Юрий ЛАЗАРЕВ

Моделирование процессов и технических систем в MATLAB

Учебный курс

Киев – 2004

УДК 681.3.06(075.8)
ББК 32.973.26-018.2 Я73
Л17

Лазарев Юрий Федорович

Л17 MatLAB 6.5. Математическое моделирование физических процессов и технических систем: Учебный курс. - К.: 2004. - 474 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.

Илл. 487. Библиогр. 21 назв.

Содержание

Предисловие

В последние годы в университетских и инженерно-технических кругах мира получила широкое распространение новая компьютерная система осуществления математических расчетов - система MatLAB. Более того, в настоящее время система MatLAB принята в качестве официального вычислительного средства при подготовке и оформлении инженерной документации и научных публикаций. В чем причина такой популярности этой системы?

Главные преимущества "языка технических вычислений" MatLAB, выгодно выделяющие 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 в нем 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: 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.

Книга состоит из одиннадцати глав (уроков). Она задумана как учебное пособие для студентов высших технических учебных заведений и естественнонаучных специальностей университетов. Первые четыре урока могут быть рекомендованы в качестве пособия по учебным дисциплинам "Моделирование процессов и систем" и "Математическое моделирование на ЭВМ". Остальная часть может быть использована как пособие по курсовому и дипломному проектированию. В целом книгу можно рассматривать и как введение в MatLAB для инженеров и научных работников в области проектирования технических систем.

В Уроке 1 читатель знакомится с возможностями системы в режиме научного калькулятора. Здесь помещены сведения об основных операторах, командах, функциях и процедурах системы. В Уроке2 описаны правила и примеры составления программ на языке MatLAB. Кроме того, в нем представлены некоторые дополнительные процедуры, которые помогают рационально организовать вычислительный процесс. Урок 3 содержит перечень некоторых процедур и команд общего назначения, которые связывают систему MatLAB с операционной системой компьютера, а также со средствами, позволяющими использовать возможности MatLAB при оформлении документов в текстовом редакторе WORD. Здесь же описан механизм формирования и чтения файлов данных в среде 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 Уроке 4.

В Уроке 5 сосредоточены сведения об особенностях использования процедур цифровой обработки сигналов пакета SIGNAL. Содержание Урока 6 - начальное ознакомление с особенностями работы с процедурами анализа и синтеза линейных стационарных систем автоматического управления пакета CONTROL.

Урок 7 знакомит с ядром пакета SimuLink интерактивного (визуального) моделирования динамических систем во временной области. Содержанием Урока 8 является более глубокое ознакомление с важнейшими средствами SimuLink, позволяющими обеспечить эффективное взаимодействие SimuLink со средой MatLAB.

Наконец Уроки 9, 10 и 11 посвящены ознакомлению читателя с основами использования трех дополнительных библиотек пакета SimuLink: моделирования динамики аэрокосмических объектов (Aerospace Blockset),

моделирования електротехнических систем (SimPowerSystems) и моделирования динамики механизмов (SimMechanics).
Изложение ведется таким образом, чтобы пользователь мог сразу применить полученные знания для проведения вычислений. Книга содержит много примеров, которые поясняют и иллюстрируют работу по использованию процедур.
В пособии использованы материалы из изданий, указанных в списке литературы.
Учебный курс, в основном, ориентирован на русифицированную версию MatLAB 6.5.0.180913a (R13)

Введение

Система MatLAB создана фирмой MathWork Inc. (США, г. Нейтик, штат Массачузетс). Хотя впервые эта система начала использоваться в конце 70-х годов, 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.
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. MatLAB 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 много других. 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. Предусмотрены 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.
В составе системы 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.
Работа 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);
- в программном режиме, то есть 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 при этом язык MatLAB обладает особенностью, которая отличает его 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.

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

Урок 1. MatLab как научный калькулятор

Командное окно

Операции с числами

Простейшие операции с векторами и матрицами

Функции прикладной численной математики

Построение простейших графиков

Операторы управления вычислительным процессом

Работа в среде MatLAB может осуществляться в двух режимах:

> путем ввода в командное окно команд и (или) операторов, которые выполняются сразу, после нажатия клавиши OK; такой режим будем называть командным режимом или режимом научного калькулятора;
>
> путем запуска из командного окна специально написанной на языке MatLAB (M-языке) программы; такой режим можно назвать программным.

В командном режиме пользователю доступны практически все возможности и функции MatLAB, за небольшим исключением. Систему MatLAB обоснованно относят к одному из наиболее мощных научных калькуляторов, которому доступны практически все численные средства решения научных и инженерных задач, разработанные в научных организациях мира на настоящий момент. Использование этих средств в командном режиме является, в большинстве случаев, очень простым. Результат получают сразу в самом командном окне в наглядной простой форме или в графическом виде в дополнительном графическом окне. Поэтому знакомство с MatLAB и освоение приемов работы в ее среде целесообразно начать с изучения возможностей системы именно в командном режиме

1.1. Командное окно

После вызова MatLAB 6.5 из среды Windows на экране появляется окно МАТЛАБ, представленное на рис 1.1. В нем могут отображаться несколько окон. Главным из них является Окно команд, или так называемое командное окно среды MatLAB.

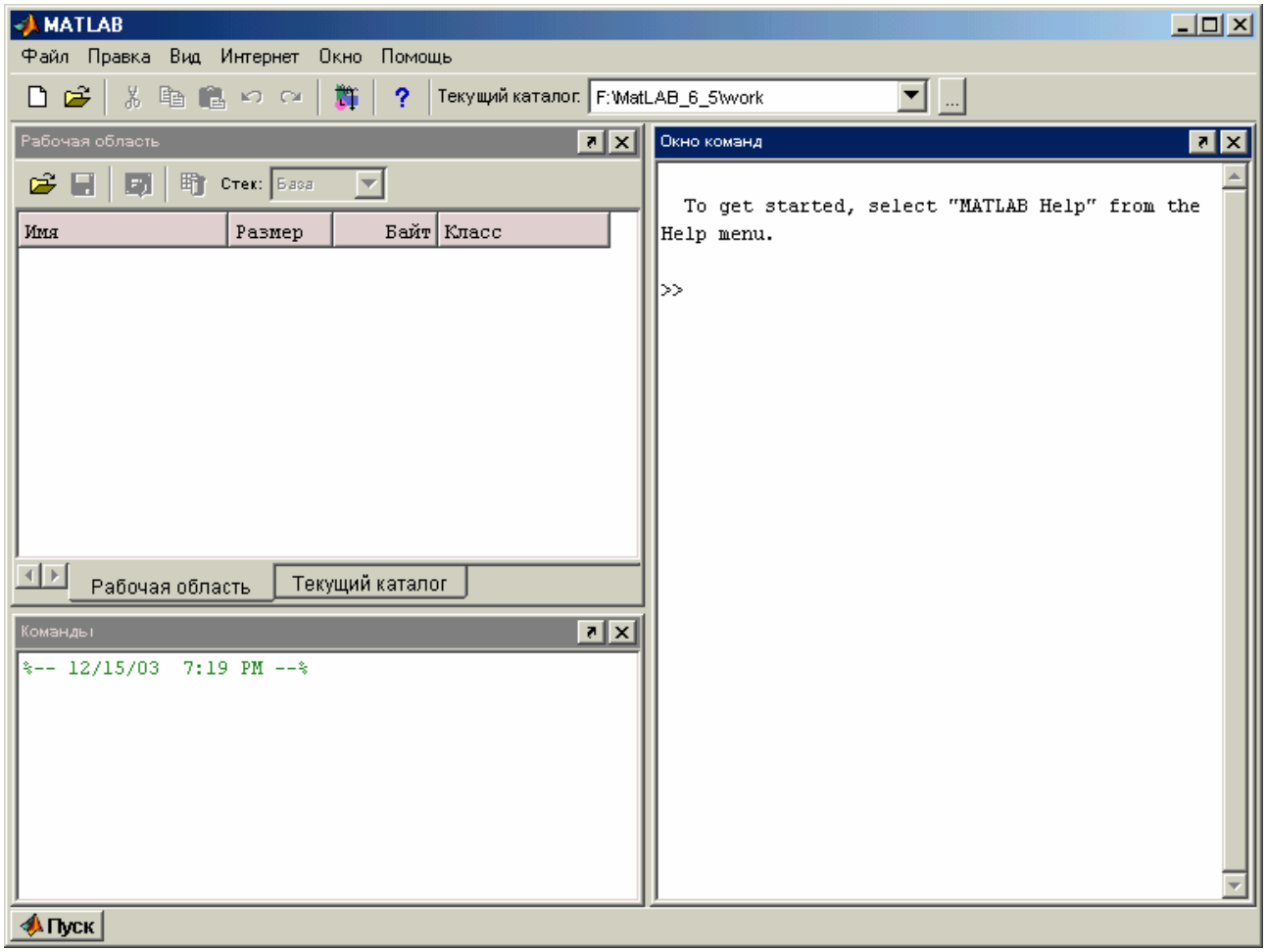


Рис. 1.1. Окно MatLAB

Если закрыть остальные окна, окно MATLAB примет вид, изображенный на рис. 1.2. Командное окно является основным в MatLAB. В нем появляются символы команд, которые набираются пользователем с клавиатуры, отображаются результаты выполнения этих команд, текст исполняемой программы и информация об ошибках выполнения программы, распознанных системой.

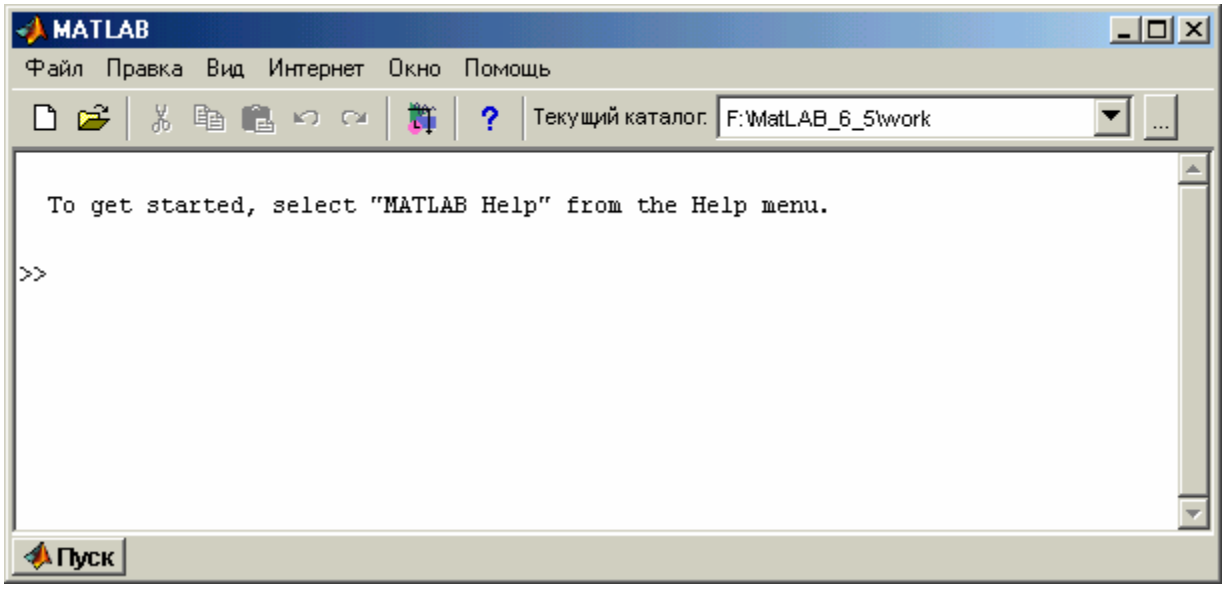


Рис. 1.2. Командное окно MatLAB

Признаком того, что MatLAB готова к восприятию и выполнению очередной команды, является наличие в последней строке командного окна знака приглашения », после которого расположена мигающий курсор.

В верхней части окна (под заголовком) расположена строка меню,. Чтобы открыть какое-либо меню, следует установить на нем курсор мыши и нажать ее левую кнопку. Здесь отметим лишь, что для выхода из среды MatLAB достаточно открыть меню Файл и выбрать в нем команду Выход из MATLAB, или просто закрыть командное окно, щелкнув мышью на кнопке закрытия окна с изображением крестика.

1.2. Операции с числами

Основные объекты системы MatLAB – числа. Операции с ними лежат в основе всей работы с этой системой.

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, показанная на рис. 1.3.

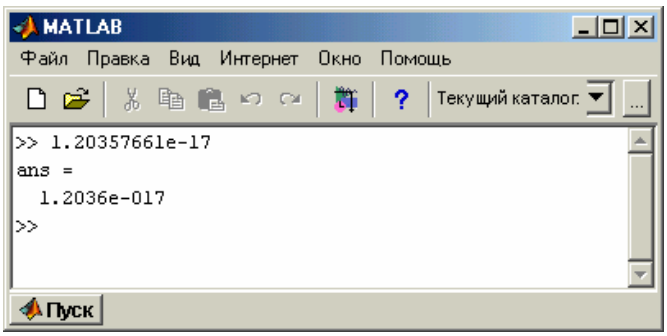


Рис. 1.3. Ввод и вывод числа

Видно, что выведенное на экран число не совпадает с введенным. Это обусловлено тем, что результат вычислений в MatLAB выводится в виде (формате), который определяется предварительно установленным форматом представления чисел. Этот формат может быть установлен с помощью команды Файл ▸ Предпочтения. После ее вызова на экране появится одноименное окно (рис. 1.4).

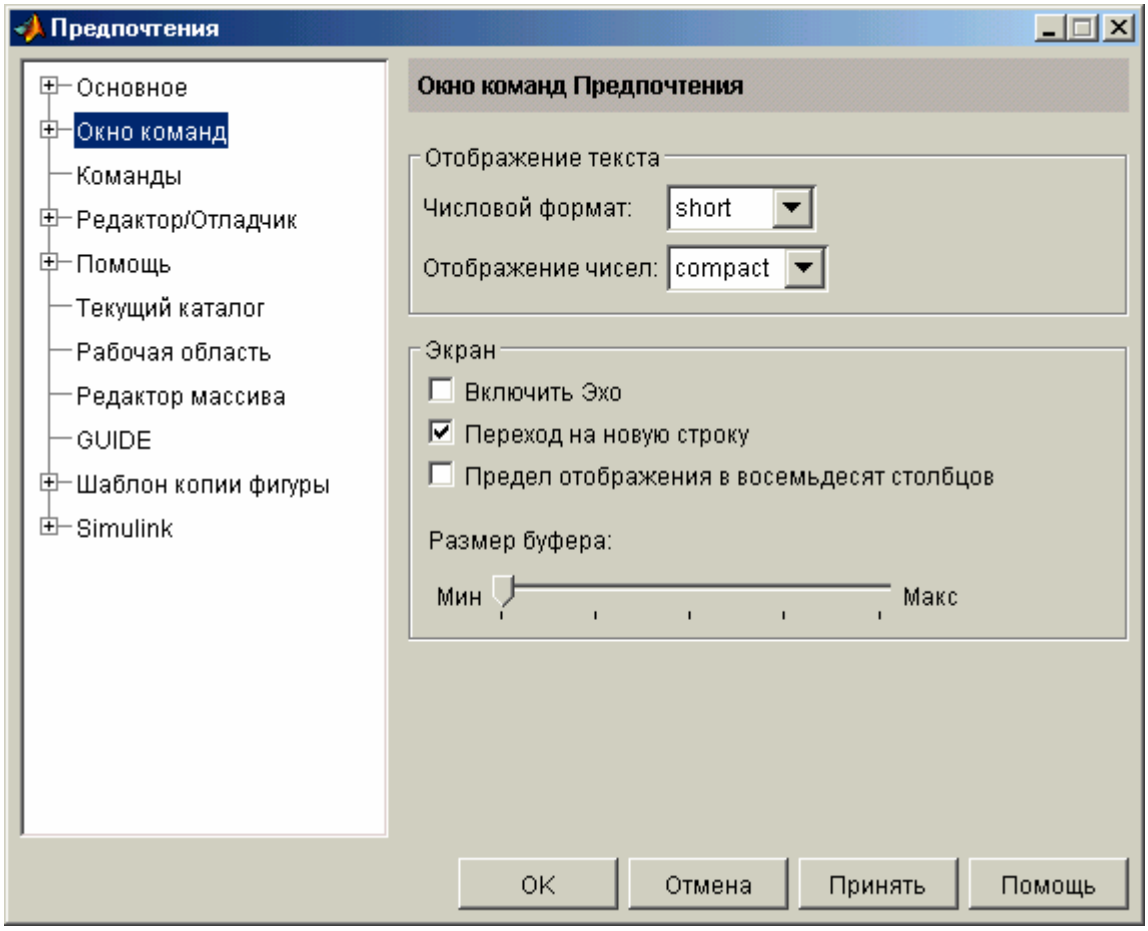


Рис. 1.4. Окно Предпочтения

Один из участков этого окна имеет название Числовой формат. Он предназначен для установки и изменения формата представления чисел, которые выводятся в командное окно в процессе расчетов. Предусмотрены такие форматы (рис 1.5):

Short (default) краткая запись (применяется по умолчанию) с фиксированной запятой;
Long длинная запись с фиксированной запятой;
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;
Hex	запись в виде шестнадцатеричного числа;
Bank	запись до сотых долей;
+	записывается только знак числа;
Rational	запись в виде рациональной дроби.

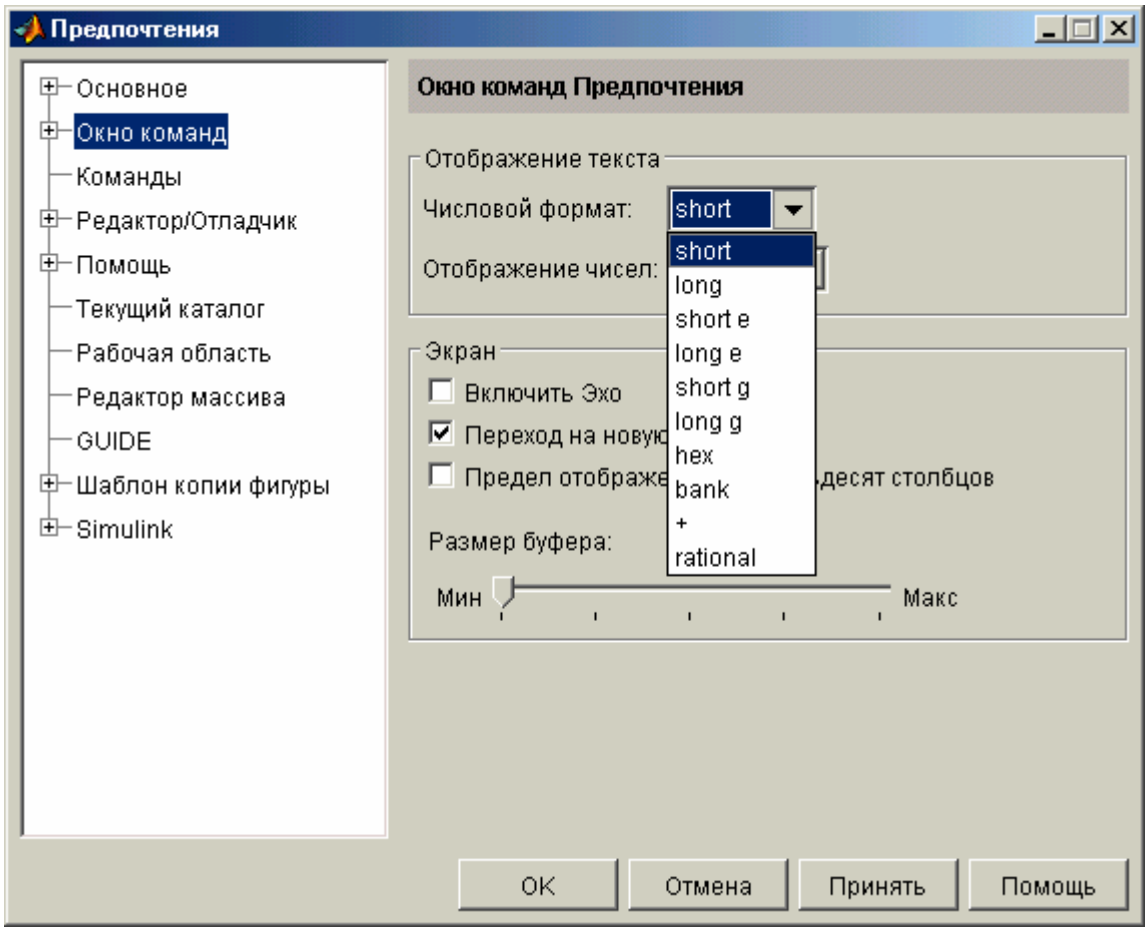


Рис. 1.5. Меню числового формата

Выбирая нужный вид представления чисел, можно обеспечить в дальнейшем выведение чисел в командное окно именно в этой форме.

Как видно из рис. 1.3, число, которое выведено на экран, не совпадает с введенным числом. Это обусловлено тем, что установленный по умолчанию формат представления чисел (Short E) не позволяет вывести больше 6 значащих цифр. На самом деле введенное число сохраняется внутри MatLAB со всеми введенными его цифрами. Например, если выбрать формат Long E, то, повторяя те же действия, получим представление числа, где все цифры отображены верно (рис. 1.6).

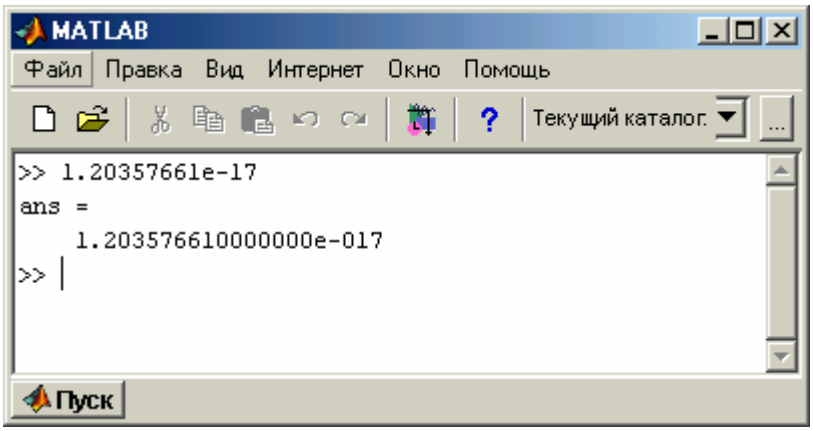


Рис. 1.6. Представление числа в формате Long E

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

Использование MatLAB в режиме калькулятора может происходить путем простой записи в командную строку последовательности арифметических действий с числами, т. е. обычного арифметического выражения, например:

```
(4.5)2*7.23 - 3.14*10.4
```

Если после ввода с клавиатуры этой последовательности нажать клавишу Enter, в командном окне возникнет результат выполнения в виде, представленном на рис. 1.7, т. е. на экран под именем системной переменной **ans** выводится результат действия последнего выполненного оператора.

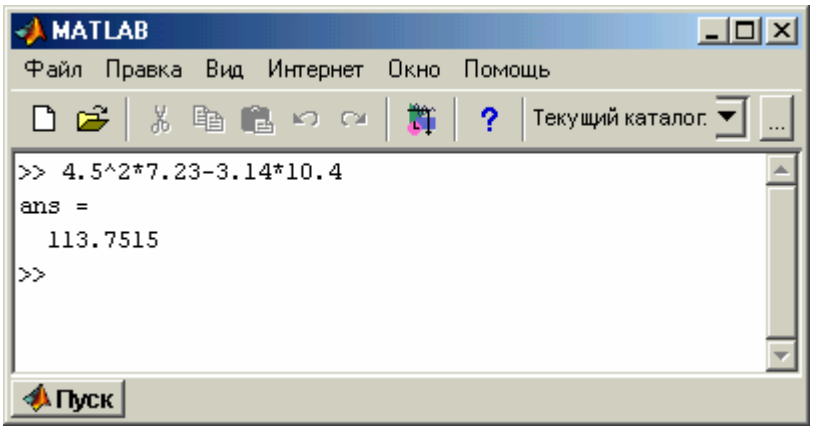


Рис. 1.7. Пример вычисления выражения

Вообще вывод промежуточной информации в командное окно подчиняется таким правилам:

если запись оператора не заканчивается символом «;», результат действия этого оператора сразу же выводится в командное окно;

если оператор заканчивается символом «;», результат его действия не отображается в командном окне;

если оператор не содержит знака присваивания (=), т. е. является просто записью некоторой последовательности действий над числами и переменными, значение результата присваивается специальной системной переменной по имени **ans**;

полученное значение переменной **ans** можно использовать в следующих операторах вычислений, используя 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. 1.8).

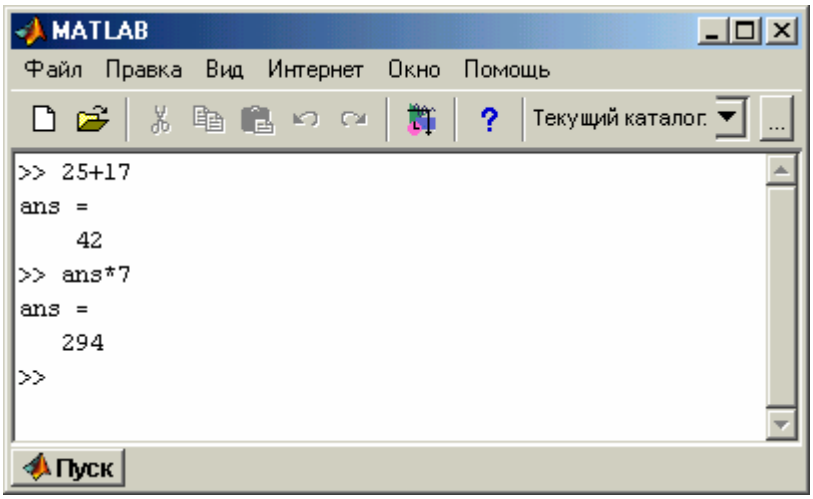


Рис. 1.8. Использование переменной ans

Чтобы предотвратить выведение промежуточного результата действия 25+17, достаточно после записи этой последовательности добавить символ « ; ». Тогда получим результаты в виде, представленном на рис. 1.9.

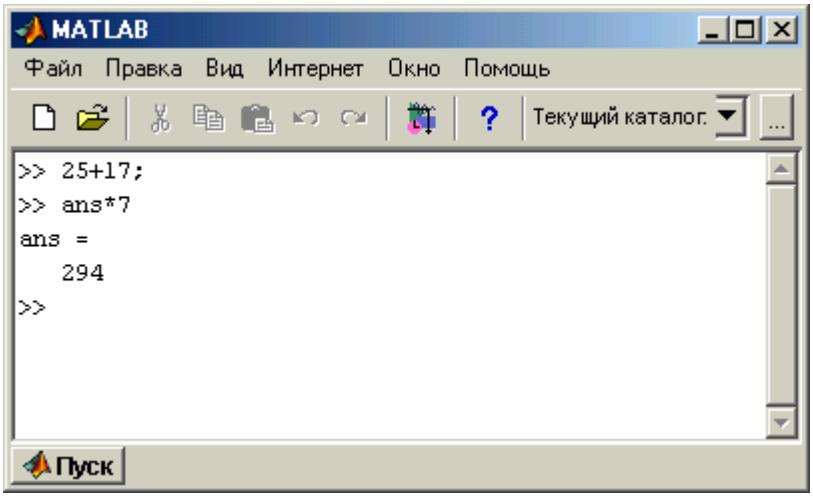


Рис. 1.9. Предотвращение вывода на экран

Используя 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 не должно совпадать с именами функций, процедур системы и системных переменных. При этом система различает большие и малые 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>.
```

Например, если ввести в командное окно строку **x = 25 + 17**, на экране появится запись (рис. 1.10):

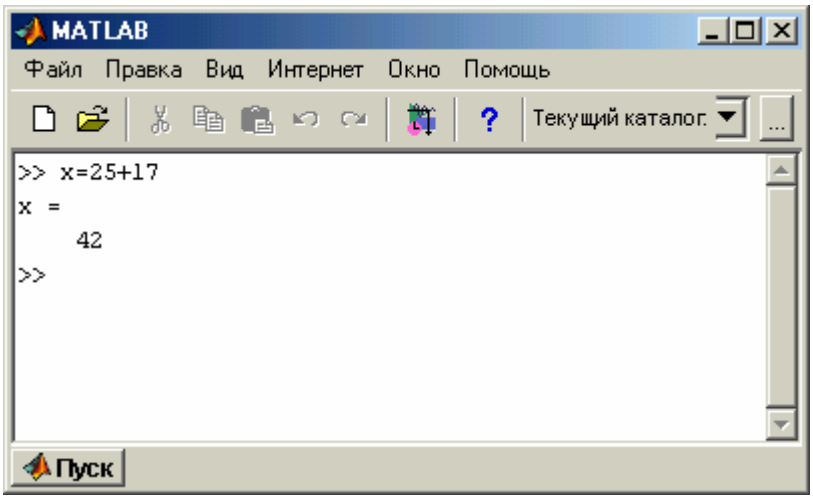


Рис. 1.10. Присвоение значения переменной

Система MatLAB имеет несколько имен переменных, которые используются самой системой и входят в состав зарезервированных (эти переменные можно использовать в математических выражениях):

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

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. Из примера, приведенного на рис. 1.11, видно, в каком виде система выводит комплексные числа на экран (и на печать).

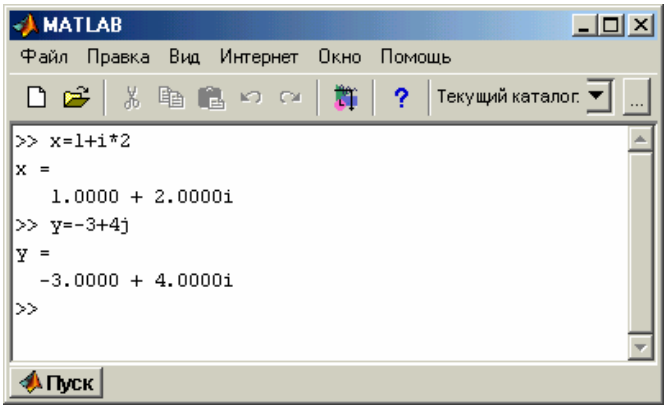


Рис. 1.11. Ввод комплексных чисел

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 математические функции.

Тригонометрические и гиперболические функции

sin(Z)	синус числа Z;
sinh(Z)	гиперболический синус;
asin(Z)	арксинус (в радианах, от π /2 до +π /2);
asinh(Z)	обратный гиперболический синус;
cos(Z)	косинус;
cosh(Z)	гиперболический косинус;
acos(Z)	арккосинус (в диапазоне от 0 до π);
acosh(Z)	обратный гиперболический косинус;
tan(Z)	тангенс;
tanh(Z)	гиперболический тангенс;
atan(Z)	арктангенс (в диапазоне от π /2 до +π /2);
atan2(X,Y)	четырехквадрантный арктангенс (угол в диапазоне (-π , +π] 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-функции числа Z (0 при Z=0, -1 при Z<0, 1 при Z>0)

1.2.5. Специальные математические функции

Кроме элементарных, в языке MatLAB предусмотрен целый ряд специальных математических функций. Ниже приведен перечень и краткое 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	усеченная Бета- функция;
betaln	логарифм Бета- функции

Гамма-функции

gamma	Гамма- функция;
gammainc	усеченная Гамма-функция;
gammaln	логарифм Гамма- функции

Эллиптические функции и интегралы

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

Функции ошибок

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

Другие функции

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

1.2.6. Элементарные действия с комплексными числами

Простейшие действия с комплексными числами - сложение, вычитание, умножение, деление и возведение в степень - осуществляются при помощи обычных арифметических знаков +, -, *, /, \ и ^ соответственно. Примеры использования приведены на рис. 1.12.

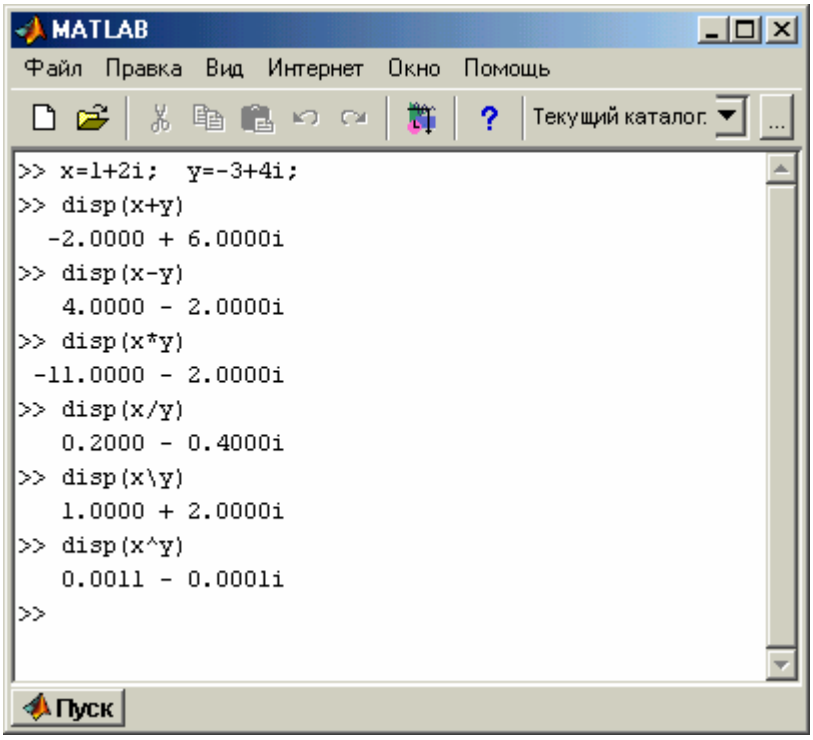


Рис. 1.12. Действия с комплексными числами

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.

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.

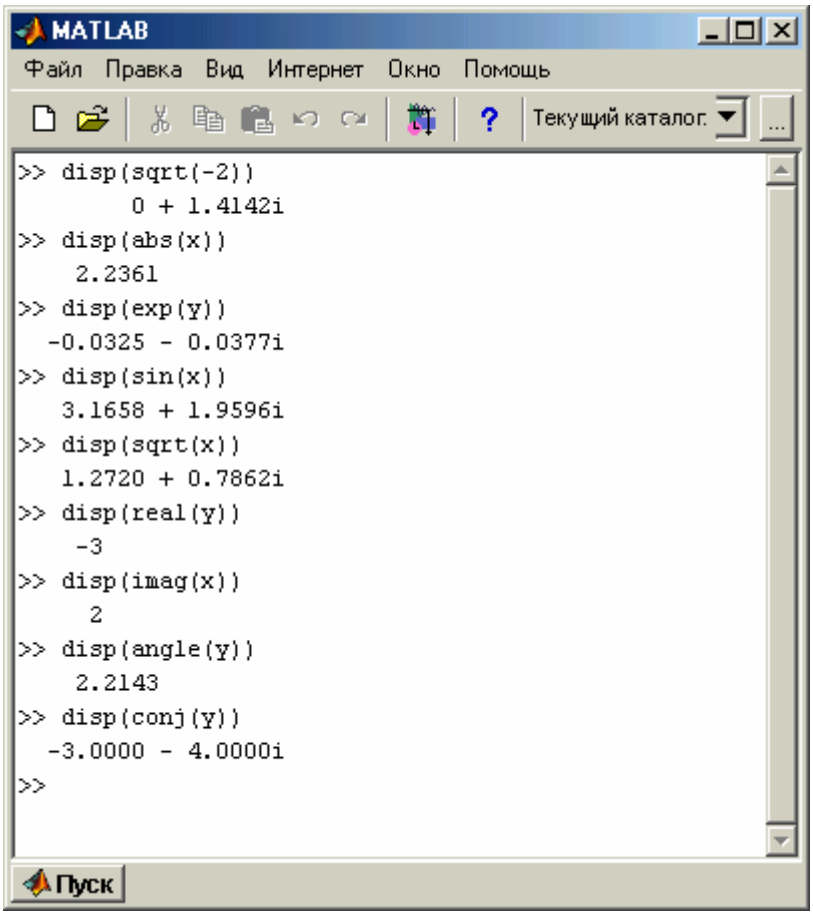


Рис. 1.13. Функции комплексного аргумента

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 =
  -1.0000   -1.0000 + 2.0000i  -5.0000    4.0000    0 + 5.0000i
  -1.0000 - 2.0000i         0 - 5.0000i

disp(cplxpair(v))
  -1.0000 - 2.0000i  -1.0000 + 2.0000i  0 - 5.0000i  0 + 5.0000i
   -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.

Прпмечание. 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.

1.3. 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):

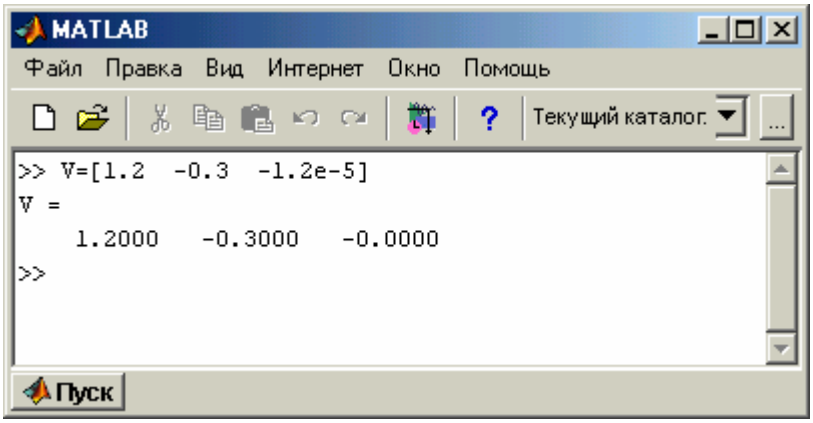


Рис. 1.14. Ввод вектора

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 не более четырех цифр после десятичной запятой.

Длинный вектор можно вводить частями, которые потом объединять с помощью операции *объединения векторов в строку* : **`v = [ v1 v2 ]`**. Например (см. рис.1.15):

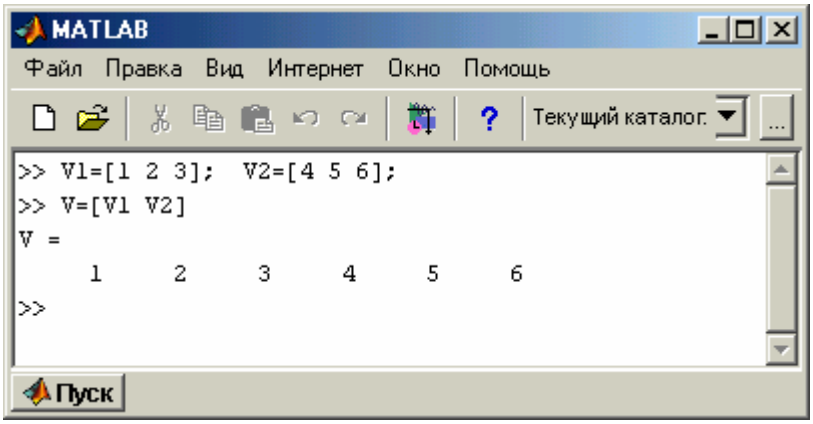


Рис. 1.15. Объединение векторов

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

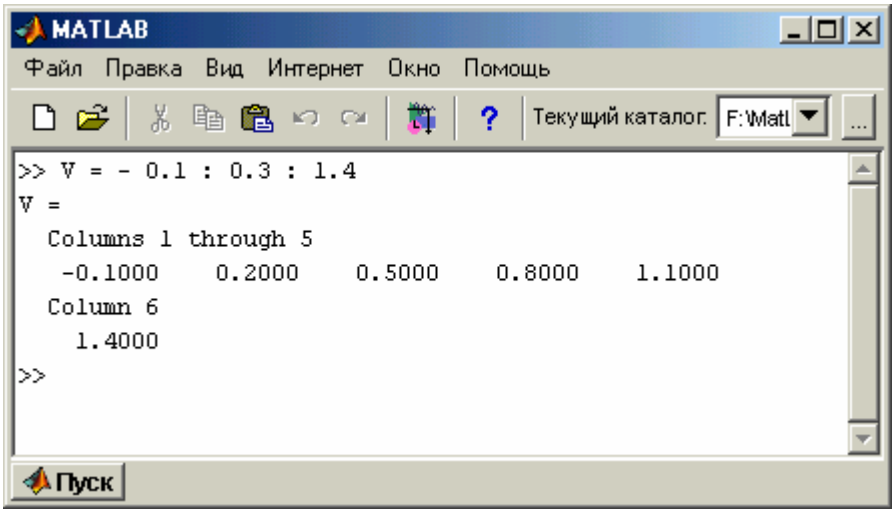


Рис. 1.16. Ввод вектора – арифметической прогрессии

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).

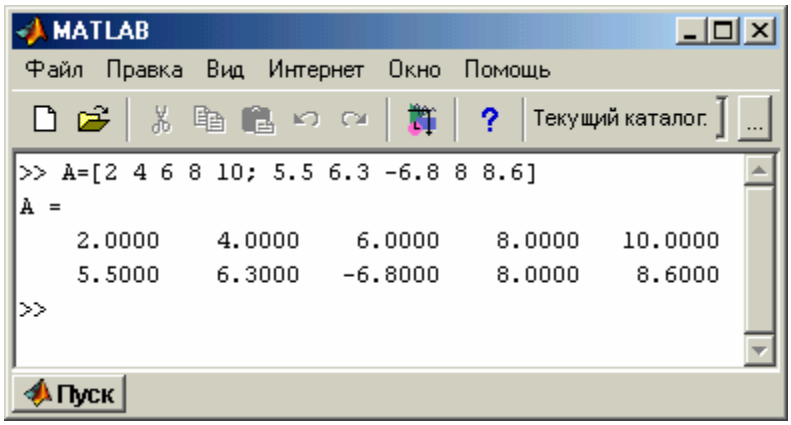


Рис. 1.17. Ввод матрицы

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 Гильберта 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, т. е. меняя местами левую и правую стороны матрицы, 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, т. е. меняя местами верхнюю и нижнюю стороны матрицы, 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 последующего распределения 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(x)** 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 оба 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 после имени транспонируемого вектора. 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 предусмотрена функция **cross**, которая позволяет найти ***векторное произведение двух векторов***. При этом, 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, хотя не являются операциями с вектором как с 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, элементы которого равны 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 является вектор того же типа и размера.*

Добавление (отнимание) числа к (из) каждому элементу 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 равна 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 их графики) сложные математические функции, не используя при этом операторов 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, все элементы которой равны единице, 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 А. Исходная матрица А 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*** отрицательном).

Приведем пример:

```
» A^2

   ans =

        8     -3    -10
       -5     10     16
       -2      4      9

» A^(-2)

   ans =

     1.5385e-001  -7.6923e-002    3.0769e-001
     7. 6923e-002  3. 0769e-001 -4. 6154e-001
     2. 1328e-018 -1. 5385e-001  3. 8462e-001
```

Оригинальными в языке MatLAB являются две новые, не определяемые в математике функции ***деления матриц***. При этом вводятся понятие ***деления матриц слева напрво*** и ***деления 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:

$X \cdot A = B$.

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:

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

В 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, равной 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
```

1.4. Функции прикладной численной математики

К преимуществам системы MatLAB относится то, что она содержит в своем составе большое число функций и процедур, выполняющих стандартные математические операции, используемые в прикладной (инженерной) математике. Сюда можно отнести операции с полиномами, обработку данных измерений, функции линейной алгебры, аппроксимацию и интерполяцию данных, векторную фильтрацию и спектральный анализ сигналов. Далее ознакомимся с важнейшими из них.

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 среде 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 выполнено соотношение

B = conv(A,Q) + R.

В системе MatLAB предусмотрена функция **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 содержит 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

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

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 (стандартное) 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, а элементы его 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 и 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 аргумента из 100 элементов. Тогда вычисления сведутся к совокупности операций:

```
» x = (0 : 0.01:1)*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 симметричную квадратную матрицу с количеством строк и столбцов, равным 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 (p=2).

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) = σ_{max}(A).

Функция **rd = rcond(A)** вычисляет величину, обратную значению числа обусловленности матрицы A относительно 1-нормы. Если матрица A хорошо обусловлена, 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 симметричных действительных 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 Хессенберга 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 Хессенберга, 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 пары матриц к форме Шура возникает в 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:

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

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

причем матрицы ***Q***, ***R*** и векторы ***b***, ***c*** и ***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

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 + λ *A1 + ... + λ^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 = [-0.45 -0.35 -0.25 -0.15 -0.05 0.05 0.15 0.25 0.35 0.45];

а массив соответствующих значений измеренной величины - вид:

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

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

```
» x = -0.45:0.1:0.45;
» y = [-1.1 0.2 0.1 0.8 0.5 0.2 0.4 0.1 0.1 -0.6];
» polyfit(x,y,1)
   ans =      0.13939         0.07
» polyfit(x,y,2)
   ans =   -6.1364      0.13939      0.57625
» polyfit(x,y,3)
   ans =    7.4592      -6.1364     -0.95338      0.57625
» polyfit(x,y,4)
   ans =   -35.548       7.4592       1.1509     -0.95338     0.40469.
```

Это означает, что заданную зависимость можно аппроксимировать или прямой

$$y(x) = 0{,}13939x + 0{,}07 ,$$

или квадратной параболой

$$y(x) = -6{,}1364x^2 + 0{,}13939x + 0{,}57625 ,$$

или кубической параболой

$$y(x) = 7{,}4592x^3 - 6{,}1364x^2 - 0{,}95338x + 0{,}57625 ,$$

или параболой четвертой степени

$$y(x) = -35{,}548x^4 + 7{,}4592x^3 + 1{,}1509x^2 - 0{,}95338x + 0{,}40469 .$$

Построим в одном графическом поле графики заданной дискретной функции и графики полученных при аппроксимации полиномов рис. 1.18:

```
x = -0.45:0.1:0.45;
y = [-1.1 0.2 0.1 0.8 0.5 0.2 0.4 0.1 0.1 -0.6];
stem(x,y); hold on
 P1=polyfit(x,y,1)
 P2=polyfit(x,y,2)
 P3=polyfit(x,y,3)
 P4=polyfit(x,y,4)
x1 = -0.5 : 0.01 : 0.5;
y1=polyval(P1,x1);
y2=polyval(P2,x1);
y3=polyval(P3,x1);
y4=polyval(P4,x1);
plot(x1,y1, '--',x1,y2,':',x1,y3, '.',x1,y4),
grid, set(gca, 'FontSize', 12),
title('Полиномиальная аппроксимация ');
xlabel('Аргумент'); ylabel('Функция');
legend('исходные','данные','Аппроксимация','линейная','квадратическая','кубическая','четв
ертой степени',0)
```

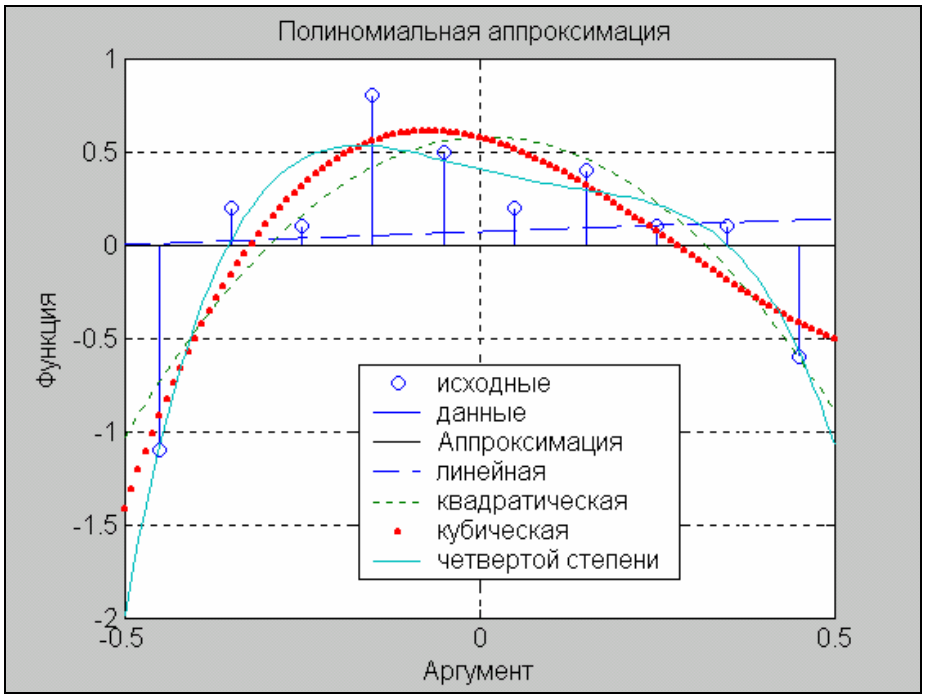


Рис. 1.18. Результаты применения функции polyfit

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

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

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 = -0.45:0.1:0.45;
 y = [-1.1 0.2 0.1 0.8 0.5 0.2 0.4 0.1 0.1 -0.6];
 x1 = -0.46:0.01:0.46;
 y2 = spline(x,y,x1);
stem(x,y); hold on
plot (x1,y2, '.'), grid
set(gca, 'FontSize',12),
```

```
title('Интерполяция процедурой SPLINE ');
xlabel('Аргумент');  ylabel('Функция')
```

Результат приведен на рис. 1.19.

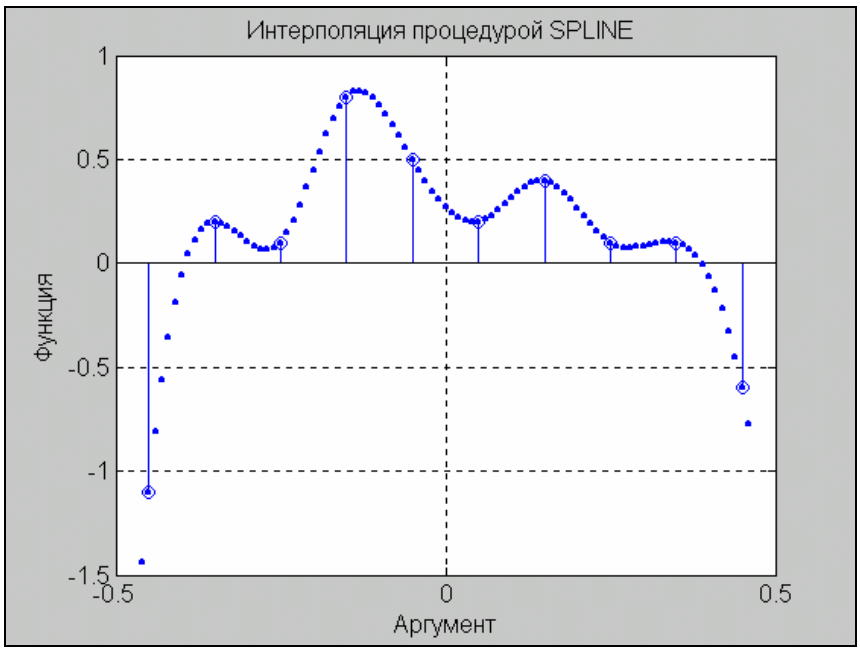


Рис. 1.19. Интерполяция функцией spline

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.

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

```
y = [-1.1 0.2 0.1 0.8 0.5 0.2 0.4 0.1 0.1 -0.6];
 x1 = -0.46:0.01:0.46;
 y1 = interp1(x,y,x1);
 y4 = interp1(x,y,x1,'nearest');
 y2 = interp1(x,y,x1,'cubic');
 y3 = interp1(x,y,x1,'spline');
 plot (x1,y1, '--',x1,y2,'.',x1,y3,x1,y4,':'), grid
set(gca, 'FontSize',12),
 legend('линейная','кубическая','сплайновая','ступенчатая',0)
 title('Интерполяция процедурой INTERP1       ');
 xlabel('Аргумент');       ylabel('Функция')
```

Результаты приведены на рис.1.20.

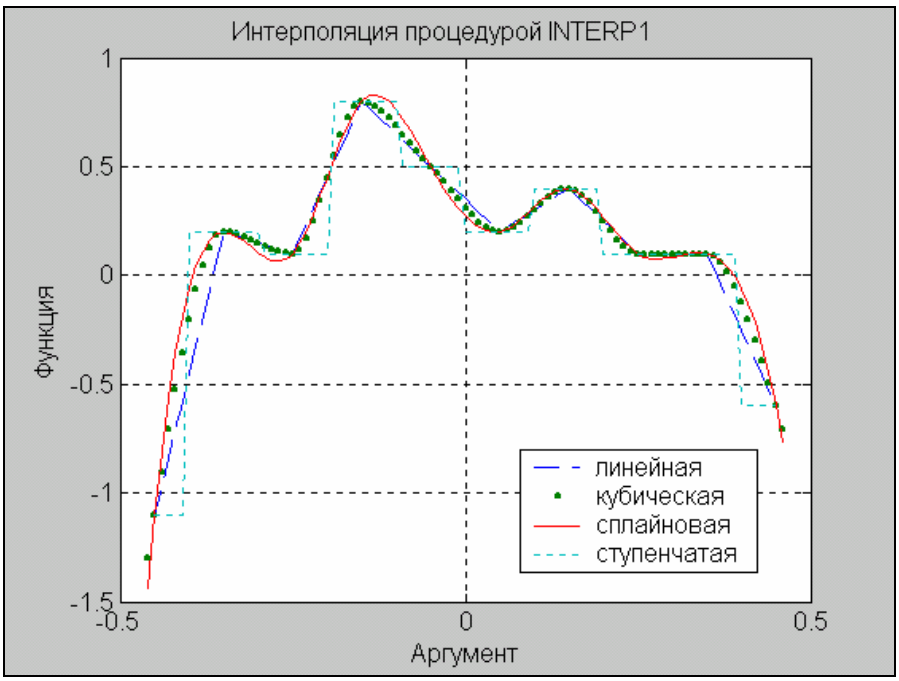


Рис. 1.20. Интерполяция функцией interp1

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) + ... + b(nb+1)*x(k-nb) --a(2)*y(k-1) - a(3)*y(k-3) - ... - a(na+1)*y(k-na), (1.1)

где вектор ***b*** имеет такой состав

b = [b(1), b(2),... , b(nb+1)],

а вектор ***a***

a = [1, a(2), a(3),... , 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 =

         0    0.1000    0.3900    0.2610   -0.5861    0.3146    3.9129
    0.2503  -13.4768    2.8466   56.4225
```

Функции **`fft`** (*Fast Fourier Transformation*) и **`ifft`** (*Inverse Fast Fourier Transformation*) осуществляют преобразования заданного вектора, соответствующие 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*** - во втором по формулам:

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

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

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);
figure, plot(t, x); grid,
set(gcf,'color','white'), set(gca,'FontName','Arial Cyr','FontSize',16),
title('Входной процесс ');xlabel('Время (с)');      ylabel('X(t)')
```

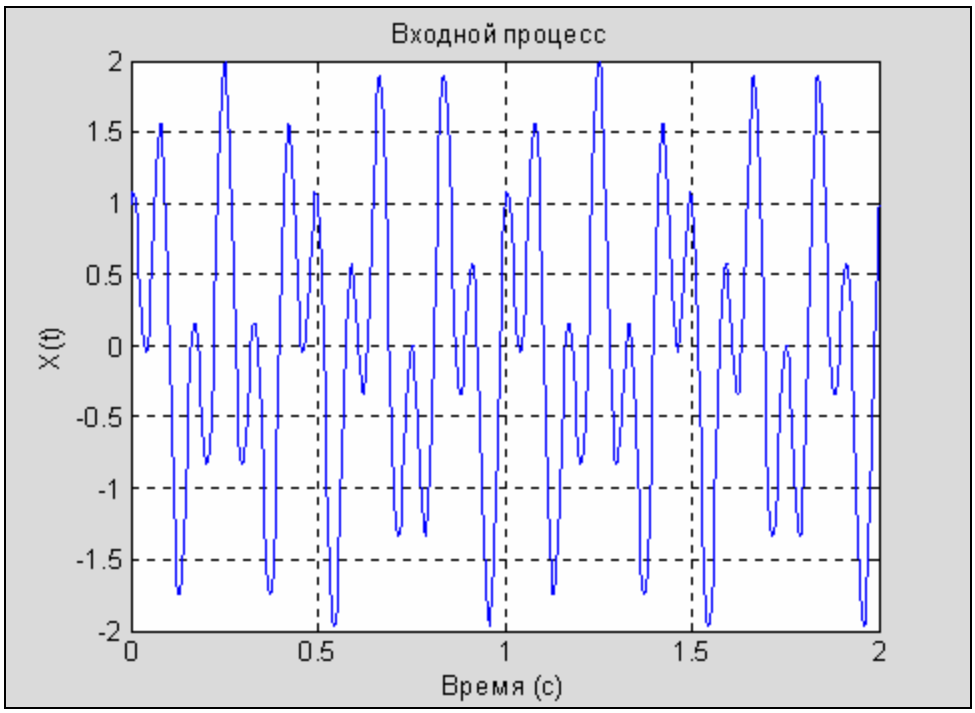


Рис. 1.21. Бигармонический процесс

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

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

Результат отображен на рис. 1.22.

Рис. 1.22. Модуль Фурье-изображения

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. Нетрудно убедиться, что воспроизведенный процесс совпадает с исходным.

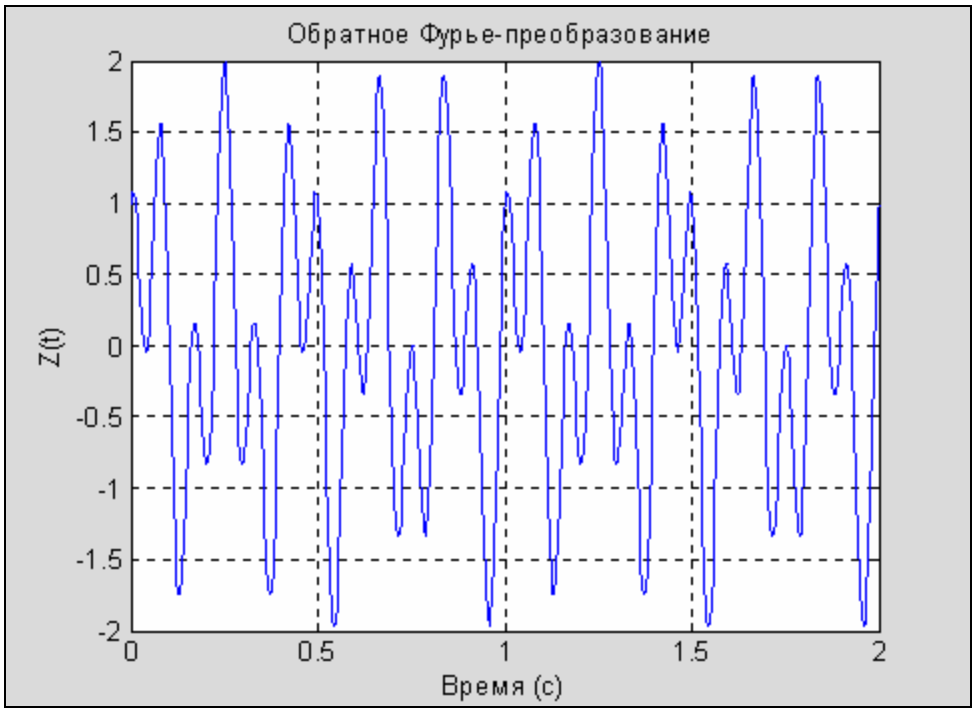


Рис. 1.23. Результат обратного преобразования Фурье

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 или частотной области, 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:

$$Df = 1/T, \tag{1.4}$$

а диапазон изменения частоты - формулой

$$F = 1/h; \tag{1.5}$$

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

$$Df = 0.5; \qquad F = 1000;$$

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 до *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 вышесказанного:

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

и выведем график 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))')
```

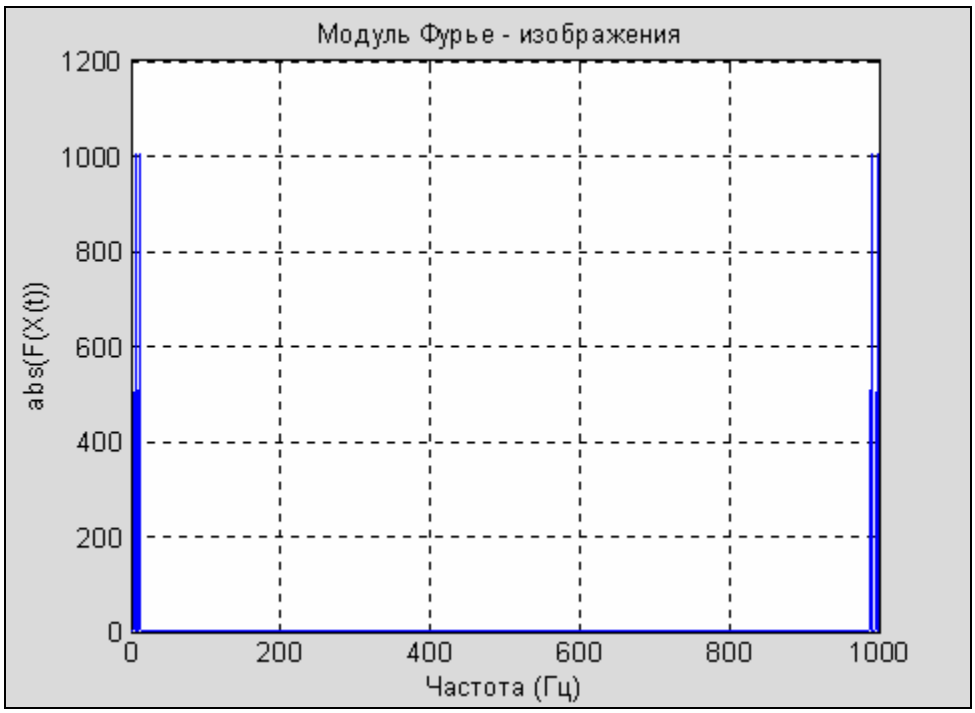


Рис. 1.24. Модуль Фурье-изображения в частотной области

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.**

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:

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

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

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 (рис. 1.25):

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

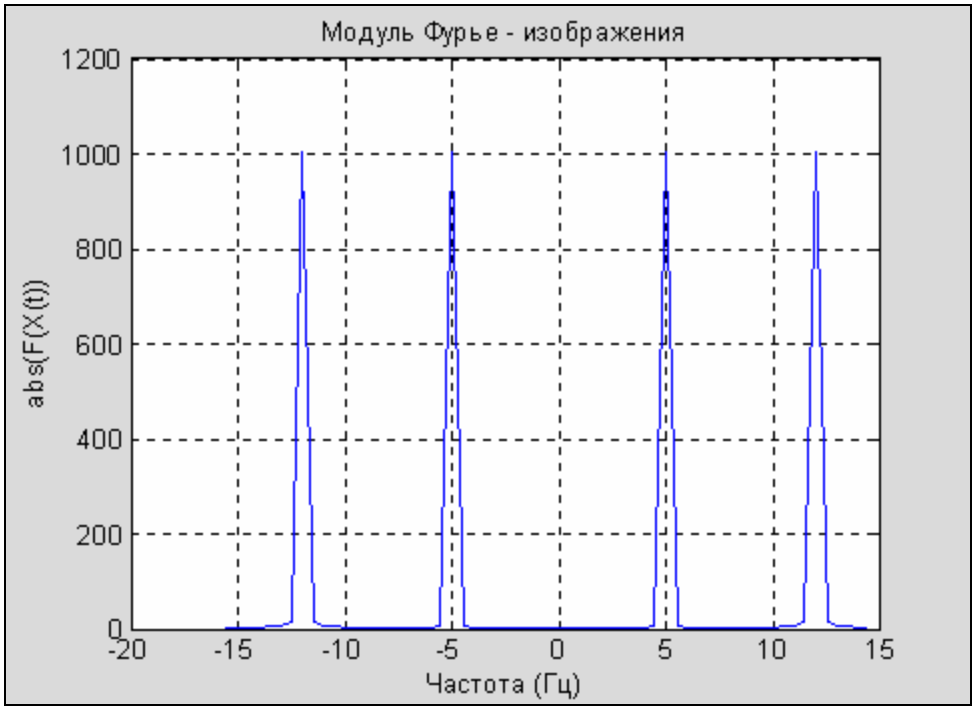


Рис. 1.25. Преобразование Фурье-изображения функцией fftshift

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',14,'Color','white'),
```

```
title('Модуль комплексного спектра');
xlabel('Частота (Гц)');ylabel('abs(F(X(t)) / N')
```

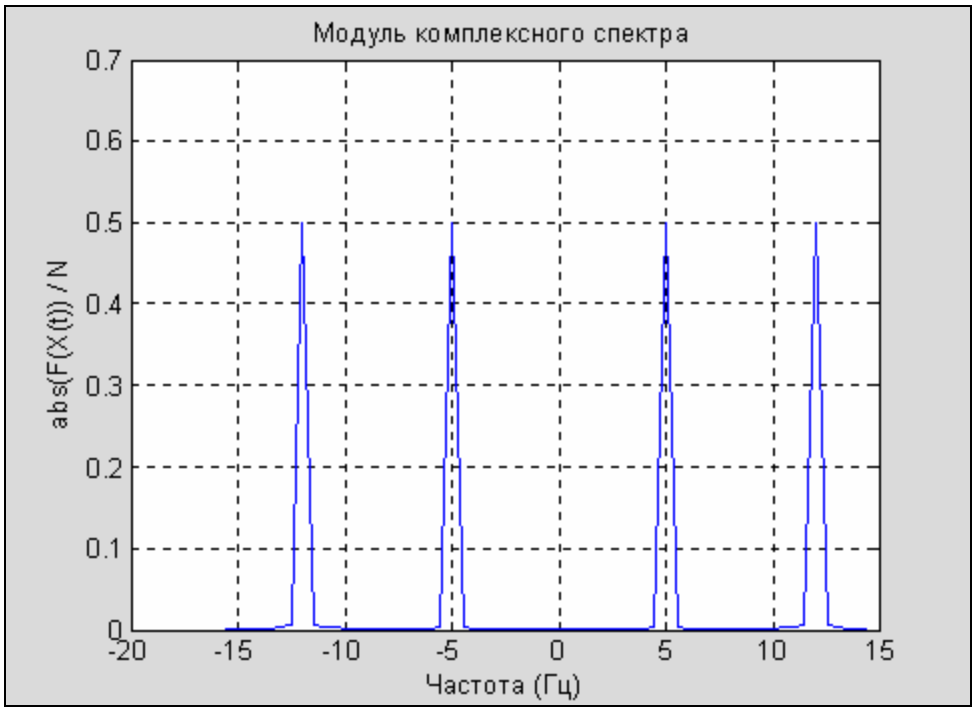


Рис. 1.26. Комплексный амплитудный спектр сигнала

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.

1.5. Построение простейших графиков

Вывод результатов вычислений в наглядной графической форме является одной из важнейших процедур в инженерной и научной практике. MatLAB предоставляет для этого широкие возможности. Ознакомимся с важнейшими из средств построения графиков, предусмотренными системой.

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 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).

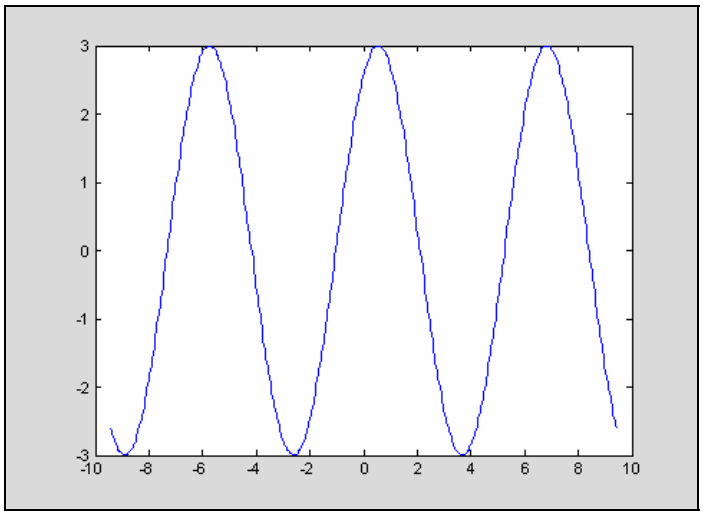


Рис. 1.27. График функции с указанием аргумента

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 следующие недостатки:

- 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.

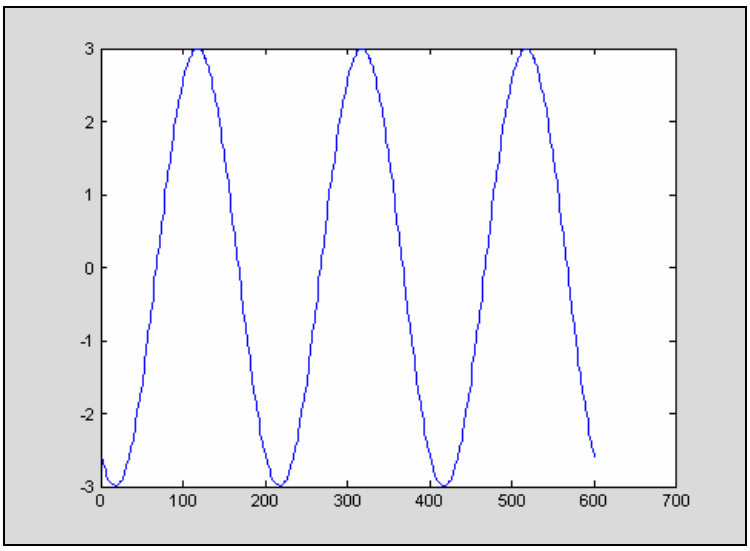


Рис. 1.28. График функции без указания аргумента

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.

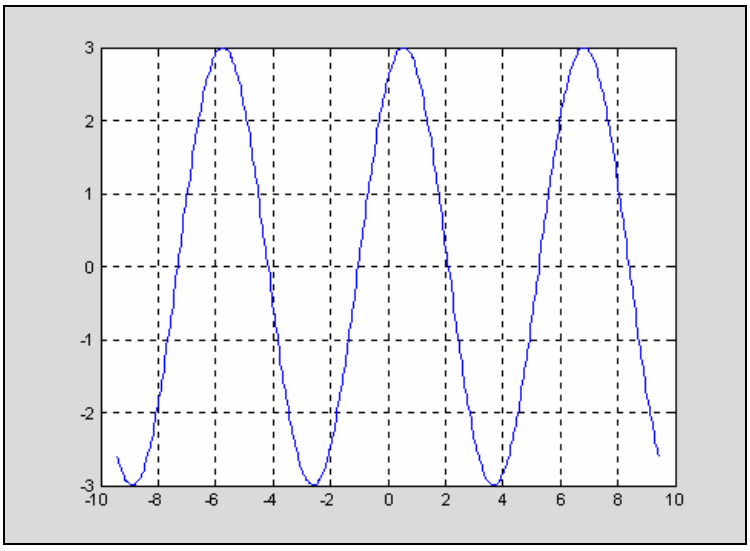


Рис. 1.29. Результат применения функции grid

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');
```

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

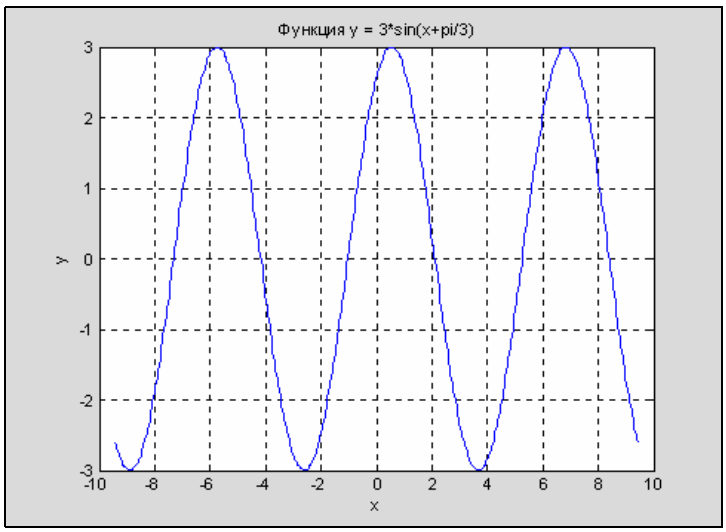


Рис. 1.30. Результат применения функций title, xlabel и ylabel

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

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

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

```
t = 0:0.1:50; x = 4*exp(-0.05*t).*sin(t);
y = 0.2*exp(-0.1*t).*sin(2*t);
plot(x,y),  grid, set(gcf,'color','white')
title('Параметрическая функция  x=4*exp(-0.05t)*sin(t);…
y= 0.2*exp(-0.1t)*sin(2t) ')
```

получим график рис. 1.31.

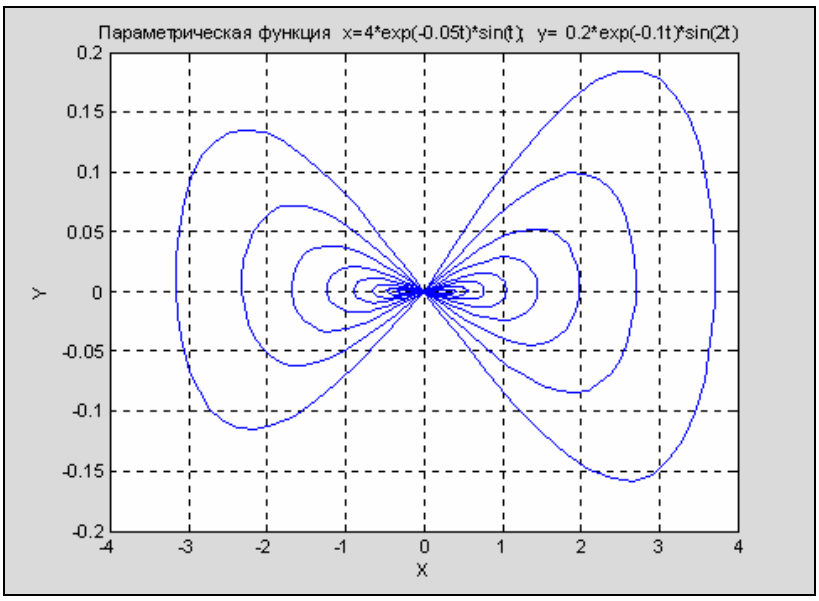


Рис. 1.31. График параметрически заданной функции

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.

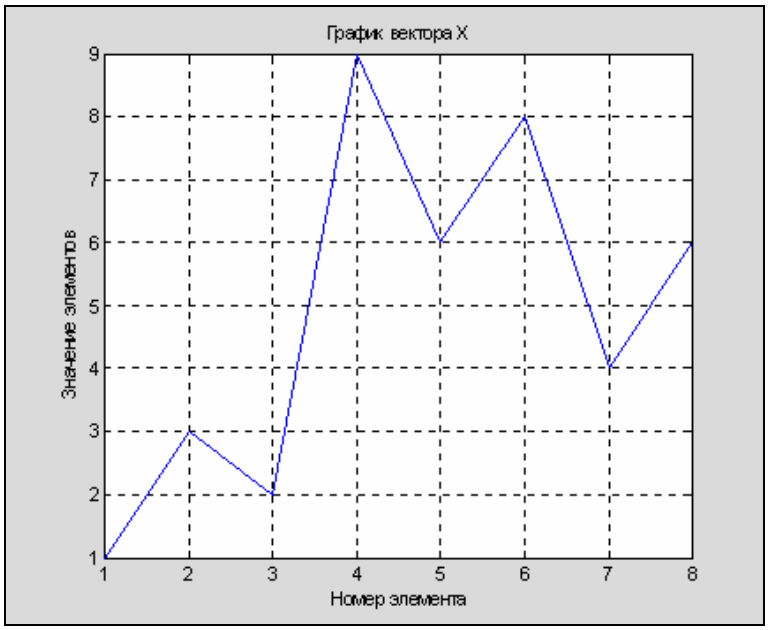


Рис. 1.32. График вектора

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')
```

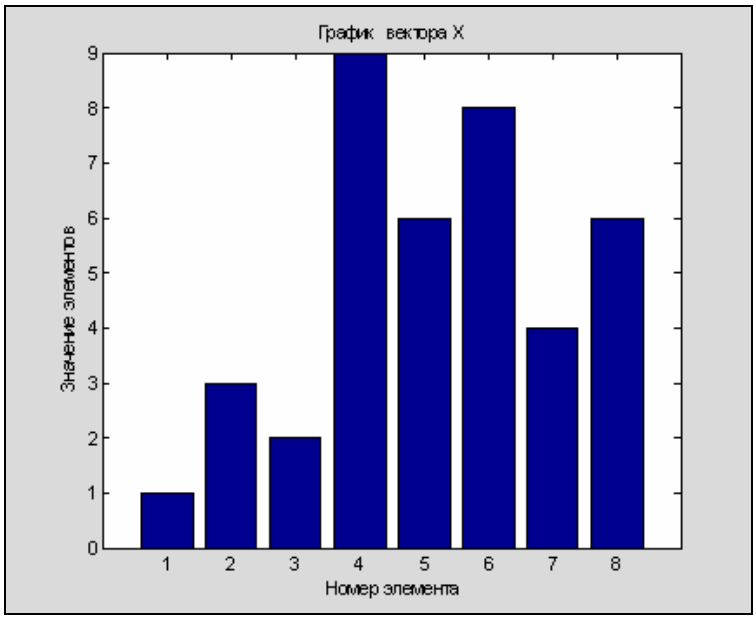


Рис 1.33. Результат применения функции bar

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,'FontSize',14),
title('График вектора X')
ylabel('Значение элементов'),  xlabel(' Номер элемента')
```

На рис. 1.34 изображен полученный при этом график.

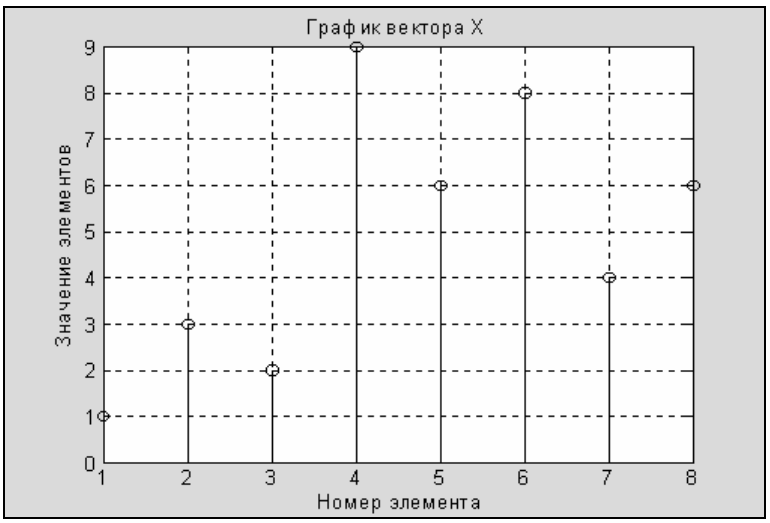


Рис. 1.34. График, полученный функцией stem

Другой пример - построение графика функции в виде столбцовой диаграммы (рис. 1.35):

```
x = - 2.9 : 0.2 : 2.9;     y=exp(-x.*x)
bar(x, y), set(gca,'FontSize',14)
title('Столбцовая  диаграмма  функции  y = exp(-x^2)')
xlabel (' Аргумент  x'),  ylabel (' Значение функции  y')
```

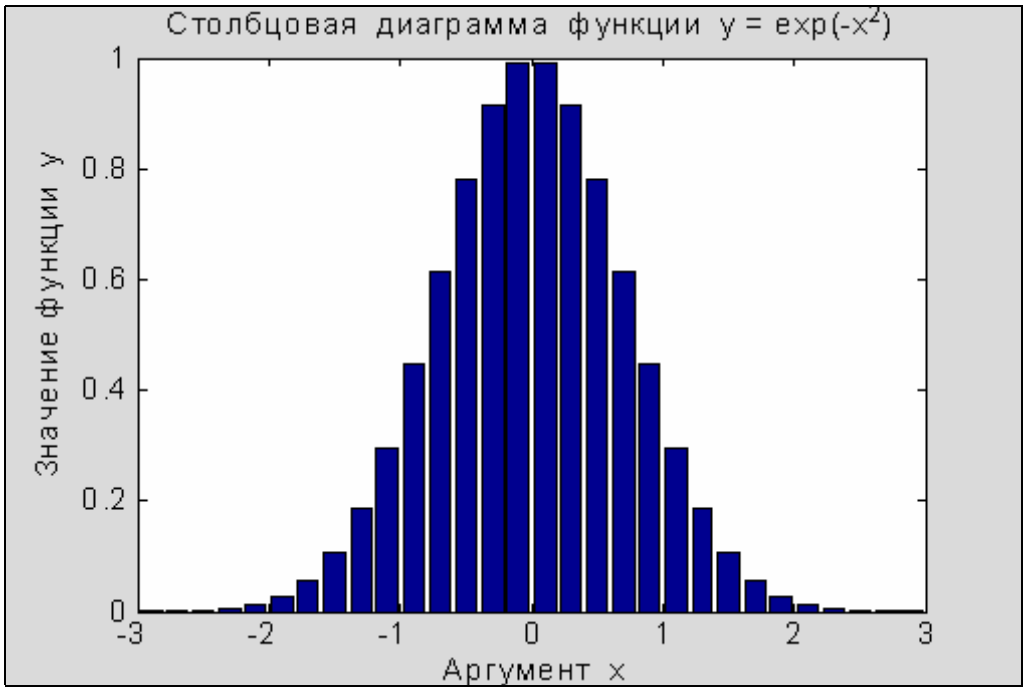


Рис. 1.35. Столбцовая диаграмма функции

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 изменения 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. 1.36. Гистограмма функции randn

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 фазо-частотной 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 изменения 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. Аналогично вычисляется комплекснозначный вектор "zn" знаменателя ЧПФ.

Вектор значений АЧХ (амплитудно-частотной характеристики) можно найти, рассчитывая модули векторов числителя и знаменателя ЧПФ и поэлементно деля полученные векторы. Чтобы найти вектор значений ФЧХ (фазо-частотной 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**.

В целом последовательность действий может быть такой:

```
P1 = [1 4];  P2 = [1 4 100]; OM = logspace(-2,2,40); p=i*OM;
ch = polyval(P1,p); zn = polyval(P2,p);
ACH = abs(ch)./abs(zn); FCH = angle(ch./zn)*180/pi;
loglog(OM,ACH);grid;       set(gca,'FontSize',14)
title('График Амплитудно-Частотной Характеристики')
xlabel('Частота   (рад/с)');     ylabel('Отношение  амплитуд')
figure, 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.

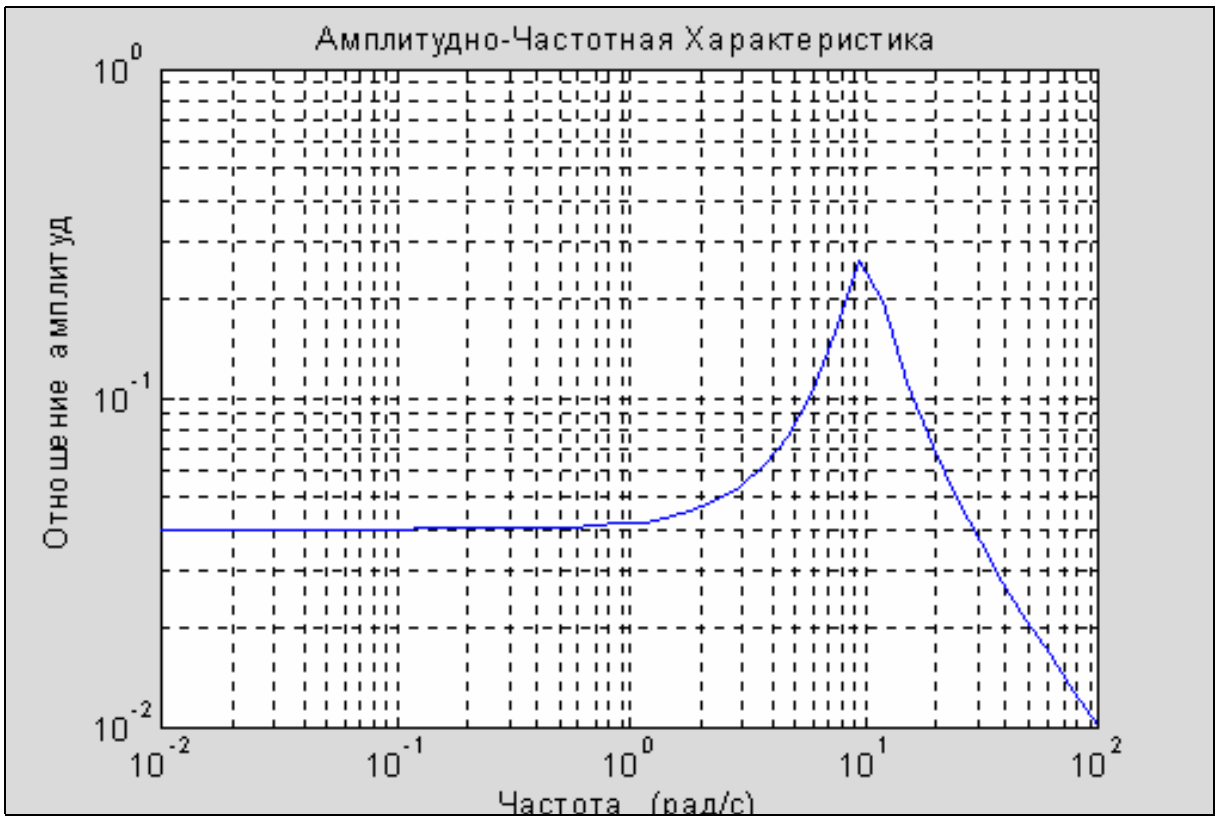


Рис. 1.37. График амплитудно-частотной характеристики

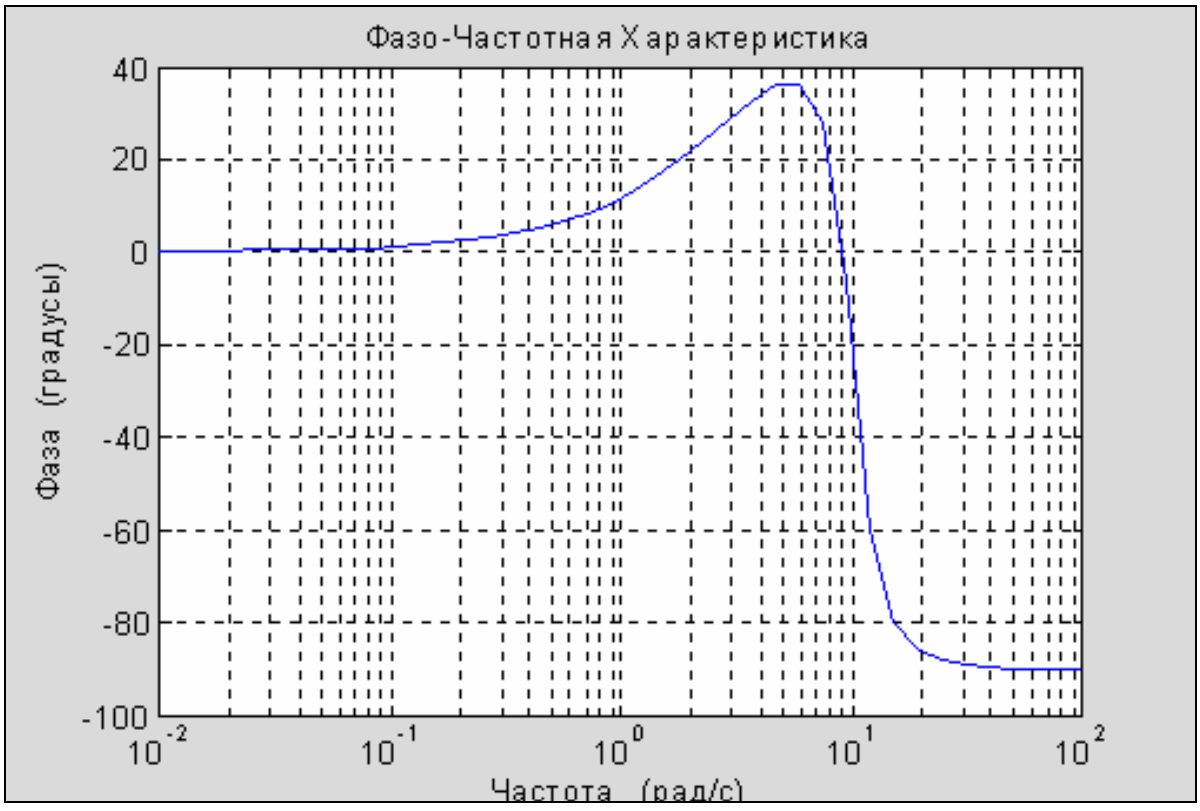


Рис. 1.38. График фазочастотной характеристики

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 обеим осям графика.

