{11}(s)=\frac{J_2}{J_1\cdot J_2\cdot s^2+H^2};\qquad W_{12}(s)=-\frac{H}{s\cdot(J_1\cdot J_2\cdot s^2+H^2)};$$

$$W_{21}(s)=\frac{H}{s\cdot(J_1\cdot J_2\cdot s^2+H^2)};\ W_{22}(s)=\frac{J_1}{J_1\cdot J_2\cdot s^2+H^2}. \qquad (12)$$

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:

$$V=\left[1,\quad 0\right].$$

Теперь создадим вектор коэффициентов второго знаменателя путем свертки этих двух векторов (это соответствует перемножению полиномов):

$V_{zn2}=$ ***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
```

$\text{den}(1,2)=\{V_{zn2}\};\qquad \text{den}(2,1)=\{V_{zn2}\}.$

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. Ниже приводится пример этого:

```
>> 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
```

Предостережение.

При манипуляциях или преобразованиях LTI-модели следует учитывать, что:

1) 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;

*2) преобразования 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**);*

3) 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, <u>следует по возможности избегать преобразований моделей из одной формы в другую и наоборот.</u>

Итак, 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 (таким, как названия входов и выходов, название системы и т.п.).

Указанные процедуры позволяют создавать как непрерывные модели , 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}. Приведем 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'.

Те же 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 перевести в дискретную* форму, воспользовавшись процедурой ***c2d*** в соответствии со схемой:

sysd = ***c2d***(sys, Ts, method).

Здесь sys - исходная непрерывная заданная модель, sysd - получаемая в результате работы процедуры дискретный аналог исходной системы, Ts - задаваемое значение шага дискретизации, method - параметр, определяющий метод дискретизации. Последний параметр может принимать одно из таких 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.

Ниже приведены примеры перевода введенного ранее непрерывного колебательного звена 'kzv1' в дискретные звенья по разным методам:

```
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*, например:

```
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  ,
```

а затем введем задержку по входу, равную 3 Ts. 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, осуществляющих расчет характеристик соединений отдельных звеньев, относятся:

- ***plus (minus)*** - осуществляет "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 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 поменены местами;
- ***vertcat*** производит так называемую вертикальную конкатенацию (сцепление) систем (звеньев), т. е. такое их объединение, когда входы их 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 каждой из объединяемых систем, а число выходов 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 последовательно соединенных двух 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 дифференциально-колебательное звено

```
GT = tf([100 0],[1 10 100])
Transfer function:
   100 s
----------------
s^2 + 10 s + 100 .
```

Параллельное соединение этих двух контуров управление можно осуществить тоже двумя путями: либо используя процедуру ***parallel***

```
lzm1=parallel(GN,GT)
 Transfer function:
2 s^2 + 120 s + 200
-------------------
 s^2 + 10 s + 100 ,
```

либо применяя операцию "сложения" моделей

```
lzm = 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,lzm)
Transfer function:
        25 s^2 + 250 s + 2500
-----------------------------------------------
100 s^4 + 1050 s^3 + 10550 s^2 + 8000 s + 5000 .
```

Конечно, несравненно *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 символьные ее описания. В частности, дать названия входам и выходам системы, дать краткий комментарий к самой системе, например:

```
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), например:

```
[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 временит.

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.

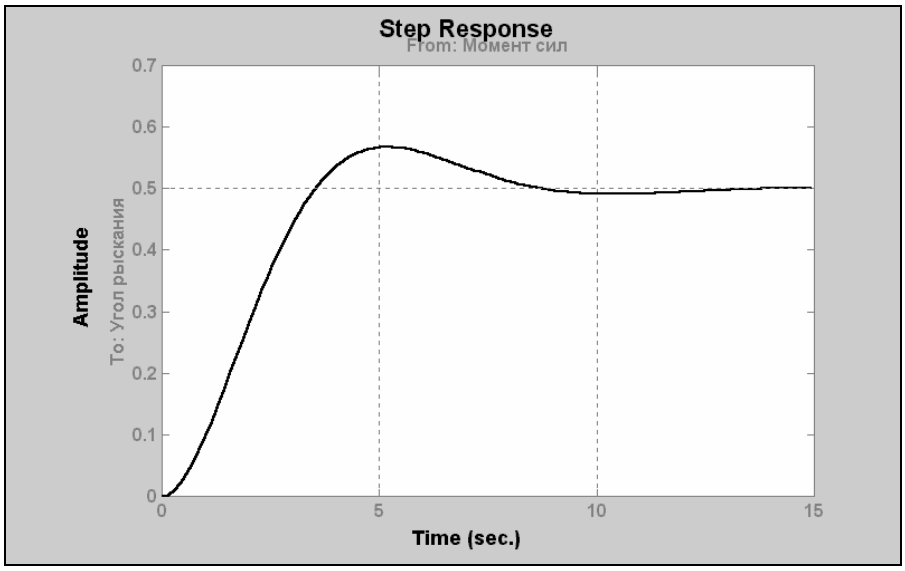

Рис. 6.1

Аналогично, использование процедуры

impulse(sys)

приведет к появлению в графическом окне графика 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. Например:

initial(ssys,[0 0 0 1],20).

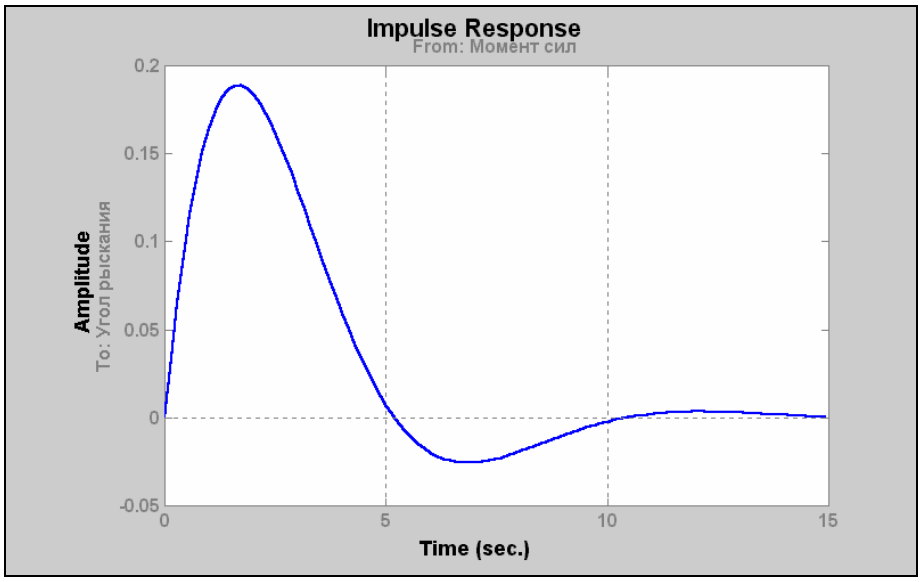

Рис. 6.2

Получим в графическом окне картину, показанную на рис. 6.3.

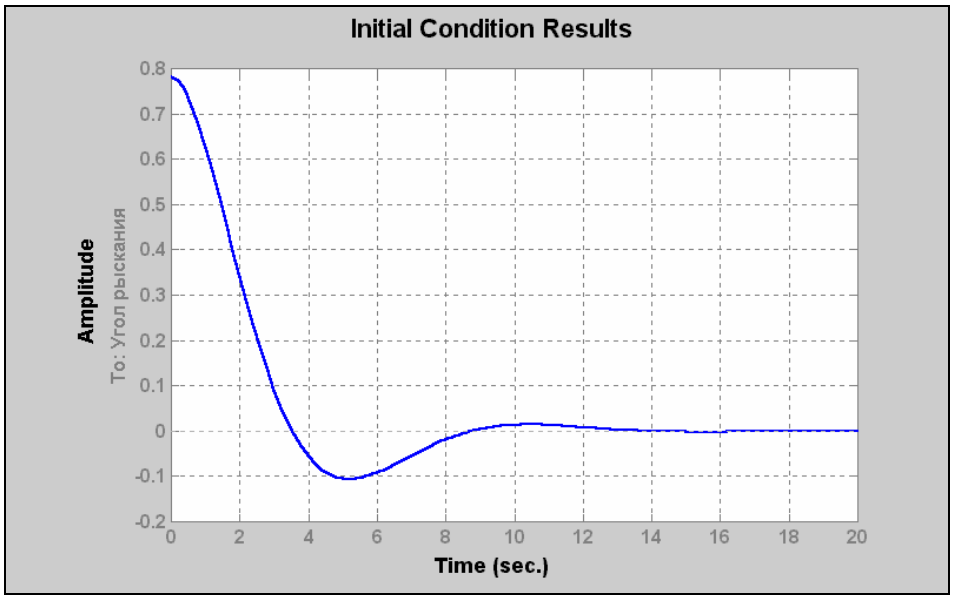

Рис. 6.3

Для применения процедуры ***lsim*** необходимо предварительно задать вектор 't' значений времени, в которых будут заданы значения входного воздействия, а затем и 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.

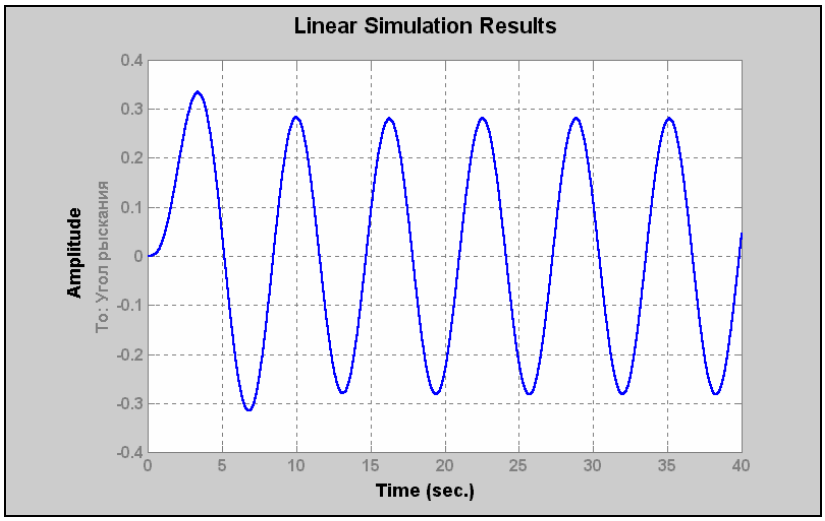

Рис. 6.4

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*.

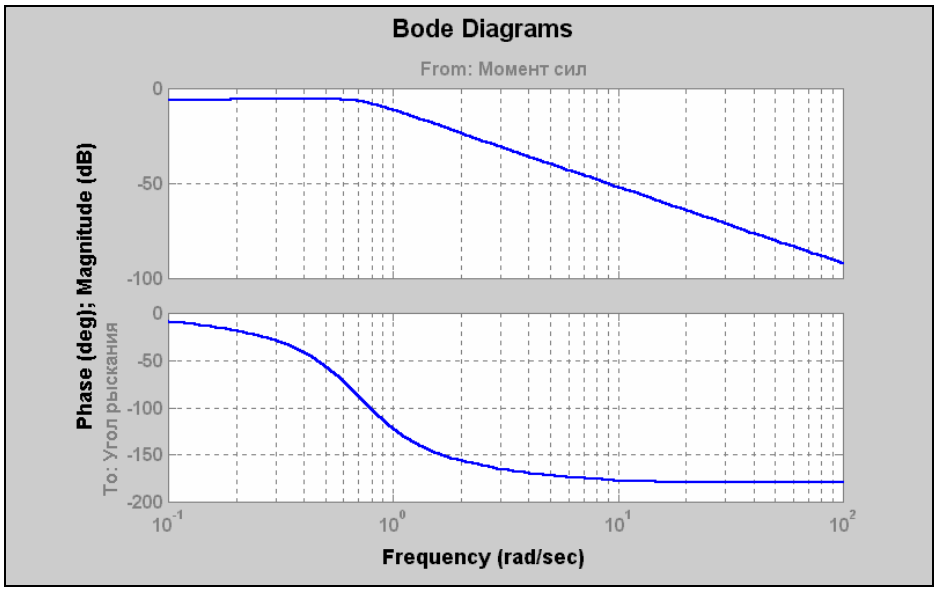

Рис. 6.5

Приведем примеры.

bode(sys) - результат приведен на рис. 6.5;

nyquist(sys); grid - результат - на рис. 6.6;
nichols(sys); grid - результат показан на рис. 6.7;
sigma(sys) - см. рис. 6.8;
margin(ssys); grid - см. рис. 6.9.

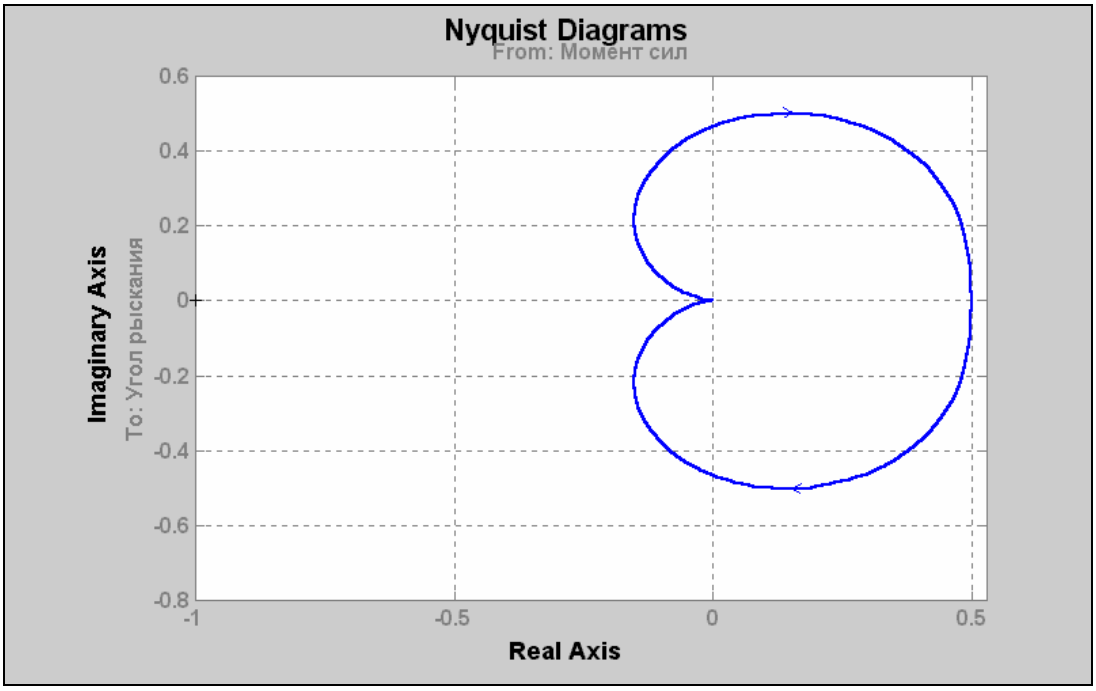

Рис. 6.6

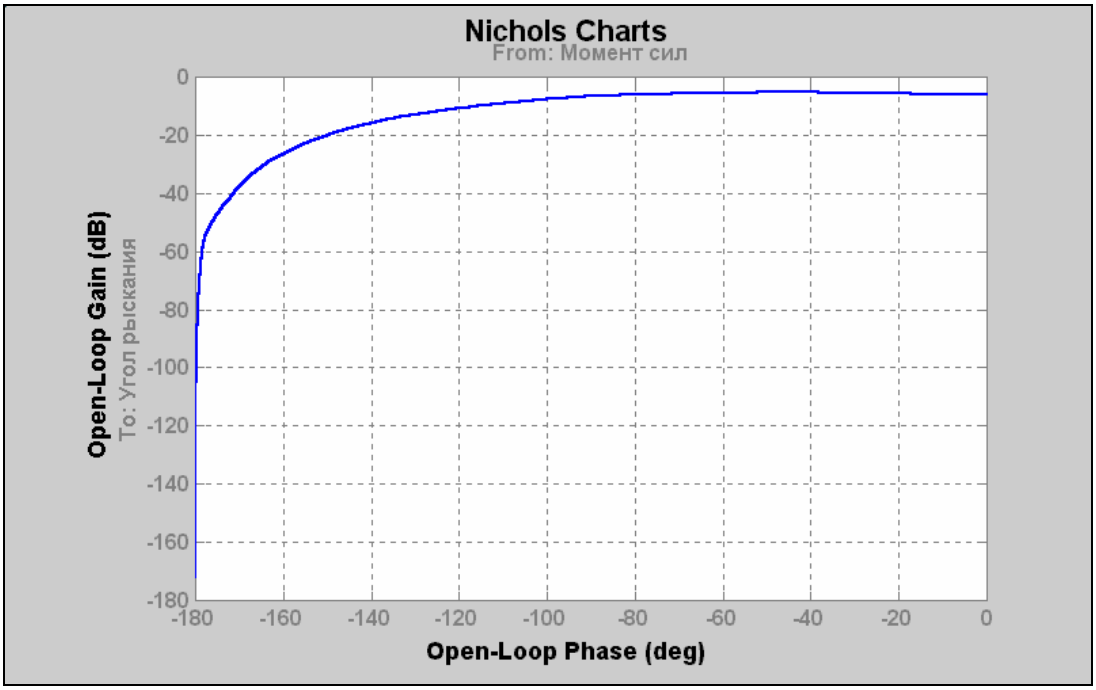

Рис. 6.7

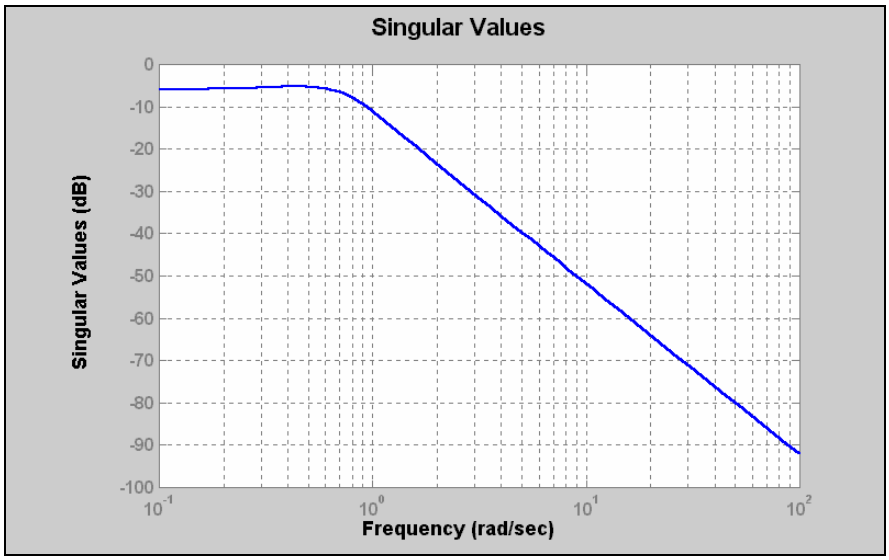

Рис. 6.8

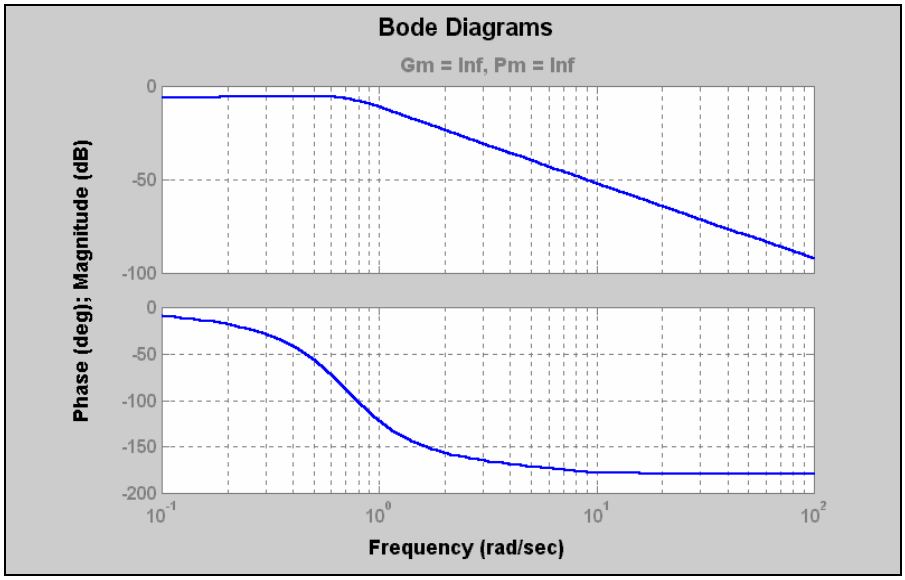

Рис. 6.9

Теперь рассмотрим процедуры, вычисляющие отдельные характеристики и графически показывающие расположение полюсов и нулей системы. К ним можно отнести

- ***pole*** - расчет полюсов системы;
- ***zpkdata*** - расчет полюсов, нулей и коэффициента передачи системы;
- ***gram*** - вычисление граммианов 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 см. 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 - на рис. 6.11.

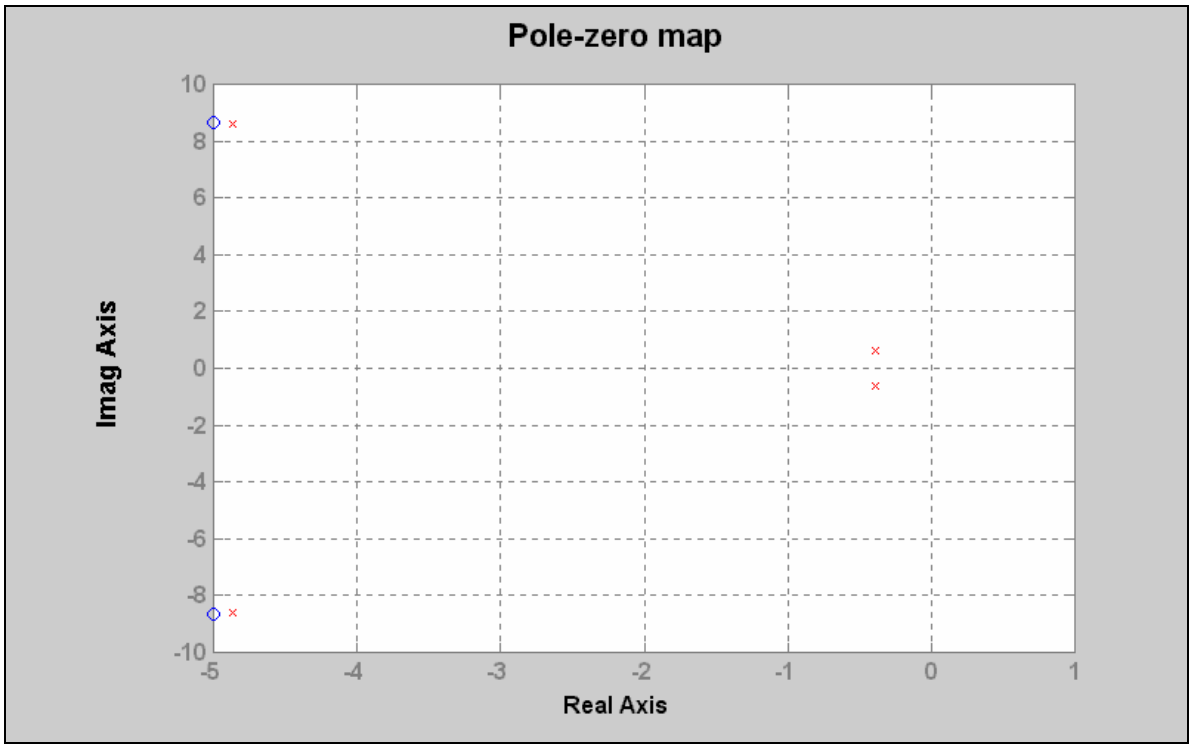

Рис. 6.10

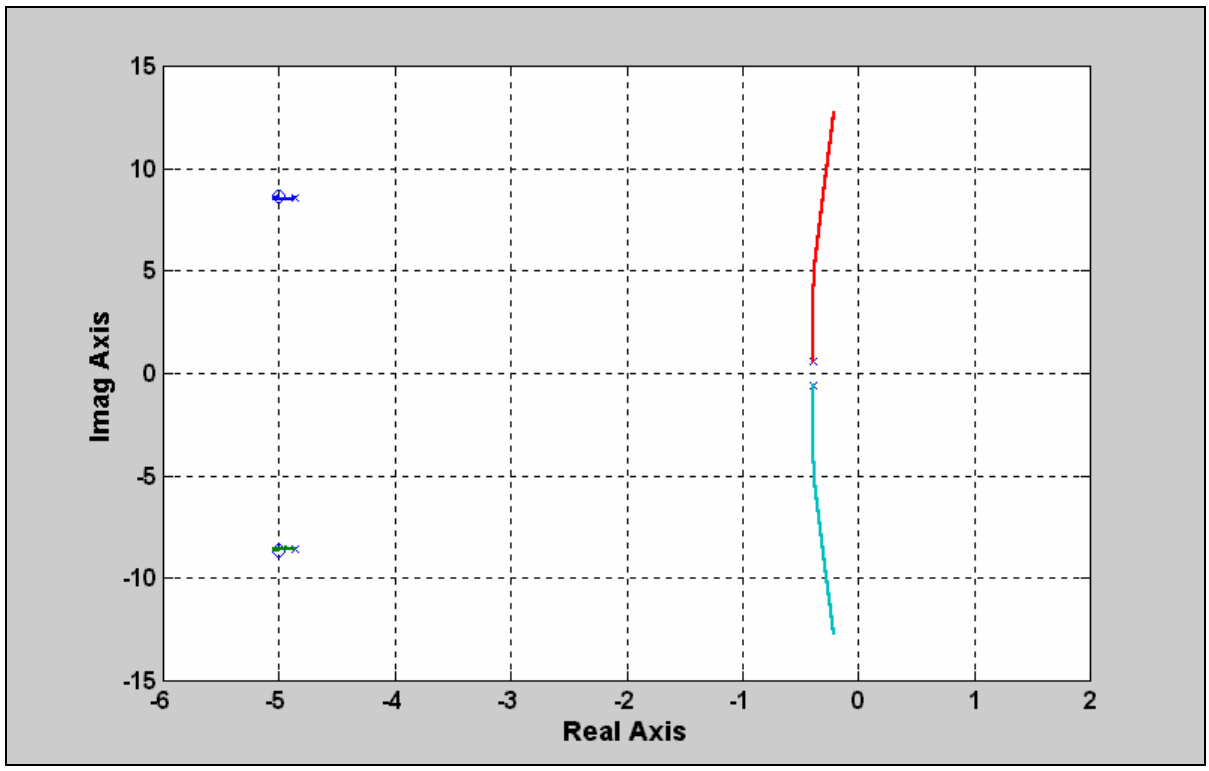

Рис. 6.11

6.4. Интерактивный "обозреватель" ltiview

Набирая в командном окне MatLAB команду ***ltiview***, можно вызывать окно так называемого "обозревателя" LTI-объектов, что позволяет в интерактивном режиме "строить" в этом окне практически все вышеуказанные графики, причем для нескольких систем синхронно:

ltiview

При этом на экране появляется новое окно ***LTIViewer*** (рис. 6.12).

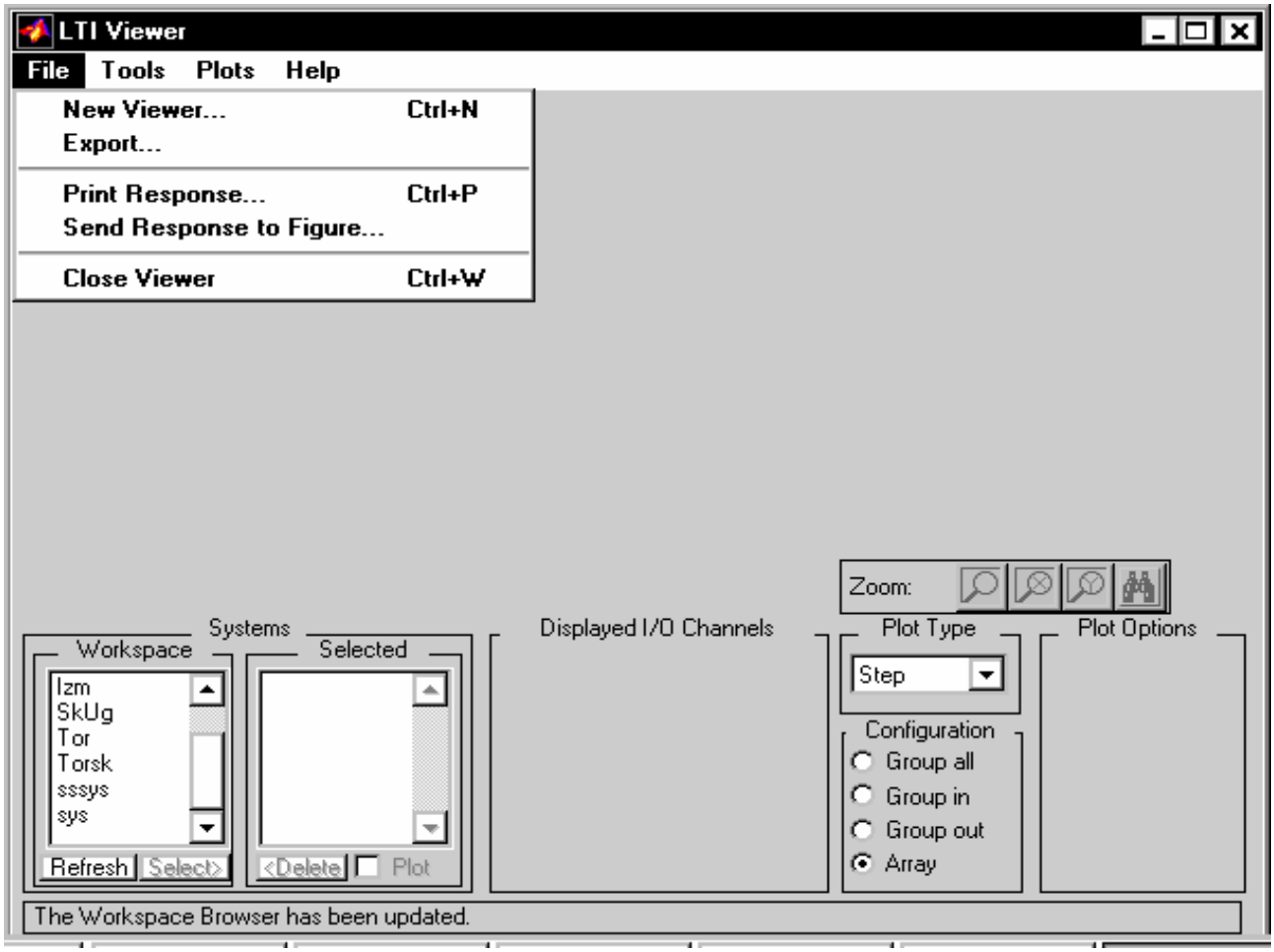

Рис. 6.12

Это окно состоит из нескольких частей. Главное место в нем занимает графическое поле, в котором строятся разнообразные графики. При первом обращении к обозревателю оно пусто. В левой нижней части расположены два окошка Workspace и Selected. В первом из них отображены те LTI-объекты, которые находятся в это время в рабочем пространстве системы MatLAB (на рис. 6.12 - это система sys бокового движения торпеды, созданная ранее). В втором отображаются LTI-объекты, которые загружены в среду обозревателя. В начале работы с обозревателем оно также является пустым.

По правой стороне внизу расположено окошко с выпадающим меню, которое содержит перечень графических процедур, которые можно выполнить. Еще правее расположены окошки, оперирование которыми позволяет определить не-

которые числовые характеристики соответствующего графика. Для этого надлежит в соответствующем окошке поставить "галочку" с помощью мыши.

Работу с обозревателем необходимо начинать с "загрузки" в его среду тех LTI-объектов, которые нужно анализировать. Это делается таким образом.

С помощью мыши следует выделить в окошке Workspace имена LTI-объектов, которые нужно исследовать. Потом "нажать мышкой" на надпись '<Select>' под окошком справа. Соответствующее имя должно появиться в окошке Selected рядом (см. рис. 6.13). Причем по левую сторону имя сопровождается знаком '+', а по правую сторону от него указан в скобках цвет линии графика, которая отвечает 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 в обычном состоянии в окошке справа установлено Step.

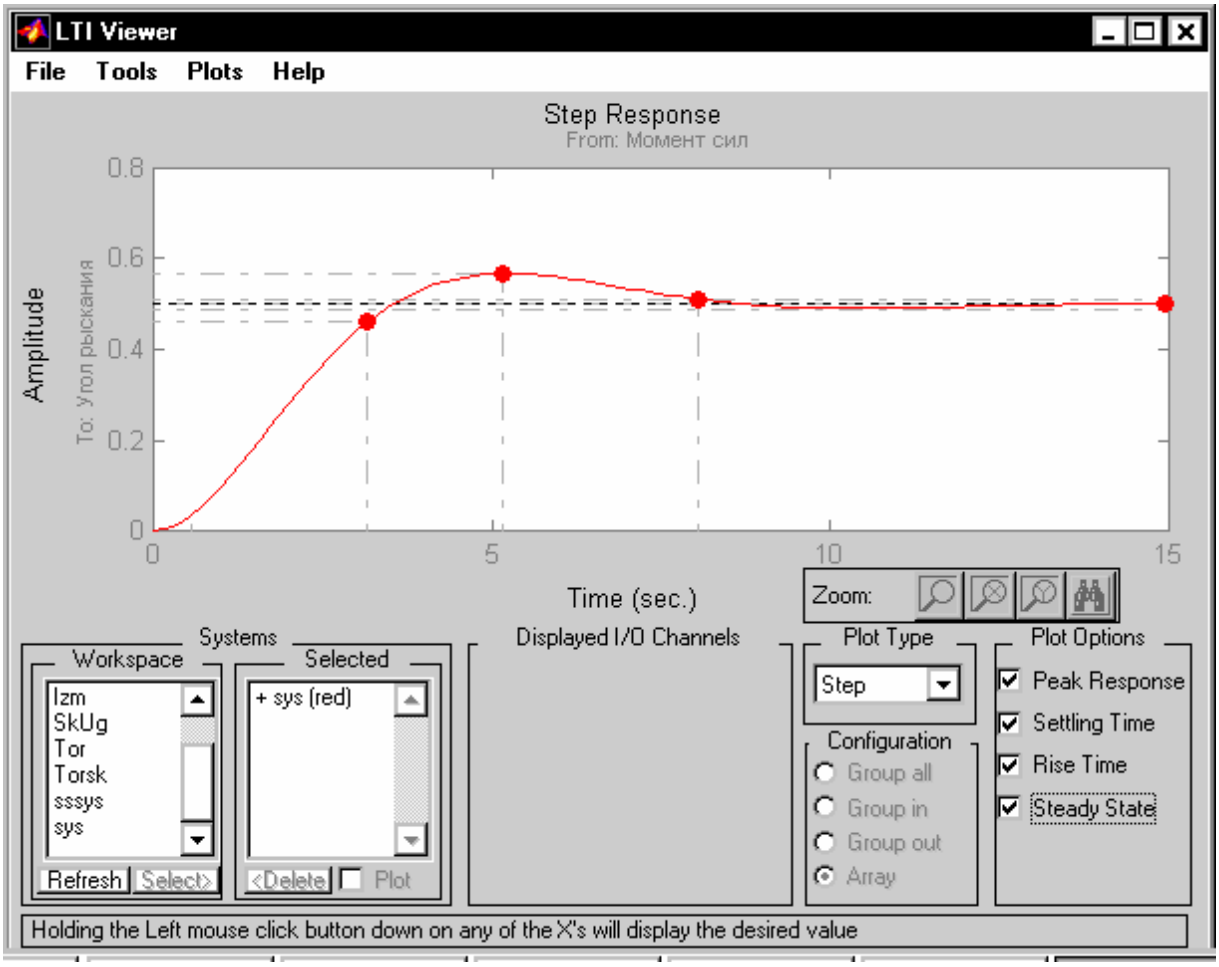

Рис. 6.13

Если в этом окошке "нажать" кнопку выпадающего меню, то оно предоставит такие альтернативные возможности обозревателя:

- ***Step***
- ***Impulse***
- ***Bode***

- ***Nyquist***
- ***Nichols***
- ***Sigma***
- ***PoleZero***
- ***Lsim***
- ***Initial***

Как видим, эти альтернативы полностью совпадают с ранее рассмотренными одноименными процедурами и потому следует ожидать, что их избрание приведет к построению тех же графиков. Например, выбирая раздел ***Impulse***, получим картину, изображенную на рис. 6.14.

В нижнем крайне правом углу окна расположены окошки, которые позволяют отметить те параметры соответствующего графика, которые можно отсчитать по графику.

В качестве примера на рис. 6.13 и 6.14 отмечены ("галочками" в соответствующих окошках) все возможные параметры. Поэтому на графиках появились точки, которые отвечают искомым параметрам.

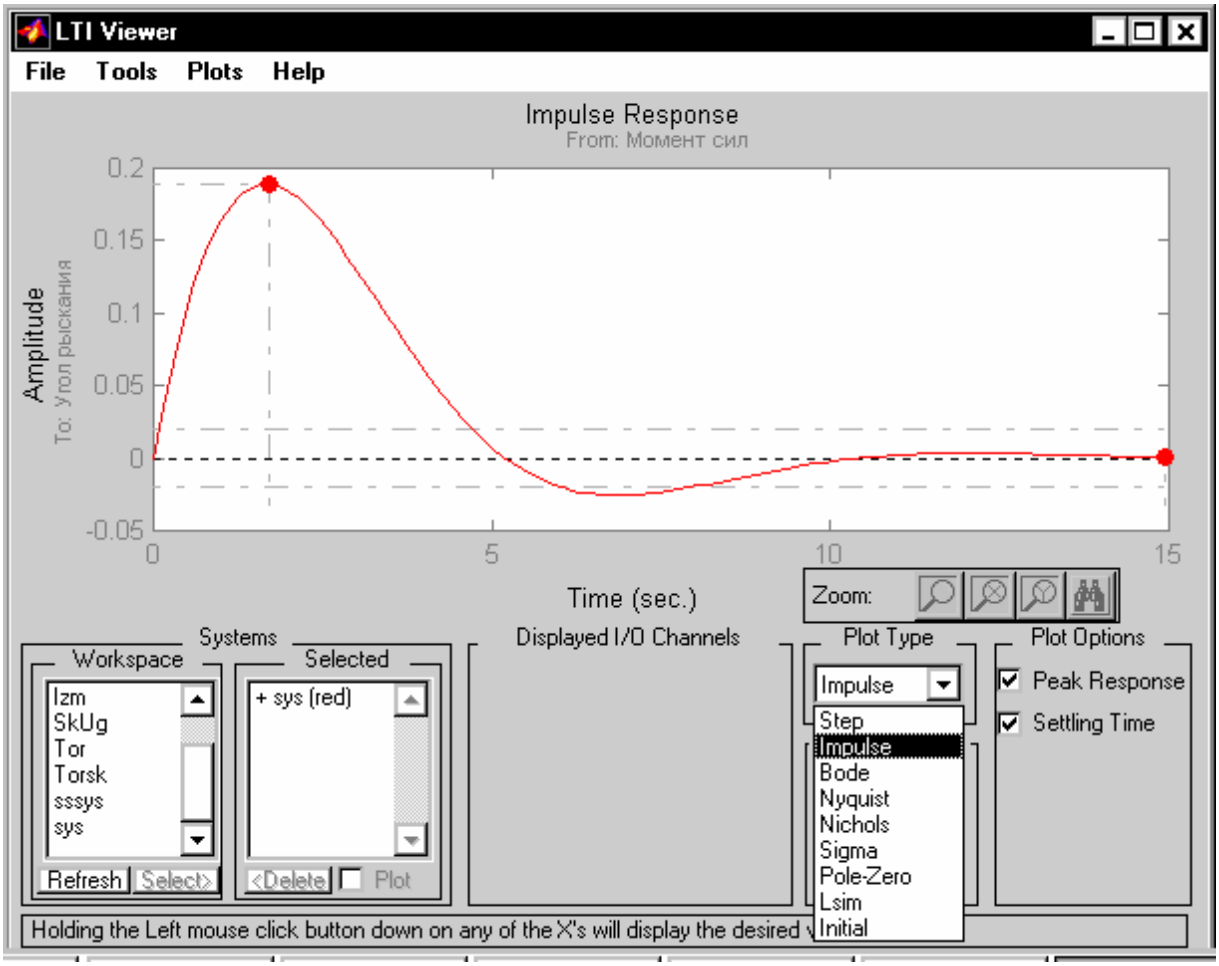

Рис. 6.14

Вид графиков можно изменять, используя меню окна обозревателя. К примеру, раздел *File* меню окна обозревателя (см. рис. 6.12) вызывает дополнительное меню с такими разделами:

- *New Viewer* - открыть новый обозреватель; при избрании этого раздела "мышкой" на экране образовывается еще одно, дополнительное окно такого же обозревателя, в котором можно построить иные нужные графики;
- *Export* - экспортировать LTI-объект; избрание этого раздела приводит к появлению на экране окна *Export LTI Models/Compensators* (рис. 6.15), которое позволяет записать использованные LTI-объекты (имена которых находятся в окошке по левую сторону) или в MAT-файл (окошко *Export to Disk*), или в рабочее пространство (*Export to Workspace*);
- *Print Response* - распечатать графики на принтере;
- *Send Response to Figure* - расположить график в отдельном графическом окне (фигуре); эта команда приводит к появлению на экране дополнительного окна *LTI Viewer Responses*, в котором в объеме всего окна располагается тот же график, который находится к тому времени в графическом поле окна *LTI Viewer*; это удобно с двух точек зрения: во-первых, график в этом окне имеет более обозримую и удобную форму для его распечатывания на принтере, а во-вторых, что более важно, новое окно дает возможность образовать "копию фигуры"; последняя создается так: используется раздел *Edit* в меню нового окна, потом в появившемся подменю выбирается раздел *Copy Figure* - и содержимое окна запоминается в буфере памяти ПК; если теперь перейти в окно какого-нибудь редактора (графического или текстового) и нажать клавиши <Shift> + <Insert>, то в окне этого редактора появится запомненное изображение; именно таким образом получен рис. 6.16, содержащий годограф Найквиста для исследуемой системы (см. рис. 6.17);

Рис. 6.15

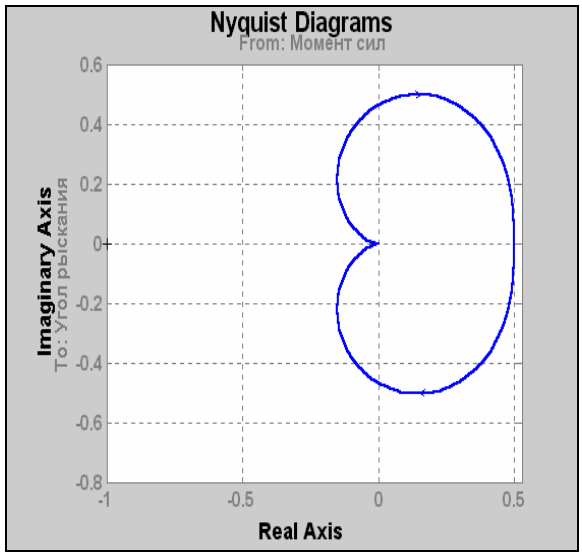

Рис. 6.16

- *Close Viewer* - "закрыть" окно обозревателя.

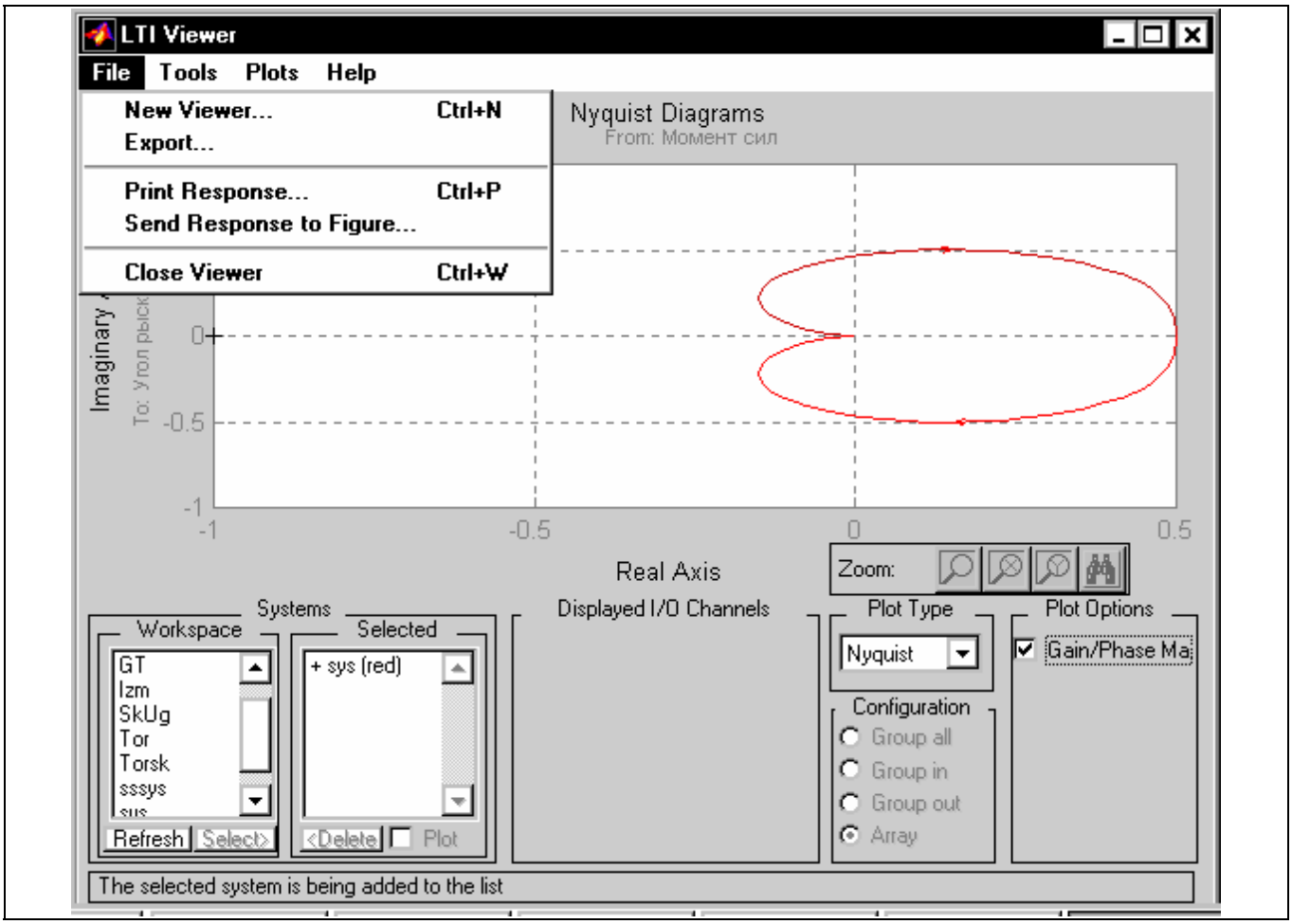

Рис. 6.17

Перейдем к следующему разделу меню окна *LTI Viewer* под названием *Tools*. "Нажав" на него, получим подменю (рис. 6.18), состоящее из трех таких разделов:

- *Viewer Controls* (*Управление Обозревателем*), которое добавляет или уничтожает нижнюю половину окна *LTI Viewer,* в которой располагаются окошки управления выводом графической информации в верхнюю половину окна; каждое нажатие на этот раздел приводит к исчезновению "галочки" рядом с названием раздела (если она перед этим была) или к появлению ее (если ее не было); по умолчанию "галочка" есть, и потому вся управляющая половина изображена в окне *LTI Viewer*; если теперь "нажать мышкой" на *Viewer Controls*, получим в окне картину, представленную на рис. 6.19 ;
- *Response Preferences* (*Свойства Графиков*); нажатие на него приводит к появлению нового окна с таким же названием (рис. 6.20); благодаря этому окну можно установить собственные желательные диапазоны изменений координат, изменить единицы измерения амплитуды и частоты, установить собственные определения времени установления и времени возрастания переходного процесса и т. п.;