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

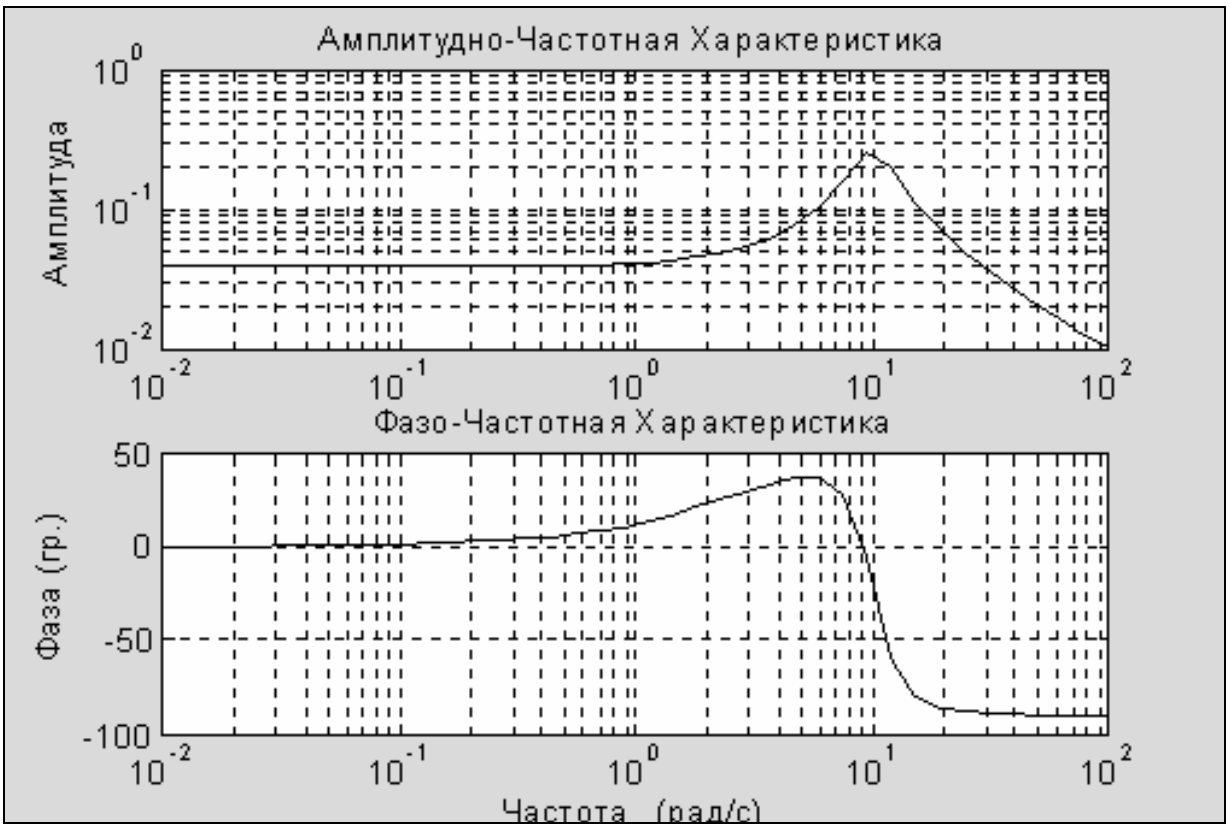


Рис. 1.39. Использование функции subplot

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(' А Ч Х')
gtext(' Ф 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. Выберите **`Файл`** ► **`Печать`**. Подготовьте принтер к работе и нажмите OK в окне печати, - принтер распечатает содержимое графического окна на отдельном листе бумаги.

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 работе MatLAB и в режиме калькулятора.

В языке MatLAB отсутствует оператор безусловного перехода и, в соответствии с этим, нет понятия 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 оператором **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,** с значениями, указанными в строках со словом **case.** Соответствующая группа операторов **case** выполняется, если значение выражения совпадает с 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)     '];
» disp(a)
» for i = 1:20
      x = i/5;
      si = sin(x);
      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.

1.7. Вопросы для самопроверки

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?

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

11. Какие функции MatLAB позволяют преобразовывать вектор поэлементно?

12. При помощи каких средств в MatLAB осуществляются основные операции с векторами?

13. Как вводятся матрицы в системе MatLAB?

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

15. Какие функции поэлементного преобразования матрицы есть в MatLAB?

16. Как осуществляются в MatLAB обычные матричные операции?

17. Как решить в MatLAB систему линейных алгебраических уравнений?

18. Какой объект в MatLAB называется полиномом?

19. Как в MatLAB осуществляется перемножение и деление полиномов?

20. При помощи каких функций можно найти корни заданного полинома, значение полинома по известному значению аргумента?

21. Какие функции позволяют найти производную от полинома?

22. Как найти характеристический полином матрицы?

23. Какие функции MatLAB осуществляют вывод графиков на экран?
24. Какими функциями обеспечивается снабжение графика координатными линиями и надписями?
25. Как вывести график в виде столбцовой диаграммы?
26. Как построить гистограмму?
27. Можно ли построить несколько графиков в одной системе координат и в одном графическом окне?
28. Как вывести несколько отдельных графиков в разных графических окнах?
29. Как построить несколько отдельных графиков в одном графическом окне в разных графических полях?
30. Какие средства управления ходом вычислительного процесса предусмотрены в языке MatLAB?
31. Как можно организовать вычисления по циклу в языке MatLAB?
32. Как организовать вывод таблицы результатов вычислений в командное окно MatLAB?

Урок 2. Программирование в среде MatLAB

Функции функций

Создание М-файлов

Создание простейших файлов-функций (процедур)

Создание Script-файлов

Графическое оформление результатов

Создание функций от функций

Пример создания сложной программы

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 функции, необходимо, чтобы пользователь предварительно создал М-файл, в котором вычислялось бы значение нужной ("внутренней") 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:

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

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

Обращение к процедурам численного интегрирования ОДУ имеет вид:

[t, y] = ode23 ('<имя_функции>', tspan, y0, options)

[t, y] = ode45 ('<имя_функции>', tspan, y0, options),

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 соответствует 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 $1 \cdot 10^{-3}$, абсолютная (по любой из переменных состояния) - $1 \cdot 10^{-6}$. 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 нулей функции осуществляется такими функциями MatLAB:

fmin	отыскание минимума функции одного аргумента;
fmins	отыскание минимума функции нескольких аргументов;
fzero	отыскание нулей 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 обозначено:

- 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 более чем в 20 раз. Последние два числа (10 и 20) могут быть изменены пользователем, для этого при обращении следует добавить эти новые значения в заголовок процедуры в указанном порядке.

Если обратиться к этой процедуре так:

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

то график указанной 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

Теперь рассмотрим приемы и особенности написания собственных программ и процедур, работающих в среде системы MatLAB.

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 Предпочтения меню Файл командного окна MatLAB. Например, это может быть редактор Notepade среды Windows. Окно предварительно установленного редактора появляется на экране, если перед этим выбрано Файл ►Новый ►М-файл команды или выбрано название одного из существующих М-файлов при вызове Файл ►Открыть 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 (оно автоматически устанавливается при сохранении файла на диске), 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 файлы-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 файлы-функции имеют первую строку вида

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

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

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

В файлах-функциях все имена переменных внутри файла, а также 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 отдельные фрагменты, 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) на языке MatLAB.

Обычно каждый оператор записывается в отдельной строке текста программы. Признаком конца оператора является символ (он не появляется в окне) возврата каретки и перехода на следующую строку, который вводится в программу при нажатии клавиши `Enter`, т. е. при переходе на следующую строку.

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

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

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

Строка программы, начинающаяся с символа « % », не выполняется. Эта строка воспринимается системой MatLAB как комментарий. Таким образом, для ввода комментария в любое место текста программы достаточно начать соответствующую строку с символа « % ».

Строки комментария, предшествующие первому выполняемому оператору программы, т. е. такому, который не является комментарием, воспринимаются системой MatLAB как описание программы. Именно эти строки выводятся в командное окно, если в нем набрана команда

help <имя файла>

В программах на языке MatLAB отсутствует символ окончания текста программы.

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».

Имена переменных могут содержать лишь буквы латинского алфавита или цифры и должны начинаться с буквы. Общее число символов в имени может достигать 30. В именах переменных могут использоваться как прописные, так и строчные буквы. Особенностью языка MatLAB является то, что строчные и прописные буквы в именах различаются системой. Например, символы «а» и «А» могут использоваться в одной программе для обозначения разных величин.

2.3. Создание простейших файлов-функций (процедур)

Создание собственных файлов-функций является неизбежным этапом написания собственных программ, а также использования стандартных функций от функций для решения ваших задач.

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), то файл-функция представляет собой уже более сложный объект, который в программировании обычно называется или 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 (М-файла).

В качестве примера рассмотрим составление М-файла для функции

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

Для этого следует выбрать в меню командного окна Файл ▶Новый ▶М-файл. На экране появится окно текстового редактора. В нем нужно набрать такой текст:

```
function y = F1(x,d)
%          Процедура,  вычисляющая 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 М-файл создан. Теперь можно пользоваться этой функцией при расчетах. Так, если ввести команду

```
» y = F1(1, 0.1)
```

то получим результат

```
y =  4. 1421e-004.
```

Следует заметить, что 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 к той же процедуре

```
» 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, вычисляющая 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 графики этих функций:

```
» fplot('myfun', [-20 20], 50, 2), grid
» set(gca,'FontSize',12); title('График  функции  "MYFUN"')
```

Результат изображен на рис. 2.1.

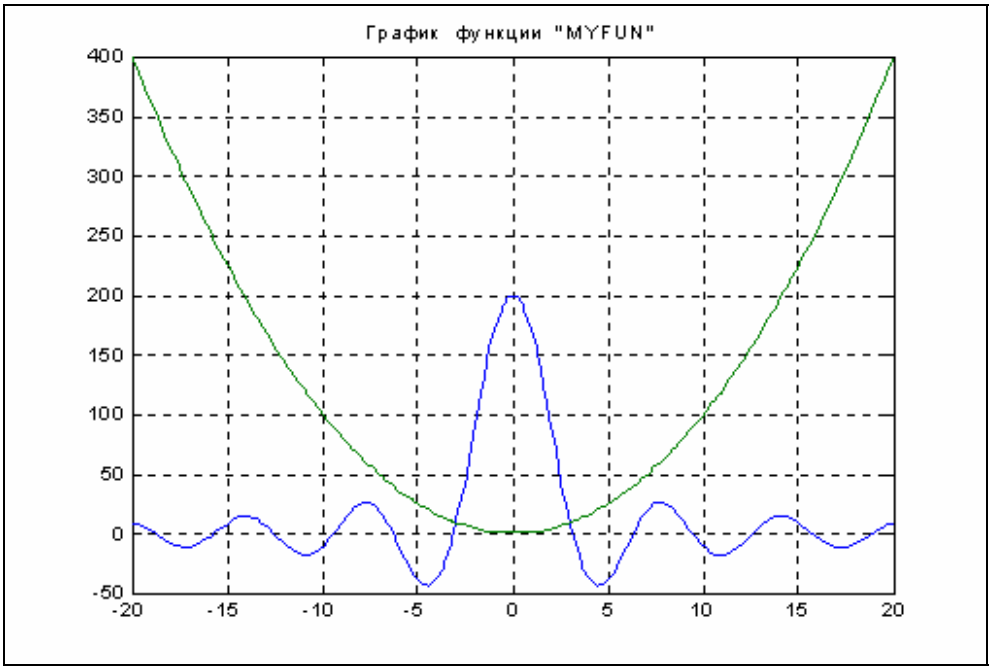


Рис. 2.1. Результат применения функции fplot

Третий пример - создание файла-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, 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 такому шаблону:

```
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>
                  < П у с т а я   с т р о к а >
 % Автор : <Указывается автор процедуры, дата создания
 %      процедуры и организация, в которой создана программа>
< Т е к с т   и с п о л н я е м о й   ч а с т и   п р о ц е д у р ы >
```

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

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

2.4. Создание Script-файлов

2.4.1. Основные особенности Script-файлов

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

- Script-файлы являются независимо (самостоятельно) исполняемыми блоками операторов и команд;
- все используемые переменные образуют так называемое рабочее пространство, которое является общим для всех исполняемых Script-файлов; из этого следует, что при выполнении нескольких Script-файлов имена переменных в них должны быть согласованы, так как одно имя означает в каждом из них один и тот же объект вычислений;
- 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, обходя заголовок файл-функции, использовать значения, которые приобретают переменные в Script-файле (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 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).

Примечание. 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 воспользоваться функцией **num2str**, которая осуществляет такое преобразование. Запись

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

превратит числовое значение переменной x в текстовое представление. При этом *форма представления определяется установленным форматом выведения чисел на экран* (Числовой формат), например:

```
» x = -9. 30876e-15

x = -9. 3088e-015

» y = num2str(x)

y = -9. 309e-015
```

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

```
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.

Другое средство достижения того же результата - использование функции **sprintf**. Обращаться к ней следует по форме:

```
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 окна 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.

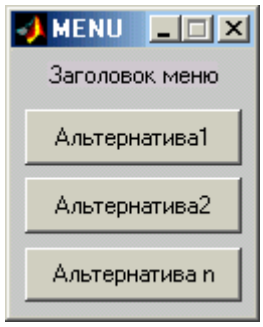


Рис. 2.2. Структура и вид окна меню пользователя в MatLAB

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**.

2.4.3. Организация повторения действий

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

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

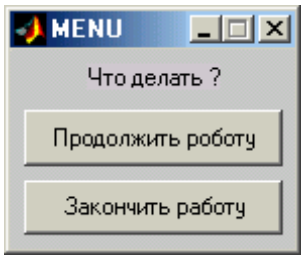


Рис. 2.3. Окно меню повторения работы с программой

2.4.4. Изменение 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('Что изменить?','Параметр1','Параметр2','Параметр3',
'Параметр4','Параметр5','Ничего не менять'),
```

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

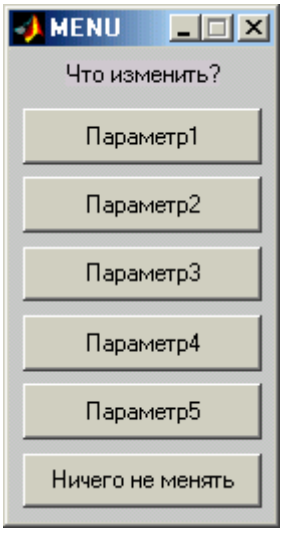


Рис. 2.4. Вариант оформления меню

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( '   Что  изменить?   ', ...
sprintf (' Параметр1   x1 = %g', x1),...
sprintf (' Параметр2   x2 = %g', x2),...
sprintf (' Параметр3   x3 = %g', x3),...
sprintf (' Параметр4   x4 =  %g', x4),...
sprintf (' Параметр5   x5 =  %g', x5),...
 '  Ничего не изменять  ')
```

Результат приведен на рис. 2.5.

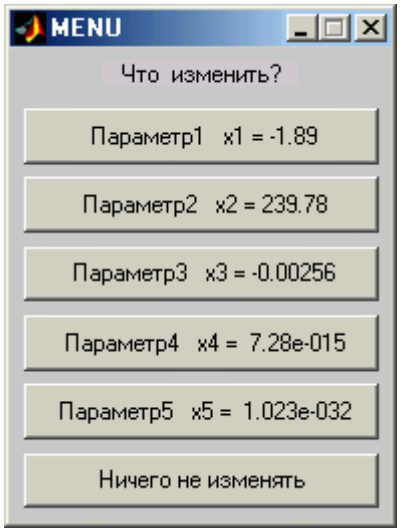


Рис. 2.5. Вариант рационального оформления меню

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
```

Для того, чтобы можно было проконтролировать правильность ввода новых значений, обеспечить возможность их корректировки и последовательного изменения всех желаемых параметров, нужно, чтобы после ввода нового значения любого параметра на экране вновь возникало то же меню, но уже со скорректированными значениями. При этом завершение работы с меню должно наступить только при условии выбора последней альтернативы меню «Ничего не изменять», соответствующей значению k, равному 6.

Поэтому предыдущие операторы следует заключить в цикл:

```
k=1;
while k<6
 k = menu( ' Что изменить ?   ', ...
             sprintf (' Параметр1 x1 = %g', x1),...
             sprintf (' Параметр2 x2 = %g', x2),...
             sprintf (' Параметр3 x3 = %g', x3),...
             sprintf (' Параметр4 x4 = %g', x4),...
             sprintf (' Параметр5 x5 = %g', x5),...
              ' Ничего не изменять ');
 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.

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.

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`.

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 значению), 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.  0., дата создания, организация>

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

2.5. Графическое оформление результатов

2.5.1. Общие требования к представлению графической информации

Вычислительная программа, создаваемая инженером-разработчиком, 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, полностью отвечают этим требованиям.

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

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

При этом нужно обратить внимание на следующее.

1. 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`**.

2. 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.

3. 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.

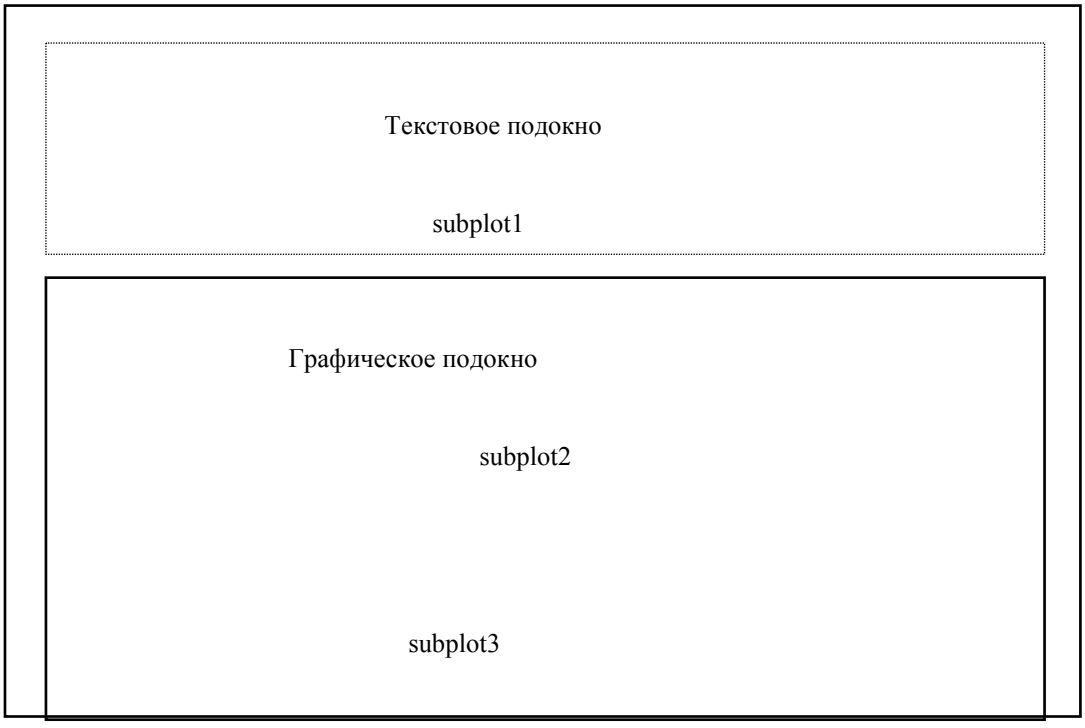


Рис. 2.6. Схема разбивки графического окна на подокна

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

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

Если требуется создать три отдельных поля графиков один под другим на 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])`**;

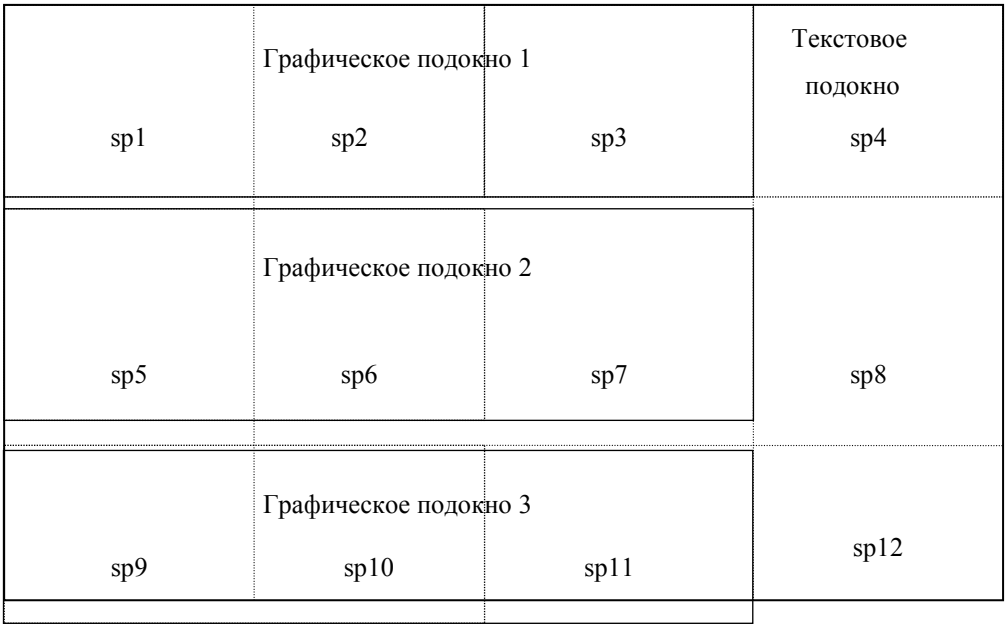


Рис. 2.7. Вторая схема разбивки графического окна

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').
```

Так, если эту команду ввести после предыдущих 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.

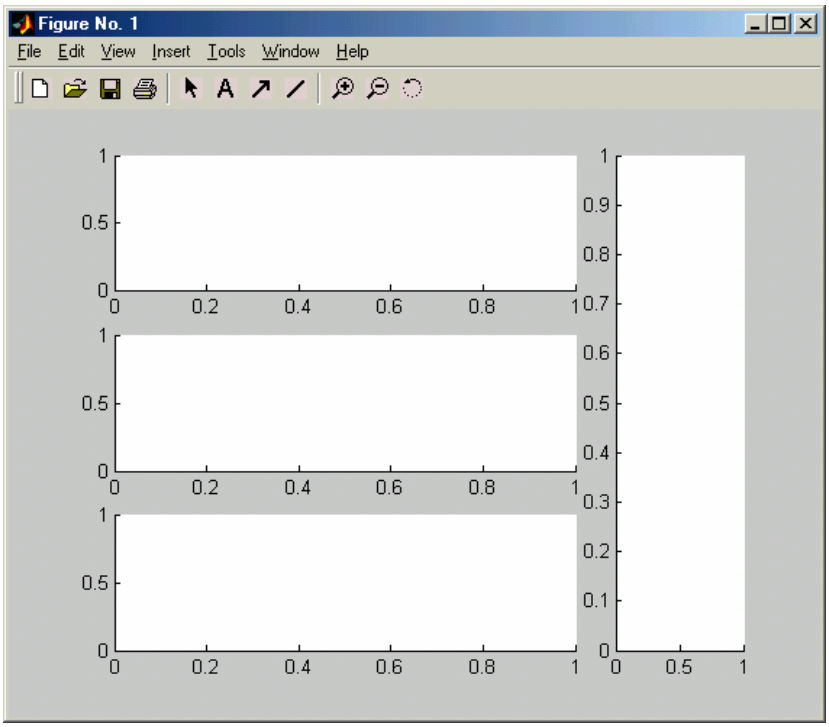


Рис. 2.8. Графическое окно после использования функции subplot

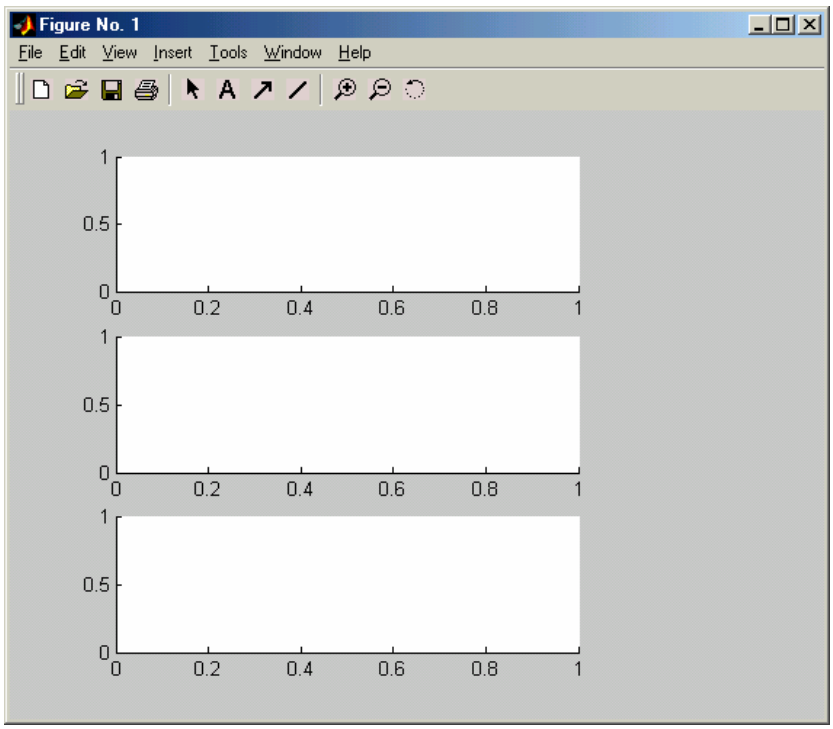


Рис. 2.9. Результат действия функции axis('off')

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(' H = %g ',D1(3)),'FontSize',10);
h1=text(-0.2,0.85,sprintf(' R = %g ',D1(4)),'FontSize',10);
h1=text(0.6,0.85,sprintf(' C = %g ',D1(5)),'FontSize',10);
h1=text(-0.2,0.8,sprintf('J1  = %g ',D1(1)),'FontSize',10);
h1=text(0.6,0.8,sprintf('J2 = %g ',D1(2)),'FontSize',10);
h1=text(0,0.75,'Внешних воздействий','FontSize',10);
h1=text(-0.2,0.7,sprintf('pst0 = %g ',D2(1)),'FontSize',10);
h1=text(0.6,0.7,sprintf(' tet0 = %g ',D2(2)),'FontSize',10);
h1=text(0.2,0.66,sprintf(' fit0 = %g ',D2(3)),'FontSize',10);
h1=text(-0.2,0.62,sprintf('psm = %g ',D2(4)),'FontSize',10);
h1=text(0.6,0.62,sprintf(' tem = %g ',D2(5)),'FontSize',10);
h1=text(0.2,0.58,sprintf(' fim = %g ',D2(6)),'FontSize',10);
h1=text(-0.2,0.54,sprintf('omps = %g ',D2(7)),'FontSize',10);
h1=text(0.6,0.54,sprintf(' omte = %g ',D2(8)),'FontSize',10);
h1=text(0.2,0.5,sprintf('omfi = %g ',D2(9)),'FontSize',10);
h1=text(-0.2,0.46,sprintf('eps = %g ',D2(10)),'FontSize',10);
h1=text(0.6,0.46,sprintf(' ete = %g ',D2(11)),'FontSize',10);
h1=text(0.2,0.42,sprintf('efi=%g ',D2(12)),'FontSize',10);
h1=text(0,0.35,'Интегрирования','FontSize',10,'FontUnderline','on');
h1=text(0,0.3,sprintf('h = %g ',D5(1)),'FontSize',10);
h1=text(0,0.25,sprintf('hpr = %g ',D5(2)),'FontSize',10);
h1=text(0,0.2,sprintf('t = %g ',D5(3)),'FontSize',10);
h1=text(0,0.15,sprintf('tfinal = %g ',D5(4)),'FontSize',10);
h1=text(-0.3,0.12,'-----------------------------------------------------
','FontSize',10);
tm=fix(clock); Tv=tm(4:6);
h1=text(-0.2,0.08,['Программа ' sprogram],'FontSize',10);
h1=text(-0.3,0.04,['Расчеты провел ' sname],'FontSize',10);
h1=text(-0.3,0,[sprintf(' %g :',Tv) ' ' date],'FontSize',10);
h1=text(-0.3,-0.04,'-----------------------------------------------------
','FontSize',10); h1=text(-0.3,-0.08,'Ukraine,   KPI,   cath. PSON','FontSize',10);
```

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

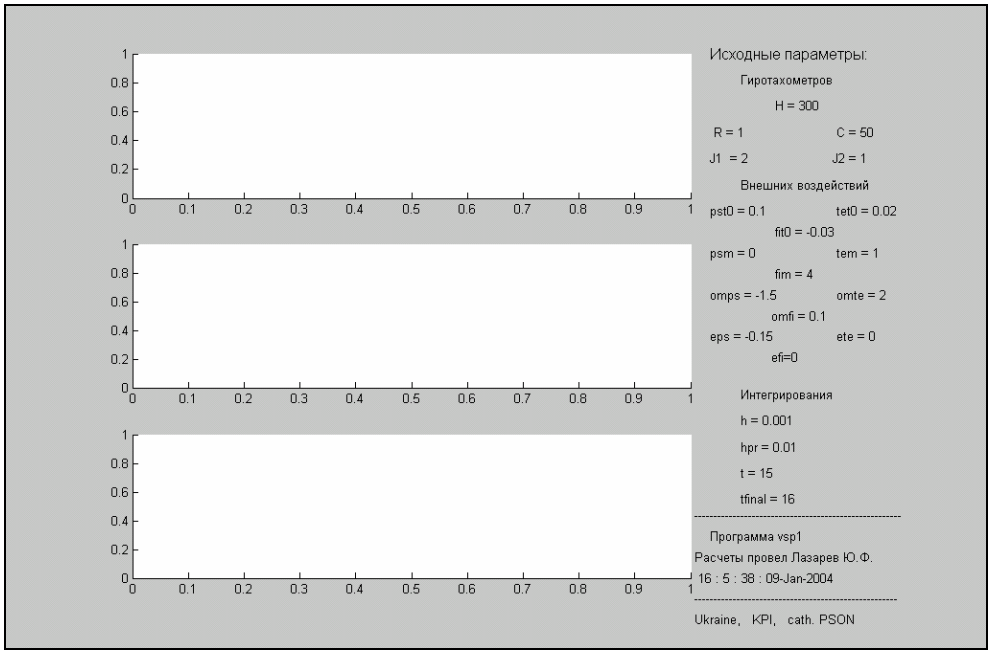


Рис. 2.10. Пример текстового оформления графического окна

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 определенного файла-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 указании 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 он не меняет 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 обращение к ней, не зная еще конкретного имени функции.

2.6.2. Примеры создания процедур от функций

Рассмотрим на примерах особенности создания собственных функций от функций.

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

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

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

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

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

yout = y + h*F(y,t).

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

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

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

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

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

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

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


  %           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 конкретизировано; 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 вычисления 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 = φ; 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 М-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 К с помощью служебного слова **global,** а потом определены все восемь 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 преобразуем исходное уравнение, записывая его в виде:

$$\varphi'' + \sin\varphi = S(\tau, \varphi, \varphi'),$$

где штрих - обозначение производной по безразмерному времени $\tau = \omega_0 \cdot t$,

$$\omega_0 = \sqrt{\frac{mgl}{J}} \ ,$$

а через $S(\tau, \varphi, \varphi')$ обозначена некоторая заданная функция безразмерного времени, угла поворота маятника и его безразмерной скорости

$$\varphi' = \frac{d\varphi}{d\tau} \ .$$

В рассматриваемом случае эта функция приобретает такой вид:

$$S(t, \varphi, \varphi') = -2 \cdot \zeta \cdot \varphi' -$$
$$-[n_{mx} \cdot \sin(\nu \cdot \tau + \varepsilon_x) \cdot \cos\varphi + n_{my} \cdot \sin(\nu \cdot \tau + \varepsilon_y) \cdot \sin\varphi],$$

причем безразмерные величины ζ и ν определяются выражениями:

$$\zeta = \frac{R}{2 \cdot \sqrt{mgl \cdot J}} \ ; \qquad \nu = \frac{\omega}{\omega_0} \ .$$

Такая безразмерная форма представления уравнений является предпочтительной, 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, действующих на Физический Маятник
  % Осуществляет расчет момента "m" сил
  % по формуле:
  %    m =-2*dz*y(2) - (nmx*sin(nu*t+ex)*cos(y(1)) +...
  %                              + nmy*sin(nu*t+ey)*sin(y(1)),
  % Коэффициенты передаются в процедуру через глобальный вектор
  %                   KM1=[dz,nmy,nmx,nu,ey,ex]

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
  %                вычисления правых частей ОДУ.
  %          Обращение: z = fun(Mpfun,t,y),
  %                где Zpfun = 'fun',
  %                Mpfun - строка с именем процедуры, к которой
  %                       обращается процедура fun;
  %                t - текущий момент времени
  %                y - вектор текущих значений переменных состояния
  %          z - вычисленные значения производных z(i) =dy(i)/dt.
  %    h - шаг интегрирования
  %    t - 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 как глобальную переменную. Тогда процедура правых частей может быть записана так:

```
function z = FM2(t,y);
  % Процедура правых частей уравнения Физического Маятника.
   % Осуществляет расчет вектора "z"
   % производных 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. Самостоятельная работа

2.3. 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 метода
1	$F=k_1$	$k_1=Z(t_m;y_m)$	Ейлера
2	$F=(k_1+k_2)/2$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h;y_m+hk_1)$	Модифицированный Ейлера
3	$F=Z(t_m+h/2;\ y_m++hk_1/2)$	$k_1=Z(t_m;y_m)$	
4	$F=(k_1+4k_2+k_3)/6$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/2;y_m+hk_1/2)$; $k_3=Z(t_m+h;y_m+h(2k_2-k_1))$	Хойне
5	$F=(k_1+3k_3)/4$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/3;y_m+hk_1/3)$; $k_3=Z(t_m+2h/3;y_m+2hk_2/3)$	
6	$F=(k_1+2k_2+2k_3+ +k_4)/6$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/2;y_m+hk_1/2)$; $k_3=Z(t_m+h/2;y_m+hk_2/2)$; $k_4=Z(t_m+h;y_m+hk_3)$	Рунге-Кутта
7	$F=(k_1+3k_2+3k_3+ +k_4)/8$	$k_1=Z(t_m;y_m)$; $k_2=Z(t_m+h/3;y_m+hk_1/3)$; $k_3=Z(t_m+2h/3;y_m+h(k_2- k_1/3))$; $k_4=Z(t_m+h;y_m+h(k_1- k_2+k_3))$	

Таблица 2.2. Многошаговые методы

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

2.7. Программа моделирования движения маятника

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.

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 предыдущем разделе.

Запись М-файла процедуры правых частей

Следующим шагом подготовки программы является написание и запись на диск текста процедуры вычисления правых частей полученной системы ОДУ 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 процедуры, 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, действующих на Физический Маятник
  % Осуществляет расчет момента "m" сил
  % по формуле:
  % m =-2*dz*y(2) - (nmx*sin(nu*t+ex)*cos(y(1)) +...
  % + nmy*sin(nu*t+ey)*sin(y(1)),
  % Коэффициенты передаются в процедуру через
  % глобальный вектор KM1=[dz,nmy,nmx,nu,ey,ex]

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 предыдущего раздела :

Файл FizMayatn2 . m

```
  % FizMayatn2
  % Управляющая программа исследования движения ФМ
  % 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 лишь вызывает 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 диалогового ввода-изменения значений параметров физического маятника, движения основания и параметров численного интегрирования в соответствия со схемой, описанной в предыдущем разделе.

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 малых собственных        колебаний)','FontSize',12);
ylabel('Угол в градусах','FontSize',12);
subplot(2,4,1:2);
plot(y(:,1)*180/pi,y(:,2));grid;
title('Фазовый портрет','FontSize',14);
xlabel('Угол в градусах','FontSize',12);
ylabel('Угл. скорость (б/р)','FontSize',12);
 % Вывод текстовой информации в графическое окно
subplot(2,4,3:4);         axis('off');
h1=text(0,1.1,'Движение физического маятника',
FontSize', 14, 'FontWeight', 'Bold');
h1=text(0.4, 1,'в соответствии с 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,'при следующих значениях 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 град.',KM1(5)*180/pi),'FontSize',12);
h1=text(0.7,0.4,sprintf('ex = %g град.',…
                      KM1(6)*180/pi),'FontSize',12);
h1=text(0.45,0.3,sprintf('nu = %g',KM1(4)),'FontSize',12);
h1=text(0,0.2,'и начальных условий:','FontSize',12);
h1=text(0,0.1,[sprintf('fi(0) = %g град.',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 Заключается в запуске главного М-файла FizMayatn2, проверке правильности функционирования всех частей программы, внесения корректив в тексты используемых М-файлов до тех пор, пока все запрограммированные действия не будут удовлетворять заданным требованиям. Сюда же входят и действия по проверке «адекватности», т. е. соответствия получаемых программой результатов отдельным априорно известным случаям поведения исследуемой системы. Очевидно, для такой проверки нужно подобрать несколько совокупностей значений параметров системы, при которых поведение системы является известным из предыдущих теоретических или экспериментальных исследований. Если полученные программой результаты полностью согласуются с известными, программа считается адекватной принятой математической модели.

В приведенном тексте программы «вшитые» начальные значения параметров отвечают свободному движению маятника при отсутствии 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.

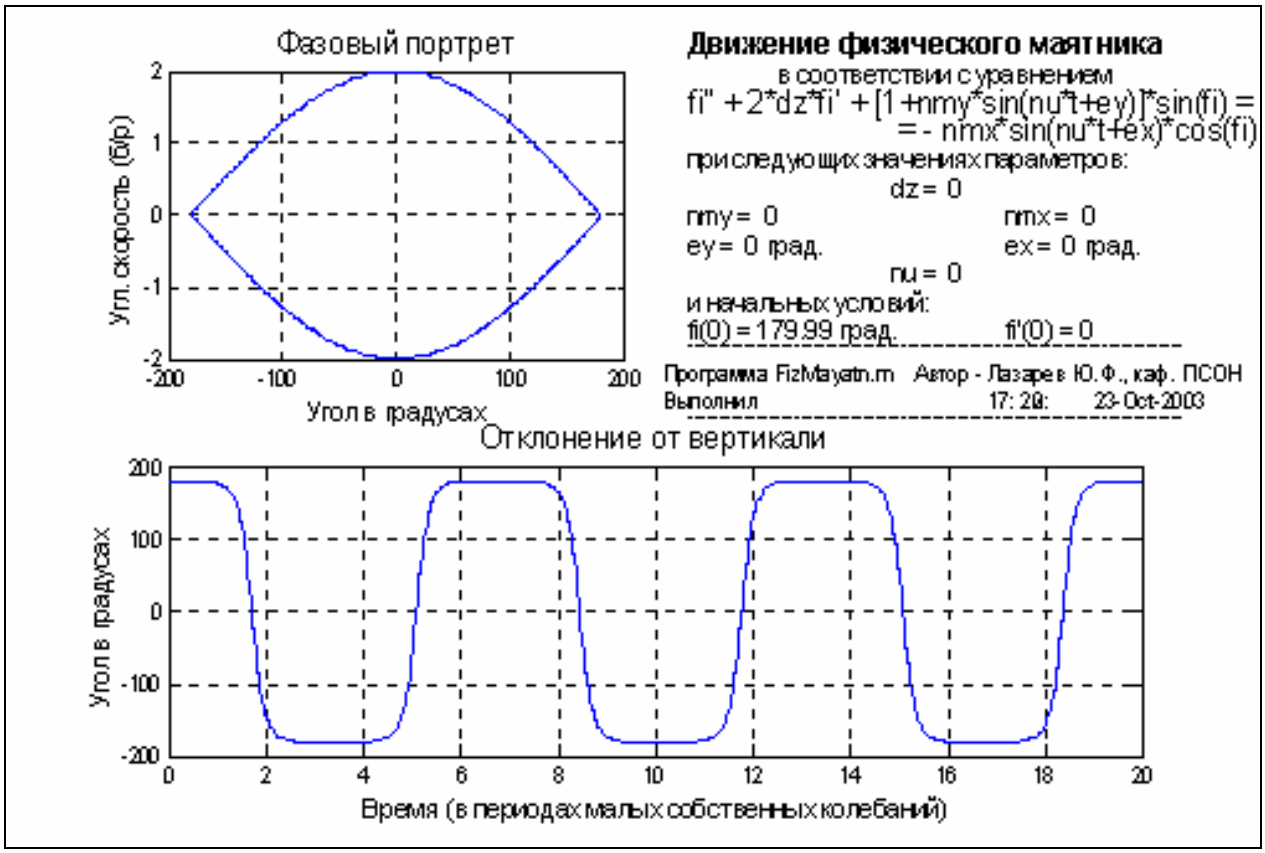


Рис. 2.11. Свободные колебания маятника с большим размахом

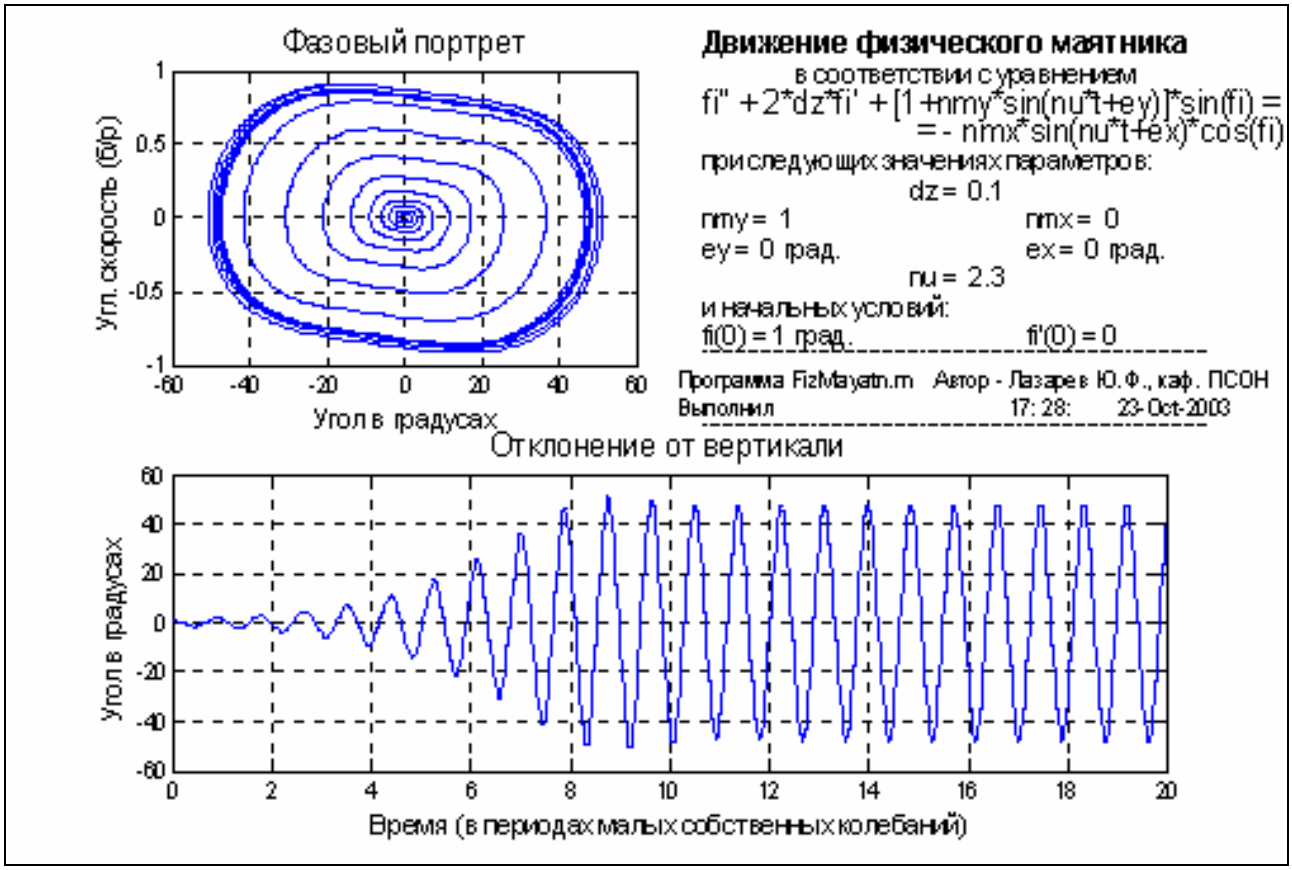


Рис. 2.12. Параметрические колебания маятника при вертикальной вибрации

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.

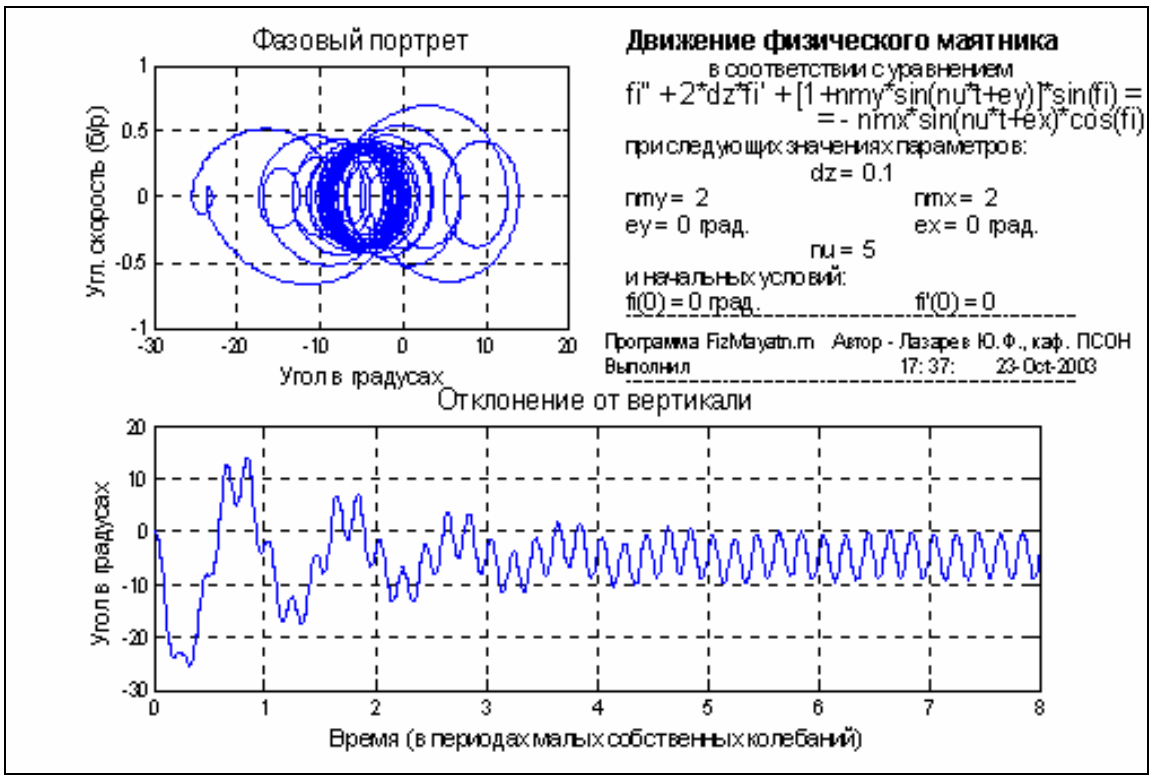


Рис. 2.13. Отклонение среднего положения маятника от вертикали

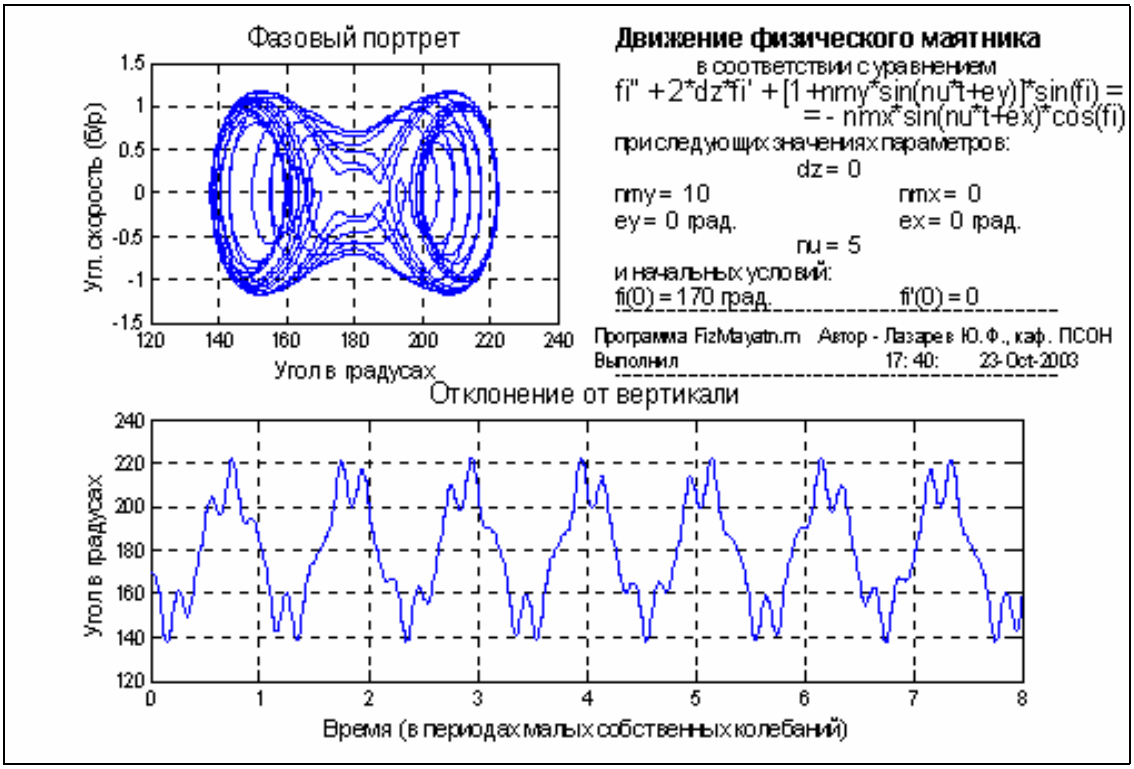


Рис. 2.14. Устойчивые колебания маятника относительного верхнего положения

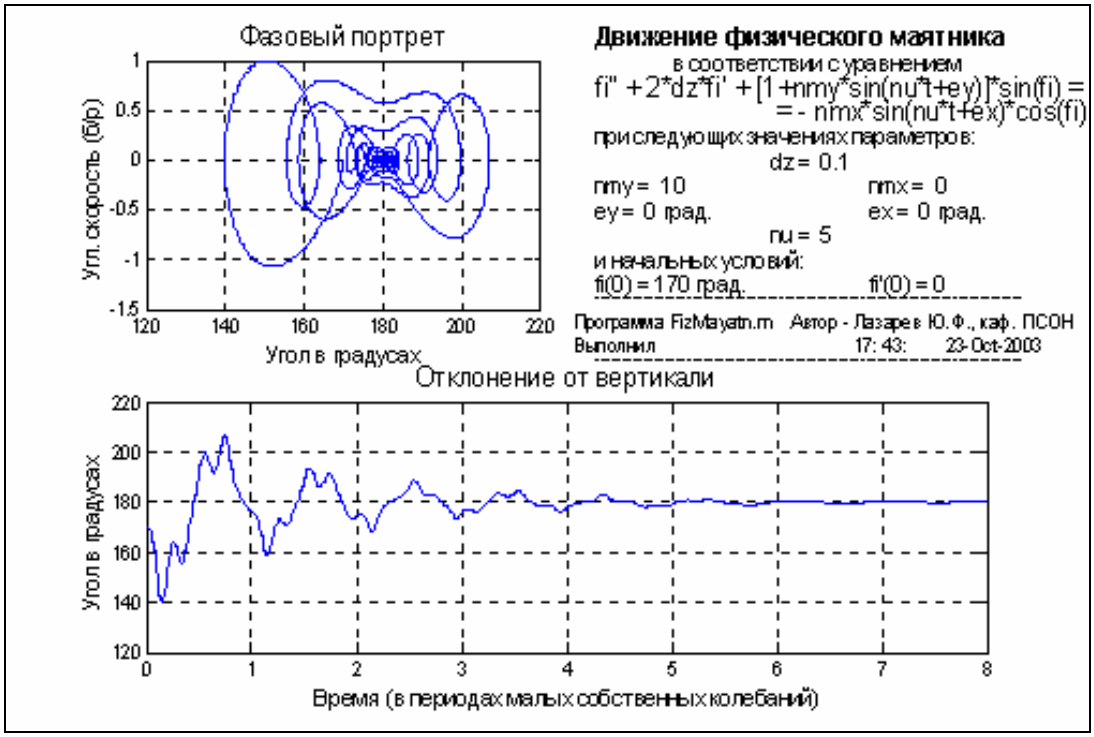


Рис. 2.15. Затухание колебаний маятника относительно верхнего положения

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 рисунками, это отклонение достигает величины свыше 40 градусов при принятых значениях параметров вибрации, причем направление отклонения зависит от начальных условий движения маятника.

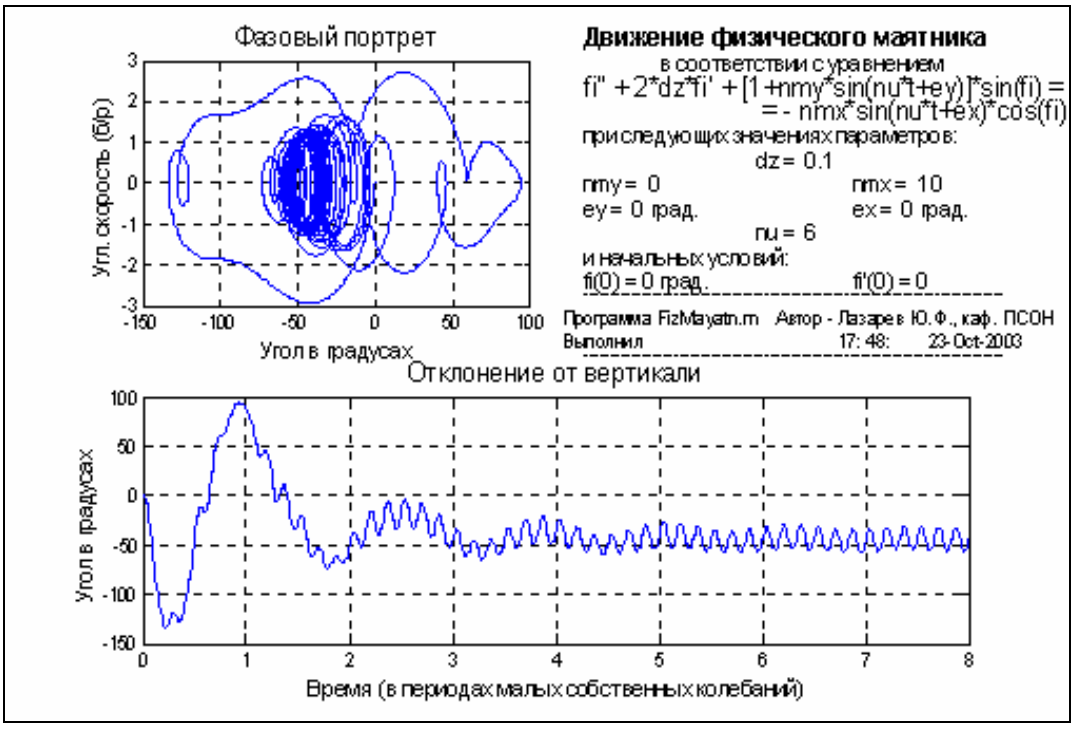


Рис. 2.16. Отклонение среднего положения маятника от вертикали при горизонтальной вибрации

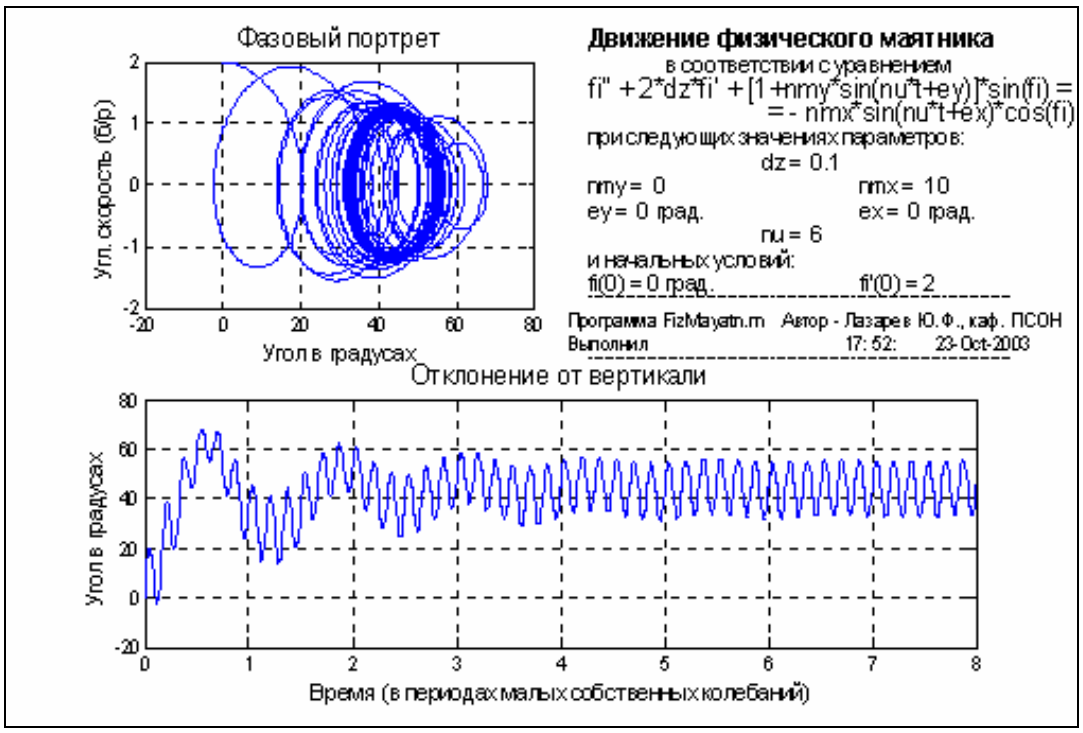


Рис. 2.17. Второе устойчивое положение равновесия колебаний маятника при горизонтальной вибрации

2.8. Вопросы для самопроверки

1. Что понимается в MATLAB под функциями функций?

2. Какие стандартные (встроенные) функции функций предусмотрены в MATLAB?

3. Для чего создаются программы в среде MatLAB?

4. Какие виды файлов программ существуют в MATLAB?

5. Чем отличаются Script-файлы от файлов-функций? Какова сфера применения каждого из этих видов файлов?

6. Каковы основные правила написания текстов программ на языке MATLAB?

7. Как создать М-файл процедуры или функции?

8. Какие процедуры обеспечения диалогового режима ввода-вывода информации предусмотрены в MATLAB?

9. Какими средствами можно обеспечить вывод текстовой информации в графическое окно (фигуру) MATLAB?

10. Каково назначение процедуры **`feval`**?

11. Какие средства существуют в MATLAB для обеспечения численного интегрирования систем дифференциальных уравнений?

12. Какими средствами в MATLAB можно осуществить повторное вычисление большого фрагмента программы?

Урок 3. Интерфейс системы MatLAB

Команды связи с операционной средой

Использование MatLAB при оформлении текстовых документов

Использование в MatLAB файлов данных

Под интерфейсом системы MatLAB далее понимаются средства, позволяющие связывать систему с различными приложениями (операционной системой, текстовым редактором Word, с файлами данных, полученных другими системами и т. п.). Из них здесь рассматриваются только важнейшие и наиболее употребительные в инженерной практике.

3.1. Команды связи с операционной средой

В MatLAB предусмотрен ряд команд общего назначения, позволяющих использовать возможности и функции операционной системы при выполнении программ в MatLAB.

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>. Те же команды можно употреблять в тексте программ, написанных на языке MatLAB.

Эти команды удобно разделить на такие группы:

- управляющие команды и функции;
- команды управления переменными и рабочим пространством;
- команды работы с файлами и операционной системой;
- команды управления командным окном;
- команды запуска и выхода с MatLAB;
- команды получения общей информации.

Рассмотрим вкратце некоторые из этих команд и функций.

3.1.1. Управляющие команды и функции

help	вывод на экран первых строк описания указанной программы или функции;
what	вывод на экран перечня имен M, MAT и MEX файлов в текущей папке;
type	вывод на экран текста указанного M-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.

3.1.2. 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.

3.1.3. Команды работы с файлами и операционной системой

cd	заменить текущий каталог указанным;
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.

3.1.4. 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.

3.1.5. Команды запуска и выхода с MatLAB

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

3.1.6. 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.

3.2. Использование MatLAB при оформлении текстовых документов

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. Документы Word, созданные с использованием связи с MatLAB, обычно называют M-книгами.

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 M-книги, нужно, чтобы этот шаблон, носящий имя M-book.dot, был предварительно подсоединен к редактору Word.

3.2.1. Создание новой M-книги

Чтобы приступить к написанию новой M-книги, нужно:

1) запустить редактор Word;

2) выбрать в окне Word команды Файл ► Создать;

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. 3.1).

3.2.2. Написание M-книги

Написание M-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 Как Входную Ячейку) в меню Notebook (см. рис. 3.1), или нажать клавиши Alt+D; после этого вид строки команды должен измениться - символы команды приобретают темно-зеленый цвет, а команда становится окаймленной квадратными скобками темно-серого цвета;

4) выбрать мышью 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 окаймлены квадратными скобками.

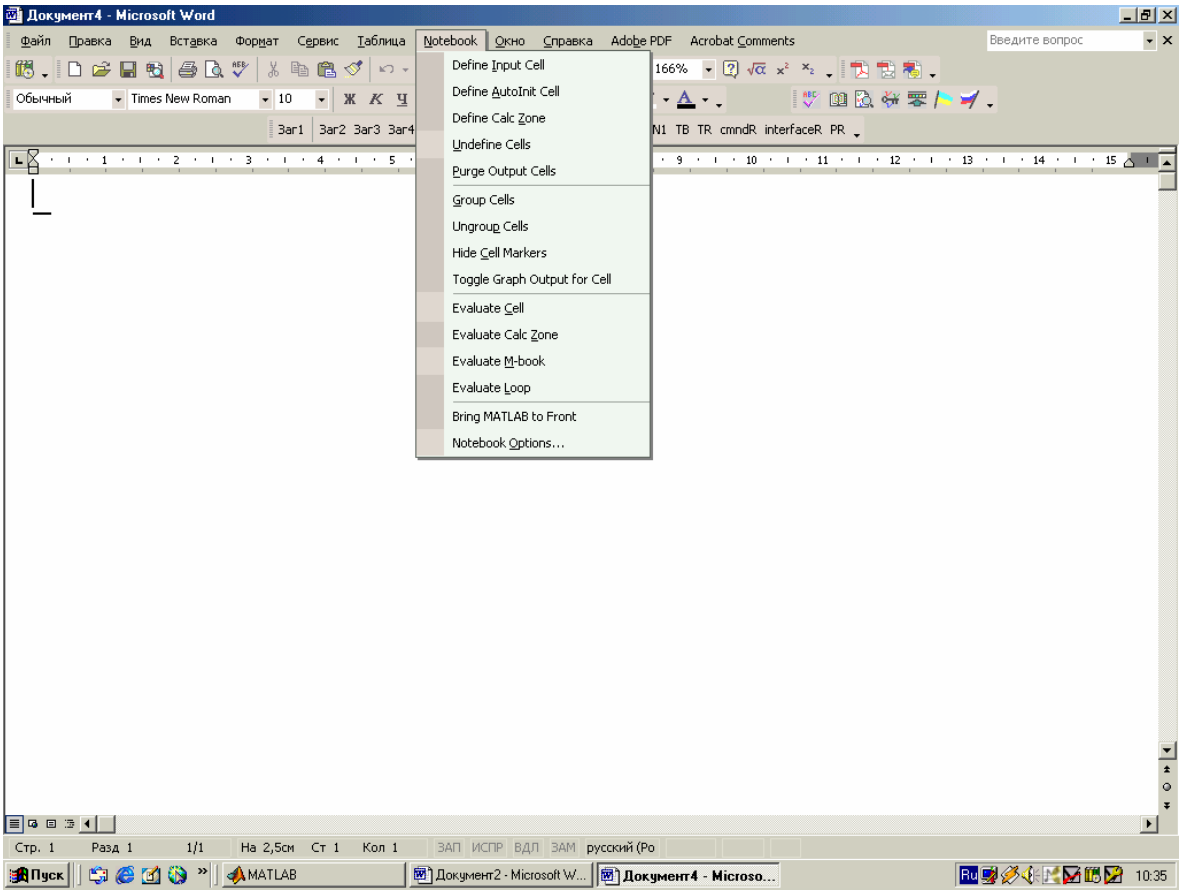


Рис. 3.1 Меню Notebook редактора Word

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. 3.2).

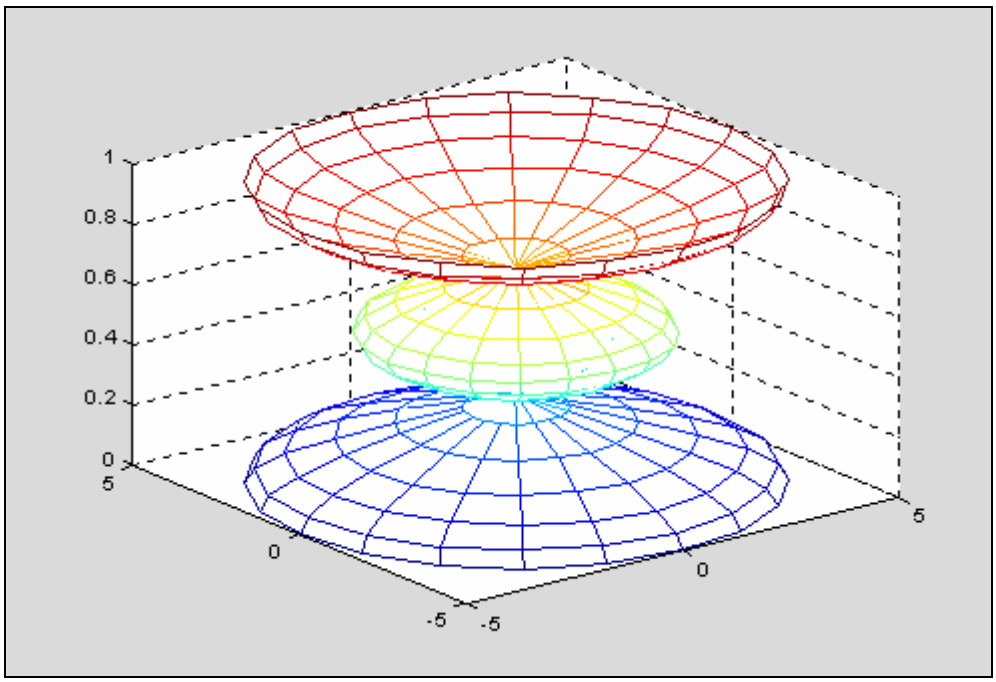


Рис. 3.2. График MatLAB, полученный в тексте Word

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

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

2) выбрать команды `Notebook ► Undefine Cells` 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.

3.2.3. Редактирование M-книги

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

- войти в редактор `Word` и открыть, используя `Файл ► Открыть`, файл M-книги, которую нужно корректировать;

- войти в редактор `Word ► Файл` и выбрать нужный файл с M-книгой из списка последних документов в нижней части;

- дважды "щелкнуть" мышью на имени документа M-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`.

3.2.4. Преобразование документа Word в M-книгу

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

- в редакторе `Word` создать новую (пока пустую) M-книгу;

- в меню редактора `Word` выбрать `Вставка ► Файл` ;

- выбрать в появившемся окне `Вставка файла`, файл, который нужно превратить в M-книгу, и нажать клавишу `Enter`.

3.2.5. Особенности использования MatLAB в среде Word

При написании M-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, т. е. готовыми M-программами, а также процедурами и функциями, доступными лишь при работе с Script-файлами (например, процедурами создания меню и т.п.).

Последнее ограничение не является непреодолимым. Его можно обойти, если воспользоваться командой `Bring MATLAB to Front` (Вывести MatLAB на передний план) 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 M-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 M-книги.

3.2.6. Изменение параметров вывода результатов

В меню `Notebook` есть команда **Notebook Options**, которая позволяет устанавливать некоторые параметры оформления результатов в M-книге по усмотрению пользователя. Если эту команду активизировать с помощью мыши, на экране возникнет окно, представленное на рис. 3.3.

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 M-книги (опция **Embed Figure in M-book**);
- использовать при выводе графических изображений в M-книгу 16 цветов (опция **Use 16-Color Figures**) или 256 цветов.

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

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

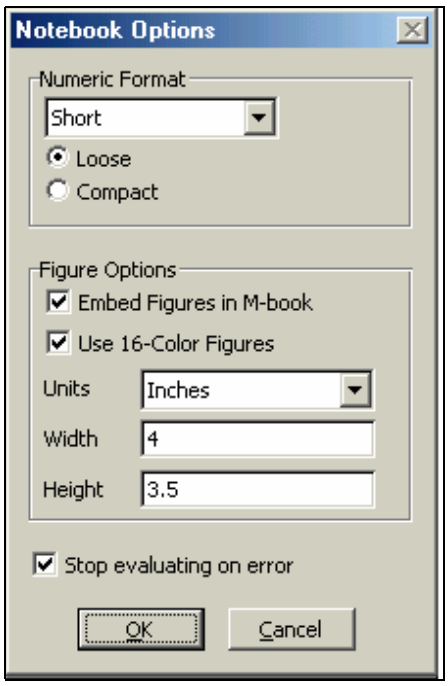


Рис. 3.3. Окно Notebook Options

3.3. Использование в MatLAB файлов данных

Система MatLAB располагает значительным набором специальных функций для работы с файлами произвольных форматов и типов, например, с файлами программ на языке C или с MAT-файлами, представляющими собой запись состояния рабочего пространства MatLAB.

Однако часто возникает задача использования данных не из MAT-файла и не с клавиатуры, а из уже сформированного файла, запись которого производилась самыми разными способами и программными средствами. Или необходимо сохранить результаты вычислений в файлах заранее оговоренного формата, предназначенных для использования в составе иных программных средств.

Такие задачи решаются путем использования файлов данных общих для всех программных средств типов, которые разделяются на *бинарные файлы* и *текстовые файлы*.

Бинарные файлы предназначены для хранения произвольных данных в виде потока байтов. Основные операции с такими файлами – записи и считывания заданного количества байтов информации.

В отличие от них, текстовые файлы содержат записи, трактуемые как символы определенной кодировки, включая набор таких управляющих символов как "возврат каретки", "перевод строки", "конец файла".

3.3.1. Открытие и закрытие файлов

Независимо от типа файла перед началом работы его нужно открыть специальной функцией **fopen**:

```
<идентификатор_файла> = fopen('<имя_файла>','<флаг>').
```

Здесь обозначено:

- <идентификатор_файла> - имя, под которым файл будет записываться в оперативную память ПК при осуществлении с ним операций чтения-записи;

- <имя_файла> - имя файла, под которым он записан (или будет записан) на внешний носитель;

- **<**флаг**>** - параметр, называемый *флагом открытия файла* и несущий информацию о способе работы с файлом.

Флаг может принимать следующие символьные значения:

'r'	только для чтения;
'w'	только для записи (предыдущее содержимое теряется, а несуществующий файл создается);
'w+'	удаление содержания существующего файла или создание нового и открытие его для записи и чтения;
'r+'	чтение и запись одновременно;
'a'	добавление в конец файла;
'a+'	создание и открытие нового файла или открытие существующего для записи, чтения и добавления в конец файла.

К указанным символам следует добавить символ ***'b'*** для открытия файла в *бинарном режиме* или ***'t'*** – для открытия файла в *текстовом режиме*.

Если файл данных больше не используется для чтения или записи, его следует закрыть функцией **fclose**:

```
fclose(<идентификатор файла>).
```

3.3.2. Чтение и запись информации в бинарные файлы

Чтение и запись информации в бинарный файл осуществляются функциями **fread** и **fwrite.**

Функция **fwrite** предназначена для записи информации в бинарный файл и имеет такую форму обращения к ней

```
fwrite(<идентификатор файла>, A,'precision'),
```

где A – числовой вектор (или матрица), элементы которого необходимо записать в файл, 'precision' символьный параметр, указывающий сколько памяти отводится на запись отдельного числа. В MatLAB для записи вещественных чисел используется тип **double**, под которую отводится 8 байт (или 64 бита) памяти. Поэтому для записи таких данных в бинарный файл нужно указать в качестве параметра 'precision' текстовую строку ***'float64'***.

Следует заметить, что считывание значений элементов некоторой матрицы происходит по столбцам, т. е. сначала считываются элементы первого столбца матрицы, затем второго и т. д. В таком же порядке располагаются записываемые элементы в бинарном файле.

Рассмотрим пример записи значений элементов вектора x размером (1×5) и квадратной матрицы y размером (3×2)

```
x=1:5
x =     1     2     3     4     5
```

```
y=[6 7;8 9;10 11]
y =     6     7
        8     9
        10    11
```

в бинарный файл tst_dat.bin:

```
F1=fopen('tst_dat.bin','wb');
fwrite(F1,x,'float64');
fwrite(F1,y,'float64');
fclose(F1);
```

Теперь осуществим чтение данных из записанного файла. Для этого откроем файл с флагом '***rb***' и применим функцию **fread**, предназначенную для чтения информации из бинарного файла. Обращение к ней осуществляется по форме

```
[A, count]= fread(<идентификатор_файла>, [m,n],'precision'),
```

где A – имя числовой матрицы, элементы которой принимают считанные из файла значения, m – число строк этой матрицы, n – число ее столбцов, count – количество действительно считанных элементов из файла данных, 'precision' - символьный параметр, указывающий сколько памяти отводится на запись в матрице A отдельного числа. Для записи в MatLAB данных из бинарного файла следует указать в качестве параметра 'precision' текстовую строку ***'float64'***. Параметр <идентификатор_файла> является символьной строкой, содержащей имя бинарного файла, из которого считывается информация.

При считывании из бинарного файла следует иметь в виду:

- считывание осуществляется с места бинарного файла, где находится указатель;
- при первом считывании только что открытого файла данных указатель расположен в самом начале бинарного файла перед первым его элементом;
- после очередного считывания функцией **fread** указатель перемещается вдоль файла данных и устанавливается после последнего считанного элемента.

Приведем несколько примеров.

В записанном нами файле tst_dat.bin содержится 11 чисел типа **double**. Сначала считаем его в едиственный вектор длиной 15 элементов:

```
F2=fopen('tst_dat.bin','rb');
[V1, c1]=fread(F2,[1,15],'float64')
fclose(F2);
```

Получим:

```
V1 =     1     2     3     4     5     6     8    10     7     9    11
c1 =    11
```

Как видим, реально было считано 11 элементов. Порядок расположения этих элементов в записанном файле данных ясен из полученного вектора V1. Можно убедиться, что элементы исходной матрицы **y** были считаны при записи по столбцам.

Теперь считаем эти же данные в вектор и матрицу тех же размеров, которые были использованы при записи:

```
F2=fopen('tst_dat.bin','rb');
[X1, c2]=fread(F2,[1,5],'float64')
[Y1, c3]=fread(F2,[3,2],'float64')
fclose(F2);
```

```
X1 =     1     2     3     4     5
c2 =     5
Y1 =
     6     7
     8     9
    10    11
c3 =     6
```

Результат считывания полностью совпадает с исходными данными.

Наконец, попробуем считать матрицу **y** в матрицу Y2 заведомо больших размеров (4×3):

```
F2=fopen('tst_dat.bin','rb');
[X2, c4]=fread(F2,[1,5],'float64')
[Y2, c5]=fread(F2,[4,3],'float64')
fclose(F2);
```

```
X2 =     1     2     3     4     5
c4 =     5
Y2 =
```

```
     6     9
     8    11
    10     0
     7     0
c5 =     6
```

Результат показывает, что заполнение новых матриц последовательно считываемыми элементами бинарного файла осуществляется по столбцам. Недостающие элементы матриц заполняются нулями.

Примечание. Нетрудно заметить, что правильное считывание данных из бинарного файла возможно только при условии, что заранее известно, в каком формате записаны данные в этот файл.

Например, если считать данные в формате ***float32***, то получим:

```
F2=fopen('tst_dat.bin','rb');
[X2, c4]=fread(F2,[1,5],'float32')
[Y2, c5]=fread(F2,[4,3],'float32')
fclose(F2);

X2 =         0    1.8750         0    2.0000         0
c4 =     5
Y2 =
    2.1250    2.3125    2.5000
         0         0         0
    2.2500    2.3750    2.5625
         0         0         0
c5 =    12
```

что ни в коей мере не отражает записанные исходные данные.

3.3.3. Чтение и запись информации в текстовые файлы

Текстовые файлы данных отличаются от бинарных прежде всего тем, что информация в них содержится в виде закодированных текстовых символов, т. е. в символьном виде. Отсюда и название таких файлов. В число записываемых символов входят и такие явно не регистрируемые символы, как символ окончания строки, перевода каретки, абзаца и др. Поэтому в текстовые файлы данных записываются такие данные, которые образуют некоторый сформированный текстовый фрагмент.

Текстовые файлы пригодны и для записи чисел, если предварительно преобразовать эти числа в символьное представление. Мы уже сталкивались при знакомстве с MatLAB с символьным представлением чисел, когда знакомились с тем, как выводятся числа в командное окно. Напомним, что в MatLAB существуют такие форматы символьного представления чисел: *Short*, *ShortE*, *ShortG*, *Long*, *LongE*, *LongG*, *Hex*, *Bank*, *Plus* и *Rational*.

Остановимся прежде всего на записи и чтении числовых данных.

Запись данных в текстовый файл осуществляется применением функции **`fprintf`**. Обращаться к ней следует в форме:

`fprintf('<имя_файла>','строка_ управляющих_символов',<ПЗВ> )`

Здесь <имя_файла> - имя файла, в который записываются данные, <ПЗВ> - перечень записываемых величин (они должны быть заданы (определены) до открытия файла для записи). Строка управляющих символов (она должна быть заключена в апострофы) содержит информацию о том, в каком формате будут записываться данные, указанные в <ПЗВ>. Она может содержать, помимо специальных управляющих символов и произвольные обычные символы. Тогда эти символы будут помещены между записываемыми данными.

К управляющим символам относятся:

`%f` спецификатор, означающий, что очередная переменная, подлежащая записи в файл, будет записана как действительное число в форме с фиксированной десятичной запятой; между символами % и f могут быть записаны два целых числа и разделяющая их точка; первое число задает полное количество символов, отводимых на запись числа, второе – число символов после десятичной точки;

`%g` спецификатор, осуществляющий запись числа в форме с плавающей десятичной запятой;

`%s` спецификатор, осуществляющий запись очередной символьной переменной;

`\n` управляющая последовательность символов, означающая конец строки и перевод каретки на следующую строку;

`\t` вставка горизонтальной табуляции;

`\r` перевод каретки на начало строки;

`\b` возврат на один символ;

`\f`	переход к новой странице;
`\" или ''`	проставить знак апострофа;
`%%`	проставить знак процента.

Приведем несколько примеров.

Рассмотрим вначале запись вектора. Сформируем вектор из четырех элементов:

```
V=[pi 1.457e-17 -0.312567 5.089e4]

V =  3.1416e+000  1.4570e-017 -3.1257e-001  5.0890e+004
```

Запишем этот вектор в текстовый файл в формате с фиксированной десятичной точкой:

```
FT=fopen('Text1.txt','w')
fprintf(FT,'%f',V);
fclose(FT)
```

Результат записи теперь можно просмотреть, вызвав записанный файл `Text1.txt` стандартным текстовым редактором `NotePade` (рис. 3.4).

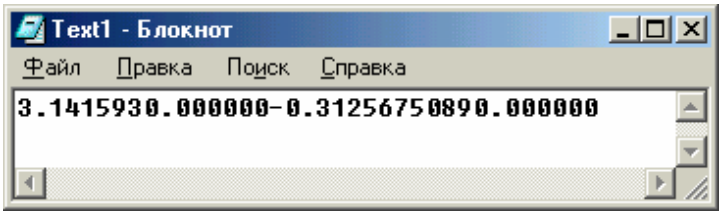


Рис. 3.4. Текстовый файл Text1.txt

Как видим, все числа записаны подряд, без разделения, причем второе число записано как нуль.

Теперь вставим по три пробела между числами:

```
FT=fopen('Text2.txt','w');
fprintf(FT,'%f   ',V);
fclose(FT);
```

Результат показан на рис. 3.5.

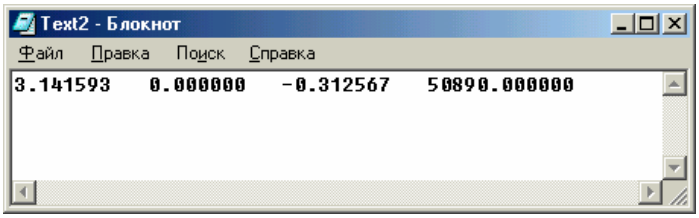


Рис. 3.5. Текстовый файл числового вектора с пробелами

Тот же вектор запишем, пользуясь спецификатором с плавающей запятой:

```
FT=fopen('Text3.txt','w');
fprintf(FT,'%g   ',V);
fclose(FT);
```

На рис. 3.6 показан результат. В отличие от предыдущих записей в формате с фиксированной запятой, теперь второй элемент вектора отражен верно.

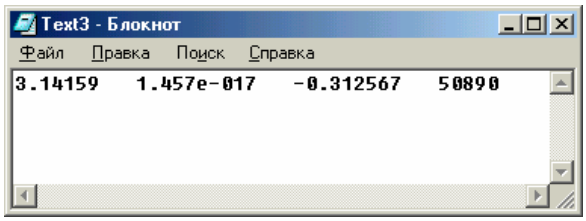


Рис. 3.6. Запись числового вектора в формате с плавающей запятой

Поэтому применение спецификатора **%g** при записи чисел в текстовый файл всегда является более предпочтительным.

Перейдем к записи в текстовый файл числовой матрицы. Сформируем матрицу

```
A= [1 -1.04e-28 7.8e45; -8.1234e-6 6.089 pi; 6 -1098 35]

A =
            1         -1.04e-028      7.8e+045
 -8.1234e-006         6.089           3.1416
            6        -1098           35
```

Запишем ее в текстовый файл:

```
FT=fopen('Text4.txt','w');
fprintf(FT,'%g ',A);
fclose(FT);
```

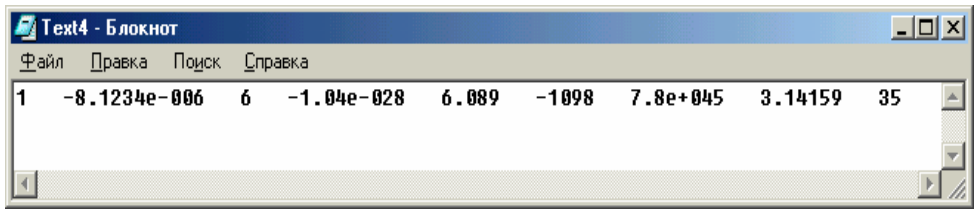


Рис. 3.7. Запись числовой матрицы в формате с плавающей запятой

Как видим из рисунка 3.7, элементы матрицы записываются в этом случае в одну строку последовательно по столбцам.

Чтение из текстового файла данных может быть осуществлено одной из трех функций: **fgetl**, **fgets** или **fscanf**.

Обращение к функции **fgetl** по форме

```
str=fgetl(fid)
```

формирует строку из символов текстового файла данных с идентификатором ***fid*** с удалением символа конца строки. Аналогично, обращение

```
str=fgets(fid)
```

формирует строку из символов текстового файла данных с идентификатором **fid** с символом конца строки.

Функция **fscanf**, вызванная в виде:

```
A=fscanf(fid,format,size),
```

осуществляет считывание из файла количества данных, указанных в параметре **size**, преобразует их из строки символов в иной формат (например, числовой) в соответствии с параметром ***format*** и присваивает полученные значения элементам матрицы **A**. Параметр ***size***, задаваемый в виде [m,n], где m и n – целые положительные числа, определяет количество строк (m) и столбцов (n) формируемой матрицы **A.** Параметр ***format*** должен быть строкой символов (a, значит, должна быть окаймлена апострофами). В число этих символов могут входить обычные символы, спецификаторы и управляющие последовательности символов, аналогичные тем, что были описаны при описании функции **fprintf**. Отличие заключается лишь в том, что теперь эти спецификаторы говорят о количестве символов, считываемых из файла, о формате, в котором они считываются и о типе данных, в который преобразуются считанные символы (тип элементов матрицы **A).**

Для считывания числовых данных наиболее удобна функция **fscanf** - единственная из функций считывания, сразу формирующая числовые данные в MatLAB.

Рассмотрим на примерах, как осуществляется воспроизведение записанных в файл данных при разных способах чтения.

Прежде всего проведем чтение файла Text1.txt функцией **fscanf**. Напомним, что запись в этот файл производилась слитно, без разделителей между числами. Такой же формат применим и для чтения данных:

```
FT=fopen('Text1.txt','r')
Vnov=fscanf(FT,'%f',[1,4]);
fclose(FT)
Vnov
```

```
Vnov =       3.1416            0     -0.31257            0
```

В результате получаем вектор, в котором неверно не только второе число (чего следовало ожидать, так как оно не записалось в текстовый файл, см. рис. 3.4), но и четвертое число, которое было правильно записано в файл.

Картина меняется, если при записи между отдельными числами ставится какой-либо разделительный символ. Например, при записи вектора в файл `Text2.txt` таким символом был пробел. Прочитаем вектор из этого файла, используя тот же разделитель:

```
FT=fopen('Text2.txt','r');
V1nov=fscanf(FT,'%f   ',[1,4]);
fclose(FT);
V1nov

V1nov =       3.1416            0     -0.31257        50890
```

Теперь четвертый элемент считан тоже верно.

Запись в файл `Text3.txt` была осуществлена в формате **g**, и все числа в нем отражены без искажений. Считаем данные из этого файла тем же форматом с тем же разделителем между числами:

```
FT=fopen('Text3.txt','r');
V2nov=fscanf(FT,'%g   ',[1,4]);
fclose(FT);
V2nov

V2nov =       3.1416   1.457e-017     -0.31257        50890
```

Получается результат со всеми верными числами.

Считаем матрицу A из файла `Text4.txt` в том же формате, в котором она была записана в этот файл:

```
FT=fopen('Text4.txt','r');
Anov=fscanf(FT,'%g   ',[3,3]);
fclose(FT);
Anov

Anov =
            1       -1.04e-028            7.8e+045
 -8.1234e-006        6.089                3.1416
            6     -1098                   35
```

В результате получаем матрицу B, полностью совпадающую с исходной матрицей A.

Вывод. При записи и чтении числовых массивов в текстовых файлах целесообразно использовать спецификатор **%g** и функцию **fscanf** для чтения. Это позволяет избежать возможных искажений чисел при записи и чтении. При этом необходимо отделять числа каким-либо разделительным символом. При чтении числовых массивов следует применять те же разделительные символы, что были применены в функции **fprintf** при их записи в файл.

3.4. Вопросы для самопроверки

1. Какие команды операционной среды можно выполнить в командном окне MATLAB?

2. Что следует понимать под M-книгой?

3. Как, находясь в текстовом редакторе Word, осуществлять расчеты и строить графики, используя MATLAB?

4. Какие существуют типы файлов данных и чем они отличаются друг от друга?

5. Какими средствами в MATLAB осуществляется чтение и запись информации в бинарные файлы данных?

6. Какие средства предусмотрены в MATLAB для осуществления чтения и записи информации в текстовые файлы данных?

7. Каковы преимущества и недостатки использования бинарных и текстовых файлов данных для хранения и считывания информации?

Урок 4. Классы вычислительных объектов

Основные классы объектов

Производные классы

Пример создания нового класса POLYNOM

Создание методов нового класса

К числу основных достижений современных языков программирования высокого уровня является наличие средств осуществления так называемого объектно-ориентированного программирования (ООП). Под этим обычно понимают возможность пользователя самому определять (задавать) классы вычислительных объектов, их свойства и операции по их преобразованию. Создание собственных классов объектов и методов оперирования с ними существенно увеличивает возможности использования вычислительной техники, упрощает программирование и делает его более удобным, прозрачным и эффективным.

В системе MatLAB предусмотрены достаточно простые средства, позволяющие решать задачи объектно-ориентированного программирования. К этим средствам относятся возможность создания новых классов вычислительных объектов и программ (методов) оперирования с ними.

4.1. Основные классы 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.

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*** - единственный метод - операция сохранения (чаще всего используется в пакете 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 (конкатенацию) 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 со строкой ***s2*** на строку ***s3***:

```
>> st = [st1 st2 st3]
st =  Это  строка  символов.
>> y = strrep(st,'о','а')
y =  Эта  страка  симвалав.
>> x = strrep(st,'а','о')
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 числовых параметров, 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 Числовой формат**.** В качестве примера рассмотрим формирование текстовой строки с включением значения переменной:

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

Аналогичным образом, при помощи процедуры **mat2str(A)** можно получить значение матрицы A в виде символьной строки:

```
>> A = [1,2,3;4 5 6; 7 8 9];
>> disp(mat2str(A))
[1 2 3;4 5 6;7 8 9]
```

Обратный переход от символьного представления числа к численному осуществляется процедурой **str2num**:

```
>> stx = num2str(x)
stx = 3. 1416

>> y = str2num(stx)
y =     3. 1416

>> y+5
ans =     8. 1416

>> z = stx+5
z =     56    51    54    57    54    59
```

Последний результат получен вследствие того, что строка символов ***stx*** при включении ее в арифметическую операцию автоматически перестраивается в класс ***double***, т. е. все символы, составляющие ее, заменяются целыми числами, равными коду соответствующего символа. После этого сложение полученного числового вектора с числом 5 происходит по обычным правилам, т. е. 5 суммируется с каждым из кодов символов. Проверим это, учитывая, что *с помощью процедуры* **double(str)** *можно получить числовое представление строки символов* ***str*** *в виде кодов 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 присвоении конкретных значений полям записи.

Обращение к любому полю с 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 добавлять 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 = 'Алексей'; PG81(2). bat = 'Иванович';
>>  PG81(3). fam = 'Попель' ;
>>  PG81(3). imya = 'Богдан'; PG81(3). bat = 'Тимофеевич';
>>  PG81(3). god = 1982
PG81 =
```

```
1x3 struct array with fields:
    fam
    imya
    bat
    god
>> PG81(2). god
ans =      []
```

Чтобы удалить некоторое поле из всех элементов массива записей, надо использовать процедуру **rmfield** по схеме **S = rmfield (S, 'имя поля ')**, где S - имя массива записей, который корректируется. Рассмотрим пример:

```
>> PG81 = rmfield(PG81, 'bat')

PG81 =

1x3 struct array with fields:

    fam

    imya

    god
```

Класс ***struct***, как видим, имеет незначительное число методов, что делает его непосредственное использование при расчетах довольно проблематичным. Однако именно *на 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 являются ячейки, которые сами могут содержать любой тип массива, в том числе и массив ячеек. Массивы ячеек позволяют хранить массивы с элементами разных типов и разных измерений. Например, одна из ячеек может содержать матрицу действительных чисел, вторая - массив символьных строк, третья - вектор комплексных чисел. Можно строить массивы ячеек любых размеров и любой структуры, включая и многомерные.

Создание массива ячеек

Создать массив ячеек можно двумя способами:

- использованием операторов присваивания;
- при помощи функции **cell** предварительно сформировать пустой массив, а потом присвоить значения отдельным ячейкам.