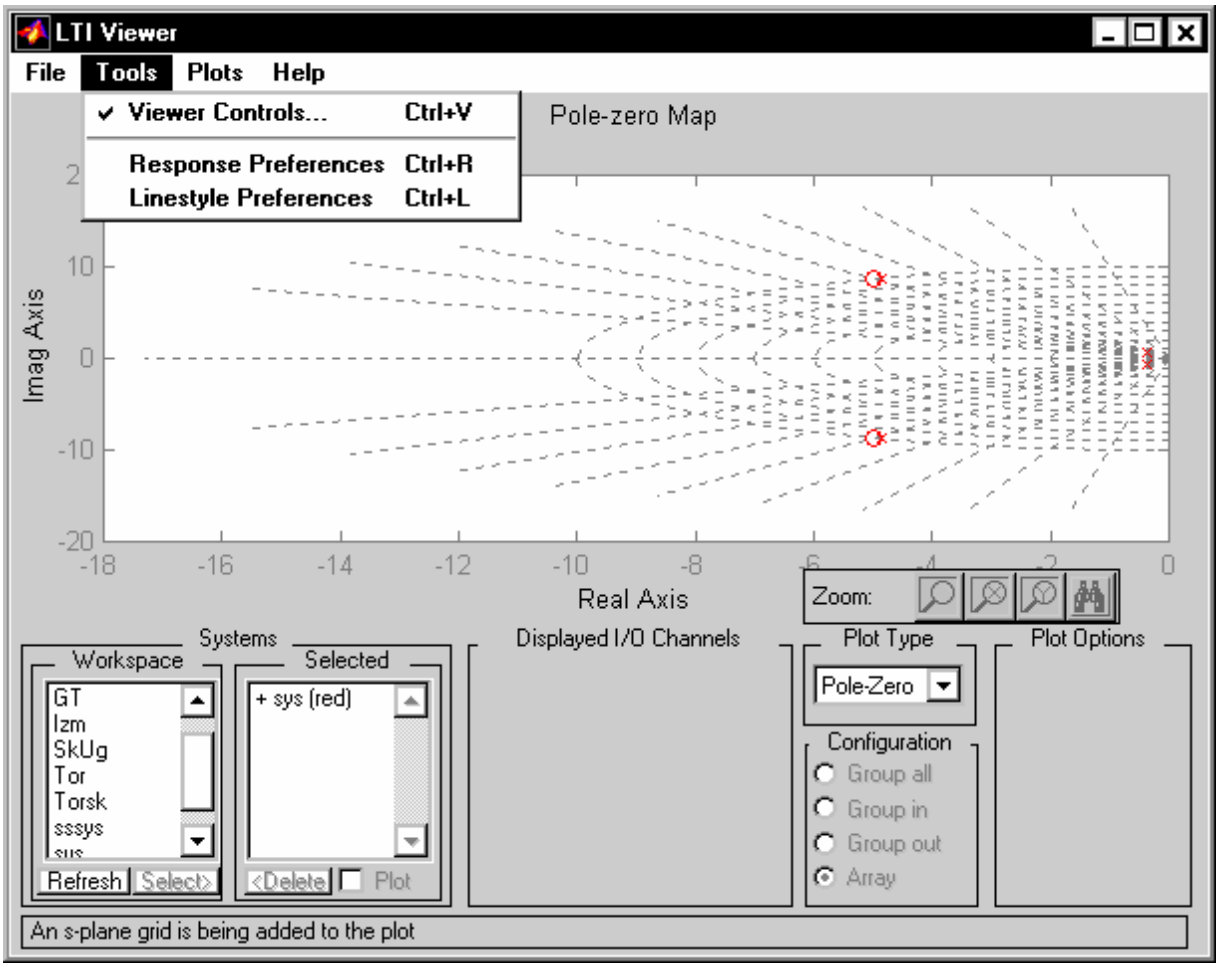

Рис. 6.18

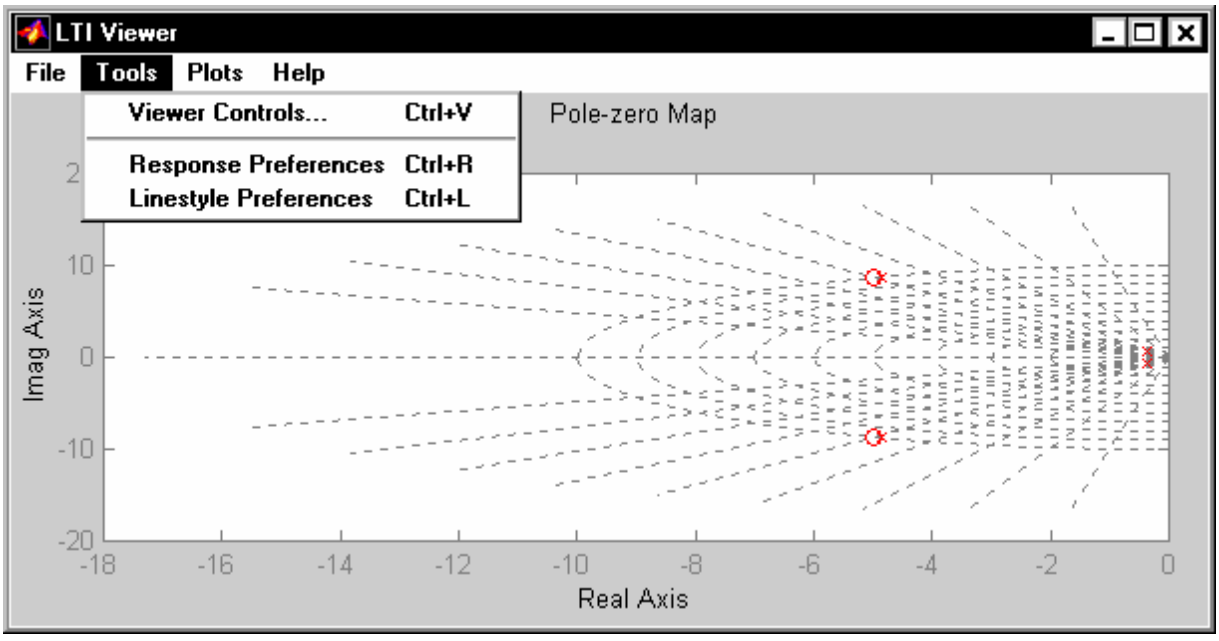

Рис. 6.19

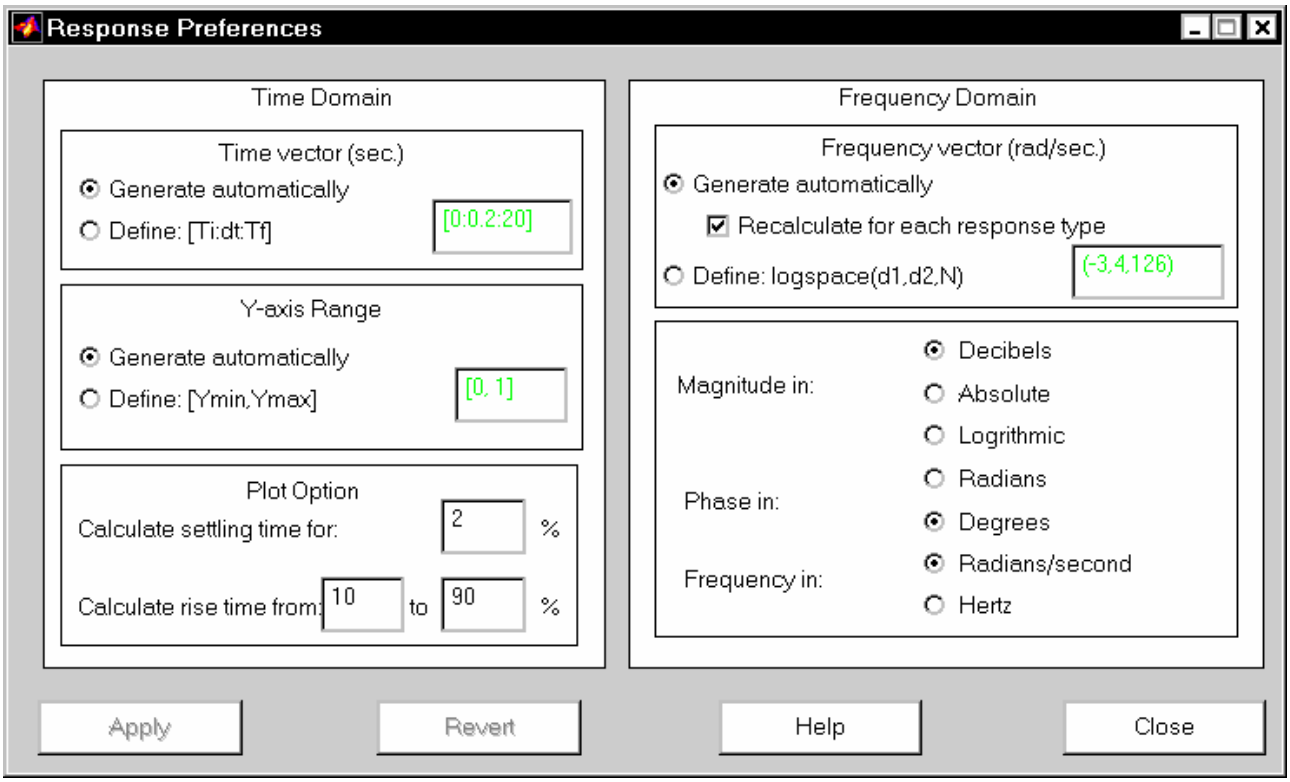

Рис. 6.20

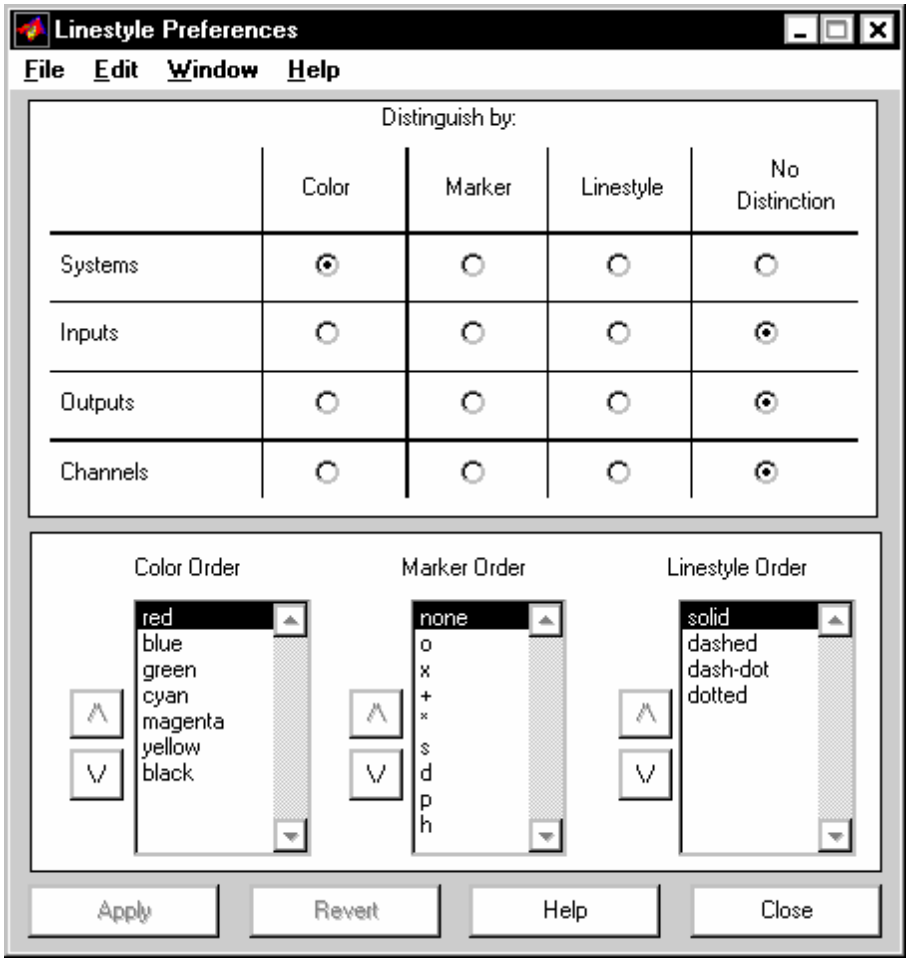

Рис. 6.21

- *Linestyle Preferences* (*Свойства и стили линий*), которое позволяет по собственному усмотрению устанавливать цвета, толщину и стили линий графического изображения; для этого достаточно "нажать мышкой" на этот раздел, и в появившемся окне (рис. 6.21) установить желательные параметры.

В следующем разделе меню обозревателя *Plots* (*Графики*) есть только один раздел дополнительного меню - *Grid On* (*Сетка Установлена*) - рядом с которым есть или нет "галочка" (рис. 6.22). Если "галочка" есть, графики сопровождаются сеткой из координатных линий, например так, как изображено на рис. 6.22.

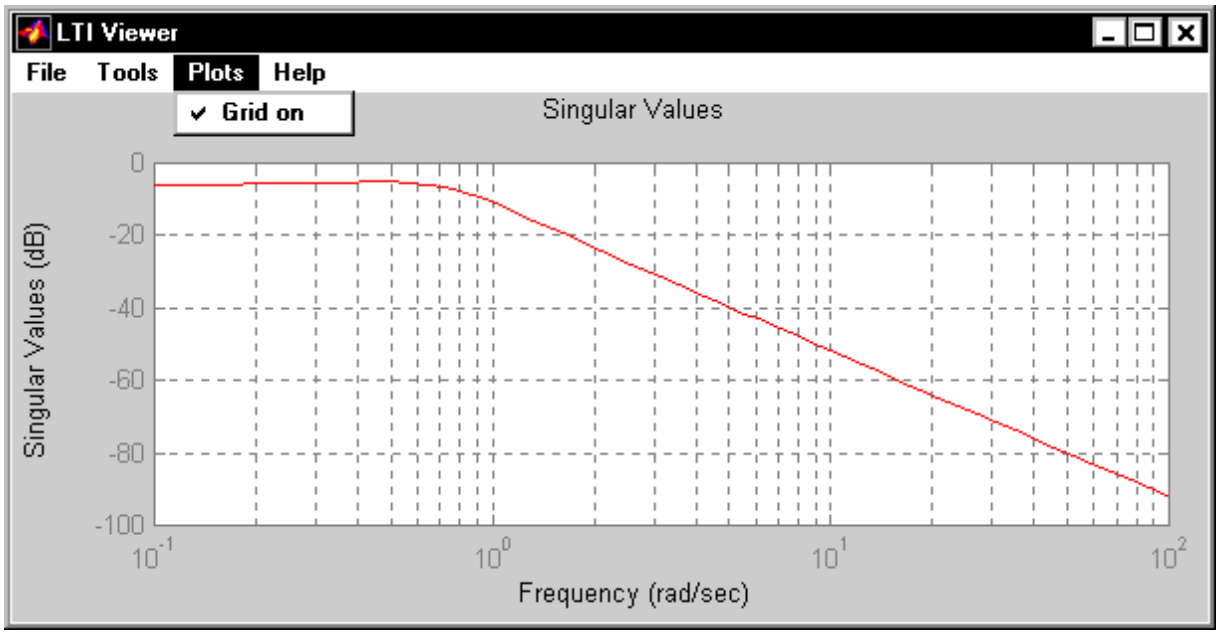

Рис. 6.22

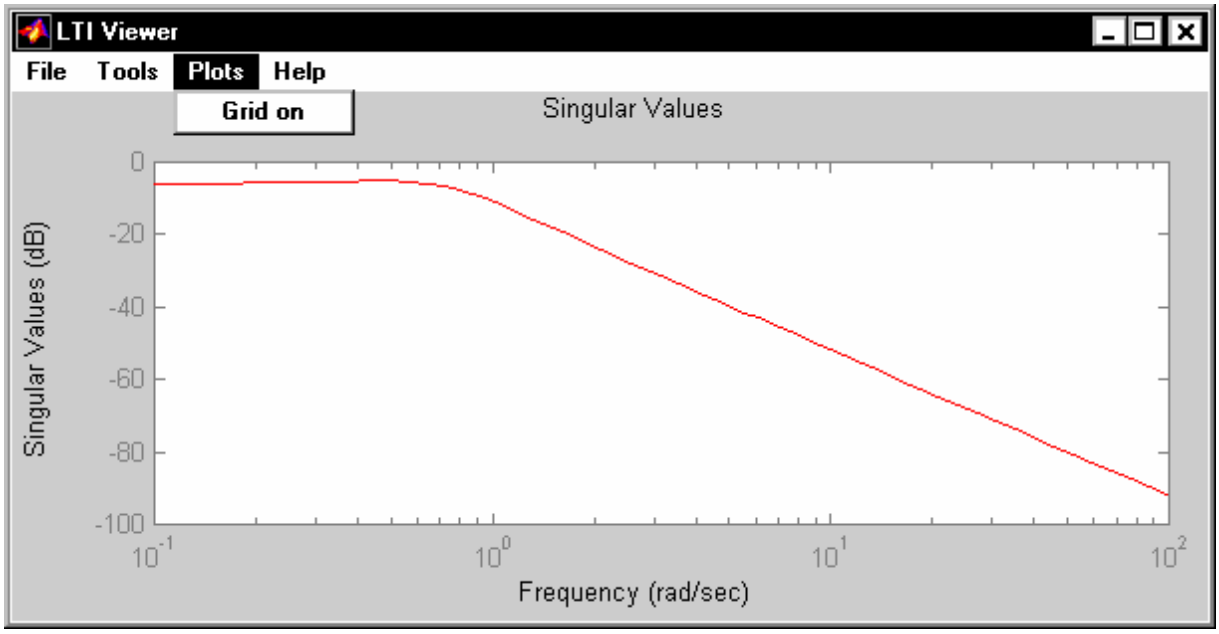

Рис. 6.23

Если ее нет - сетка координатных линий на графики не наносится (рис. 6.23). Установление или исключение "галочки" осуществляется "нажатием" мышью на раздел *Grid On*.

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.

К примеру, процедура ***lqr*** осуществляет проектирование линейно-квадратичного оптимального регулятора для систем непрерывного времени.. При обращении к ней вида [K,S,E] = ***lqr***(A,B,Q,R,N) она рассчитывает оптимальное статическое матричное звено К 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

$$E = EIG(A-B*K)\ . \tag{6.5}$$

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 все уже существующие) и охватывающая только регулируемый объект. Во-вторых, минимизируется функционал не по вектору состояния, а по выходной величине (величинам) системы

$$J = \int \{y'Qy + u'Ru + 2*y'Nu\}\, dt. \qquad (6.6)$$

В этом случае входным параметром процедуры является сама ***ss-***модель системы в форме

$$\frac{dx}{dt} = A \cdot x + B \cdot u, \qquad y = C \cdot x + D \cdot u. \qquad (6.7)$$

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

$$x[n+1] = Ax[n] + Bu[n], \quad y[n] = Cx[n] + Du[n]. \qquad (6.8)$$

при этом минимизируется функционал

$$J = \mathrm{Sum}\ \{y'Qy + u'Ru + 2*y'Nu\}. \qquad (6.9)$$

Применим процедуру к рассматриваемой системе. Получаем:

```
 Q=1; R=1;
 [K,S,E] = LQRY(sssys,Q,R)
K =
   0.1501 0.0988   0.2471   0.1844
   0.1502
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 К 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

$$A'SA - S - (A'SB+N)(R+B'SB)\,(B'SA+N') + Q = 0. \qquad (6.10)$$

Процедура ***kalman*** осуществляет расчет (проектирование) 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

$$\frac{dx}{dt} = A \cdot x + B \cdot u + G \cdot w; \qquad \text{(уравнения состояния)} \qquad (6.11)$$

$$y = C \cdot x + D \cdot u + H \cdot w + v, \qquad \text{(уравнение измерения)} \qquad (6.12)$$

с известными входами 'u', шумовым процессом 'w', шумом измерения 'v' и шумами ковариаций

$$E\{ww'\} = Qn, \quad E\{vv'\} = Rn, \quad E\{wv'\} = Nn, \qquad (6.13)$$

фильтр KEST имеет вход [u;y] и генерирует оптимальные оценки y_e и x_e соответственно величин 'y' и 'x' путем решения уравнений:

$$\frac{dx_e}{dt} = A \cdot x_e + B \cdot u + L \cdot (y - C \cdot x_e - D \cdot u); \qquad (6.14)$$

$$y_e = C \cdot x_e + D \cdot u. \qquad (6.15)$$

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

$$AP + PA' - (PC'+G*N)R^{-1}\,(CP+N'*G') + G*Q*G' = 0\,. \qquad (6.16)$$

Если система SYS задана конечно-разностными уравнениями

$$x[n+1] = Ax[n] + Bu[n] + Gw[n] \qquad \{\text{уравнение состояния}\} \qquad (6.17)$$

$$y[n] = Cx[n] + Du[n] + Hw[n] + v[n] \qquad \{\text{уравнение измерения}\} \qquad (6.18)$$

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

$$\begin{aligned} x[n+1|n] &= Ax[n|n-1] + Bu[n] + L(y[n] - Cx[n|n-1] - Du[n]) \\ y[n|n] &= Cx[n|n] + Du[n] \\ x[n|n] &= x[n|n-1] + M(y[n] - Cx[n|n-1] - Du[n]) \end{aligned} \qquad (6.19)$$

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

$$P = E\{(x - x[n|n-1])(x - x[n|n-1])'\} \quad \text{(решение уравнений Риккати)}$$

$$Z = E\{(x - x[n|n])(x - x[n|n])'\} \quad \text{(апостериорная оценка)}$$

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 этой процедуре RLQG = ***lqgreg***(KEST,K), то она создает в матрице RLQG 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 $u = -K x_e$, причем x_e является оценкой Калмана вектора состояния, основанной на измерениях 'y'. Этот регулятор 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 ее '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 генерирует оценки y_e и x_e соответственно выходов и вектора состояния:

$$\frac{dx_e}{dt} = (A_e - L \cdot C) \cdot x_e + L \cdot y;$$

$$y_e = C \cdot x_e,$$

Похожим 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

$$A'XE + E'XA - (E'XB + S)\, R^{-1}\, (B'XE + S') + Q = 0,$$

или, 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

$$E'XE = A'XA - (A'XB + S)(B'XB + R)^{-1}(A'XB + S)' + Q$$

или, 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 к ней 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

$$A*X*A' - X + Q = 0.$$

7. Моделирование нелинейных систем (пакет SimuLink)

7.1. Общая характеристика пакета SimuLink

Одной из наиболее привлекательных особенностей системы MatLAB является наличие в ее составе наиболее наглядного и эффективного средства составления программных моделей – пакета визуального программирования ***SimuLink***.

Пакет SimuLink позволяет осуществлять исследование (моделирование во времени) поведения динамических нелинейных систем, причем введение характеристик исследуемых систем осуществлять в диалоговом режиме, *путем графической сборки схемы соединений элементарных* (стандартных или пользовательских) *звеньев*. В результате такого составления получается модель исследуемой системы, которую в дальнейшем будем называть S-моделью и которая сохраняется в файле с розширением **.mdl**. Такой процесс образования вычислительных программ принято называть *визуальным программированием*.

Создание моделей в пакете SimuLink основывается на использовании технологии Drag-and-Drop (*Перетяни и оставь)*. В качестве "кирпичиков" при построении S-модели используются визуальные блоки (модули), которые сохраняются в библиотеках SimuLink. S-модель может иметь иерархическую структуру, т. е. состоять из моделей более низкого уровня, причем количество уровней иерархии практически не ограничено. На протяжении моделирования есть возможность наблюдать за процессами, которые происходят в системе. Для этого используются специальные блоки ("обзорные окна"), входящие в состав библиотек SimuLink. Состав библиотек SimuLink может быть пополнен пользователем за счет разработки собственных блоков.

7.1.1. Запуск SimuLink

Запуск SimuLink можно осуществить из командного окна MatLAB или избрав команду **New Model** ("Новая модель") в меню **File**, или нажав соответствующую пиктограмму в линейке инструментов.

При запуске SimuLink открываются два окна (рис. 3.3):

- пустое окно **untitled** (окно, куда будет выводиться схемное представление моделируемой системы, новой S-модели, MDL-файла);
- окно **Simulink Library Browser**, которое содержит перечень основных библиотек SimuLink (рис. 7.1).

Начнем со знакомства с библиотеками SimuLink.

В окне **Simulink Library Browser** представлен перечень SimuLink-библиотек, входящих в состав установленной конфигурации системы MatLAB. Из них главной является библиотека ***SimuLink***, расположенная в первой строке

браузера. Другие библиотеки не являются обязательными. Они включаются в состав общей библиотеки в зависимости от вкусов пользователя.

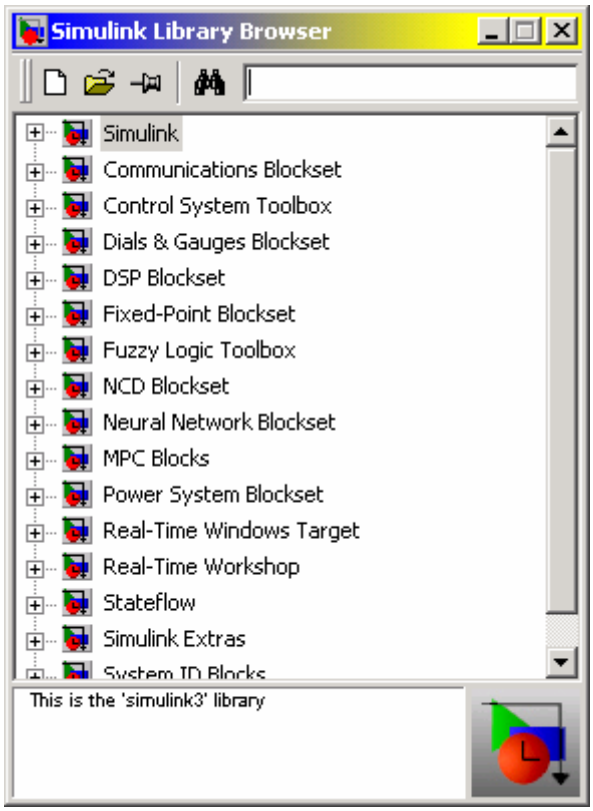

Рис. 7.1

Вторым и более наглядным представлением библиотеки является окно **Library: simulink3**, которое вызовется, если в командном окне MatLAB набрать команду ***simulink3***

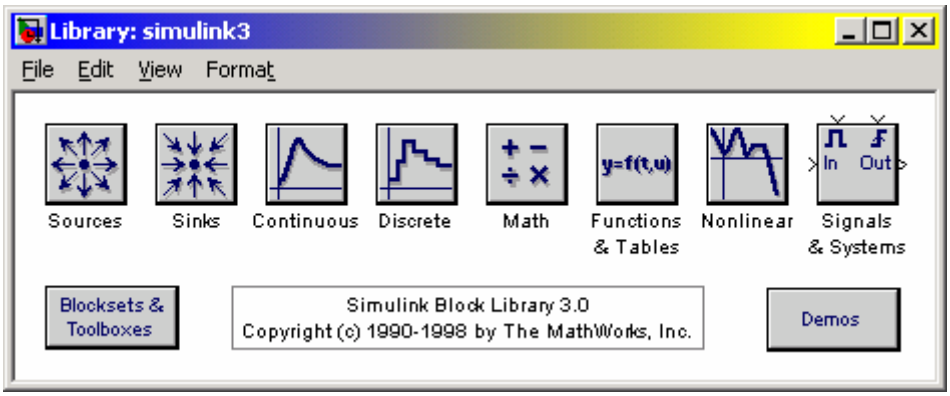

Рис. 7.2

После нажатия клавиши <Enter> на экране возникает окно, представленное на рис. 7.2, в котором отображенные главные разделы лишь основной библиотеки, а именно библиотеки ***SimuLink***.

Окно блок-схемы модели (рис. 7.3) содержит строку меню, панель инструментов (версии MatLAB 5.2 и 5.3) и рабочее поле.

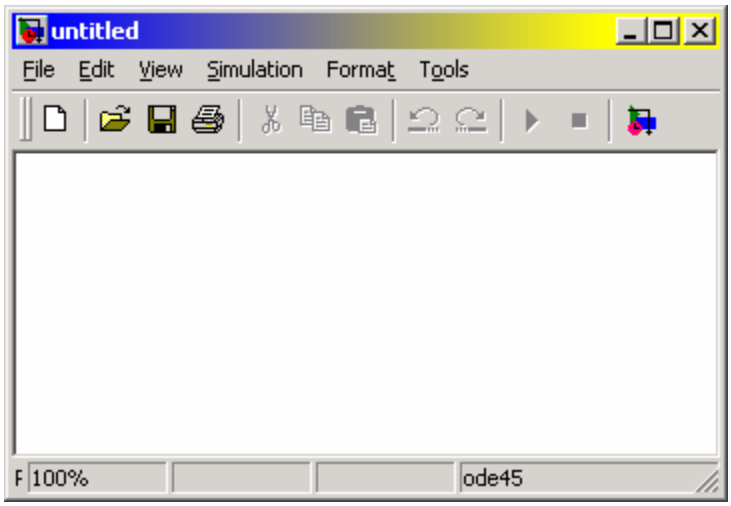

Рис. 7.3

Меню **File** (Файл) содержит команды работы с MDL-файлами, меню **Edit** (Редактирование) - команды редактирования блок-схемы, а меню **View** (Представление) (начиная из версии 5.2) - команды изменения внешнего вида окна. В меню **Simulation** (Моделирование) содержатся команды управления моделированием, а в меню **Format** (Формат) - команды редактирования формата (т. е. внешнего изображения) блоков схемы и блок-схемы в целом. Меню **Tools** (Инструменты) содержит некоторые дополнительные сервисные средства работы с Simulink-моделью.

7.1.2. Библиотека модулей (блоков)

Библиотека блоков SimuLink является набором визуальных объектов, используя которые, можно, соединяя отдельные модули между собою линиями функциональных связей, составлять функциональную блок-схему любого устройства.

Библиотека ***SimuLink*** блоков (рис. 7.2) состоит из 9 разделов. Восемь из них являются главными и не могут изменяться пользователем:

- ***Sources*** (Источники);
- ***Sinks*** (Приемники);
- ***Continuous*** (Непрерывные элементы);
- ***Discrete*** (Дискретные элементы);
- ***Math*** (Математические блоки);
- ***Functions & Tables*** (Функции и таблицы)
- ***Nonlinear*** (Нелинейные элементы);
- ***Signals & Systems*** (Сигналы и системы).

Девятый раздел - ***Blocksets & Toolboxes*** *(Наборы блоков и инструменты)* - содержит дополнительные блоки, включенные в рабочую конфигурацию пакета.

Блоки, которые входят в раздел ***Sources*** (Источники), предназначены для *формирования сигналов*, которые обеспечивают роботу S-модели в целом или отдельных ее частей при моделировании. 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 3 вида:

- блоки, которые используются как обзорные окна при моделировании;
- блоки, обеспечивающие сохранение промежуточных и исходных результатов моделирования;
- блок управления моделированием, который позволяет перерывать моделирование при выполнении тех или других условий.

Раздел ***Continuous*** (непрерывные элементы) содержит блоки, которые можно условно поделить на три группы:

- блоки общего назначения (интеграторы, дифференциатори);
- блоки задержки сигнала;
- блоки линейных стационарных звеньев.

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, которые входят в раздел ***Discrete*** обеспечивают моделирование систем с дискретным временем.

Раздел ***Math*** (математические элементы) - наибольший по составу. Он содержит 20 блоков, которые можно разделить на несколько групп:

- блоки, реализующие элементарные математические операции (умножения, суммирования разных математических объектов);
- блоки, реализующие элементарные математические функции;
- блоки, обеспечивающие логическую обработку входных сигналов;
- блоки, которые преобразуют комплекснозначный сигнал в два действительных и наоборот тем или другим способом;
- блок, который реализует отыскание нуля алгебраической функции.

В разделе ***Functions & Tables*** (функции и таблицы) сосредоточены блоки двух видов:

- блоки, формирующие выходный сигнал по входному в соответствии с заданной таблицей соответствий, осуществляя линейную интерполяцию по этим значениям;
- блоки, позволяющие пользователю создавать собственные блоки с произвольными функциями.

Раздел ***Nonlinear*** (нелинейные элементы) содержит 10 элементов, из которых 7 блоков реализуют разного вида кусочно-линейные зависимости выхода от входа, а три осуществляют разного вида переключения сигнала.

Большинство блоков раздела ***Signals & Systems*** (сигналы и системы) предназначено для разработки сложных S-моделей, содержащих модели более

низкого уровня (подсистемы), и обеспечивают установление необходимых связей между несколькими S-моделями.

Чтобы перейти в окно соответствующего раздела библиотеки, в котором расположены графические изображения блоков, достаточно дважды щелкнуть мышью на пиктограмме этого раздела

Сборка блок-схемы S-модели заключается в том, что графические изображения выбранных блоков с помощью мыши перетягиваются из окна раздела библиотеки в окно блок-схемы, а затем выходы одних блоков в окне блок-схемы соединяются с входами других блоков также при помощи мыши.

Технология перетягивания изображения блока такова: курсор мыши нужно установить на изображении выбранного блока в окне раздела библиотеки, потом нажать левую клавишу мышки и, не отпуская ее, передвинуть курсор на поле блок-схемы, после чего отпустить клавишу. Аналогично осуществляются соединения в блок-схеме линиями выходов одних блоков с входами других блоков: курсор мышки подводят к нужному выходу некоторого блока (при этом курсор должен приобрести форму крестика), нажимают левую клавишу и, не отпуская ее, курсор перемещают к нужному входу другого блока, а потом отпускают клавишу. Если соединение осуществлено верно, на входе последнего блока появится изображение черной затушеванной стрелки.

7.1.3. Раздел Sinks (приемники)

После перехода к разделу ***Sinks*** на экране появляется окно этого раздела, изображенное на рис. 7.4.

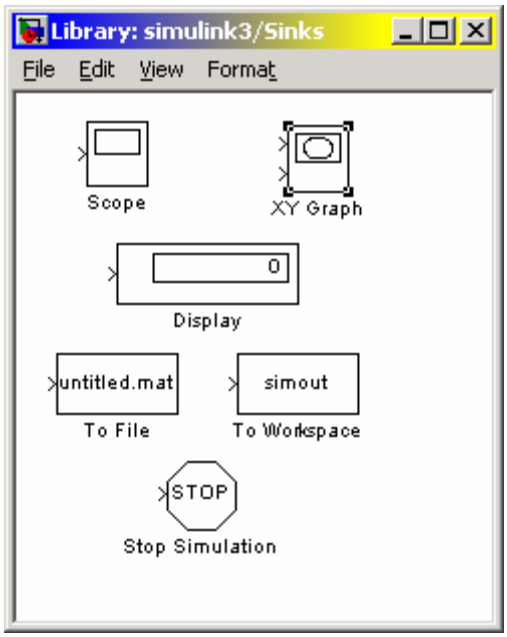

Рис. 7.4

Из его рассмотрения вытекает, что в этом разделе размещены три группы блоков, которые не имеют выходов, а только входы:

1) блоки, которые при моделировании играют роль обзорных окон; к ним относятся:

- блок ***Scope*** с одним входом, который выводит в графическое окно график зависимости величины, подаваемой на его вход, от модельного времени;
- блок ***XYGraph*** с двумя входами, который обеспечивает построение графика зависимости одной моделируемой величины (второй сверху вход) от другой (первый вход);
- блок ***Display*** с одним входом, предназначенный для отображения численных значений входной величины;

2) блоки для сохранения результатов:

- блок ***To File***, который обеспечивает сохранение результатов моделирования на диске в MaAT-файле (с расширением **.mat**);
- блок ***To Workspace***, который сохраняет результаты в рабочем пространстве;

3) блок управления моделированием - ***Stop Simulation***, позволяющий прерывать моделирование при выполнении тех или иных условий; блок срабатывает в том случае, когда на его вход поступает ненулевой сигнал.

Блок ***Scope***

Этот блок позволяет в процессе моделирования наблюдать по графику процессы, которые интересуют исследователя.

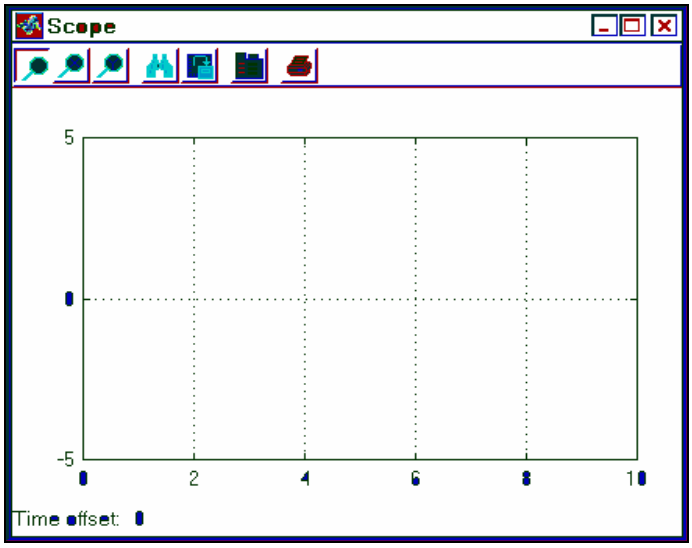

Рис. 7.5

Для настраивания параметров этого блока нужно *после установки изображения блока в окно блок-схемы дважды щелкнуть мышкой на этом изображении*. В результате на экране появится окно **Scope** (рис. 7.5). Размер и пропорции окна можно изменять произвольно, пользуясь мышью. По горизонтальной оси откладываются значения модельного времени, а по вертикальной - значения входной величины, отвечающие этим моментам времени.

Если входная величина блока ***Scope*** является вектором, в окне строятся графики измения всех элементов этого вектора, т. е. столько кривых, сколько элементов в входном векторе, причем каждая - своего цвета. Одновременно в окне может отображаться до 30 кривых.

Для управления параметрами окна в нем предусмотрена панель инструментов, который содержит 7 пиктограмм такого назначения (слева направо):

- изменение масштаба одновременно по обеим осям графика;
- изменение масштаба по горизонтальной оси;
- изменение масштаба по вертикальной оси;
- автоматическое установление оптимального масштаба осей (полный обзор, автошкалирование);
- сохранение установленного масштаба осей;
- вызов диалогового окна настраивания параметров блока ***Scope***;
- распечатка содержимого окна ***Scope*** на принтере.

Первые три пиктограммы являются альтернативными, т. е. в каждый момент времени может быть нажата лишь одна из них. Первые пять пиктограмм не активны до тех пор, пока нет графика в окне ***Scope.*** Активны с самого начала лишь последние две пиктограммы. Нажатие шестой пиктограммы приводит к появлению окна настраивания параметров (свойств) блока **'Scope' properties** (рис. 7.6).

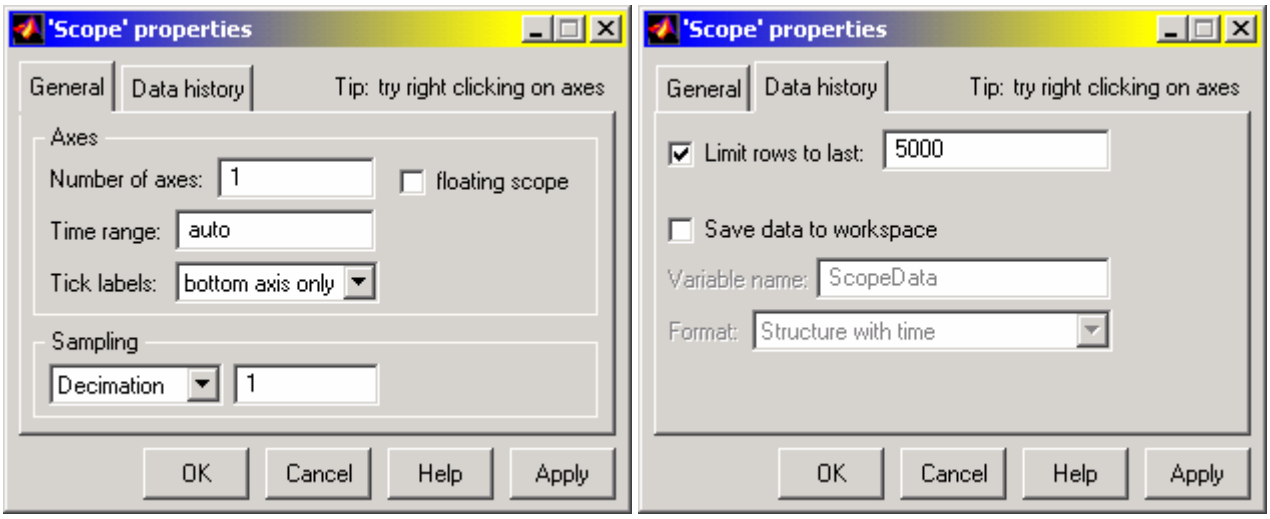

Рис. 7.6

Это окно имеет две вкладки:

- ***General*** (Общие), она позволяет установить параметры осей;
- ***Data history*** (Представление данных), которая предназначена для введения параметров представления данных блока ***Scope.***

В нижней части окна расположены кнопки: *Apply (Применить), Help (Вызов справки) , Cancel (Вернуться назад)* и *OK (Подтвердить установку).*

На вкладке ***General*** имеются поля *Axes* и *Sampling*.

В поле *Axes* можно установить:

- в окошке *Number of axes* (Количество осей) - количество графических полей в графическом окне ***Scope*** (одновременно изменяется количество входов в блок ***Scope***);
- верхнюю границу отображаемого модельного времени по оси абсцисс (окошко *Time range*); при этом следует принимать во внимание следующее: *если размер заданного интервала моделирования ($T_м$) не превышает установленное значение Time range (т. е. весь процесс умещается в окне* ***Scope****), то под графиком в строке Time offset выводится 0. В случае же, когда интервал моделирования превышает значения Time range, в окне* ***Scope*** *отображается только последний отрезок времени, меньший по размеру, чем Time range и равный $T_м$-n*Time range, где n - целое число; при этом в строке Time offset выводится размер "скрытого" интервала времени - n*Time range;* например, если значения *Time range* равняется 3, а продолжительность интервала моделирования установлена 17, то в окне ***Scope*** будет выведен график моделируемого процесса за последние 2 единицы времени, а строка под графиком будет иметь вид: *Time offset: 15*;
- в окошке *Tick Labels* – вид оформления осей координат в графиках графичного окна; если вызвать нисходящий список в нем, то в нем увидим три альтернативы – *all* (все), *none* (нет), *bottom axis only* (только нижней оси); избрания *all* приводит к тому, что деления по осям наносятся вдоль каждой из осей всех графиков; выбор *bottom axis only* вызовет исчезновение делений по всем горизонтальным осям графических полей (если их несколько), при этом останутся лишь деления по самой нижней из них; наконец, если выбрать *none*, то исчезнут все деления по осям графиков и надписи на них, график займет все поле окна и окно примет вид, представленный на рис. 7.7.

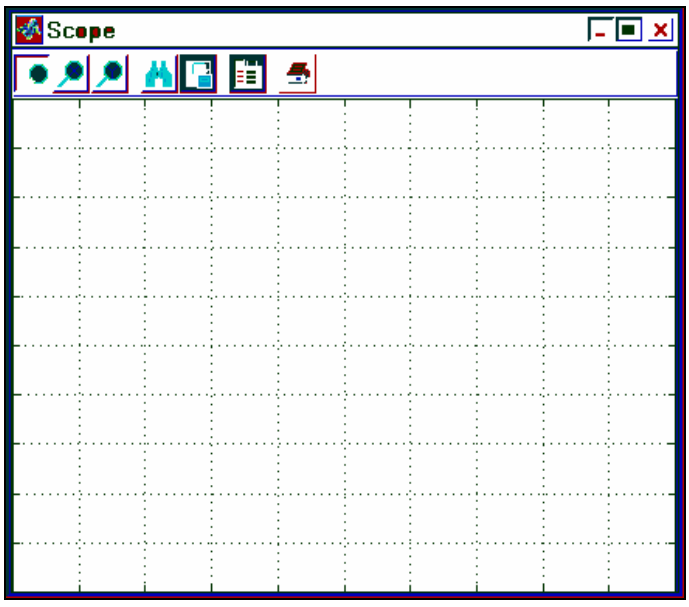

Рис. 7.7

Окошко рядом с надписью *floating scope* предназначено для отключения входов в блок ***Scope***. Для этого достаточно поместить в него "галочку", щелкнув в нем мішью. Если "галочка" установлена, то ***Scope*** отображается как блок без входа, и если он был связан по входу с другими блоками, то эти связи "обрываются".

Поле *Sampling* содержит лишь одно окошко с надписью *Decimation*. В нем можно задать целое положительное число, которое равно количеству шагов (дискретов) времени, в которых используются полученные данные для построения графиков в окне ***Scope***.

Вторая вкладка ***Data history*** (рис. 7.6) позволяет задать максимальное количество элементов массивов данных, которые используются для построения графиков в окне ***Scope*** (окошко рядом с надписью *Limit rows to last* (Максимальная граница строк)). Другие окошки этой вкладки становятся досягаемыми, если в окошке рядом с надписью *Save data to work space* (Записать данные в рабочее пространство) поставить "галочку" (мышью). При этом становится возможным записать данные, которые выводятся на графики окна ***Scope,*** в рабочее пространство системы MatLAB, и становятся активными окошки с надписями *Variable name* (Имя переменной) и *Format* (Формат). В первое из них можно ввести имя переменной, под которым будут сохраняться данные в рабочем пространстве системы (по умолчанию эти данные будут записаны под именем *ScopeData*). Окошко *Format* дает возможность выбрать один из трех форматов записи данных – *Matrix* (Матрица), *Structure* (Структура) и *Structure with time* (Структура с временем).

Любые изменения, сделанные в окне **Properties**, изменяют окно ***Scope*** лишь в случае, если после введения этих изменений нажата кнопка *Apply* в нижней части окна.

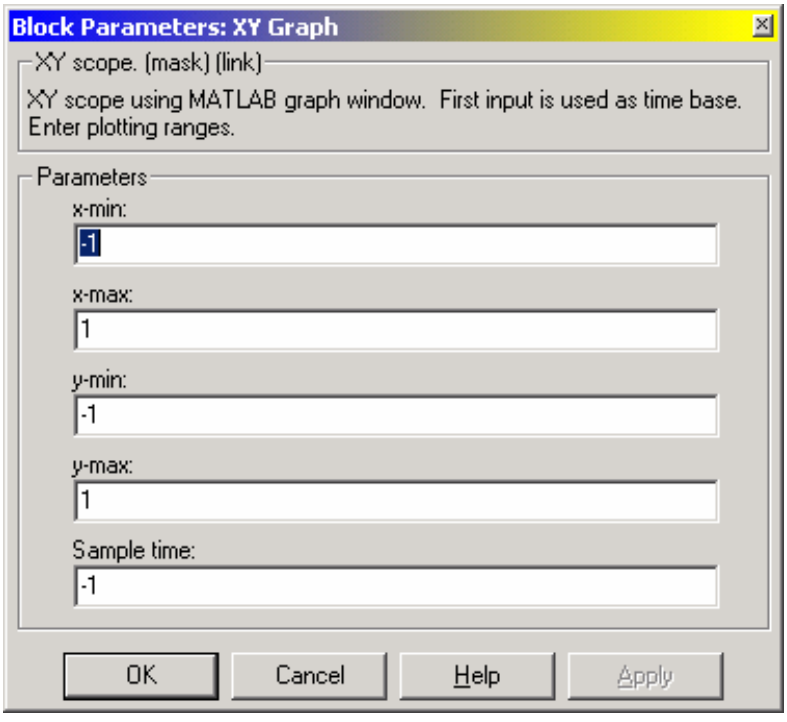

Рис. 7.8

*Блок **XYGraph***

Этот блок также является обзорным окном. В отличие от ***Scope***, он имеет два входа: на первый (верхний) подается сигнал, значения которого откладываются по горизонтальной оси графика, а на второй (нижний) - по вертикальной оси.

Если перетянуть этот блок на поле блок-схемы, а потом на изображении его щелкнуть дважды мышкой, то на экране появится окно настраивания блока (рис. 7.8), которое позволяет установить границы изменений обеих входных величин, между которыми будет построен график зависимости второй величины от первой, а также задать дискрет по времени.

Приведем пример использования блока ***XYGraph***. Для этого перетянем в окно блок-схемы изображение этого блока из окна *Library: simulink3/Sinks*, а из окна *Library: simulink3/Sources* **-** два блока-источника ***Clock*** и ***Sine Wave***. Соединим выходы блоков-источников с входами блока ***XYGraph.*** Получим блок-схему, приведенную на рис. 7.9. Если теперь выбрать мышью меню *Simulation* в строке меню окна блок-схемы, а в нем - команду *Start*, то по окончании расчетов на экране возникнет окно ***XYGraph*** и в нем будет построено изображение, представленное на рис. 7.10.

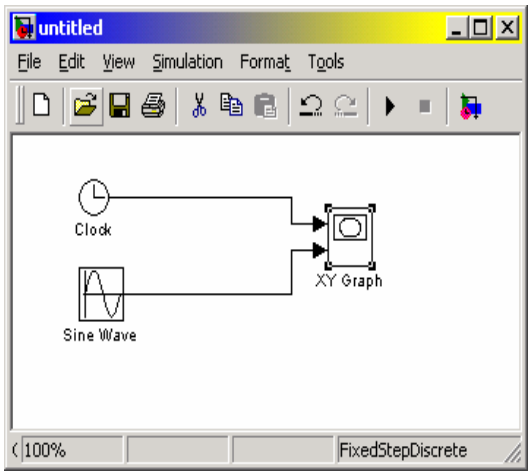

Рис. 7.9

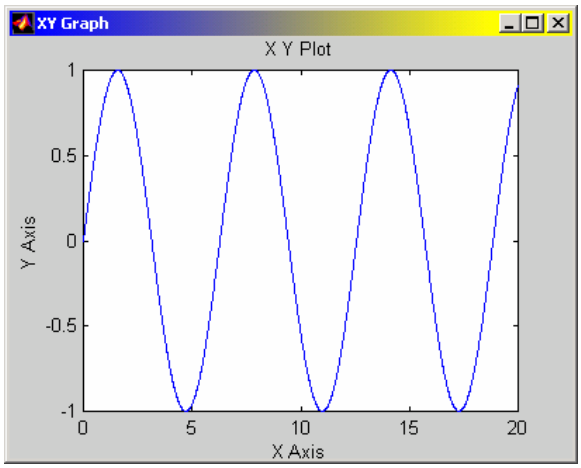

Рис. 7.10

*Блок **Display***

Этот блок предназначен для вывода на экран численных значений величин, которые фигурируют в блок-схеме.

Блок имеет 4 параметра настраивания (рис. 7.11). Список *Format* задает формат вывода чисел; вид формата избирается с помощью нисходящего меню, содержащего 5 пунктов: *short, long, short_e, long_e, bank*. Поле введения *Decimation* позволяет задать периодичность (через сколько шагов времени) вывода значений в окне *Display*. Переключатель *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

представление 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 видимыми, необходимо растянуть изображение блока. Для этого следует выделить блок, подвести курсор мышки к одному из его углов, нажать левую клавишу мыши и, не отпуская ее, растянуть изображение блока, пока не исчезнет черный треугольник.

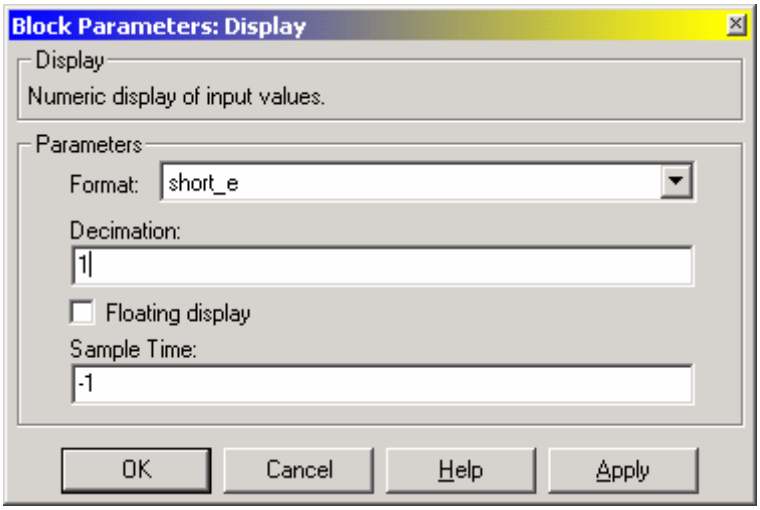

Рис. 7.11

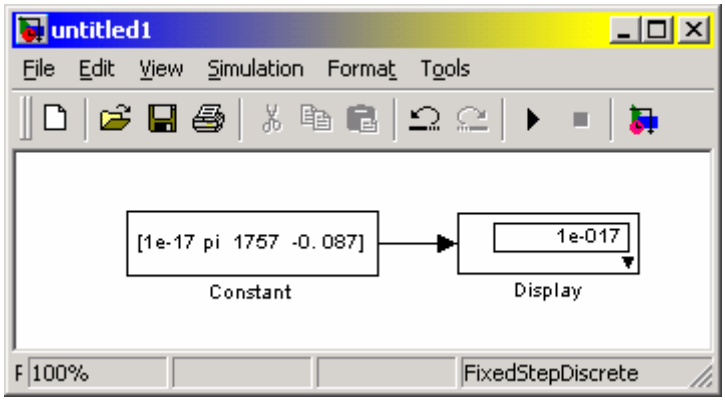

Рис. 7.12

Для примера создадим блок-схему (рис. 7.12) из двух элементов – блока-источника ***Constant*** и блока-приемника ***Display***.

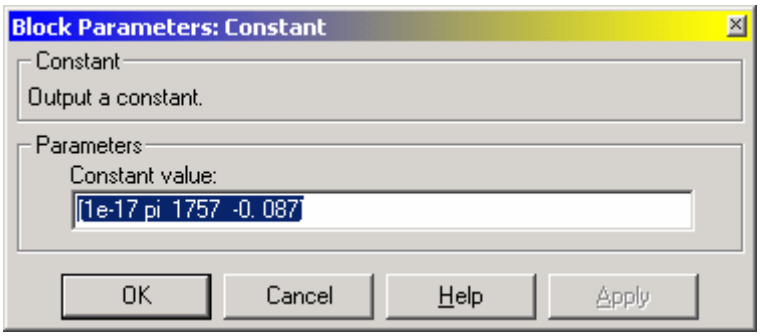

Рис. 7.13

Вызвав окно настраивания блока ***Constant*** (рис. 7.13), установим в нем значения константы-вектора, который состоит из четырех элементов [1e-17 pi 1757 -0.087]. Вызывая окно настраивания блока ***Display***, установим с его помощью формат вывода чисел *short_e*. После активизации команды *Start* из меню Simulation, получим изображение окна блок-схемы, показанное на рис. 7.12. Растягивая изображение блока ***Display*** на блок-схеме, получим картину, представленную на рис. 7.14.

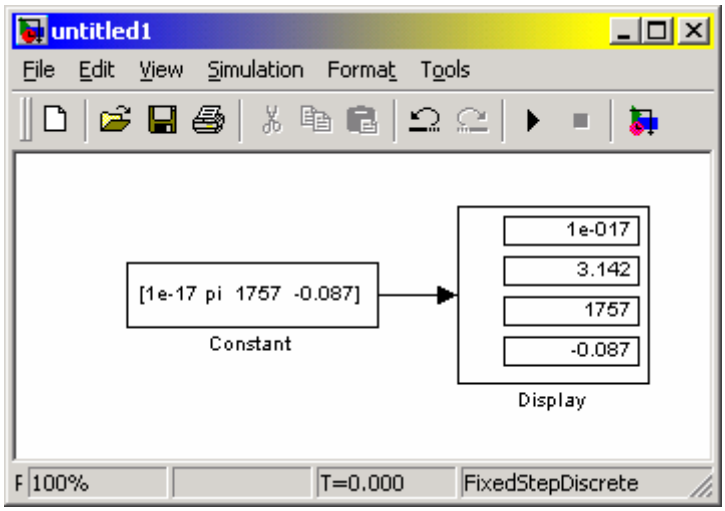

Рис. 7.14

*Блок **To File***

Этот блок обеспечивает запись значений величины, поданной на его вход, в MAT-файл данных для использования их в последующем в других S-моделях.

Блок имеет такие параметры настраивания (см. рис. 7.15):

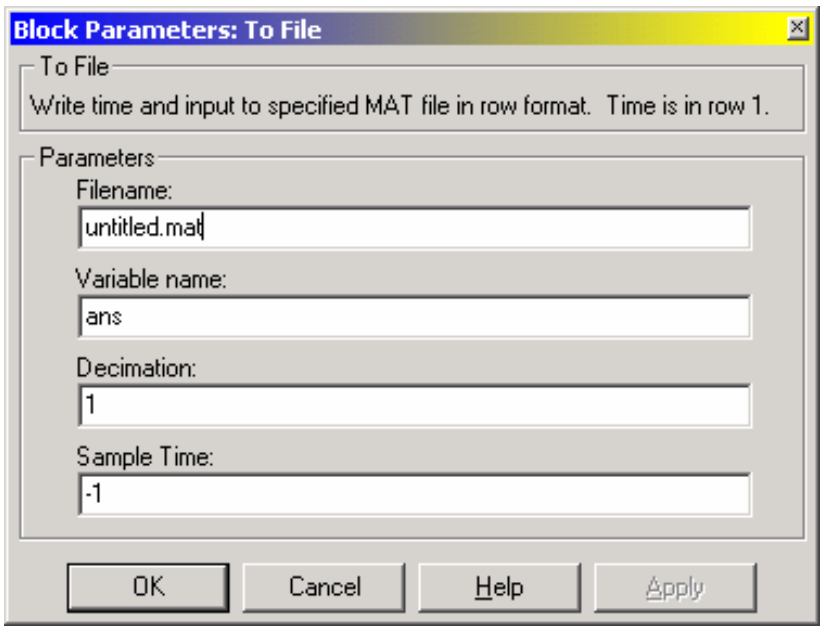

Рис. 7.15

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 - размер дискрета времени для данного блока.

Следует отметить, что значения данных, которые подаются во вход блока записываются в выходную переменную (например, ***ans***) так: первую строку матрицы образуют значения соответствующих моментов времени; вторая строка содержит соответствующие значения первого элемента входного вектора, третья строка - значения второго элемента и т.д. В результате записывается матрица размером (k+1)*N, где k - количество элементов входного вектора, а N - количество точек измерения (или количество моментов времени, в которые осуществлены измерение).

*Блок **To Workspace***

Этот блок предназначен для сохранения данных в рабочем пространстве системы MatLAB. Данные сохраняются в виде матрицы размером (N*k), структура которой отличается от структуры данных в MAT-файле тем, что:

- значения величин, которые сохраняются, расположены по столбцам, а не по строкам;
- не записываются значения модельного времени.

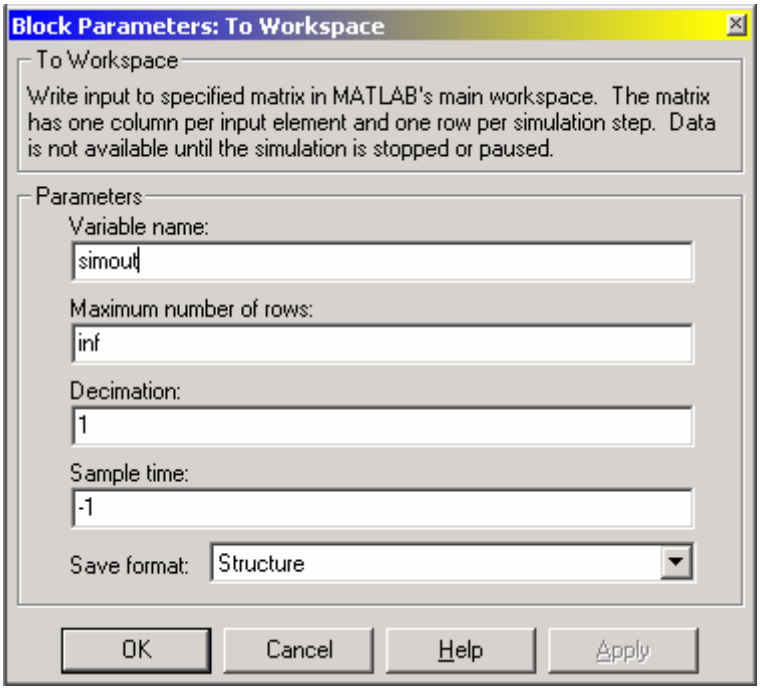

Рис. 7.16

Блок имеет 4 параметры настраивания (см. рис. 7.16):

Variable name - имя, под которым данные сохраняются в рабочем пространстве (по умолчанию - *simout*);

Maximum number of rows - максимально допустимое количество строк, т. е. значений данных, которые записываются; по умолчанию она задается константой ***inf***, т. е. данные регистрируются на всем интервале моделирования;

Decimation и *Sample Time* имеют то самое содержание, которое и раньше;

Окошко *Save format* (Формат записи) позволяет выбрать один из трех вариантов записи данных: *Matrix* (Матрица), *Structure* (Структура) и *Structure with time* (Структура с временем).

7.1.4. Раздел Sources (Источники)

После выбора раздела ***Sources*** библиотеки SimuLink на экране появится дополнительное окно, показанное на рис. 7.17.

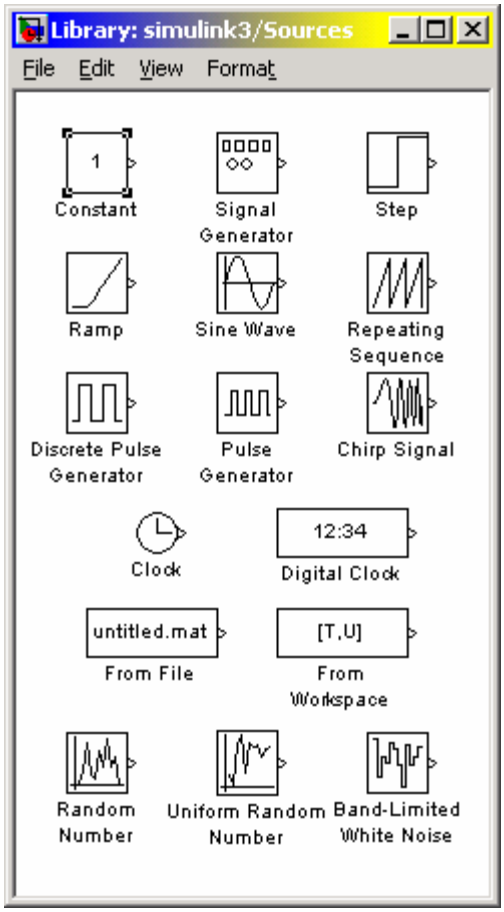

Рис. 7.17

Как видим, в этом разделе библиотеки в качестве источников сигналов предусмотрены такие блоки:

- ***Constant*** - формирует постоянную величину (скаляр, вектор или матрицу);

- ***Signal Generator*** - создает (генерирует) непрерывный колебательный сигнал одной из волновых форм на выбор - синусоидальный, прямоугольный, треугольный или случайный;
- ***Step*** - генерирует сигнал в виде одиночной ступеньки (ступенчатый сигнал) с заданными параметрами (начала ступеньки и ее высоты);
- ***Ramp*** - создает линейно восходящий (или нисходящий) сигнал;
- ***Sine Wave*** - генерирует гармонический сигнал;
- ***Repeating Sequence*** - генерирует периодическую последовательность;
- ***Discrete Pulse Generator*** - генератор дискретных импульсных сигналов;
- ***Pulse Generator*** - генератор непрерывных прямоугольных импульсов;
- ***Chirp Signal*** - генератор гармонических колебаний с частотой, которая линейно изменяется с течением времени;
- ***Clock*** (Часы) - источник непрерывного сигнала, пропорционального модельному времени;
- ***Digital clock*** (Цифровые часы) - формирует дискретный сигнал, пропорциональный времени;
- ***Random Number*** - источник дискретного сигнала, значения которого являются случайной величиной, распределенной по нормальному (гауссовому) закону;
- ***Uniform Random Number*** - источник дискретного сигнала, значения которого являются случайной равномерно распределенной величиной;
- ***Band-Limited White Noise*** - генератор белого шума с ограниченной полосой частот.

Оставшиеся два блока из раздела ***Sources*** обеспечивают использование в модели данных, полученных раньше. Первый из них - ***From File*** - предназначен для введения в S-модель данных, которые сохраняются на диске в MAT-файле. Второй - ***From Workspace*** - обеспечивает введения в модель данных непосредственно из рабочего простора MatLAB. Напомним, что структура данных в MAT-файле является многомерным массивом с переменным количеством строк, которое определяется количеством регистрируемых переменных. Элементы первой строки содержат последовательные значения модельного времени, элементы в других строках - значения переменных, соответствующие отдельным моментам времени.

Как и другие блоки библиотеки SimuLink, блоки-источники могут настраиваться пользователем, за исключением блока ***Clock***, работа которого основана на использовании аппаратного таймера компьютера.

Блок ***Constant***

Блок предназначен для генерирования процессов, который являются неизменными во времени, т. е. имеют постоянное значение. Он имеет один параметр настраивания (см. рис. 7.13) - *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 настраивания этого блока выглядит так, как показано на рис. 7.18. Как видно, в параметры настраивания входят:

- *Wave form* - форма волны - позволяет выбрать одну из таких форм периодического процесса
  1) *Sine* - синусоидальные волны;
  2) *Square* - прямоугольные волны;
  3) *Sawtooth* - треугольные волны;
  4) *Random* - случайные колебания;

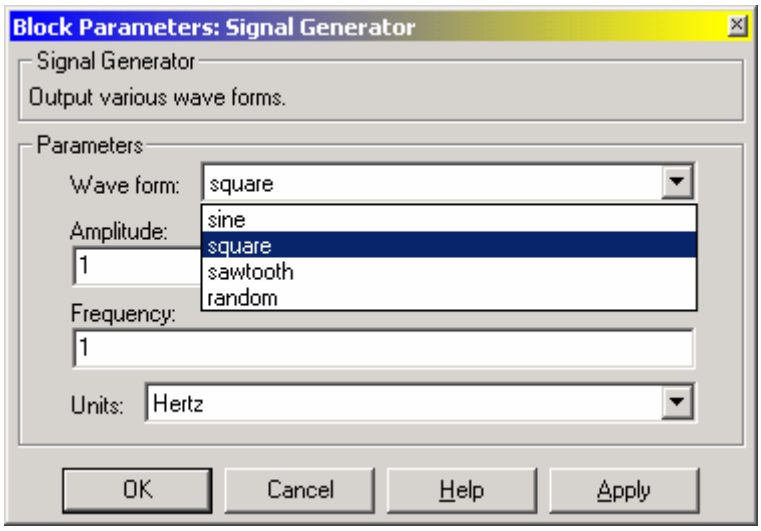

Рис. 7.18

- *Amplitude* - определяет значения амплитуды колебаний, которые генерируются;
- *Frequency* - задает частоту колебаний;
- *Units* - позволяет выбрать одну из единиц измерения частоты с помощью нисходящего меню - *Hertz* (в Герцах) и *Rad/Sec* (в радианах в секунду).

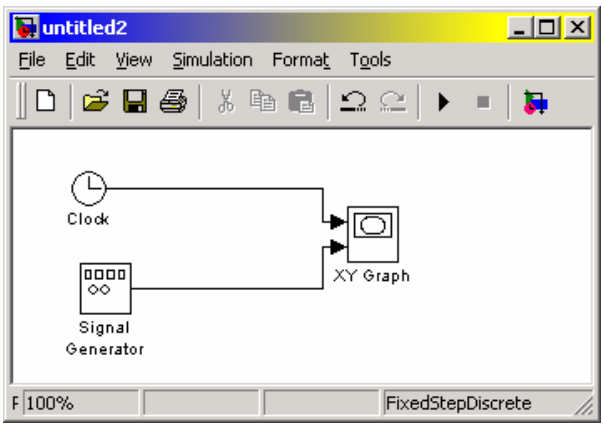

Рис. 7.19

На рис. 7.19 показана простейшая блок-схема S-модели, которая состоит из блока ***Signal Generator*** и блока отображения ***XY Graph***. Содержимое блока отображения после проведения моделирования при таких параметрах настраивания: вид колебаний - *Sine*; амплитуда - 1; частота - 1 *радиан/сек* отображено на рис. 7.10.

На рис. 7.20.. 7.22 представлены результаты, отображаемые в окне ***XYGraph*** в случае выбора соответственно прямоугольных, треугольных и случайных колебаний при тех же значениях остальных параметров настраивания.

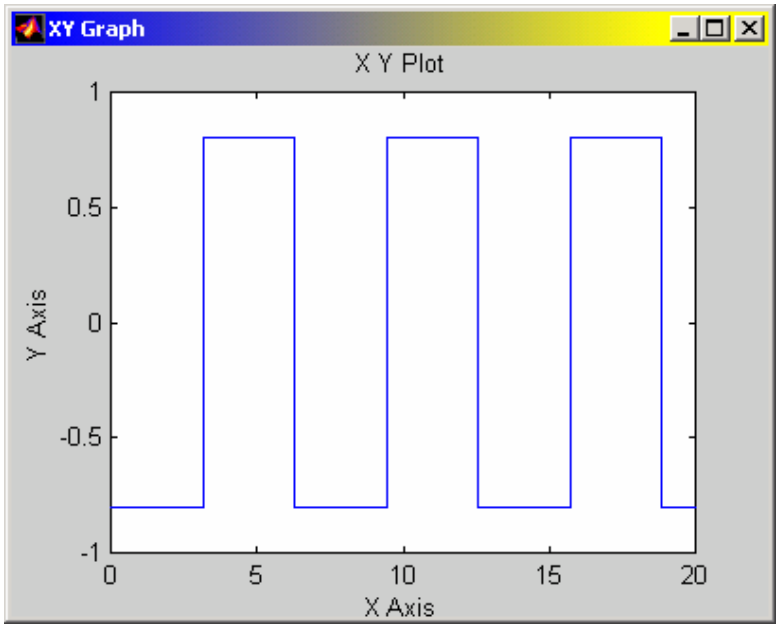

Рис. 7. 20

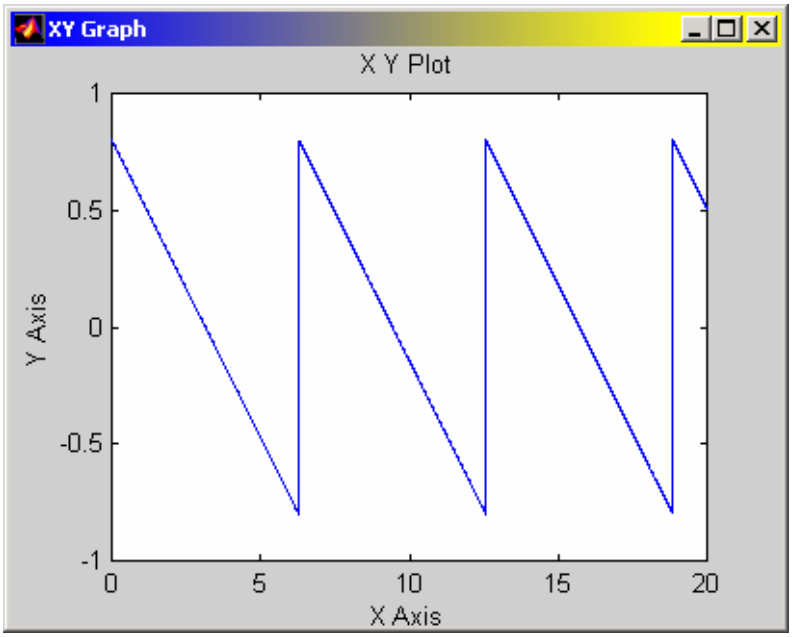

Рис. 7. 21

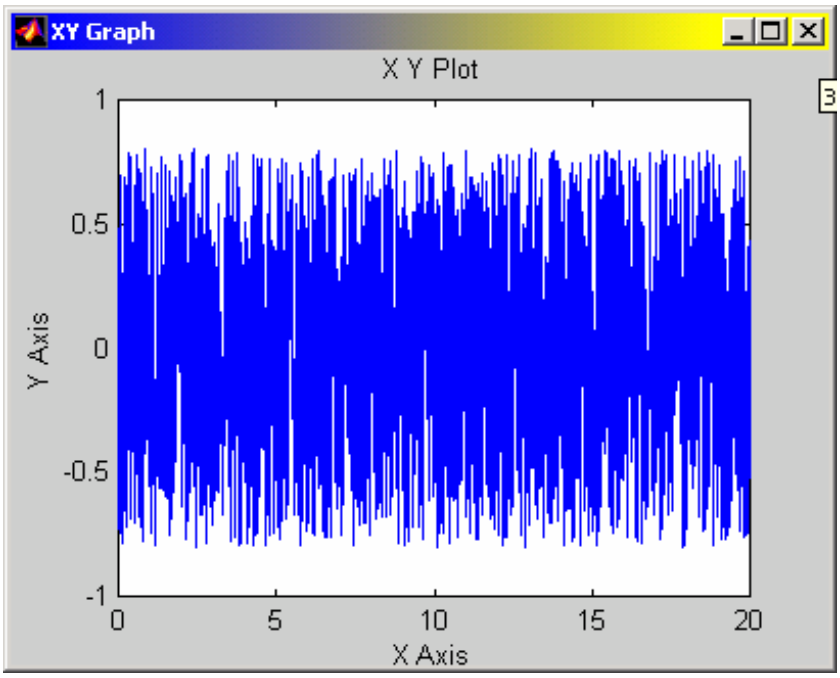

Рис. 7.22

*Блок **Step***

Блок обеспечивает формирование сигнала в форме ступеньки (или, как говорят, - ступенчатого сигнала).

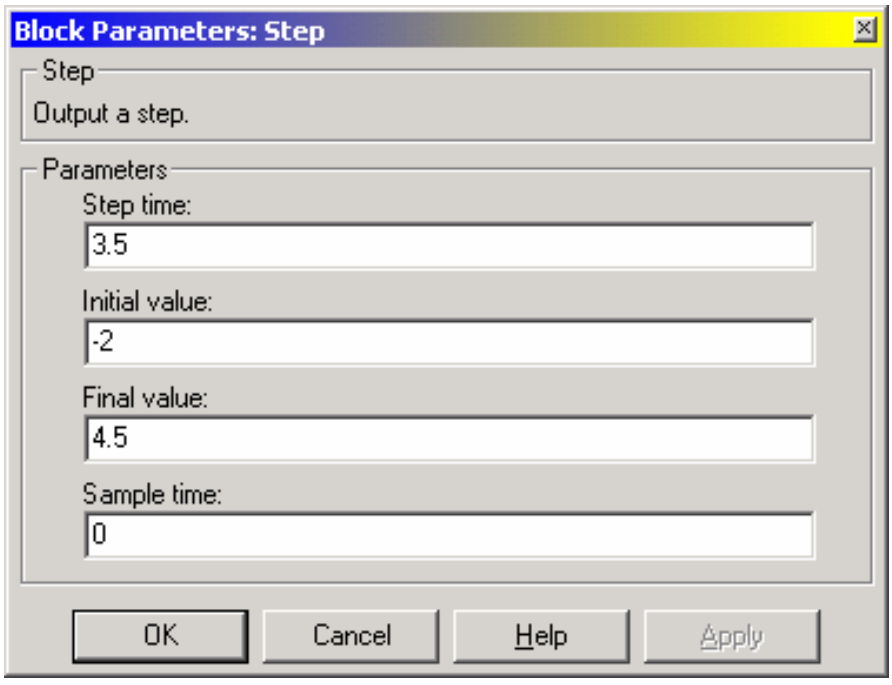

Рис. 7. 23

Блок имеет 3 параметра настраивания (рис. 7.23):

- *Step time* - время начала ступеньки (момент скачка сигнала) - определяет момент времени, в который происходит скачкообразное изменение величины сигнала; по умолчанию принимается равным 1;

- *Initial value* - начальное значение - задает уровень сигнала до скачка; значение по умолчанию - 0;
- *Final value* - конечное значение - задает уровень сигнала после скачка; значение его по умолчанию - 1.

Рассмотрим пример использования блока. Перетянем в окно блок-схемы из окна библиотеки источников блоки ***Step*** и ***Clock***, а из окна библиотеки приемников - блок ***XY Graph*** и соединим их (рис. 7.24)

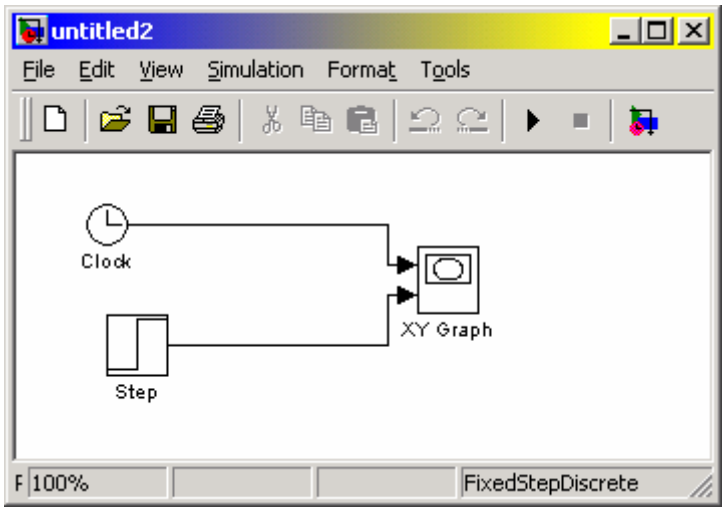

Рис. 7.24

Установим такие параметры настраивания блока: *Step time* - 3,5, *Initial value* - (-2), *Final value* – 4.5. После активизации моделирования (*Simulation/Start)* получим в окне ***Scope*** картину, представленную на рис. 7.25.

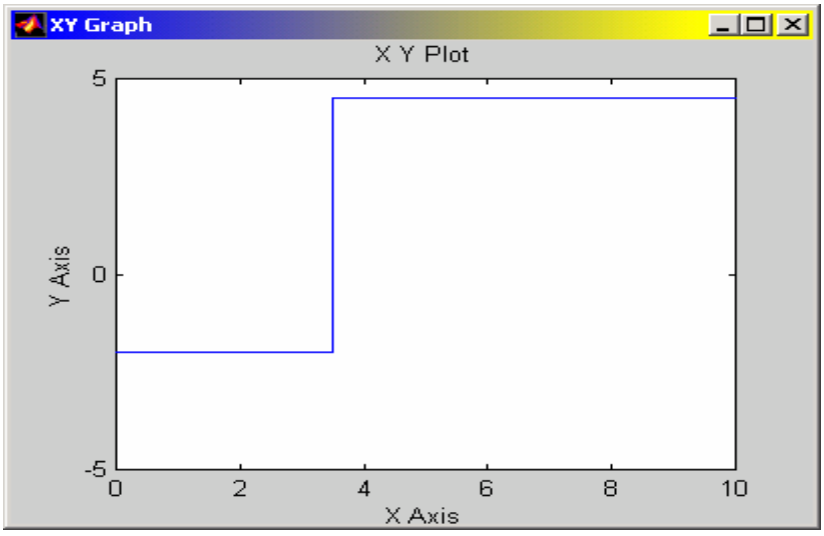

Рис. 7.25

*Блок **Ramp***

Блок формирует непрерывно нарастающий сигнал и имеет такие параметры настраивания (рис. 7.26):

- *Slope* - значение скорости нарастания сигнала;

- *Start time* - время начала нарастания сигнала;
- *Initial output* - значение сигнала до момента начала его нарастания.

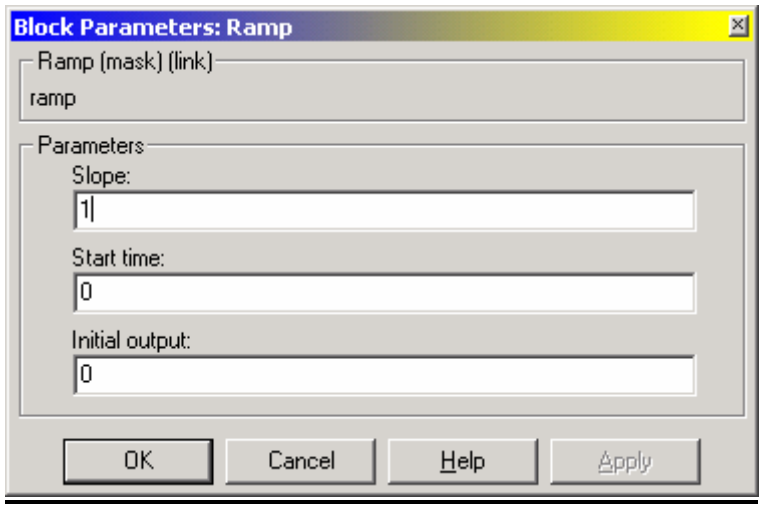

Рис. 7. 26

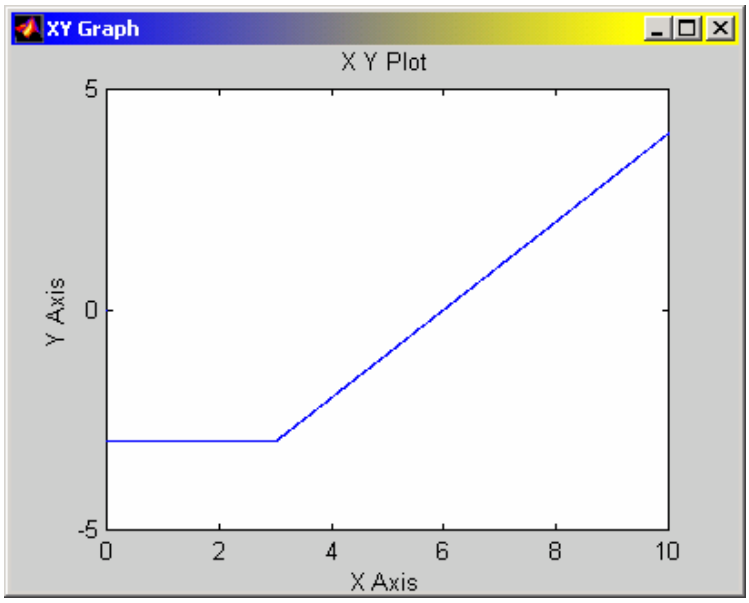

Рис. 7.27

На рис. 7.27 приведен пример результата применения блока ***Ramp*** при таких значениях параметров: *Slope* = 1; *Start time* = 3; *Initial output* = -3.

*Блок **Sine Wave***

Блок ***Sine Wave*** имеет такие настройки (рис. 7.28):

- *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 дискрета времени задания значений синусоидального сигнала.

Результат применения блока (при значениях параметров настраивания: амплитуда - 4,5; частота - 1 радиан в секунду и начальная фаза - π./2 радиан) показан на рис. 7.29.

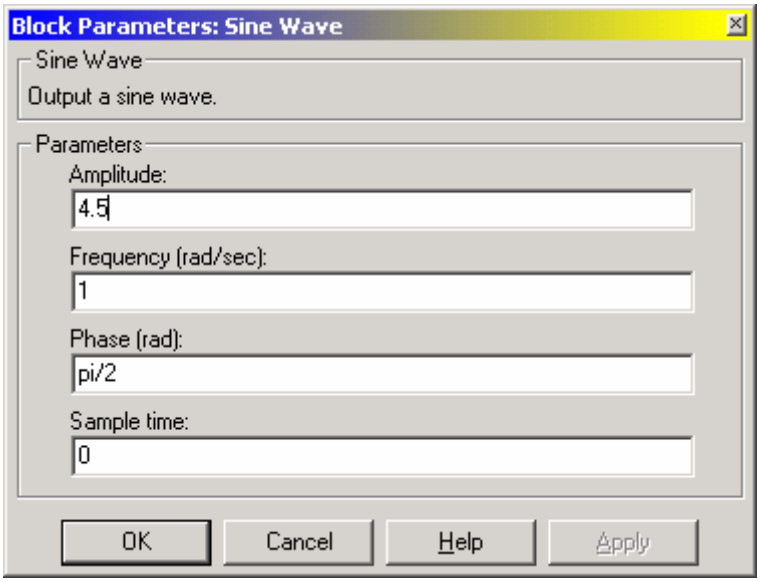

Рис. 7. 28

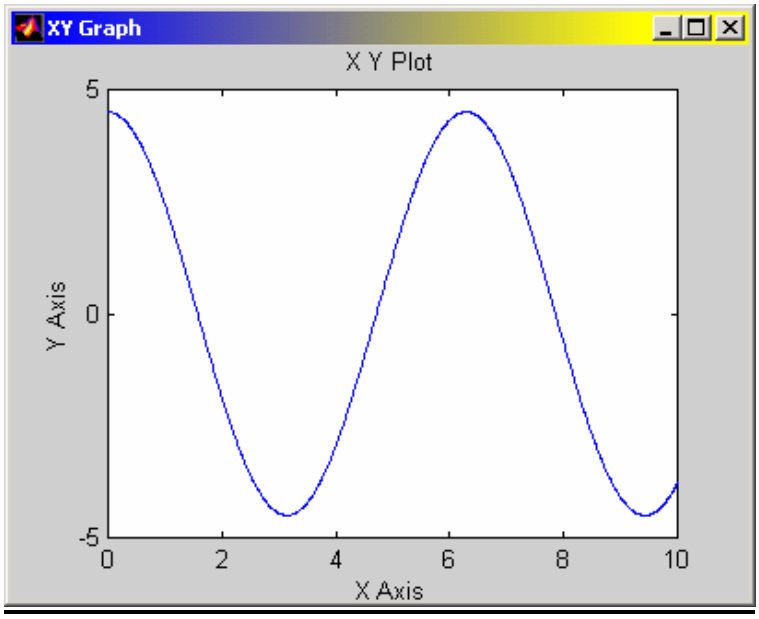

Рис. 7.29

Отличия этого блока от генератора синусоидальных колебаний в блоке ***Signal Generator*** состоят в следующем:

1) в анализируемом блоке можно устанавливать произвольную начальную фазу;
2) в нем нельзя задать частоту в Герцах.

*Блок **Repeating Sequence***

Этот блок содержит две настройки (рис. 7.30):

- *Time values* - вектор значений времени, в которые заданы значения исходной величины;
- *Output values* - вектор значений исходной величины, которые она должна принять в указанные в первом векторе соответствующие моменты времени.

Блок обеспечивает генерирование колебаний с периодом, равным различию между последним значением вектора *Time values* и значением первого его элемента. Форма волны внутри периода является ломаной, проходящей через точки с указанными в векторах *Time values* и *Output values* координатами.

В качестве примера на рис. 7.31 приведено изображение процесса, сгенерированного блоком ***Repeating Sequence*** при параметрах настраивания, указанных на рис. 7.30.

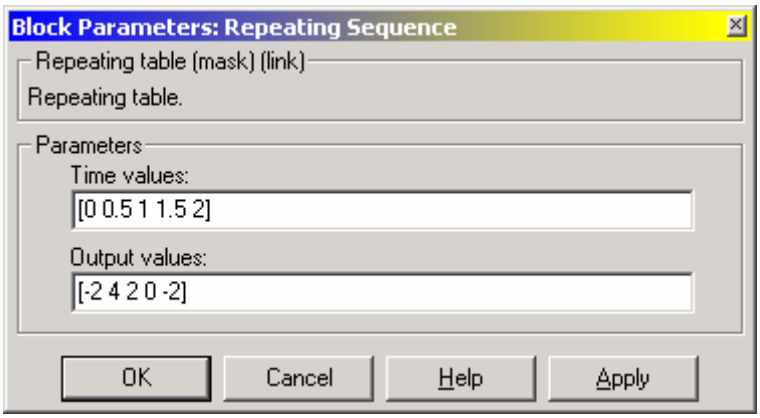

Рис. 7. 30

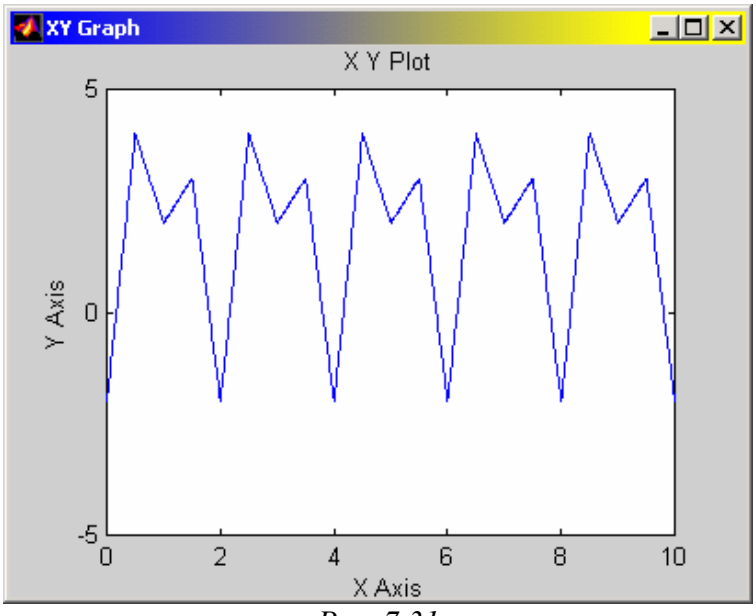

Рис. 7.31

Блок ***Diskret Pulse Generator***

Блок генерирует последовательности прямоугольных импульсов. В число настраиваемых параметров этого блока входят (рис. 7.32):

- амплитуда сигнала (*Amplitude*), т. е. высота прямоугольного импульса;

- размер периода сигнала (*Period*), который отсчитывается в целых числах - количестве дискретов времени;
- ширина импульса (*Pulse width*), тоже в дискретах времени;
- размер задержки импульса относительно *t*=0 (*Phase delay*) - в дискретах времени;
- размер дискрета времени (*Sample time*).

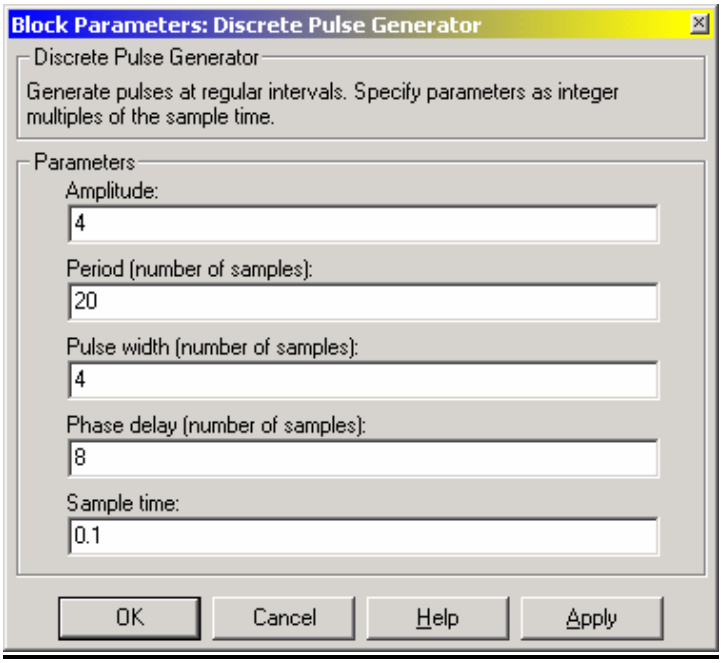

Рис. 7. 32

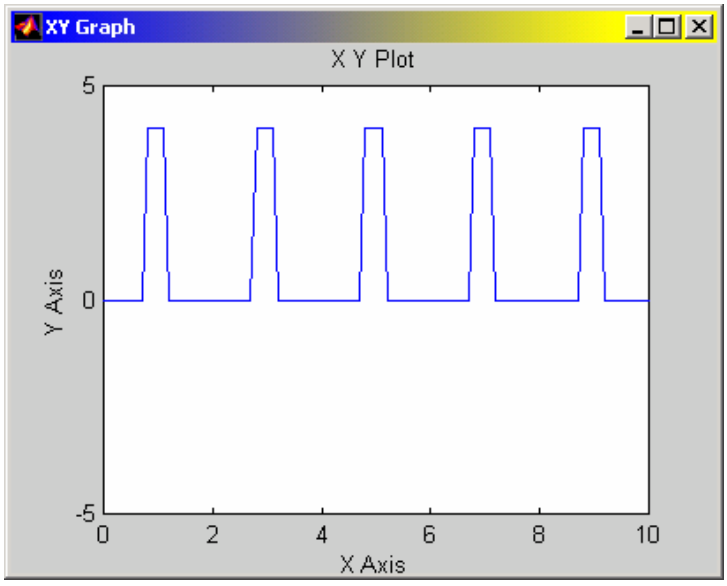

Рис. 7.33

Установление параметров этого блока целесообразно начинать с 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. 7.32, приведен на рис. 7.33.

*Блок **Pulse Generator***

Блок ***Pulse Generator*** осуществляет практически ту же функцию, что и блок ***Discret Pulse Generator***. Но параметры импульсов задаются иначе (рис. 7.34).

Здесь требуют задания такие параметры:

- *Period* - период прохождения импульсов;
- *Duty cycle (% of period)* - продолжительность прямоугольного импульса (в процентах от периода);
- *Amplitude* - амплитуда (высота) импульса;
- *Start time* - начальный момент времени первого из импульсов.

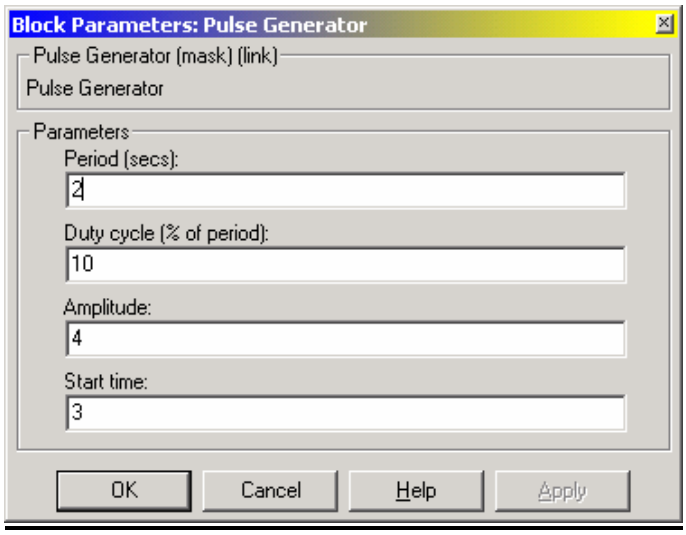

Рис. 7.34

Результат применения блока при значениях параметров, указанных на рис. 7.34, можно увидеть на рис. 7.35.

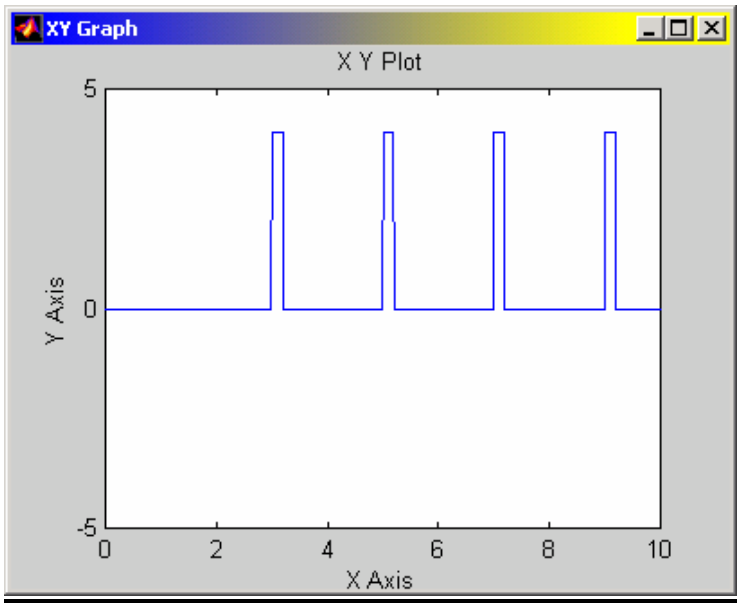

Рис. 7.35

Блок ***Chirp Signal***

Этот блок генерирует синусоидальный сигнал единичной амплитуды переменной частоты, причем значения частоты колебаний изменяется с течением времени по линейному закону. Соответственно этому в нем предусмотрены такие параметры настраивания (рис. 7.36):

- *Initial frequency (Hz)* - начальное значение (при t=0) частоты в герцах;
- *Target time (secs)* - второй (положительная величина) момент времени (в секундах);
- *Frequency at target time (Hz)* - значение частоты в этот второй момент времени.

Рис. 7.37 демонстрирует результат использования блока при параметрах, указанных на рис. 7.36.

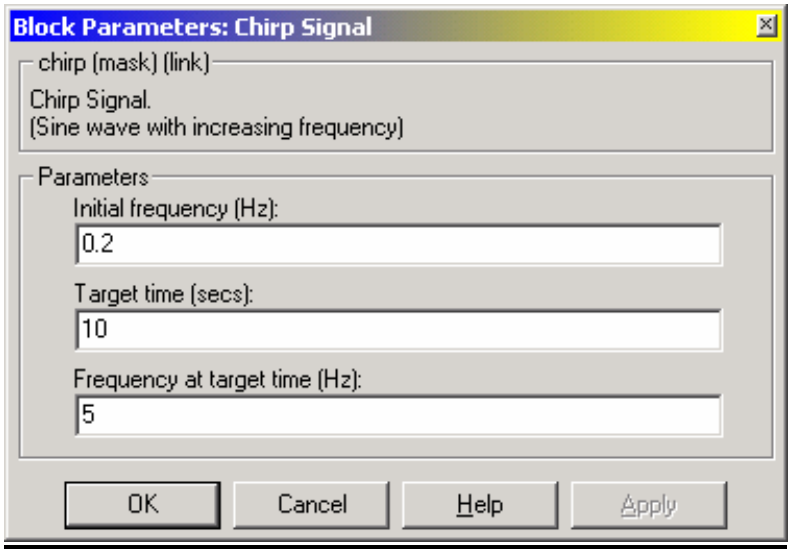

Рис. 7.36

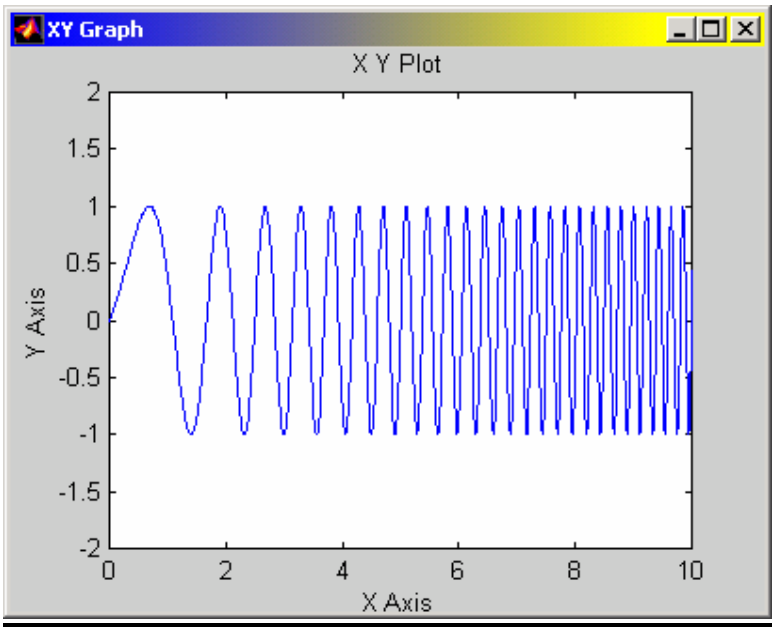

Рис. 7.37

Блок ***Random Number***

Блок ***Random Number*** обеспечивает формирование сигналов, значения которых в отдельные моменты времени являются случайной величиной, распределенной по нормальному (гауссовому) закону с заданными параметрами.

Блок имеет четыре параметра настраивания (рис. 7.38). Первые два - *Mean* и *Variance* - являются средним и среднеквадратичным отклонением 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*), как и ранее, задает величину дискрета времени. Рис. 7.39 показывает результат использования блока при значениях параметров, приведенных на рис. 7.38.

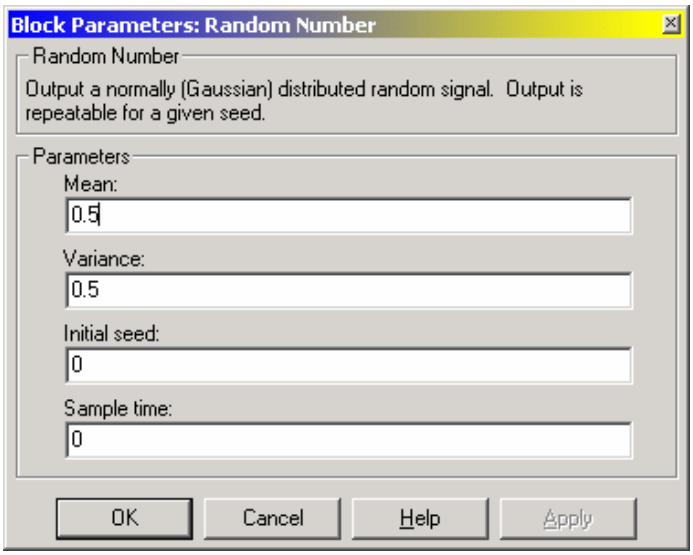

Рис. 7.38

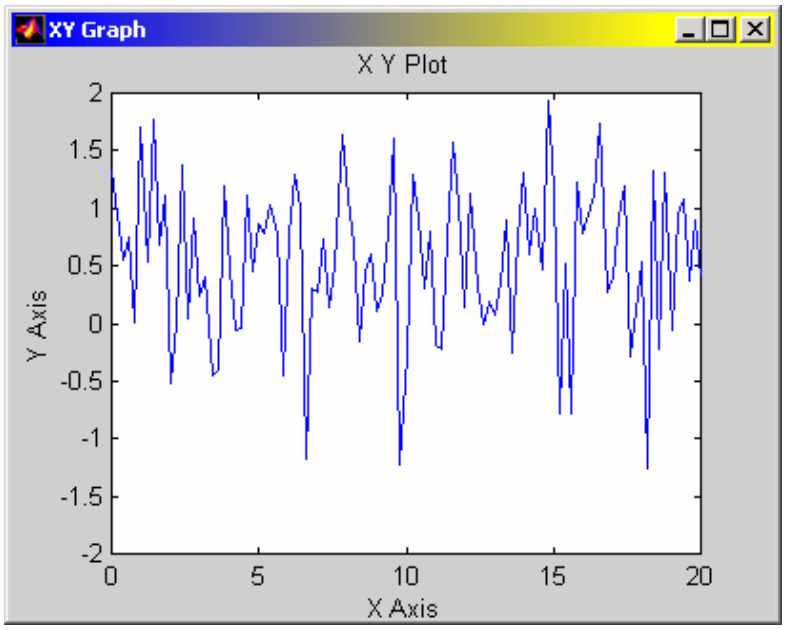

Рис. 7.39

Блок ***Uniform Random Number***

Этот блок формирует сигналы, значения которых в отдельные моменты времени являются случайной величиной, равномерно распределенной в заданном интервале. В число параметров настраивания блока входят (рис. 7.40):

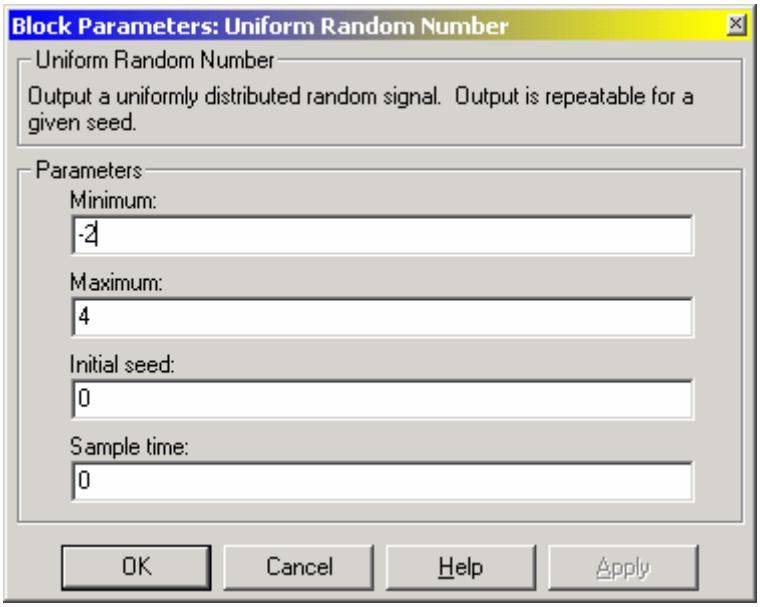

Рис. 7.40

- *Minimum* - нижний предел случайной величины;
- *Maximum* - верхний предел;
- *Initial seed* - начальное значение базы генератора случайных чисел;
- *Sample time* - дискрет времени.

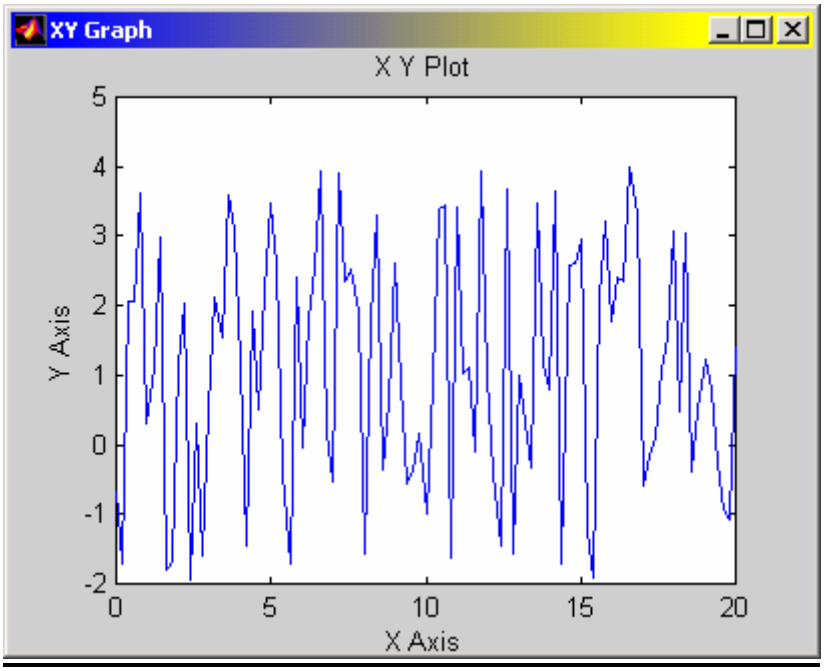

Рис. 7.41

Пример процесса, сгенерированного блоком по параметрам, указанным на рис. 7.40, приведен на рис. 7.41

Блок ***Band-Limited White Noise***

Этот блок формирует процесс в виде частотно-ограниченного белого шума. Параметры настраивания у него такие (рис. 7.42):

- *Noise power* - значения интенсивности белого шума;
- *Sample time* - значения дискрета времени (определяет верхнее значение частоты процесса);
- *Seed* - начальное значение базы генератора случайной величины.

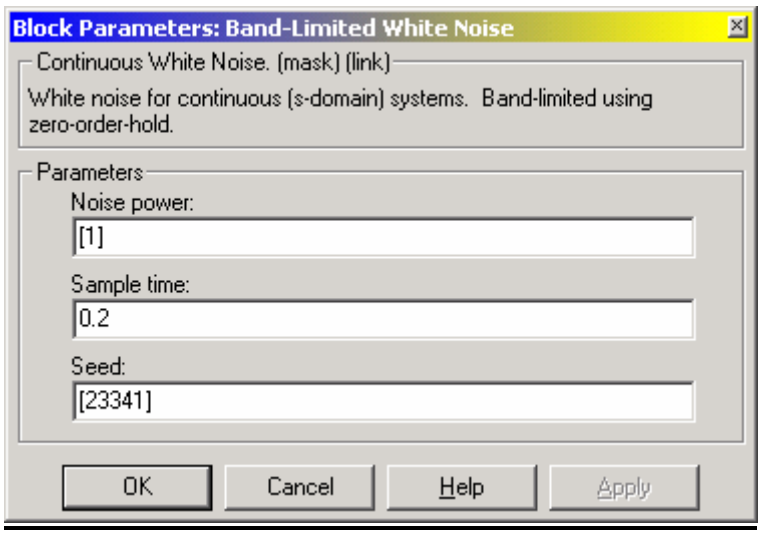

Рис. 7.42

Ниже (рис. 7.43) приведен пример реализации процесса с помощью блока ***Band-Limited White Noise*** при параметрах, указанных на рис. 7.42.

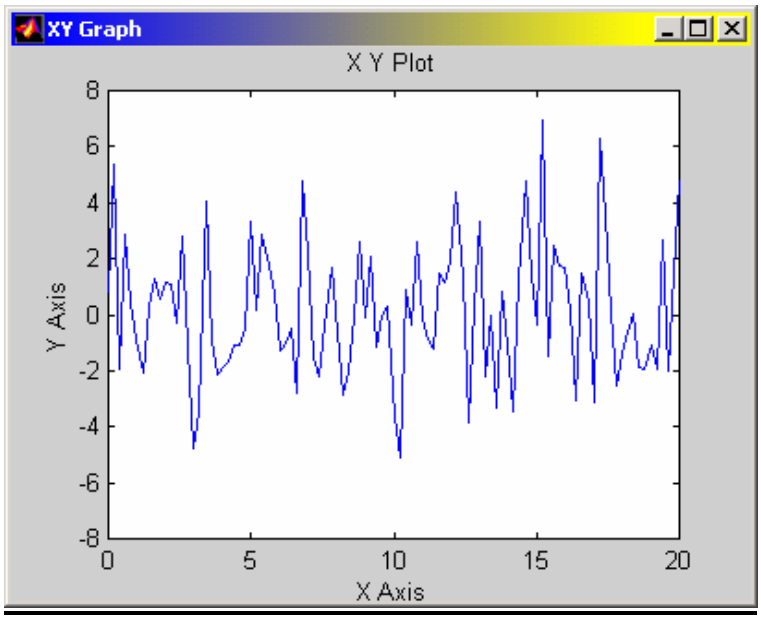

Рис. 7.43

7.1.5. Раздел Continuous

Этот раздел библиотеки содержит блоки (рис. 7.44):

- ***Integrator*** - идеальное интегрирующее звено (интегратор);
- ***Derivative*** - идеальное дифференцирующее звено;
- ***State-Space*** - определение линейного звена через задание четырех матриц его пространства состояний;
- ***Transfer Fcn*** - определение линейного звена через задание его передаточной функции;
- ***Zero-Pole*** - задание звена через указание векторов значений его полюсов и нулей, а также значения коэффициента передачи;
- ***Memory*** *(Память)* выполняет задержку сигнала на один шаг модельного времени;
- ***Transport Delay*** обеспечивает задержку сигнала на заданное количество шагов модельного времени, причем необязательно целое;
- ***Variable Transport Delay*** позволяет задавать управляемую извне величину задержки.

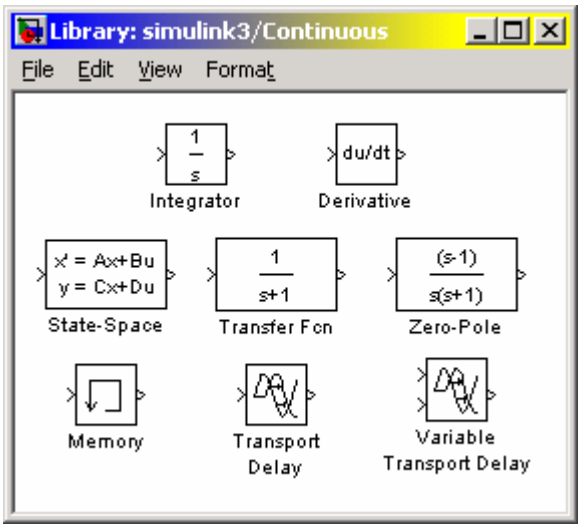

Рис. 7.44

Блок ***Integrator*** осуществляет интегрирование в непрерывном времени входной величины. Он имеет такие параметры настраивания (рис. 7.45):

- подключение дополнительного управляющего сигнала (*External reset*);
- определение источника (внутренний или внешний) установления начального значения выходного сигнала (*Initial condition source*);
- начальное значение выходной величины (*Initial condition*); значение вводится в строке редактирования или как числовая константа, или в виде выражения, которое вычисляется;
- флажок *Limit output* (*Ограничения исходного значения)* определяет, будут ли использоваться следующие 3 параметры настраивания;
- верхнее предельное значение выходной величины (*Upper saturation limit*); по умолчанию - не ограничено (*inf*);

- нижнее предельное значение выходной величины (*Lower saturation limit*); по умолчанию параметр имеет значения (*-inf*);
- флажок *Показать порт насыщения* (*Show saturation port*);
- флажок *Показать порт состояния* (*Show state port*);
- допустимая предельная величина абсолютной погрешности (*Absolute tolerance*).

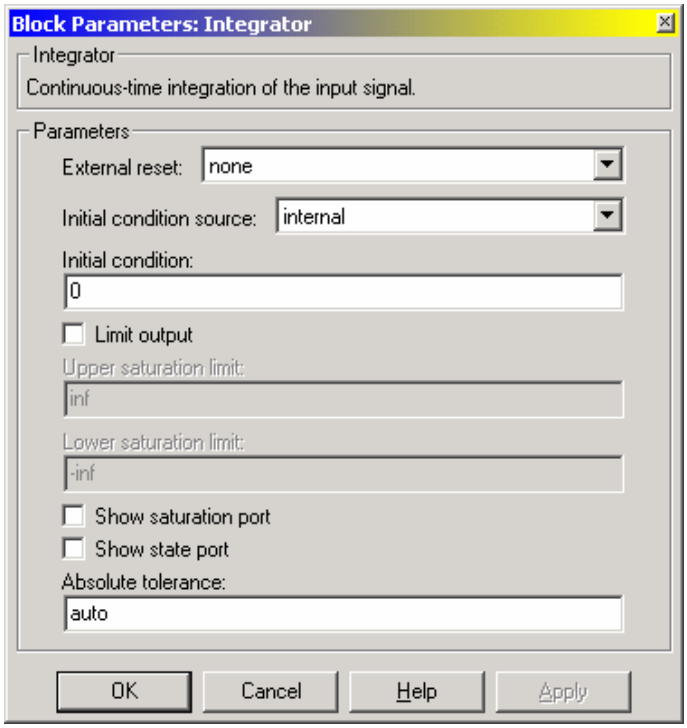

Рис. 7.45

Параметр *External reset* может принимать такие значения (см. его выпадающее меню):

none - дополнительный управляющий сигнал не используется;

rising - для управления используется нарастающий сигнал;

falling - для управления используется убывающий сигнал;

either - на работу блока влияет изменение управляющего сигнала в любом направлении.

Параметр *Initial condition source* принимает одно из двух значений:

internal - используется внутреннее установка начального значения выходной величины;

external - установления начальных условий будет осуществляться извне.

Если выбранные пользователем значения этих двух параметров предполагают наличие дополнительных входных сигналов, то на графическом изображении блока появляются дополнительные входные порты (после нажатия кнопки *Apply* в окне настраивания блока). Если флажок *Limit output* установлен, то при переходе выходного значения интегратора через верхнюю или нижнюю границу на дополнительном выходе блока (*saturation port*) формируется единичный сигнал. Чтобы этот сигнал можно было использовать для управления

работой S-модели, флажок *Show saturation port* должен быть включен. При этом на графическом изображении блока появляется обозначение нового выходного порта на правой стороне изображения блока-интегратора. Установление флажка *Show state port* также приводит к появлению дополнительного выхода *state port* блока (он возникает, конечно, на нижней стороне изображения блока). Сигнал, который подается на этот порт, совпадает с главным выходным сигналом, но, в отличие от него, может быть использован только для прерывания алгебраического цикла или для согласования состояния подсистем модели.

Использование блоков-звеньев ***State-Space***, ***Transfer Fcn*** и ***Zero-Pole*** является достаточно ясным для тех, кто знаком с основами теории автоматического управления.

Блок ***Memory*** *(Память)* выполняет задержку сигнала только на один шаг модельного времени. Блок имеет два параметра настраивания: *Initial condition (Начальное условие)* задает значения входного сигнала в начальный момент времени; флажок *Inherit sample time (Наследование шага времени)* позволяет выбрать величину промежутка времени, на который будет осуществляться задержка сигнала:

- если флажок снят, то используется минимальная задержка, равная 0,1 единицы модельного времени;
- если флажок установлен, то величина задержки равна значению дискрету времени блока, который предшествует блоку ***Memory.***

Блок ***Transport Delay*** обеспечивает задержку сигнала на заданное количество шагов модельного времени, причем необязательно целое. Настраивание блока происходит по трех параметрам:

Time delay (Время задержки) - количество шагов модельного времени, на который следует задержать сигнал; может вводиться или в числовой форме, или в форме выражения, которое вычисляется;

Initial input (Начальное значение входа) - по умолчанию равняется 0;

Initial buffer size (Начальный размер буфера) - объем памяти (в байтах), который выделяется в рабочем пространстве MatLAB для сохранения параметров задержанного сигнала; должен быть кратной до 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 продолжительность задержки.

7.1.6. Раздел Discrete

Ранее рассмотренные разделы библиотеки позволяют формировать непрерывную динамическую систему. Раздел ***Discrete*** содержит элементы (блоки), присущие только дискретным системам, а также те, которые превращают непрерывную систему в дискретную (рис. 7.46):

- ***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 линейного звена матрицами его состояния;
- ***Discrete Filter*** - блок задания дискретного звена через дискретную передаточную дробно-рациональную функцию относительно 1/z;
- ***Discrete Transfer Fcn*** - блок задания линейного 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.

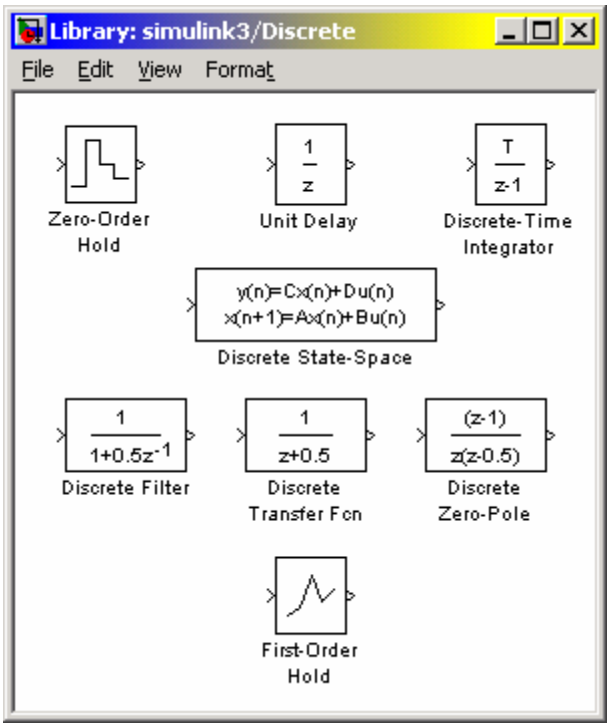

Рис. 7.46

Блок ***Unit Delay*** обеспечивает задержку входного сигнала на заданное число шагов модельного времени. Параметрами настраивания для этого блока являются начальное значение сигнала (*Initial condition)* и время задержки *(Sample time)*, которое задается количеством шагов модельного времени.

Блок ***Discrete-Time Integrator*** выполняет численное интегрирование входного сигнала. Большинство параметров настраивания 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 ниспадающего меню можно выбрать один из трех методов: прямой метод Ейлера (левых прямоугольников); обратный метод Ейлера (правых прямоугольников); метод трапеций. Второе отличие - вместо параметра *Absolute tolerance* введен параметр

Sample time, который задает шаг интегрирования в единицах шагов модельного времени.

7.1.7. Раздел Math

В этом разделе содержатся блоки, которые реализуют некоторые встроенные математические функции системы MatLAB (рис. 7.47):

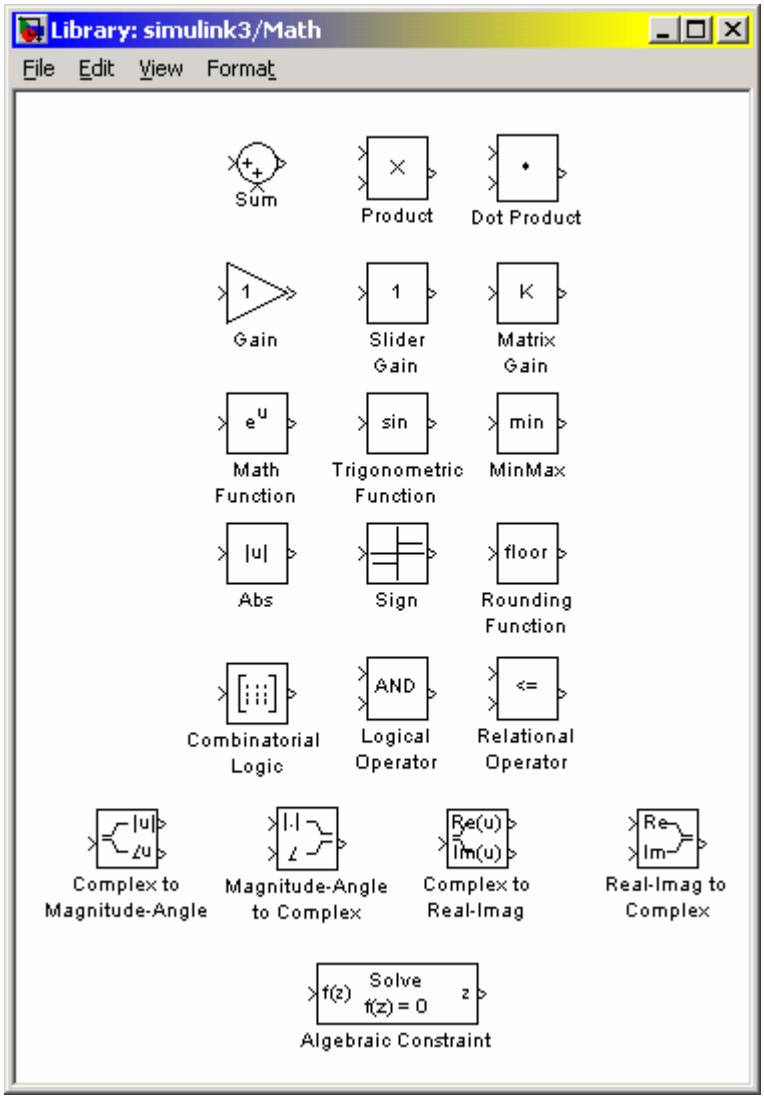

Рис. 7.47

- ***Sum***, который осуществляет суммирование сигналов, поступающих в него;
- ***Product***, выполняющий умножение или деление входных сигналов;
- ***Dot Product*** - блок, осуществляющий перемножение двух входных величин, если они являются скалярами, или определяющий сумму поэлементных произведений элементов двух входных векторов (одинаковой длины);
- ***Gain*** - линейное усилительное звено;
- ***Slider Gain*** - звено интерактивного измения коэффициента усиления;

- ***Matrix Gain*** - матричное усилительное звено для многомерной системы;
- шесть блоков математических стандартных операций (***MathFunction***, ***TrigonometricFunction***, ***MinMax***, ***Abs***, ***Sign*** и ***Rounding function***);
- три блока логических операций (***Combinatorial Logic***, ***LogicalOperator*** и ***Relation Operator***);
- четыре блока трансформирования комплексных сигналов в действительные и наоборот (***Complex to Magnitude-Angle, Complex to Real-Imag, Magnitude-Angle to Complex*** и ***Real-Imag to Complex***);
- ***AlgebraicConstraint*** – блок решения алгебраических уравнений.

Блок ***Sum*** может использоваться в двух режимах:

- суммирования входных сигналов (в том числе с разными знаками);
- суммирования элементов вектора, который поступает на вход блока.

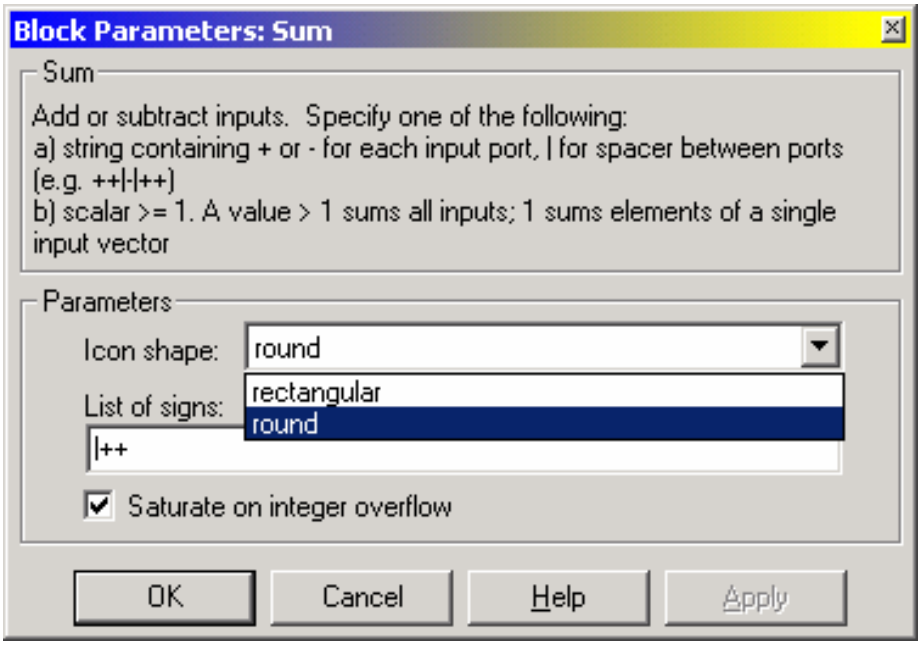

Рис. 7.48

Для управления режимами работы блока используется два параметра настраивания – *Icon Shape* (Форма изображения) и *List of signs* (Список знаков) (рис. 7.48). Первый может принимать два значения – *round* (круглый) и *rectangular* (прямоугольный). Значения второго параметра могут задаваться одним из трех способов:

- в виде последовательности знаков "+" или "-"; при этом *количество знаков определяет количество входов блока, а самый знак - полярность соответствующего входного сигнала*;
- в виде целой положительной и больше 1 константы; *значения этой константы определяет количество входов блока, а все входы считаются положительными*;
- в виде символа "1", который указывает, что *блок используется в втором режиме*

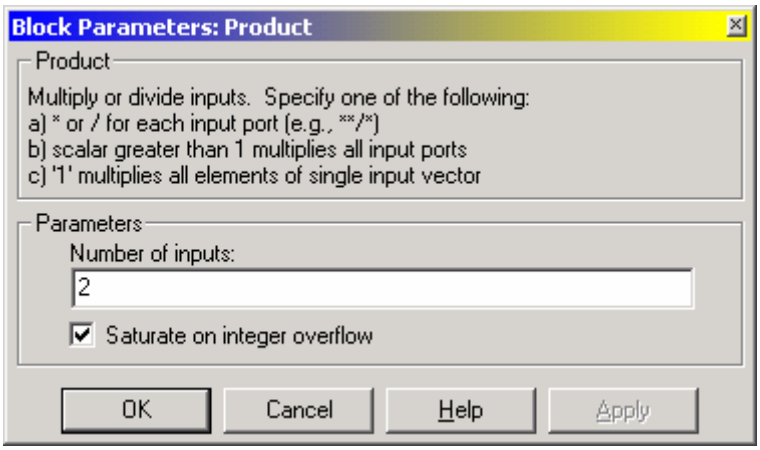

Рис. 7.49

Блок ***Product*** выполняет умножение или деление нескольких входных сигналов. В параметры настраивания входят количество входов блока и вид выполняемой операции. Окно настраивания блока содержит лишь один параметр (рис. 7.49) – *Number of inputs* (количество входов). Если этот параметр (а, значит, количество входов блока) – положительное число, большее 1, то все входные величины перемножаются. Если в качестве значения параметра настраивания блока ввести "1", будет вычисляться произведение элементов входного вектора. При этом на изображении блока выводится символ ***P.*** В случае, когда результат выполнения должен содержать деления на некоторые входные величины, в окошко *Number of inputs* следует вводить последовательность символов '*' или '/' по числу входов блока в соответствии с тем умножается или делится результат на соответствующую входную величину. Задание значений этих параметров аналогично настраиванию блока ***Sum*** из библиотеки ***Linear.***