Применение операторов присваивания

Есть два способа присвоить значения отдельным ячейкам - *индексация ячеек* и *индексация содержимого*.

Индексация ячеек. При присваивании значений отдельным элементам массива ячеек индексы ячейки в левой от знака присваивания части размещают в скобках, используя стандартные обозначения для массива, а в правой части *присваиваемое значение ячейки помещают в фигурные скобки*.

Для примера рассмотрим создание массива 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. 0000i]            [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 3
   -3.1416   -2.5133   -1.8850
  Columns 4 through 6
   -1.2566   -0.6283         0
  Columns 7 through 9
    0.6283    1.2566    1.8850
  Columns 10 through 11
    2.5133    3.1416
```

Для отображения структуры массива ячеек в виде графического изображения на экране предназначена функция **cellplot**

```
>> cellplot(C)
```

Результат приведен на рис. 4.1.

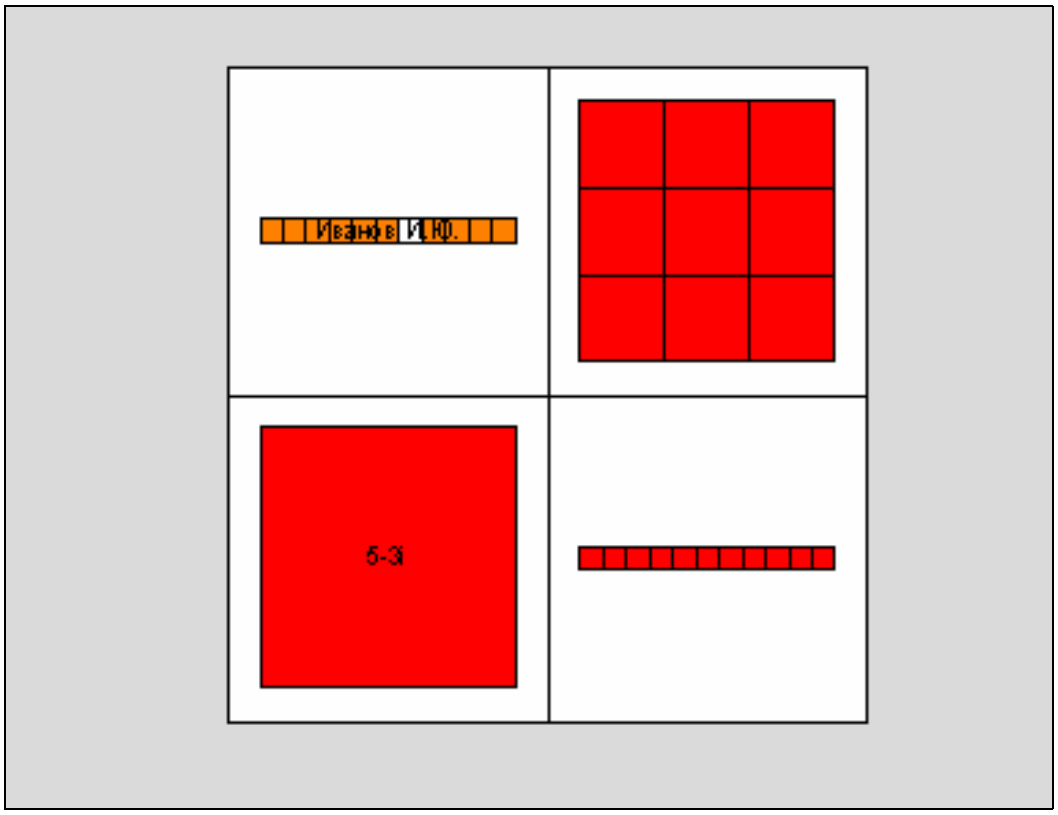


Рис. 4.1. Результат действия функции cellplot

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 C ячеек может быть построен так:

```
>> C = { ' Иванов И. Ю.', [1 2 3; 4 5 6; 7 8 9]; 5-3i,-pi:pi/5:pi }

C =     ' Иванов И. Ю.'        [3x3  double]
```

```
[5. 0000- 3. 0000i]    [1x11 double]
```

Применение функции cell

Функция **cell** позволяет создать шаблон массива ячеек, заполняя его пустыми ячейками.

Пример. Создадим пустой массив ячеек размером (2*3):

```
>> A = cell(2,3)
A =
     []     []     []
     []     []     []
```

Заполним одну из ячеек, используя оператор присваивания:

```
>> A(2,2) = {0 : pi/10:2*pi}
A =      []              []              []
         []          [1x21 double]       []
```

Извлечение данных из массива ячеек

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 ячейки производится дополнительным указанием в скобках индексов элемента массива, находящегося в нужной ячейке:

```
>> x =C{1,2}(2,3)
x =     6
>> y= C{1,1}(1:5)
y = Иван
```

Второй способ позволяет *извлекать из массива ячеек другой массив ячеек*, составляющий часть первого:

```
>> D = A(2,2:3)
D =     [1x21 double]     []
```

В этом случае применяются обычные скобки.

Массивы ячеек используются для объединения массивов данных разных типов и размеров. 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. относятся 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 имеет 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 вычисление функции при заданных значениях аргумента ***x*** и параметров ***P1, P2, ...*** может осуществляться путем обращения к ней в естественной форме, например, **`F(0.6, -0.5, 3)`**.

Классу ***inline*** соответствует подкаталог @INLINE каталога TOOLBOX/MATLAB/FUNFUN. В нем содержатся 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 к конструктору таким образом

F = inline('<математическое выражение>', 'имя1', 'имя2', ...),

то формируется функция, имеющая заданные в 'имя1', 'имя2', ... обозначения аргументов:

```
>>FUN2=inline('cos(alfa)*cos(beta)+...
sin(alfa)*sin(beta)*cos(gamma)','alfa','beta','gamma')

FUN2 =

     Inline function:
     FUN2(alfa,beta,gamma) = cos(alfa)*cos(beta)+sin(alfa)*sin(beta)*cos(gamma)
```

Наконец, при обращении вида

F = inline('<математическое выражение>', n),

создается функция, которая имеет один аргумент ***x*** и ***n*** параметров с заданными именами P1, P2, ... , Pn**.** Т. е. в выражении, кроме некоторых заданных чисел, 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 к алгоритму.

Таким образом, *процедура* **feval** *позволяет использовать внешние функции при программировании* в среде MatLAB. Общий вид обращения и примеры применения процедуры **feval** приведены в 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)]

[$(s^2-5s+6)/(s^2+s)$]

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 - матрицы A, B, C, D, соответствующие уравнениям в пространстве состояния:

$$Edx/dt = Ax + Bu\ ,\quad y = Cx + Du\ ,$$

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 подкаталоге @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 = []
```

Подробнее с методами класса CONTROL и их использование можно ознакомиться в главе 6.

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 вызовите из командного окна Файл ►Открыть, а затем в появившемся окне перейдите к папке Toolbox\Matlab\Polyfun**.** Воспользуйтесь пиктограммой создания новой папки в этом окне, чтобы открыть новую папку по имени @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 n=2:k                        % вектора
       p.c=conv(p.c,vs(n,:));     % 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) & a==1
                        s=[s 'x^' int2str(d)];
                  end
                  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
```

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*x^8 + 2*x^7 + 3*x^6 + 4*x^5 - 6*x^2 - 5*x - 7;
```

Создадим теперь полином-объект по заданным его корням и значению старшего коэффициента:

```
>> Pol2=polynom([1 2 3 4 5],-5)
 Pol2 = -5*x^5 + 75*x^4 - 425*x^3 + 1125*x^2 - 1370*x + 600;
```

Проверим корни созданного полинома, используя процедуру ***roots*** (см. далее):

```
>> roots(Pol2)
ans =
    5.0000
    4.0000
    3.0000
    2.0000
    1.0000
```

Как 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 деление 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)	Формирование 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 - изображение на терминале символьного представления аргумента .
[a b]	horzcat(a,b,...)	Объединение в строку	Два или больше аргумента. Результат - вектор-строка из аргументов
[a; b]	vertcat(a,b,...)	Объединение в столбец	Два или больше аргумента. Результат - вектор-столбец из аргументов
a(s1,...sn)	subsref(a,s)	Индексная ссылка	
a(s1,...sn)=b	subsasgn(a,s,b)	Индексное выражение	
b(a)	subsindex(a,b)	Индекс подмассива	

Перечисленные процедуры в MatLAB могут быть переопределены под теми же именами во всех новообразованных подкаталогах новых классов. После этого обычные операторы арифметических действий и операций сравнения могут применяться и при оперировании объектами новых классов. Смысл этих операций, конечно, может значительно отличаться от обычного и будет определяться содержимым соответствующих М-файлов в подкаталогах классов.

Учитывая это, можно сделать вывод, что М-файлов с названиями, указанными в 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*x^3 - 3*x^2 + 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*x^7 + 9*x^4 + 2*x^3 - 3*x^2 - 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*x^7 + 9*x^4 + 2*x^3 - 3*x^2 - 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 полиномов-объектов:

```
function r=minus(p,q)
  % POLYNOM/MINUS Вычитание полиномов r = p - q
p = polynom(p);
q = polynom(q);
k = q.n - p.n;
r = polynom([zeros(1,k) p. c]-[zeros(1,-k) q. c]);
  % Завершение POLYNOM/MINUS
```

Проверим ее работу на примере:

```
>> P1-P2
 ans = -2*x^7 - 9*x^4 + 2*x^3 - 3*x^2 + 3*x + 1;
```

Создадим еще такие методы класса ***polynom***:

- метод **double**, который осуществляет обратную относительно создания полинома-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 - 0.8579i
>> plot(Pol)
```

Результат представлен на рис. 4.2.

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      polyval   roots
diff  double    mrdivide  plot      polynom   rdivide
```

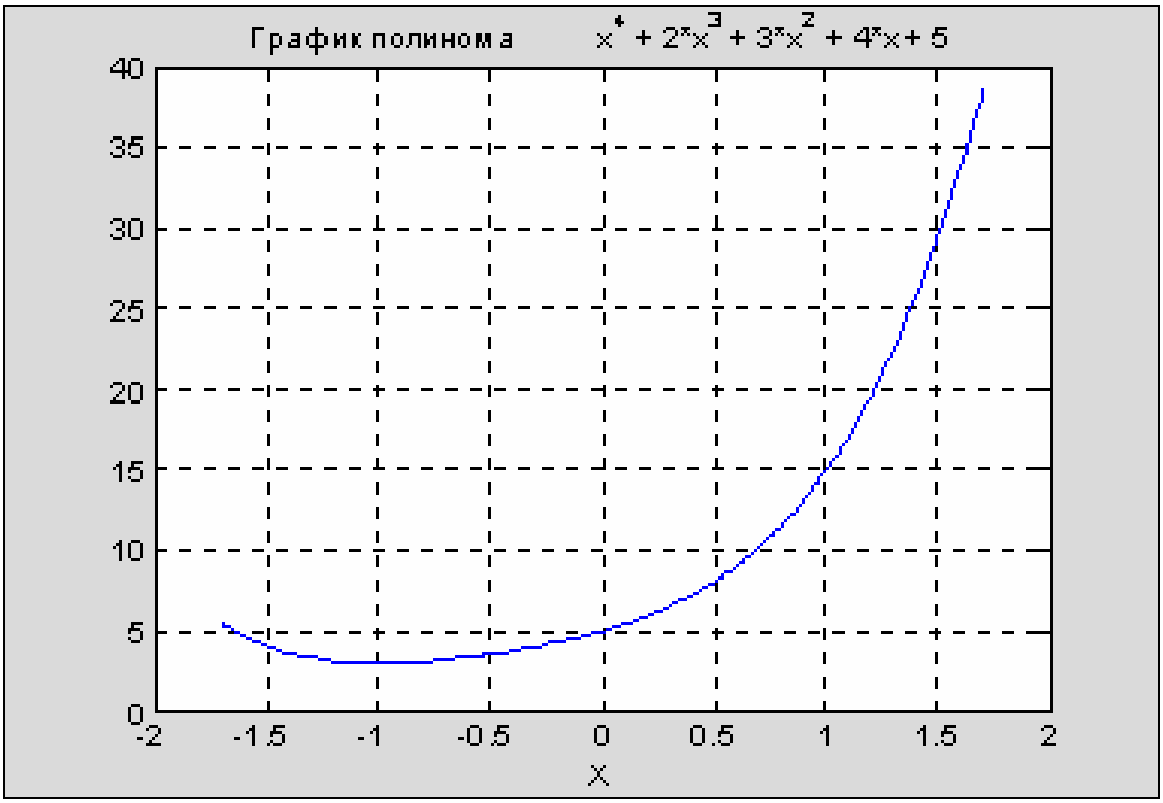


Рис. 4.2. Результат выполнения процедуры plot для полинома-объекта

Теперь мы имеем удобный 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.

4.5. Вопросы для самопроверки

1. Что понимается в MATLAB под классом вычислительных объектов? под методами класса?
2. Какие классы вычислительных объектов составляют основу MATLAB?
3. Какие производные классы используются в MATLAB?
4. На какой основе можно создать в MATLAB новые классы вычислительных объектов?
5. Для каких целей можно использовать классы inline, cell, struct, char?
6. Как создать собственный класс вычислительных объектов?
7. Как создать методы собственного класса вычислительных объектов?
8. Каким образом можно обеспечить использование знаков арифметических операций для выполнения действий над объектами создаваемого класса?

Урок 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 преобра-зования в целях получения информации о величине, которая должна быть измерена, представлена на рис. 5.1.

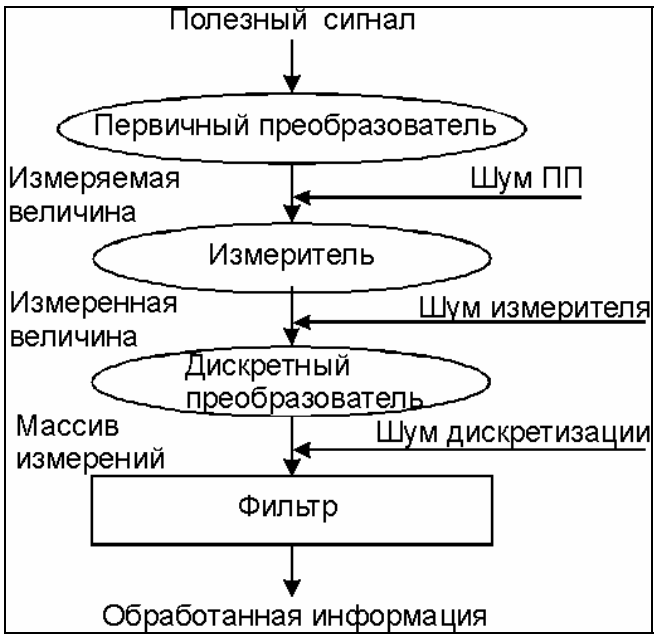


Рис. 5. 1. Схема измерения и преобразования сигнала

Физическая величина, являющаяся полезной (несущей в себе необходимую информацию), редко имеет такую физическую форму, что может быть непосредственно измеренной. Обычно она представляет лишь некоторую составляющую (сторону, часть, черту) некоторой другой физической величины, которая может быть непосредственно измерена. Связь между этими двумя вели-чинами обозначим введением звена, которое назовем *"первичным преобразователем"* (ПП). Обычно закон преобразования известен заранее, иначе восстановить 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 информационной составляющей, другие, вредные составляющие или черты, искажающие полезную информацию. И, хотя зависимость выхода ПП от этих других величин также известна, однако вследствие неконтролируемого возможного изменения последних со временем, часто трудно спрогнозировать их влияние на искажение полезной составляющей. Назовем вносимую ПП вредную составляющую *шумом ПП*.

Пусть образованная таким образом непосредственно измеряемая величина измеряется некоторым измерителем. Любой реальный измеритель вносит собственные искажения в измеряемую величину и дополнительные зависимости от некоторых других физических величин, не являющихся объектом измерения. Назовем эти искажения *шумами измерителя*. Не ограничивая общности, будем полагать, что выходной величиной измерителя является электрический сигнал (*измеренная величина)*, который можно в дальнейшем довольно просто пребразовывать электрическими устройствами.

Для осуществления цифровой обработки измеренная величина должна быть преобразована в дискретную форму при помощи специального устройства, которое содержит э*кстраполятор* и *аналого-цифровой преобразователь* (АЦП). Первый производит фиксацию отдельного текущего значения измеренной величины в отдельные моменты времени через определенный постоянный промежуток времени, называемый *дискретом времени*. Второй переводит это значение в цифровую форму, которая позволяет в дальнейшем осуществлять преобразования с помощью цифровых ЭВМ. Хотя оба устройства могут вносить при таких преобразованиях собственные искажения в выходной (дискретный) сигнал, однако ими обычно пренебрегают, так как в большинстве случаев эти дополнительные искажения значительно меньше шумов ПП и измерителя.

Чтобы на основе полученного дискретизированного сигнала получить полезный сигнал, нужно рассчитать и создать устройство (программу для ЭВМ), которое осуществляло бы такие преобразования входного дискрет-

ного 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 фазочастотным их характеристикам, формировать последовательности типовых временных сигналов и обрабатывать их спроектированными фильтрами. В пакет входят процедуры, осуществляющие преобразования Фурье, Гильберта, а также статистический анализ. Пакет позволяет рассчитывать корреляционные функции, спектральную плотность мощности сигнала, оценивать параметры фильтров по измеренным отсчетам входной и выходной последовательностей.

5.1. Формирование типовых процессов

Моделирование процессов фильтрации невозможно без предварительного моделирования (генерирования) сигналов заданного вида. Поэтому ознакомимся с основными процедурами, позволяющими генерировать сигналы различных типовых форм.

5.1.1. Формирование одиночных импульных процессов

В пакете Signal предусмотрено несколько процедур, образующих последовательности данных, представляющие некоторые одиночные импульсные процессы типовых форм.

Процедура **`rectpuls`** обеспечивает формирование одиночного импульса прямоугольной формы. Обращение вида

```
y = rectpuls(t, w)
```

позволяет образовать вектор ***y*** значений сигнала такого импульса единичной 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,'FontSize',12)
title(' Пример применения процедуры RECTPULS')
xlabel('Время ( с )')
ylabel('Выходной процесс  Y(t)')
```

Рис. 5. 2. График процесса, сгенерированного функцией RECTPULS

Формирование импульса треугольной формы единичной амплитуды можно осуществить при помощи процедуры **tripuls**, обращение к которой имеет вид

```
y =tripuls(t,w,s).
```

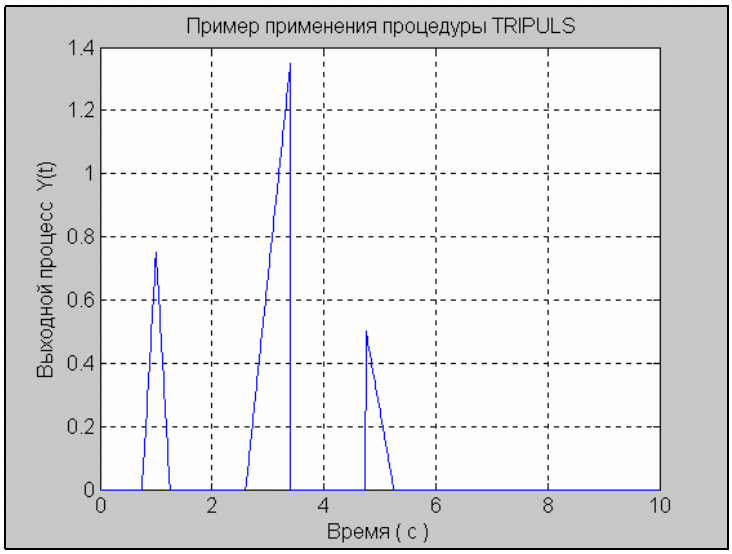


Рис. 5. 3. График процесса, сгенерированного функцией TRIPULS

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,'FontSize',12)
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.

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,'FontSize',12)
title(' Пример применения процедуры GAUSPULS')
xlabel('Время ( с )')
ylabel('Выходной процесс  Y(t)')
```

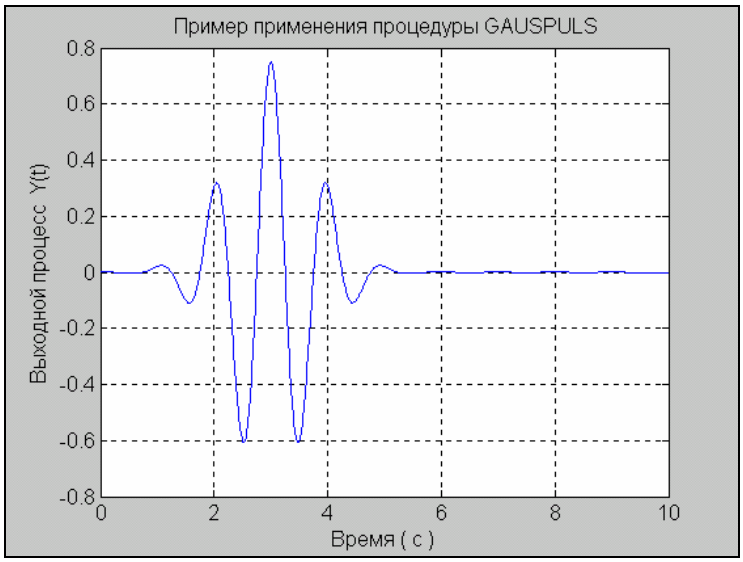


Рис. 5. 4. График процесса, сгенерированного функцией GAUSPULS

Наконец, рассмотрим процедуру ***sinc***, формирующую вектор значений функции **`sinc(t)`**, которая определяется формулами:

$$\sin c(t) = \begin{cases} 1t = 0, \\ \dfrac{\sin(\pi t)}{\pi t} t \neq 0. \end{cases}$$

Эта функция является обратным преобразованием Фурье прямоугольного импульса шириной 2π и высотою 1:

$$\sin c(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega t} d\omega .$$

Приведем пример ее применения:

```
t=0 : 0.01 : 50;
y1=0.7*sinc(pi*(t-25)/5);
plot(t,y1),grid,set(gca,'FontSize',12)
title('Функция SINC  Y(t) = 0.7* SINC(pi*(t-25)/5)')
xlabel('Время ( с )')
ylabel('Выходной процесс Y(t)')
```

Результат изображен на рис. 5.5.

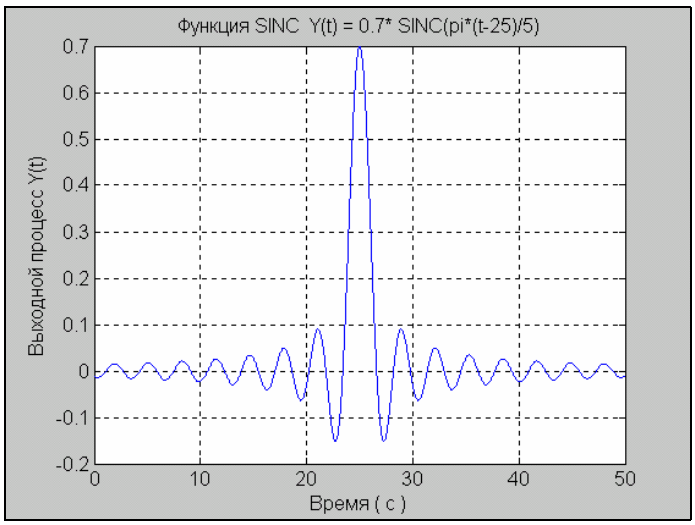


Рис. 5. 5. Графическое представление функции `0.7*SINC(pi*(t-25)/5)`

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,'FontSize',12)
title('Гармонические колебания  Y(t) = 0.7* SIN(pi*t/5)')
xlabel('Время ( с )')
ylabel('Выходной процесс Y(t)')
```

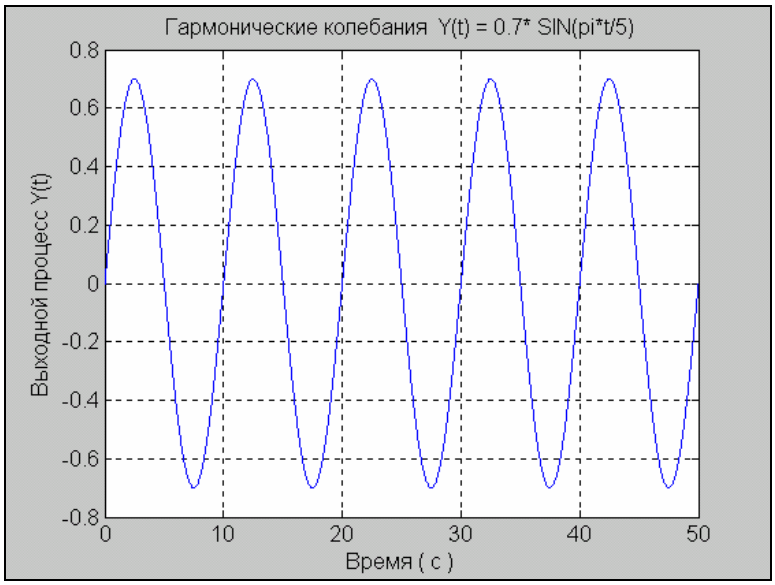


Рис. 5. 6. График синусоидального процесса

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 про-центах от периода волны. Например (результат приведен на рис. 5.7):

```
y=0.7*square(pi*t/5, 40);
plot(t,y), grid,set(gca,'FontSize',12)
title('Прямоугольные волны   Y(t) = 0.7* SQUARE(pi*t/5, 40)')
xlabel('Время ( c )')
ylabel('Выходной процесс  Y(t)')
```

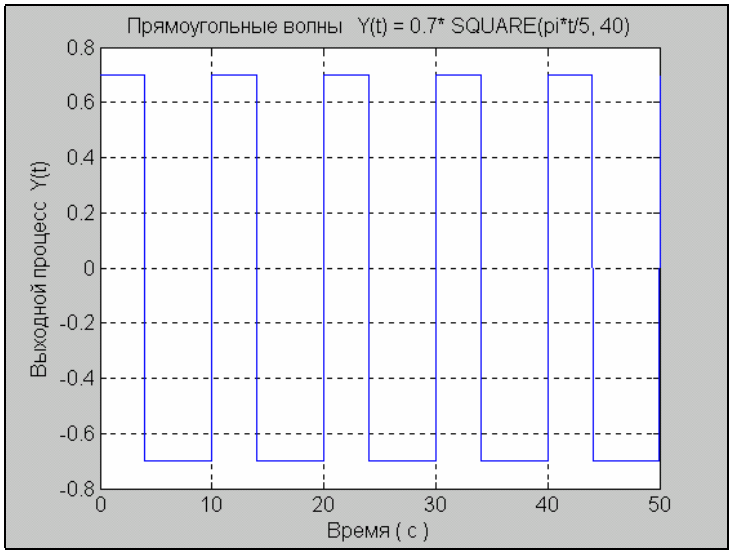


Рис. 5. 7. Результат применения функции SQUARE

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 пило-образные волны с периодом 2π в моменты времени, которые задаются вектором ***t***. При этом параметр ***width*** определяет часть периода, в которой сигнал увеличивается. Ниже приведен пример применения этой процедуры:

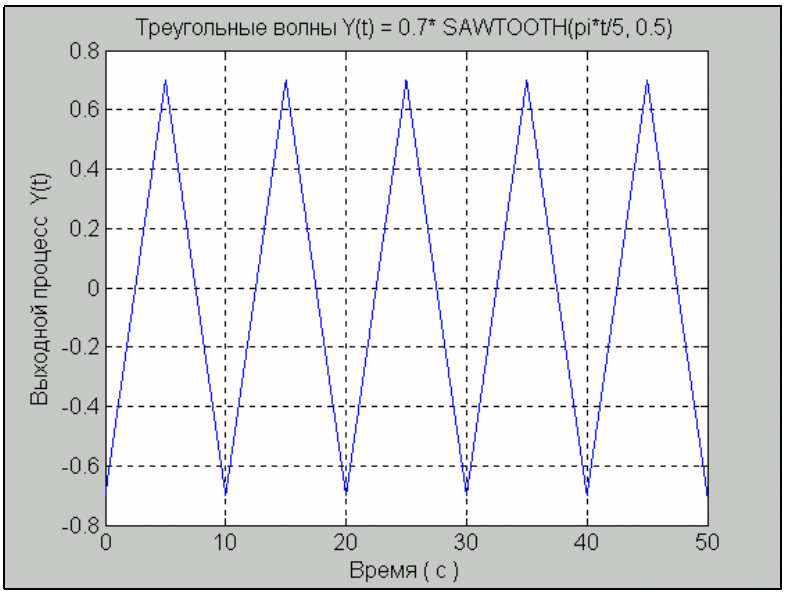


Рис. 5. 8. Результат применения функции SAWTOOTH

```
y=0.7*sawtooth(pi*t/5, 0.5);
plot(t,y), grid,set(gca,'FontSize',12)
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.

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 соответствии с формой обращения к процедуре, определяющей этот импульс.

Ниже приведены три примера применения процедуры **pulstran** для разных форм составляющих импульсов:

- 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,'FontSize',12)
title('Y(t) = 0.7*PULSTRAN(t,d,''tripuls'',5)' )
xlabel('Время ( с )')
ylabel('Выходной процесс  Y(t)')
```

результат представлен на рис. 5.9;

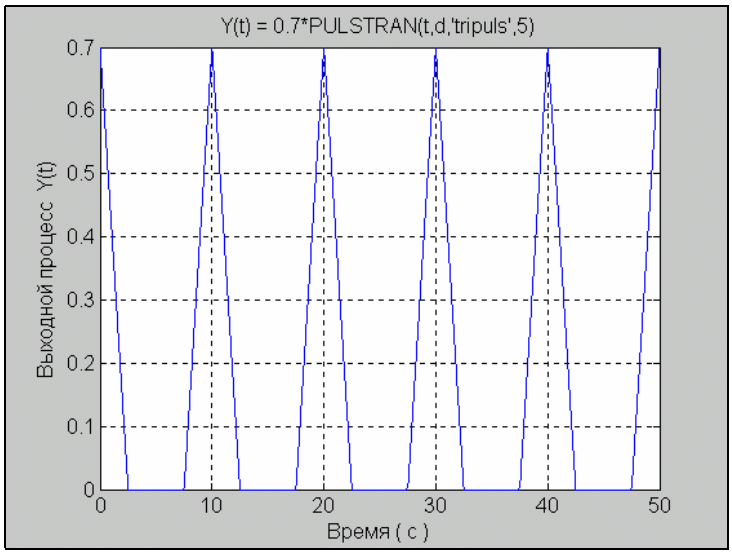


Рис. 5. 9. Результат применения функции 0.7*PULSTRAN(t,d,'TRIPULS',5)

- 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,'FontSize',12)
title('Y(t) = 0.75*PULSTRAN(t,d,''rectpuls'',3)' )
xlabel('Время ( с )')
ylabel('Выходной процесс  Y(t)')
```

результат приведен на рис. 5.10;

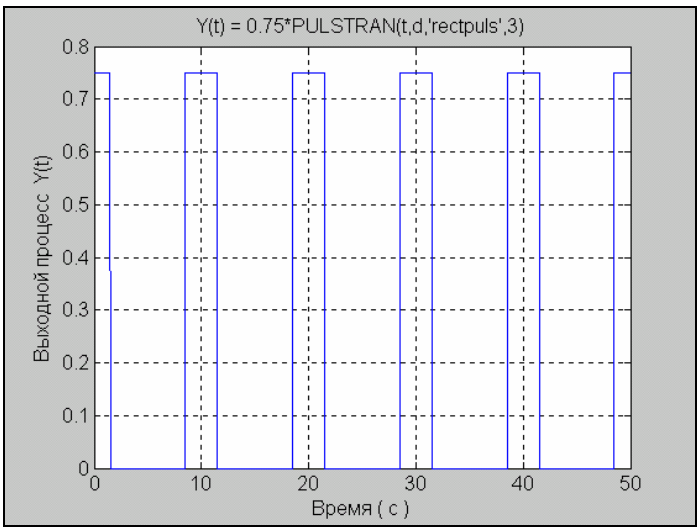


Рис. 5. 10. Результат применения функции 0.75*PULSTRAN(t,d,'RECTPULS',3)

- 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,'FontSize',12)
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.

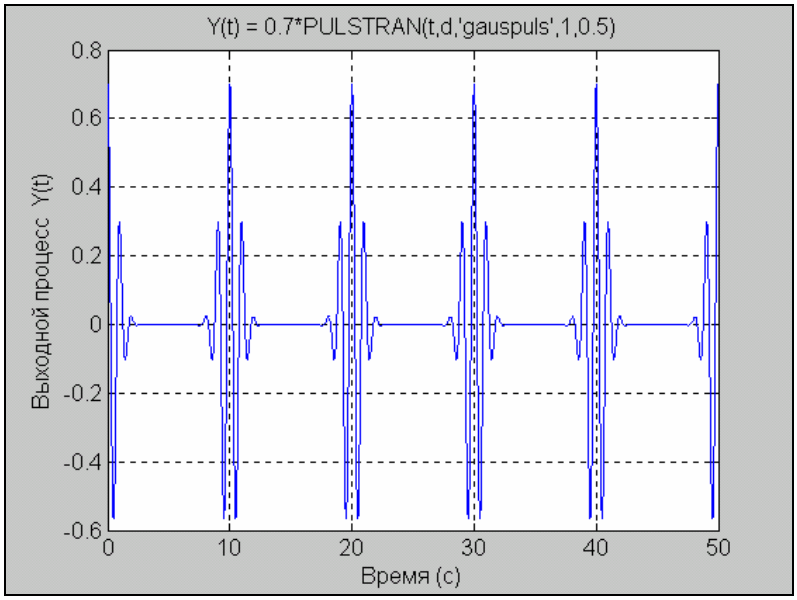


Рис. 5. 11. Результат применения функции 0.7*PULSTRAN(t,d,'GAUSPULS',1,0.5)

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,'FontSize',12)
```

```
title(' Пример процедуры CHIRP')
xlabel('Время ( с )'), ylabel('Выходной процесс  Y(t)')
```

Результат показан на рис. 5.12.

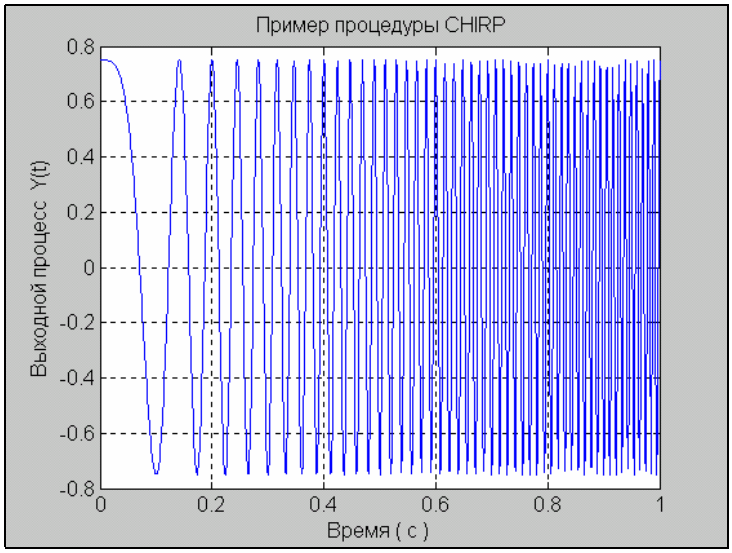


Рис. 5. 12. Результат применения функции CHIRP

Еще одна процедура **diric** формирует массив значений так называемой *функции Дирихле*, определяемой соотношениями:

$$diric(t) = \begin{cases} -1^{k(n-1)} & при \quad t = 2\pi k, \quad k = 0,\pm 1,\pm 2,.. \\ \dfrac{\sin(nt/2)}{n \cdot \sin(t/2)} & при \quad других \quad t \end{cases}$$

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).
```

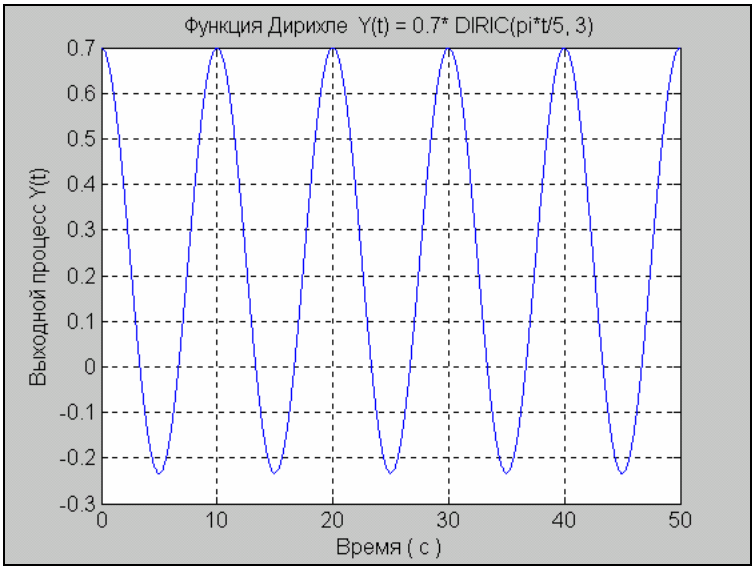


Рис. 5. 13. Результат применения функции **y1=0.7*DIRIC(pi*t/5, 3)**

Далее 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,'FontSize',12)
title('Функция Дирихле  Y(t) = 0.7* DIRIC(pi*t/5, 3)')
xlabel('Время ( c )'), ylabel('Выходной процесс Y(t)')
```

Результаты представлены на рис. 5.13...5.15.

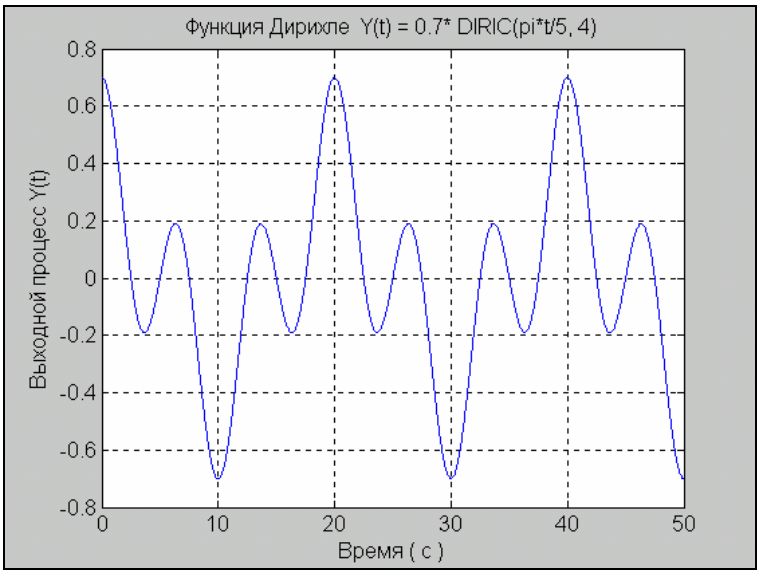


Рис. 5. 14. Функция Дирихле при n=4

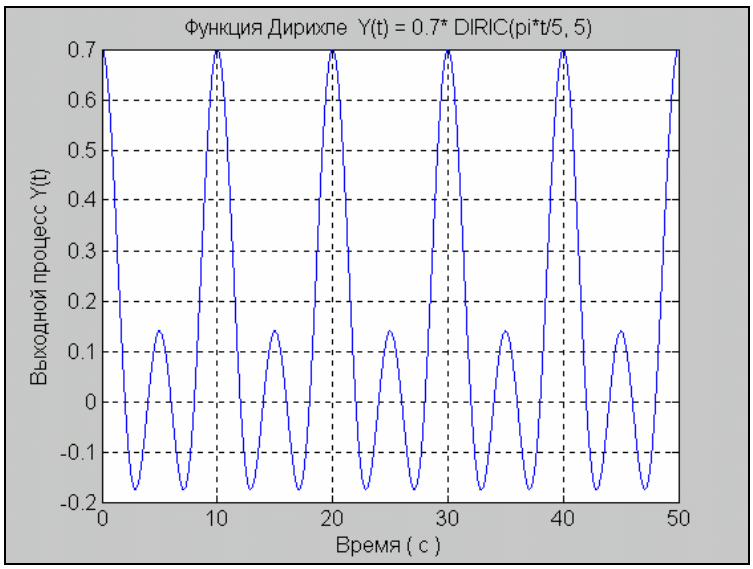


Рис. 5. 15. Функция Дирихле при n=5

5.2. Общие средства фильтрации. Формирование случайных процессов

Следующим необходимым этапом моделирования процессов фильтрации является осуществление непосредственно фильтрации. Для этого необходимо вначале задать сам фильтр как некоторое динамическое звено, а затем провести обработку сформированных ранее сигналов с помощью этого звена, применяя специальную процедуру фильтрации.

5.2.1. Основы линейной фильтрации

Рассмотрим основы линейной фильтрации на примере линейного стационарного фильтра, который в непрерывном времени описывается дифференциальным уравнением второго порядка:

$$\ddot{y} + 2\zeta\omega_o \cdot \dot{y} + \omega_o^2 \cdot y = A \cdot x\,, \tag{5.1}$$

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:

$$W(s) = \frac{y(s)}{x(s)} = \frac{A}{s^2 + 2\zeta\omega_o \cdot s + \omega_o^2}. \tag{5.2}$$

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.

Примечание. Если не указана выходная величина, т.е. обращение имеет вид
`freqs(b, a, w),`
процедура выводит в текущее графическое окно два графика - АЧХ и ФЧХ.

Приведем пример. Пусть для передаточной функции (5.2) выбраны такие значения параметров:

$$A = 1; \quad \zeta = 0.05; \quad T_o = 2\pi / \omega_o = 1.$$

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)
title('А Ч Х   и   Ф Ч Х  фильтра')
```

В результате получим рис. 5.16.

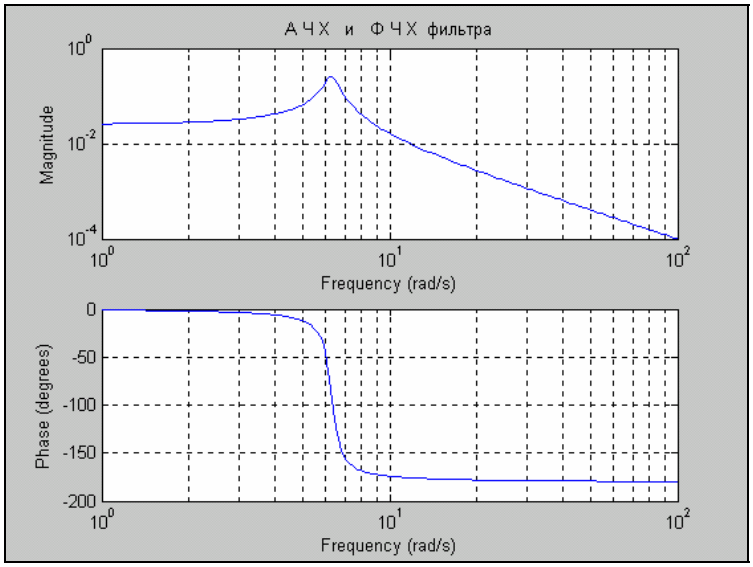


Рис. 5. 16. Результат работы процедуоы FREQS

Допустим, что заданный процесс $x\,(t)$ представлен в виде отдельных его значений в дискретные моменты времени, которые разделены одинаковыми промежутками T_S времени (дискретом времени). Обозначим через $x(k)$ значение процесса в момент времени $t = k \cdot T_S$, где k - номер измерения с начала процесса.

Запишем уравнение (5.1) через конечные разности процессов x и y, учитывая, что конечно-разностным эквивалентом производной $\dot{y}$ является конечная разность

$$\frac{\Delta y(k)}{T_S} = \frac{y(k) - y(k-1)}{T_S},$$

а эквивалентом производной второго порядка $\ddot{y}$ является конечная разность второго порядка

$$\frac{\Delta^2 y(k)}{T_S^2} = \frac{\Delta y(k) - \Delta y(k-1)}{T_S^2} = \frac{y(k) - 2y(k-1) + y(k-2)}{T_S^2}.$$

Тогда разностное уравнение

$$(1 + 2\zeta\omega_o \cdot T_S + \omega_o^2 \cdot T_S^2) \cdot y(k) - 2(1 + \zeta\omega_o \cdot T_S) \cdot y(k-1) + y(k-2) = A \cdot T_S^2 \cdot x(k) \tag{5.3}$$

является дискретным аналогом дифференциального уравнения (5.1).

Применяя к полученному уравнению Z-преобразование, получим:

$$y(z) \cdot [a_o + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}] = A \cdot T_S^2 \cdot x(z), \tag{5.4}$$

где

$$a_o = 1 + 2\zeta\omega_o \cdot T_S + \omega_o^2 \cdot T_S^2;$$

$$a_1 = -2(1 + \zeta\omega_o \cdot T_S); \tag{5.5}$$

$$a_2 = 1.$$

Дискретная передаточная функция фильтра определяется из уравнения (5.4):

$$G(z) = \frac{y(z)}{x(z)} = \frac{A \cdot T_S^2}{a_o + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}, \tag{5.6}$$

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)
title('А Ч Х   и   Ф Ч Х  дискретного фильтра')
```

Результат приведен на рис. 5.17.

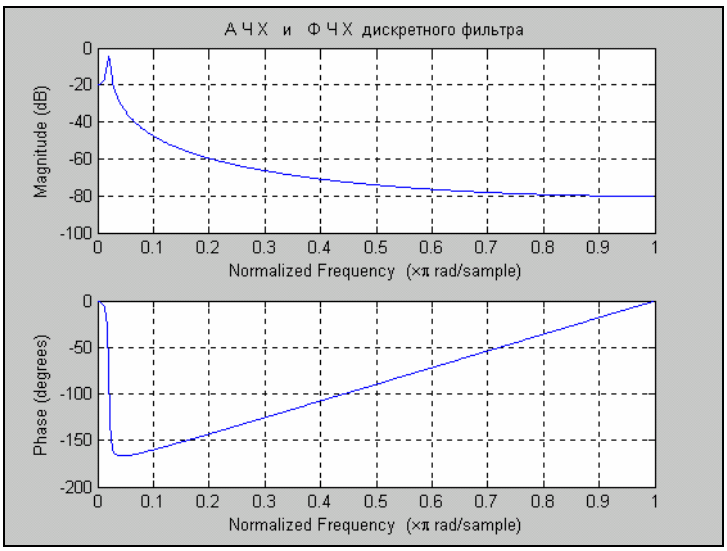


Рис. 5. 17. Результат действия процедуры FREQZ

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 коэффициентов числителя дискретной передаточной функции (5.7) фильтра; ***a*** - вектор коэффициентов знаменателя этой функции.

В качестве примера рассмотрим такую задачу. Пусть требуется получить достаточно верную информацию о некотором «полезном» сигнале, имеющем синусоидальную форму с известным периодом 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 (рис. 5.18):

```
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,'FontSize',12)

title('"Полезный" процесс ');    xlabel('Время (с)');
ylabel('Yp(t)')
```

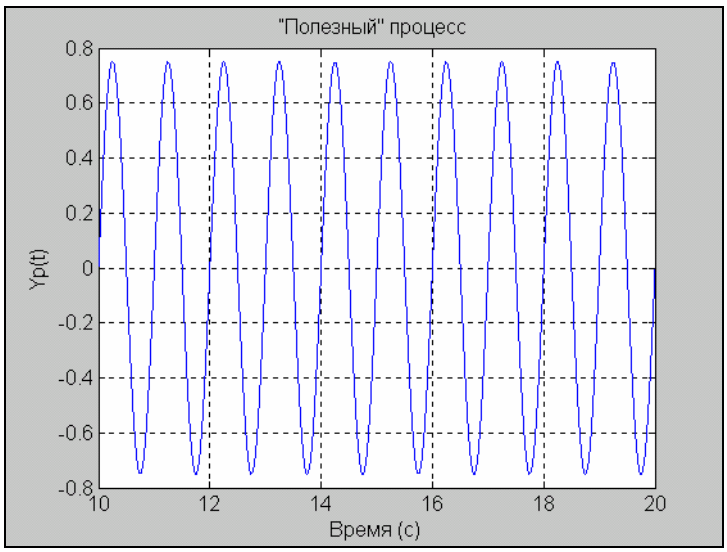


Рис. 5. 18. Вид полезного сигнала

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,'FontSize',12),
title('Входной процесс ');       xlabel('Время (c)');
ylabel('X(t)')
```

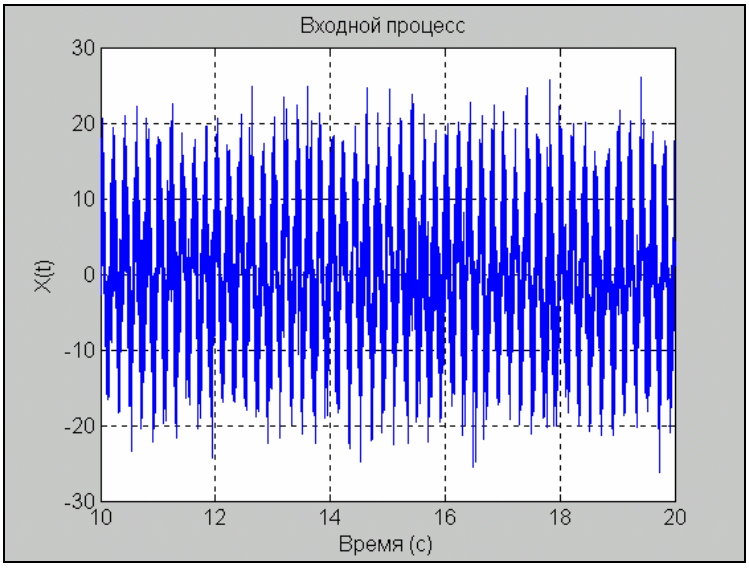


Рис. 5. 19. Вид измеренного сигнала

Требуется так обработать измеренные данные ***x***, чтобы восстановить по ним полезный процесс как можно точнее.

Так как частота полезного сигнала заранее известна, восстановление его можно осуществить при помощи резонансного фильтра отмеченного выше вида. При этом необходимо создать такой фильтр, чтобы период его собственных колебаний $T_ф$ был равен периоду колебаний полезного сигнала ($T_ф = T_1$). Для того, чтобы после прохождения через такой фильтр амплитуда восстановленного сигнала совпадала с амплитудой полезного сигнала,

нужно входной сигнал фильтра умножить на постоянную величину $2\zeta\omega_o^2$ (ибо при резонансе амплитуда выходного сигнала «уменьшается» именно во столько раз по сравнению с амплитудой входного сигнала).

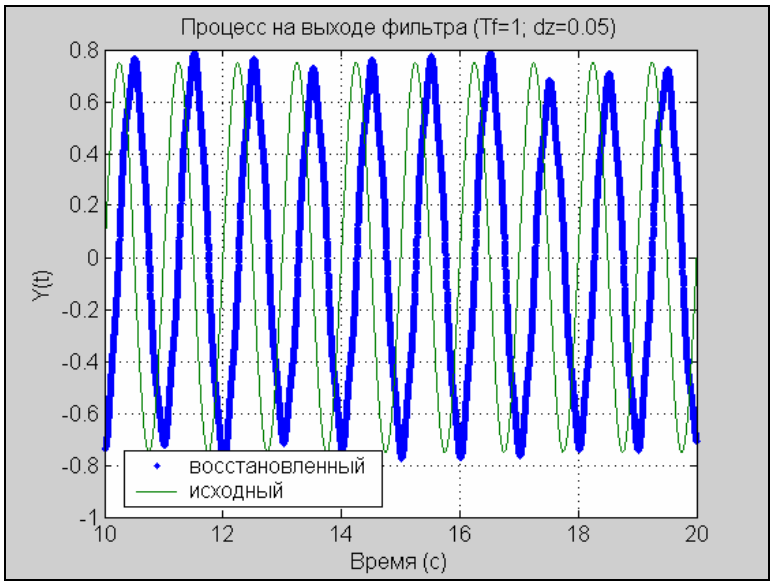


Рис. 5. 20. Результат фильтрации функцией FILTER

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,'FontSize',12)
title('Процесс на выходе фильтра (Tf=1; dz=0.05)');
xlabel('Время (c)');             ylabel('Y(t)')
legend('восстановленный','исходный',0)
```

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,'FontSize',12)
title('Процесс на выходе фильтра (Tf=1; dz=0.05)');
xlabel('Время (c)');             ylabel('Y(t)')
legend('восстановленный','исходный',0)
```

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.

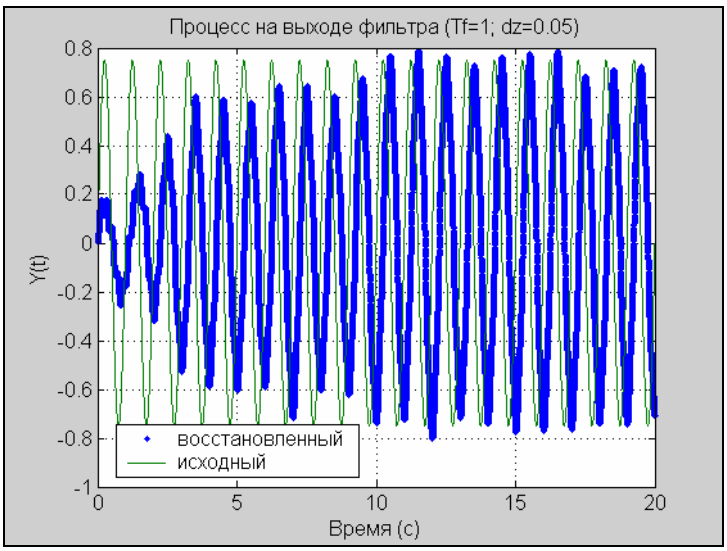


Рис. 5. 21. Переходной процесс фильтра

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 $\boldsymbol{x}$ 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,'FontSize',12)
title('Применение процедуры FILTFILT');
xlabel('Время (с)');              ylabel('Y(t)')
legend('восстановленный','исходный',0)
```

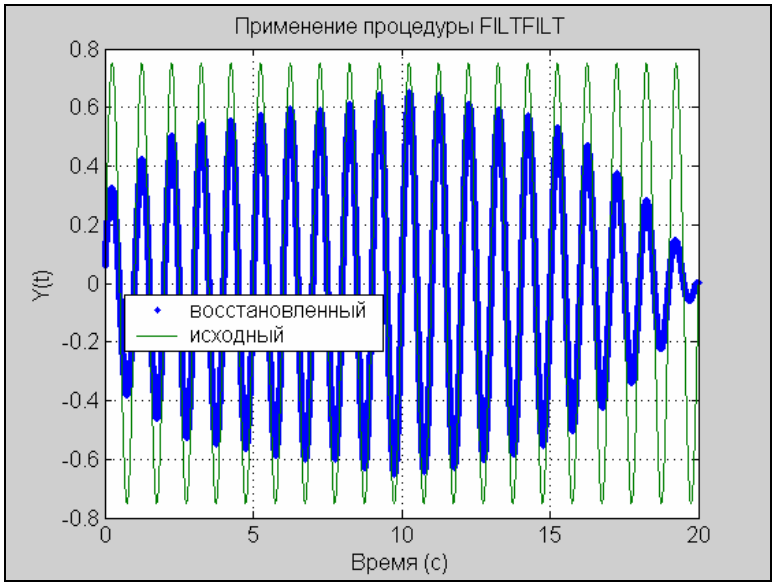


Рис. 5. 22. Результат применения процедуры FILTFILT

5.2.2. Формирование случайных процессов

В соответствии с теорией, сформировать случайный процесс с заданной корреляционной функцией можно, если вначале сформировать нормально (по гауссовому закону) распределенный случайный процесс, 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.

Гауссовый случайный процесс 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,'FontSize',12)
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.

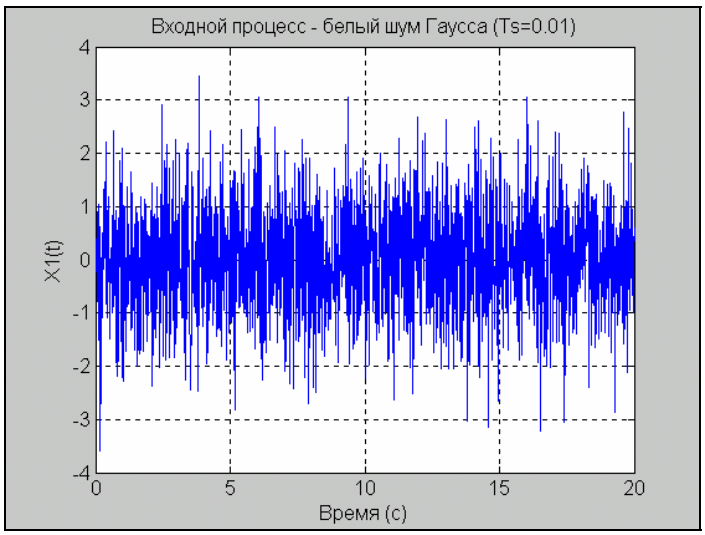


Рис. 5. 23. Нормально распределенный случайный процесс с Ts=0.01 с

Для другого значения дискрета времени (Ts=0.001с), повторяя аналогичные операции, получим процесс $x_2(t)$, изображенный на рис. 5.24.

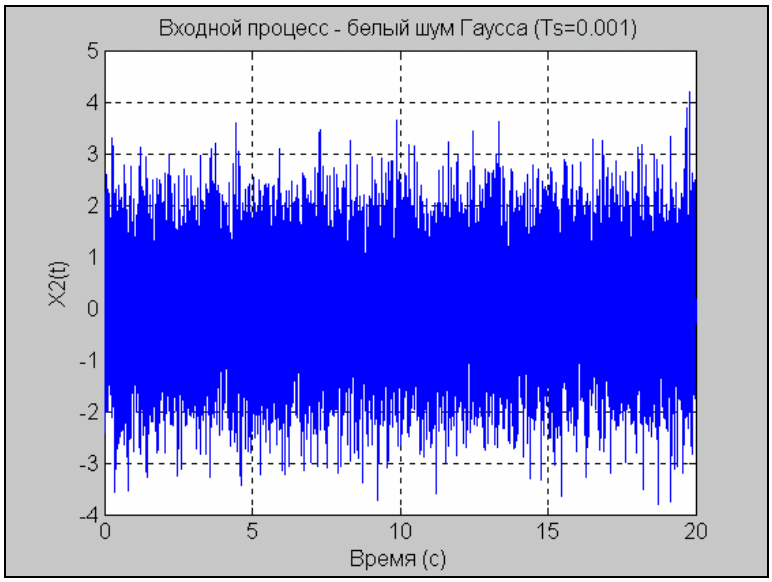


Рис. 5. 24. Нормально распределенный случайный процесс с Ts=0.001 с

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,'FontSize',12)
title('Процесс на выходе фильтра (T0=1; dz=0.05, Ts= 0.01)');
xlabel('Время (c)');            ylabel('Y1(t)')
```

Результат представлен на рис. 5.25.

Рис. 5. 25. Случайный процесс на выходе динамического фильтра (Ts=0.01 с)

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;
y2=filter(b,a,x2); t=0 : Ts : 20;
plot(t,y2),grid, set(gca,'FontSize',12)
title('Процесс на выходе фильтра (T0=1;dz=0.05,Ts= 0.001)');
xlabel('Час (c)');          ylabel('Y2(t)')
```

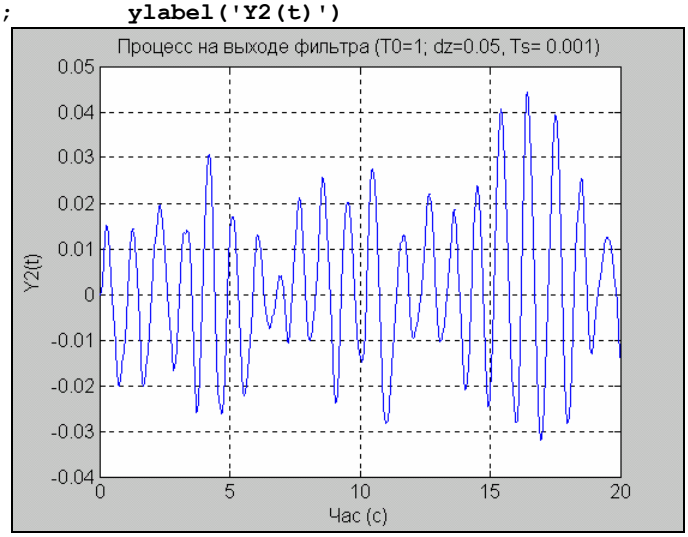


Рис. 5. 26. Случайный процесс на выходе динамического фильтра (Ts=0.001 с)

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

Чтобы убедиться, что качества проверяемого (разработанного) фильтра соответсвуют ожидаемым, следует проанализировать свойства профильтрованного сигнала. Ознакомимся с основными процедурами спектрального и статистического анализа, для этого предназначенными. 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*). К задачам статистического анализа двух процессов обычно относят определение характеристик связи этих процессов таких, как взаимная корреляционная функция и связанная с ней взаимная спектральная плотность.

5.3.1. Основы спектрального и статистического анализа

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

$$f_m = m \cdot f \, (m = 0,1,2,...) \qquad (5.10)$$

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

$$\mathrm{Re}\{X(-m)\} = \mathrm{Re}\{X(m)\}; \mathrm{Im}\{X(-m)\} = -\mathrm{Im}\{X(m)\}, \qquad (5.13)$$

т. е. *действительная часть спектра является четной функцией частоты*, а *мнимая часть спектра - нечетной функцией частоты*.

Разложения (5.12) и (5.8) позволяют рассматривать совокупность комплексных амплитуд (5.9) как изображение периодического процесса в частотной области. Желание распространить такой подход на произвольные, 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, если 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:

$$X(k) = Ts \cdot \sum_{m=1}^{n} x(m) \cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} . \qquad (5.18)$$

Как было отмечено в разделе 1.4.5 (формулы (2) и (3)), процедуры MatLAB **fft** и **ifft** осуществляют вычисления в соответствии с формулами:

$$y(k) = \sum_{m=1}^{n} x(m) \cdot e^{-j\cdot 2\pi\cdot(m-1)\cdot(k-1)/n} ; \qquad (5.19)$$

$$x(m) = \frac{1}{n} \sum_{k=1}^{n} y(k) \cdot e^{j\cdot 2\pi\cdot(m-1)\cdot(k-1)/n} \qquad (5.20)$$

соответственно. Сравнивая (5.18) с (5.19), можно сделать вывод, что *процедура* **fft** *находит дискретное Фурье-изображение заданного дискретного во времени процесса* $x(t)$ *, деленное на дискрет времени:*

$$y(k) = \frac{X(k)}{Ts} . \qquad (5.21)$$

Осуществляя аналогичную операцию дискретизации соотношения (5.9) для комплексной амплитуды $X^*(k)$, получим

$$X^*(k) = \frac{Ts}{T} \cdot \sum_{m=1}^{n} x(m) \cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} =$$

$$= \frac{1}{n} \cdot \sum_{m=1}^{n} x(m) \cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(m-1)} = \frac{y(k)}{n} . \qquad (5.22)$$

Из этого следует, что *комплексный спектр разложения стационарного процесса равен деленному 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:

$$R_{12}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{+T/2} x_1(t) \cdot x_2(t+\tau) \cdot dt , \qquad (5.23)$$

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:

$$S_{12}(f) = \int_{-\infty}^{+\infty} R_{12}(\tau) \cdot e^{-j(2\pi f)\tau} d\tau \qquad (5.24)$$

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:

$$R_{12}(l) = \frac{1}{n-l} \sum_{m=1}^{n-l} x_1(m) \cdot x_2(m+l-1) , \qquad (l = 1,2,..n/2); \qquad (5.25)$$

или в несколько более простое соотношение

$$R_{12}(l) = \frac{2}{n}\sum_{m=1}^{n/2} x_1(m)\cdot x_2(m+l-1), \qquad (l = 1,2,..n/2); \tag{5.26}$$

а вместо (5.24) использовать

$$S_{12}(k) = Ts\cdot\sum_{l=1}^{n/2} R_{12}(l)\cdot e^{-j\cdot(2\pi/n)\cdot(k-1)\cdot(l-1)}, \quad (k = 1,2,..n/2). \tag{5.27}$$

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:

$$S_{12}(k) = Ts\cdot\left\{\frac{2}{n}y_2(k)\right\}\cdot\overline{y}_1(k); \qquad (k = 1,2,..n/2), \tag{5.28}$$

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:

Fmax = 1/Ts; (5.30)

- по заданной длительности заданного процесса T рассчитать дискрет частоты df по формуле:

df =1/T; (5.31)

- по вычисленным данным сформировать вектор значений частот, в которых будет вычислено Фурье-изображение.

Последнее проще (но не наиболее правильно) сделать таким образом:

```
f1=0 : df : Fmax.
```
(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

f = -Fmax/2 : df : Fmax/2. (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,'FontSize',12),
title('Процесс из одиночного прямоугольного импульса ');
xlabel('Время (с)');              ylabel('Y(t)')
```

Результат показан на рис. 5.27.

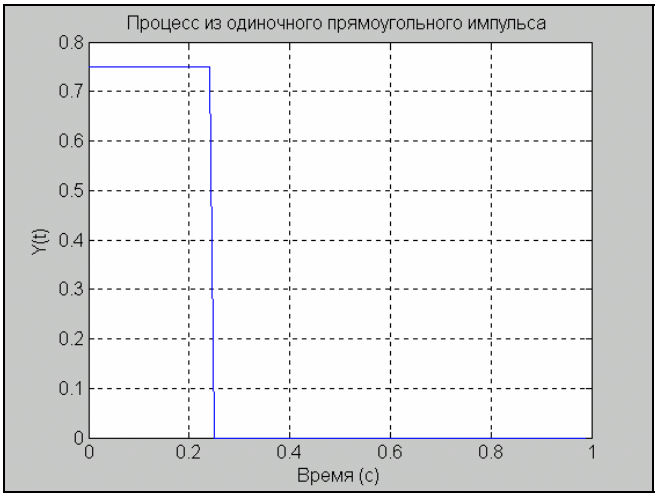


Рис. 5. 27. Одиночный прямоугольный импульс

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,'FontSize',12),
title('Модуль FFT-преобразования прямоугольного импульса ');
xlabel('Частота (Гц)');    ylabel('Модуль')
```

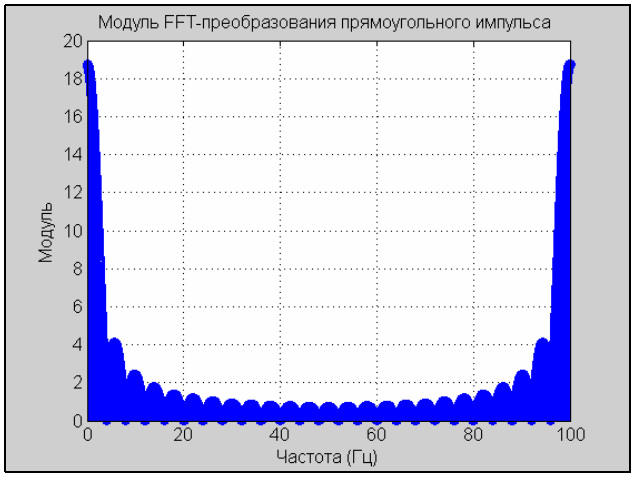


Рис. 5. 28. Модуль FFT-преобразования прямоугольного импульса

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,'FontSize',12),
```

```
title('Модуль Фурье-изображения прямоугольного импульса ');
xlabel('Частота (Гц)');   ylabel('Модуль')
```

Получим результат, приведенный на рис. 5.29.

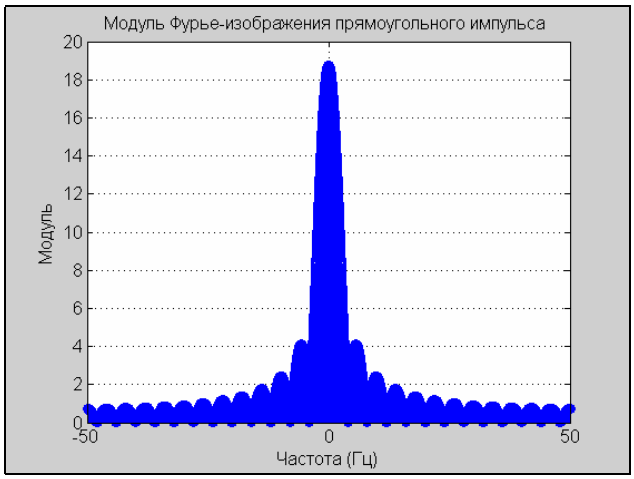


Рис. 5. 29. Модуль Фурье-изображения прямоугольного импульса

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,'FontSize',12),
title('Фурье-изображение прямоугольного импульса ');
ylabel('Действит. и Мнимая части'),
xlabel('Частота (Гц)');
legend('Действительная','Мнимая',0)
```

Они представлены на рис. 5.30.

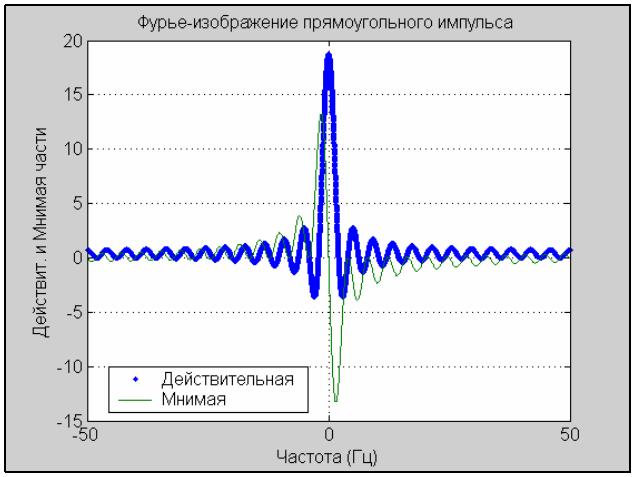


Рис. 5. 30. Действительная и мнимая части Фурье-изображения прямоугольного импульса

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,'FontSize',12),
title(' Трехчастотный полигармонический процесс ');
xlabel('Время (с)');      ylabel('Y(t)')
```

График процесса показан на рис. 5.31.

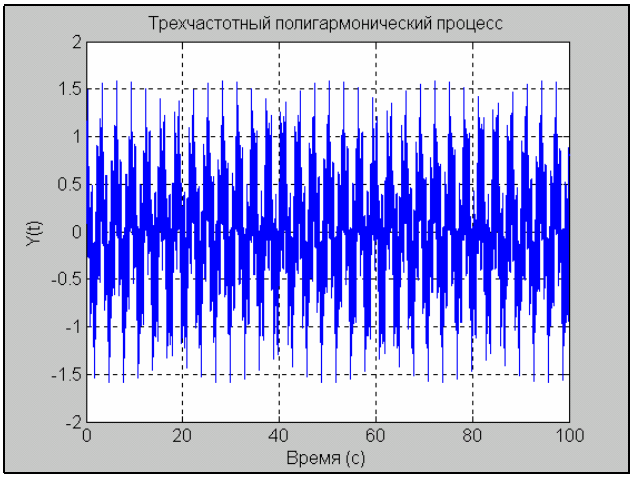


Рис. 5. 31. График трехчастотного полигармонического процесса

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,'FontSize',12),
title('Модуль          Фурье-изображения          полигармонического          процесса');
xlabel('Частота (Гц)');   ylabel('Модуль')
```

Результат представлен на рис. 5.32.

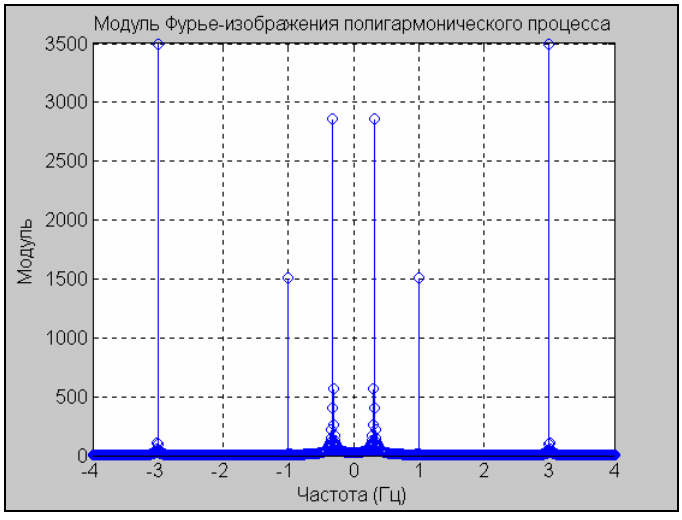


Рис. 5. 32. Модуль Фурье-изображения полигармонического процесса при Ts=0.01 с

Если изменить дискрет времени на Ts=0.02, получим результат, изображенный на рис. 5.33.

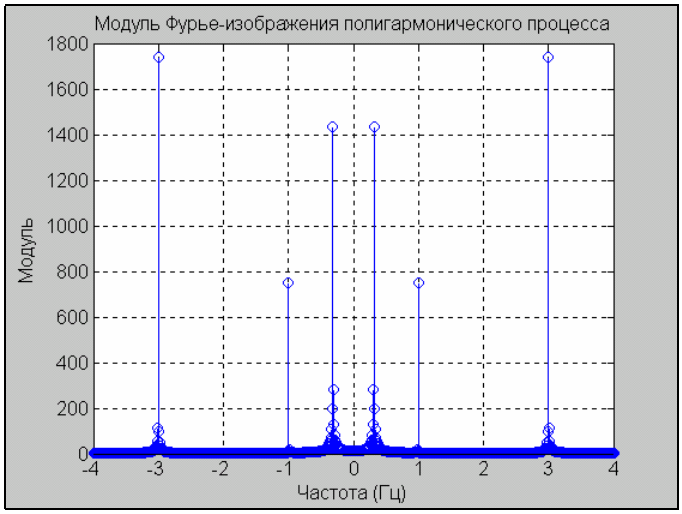


Рис. 5. 33. Модуль Фурье-изображения полигармонического процесса при Ts=0.02 с

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.

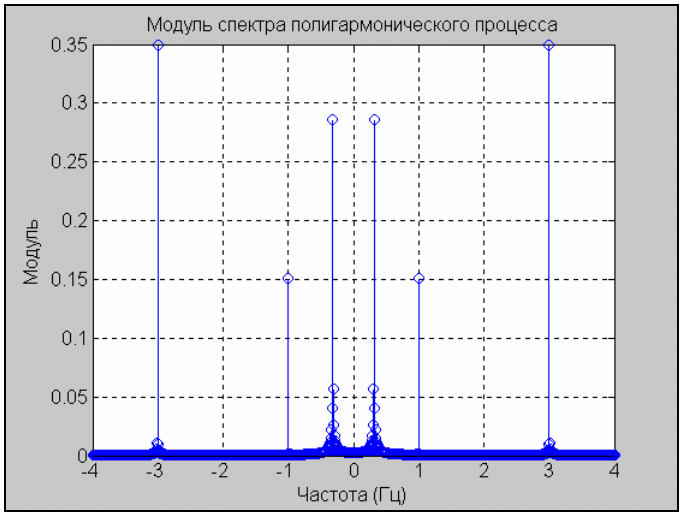


Рис. 5. 34. Модуль спектра полигармонического процесса

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,'FontSize',12),
title('Комплексный спектр полигармонических колебаний');
ylabel('Действит. часть');
subplot(2,1,2) ,            plot(f(s1:s2),mch(s1:s2)), grid,
set(gca,'FontSize',12),
xlabel('Частота (Гц)');          ylabel('Мнимая часть')
```

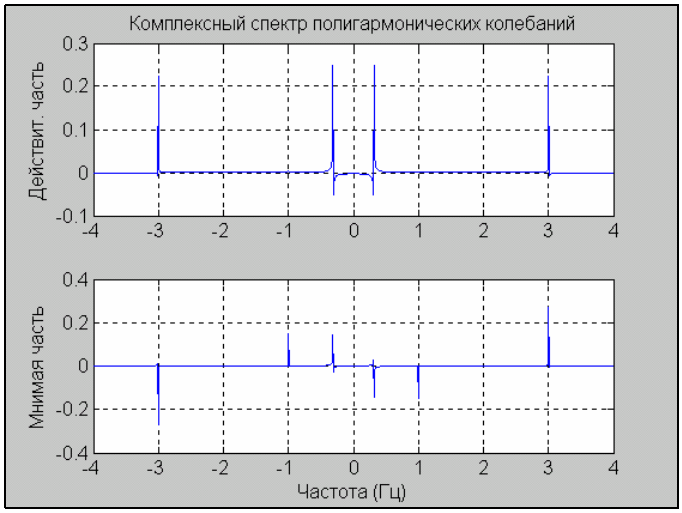


Рис. 5. 35. Комплексный спектр полигармонических колебаний

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). Аналогично тому, как это было сделано в п. 5.2.2., сформируем процесс в виде белого гауссового шума с шагом во времени 0.01 и длительностью в 100 с , создадим формирующий фильтр, «пропустим» через него белый шум и результат выведем на рис. 5.36:

```
Ts=0.01;     T=100; t=0 : Ts : T;
x1=randn(1,length(t));
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,x1);
plot(t,y1),grid, set(gca,'FontSize',12)
title('Процесс на выходе фильтра (T=1; dz=0.05, Ts= 0.01)');
xlabel('Время (c)');             ylabel('Y1(t)')
```

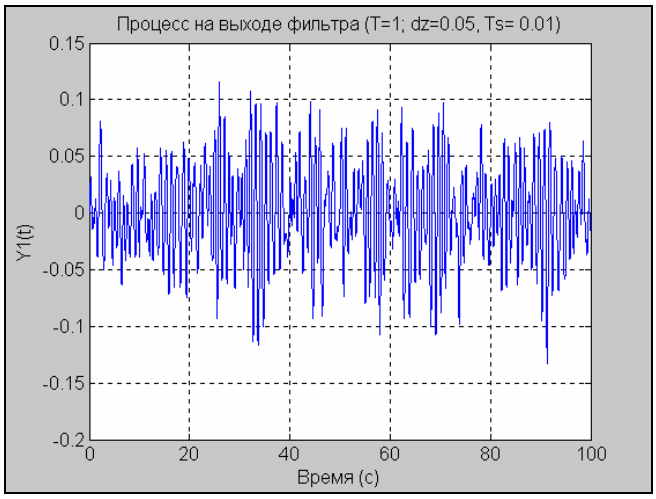


Рис. 5. 36. Случайный нормальный процесс с преобладающей частотой 1 Гц

Вычислим Фурье-изображение (ФИ) для процесса-шума с учетом замечания, сделанного для установившихся процессов и построим на рис. 5.37 графики модуля ФИ и спектральной плотности мощности (СПМ):

```
%  Формирование  массива частот
df = 1/T;     Fmax = 1/Ts; f = - Fmax/2 : df : Fmax/2;
dovg = length(f);
%  Расчет скорректированных массивов Фурье-изображений
Fu1 = fft(x1)/dovg;  Fu2 = fft(y1)/dovg;
Fu1p = fftshift(Fu1);Fu2p = fftshift(Fu2);
% Формирование массивов модулей ФИ
A1 = abs(Fu1p);            A2 = abs(Fu2p);
% Вычисление Спектральных Плотностей Мощности
S1 = Fu1p.*conj(Fu1p)*dovg;      S2 = Fu2p.*conj(Fu2p)*dovg;
```

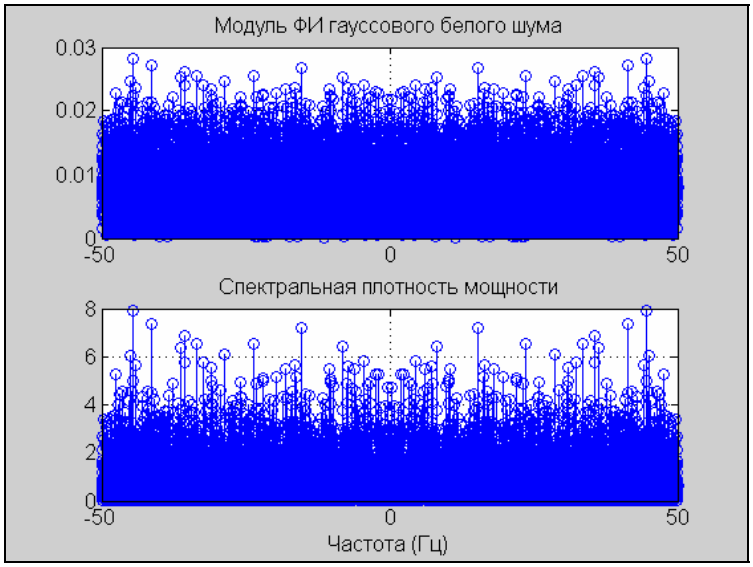


Рис. 5. 37. Модуль Фурье-изображения и СПМ белого шума

```
%  Вывод графиков белого шума
subplot(2,1,1);            stem(f,A1),grid,
set(gca,'FontSize',12)
title('Модуль ФИ гауссового белого шума');
subplot(2,1,2);            stem(f,S1),grid,
set(gca,'FontSize',12)
title('Спектральная плотность мощности');
xlabel('Частота (Гц)');
```

Рассматривая рис. 5.37, 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. 5.38):

```
%  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,'FontSize',12)
title('Модуль ФИ случайного стационарного процесса');
subplot(2,1,2);     stem(f(c1:c2),S2(c1:c2)),grid,
set(gca,'FontSize',12)
title('Спектральная плотность мощности');
xlabel('Частота (Гц)');
```

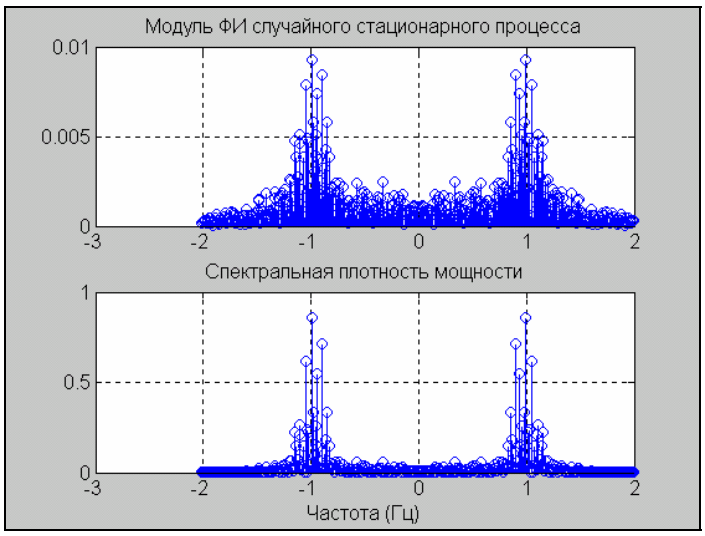


Рис. 5. 38. Модуль ФИ и СПМ нормального случайного процесса с преобладающей частотой 1 Гц

Проводя эти вычисления еще раз с новой длительностью процесса T=20 с, 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 отдельная 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 с преобладающей частотой 1 Гц:

```
[C,f] = psd(y1,dovg,Fmax);
stem(f(1:200),C(1:200)),grid,
set(gca,'FontSize',12)
title('       Спектральная плотность мощности');
xlabel('Частота (Гц)');
```

В результате получим картину, представленную на рис. 5.39

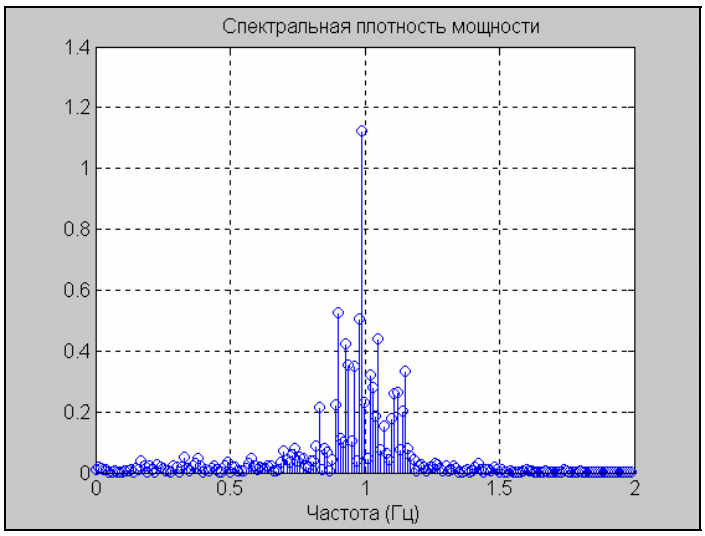


Рис. 5. 39. СПМ, полученная применением процедуры PSD

Если ту же процедуру вызвать *без указания выходных величин*, то результатом ее выполнения станет *выведение графика СП от частоты*.

Например, обращение

```
psd(y1, dovg, Fmax)
```

приведет к построению в графическом окне (фигуре) графика рис. 5.40. При этом значения СП будут откладываться в логарифмическом масштабе в децибелах.