Блок ***DotProduct*** имеет лишь два входа и не имеет параметров настраивания. Его входные сигналы должны быть векторами одинаковой длины. Выходная величина блока в каждый момент времени равна сумме произведений соответствующих элементов этих двух векторов.

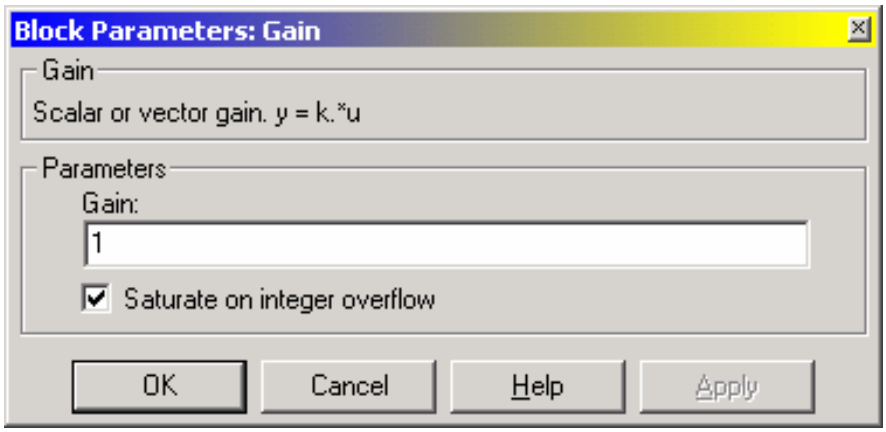

Рис. 7.50

Блок ***Gain*** осуществляет умножение входного сигнала на постоянную величину, значения которой задается в окне настраивания (рис. 7.50). В случае,

когда входной сигнал является вектором длиной N элементов, коэффициент усиления должен быть вектором той же длины.

Блок ***Slider Gain*** является одним из элементов взаимодействия пользователя с моделью. Он позволяет в удобной диалоговой форме изменять значение некоторого параметра в процессе моделирования. Блок становится активным после того, как будет перемещен в окно блок-схемы создаваемой модели. Чтобы отворить окно с "ползунковым" регулятором (рис. 7.51), необходимо дважды щелкнуть мышью на изображении блока.

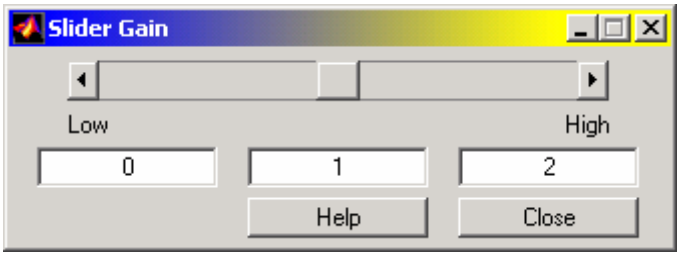

Рис. 7.51

Окно ***Slider Gain*** имеет три поля ввода информации:

- для указания нижней границы изменения параметра (*Low*);
- для указания верхней границы изменения параметра (*High*);
- для указания текущего значения.

Текущее значение должно располагаться внутри диапазона [*Low, High*]. Тем не менее при выборе нового диапазона необходимо сначала указать новое значение параметра, а потом изменить границы диапазона.

Блок ***Matrix Gain*** осуществляет перемножение входного вектора на постоянную матрицу, указанную в окне настраивания (рис. 7.52). При этом количество строк матрицы усиления должно совпадать с длиной входного вектора. На выходе одержується векторный сигнал длиной, которая равна количеству столбцов матрицы усиления.

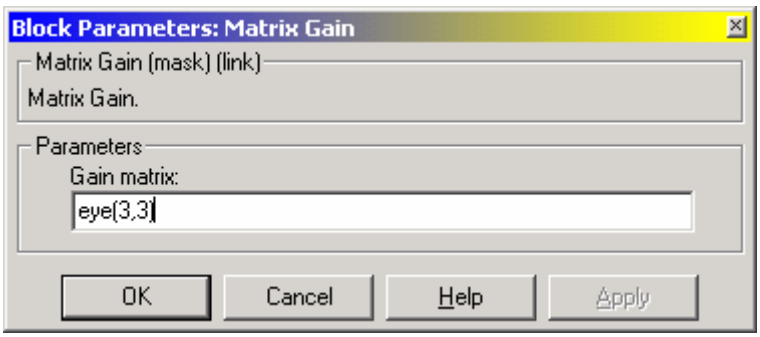

Рис. 7.52

Ниже приводятся особенности той части блоков, которая реализует математические функции.

Блок ***Abs*** формирует абсолютное значение входного сигнала. Он не имеет параметров настраивания. Блок ***Trigonometric Function*** обеспечивает

преобразования входного сигнала с помощью одной из таких функций MatLAB: ***sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh.*** Избрание необходимой функции осуществляется в окне настраивания блока с помощью ниспадающего меню. Блок ***Math Function*** позволяет выбрать для преобразования входного сигнала элементарные не тригонометрические и не гиперболические функции, такие как ***exp***, ***log***, ***10^u***, ***log10***, ***magnitude^***2, ***square***, ***sqrt***, ***pow***, ***reciprocal***, ***hypot***, ***rem***, ***mod***. Нужная функция выбирается с помощью выпадающего меню в окне настраивания. Блок ***Rounding Function*** содержит разнообразные функции округления, предусмотренные в MatLAB. Он осуществляет округление значений входного сигнала. Выбор конкретного метода округления осуществляется также с помощью выпадающего меню в окне настраивания. Блок ***MinMax*** осуществляет поиск минимального или максимального элемента входного вектора. Если входом является скалярная величина, то выходная величина совпадает с входной. Если входов несколько, ищется минимум или максимум среди входов. В число настроек входит выбор метода и количество входов блока. Блок ***Sign*** реализует нелинейность типа сигнум-функції. В нем нет параметров настраивания. Блок формирует исходный сигнал, который принимает только три возможных значения: "+1" - в случае, когда входной сигнал положителен, "-1" - при отрицательном входном сигнале и "0" при входном сигнале, равном нулю.

Для указанных выше блоков имя выбранной функции выводится на графическом изображении блока.

Следующую группу образуют блоки, которые обеспечивают логическую обработку входного сигнала. Общим для всех их является то, что выходная величина в них является булевой, то есть может достигать лишь двух значений: "1" ("истина") или "0" ("ложь"). Во многих из них булевыми должны быть и все входные величины.

Блок ***Relational Operator*** реализует операции отношения между двумя входными сигналами >, <, ≤, ≥, ==, ~= (соответственно: больше, меньше, меньше или равно, больше или равно, тождественно равно, не равно). Конкретная операция выбирается при настраивании параметров блока с помощью выпадающего меню. Знак операции выводится на изображении блока.

Блок ***Logical Operator*** содержит набор основных логических операций. Входные величины должны быть булевими. Выбор необходимой логической операции осуществляется в окне настраивания блока с помощью выпадающего меню. Вторым параметром настраивания является количество входных величин (портов) блока (*Number of input ports*), то есть количество аргументов логической операции.

Блок ***Combinatorial Logic*** обеспечивает преобразование входных булевых величин в выходную в соответствия с заданной таблицей истинности. Блок имеет единственный параметр настраивания - *Truth table* (таблица истинности).

Четыре следующих блока осуществляют преобразование комплексного входного сигнала в один или два действительных выходных сигнала, которые являются модулем, аргументом, действительной или мнимой частью входного сигнала (блоки

Complex to Magnitude-Angle и ***Complex to Real-Imag***), а также один или два входных действительных сигнала в комплексный выходный сигнал (блоки ***Magnitude-Angle to Complex*** и ***Real-Imag to Complex***). Количество входов или выходов определяется в окне настраивания блока.

7.1.8. Раздел Functions & Tables

В этот раздел входят две группы блоков (рис. 7.53):

- два блока, осуществляющие линейную интерполяцию значений выходного сигнала по значениям входного сигнала, которые поступают в блок, в соответствии с заданной в окне настраивания таблицы соответствий;

- три блока, которые являются заготовками, дающими пользователю возможность разработать собственные блоки программы, выполняющие необходимые ему преобразования сигналов.

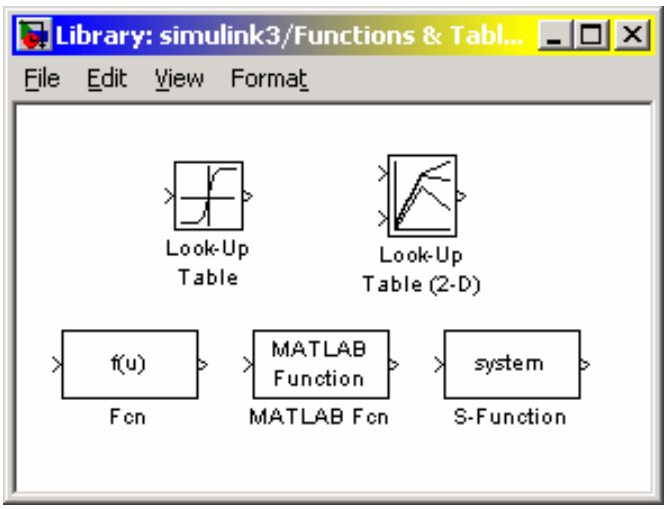

Рис. 7.53

Блок ***Look-upTable*** выполняет линейную интерполяцию входного сигнала в соответствии с табличной функцией, которая задается. Блок ***Look-upTable(2D)*** осуществляет двумерную линейную интерполяцию двух входных сигналов.

Блок ***Fcn*** позволяет пользователю ввести любую скалярную функцию от одного (скалярного или векторного) аргумента, которая выражается через стандартные функции MatLAB. Выражение функции вводится в окне настраивания блока. Для обозначения входного сигнала (аргумента функции) используется символ ***u***.

Блок ***MATLAB Fcn*** позволяет применить к входному сигналу любую подпрограмму обработки, реализованную в виде М-файла. В отличие от предыдущего блока, здесь к числу параметров настраивания добавлен параметр *Output width (Ширина выходного сигнала)*, который определяет количество элементов выходного вектора. Применение этого блока приводит к значительной потере быстродействия в выполнении (в сравнении с предыдущим блоком).

С помощью блока ***S-function*** пользователь может реализовать в виде Simulink-блока любую программу обработки входного сигнала, включая

интегрирование дифференциальных уравнений и обработку дискретных во времени сигналов. Более подробно работа с ***S-function*** изложена в п. 7.4.6.

7.1.9. Раздел Nonlinear

Состав этого раздела приведен на рис. 7.54.

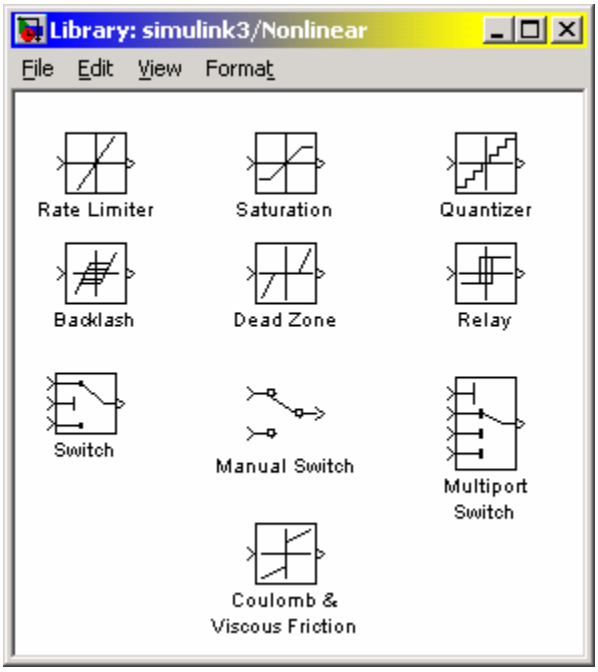

Рис. 7.54

Здесь расположены 6 блоков, которые реализуют некоторые типовые нелинейные (кусочно-линейные) зависимости выходной величины от входной и 3 блока-переключателя.

Блок ***RateLimiter*** (Ограничитель скорости) обеспечивает ограничивание сверху и снизу скорости изменения сигнала, проходящего через него. Окно настраивания содержит два параметра *Rising slew rate* и *Falling slew rate*. Блок работает по следующему алгоритму. Сначала рассчитывается скорость изменения сигнала, который проходит через блок по формуле

$$rate = \frac{u(i) - y(i-1)}{t(i) - t(i-1)},$$

где $u(i)$ - значения входного сигнала в момент времени $t(i)$; $y(i-1)$ - значения выходного сигнала в момент $t(i-1)$. Далее, если вычисленное значение *rate* больше, чем *Rising slew rate* (*R*), *вы*ходная величина определяется по формуле

$$y(i) = y(i-1) + R \cdot \Delta t;$$

если *rate* меньше, чем *Falling slew rate* (*F*), то выход определяется так

$$y(i) = y(i-1) + F \cdot \Delta t.$$

Если же значения *rate* содержится между *R* и *F*, то исходная величина совпадает с входной

$$y(i) = u(i).$$

Блок ***Saturation*** (*Насыщение*) реализует линейную зависимость с насыщением (ограничением). Выходная величина этого блока совпадает со входной, если последняя находится внутри указанного диапазона. Если же входная величина выходит за рамки диапазона, то выходный сигнал принимает значение ближайшей из границ. Значения границ диапазона устанавливаются в окне настраивания блока.

Блок ***Quantizer*** *(Квантователь)* осуществляет дискретизацию входного сигнала по величине. Единственным параметром настраивания этого блока является *Quantization interval* - *Интервал квантования* - размер дискрета по уровню входного сигнала.

Блок ***BackLash*** *(Люфт)* реализует нелинейность типа зазора. В нем предусмотрено два параметра настраивания: *Deadband width* - величина люфта и *Initial output* - начальное значение выходной величины.

Блок ***Dead Zone*** *(Мертвая зона)* реализует нелинейность типа зоны нечувствительности. Параметров настраивания здесь два - начало и конец зоны нечувствительности.

Блок ***Relay*** *(Реле)* работает по аналогии с обычным реле: если входной сигнал превышает некоторое предельное значение, то на выходе блока формируется некоторый постоянный сигнал. Блок имеет 4 параметра настраивания (рис. 7.55):

- *Switch on point (Точка включения)* - задает предельное значение, при превышении которого происходит включение реле;
- *Switch off point (Точка выключения)* - определяет уровень входного сигнала, ниже которого 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.

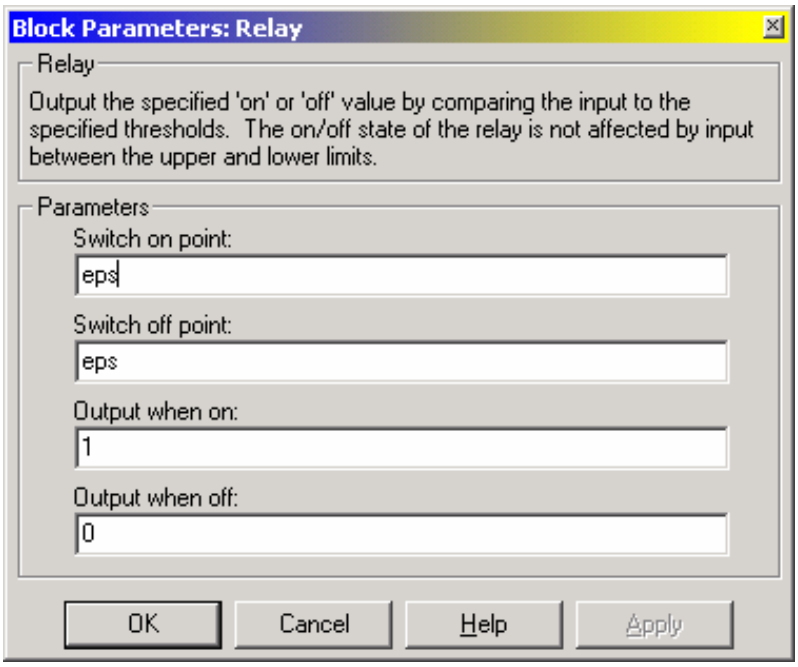

Рис. 7.55

Блок ***Coulomb & Viscous Friction*** *(Кулоново и вязкое трение)* реализует нелинейную зависимость типа "линейная с натягом". Если вход положителен, то выход пропорционален входу с коэффициентом пропорциональности - "коэффициентом вязкого трения" - и увеличен на величину "натяга" ("кулонова, сухого трения"). Если вход отрицателен, то выход также пропорционален входу (с тем же коэффициентом пропорциональности) за вычетом величины "натяга". При входе равном нулю выход тоже равняется нулю. В параметры настраивания блока входят величины "кулонова трения" ("натяга") и коэффициента "вязкого трения".

В завершение рассмотрим группу блоков-переключателей, которые руководят направлением передачи сигнала. Таких блоков три: ***Switch*** *(Переключатель)*, ***Manual Switch*** *(Ручной переключатель)* и ***Multiport Switch*** *(Многовходовый переключатель)*.

Блок ***Switch*** имеет три вход: (два (1-й и 3-й) информационные и один (2-й) – управляющий) и один выход. Если величина управляющего сигнала, который поступает на 2-й вход, не меньше некоторого заданного граничного 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 настраивания. У него два входа и один выход. На изображении блока показано перемычкой, какой именно из двух входов подключен к выходу. Блок позволяет "вручную" переключать входы. Для этого необходимо дважды щелкнуть мышкой на изображении блока. При этом изменится и изображение блока - на нем выход уже будет соединен перемычкой с другим входом.

Блок ***Multiport Switch*** имеет не меньше трех входов. Первый (верхний) из них является управляющим, другие - информационными. Блок имеет один

параметр настраивания *Number of inputs (Количество входов)*, который определяет количество информационных входов. Номер входа, который соединяется с выходом, равняется значению управляющего сигнала, который поступает на верхний вход. Если это значение является дробным числом, то оно округляется до целого по обычным правилам. Если оно меньше единицы, то оно считается равным 1; если оно больше количества информационных входов, то оно принимается равным наибольшему номеру (входы нумеруются сверху вниз, кроме самого верхнего - управляющего).

7.1.10. Раздел Signals & Systems

Раздел библиотеки ***Signals & Systems*** (сигналы и системы) предназначен для построения сложных S-моделей, которые состоят из других моделей более низкого уровня. Состав раздела приведен на рис. 7.56.

Блоки ***In*** *(Входной порт)* и ***Out*** *(Исходный порт)* обеспечивают информационную связь между подсистемами модели и с рабочим пространством системы MatLAB.

Блок ***Mux*** *(Мультиплексор)* выполняет объединение входных величин в единый выходной вектор. При этом входные величины могут быть как скалярными, так и векторными. Длина результирующего вектора равняется сумме длин всех векторов. Порядок элементов в векторе выхода определяется порядком входов (сверху вниз) и порядком расположения элементов внутри каждого входа. Блок имеет один параметр настраивания - *Number of inputs (Количество входов)*.

Блок ***Demux*** *(Разделитель, Демультиплексор)* выполняет обратную функцию - *разделяет входной вектор на заданное количество компонентов*. Он также имеет единственный параметр настраивания *Number of outputs (Количество выходов)*. В случае, когда указанное число выходов (N) задается меньшим длины входного вектора (M), блок формирует исходные векторы следующим образом. Первые (N-1) выходов будут векторами одинаковой длины, равной целой части отношение M/(N-1). Последний выход будет иметь длину, равную остатку от деления.

Блоки ***From*** *(Принять от)*, ***Goto Tag Visibility*** *(Признак видимости)* и ***Goto*** *(Передать к)* используются совместно и предназначены для обмена данными между разнообразными частями S-модели с учетом досягаемости (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 сохранение на протяжении моделирования.

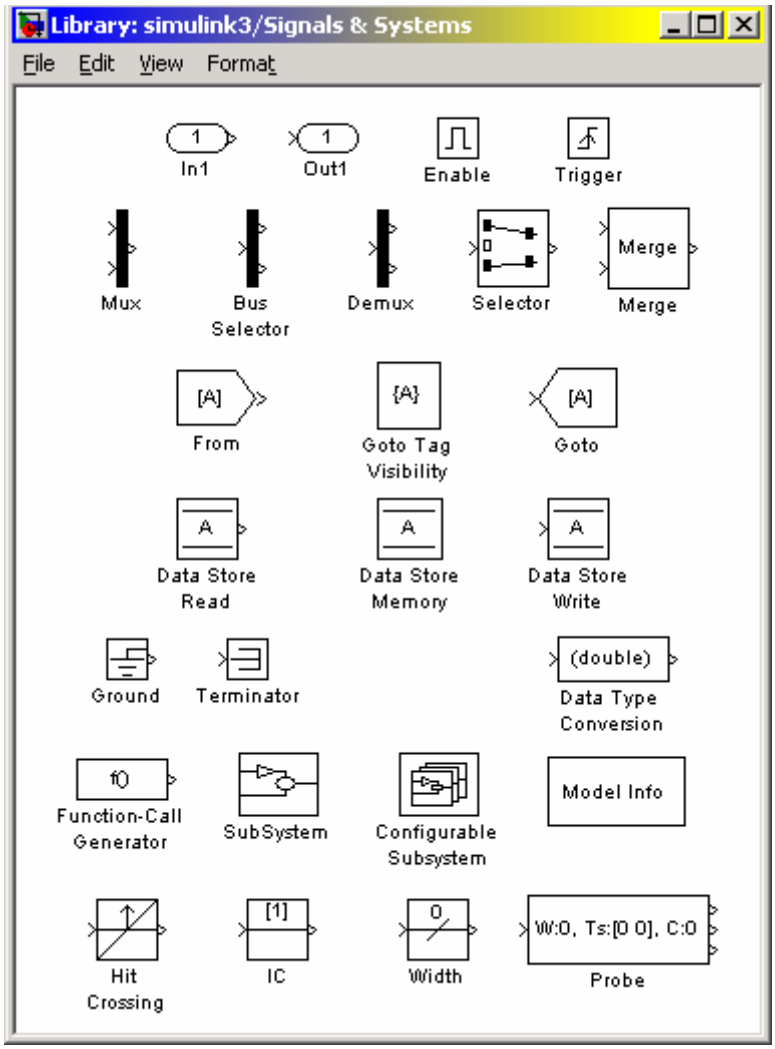

Рис. 7.56

Блоки ***Enable*** *(Разрешить)* и ***Trigger*** *(Задвижка)* предназначены для логического управления работой подсистем S-модели.

Блоки ***Ground*** *(Земля)* и ***Terminator*** *(Ограничитель)* могут использоваться как "заглушки" для тех портов, которые по какой-либо причине оказались не подсоединенными 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)* является "заготовкой" для создания подсистемы. Подсистема - эта довольно самостоятельная S-модель более низкого уровня, которая, в свою очередь, может содержать подсистемы произвольного уровня вложенности.

Блок ***Selector*** выбирает в входном векторе и передает на выход только те элементы, номера которых указаны в одном из параметров настраивания блока. Существенным преимуществом блока является то, что значения параметров его настройки отображаются в графической форме в изображении блока.

Блок ***Width*** *(Размер)* определяет размерность сигнала, который поступает на его вход. Значения размерности выводится непосредственно на изображении блока. Параметров настраивания блок не имеет.

Блок ***Merge*** *(Слияния)* выполняет объединение сигналов, которые поступают в его входы, в единый сигнал.

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 параметры настраивания (рис. 7.57):

- *Hit crossing offset* - определяет значения, пересечение которого необходимо идентифицировать;
- *Hit crossing direction* позволяет указать направление пересечения, при котором это пересечение должно віявляться; значения этого параметра выбирается с помощью выпадающего меню, которое содержит три альтернативы: *rising (восхождение), falling (спадание), either (в любом направлении);*
- *Show output port (указать порт выхода)* - флажок, с помощью которого выбирается вид представления блока.

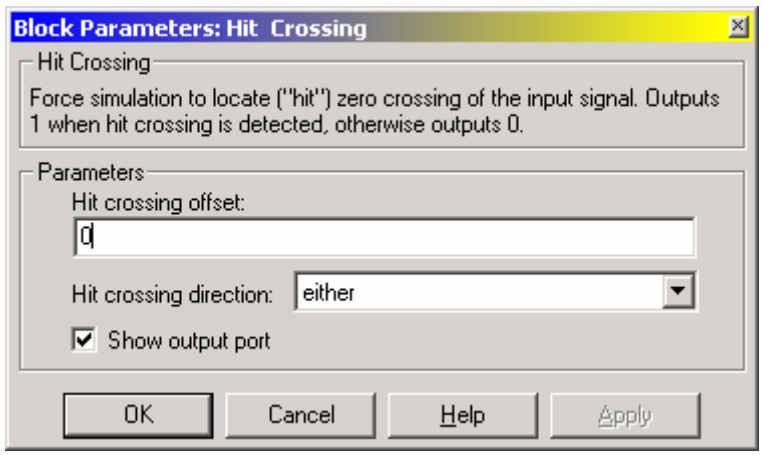

Рис. 7.57

При одновременном выполнении условий, которые задаются параметрами *Hit crossing offset* и *Hit crossing direction*, на выходе блока формируется единичный сигнал. Его продолжительность определяется значением дискрета времени (параметр *Sample time*) блока, который предшествует в модели блоку ***Hit crossing.*** Если этот параметр отсутствует, то единичный сигнал на выходе блока сохраняется до его следующего срабатывания.

7.2. Построение блок-схем

Рассмотрим операции, которые позволяют образовывать блок-схемы сложных динамических систем.

7.2.1. Выделение объектов

При создании и редактировании S-модели нужно выполнять такие операции, как копирование или изъятие блоков и линий, для чего необходимо сначала выделить одних или несколько блоков и линий (объектов).

Чтобы ***выделить отдельный объект,*** нужно щелкнуть на нем один раз. При этом появляются маленькие кружки по углам выделенного блока или в начале и конце линии. *При этом становятся невыделенными все другие перед этим выделенные объекты*. *Если щелкнуть по блоку второй раз, он становится невыделенным*.

На рис. 7.58 справа приведен результат выделения блока ***Clock*** (сравните с рисунком слева).

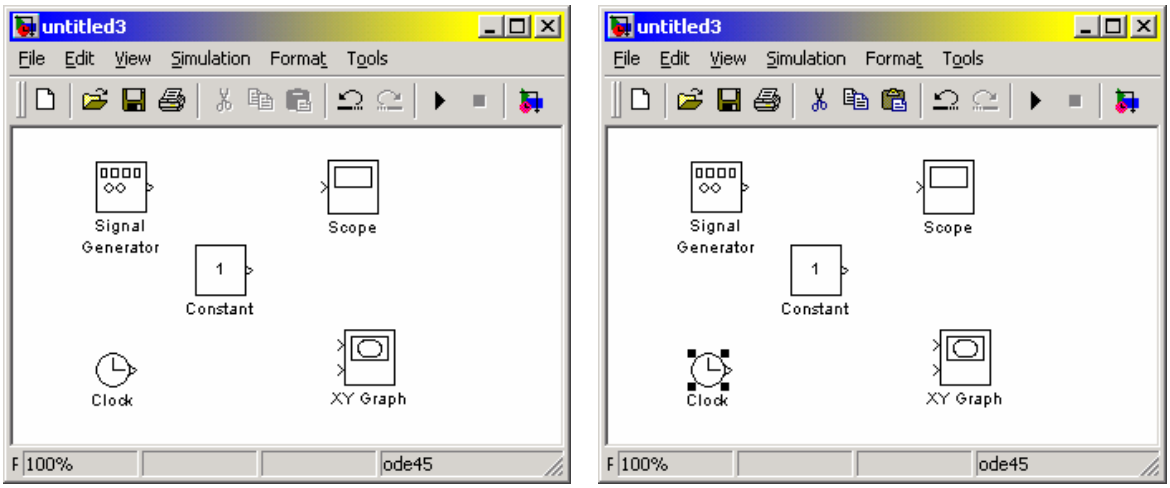

Рис. 7.58

Выделение нескольких объектов по одному осуществляется таким образом:

- нажать клавишу <Shift> и держать ее нажатой;
- щелкнуть на каждом из объектов, которые выделяются, не отпуская клавишу <Shift>;
- отпустить клавишу <Shift> .

Именно таким способом на рис. 7.59 выделены блоки ***Signal Generator***, ***Constant*** и ***XYGraph***.

Выделить несколько объектов с помощью объединяющего бокса можно следующим чином:

- определить стартовый угол прямоугольника-бокса, который будет содержать блоки, которые нужно выделить;
- нажать клавишу мыши в этой точке;

- не отпуская клавишу мыши, переставить ее курсор в противоположный угол бокса; при этом пунктирные линии должны окружить нужные блоки и линии;
- отпустить клавишу мыши; блоки и линии внутри бокса будут выделены.

На рис. 7.60 показан результат выделения боксом блоков ***Signal Generator***, ***Constant*** и ***Clock***.

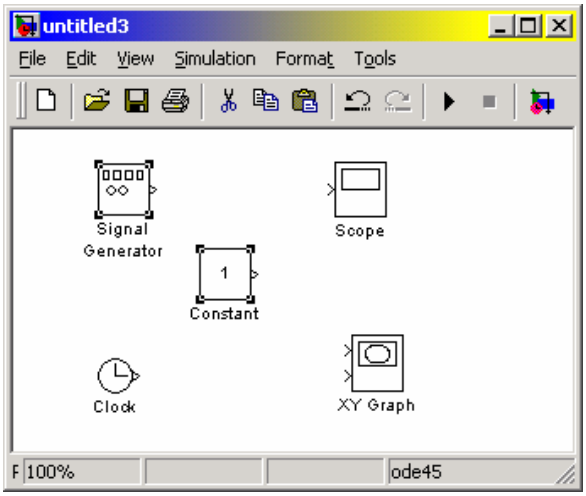

Рис. 7.59

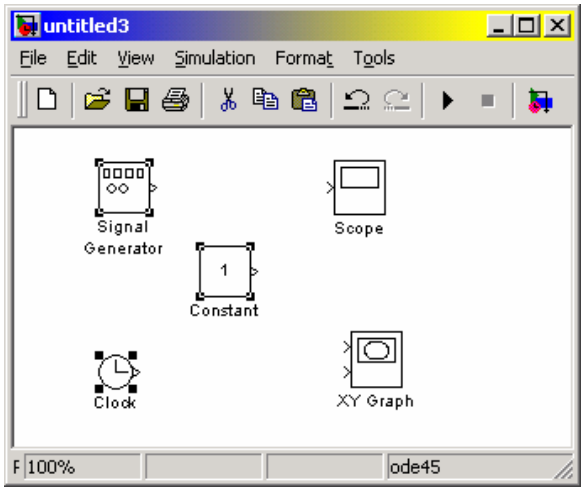

Рис. 7.60

Выделение всей модели, то есть всех объектов в активном окне блок-схемы, осуществляется одним из двух путей:

1) выбором команды *Select All* в меню *Edit* окна блок-схемы;
2) нажатием совокупности клавиша <Ctrl>+<A>.

7.2.2. Оперирование с блоками

Копирование блоков из одного окна в другое

В процессе создания и редактирования модели нужно копировать блоки из библиотеки или другой модели в текущую модель. Для этого достаточно:

- открыть нужный раздел библиотеки или окно модели-прототипа;
- перетянуть мышкой нужный блок в окно создаваемой (редактируемой) модели.

Другой способ заключается в следующем:

1) выделить блок, который нужно скопировать;
2) выбрать команду *Copy (Копировать)* из меню *Edit (Редактирование)*;
3) сделать активным окно, в которое нужно скопировать блок;
4) выбрать в нем команду *Paste (Вставить)* из меню *Edit*.

SimuLINK присваивает имя каждому из скопированных блоков. *Первой 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

Чтобы ***переставить блок внутри модели*** с одного места в другое, достаточно перетянуть его в это положение с помощью мыши. ***SimuLINK*** автоматически перерисовывает линии связей других блоков с тем, который переставлен.

Переставить несколько блоков одновременно, включая соединительные линии можно так:

- выделить блоки и линии (см. предыдущий раздел);
- перетянуть мышью один из выделенных блоков на новое место; остальные блоки, сохраняя все относительные расстояния, займут новые места.

На рис. 7.61 показан результат таких действий с блоками, выделенными на рис. 7. 60.

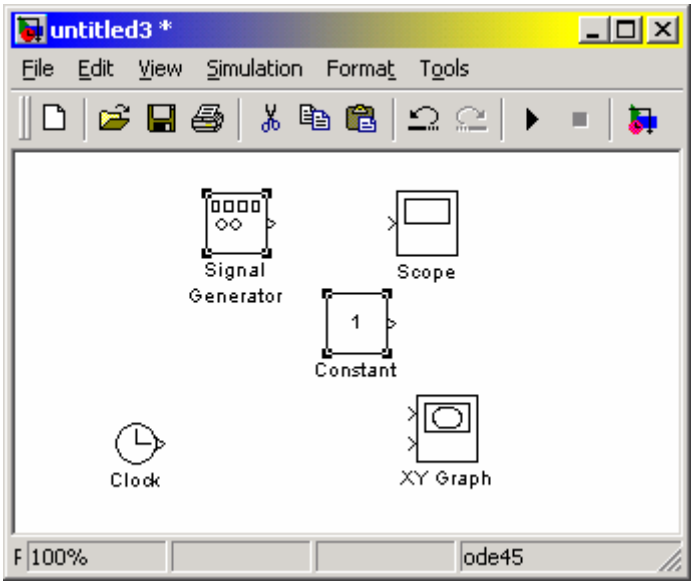

Рис. 7.61

Дублирование блоков внутри модели

Чтобы ***скопировать блоки внутри модели*** нужно сделать следующее:

1) нажать клавишу <Ctrl>;
2) не отпуская клавишу <Ctrl>, установить курсор на блок, что необходимо скопировать и перетянуть его в новое положение.

Того же результата можно достичь, если просто *перетянуть* мышкой блок в новое положение, но *с помощью правой клавиши мыши*.

На рис. 7.62 представлен результат копирования блоков ***Scope*** и ***XYGraph***.

Задание параметров блока

Функции, которые выполняет блок, зависят от значений параметров блока. Установления этих значений осуществляется в окне настраивания блока, которое вызовется, если дважды щелкнуть 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 <Delete> или <Backspace>. Можно также использовать команду *Clear* или *Cut* из меню *Edit* окна блок-схемы. Если использована команда *Cut*, то в дальнейшем удаленные блоки можно скопировать обратно в модель, если воспользоваться командой *Paste* того же меню окна схемы.

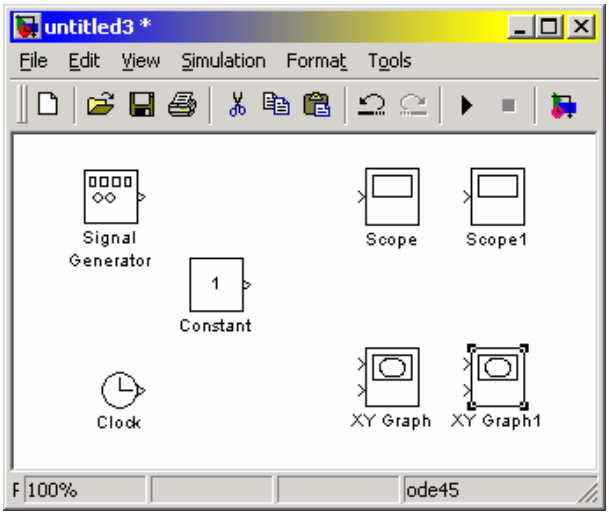

Рис. 7.62

Отсоединение блока

Для ***отсоединения блока от соединяющих линий*** достаточно нажать клавишу <Shift> на изображении блока, и, не отпуская ее, перетянуть блок в некоторое другое место.

Изменение угловой ориентации блока

В обычном изображении блоков сигнал проходит сквозь блок слева направо (по левую сторону размещены входы блока, а по правую сторону - выходы). Чтобы ***изменить угловую ориентацию блока*** нужно:

- выделить блок, который нужно повернуть;
- избрать меню *Format* в окне блок-схемы;
- в дополнительном меню, которое появится на экране, выбрать команду *Flip Block* - поворот блока на 180 градусов, или *Rotate Block* - поворот блока по часовой стрелке на 90 градусов.

На рис. 7.63 показан результат применения команды *Rotate Block* к блоку ***Constant*** и команд *Rotate Block* и *Flip Block* - к блоку ***SignalGenerator***.

Изменение размеров блока

Чтобы изменить размеры блока, нужно сделать следующее:

- выделить блок, размеры которого надо изменить;
- привести курсор мыши на одну из угловых меток блока; при этом на экране у этой метки должен возникнуть новый курсор в виде обоюдной стрелки под наклоном 45 градусов;
- захватить эту метку мышью и перетянуть в новое положение; при этом противоположная метка этого блока останется неподвижной.

На рис. 7.64 показан результат применения этих операций для растяжения блока ***XYGraph***, а также середина процесса увеличения размеров блока ***Scope***.

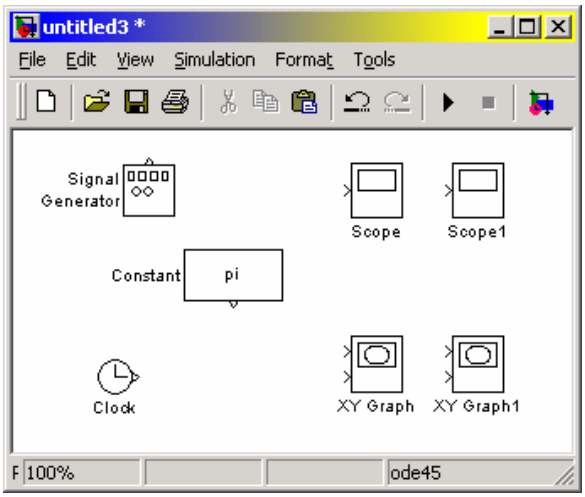

Рис. 7.63

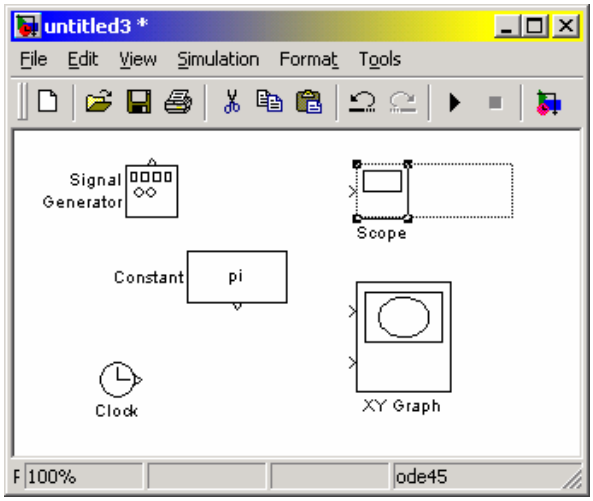

Рис. 7.64

Изменение имен блоков и манипулирования с ними

Все имена блоков в модели должны быть уникальными и иметь, как минимум один символ. Если блок ориентирован слева направо, то имя, по умолчанию, находится под блоком, если справа налево - выше блока, если же сверху вниз или снизу вверх - по правую сторону блока (см. рис. 7.63).

Изменение имени блока осуществляется так: надо щелкнуть на существующем имени блока, потом, используя клавиши обычного редактирования текста, изменить это имя на нужное.

Для ***изменения шрифта*** следует выделить блок, потом выбрать команду *Font* из меню *Format* окна модели и, наконец, выбрать нужный шрифт из представленного перечня.

Чтобы ***изменить местоположение имени выделенного блока,*** существуют два пути:

- перетянуть имя на противоположную сторону мышью;
- воспользоваться командой *Flip Name* из раздела *Format* меню окна модели - она тоже переносит имя на противоположную сторону.

Удалить имя блока можно, используя команду *Hide Name* из меню *Format* окна модели. Чтобы ***восстановить*** потом ***отображение имени*** рядом с изображением блока, следует воспользоваться командой *Show Name* того же меню.

7.2.3. Проведение 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 соединять выходной порт одного блока с входными портами нескольких блоков через разветвление линии.

Создание линии между блоками

Для соединения выходного порта одного блока с входным портом другого блока следует выполнить такую последовательность действий:

- установить курсор внутрь выходного порта первого блока; при этом *курсор должен превратиться на перекрестие*;
- нажав левую клавишу мыши и, удерживая ее в этом положении, передвинуть перекрестие к входному порту второго блока;
- отпустить ЛКМ; ***SimuLINK*** заменит символы портов соединительной линией с представлением направления передачи сигнала.

Именно таким образом соединен на рис. 7.65 выход блока ***Clock*** с входом блока ***XYGraph***.

Линии можно рисовать как от выходного порта к входному, так и наоборот.

SimuLINK рисует соединительные линии, используя лишь горизонтальные и вертикальные сегменты. *Для образования диагональной линии нажмите и удерживайте клавишу <Shift> на протяжении рисования.*

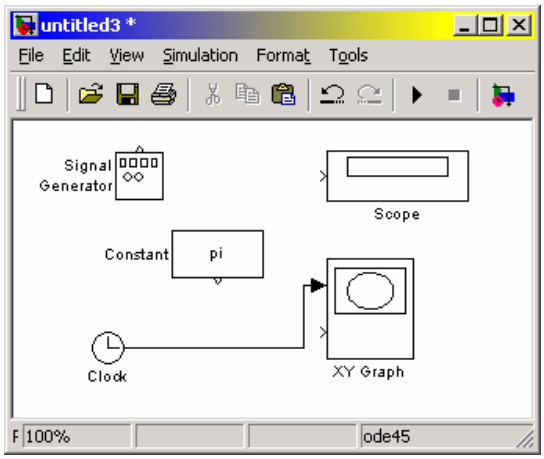

Рис. 7.65

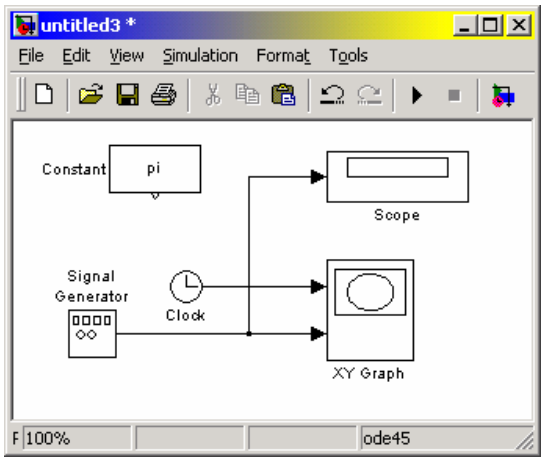

Рис. 7.66

Создание разветвления линии

Линия, которая разветвляется, начинается с существующей и передает ее сигнал к входному порту другого блока. Как существующая, так и ответвленная линия передают тот самый сигнал. Разветвленная линия дает возможность передать тот самый сигнал до нескольких блоков.

Чтобы ***образовать ответвление*** от существующей линии, нужно:

- установить курсор на точку линии, от которой должна ответвляться другая линия;
- нажав и удерживая клавишу <Ctrl>, нажать и удерживать ЛКМ;

- провести линию к входному порту нужного блока; отпустить клавишу <Ctrl> и ЛКМ (см. рис. 7.66).

Создание сегмента линии

Линии могут быть нарисованы по сегментам. В этом случае для создания следующего сегмента следует установить курсор в конец предыдущего сегмента и нарисовать (с помощью мыши) следующий сегмент. Таким образом, например, соединены на рис. 7.67 блоки ***Clock*** с ***XYGraph*** и ***SignalGenerator*** с ***XYGraph***.

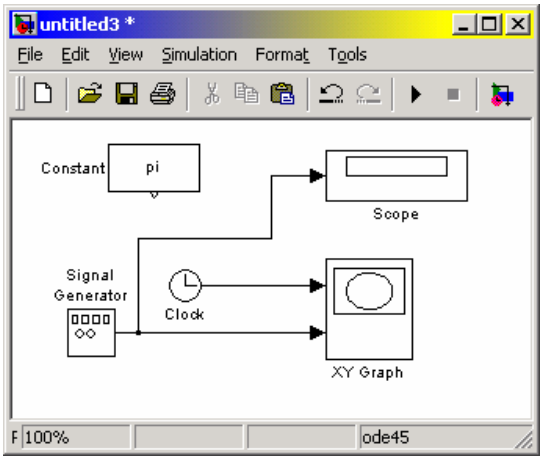

Рис. 7.67

Передвижение сегмента линии

Чтобы ***передвинуть отдельный сегмент*** линии, необходимо выполнить следующее:

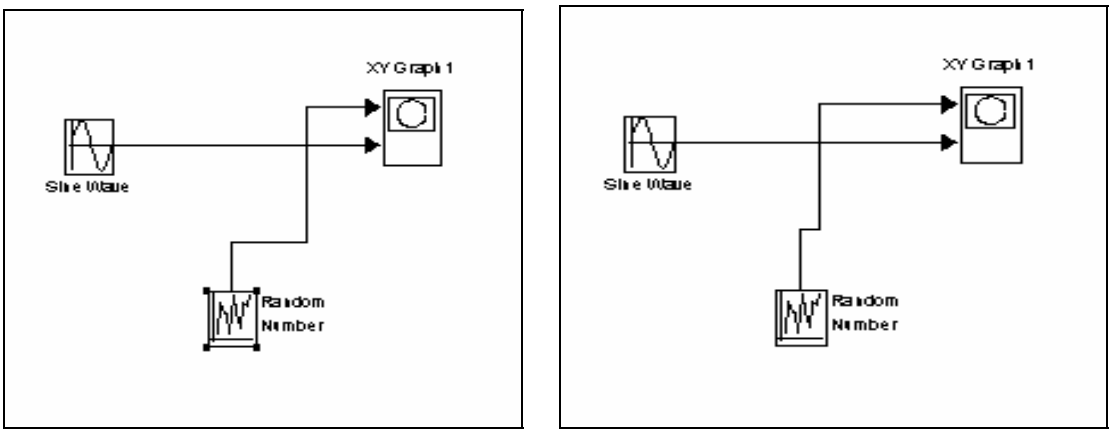

Рис. 7.68

- установить курсор на сегмент, который нужно передвинуть;
- нажать и удерживать левую клавишу мыши (ЛКМ); при этом курсор должен превратиться на "крест";
- передвинуть "крест" к новому положению сегмента;
- отпустить ЛКМ.

На рис. 7.68 показан результат передвижения вертикального сегмента линии, которая соединяет блоки ***Random Number*** с ***XYGraph1***.

Нельзя передвинуть сегмент, который непосредственно прилегает к порту блока.

Разделение линии на сегменты

Чтобы ***разделить линию на два сегмента,*** нужно:

- выделить линию;
- установить курсор в ту точку выделенной линии, в которой линия должна быть разделена на два сегмента;
- удерживая нажатой клавишу <Shift>, нажать и удерживать ЛКМ; курсор превратится на маленький круг; на линии образуется слом;
- передвинуть курсор в новое положение слома;
- отпустить ЛКМ и клавишу <Shift>.

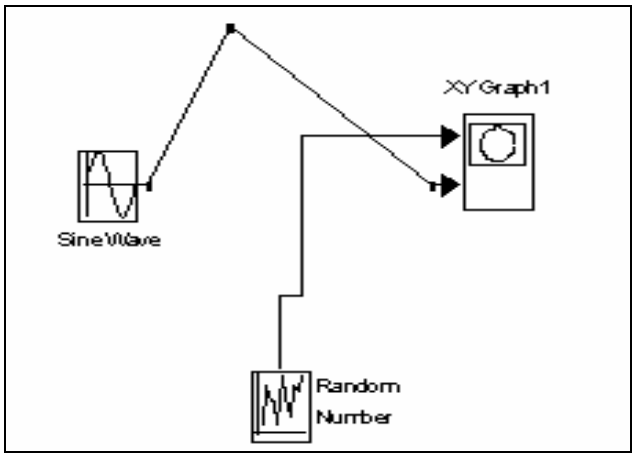

Рис. 7.69

Пример проведения этих действий представлен на рис. 7.69, где линия, которая соединяет блоки ***Sine Wave*** и ***XYGraph1*** разделена на два сегмента.

Передвижение слома линии

Для ***передвижения слома*** линии достаточно перетянуть точку этого слома в новое положение на блок-схеме.

7.2.4. Проставление меток сигналов и комментариев

Для наглядности оформления блок-схемы и удобства пользованием ею можно сопровождать линии метками сигналов, протекающим по ним. Метка размещается под или над горизонтальной линией, по левую сторону или по правую сторону вертикальной линии. Метка может быть расположена в начале, в конце или посреди линии.

Создание и манипулирование метками сигналов

Чтобы ***создать метку сигнала,*** надо дважды щелкнуть на сегменте линии и ввести метку (рис. 7.70). *При создании метки сигнала необходимо дважды щелкнуть именно точно на линии, так как иначе будет создан комментарий к модели.*

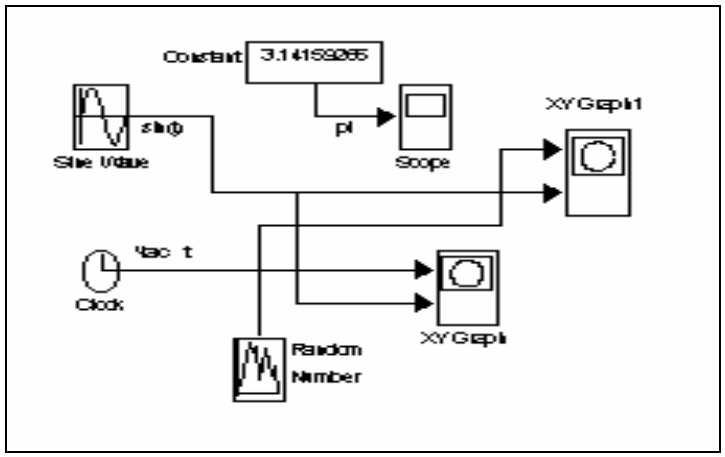

Рис. 7.70

Для ***передвижения метки*** надо ее просто перетянуть на новое место мышью. Чтобы ***скопировать метку,*** следует нажать и удерживать клавишу <Ctrl> и перетянуть метку к новому положению на линии, или избрать другой сегмент линии, на котором нужно установить копию метки и дважды щелкнуть по этому сегменту линии. ***Отредактировать метку,*** можно щелкнув на ней и осуществляя потом соответствующие изменения как в обычном текстовом редакторе. Чтобы ***удалить метку***, нажмите и удерживайте клавишу <Shift>, выделите метку и уничтожьте ее, используя клавиши <Delete> или <Backspace>. При этом будут удалены все метки этой линии.

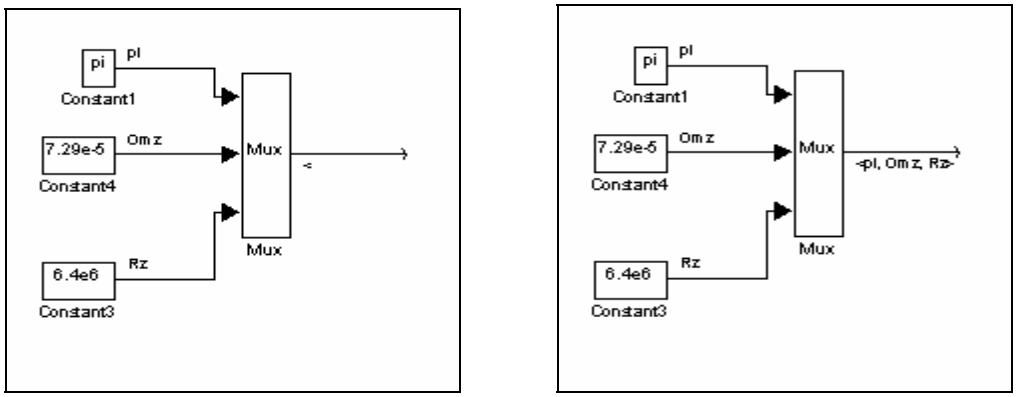

Рис. 7.71

Распространение меток линии

Распространение меток линии - это процесс автоматического переноса имени метки к сегментам одной линии, которые разорваны блоками ***From/Goto***, ***Mux*** (рис. 7.71).

Чтобы ***распространить метки,*** создайте во втором и следующих сегментах линии метки с именем '<'. После выполнения команды *Update Diagram* из раздела *Edit* меню окна модели, или одновременного нажатия клавиш <Ctrl>+<D> в этих сегментах автоматически будут проставлены метки (см. рис. 7.71)

Создание комментария и манипулирование ним

Комментарии дают возможность снабжать блок-схемы текстовой информацией о модели и отдельных ее составляющих. Комментарии можно проставлять в любом свободном месте блок-схемы (см. рис. 7.72).

Для ***создания комментария*** дважды щелкните в любом свободном месте блок-схемы, а потом введите комментарий в возникшем прямоугольнике. Для ***перемещения комментария*** в другое место его нужно перетянуть на это место с помощью мыши. Чтобы ***скопировать комментарий,*** достаточно нажать клавишу <Ctrl> и, удерживая ее в этом положении, перетянуть комментарий в новое место. Для ***редактирования комментария*** надо щелкнуть на нем, а потом внести нужные изменения как в обычном текстовом редакторе. Чтобы ***изменить*** при этом ***шрифт***, его размер или стиль, следует выделить текст комментария, который нужно изменить, а потом избрать команду *Font* из меню *Format* окна блок-схемы, выбрать в окне, которое возникнет, название шрифта, его размер, атрибуты и стиль и нажать кнопку <Ok> в этом окне.

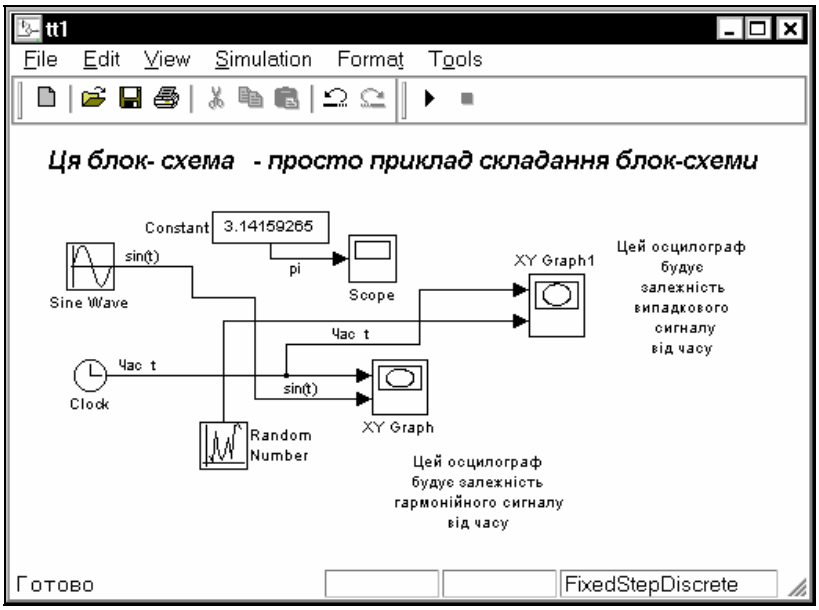

Рис. 7.72

Чтобы ***удалить комментарий,*** нажмите и удерживайте клавишу <Shift>, выделите комментарий и нажмите клавишу <Delete> или <Backspace>.

7.2.5. Создание подсистем

Если сложность и размеры блок-схемы модели становятся весьма большими, ее можно существенно упростить, группируя блоки в подсистемы. Использование подсистем дает следующие преимущества:

- сокращения количества блоков, которые выводятся в окне модели;
- объединение в единую группу (подсистему) функционально связанных блоков;
- возможность создания иерархических блок-схем.

Существуют две возможности создания подсистем:

- добавить блок ***Subsystem*** в модель, потом войти в этот блок и создать подсистему в возникшем окне подсистемы;
- выделить часть блок-схемы модели и объединить ее в подсистему.

Создание подсистемы через добавление блока ***Subsystem***

В этом случае следует поступить так:

- скопировать блок ***Subsystem*** в окно модели, перетянув его из библиотеки ***Sygnals&Systems***;
- раскрыть окно блока ***Subsystem***, дважды щелкнув на изображении блока в блок-схеме;
- в пустом окне модели создать подсистему, используя блоки ***In*** и ***Out*** для создания входов и выходов подсистемы.

Создание подсистемы путем группирования существующих блоков

Если блок-схема уже содержит блоки, которые нужно объединить в подсистему, то последнюю можно образовать таким образом:

- выделить объединяющим боксом блоки и соединительные блоки, которые нужно включить в состав подсистемы (рис. 7.73);
- избрать команду *Create Subsystem* из меню *Edit* окна модели; при этом ***SimuLINK*** заменит выделенные блоки одним блоком ***Subsystem*** (см. рис. 7.73)

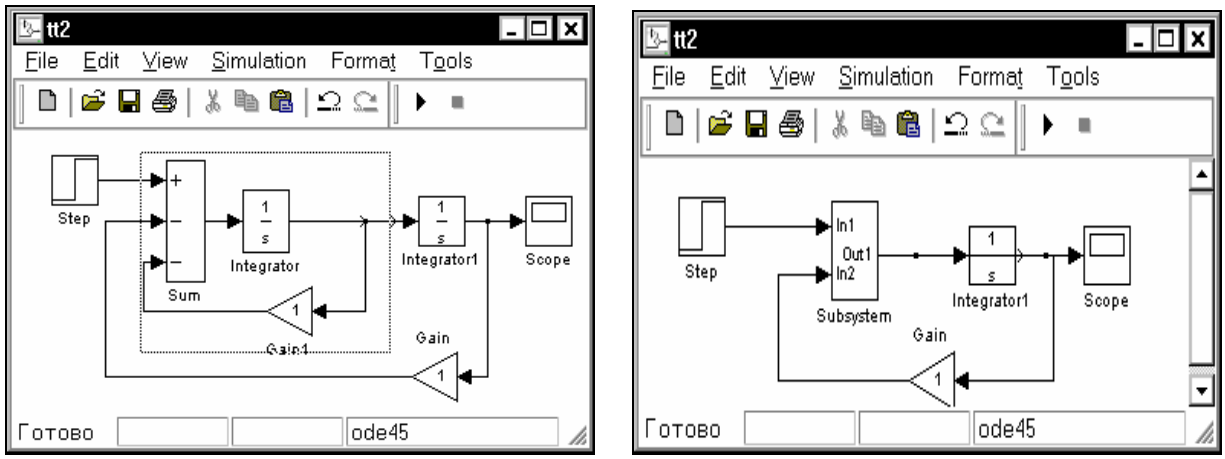

Рис. 7.73

Если раскрыть окно блока ***Subsystem***, дважды щелкнув на нем, то ***SimuLINK*** отобразит блок-схему созданной подсистемы (рис. 7.74). Как видно, ***SimuLINK*** прибавил блоки ***In*** и ***Out*** для отображения входов и выходов в систему высшего уровня.

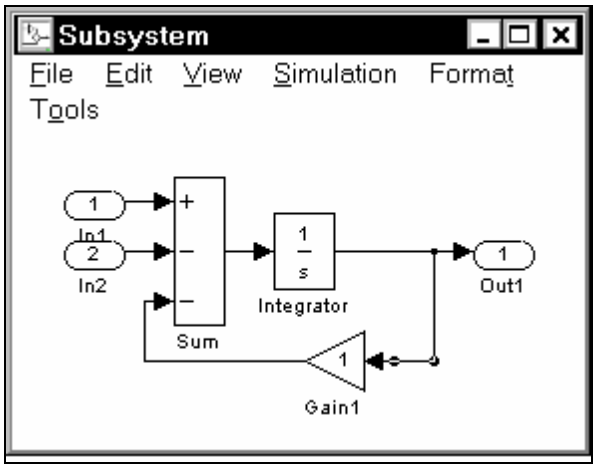

Рис. 7.74

7.2.6. Запись и распечатка блок-схемы S-модели

Для ***записи модели*** (блок схемы) на диск нужно выполнить команду *Save* или *Save As* в меню *File* окна модели. При этом ***SimuLINK*** записывает в указанную директорию файл с указанным (введенным из клавиатуры) именем, присваивая нему расширение MDL.

Чтобы распечатать модель (блок-схему) на принтере, достаточно воспользоваться командой *Print* из меню *File* окна модели.

Довольно интересной является возможность "***распечатать*** *блок-схему в документе* любого текстового редактора, например, Word. Для этого следует использовать команду *Copy Model* из меню *Edit* окна модели, которая запоминает в буфере содержимое окна модели. Если после этого войти в окно текстового редактора и нажать клавиши <Shift>+<Insert>, в открытом документе редактора возникнет изображение блок-схемы модели. Именно таким образом получены рисунки 7.68...7.71.

7.3. Примеры моделирования

7.3.1. Моделирование поведения физического маятника

Рассмотрим ту же задачу моделирования поведения физического маятника при гармонической вибрации точки его опоры.

Пользуясь результатами ранее проведенных (п. 2.6.2) преобразований, исходное уравнение движения маятника примем в такой безразмерной форме

$$\varphi'' + \sin\varphi = S(\tau, \varphi, \varphi'), \qquad (7.1)$$

где обозначено

$$S(\tau,\varphi,\varphi') = -2\cdot\zeta\cdot\varphi' - \\ -[n_{mx}\cdot\sin(\nu\cdot\tau+\varepsilon_x)\cdot\cos\varphi + n_{my}\cdot\sin(\nu\cdot\tau+\varepsilon_y)\cdot\sin\varphi], \quad (7.2)$$

причем безразмерные величины ζ и ν определяются выражениями:

$$\zeta = \frac{R}{2\cdot\sqrt{mgl\cdot J}}; \qquad \nu = \frac{\omega}{\omega_0}; \qquad \omega_0 = \sqrt{\frac{mgl}{J}}.$$

Исходными (задаваемыми) параметрами для моделирования будем считать:

1) параметры самого маятника; к ним в анализируемом случае относятся только относительный коэффициент затухания ζ;

2) параметры, характеризующие внешнее ввоздействие; сюда входят:

- амплитуды виброперегрузок в вертикальном n_{my} и горизонтальном n_{mx} направлениях;
- относительная (относительно частоты 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 от вертикали;
- начальная безразмерная угловая скорость маятника $\varphi' = \dot{\varphi}\ /\omega_o$.

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 другой форме:

$$\varphi'' = S(\tau,\varphi,\varphi') - \sin\varphi. \quad (7.3)$$

В основу воплощения этого уравнения в блок-схему положим такую идею: *если сформировать правую часть уравнения по "известным" процессам* $\varphi(\tau)$ *и* $S(\tau,\varphi,\varphi')$, *то тем самым станет известным угловое ускорение* $\varphi''(\tau)$. *Если теперь проинтегрировать ускорение, можно получить угловую скорость* $\varphi'(\tau)$. *Наконец, проинтегрировав и ее, можно получить закон изменения угла* $\varphi(\tau)$ *от времени. Последние полученные две величины (процессы) можно отныне использовать для формирования правой части уравнения (7.3)*.

Итак, для формирования блок-схемы, осуществляющей численное интегрирование уравнения (7.1), можно поступить так:

- в основу блок-схемы положить два последовательно соединенных интегратора (блоки ***Integrator***) с начальными условиями, которые задаются внешне (external); на вход первого интегратора подать угловое ускорение, а как начальное условие 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 $\varphi(0)$; выходом этого блока будет искомый процесс $\varphi(\tau)$;

- сформировать отдельным блоком в виде подсистемы (блок ***Subsystem***) функцию $S(\tau,\varphi,\varphi')$, используя как входные "полученные" процессы $\varphi(\tau)$ и $\varphi'(\tau)$, а как входные параметры - $2\cdot\zeta$, n_{mx}, ε_x, n_{my}, ε_y; один из вариантов воплощения такой подмодели соответствующей выражению (7.2) представлено на рис. 7.75;

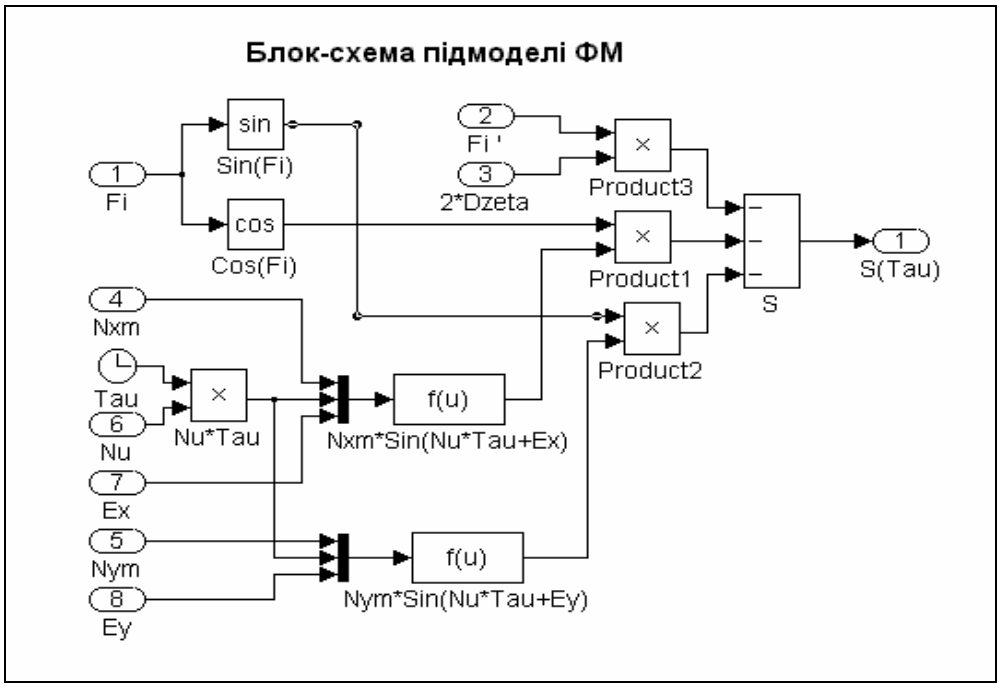

Рис. 7.75

- используя созданную подмодель, в основной модели соединить "сформированые" процессы $\varphi(\tau)$ и $\varphi'(\tau)$ с соответствующими входами подмодели, а выход подмодели связать с суматором, на который подать также предварительно сформированный сигнал $\sin(\varphi)$; сигнал с выхода суматора подать на вход первого интегратора, замыкая цепь интегрирования;
- для отображения результатов интегрирования в графической форме присоединить блок ***Scope*** к выходу системы - сформированного сигнала $\varphi(\tau)$; для получения фазового портрета маятника использовать блок ***XY Graph***, на входы которого направить сигналы $\varphi(\tau)$ и $\varphi'(\tau)$.

На рис. 7.76 приведена блок-схема, которая реализует указанные идеи.

Следует обратить внимание на то, как схемно реализуются заданные колебания точки подвеса. Для этого используется "генератор времени" - блок ***Clock*** (рис. 7.75). 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. 7.75), созданный на основе стандартного блока ***Fcn*** и реализующий такую зависимость:

$$f(u) = u(1) * \sin(u(2) + u(3)),$$

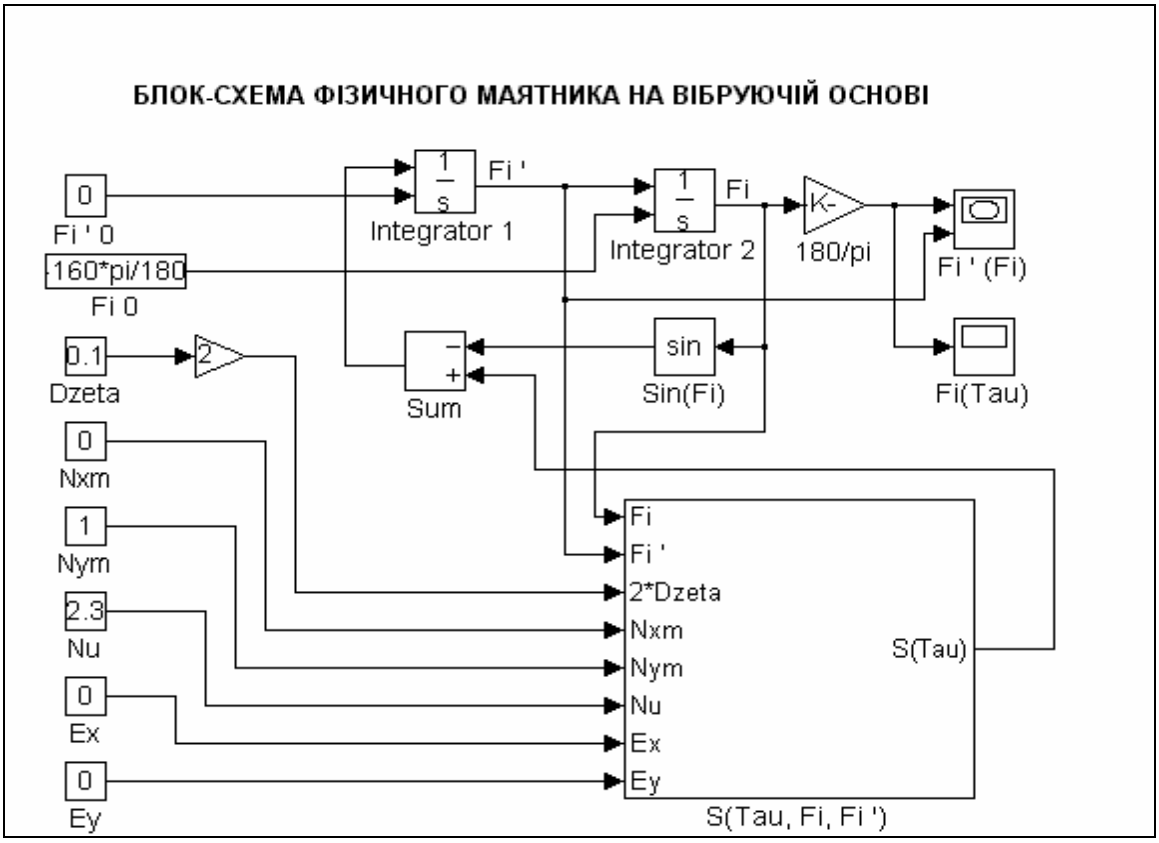

Рис. 7.76

где $u(1), u(2), u(3)$ - элементы вектора *u*, подаваемого на вход блока. Таких блоков в подсистеме рис. 7.52 два. В первом из них в качестве первого элемента входного вектора используется величина n_{mx}, второго элемента - $\nu\tau$ и третьего - ε_x. Во втором первым элементом является n_{my}, вторым - $\nu\tau$ и третьим - ε_y.

Отметим также, что при построении блок-схемы подсистемы (см. рис. 7.69) связь подсистемы с основной системой осуществляется через введение в подсистему стандартных блоков типа ***In*** (Вход) и ***Out*** (Выход). Все величины, которые формируются в основной модели, а потом должны быть использованы в подмодели, должни "проникать" в подсистему через блоки ***In***, а величины, сформированные в подсистеме и потом используемые в основной системе, должны выходить из подсистемы через блоки ***Out***.

При этом на изображении блока подсистемы в блок-схеме основной модели автоматически появляется изображения такого количества входов, которое совпадает с количеством введенных в подсистеме блоков ***In,*** и выходов, равных числу блоков ***Out***, использованных в подсистеме.

Используя меню *Simulation* окна главной блок-схемы, а в выпадающем меню, которое возникает при этом, - команду *Start*, можно активизировать процесс моделирования созданной S-модели маятника. По завершении этого процесса возникнет дополнительное графическое окно, созданное блоком ***XY Graph*** (см. рис. 7.77), на котором изображен фазовый портрет маятника при выбранных параметрах маятника и возмущений. Если теперь дважды щелкнуть на изображении блока ***Scope*** в блок-схеме маятника, то возникнет еще одно графическое окно (рис. 7.78) с графиком зависимости угла от времени.

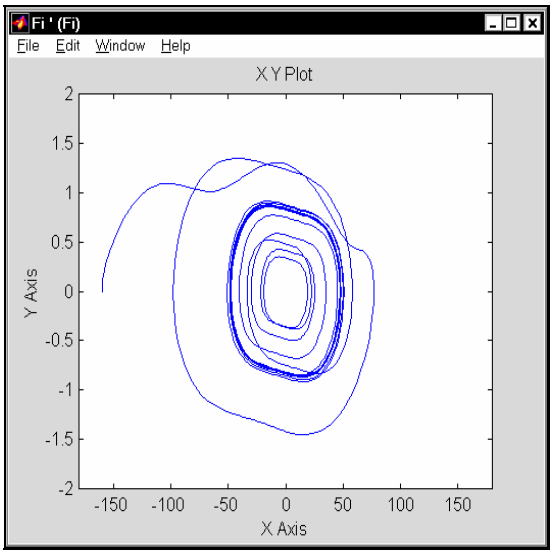

Рис. 7.77

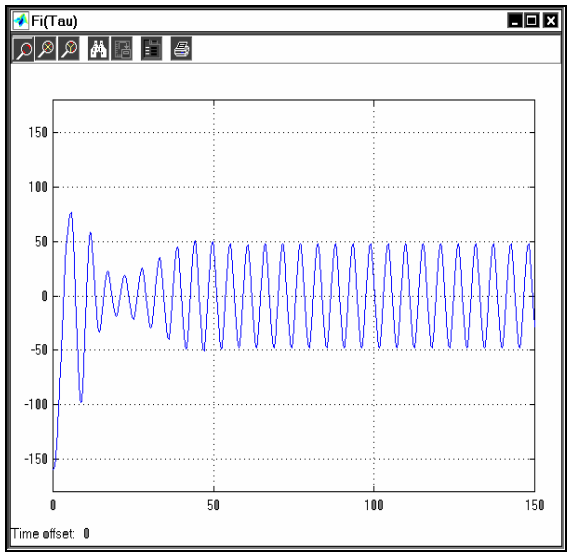

Рис. 7.78

Графики, представленные на рис. 7.77 и 7.78, отвечают значениям входных данных, указанным на схеме черт. 7.76, 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 понятны значительные преимущества и некоторые недостатки моделирования динамических систем с помощью пакета SIMULINK в сравнении с аналогичными исследованиями с помощью программы:

- составление блок-схемы уравнений движения вместо набора текста процедуры правых частей значительно более наглядно, позволяет контролировать правильность набора путем осознания физического содержания отдельных блоков и их взаимосвязей;
- при проведении самого процесса моделирования в среде SIMULINK исчезает потребность в организации самого процесса численного интегрирования дифференциальных уравнений и даже выведения результатов в графической форме;
- однако форма вывода результатов в графической форме в SIMULINK является недостаточно гибкой: нельзя добавить собственные надписи в заголовок и по осям графика, нельзя установить сетку координатных

линий; в особенности неудобно то, что здесь не предусмотрены средства вывода текстовой информации на поле графика, что делает графическое представление безадресным.

Последний недостаток существенен. Он может быть устранен существующими в пакете SIMULINK средствами. Например, можно записать полученные значения исходных величин в MAT-файл (посылая их на блок ***To File***), а потом 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 разделах 2.5 и 2.7. Такой путь использован в следующем примере. Неудобством применения обзорного окна ***XY Graph*** является также то, что предварительно нужно установить диапазоны изменения обеих входных величин по осям графика. Если эти диапазоны установлены неверно, в обзорном окне может вообще не возникнуть изображение графика, или появится такой его фрагмент, по которому невозможно сделать правильный вывод о поведении исследуемой системы. А при исследовании системы часто невозможно заранее предусмотреть диапазоны изменений величин, или сделать это слишком сложно.

7.3.2. Моделирование поведения гироскопа в карданове подвесе

Уравнения движения гироскопа в карданове подвесе на неподвижном основании представим в виде (см. Задача 2.19, раздел 2.6)

$$\begin{cases} (J_1 + J_2\cos^2\beta)\ddot{\alpha} - 2J_2\dot{\alpha}\dot{\beta}\sin\beta\cos\beta + H\dot{\beta}\cos\beta = \\ = -f_2\dot{\alpha} + N_0 + N_m\sin(\omega t + \varepsilon_N) - [R_0 + R_m\sin(\omega t + \varepsilon_R)]\sin\beta, \\ J_3\ddot{\beta} + J_2\dot{\alpha}\sin\beta\cos\beta - H\dot{\alpha}\cos\beta = -f_2\dot{\beta} + L_0 + \\ + L_m\sin(\omega t + \varepsilon_L), \\ \dfrac{dH}{dt} = R_0 + R_m\sin(\omega t + \varepsilon_R), \end{cases}$$

Воплощение этой системы уравнений в блок-схему S-модели показано на рис. 7.79...7.82.

Рисунок 7.79 представляет основную модель.

Основная S-модель содержит в себе такие основные компоненты:

- блоки задания исходных данных; это совокупность блоков типа ***Constant***, определяет значения постоянных величин, входящих в уравнения гироскопа:
- две основных подсистемы (построенные на основе стандартных блоков ***Subsystem***): ***ГКП*** и ***Моменты***;
- ряд блоков построения графических изображений исходных процессов; среди них 4 блока типа ***Scope*** и два блока типа ***XYGraph***; назначение каждого из них становится понятным из рассмотрения блок-схемы;

- блок типа ***To File***, записывающий 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.***

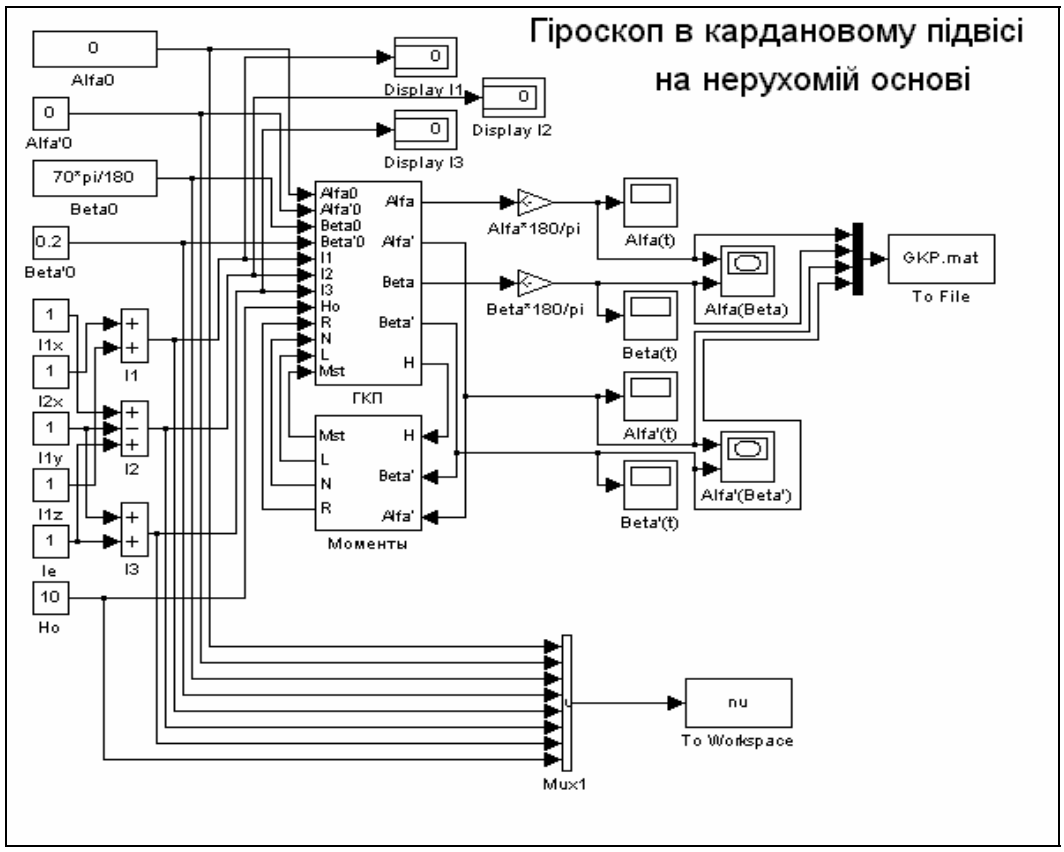

Рис. 7.79

Схема подсистемы ***ГКП*** представлена на следующем рисунке (7.80).

Подсистема осуществляет численное интегрирование исходной системы дифференциальных уравнений. Здесь размещены 5 интеграторов (так как принятая система дифференциальных уравнений имеет пятый порядок). Входы с 1 по 8, 12 и 13 представляют величины, которые не изменяются в процессе моделирования. Входы же из 9 по 11 являются моментами сил, действующие по осям подвеса гироскопа, значения которых формируются в подсистеме ***Моменты***. Число выходов в этой подсистеме - 5. Сюда входят два угла поворота гироскопа, две их производные по времени и текущее значение собственного кинетического момента гироскопа.

Подсистема ***Моменты*** (рис. 7.81) использует данные об угловых скоростях поворота гироскопа и текущее значение кинетического момента для формирования величин моментов сил сопротивления (трения) по осям подвеса. Блок ***Моменты*** имеет 3 входа. Кроме этого, внутри подсистемы осуществляется установка 13 постоянных параметров, которые определяют коэффициенты трения по осям и параметры гармонических колебаний моментов. Выходами подсистемы являются текущие значения моментов сил по каждой из трех осей карданового подвеса. Формирование составляющих моментов сил, изменяющихся

гармонически, происходит по однотипному правилу, воплощенному в трех полностью аналогичных под-подсистемах, одна из которых представлена на рис. 7.82.

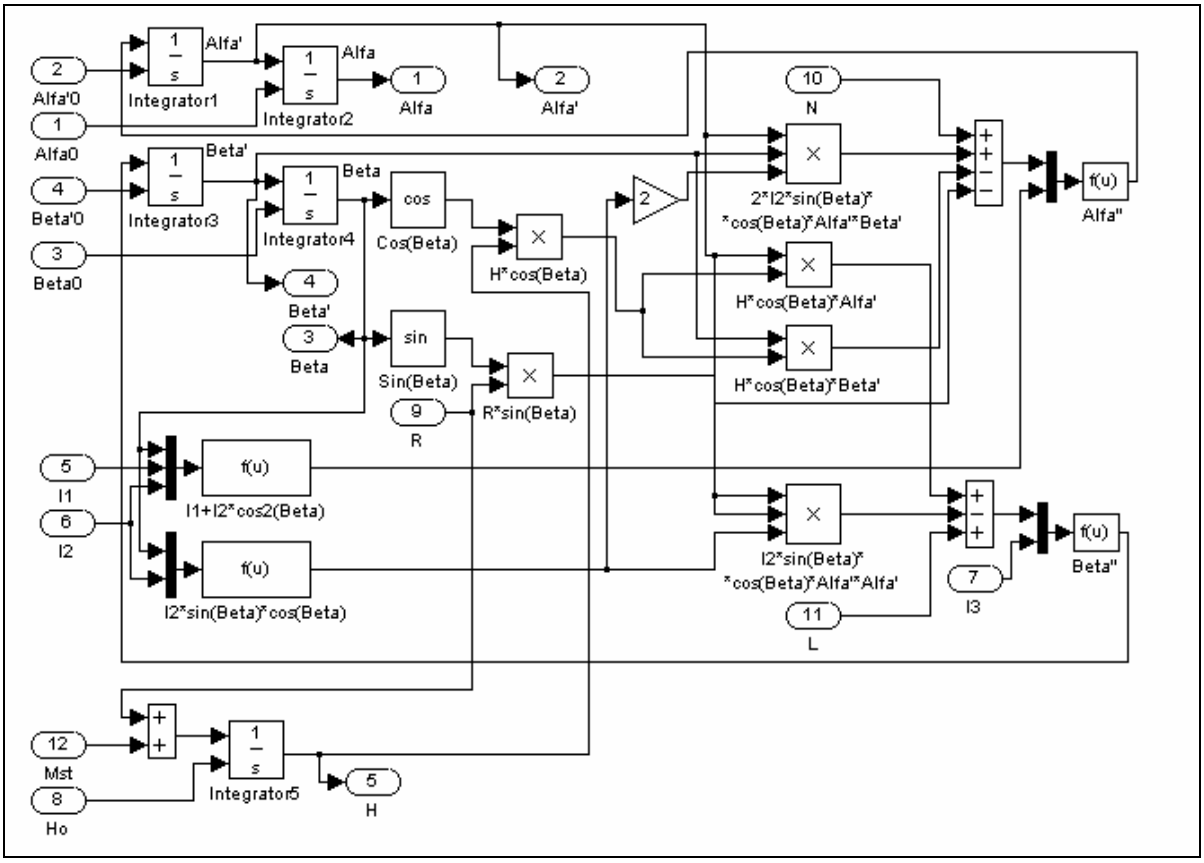

Рис. 7.80

Как вытекает из схемы черт. 7.79, на вход конечного файла ***GKP. mat*** подаются 4 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 (относительно 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.

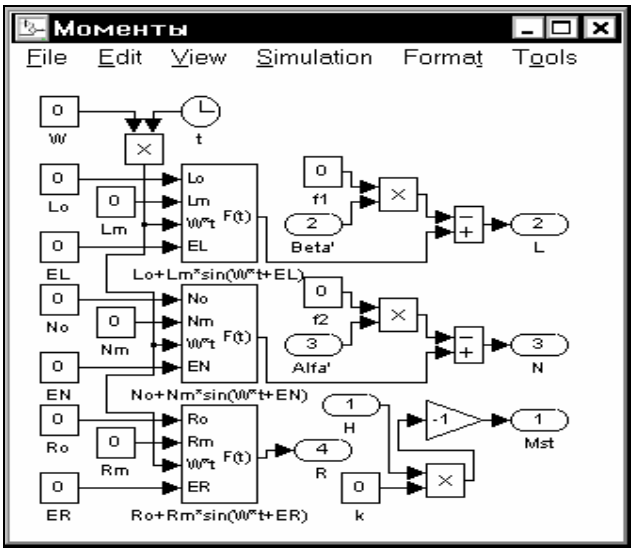

Рис. 7.81

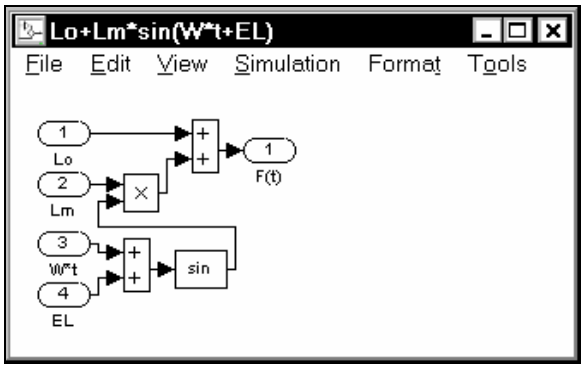

Рис. 7.82

Если файл ***GKP. mat*** сформирован (а это произойдет автоматически при запуске основной S-модели путем нажатия команды *Start* меню *Simulation*), появляется возможность использовать записанные данные в специальном Script-файле для оформления графического окна с необходимым текстовым оформлением.

Далее приведен текст варианта такого Script-файла. Запуская этот Script-файл после проведения вычислений, получим графическое окно, представленное на рис. 7.83.

Script-файл ***GKP_graf***