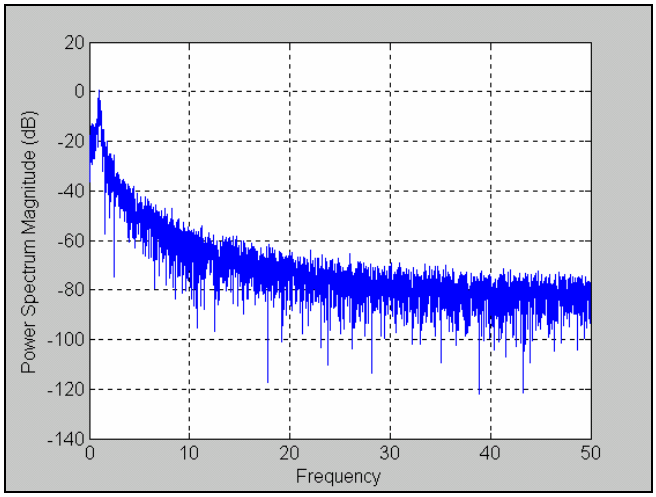


Рис. 5. 40. График СПМ, полученный процедурой PSD

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, 'FontSize', 12)
title('АКФ случайного процесса') ,      xlabel('Запаздывание (c)')
```

На рис. 5.41 представлен результат применения процедуры **xcorr.**

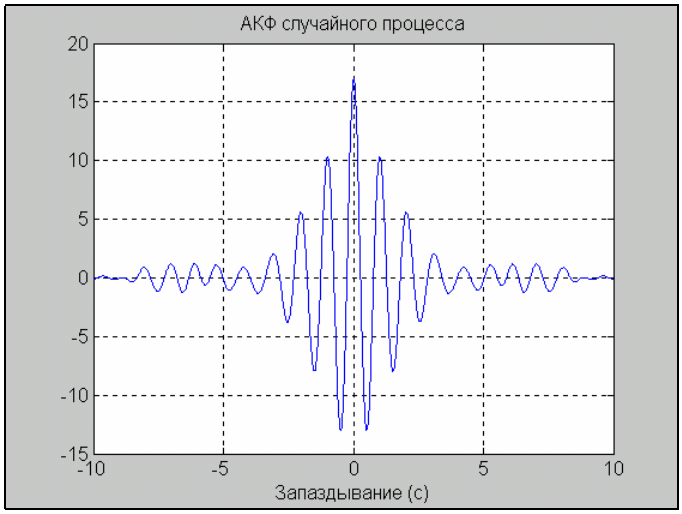


Рис. 5. 41. Корреляционная функция случайного процесса, полученная процедурой XCORR

5.4. Проектирование фильтров

Теперь перейдем к изучению процедур, помогающих разработать фильтр с заданными свойствами.

5.4.1. Формы представления фильтров и их преобразования

Фильтр как звено системы автоматического управления может быть представлен в нескольких эквивалентных формах, каждая из которых полностью описывает его:

- в форме рациональной ***передаточной функции*** (***tf*** - *представление)*; если звено является непрерывным (аналоговым), то оно описывается непрерывной передаточной функцией

$$W(s)=\frac{b(s)}{a(s)}=\frac{b(1)\cdot s^{m}+b(2)\cdot s^{m-1}+\ldots+b(m+1)}{a(1)\cdot s^{n}+a(2)\cdot s^{n-1}+\ldots+a(n+1)}, \tag{5.34}$$

а в случае дискретного фильтра последний может быть представлен дискретной передаточной функцией вида

$$W(z)=\frac{b(z)}{a(z)}=\frac{b(1)+b(2)\cdot z^{-1}+\ldots+b(m+1)\cdot z^{-m}}{a(1)+a(2)\cdot z^{-1}+\ldots+a(n+1)\cdot z^{-n}}; \tag{5.35}$$

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):

$$\frac{b(z)}{a(z)}=\frac{r(1)}{1-p(1)\cdot z^{-1}}+\ldots+\frac{r(n)}{1-p(n)\cdot z^{-1}}+k(1)+k(2)\cdot z^{-1}+\ldots$$
$$\ldots+k(m-n+1)\cdot z^{-(m-n)}; \tag{5.36}$$

в этой форме звено описывается тремя векторами: вектором-столбцом ***r*** вычетов передаточной функции, вектором столбцом ***p*** полюсов и вектором-строкой ***k*** коэффициентов целой части дробно-рациональной функции;

- в каскадной форме (***sos*** - *представление)*, когда передаточная функция звена представлена в виде произведения передаточных функций не выше второго порядка:

$$H(z)=\prod_{k=1}^{L}H_k(z)=\prod_{k=1}^{L}\frac{b_{ok}+b_{1k}\cdot z^{-1}+b_{2k}\cdot z^{-2}}{a_{ok}+a_{1k}\cdot z^{-1}+a_{2k}\cdot z^{-2}}; \tag{5.37}$$

параметры каскадного представления задаются в виде матрицы ***sos***, содержащей вещественные коэффициенты:

$$sos=\begin{bmatrix} b_{o1}b_{11}b_{21}a_{o1}a_{11}a_{21} \\ b_{o2}b_{12}b_{22}a_{o2}a_{12}a_{22} \\ \ldots\ldots\ldots\ldots \\ b_{oL}b_{1L}b_{2L}a_{oL}a_{1L}a_{2L} \end{bmatrix}; \tag{5.38}$$

- в пространстве состояний (***ss*** *-представление)*, т. е. с помощью уравнений звена в форме

$$\begin{aligned} \dot{x} &= A\cdot x + B\cdot u \\ y &= C\cdot x + D\cdot u \end{aligned}; \tag{5.39}$$

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)*:

$$W(s)=k\frac{[s-z(1)]\cdot[s-z(2)]\cdots[s-z(m)]}{[s-p(1)]\cdot[s-p(2)]\cdots[s-p(n)]}; \tag{5.40}$$

- 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 проверить, что все элементы вектора k по абсолютной величине не превышают единицы.

Обратная задача *вычисления коэффициентов полинома по коэффициентам решетчатого представления* решается путем применения функции **rc2poly**:

```
a = rc2poly(k).
```

2. Процедура **sos2ss** *определяет матрицы (5.39) A, B. C и D, описывающие звено в пространстве состояний, по заданной матрице SOS каскадной формы (5.38)*:

```
[A,B,C,D] = sos2ss (SOS).
```

Элементы матрицы SOS должны быть вещественными.

Обратный переход осуществляется при помощи функции **ss2sos**

```
SOS = ss2sos(A,B,C,D)
```

3. Функция **sos2zp** дает возможность определить (5.40) векторы *z* нулей, *p* - полюсов и коэффициент усиления *k* звена, заданного каскадной формой передаточной функции (т. е. матрицей SOS (5.38)):

```
[z,p,k] = sos2zp (SOS).
```

Обратный переход осуществляется при помощи функции **zp2sos**

```
SOS = zp2sos(z,p,k).
```

4. *Нахождение нулей* z, *полюсов* p *и коэффициента усиления* k звена по его описанию в пространстве состояний можно произвести путем обращения к процедуре **ss2zp** по форме

```
[z,p,k] = ss2zp (A,B,C,D, iu),
```

где iu - номер входа, по которому ищется передаточная функция.

Обратное преобразование осуществляется процедурой **zp2ss**:

```
[A,B,C,D] = zp2ss (z,p,k).
```

5.4.2. Разработка аналоговых фильтров

Цель разработки фильтров заключается в обеспечении частотно-зависи-мого 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

$$\tau = -\frac{d\vartheta(\omega)}{d\omega} \quad . \tag{5.44}$$

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 $\pm z_i$.

Идеальный фильтр низких частот (ФНЧ) пропускает только низкочастотные составляющие. Его характеристика затухания имеет вид, показанный на рис. 5.42*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. 5.42 *б, в* и *г*.

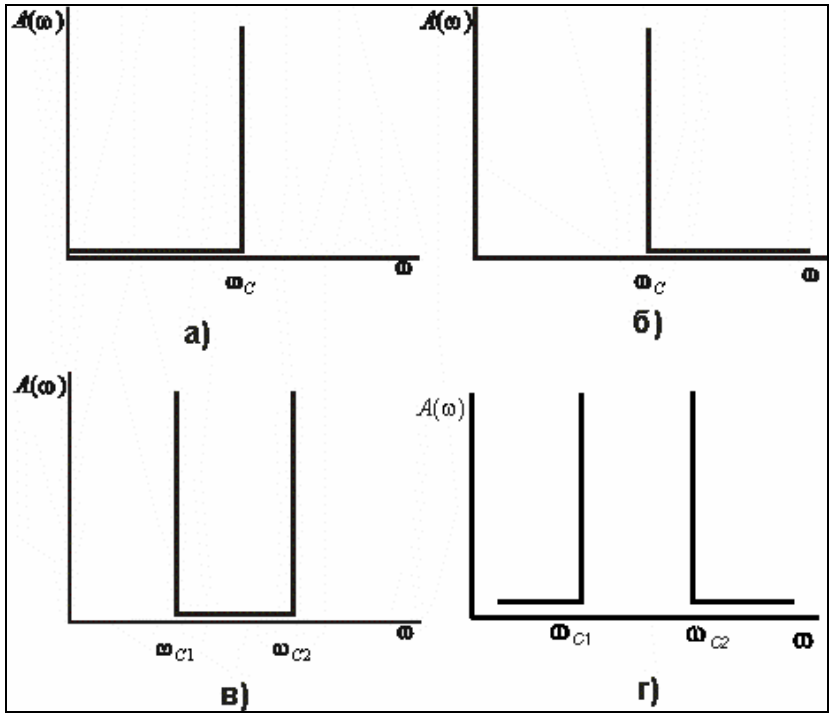


Рис. 5. 42. Характеристики затухания идеальных фильтров

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. 5.43*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 $\omega\_$ 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. 5.43 *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.

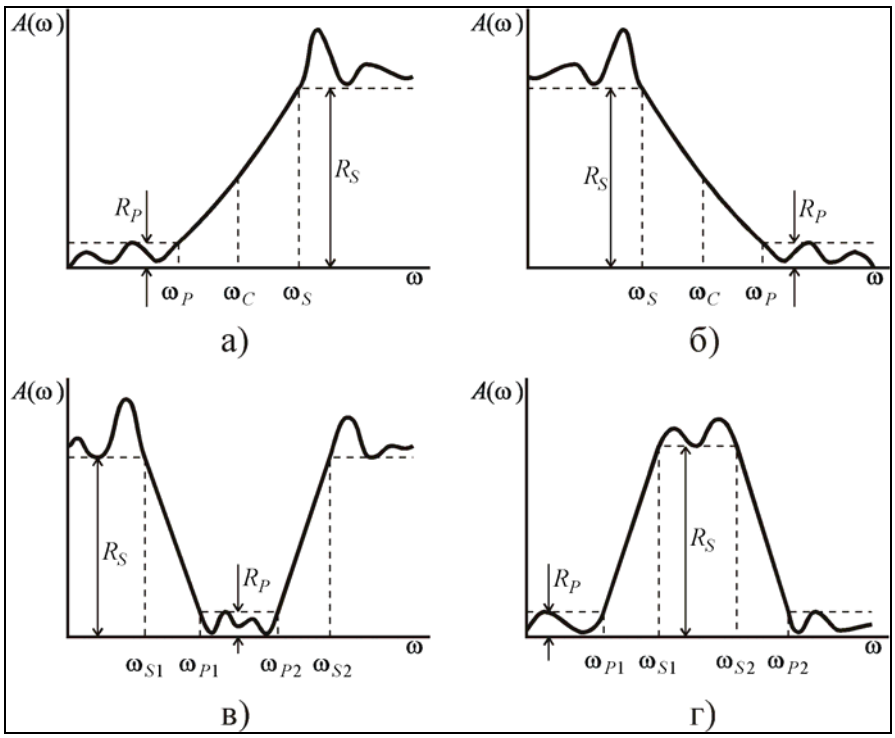


Рис. 5. 43. Характеристики затухания реальных фильтров

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 фильтра по заданным аналогичным параметрам аналогового прототипа. И, наконец, третья форма определяет матрицы дискретного пространства состояний фильтра по известным матрицам непрерывного пространства состояний.

Второй способ построения цифрового фильтра по его аналоговому прототипу заключается в таком преобразовании параметров аналогового фильтра в параметры дискретного фильтра, при котором импульсная характеристика последнего совпадала бы с импульсной характеристикой аналогового фильтра в точках через дискрет по времени. Это в MatLAB осуществляется применением процедуры **impinvar**:

```
[bz, az] = impinvar(b, a,  Fs).
```

Здесь b и a - заданные векторы коэффициентов числителя и знаменателя передаточной функции аналогового прототипа фильтра, bz и az - вычисляемые коэффициенты числителя и знаменателя дискретной передаточной функции дискретного фильтра, Fs - заданная частота дискретизации сигнала в герцах. Если параметр Fs при обращении не указан, то по умолчанию он принимается равным 1 Гц.

Третий способ формирования дискретных фильтров - использование ранее рассмотренных процедур формирования фильтров **butter, cheby1, cheby2** и **ellip**. Если при обращении к этим процедурам не указывать в конце списка входных параметров «флажка» **'s'**, то результатом работы этих процедур будут параметры именно цифровых фильтров.

Основное отличие применения этих функций для разработки цифровых фильтров заключается в другом представлении задаваемых частот в векторе Wc. *Все частоты должны задаваться по отношению к так называемой "частоте Найквиста". Частотой Найквиста называют половину частоты дискретизации сигнала.*

Так как диапазон частот изменения дискретного сигнала всегда меньше частоты дискретизации, то все частотные характеристики дискретных фильтров определяются только внутри диапазона от 0 до частоты Найквиста. Поэтому все задаваемые в векторе Wc граничные частоты должны быть меньше единицы.

Существуют еще две процедуры расчета БИХ-фильтров.

Процедура **maxflat** производит расчет обобщенного цифрового фильтра Баттерворта. Формы обращения к ней таковы:

```
[ b, a] = maxflat(nb, na, Wc)

[ b, a] = maxflat(nb, 'sym', Wc)

[ b, a, b1, b2] = maxflat(nb,na, Wc)

[ b, a] = maxflat(nb,na, Wc, 'design_flag').
```

Первое обращение позволяет вычислить коэффициенты b - числителя и a -знаменателя дискретной передаточной функции H(z) цифрового ФНЧ Баттерворта с частотой среза Wc, порядок числителя которой равен nb, а знаменателя - na.

При обращении второго вида вычисляются коэффициенты цифрового симметричного КИХ-фильтра Баттерворта. В этом случае na принимается равным 0. Параметр nb должен быть четным.

Если обратиться к процедуре так, как указано в третьем обращении, т. е. указать в качестве выходных четыре величины, то дополнительные параметры b1 и b2 определят коэффициенты двух полиномов, произведение ко-

торых 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 = 0.6;
>> [b,a,b1,b2] = maxflat(nb,na,w,'trace')
```

```
Table:

   L         M         N         wo_min/pi wo_max/pi


 Columns 1 through 4
         10            0            2            0
          9            1            2      0.23938
          8            2            2      0.32591
          7            3            2       0.3991
          6            4            2      0.46688
          5            5            2      0.53312
          4            6            2       0.6009
          3            7            2      0.67409
          2            8            2      0.76062
 Column 5
      0.23938
      0.32591
       0.3991
      0.46688
      0.53312
       0.6009
      0.67409
      0.76062
            1
b =
      0.11478      0.50222      0.81309      0.54459
     0.070336    -0.052259    0.0040606    0.0050236
   -0.0022396   0.00042191 -3.2299e-005
a =   1      0.46247      0.53752
b1 =  1     5    10    10     5     1
b2 =
      0.11478    -0.071679     0.023681   -0.0048336
    0.0005834 -3.2299e-005
```

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 =   0.11478      0.50222      0.81309      0.54459
     0.070336    -0.052259    0.0040606    0.0050236
   -0.0022396   0.00042191 -3.2299e-005
a =   1      0.46247      0.53752
b1 =  1     5    10    10     5     1
b2 =  0.11478    -0.071679     0.023681   -0.0048336
    0.0005834 -3.2299e-005
```

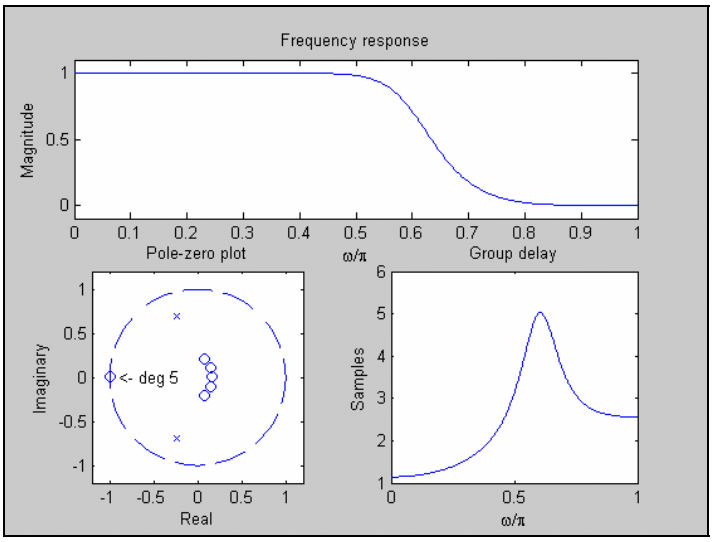


Рис. 5. 44. Результат выполнения процедуры MAXFLAT(nb,na,w,'plots')

Расчет БИХ-фильтра по заданной амплитудно-частотной характеристике производится процедурой **yulewalk**. Если набрать в командном окне MatLAB строку

```
[ b, a] = yulewalk(n, f, m),
```

то будут вычислены коэффициенты b - числителя и a - знаменателя дискретной передаточной функции БИХ-фильтра порядка n, АЧХ которого задана векторами f (частоты в нормированных значениях) и m - соответствующих значений отношений амплитуд выхода и входа. Первый элемент вектора f должен быть равен 0, а последний 1. Все остальные элементы должны быть расположены в неубывающем порядке. Частоты, при которых происходит скачок АЧХ, указываются два раза с разными значениями соответствующих им «амплитуд».

Приведем пример расчета ФНЧ 8-го порядка и построим желаемую АЧХ и АЧХ полученного фильтра. Результат приведен на рис. 5.45.

```
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))
set(gca,'FontSize',12)
grid, title('Пример применения процедуры YULEWALK')
xlabel('Нормализованная частота'),     ylabel('A Ч X')
```

Рис. 5. 45. Пример применения процедуры YULEWALK

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 Блекмана;

`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.

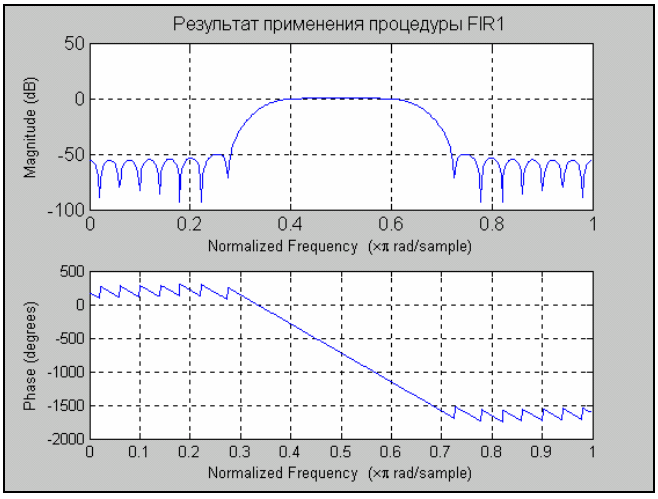


Рис. 5. 46. Результат применения процедуры FIR1

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)
set(gca,'FontSize',12)
title('Результат применения процедуры FIR1')
```

Результат показан на рис.5.46.

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
set(gca,'FontSize',12)
title('АЧХ КИХ-фильтра  (процедура FIR2)')
xlabel('Нормализованная частота'),     ylabel('А Ч Х')
```

На рис. 5.47 приведены желаемая АЧХ и АЧХ, полученная в результате расчета.

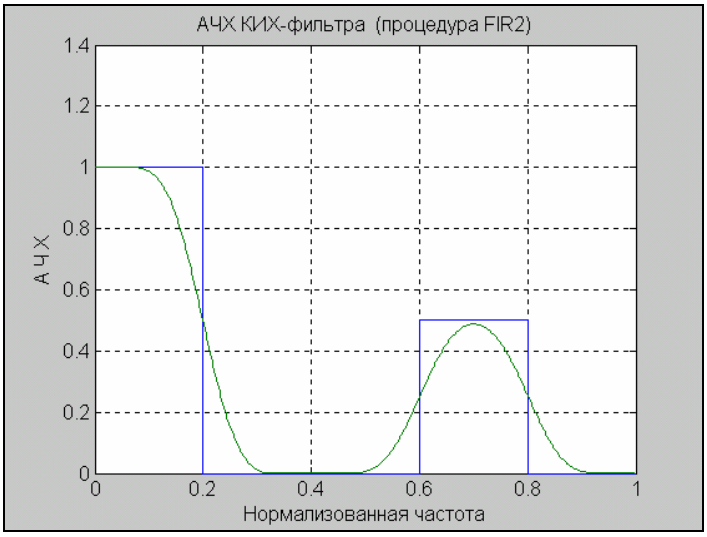


Рис. 5. 47. АЧХ КИХ-фильтра (процедура FIR2)

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 =
 -7.4344e-005   -0.0030717    0.022633    0.010051
    0.0059551    0.001063   -0.010522   -0.023061
   -0.062583    0.0089569 -6.0306e-005   -0.014469
     0.17747      0.11938     0.12718      0.3023
     0.12718      0.11938     0.17747   -0.014469
 -6.0306e-005    0.0089569   -0.062583   -0.023061
   -0.010522    0.001063    0.0059551    0.010051
    0.022633   -0.0030717 -7.4344e-005
```

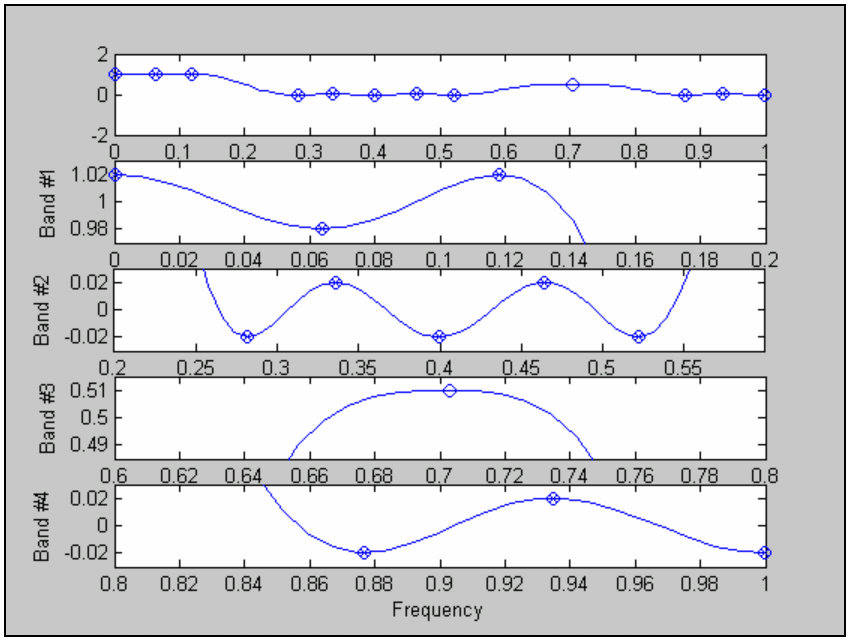


Рис. 5. 48. Результат применения процедуры FIRCLS

Для сравнения с результатами работы процедуры **fir2** построим график полученной АЧХ, аналогичный приведенному на рис. 5.47:

```
[h,w] = freqz(b,1,512);
```

```
plot(w/pi,abs(h)), grid
set(gca,'FontSize',12)
title('АЧХ КИХ-фильтра  (процедура FIRCLS)')
xlabel('Нормализованная частота'),     ylabel('А Ч Х')
```

Результат представлен на рис. 5.49.

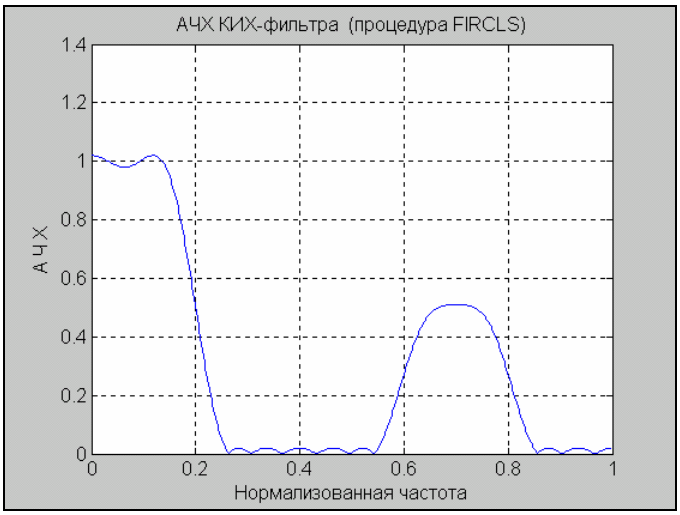


Рис. 5. 49. АЧХ КИХ-фильтра (процедура FIRCLS)

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, записанных в возрастающем порядке 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 соответ-

ствующими 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 (а, 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
set(gca,'FontSize',12)
title('АЧХ КИХ-фильтра  (процедура REMEZ)')
xlabel('Нормализованная частота'),  ylabel('A Ч X')
```

Результат приведен на рис.5.50.

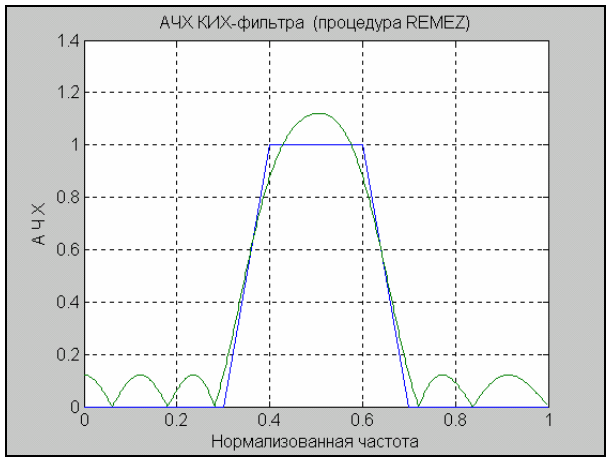


Рис. 5. 50. АЧХ КИХ-фильтра (процедура REMEZ)

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)
set(gca,'FontSize',12)
title('АЧХ КИХ-фильтра  (процедура CREMEZ)')
```

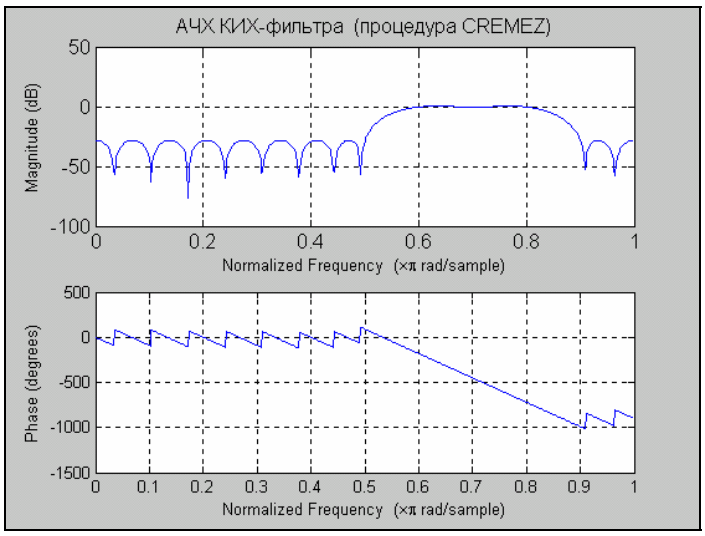


Рис. 5. 51. Результат применения процедуры CREMEZ

5.5. Графические и интерактивные средства

Важным инструментарием моделирования процесса фильтрации является наглядное графическое представление как характеристик сигналов, так и динамических характеристик фильтров. Рассмотрим процедуры пакета SIGNAL, осуществляющие такое представление.

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, а функции **`freqz`** - на рис. 5.17. 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 логарифмическом масштабе, а во втором - в виде отношения к частоте Найквиста, в равномерном масштабе и в диапазоне от 0 до 1;

- форма оформления графиков достаточно жесткая и не предусматривает возможности изменения размеров графиков, надписей по осям и вывода заголовка.

Некоторые процедуры расчета фильтров, такие как **`fircls, fircls1, cremez`** и **`maxflat`** предусматривают выведение соответствующих графических изображений некоторых параметров спроектированного фильтра, если в качестве последнего входного параметра при обращении к процедуре указан флаг **`'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(10,2,0.6,'plots')
```

приводит к появлению в графическом окне изображения, показанного на рис. 5.52.

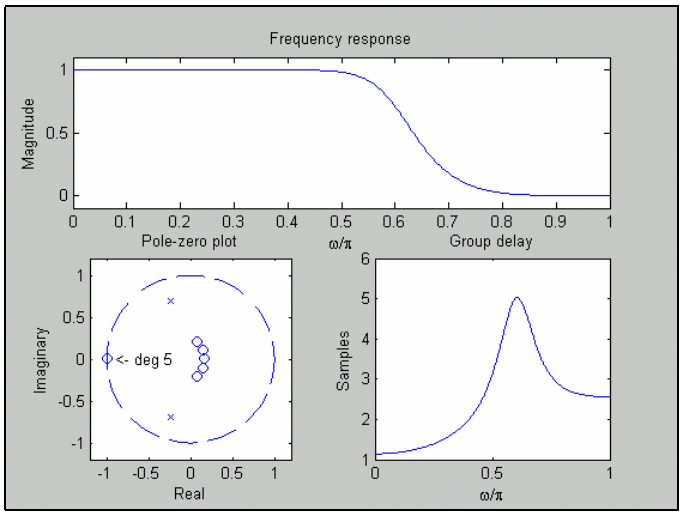


Рис. 5. 52. Графическое окно функции MAXFLAT

При вызове функции **`fircls`** с этим флагом на график выводятся фрагменты АЧХ с максимальными отклонениями от требуемой АЧХ (см. рис. 5.53):

```
n= 30;f = [0 0.2 0.6 0.8 1];     amp = [1 0 0.5 0];
up = [1.02 0.02 0.51 0.02]; lo = [0.98 -0.02 0.49 -0.02 ];
fircls(n,f,amp,up,lo,'plots');
```

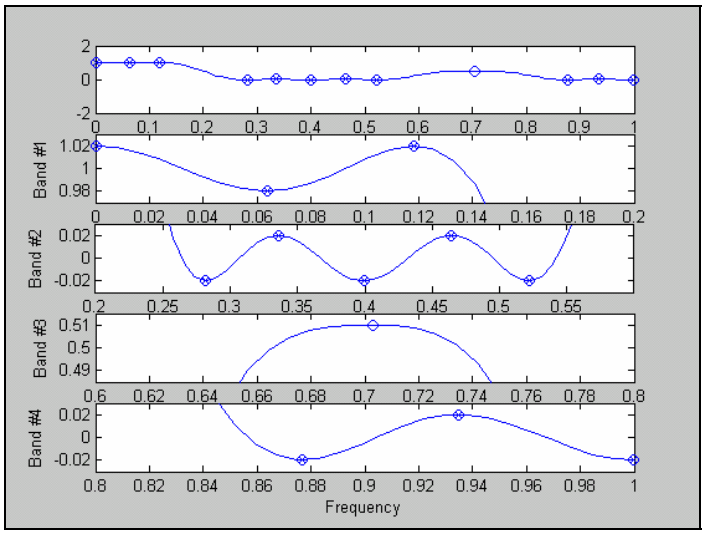


Рис. 5. 53. Графическое окно функции FIRCLS

Аналогичные графики строятся и при вызове функции **`fircls1`**. Отличие в том, что теперь графики не орнаментированы никаким текстом (рис. 5.54):

```
fircls1(n,0.5,0.01,0.01,'plots');
```

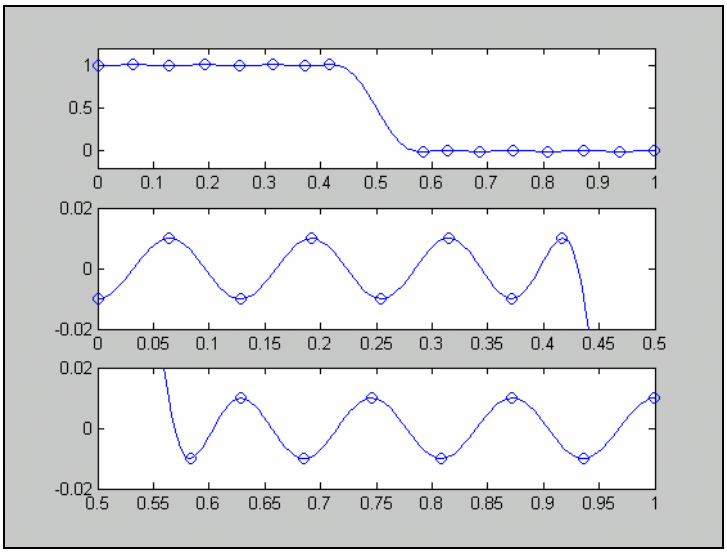


Рис. 5. 54. Графическое окно функции FIRCLS1

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. 5.55:

```
cremez(30,[0 0.5 0.6 0.8 0.9 1],'bandpass','plots');
```

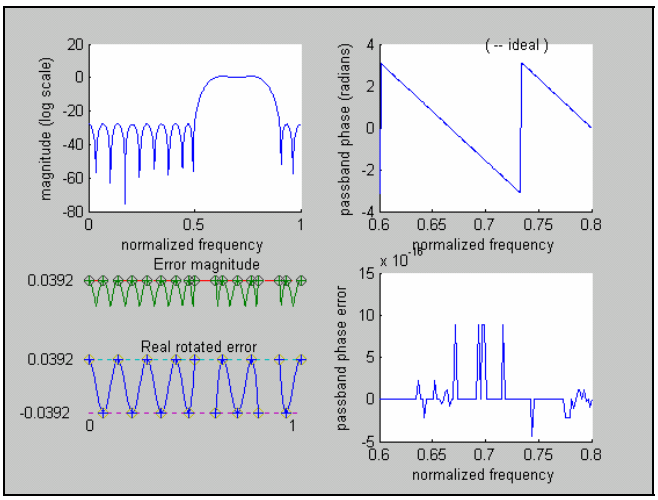


Рис. 5. 55. Графическое окно функции CREMEZ

В пакете SIGNAL имеются еще три 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.

Рассмотрим в качестве примера применение этих процедур к БИХ-фильтру, созданному процедурой **maxflat**:

```
[b,a] = maxflat(10,2,0.6) ;      grpdelay(b,a,128)
```

Результат применения функции **grpdelay** приведен на рис. 5.56.

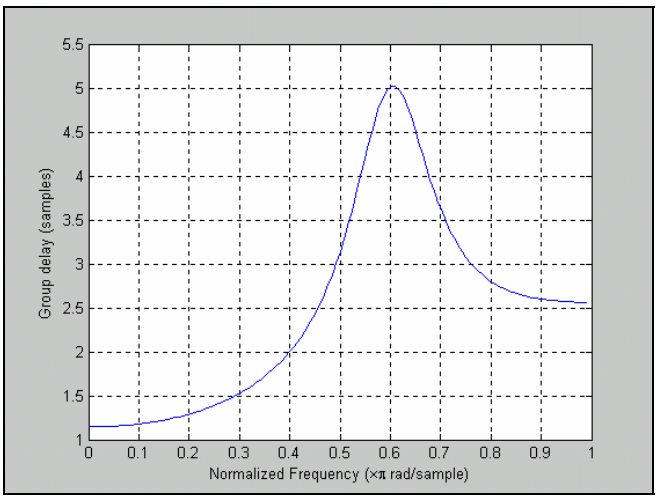


Рис. 5. 56. Результат применения процедуры GRPDELAY

Применяя процедуру **`impz`** к тому же фильтру, получим график импульсной дискретной характеристики фильтра, изображенный на рис. 5.57:

```
impz(b,a)
```

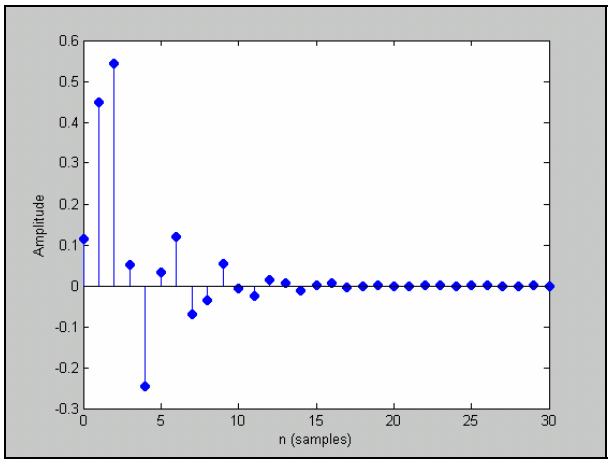


Рис. 5. 57. Результат применения процедуры IMPZ

Использование процедуры **`zplane`** для этого фильтра:

```
zplane(b,a)
```

приводит к построению графика рис. 5.58.

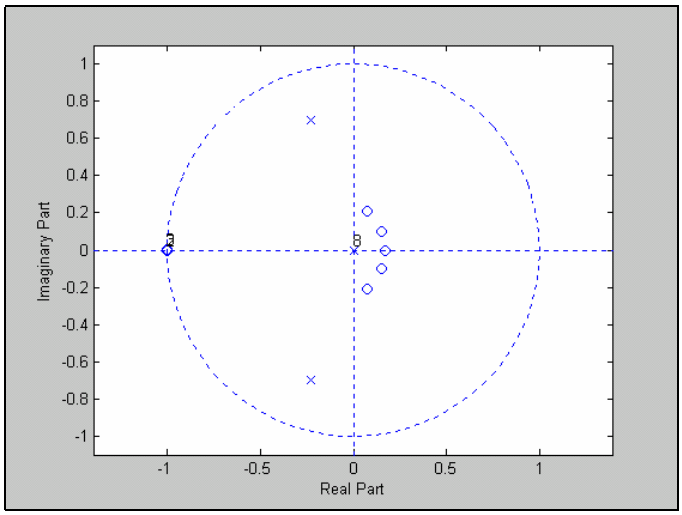


Рис. 5. 58. Результат применения процедуры ZPLANE

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,'FontSize',12)
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,'FontSize',12)
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,'FontSize',12)
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. 59.

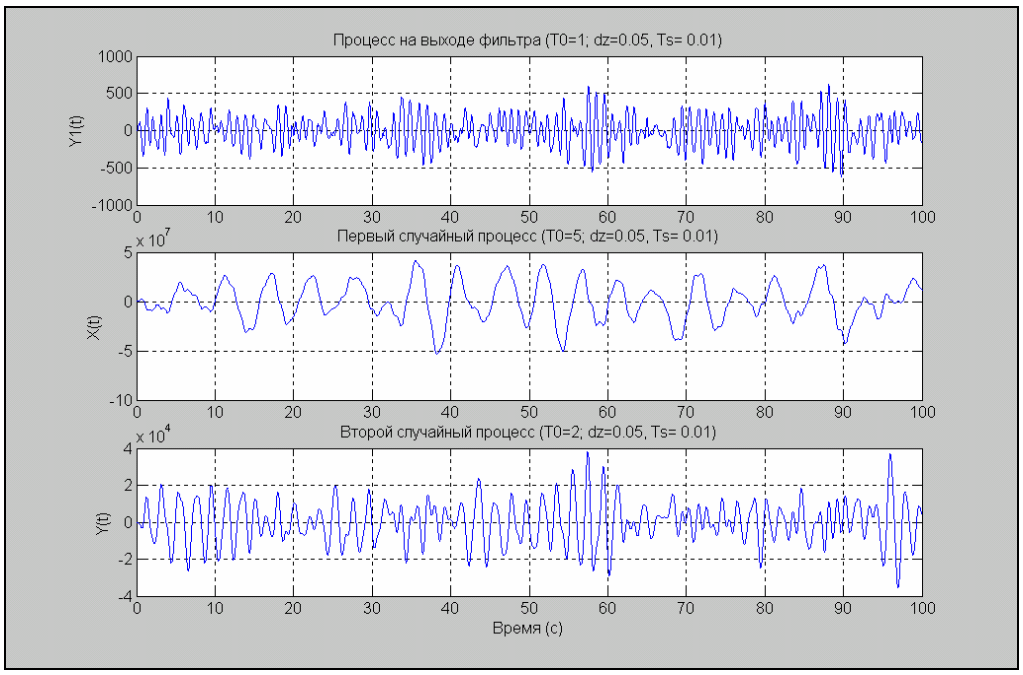


Рис. 5. 59. Графики случайных процессов с различными преобладающими частотами

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 изменения значения функции в каждом фрагменте от -2 до 2:

```
strips(y1,20,100, 2),grid
set(gca,'FontSize',12)
title ('Применение процедуры STRIPS для выведения Y1(t)');
xlabel('Время, с')
```

На рис. 5.60 представлен результат.

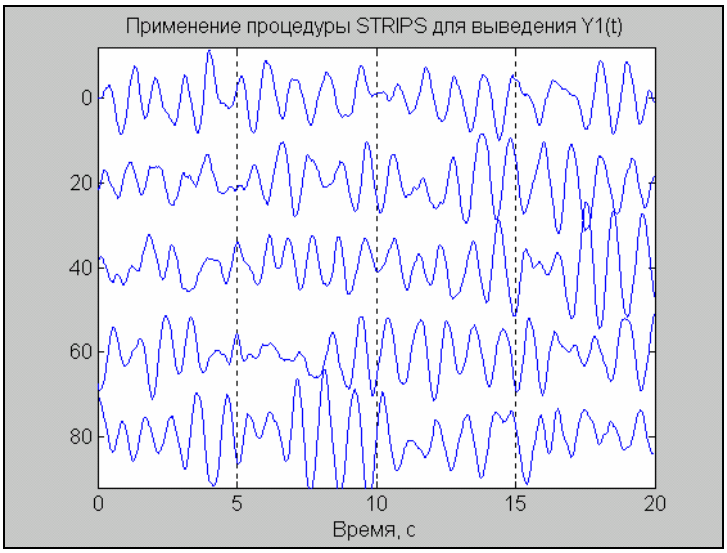


Рис. 5. 60. Применение процедуры STRIPS для выведения графиков

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. 5.61 и 5.62.

```
[Sx,f]=psd(x,10000,100);
plot(f(1:100),Sx(1:100)), grid, set(gca,'FontSize',12)
title (' Применение процедуры PSD к процессу X(t)');
ylabel('Спектральная    плотность'); xlabel('Частота, Гц ')
```

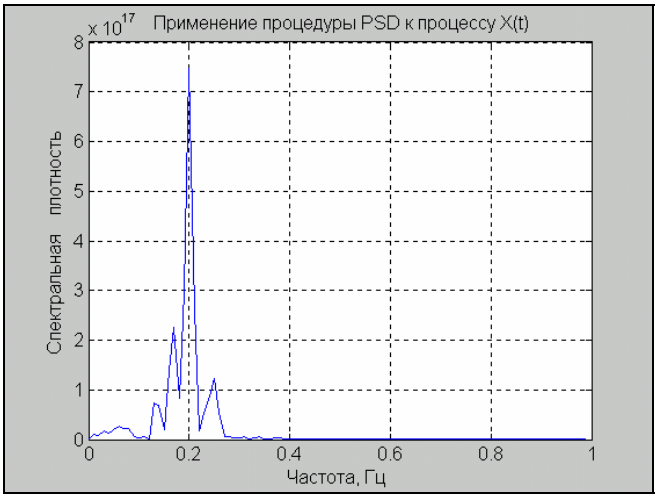


Рис. 5. 61. Применение процедуры PSD к процессу X(t)

```
[Sxy,f]=csd(x,y,10000,100);
plot(f(1:100),abs(Sxy(1:100))), grid, set(gca,'FontSize',12)
title(' Применение процедуры CSD к процессам X(t)  и Y(t)');
ylabel('Модуль взаимной    С  П'); xlabel('Частота, Гц ')
```

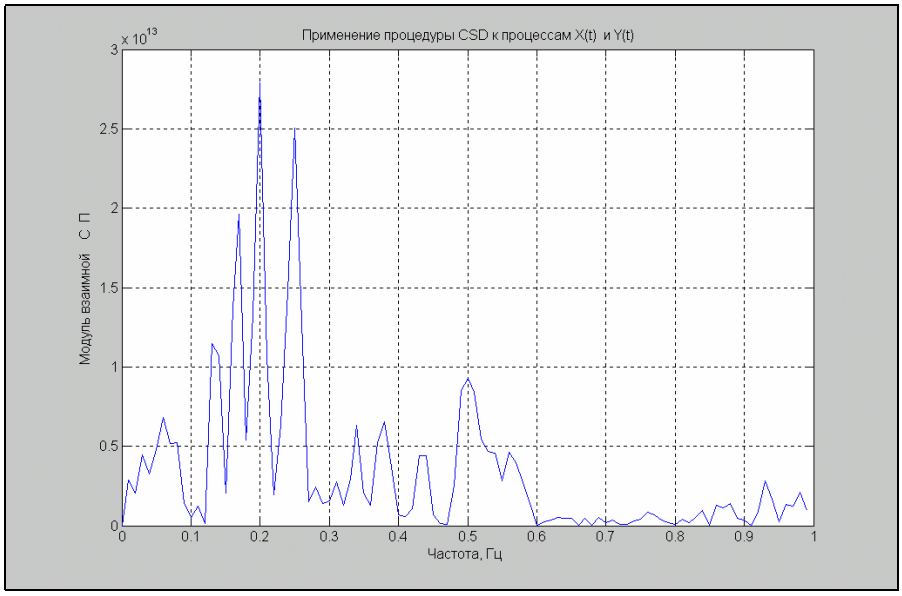


Рис. 5. 62. Применение процедуры CSD к процессам X(t) и Y(t)

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. 5.63:

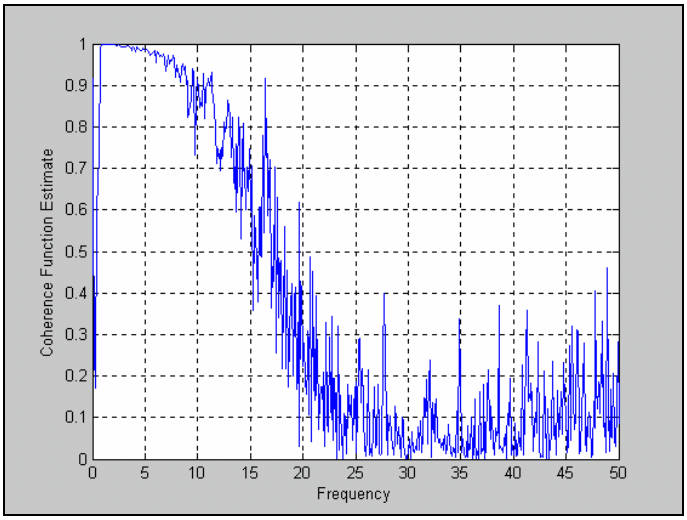


Рис. 5. 63. Применение функции COHERE к процессам X(t) и Y(t)

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:

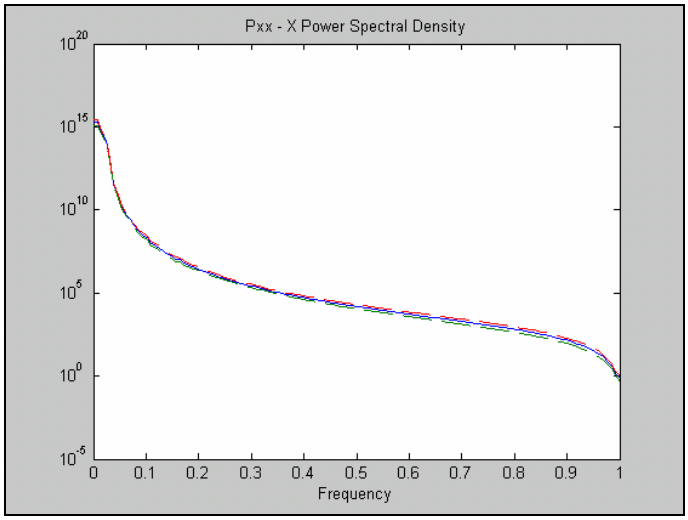


Рис. 5. 64. Оценки осредненного значения СПМ процесса X(t)

- зависимости СПМ первого сигнала от нормализованной частоты (рис. 5.64); 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;

- после нажатия клавиши <Enter> прежние кривые исчезнут и на том же поле появятся три аналогичные кривые (рис. 5.65) для второго процесса Y(t);

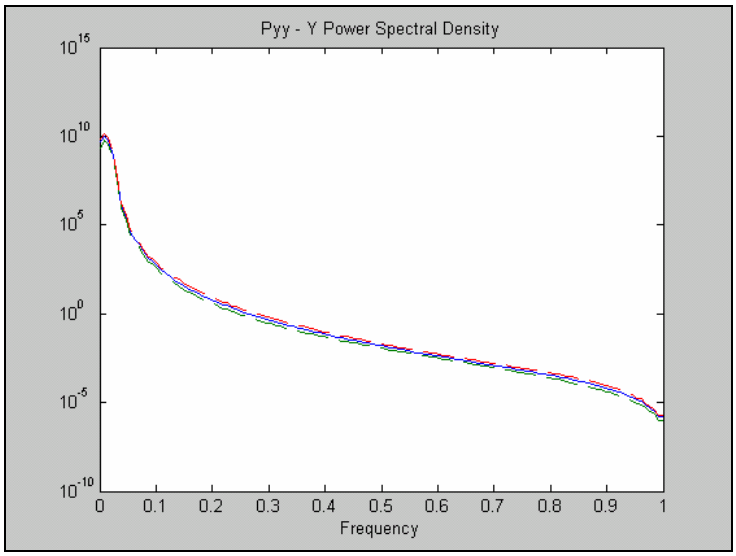


Рис. 5. 65. Оценки осредненного значения СПМ процесса Y(t)

- следующее нажатие <Enter> приведет (рис. 5.66) к появлению кривой зависимости модуля «передаточной функции» взаимной спектральной плотности указанных процессов от частоты;

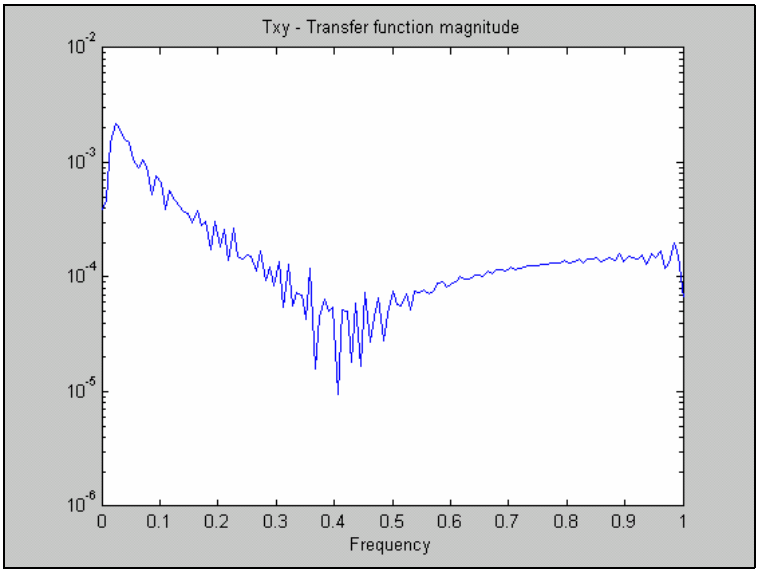


Рис. 5. 66. Модуль «передаточной функции» взаимной спектральной плотности

- дальнейшее нажатие <Enter> приводит к появлению графика зависимости аргумента «передаточной функции» ВСП от частоты (рис. 5.67);

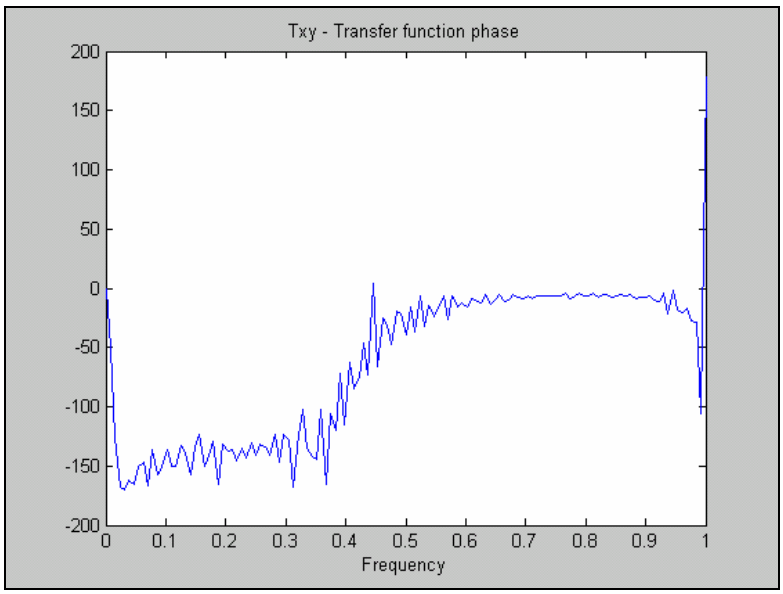


Рис. 5. 67. Аргумент «передаточной функции» взаимной спектральной плотности

- последнее нажатие <Enter> вызовет появление в поле графика функции когерентности (рис. 5.68).

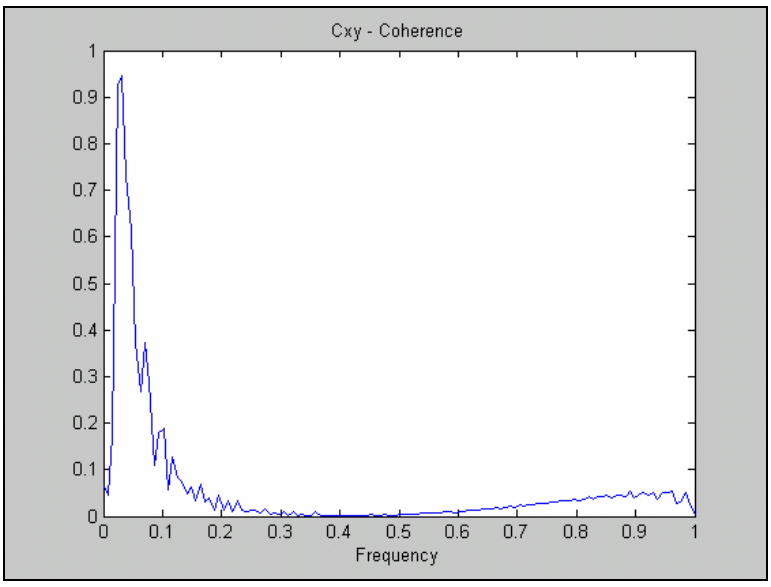


Рис. 5. 68. Функция когерентности

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. 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.

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. 5.69.

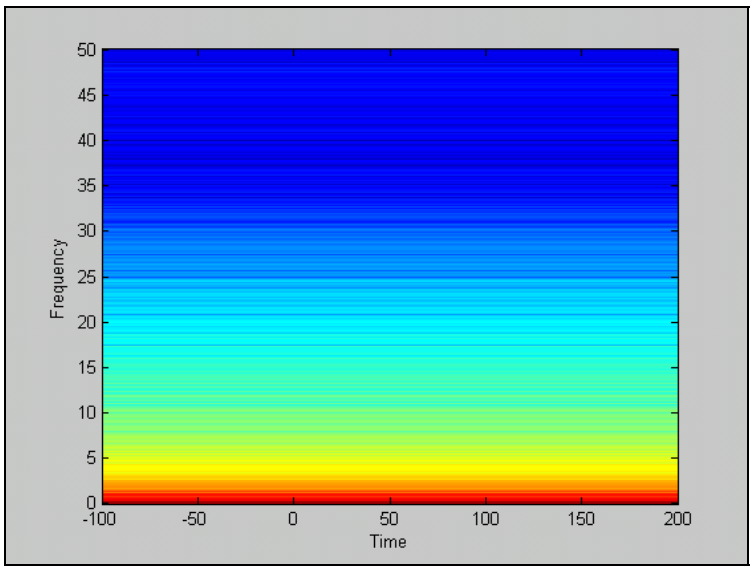


Рис. 5. 69. Спектрограмма процесса X(t)

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.

Применяя процедуру к ранее сформированным процессам X(t) и Y(t):

```
tfe(x, y, 10000,100)
```

получим график рис. 5.70.

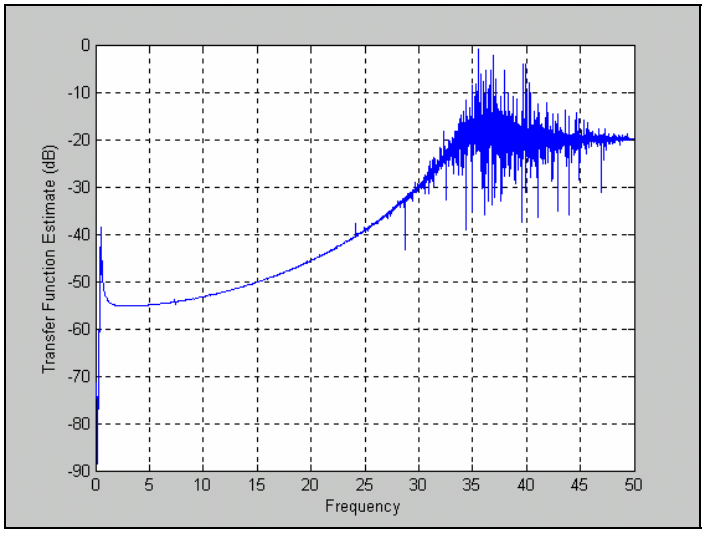


Рис. 5.70. Процедура TFE оценки передаточной функции процессов X(t) и Y(t)

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. 5.71.

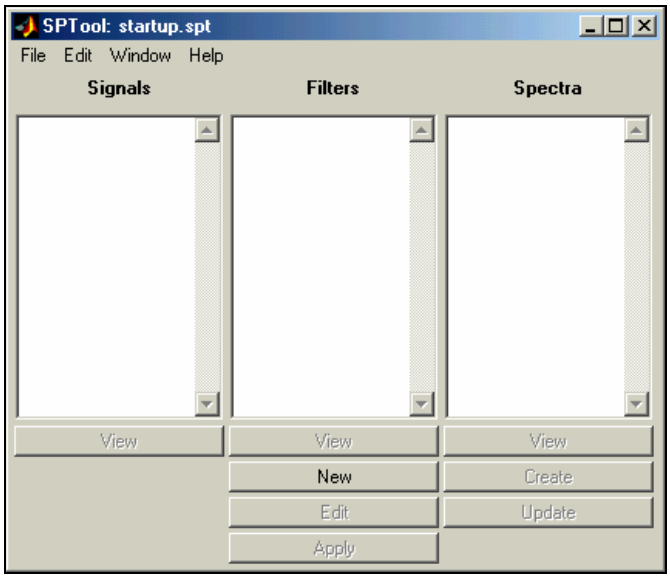


Рис. 5.71. Окно интерактвной среды SPTOOL

Как видим, окно ***SPTool*** состоит из трех полей - *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* - *Новый*);
- отредактированы (надпись *Edit* - *Правка*);
- просмотрены (надпись *View*);
- применены к одному или нескольким объектам, выделенным в окошке *Signals* (надпись *Apply* - *Применить*).

Аналогично, с объектами окошка *Spectra* - (спектрами) можно производить 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 *startup.spt*, все три окошка - пустые, а из команд, расположенных ниже их, активной является только одна *New*. 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 сигнале. 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.***

Для этого в окне ***SPTool*** выберите `File`. В результате появляется список (рис. 5.72), одной из команд которого является `Import.` Выбрав ее, полуяим на экране новое окно ***Import to SPTool***, представленное на рис. 5.73.

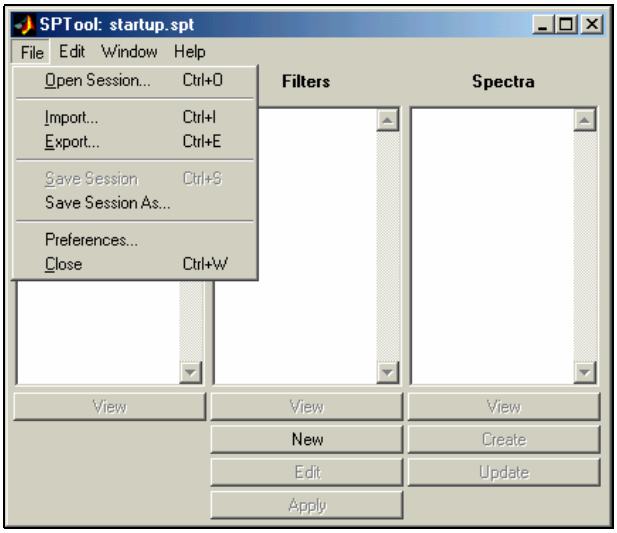


Рис. 5. 72. Команды меню File

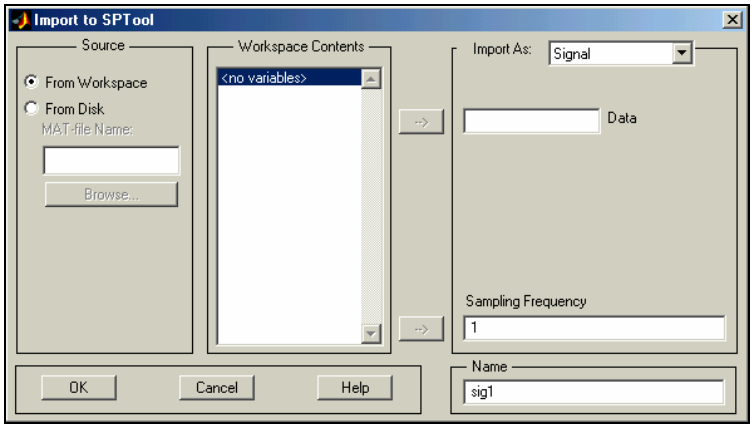


Рис. 5. 73. Окно Import to SPTool

В разделе *Source (Источник)* этого окна отмечен переключатель *From Workspace (Из Рабочего Пространства)*. Это означает что окно настроено на импорт сигналов из рабочего пространства MatLAB. Поэтому все имена переменных рабочего пространства представлены во втором окошке *Workspace Contents (Содержимое Рабочего Пространства)*. В начале сеанса работы это окошко пусто.

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***. В результате изменится и содержимое окошка *Workspace Contents* (рис. 5.74). Внем появится список всех переменных рабочего пространства MatLAB.

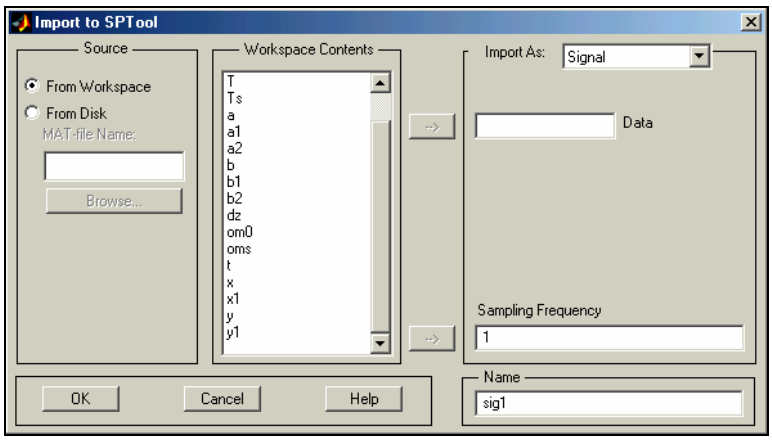


Рис. 5. 74. Окно Import to SPTool при непустом рабочем пространстве

Выбрав в этом списке необходимую переменную, необходимо затем «нажать» кнопку со стрелкой, указывающей на окошко с надписью *Data*. После этого в окошке *Data* должно появиться имя выбранной переменной.

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. 5.75 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)

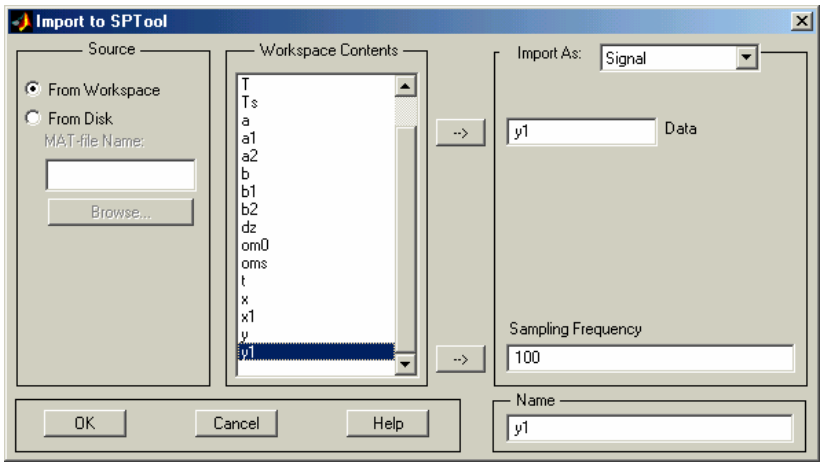


Рис. 5. 75. Импортирование в SPTool сигнала Y1

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 окно ***sptool*** изменит свой вид (рис. 5.76): в окошке `Signals` появится запись имени вектора сигнала, активизируется подпись *View* под этим окошком и станет активной команда *Create* под окошком `Spectra`. Это означает, что можно находить спектральные характеристики импортированного сигнала.

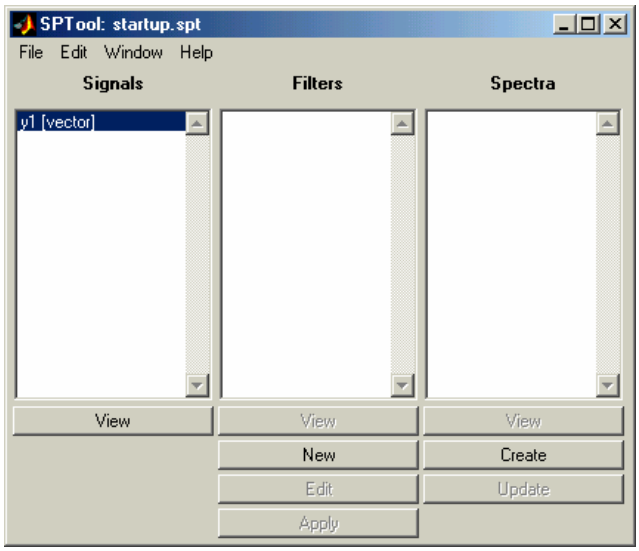


Рис. 5. 76. Окно SPTool после импорта сигнала Y1

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 переключатель `From disk`. В результате будут активизированы два нижерасположенных раздела - окошко *MAT-file Name* и *Browse* (рис 5.77) .

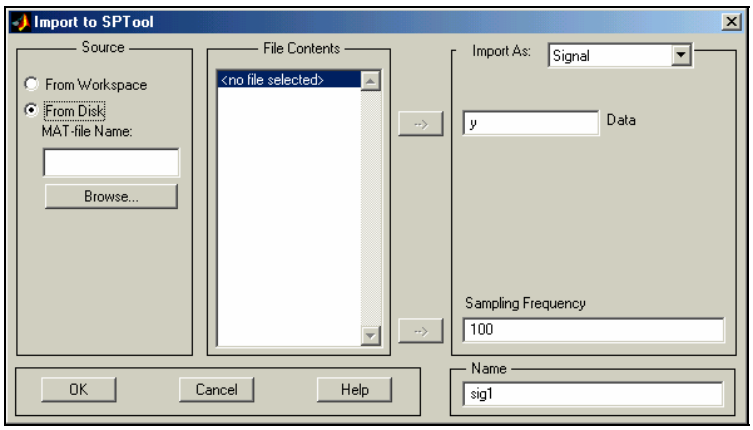


Рис. 5. 77. Окно Import to SPTool с активным переключателем From Disk

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` этот сигнал и нажать на надпись *View* под окошком. В результате должно появиться новое окно `Signal Browser`.

В нашем случае, выбирая сигнал ***y1*** в окошке *Signals* окна ***SPTool***, получим окно, изображенное на рис. 5.78.

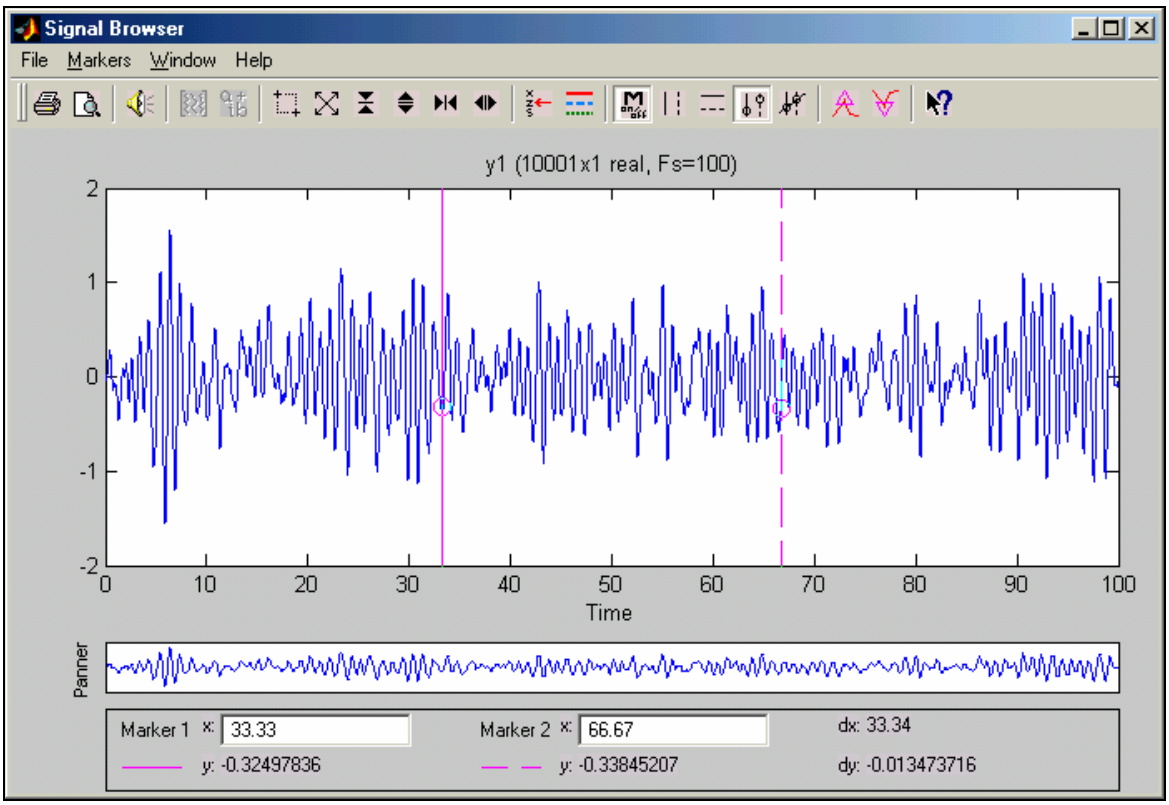


Рис. 5. 78. Окно Signal Brouser

На рис. 5.79 отображены все три процесса.

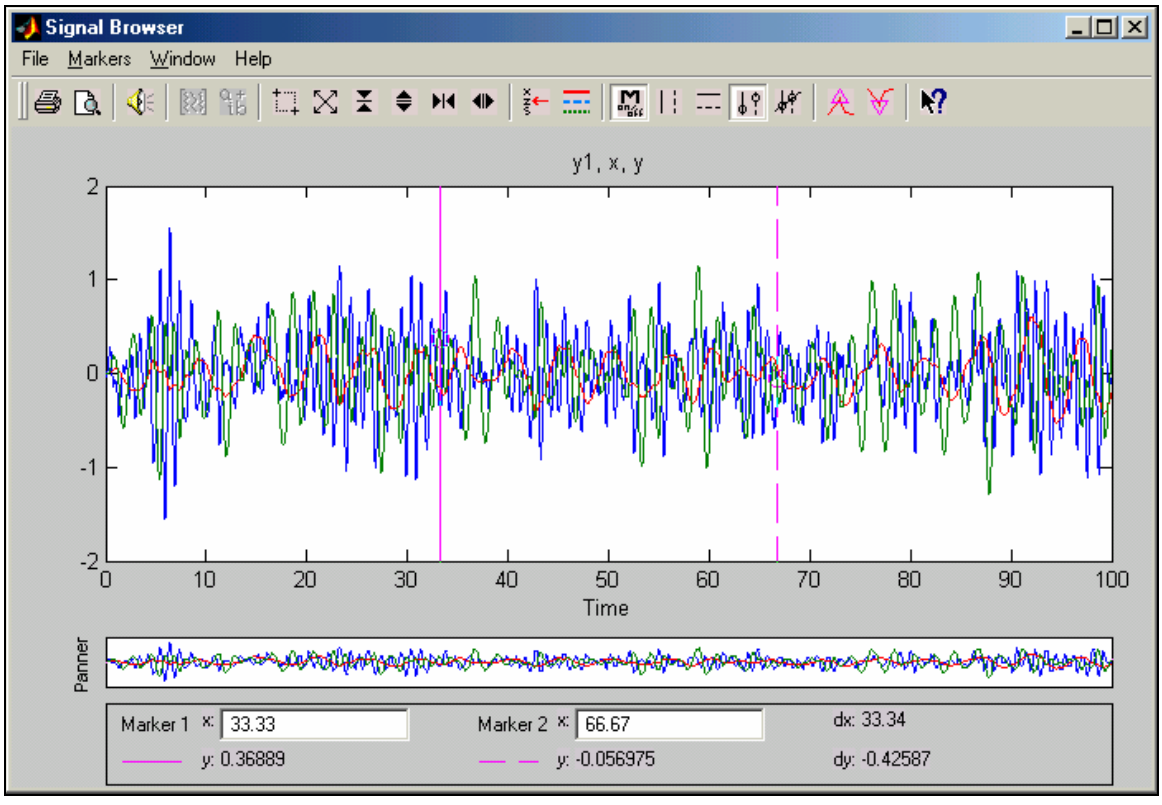


Рис. 5. 79. Отображение трех процессов в окне Signal Browser

Как видим, в заголовке окна указываются имена сигналов, изображенных на графике, размерность соответствующих векторов и частота дискретизации.

Центральную часть окна занимает изображение кривых зависимости выделенных процессов от времени. Там же расположены две вертикальные линии (маркеры), передвигая которые в горизонтальном направлении с помощью мыши, можно определить координаты двух любых точек представленной кривой при установленных маркерами значениях аргумента. Результаты этих отсчетов приводятся в нижней части окна. Там же обычно приводятся значения разностей аргументов и координат этих двух точек

В верхней части окна расположена строка меню, состоящая из четырех меню `File`, `Markers`, `Windows` и `Help`.

Меню `File` содержит команды подготовки и осуществления вывода на принтер содержимого окна.

Содержимое меню `Markers` показано на рис. 5.80.

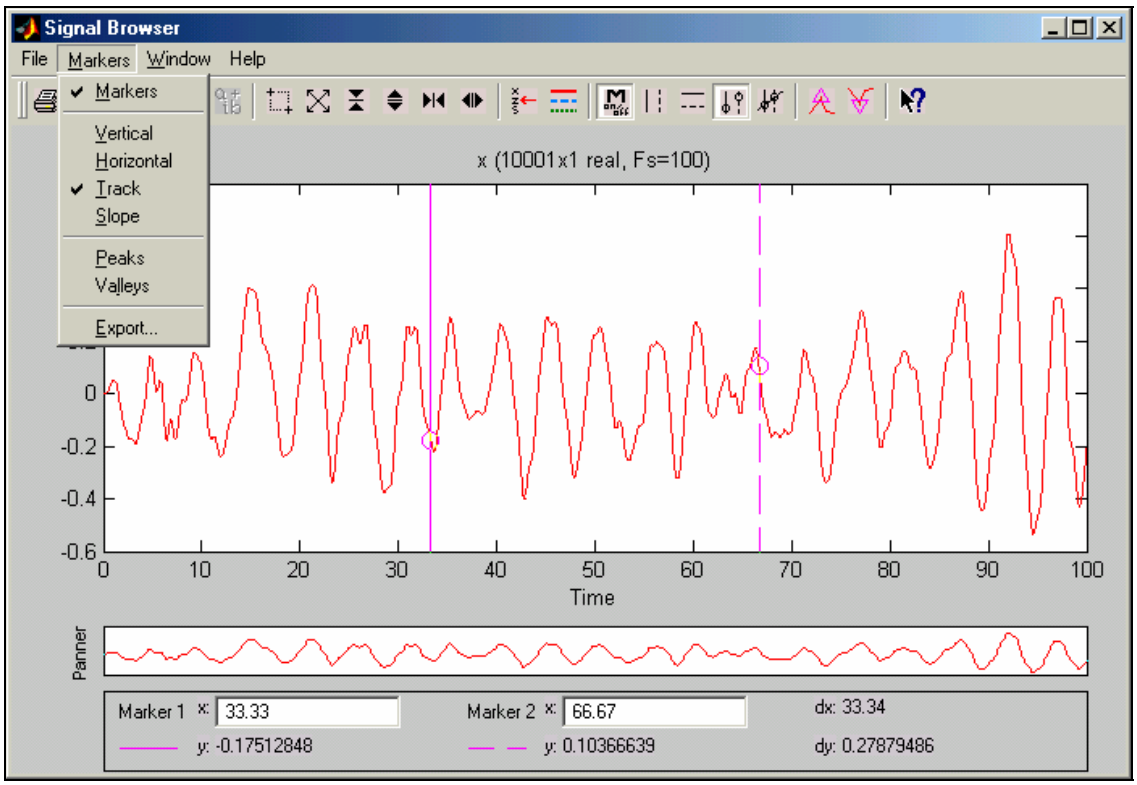


Рис. 5. 80. Список меню Markers

Первая команда *Markers* позволяет отключать (включать) маркеры на изображении процессов. Остальные команды активны только при включении маркеров. Первая из них (*Vertical*) включает отображение только аргументов точек пересечения маркерами с графиком процесса. Вторая (*Horizontal*) создает горизонтальные линии маркеров и позволяет регистрировать только их положение по вертикали. Команда *Track* устанавливает тот режим работы с маркерами, который был описан вначале. Использование команды *Slope* приводит к появлению на графике еще одной линии, соединяющей точки пересечения графика сигнала с маркерными линиями. При этом внизу экрана выводится значение тангенса угла наклона этой прямой к оси абсцисс.

Команда *Peaks* приводит к тому, что линии маркеров могут быть установлены только в точках максимальных значений сигнала. При этом внизу окна появляются точные значения этих максимумов и их аргументов. Аналогично, с помощью команды *Valleys* определяются точки минимумов сигнала.

Наконец, команда *Export* позволяет записать маркерную структуру, изображенную в окне, в виде массива с указанным именем в рабочее пространство MatLAB.

Ниже строки меню расположена линейка инструментов, с помощью которых можно произвести действия, предусмотренные меню `File`, `Markers`, `Windows`, а также следующие операции:

- изменить цвета кривых, изображаемых в окне графиков;
- изменить масштабы изображения по обеим осям графика;
- выделить для укрупненного изображения отдельную область графика.

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 вызвать команду **`Create`** под окошком `Spectra`. При этом на экране появится новое окно - `Spectrum Viewer` *(Обозреватель спектра)* - рис. 5.81.

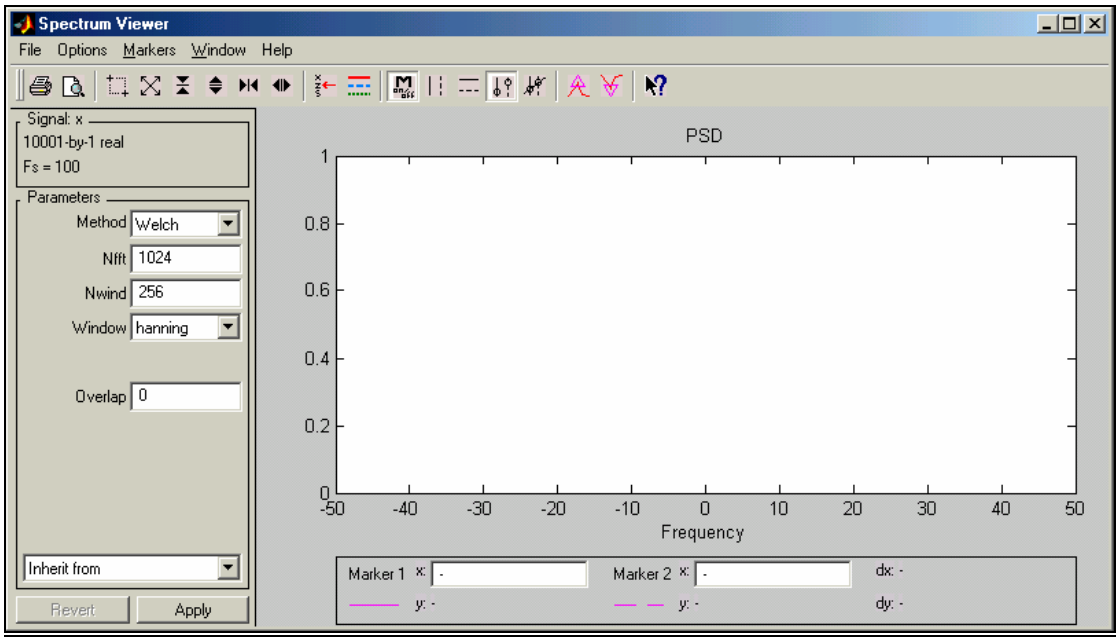


Рис. 5. 81. Окно Specrum Viewer

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;
- выбрать тип окна сглаживания;

Метод вычисления спектра выбирается при помощи списка в окошке под названием *Method*. Этот список содержит такие альтернативы:

Burg;
FFT;
MEM;
MTM;
MUSIC;
Welch;
YuleAR.

Каждому из названий соответствует аналогичный метод (процедура) вычисления спектра сигнала.

Тип используемого при вычислении спектра окна выбирается при помощи списка в окошке под названием `Window`. Список предусматривает использование таких видов окон:

bartlett;
blackman;

boxcar;

chebwin;

hamming;

hanning;

kaiser;

triang.

Для проведения вычислений после выбора метода следует нажать кнопку *Apply* внизу левого поля. Например, выделим для обработки процесс X, вызовем команду **`Create`** и выберем метод FFT. После нажатия кнопки *Apply* в окне `Spectrum Viewer` появится картина, представленная на рис. 5.82.

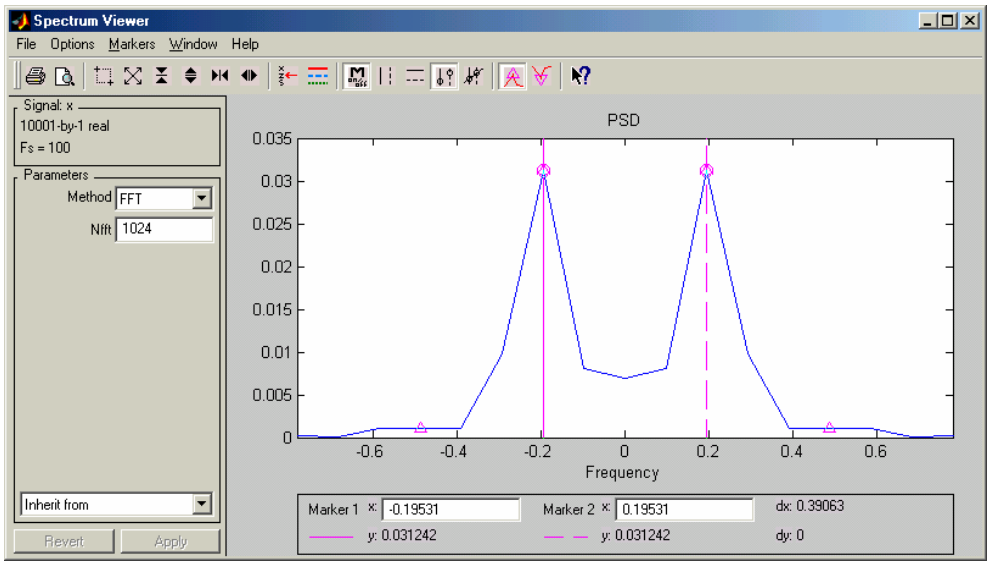


Рис. 5. 82. Спектр сигнала X(t)

Проектирование фильтра

Если в окне `SPTool` выбрать команду **`New`**, то на экране возникнет окно `Filter Designer`, показанное на рис. 5.83.

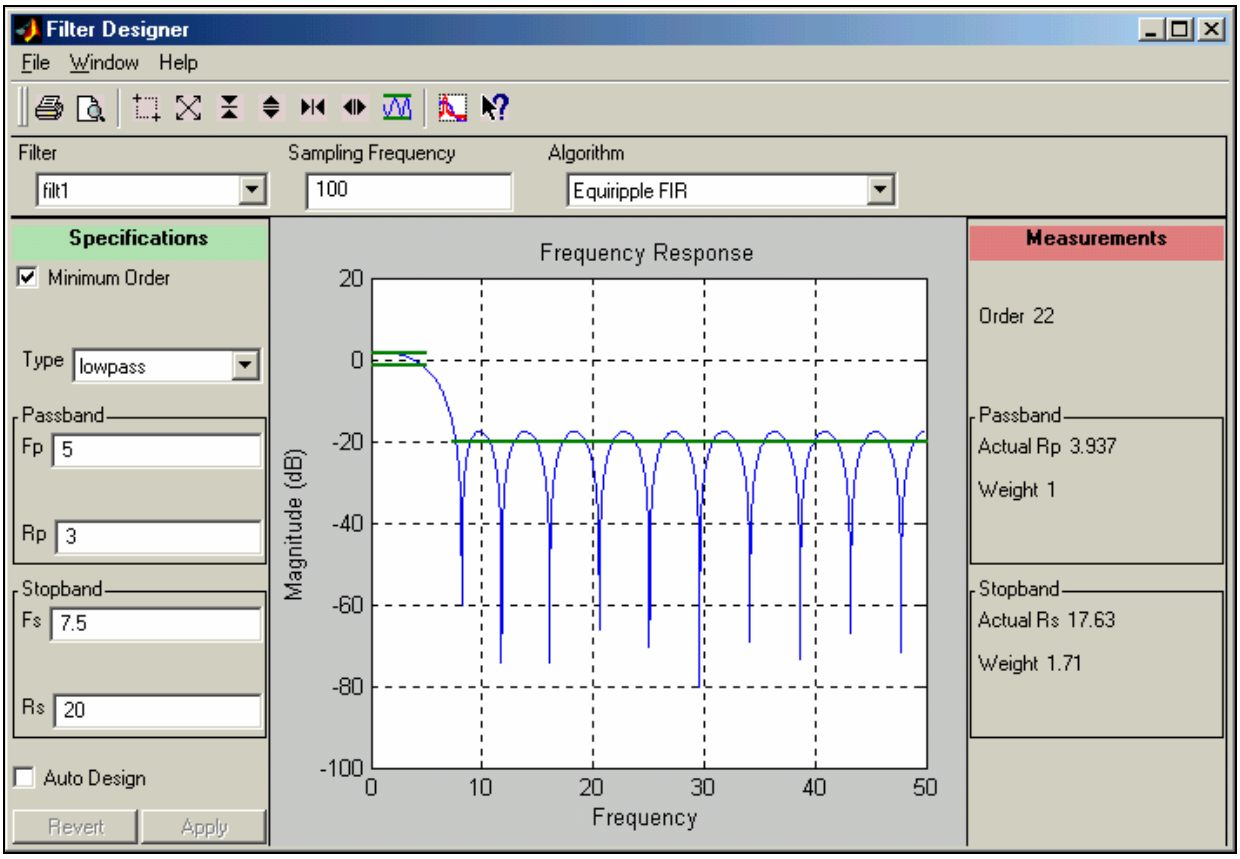


Рис. 5. 83. Окно Filter Designer

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 `'filt1'` .

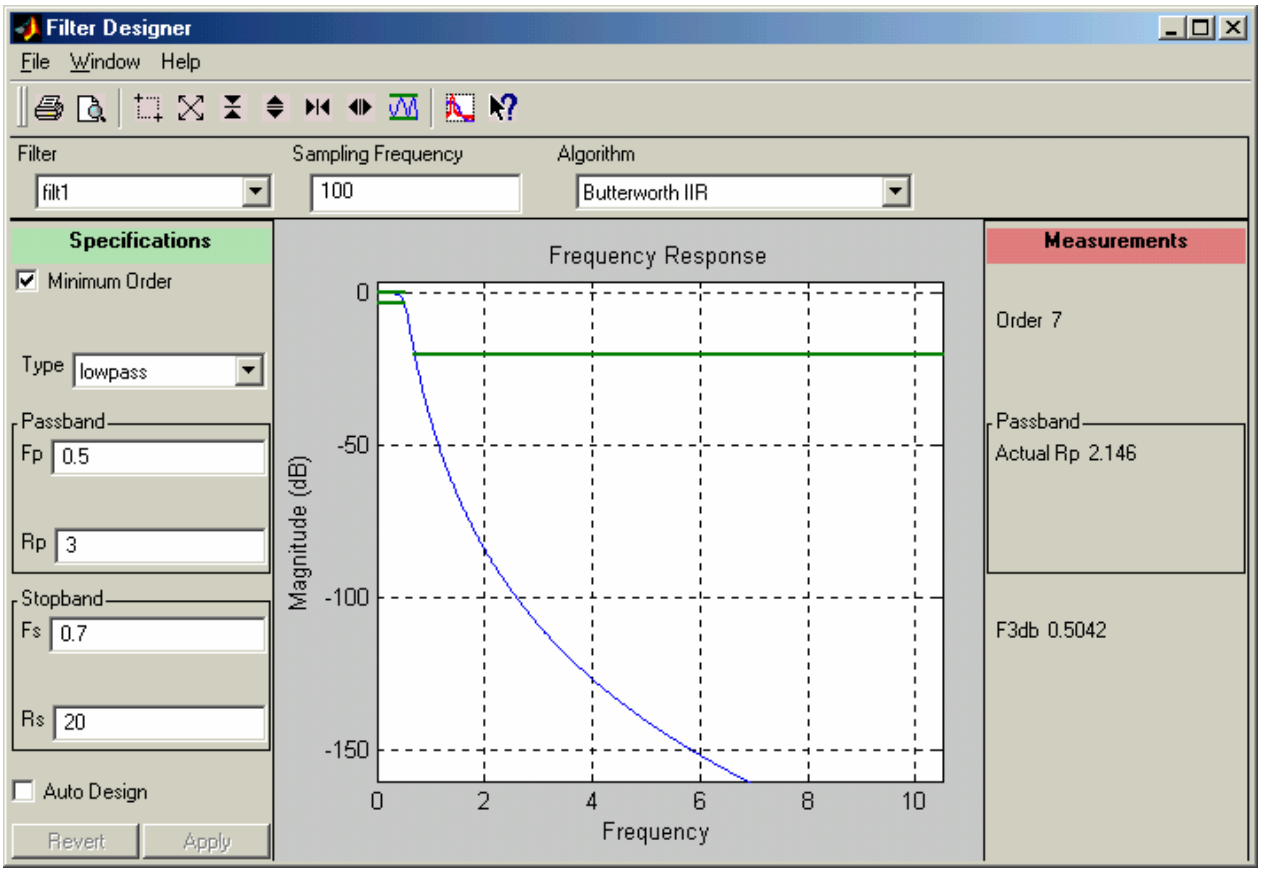


Рис. 5. 84. Проект фильтра низких частот Баттерворта

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.

Например, для только что созданного фильтра `'filt1'` мы получим в результате на экране новое окно `Filter Viewer` с картиной, показанной на рис. 5.85.

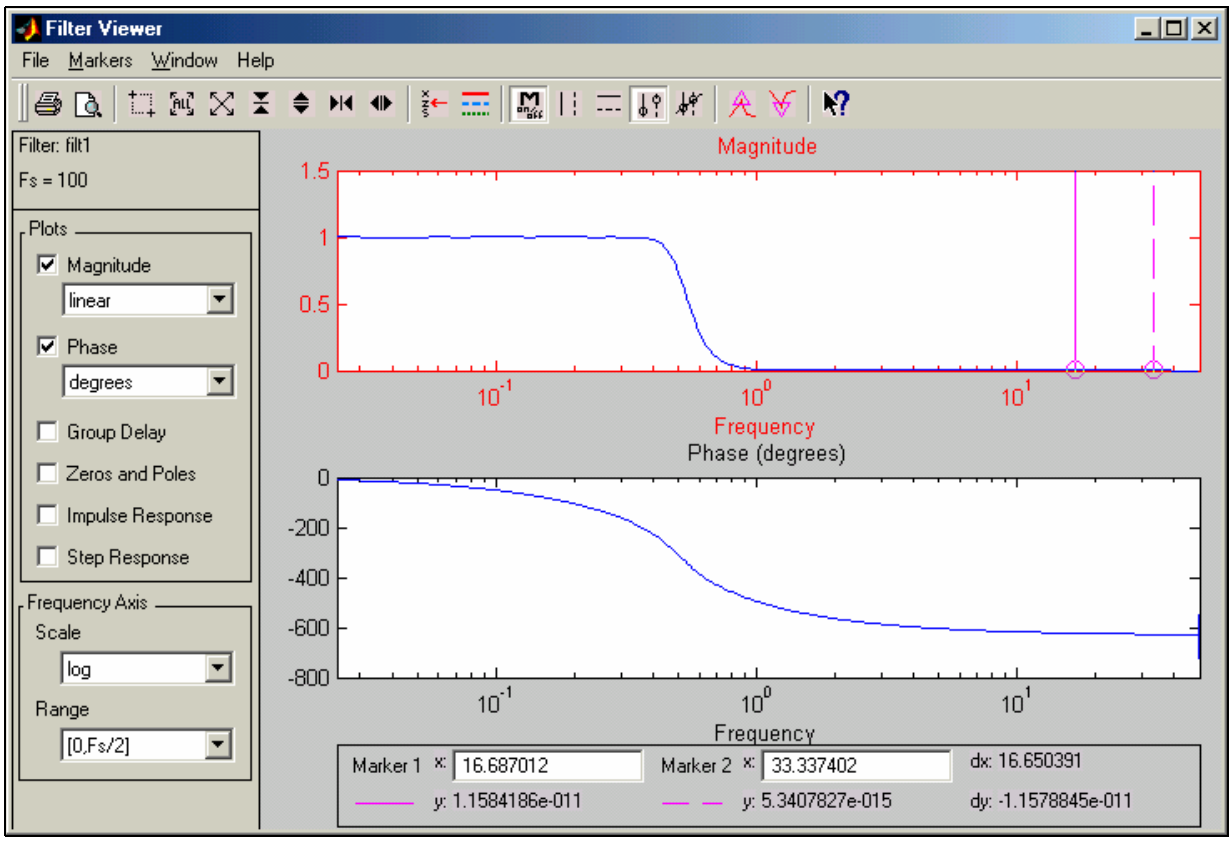


Рис. 5. 85. Окно Filter Viewer

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 установить флажки на нужных видах 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 одновременно всех доступных графиков показан на рис. 5. 86.

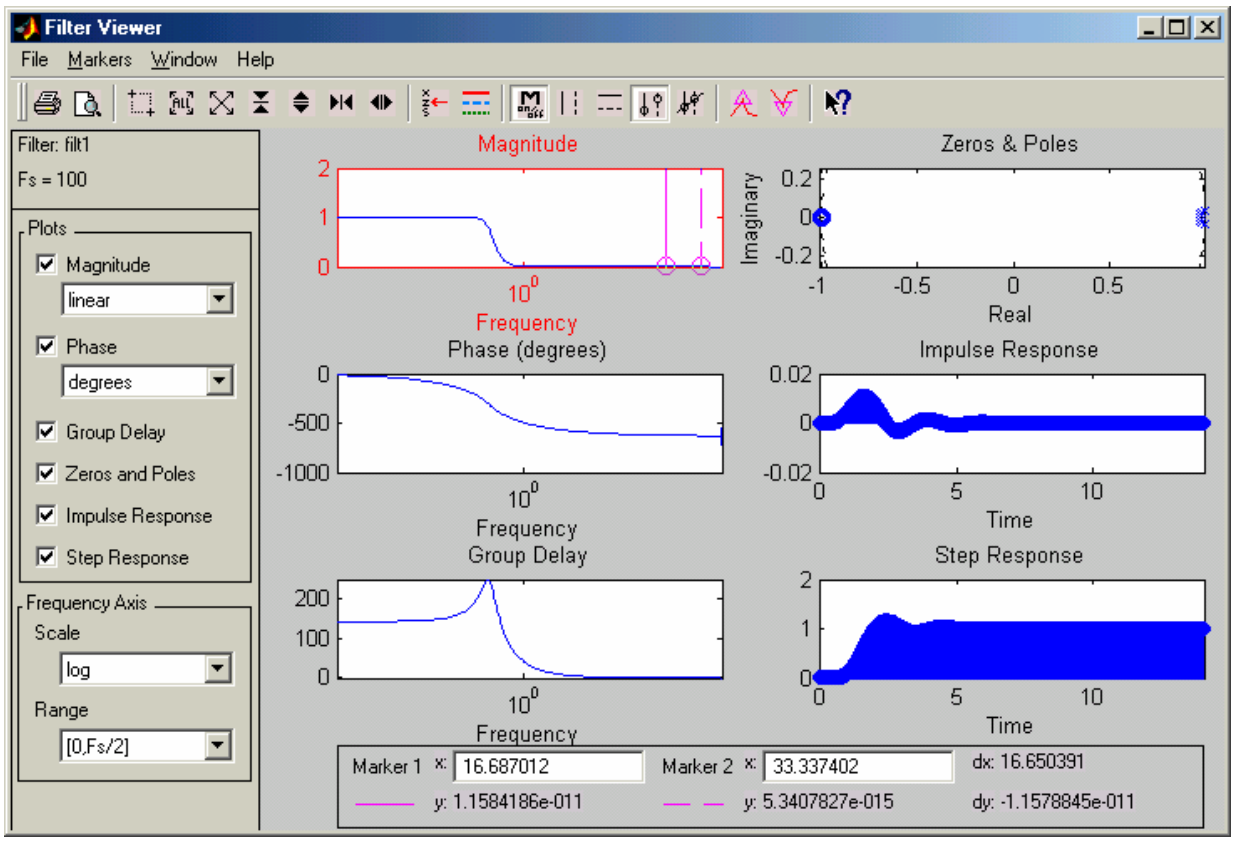


Рис. 5. 86. Возможные виды графиков в окне Filter Viewer

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 'filt1' к процессу X(t), получим процесс, изображенный на рис. 5. 87.

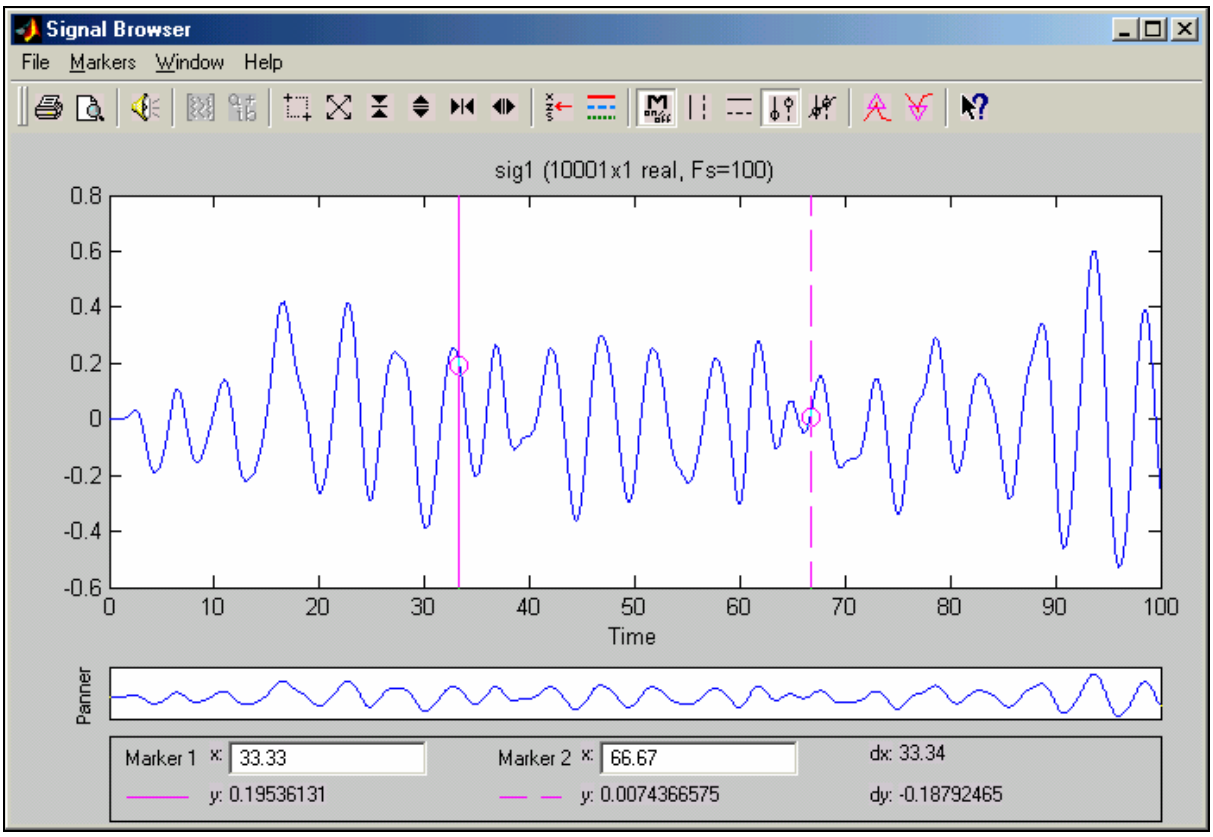


Рис. 5. 87. Результат прохождения сигнала X(t) через фильтр Filt1

Спектральные характеристики полученного процесса можно изучить, применяя раздел *Spectra*, как это было описано.

Вторичное использование результатов SPTOOL

При завершении сеанса работы с ***sptool*** система запрашивает, нужно ли записать полученные результаты на диск. В случае положительного ответа она сохраняет все данные в файле с расширением SPT. Кроме того, в меню ***File*** окна `SPTool` предусмотрены команды записи в файл (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.

5.6. Вопросы для самопроверки

1. Что входит в понятие цифровой обработки сигналов?
2. Какие задачи можно решить с помощью пакета **Signal**?
3. Что такое фильтрация сигналов, какими средствами она обеспечивается?
4. Что такое фильтр низких частот, фильтр высоких частот, полосовой фильтр и режекторный фильтр?
5. Что такое БИХ и КИХ фильтры?
6. Какими средствами в **Signal** обеспечивается проектирование фильтров?
7. Какие интерактивные средства предусмотрены в пакете **Signal**?
8. Какими средствами генерирования процессов обладает пакет **Signal**?
9. Как в пакете **Signal** обеспечить генерирование и анализ случайных процессов?

Урок 6. Исследование линейных стационарных систем (пакет CONTROL Toolbox)

В теории автоматического управления сложился собственный, чрезвычайно удобный и практичный математический аппарат, позволяющий эффективно исследовать поведение линейных стационарных систем автоматического управления.

Линейными стационарными системами (в дальнейшем – ЛСС) принято называть такие системы, поведение которых достаточно удовлетворительно описываются системой обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами. Математическая теория таких систем достаточно полно и хорошо разработана, т. е. полные решения соответствующей системы дифференциальных уравнений можно найти практически для любого вида внешних воздействий и возмущений. Особенно эффективен для исследования таких систем так называемый частотный подход, когда анализируются частотные свойства системы при изменении частоты внешнего гармонически изменяющегося во времени воздействия в широких пределах. Соответствующие частотные характеристики системы (амплитудная и фазовая) в этом случае полностью характеризуют и временные свойства системы при произвольном законе изменения воздействий во времени. Поэтому для анализа ЛСС используются такие специфические характеристики ЛСС, как передаточные функции, частотные передаточные функции, амплитудно-частотные и фазо-частотные характеристики, а также такие методы представления систем, как пространство состояния и др. Хотя эти методы и характеристики разработаны и наиболее эффективны для анализа и синтеза систем автоматического управление, они могут быть с успехом применены для исследования любых динамических систем, описываемых линейными дифференциальуравнениями с постоянными коэффициентами

Пакет CONTROL предназначен для исследования линейных стационарных систем средствами теории автоматического управления.

6.1. Общая характеристика процедур пакета CONTROL

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	Коэффициент передачи при нулевой (низкой) частоте.
norm	Нормы LTI-систем
covar	Ковариация отклика на белый шум
damp	Частота собственных колебаний и демпфирование по полюсам системы.
esort	Сортировка полюсов непрерывной системы по их действительным частям
dsort	Сортировка полюсов дискретной системы по их модулям
pade	Аппроксимация Паде задержек по времени

Модели пространства состояния

rss,drss	Генерирование случайных моделей пространства состояния.
ss2ss	Преобразование переменных состояния
canon	Каноническая форма пространства состояния
ctrb, obsv	Матрицы управляемости и наблюдаемости
gram	Определители 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.

6.2. 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 можно представить в виде:

$$\begin{cases} \ddot{\alpha} + \lambda\dot{\beta} = n(t) \\ \ddot{\beta} - \lambda\dot{\alpha} = l(t) \end{cases} \tag{6.1}$$

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:

$$\begin{cases} \dfrac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u \end{cases} \tag{6.2}$$

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:

$$x_1 = \alpha; \quad x_2 = \beta; \quad x_3 = \dot{\alpha}; \quad x_4 = \dot{\beta}. \tag{6.3}$$

Тогда уравнения гироскопа в форме Коши приобретут вид:

$$\begin{cases} \dot{x}_1 = x_3; \\ \dot{x}_2 = x_4; \\ \dot{x}_3 = -\lambda \cdot x_4 + n(t); \\ \dot{x}_4 = \lambda \cdot x_3 + l(t). \end{cases} \tag{6.4}$$

Теперь нужно образовать матрицы A, B, C и D в соответствии с формой (2) представления системы в пространстве состояния. В рассматриваемом случае в качестве выходного вектора *y* примем:

$$y = [\alpha, \beta]^T; \tag{6.5}$$

а в качестве входного вектора - вектор моментов сил:

$$u = [n(t), l(t)]^T. \tag{6.6}$$

Полагая

$$x = [x_1,\ x_2,\ x_3,\ x_4]^T, \tag{6.7}$$

значения указанных матриц должны быть такими:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\lambda \\ 0 & 0 & \lambda & 0 \end{bmatrix}; \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix};$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{6.8}$$

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 model.
```

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 - 8.882e-016
 #1:  --------------
       s^3 + 100 s

          10
 #2:  -----------
      s^3 + 100 s

Transfer function from input 2 to output...

          -10
 #1:  -----------
      s^3 + 100 s


      s - 8.882e-016
 #2:  --------------
       s^3 + 100 s
```

Теперь преобразуем введенную ***ss***-модель в ***zpk***-модель при помощи процедуры **`zpk`**:

```
GYROzp=zpk(GYROss)
```

```
Zero/pole/gain from input 1 to output...
            s
 #1:  --------------
      s (s^2  + 100)

            10
 #2:  --------------
      s (s^2  + 100)


Zero/pole/gain from input 2 to output...
           -10
 #1:  --------------
      s (s^2  + 100)

            s
 #2:  --------------
      s (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:

$$\begin{cases} E \cdot \dfrac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u, \end{cases} \tag{6.9}$$

где E - произвольная квадратная матрица размером ($n \times n$), 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

$$\begin{cases} J_1 \cdot \ddot{\alpha} + H \cdot \dot{\beta} = N(t) \\ J_2 \cdot \ddot{\beta} - H \cdot \dot{\alpha} = L(t). \end{cases} \tag{6.10}$$

Вводя те же переменные (см. (3), (5)...(7)), получим систему уравнений в виде (9), где матрицы A и E будут иметь вид:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -H \\ 0 & 0 & H & 0 \end{bmatrix} \qquad E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & J_1 & 0 \\ 0 & 0 & 0 & J_1 \end{bmatrix}. \tag{6.11}$$

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 model.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 - 1.11e-016
 #1:  -----------------
        s^3 + 16.67 s

          1.667
 #2:  -------------
      s^3 + 16.67 s

Transfer function from input 2 to output...

         -1.667
 #1:  -------------
      s^3 + 16.67 s

      0.3333 s + 1.48e-016
 #2:  --------------------
         s^3 + 16.67 s
```

```
Gyrozp=zpk(Gyross)
```

```
Zero/pole/gain from input 1 to output...

      0.5 s
#1: ----------------
   s (s^2  + 16.67)

      1.6667
```

```
#2:  ----------------
    s (s^2  + 16.67)

Zero/pole/gain from input 2 to output...
        -1.6667
#1:  ----------------
    s (s^2  + 16.67)

       0.33333 s
#2:  ----------------
    s (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. Примем, 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:

$$W_{11}(s)=\frac{J_2}{J_1\cdot J_2\cdot s^2+H^2};\qquad W_{12}(s)=-\frac{H}{s\cdot(J_1\cdot J_2\cdot s^2+H^2)};$$

$$W_{21}(s)=\frac{H}{s\cdot(J_1\cdot J_2\cdot s^2+H^2)};\qquad W_{22}(s)=\frac{J_1}{J_1\cdot J_2\cdot s^2+H^2}. \qquad (6.12)$$

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:

$$V_{zn1}=\left[J_1\cdot J_2,\quad 0,\quad H^2\right],$$

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)={V_zn1};
        end
end
```

$\text{den}(1,2)=\{V_{zn2}\};$ $\text{den}(2,1)=\{V_{zn2}\}.$

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; старайтесь 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, следует по возможности избегать преобразований моделей из одной формы в другую и наоборот.

Проиллюстрируем это на примере. Преобразуем созданную основную TF-модель gyrotf в SS-модель gyross:

gyross=ss(gyrotf)

```
a =
        x1     x2     x3     x4
  x1     0  -4.167      0      0
  x2     4      0      0      0
  x3     0      0      0  -4.167
  x4     0      0      4      0
  x5     0      0      0      4
  x6     0      0      0      0
  x7     0      0      0      0
  x8     0      0      0      0
  x9     0      0      0      0
  x10    0      0      0      0

        x5     x6     x7     x8
  x1     0      0      0      0
  x2     0      0      0      0
  x3     0      0      0      0
  x4     0      0      0      0
  x5     0      0      0      0
  x6     0      0  -4.167      0
  x7     0      4      0      0
  x8     0      0      4      0
  x9     0      0      0      0
  x10    0      0      0      0

        x9    x10
  x1     0      0
  x2     0      0
  x3     0      0
  x4     0      0
  x5     0      0
  x6     0      0
  x7     0      0
  x8     0      0
  x9     0  -4.167
  x10    4      0

b =
       u1    u2
```

```
  x1   0.25     0
  x2      0     0
  x3   0.25     0
  x4      0     0
  x5      0     0
  x6      0  0.25
  x7      0     0
  x8      0     0
  x9      0  0.25
  x10     0     0

c =
         x1      x2      x3
  y1      0     0.5       0
  y2      0       0       0

         x4      x5      x6
  y1      0       0       0
  y2      0  0.4167       0

         x7      x8      x9
  y1      0 -0.4167       0
  y2      0       0       0

        x10
  y1      0
  y2 0.3333

d =
     u1  u2
  y1  0   0
  y2  0   0
```

Continuous-time model.

Нетрудно убедиться, что мы получили систему, совсем не похожую на ранее введенную S-модель (11), хотя обе они описывают одну и ту же ЛСС. Новая модель отличается от предыдущей не только иными значениями элементов основных матриц, но и, что совсем необычно и непонятно, числом переменных состояния. Из теории следует, что число переменных состояния должно быть равно порядку выбранной системы дифференциальных уравнений. Поэтому в системе (10), имеющей четвертый порядок, должно быть четыре переменных состояния. В последнем случае, как видим, число переменных состояния возросло до десяти.

Резюмируя, отметим, что 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 система, можно перевести в дискретную* форму, воспользовавшись процедурой **c2d** в соответствии со схемой:

```
sysd = c2d(sys, Ts, method).
```

Здесь sys - исходная непрерывная заданная модель, sysd - получаемый в результате работы процедуры дискретный аналог исходной системы, Ts - задаваемое значение шага дискретизации, method - параметр, определяющий метод дискретизации. Последний параметр может принимать одно из 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(kd1,[],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 в 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
```

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**

```
Izm1=parallel(GN,GT)

    Transfer function:

   2 s^2 + 120 s + 200
   -------------------
    s^2 + 10 s + 100
```

либо применяя операцию «сложения» моделей

```
      Izm = GN+GT

   Transfer function:

   2 s^2 + 120 s + 200

   -------------------

    s^2 + 10 s + 100
```

Теперь найдем модель всей системы автоматического управления угловым движением торпеды, рассматривая цепь управления как цепь отрицательной обратной связи для торпеды и пользуясь для объединения прямой и обратной цепи процедурой **feedback**:

```
      sys=feedback(Tor,Izm)

   Transfer function:

               25 s^2 + 250 s + 2500

   ---------------------------------------------

   100 s^4 + 1050 s^3 + 10550 s^2 + 8000 s + 5000
```

Конечно, 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,'InputName',' 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
```

6.3. 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)
- **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 =
         -10.5      -6.5938       -0.625    -0.048828
            16            0            0            0
             0            8            0            0
             0            0            8            0
   B =
          0.25
             0
             0
             0
   C =
             0       0.0625     0.078125     0.097656
   D =
        0
```

```
[z,p,k] = zpkdata(sys,'v')
```

```
   z =
           -5 +     8.6603i
           -5 -     8.6603i
   p =
      -4.8653 +     8.5924i
      -4.8653 -     8.5924i
     -0.38466 +    0.60403i
     -0.38466 -    0.60403i
   k =
          0.25
```

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
          ioDelay: 0
       InputDelay: 0
      OutputDelay: 0
        InputName: {' Момент сил'}
       OutputName: {'Угол рыскания'}
       InputGroup: {0x2 cell}
      OutputGroup: {0x2 cell}
            Notes: {'Угловое движение торпеды'}
         UserData: []
   Continuous-time system.
```

get(sssys)

```
                a: [4x4 double]
                b: [4x1 double]
                c: [0 0.0625 0.0781 0.0977]
                d: 0
                e: []
        StateName: {4x1 cell}
               Ts: 0
          ioDelay: 0
       InputDelay: 0
      OutputDelay: 0
        InputName: {' Момент сил'}
       OutputName: {'Угол рыскания'}
       InputGroup: {0x2 cell}
      OutputGroup: {0x2 cell}
            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).
```

6.4. 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), grid

можно получить график, представленный на рис. 6.1.

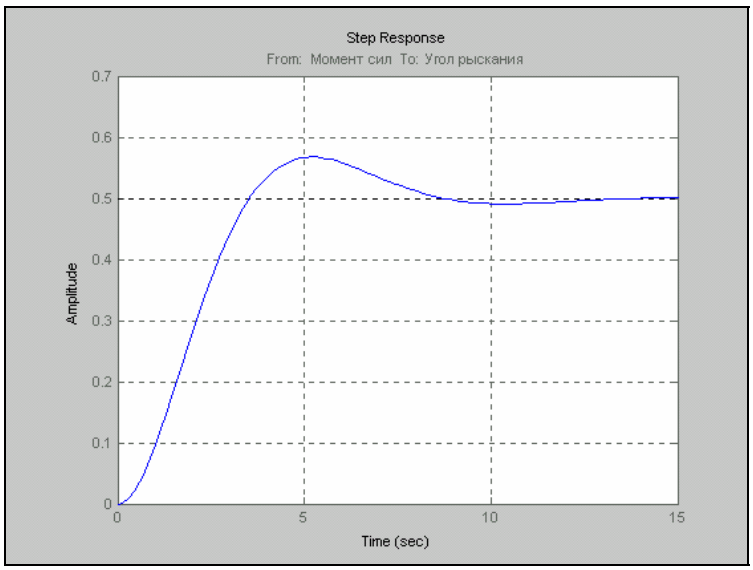


Рис. 6. 1. Отклик системы SYS на единичное ступенчатое воздействие

Аналогично, использование процедуры

```
impulse(sys), grid
```

приведет к появлению в графическом окне графика рис. 6.2.

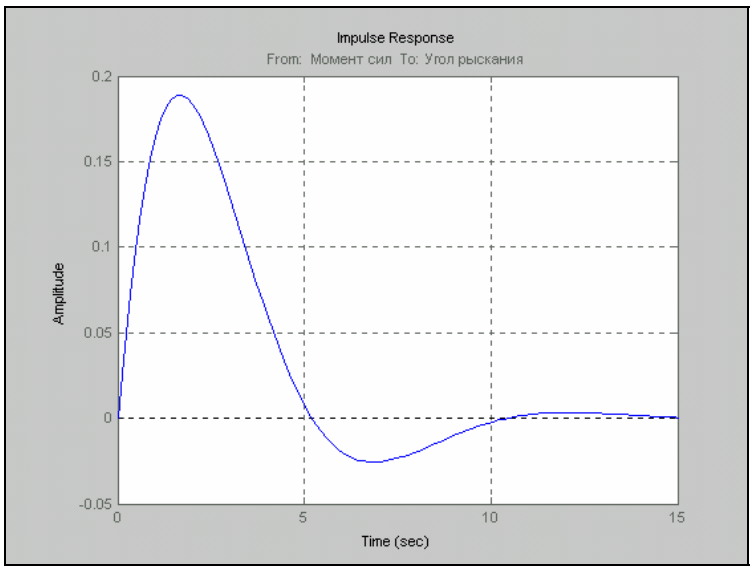


Рис. 6. 2. Отклик системы SYS на единичное импульсное воздействие

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(sssys,[0 0 0 1],20), grid
```

Получим в графическом окне картину, показанную на рис. 6.3.

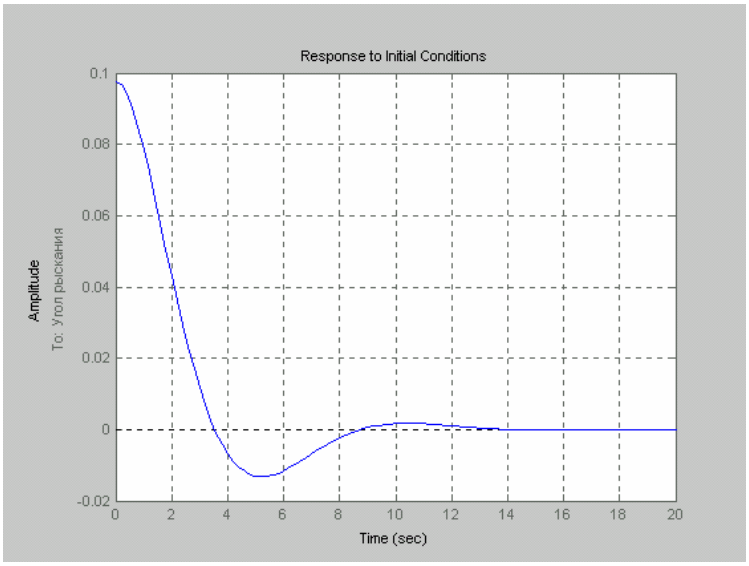


Рис. 6. 3. Переходный процесс в системе SSSYS при заданных начальных условиях

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(sssys,u,t);grid
```

Результат изображен на рис. 6.4. На нем одна кривая отображает входное воздействие, а другая – реакцию на него системы.

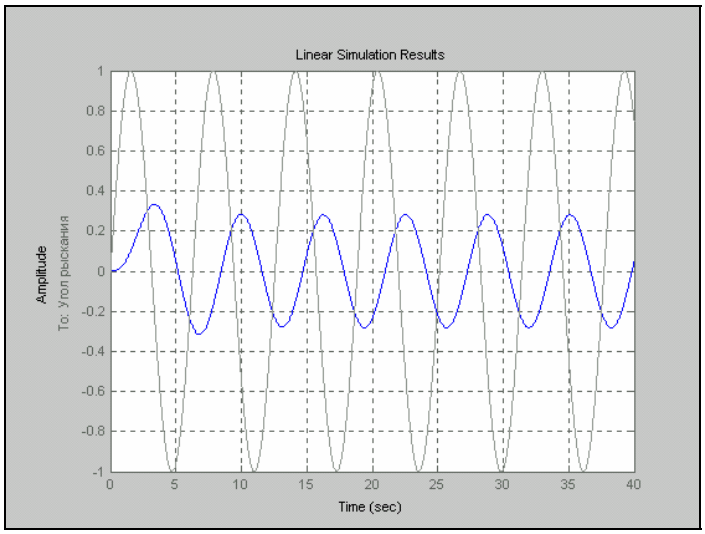


Рис. 6. 4. Реакция системы SSSYS на заданное воздействие

Следующая группа процедур представляет в частотной области реакцию системы на внешние гармонические воздействия. К таким процедурам относятся:

- **bode** - строит графики АЧХ и ФЧХ (диаграмму Боде) указанной системы;

- **nyquist** - строит в комплексной плоскости график Амплитудно-Фазовой Характеристики (АФХ) системы в полярных координатах;

- **nichols** - строит карту Николса системы, т. е. график АФХ разомкнутой системы в декартовых координатах;

- **sigma** - строит графики зависимости от частоты сингулярных значений системы; обычно совпадает с АЧХ системы;

- **margin** – строит диаграмму Боде с указанием *запасов по амплитуде и по фазе*.

Приведем примеры.

```
bode(sys), grid
```

- результат приведен на рис. 6.5;

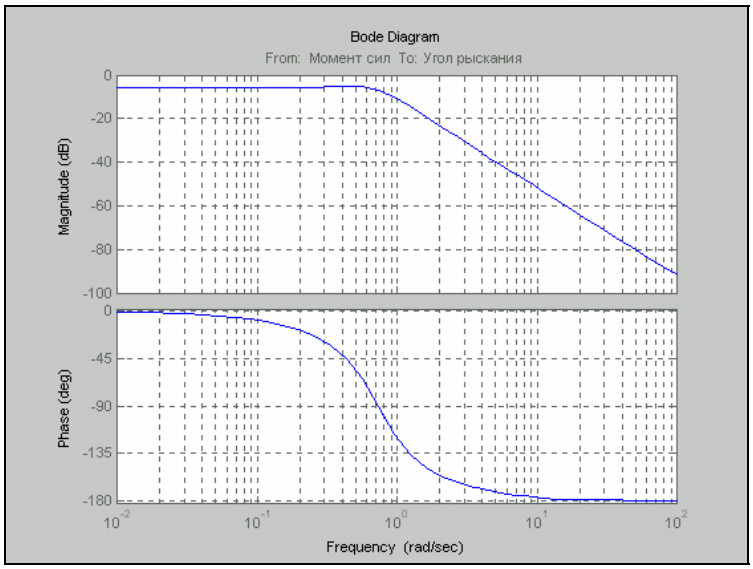


Рис. 6. 5. Диаграммы Боде (АЧХ и ФЧХ) системы SYS

```
nyquist(sys); grid
```

- результат - на рис. 6.6;

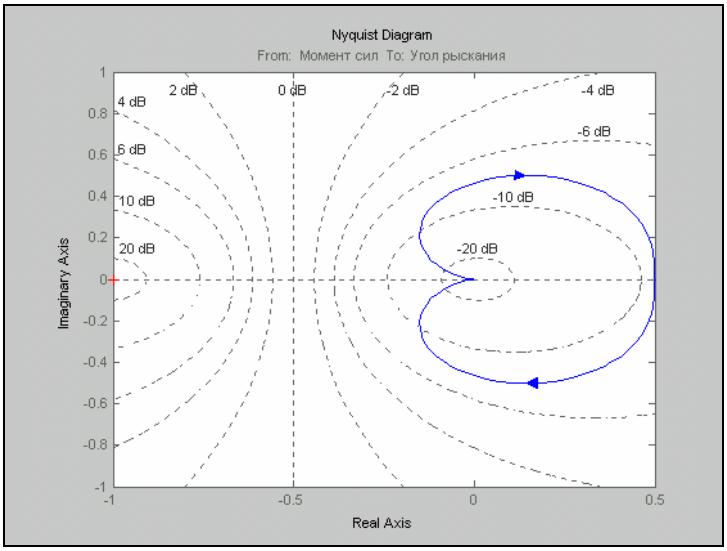


Рис. 6. 6. Диаграмма Найквиста системы SYS

```
nichols(sys);  grid
```

- результат показан на рис. 6.7;

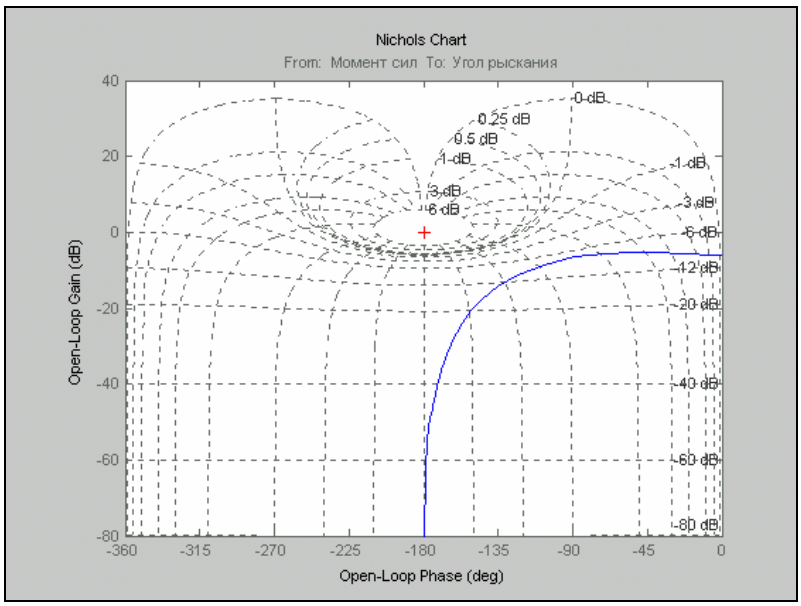


Рис. 6. 7. Карта Николса для разомкнутой системы SYS

```
sigma(sys), grid
```

- см. рис. 6.8;

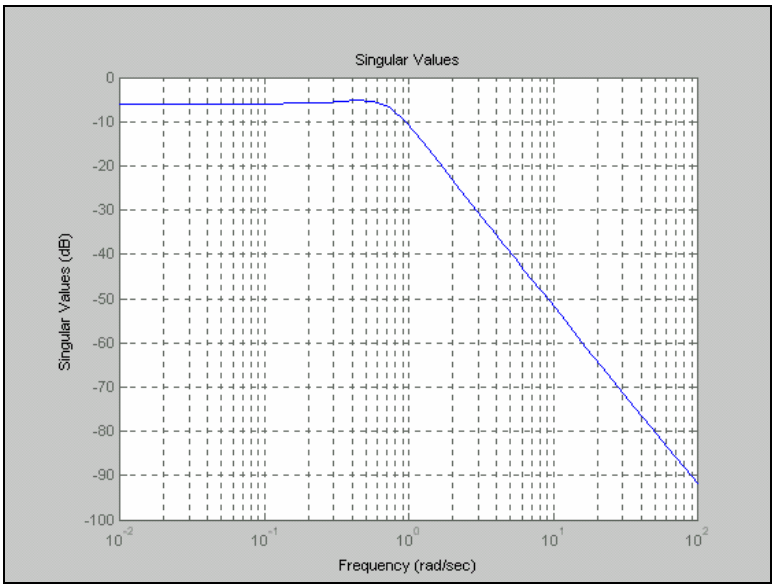


Рис. 6. 8. Частотная зависимость сингулярных чисел системы SYS

```
margin(sssys); grid
```

- см. рис. 6.9.

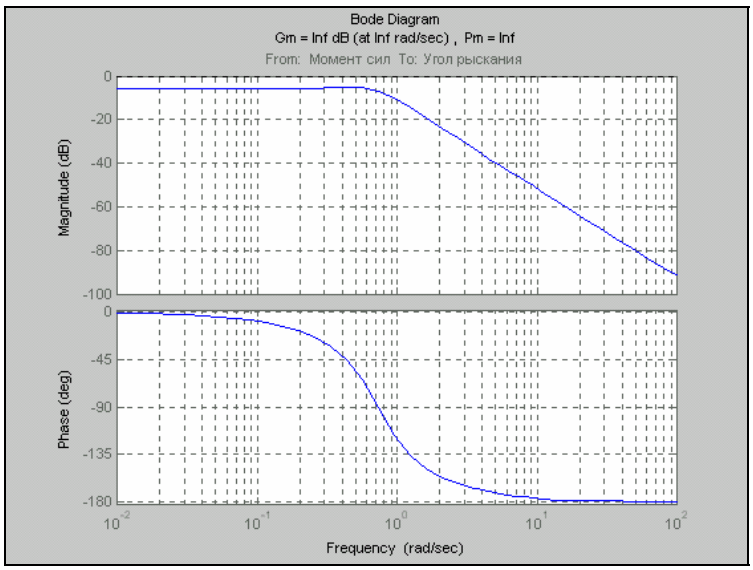


Рис. 6. 9. АЧХ и ФЧХ системы SYS с указанием запасов по амплитуде и фазе

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 =
       0.0032245  1.3753e-016   -0.0041717 -1.5179e-015
     1.3753e-016    0.0083434 -1.4832e-016    -0.070084
      -0.0041717 -1.4832e-016     0.070084 -7.7542e-016
    -1.5179e-015    -0.070084 -7.7542e-016       8.7807
```

```
Wo=gram(sssys,'o')
```

```
   Wo =
       1.3335    0.8751    1.0938    0.0977
       0.8751    0.5770    0.7210    0.0682
       1.0938    0.7210    0.9011    0.0851
       0.0977    0.0682    0.0851    0.0134
```

```
pzmap(sys), grid
```

- результат см. рис. 6.10

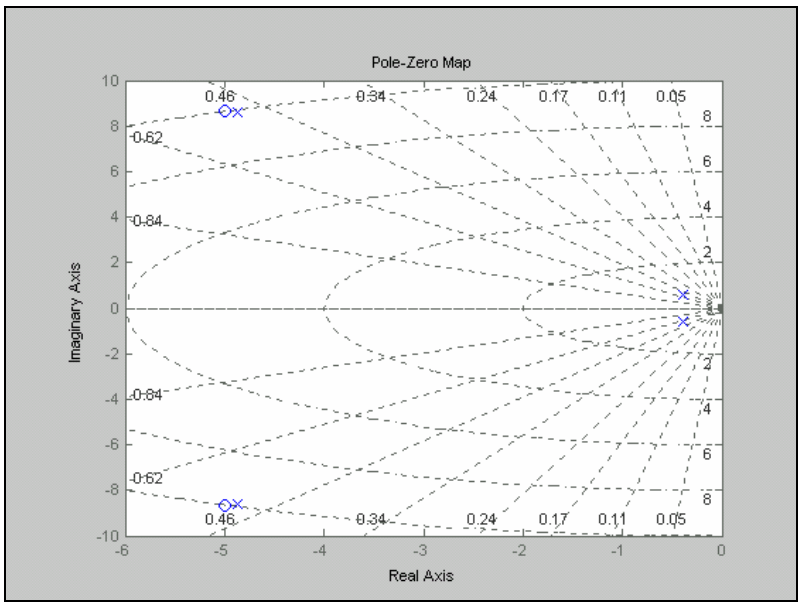


Рис. 6. 10. Изображение нулей и полюсов системы SYS

```
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), grid
```

- результат - на рис. 6.11.

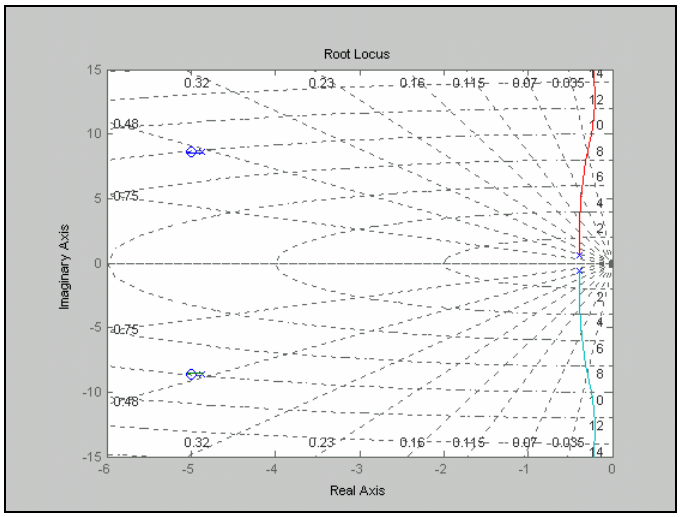


Рис. 6. 11. Траектории полюсов системы SYS при изменении коэффициента передачи

6.5. Интерактивный обозреватель ltiview

Набирая в командном окне MatLAB команду **`ltiview`**, можно вызывать окно так называемого «обозревателя» LTI-объектов, который 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).

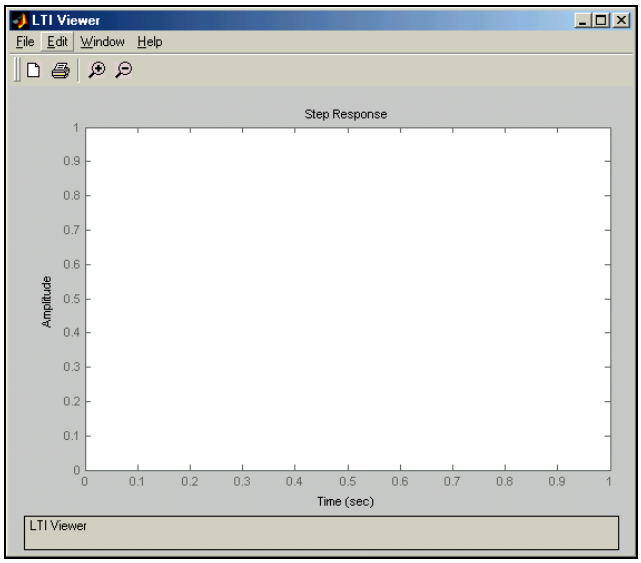


Рис. 6. 12. Окно LTI Viewer

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 верхней части расположены строка меню и линейка инструментов.

Меню `File` содержит такие команды (рис. 6.13):

`New Viewer` открыть новый обозреватель;

`Import` ввести новые LTI объекты в обозреватель;

`Export` вывести объекты из обозревателя (в рабочее пространство);

`ToolboxPreferences` установление (изменение) свойств графического вывода в обозревателе;

`Page Setup` установка свойств расположения графического изображения на листе бумаги;

`Print` выведение графического изображения на принтер;

`Print to Figure` выведение графического изображения в окно фигуры (это удобно для того, чтобы воспользоваться в дальнейшем опцией **Copy Figure** для вывода графиков на печать);

`Close` закрыть обозреватель.

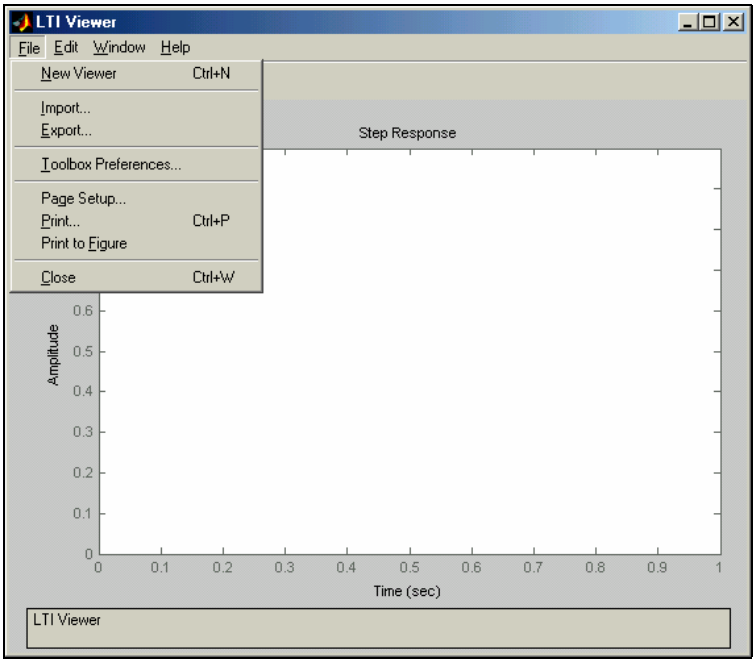


Рис. 6. 13. Меню File окна LTI Viewer

Работу с обозревателем необходимо начинать с «загрузки» в его среду тех LTI-объектов, которые нужно анализировать. Для этого следует воспользоваться командой `Import` меню `File`. В результате на экране возникнет новое окно **Import System Data** (рис. 6.14)

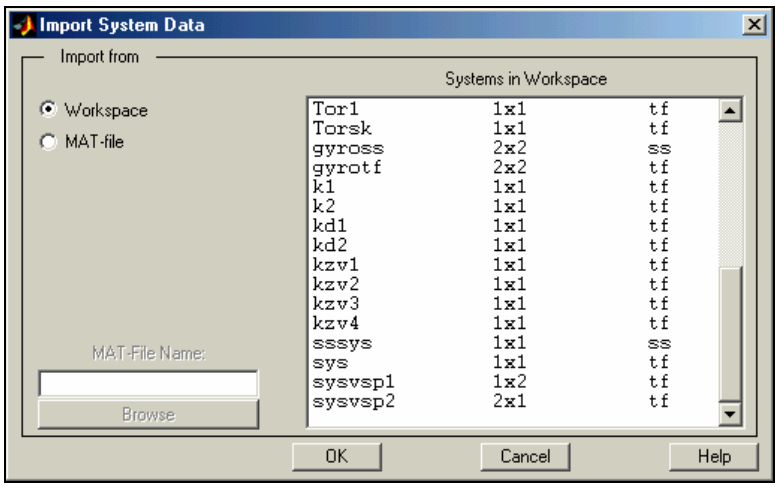


Рис. 6. 14. Окно Import System Data

Как видим, загрузить LTI-объекты можно из рабочего пространства (переключатель *WorkSpace*) или из MAT-файла (переключатель *MAT-file*).

Отметим по очереди в окошке справа LTI-объекты *Tor* и *sssys*, представляющие соответственно неуправляемое и управляемое движение торпеды по углу рыскания, и нажмем кнопку <OK>. Окно **Import System Data** исчезнет, а в окне **LTI Viewer** появятся две кривые, отражающие движение торпеды под действием единичного момента сил (рис. 6.15).

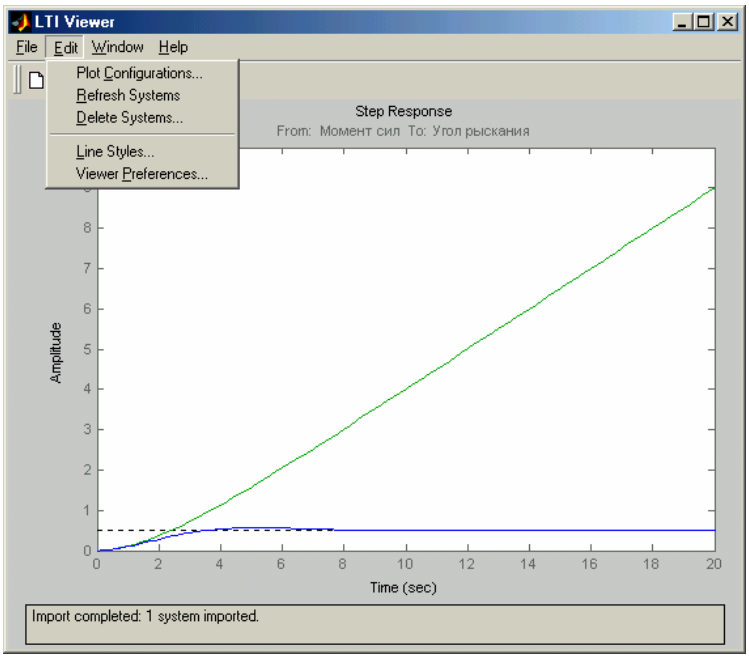


Рис. 6. 15. Окно LTI Viewer с графиками систем Tor и sssys

Второе меню `Edit` окна **LTI Viewer** содержит следующие команды (рис. 6.15):

`Plot Configurations` установка вида графиков, выводимых в графическое окно **LTI Viewer,** и их количества;

`Refresh Systems` обновление LTI-объектов;

`Delete Systems` удаление LTI-объектов;

`Lyne Styles` установка стилей линий на графиках;

`Viewer Preferences` установка свойств графиков.

Прежде всего, следует определиться с количеством и видом графиков, выводимых в окно **LTI Viewer.** Их можно установить с помощью команды `Plot Configurations` меню ***Edit***. Вызов ее приводит к появлению на экране окна ***Plot Configurations*** (рис.6.16).

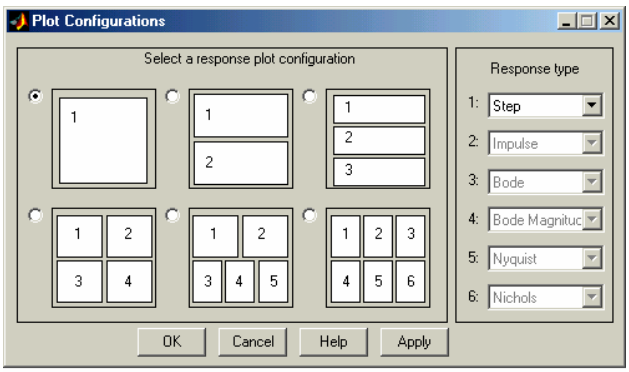


Рис. 6. 16. Окно Plot Configurations

Из рис. 6.16 видно, что предусмотрено выведение в окно **LTI Viewer** от одного до шести графических подокон. По умолчанию установлен вывод одного подокна, в которое выводится график реакции системы на единичное воздействие. Выбор количества подокон производится переключателем в верхнем левом углу соответствующего изображения.

Тип графика, выводимого в подокно с указанным номером, устанавливается с помощью выпадающего меню, которым снабжено каждое окошко справа с этим номером (рис. 6.17).

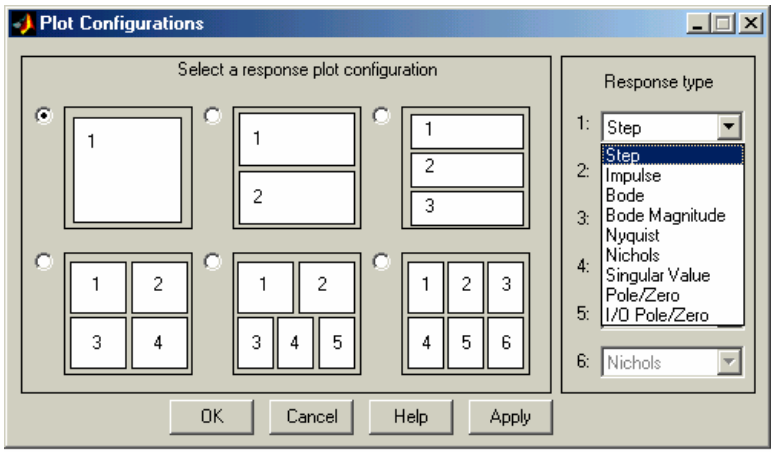


Рис. 6. 17. Список графиков, выводимых в окно LTI Viewer

Возможны девять вариантов графиков (рис. 6.17):

Step	– реакции системы на единичное ступенчатое воздействие;
`Impulse`	реакции на единичное импульсное воздействие;
`Bode`	АЧХ и ФЧХ системы;
`Bode Magnitude`	АЧХ системы;
`Nyquist`	диаграммы Найквиста;
`Nichols`	карты Николса;
`Singular Value`	зависимости сингулярных значений системы от частоты;
`Pole/Zero`	расположения нулей и полюсов системы;
`I/O Pole/Zero`	то же.

Выбирая соответствующую команду, устанавливают нужное изображение в указанное подокно.

Выберем графическое окно с четырьмя подокнами. Установим в первое подокно **`Bode Magnitude`**, во второе – **`Impulse`**, в третье - **`Pole/Zero`**, а в четвертое – **`Step`**. Нажимая клавишу <OK>, получим в окне **LTI Viewer** изображение, представленное на рис. 6.18.

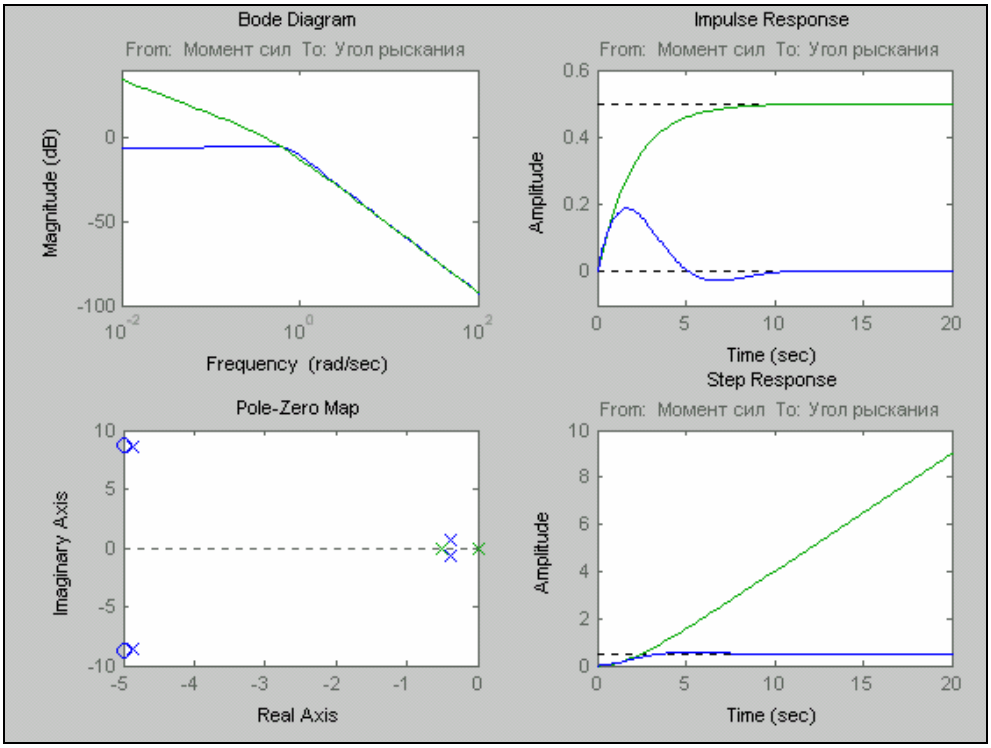


Рис. 6. 18. Графики систем Tor и sssys, выведенные в окно LTI Viewer

Рассмотрим теперь команду **`Viewer Preferences`** меню `Edit` окна **LTI Viewer**.

Вызывая ее, получим на экране окно **LTI Viewer Preferences** (рис. 6.19).

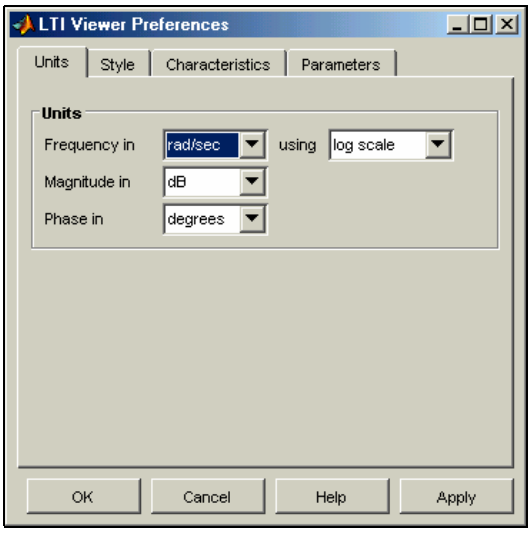


Рис. 6. 19. Окно LTI Viewer Preferences

Оно содержит четыре вкладки:

`units` установки единиц измерения, в которых будут откладываться величины по осям графиков;

`style` установки кеглей текстовых символов, наносимых на графики;

`characteristics` установки параметров некоторых численных характеристик процессов;

`parameters` установки диапазонов изменения аргументов, отличных от принятых по умолчанию.

Как видно из рис. 6.19, по умолчанию принимаются следующие единицы измерения:

- для частоты – *радианы в секунду* и используется логарифмическая шкала;
- для амплитуды – *дециБеллы*;
- для фазы – *градусы*.

При помощи списков можно изменить единицы измерения частоты на *Герцы*, амплитуды – на абсолютные единицы, фазы – на радианы, а шкалу по частоте сделать равномерной.

Содержание вкладки *style* показано на рис. 6.20.

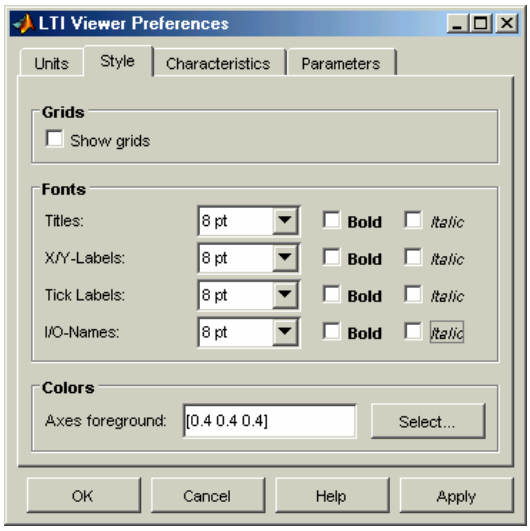


Рис. 6. 20. Вкладка Style окна LTI Viewer Preferences

С ее помощью можно:

- установить сетку координатных линий на графиках;
- установить размер (кегль) символов, выводимых в заголовок, в надписи по осям координат, в деления по осям, в названия входа и выхода системы;
- установить стиль символов (жирный, курсив);
- установить цвет фона.

Установим разметку графиков, увеличим размеры символов в заголовках графиков до 12 кеглей, а надписей по осям координат до 10 кеглей. Кроме того, установим жирный шрифт курсивом в заголовках (рис. 6.21).

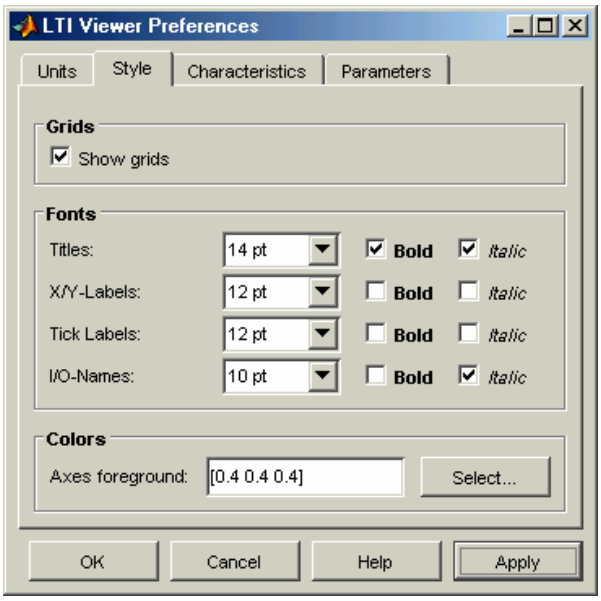


Рис. 6. 21. Вкладка Style с измененными характеристиками

Закрепляя эти изменения нажатием клавиши <OK>, мы получим графическое изображение в виде, представленном на рис. 6.22.

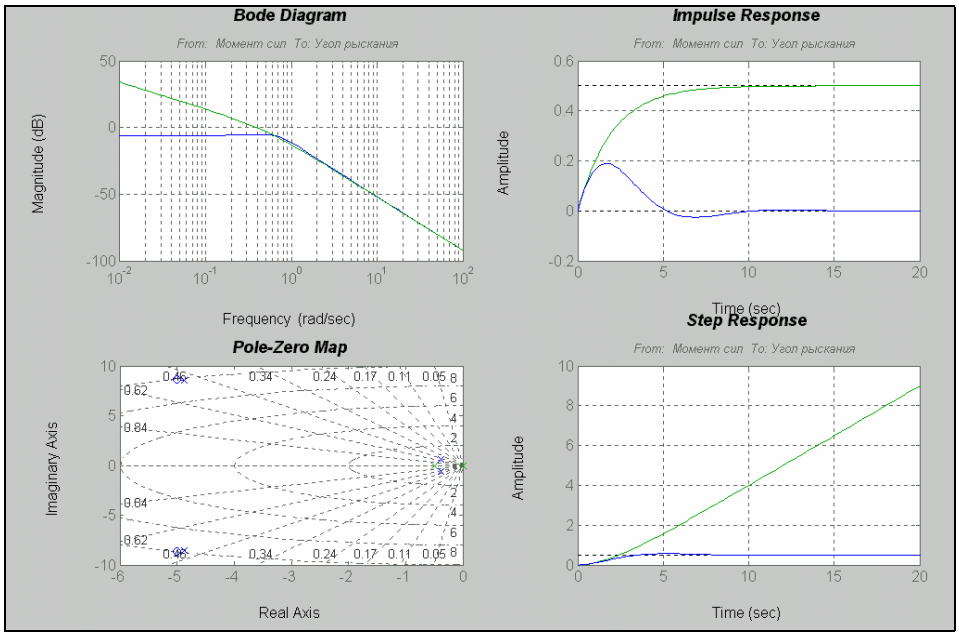


Рис. 6. 22. Графики систем Tor и sssys с измененными параметрами графики

Следующая вкладка - **`characteristics`** - окна **LTI Viewer Preferences** показана на рис. 6.23. В ней указано, что время установления переходного процесса определяется по уровню 2%, а время возрастания – по промежутку времени от момента, когда значение процесса равно 10%, до момента, когда оно достигает 90% установившегося значения. "Галочка" в окошке *Unwrap phase* означает, что при вычислении фазы в районе перехода ее через $\pm\pi$ предприняты меры, чтобы она не претерпевала разрыва на $\pm 2\pi$.

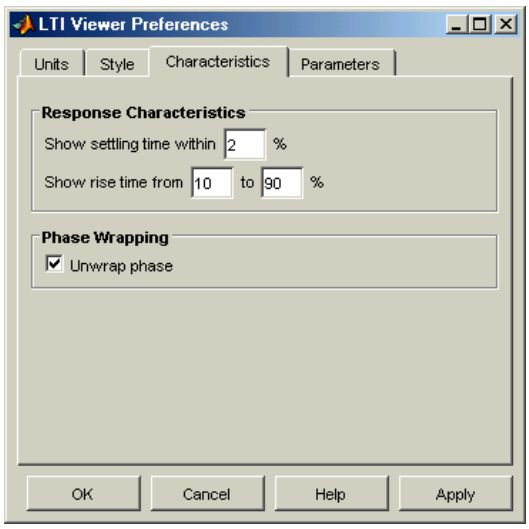


Рис. 6. 23. Вкладка Characteristics окна LTI Viewer Preferences

Первые три характеристики (численные) могут быть изменены пользователем по своему усмотрению. Может быть также снята галочка в окошке *Unwrap phase*. Тогда фаза может на графиках претерпевать разрывы на $\pm 2\pi$ радиан (что, кстати, не соответствует реальным особенностям непрерывной системы).

Вкладка *parameters* окна **LTI Viewer Preferences** представлена на рис. 6.24.

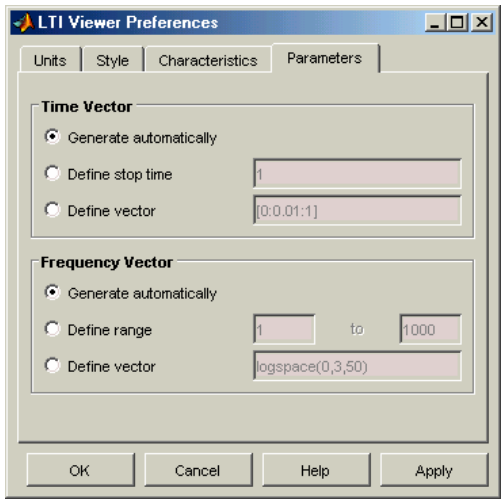


Рис. 6. 24. Вкладка Parameters окна LTI Viewer Preferences

Используя ее, можно, в случае необходимости, задать диапазоны изменения времени и частоты по своему усмотрению.

Наконец, рассмотрим команду **`Lyne Styles`** меню `Edit` окна **LTI Viewer.** При ее вызове возникает окно **Line Styles**, показанное на рис. 6.25.

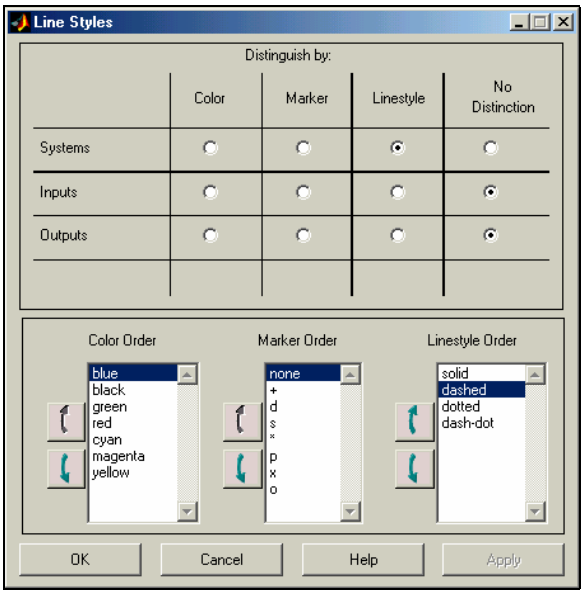


Рис. 6. 25. Окно Line Styles

Установка стилей линий происходит в следующем порядке.

Сначала, с помощью верхней таблицы рис. 6.25, устанавливают какую либо одну особенность линий, которой и будут отличаться линии на графиках, относящиеся к разным системам (цвет, маркер или стиль линии). Для этого соответствующее свойство должно быть отмечено переключателем под его именем в таблице.

Затем, в одном из окошек в нижней части окна, которое соответствует выбранному свойству линии, выделяют значение этого свойства для первой LTI-системы и перемещают его с помощью ползунков слева от окошка вверх, на верхнюю позицию. Точно так же устанавливают на второе сверху место значение этого свойства, принимаемое для второй системы и т. д.

Так, на рис. 6.25 установлено, что линии будут синими и различаться стилем линий. Линии первой LTI-системы (sssys) будут сплошными, а второй (Tor) – штриховыми. Результат такой установки отражен на рис. 6.26.

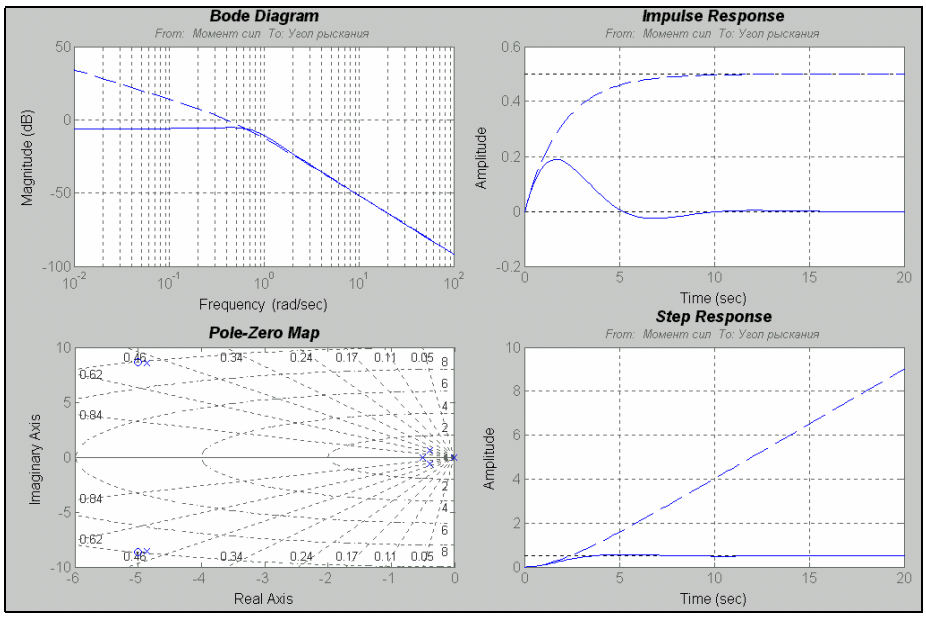


Рис. 6. 26. Графики систем Tor и sssys с измененными параметрами линий

Для вывода содержимого графического окна **LTI Viewer** на печать можно использовать команду **`Print to Figure`** меню `File`, которая осуществляет выведение графика предварительно в графическое окно фигуры. Затем содержимое фигуры по обычным правилам может быть либо перенесено в окно документа текстового редактора, либо выведено на принтер. Именно таким способом были получены рис. 6.22 и 6.26.

6.6. 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

$$u = - Kx \tag{6.13}$$

минимизирует функционал

$$J = \int \{x'Qx + u'Ru + 2*x'Nu\}\, dt, \tag{6.14}$$

если объект регулирования описывается уравнениями состояния

$$\frac{dx}{dt} = A \cdot x + B \cdot u\ . \tag{6.15}$$

Если последняя матрица N при обращении к процедуре не указана, то она принимается по умолчанию нулевой. Одновременно вычисляется решение S алгебраических уравнений Риккати

$$S \cdot A + A' \cdot S - (S \cdot B + N) \cdot R^{-1} \cdot (B'S + N') + Q = 0 \tag{6.16}$$

и находятся собственные значения E замкнутой системы

$$\mathbf{E = eig(A-B*K)}\ . \tag{6.17}$$

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

$$J = \int \{y'Qy + u'Ru + 2*y'Nu\}\, dt. \tag{6.18}$$

В этом случае входным параметром процедуры является сама ***ss***-модель системы в форме

$$\frac{dx}{dt} = A \cdot x + B \cdot u\,, \qquad y = C \cdot x + D \cdot u\,. \tag{6.19}$$

а вызываться процедура должна таким образом **`[K,S,E] = lqry(sys,Q,R,N)`**, где sys - имя ***lti***-модели оптимизируемой САУ. Та же процедура может быть применена для дискретной системы (модели), уравнения состояния которой заданы в виде конечно-разностных уравнений вида

$$x[n+1] = Ax[n] + Bu[n], \quad y[n] = Cx[n] + Du[n]. \tag{6.20}$$

при этом минимизируется функционал

$$J = \text{Sum } \{y'Qy + u'Ru + 2*y'Nu\}. \tag{6.21}$$

Применим процедуру к рассматриваемой системе. Получаем:

```
Q=1; R=1;
[K,S,E] = LQRY(sssys,Q,R)
  K =
       0.30016      0.19769      0.24705     0.023054
  S =
        1.2007      0.79074      0.98822     0.092214
       0.79074      0.52333      0.65394     0.064612
       0.98822      0.65394      0.81717     0.080638
      0.092214     0.064612     0.080638     0.012994
  E =
      -4.8653 +     8.5924i
      -4.8653 -     8.5924i
     -0.42218 +    0.62857i
     -0.42218 -    0.62857i
```

Процедура **`lqrd`** позволяет спроектировать *дискретный* оптимальный линейно-квадратичный регулятор, минимизирующий непрерывный функционал (6.14). Обращение к процедуре **`[K,S,E] =lqrd(A,B,Q,R,N,Ts)`** , где Ts - заданный период дискретизации, приводит к расчету матрицы K статического звена (6.13) обратной связи по вектору состояния системы. При этом модель системы должна быть задана в конечно-разностной форме (6.20).

Проектирование оптимального линейного дискретного регулятора для *дискретной* системы с использованием *дискретного* функционала (6.21) 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 (6.20). Матрица S в этом случае представляет собой решение уравнения Риккати в виде

$$A'SA - S - (A'SB+N)(R+B'SB)\,(B'SA+N') + Q = 0. \tag{6.22}$$

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

$$\frac{dx}{dt} = A \cdot x + B \cdot u + G \cdot w\,; \qquad \text{(уравнения состояния)} \tag{6.23}$$

$$y = C \cdot x + D \cdot u + H \cdot w + v\,, \quad \text{(уравнение измерения)} \tag{6.24}$$

с известными входами u, шумовым процессом w, шумом измерения v и шумами ковариаций

$$E\{ww'\} = Qn, \quad E\{vv'\} = Rn, \quad E\{wv'\} = Nn, \tag{6.25}$$

фильтр KEST имеет вход [u;y] и генерирует оптимальные оценки y_e и x_e соответственно величин y и x путем решения уравнений:

$$\frac{dx_e}{dt} = A \cdot x_e + B \cdot u + L \cdot (y - C \cdot x_e - D \cdot u)\,; \tag{6.26}$$

$$y_e = C \cdot x_e + D \cdot u\,. \tag{6.27}$$

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

$$AP + PA' - (PC'+G*N)R^{-1}\ (CP+N'*G') + G*Q*G' = 0\ . \qquad (6.28)$$

Если система SYS задана конечно-разностными уравнениями

$$x[n+1] = Ax[n] + Bu[n] + Gw[n] \qquad \{\text{уравнение состояния}\} \qquad (6.29)$$

$$y[n] = Cx[n] + Du[n] + Hw[n] + v[n] \quad \{\text{уравнение измерения}\} \qquad (6.30)$$

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

$$x[n+1|n] = Ax[n|n-1] + Bu[n] + L(y[n] - Cx[n|n-1] - Du[n])$$

$$y[n|n] = Cx[n|n] + Du[n] \qquad (6.31)$$

$$x[n|n] = x[n|n-1] + M(y[n] - Cx[n|n-1] - Du[n])$$

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

$P = E\{(x - x[n|n-1])(x - x[n|n-1])'\}$ (решение уравнений Риккати)

$Z = E\{(x - x[n|n])(x - x[n|n])'\}$ (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

$$A*X*A' - X + Q = 0.$$

6.7. Вопросы для самопроверки

1. Что такое линейная стационарная система (ЛСС)?
2. Какие задачи можно решить с помощью пакета **CONTROL**?
3. Какой класс объектов составляет основу пакета **CONTROL**?
4. Какими способами и средствами обеспечивается ввод информации об ЛСС-системе?
5. Как преобразовать LTI-объект из одной формы его представления в другую?
6. Какими средствами в пакете **CONTROL** обеспечивается анализ системы?
7. Какие интерактивные средства предусмотрены в пакете **CONTROL**?
8. Какими средствами синтеза систем обладает пакет **CONTROL**?
9. Как обеспечить получение информации о системе?

Урок 7. Основы визуального моделирования динамических систем (пакет 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 линейных и нелинейных систем, причем составление «программы» и ввод характеристик исследуемых систем осуществлять в диалоговом режиме, *путем графической сборки на экране схемы соединений элементарных* (стандартных или пользовательских) *блоков*. В результате такой сборки получается модель исследуемой системы, которую в дальнейшем будем называть S-моделью и которая сохраняется в файле с расширением **.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. В процессе моделирования есть возможность наблюдать за процессами, которые происходят в системе. Для этого используются специальные блоки («обзорные окна»), входящие в состав библиотеки Simulink. Библиотека Simulink может быть пополнена пользователем за счет разработки собственных блоков.

7.1. Библиотека SIMULINK - ядро пакета Simulink

Основой пакета Simulink, его ядром является библиотека SIMULINK, в которой сосредоточены все основные средства, графические блоки и программы, позволяющие составлять модели сложных динамических систем, описываемых сложными нелинейными дифференциальными уравнениями.

7.1.1. Запуск Simulink

Запуск Simulink можно осуществить из командного окна MatLAB, нажав соответствующую пиктограмму в линейке инструментов (рис. 7.1).

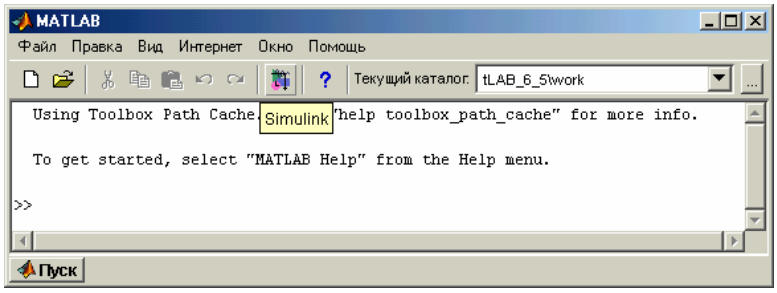


Рис. 7. 1. Вызов пакета Similink

При этом на экране должно появиться окно **Simulink Library Browser** браузера библиотек **Simulink** (рис. 7.2).

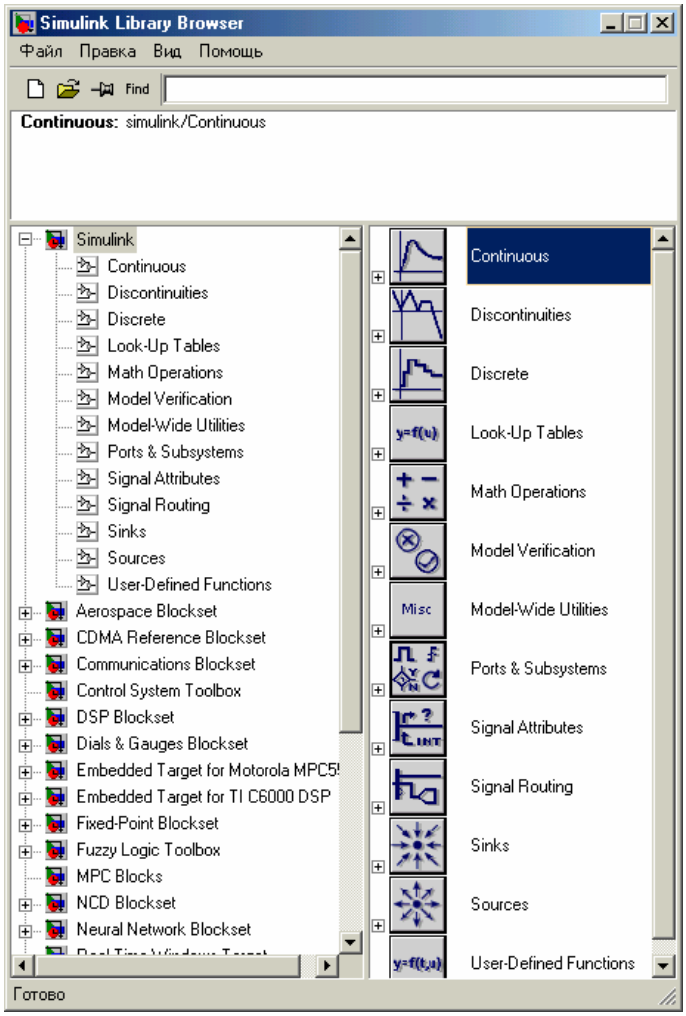


Рис. 7.2. Окно Simulink Library Browser

В левой половине окна браузера приведен перечень библиотек, подключенных в состав **Simulink**, а в правой – перечень разделов соответствующей библиотеки, либо изображения блоков соответствующего раздела.

Чтобы начать сборку блок-схемы моделируемой системы необходимо вызвать на экран отдельное окно, в котором и будет осуществляться эта сборка. Для этого достаточно в командном окне MATLAB вызвать команду `Файл` ► `Новый` ► `Модель`. При этом появляется новое (пустое) окно (рис. 7.3) **untitled** (окно, куда будет выводиться схемное представление моделируемой системы, новой S-модели, MDL-файла).

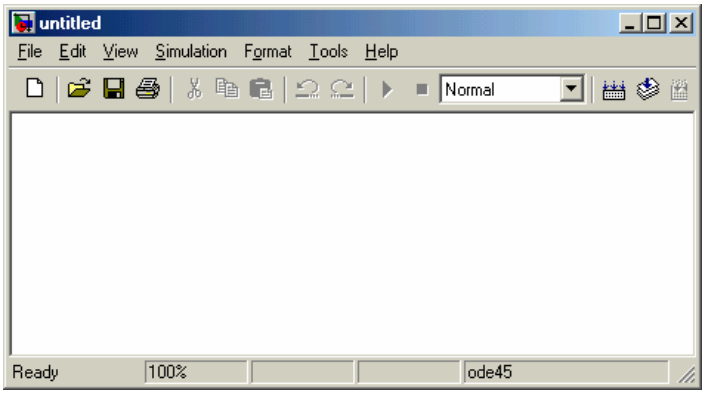


Рис. 7. 3. Окно блок-схемы

Окно блок-схемы модели (рис. 7.3) содержит строку меню, панель инструментов (версии MatLAB 5.2 и 5.3) и рабочее поле.

Меню **File** (Файл) содержит команды работы с MDL-файлами, меню **Edit** (Правка) - команды редактирования блок-схемы, а меню **View** (Вид) - команды изменения внешнего вида окна. В меню **Simulation** (Моделирование) содержатся команды управления процессом моделированием, а в меню **Format** (Формат) - команды редактирования формата (т. е. внешнего вида) блоков схемы и блок-схемы в целом. Меню **Tools** (Инструменты) содержит некоторые дополнительные сервисные средства работы с Simulink-моделью.

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. Она является ядром пакета. Другие библиотеки не обязательны. Они включаются в состав общей библиотеки в зависимости от вкусов пользователя.

Чтобы познакомиться с составом какой либо из библиотек, достаточно воспользоваться контекстным меню. Для этого нужно нажать правую клавишу мыши, наведя ее курсор на название библиотеки в левой половине браузера библиотек. Например, проделав эту операцию с библиотекой ***SIMULINK***, получим на экране приглашение ***Открыть библиотеку "Simulink"***, нажав на изображение которого, получим на экране новое окно (рис. 7.4), в котором представлены графические изображения разделов этой библиотеки.

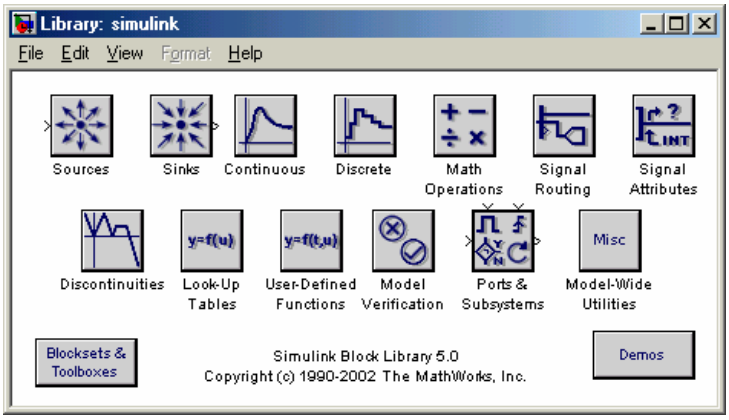


Рис. 7. 4. Окно Library: simulink

7.1.2. Общая характеристика библиотеки блоков *SIMULINK*

Библиотека блоков SIMULINK - это набор визуальных объектов, используя которые, можно, соединяя отдельные модули между собою линиями функциональных связей, составлять функциональную блок-схему любого устройства.

Библиотека SIMULINK (рис. 7.5) состоит из 15 разделов. Тринадцать из них являются главными и не могут изменяться пользователем:

Sources	Источники
Sinks	Приемники
Continuous	Непрерывные элементы
Discrete	Дискретные элементы
Math Operations	Математические операции
Signals Routing	Пересылка сигналов
Signals Attributes	Атрибуты сигналов
Discontinuities	Нелинейные элементы
Look Up Tables	Табличные функции
User Defined Functions	Функции, определяемые пользователем
Model Verification	Проверка моделей
Ports & Subsystems	Порты и подсистемы
ModelWide Utilities	Утилиты расширения модели

Четырнадцатый раздел - **Blocksets & Toolboxes** (Наборы блоков и инструменты) - содержит дополнительные блоки, включенные в рабочую конфигурацию пакета. Пятнадцатый раздел **Demos** позволяет вызвать демонстрационные программы для иллюстрации работы блоков.

Любая блок-схема моделируемой системы должна необходимо включать в себя один или несколько блоков-источников, генерирующих сигналы, которые, собственно, и вызывают «движение» моделируемой системы, и один или несколько блоков-приемников, которые позволяют получить информацию об выходных сигналах этой системы (увидеть результаты моделирования).

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, дифференциаторы);
- 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 Operations** (математические операции) - 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 **Signals Routing** включены блоки, обеспечивающие разного рода пересылки сигналов, таких как переключения сигналов, объединение нескольких сигналов в шину, разведение сигналов из шины и т. п.

Раздел **Signals Attributes** состоит из блоков обеспечивающих либо определение, либо изменение некоторых атрибутов сигнала (таких как размер сигнала (количество элементов в векторном или матричном сигнале), тип данных, начальные условия, скорость передачи данных и т. п.).

Раздел **Discontinuities** (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.

В разделе **Look Up 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.

User Defined Functions (функции, определяемые пользователем) содержит блоки, на основе которых пользователь может создавать собственные S-блоки, выполняющие необходимые ему функции.

В разделе **Model Verification** сосредоточены блоки, позволяющие осуществлять проверку некоторых динамических свойств S-модели.

Большинство блоков раздела **Ports & Subsystems** (порты и подсистемы) предназначено для разработки сложных S-моделей, содержащих модели более низкого уровня (подсистемы), и обеспечивают установление необходимых связей между несколькими S-моделями.

ModelWide Utilities включает блоки, позволяющие линеаризовать динамическую модель и оформить документацию к модели.

Чтобы перейти в окно соответствующего раздела библиотеки, в котором расположены графические изображения блоков, достаточно дважды щелкнуть мышью на значке 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)

Вначале рассмотрим блоки раздела **Sinks**, так как именно они обеспечивают визуализацию результатов, получаемых при моделировании, и без них невозможно проконтролировать правильность работы того или иного блока или моделируемой системы в целом.

После перехода к разделу **Sinks** на экране появляется окно этого раздела, изображенное на рис. 7. 5.

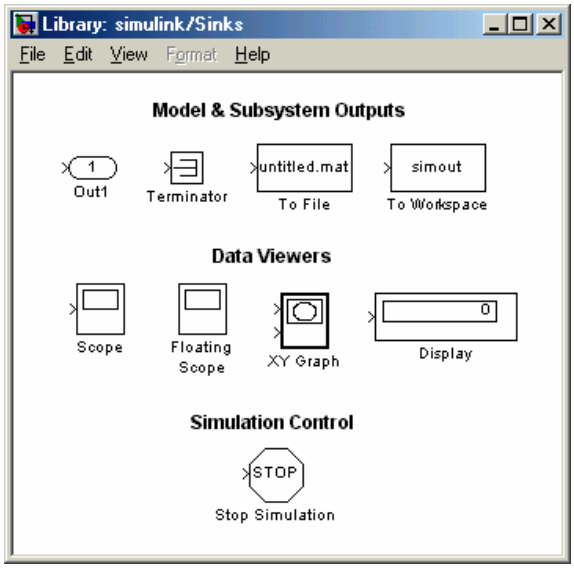


Рис. 7. 5. Окно раздела Sinks

Из его рассмотрения вытекает, что в разделе размещены три группы блоков:

1) блоки, которые при моделировании играют роль обзорных окон; к ним относятся:

Scope блок с одним входом, который выводит в графическое окно график зависимости величины, подаваемой на его вход, от модельного времени;

Floating Scope блок с одним входом, с аналогичными функциями;

XYGraph блок с двумя входами, который обеспечивает построение графика зависимости одной моделируемой величины (второй сверху вход) от другой (первый вход);

Display блок с одним входом, предназначенный для отображения численных значений входной величины;

2) блоки для пересылки и сохранения результатов:

Out выходной порт для вывода результатов вне модели;

Terminator (заглушка) порт для вывода результата «в никуда»;

To File , который обеспечивает сохранение результатов моделирования на

диске в MAT файле (с расширением .mat);

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.

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. 6).

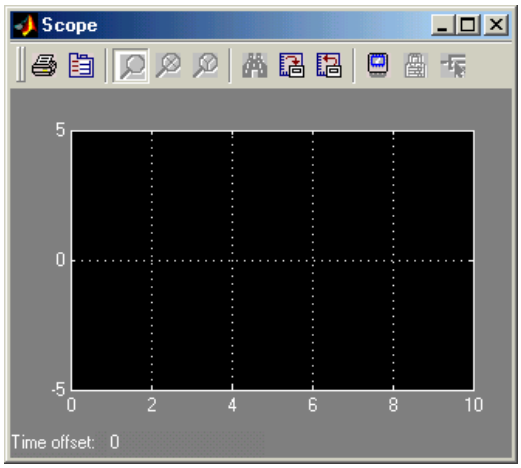


Рис. 7. 6. Окно блока Scope

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 изменения 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 11 значков такого назначения (слева направо):

- распечатка содержимого окна Scope на принтере;
- вызов диалогового окна настраивания параметров блока **Scope**;
- 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 установок параметров осей;
- восстановление установок параметров осей;
- включение холостого подсоединения блока;
- шлюз селектора сигналов;
- селектор сигналов.

Третий, четвертый и пятый значки являются альтернативными, т. е. в каждый момент времени может быть использован лишь один из них. Они не активны до тех пор, пока нет графика в окне Scope. Активны с самого начала лишь первые два, шестой, седьмой и девятый значки. Нажатие второго значка приводит к появлению окна 'Scope' parameters настраивания параметров (свойств) блока (рис. 7. 7).

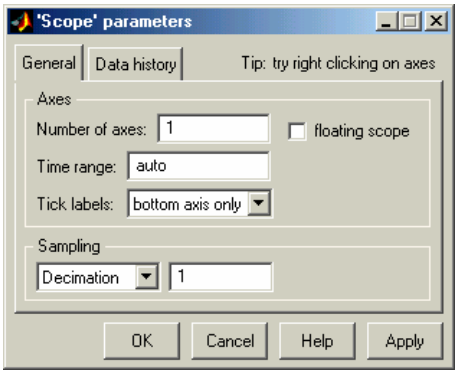


Рис. 7. 7. Окно настройки блока Scope

Это окно имеет две вкладки:

- General (Общие), позволяющая установить параметры осей;

- Data history (Представление данных), предназначенная 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 его список, 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. 8.

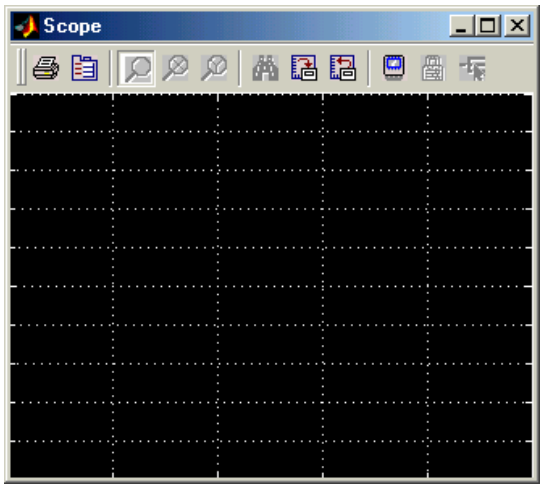


Рис. 7. 8. Окно Scope при установке значения none параметра Tick Labels

Флажок *Floating scope* предназначен для отключения входов в блок **Scope**. Для этого достаточно отметить его, щелкнув в нем мышью. Если флажок установлен, то **Scope** отображается как блок без входа, и если он был связан по входу с другими блоками, то эти связи «обрываются». Тот же эффект оказывает нажатие значка с тем же названием в линейке инструментов блока.

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. 7.9) позволяет задать максимальное количество (начиная с конца) элементов массивов данных, которые используются для построения графиков в окне Scope (окошко рядом с надписью *Limit data rows to last* (Максимальный размер строки данных)).

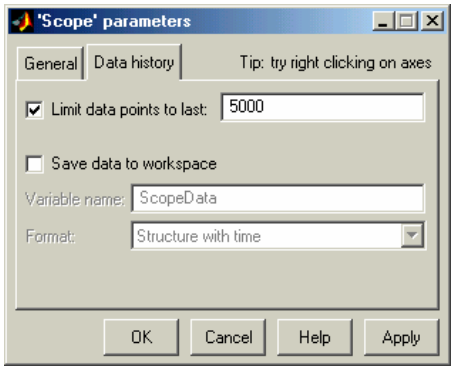


Рис. 7. 9. Вкладка Data history окна настройки блока Scope

Другие окошки этой вкладки становятся активными, если установить флажок в переключателе *Save data to work space* (Записать данные в рабочее пространство). 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 – *Array* (Массив, матрица), *Structure* (Структура) и *Structure with time* (Структура с временем).

Любые изменения, сделанные в окне 'Scope' parameters, изменяют окно Scope лишь в случае, если после введения этих изменений нажата кнопка *Apply* в нижней части окна.

Продемонстрируем работу блока **Scope** на простейшем примере.

Перетянем в окно будущей блок-схемы из окна раздела **Sources** блок **Sine Wave** (Волна синусоиды), а из окна раздела **Sinks** – блок **Scope** и соединим выход первого блока со входом второго. Получим схему, показанную на рис. 7.10.