```
% Считывание исходных данных из записанной в переменной "nu" рабочего пространства
 %  информации          (см. блок  To Workspace  схемы рис. 7.33)
Alfa0=nu(1,1)*180/pi;  Beta0=nu(1,3)*180/pi;
Alfat0=nu(1,2);          Betat0=nu(1,4);
l1=nu(1,5);       l2=nu(1,6);       l3=nu(1,7);
H0=nu(1,8);
load GKP;      % Загрузка (чтения) файла  GKP. mat
 %  ( Происходит присваивание системной переменной ANS значений
 %          всей матрицы, которая содержится в файле GKP.  mat.
 %      При этом модельное время занимает первую строку этой матрицы,
 %          а массивы входных величин - следующие 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,:);
set(gcf,'Color','white')
 %       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',14);
xlabel('Альфа (градусы)','FontSize',10);
ylabel('Бета (градусы)','FontSize',10);
```

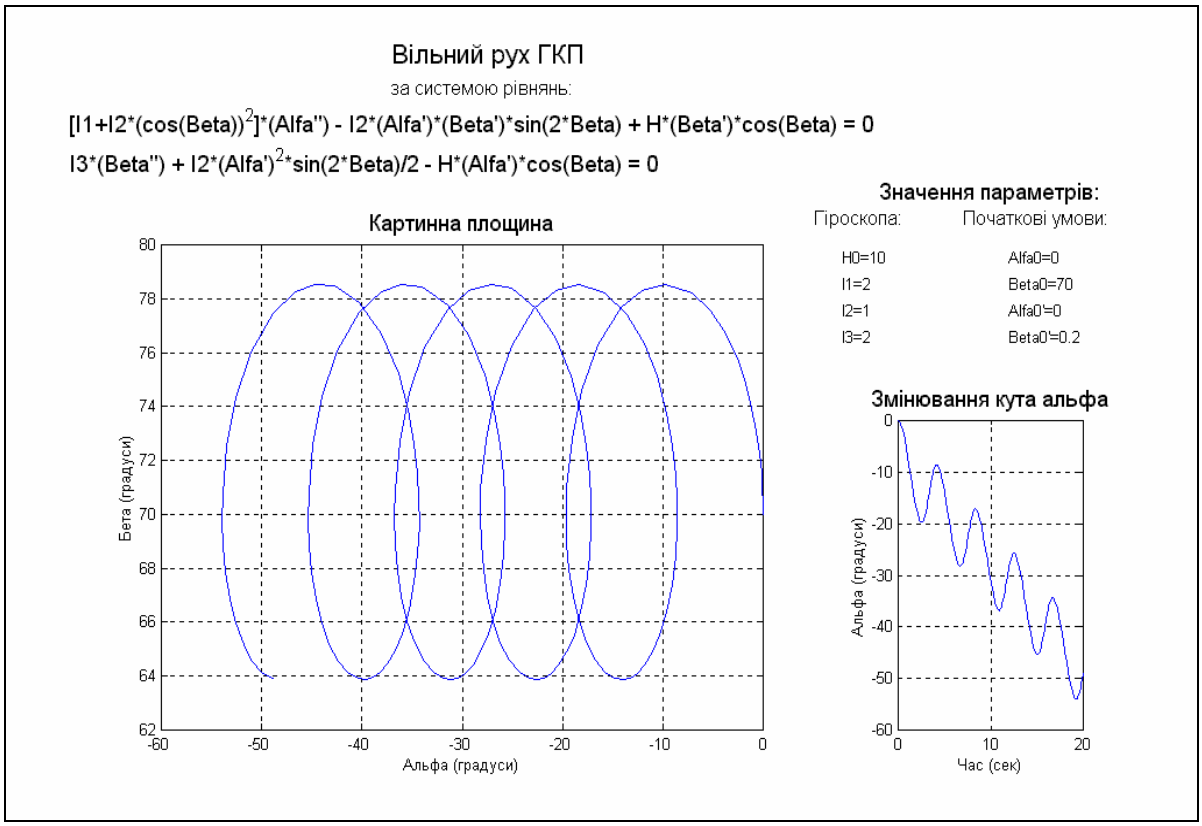

Рис. 7.83