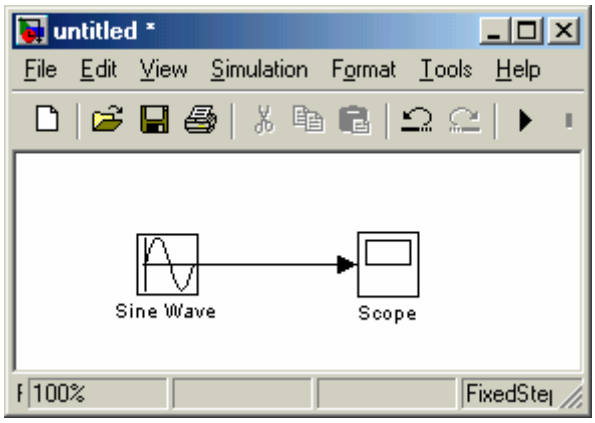


Рис. 7. 10. Простейшая блок-схема с блоком ***Scope***

Вызовем в окне этой блок схемы Simulation ► Start, а затем дважды щелкнем на изображении блока **Scope**. На экране возникнет графическое окно этого блока с изображением графика изменения во времени гармонически изменяющегося сигнала (рис. 7. 11).

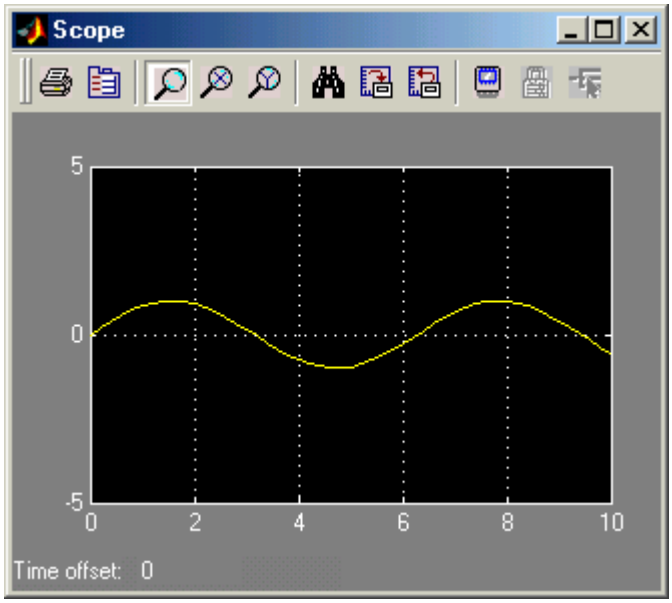


Рис. 7. 11. Окно ***Scope*** с изображением синусоиды

Блок XY Graph

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. 7.12), которое позволяет установить границы изменений обеих входных величин, в которых будет построен график зависимости второй величины от первой, а также задать дискрет по времени.

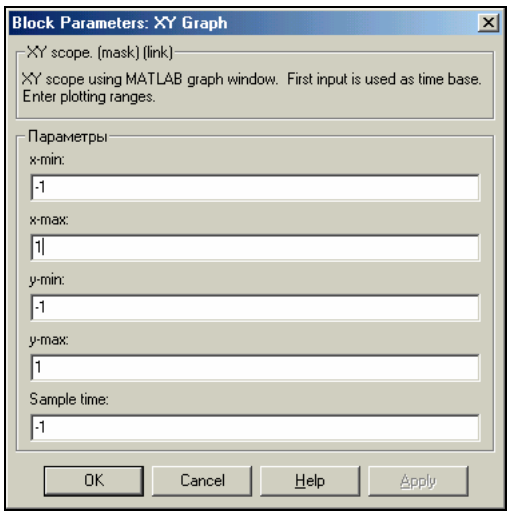


Рис. 7. 12. Окно настраивания блока XY Graph

Приведем пример использования блока **`XY Graph`**. Для этого перетянем в окно блок-схемы изображение этого блока из окна Library: Simulink/Sinks, а из окна Library:Simulink/Sources - два блока-источника **`Clock`** и **`Sine Wave`**. Соединим выходы блоков-источников с входами блока **`XY Graph`**. Получим блок-схему, приведенную на рис. 7.13.

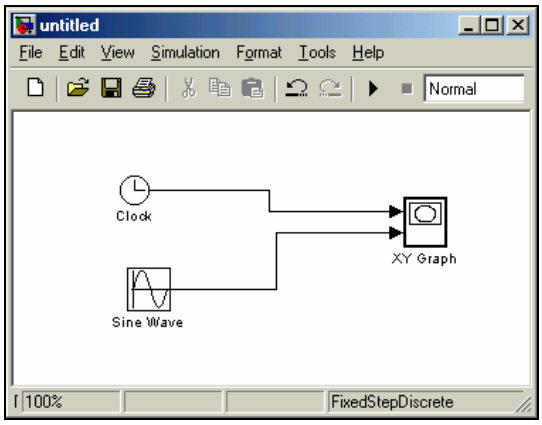


Рис. 7. 13. Блок-схема проверки работы блока XY Graph

Прежде чем запустить эту схему на моделирование, необходимо настроить блок **`XY Graph`**, вводя в его окне настраивания (рис. 7. 12) ожидаемые диапазоны изменения величины ***x*** (в нашем случае – времени: x-min=0, x-max=10) и ***y*** (синусоидального сигнала: ymin = -1, ymax = 1). Отметим, что в случае использования блока **`Scope`** ввода диапазонов изменения величин не требуется, - они устанавливаются автоматически.

Вызывая Simulation ► Start, получим на экране обзорное окно XY Graph и в нем будет построено изображение, представленное на рис. 7.14.

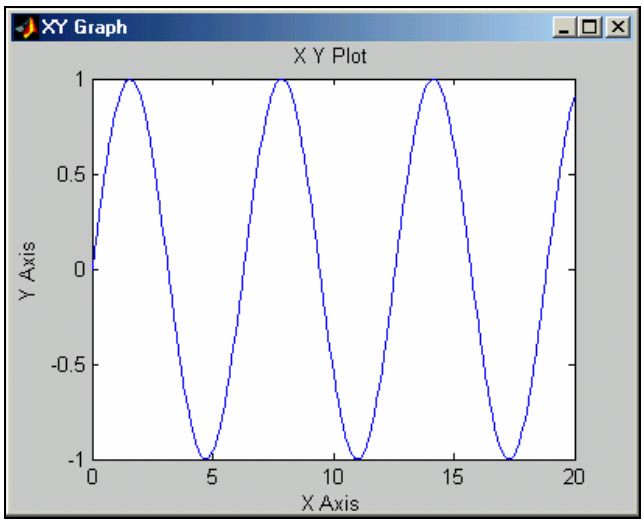


Рис. 7. 14. Обзорное окно блока XY Graph с графиком синусоиды

Блок Display

Этот блок предназначен для вывода на экран численных значений величин, которые фигурируют в блок-схеме.

Перетянем блок **`Display`** в окно блок схемы и дважды щелкнем на нем. Появится окно настраивания блока (рис. 7.15).

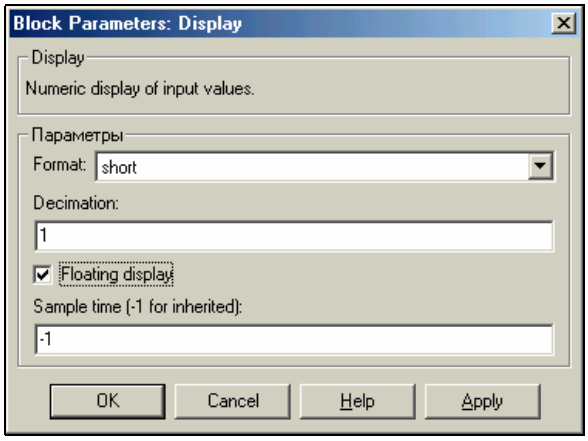


Рис. 7. 15. Окно настраивания блока Display

Как видим, блок имеет 4 параметра настраивания. Поле *Format* задает формат вывода чисел; вид формата избирается с помощью списка, содержащего 5 пунктов: *short*, *long*, *short_e*, *long_e*, *bank*. Поле ввода *Decimation* позволяет задать периодичность (через сколько шагов времени) вывода значений в окне **`Display`**. Переключатель *Floating display* (если сбросить флажок) позволяет определять блок **`Display`** как блок без входа, обрывая его связи. Четвертый параметр *Sample Time* используется только для дискретных во времени процессов. Установленное по умолчанию его значение (на рис. 7.15 – (-1)) для непрерывных процессов и блоков рекомендуется не изменять.

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.

Для примера создадим блок-схему (рис. 7.16) из двух элементов – блока-источника **`Constant`** и блока-приемника **`Display`**. Вызвав окно настраивания блока **`Constant`** (рис. 7.17), 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** меню Simulation, получим изображение окна блок-схемы, показанное на рис. 7.16.

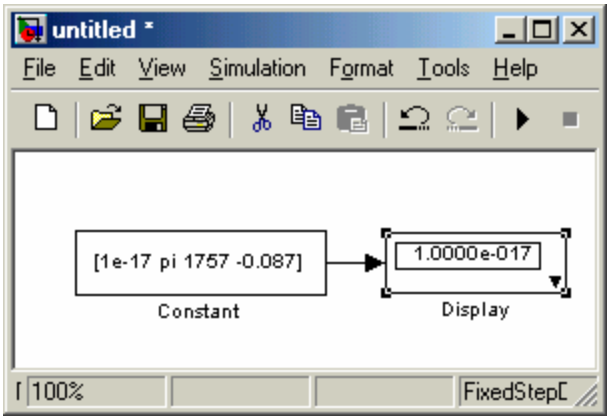


Рис. 7. 16. Блок-схемо проверки функционирования блока Display

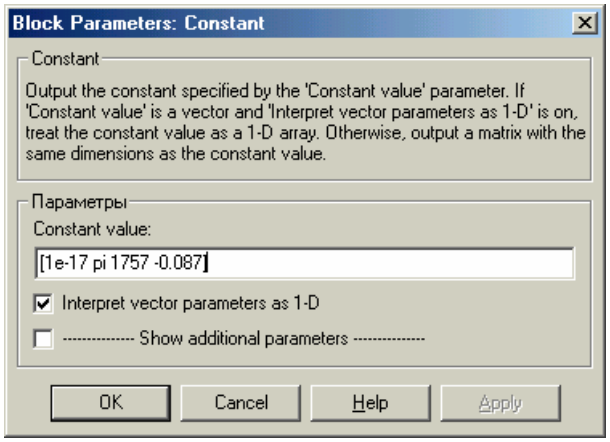


Рис. 7. 17. Окно настраивания блока Constant

Растягивая изображение блока **Display** на блок-схеме, получим картину, представленную на рис. 7.18.

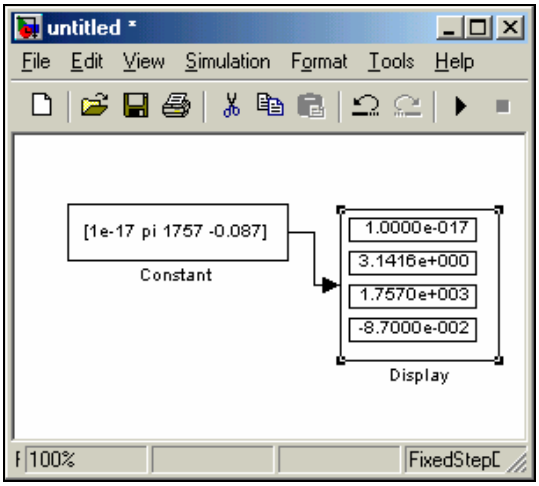


Рис. 7. 18. Полное изображение содержимого обзорного блока Display

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. 7.19):

Filename 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 (через сколько шагов времени) записи данных в файл;

Sample Time размер дискрета времени для данного блока.

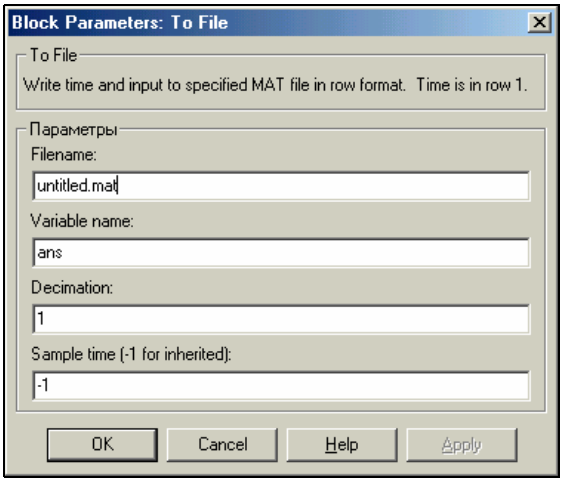


Рис. 7. 19. Окно настраивания блока To File

Следует отметить, что значения данных, которые подаются в 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.

Блок имеет 4 параметра настраивания (см. рис. 7.20):

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 оно задается константой ***inf***, т. е. данные регистрируются на всем интервале моделирования;

Decimation и имеют то же смысл, что и ранее;

Sample Time

Поле **`Save format`** (Формат записи) позволяет выбрать один из трех вариантов записи данных: *Array* (Массив, матрица), *Structure* (Структура) и *Structure with time* (Структура со временем).

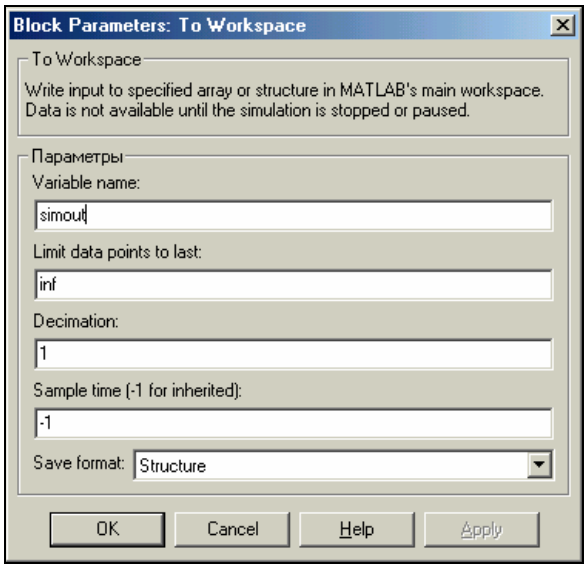


Рис. 7. 20. Окно настраивания блока To Workspace

7.1.4. Раздел Sources (Источники)

После выбора раздела **Sources** библиотеки **SIMULINK** на экране появится окно, показанное на рис. 7.21.

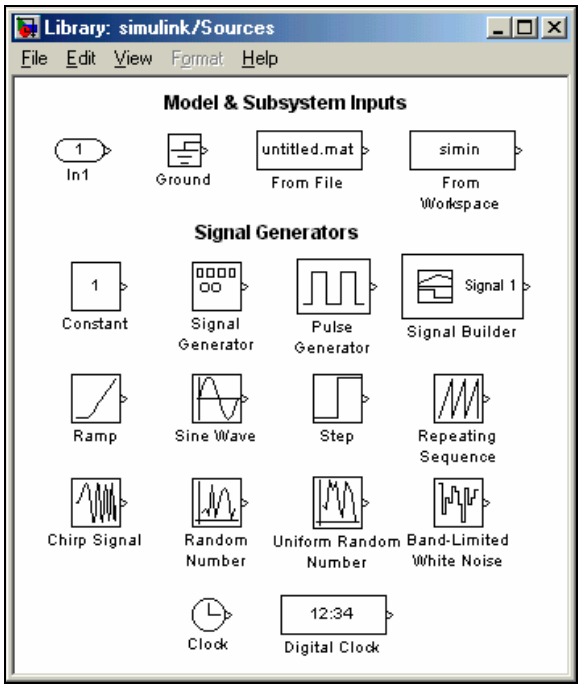


Рис. 7. 21. Содержимое раздела Sources библиотеки SIMULINK

Как видим, в этом разделе библиотеки блоки разделены на две группы:

- Входы моделей и подсистем;

- Генераторы сигналов.

В первую группу входят блоки, обеспечивающие поступление сигналов в S-модель или подсистему Simulink извне – из других S-моделей, рабочего пространства или MAT-файлов:

In – входной порт S-модели или подсистемы; он обеспечивает поступление в подсистему сигналов из системы более высокого порядка, формируя вход этой подсистемы; в S-модели верхнего уровня обеспечивает поступление в нее процесса из рабочего пространства;

Ground - (Заземление) создает нулевой по уровню сигнал;

From File - предназначен для введения в S модель данных, которые сохраняются на диске в MAT-файле;

From Workspace - обеспечивает введение в модель данных непосредственно из рабочего пространства 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.

Вторую группу образуют блоки, формирующие выходную величину как заданную функцию времени. Их можно рассматривать как своеобразные генераторы сигналов заданного вида. Сюда включены 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;

Pulse Generator генератор непрерывных прямоугольных импульсов;

Signal Builder создает (генерирует) один или несколько процессов, аппроксимируемых отрезками прямых (до пяти отрезков в каждом);

Ramp создает линейно восходящий (или нисходящий) сигнал;

Sine Wave генерирует гармонический сигнал;

Step генерирует сигнал в виде одиночной ступеньки (ступенчатый сигнал) с заданными параметрами (начала ступеньки и ее высоты);

Repeating Sequence генерирует периодическую последовательность;

Chirp Signal генератор гармонических колебаний с частотой, которая линейно изменяется с течением 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;

Limited White Noise генератор белого шума с ограниченной полосой частот.

Clock (Часы) источник непрерывного сигнала, пропорционального модельному времени;

Digital clock (Цифровые часы) формирует дискретный сигнал, пропорциональный 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. 7.17) - *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. 7.22.

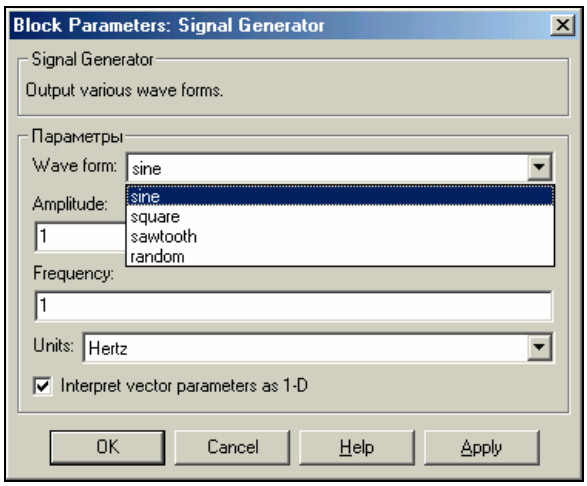


Рис. 7. 22. Окно настраивания блока Signal Generator

Как видно, в параметры настраивания входят:

Wave form форма волны позволяет выбрать одну из таких форм периодического процесса:
Sine - синусоидальные волны;
Square - прямоугольные волны;
Sawtooth - треугольные волны;
Random - случайные колебания;

Amplitude определяет значения амплитуды колебаний, которые генерируются;

Frequency задает частоту колебаний;

Units позволяет выбрать одну из единиц измерения частоты с помощью списка:
Hertz (в Герцах)
Rad/Sec (в радианах в секунду).

На рис. 7.23 показана простейшая блок-схема S-модели, которая состоит из блока **`Signal Generator`** и блока отображения **`XY Graph`**. Окно блока **`XY Graph`** после проведения моделирования при следующих параметрах настраивания: вид колебаний - *Sine*; амплитуда - 1; частота - 1 Герц (см. рис. 7. 22), - отображено на рис. 7. 24.

На рис. 7. 25.. 7. 27 представлены результаты, отображаемые в окне **`XY Graph`** в случае выбора соответственно прямоугольных, треугольных и случайных колебаний при тех же значениях остальных параметров настраивания.

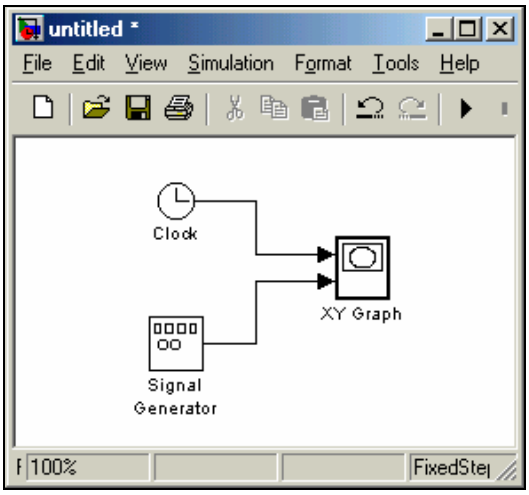


Рис. 7. 23. Блок-схема проверки функционирования блока Signal Generator

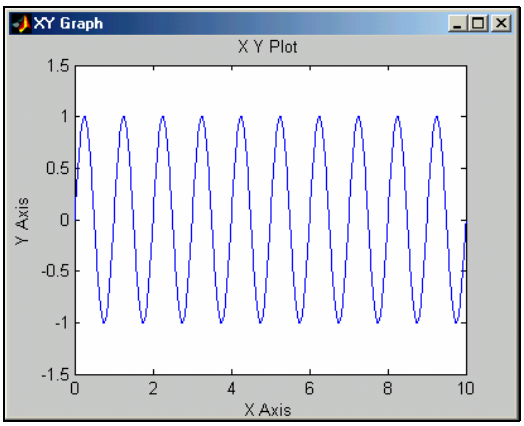


Рис. 7. 24. Результат генерирования синусоидального сигнала

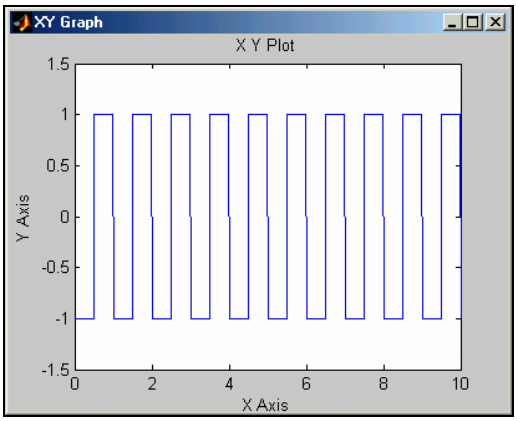


Рис. 7. 25. Результат генерирования прямоугольных волн

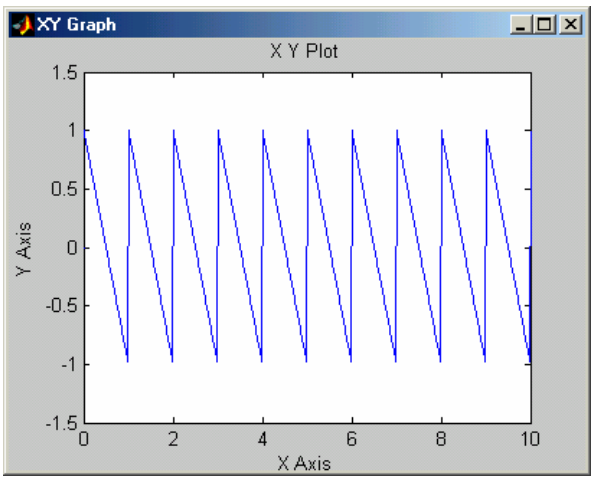


Рис. 7. 26. Результат генерирования треугольных волн

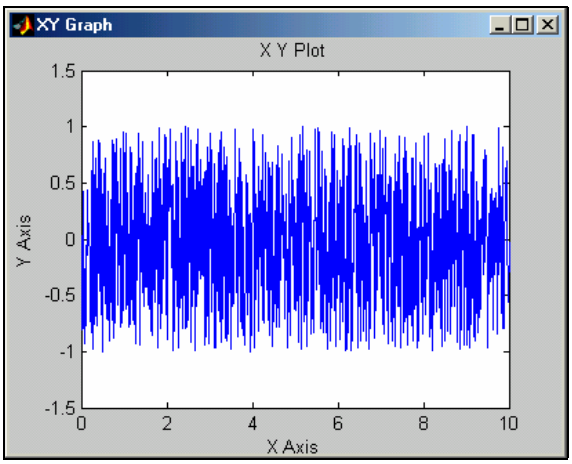


Рис. 7. 27. Результат генерирования случайных колебаний

Примечание. В последнем случае случайных колебаний генерируется сигнал, значения которого равномерно случайно распределены в диапазоне, указанном параметром *Amplitude*, а значения времени, в которых происходят скачкообразные изменения сигнала, разделены на величину шага моделирования, устанавливаемом в Simulation ► Simulation Parameters ► Solver

Блок Step

Блок обеспечивает формирование сигнала в форме ступеньки (или, как говорят, - ступенчатого сигнала).

Блок имеет 3 параметра настраивания (рис. 7. 28):

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.

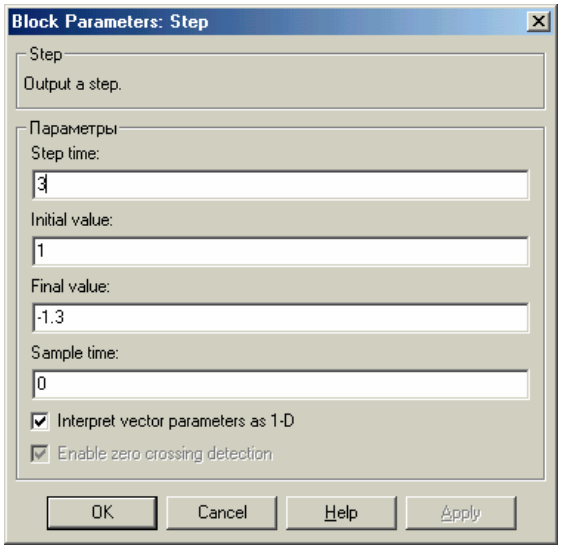


Рис. 7. 28. Окно настраивания блока Step

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. 7.29)

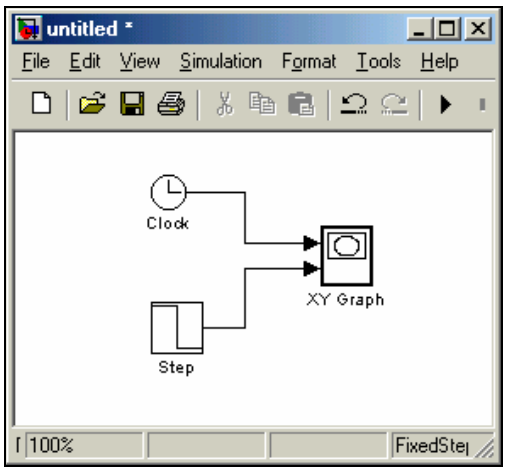


Рис. 7. 29. Блок-схема проверки функционирования блока Step

Установим такие параметры настраивания блока: *Step time* - 3, *Initial value* - 1, *Final value* – (-1.3). После активизации моделирования (Simulation ► Start) получим в окне **`XY Graph`** картину, представленную на рис. 7. 30.

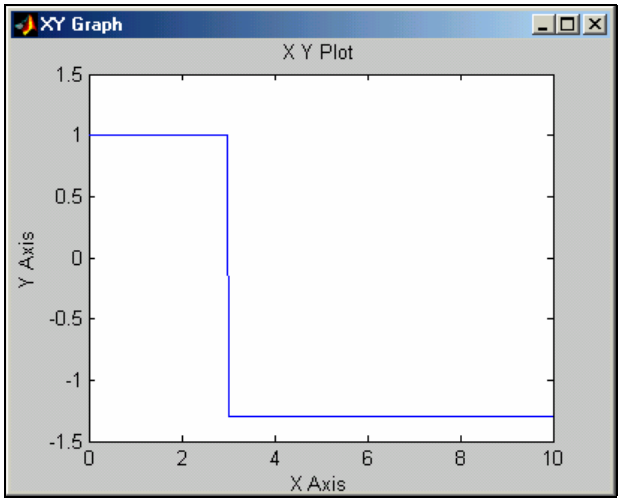


Рис. 7. 30. Результат работы блока Step

Блок Ramp

Блок формирует непрерывно нарастающий сигнал и имеет такие параметры настраивания (рис. 7. 31):

Slope	значение скорости нарастания сигнала;
Start time	время начала нарастания сигнала;
Initial output	значение сигнала до момента начала его нарастания.

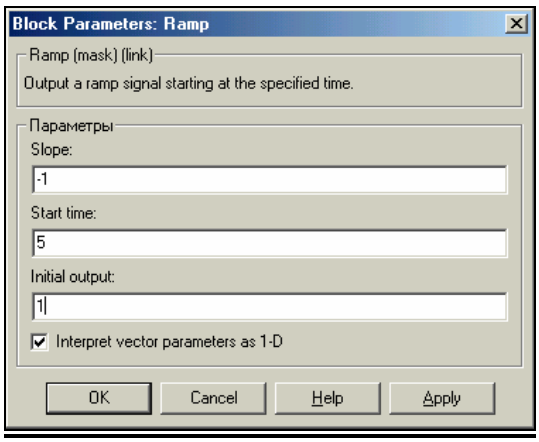


Рис. 7. 31. Окно настраивания блока Ramp

На рис. 7.32 приведен пример результата применения блока ***Ramp*** при значениях параметров, указанных на рис. 7.31.

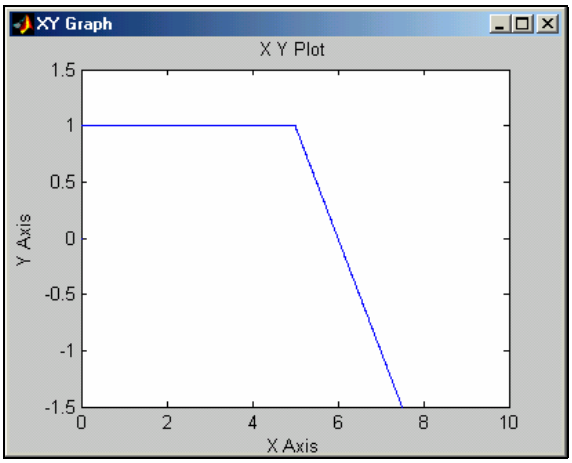


Рис. 7. 32. Результат работы блока Ramp

Блок Sine Wave

Блок ***Sine Wave*** имеет такие параметры настройки (рис. 7. 33):

Sine Type - задает тип задания синусоидальной волны:
Time based - для непрерывного сигнала (аргумент- время);
Sample based - для дискретного времени (аргумент - число дискретов времени) ;

Amplitude - задает амплитуду синусоидального сигнала;

Bias – задает смещение (постоянную составляющую синусоиды);

Frequensy (rad/sec) - задает частоту колебаний в радианах в секунду;

Phase (rad) - позволяет установить начальную фазу в радианах;

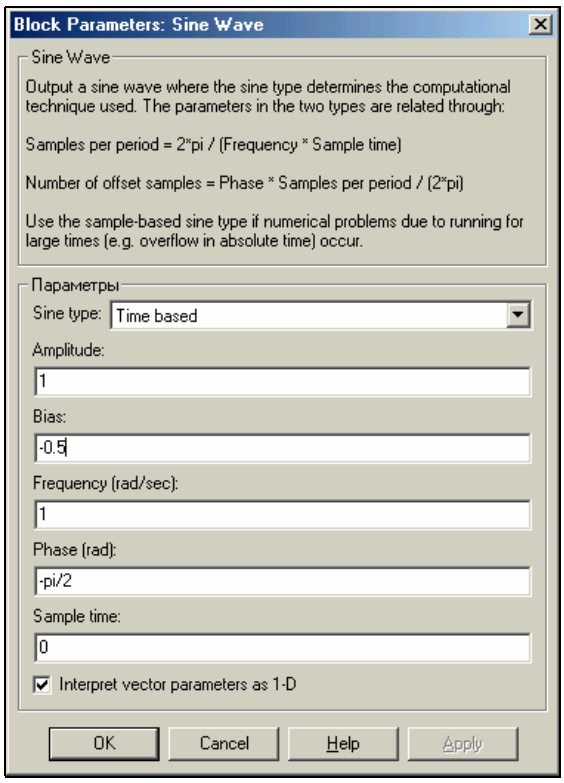


Рис. 7. 33. Окно настраивания блока Sine Wave

Результат применения блока при значениях параметров настраивания, приведенных на рис. 7. 33 показан на рис. 7.34.

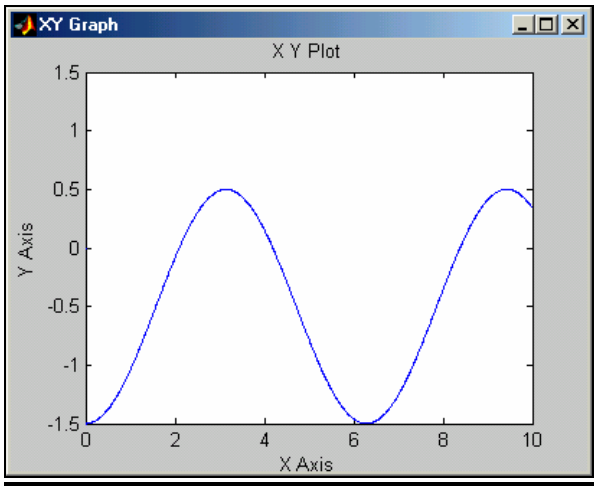


Рис. 7. 34. Результат работы блока Sine Wave

Отличия этого блока от генератора синусоидальных колебаний в блоке **`Signal Generator`** состоят в следующем:

- в блоке **`Sine Wave`** можно устанавливать произвольную начальную фазу;
- можно задать средний уровень синусоиды;
- в нем нельзя задать частоту в Герцах.

Блок Repeating Sequence

Этот блок содержит два параметра настройки (рис. 7.35):

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. 36 приведено изображение процесса, сгенерированного блоком **`Repeating Sequence`** при параметрах настройки, указанных на рис. 7. 35.

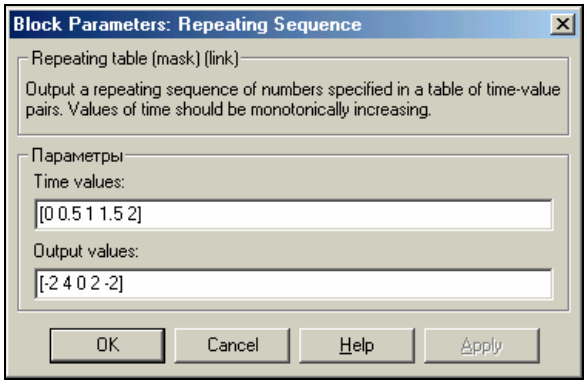


Рис. 7. 35. Окно настраивания блока Repeating Sequence

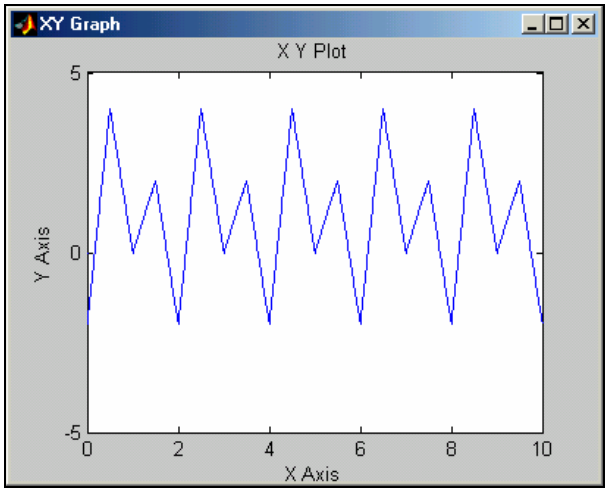


Рис. 7. 36. Результат работы блока Repeating Sequence

Блок Pulse Generator

Блок генерирует последовательности прямоугольных импульсов. В число настраиваемых параметров этого блока входят (рис. 7.37):

Pulse Type - задает тип задания импульсов:
Time based - для непрерывного сигнала (аргумент- время);
Sample based - для дискретного времени (аргумент - число дискретов времени) ;
Amplitude амплитуда сигнала (высота прямоугольного импульса);
Period размер периода сигнала;
Pulse width ширина импульса, в процентах от периода;
Phase delay величина задержки первого импульса относительно *t*=0 ;

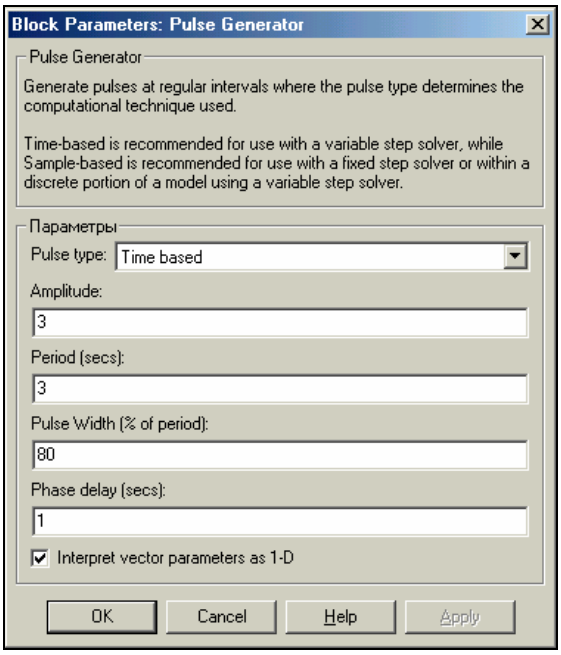


Рис. 7. 37. Окно настраивания блока Pulse Generator

Пример применения этого блока при значениях параметров, указанных на рис. 7.37, приведен на рис. 7.38.

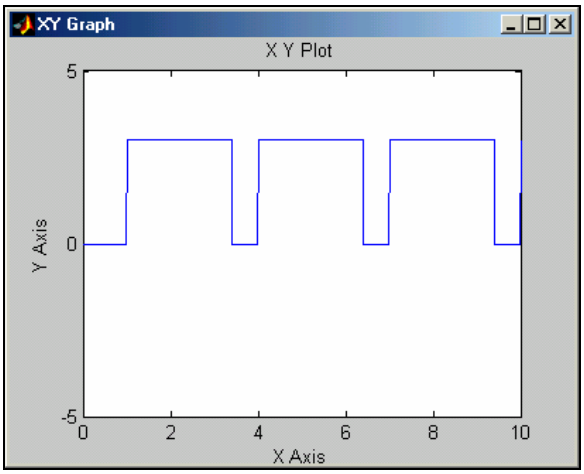


Рис. 7. 38. Результат работы блока Pulse Generator

Блок Chirp Signal

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. 39):

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.

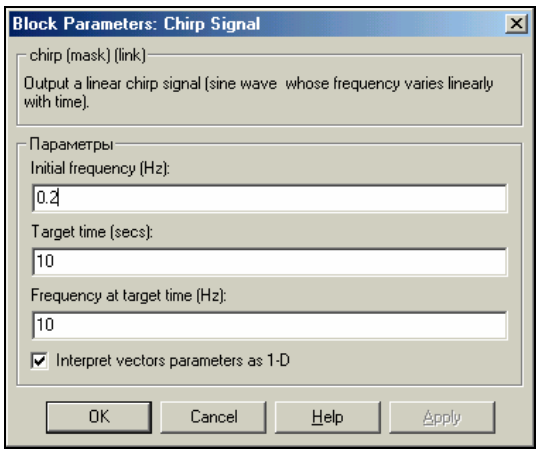


Рис. 7. 39. Окно настраивания блока Chirp Signal

Рис. 7. 40 демонстрирует результат использования блока при параметрах, указанных на рис. 7. 39.

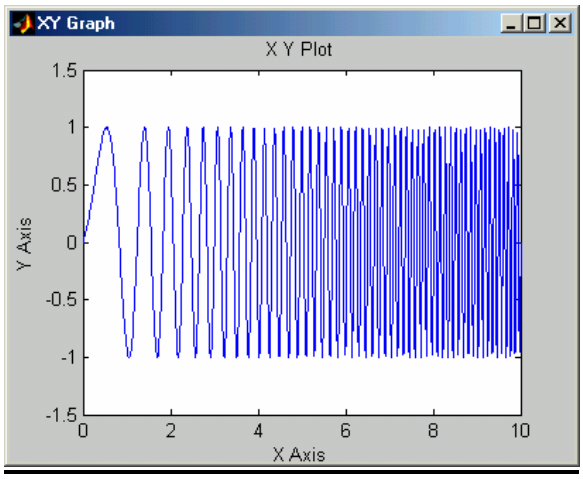


Рис. 7. 40. Результат работы блока Chirp Signal

Блоки генерирования случайных процессов (Random Number, Uniform Random Number и Band-Limited White Noise)

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. 41). Первые два - *Mean* и *Variance* - являются средним значением генерируемого процесса 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.

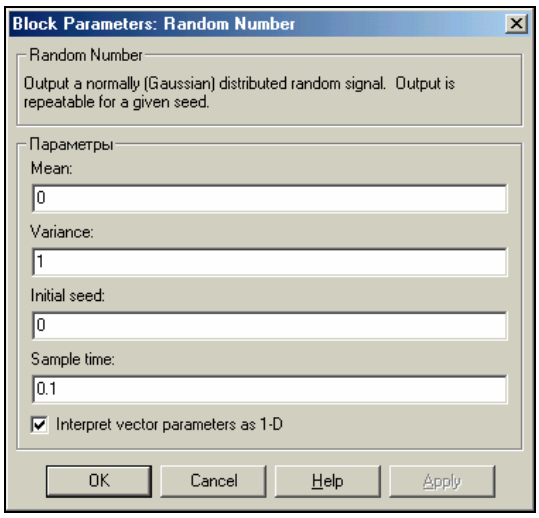


Рис. 7. 41. Окно настраивания блока Random Number

Блок **`Uniform Random Number`** формирует сигналы, значения которых в отдельные моменты времени являются случайной величиной, равномерно распределенной в заданном интервале. В число параметров настраивания блока входят (рис. 7. 42):

Minimum **`нижний предел случайной величины;`**
Maximum **`верхний предел;`**
Initial seed **`начальное значение базы генератора случайных чисел;`**
Sample time **`дискрет времени.`**

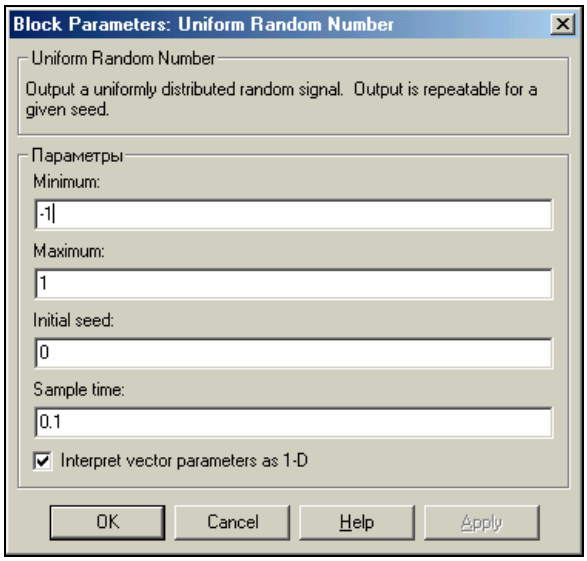


Рис. 7. 42. Окно настраивания блока Uniform Random Number

Блок **`Band-Limited White Noise`** формирует процесс в виде частотно-ограниченного белого шума. Параметры настраивания у него такие (рис. 7.43):

Noise power значение интенсивности (мощности) белого шума;
Sample time значение дискрета времени (определяет верхнее значение частоты процесса);
Seed начальное значение базы генератора случайной величины.

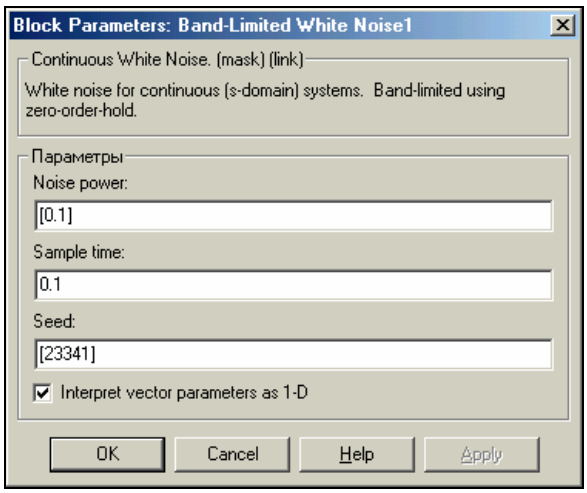


Рис. 7. 43. Окно настраивания блока Band-Limited White Noise

Сформируем блок-схему (рис. 7. 44) для иллюстрации работы блоков.

Чтобы в обзорном окне Scope можно было разместить три графика, каждый из которых отображал бы отдельный процесс, необходимо вызвать Scope ► 'Scope' parameters ► General и установить в окошке *Number of axes* число осей – 3. Вид блока ***Scope*** на блок- схеме при этом изменится, - на нем появятся изображения трех входов. Там же, в окошке *Tick labels* установим «all». Это необходимо, чтобы можно было проставить заголовки над каждым из выводимых трех графиков. Тексты заголовков надписываются на линиях, ведущих ко входам блока **`Scope`** (см. рис. 7.44).

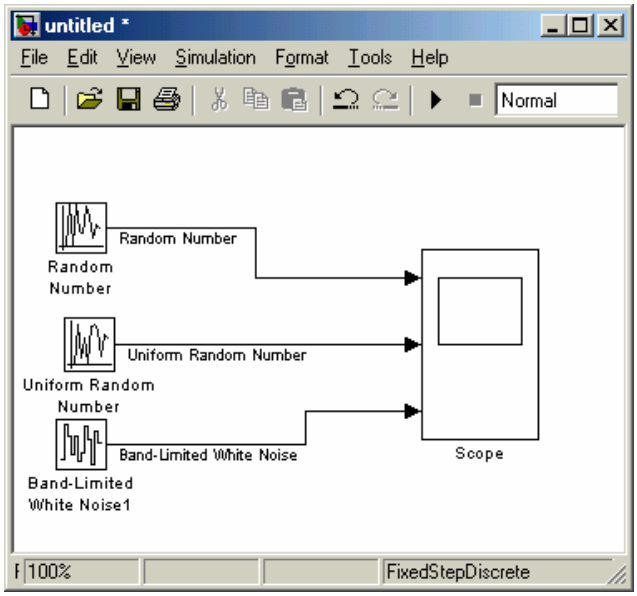


Рис. 7. 44. Блок-схема проверки функционирования блоков – генераторов случайных процессов

Установим в окнах настраивания блоков-генераторов случайных процессов одинаковую величину дискрета времени **`Sample time`** (0.1 с). Перед запуском в окне блок схемы выберем Simulation ► Simulation parameters ► Solver и установим:
- в окошке *Type* - значение *Fixed-step* из списка и *discrete (no continuous states)*;
- в окошке *Fixed step size* – значение шага изменения времени - 0.01 с.

Примечание. Следует различать шаг изменения модельного времени, который устанавливается при проведении моделирования в окне Simulation parameters и определяет собой интервалы времени, через которые производятся вычисления отдельных состояний моделируемой системы, и параметр *Sample Time* (дискрет времени), который определяет интервал времени, внутри которого выходная величина блока не изменяет своего значения.

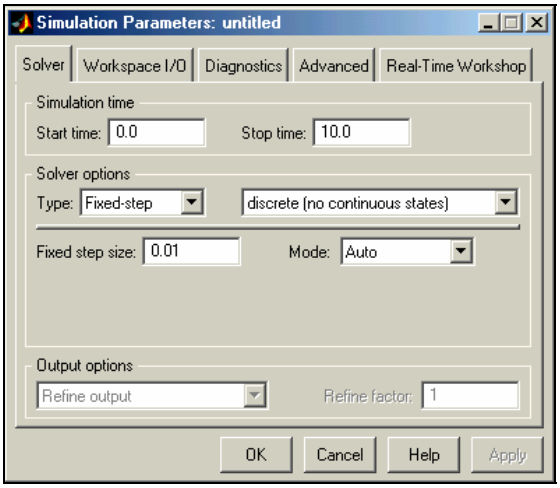


Рис. 7. 45. Окно настраивания параметров процесса моделирования

Осуществляя теперь моделирование, получим результаты, представленные на рис. 7.46.

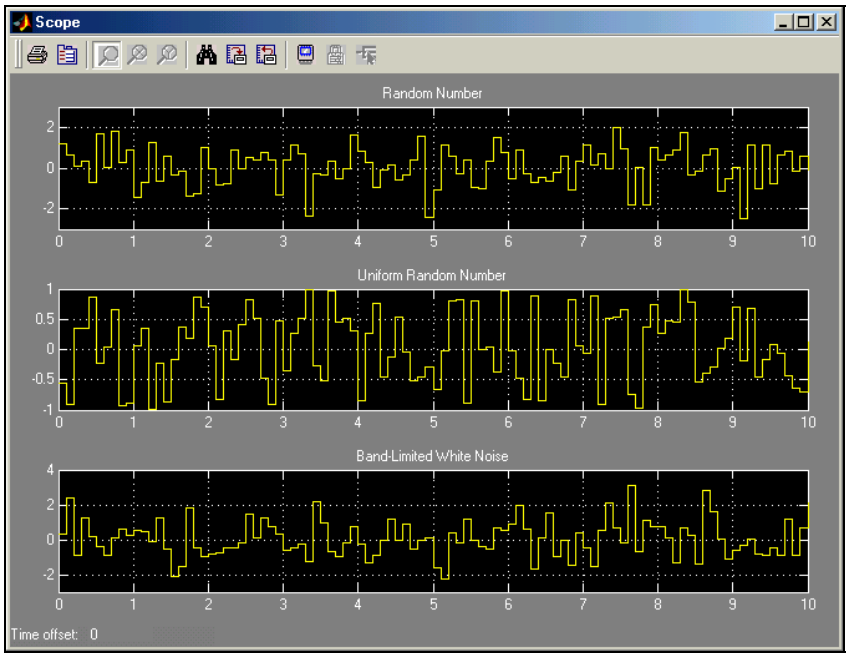


Рис. 7. 46. Результат генерирования случайных процессов

Как видим, моделируемые процессы сохраняют неизменные значения внутри интервалов времени, определяемых значением параметра *Sample Time* (дискрета времени). Случайное изменение значения процесса происходят скачкообразно на границе соседних дискретов времени.

Следует отметить существенное отличие первых двух блоков-генераторов от блока **`Band-Limited White Noise`**. Задание значения параметра *Sample Time*, отличного от нуля, в первых двух блоках является не обязательным. В блоке **`Band-Limited White Noise`** этот параметр определяет наибольшую частоту в спектре выходного сигнала. Она равна, в Герцах, величине, обратной *Sample Time*, а потому не может быть равной нулю.

На рис. 7. 47. показан результат моделирования первых двух блоков при значении *Sample Time* = 0.

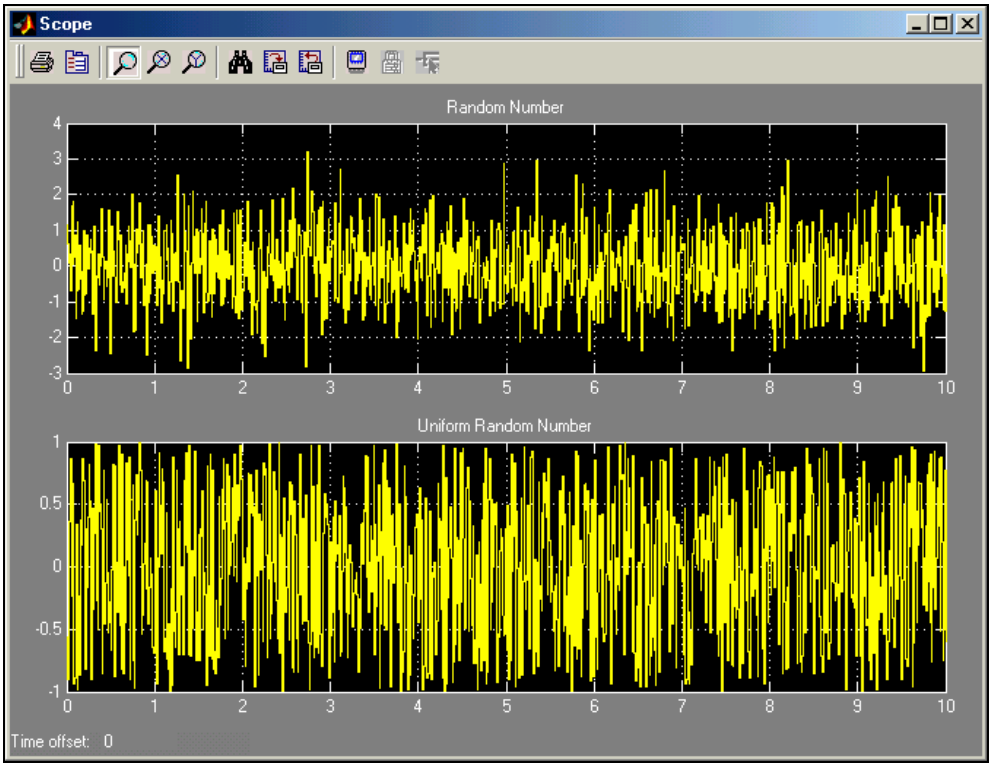


Рис. 7. 47. Результат генерирования случайных процессов при *Sample Time=0*

Из него следует, что изменение величины случайного процесса в этом случае происходит на стыке шагов моделирования (0.01 с), величина которых устанавливается в Simulation ► Simulation parameters ► Solver.

7.1.5. Раздел Continuous

Раздел **Continuous** (*Непрерывные элементы*) библиотеки содержит блоки (рис. 7.48), реализующие динамические звенья, описываемые линейными дифференциальными уравнениями с постоянными коэффициентами, т. е. линейные стационарные звенья:

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;
ZeroPole	задание звена через указание векторов значений его полюсов и нулей, а также значения коэффициента передачи;
Transport Delay	обеспечивает задержку сигнала на заданное количество шагов модельного времени, причем необязательно целое;
Variable Transport Delay	позволяет задавать управляемую извне величину задержки.

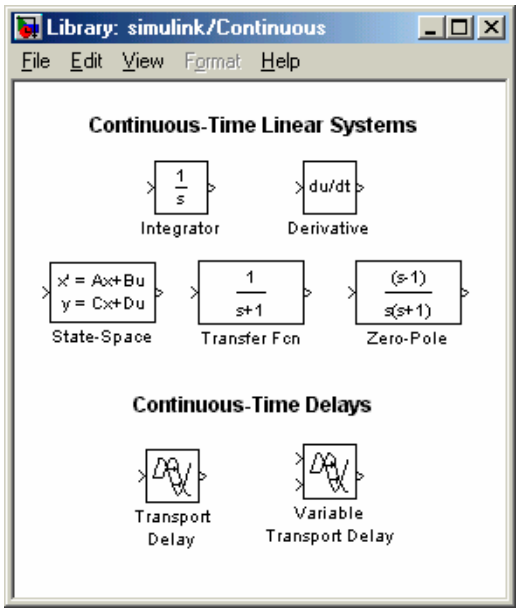


Рис. 7. 48. Содержимое раздела Continuous

Пользуясь этими блоками, можно, собирая в окне блок-схемы различные звенья и соединяя их между собой, составлять модели сложных систем автоматического управления, а затем осуществлять моделирование во времени поведения этих систем при различных заданных вхолных воздействиях. В этом случае моделирование в среде **Simulink** осуществляет те же функции, что и пакет **Control**. Однако этот процесс осуществляется в **Simulink** значительно проще, более нагляден и надежен, дает возможность проводить моделирование при значительно более сложных внешних воздействиях на систему.

Блок ***Integrator*** осуществляет интегрирование в непрерывном времени входной величины. Он имеет такие параметры настраивания (рис. 7.49):

External reset подключение дополнительного управляющего сигнала;
Initial condition source определение источника (внутренний или внешний) установления начального значения выходного сигнала;
Initial condition начальное значение выходной величины; значение вводится в строке редактирования или как числовая константа, или в виде выражения, которое вычисляется;
Limit output *Ограничение величины выхода* – флажок, определяющий, будут ли использоваться следующие три параметра настраивания;
Upper saturation limit верхнее предельное значение выходной величины; по умолчанию не ограничено (*inf*);
Lower saturation limit нижнее предельное значение выходной величины; по умолчанию параметр имеет значенит (*-inf*);
Show saturation port флажок *Показать порт насыщения*;
Show state port флажок *Показать порт состояния*;
Absolute tolerance допустимая предельная величина абсолютной погрешности.

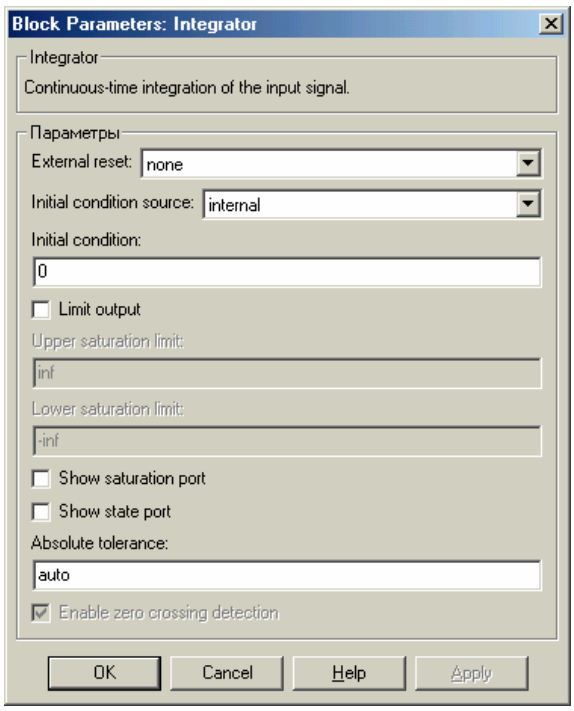


Рис. 7. 49. Окно настраивания блока Integrator

Параметр *External reset* может принимать такие значения (см. его список):

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 внутренняя установка начального значения выходной величины;
external установка 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 **`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 линейную 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;
`Discrete Filter`	блок задания дискретного звена через дискретную передаточную дробнорациональную функцию относительно 1/z;
`Discrete Transfer Fcn`	блок задания линейного дискретного звена через дискретную передаточную дробнорациональную функцию относительно z;
`Discrete Zero-Pole`	блок задания дискретного звена через указание значений нулей и полюсов дискретной передаточной функции относительно z.
`Discrete State-Space`	блок задания дискретного линейного звена матрицами его состояния;
`Memory`	задержка сигнала на один шаг модельного времени.
`Zero-Order Hold`	экстраполятор нулевого порядка;
`First-Order Hold`	экстраполятор первого порядка;

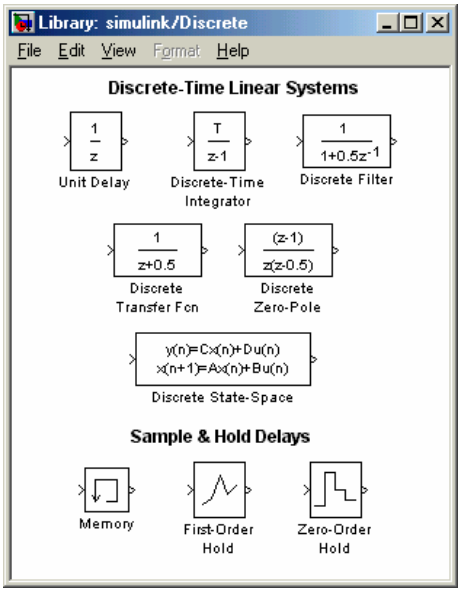


Рис. 7. 50. Содержимое раздела Discrete

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 **Continuous**. Отличия состоят в следующем. В блоке дискретного интегратора есть дополнительный параметр - метод численного интегрирования (*Integrator method*). С помощью списка можно выбрать один из трех методов: прямой метод 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.

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`.**

Блоки **Zero-Order Hold** и **First-Order Hold** служат для преобразования непрерывного сигнала в дискретный. На рис. 7. 51 приведен результат прохождения синусоидального сигнала через блоки **Zero-Order Hold** и **First-Order Hold.**

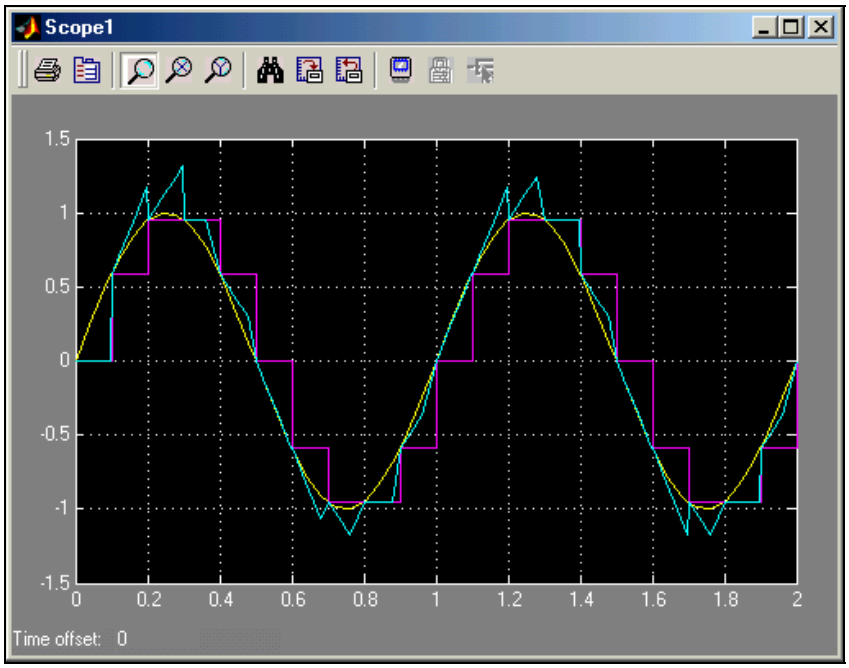


Рис. 7. 51. Результат прохождения синусоидального сигнала через блоки Zero-Order Hold и First-Order Hold

7.1.7. Раздел Math Operations

В разделе **Math Operations** (*Математические операции*) содержатся блоки, которые реализуют некоторые встроенные математические функции системы MatLAB (рис. 7.52).

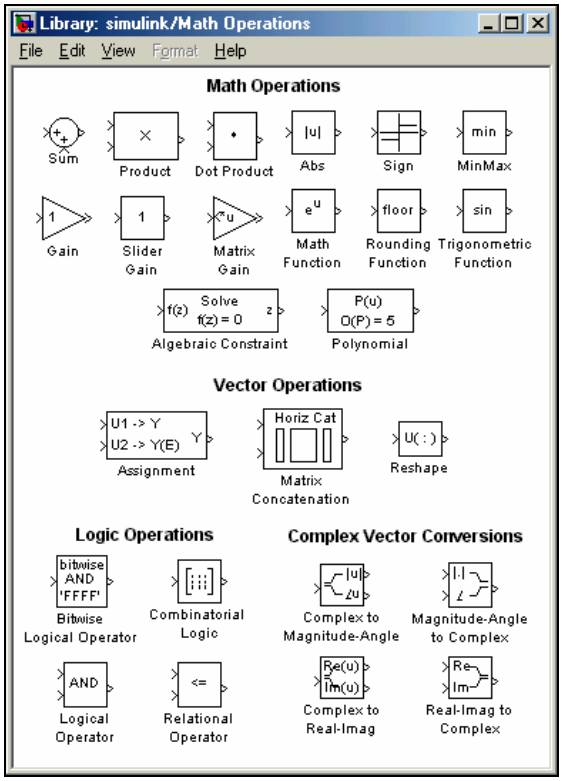


Рис. 7. 52. Содержимое раздела Math Operations

Они объединены в четыре группы:

Math Operations содержит блоки, осуществляющие математические преобразования входных величин;

Vector Operations содержит блоки, осуществляющие векторные операции;

Logic Operations содержит блоки, осуществляющие логические операции;

Complex Vector Conversions содержит блоки, осуществляющие преобразования комплексных векторных величин.

Рассмотрим вначале группу **Math Operations**. В нее включены такие блоки:

Sum блок,осуществляющийт суммирование сигналов, поступающих на него;

Product Блок, 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 изменения коэффициента усиления;

Matrix Gain матричное усилительное звено для многомерной системы;

MathFunction, TrigonometricFunction, MinMax, Abs, Sign и Rounding function шесть блоков математических стандартных операций;

Algebraic Constraint блок, осуществляющий нахождение решения алгебраического уравнения;

Polynomial блок, вычисляющий корни полинома.

Группа **Vector Operations** включает три блока:

Assignment (*Присваивание*) осуществляет включение сигнала в другой векторный сигнал;

Matrix Concatenation (*Конкатенация матриц*) осуществляет расширение матричного сигнала за счет его сцепления с другим матричным сигналом

Reshape (*Изменение формы*) преобразует входной векторный или матричный сигнал в одну из форм одномерного массива

Группа **Logic Operations** состоит из четырех блоков логических операций:

Logical Operator логический оператор;

Relation Operator оператор отношения;

Combinatorial Logic блок комбинаторной логики;

Bitwise Logical Operator поразрядный логический оператор;

Последняя четвертая группа состоит из четырех блоков преобразования комплексных сигналов в действительные и наоборот, - **Complex to Magnitude-Angle, Complex to Real-Imag, Magnitude-Angle to Complex и Real-Imag to Complex.**

Блок ***Sum*** может использоваться в двух режимах:
- суммирования входных сигналов (в том числе с разными знаками);
- суммирования элементов вектора, который поступает на вход блока.

Для управления режимами работы блока используется два параметра настраивания – *Icon Shape* (Форма изображения) и *List of signs* (Список знаков) (рис. 7.53).

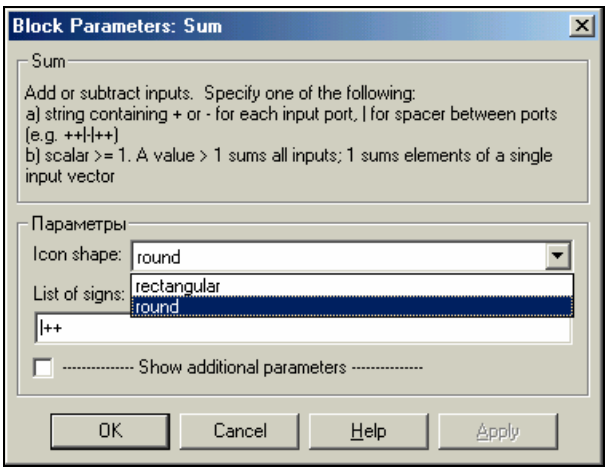


Рис. 7. 53. Окно настраивания блока Sum

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 во втором режиме

Блок **`Product`** выполняет умножение или деление нескольких входных сигналов. В параметры настраивания входят количество входов блока и вид выполняемой операции. Окно настраивания блока содержит два параметра (рис. 7. 54) – *Number of inputs* (количество входов) и *Multiplication*.

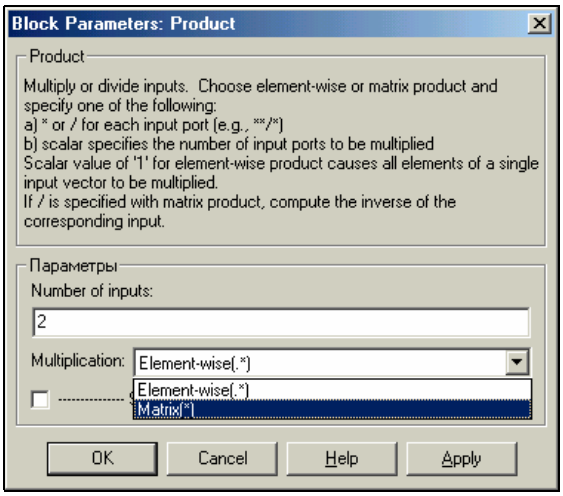


Рис. 7. 54. Окно настраивания блока Product

Входные величины могут быть векторными или матричными.

Параметр *Multiplication* устанавливает вид умножения входных величин:
Element wise поэлементное умножение входных векторов или матриц;
Matrix матричное умножение векторов или матриц.

Если параметр *Number of inputs* (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 единственного 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.`** Если при этом установлено

матричное умножение, то знаку « / » соответствует умножение на матрицу, обратную матрице соответствующей входной величины.

Блок **`DotProduct`** (*Внутреннее, скалярное произведение*) 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. Если векторы являются комплексными, то перед перемножением первый вектор (верхний входной порт) заменяется на комплексно сопряженный..

Блок **`Gain`** осуществляет умножение входного сигнала на постоянную величину (или вектор), значения которой (элементы которого) задаются в окне настраивания (рис. 7.55) в окошке параметра *Gain* (коэффициент усиления).
Входная величина блока (u) может быть скалярной, векторной или матричной. В случае, когда входной сигнал является вектором длиной *N* элементов, коэффициент усиления должен быть вектором той же длины. Параметр *Multiplication* устанавливает один из следующих видов умножения входной величины на вектор K коэффициентов усиления:

*Element wise (K.*u)* поэлементное умножение входного вектора на вектор коэффициентов усиления;
*Matrix (K*u)* матричное умножение вектора коэффициентов усиления на матрицу входной величины;
*Matrix (u*K)* матричное умножение матрицы входной величины на вектор коэффициентов усиления;
*Matrix (K*u) (u vector)* матричное умножение векторов K и u.

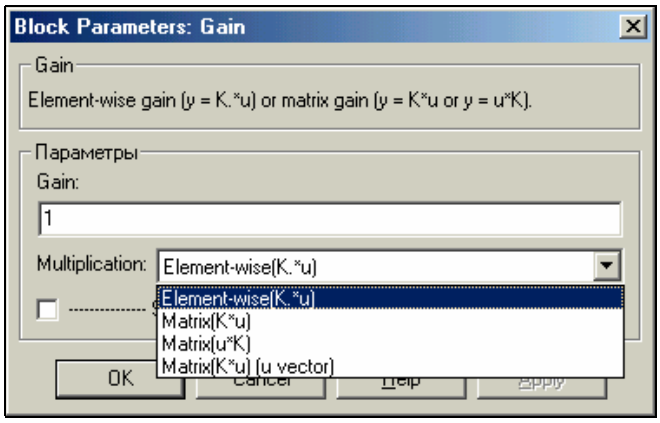


Рис. 7. 55. Окно настраивания блока Gain

Блок **`Matrix Gain`** (рис. 7.56) отличается от блока **`Gain`** только тем, что коэффициент передачи задается как матрица.

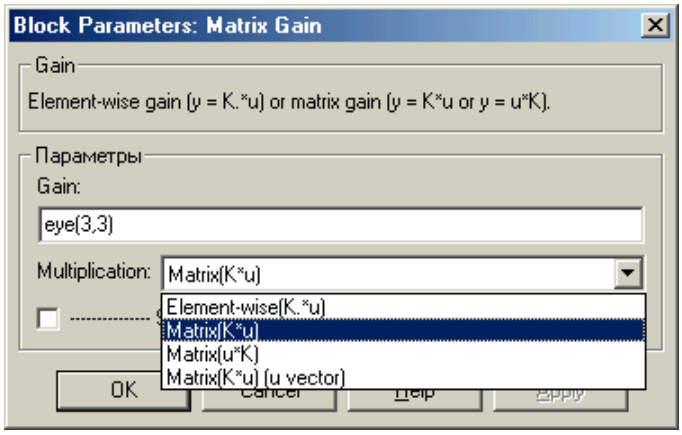


Рис. 7. 56. Окно настраивания блока Matrix Gain

Блок **`Slider Gain`** является разновидностью простейшего усилительного звена и 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 открыть окно с «ползунковым» регулятором (рис. 7.57), необходимо дважды щелкнуть мышью на изображении блока.

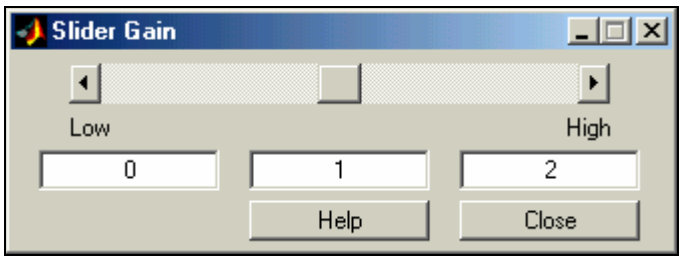


Рис. 7. 57. Окно настраивания блока Slider Gain

Окно Slider Gain имеет три поля ввода информации, два именованых
Low для указания нижней границы изменения параметра;
High для указания верхней границы изменения параметра;
и одно, среднее, - 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. После ввода значений этих трех числовых величин, можно, используя изображенный выше ползунковый регулятор, установить, передвигая ползунок мышью, любое другое значение внутри указанного диапазона. Установленное значение отобразится в числовом виде в среднем поле воода.

Далее приводятся особенности той части блоков, которая реализует математические функции.

Блок **Abs** формирует абсолютное значение вектора 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, asinh, acosh, atanh.** Выбор необходимой функции осуществляется в окне настраивания блока с помощью списка.

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, conj, reciprocal, hypot, rem, mod, transpose, hermitian**. Нужная функция выбирается с помощью списка 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 списка 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 (минимум или максимум) и количество входов блока.

Блок **Sign** реализует нелинейность типа сигнум-функции. В нем нет параметров настраивания. Блок формирует выходный 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.

Блоки группы **Logic Operations** имеют между собой то общее, 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 списка. Знак операции в дальнейшем отображается на изображении блока.

Блок **`Logical Operator`** содержит набор основных логических операций – AND, OR, NAND, NOR, XOR, NOT. Входные величины должны быть булевыми. Выбор необходимой логической операции осуществляется в окне настраивания блока с помощью списка. 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 Discontinuities

В разделе **Discontinuities** (*Разрывные элементы*) расположены (рис. 7.58) восемь блоков, которые реализуют некоторые типовые нелинейные (кусочно-линейные) зависимости выходной величины от входной

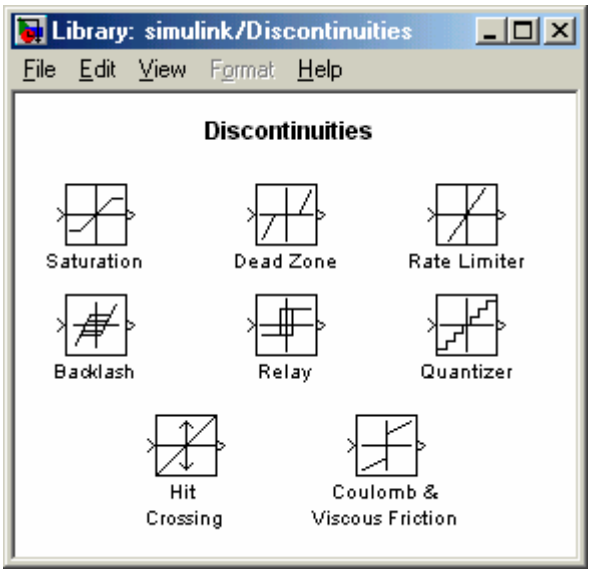


Рис. 7. 58. Содержимое раздела Discontinuities

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 **`Dead Zone`** *(Мертвая зона)* реализует нелинейность типа зоны нечувствительности. Параметров настраивания здесь два - начало и конец зоны нечувствительности. Выходная величина равна нулю, если входная величина принимает значения внутри зоны нечувствительности. Если входная величина больше верхней границы зоны, выходная равна входной минус эта верхняя граница. При входе, меньшем нижней границы зоны, выход равен входу плюс нижняя граница.

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*), выходная 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 со входной

$$y(i) = u(i).$$

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 **`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. 7.59):

`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.

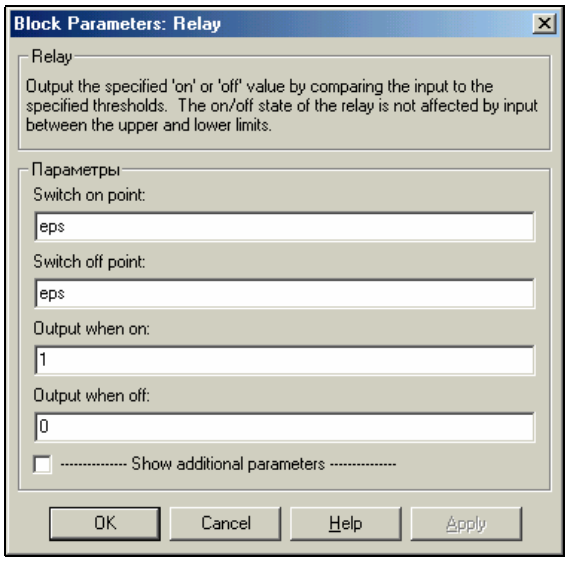


Рис. 7. 59. Окно настраивания блока Relay

Блок **`Quantizer`** (*Квантователь*) осуществляет дискретизацию (квантование) входного сигнала по его величине. Параметр настраивания блока один – *Quantization Interval* (*Интервал квантования*) – величина дискрета сигнала по его уровню.

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 три параметра настраивания (рис. 7. 60):

`Hit crossing offset` определяет значения, пересечение которого необходимо идентифицировать;

`Hit crossing direction` позволяет указать направление пересечения, при котором это пересечение должно выявляться; значения этого параметра выбирается с помощью списка, содержащего

Show output port (указать порт выхода)	три альтернативы: *rising (восхождение),* *falling (спадание),* *either (в любом направлении);* флажок, с помощью которого выбирается вид представления блока.

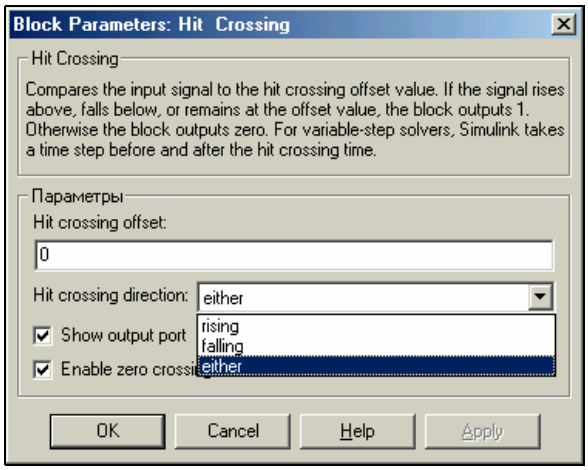


Рис. 7. 60. Окно настраивания блока Hit Crossing

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.

Блок **`Coulomb & Viscous Friction`** *(Кулоново и вязкое трение)* реализует нелинейную зависимость типа линейная с предварительным 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 равен нулю. В параметры настраивания блока входят величины кулонова трения (натяга) и коэффициента вязкого трения.

7.1.9. Раздел User Defined Functions

В разделе **User Defined Functions** (*Функции, определяемые пользователем*) сосредоточены блоки, позволяющие самому пользователю устанавливать их назначение.

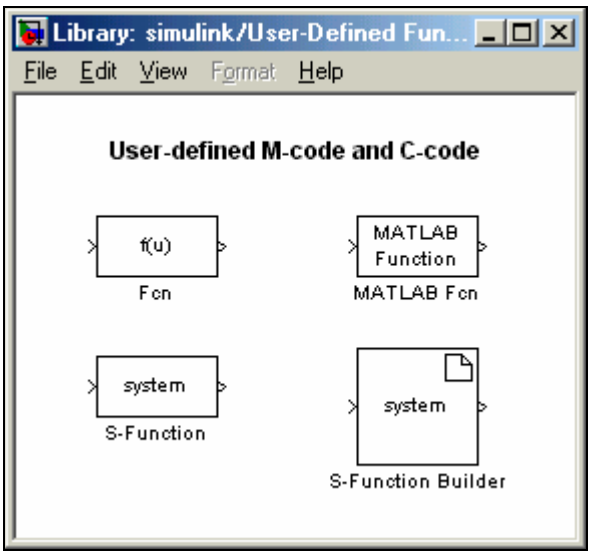


Рис. 7. 61. Содержимое раздела User-Defined Functions

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 формировать из входного сигнала не только скалярный, но и векторный выход. В отличие от предыдущего блока, здесь к числу параметров настраивания добавлен параметр *Output width (Ширина выходного сигнала)*, который определяет количество элементов выходного вектора. Значение (-1) этого параметра задает размер выходного вектора, совпадающую с размером входа. Отдельный *i*-ый элемент выходного вектора в окне параметра *MATLAB function* задается в виде функции, записанной на М-языке, предваряемой записью
`«u(i)=».`

С помощью блока **`S-function`** (*S-функция*) пользователь может реализовать в виде Simulink-блока любую программу обработки входного сигнала, включая создание сложных моделей систем, описываемых системой нелинейных дифференциальных или конечно-разностных уравнений, и обработку дискретных во времени сигналов. Более подробно работа с S-функциями изложена в уроке 8.

Блок **`S-function Builder`** (*Построитель S-функций*) позволяет пользователю создавать S-функцию в диалоговом режиме.

7.1.10. Раздел Signals Routing

Раздел **Signals Routing** (Маршрутизация сигналов) включает блоки, обеспечивающие различные необходимые пересылки сигналов. Состав этого раздела приведен на рис. 7. 62.

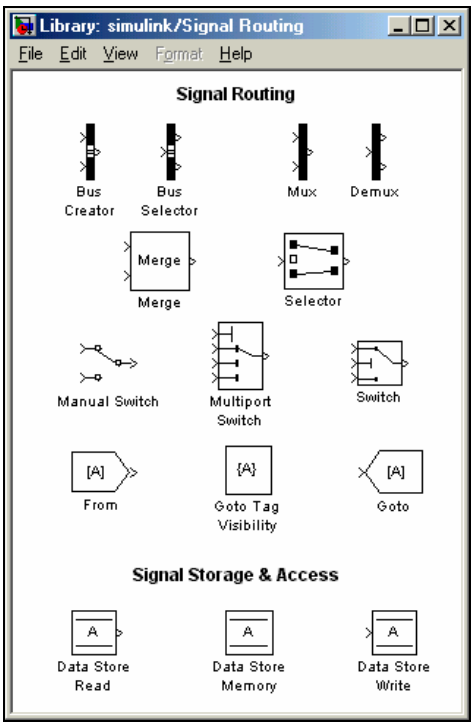


Рис. 7. 62. Содержимое раздела Signals Routing

Блок **`Mux`** *(Мультиплексор)* выполняет объединение входных величин в единый выходной вектор (шину). При этом входные величины могут быть как скалярными, так и векторными. Длина результирующего вектора равна сумме длин всех векторов. Порядок элементов в векторе выхода определяется порядком входов (сверху вниз) и порядком расположения элементов внутри каждого входа. Блок имеет два параметра настраивания - *Number of inputs (Количество входов)* и *Display option* (*Вид изображения*). Последний параметр определяет вид блока, в котором он будет изображен на блок-схеме

Блок **`Demux`** *(Разделитель, Демультиплексор)* выполняет обратную функцию - *разделяет входной вектор на заданное количество компонентов*. Он также имеет два параметра настраивания *Number of outputs (Количество выходов)* и *Display option* (*Вид изображения*). 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.

Два блока **`Bus Creator`** (*Построитель шины*) и **`Bus Selector`** (*Разделитель шины*), в общем, имеют те же функции, что, соответственно, и блоки **`Mux и Demux`**, но имеют большие возможности для перераспределения сигналов внутри шины.

Блок **`Merge`** (*Слияние*) производит объединение поступающих на его входы сигналов в один.

Блок **`Selector`** (*Селектор*) выбирает во входном векторе и передает на выход только те элементы, номера которых указаны в параметре *Elements* (рис. 7. 63).

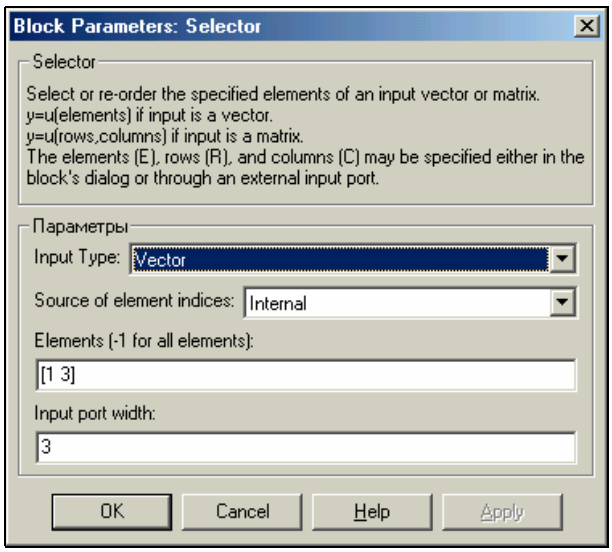


Рис. 7. 63. Окно настраивания блока Selector

Существенным достоинством блока является то, что значения параметров его настройки отображаются в графической форме на изображении блока.

Блок **`Manual Switch`** (*Ручной переключатель*) 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`** (*Многопортовый переключатель*) 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, равен 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).

Блок **`Switch`** имеет три входа: 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.

Блоки **`From`** *(Принять от)*, **`Goto Tag Visibility`** *(Признак видимости)* и **`Goto`** *(Передать в)* 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.

7.1.11. Раздел Signals Attributes

Раздел Signals Attributes (Атрибуты сигналов) содержит блоки, меняющие или определяющие значения некоторых атрибутов сигналов (рис. 7. 64).

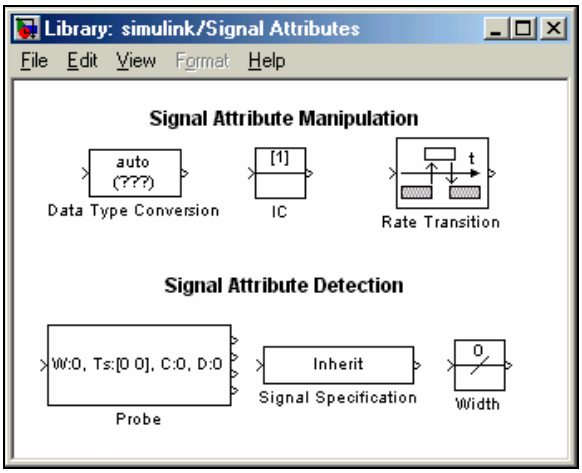


Рис. 7. 64. Содержимое раздела Signals Attributes

Блок **`Data Type Conversion`** (*Изменение типа данных*) позволяет изменить тип данных входного сигнала. Список единственного параметра настройки *Data Type* содержит такие альтернативы установки типа данных:

`auto`	автоматическое установление типа данных;
`double`	установление типа данных double;
`single`	установление типа данных single;
`int8`	установление типа данных целых 1-байтовых чисел;
`uint8`	установление типа данных целых 1-байтовых чисел только для хранения и считывания;
`int16`	установление типа данных целых 2-байтовых чисел;
`uint16`	установление типа данных целых 2-байтовых чисел только для хранения и считывания;
`int32`	установление типа данных целых 4-байтовых чисел;
`uint32`	установление типа данных целых 4-байтовых чисел только для хранения и считывания;
`boolean`	установление булевского типа данных.

Блок **`IC`** *(Initial Condition - начальное условие)* позволяет установить произвольное начальное значение входного сигнала. Применяется для внешней установки начального условия перед блоком **`Integrator.`**

Блок Probe (Зонд) служит для определения таких атрибутов сигнала

`width`	ширина сигнала;
`sample time`	величины дискрета времени;
`complex signal`	наличия комплексных сигналов
`signal dimensions`	размерности сигнала

Блок **`Width`** *(Размер)* определяет длину векторного сигнала, который поступает на его вход. Значения размерности выводится непосредственно на изображении блока. Параметров настраивания блок не имеет.

7.1.12. Раздел Ports & Subsystems

Раздел Ports & Subsystems (Порты и подсистемы) содержит блоки, позволяющие создавать подсистемы S-модели (рис. 7. 65).

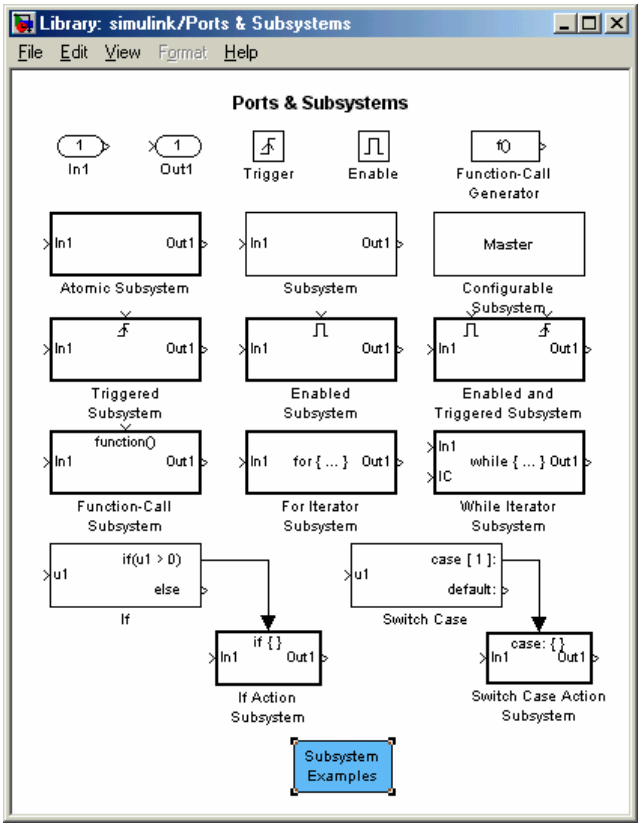


Рис. 7. 65. Содержимое раздела Ports & Subsystems

Подсистема - эта довольно самостоятельная S-модель более низкого уровня, которая, в свою очередь, может содержать подсистемы произвольного уровня вложенности.

Подсистемы не могут функционировать самостоятельно, а лишь в составе основной S-модели. Связь с основной S-моделью осуществляется через входные порты **`In`** и выходные порты **`Out`** подсистемы.

Подсистемы в S-модели играют ту же роль, что и функции (процедуры) в основной (вызывающей их) M-программе. При этом входные порты подсистемы определяют собой входные величины подсистемы, а выходные порты – выходные величины (результаты выполнения действий в подсистеме), которые в дальнейшем могут быть использованы в вызывающей S-модели (или подсистеме более высокого уровня иерархии).

Использование подсистем позволяет свести составление сложной S- модели к совокупности вложенных простых подсистем более низкого уровня, что делает моделирование более наглядным и упрощает отладку модели.

Опишем основные блоки раздела.

Блоки ***In*** *(Входной порт)* и ***Out*** *(Выходной порт)* обеспечивают информационную связь между подсистемой и вызывающей ее S-моделью.

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 ***Subsystem*** *(Подсистема)* является «заготовкой» для создания подсистемы.

Двойной щелчок на изображении этого блока в блок-схеме приводит к появлению на экране окна дополнительной блок-схемы (рис. 7. 66), в которой размещены только изображения одного входного и одного выходного порта, соединенные между собой. Это фактически напоминание пользователя, что создаваемая им подсистема должна обязательно содержать входные и выходные порты, которые должны быть соединены между собой (возможно, через посредство других блоков).