```
 %      Оформление  текстового подокна  "Модель"
subplot(4,4,1:4);
axis('off');
h1=text(0.25,1.1,'Свободное движение ГКП','FontSize',16);
h1=text(0.25,0.85,'по системе уравнений:','Fontsize',12);
h1=text(-0.1,0.6,'[I1+I2*(cos(Beta))2]*(Alfa"")  -  I2*(Alfa")*(Beta")*sin(2*Beta)  +
H*(Beta")*cos(Beta) = 0','Fontsize',14);
h1=text(-0.1,0.3,'I3*(Beta"")  +  I2*(Alfa")2*sin(2*Beta)/2  -  H*(Alfa")*cos(Beta)  =
0','Fontsize',14);
 %      Оформление  текстового подокна  "Параметры"
subplot(4,4,8);
axis('off');
h1=text(-0.1,1.4,'Значения параметров:','Fontsize',14);
h1=text(-0.45,1.2,'Гироскопа:','Fontsize',12);
h1=text(0.35,1.2,'Начальные условия:','Fontsize',12);
h1=text(-0.3,0.9,sprintf('H0=%g',H0),'Fontsize',10);
h1=text(-0.3,0.7,sprintf('I1=%g',I1),'Fontsize',10);
```

```
h1=text(-0.3,0.5,sprintf('I2=%g',I2),'Fontsize',10);
h1=text(-0.3,0.3,sprintf('I3=%g',I3),'Fontsize',10);
h1=text(0.6,0.9,sprintf('Alfa0=%g',Alfa0),'Fontsize',10);
h1=text(0.6,0.7,sprintf('Beta0=%g',Beta0),'Fontsize',10);
h1=text(0.6,0.5,sprintf('Alfa0''=%g',Alfat0),'Fontsize',10);
h1=text(0.6,0.3,sprintf('Beta0''=%g',Betat0),'Fontsize',10);
```

Результаты, представленные на рис. 7.83, иллюстрируют нелинейное свойство гироскопа в кардановом подвесе - наличие даже в случае, когда отсутствуют моменты сил по осям подвеса (т. е. когда гироскоп является свободным), систематически (с постоянной угловой скоростью) нарастающего угла поворота оси гироскопа вокруг внешней оси карданового подвеса. Это так называемый "уход Магнуса".

7.4. Объединение S-моделей с программами MatLAB

Как уже отмечалось, моделирование процессов с помощью S-моделей, несмотря на весьма значительные удобства и преимущества, имеют и некоторые существенные недостатки.

К преимуществам использования SimuLink-моделей относятся:

- весьма удобный, наглядный и эффективный способ образования программ моделирования даже довольно сложных динамических систем – *визуальное* программирование, - путем сборки блок-схемы системы из "стандартных" готовых блоков;
- довольно удобные и наглядные средства вмешательства в готовую блок-схему системы с целью ее преобразования или получения дополнительной информации об изменении промежуточных процессов;
- широкий набор эффективных программ решателей (Solvers, интеграторов) дифференциальных уравнений (с фиксированным шагом интегрирования, с переменным шагом, а также решателей так называемых "жестких" систем дифференциальных уравнений);
- отсутствие необходимости в специальной организации процесса численного интегрирования;
- уникальные возможности интегрирования нелинейных систем с "существенными" нелінійностями (когда нелинейная зависимость имеет скачкообразный характер);
- весьма быстрое и удобное получение графической информации об изменении моделируемых величин по аргументу (времени).

Недостатками же использования S-моделей являются:

- жесткая и неудобная форма графического представления сигналов в блоках *Scope* и *XY Graph* (в отличие от средств среды MatLab);
- невозможность автоматической (программной) обработки полученных результатов многоразового моделирования одной или нескольких S-моделей;

- невозможность рациональной организации процесса изменения исходных данных S-модели и параметров ее блоков (например, в диалоговой форме).

Кроме того, для отдельных видов дифференциальных уравнений намного удобнее и быстрее составлять процедуры их правых частей в виде программы, чем составлять соответствующую блок-схему. Это довольно наглядно видно из сравнения, например, блок-схем гироскопа в кардановом подвесе (рис. 7.73...7.76) с его дифференциальными уравнениями (п. 7.3.2).

Указанное свидетельствует о том, что программная реализация процесса моделирования и моделирования в виде S-моделей имеют взаимодополняющие свойства. Желательно уметь объединять преимущества этих двух средств моделирования, соединяя программную реализацию с использованием S-моделей.

Очевидно, чтобы осуществить объединение программы с моделированием с помощью S-модели надо иметь в наличии:

- *средства передачи данных из среды MatLab в S-модель и обратно*;
- *средства запуска процесса моделирования S-модели из среды MatLab, а также изменения параметров моделирования из этого среды;*
- *средства создания S-блоков* не только из других готовых блоков, а и *путем использования программ языком MatLab*.

Рассмотрим подробнее эти средства, предоставленные системой MatLab.

7.4.1. Принципы функционирования блоков системы SimuLink

Каждый блок S-модели имеет такие характеристики (рис. 7.84):

- вектор входных величин ***u***;
- вектор выходных величин ***y***;
- вектор состояния ***x***.

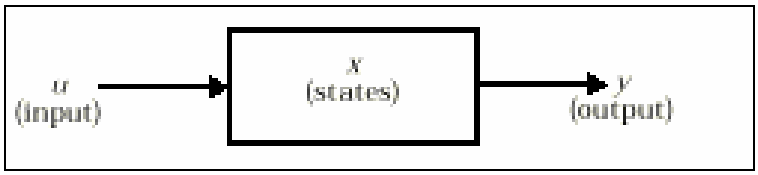

Рис. 7.84

Вектор состояния может состоять из непрерывных состояний x_c, дискретных состояний x_d, или комбинации их обоих. Математические связи между этими величинами могут быть представлены в виде уравнений:

$y = f_o(t, x, u)$ формирования выхода

$x_{d_{k+1}} = f_u(t, x, u)$ обновления (формирования нового значения) состояния

$\frac{dx}{dt} = f_d(t, x, u)$ формирования значений производной состояния

где $$x = \begin{cases} x_c \\ x_{d_k} \end{cases}$$

Моделирование состоит из двух фаз - инициализации и собственно моделирования. В фазе инициализации осуществляются такие действия:

- блочные параметры передаются в MatLAB для оценивания (вычисления); результаты числовых вычислений используются как фактические параметры блоков;
- иерархия модели сглаживается; каждая не условно выполняемая подсистема заменяется блоками, из которых она складывается;
- блоки сортируются в порядке, в котором их нужно изменять; алгоритм сортировки образует такой порядок, что любой блок с прямым подключением не изменяется, пока изменяются блоки, которые определяют входные величины; на этом шаге выявляются алгебраические циклы;
- проверяются связи между блоками, чтобы гарантировать, что длина вектора выхода каждого блока совпадает с ожидаемой длиной векторов входа блоков, которые управляются ним.

Собственно моделирование осуществляется путем численного интегрирования. Каждый из имеющихся в наличии методов интегрирования (ODE) зависит от способности модели определять производные ее непрерывных состояний. Вычисления этих производных является двухшаговым процессом. Сначала каждая выходная величина блока вычисляется в порядке, определенном в процессе сортировки. На втором шаге каждый блок вычисляет свои производные для текущего момента времени, входные переменные и переменные состояния. Полученный вектор производных используется для вычисления нового вектора переменных состояния в следующий момент времени. Как только новый вектор состояния вычислен, блоки данных и блоки - обзорные окна обновляются.

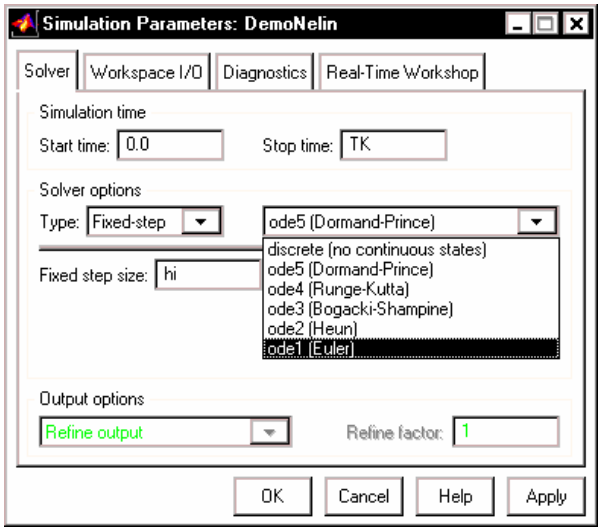

Рис. 7.85

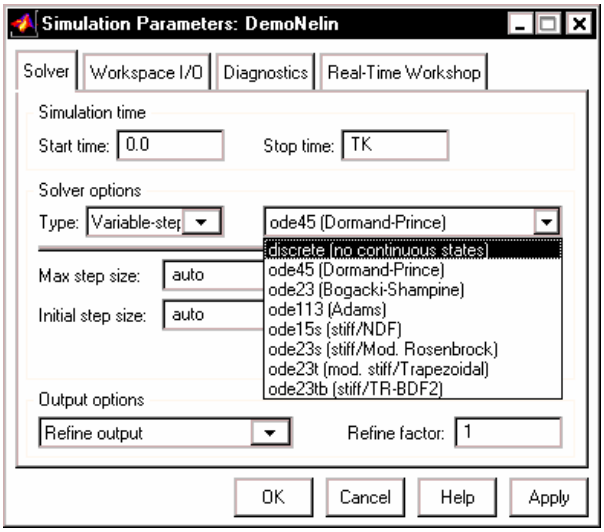

Рис. 7.86

С перечнем программ решателей (интеграторов), которые прилагаются к пакету SimuLink, можно ознакомиться через меню *Simulation/Parameters* (вкладка *Solvers*) – см. рис. 7.85…7.88.

Вкладка *Solvers* (рис. 7.85) содержит в верхней половине четыре окошка:

- *Start time,* – в котором устанавливается начальное значение аргумента (времени);
- *Stop time,* – где записывается конечное значение времени;
- *Type,* – где выбирается вид решателей;
- *Solver options* – в котором выбирается конкретный решатель.

Если в окошке *Type* выбран вид решателей *Fixed step* (с фиксированным шагом), в окошке *Solver options* появляется такой (рис. 7.85) набор решателей с фиксированным шагом:

- *discrete (no continuous states)* - дискретный (не непрерывные состояния);
- *ode5* – метод Дормана-Пренса (пятого порядка);
- *ode4* – метод Рунге-Кутта (четвертого порядка);
- *ode3* – метод Богацкого-Шампена (третьего порядка);
- *ode2* – метод Хойне (второго порядка);
- *ode1* – метод Ейлера (первого порядка).

При этом в нижней половине вкладки возникает окошко с надписью *Fixed step size* (Размер фиксированного шага), в которое нужно ввести значение шага интегрирования. В Matlab, начиная из версии 5.3 и выше, возникает еще одно окошко (рис. 7.87) под именем *Mode* (Режим), в котором выбирается один из трех возможных режимов работы:

- *Auto* – автоматический режим;
- *Single Tasking* – однозадачный режим;
- *Multi Tasking* – многозадачный режим.

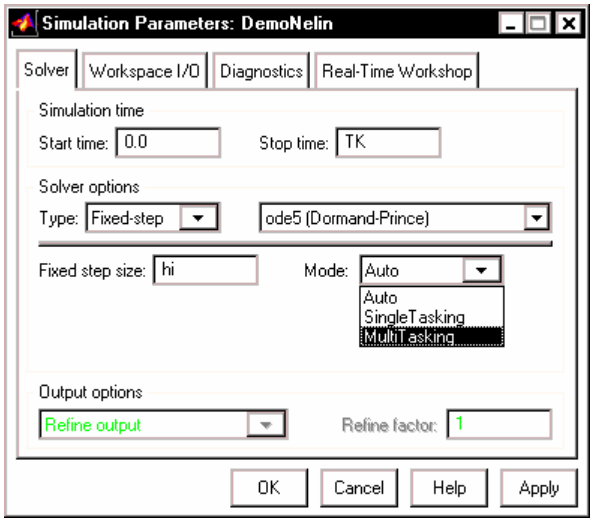

Рис. 7.87

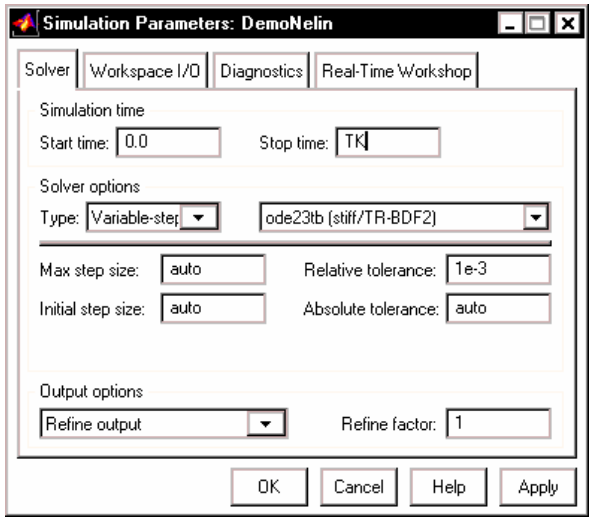

Рис.7.88

Если же выбрать тип решателя *Variable step*, в окошке *Solver options* (рис. 7.86) возникнет другая подборка интеграторов (методов численного интегрирования):

- *ode45* – метод Дормана-Пренса с переменным шагом;
- *ode23* – метод Багацкого-Шампена с переменным шагом;
- *ode113* – метод Адамса с переменным шагом;
- *ode15s* – жесткий метод NDF с переменным шагом;
- *ode23s* – жесткий метод Розенброка с переменным шагом;
- *ode23t* – жесткий метод трапеций с переменным шагом;
- *ode23tb* – жесткий метод TR/BDF2 с переменным шагом.

В этом случае в нижней половине вкладки возникают другие окошки:

- *Max step size* – максимальная величина шага;
- *Initial step size* - начальная величина шага;
- *Relative tolerance* – допустимая относительная погрешность;
- *Absolute tolerance* - допустимая абсолютная погрешность.

Первые две и последняя из этих величин по умолчанию установлены *auto*, т. е. устанавливаются автоматически и требуют задания лишь в случае, если у пользователя имеются весомые причины изменить их значение на некоторые определенные. Относительная точность (точнее, относительная погрешность) по умолчанию всегда устанавливается $1 \cdot 10^{-3}$. По желанию, можно установить ее другой.

7.4.2. Функции пересечения нуля

SimuLINK использует функцию пересечения нуля для того, чтобы обнаружить резкие изменнения в непрерывных сигналах. Эта функция играет важную роль:

- в управлении скачками состояния;
- при точном интегрировании прерывистых сигналов.

Система испытывает скачок состояния, когда изменение значений переменных состояния заставляет систему испытать значительные мгновенные изменения. Простой пример скачков состояния - мячик, отскакивающий от пола. При моделировании такой системы используется метод интегрирования с изменяемым шагом. Метод численного интегрирования обычно не предусматривает мер, которые бы позволили точно зафиксировать момент времени, когда мяч контактирует с полом. Вследствие этого при моделировании мяч, переходя через контактную точку, как бы проникает сквозь пол. SimuLINK использует функцию пересечения нуля, чтобы гарантировать, что момент скачка состояния системы определен точно (с машинной точностью). Вследствие того, что эти моменты скачков состояния определяются точно, во время моделирования не происходит никакого проникновения сквозь пол, и переход от отрицательной скорости мяча к положительной в момент контакта является чрезвычайно резким.

Чтобы увидеть демонстрацию подскакивающего мяча, введите в командном окне MatLAB команду ***bounce***. В результате выполнения этой команды возникнет окно с изображением блок-схемы модели поведения мяча (рис. 7.89). Эта модель численно интегрирует методом ***ode23*** (с автоматическим изменением шага интегрирования) дифференциальное уравнение

$$\ddot{x} = -g,$$

(g - ускорение свободного падения; x - текущая высота мяча над полом) в промежутках между моментами времени, соответствующими контактам мяча с полом.

Интегрирование осуществляется с помощью двух блоков-интеграторов. На выходе первого из них получают значение текущей скорости движения мяча, а на выходе второго - текущую высоту мяча.

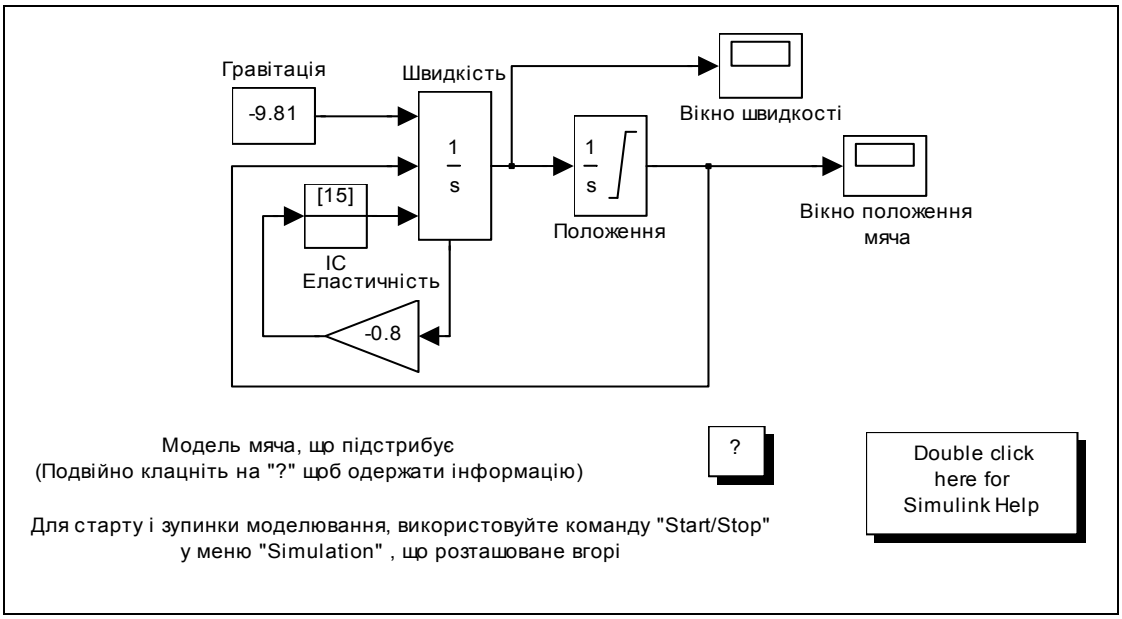

Рис. 7.89

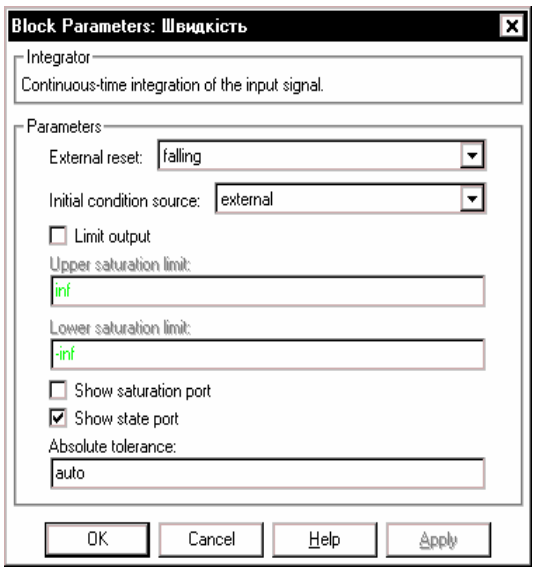

Рис. 7.90

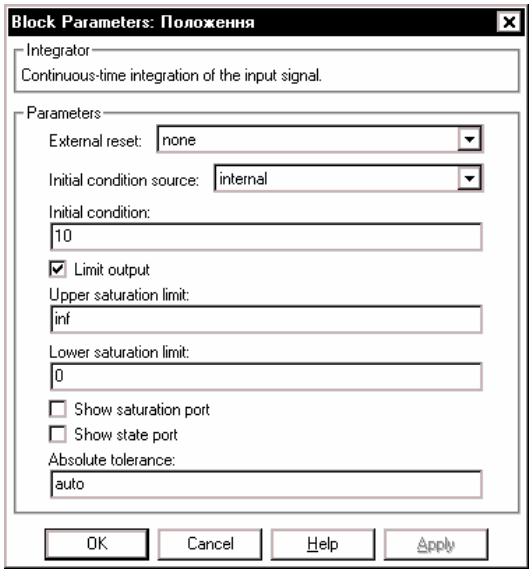

Рис. 7.91

Более близкое знакомство со вторым интегратором (рис.7.91) дает возможность убедиться, что в нем установлена нижняя граница (нуль) изменений высоты мяча, а в первом (рис. 7.90) введено внешнее управление (*falling* - при уменьшении) от выхода второго интегратора. Во втором интеграторе начальное условие установлено как внутреннее и равняется 10 (м), а в первом, - начальное условие является внешнм (15 м/с). Кроме того, в первом интеграторе установлены отображение порта состояния. На этот порт подаются значения текущей скорости.

Моделирование происходит так. Интегрирование начинается при указанных начальных условиях. Когда на втором интеграторе фиксируется пересечение мячом нуля высоты, на этом шаге осуществляется точное (с машинной точностью) вычисление момента времени, когда мяч пересекает пол, пересчитывается значение скорости в первом интеграторе на момент пересечения, и этот момент устанавливается как новый начальный момент времени. Найденное значение скорости через выходный порт первого интегратора (внизу блока интегратора) изменяет свой знак на противоположный, уменьшается по величине на 20 процентов (этим учитывается уменьшение скорости за счет потерь энергии вследствие неидеальной упругости мяча) и используется как новое начальное условие по скорости, и интегрирование продолжается при новых начальных условиях.

Следует отметить, что управление процессом прерывания интегрирования и продолжение его при новых начальных условиях, осуществляется вторым интегратором при пересечении величины на его выходе установленного уровня (нуля) при уменьшении (*falling* на первом интеграторе). При этом значение величины на выходе первого интегратора не может быть использовано для расчета нового ее начального значения. Надо обязательно для этой цели использовать дополнительный выходный порт интегратора, т. е. установить "галочку" в окошко *Show state port* (Показать порт состояния) и подать это рассчитанное значение на входной порт блока внешнего начального условия интегратора не непосредственно, а обязательно через блок ***IC*** начального условия.

Численные методы интегрирования обычно сформулированы в предположении, что сигналы, которые они интегрируют, являются непрерывными и имеют непрерывные производные.

В пакете *SimuLink* предусмотрены следующие блоки, использующие функции определения пересечения нуля:

- ***Abs*** (блок формирования абсолютного значения входа), - один раз: выявление момента, когда входной сигнал пересекает нуль в любом направлении – уменьшаясь или увеличиваясь;
- ***Backlash*** (блок формирования люфта), - дважды: когда входной сигнал сталкивается с верхним порогом и когда сталкивается с нижним порогом;
- ***Dead Zone*** (блок формирования зоны нечувствительности (мертвой зоны)), - дважды: когда сигнал входит в зону нечувствительности (до этого выходной сигнал равнялся входному сигналу минус нижняя граница зоны) и когда оставляет эту зону (выходный сигнал становится равным входному минус верхняя граница);
- ***Hit Crossing*** (блок улавливания пересечения), - один раз: выявляет момент, когда вход пересекает заданный уровень;
- ***Integrator*** (блок интегратора), - если представлен порт сброса, выявляет момент, когда сброс происходит; если выход ограничен, то тройное выявление пересечения нуля: когда достигается верхняя граница насыще-

ния, когда достигается нижняя граница насыщения и когда зона насыщения покинута;
- ***MinMax*** (блок поиска минимума или максимума входной величины), - один раз: для каждого элемента выходного вектора выявление момента, когда входной сигнал становится минимальным или максимальным;
- ***Relay*** (блок формирования релейного изменения выхода), - один раз: если реле выключено – выявление момента, когда его нужно включить; если реле включено – выявление момента, когда его нужно выключить;
- ***Relational Operator*** (блок операторов отношения), - один раз: выявление момента, когда выход изменяется;
- ***Saturation*** (блок насыщения), - дважды: один раз выявляет, когда верхний порог достигается или покидается, и один раз выявляет, когда нижний порог достигается или покидается;
- ***Sign*** (блок формирования функции sign от входа), - один раз: выявление момента прохождения входа через нуль;
- ***Step*** (блок формирования скачкообразного изменения выхода), - один раз: выявляет момент, когда будет осуществляться скачкообразное изменение.

Эти средства пакета SimuLink позволяют моделировать очень важные и труднопрограммируемые особенности поведения существенно нелинейных систем, таких как:
- "сцепление" подвижных частей механизмов силами сухого трения;
- удароподобные процессы и режимы с ними;
- скользящий режим;
- автоколебания;
- внезапные переходы от одного из режимов к другому.

7.4.3. Передача данных между средой MatLab и S-моделью

Прежде всего, укажем, что рабочее пространство среды MatLab всегда досягаемо для используемой S-модели. Это означает, что если в качестве значений параметров в окнах настройки блоков S-модели использованы некоторые имена, а значения этим именам предварительно присвоены в рабочем пространстве, то эти значения сразу передаются соответствующим блокам S-модели. Из этого следует простое правило:

чтобы организовать удобное (например, диалоговое) изменение параметров блоков S-модели достаточно

- *в окнах настраивания блоков S-модели в качестве параметров указать идентификаторы (имена) вместо чисел*;
- *организовать средствами среды MatLab (например, программно) присвоение численных значений этим идентификаторам, а также (в случае необходимости) диалоговое изменение их*;

- *после присвоения численных значений всем идентификаторам (например, путем запуска соответствующей М-программы) провести запуск на моделирование S-модели.*

Некоторые средства обмена данными были уже рассмотрены раньше. Это – блоки ***From Workspace*** раздела ***Sources*** и ***To Workspace*** раздела ***Sinks*** стандартной библиотеки SimuLink. Они служат: блок ***From Workspace*** - для подключения сигналов, которые предварительно получены в результате вычислений в среде MatLab, к процессу моделирования S-модели; блок ***To Workspace*** - для возможности записи результатов моделирования процессов, полученных путем моделирования на S-модели, в рабочее пространство среды MatLab.

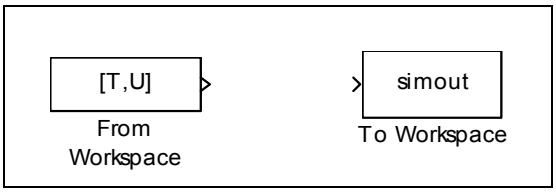

Рис. 7.92

На рис. 7.92 показанный внешний вид этих блоков. Рис. 7.93 и 7.94 представляют их окна настраивания.

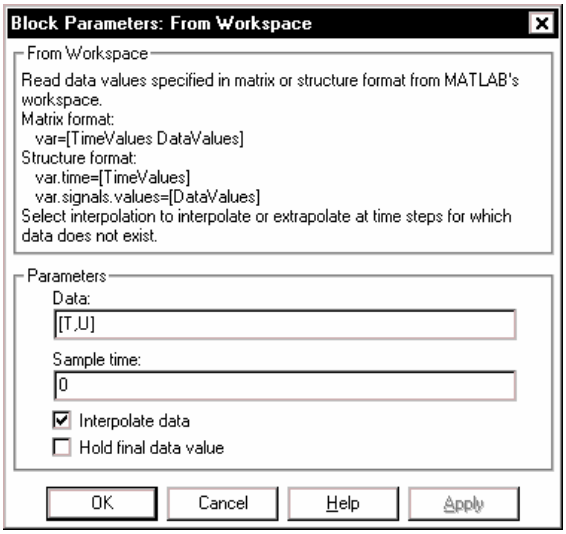

Рис. 7.93

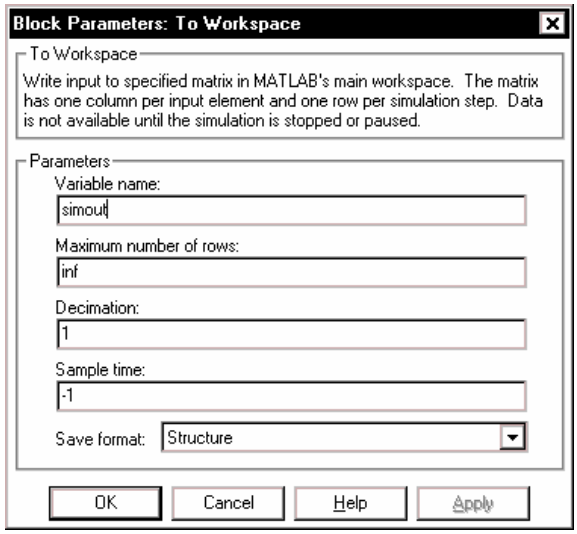

Рис.7.94

Как видим (рис. 7.93), для определения процесса, который будет использоваться в S-модели, следует указать в первом окошке *Data* вектор из двух имен – первого – имени массива значений аргумента (времени), в которые определен этот процесс, и второго - имени массива значений процесса при этих значениях аргумента. Для записи полученного процесса в рабочее пространство (рис. 7.94) следует указать в окошке *Variable name* имя, под которым его нужно сохранить в рабочем пространстве системы MatLab.

Те же операции можно осуществить еще проще, не используя эти блоки.

Чтобы подключить процесс, определенный в программе MatLab, как входной в S-модель, предусмотрен механизм включения портов входа и выхода.

Для этого нужно сделать следующее:

- в блок-схему S-модели вставить блок порта входа *In* и подключить к S-модели;
- в меню *Simulation* окна S-модели вызвать вкладку *Workspace I/O* (рис. 7.95) окна *Simulation Parameters* (команды *Parameters*);
- установить "галочку" в правом окошке рядом с надписью *Input*, записав по левую сторону от этой надписи вектор с двумя именами – первое – имя вектора значений аргумента, второе - имя вектора значений входного сигнала при этих значениях аргумента;
- установить значение этих векторов в среде MatLab, например, таким образом

```
t = (0:0.1:1)';
u = [sin(t), cos(t), 4*cos(t)];
```

- обратиться к выполнению моделирования S-модели.

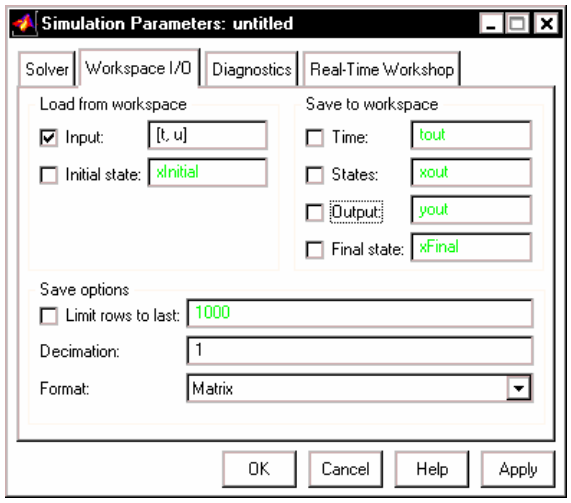

Рис. 7.95

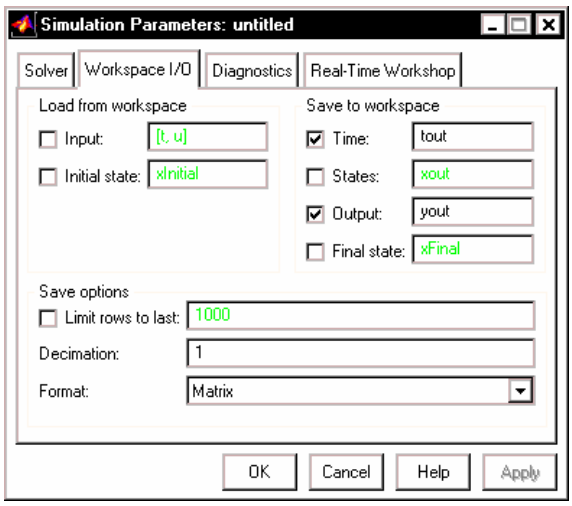

Рис.7.96

Наоборот, чтобы вывести некоторые сигналы, которые формируются в S-модели, в рабочее пространство MatLab, нужно:

- в блок-схему S-модели вставить блоки портов выхода *Out* и подсоединить к ним необходимые выходные величины других блоков;
- в меню *Simulation* окна S-модели вызвать вкладку *Workspace I/O* (рис. 7.96) окна *Simulation Parameters* (команды *Parameters*);
- установить "галочку" в правом окошке с надписью *Time*;
- установить "галочку" в правом окошке с надписью *Output.*

В этом случае значения времени будут записываться в рабочее пространство как вектор с именем *tout*, а соответствующие значения выходных процессов при этих значениях времени – в столбце матрицы *yout* (в первые столбцы – процессы, которые поданы на первый выходной порт *Out1,* далее, - процессы, которые поданы на второй порт *Out2* и т.п.). Конечно, если изменить имена, которые

записаны по правую сторону от надписей соответствующих окошек, то эти же данные будут записаны под этими именами.

Так же, активизируя (рис. 7.96) окошко *Initial state* (начальные значения переменных состояния), можно ввести в S-модель начальные значения переменных состояния системы. Активизировав окошко *States* (Переменные состояния), можно записать текущие значения переменных состояния системы в рабочее пространство под именем *xout* (или другим именем, если его записать по правую сторону от этой надписи).

Наконец, можно записать и конечные значения переменных состояния в вектор *xFinal*, если активизировать окошко *Final state*.

7.4.4. Запуск процесса моделирования S-модели из среды MatLab

Рассмотрим теперь средства, которые позволяют запускать на моделирование созданные S-модели из программы MatLab.

S-модель запускается на выполнение, если в программе MatLab вызвать процедуру ***sim*** по следующему образцу

[t,x,y1, y2, ..., yn] = ***sim****(model,timespan,options,ut),*

где *model* – текстовая переменная, являющаяся именем mdl-файла, который содержит запись соответствующей S-модели; *timespan* – вектор из двух элементов, - значения начального и конечного моментов времени моделирования; *option*s – вектор значений опций интегрирования; устанавливается процедурой ***simset***:

options = ***simset****('Свойство1', Значения1,' Свойство2', Значения2, ...);*

t – массив выходных значений моментов времени; *x* – массив (вектор) переменных состояния системы; *y1* – первый столбец матрицы выходных переменных системы (которые подаются на выходные порты) и т.д.

Устанавливать (и изменять) параметры решателя и процесса интегрирования в программе MatLab можно при помощи функции ***simset*** как это показано выше. Так можно установить значение следующих (среди прочих) свойств решателя или интегрирования:

'Solver', - название решателя; значения, которые может принять это свойство, может быть одним из следующих (указывается в апострофах): ode45 | ode23 | ode113 | ode15s | ode23s – для интегрирования с автоматически изменяемым шагом, а для фиксированного шага - ode5 | ode4 | ode3 | ode2 | ode1;

'RelTol' – относительная допустимая погрешность; значение может быть положительным скаляром; по умолчанию устанавливается 1e-3;

'AbsTol' – абсолютная допустимая погрешность; значение может быть положительным скаляром; по умолчанию устанавливается 1e-6;

'FixedStep' – фиксированный шаг (положительный скаляр);

'MaxOrder' – максимальный порядок метода (применяется лишь для метода *ode15*); может быть одним из целых 1 | 2 | 3 | 4 |; по умолчанию равняется 5;

'MaxRows' – максимальное количество строк в выходном векторе; неотрицательное целое; по умолчанию равно 0;

'InitialState' – вектор начальных значений переменных состояния; по умолчанию он пустой ([]);

'FinalStateName' – имя вектора, в который будет записываться конечные значения вектора состояния модели; символьная строка, по умолчанию – пустое ('');

'OutputVariables' – выходные переменные; по умолчанию имеет значение {txy}; возможные варианты | tx | ty | xy | t | x | y; все они неявно указывают, какие именно выходные переменные не будут выводиться.

7.4.5. Образование S-блоков путем использования программ MatLab. S-функции

В системе MatLab предусмотрен механизм преобразования некоторых процедур, написанных языками высокого уровня, в блок SimuLink-модели. Осуществляется этот механизм с помощью так называемых S-функций.

S-функция – эта относительно самостоятельная программа, которая написана на языке MatLab или C. Главное назначение S-функции – решать следующие задачи:

- образования новых блоков, которые дополняют библиотеку пакета SimuLink;
- описания моделируемой системы в виде системы математических уравнений;
- включения ранее созданных программ на языке C или MatLab в S-модель.

Программа S-функции имеет определенную четкую структуру. Для случая, когда S-функция создается на основе M-файла, эта структура приведена в виде файла ***SfunTMPL.m*** в директории TOOLBOX\SIMULINK\BLOCKS. Из рассмотрения этого файла-шаблона вытекает, что заголовок S-функции в общем случае может иметь следующий вид:

function [sys,x0,str,ts] = **<*Имя_S-функции*>** (t,x,u,flag{, <Параметры>})

Стандартными аргументами S-функции являются:

t – текущее значение аргумента (времени);

x – текущее значение вектора переменных состояния;

u – текущее значение вектора входных величин;

flag – целочисленная переменная, отражающая форму представления результатов действия S-функции;

<Параметры> - дополнительные идентификаторы, которые характеризуют значения некоторых параметров системы, которые используются в S-функции; наличие их не является обязательным.

В результате вычислений, осуществляемых при работе S-функции, получают значение такие переменные:

sys – системная переменная, содержание которой зависит от значения, которое приобретает переменная flag;

x0 – вектор начальных значений переменных состояния;

str – символьная переменная состояния (обычно она пуста []);

ts – матрица размером (m*2), которая содержит информацию о дискретах времени.

Текст S-функции состоит из текста самой S-функции и текстов собственных (внутренних) подпрограмм, которые она вызывает, а именно:

mdlInitializeSizes, устанавливающей размеры переменных S-функции и начальные значения переменных состояния;

mdlDerivatives, используемой как процедура правых частей системы дифференциальных уравнений модели в форме Коши в случае, когда переменные состояния объявлены как непрерывные;

mdlUpdate, которая используется как процедура обновления на следующем интервале дискрета времени значений переменных состояния, которые объявлены как дискретные;

mdlOutputs, которая формирует вектор значений выходных переменных в блоке S-функции;

mdlGetTimeOfNextVarHit – вспомогательная функция, которая используется для определения момента времени, когда определенная переменная состояния пересекает заданный уровень;

mdlTerminate – функция, которая завершает работу S-функции.

В зависимости от типа уравнений (алгебраические, или дифференциальные, или разностные), которыми описывается блок, который моделируется через S-функцию, некоторые из указанных функций не используются. Так, если блок описывается алгебраическими уравнениями, то не используются почти все внутренние указанные процедуры, за исключением процедуры ***mdlOutputs***, в которой и вычисляются соответствующие алгебраические соотношения, определяющие связь между входными переменными ***u*** и выходными ***y***. В случае, когда поведение блока описывается системой непрерывных дифференциальных уравнений, не используется процедура ***mdlUpdate***, а если уравнения блока являются разностными, не используется функция ***mdlDerivatives***. Обязательными являются лишь процедуры инициализации ***mdlInitializeSizes*** и формирования выхода ***mdlOutputs***.

Главная процедура S-функции содержит, главным образом, обращения к той или другой внутренней процедуре в соответствия со значением переменной *flag* на манер нижеследующего:

```
switch flag,
 case 0,
        [sys,x0,str,ts]=mdlInitializeSizes;
 case 1,
            sys=mdlDerivatives(t,x,u);
 case 2,
            sys=mdlUpdate(t,x,u);
 case 3,
            sys=mdlOutputs(t,x,u);
 case 4,
```

```
            sys=mdlGetTimeOfNextVarHit(t,x,u);
 case 9,
            sys=mdlTerminate(t,x,u);
 otherwise
            error(['Unhandled flag = ',num2str(flag)]);
end
```

В соответствии со значением переменной *flag* выполняется та или другая внутренняя процедура:

flag=0 – выполняется инициализация блока S-функции;

flag=1 – осуществляется обращение к процедуре правых частей непрерывных дифференциальных уравнений;

flag=2 – вычисляются новые значения переменных состояния на следующем шаге дискретизации (для дискретной S-функции);

flag=3 – формируется значения вектора выходных величин блока S-функции;

flag=4 – формируется новое значение модельного времени, которое отсчитывается от момента пересечения заданного уровня определенной переменной состояния;

flag=9 – прекращается работа блока S-функции.

Установление и изменение значения переменной *flag* осуществляется автоматически, без вмешательства пользователя, в соответствии с логикой функционирования блоков SimuLink при моделировании (см. п. 7.4.1).

Итак, использование S-функции позволяет моделировать работу как обычных алгебраических, так и динамических (непрерывных или дискретных) звеньев.

Процесс ***образования блока S-функции*** состоит из следующих этапов:

1) написания текста S-функции, например, в виде M-файла; текст составляется на основе файла-шаблона ***SfunTMPL.m*** с учетом заданных уравнений поведения блока;
2) перетаскивания из библиотеки SimuLink (*SimuLink> Functions & Tables>*) стандартного блока ***S-function*** (рис. 7.97) в окно блок-схемы, внутри которой будет создаваться новый S-блок;

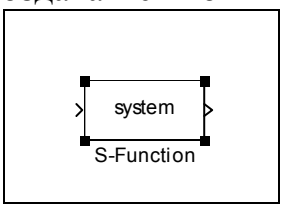

Рис. 7.97

3) двойного нажатия мышью на изображении этого блока, что приводит к возникновению на экране нового окна (рис. 7.98), содержащего два окошка ввода:

S-function name (имя S-функции), в которое вводится имя новой написанной S-функции;

S-function parameters (параметры S-функции), в которое вводятся имена или значения тех параметров блока, которые указаны в разделе <Параметры> составленного М-файла S-функции;

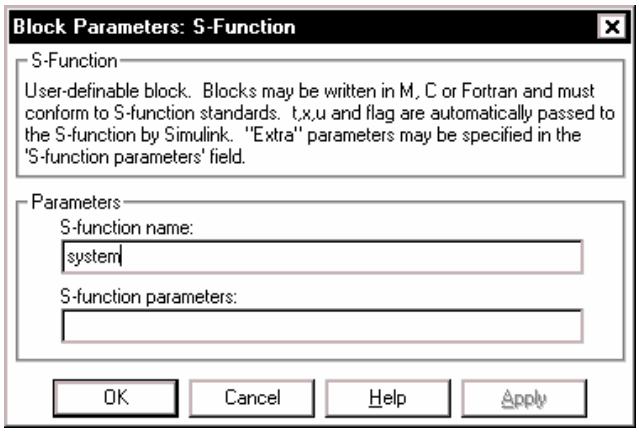

Рис. 7.98

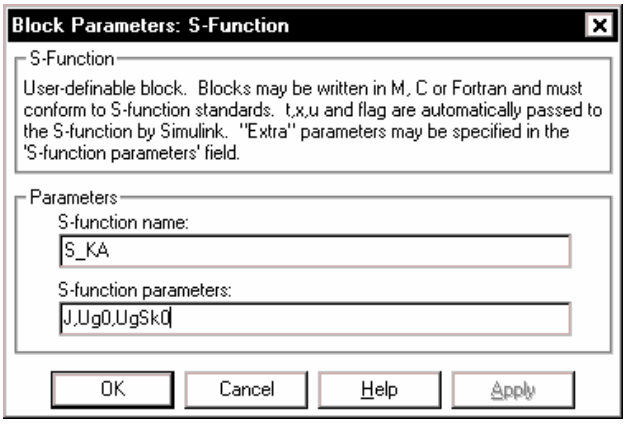

Рис. 7.99

4) введения в указанные окошки имени написанного М-файла S-функции и списка значений параметров S-функции; если, например, ввести в качестве имени ***S_KA***, а в окошко параметров, - строку ***J, Ug0, UgSk0***, то это окно изменит свой вид на следующий (рис. 7.99):

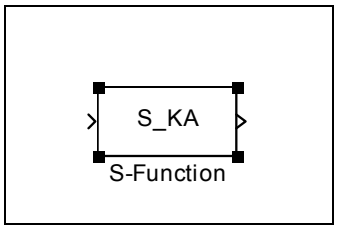

Рис. 7.100

5) теперь нужно нажать мышью на кнопку <OK> этого окна; если система обнаружит М-файл с введенным именем в достижимых ей путях, окно (рис. 7.99) исчезнет, а изображение блока в окне блок-схемы изменит свой вид (рис. 7.100) – внутри него возникнет введенное имя S-функции.

S-блок на основе S-функции, которая содержится в M-файле по имени S_KA.m, создан.

Теперь его можно использовать как обычный S-блок в блок схеме S-модели. При этом на вход этого блока должен поступать векторный сигнал ***u***. Выходом блока является также векторный сигнал ***y***, который сформирован S-функцией во внутренней процедуре ***mdlOutputs***.

7.4.6. Пример создания S-функции

Образуем S-функцию, которая реализует динамические свойства твердого тела в его вращательном движении. Для описания динамики тела воспользуемся динамическими уравнениями Ейлера в матричной форме:

$$\mathbf{J}\cdot\frac{d\boldsymbol{\omega}}{dt}+(\boldsymbol{\omega}\times)\cdot(\mathbf{J}\cdot\boldsymbol{\omega})=\mathbf{M}, \tag{7.4}$$

здесь $\mathbf{J}$ - матрица моментов инерции тела относительно осей, связанных с телом; $\boldsymbol{\omega}$ - матрица-столбец из проекций абсолютной угловой скорости тела на одни и те же оси; $(\boldsymbol{\omega}\times)$ - кососиметрична матрица вида

$$(\boldsymbol{\omega}\times)=\begin{bmatrix}0 & -\omega_z & \omega_y\\ \omega_z & 0 & -\omega_x\\ -\omega_y & \omega_x & 0\end{bmatrix}, \tag{7.5}$$

составленная из тех же проекций; $\mathbf{M}$ - матрица-столбец из проекций вектора момента внешних сил на связанные оси.

Создадим M-файл соответствующей S-функции. Назовем его ***S_DUE.m*** :

Файл ***S_DUE.m***

```
function [sys,x0,str,ts] = S_DUE(t,x,M,flag,J,UgSk0)
 % S-функция S_DUE Динамических Уравнений Ейлера
 % Реализует динамику вращательного движения твердого тела,
 % отыскивая вектор абсолютной угловой скорости тела
 % по заданному вектору моментов внешних сил,
 % действующих на тело
 % ВХОД блока:
 %       M - вектор проекций момента внешних сил на оси
 %             X, Y i Z связанной с телом системы координат
 % ВЫХОД блока:
 %       y - вектор из шести элементов: первые три - проекции
 %             абсолютной угловой скорости om тела на указанные оси,
 %             последние три - проекции на те же оси
 %             углового ускорения тела
 % Входные ПАРАМЕТРЫ S-функции:
 %       J - матрица моментов инерции тела в указанных осях;
 %       UgSk0 - вектор начальных значений проекций
 %                   угловой скорости тела

 % Лазарев Ю.Ф., Украина, 18-12-2001

IJ=inv(J);        % вычисление обратной матрицы моментов инерции
switch flag,
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(UgSk0);
```

```
  case 1,
    sys = mdlDerivatives(t,x,M,J,lJ);
  case 3,
    sys = mdlOutputs(x,M,J,lJ);
  case 9
    sys = [];
end
 %           Конец процедуры
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(UgSk0,Ug0)
sizes = simsizes;
sizes.NumContStates  = 3;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = UgSk0;
str = [];
ts  = [0 0];
 % Конец процедуры mdlInitializeSizes
 %=======================================
function z = mdlDerivatives(t,x,M,J,lJ)
 % ВХОДНОЙ вектор "M" является вектором проекций моментов внешних сил,
 %    действующих на космический аппарат соответственно по осям X Y Z
 % x(1)=om(1); x(2)=om(2); x(3)=om(3);
 % z(1)=d(om(1))/dt; z(2)=d(om(2))/dt; z(3)=d(om(3))/dt;
 %     |Jx    Jxy   Jxz|
 % J=|Jxy   Jy    Jyz| - матрица моментов инерции КА
%     |Jxz  Jyz    Jz  |
om=x(1:3); omx=vect2ksm(om);
z=lJ*(M-cross(om,J*om));        % ДИНАМИЧЕСКИЕ уравнения Ейлера
 % Конец процедуры mdlDerivatives
%================================
function y = mdlOutputs(x,M,J,lJ)
y(1:3)=x;
om=x(1:3);
zom=lJ*(M-cross(om,J*om))    ;     % Определения УСКОРЕНИЙ
y(4:6)=zom;
 % Конец процедуры mdlOutputs
```

В качестве входного вектора создаваемого S-блока принят вектор M, состоящий из трех значений текущих проекций момента внешних сил, которые действуют на тело, на оси системы декартовых координат, связанной с телом. Образуем выходной вектор ***y*** из шести элементов: первые три – текущие значения проекций абсолютной угловой скорости тела, вторые три, - проекции на те же оси абсолютного углового ускорения тела:

$$\boldsymbol{y}=[\text{omx,omy,omz,epsx,epsy,epsz}].$$

Создаваемый S-блок рассматривается как непрерывная система (с тремя непрерывными состояниями ***x***=[omx,omy,omz]). Поэтому в тексте S-функции изъята процедура ***mdlUpdate*** и оставлена процедура ***mdlDerivative***, которая фактически является подпрограммой правых частей динамических уравнений Ейлера.

Создадим новое (пустое) окно блок-схемы SimuLink. Перетянем у него стандартный блок S-функции из раздела *Functions & Tables*.

Дважды щелкнув мышью на изображении этого блока, вызовем его окно настраивания и запишем в него название М-файла созданной S-функции и его параметры (рис. 7.101). Нажмем кнопку <OK>.

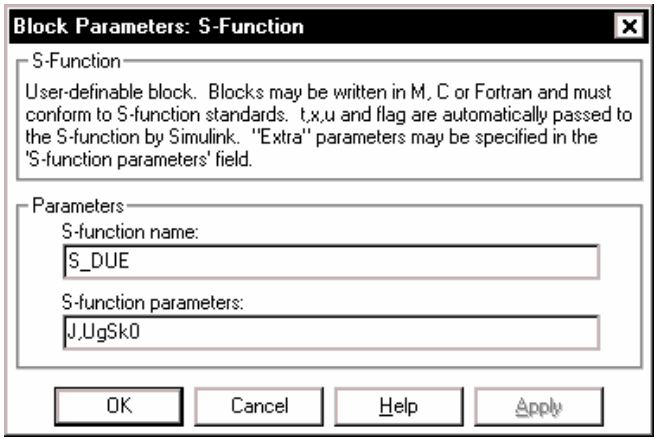

Рис. 7.101

Вследствие этого на изображении блока возникнет надпись имени S-функции, а окно настраивания исчезнет.

Теперь в том же окне с S-блоком S-функции создадим цепь для проверки правильности его работы.

Но для этого предварительно нужно продумать условия тестового примера.

Рассмотрим, например, такой случай:

- на тело не действуют внешние силы, т. е. тело свободно вращается в пространстве;
- оси декартовой системы координат, которая связана жестко с телом, направлены по главным осям инерции тела; при таком условии матрица моментов инерции будет диагональной;
- тело является динамично симметричным, а его ось фигуры направлена вдоль второй оси (Y) связанной системы координат; это означает, что матрица моментов инерции будет иметь вид:

$$\mathbf{J} = \begin{bmatrix} J_e & 0 & 0 \\ 0 & J & 0 \\ 0 & 0 & J_e \end{bmatrix},$$

где обозначено: J_e - экваториальный момент инерции, J - момент инерции тела относительно его оси фигуры (осевой момент инерции тела), причем $J > J_e$ в случае, если тело является сплющенным в направлении оси Y;

- тело предварительно "раскручено" с угловой скоростью Ω вокруг оси его фигуры и имеет незначительную (по сравнению с Ω) начальную угловую скорость ω_0 вокруг оси X.

Рассмотрим теоретически движение тела при этих условиях. Уравнения Ейлера в этом случае приобретут следующий вид:

$$\begin{cases} J_e \cdot \dfrac{d\omega_X}{dt} = (J - J_e) \cdot \omega_Y \cdot \omega_Z \\ J \cdot \dfrac{d\omega_Y}{dt} = 0 \\ J_e \cdot \dfrac{d\omega_Z}{dt} = -(J - J_e) \cdot \omega_Y \cdot \omega_X \end{cases}$$

и имеют такие решения при заданных начальных условиях:

$$\omega_X = \omega_0 \cdot \cos(k\Omega t); \qquad \omega_Y = \Omega; \qquad \omega_Z = -\omega_0 \cdot \sin(k\Omega t), \qquad (7.6)$$

причем

$$k = \frac{J - J_e}{J_e}. \qquad (7.7)$$

Итак, если для образованной модели обеспечить указанные условия, то, если она правильна (адекватна), при моделировании должны получить результаты соответствующие (7.6).

Добавим в блок-схему блок констант, который формирует нулевой вектор моментов внешних сил (рис. 7.102), а также блоки *Scope* по которым можно проконтролировать результаты моделирования в виде зависимостей от времени проекций угловой скорости и углового ускорения тела.

Перед началом моделирования нужно присвоить значение матрицы моментов инерции. Это можно сделать в командном окне MatLab, вводя строку

J=diag([400,600,400])

Результатом будет возникновение в том же окне записи

```
J =
  400    0    0
    0  600    0
    0    0  400
```

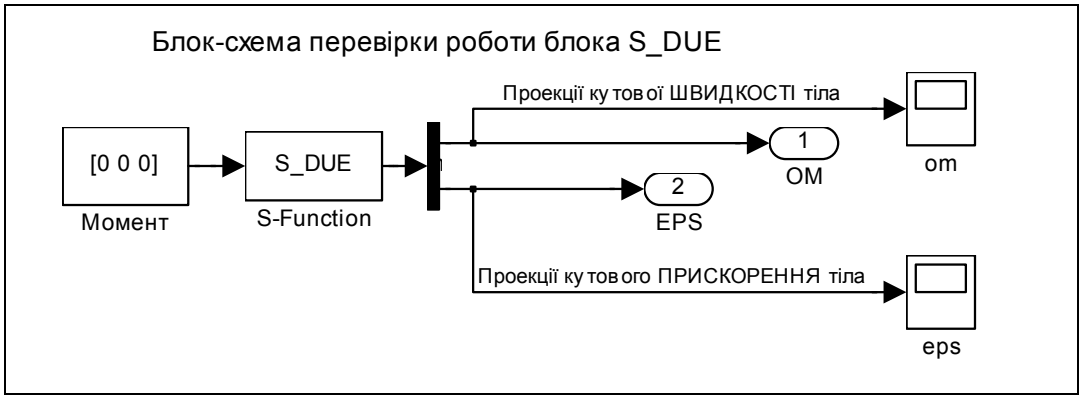

Рис. 7.102

Аналогично нужно ввести вектор начальных условий

UgSk0=[0.001 0.01 0]

Получим

```
UgSk0 =  1.0000e-003  1.0000e-002          0
```

Теперь следует перейти к окну блок-схемы, установить параметры интегрирования, указанные на рис. 7.103 и запустить блок схему на моделирование

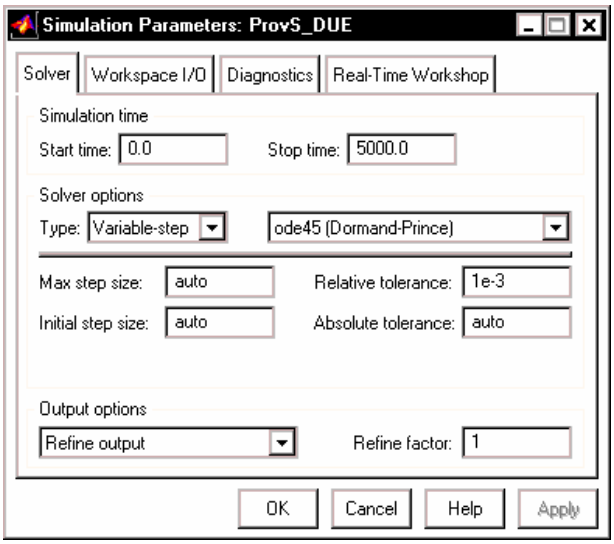

Рис. 7.103

По окончании процесса моделирования, обращаясь к окнам блоков *Scope,* можно убедиться, что созданная модель работает полностью адекватно.

Но убедить в этом читателя довольно трудно из-за следующих обстоятельств. Графические результаты блоков *Scope* представлены на черном поле. Поэтому скопировать соответствующее графическое окно означает получить некачественное изображение на бумаге, вдобавок израсходовав нерациональное количество красящего материала. Можно скопировать изображение этого окна при помощи команды печати в графическом окне блока *Scope*, но тогда соответствующее изображение займет целый лист, его невозможно уменьшить средствами текстового редактора. Вдобавок, линии на графике после печати на черно-белом принтере, не будут отличаться один от другого. На них невозможно нанести надписи, чтобы указать особенности кривых, нельзя довольно просто изменить стиль линии.

Все это говорит о том, что наиболее рациональн передать результаты в рабочее пространство с помощью введения выходных портов (рис. 7.102) и подания на них тех сигналов, которые нужно представить графически, а потом построить необходимые графики, пользуясь огромными графическими возможностями MatLab.

Последнее можно сделать непосредственно, пользуясь командами MatLab в его командном окне, но более рационально сделать это программно, причем желательно объединить в этой программе все действия:

- введение значений параметров, начальных условий и т.п.;
- установление параметров интегрирования;
- обращение к S-модели и запуск ее на моделирование;

- обработку полученных результатов моделирования, построение и оформление графиков.

Пример такой программы приведен ниже.

```
% Prov_DUEupr.m
% Управляющая программа
% для запуска модели Prov_DUE.mdl

% Лазарев Ю.Ф.  18-12-2001

J=[400 0 0; 0 600 0; 0 0 400]; % Введения значений матрицы инерции
UgSk0=[0.001 0.01 0]; % Введения начальных значений
                                % проекций угловой скорости тела

% Установления параметров моделирования
options=simset('Solver','ode45','RelTol',1e-6);
sim('Prov_DUE',5000,options); % МОДЕЛИРОВАНИЯ на S-модели

% Формирования данных и вывод ГРАФИКОВ
tt=tout;
omx=yout(:,1); omy=yout(:,2); omz=yout(:,3);
epsx=yout(:,4);     epsy=yout(:,5);epsz=yout(:,6);

subplot(2,1,1)
h=plot(tt,omx,tt,omy,'.',tt,omz,'--');grid
set(h,'LineWidth',2);
set(gca,'LineWidth',2)
set(gca,'FontAngle','italic','Fontsize',16)
title(' Проекции угловых скоростей')
ylabel('радианы в секунду')
legend('omx','omy','omz',0)

subplot(2,1,2)
h=plot(tt,epsx,tt,epsy,'.',tt,epsz,'--');grid
set(h,'LineWidth',2);
set(gca,'LineWidth',2)
set(gca,'FontAngle','italic','Fontsize',16)
title('Проекции угловых ускорений ')
ylabel('1/c2)')
xlabel('Время (c)')
set(gcf,'color','white')
legend('epsx','epsy','epsz',0)
```

Обратившись к этой программе, получим графики, приведенные на рис. 7.104.

Теперь читатель может наглядно убедиться в правильности модели.

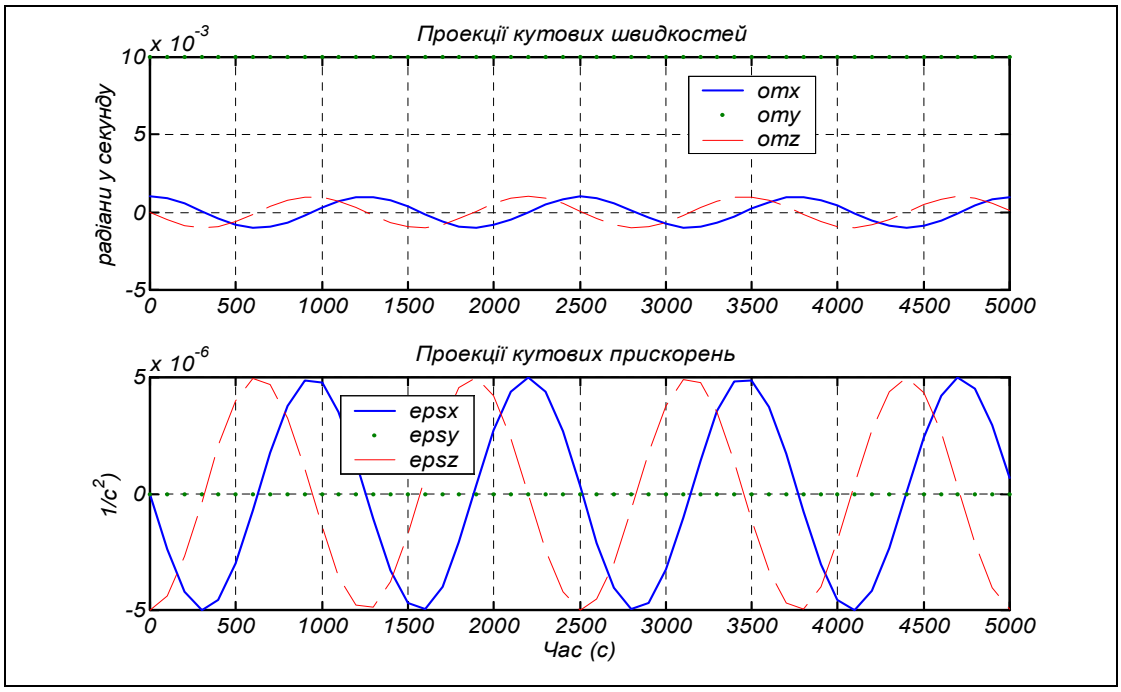

Рис. 7.104

Следует указать еще один, более удобный, способ объединения S-модели с программами на языке MatLab, который состоит в возможности вызова m-файлов непосредственно из S-модели специально предусмотренными для этого средствами.

Например, предыдущий (перед началом загрузки S-модели) вызов m-файла "PERVdan.m", содержащий операции присваивания исходных значений всех данных, которые используются при моделировании в S-модели по имени "MODEL.mdl", можно осуществить, если при создании S-модели с этим именем в командном окне MatLab ввести команду

```
set_param('MODEL','PreLoadFcn','PERVdan')
```

Если после выполнения этой команды записать на диск эту S-модель, то при дальнейших ее вызовах сначала автоматически будет вызван и выполнен файл ***PERVdan.m*** и лишь после этого на экране возникнет блок-схема S-модели, готовая к моделированию.

Проверить, какой именно m-файл используется в данной S-модели как предварительно выполняемый, можно путем привлечения команды

```
get_param('имя S-модели','PreLoadFcn')
```

Если же нужно вызвать некоторый m-файл перед или после проведения собственно моделирования на S-модели (например, нужно вызвать программу, которая позволяет изменить установленные значения параметров модели в диалоговом режиме, или использовать программу вывода результатов моделирования в графической форме), можно установить на свободном месте блок-схемы пустые блоки ***Subsystem*** (из раздела ***Sygnals & System***), каждый из которых будет осуществлять вызов соответствующего m-файла.

После установления блока ***Subsystem*** в поле блок-схемы модели следует щелкнуть правой клавишою мышки на его изображении. В окне, которое возник-

нет, следует выбрать команду *Block properties*. Возникнет окно настраивания блока, изображенное на рис. 7.105.

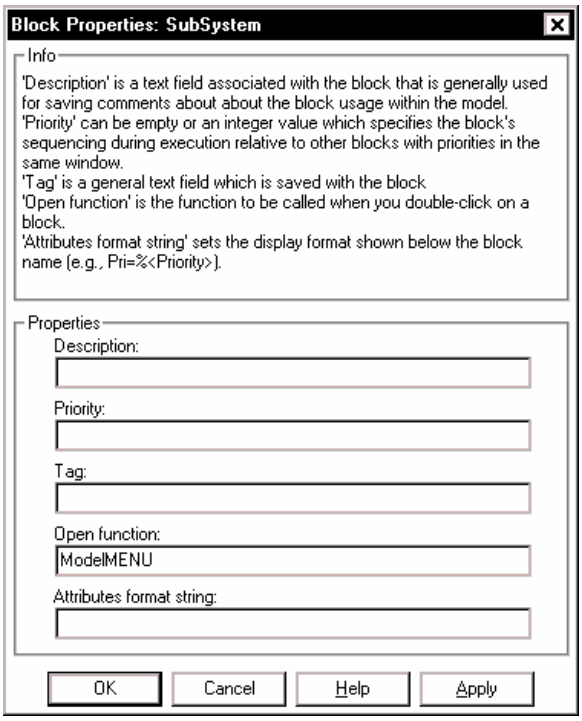

Рис. 7.105

Теперь достаточно внести имя m-файла (например, как на рис. 7.105, - ***ModelMENU***) в окошко *Open function* окна настраивания блока, возвратиться в окно блок-схемы и сделать надпись на установленном блоке ***Subsystem***, которая раскрывала бы его назначение.

Если m-файл, имя которого вписано в окно настраивания блока ***Subsystem***, существует, то *его выполнение теперь можно осуществить, дважды щелкнув мышью на изображении этого блока*.

Осуществление связи S-модели с определенными m-файлами описанным способом, наверное, является, пожалуй, наиболее удобным, так как, во-первых, позволяет вызвать m-файлы лишь в случае необходимости и в произвольном порядке, а во-вторых, все управление моделированием и вызовом программ осуществляется только из блок-схемы S-модели.

7.5. Создание библиотек S-блоков пользователя

Когда серьезный пользователь углубленно занимается моделированием систем, он неизбежно, рано или поздно, соприкоснется с необходимостью подготовки собственных блоков, имеющих свойства стандартных библиотечных блоков пакета SimuLink. Потребность в этом возникает, когда пользователь при разных задачах моделирования в собственной предметной области вынужден или неоднократно использовать элементарные созданные ним блоки, которые являются оригинальными и не входят в состав стандартных библиотек SimuLink, или использовать одни и те же блоки многократно в определенных устойчивых их соединениях. В таких случаях время создания новой модели можно значительно сократить и при этом предотвратить многочисленные ошибки, если оформить эти новые блоки или соединения блоков в виде новых блоков и разместить их в библиотеке.

Преимущество использования собственных блоков в составе библиотек состоит в следующем:

- их возможно использовать неоднократно путем перетягивания изображения блока из библиотеки в окно блок-схемы модели;
- пользоваться ими удобнее всего, когда общение с ними осуществляется через специальные диалоговые окна настраивания блоков, аналогичные тем, которые рассматривались при описании стандартных блоков SimuLink.

Создания окон настраивания блоков осуществляется через так называемую *маскировку* блока, т. е. создание *маски блока*.

7.5.1. Создание библиотеки

Рассмотрим процесс создания новой собственной библиотеки S-блоков на конкретных примерах.

Образование новой библиотеки начинается с обращения к браузеру SimuLink через предпоследнюю пиктограмму линейки инструментов командного окна MatLab. В результате на экране возникает окно этого браузера (рис. 7.106). Нажав мышкой на изображении первой пиктограммы в окне браузера, вызовем появление еще одного пустого окна *untitled* для введения блок-схемы модели. В этом новом окне следует вызвать меню *File*, а в нем – раздел *New,* а в возникающем нисходящем меню – раздел *Library* (см. рис. 7.107). В результате этих действий на экране возникнет пустое окно новосоздаваемой библиотеки (рис. 7.108) с именем *Library untitled1*. Теперь в нем можно создавать, или перетягивать в него созданные S-блоки.

Напомним, что в общем случае образовать S-блок можно на основе двух видов стандартных блоков:

- блока ***S-Function*** из раздела *Function & Ta*ble библиотеки SimuLink;
- блока ***SubSystem*** из раздела *Signals & Systems* той же библиотеки.

Блок на основе ***S-Function*** получается на грунте написанных языком MatLab файлов S-функции, и имеет лишь один вход (возможно векторный) и один выход (тоже векторный). Блок на основе ***SubSystem*** состоит как схема из существующих блоков и может иметь произвольное количество входов и выходов произвольного вида.

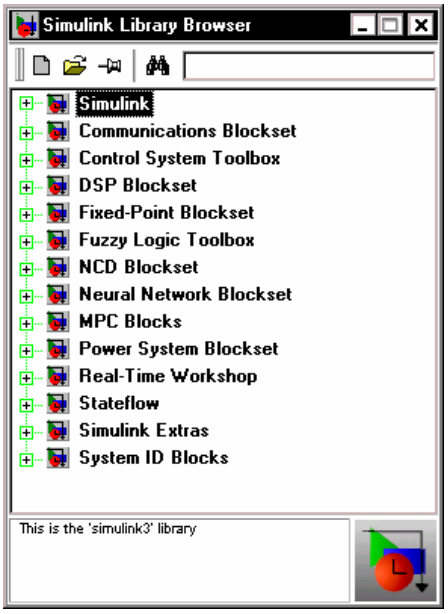

Рис. 7.106

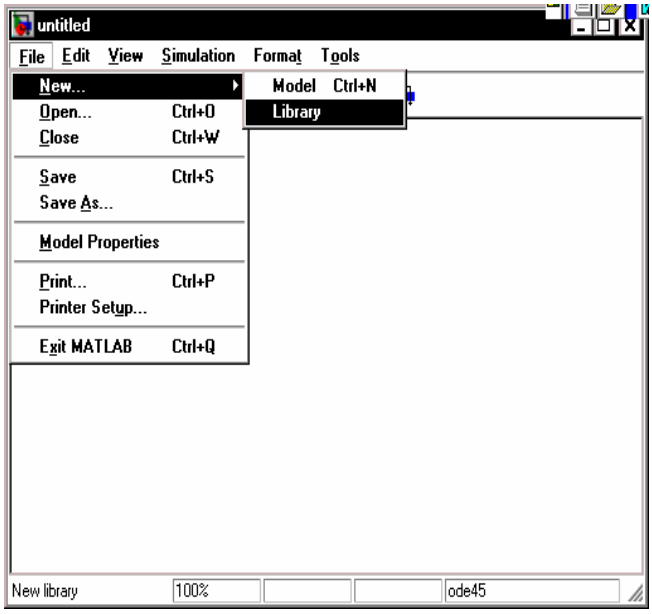

Рис. 7.107

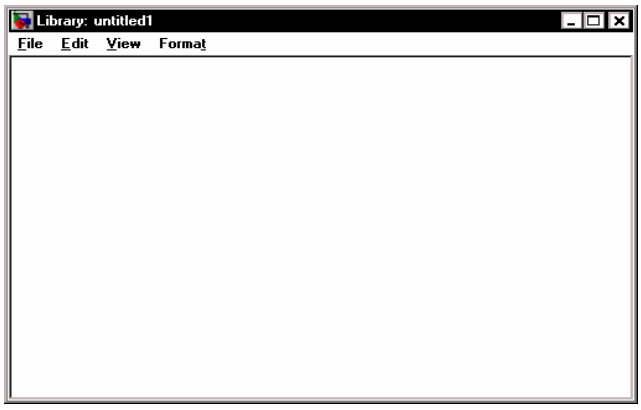

Рис. 7. 108

Образуем в этой библиотеке S-блок ***S_DUE***, на основе созданной прежде одноименной S-функции.

Прежде всего, перетянем в окно создаваемой библиотеки блок ***S-Function*** из указанного раздела библиотеки *Function & Ta*ble библиотеки SimuLink. Окно библиотеки приобретет вид, приведенный на рис. 7.109.

Теперь, дважды щелкнув мышкой на изображении этого блока, вызовем его окно настраивания (рис. 7.110). В окошко *S-Function name* введем имени S-функции (***S_DUE***), а в окошко *S-Function parameters* – ее параметры (*J, UgSk0*) (рис. 7.111) и нажмем мышкой на кнопку <OK> внизу этого окна. В результате (если соответствующий файл существует в путях, достижимых для MatLab, а список введенных параметров отвечает списку параметров, указанных в S-функции)

окно настраивания исчезнет, а изображение блока в окне библиотеки изменится (см. рис. 1.112).

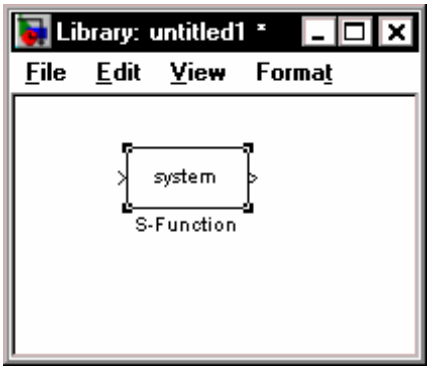

Рис. 7.109

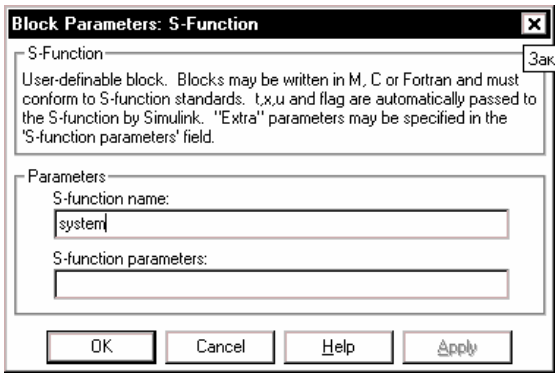

Рис.7.110

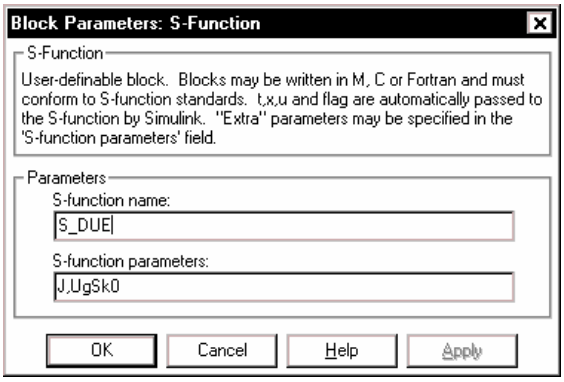

Рис. 1.111

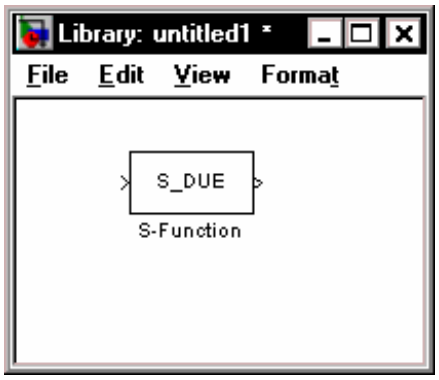

Рис. 1.112

В завершение изменим название под блоком на такое "Динамические уравнения Эйлера", чтобы точнее отобразить сущность преобразований, которые осуществляет блок.

В дальнейшем, для моделирования процесса управления ориентацией, например, космического аппарата (КА), обращающегося вокруг планеты по определенной замкнутой орбите, станет необходимым еще один блок, осуществляющий интегрирование кинематических уравнений ориентации, вычисляя значения параметров, определяющих текущее угловое положение корпуса КА относительно орбитальной системы отсчета. Например, если в качестве таких параметров избрать кватернион поворота $\mathbf{Q}$, который переводит текущее положение КА в нужное, соответствующие кинематические уравнения будут иметь вид

$$\frac{d\mathbf{Q}}{dt} = \frac{1}{2}(\mathbf{Q} \circ \boldsymbol{\omega} - \boldsymbol{\Omega} \circ \mathbf{Q}),$$

где обозначено: $\boldsymbol{\omega}$ - вектор-кватернион абсолютной угловой скорости КА; $\boldsymbol{\Omega}$ - вектор-кватернион абсолютной угловой скорости орбитальной системы отсчета (жестко связанной с положением КА на орбите); $\circ$ - знак кватернионного умножения.

Кватернионное кинематическое уравнение недостаточно удобно для проведения вычислений из-за того, что действия над кватернионами существенно отли-

чаются от действий над матрицами и не предусмотрены в системе MatLab. Значительно удобнее превратить его в совокупность сугубо матричных уравнений:

$$\begin{cases} \dfrac{dq_0}{dt} = -\dfrac{1}{2}\mathbf{q}^t \cdot (\boldsymbol{\omega} - \boldsymbol{\Omega}) \\ \dfrac{d\mathbf{q}}{dt} = \dfrac{1}{2}[q_0 \cdot (\boldsymbol{\omega} - \boldsymbol{\Omega}) + (\mathbf{q}\times) \cdot (\boldsymbol{\omega} + \boldsymbol{\Omega})] \end{cases}$$

В этих уравнениях величины $\mathbf{q}$, $\boldsymbol{\omega}$ и $\boldsymbol{\Omega}$ суть векторы-столбцы из проекций, соответственно, векторной части кватерниона поворота, вектора абсолютной угловой скорости КА на оси связанной системы координат и вектора угловой скорости орбитальной системы координат на ее же оси; q_0 скалярная часть кватерниону поворота; $(\mathbf{q}\times\ )$ – обозначения кососиметричной матрицы, составленной из проекций вектора $\mathbf{q}$.

Создадим М-файл S-функции, осуществляющей интегрирование этих кинематических уравнений. Ниже приведен текст этого М-файла по имени ***S_KUqwat***.

```
function [sys,x0,str,ts] = S_KUqwat(t,x,u,flag,OM0,Qw0)
 % S-функция S_KUqwat Кинематических Уравнений в кватернионах
 % Реализует переход от заданного вектора абсолютной
 % угловой скорости орбитального космического аппарата
 % к кватерниону поворота КА относительно орбитальной
 %                    системы отсчета
 % ВХОД блока:
 %       u = [omx,omy,omz]- вектор проекций абсолютной угловой
 %                скорости КА на оси СК, жёстко связанной с ним
 % ВЫХОД блока:
 %       y=[qw0,qw1,qw2,qw3] - вектор компонентов кватерніону поворота КА
 %                относительно орбитальной декартовоï системы координат
 % Входные ПАРАМЕТРЫ S-функции:
 %       OM0 - орбитальная угловая скорость;
 %       Ug0 - вектор начальных значений углов поворота

 % Лазарев Ю.ф. Украина 18-12-2001

switch flag,
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(Ug0);
  case 1,
    sys = mdlDerivatives(t,x,u,OM0);
 case 3,
    sys = mdlOutputs(x);
  case 9
    sys = [];
end
 % Конец процедуры
%
%=============================================
 % Далее идут тексты внутренних процедур
%=============================================
%
function [sys,x0,str,ts] = mdlInitializeSizes(Ug0)
sizes = simsizes;
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
```

```
sizes.NumOutputs     = 4;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = Qw0;
str = [];
ts  = [0 0];
 % Конец процедуры mdlInitializeSizes

%=============================================
function z = mdlDerivatives(t,x,u,OM0)
 % ВХОДНОЙ вектор "u" является вектором проекций моментов внешних сил,
 %    действующих на космический аппарат соответственно по осям X Y Z
 % x(1)=qw0 - скалярная часть кватерниона;
 % x(2:4)=qw(1:3) - векторная часть кватерниона;
 % z(1)=d(qw0)/dt; z(2:4)=d(qw)/dt;

 % Формирования векторов угловых скоростей и кватерниона
om=u;       OM = [0 OM0 0]';    v=x(2:4);
omMOM=om-OM;  omPOM=om+OM;
z(1)=-(v'*omMOM)/2;                    % уравнения скалярной части кватерниона
z4=(x(1)*omMOM+cross(v,omPOM))/2;      % уравнения векторной части кватерниона
z(2:4)=z4;
 % Конец процедуры mdlDerivatives

%=============================================%
function y = mdlOutputs(x)
y=x;
 % Конец процедуры mdlOutputs
```

Полностью аналогично предыдущему создадим новый S-блок за именем ***S_Kuqwat***. В нем входом является вектор проекций абсолютной угловой скорости КА, а выходом – вектор, состоящий из четырех компонентов кватерниона поворота КА относительно орбитальной системы координат. Первый компонент представляет собой скалярную часть, остальные три – проекции векторной части этого кватерниона. В результате получим окно библиотеки в виде, представленном на рис. 1.113.

Наконец, довольно важно создать S-блок, который осуществлял бы операцию векторного умножения двух векторов, аналогичную М-функции ***cross***.

Для этого удобнее использовать другой стандартный блок ***SubSystem*** из раздела *Signals & Systems*. Перетянем его мышью в окно новой библиотеки (рис. 1.114). Дважды щелкнув на изображении этого блока, получим пустое окно, в котором составим блок-схему подсистемы, приведенную на рис. 1.115.

В ней использован блок ***MATLAB Function*** из раздела *Functions & Tables*, окно настраивания которого представлено на рис. 7.116. Именно он, собственно, и выполняет операцию векторного умножения двух входных векторов, используя для этого стандартную функцию ***cross*** системы MatLab.

Рис. 1.113

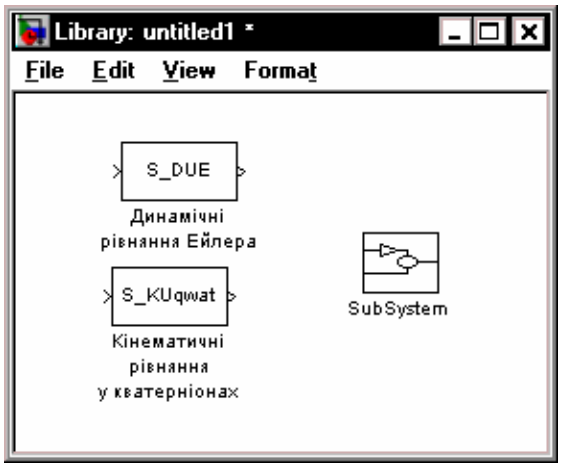

Рис. 1.114

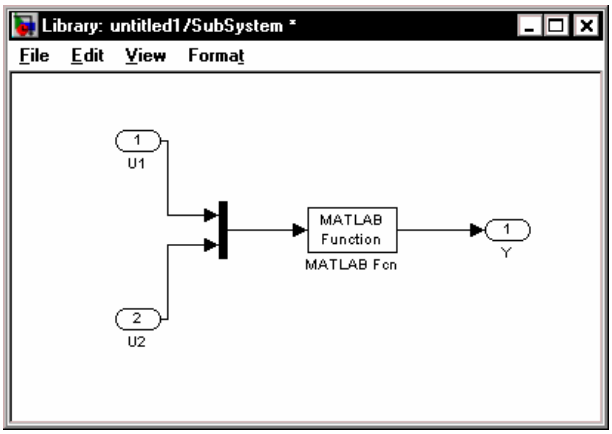

Рис. 7.115

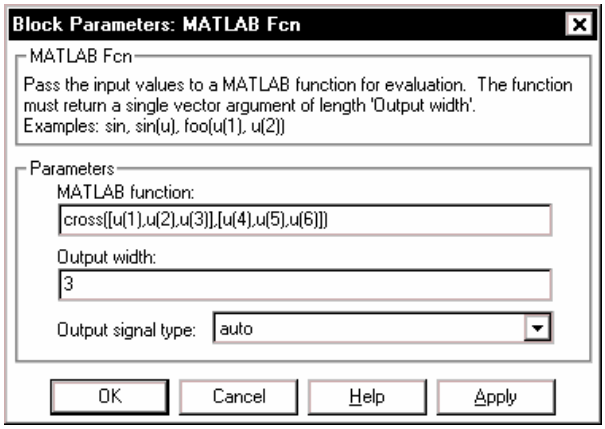

Рис. 7.116

В итоге всего этого будем иметь новую библиотеку из трех новых собственных S-блоков. Запишем ее пока что как ***untitled1*** (рис. 7.117).

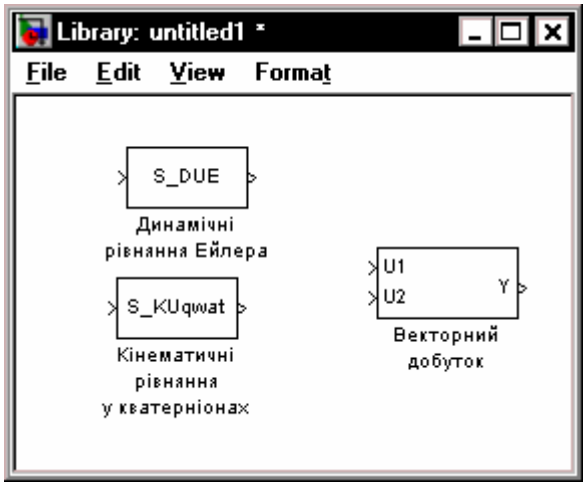

Рис. 7.117

7.5.2. Маскировка блоков

Теперь рассмотрим процесс маскировки блока, т. е. образования окна настраивания блока, которое является более удобным механизмом оперирования с блоком.

Прежде всего, нужно выделить тот блок в библиотеке, для которого желательно образовать маску. Пусть это будет блок ***S_DUE*** новой библиотеки. Выделим его, щелкнув мышкой на его изображении.

Теперь нужно перейти в меню *Edit* окна библиотеки, в которой расположен выделенный блок. При этом возникнет перечень команд этого меню (рис. 7.118).

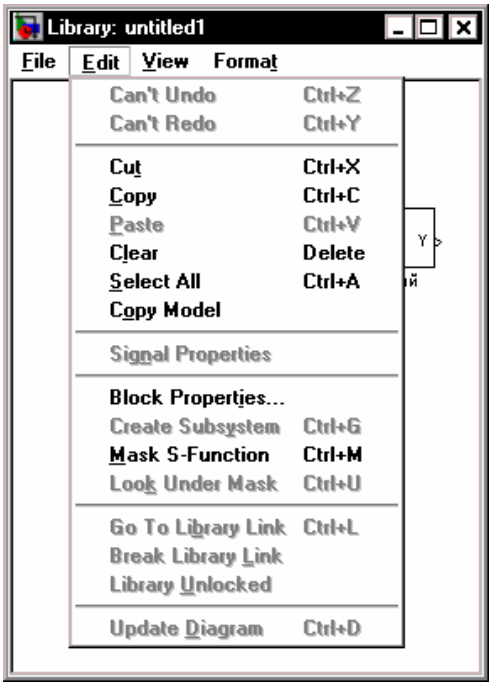

Рис. 7.118

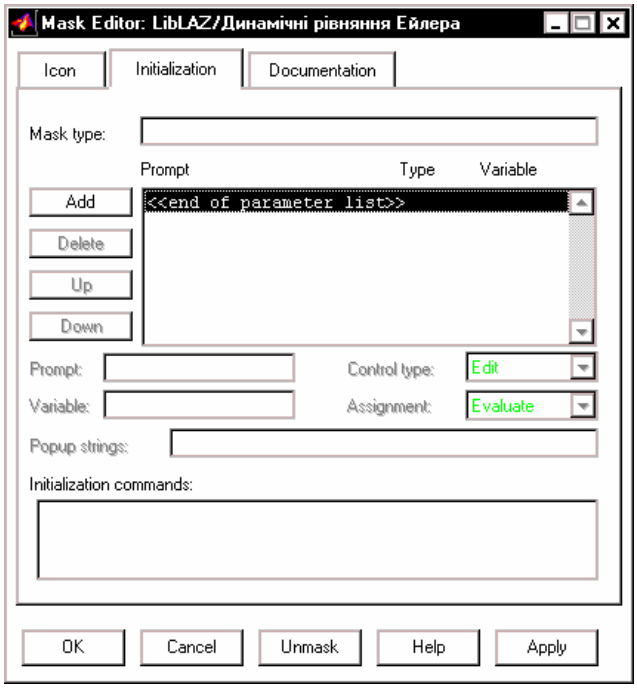

Рис. 7.119

В этом перечне следует выбрать команду *Mask S-Function*. Тогда на экране возникнет окно редактора маски, представленное на рис. 7.119.

Примечание. *При повторном вызове вашей библиотеки возможно, что эта команда не является активной. Тогда обратите внимание на предпоследнюю команду в перечне меню. Она должна быть активной и иметь такой вид* ***Unlock Library****. Нажмите на нее мышью. Вид перечня меню изменится, и команда* ***Mask S-Function*** *должна стать активной.*

Окно **Mask Editor** (рис. 7.119) имеет три вкладки:

- *Icon* – для создания и редактирования изображений на пиктограмме блока;
- *Initialization* – для создания и редактирования диалоговой части (введения параметров) окна настраивания;
- *Documentation* – для оформления и редактирования текстовой части окна настраивания блока и справочной части маски.

Перейдем (с помощью мыши) к важнейшей из них – *Initialization,* - так как именно она, собственно, образовывает маску, ее главную часть, - окно настраивания. Она имеет вид, приведенный на рис. 7.119.

Укажем, что из рассмотрения окон настраивания (масок) стандартных S-блоков вытекает, что эти окна имеют, в общем случае, три части:

- первая (верхняя) часть содержит справочную информацию о назначении блока, его главных параметрах и правилах, которыми следует пользоваться при введении параметров блока и его использовании;
- вторая (нижняя) часть по имени *Parameters* содержит окошка введения параметров блока и надписи над этими окошками, которые объясняют содержание этих параметров;
- третья, самая нижняя, стандартная часть содержит стандартные кнопки *OK*, *Cancel*, *Help* и *Apply*.

Редактор маски предназначен для оформления первых двух частей окна настраивания, а также создания справки, которая вызовется при нажатии кнопки *Help* в этом окне.

Рассматривая вкладку *Initialization*, можно сделать вывод, что с ее помощью можно сконструировать вторую, важнейшую, часть маски, - ее диалоговую часть.

Сконструируем диалоговую часть маски избранного блока. В нем всего два параметра, значения которых нужно вводить – матрица моментов инерции тела *J* размером (3*3) и вектор *UgSk0* начальных значений трех проекций угловой скорости тела. Поэтому нужно создать два окошка введения значений этих параметров и создать надписи на них.

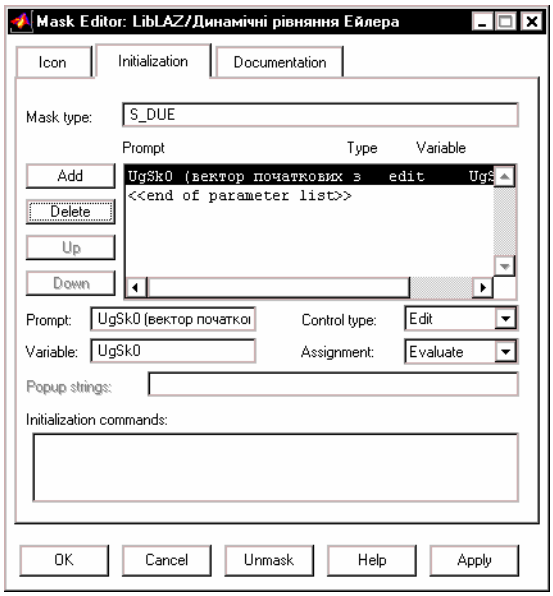

Рис. 7.120

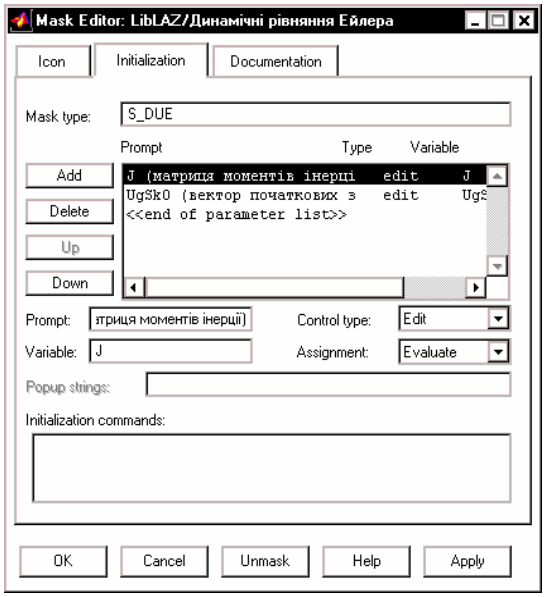

Рис. 7.121

Сначала в верхнем окошке введения *Mask type* вкладки введем имя блока ***S_DUE***.

Введение очередного окошка в маску осуществляется нажатием мышью кнопки *Add* по левую сторону от наибольшего окна вкладки, где отображаются результаты редактирования. При этом надпись *<<end of parameter list>>* в этом окне перемещается на одну строку вниз, освобождая верхнюю строку для отобра-

жения надписи над создаваемым окошком введения. Сама запись этой надписи осуществляется в окошке с надписью *Prompt*.

Запишем в это окошко такую строку

UgSk0 (вектор начальных значений угловой скорости).

Ниже в окошко с надписью *Variable* записывается имя идентификатора, содержащего этот параметр в правой части обращения к соответствующей S-функции (***UgSk0***).

Результат этих действий показан на рис. 1.120.

Снова нажмем кнопку *Add* и введем в окошко с надписью *Prompt* строку *J (матрица моментов инерции)*, а в окошко *Variable* введем имя ***J***. В результате окно редактора маски приобретет вид, представленный на рис. 1.121.

Теперь можно перейти к вкладке *Documentation* (рис. 7.122). В окно *Mask type* следует ввести имя блока. В окно *Block description* записывается информация, которую бы вы желали иметь в верхней (справочной) части окна настраивания блока, а в окне *Block Help* записывается дополнительная справочная информация, которая вызовется, если в окне маски нажать кнопку *Help*.

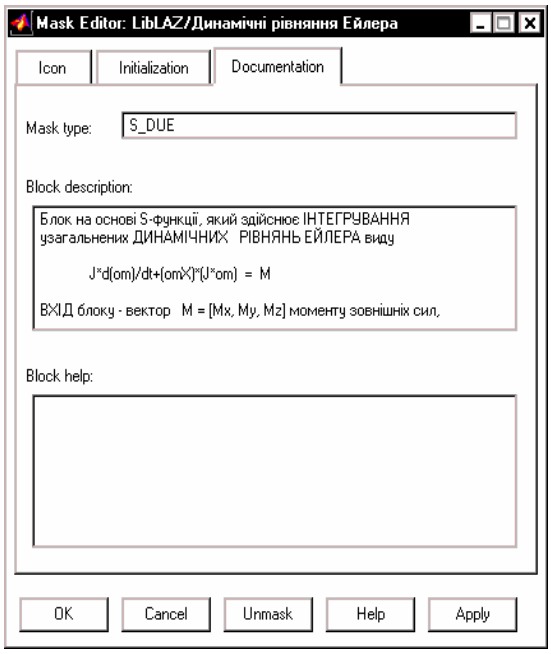

Рис.7.122

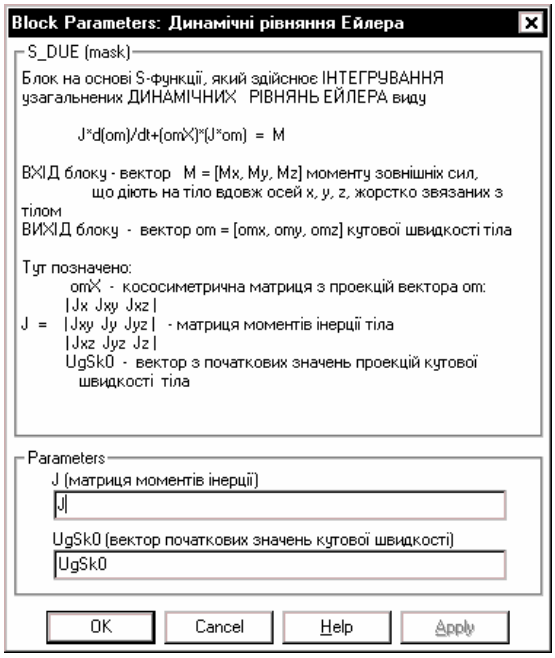

Рис. 7.123

Если теперь завершить редактирование маски нажатием кнопки <OK> в окне редактора маски, перейти в окно библиотеки и дважды щелкнуть на изображении блока ***S_DUE***, то возникнет не окно черт. 7.111 (которое возникало, когда блок не был маскирован), а окно маски блока, изображенное на рис. 7.123.

Аналогично получается маска блока ***S_KUqwat***, представленная на рис. 7.124, и маска блока "Векторное произведение", приведенная на рис. 7.126. Единственным исключением является то, что при создании последней маски была использована и вкладка *Icon* (рис. 7.125), в окно *Drawing commands* которой была введена команда