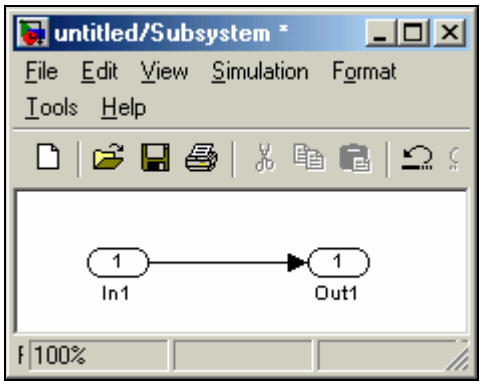


Рис. 7. 66. Окно заготовки блока Subsystem

Пользователь в этом окне создает блок-схему подсистемы по обычным правилам создания S-модели, а затем записывает ее на диск. При этом:

- дополнительные входные и выходные порты в подсистеме приведут к появлению на изображении блока Subsystem дополнительных изображений входов и выходов;
- надписи на входных и выходных портах подсистемы появятся рядом с соответствующими входами и выходами блока Subsystem на изображении блока.

7.1.13. Раздел Look Up Tables

В раздел **Look Up Tables** входят блоки (рис. 7. 67), позволяющие создавать непрерывные сигналы по заданным таблицам его значений путем интерполяции и экстраполяции заданных значений.

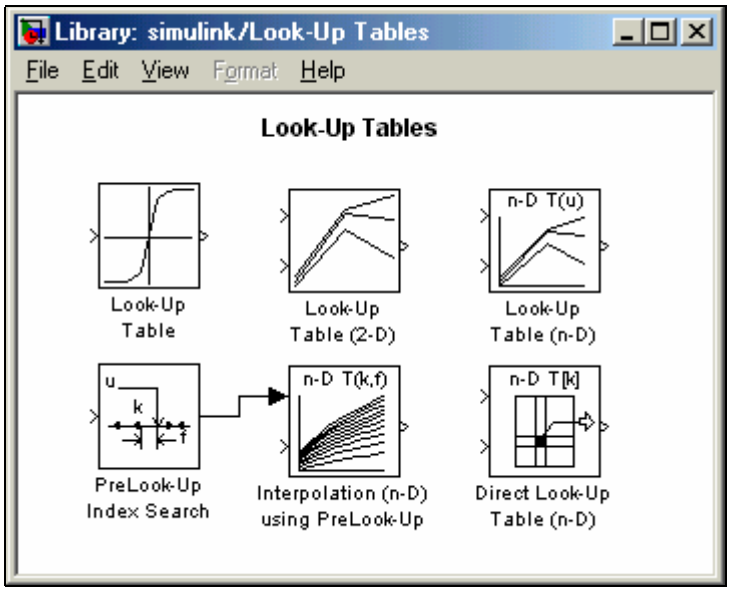


Рис. 7. 67. Содержимое раздела Look Up Tables

Блок **`Look-Up Table`** выполняет линейную интерполяцию входного сигнала в соответствии с табличной функцией, которая задается. Блок **`Look-Up Table(2D)`** осуществляет двумерную линейную интерполяцию двух входных сигналов.

7.1.14. Разделы Model Verification и ModelWide Utilities

Содержимое разделов Model Verification (Проверка модели) и ModelWide Utilities (Утилиты расширения модели) представлено на рис. 7. 68 и 7.69.

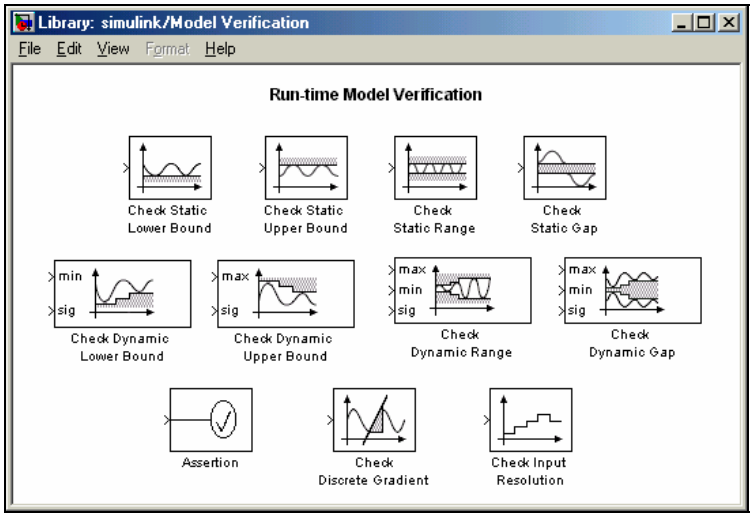


Рис. 7. 68. Содержимое раздела Model Verification

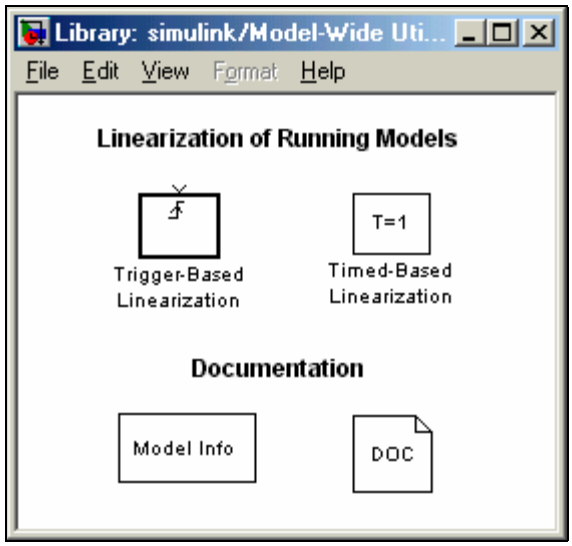


Рис. 7. 69. Содержимое раздела Model-Wide Utilities

В первом сосредоточены блоки, осуществляющие контроль некоторых статических и динамических характеристик модели. Во втором – блоки, позволяющие линеаризовать модель и оформить документацию на нее.

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 однин или несколько блоков и линий (объектов).

Чтобы ***выделить отдельный объект,*** нужно щелкнуть на нем один раз. При этом появляются маленькие затушеванные квадратики 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. 70 справа приведен результат выделения соединительной линии, а на рис. 7. 71 - выделения блока **`Clock`**

.

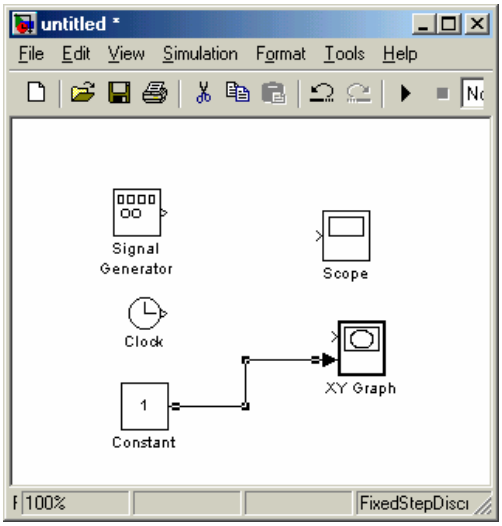


Рис. 7. 70. Результат выделения линии

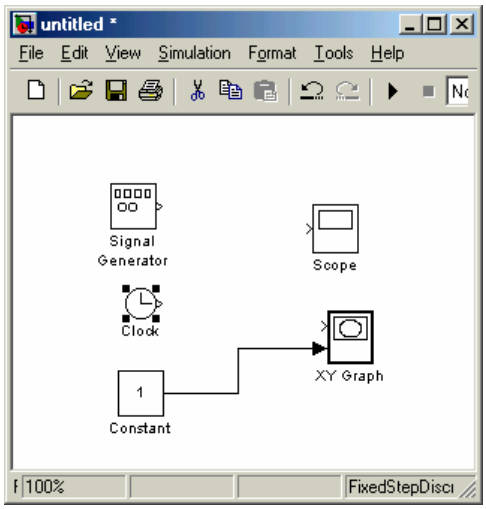


Рис. 7. 71. Результат выделения блока Clock

Для ***выделения нескольких объектов по одному*** нужно совершить такие действия:

1) нажать клавишу <Shift> и держать ее нажатой;
2) щелкнуть на каждом из объектов, которые выделяются, не отпуская клавишу <Shift>;
3) отпустить клавишу <Shift> .

Именно таким способом на рис. 7. 72 выделены блоки **`Signal Generator`**, **`Constant`** и **`XY Graph`**.

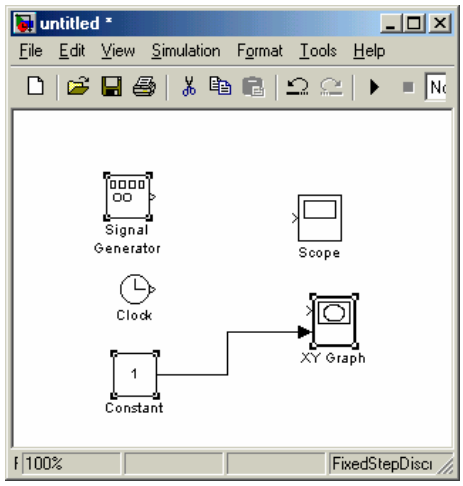


Рис. 7. 72. Результат выделения нескольких блоков

Выделить несколько объектов с помощью объединяющего бокса можно следующим образом:

1) определить стартовый угол прямоугольника-бокса, который будет содержать блоки, которые нужно выделить;
2) нажать клавишу мыши в этой точке;
3) не отпуская клавишу мыши, переставить ее курсор в противоположный угол бокса; при этом пунктирные линии должны окружить нужные блоки и линии;
4) отпустить клавишу мыши; блоки и линии внутри бокса будут выделены.

На рис. 7. 73 показан процесс выделения боксом блоков **`Signal Generator, Constant и Clock`**.

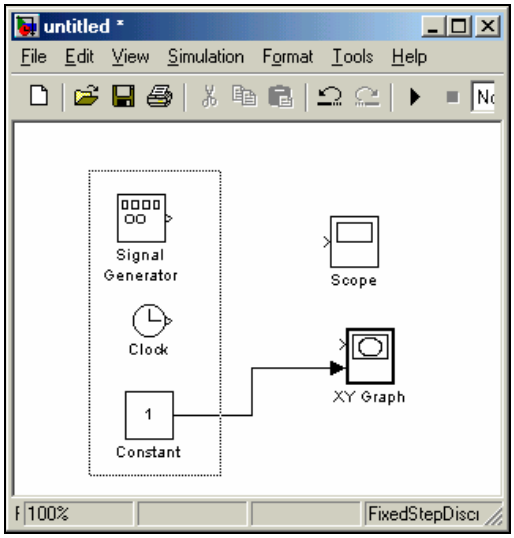


Рис. 7. 73. Выделение нескольких блоков боксом

Выделение всей модели, то есть всех объектов в активном окне блок-схемы, осуществляется одним из двух путей:

- выбором команды *Select All* в меню *Edit* окна блок-схемы;

- 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:

1) открыть нужный раздел библиотеки или окно модели-прототипа;
2) 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. *Первый скопированный блок будет иметь то же имя, что и блок в библиотеке*. *Каждый следующий блок того же типа будет иметь такое же имя с добавлением порядкового номера*. Пользователь может переименовать блок (см. далее).

При копировании блок получает те же значения настраиваемых параметров, что и блок-оригинал.

Перестановка блоков в модели

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:

1) выделить блоки и линии боксом (см. предыдущий раздел);
2) перетянуть мышью один из выделенных блоков на новое место; остальные блоки, сохраняя все относительные расстояния, займут новые места.

На рис. 7. 74 показан результат таких действий с блоками, выделенными на рис. 7. 73.

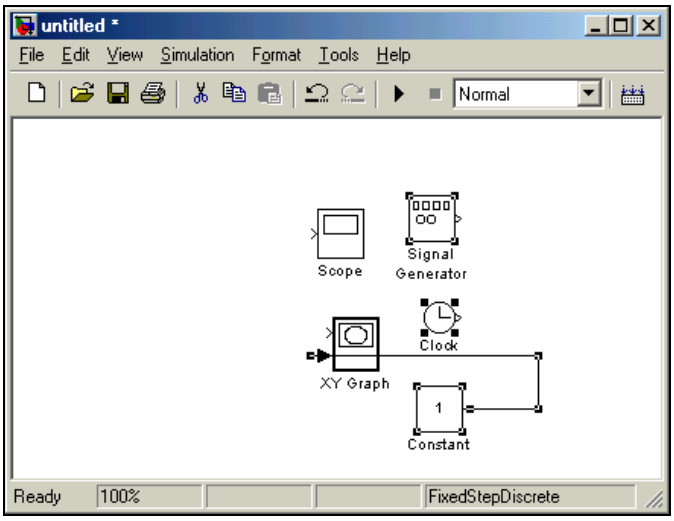


Рис. 7. 74. Результат перемещения блоков, выделенных боксом

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, который 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. 75 представлен результат копирования блоков **`Scope`** и **`XY Graph`**.

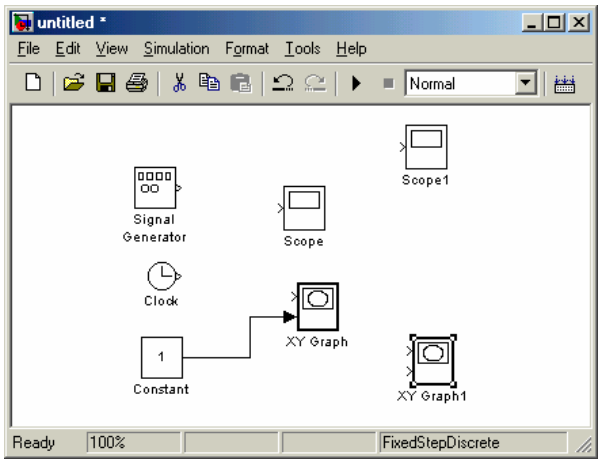


Рис. 7. 75. Результат копирования блоков

Задание параметров блока

Функции, которые выполняет блок, зависят от значений параметров блока. Установление 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.

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:

1) выделить блок, который нужно повернуть;
2) избрать меню *Format* в окне блок-схемы;
3) 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. 76 показан результат применения команды *Rotate Block* к блоку **`Constant`** и команд *Rotate Block* и *Flip Block* - к блоку **`SignalGenerator`**.

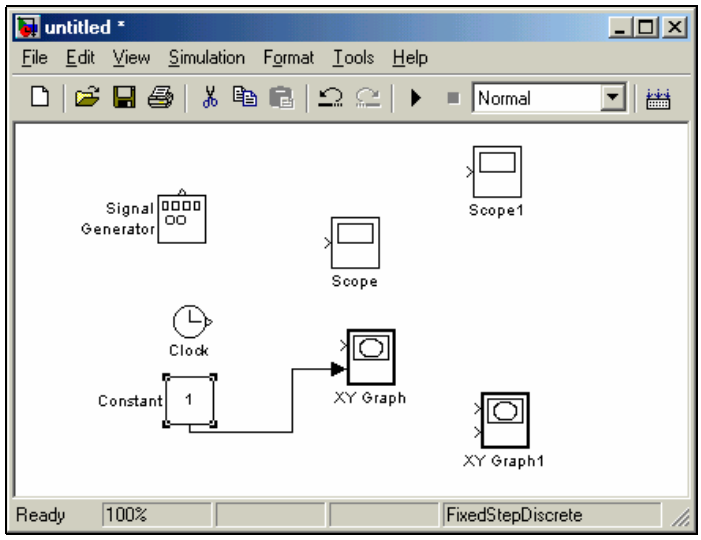


Рис. 7. 76. Результат изменения угловой ориентации блоков

Изменение размеров блока

Чтобы изменить размеры блока, нужно сделать следующее:

1) выделить блок, размеры которого надо изменить;
2) 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;
3) захватить эту метку мышью и перетянуть в новое положение; при этом противоположная метка этого блока останется неподвижной.

На рис. 7. 77 показан результат применения этих операций для растяжения блока **`XY Graph`**, а также середина процесса увеличения размеров блока **`Scope`**.

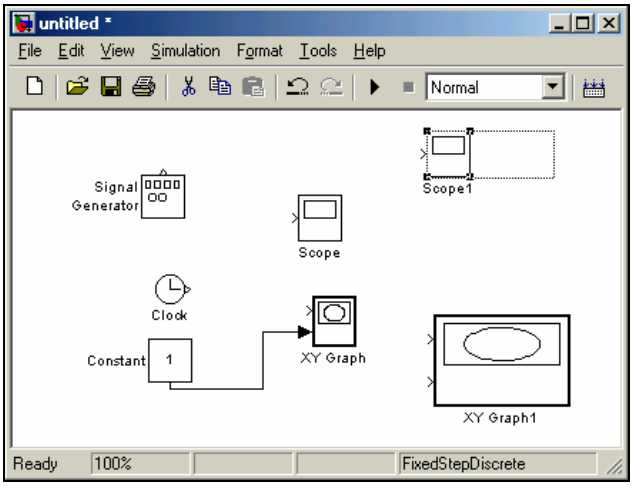


Рис. 7. 77. Результат растяжения изображения блоков

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. 77).

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:

1) перетянуть имяи на противоположную сторону мышью;

2) 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 со входным портом другого блока следует выполнить такую последовательность действий:

1) установить курсор внутрь выходного порта первого блока; при этом *курсор должен превратиться на перекрестие*;
2) нажав левую клавишу мыши и, удерживая ее в этом положении, передвинуть перекрестие ко входному порту второго блока;
3) отпустить ЛКМ; **Simulink** заменит символы портов соединительной линией с представлением направления передачи сигнала.

Именно таким образом соединен на рис. 7. 70 выход блока **`Clock`** с входом блока **`XY Graph1`**.

Линии можно рисовать как от выходного порта ко 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.*

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 один и тот же сигнал к нескольким блокам.

Чтобы ***образовать ответвление*** от существующей линии, нужно:

1) установить курсор на точку линии, от которой должна ответвляться другая линия;
2) нажав и удерживая клавишу <Ctrl>, нажать и удерживать ЛКМ;

3) провести линию к входному порту нужного блока; отпустить клавишу <Ctrl> и ЛКМ (см. рис. 7. 78).

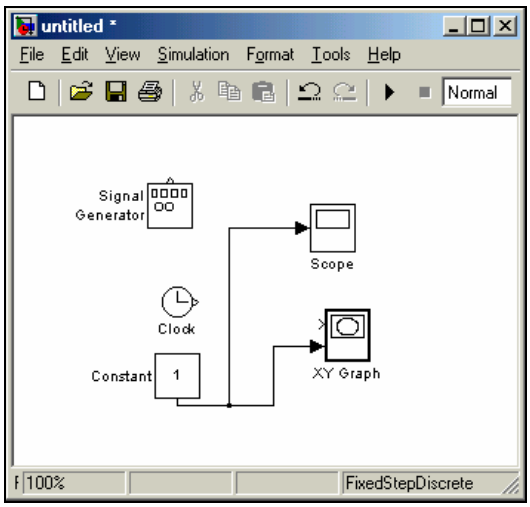


Рис. 7. 78. Создание разветвления линии

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. 79 блоки **`SignalGenerator`** с **`XY Graph`**.

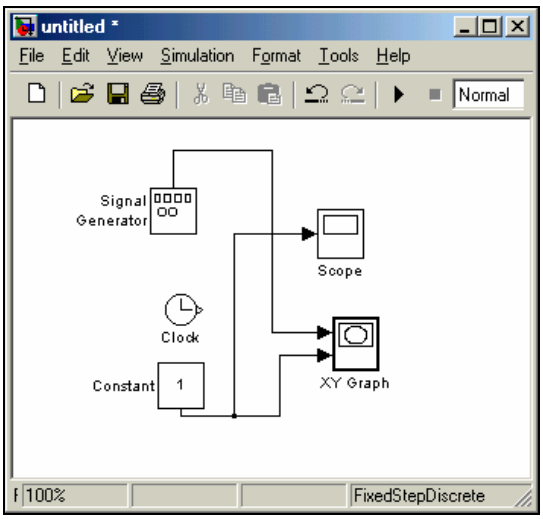


Рис. 7. 79. Проведение линии по сегментам

Передвижение сегмента линии

Чтобы ***передвинуть отдельный сегмент*** линии, необходимо выполнить следующее:

1) установить курсор на сегмент, который нужно передвинуть;
2) нажать и удерживать левую клавишу мыши (ЛКМ); при этом курсор должен превратиться в крест;
3) передвинуть крест к новому положению сегмента;
4) отпустить ЛКМ.

На рис. 7. 80 показан результат передвижения вертикального сегмента линии, которая соединяет блоки **`SignalGenerator`** с **`XY Graph`**.

Нельзя передвинуть сегмент, который непосредственно прилегает к порту блока.

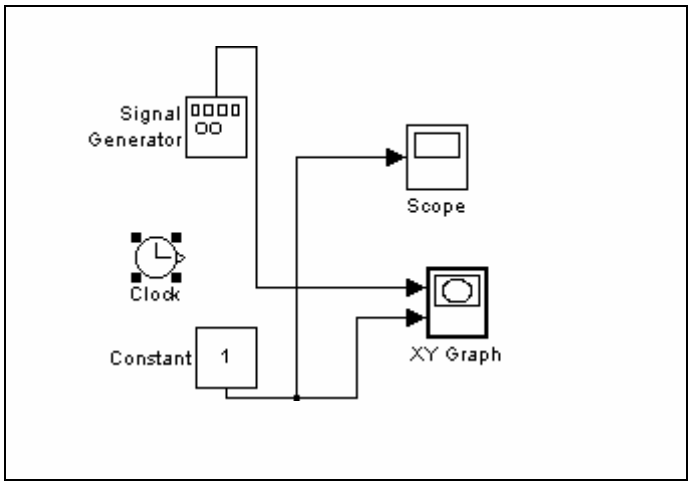


Рис. 7. 80. Передвижение сегмента линии

Разделение линии на сегменты

Чтобы ***разделить линию на два сегмента,*** нужно:

1) выделить линию;
2) установить курсор в ту точку выделенной линии, в которой линия должна быть разделена на два сегмента;
3) удерживая нажатой клавишу Shift, нажать и удерживать ЛКМ; курсор превратится на маленький круг; на линии образуется слом;
4) передвинуть курсор в новое положение слома;
5) отпустить ЛКМ и клавишу Shift.

Пример проведения этих действий представлен на рис. 7. 81, где линия, которая соединяет блоки ***Sine Wave*** и ***XY Graph1*** разделена на два сегмента.

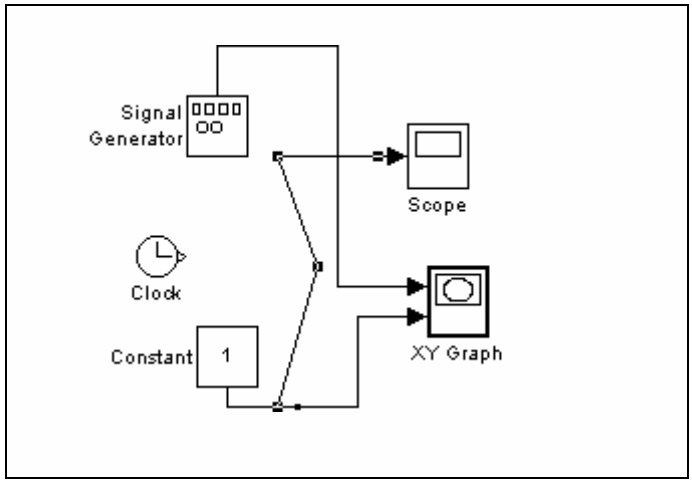


Рис. 7. 81. Разделение линии на два сегмента

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 пользования нею можно сопровождать линии метками сигналов, протекающих 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. 82). *При создании метки сигнала необходимо дважды щелкнуть именно точно на линии, так как иначе будет создан комментарий к модели.*

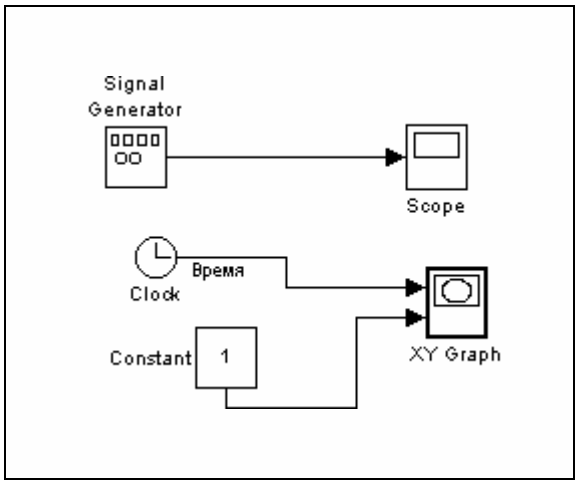


Рис. 7. 82. Создание метки «Время» на соединительной линии

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.

Распространение меток линии

Распространение меток линии - это процесс автоматического переноса имени метки к сегментам одной линии, которые разорваны блоками **`From/Goto, Mux`** (рис. 7. 83).

Чтобы ***распространить метки,*** создайте в втором и следующих сегментах линии метки с именем « < ». После выполнения команды *Update Diagram* из меню Edit окна модели, или одновременного нажатия клавиш Ctrl+D в этих сегментах автоматически будут проставлены метки (см. рис. 7. 84)

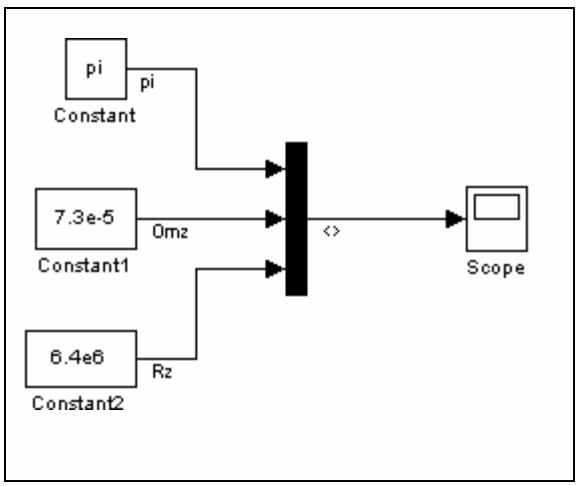


Рис. 7. 83. Распространение меток (процесс)

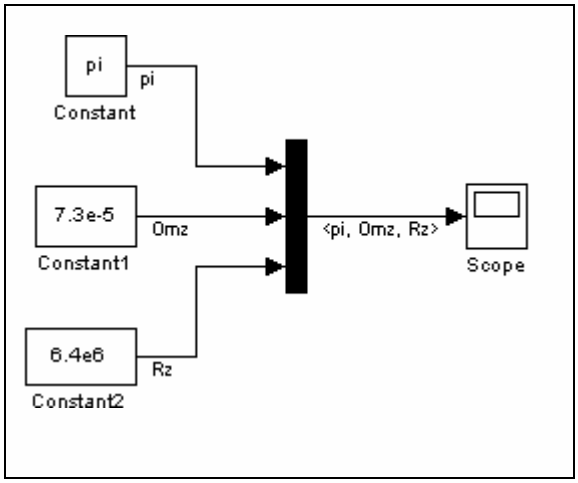


Рис. 7. 84. Распространение меток (результат)

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. 85).

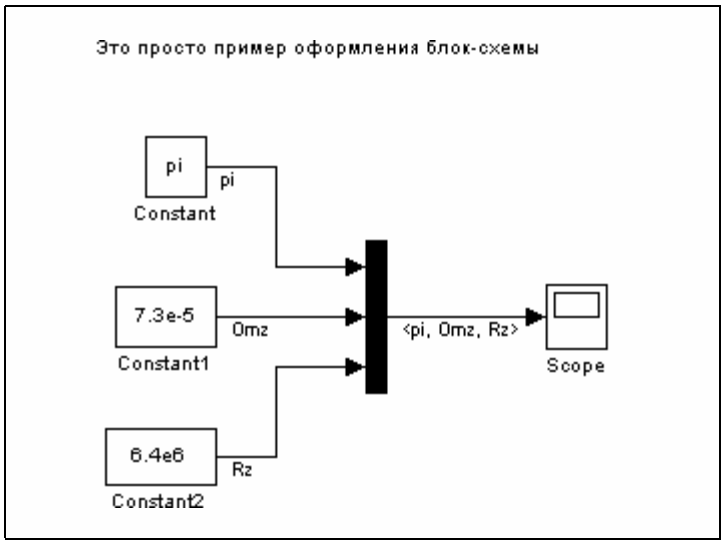


Рис. 7. 85. Пример проставления комментария

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.

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:

1) скопировать блок **`Subsystem`** в окно модели, перетянув его из библиотеки **`Ports&Systems`**;
2) раскрыть окно блок-схемы блока **`Subsystem`**, дважды щелкнув на изображении блока в блок-схеме S-модели;
3) в пустом окне блок-схемы подсистемы 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:

1) выделить объединяющим боксом блоки и соединительные блоки, которые нужно включить в состав подсистемы (рис. 7. 86);
2) избрать команду *Create Subsystem* из меню *Edit* окна модели; при этом **Simulink** заменит выделенные блоки одним блоком **`Subsystem`** (см. рис. 7. 87)

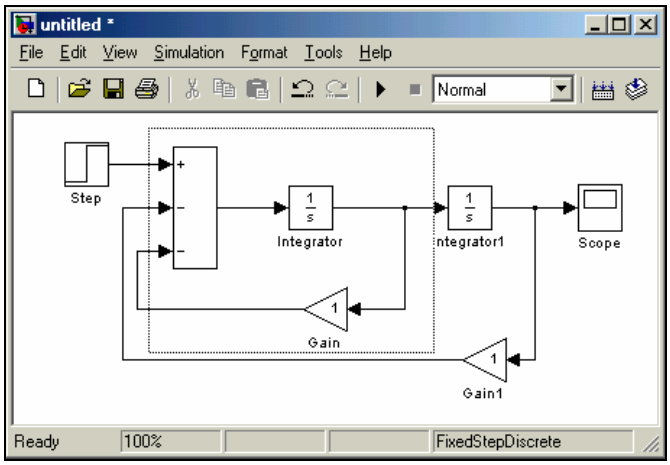


Рис. 7. 86. Создание подсистемы из части блок-схемы (процесс)

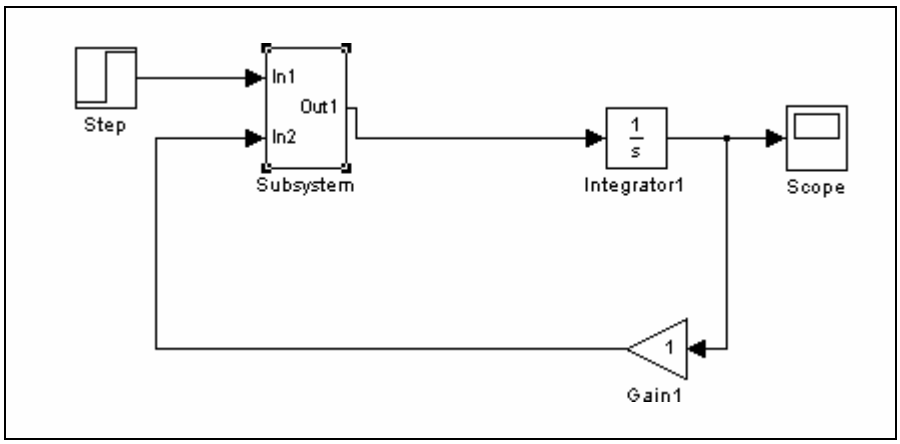


Рис. 7. 87. Создание подсистемы из части блок-схемы (результат)

Если раскрыть окно блока **`Subsystem`**, дважды щелкнув на нем, то **Simulink** отобразит блок-схему созданной подсистемы (рис. 7. 88).

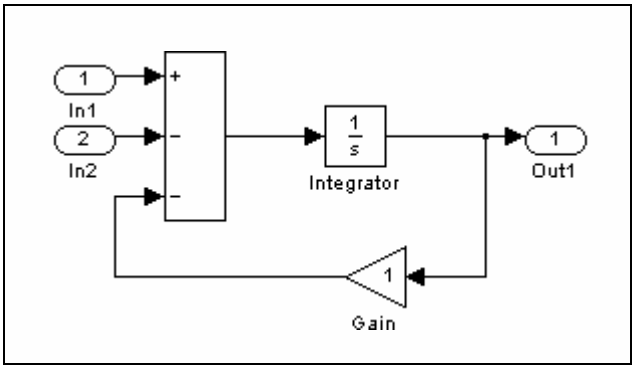


Рис. 7. 88. Созданная подсистема

Как видно, **Simulink** прибавил блоки **`In`** и **`Out`** для отображения входов и выходов в систему высшего уровня.

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 to clipboard* 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. 80...7. 88.

7.3. Примеры создания S-моделей

Рассмотрим несколько примеров составления S-моделей.

7.3.1. Моделирование поведения физического маятника

Рассмотрим процесс построения S-модели на примере задачи моделирования поведения физического маятника при гармонической вибрации точки его опоры, решенной ранее (урок 2).

Пользуясь результатами ранее проведенных (п. 2.6.2) преобразований, исходное уравнение движения маятника примем в такой безразмерной форме

$$\varphi'' + \sin\varphi = S(\tau, \varphi, \varphi'), \tag{7.1}$$

где обозначено

$$\begin{aligned} S(\tau,\varphi,\varphi') &= -2\cdot\zeta\cdot\varphi' - \\ &-[n_{mx}\cdot\sin(\nu\cdot\tau+\varepsilon_x)\cdot\cos\varphi + n_{my}\cdot\sin(\nu\cdot\tau+\varepsilon_y)\cdot\sin\varphi], \end{aligned} \tag{7.2}$$

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 воздействие; сюда входят:

- амплитуды виброперегрузок в вертикальном n_{my} и

горизонтальном n_{mx} направлениях;

- относительная (по отношению к частоте собственных малых 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:

$$\varphi'' = S(\tau, \varphi, \varphi') - \sin\varphi. \tag{7.3}$$

Прежде, чем приступить к составлению S- модели, отметим одну очень важную особенность взаимодействия рабочего пространства **MatLab** и среды **Simulink**.

ВНИМАНИЕ. Все рабочее пространство (WorkSpace) системы **MatLab** доступно для исполняемых S-моделей среды **Simulink**. Поэтому при задании значений настраиваемым параметрам S-блоков можно вместо конкретных числовых значений вводить имена переменных (идентификаторы), которые существуют в данный момент времени в рабочем пространстве **MatLab** и даже выражения из них, записанные на М-языке.

Это свойство значительно облегчает составление блок-схем, позволяя вначале формировать массив переменных в рабочем пространстве, а при составлении блок-схем значения настраиваемых параметров блоков выражать через эти переменные.

В дальнейшем будем использовать следующие обозначения исходных и искомых величин в рабочем пространстве **MatLab**.

В формулах	В рабочем пространстве	Примечание
$\varphi(0)$	`fi0`	начальное значение угла отклонения маятника от вертикали
$\varphi'(0)$	`fit0`	начальная безразмерная угловая скорость маятника
ς	`dz`	относительный коэффициент затухания
n_{mx}	`nmx`	амплитуда виброперегрузки вдоль горизонтальной оси
n_{my}	`nmy`	амплитуда виброперегрузки вдоль вертикальной оси

ν	`nu`	относительная частота вибрации тоски подвеса
ε_x	`ex`	начальная фаза виброперегрузки в горизонтальном направлении
ε_y	`ey`	начальная фаза виброперегрузки в вертикальном направлении

Будем предполагать, что тем или иным способам этим переменным присвоены значения и они находятся в рабочем пространстве.

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 теперь 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`**); 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 $\varphi(\tau)$.

Оформим эту часть блок-схемы в виде подсистемы. Для этого в пустое окно будущей блок-схемы перетянем из **Ports&Subsystem** блок **`Subsystem`** и дважды щелкнем на нем. В появившемся окне соберем блок-схему подсистемы, показанную на рис. 7.89, и назовем ее «Маятник».

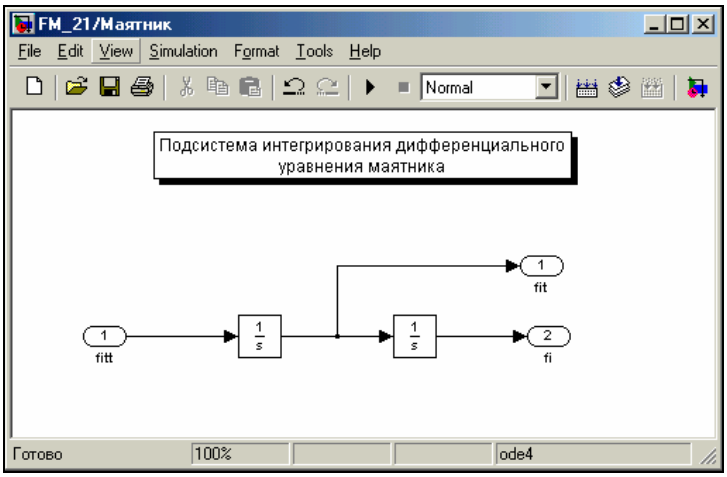


Рис. 7. 89. Подсистема «Маятник»

При настраивании первого блока **`Integrator`** запишем `fit0` в качестве его параметра *Initial condition*. Во втором интеграторе этому параметру присвоим значение `fi0`.

Выходными величинами созданной подсистемы являются текущий угол φ отклонения маятника от вертикали и его текущая безразмерная угловая скорость φ'.

Используя их, можно теперь аналогично сформировать подсистему, вычисляющую текущее значение безразмерного «момента» внешних сил, действующих на маятник при вибрации точки его опоры. Назовем новую подсистему «Внешние моменты сил» (рис. 7. 90).

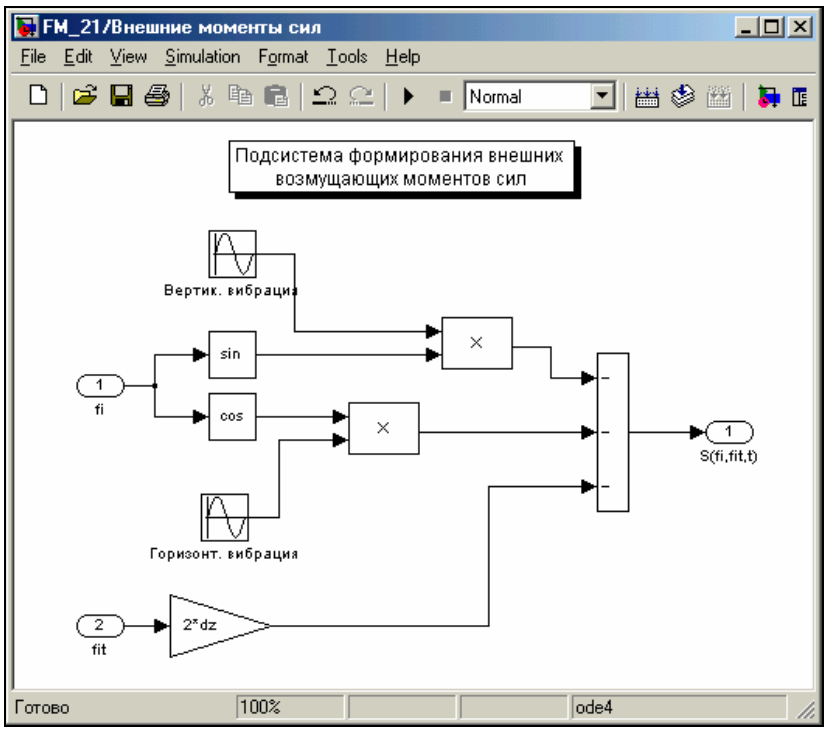


Рис. 7. 90. Подсистема «Внешние моменты сил»

Наглядно видно, что общая структура подсистемы осуществляет действия по формированию выражения (7.2). Выходной порт ее выдает величину $S(\varphi, \varphi', \tau)$.

Можно заметить, что величина `dz` (относительного коэффициента затухания) использована в блоке **`Gain внизу блок-схемы`**. Кроме того, при формировании перегрузок по горизонтальной и вертикальной осям (блоки «Вертик. вибрация» и «Горизонт. вибрация») в параметрах настраивания использованы переменные `nmx, nmy, nu, ex` и `ey` (рис. 7. 91).

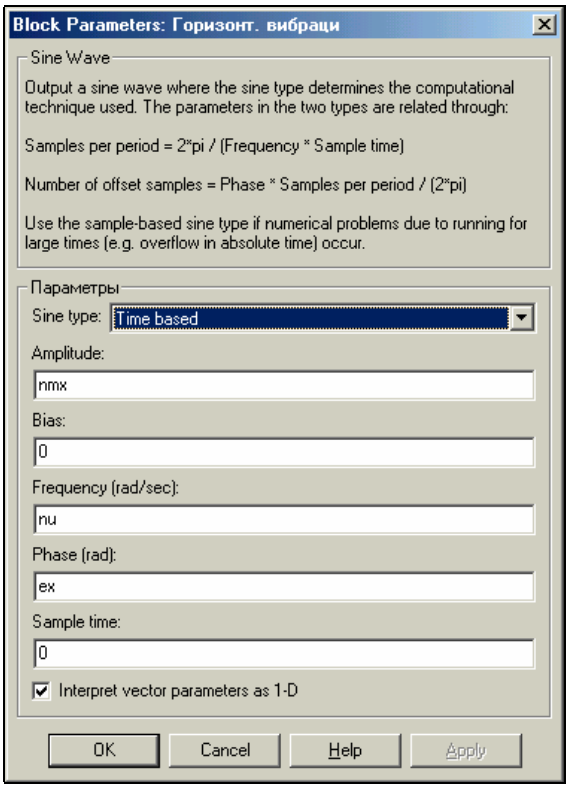


Рис. 7. 91. Установки в блоке «Горизонт. вибрация»

Теперь, используя эти две созданные подсистемы, можно и собрать основную блок-схему, реализующую уравнение (7.3). Она изображена на рис. 7. 92. Назовем ее «FM_21»

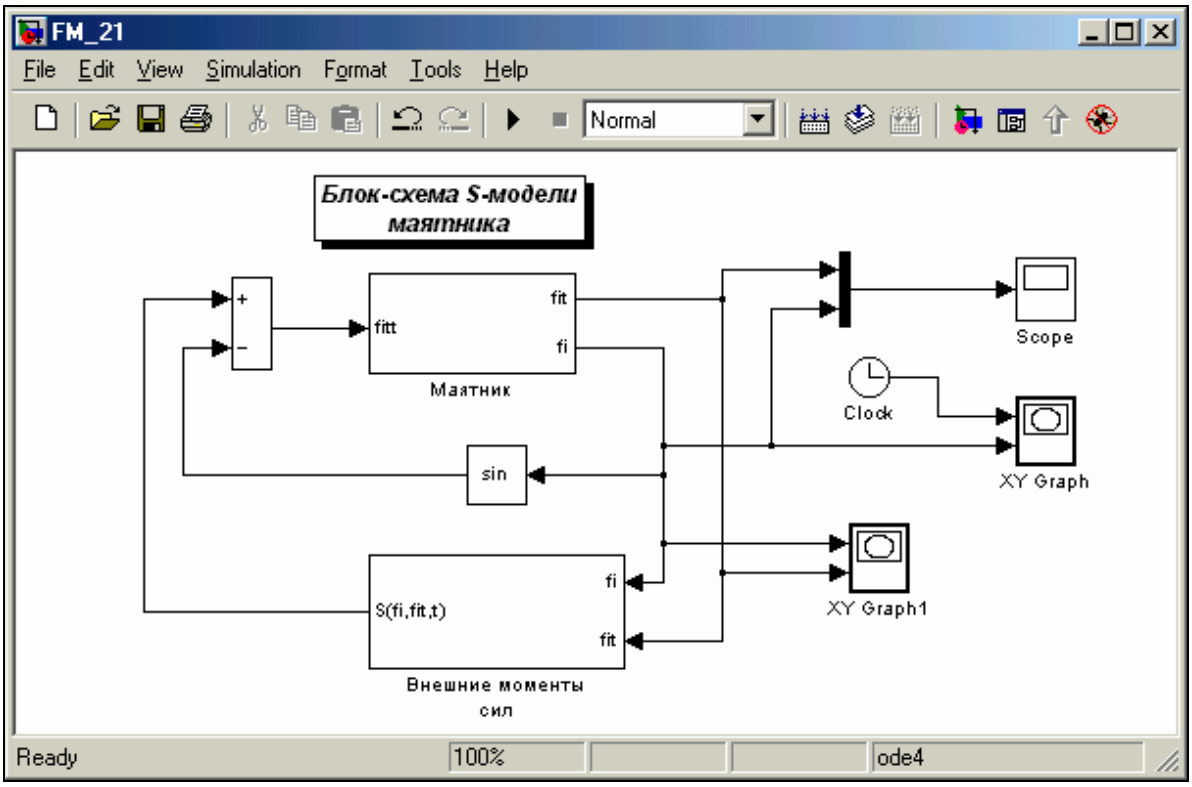


Рис. 7. 92. Основная блок-схема «FM_21»

Перед моделированием следует вначале задать параметры численного интегрирования дифференциального уравнения маятника на вкладке Solver (рис. 7. 93).

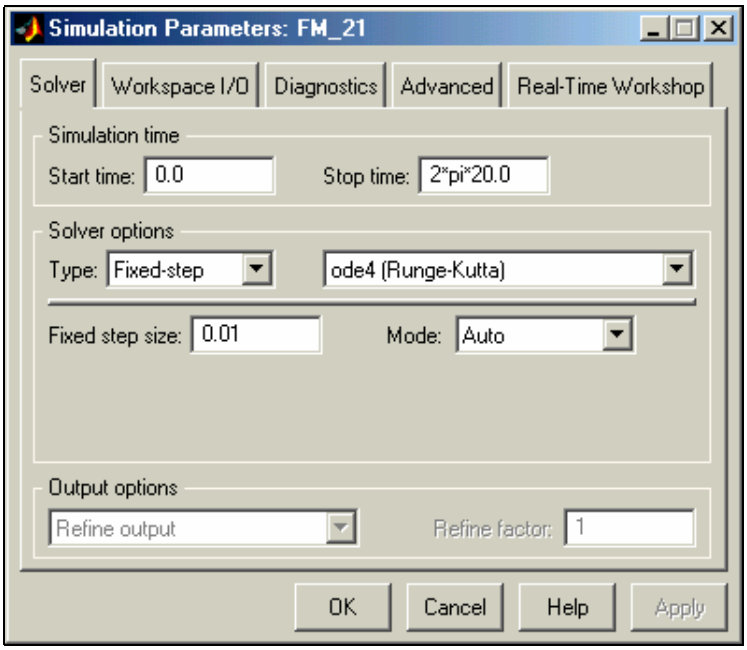


Рис. 7. 93. Установка параметров численного интегрирования

Установим интервал моделирования (по безразмерному времени) от *Start time*=0 до *Stop time*=2*pi*20, что соответствует двадцати периодам собственных малых колебаний маятника. Тип решателя (метода интегрирования) установим с фиксированным шагом (*Type= Fixed-step*), метод интегрирования – Рунге-Кутта 4-го порядка (*ode4*) а значение шага интегрирования – 0,01 с (Step size=0.01).

Модель практически готова к моделированию.

Осталось лишь обеспечить присвоение значений всем переменным, которые были использованы в параметрах настраивания блоков. Для этого запишем небольшую программу ***FM_21_dat.m*** (см. ниже):

```
% FM_21_dat
% Программа ввода данных для S-модели FM_21

% Лазарев Ю. Ф. 23-01-2004

nmy=1;
nmx=0;
ey=0;
ex=0;
nu=2.3;
dz=0.1;
fi0=160*pi/180;
fit0=0;
```

Данные, введенные в файле FM_21_dat.m соотвествуют чисто вертикальной вибрации основания с частотой в 2,3 большей частоты собственных незатухающих колебаний маятника с амплитудой по ускорению в 1g и начальному отклонению маятника от вертикаль в 160 градусов.

Запуская вначале программу FM_21_dat.m из командного окна **MatLab**, а затем S-модель `FM_21` из окна ее блок-схемы, получим в обзорных окнах модели результаты, представленные на рис. 7. 94…7.96.

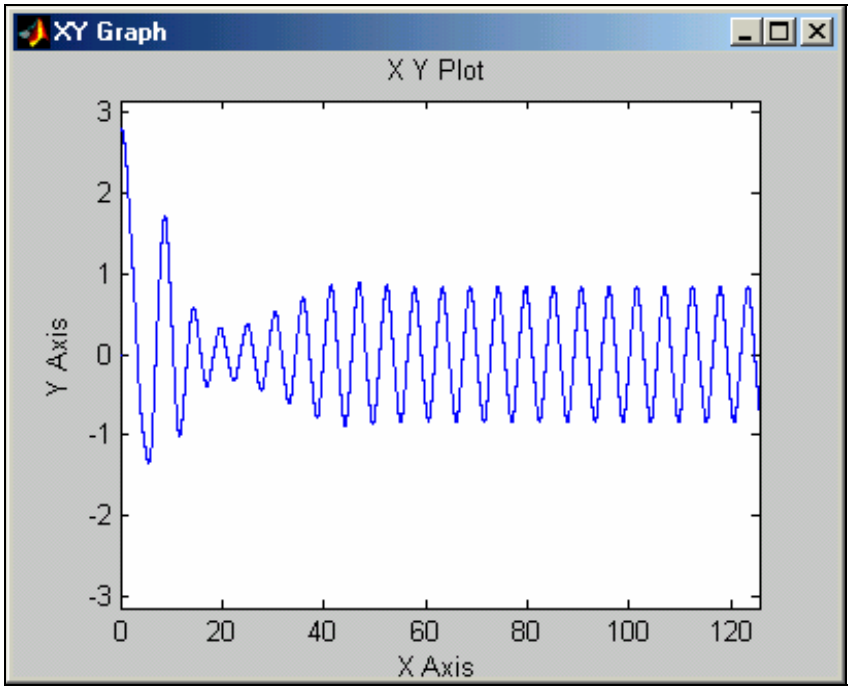


Рис. 7. 94. Результат моделирования, представленный в окне XY Graph

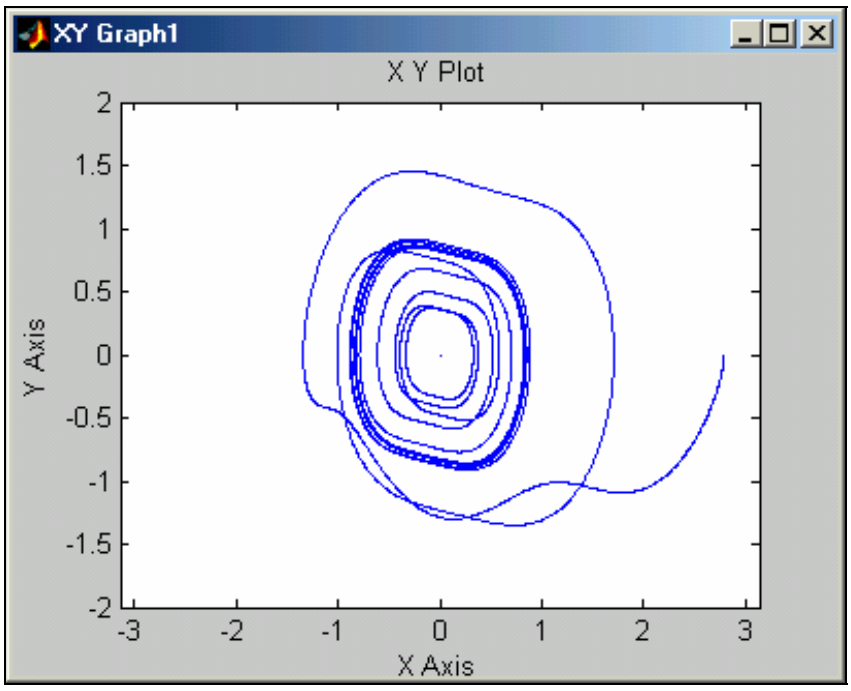


Рис. 7. 95. Результат моделирования, представленный в окне XY Graph1

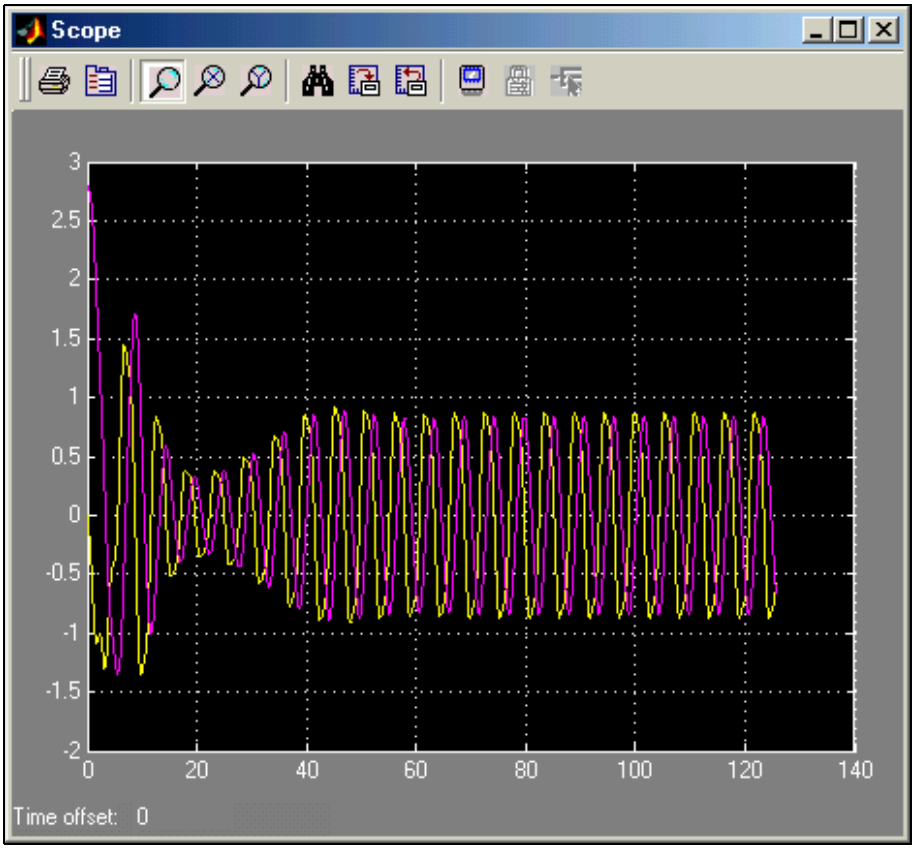


Рис. 7. 96. Результат моделирования, представленный в окне Scope

В графическом окне блока **`XY Graph1`** (см. рис. 7. 95) изображен фазовый портрет маятника при выбранных параметрах маятника и возмущений. В графическом окне блока **`Scope`** представлены (рис. 7. 96) графики зависимости угла и угловой скорости от времени.

Результаты полностью соответствуют полученным ранее (урок 2) и иллюстрируют возникновение параметрических колебаний маятника при вертикальной вибрации точки его подвеса.

Изменяя данные настройки в файле FM_21_dat.m, 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 **MatLab**:

- 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 графические окна блоков XY Graph; 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 движения трех гравитирующих тел

Здесь рассматривается классическая задача небесной механики – определение движения трех гравитирующих материальных точек – с точки зрения численного моделирования его в среде **Simulink**.

Пусть существуют три изолированные материальные точки с массами соответственно m_1, m_2 и m_3.

Обозначим радиус-векторы этих точек относительно некоторой неподвижной в инерциальном пространстве точки О через $\mathbf{R_1}$, $\mathbf{R_2}$ и $\mathbf{R_3}$.

Тогда дифференциальные уравнения движения этих трех точек могут быть записаны в виде:

$$\begin{cases} m_1 \dfrac{d^2\mathbf{R_1}}{dt^2} = G \cdot \dfrac{m_1 \cdot m_2}{R_{21}^3} \cdot \mathbf{R_{21}} - G \cdot \dfrac{m_1 \cdot m_3}{R_{31}^3} \cdot \mathbf{R}_{13}; \\ m_2 \dfrac{d^2\mathbf{R}_2}{dt^2} = -G \cdot \dfrac{m_1 \cdot m_2}{R_{21}^3} \cdot \mathbf{R_{21}} + G \cdot \dfrac{m_2 \cdot m_3}{R_{32}^3} \cdot \mathbf{R}_{32}; \\ m_3 \dfrac{d^2\mathbf{R}_3}{dt^2} = -G \cdot \dfrac{m_3 \cdot m_2}{R_{32}^3} \cdot \mathbf{R_{32}} + G \cdot \dfrac{m_3 \cdot m_1}{R_{31}^3} \cdot \mathbf{R}_{13}; \end{cases} \qquad (7.4)$$

где обозначено $\mathbf{R_{21}} = \mathbf{R}_2 - \mathbf{R_1}$; $\mathbf{R_{32}} = \mathbf{R}_3 - \mathbf{R}_2$; $\mathbf{R}_{13} = \mathbf{R}_1 - \mathbf{R}_3$.

Исследования уравнений движения (и, особенно, численное) всегда удобнее производить по уравнениям в безразмерном виде, - в этом случае сокращается число параметров, от которых зависит решение.

Приведем уравнения (4) к безразмерной форме. Для этого в качестве базовых используем такие физические величины:

G - гравитационная постоянная размерности $L^3M^{-1}T^{-2}$ (L – единица длины, M – единица массы, T – единица времени);

m_1 - масса первого тела, которое будем считать основным; им обычно является наиболее массивное тело (например, Солнце, которое находится под действием гравитационного притяжения Земли (второе, среднее по массе тело) и Луны (третье, наименее массивное тело)); размерность M;

R_{210} - начальное значение расстояния второго тела от первого, размерностью L.

Теперь введем безразмерные величины:

1) безразмерная масса первого тела, очевидно, будет равна единице;

2) безразмерная масса второго тела равна

$$\mu_2 = \frac{m_2}{m_1}; \tag{7.5}$$

3) безразмерная масса третьего тела равна

$$\mu_3 = \frac{m_3}{m_1} \quad ; \tag{7.6}$$

4) безразмерные длины радиус-векторов

$$\rho_1 = \frac{R_1}{R_{210}}; \quad \rho_2 = \frac{R_2}{R_{210}}; \quad \rho_3 = \frac{R_3}{R_{210}}: \tag{7.7}$$

5) безразмерные радиус-векторы

$$\mathbf{r}_i = \frac{\mathbf{R}_i}{R_{210}}; \qquad \mathbf{r}_{ij} = \frac{\mathbf{R}_{ij}}{R_{210}}; \tag{7.8}$$

5) безразмерное время определим формулой

$$\tau = \sqrt{\frac{G \cdot m_1}{{R_{210}}^3}} \cdot t; \tag{7.9}$$

Последняя формула означает, что в качестве единицы измерения времени используется величина

$$T_o = 2\pi \sqrt{\frac{{R_{20}}^3}{G \cdot m_1}}. \tag{7.10}$$

В случае, например, Солнца, Земли и Луны эта величина равна году.

Принятое безразмерное время таково, что *безразмерный период кругового обращения второго тела вокруг первого равен* 2π.

С учетом этого уравнения (4) в безразмерной форме приобретут вид

$$\begin{cases} \dfrac{d^2\mathbf{r_1}}{d\tau^2} = \dfrac{\mu_2}{\rho_{21}^3} \cdot \mathbf{r}_{21} - \dfrac{\mu_3}{\rho_{31}^3} \cdot \mathbf{r}_{13}; \\ \dfrac{d^2\mathbf{r}_2}{d\tau^2} = -\dfrac{1}{\rho_{21}^3} \cdot \mathbf{r}_{21} + \dfrac{\mu_3}{\rho_{32}^3} \cdot \mathbf{r}_{32}; \\ \dfrac{d^2\mathbf{r}_3}{d\tau^2} = -\dfrac{\mu_2}{\rho_{32}^3} \cdot \mathbf{r_{32}} + \dfrac{1}{\rho_{31}^3} \cdot \mathbf{r}_{13} \end{cases} \tag{7.11}$$

где обозначено

$$\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j; \qquad \rho_{ij} = \left|\mathbf{r}_{ij}\right|. \tag{7.12}$$

Три вектора (12) связаны между собой соотношением

$$\mathbf{r}_{13} + \mathbf{r}_{32} + \mathbf{r}_{21} = 0, \tag{7.13}$$

Уравнения движения удобнее полностью выразить в векторах (12), характеризующих положение тел относительно друг друга. Для этого вычтем по очереди уравнения (11) одно из другого. Из полученных трех уравнений исключим вектор $\mathbf{r}_{13}$. Останутся два уравнения

$$\begin{cases} \dfrac{d^2\mathbf{r}_{21}}{d\tau^2} = -\left(\dfrac{1+\mu_2}{\rho_{21}^3} + \dfrac{\mu_3}{\rho_{31}^3}\right)\cdot\mathbf{r}_{21} + \mu_3\left(\dfrac{1}{\rho_{32}^3} - \dfrac{1}{\rho_{31}^3}\right)\cdot\mathbf{r}_{32}; \\ \dfrac{d^2\mathbf{r}_{32}}{d\tau^2} = -\left(\dfrac{\mu_2+\mu_3}{\rho_{32}^3} + \dfrac{1}{\rho_{31}^3}\right)\cdot\mathbf{r}_{32} + \left(\dfrac{1}{\rho_{21}^3} - \dfrac{1}{\rho_{31}^3}\right)\cdot\mathbf{r}_{21} \end{cases} \tag{7.14}$$

Эти уравнения и положим в основу моделирования. При этом надо принять во внимание, что $\rho_{21}(0) = 1$.

При составлении S-модели будем использовать следующие переменные рабочего пространства:

В формуле	В рабочем пространстве	Примечание
$\mu_2 = \dfrac{m_2}{m_1}$	mu2	отношение масс второго и первого тела
$\mu_3 = \dfrac{m_3}{m_1}$	mu3	отношение масс третьего и первого тела
$x_{21}(0), y_{21}(0), z_{21}(0)$	X210, Y210, Z210	Начальные безразмерные координаты второго тела относительно первого
$x_{32}(0), y_{32}(0), z_{32}(0)$	X320, Y320, Z320	Начальные безразмерные координаты третьего тела относительно второго
$V_{21x}(0), V_{21y}(0), V_{21z}(0)$	V21x, V21y, V21z	Начальные безразмерные проекции вектора скорости второго тела относительно первого
$V_{32x}(0), V_{32y}(0), V_{32z}(0)$	V32x, V32y, V32z	Начальные безразмерные проекции вектора скорости третьего тела относительно второго

Вариант S-модели под названием GR_3_TEL. mdl приведен на рис. 7. 97. Она включает 6 подсистем – **`Uravnenye 21`**, **`Uravnenye 32`**, **`Prav 21`**, **`Prav 32`**, **`RV13`** и **`Absolute Coordinates`** и блок **`To File`**. Последний обеспечивает запись результатов моделирования в MAT-файл под названием GR_3_TEL.mat, в котором они сохраняются в матрице под именем RV.

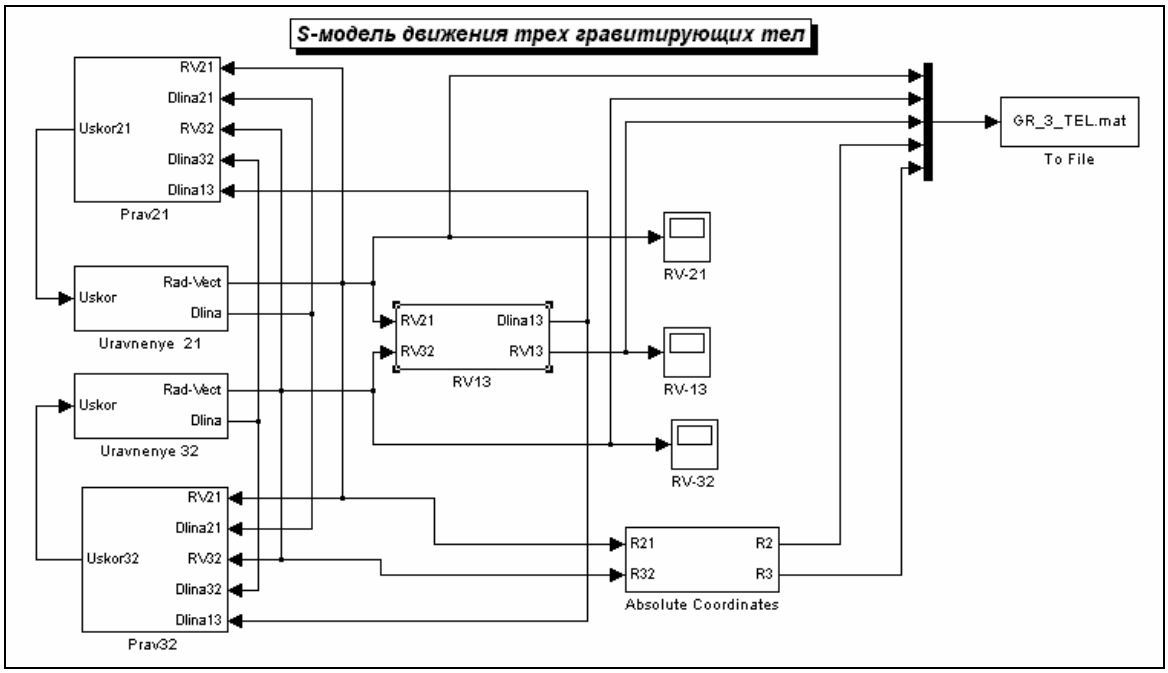


Рис. 7. 97. Блок-схема S-модели GR_3_TEL. mdl

Основные действия по численному интегрированию дифференциальных уравнений (14) сосредоточены в подсистемах **`Uravnenye 21`** и **`Uravnenye 32`**. Здесь (рис. 7. 98) производится двойное интегрирование ускорений, а также вычисляется текущее расстояние (длина вектора) между вторым и первым телами. Входной величиной обоих подсистем является векторная величина, элементы которого представляют собой проекции вектора ускорения на оси X, Y и Z. Выходных величин две. Одна их них – вектор из трех элементов, являющихся текущими проекциями радиус-вектора на те же оси. Второй выход – скалярная величина – длина этого радиус-вектора.

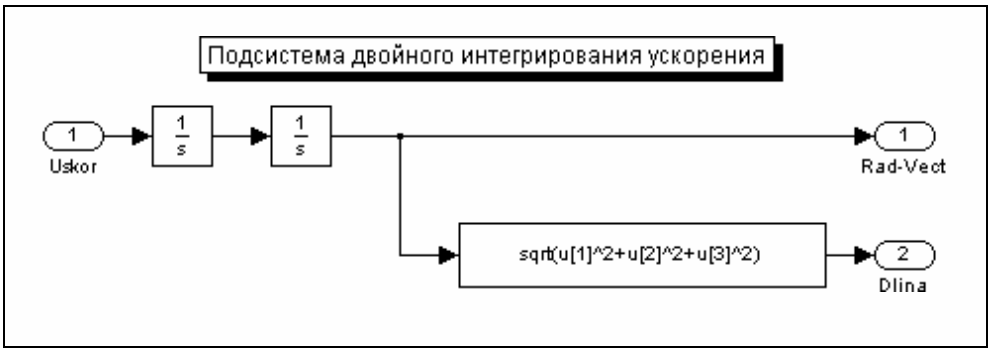


Рис. 7. 98. Блок-схема подсистемы Uravnenye 21

Блок, реализующий вычисление длины вектора по его составляющим, реализован на основе блока **`Fcn`** из раздела **User-Defined Functions**.

Примечание. При написании арифметических выражений в окнах настраивания блоков Simulink используются правила М-языка. Однако *указание индексов для векторов и матриц должно осуществляться в квадратных скобках* (а не в круглых, как в М-языке)

Важной и характерной особенностью этих подсистем, отличающей их от использованной ранее подсистемы **`Маятник`** (рис. 7. 89), является то, что в них интегрируется не скалярная функция времени, а векторная, состоящая сразу из трех скалярных величин. Например, в подсистеме **`Uravnenye 21`** производится одновременное интегрирование ускорений x''_{21}, y''_{21} и z''_{21}. Достигается это с помощью только двух блоков **`Integrator`**. Для этого достаточно в блоке настраивания интеграторов ввести в качестве начального условия не одну величину, а вектор из трех величин, каждая из которых является начальным условием соответствующего элемента вектора, поступающего на вход интегратора. Например, вектором начальных условий для первого интегратора является **`[V21x,V21y,V21z]`** (см. рис. 7. 99).

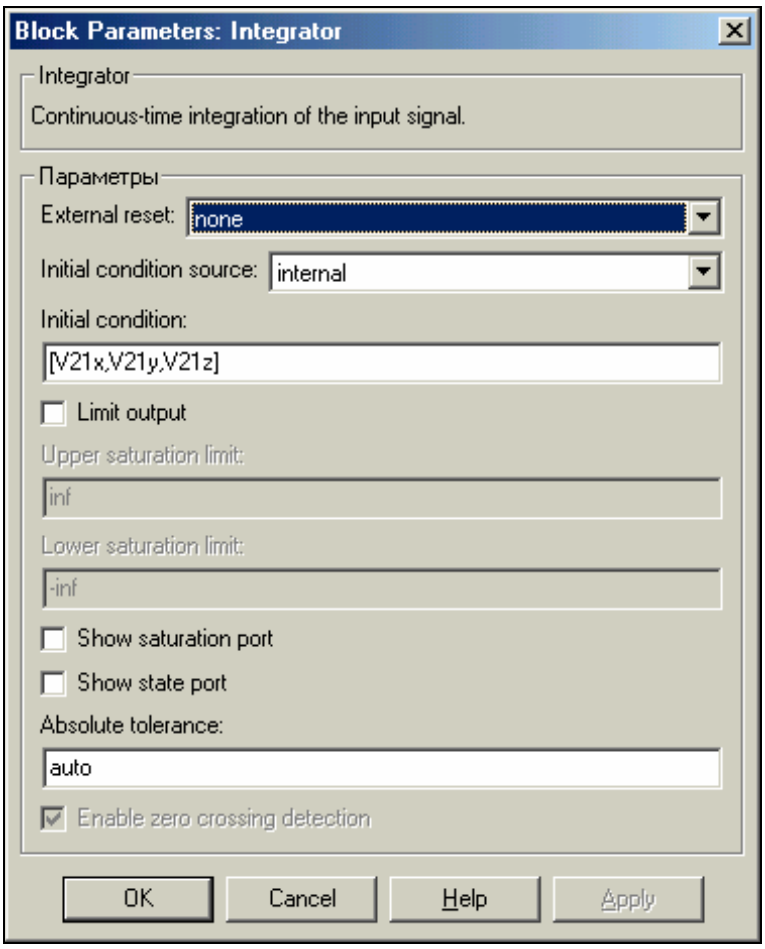


Рис. 7. 99. Окно настраивания первого блока Integrator подсистемы Uravnenye 21

Аналогично, во втором интеграторе установлен вектор **[X210,Y210,Z210]** начальных условий.

То же сделано в подсистеме **Uravnenye 32**, которая вычисляет текущие составляющие вектора $\mathbf{r}_{32}$, полученные двойным интегрированием ускорения $\mathbf{r}''_{32}$. Здесь в интеграторах установлены начальные условия в виде векторов **[V32x,V32y,V32z]** и **[X320,Y320,Z320].**

Подсистемы **Prav 21 и Prav 32** формируют правые части первого и второго уравнений (14), т. е. ускорения в виде вектора из трех элементов. Выходная величина у этих подсистем – одна. Например, в подсистеме **Prav 21** (рис. 7. 100) выходом является вектор Uskor21, составляющими которого являются проекции ускорения на оси X, Y и Z.

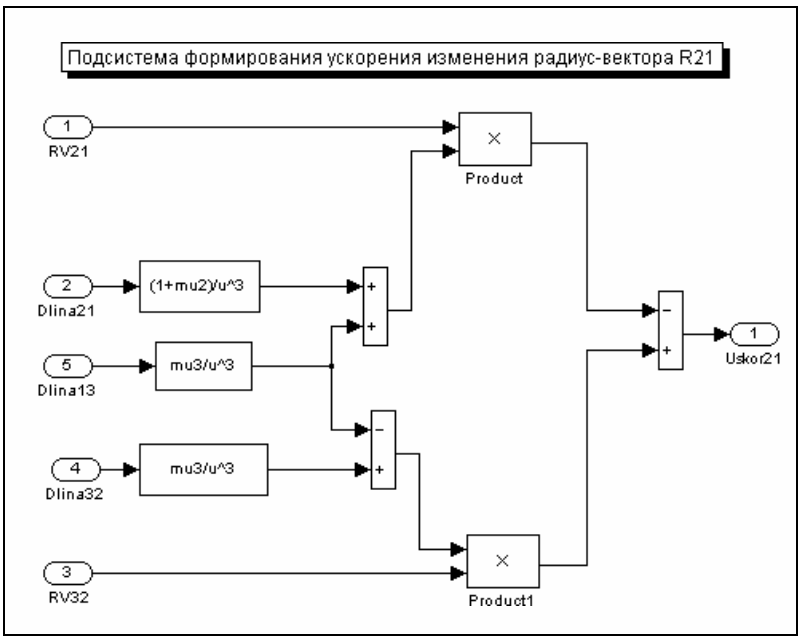


Рис. 7. 100. Блок-схема подсистемы Prav 21

Входами являются два вычисленных ранее вектора RV21 и RV32, состоящие из проекций радиус-векторов $\mathbf{r}_{21}$ и $\mathbf{r}_{32}$, и три скалярные величины, равные текущим значениям длин трех радиус векторов, соединяющих три тела.

В подсистеме **`Prav 21`** используются три блока типа **`Fcn`** для вычисления коэффициентов при векторах в правой части первого уравнения (14) и два блока **`Product`** для перемножения вектора на скалярный коэффициент.

Блок схема подсистемы **`RV13`**, вычисляющей вектор $\mathbf{r}_{13}$ и его длину по известным векторам $\mathbf{r}_{21}$ и $\mathbf{r}_{32}$, приведена на рис. 7. 101.

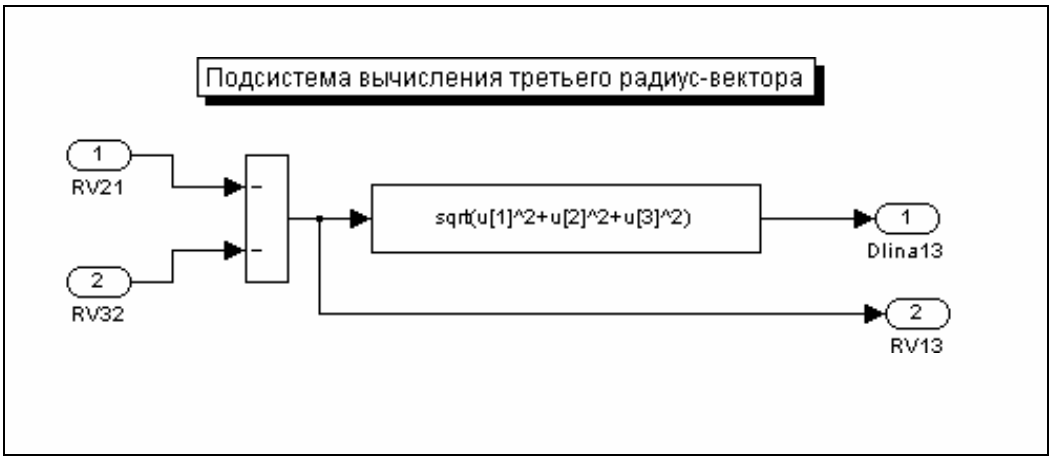


Рис. 7. 101. Блок-схема подсистемы RV13

На рис. 7.102 показана блок-схема подсистемы **`Absolute Coordinates`**. Она вычисляет абсолютные координаты второго и третьего тела относительно центра масс системы трех тел.

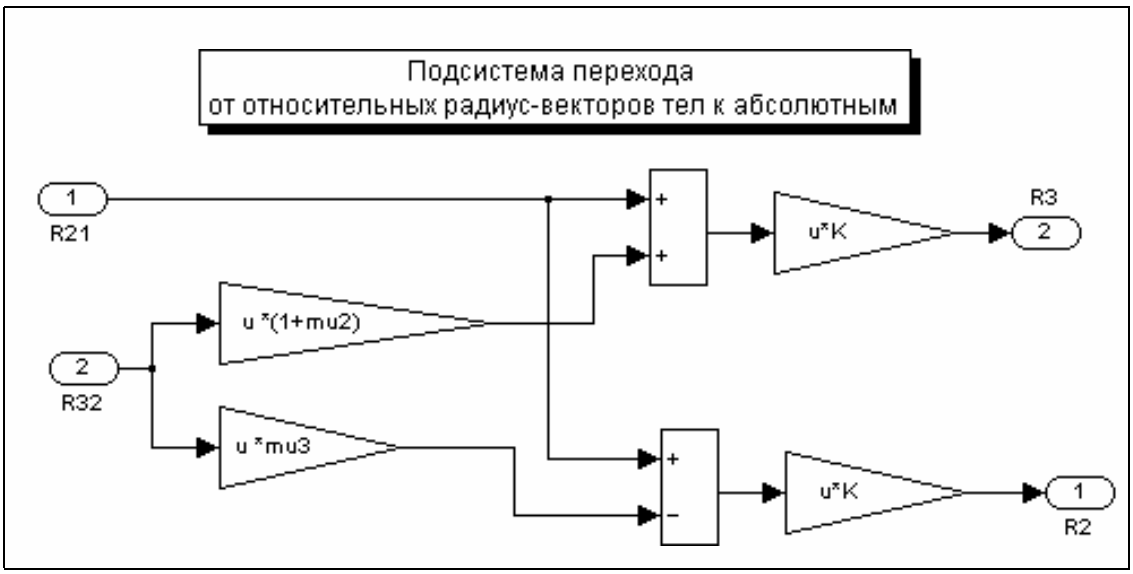


Рис. 7. 102. Блок-схема подсистемы Absolute Coordinates

Блок **`To File`** производит запись поступающих на его вход данных в MAT-файл GR_3_TEL. mat. Эти данные записываются на диск в виде матрицы под именем RV (рис. 7. 103) в следующем порядке:

- первую строку образует массив значений модельного времени, для которых вычислены подаваемые на вход блока величины (хотя, как видно из рис. 7.97, на вход не подается время);
- вторую строку образуют значения первого элемента входного векторного сигнала (ему соответствует верхний сигнал, поступающий на вход блока Mux, выходной сигнал которого поступает на вход блока **`To File`**), соответствующие моментам времени, записанным в первой строке;
- остальные строки заполняются значениями остальных элементов векторного сигнала входа в порядке следования элементов этого вектора.

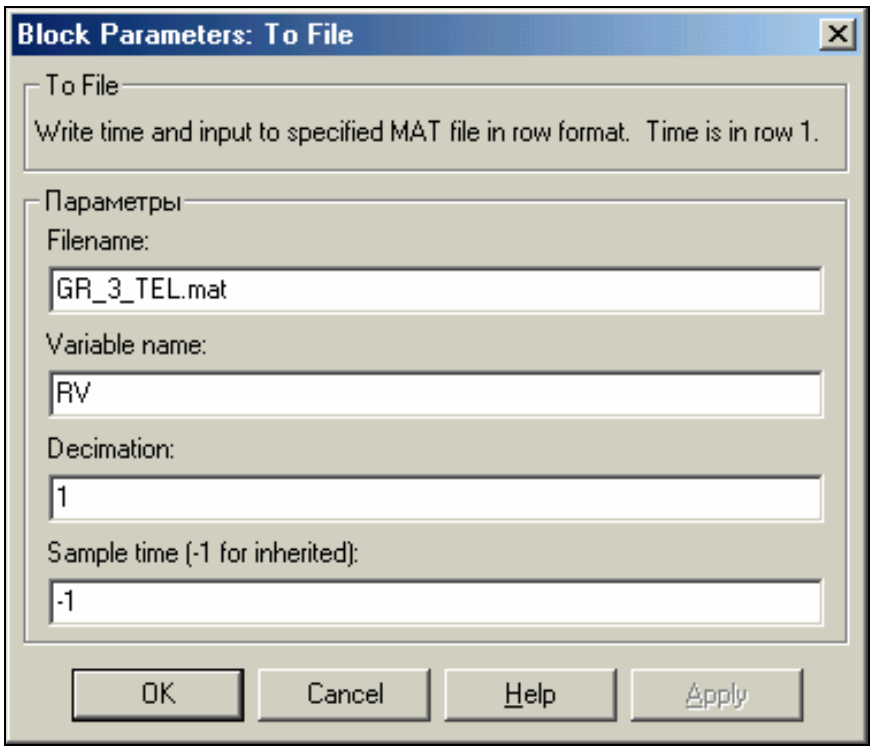


Рис. 7. 103. Заполнение окна настраивания блока To File

Это обстоятельство следует учитывать при считывании данных из сформированного MAT-файла.

Для того, чтобы провести моделирование по созданной S-модели и оформить графически полученные результаты следует произвести следующие операции:

1) ввести исходные данные в рабочее пространство;
2) запустить S-модель GR_3_TEL.mdl на моделирование; при этом результаты моделирования будут записаны в MAT-файл GR_3_TEL. mat;
3) загрузить содержимое MAT-файла GR_3_TEL. mat в рабочее пространство, введя в командном окне **MatLab** команду

```
load GR_3_TEL
```

при этом все данные, записанные в этом MAT-файле, будут записаны в рабочее пространство в виде матрицы под именем RV размером (16×n), где n – размер вектора значений времени;
4) используя данные матрицы RV, создать графическое и текстовое оформление результатов моделирование в графическом окне **MatLab**.

Все эти операции можно автоматизировать, запрограммировав их в отдельной M-программе, которую удобно оформить в виде M-файла ***GR_3_TEL_upr*** (управляющая программа для S-модели GR_3_TEL). При этом вызвать на моделирование S-модель из программы можно с помощью команды

```
sim('имя_S-модели')
```

Ниже приведен текст управляющей программы.

```
% GR_3_TEL_upr
% Программа управления для S-модели GR_3_TEL
% Лазарев Ю. Ф. 25-01-2004
%       1. Ввод данных в рабочее пространство
mu2=0.1;    mu3=0.01;
X210=1; Y210=0; Z210=0;
V21x=0; V21y=1; V21z=0;
X320=0.1; Y320=0.0; Z320=0;
V32x=0; V32y=-1; V32z=0;
%       2. Запуск S-модели
sim('GR_3_TEL')
%       3. Загрузка MAT-файла
load GR_3_TEL;
%       4. Присвоение значений из данных MAT-файла новым переменным
t=RV(1,:);
X21=RV(2,:);    Y21=RV(3,:);    Z21=RV(4,:);
X32=RV(5,:);    Y32=RV(6,:);    Z32=RV(7,:);
X13=RV(8,:);    Y13=RV(9,:);    Z13=RV(10,:);
X2=RV(11,:);    Y2=RV(12,:);    Z2=RV(13,:);
X3=RV(14,:);    Y3=RV(15,:);    Z3=RV(16,:);
n=length(t);
%       5. Построение графика зависимости координат второго тела
%           движения второго и третьего тел
subplot(2,2,1)
plot(t,X21,t,Y21,'--',t,Z21,'.'), grid
title('Движение второго тела относительно первого')
xlabel('Время (безразмерное)')
ylabel('Координаты (безразмерные)')
legend('X','Y','Z',0)
%       6. Построение графика зависимости координат второго тела
%           относительно первого от времени
subplot(2,2,3)
n1=round(n);
plot(t(1:n1),X32(1:n1),t(1:n1),Y32(1:n1),...
      '—',t(1:n1),Z32(1:n1),'.'),grid
title('Движение третьего тела относительно второго')
xlabel('Время (безразмерное)')
ylabel('Координаты (безразмерные)')
legend('X','Y','Z',0)
%       7. Построение пространственных траекторий
%           относительно первого от времени
subplot(4,4,[3,4,7,8,11,12])
plot3(X2,Y2,Z2,'.',X3,Y3,Z3), grid
title('Движение трех гравитирующих тел','FontSize',14)
xlabel('Координата X')
ylabel('Координата Y')
zlabel('Координата Z')
legend('первое тело','второе тело',0)
```

```
%       8. Вывод текстового оформления
subplot(4,4,[15,16])
axis('off')
h= text(0,1.0,'Массовые параметры (относительные):');
h= text(-0.1,0.9,['mu1 = 1;    ',sprintf('mu2 = %g;    ',mu2),sprintf('mu3 = %g',mu3)]);
h= text(0.0,0.8,'Начальные координаты (безразмерные):');
h= text(-0.1,0.7,['второго тела относительно первого:',...
        sprintf('X21 = %g;    ',X210),sprintf('Y21 = %g;    ',Y210),...
         sprintf('Z21 = %g;    ',Z210)]);
h= text(-0.1,0.6,['третьего тела относительно второго:    ',
      sprintf('X32 = %g;    ',X320),sprintf('Y32 = %g;    ',Y320),...
      sprintf('Z32 = %g;    ',Z320)]);
h= text(0,0.5,'Начальные скорости (безразмерные):');
h= text(-0.1,0.4,['второго тела относительно первого:     ',...
      sprintf('V21x = %g;   ',V21x),sprintf('V21y = %g; ',V21y),...
      sprintf('V21z = %g;    ',V21z)]);
h= text(-0.1,0.3,['третьего тела относительно второго:    ',...
      sprintf('V32x = %g;   ',V32x),sprintf('V32y = %g;   ',V32y),...
      sprintf('V32z = %g;    ',V32z)]);
h= text(-0.1,0.1,'_____________________________________________ ');
h= text(-0.1,-0.1,'Программа GR-3-TEL-upr  Автор - Лазарев Ю. Ф.     25-01-2004');
h= text(-0.1,-0.2,' _____________________________________________ ');
```

Запуск этой программы позволяет сразу получать результаты моделирования в виде, приведенном на рис. 7. 104…7. 106.

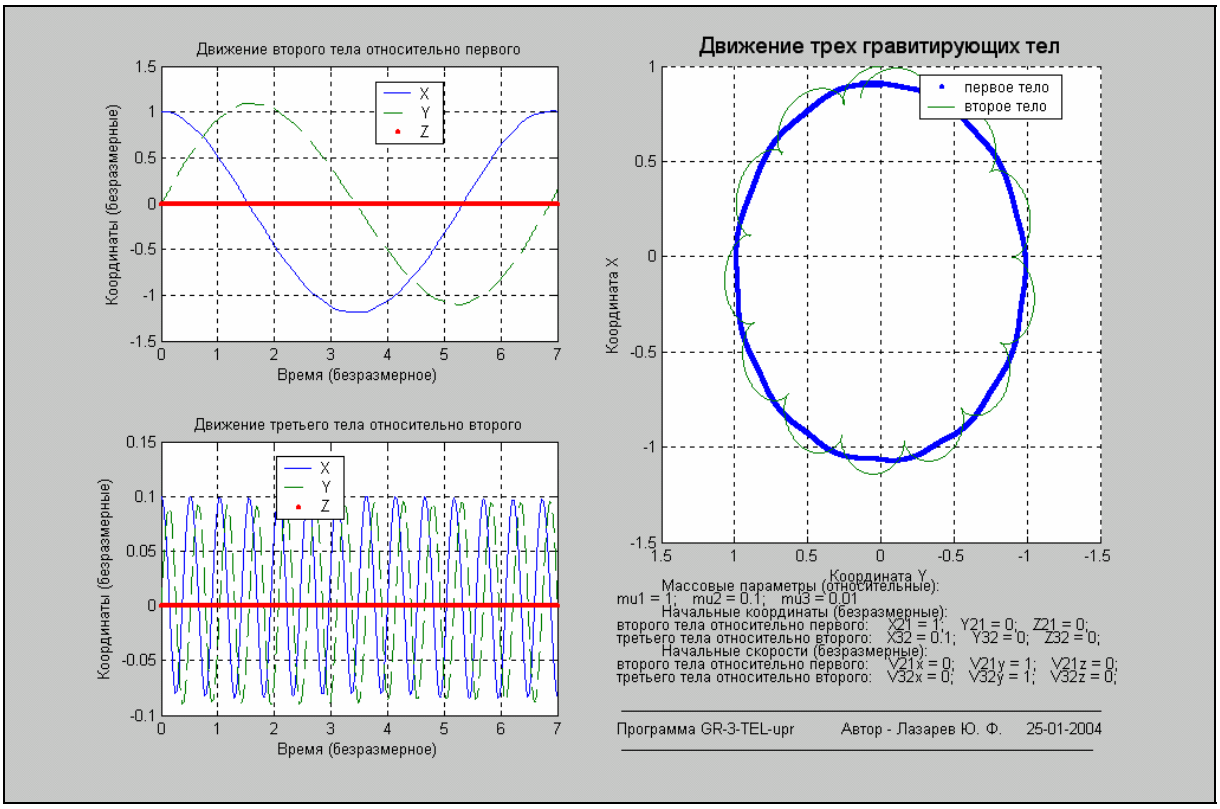


Рис. 7. 104. Плоское орбитальное движение трех тел в одном направлении

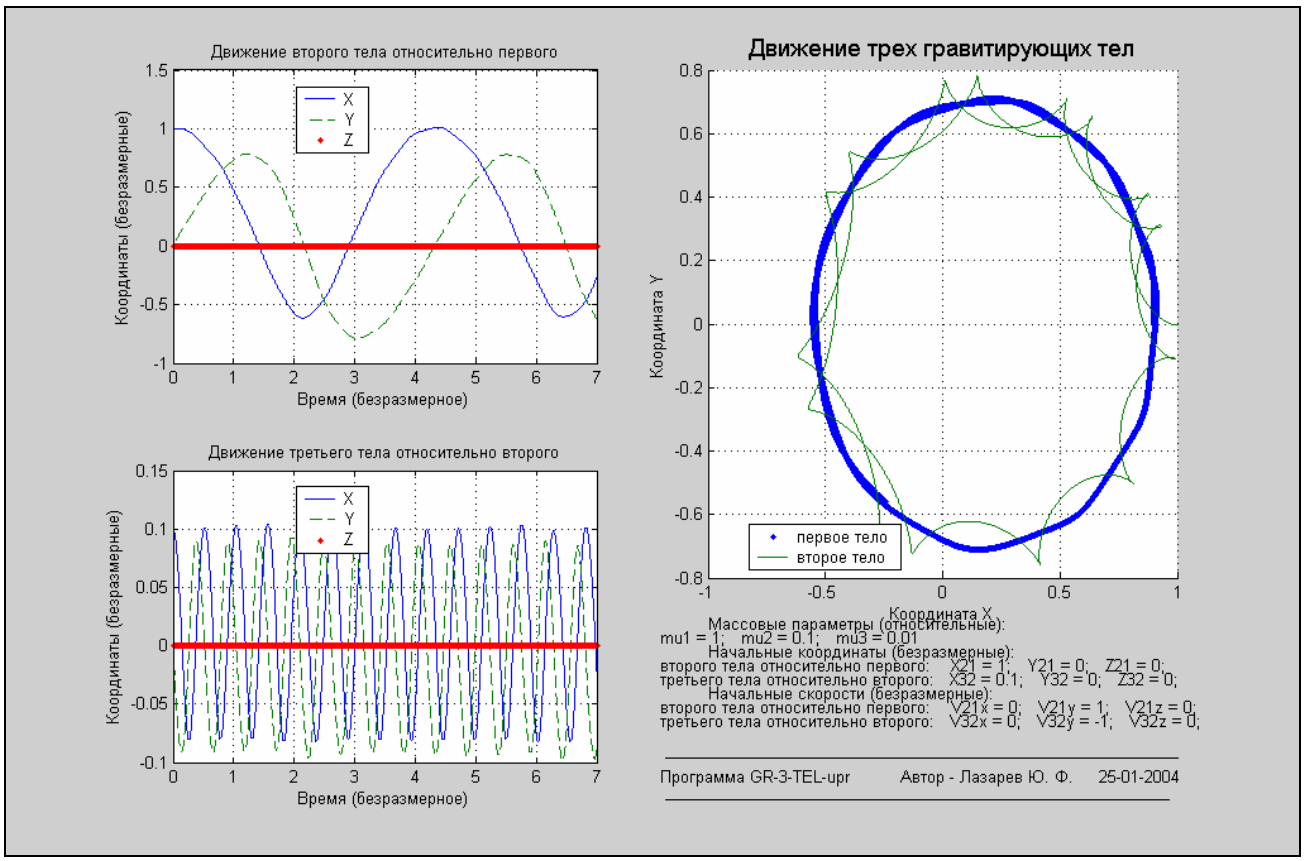


Рис. 7. 105. Плоское орбитальное движение трех тел в противоположных направлениях

Представленные результаты соответствуют трем случаям:

1) относительные орбитальные движения второго тела относительно первого и третьего относительно второго происходят в одной плоскости (XY) в одном направлении (орбитальные моменты однонаправлены – вдоль оси Z);
2) указанные орбитальные движения происходят в одной плоскости (XY) в противоположных направлениях (орбитальные моменты направлены противоположно);
3) указанные орбитальные движения происходят во взаимно-перпендикулярных плоскостях.