```
disp('cross(U1,U2)')
```

которая, собственно, выводит надпись ***cross(U1,U2)*** на изображении блока.

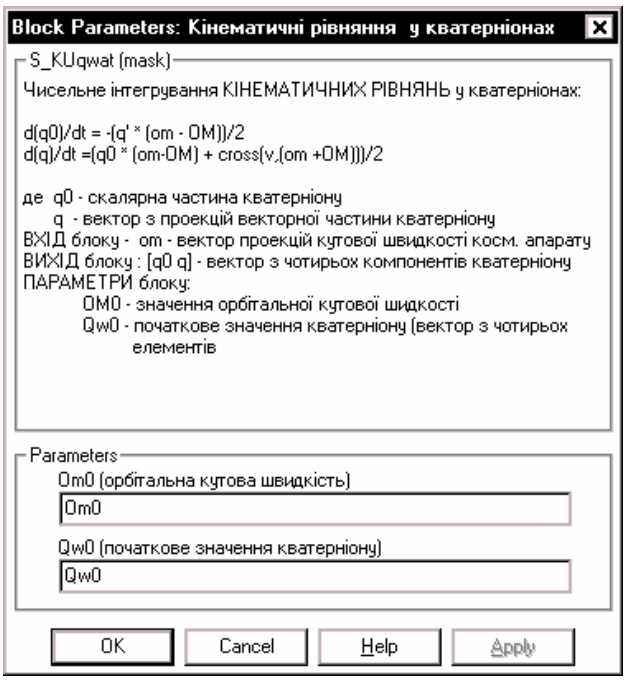

Рис.7.124

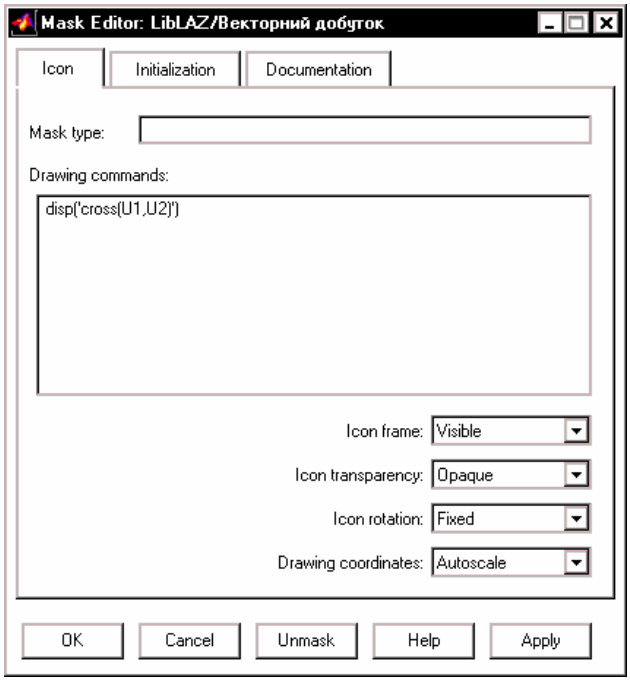

Рис. 7.125

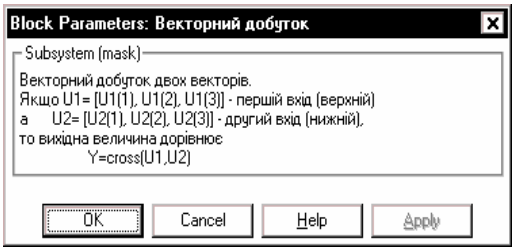

Рис. 7.126

Образованную библиотеку запишем под именем ***LibLAZ. mdl.***

7.5.3. Моделирование процесса ориентации космического аппарата

В качестве примера применения блоков собственной библиотеки рассмотрим процесс образования S-модели и комплекса М-программ, предназначенных для моделирования процесса управления ориентацией космического аппарата (в частности, искусственного спутника Земли – ШСЗ).

Для простоты будем рассматривать космический аппарат как одно твердое тело, которое обращается вокруг Земли по замкнутой орбите. Управление ориен-

тацией (т. е. угловым положением относительно орбитальной системы координат) осуществляется с помощью трех маховичних двигателей, оси которых совпадают с осями декартової системы координат, жестко связанной с корпусом космического аппарата (КА). Маховичные двигатели, с одной стороны, осуществляют разгон соответствующего ротора в соответствии с уравнениями

$$\frac{dH_k}{dt} = M_k, \qquad (k = x, y, z);$$

а, с другой стороны, осуществляют наложение соответствующего момента сил (но в противоположном направлении) вокруг оси вращения ротора на корпус самого космического аппарата. Последний момент вызывает изменение углового движения КА вокруг этой оси, т. е. осуществляет управление ориентацией.

Изменение угловой ориентации космического аппарата в пространстве подчиняется основным законам механики, из которых вытекают уравнения, воплощенные в блоках ***S_DUE*** и ***S_KUqwat***.

Составим S-модель системы ориентации и сохраним ее в файле ***SKOKA.mdl***. Она приведена на рис. 7.127.

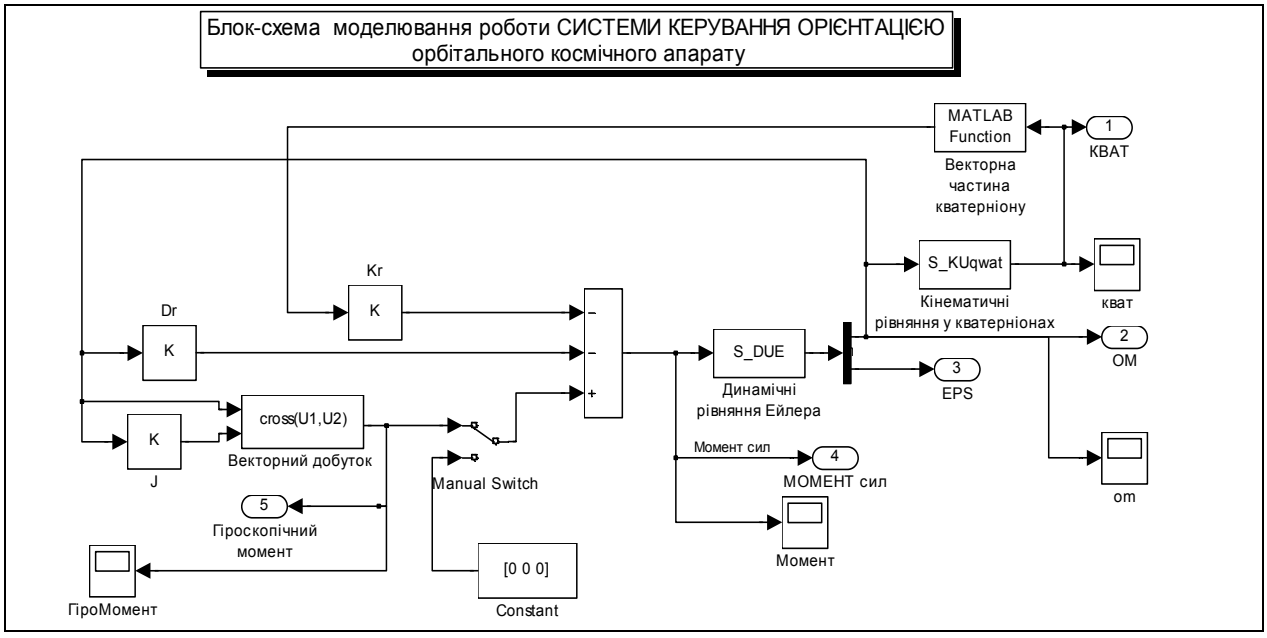

Рис. 7.127

Система ориентации состоит из последовательно соединенных блоков ***S_DUE*** и ***S_KUqwat***. Полученное на выходе блока ***S_KUqwat*** значение кватерниона поворота подается на вход блока *MATLAB Function,* который выделяет векторную часть этого кватерниона, необходимую для формирования вектора момента управления угловым положением КА.

В целом момент сил управления формируется по такому закону:

$$\mathbf{M} = -K_r \cdot \mathbf{q} - D_r \cdot \boldsymbol{\omega} + (\boldsymbol{\omega}\times) \cdot (J \cdot \boldsymbol{\omega}). \qquad (7.8)$$

В нем можно различить три составляющие:

- составляющую, пропорциональную векторной части кватерниону отклонения текущего положения КА от заданного его положения (в этой программе задан-

ное положение отвечает нулевому значению векторной части кватерниона); именно эта составляющая момента управления заставляет приближаться КА к заданному угловому положению;

- составляющую, пропорциональную вектору угловой скорости КА; она образует демпфирование процесса приближения КА к заданному положению;
- третья, последняя составляющая вводится для того, чтобы компенсировать возникающий при угловом движении КА гироскопический момент, который стремится повернуть КА вокруг оси, перпендикулярной оси действия момента сил управления; именно введение этой составляющей заставляет корпус КА возвращаться к заданному положению по кратчайшему пути, т. е., уменьшает энергетические затраты на переориентацию КА.

Матрицы K_r и D_r определяют закон управления и должны быть предварительно известными.

Формирование первой составляющей момента управления на блок-схеме осуществляется верхней обратной цепью с матричным усилителем Kr. Вторая составляющая формируется цепью с матричным усилителем Dr. Третья составляющая получается третьей обратной цепью, в состав которой входят матричный усилитель J и образованный ранее блок ***cross(U1,U2)*** (см. рис. 7.127).

Эти три образованные составляющие суммируются на сумматоре и подаются на вход блока ***S_DUE***. Этим получается замкнутая система управления ориентацией.

Поставим задачу промоделировать поведение системы ориентации космического аппарата, управляемого по компонентам кватерниона при разных законах регулирования в соответствии с данными, приведенными в статье [6], т. е. при значении матрицы инерции КА

```
J=[1200 100 -200;100 2200 300;-100 300 3100]
J =
      1200       100      -200
       100      2200       300
      -100       300      3100
```

матрицы моментов демпфирования

```
Dr=0.315*diag([1200 2200 3100])
Dr =
  378.0000         0         0
         0  693.0000         0
         0         0  976.5000
```

и четырех значениях матрицы позиционного управления:

- случай 1 - матрица управления пропорциональна обратной матрице моментов инерции

```
Kr=k/J=diag([201,110,78]);
Kr =
   201     0     0
     0   110     0
     0     0    78
```

- случай 2 - матрица управления пропорциональна единичной матрице E

```
Kr=k*E=diag([110,110,110]) ;
Kr =
   110     0     0
     0   110     0
```

```
     0    0  110
```

- случай 3 - матрица управления представляет собой комбинацию

Kr=(αJ+βE) =diag([72,110,204]);

```
Kr =
     72    0    0
      0  110    0
      0    0  204
```

- случай 4 - матрица управления пропорциональна матрице J моментов инерции

Kr=k*J=diag([60,110,155])

```
 Kr =
     60    0    0
      0  110    0
      0    0  155
```

Будем предполагать, что орбитальная угловая скорость равняется нулю, а начальное отклонение положения КА от заданного определяется кватернионом

Qw0=[0.159 0.57 0.57 0.57],

что отвечает начальному отклонению от требуемого положения, равному 161,7 градусов.

Чтобы начать моделирование, нужно присвоить исходные значения всем параметрам. После моделирования необходимо на основе полученных при моделировании данных построить ряд графиков, которые отображали бы процесс переориентации КА. Сделаем это (а также собственно моделирование) при помощи специального М-файла ***SKOKAupr.m***. Текст этого файла приведен ниже

```
% SKOKAupr.m
% Управляющая программа  для запуска модели SKOKA.mdl

% Лазарев Ю.Ф.  18-12-2001
% Последние изменения 24-12-2001
clear all
clc
K=[3,1,2];
OM0=0;%.01;
% Введение значений матрицы инерции
J=[1200 100 -200; 100 2200 300; -200 300 3100];
% Введение начальных значений:
% 1) проекций угловой скорости тела
UgSk0=[0 0 0];
% 2)_компонентов кватерниона поворота
Ug0=[0.159,0.57,0.57,0.57];
qw0=Ug0(1);    qw=Ug0(2:4);
U0=2*acos(qw0); Cs0=qw/sin(U0/2);
Zk=[1 0 0 0];
% Матрицы моментов УПРАВЛЕНИЯ
Dr=0.315*diag(diag(J))
V=[201,110,78;110 110 110;72 110 204;60 110 155]
for k=1:4
   % Установление параметров моделирования
  Kr=diag(V(k,:))
  options=simset('Solver','ode45','RelTol',1e-6);
  sim('SKOKA2',60,options);                    % МОДЕЛИРОВАНИЕ на S-модели
   % Формирование данных для вывода ГРАФИКОВ
  tt=tout;
  q0=yout(:,1); qx=yout(:,2);   qy=yout(:,3);          qz=yout(:,4);
  omx=yout(:,5);    omy=yout(:,6); omz=yout(:,7);
  Mx=yout(:,11);    My=yout(:,12); Mz=yout(:,13);
  if k==1
    t1=tt;
```

```
    q01=q0;     qx1=qx;     qy1=qy;     qz1=qz;
    omx1=omx;     omy1=omy;     omz1=omz;
    Mx1=Mx;    My1=My;    Mz1=Mz;
  elseif k==2
    t2=tt;
    q02=q0;     qx2=qx;     qy2=qy;     qz2=qz;
    omx2=omx;     omy2=omy;     omz2=omz;
    Mx2=Mx;    My2=My;    Mz2=Mz;
  elseif k==3
    t3=tt;
    q03=q0;          qx3=qx;          qy3=qy;          qz3=qz;
    omx3=omx;     omy3=omy;     omz3=omz;
    Mx3=Mx;          My3=My;          Mz3=Mz;
  elseif k==4
    t4=tt;
    q04=q0;          qx4=qx;          qy4=qy;          qz4=qz;
    omx4=omx;     omy4=omy;     omz4=omz;
    Mx4=Mx;          My4=My;          Mz4=Mz;
  end
  clear tt q0 qx qy qz omx omy omz Mx My Mz
end
A=180/pi;
D1=2*acos(q01);    D2=2*acos(q02);    D3=2*acos(q03);    D4=2*acos(q04);
dt1=diff(t1);
dHx1=diff(Mx1);          DHx1=cumsum([0;dHx1.*dt1]);
dHy1=diff(My1);          DHy1=cumsum([0;dHy1.*dt1]) ;
dHz1=diff(Mz1);          DHz1=cumsum([0;dHz1.*dt1]);
dH1=DHx1+DHy1+DHz1;
dt2=diff(t2);
dHx2=diff(Mx2);          DHx2=cumsum([0;dHx2.*dt2]);
dHy2=diff(My2);          DHy2=cumsum([0;dHy2.*dt2]) ;
dHz2=diff(Mz2);          DHz2=cumsum([0;dHz2.*dt2]);
dH2=DHx2+DHy2+DHz2;
dt3=diff(t3);
dHx3=diff(Mx3);          DHx3=cumsum([0;dHx3.*dt3]);
dHy3=diff(My3);          DHy3=cumsum([0;dHy3.*dt3]) ;
dHz3=diff(Mz3);          DHz3=cumsum([0;dHz3.*dt3]);
dH3=DHx3+DHy3+DHz3;
dt4=diff(t4);
dHx4=diff(Mx4);          DHx4=cumsum([0;dHx4.*dt4]);
dHy4=diff(My4);          DHy4=cumsum([0;dHy4.*dt4]) ;
dHz4=diff(Mz4);          DHz4=cumsum([0;dHz4.*dt4]);
d4=DHx4+DHy4+DHz4;
 % Графики проекций компонентов кватерниона на плоскости
subplot(2,2,1)
plot(qx1,qy1, qx2,qy2,':',qx3,qy3,'--',qx4,qy4,'.'), grid
title('          Проекции КОМПОНЕНТОВ кватерниона')
xlabel('Qx')
ylabel('Qy')
subplot(2,2,2)
plot(qy1,qz1,qy2,qz2,':',qy3,qz3,'--',qy4,qz4,'.'), grid
title('  на координатные плоскости              ')
xlabel('Qy')
ylabel('Qz')
subplot(2,2,3)
plot(qx1,qz1, qx2,qz2,':',qx3,qz3,'--',qx4,qz4,'.'), grid
xlabel('Qx')
ylabel('Qz')
legend('Kr = k/J','Kr = kE','Kr = k/(\alphaJ+\betaE)','Kr = kJ',4)
subplot(2,2,4)
c1=D1./sin(D1/2)*A;
cx1=c1.*qx1;    cy1=c1.*qy1; cz1=c1.*qz1;
```

```
c2=D2./sin(D2/2)*A;
cx2=c2.*qx2;    cy2=c2.*qy2; cz2=c2.*qz2;
c3=D3./sin(D3/2)*A;
cx3=c3.*qx3;    cy3=c3.*qy3; cz3=c3.*qz3;
c4=D4./sin(D4/2)*A;
cx4=c4.*qx4;    cy4=c4.*qy4; cz4=c4.*qz4;
plot3(cx1,cy1,cz1,cx2,cy2,cz2,':',cx3,cy3,cz3,'--',cx4,cy4,cz4,'.'),grid
title('Вектор ЕЙЛЕРОВОГО поворота в пространстве')
xlabel('Ex (градусы)')
ylabel('Ey (градусы)')
zlabel('Ez (градусы)')
 % Графики зависимостей компонентов кватерниона от времени
figure
subplot(2,2,1)
plot(t1,qx1,t2, qx2,':',t3,qx3,'--',t4,qx4,'.'), grid
title('        Зависимость КОМПОНЕНТОВ кватерниона  от времени')
xlabel('Время (c)')
ylabel('Qx')
subplot(2,2,2)
plot(t1,qy1,t2,qy2,':',t3,qy3,'--',t4,qy4,'.'), grid
ylabel('Qy')
xlabel('Время (c)')
subplot(2,2,3)
plot(t1,qz1,t2,qz2,':',t3,qz3,'--',t4,qz4,'.'), grid
ylabel('Qz')
xlabel('Время (c)')
subplot(2,2,4)
plot(t1,D1*A,t2,D2*A,':',t3,D3*A,'--',t4,D4*A,'.'), grid
title('Поворот вокруг оси Ейлера')
ylabel('Угол поворота в градусах')
xlabel('Время (c)')
legend('Kr = k/J','Kr = k','Kr = k/(\alpha+\beta)','Kr = k')
 % Графики зависимостей проекций момента сил от времени
figure
subplot(2,2,1)
plot(t1,Mx1,t2, Mx2,':',t3,Mx3,'--',t4,Mx4,'.'), grid
title('Зависимость проекций МОМЕНТА УПРАВЛЕНИЯ от времени')
ylabel('Mx')
xlabel('Время (c)')
subplot(2,2,2)
plot(t1,My1,t2,My2,':',t3,My3,'--',t4,My4,'.'), grid
ylabel('My')
xlabel('Время (c)')
subplot(2,2,3)
plot(t1,Mz1,t2,Mz2,':',t3,Mz3,'--',t4,Mz4,'.'), grid
ylabel('Mz')
xlabel('Время (c)')
subplot(2,2,4)
plot(t1,d1,t2,d2,':',t3,d3,'--',t4,d4,'.'), grid
title('Сумма приростов кинетических моментов ДВИГАТЕЛЕЙ-МАХОВИКОВ')
ylabel('\Delta ')
xlabel('Время (c)')
legend('Kr = k/J','Kr = k','Kr = k/(\alpha+\beta)','Kr = k',0)
```

Запуск этого М-файла приводит к результатам, представленным на рис. 7.128…7.130.

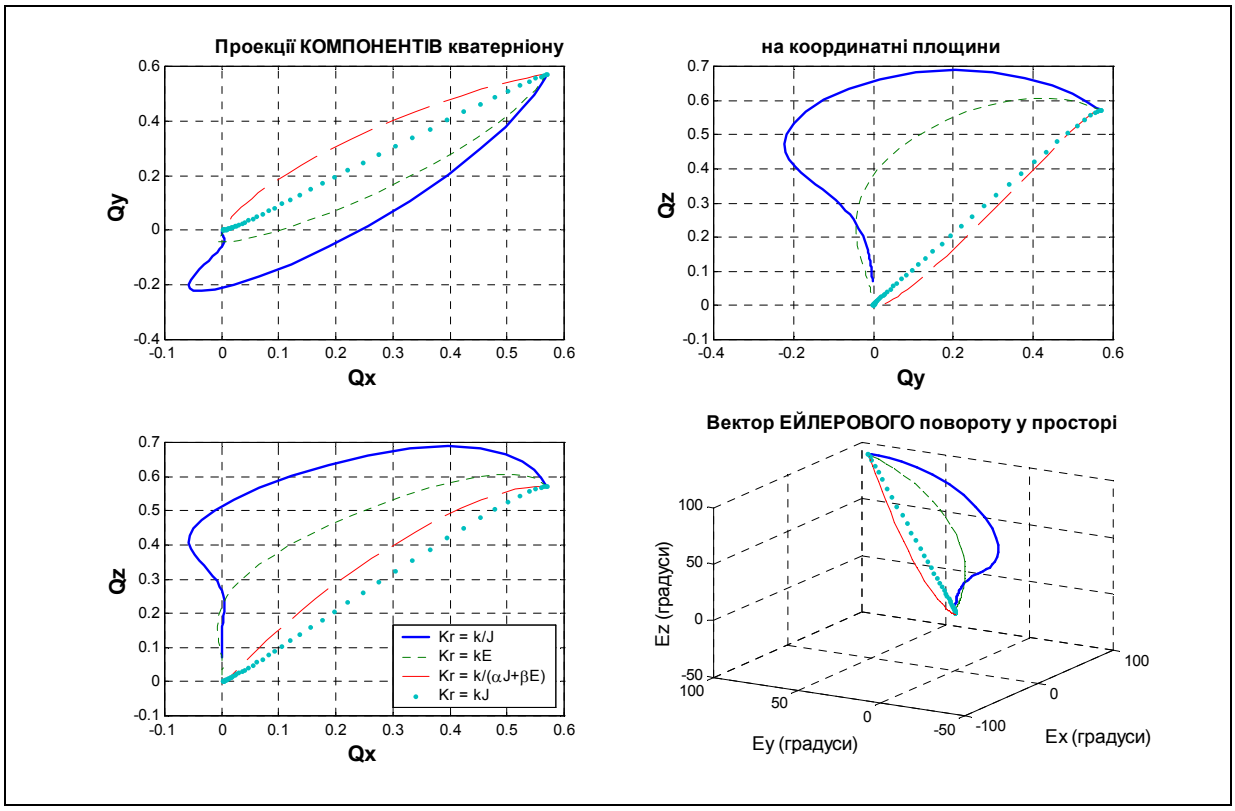

Рис. 7.128

На рис. 7.128 приведены проекции траекторий кватерниона на все три координатные плоскости, а также траектории в пространстве вектора Ейлерового поворота, на рис. 7.129, - графики зависимостей от времени компонентов кватернионов, а также проекций вектору Ейлерова поворота. Рис 7.130 представляет зависимость проекций момента управления от времени, а также общие (сумма по трем ортогональным осям) приращения кинетических моментов двигателей-маховиков, которые обеспечивают выполнение этих поворотов КА. Последние характеризуют в определенной степени затраты энергии на поворот КА.

Рассматривая полученные графики, можно сделать вывод, что наиболее оптимальным является управление по закону, когда матрица позиционного управления пропорциональна матрице моментов инерции КА.

Такого же вывода можно дойти и сугубо теоретическим путем, если подставить выражение (7.5) момента управления у уравнение (7.1) движения КА с учетом последней зависимости матрицы *Kr* от матрицы *J*. Если предположить также, что и матрица демпфирования *Dr* является также пропорциональной матрице *J* с коэффициентом пропорциональности *f*, то нетрудно убедиться, что векторное уравнение движения в таком случае будет иметь вид:

$$\frac{d\boldsymbol{\omega}}{dt} + f \cdot \boldsymbol{\omega} + k \cdot \mathbf{q} = 0\,,$$

и оно распадается на три одинаковых независимых уравнения поведения КА относительно трех его координатных осей.

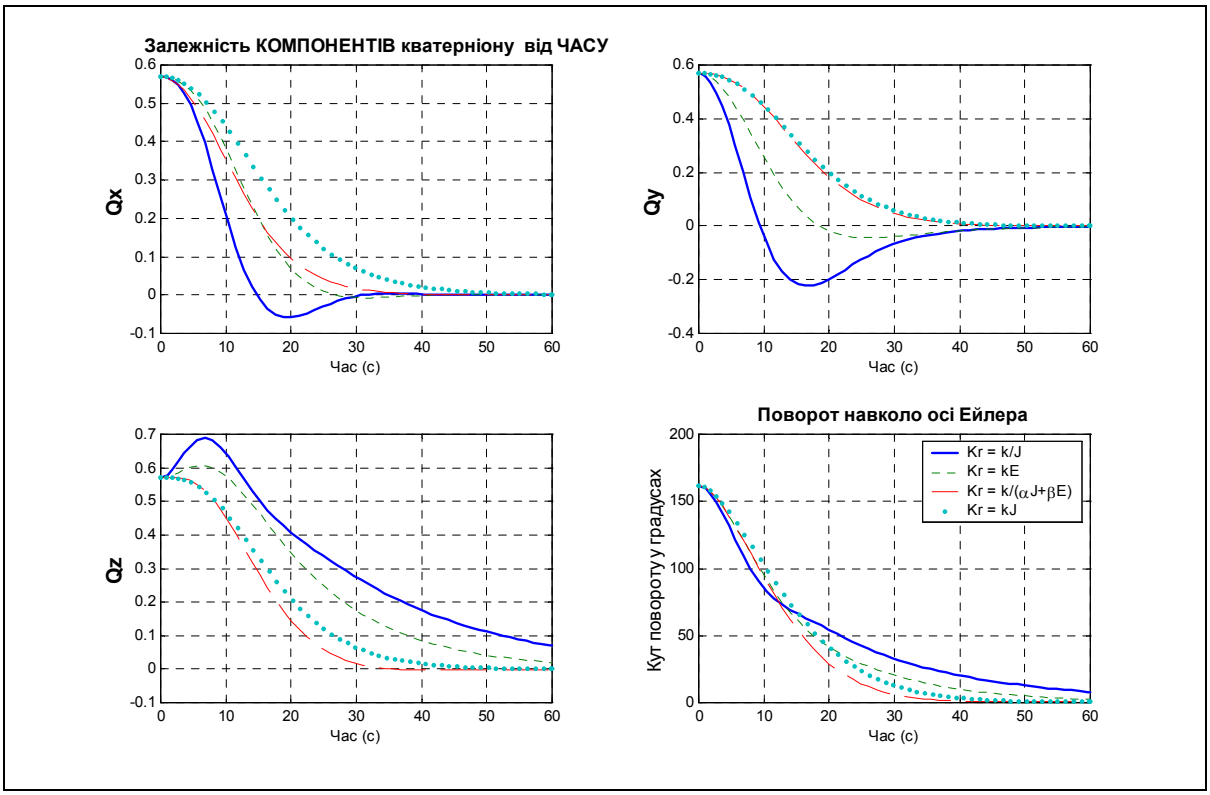

Рис. 7.129

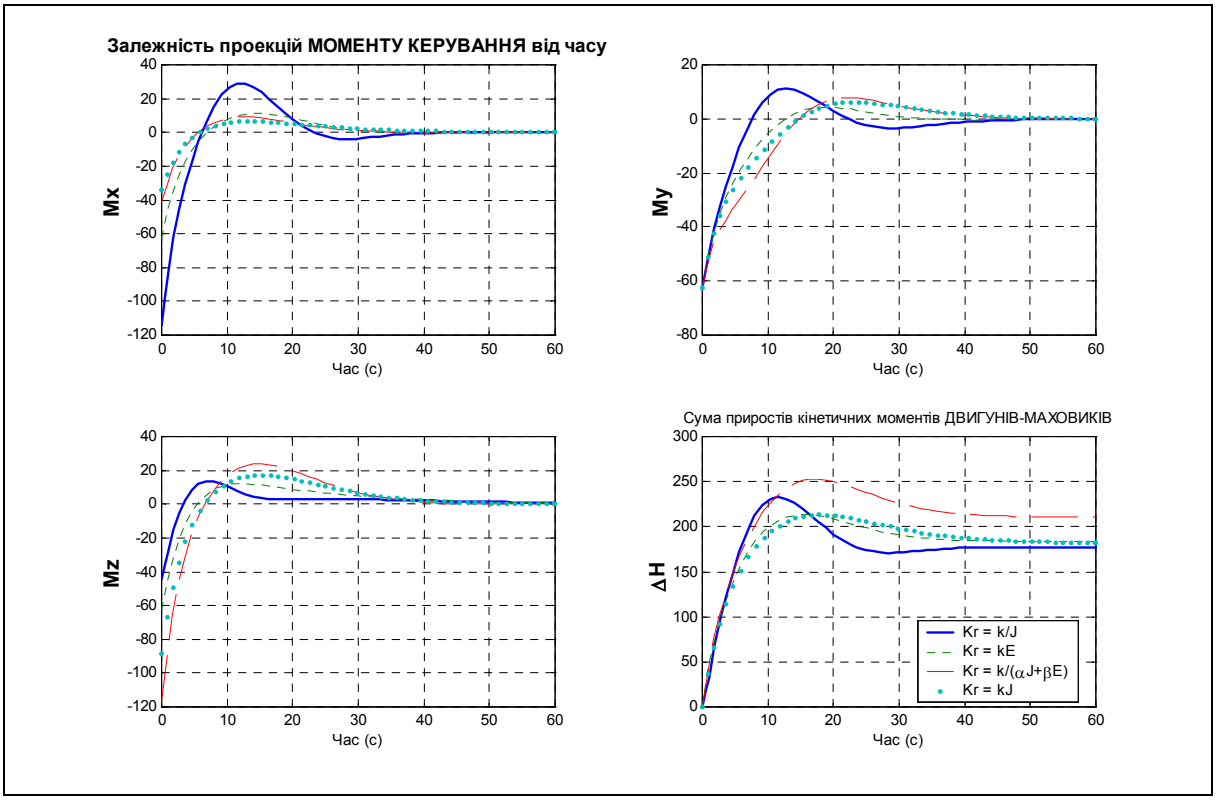

Рис. 7.130

Послесловие

Сведения, изложенные в этой книге, являются лишь начальными, необходимыми для усвоения MatLAB как мощного вычислительного средства проектирования и исследования технических устройств. В большинстве случаев их недостаточно для решения специфических задач проектирования.

Более подробные сведения о функциях, процедурах системы MatLAB читатель найдет в специальной справочной литературе, например, приведенной в списке литературы [3, 4, 8, 9, 10, 11, 12, 13]. Там же можно ознакомиться с содержанием некоторых пакетов прикладных программ (ППП), которые поставляются с той или иной версией MatLAB. К таким пакетам, кроме описанных в пособии пакетов ***SIMULINK, CONTROL*** и ***SIGNAL***, относятся в частности:

а) расширенный ППП из *математики*, которая содержит разделы *символьной математики, статистики, сплайн-апроксимации, оптимизации, уравнений в частных производных* и расширенную *NAG-библиотеку математических функций* (The Numerical Algorithms Group Ltd);

б) ППП по *анализу и синтезу систем управления*, который содержит, кроме рассмотренного пакета CONTROL, пакеты по *проектированию нелинейных систем управления, робастному управлению, управленю с эталонной моделью*, по μ-*анализу и синтезу, проектированию робастних систем с обратной связью* и по *синтезу систем управления на основе линейных матричных неравенств*;

в) ППП по *идентификации систем управления,* который включает *идентификацию параметров, идентификацию в частотной области* и *идентификацию в пространстве состояния*;

г) ППП по *обработке сигналов и изображений*, в который, кроме пакета SIGNAL, входят пакеты *многомерного спектрального анализа, обработки изображений* и *импульсной декомпозиции*.

Кроме этого, в MatLAB обычно входят пакеты по *системам связи и коммуникаций* и по финансам.

Основным же средством углубленного освоения средств и возможностей MatLAB является, во-первых, самостоятельное изучение описаний процедур, которые возникают (на английском языке) в командном окне MatLAB, если использовать команду ***help*** с указанием имени процедуры; во-вторых, изучение текстов самих программ (команда ***type*** <имя процедуры>) и, в-третьих, конечно, самостоятельное составление (написание и воплощение) программ в среде MatLAB с использованием разнообразных ее возможностей.

Список литературы

1. Антонью А. Цифровые фильтры: анализ и проектирование. - Г.: Радио и связь, 1983. - 320 с.
2. Барановская Г. Г., Любченко И. Н. Микрокалькуляторы в курсе высшей математики: Практикум. - К.: Высшая шк., 1987.- 288 с.
3. Герман-Галкин С. Г. Компьютерное моделирование полупроводниковых систем в MATLAB 6.0. Учебное пособие. – Спб.: "Корона принт", 2001. – 320 с.
4. Гультяев А. К. MatLAB 5.2. Имитационное моделирование в среде Windows: Практическое пособие. - Спб.: КОРОНА принт, 1999. - 288 с.
5. Гультяев А. К. Визуальное моделирование в среде MATLAB: Учебный курс. - Спб. : ПИТЕР, 2000. - 430 с.
6. Ви В., Уэйс Х., Эрепостатис Э. Управление поворотами космического аппарата вокруг собственной оси с обратной связью по компонентам кватерниона/ Аэрокосмическая техника, №3, март, 1990.
7. Дьяконов В. П. Справочник по применению системы PC MatLAB. - М.: Физматлит, 1993. - 113с.
8. Дьяконов В. Simulink 4. Специальный справочник. - Спб: Питер, 2002. – 518 с.
9. Дьяконов В., Круглов В. Математические пакеты расширения MatLAB. Специальный справочник. - СПб.: Питер, 2001. - 475с.
10. 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.
11. Лазарев Ю. Ф. Початки програмування в среде MatLAB: Уч. пособие. - К.: "Корнійчук", 1999. - 160с.
12. Лазарев Ю. MatLAB 5.x. – К.: "Ирина" (BHV), 2000. – 384 с.
13. Мартынов Н. Н. Введение в MATLAB 5. – Г.: Кудиц-образ, 2002. – 348 с.
14. Медведев В. С., Потемкин В. Г. Control System Toolbox. MatLAB 5 для студентов. - Г.: "ДИАЛОГ-МИФИ", 1999. – 287 с.
15. Потемкин В. Г. Система MatLAB: Справь. пособие. - М.: "ДИАЛОГ-МИФИ", 1997. - 350 с.
16. Потемкин В.Г. MatLAB 5 для студентов: Справ. пособие. - М.: "ДИАЛОГ-МИФИ", 1998. - 314 с.
17. Потемкин В. Г., Рудаков П. И. MatLAB 5 для студентов. - 2-е изд., испр. и дополн. - М.: "ДИАЛОГ-МИФИ", 1999. - 448 с.
18. Сулима И. М., Гавриленко С. И., Радчик И. А., Юдицкий Я. А. Основные численные методы и их реализация на микрокалькуляторах. - К.: Высшая шк., 1987. - 312 с.

Предметный указатель

Автокорреляционная функция 241
Алгебраические уравнения
- Ляпунова 311-312
- Риккати 311-312
Амплитуда комплексная 226
Амплитудно-частотная характеристика 78
Анализ линейной стационарной системы 294-300
Аппроксимация
- полиномиальная 60
- фильтра 246
- - аналоговая 247
Арифметические действия 11

Базис матрицы ортонормированный 52
Базис нуль-пространства матрицы 52
Боде диаграмма 303

Ввод
- данных из клавиатуры 9, 104
- действительных чисел 9
- векторов 26, 27
- комплексных чисел 13, 14
- линейных стационарных систем 183, 186-194, 287-292
- матриц 27
Взаимная корреляционная функция 237, 249
Взаимная спектральная плотность 237, 278
Взаимный корреляционный момент 51
Возведение в степень
- действительных чисел 11
- комплексных чисел 17
- матриц 38, 39
Вывод информации 107
- о линейной стационарной системе 304-305
Вычитание
- действительных чисел 11
- комплексных чисел 17
- векторов 34
- матриц 38
Вычисление
- произведения элементов вектора 48
- значения матричного полинома 59
- значения полинома 45
- интеграла 90
- максимального элемента вектора 48, 348
- минимального элемента вектора 48, 348
- минимума функции 92
- нулей передаточной функции 58
- полюсов передаточной функции 58
- производной от полинома 46
- среднего значения элементов вектора 48
- среднеквадратичного отклонения элементов вектора от их среднего значения 48
- суммы элементов вектора 48
Входной порт 353
Выходной порт 353

Генерирование случайных величин
- нормально распределенных 28, 77
- равномерно распределенных 28
Гистограмма 75
Годограф Найквиста 308
График вектора 75
Графическое окно MatLAB 72

Диагональ матрицы 30
Дискрет
- частоты 66, 238

- времени 66, 225, 238, 334
Дискретная передаточная функция 230
Дисперсия 51
Деление слева направо
- действительных чисел 11
- комплексных чисел 17
- матриц 40
Деление полиномов 44
Деление справа налево
- действительных чисел 11
- комплексных чисел 17
- матриц 40

Запас по амплитуде 308
Запас по фазе 308
Задержка сигнала 355, 348

Интегралы еллиптические 16
Интегратор
- непрерывный 349-350
- дискретный 348-353
Интегрирование
- дифференциальных уравнений 91, 117
- методом трапеций 49
- методом квадратур 90
Интерполяция 62, 63
- сплайнами 61

Карта Николса 308
Карта нулей-полюсов системы 310
Квантователь 353
Класс
- записей (структур) 174, 178-180, 184
- ячеек 174, 180-183
- вычислительных объектов 174
- разреженных двумерных числовых матриц 174
- символов 174, 175-178
- числовых матриц 174
- INLINE 184-187
- LTI 184, 187-202
- SYM 184
Комплексное сопряжение 17
Корень квадратный 15
- из комплексного числа 17
- из матрицы 42
Командное окно 8, 140
Конкатенация
- горизонтальная 33
- вертикальная 33

Линейная стационарная система 187, 297
Линейно-квадратичный оптимальный регулятор 308-310, 311
Логарифм
- от матрицы 41
- двоичный 16
- десятичный 15
- натуральный 15
Люфт 351

М-книга 169-173
М-файл 93, 94
Мантисса числа 9
Матрица
- Адамара 28
- Ганкеля 30
- Гильберта 29
- единичная 28
- из единиц 28
- управляемости системы 310
- ковариаций 51
- коэффициентов корреляции 51
- нулевая 28
- Паскаля 29
- наблюдаемости 310
Матричная экспонента 41
Меню командного окна 8, 140
Мертвая зона 350
Методы численного интегрирования дифференциальных уравнений
- Рунге-Кутта 117, 121, 122
- многошаговые 123

- с автоматическим выбором шага 91
Умножение
- действительных чисел 11
- вектора на число 34
- векторов (скалярное) 35
- векторов (векторное) 35
- комплексных чисел 17
- матриц 38
-полиномов 44

Насыщение 353
Норма матрицы 52

Обращение матрицы 39, 54
Окно
- Бартлетта 260
- Блекмана 260
- Кайзера 260
- прямоугольное 260
- треугольное 260
- Хемминга 260
- Хеннинга 260
- Чебышева 260
Оператор
- безусловного перехода 83
- переключения 85
- условного перехода 83
- цикла арифметического 86
- цикла с предпосылкой 85
Описание программы 95
Определитель матрицы 52
Оптимальный регулятор 323
- линейно-квадратичный 322-324
- дискретный 325
- для дискретной системы 325
Организация смены данных в диалоговом режиме 105-108
Отыскание корней
- полиномов 44
- системы линейных алгебраических уравнений 40
- функции 92

Передаточная функция 187-192, 250, 348
Переключатель 355
Переменные
- глобальные 101
- локальные 94
- рабочего пространства 94
- состояния 91, 122
- фазовые 91
Подсистема 359, 369-371
Показатель числа десятичный 9
Полоса задержки фильтра 255
Полоса пропускания фильтра 255
Поэлементное преобразование
- векторов 35
- матриц 37
Представление информации
- графическое 109-112
- текстовое 112-115
Преобразования билинейное 260
Преобразование форм представления фильтра
- от передаточной функции 256-257
- к передаточной функции 255-256
Производная от полинома 46, 47
Пространство состояния 187, 297 348
Псевдообращение матрицы 54

Рабочее пространство 99
Разложение матрицы
- сингулярное 55
- Холецького 53
- LU 53
- QR 54
Ранг матрицы 52
Реле 355
Ряд Фурье 235

Свертка векторов 44
Сигнум-функция 15, 353
Сингулярные числа матрицы 55, 56
Сингулярные значения системы 308
Синтез линейных стационарных систем 319-323

Система
- многомерная 188
- линейная стационарная 187, 297
- одномерная 188
След матрицы 52
Сложение
- действительных чисел 11
- комплексных чисел 17
- векторов 34
- матриц 38
Собственные векторы матрицы 55
Собственные значения
- матричного полинома 59
- матрицы 55
Сортировка комплексных чисел 18
Спектр
- амплитудный 234
- действительный 234
- комплексный 240
- мысленный 234
- фазовый 234
- частотный 234
Спектральная плотность 237, 248
Спектральный анализ 234-247, 280
Спектрограмма 282
Статистический анализ 248-250
Создание нового класса вычислительных объектов 194-204
Столбцовая диаграмма 75
Сумматор 348, 350
Сухое и вязкое трение 353

Транспонирование
- вектора 34
- матрицы 39

Фазочастотная характеристика 83
Файл-сценарий 93, 94, 101-108
Файл-функция 93, 94 96-99
Фильтр 218
- аналоговый 258
- Баттерворта 261
- Бесселя 261
- БІХ 259
- высоких частот 259
- эллиптический 259
- идеальный 259
- Калмана 310
- - дискретный 311
- КІХ 259
- нерекурсивный 259
- низких частот 259
- режекторный 259
- резонансный 232
- рекурсивный 259
- полосовой 259
- формирующий 235
- Чебышева 261
Фильтрация
- векторная 63, 231
- двойная 231
Форма Коши дифференциальных уравнений 91
Форма матрицы
- Гессенберга 55
- Шура 56
Форма представления фильтра
- решетчатая 251
- каскадная 251
- нули-полюса 251
- передаточная функция 250
- сумма простых дробей 250
- в пространстве состояний 251
Формат представления чисел 9
Формирование
- случайных процессов 231-233, 338, 345-347
- импульсных процессов 215-218
- колебаний 218-224, 335-338, 344-347
- матриц 28-31
Функции
- Бесселя 16
- бета 16
- гамма 16
- гиперболические 14
- экспоненциальные 15
- элементарные 14
- эллиптические 16

- комплексного аргумента 17
- логарифмические 15, 16
- округления 15, 352
- преобразования координат 16
- погрешностей 16
- специальные 15-16
- тригонометрические 14
- целочисленные 15
Функция
- Лежандра 16
- затухания фильтра 255
- когерентности сигналов 279
- функции 90, 116-122
Фурье-изображение 235
- случайного процесса 244-247
- дискретное 64, 236
- полигармонических колебаний 242-244
- прямоугольного импульса 239-241
Фурье-преобразование
- дискретное 64
- обратное 235
- обратное дискретное 64
- прямое 235

Характеристика фильтра
- групповой задержки 252
- затухания 252
- фазовая 252
Характеристический полином 45, 55

Частота
- среза 255
- Найквиста 66
Частотная передаточная функция 71
Численное интегрирование системы дифференциальных уравнений 91, 117-122
Число обусловленности матрицы 51-52

Указатель операторов, команд, функций и функциональных блоков MatLAB

+

- 11, 17, 34, 36, 38, 184
! 162
\ 11, 17, 184
% 95
& 84
*** 11, 17, 34, 35, 38, 184**
' 34, 184
.* 36, 37
./ 36, 37
.^ 36, 37
/ 11, 17, 40, 184
: 27, 32, 33
; 27, 95, 170
^ 11, 17, 39
| 84
~ 84
~ = 84
+ 11, 17, 34, 36, 38, 184
< 84
<= 84
= = 84
> 84
>= 84
[,] 26, 185
(,) 31

A

abs 15, 17
Abs 341, 376
acker 186
acos 14
acosh 14
acot 15
acoth 15
acsc 15
acsch 15
angle 17
ans 11, 13
append 186, 291
argnames 177, 179
array 169
asec 15
asech 15
asin 14
asinh 14
atan 14
atan2 15
atanh 15
augstate 182, 185
axis 79, 113

B

BackLash 343, 376
balreal 182, 185
Band-Limited White Noise 325, 336
bar 75
bartlett 256
besselap 249
besself 250
besseli 16
besselj 16
besselk 16
bessely 16
beta 16
betainc 16
betaln 16
bilinear 251
blackman 256
bode 182, 297
boxcar 256
break 86
buttap 249
butter 250
buttord 248

C

c2d 184, 288
canon 182, 185
care 186, 311, 312
cart2pol 16
cart2sph 16
case 85
cat 177
cd 162
cedit 163
ceil 15
cell 168, 169, 173, 174
celldisp 174
cellplot 175
char 168, 169, 174, 177, 178, 196
cheb1ap 249
cheb2ap 249
cheb1ord 248
cheb2ord 248
chebwin 256
cheby1 250
cheby2 250
chirp 214
Chirp Signal 324, 333
chol 53
class 176, 184, 195
clc 163
clear 162
Clock 324
cohere 271
colon 199
Combinatorial Logic 342
comet 77
cond 51
conj 17
connect 108, 182, 186, 291
Constant 324, 325
conv 44
corrcoeff 51
cos 14, 210
cosh 14
cot 15
coth 15
Coulomb & Viscous Friction 343
cov 51
covar 182, 185
cplxpair 18
cremez 261, 264, 266
cross 35, 395
csc 15

csch 15
csd 270
ctranspose 199
ctrb 182, 185
ctrldemo 186
cumprod 48, 49
cumsum 48, 49

D

d2c 184, 289
d2d 184, 289
damp 182, 185, 299
dare 186, 311, 312
Data Store Memory 348
Data Store Read 348
Data Store Write 348
dcgain 185
Dead Zone 343, 376
deconv 44
delete 162
demo 162
Demux 347
Derivative 337
det 52
diag 30
diary 162
diff 48, 49, 201
Digital clock 324
dir 162
diric 214
Discrete Filter 346
Discrete Pulse Generator 324, 331
Discrete State-Space 346
Discrete Transfer Fcn 346
Discrete Zero-Pole 346
Discrete-Time Integrator 345
diskdemo 186
disp 17, 86, 102, 103, 162, 177, 178
display 177, 178, 182, 97, 198,199
Display 316, 320
dlqr 310
dlyap 186, 311, 312
Dot Product 338
double 168, 171, 201
drss 185
dsort 185
dss 184, 190, 287
dssdata 182, 184

E

echo 182
eig 55, 182, 185
ellip 250
ellipap 249
ellipj 16
ellipke 16
ellipord 248
else 83
elseif 84
Enable 348
end 32, 83, 95
eps 13
eq 199
erf 16
erfc 16
erfcx 16
erfinv 16
esort 185
estim 182, 186, 311
evalfr 182, 185
exp 15
expint 16
expm 41
expm1 41
expm2 41
expm3 41
eye 28

F

Fcn 342
feedback 186, 291, 292
feval 116, 120, 177, 180
fft 64, 228
fftshift 67, 230
fgrid 182
fieldnames 172
figure 82
filt 184, 287
filter 63, 219, 251
filtfilt 221
find 169
findstr 170
fir1 255
fir2 256
fircls 257, 264, 265
fircls1 259, 264, 266
First-Order Hold 346
fix 15
fliplr 29
flipud 29
floor 15
fmin 92
fmins 92
for 83, 86
format 163
formula 177, 178
fplot 92
freqresp 185
freqs 216, 263
freqz 218, 263
From 348
From File 325
From Workspace 325, 378
function 94, 96
fzero 92

G

Gain 337
gamma 16
gammainc 16
gammaln 16
gauspuls 209
gcd 16
ge 199
get 182, 183, 184, 294
get_param 391
getenv 162
global 101
Goto 348
Goto Tag Visibility 348
gram 182, 183, 299
grid 73
Ground 348
grpdelay 266
gt 199
gtext 81

H

hadamard 28
hamming 256
hankel 30
hanning 256
help 15, 95, 97, 162
hess 56
hilb 29
hist 76
Hit Crossing 343, 377
hold off 82
hold on 82
home 163
horzcat 177, 199, 291

I

i 13, 14
IC 348
if 83
ifft 64, 228
imag 17, 169
impinvar 252
impulse 182, 185, 295
impz 266
In 347
inf 13
info 163
inherit 182
initial 182, 185, 295
inline 177, 178
input 102, 103, 106
Integrator 338, 377
interp1 61
inv 39, 54, 185, 291
invhilb 29, 196
isa 184
isct 182
isdt 184
isempty 184
isproper 184
issiso 182, 184

J

j 13, 14
jetdemo 186

K

kaizer 256
kalman 182, 186, 310
kalmd 182, 186, 311
kalmdemo 186
keyboard 102, 104

L

latc2tf 244
lcm 16
ldivide 199
le 199
legendre 16
length 162, 169
load 162
log 15
log10 15
log2 16
Logical Operator 342
loglog 78, 79
logm 41
logspace 78
lookfor 162
Look-up Table 343
Look-up Table(2D) 343
lower 169, 170
lp2bp 249
lp2bs 249
lp2hp 249
lp2lp 249
lqgreg 182, 186, 311
lqr 186, 308
lqrd 186, 310
lqry 182, 186, 309
lsim 182, 185, 295
lt 199
lti 177, 182, 183
lticheck 182
ltiprops 184
ltiview 185, 301
lu 53
lyap 186, 311, 312

M

Manual Switch 345
margin 182, 185, 297
mat2str 171
Math Function 341
MATLAB Fcn 342
matlabrc 163
Matrix Gain 338
max 48, 49
maxflat 252, 264
mdlDerivatives 382
mdlGetTimeOfNext VarHit 382
mdlInitializeSizes 382
mdlOutputs 382
mdlTerminate 382
mdlUpdate 382
mean 48, 49
Memory 344
menu 102, 103, 105
Merge 348
methods 203
milldemo 186
min 48, 49
MinMax 341, 377
minreal 185
minus 198, 201
mldivide 198
mod 15
modred 182, 185
more 163
mpower 199
mrdivide 198, 202
mtimes 198, 202, 291
Multiport Switch 345
Mux 375, 395

N

NaN 13
nargin 177
nargout 177
ndims 169
nichols 182, 297
norm 51, 182, 185
null 52
num2str 102, 171
nyquist 282, 297

O

obsv 182, 185
ode23 91, 121
ode45 91, 92,

121, 130
odeset 91
ones 28
orth 52
otherwise 85
Out 347

P

pack 162
pade 182, 185
parallel 182, 186, 290, 292
pascal 29
path 162
pause 102, 104
permute 169
pi 13
pinv 54
place 186
plot 72, 110, 202
plus 185, 299
pol2cart 16
pole 208, 318
poly 45, 55
poly2rc 245
polyder 46
polyeig 59
polyfit 60
polyval 45, 78, 202
polyvalm 59
pow2 16
power 199
prod 48, 49
Product 342
psd 240, 270
Pulse Generator 324, 332
pulstran 212
pzmap 182, 185, 299

Q

qr 54
quad 90
quad8 91
Quantizer 343
quickset 182
quit 163
qz 57

R

Ramp 324, 329
rand 28, 31
randn 28, 31, 77, 223
Random Number 325, 334
rank 52
rat 16
rats 16
rcond 52
rdivide 199, 203
real 17, 169
realmax 13
realmin 13
rectpuls 207
reg 182, 186
Relational Operator 342, 377
Relay 344, 377
rem 15
remez 259
Repeating Sequence 324, 330
reshape 30, 169
residue 244
residuez 244
return 104
rlocfind 182, 186
rlocus 182, 186, 299
rmfield 173
roots 44, 202
rot90 29
round 15
Rounding Function 341
rref 52
rsf2csf 57
rss 185

S

Saturation 343, 377
save 162
sawtooth 211
schur 56
Scope 316
sec 15
sech 15
Selector 348
semilogx 78, 79
semilogy 78, 110
series 182, 186, 291
set 182, 183, 184, 287, 293
set_param 390
S-Function 383, 394
SfunTMPL 381
sigma 182, 297
sign 15
Sign 343, 377
Signal Generator 324, 325
sim 380
simset 380
sin 14, 210
sinc 209
Sine Wave 324, 329
sinh 14
size 47, 49, 162, 169, 184, 294
Slider Gain 338, 340
sort 48, 49
sos2ss 245
sos2tf 243
sos2zp 245
sparse 168
specgram 274
spectrum 271
sph2cart 16
spline 61
sprintf 102, 103, 106
sptool 275
sqrt 15, 17
sqrtm 42
square 210
ss 184, 188, 284
ss2sos 245
ss2ss 182, 185
ss2tf 243
ss2zp 245
ssbal 185
ssdata 184, 293
star 186
startup 163
State-Space 337
std 48, 49
stem 75
step 182, 185, 295
Step 324, 327, 377
Stop Simulation 316

str2mat 171
str2num 171
strcat 169
strcmp 169
strips 268
strrep 170
struct 168, 169, 172, 177
strvcat 170
subplot 80, 110, 111
subsasgn 199
subscribe 163
subsindex 169, 199
subspace 59
subsref 177, 199
Subsystem 348, 391, 394
sum 48, 49
Sum 338, 339
svd 55
switch 83, 85
Switch 345
sym 177

T

tan 14
tanh 14
Terminator 348
text 81, 113
tf 184, 188, 190, 192, 287
tf2latc 245
tf2ss 244
tf2zp 244
tfdata 182, 184, 293
tfe 275
times 199
title 73
To File 316, 322
To Workspace 316, 323, 378
trace 52
trange 182
Transfer Fcn 337
Transport Delay 344
transpose 169
trapz 49
triang 256
Trigger 348
Trigonometric Function 341
tril 30
tripuls 208
triu 30
type 162
tzero 182, 185

U

uint8 168, 169
uminus 198
Uniform Random Number 350, 361
Unit Delay 345
unix 162
uplus 182, 198
upper 170

V

Variable Transport Delay 345
vectorize 177, 179
ver 163
version 163
vertcat 177, 199, 291

W

what 162
whatsnew 163
which 162
while 83, 85
who 162
whos 162
Width 348

X

xcorr 241
xlabel 73
xor 84
XY Graph 316, 319

Y

ylabel 73
yulewalk 254

Z

Zero-Order Hold 345
Zero-Pole 337
zeros 28
zp2sos 245
zp2ss 245
zp2tf 243
zpk 184, 189, 191, 287
zpkdata 182, 184, 293, 299
zplane 266