Во всех случаях отношение масс тел первого, второго и третьего тел равно 100:10:1.

Относительные начальные скорости тел во всех случаях приняты одинаковыми по величине ($V_{21} = 1$ и $V_{32} = 1$).

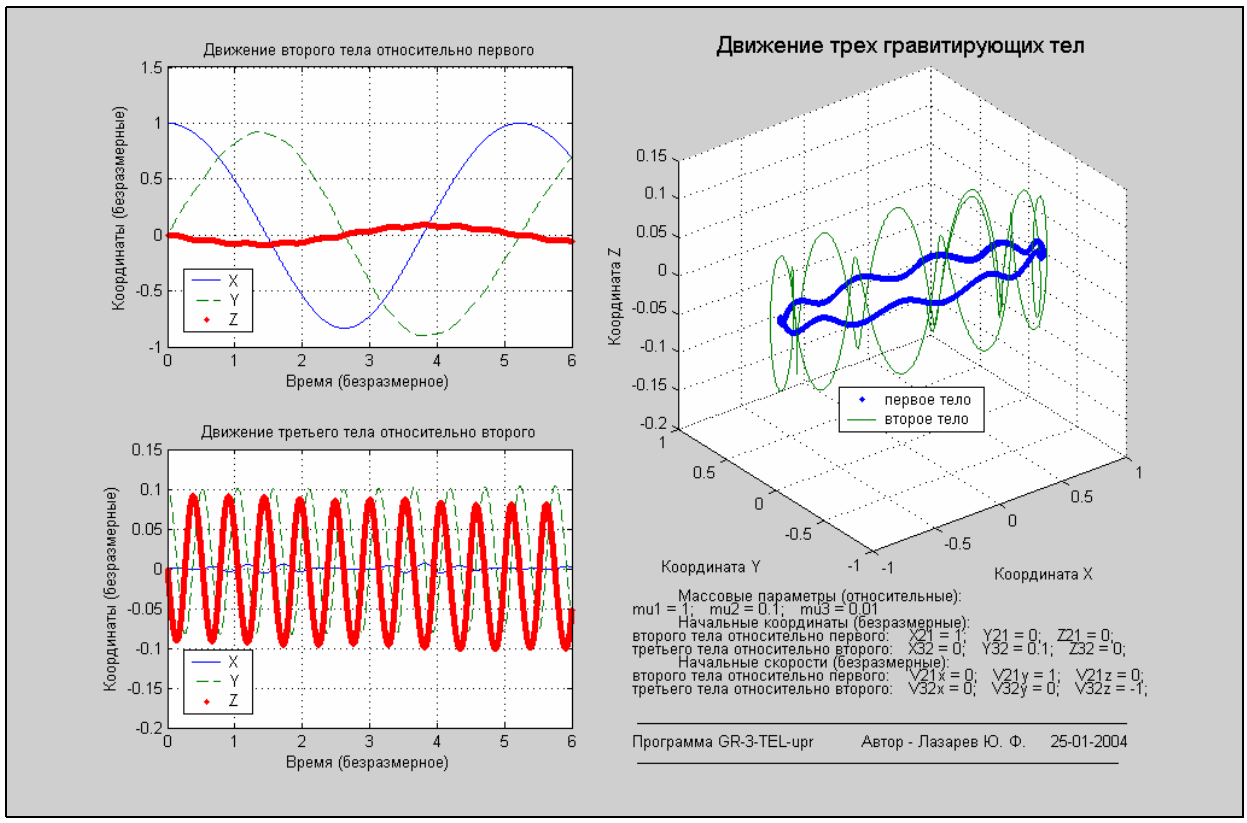


Рис. 7. 106. Орбитальное движение в случае перпендикулярности плоскостей орбит

Обратите внимание на различия в параметрах орбиты среднего (второго) тела и периода этого орбитального движения.

7.4. Вопросы для самопроверки

1. Каковы преимущества использования пакета **Simulink** для решения вычислительных задач по сравнению с программированием их непосредственно в среде MatLab?

2. Присутствие блоков каких разделов библиотеки **SIMULINK** совершенно необходимо в блок-схеме любой S-модели?

3. Каково основное предназначение блоков раздела **Source** библиотеки **SIMULINK**?

3. Каково основное предназначение блоков раздела **Sinks** библиотеки **SIMULINK**?

4. Блоки какого раздела библиотеки **SIMULINK** обеспечивают возможность пользователю создавать собственные блоки?

5. Что такое подсистема и как ее создать?

6. Какими средствами можно обеспечить передачу данных из среды MatLab в S-модель и обратно?

7. В чем основное функциональное отличие блока **`Fcn`** от блока **`MATLAB Fcn`**?

8. В чем главное преимущество блока **`S-Function`** по сравнению со всеми остальными блоками библиотеки **SIMULINK**, позволяющими пользователю создавать собственные блоки?

9. В чем заключаются преимущества использования подсистем?

10. Можно ли обеспечить одновременное интегрирование нескольких процессов одним блоком Integrator?

Урок 8. Взаимодействие MatLab с Simulink

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, методов численного интегрирования) 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» нелинейностями (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, проще и быстрее составлять процедуры их правых частей в виде программы, чем составлять соответствующую блок-схему.

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.

8.1. Объединение S-моделей с программами MatLAB

Очевидно, чтобы осуществить объединение программы с S-моделью 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 этой среды;
- средства вызова программ MatLab из S-модели;
- средства создания S-блоков не только из других готовых блоков, а и путем использования программ, написанных на M-языке MatLab.

Рассмотрим подробнее эти средства, предоставляемые системой MatLab.

8.1.1. Управление процессом моделирования в системе Simulink

Каждый блок S-модели имеет такие внутренние характеристики (рис. 8.1):
- вектор входных величин ***u***;
- вектор выходных величин ***y***;
- вектор состояния ***x***.

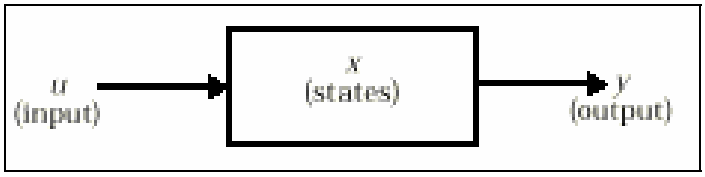


Рис. 8. 1. Схема взаимодействия величин, определяющих текущее состояние S-блока

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. Вычисление 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.

С перечнем программ решателей (интеграторов), которые прилагаются к пакету **Simulink**, можно ознакомиться в окне модели через меню Simulation ► Simulation Parameters ► Solvers – см. рис. 8. 2…8.5.

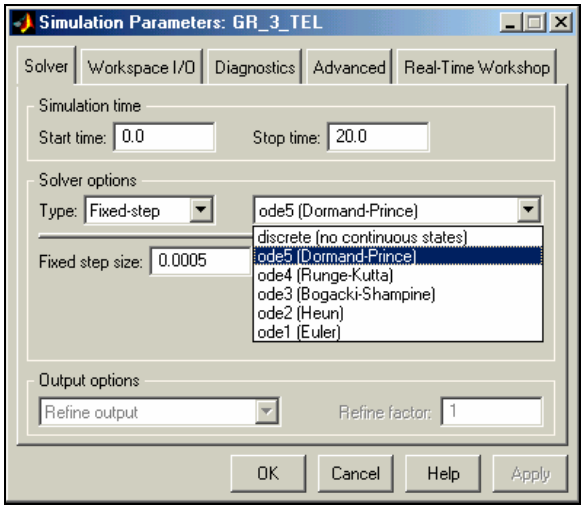


Рис. 8. 2. Вкладка Solver. Список решателей с фиксированным шагом

Вкладка *Solvers* (рис. 8.2) 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 тип решателей;

- *Solver options* – в котором выбирается конкретный решатель.

Если в окошке *Type* выбран тип решателей *Fixed step* (с фиксированным шагом), в окошке *Solver options* появляется такой (рис. 8. 2) 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 Эйлера (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. Кроме того, возникает еще одно окошко (рис. 8.3) под именем *Mode* (Режим), в котором выбирается один из трех возможных режимов работы:

Auto	автоматический режим;
Single Tasking	однозадачный режим;
Multi Tasking	многозадачный режим.

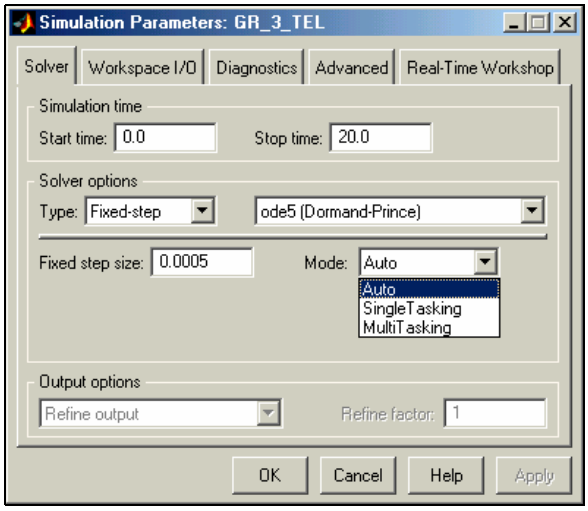


Рис. 8. 3. Вкладка Solver. Список режимов

Если же выбрать тип решателя *Variable step*, в окошке *Solver options* (рис. 8. 4) 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 следующие поля ввода (рис. 8.5):

Max step size	максимальная величина шага;
Min step size	минимальная величина шага;
Initial step size	начальная величина шага;
Relative tolerance	допустимая относительная погрешность;
Absolute tolerance	допустимая абсолютная погрешность.

Все величины (кроме *Relative tolerance*) 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, ее можно установить иной.

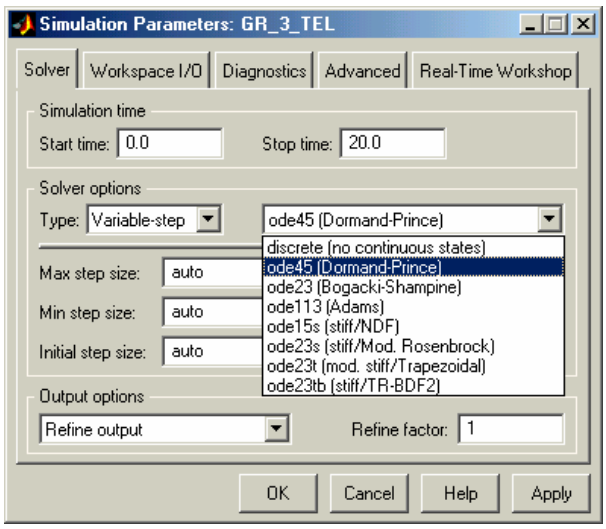


Рис. 8. 4. Вкладка Solver. Список решателей с переменным шагом

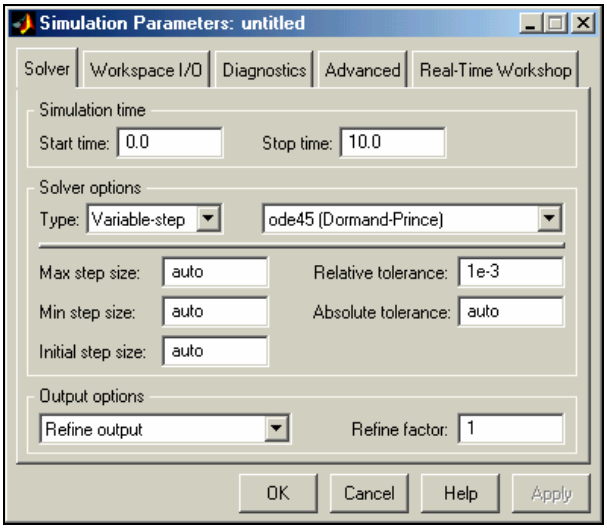


Рис. 8. 5. Вкладка Solver. Параметры решателей с переменным шагом

8.1.2. Обнаружение пересечения нуля

При моделировании систем, содержащих элементы с разрывными характеристиками, такие как реле, люфты, сухое трение, возникает задача максимально точно воспроизвести особенности поведения такой системы в моменты времени, когда разрывная характеристика скачком изменяет свое значение. Обычно в такой момент времени изменяются скачком свойства самой системы, а также начальные условия для продолжения процесса.

Поэтому становится очень важным максимально точно (с машинной точностью) определить момент времени, в который происходит скачкообразное изменение разрывной характеристики и зафиксировать параметры движения системы в этот момент времени. Это необходимо для того, чтобы на следующем шаге начать процесс интегрирования точно с этого момента времени и с новыми начальными условиями, определяемыми особенностями прохождения системы через скачок соответствующей характеристики.

Обычно скачкообразное изменение разрывной характеристики происходит в момент, когда значение одной из переменных состояния системы при своем изменении во времени переходит через некоторый уровень. Причем иногда важно, в каком именно направлении происходит пересечение переменной этого уровня – при увеличении переменной или при ее уменьшении – от этого может зависеть, как именно изменятся (скачкообразно) характеристики системы и значения ее переменных состояния.

Задача обнаружения пересечения сигналом какого либо постоянного уровня легко сводится к задаче выявления пересечения нулевого уровня. Поэтому в дальнейшем будем называть процесс точного определения параметров

состояния системы в момент, когда происходит скачкообразное изменение какой-либо характеристики системы *обнаружением пересечения нуля*.

Simulink использует обнаружение пересечения нуля для того, чтобы выявить резкие изменения 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 претерпевать значительные мгновенные изменения. Простой пример скачков состояния - мяч, 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 возникает ситуация, когда мяч, переходя через контактную точку, как бы проникает сквозь пол. **Simulink** использует обнаружение пересечения нуля, чтобы гарантировать, что момент скачка состояния системы определен точно (с машинной точностью). Вследствие того, что эти моменты скачков состояния определяются точно, в результате численного моделирования не происходит никакого проникания мяча сквозь пол, и переход от отрицательной скорости мяча к положительной в момент контакта происходит чрезвычайно резко.

Чтобы увидеть демонстрацию поведения подскакивающего мяча, введите в командном окне MatLAB команду **`bounce`**. В результате выполнения этой команды возникнет окно с изображением блок-схемы модели поведения мяча (рис. 8. 6). 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.

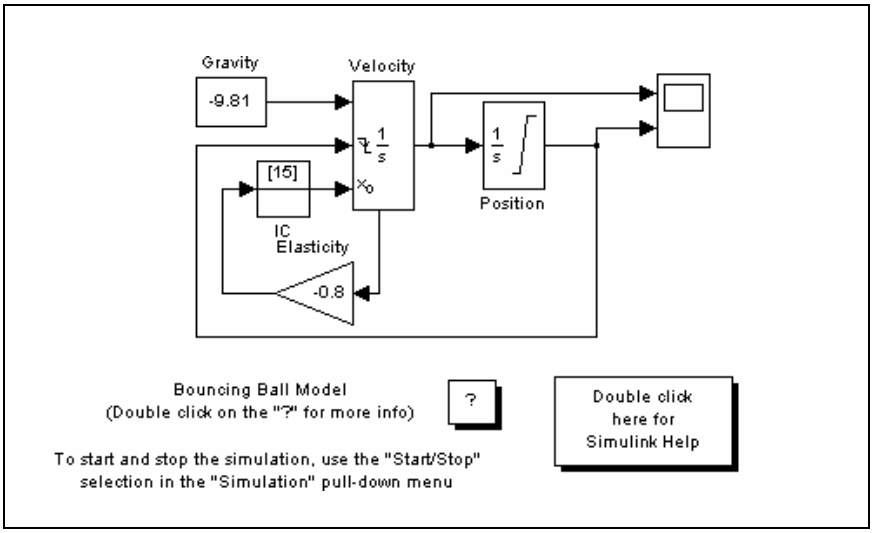


Рис. 8. 6. Блок-схема S-модели Bounce

Более близкое знакомство со вторым интегратором (Position) (рис. 8. 7) дает возможность убедиться, что в нем установлена нижняя граница (нуль) изменений высоты мяча, а в первом (рис. 8. 8) 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 внешним (15 м/с). Кроме того, в первом интеграторе установлены отображение порта состояния. На этот порт подаются значения текущей скорости.

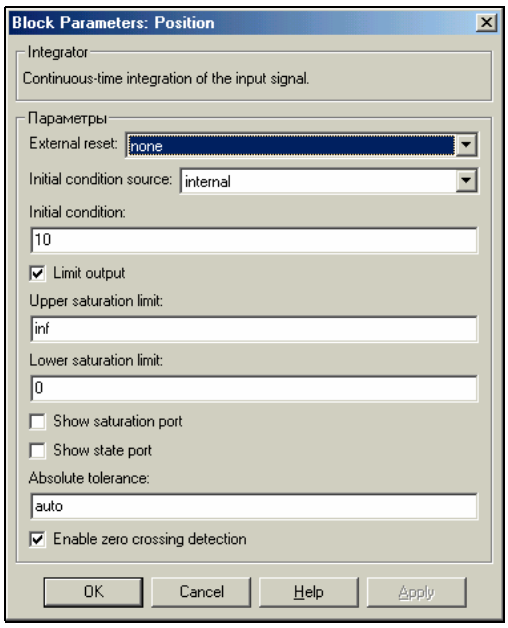


Рис. 8. 7. Окно настраивания блока-интегратора Position

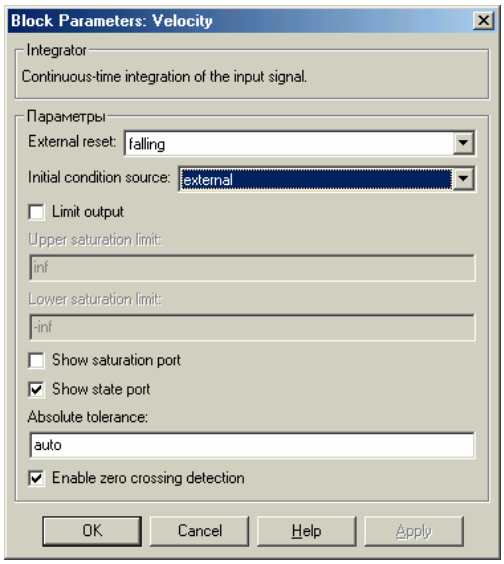


Рис. 8. 8. Окно настраивания блока-интегратора Velocity

Моделирование происходит так. Интегрирование начинается при указанных начальных условиях. В момент, 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 выходной 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. Для этой цели необходимо использовать дополнительный выходной порт интегратора, т. е. установить флажок в окошке *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, которые используют обнаружение 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 (выходной 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 насыщения, 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;
- скользящие режимы;
- автоколебания;
- внезапные переходы от одного из режимов к другому.

8.1.3. 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 M-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** (рис. 8.9)**.** 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**.

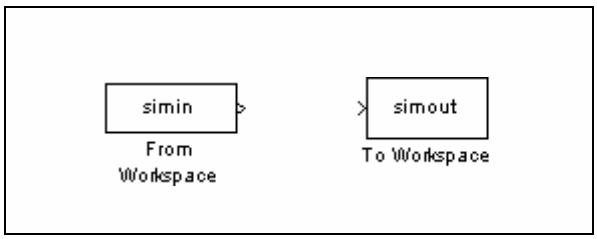


Рис. 8. 9. Внешний вид заготовок блоков From Workspace и To Workspace

Окно настраивания блока From Workspace приведено на рис. 8. 10, а блока To Workspace – на рис. 7.20.

Для определения процесса, который будет использоваться в S-модели, следует (см. рис. 8. 10) 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, например: [T,D]. В этом случае из массива T рабочего пространства будут считаны все значения, которые будут играть в S-модели роль значений модельного времени, а выходная величина блока при моделировании будет принимать в моменты времени, соответствующие записанным в массиве T, значения, записанные в массиве D.

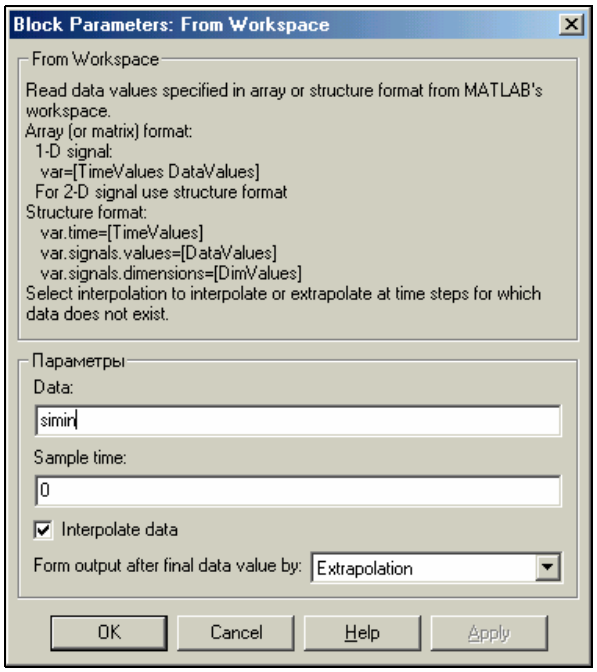


Рис. 8. 10. Окно настраивания блока From Workspace

При этом, если реальные значения времени при моделировании не совпадут с записанными в T, происходит линейная интерполяция значений массива D, соответствующих предшествующему и последующему значениям времени в массиве T.

Для записи полученного процесса в рабочее пространство (рис. 7. 20) следует указать в окошке *Variable name* имя, под которым его нужно сохранить в рабочем пространстве системы MatLab. Соответствующие моменты модельного времени при этом не записываются.

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:

1) в блок-схему S-модели вставить блок порта входа **`In`** и подключить к одному из блоков S-модели;

2) в окне S-модели вызвать Simulation ► Simulation Parameters ► Workspace I/O (рис. 8. 11);

3) установить флажок *Input* в области **`Load from workspace`**, и в поле справа ввести вектор из двух имен – первое (например, t) – имя вектора значений аргумента, второе (к примеру, u) - имя вектора значений входного сигнала при этих значениях аргумента;

4) установить значение этих векторов в среде MatLab, например, таким образом

```
t = (0:0.1:1)';
u = [sin(t), cos(t), 4*cos(t)];
```

5) запустить S-модель на моделирование.

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 окне S-модели вызвать Simulation ► Simulation Parameters ► Workspace I/O;
- в области **`Save to workspace`** установить флажки *Time* и *Output.*

В этом случае значения модельного времени будут записываться в рабочее пространство в массив под именем *tout*, а соответствующие значения выходных процессов при этих значениях времени – в столбцы матрицы *yout* (в первый столбец – процесс, который подан 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 новыми именами.

Так же, активизируя (рис. 8. 11) 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*.

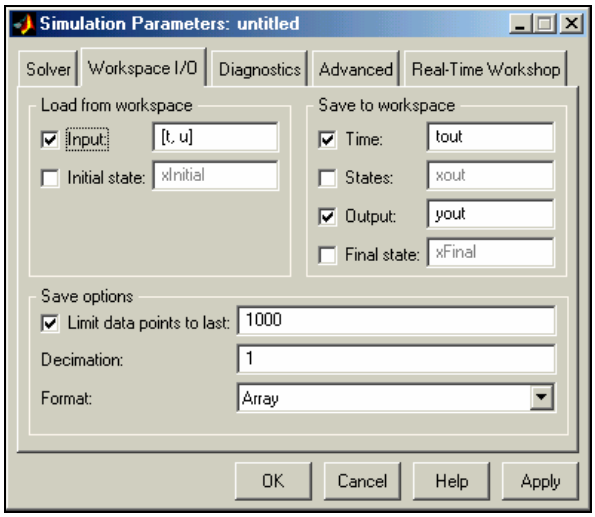


Рис. 8. 11. Установка входного процесса в S-модель

8.1.4. 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.

8.1.5. Образование S-блоков путем использования программ MatLab. S-функции

В системе MatLab предусмотрен механизм преобразования некоторых процедур, написанных на языках высокого уровня, в блок S-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, дифференциальные или разностные), которыми описывается блок, моделируемый 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 ►User-Defined Functions**) стандартного блока **`S-function`** (рис. 8. 12) в окно блок-схемы, внутри которой будет создаваться новый S-блок;

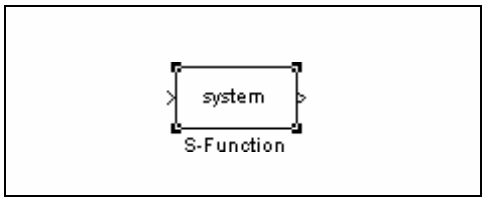


Рис. 8. 12. Внешний вид заготовки S-функции

3) двойного щелчка мышью на изображении этого блока, что приводит к возникновению на экране нового окна (рис. 8. 13), 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 M-файла S-функции;

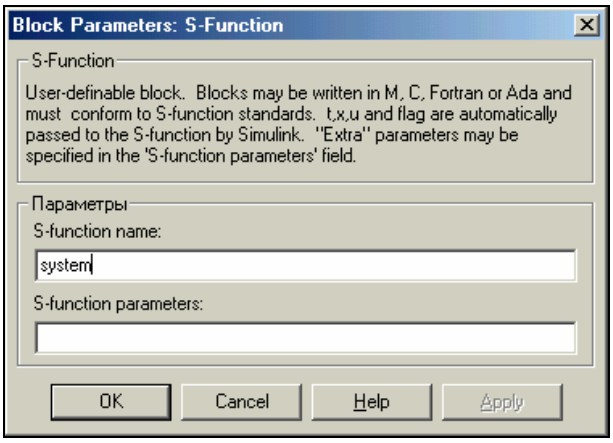


Рис. 8. 13. Окно настраивания блока S-function

4) введения в указанные окна имени написанного M-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. 8. 14):

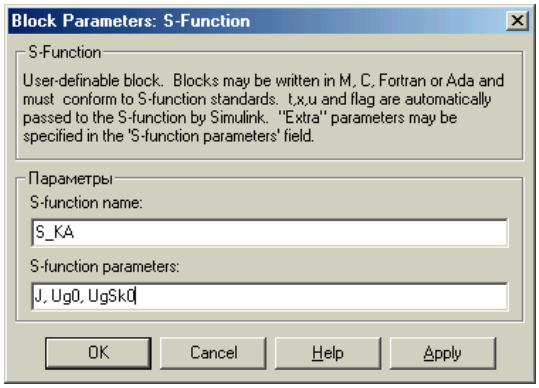


Рис. 8. 14. Окно настраивания блока S-function после ввода параметров

5) теперь нужно нажать мышью на кнопку <OK> этого окна; если система обнаружит М-файл с введенным именем в достижимых ей путях, окно (рис. 8. 14) исчезнет, а изображение блока в окне блок-схемы изменит свой вид (рис. 8. 15) – внутри него возникнет введенное имя S-функции.

S-блок на основе S-функции, которая содержится в М-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 станет векторный сигнал ***y***, который сформирован S-функцией во внутренней процедуре **mdlOutputs**.

8.1.6. Примеры создания S-функций

Образуем S-функцию, которая реализует динамические свойства твердого тела в его вращательном движении. Для описания динамики тела воспользуемся динамическими уравнениями Эйлера в матричной форме:

$$\mathbf{J}\cdot\frac{d\boldsymbol{\omega}}{dt}+(\boldsymbol{\omega}\times)\cdot(\mathbf{J}\cdot\boldsymbol{\omega})=\mathbf{M}\,, \tag{8.1}$$

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; $(\boldsymbol{\omega}\times)$ - кососимметричная матрица вида

$$(\boldsymbol{\omega}\times)=\begin{bmatrix}0 & -\omega_z & \omega_y\\ \omega_z & 0 & -\omega_x\\ -\omega_y & \omega_x & 0\end{bmatrix}, \tag{8.2}$$

составленная из тех же проекций; $\mathbf{M}$ - матрица-столбец из проекций вектора момента внешних сил на связанные оси. Такими уравнениями описывается, например, вращательное движение космического аппарата. Поэтому далее иногда тело будет отождествляться с космическим аппаратом.

Создадим М-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 от 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,IJ);
  case 3,
     sys = mdlOutputs(x,M,J,IJ);
  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,IJ)
 % 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);
z=IJ*(M-cross(om,J*om));       % ДИНАМИЧЕСКИЕ уравнения Ейлера
 % Конец процедуры mdlDerivatives
 %=================================
function y = mdlOutputs(x,M,J,IJ)
y(1:3)=x;
om=x(1:3);
zom=IJ*(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:

y=[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. Перетянем в него стандартный блок S-функции из раздела **User-Defined Functions**.

Дважды щелкнув мышью на изображении этого блока, вызовем его окно настраивания и запишем в него название M-файла созданной S-функции и его параметры (рис. 7.101). Нажмем кнопку <OK>.

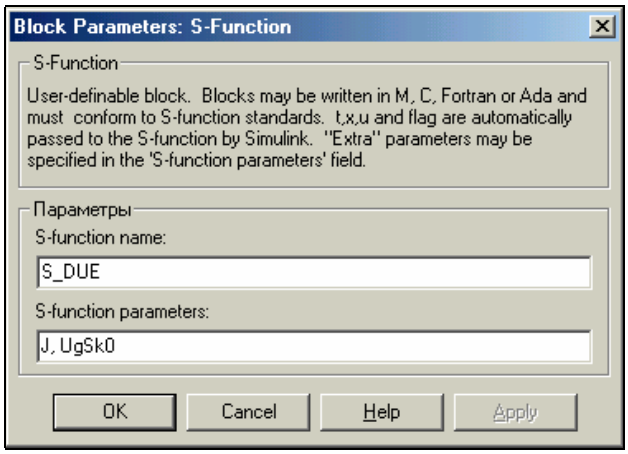


Рис. 8. 15. Окно настраивания блока S_DUE

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 блок-схему 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 приведено во вращение 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 Эйлера 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:

$$\omega_X = \omega_0 \cdot \cos(k\Omega t); \qquad \omega_Y = \Omega; \qquad \omega_Z = -\omega_0 \cdot \sin(k\Omega t), \tag{8.3}$$

причем

$$k = \frac{J - J_e}{J_e}. \tag{8.4}$$

Итак, если для образованной модели обеспечить указанные условия, то, если она правильна (адекватна), при моделировании должны получить результаты соответствующие (8.3).

Добавим в блок-схему блок констант, который формирует нулевой вектор моментов внешних сил (рис. 8. 16), 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
```

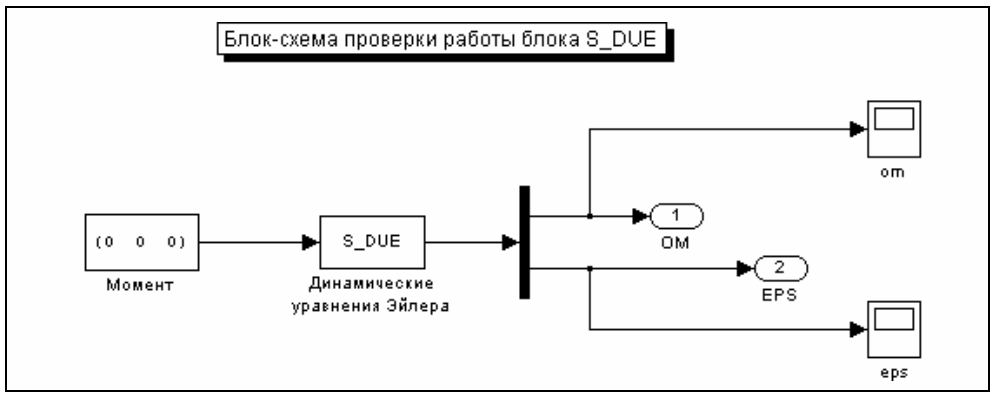


Рис. 8. 16. Блок-схема проверки работы блока S_DUE

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. 8. 17 и запустить блок схему на моделирование. По окончании процесса моделирования, обращаясь к окнам блоков **Scope**, можно убедиться, что созданная модель работает полностью адекватно.

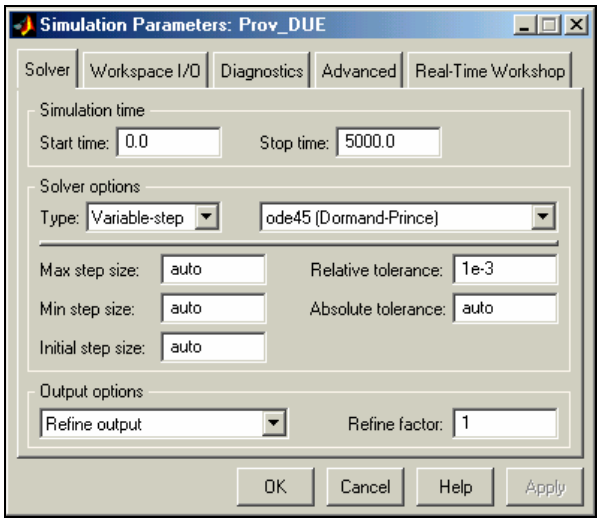


Рис. 8. 17. Параметры процесса интегрирования в S-модели Prov_DUE

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 рационально передать результаты в рабочее пространство с помощью введения выходных портов (рис. 8. 16) и посылания на них тех сигналов, которые нужно представить графически, а потом построить необходимые графики, пользуясь 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]; % Ввод значений матрицы инерции
UgSk0=[0.001 0.01 0];      % Ввод начальных значений
                           % проекций угловой скорости тела
% Установка параметров моделирования
options=simset('Solver','ode45','RelTol',1e-6);
sim('Prov_DUE',5000,options); % МОДЕЛИРОВАНИЕ на S-модели
% Формирование 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. 8. 18. Теперь читатель может наглядно убедиться в правильности модели.

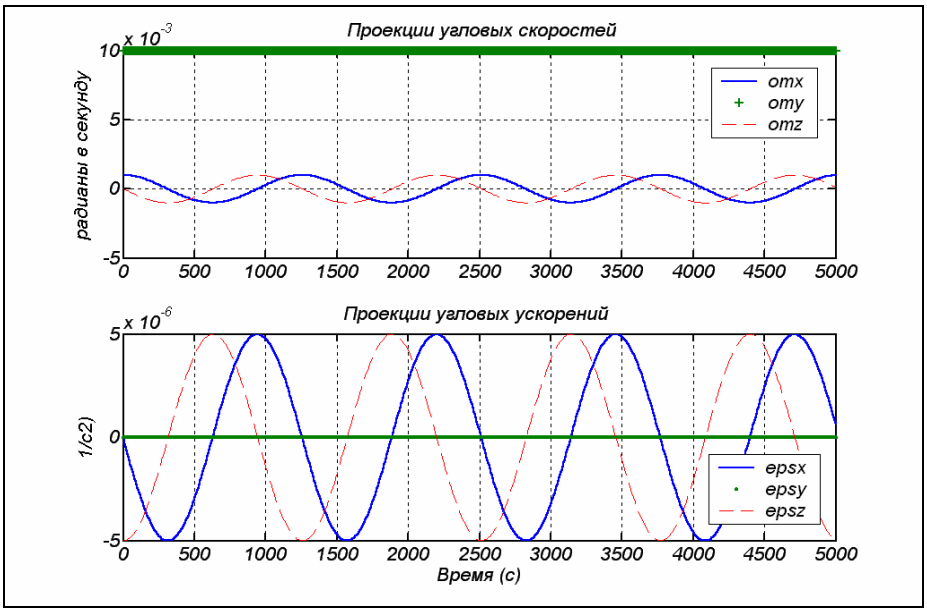


Рис. 8. 18. Свободное движение симметричного гироскопа (космического аппарата)

Отметим, что промоделированное движение соответствует свободному движению симметричного гироскопа – его нутационным колебаниям. В гироскоп тело превращает приведение его в сравнительно быстрое вращение

вокруг одной из его осей ($\omega_Y = 0{,}01c^{-1}$). Матрица моментов инерции принята диагональной, т. е. тело предполагается динамически сбалансированным относительно осей XYZ. Наконец, моменты инерции относительно осей, ортогональных оси собственного вращения тела приняты равными. Это означает, что гироскоп является динамически симметричным телом с осью фигуры Y.

8.1.7. Запуск M-программ из S-модели

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, вызов перед началом загрузки S-модели по имени, к примеру, **MODEL.mdl** некоторого M-файла, например, по имени **PERVdan**, который содержит операции присваивания исходных значений всех данных, можно осуществить, если при создании S-модели с этим именем в командном окне MatLab ввести команду

```
set_param('MODEL','PreLoadFcn','PERVdan')
```

Эта команда свяжет файл **PERVdan.m** с S-моделью **MODEL.mdl** так, что он будет автоматически вызываться и исполняться при вызове этой S-модели. 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')
```

Вообще при помощи функции set_param можно установить в S-модели значения многих ее параметров, в том числе и параметров отдельных ее блоков.

В общем виде обращение к ней может иметь вид

```
set_param('Имя_S-модели/Имя_блока','Параметр1',Значение1,
                    'Параметр2',Значение2,...)
```

Если указано Имя_блока, то последующие значения присваиваются параметрам этого блока.

Приведем примеры.

Вызов вида
```
set_param('MODEL','Solver','ode15s','StopTime','3000')
```
приведет к установлению в S-модели MODEL решателя `ode15s` и времени окончания процесса моделирования 5000.

Если обратиться к этой функции так:
```
set_param('MODEL/Уравнение','Gain','1000')
```
то в блоке **Уравнение** S-модели **MODEL** параметру *Gain* будет присвоено значение 1000.

Команда
```
set_param('MODEL/Fcn','Position',[50 100 110 120])
```
установит изображение блока **Fcn** в S-модели **MODEL** в прямоугольник в окне бло-схемы с координатами **[50 100 110 120].**

Если же осуществить такое обращение:
```
set_param('mymodel/Compute','OpenFcn','my_open_fcn')
```
то блок **Compute** S-модели **mymodel** будет связан с M-программой MatLab, записанной в файле my_open_fcn.m. После этого файл my_open_fcn.m будет вызываться на исполнение всякий раз, когда на изображении блока **Compute** дважды щелкнуть мышью.

В случае, когда 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 **Ports & Subsystems**), каждый из которых будет осуществлять вызов соответствующего M-файла.

Эти пустые блоки **Subsystem** в поле блок-схемы модели можно соединить с определенной M-программой, набрав в командном окне команду, аналогичную последней.

Попробуем организовать такую форму управления процессом моделирования для предыдущей модели Prov_DUE.

Для этого создадим на основе ранее созданного файла Prov_DUEupr.m три отдельных файла

Prov_DUE_Pred. m	осуществляющий предварительное присвоение значений исходным величинам;
MENU_DUE.m	осуществляющий диалоговое изменение исходных данных;
Graf_DUE.m	осуществляющий вывод результатов моделирования в графическое окно

Тексты этих программ приведены ниже.

% Prov_DUE_Pred.m

```
% Программа установки начальных (вшитых) значений
% параметров модели Prov_DUE.mdl

% Лазарев Ю.Ф.  28-01-2004

J=[400 0 0; 0 600 0; 0 0 400]; % Ввод значений матрицы инерции
UgSk0=[0.001 0.01 0];          % Ввод начальных значений
                               % проекций угловой скорости тела
% Установка параметров моделирования
hi=10;  TK=5000;
```

% MENU_DUE.m

```
% Программа изменения исходных значений
% параметров модели Prov_DUE.mdl

% Лазарев Ю.Ф.  28-01-2004

k=1;
while k<12
    k=menu('Данные для модели Prov_DUE. Что изменить?',...
            sprintf('Jx = %g',J(1,1)),...
            sprintf('Jy = %g',J(2,2)), ...
            sprintf('Jz = %g',J(3,3)),...
           sprintf('Jxy = %g',J(1,2)),...
            sprintf('Jxz = %g',J(1,3)),...
            sprintf('Jyz = %g',J(2,3)),...
            sprintf('OMx(0) = %g',UgSk0(1)),...
            sprintf('OMy(0) = %g',UgSk0(2)),...
            sprintf('OMz(0) = %g',UgSk0(3)),...
            sprintf('hi = %g',hi),sprintf('TK = %g',TK),...
            'Ничего не менять');
    if k==1
        J(1,1)=input([sprintf('Текущее значение J(1,1)=%g;
      ',J(1,1)),'Установите новое значение J(1,1)=']);
    end
    if k==2
        J(2,2)=input([sprintf('Текущее значение J(2,2)=%g;
      ',J(2,2)),'Установите новое значение J(2,2)=']);
    end
    if k==3
        J(3,3)=input([sprintf('Текущее значение J(3,3)=%g;
      ',J(3,3)),'Установите новое значение J(3,3)=']);
    end
    if k==4
        J(1,2)=input([sprintf('Текущее значение J(1,2)=%g;
      ',J(1,2)),'Установите новое значение J(1,2)=']);
    end
    if k==5
        J(1,3)=input([sprintf('Текущее значение J(1,3)=%g;
      ',J(1,3)),'Установите новое значение J(1,3)=']);
    end
    if k==6
        J(2,3)=input([sprintf('Текущее значение J(2,3)=%g;
      ',J(2,3)),'Установите новое значение J(2,3)=']);
    end
    if k==7
```

```
        UgSk0(1)=input([sprintf('Текущее значение OMx(0)=%g;
      ',UgSk0(1)),'Установите новое значение OMx(0)=']);
    end
    if k==8
        UgSk0(2)=input([sprintf('Текущее значение OMy(0)=%g;
      ',UgSk0(2)),'Установите новое значение OMy(0)=']);
    end
    if k==9
        UgSk0(3)=input([sprintf('Текущее значение OMz(0)=%g;
      ',UgSk0(3)),'Установите новое значение OMz(0)=']);
    end
    if k==10
        hi=input([sprintf('Текущее значение hi=%g;   ',hi),...
            'Установите новое значение hi=']);
    end
    if k==11
        TK=input([sprintf('Текущее значение TK=%g;   ',TK),...
                  'Установите новое значение TK=']);
    end
end
J(2,1)=J(1,2); J(3,1)=J(1,3); J(3,2)=J(2,3);
```

% Graf_DUE.m

```
% Программа построения в графическом окне графиков
% результатов работы модели Prov_DUE.mdl

% Лазарев Ю.Ф.  28-01-2004

% Формирование данных и вывод ГРАФИКОВ
tt=tout;
omx=yout(:,1);      omy=yout(:,2);      omz=yout(:,3);
epsx=yout(:,4);     epsy=yout(:,5);     epsz=yout(:,6);
StrJ=[sprintf('Jx= %g; ',J(1,1)),sprintf('Jy= %g; ',J(2,2)),sprintf('Jz= %g; ',J(3,3))]
StrU=[sprintf('OMx= %g; ',UgSk0(1)),sprintf('OMy= %g; ',UgSk0(2)),sprintf('OMz= %g;
',UgSk0(3))]
StrJ1=[sprintf('Jxy= %g; ',J(1,2)),sprintf('Jxz= %g; ',J(1,3)),sprintf('Jyz= %g;
',J(2,3))]
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')
set(gca,'Fontsize',14)
ylabel('радианы в секунду')
legend('omx','omy','omz',0)
xlabel([StrJ,'           ',StrJ1,'           ',StrU])
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 ')
set(gca,'Fontsize',14)
ylabel('радианы/c^2')
xlabel('     Время (c)       ')
set(gcf,'color','white')
legend('epsx','epsy','epsz',0)
```

Теперь введем в блок-схему Prov_DUE два блока Subsystem (рис. 8. 19).

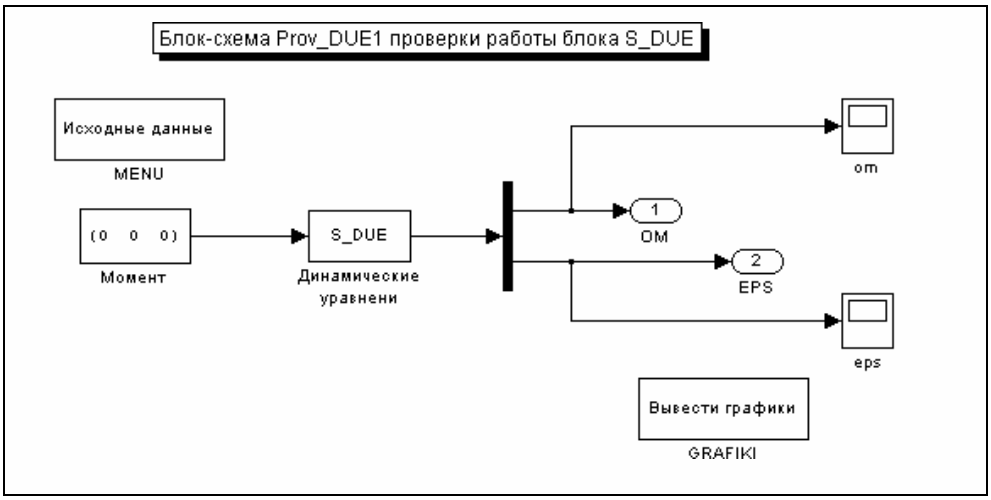


Рис. 8. 19. Блок-схема S-модели Prov_DUE1

Произведем в этих блоках следующие изменения:

1) изменим названия блоков (подписи под их изображениями); под первым блоком запишем MENU, под вторым GRAFIKI;

2) дважды щелкнув на каждом из блоков, откроем их блок-схемы и уничтожим все их содержимое, - сделаем блоки пустыми;

3) выйдя вновь в окно блок-схемы, щелкнем на изображении блоков правой клавишей мыши; при этом появится список команд, из которых следует выбрать команду Mask Subsystem; при этом появится окно Mask Editor (рис. 8. 20);

4) введем в поле Drawing commands вкладки Icon этого окна команду **`disp`** с указанием в качестве ее аргумента текста, который мы желаем поместить на изображении блока; для первого блока запишем *Исходные данные*, для второго – *Вывести графики*;

5) закроем окно Mask Editor, нажав в нем кнопку OK.

В результате получим изображения блоков, представленные на рис. 8. 19.

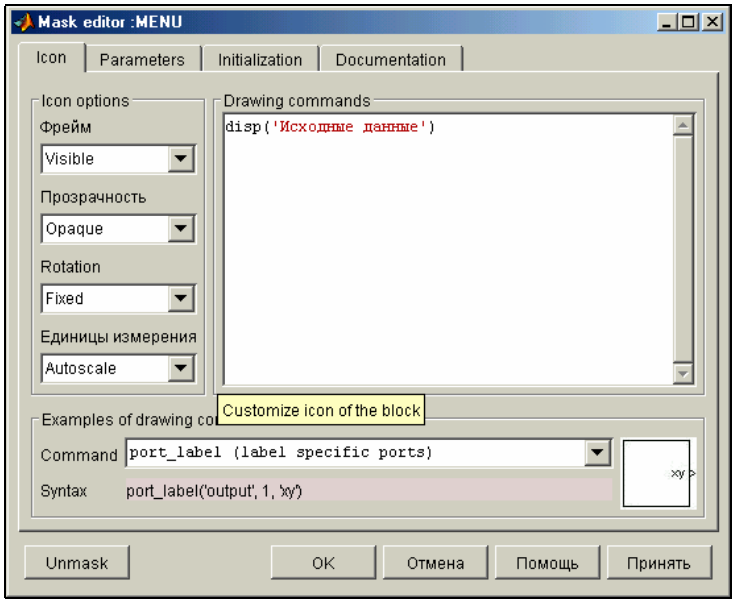


Рис. 8. 20. Окно Mask Editor

Введем в командном окне команды, связывающие составленные M-программы с S-моделью:

```
set_param('Prov_DUE','PreLoadFcn','Prov_DUE_Pred')
set_param('Prov_DUE/MENU','OpenFcn','MENU_DUE')
set_param('Prov_DUE/GRAFIKI','OpenFcn','Graf_DUE')
```

После выполнения этих команд управление всеми действиями по моделированию будет осуществляться из окна самой S-модели. Назовем эту модификацию S-модели **Prov_DUE1**.

Теперь моделирование можно производить в следующем порядке:

1) вызвать на экран окно блок-схемы **Prov_DUE1**; при этом начальные значения исходных данных уже будут записаны в рабочем пространстве, так как перед появлением окна блок-схемы будет запущена на выполнение программа **Prov_DUE_Pred**;

2) дважды щелкнуть в окне блок-схемы на блоке **MENU** (Исходные данные); появится окно меню (рис. 8. 21);

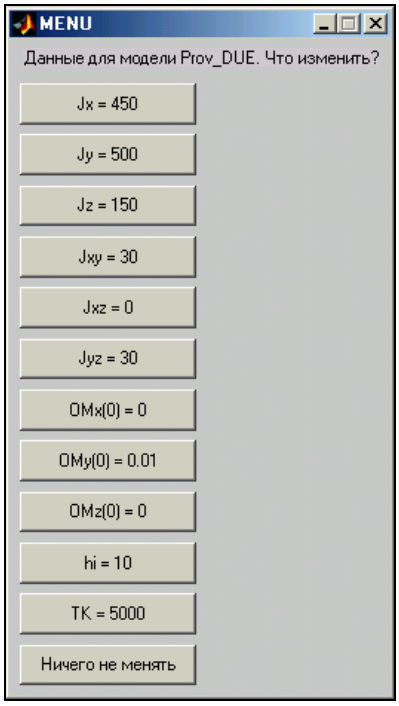


Рис. 8. 21. Меню пользователя программы MENU_DUE

3) пользуясь меню, установить нужные значения исходных параметров и закончить работу с меню, нажав кнопку «Ничего не менять»;

4) запустить S-модель на моделирование, нажав значок «►» в линейке инструментов окна блок-схемы;

5) по окончании процесса моделирования для вывода графиков в графическое окно дважды щелкнуть на блоке GRAFIKI (Вывести графики).

Осуществление связи S-модели с известными 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.

На рис. 8. 22…8.24 представлены результаты моделирования движения тела при разных сочетаниях параметров. Во всех случаях тело «раскручено» вокруг оси Y с угловой скоростью $\omega_Y = 0{,}01$ радиан в секунду. Первый случай соответствует динамически несимметричному телу ($J_X = 550$, $J_Z = 150$), имеющему начальную скорость вокруг оси X, равную $\omega_X = 0{,}001$ радиан в секунду.

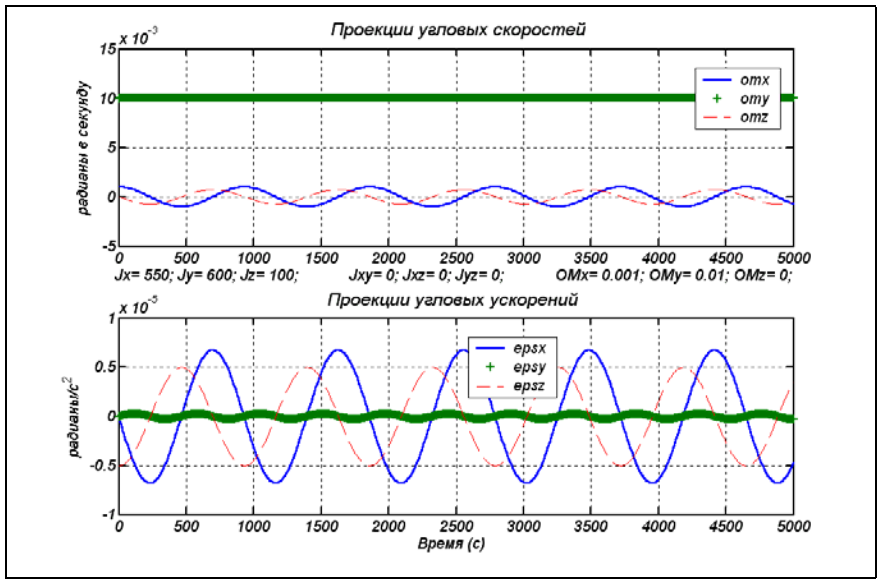


Рис. 8. 22. Нутационные колебания несимметричного гироскопа

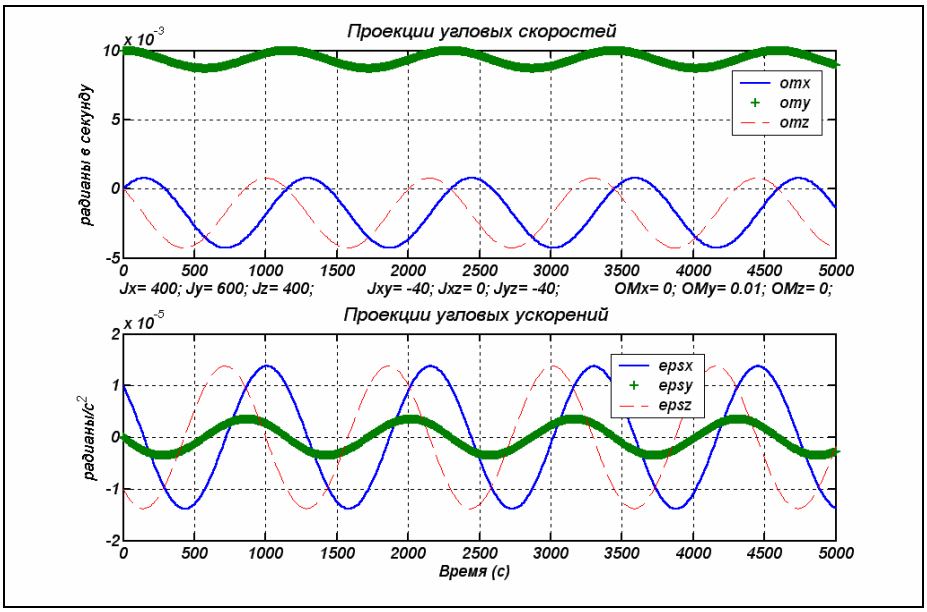


Рис. 8. 23. Свободное движение динамически несбалансированного гироскопа

Во втором случае тело является динамически несбалансированным относительно принятых осей и вращается в начальный момент времени только вокруг оси Y.

Третий вариант соответствует случаю, когда тело и динамически несбалансировано, и осевые моменты инерции значительно различаются по величине.

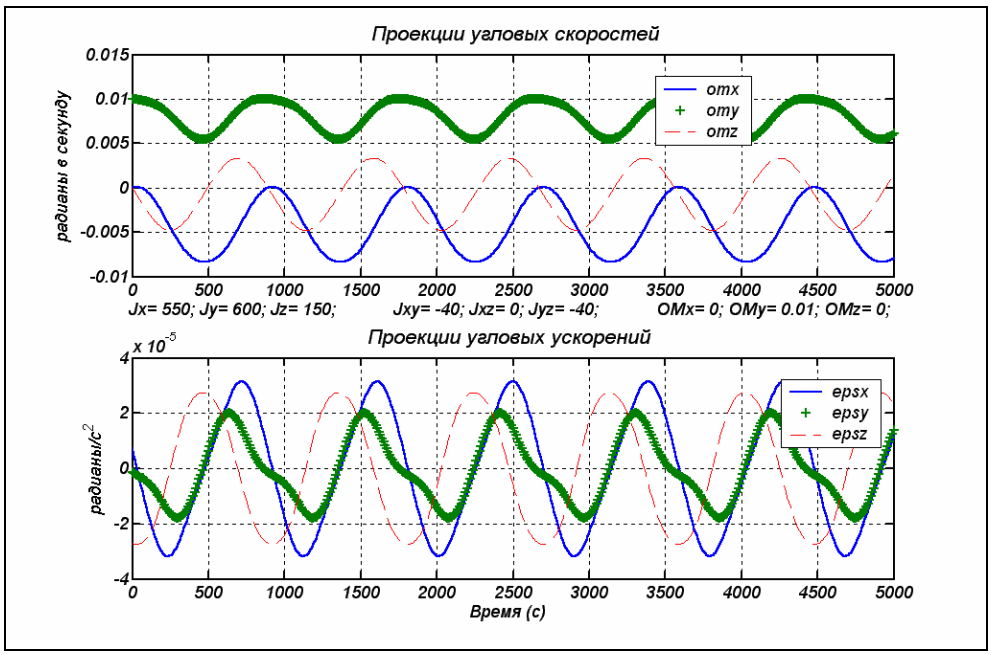


Рис. 8. 24. Свободное движение несимметричного и несбалансированного гироскопа

Как можно убедиться по этим экспериментам, вращательное движение тела существенно зависит от его инерционных характеристик.

8.2. 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**.

Создание окон настраивания блоков осуществляется через так называемую *маскировку* блока, т. е. создание *маски блока*.

8.2.1. Создание библиотеки

Рассмотрим процесс создания новой собственной библиотеки S-блоков на конкретных примерах.

Образование новой библиотеки начинается с вызова на экран окна новой блок-схемы модели. В этом новом окне следует вызвать File ► New ► Library. В результате этих действий на экране возникнет пустое окно создаваемой библиотеки (рис. 7.108) с именем Library: untitled1. Теперь в нем можно создавать, или перетягивать в него созданные S-блоки.

В общем случае образовать S-блок можно на основе двух видов стандартных блоков:

- блока **`S-Function`** из раздела **User-Defined Function** библиотеки **SIMULINK**;
- блока **`SubSystem`** из раздела **Ports & Systems** той же библиотеки.

Блок на основе **`S-Function`** получается на основе написанных на языке MatLab файлов S-функций, и имеет лишь один вход (возможно, векторный) и один выход (тоже векторный). Блок на основе **`SubSystem`** представляет собой блок-схему из существующих блоков и может иметь произвольное количество входов и выходов произвольного вида.

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 & Table** библиотеки **SIMULINK**. Окно библиотеки приобретет вид, приведенный на рис. 8. 25.

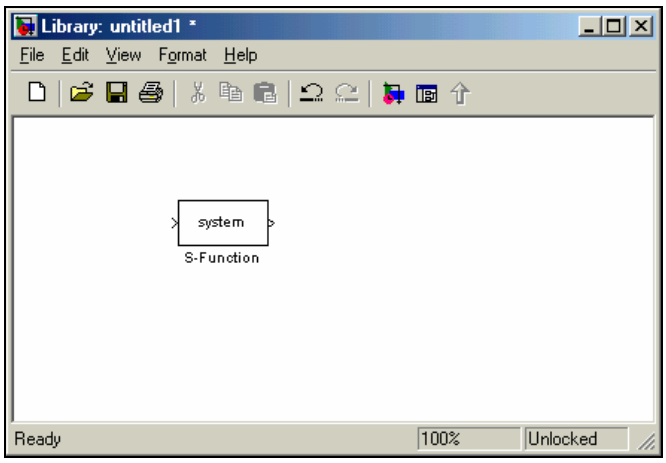


Рис. 8. 25. Окно создаваемой библиотеки

Теперь, дважды щелкнув на изображении этого блока, вызовем его окно настраивания (рис. 8. 26). В окошко *S-Function name* введем имени S-функции (***S_DUE***), а в окошко *S-Function parameters* – ее параметры (*J, UgSk0*) (рис. 8. 26) и щелкнем на кнопке OK внизу этого окна.

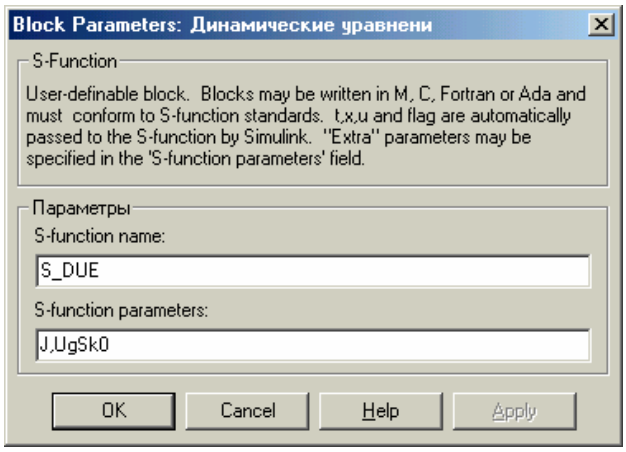


Рис. 8. 26. Окно настраивания блока S_DUE

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. 8. 27).

Рис. 8. 27. Внешний вид блока S_DUE

В завершение изменим название под блоком на такое "Дин. Уравн. 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. Если в качестве такого параметра принять 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 отличаются от действий над матрицами и не предусмотрены в системе MatLab. Значительно удобнее превратить его в совокупность сугубо матричных уравнений:

$$\begin{cases} \dfrac{dq_0}{dt} = -\dfrac{1}{2}\mathbf{q}^t \cdot (\boldsymbol{\omega} - \boldsymbol{\Omega}) \\ \dfrac{d\mathbf{q}}{dt} = \dfrac{1}{2}[q_0 \cdot (\boldsymbol{\omega} - \boldsymbol{\Omega}) + (\mathbf{q}\times) \cdot (\boldsymbol{\omega} + \boldsymbol{\Omega})] \end{cases} \tag{8.5}$$

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 кватерниона 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, жестко связанной с ним
 % ВЫХОД блока:
 %     y=[qw0,qw1,qw2,qw3] - вектор компонентов кватерниона поворота
 %     относительно орбитальной декартовоі системы координат
 % Входные ПАРАМЕТРЫ S-функции:
 %            OM0 - орбитальная угловая скорость;
 %            Qw0 - вектор начального кватерниона поворота

 % Лазарев Ю.ф. Україна 18-12-2001
```

```
switch flag,
  case 0
     [sys,x0,str,ts] = mdlInitializeSizes(Qw0);
  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(Qw0)
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" - вектор 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; ;
% Формирование 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 S-блок под именем **`S_Kuquat`**. 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. 8. 28.

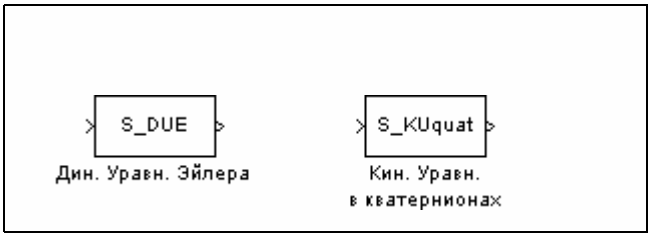


Рис. 8. 28. Новый вид библиотеки LAZlibrary

Примечание. Если блок-схема библиотеки была закрыта, изменение ее после повторного вызова возможно только после того, как вызвана команда в окне этой библиотеки Edit ► Update diagram.

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 **Ports & Subsystems**. Перетянем его мышью в окно новой библиотеки (рис. 8. 29). Дважды щелкнув на изображении этого блока, получим пустое окно, в котором составим блок-схему подсистемы, приведенную на рис. 8. 30.

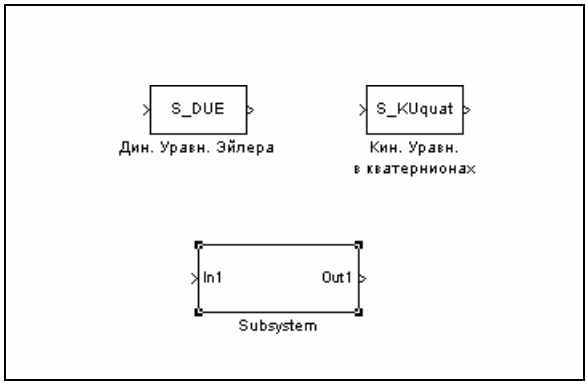


Рис. 8. 29. Включение в библиотеку блока Subsystem

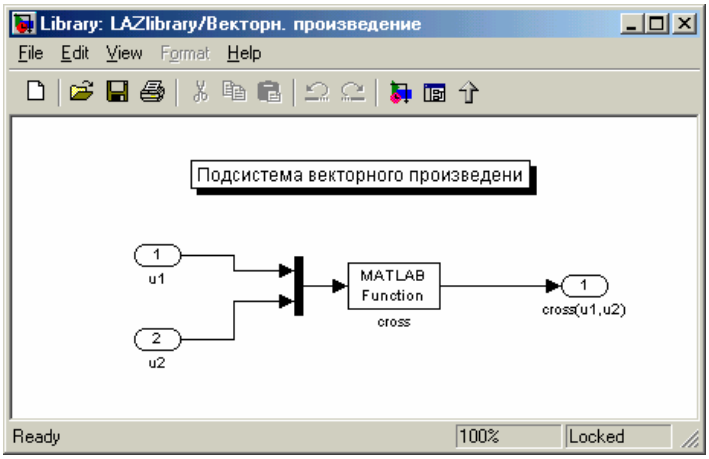


Рис. 8. 30. Блок-схема подсистемы Векторное произведение

В ней использован блок **MATLAB Function** из раздела **User-Defined Functions**, окно настраивания которого представлено на рис. 8. 31. Именно он, собственно, и выполняет операцию векторного умножения двух входных векторов, используя для этого стандартную функцию **cross** системы MatLab.

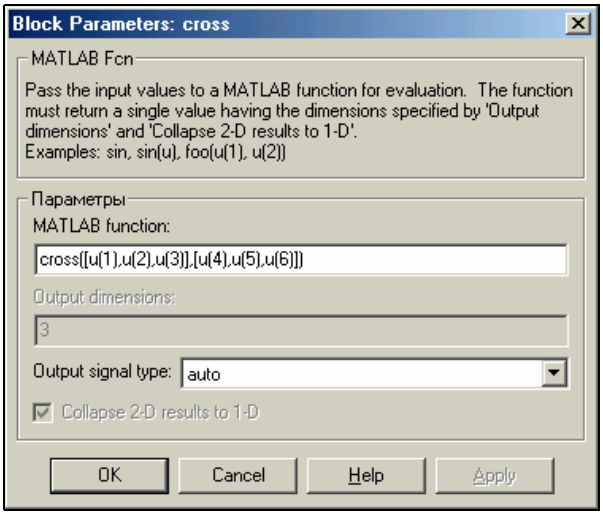


Рис. 8. 31. Окно настраивания блока Cross (Matlab Function)

В итоге всего этого получим новую библиотеку из трех новых собственных S-блоков. Запишем ее как **LAZlibrary** (рис. 8. 32).

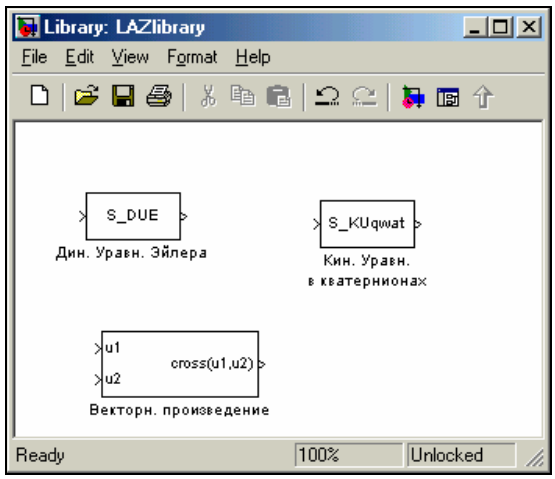


Рис. 8. 32. Библиотека пользователя LAZlibrary

Примечание. В русифицированной версии MatLab 6.5 появление в тексте документов или имени блоков русской буквы «я» приводит к необратимым нарушениям работы блока. *При составлении русского текстового оформления блока избегайте употребления буквы «я».*

8.2.2. 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 следует выполнить команду Edit ► Mask S-Function окна библиотеки, в которой расположен выделенный блок. На экране возникнет окно Mask editor редактора маски, представленное на рис. 8. 33.

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.

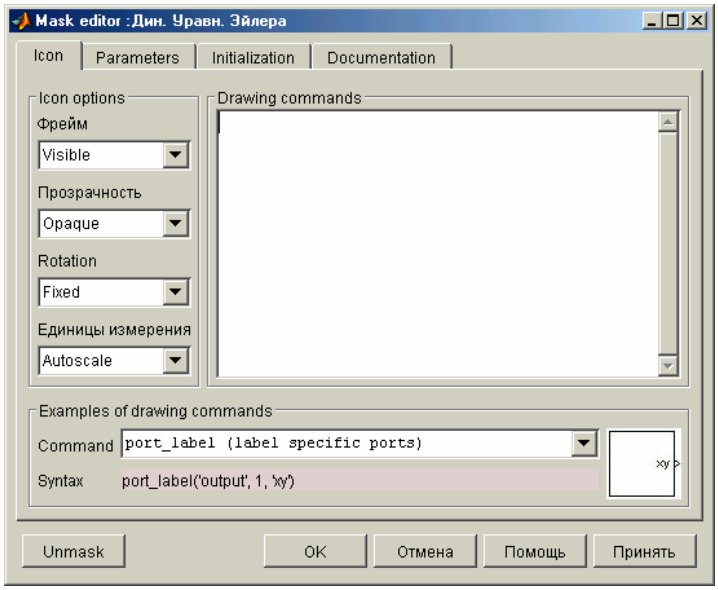


Рис. 8. 33. Окно редактора маски (Mask editor)

Окно Mask Editor (рис. 8. 33) имеет четыре вкладки:

Icon для создания и редактирования изображений на изображении блока;

Parameters для создания и редактирования диалоговой части (ввода параметров) окна настраивания;

Initialization для введения некоторых команд среды MatLab при инициализации блока;

Documentation для оформления и редактирования текстовой части окна настраивания блока и справочной части маски.

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 окошки ввода параметров блока и надписи над этими окошками, которые объясняют смысл 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.

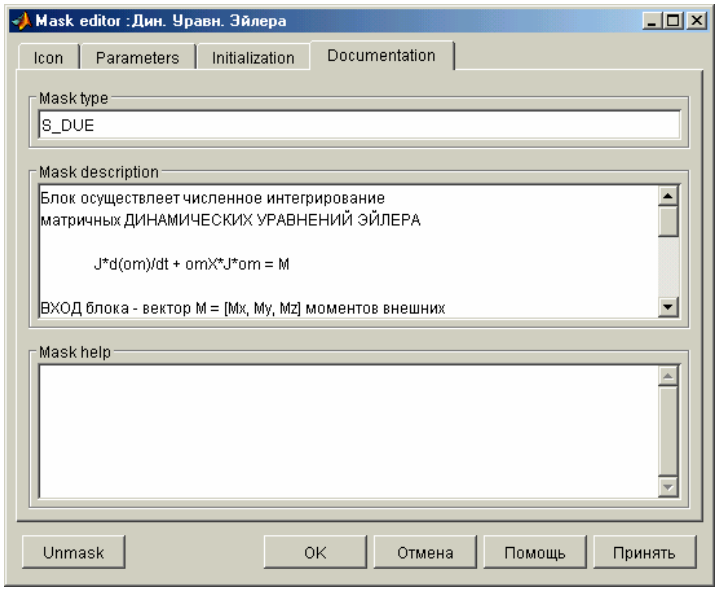


Рис. 8. 34. Заполнение вкладки Documentation окна Mask editor

Перейдем (с помощью мыши) к вкладке *Documentation* (рис. 8. 34). В окно *Mask type* следует ввести имя блока. В окно *Mask description* записывается информация, которую бы вы желали иметь в верхней (справочной) части окна настраивания блока, а в окне *Mask Help* записывается дополнительная справочная информация, которая вызовется, если в окне маски нажать кнопку *Help*.

Примечание. Грамматические ошибки в тексте документации маски вызваны тем, что в русифицированной версии MatLab 6.5 появление в тексте документов или имени маски и блоков русской буквы «я» часто приводит к необратимым нарушениям работы маски или блока. *При составлении русского текстового оформления маски избегайте употребления буквы «я».*

Перейдем ко вкладке *Parameters* (рис. 8. 35), можно сделать вывод, что с ее помощью можно сконструировать вторую, важнейшую, часть маски, - ее диалоговую часть.

Как видим, основное пространство вкладки занимает окно (пока неактивное) *Dialog Parameters* (Параметры диалога). В нем вводятся параметры, определяющие количество окошек ввода на маске, надписи над ними и имена переменных, под которыми они будут фигурировать в блоке. Слева от этого окна (вверху) находится единственная активная кнопка с изображением стрелки. Это кнопка Add (Присвоить). Нажатие на нее приводит к активизации окна, - в нем появляется активная строка, в которую следует ввести диалоговые параметры.

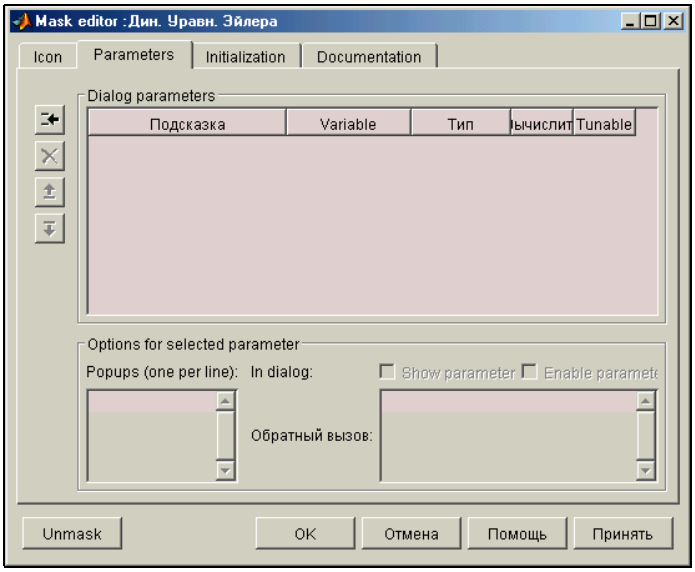


Рис. 8. 35. Вкладка Parameters окна Mask editor

В блоке **S_DUE** всего два параметра, значения которых нужно вводить в диалоговой форме – матрица моментов инерции тела *J* размером (3*3) и вектор *UgSk0* начальных значений трех проекций угловой скорости тела. Поэтому нужно создать два окошка ввода значений этих параметров и создать надписи на них.

Введение очередного окошка в маску осуществляется нажатием мышью кнопки *Add* по левую сторону от наибольшего окна вкладки, где отображаются результаты редактирования. Сама запись этой надписи осуществляется в окне *Dialog Parameters* (рис. 8. 36).

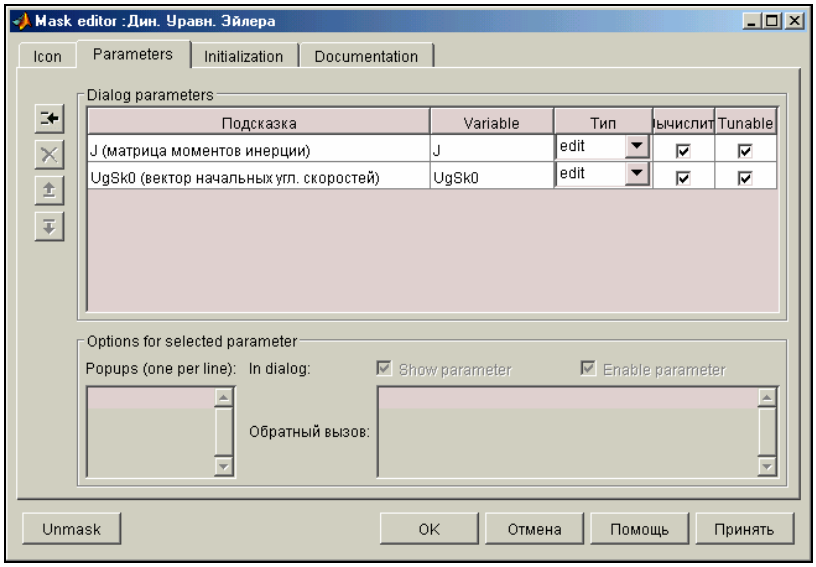


Рис. 8. 36. Ввод диалоговых параметров маски блока S_DUE

При этом в колонку *Подсказка* записывается надпись над будущим окошком маски, а в колонку *Variable* – имя, под которым введенная величина будет фигурировать внутри блока. Результат ввода этих параметров для блока **S_DUE** представлен на рис. 8. 36.

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 окно маски блока, изображенное на рис. 8. 37. Маска блока S_DUE создана.

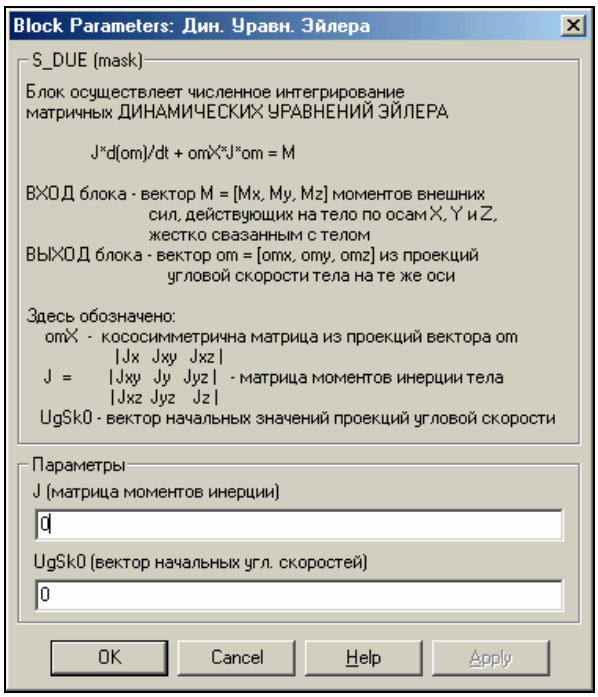


Рис. 8. 37. Окно настраивания блока S_DUE

Аналогично получается маска блока **`S_KUqwat`**, представленная на рис. 8. 38.

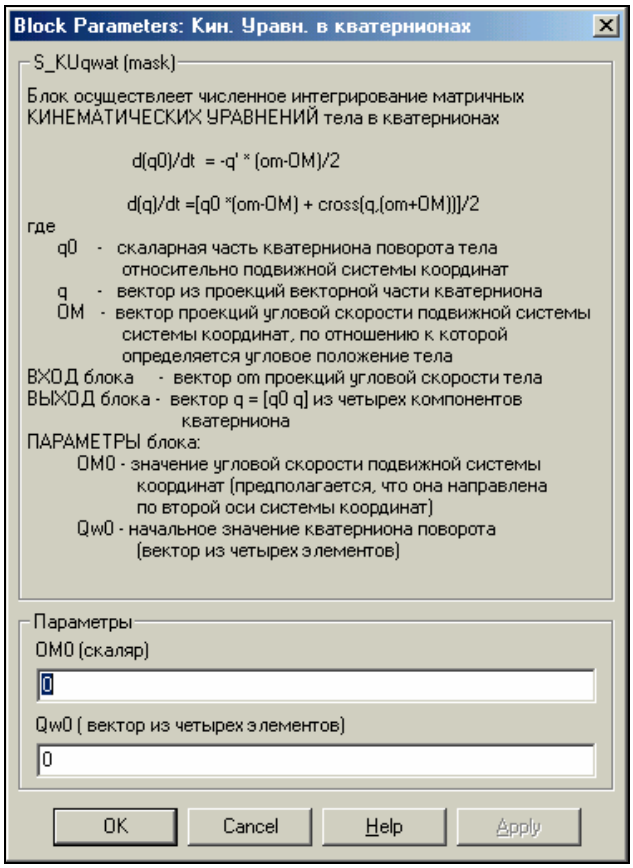


Рис. 8. 38. Окно настраивания блока S_KUqwat

8.3. Примеры использования библиотеки пользователя

8.3.1. Моделирование процесса ориентации космического аппарата

В качестве примера применения блоков собственной библиотеки рассмотрим процесс образования S-модели и комплекса M-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 ориентацией (т. е. угловым положением относительно орбитальной системы координат) осуществляется с помощью трех маховичних двигателей, оси которых совпадают с осями декартовоï системы координат, жестко связанной с корпусом космического аппарата (КА). Маховичные двигатели, с одной стороны, осуществляют разгон соответствующего ротора в соответствии с уравнениями

$$\frac{dH_k}{dt} = M_k, \qquad (k = x, y, z\ ); \tag{8.6}$$

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 ***SUO_KA.mdl***. Она приведена на рис. 8. 39.

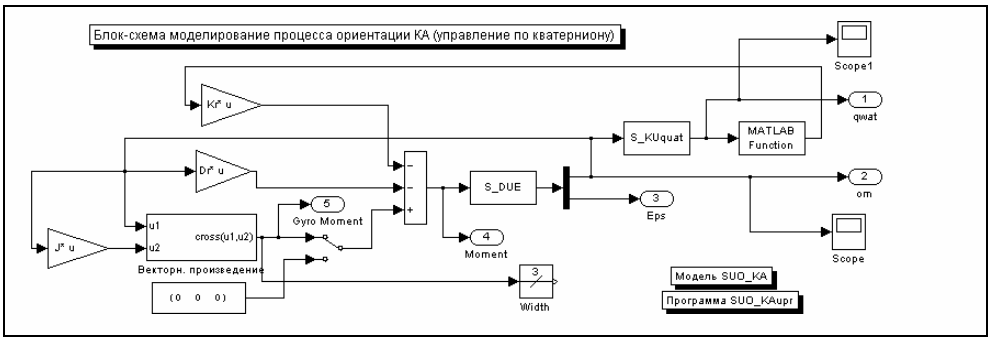


Рис. 8. 39. Блок-схема системы управления ориентацией космического аппарата

Блок-схема системы ориентации состоит из последовательно соединенных блоков **`S_DUE`** и **`S_KUquat`**, охваченных тремя цепями обратной связи, обеспечивающими управление космическим аппаратом по кватерниону, угловой скорости и компенсацию гироскопического момента, возникающего при поворотах КА. Полученное на выходе блока **`S_KUquat`** значение кватерниона поворота поступает 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 матричному закону:

$$\mathbf{M} = -K_r \cdot \mathbf{q} - D_r \cdot \boldsymbol{\omega} + (\boldsymbol{\omega}\times) \cdot (J \cdot \boldsymbol{\omega})\,. \tag{8.7}$$

В нем можно различить три составляющие:

- составляющую, пропорциональную векторной части кватерниону отклонения текущего положения КА от заданного его положения (в этой программе заданное 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. 8. 39).

Эти три образованные составляющие суммируются на сумматоре и подаются на вход блока **S_DUE**. Так образуется 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:

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
```

2) матрица управления пропорциональна единичной матрице E **Kr=k*E=diag([110,110,110]) ;**

```
Kr =
   110     0     0
     0   110     0
     0     0   110
```

3) матрица управления представляет собой комбинацию

```
  Kr=(αJ+βE) =diag([72,110,204]);
Kr =
    72     0     0
     0   110     0
     0     0   204
```

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 ***SUO_KAupr.m***. Текст этого файла приведен ниже

```
% SUO_KAupr.m
 % Управляющая программа  для запуска модели SUO_KA.mdl

 % Лазарев Ю.Ф.  18-12-2001
 % Последние изменения 2-2-2004
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
Qw0=[0.159,0.57,0.57,0.57];
qw0=Qw0(1);  qw=Qw0(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('SUO_KA',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=[0;diff(t1)];
dHx1=Mx1.*dt1;      dHy1=My1.*dt1; dHz1=Mz1.*dt1;
domx1=[0;diff(omx1)]; domy1=[0;diff(omy1)]; domz1=[0;diff(omz1)];
DEx1=cumsum(abs(dHx1.*domx1));
DEy1=cumsum(abs(dHy1.*domy1));
DEz1=cumsum(abs(dHz1.*domz1));
d1=DEx1+DEy1+DEz1;
dt2=[0;diff(t2)];
```

```
dHx2=Mx2.*dt2;      dHy2=My2.*dt2; dHz2=Mz2.*dt2;
domx2=[0;diff(omx2)]; domy2=[0;diff(omy2)]; domz2=[0;diff(omz2)];
DEx2=cumsum(abs(dHx2.*domx2));
DEy2=cumsum(abs(dHy2.*domy2));
DEz2=cumsum(abs(dHz2.*domz2));
d2=DEx2+DEy2+DEz2;
dt3=[0;diff(t3)];
dHx3=Mx3.*dt3;      dHy3=My3.*dt3; dHz3=Mz3.*dt3;
domx3=[0;diff(omx3)]; domy3=[0;diff(omy3)]; domz3=[0;diff(omz3)];
DEx3=cumsum(abs(dHx3.*domx3));
DEy3=cumsum(abs(dHy3.*domy3));
DEz3=cumsum(abs(dHz3.*domz3));
d3=DEx3+DEy3+DEz3;
dt4=[0;diff(t4)];
dHx4=Mx4.*dt4;      dHy4=My4.*dt4; dHz4=Mz4.*dt4;
domx4=[0;diff(omx4)]; domy4=[0;diff(omy4)]; domz4=[0;diff(omz4)];
DEx4=cumsum(abs(dHx4.*domx4));
DEy4=cumsum(abs(dHy4.*domy4));
DEz4=cumsum(abs(dHz4.*domz4));
d4=DEx4+DEy4+DEz4;
 % 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 кватерніону 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 Эйлера')
ylabel('Угол поворота в градусах')
xlabel('Время (c)')
legend('Kr = k/J','Kr = k','Kr = k/(\alpha+\beta)','Kr = kJ')
```

```
 % 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 (с)')
subplot(2,2,2)
plot(t1,My1,t2,My2,':',t3,My3,'--',t4,My4,'.'), grid
ylabel('My')
xlabel('Время (с)')
subplot(2,2,3)
plot(t1,Mz1,t2,Mz2,':',t3,Mz3,'--',t4,Mz4,'.'), grid
ylabel('Mz')
xlabel('Время (с)')
subplot(2,2,4)
plot(t1,d1,t2,d2,':',t3,d3,'--',t4,d4,'.'), grid
title('Суммарные затраты ЭНЕРГИИ')
ylabel('\Sigma\Delta H\Delta\omega')
xlabel('Время (с)')
legend('Kr = k/J','Kr = k','Kr = k/(\alpha+\beta)','Kr = kJ',0)
```

Запуск этого М-файла приводит к результатам, представленным на рис. 8. 40…8. 42.

На рис. 8. 40 приведены проекции траекторий кватерниона на все три координатные плоскости, а также траектории в пространстве вектора эйлерового поворота.

На рис. 8. 41 показаны графики зависимостей от времени компонентов кватернионов, а также проекций вектора эйлерова поворота.

Рис 8. 42 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.

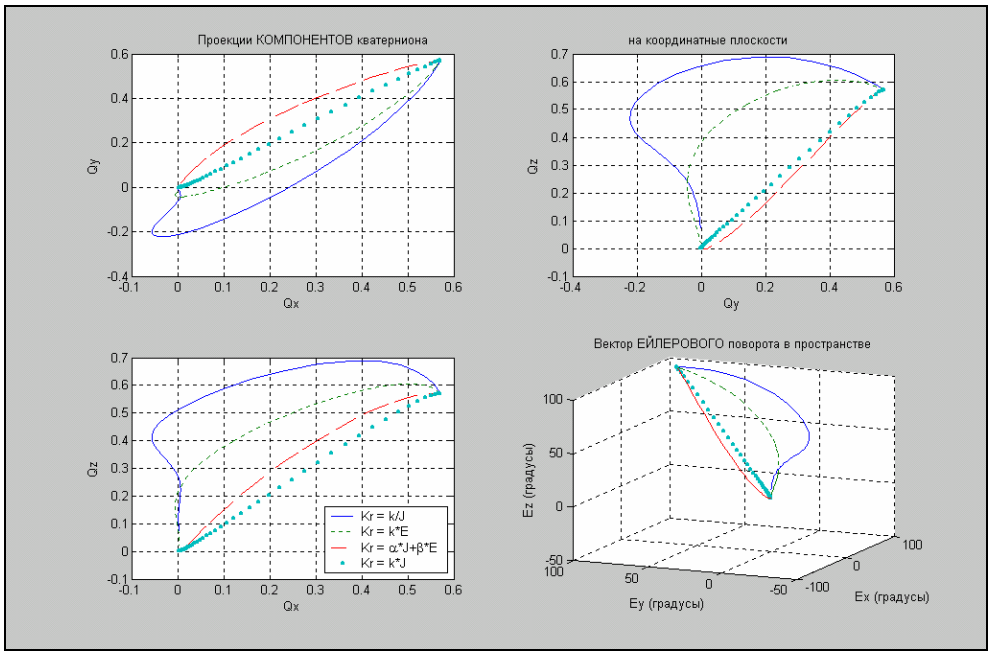


Рис. 8. 40. Проекции кватерниона поворота КА на координатные плоскости

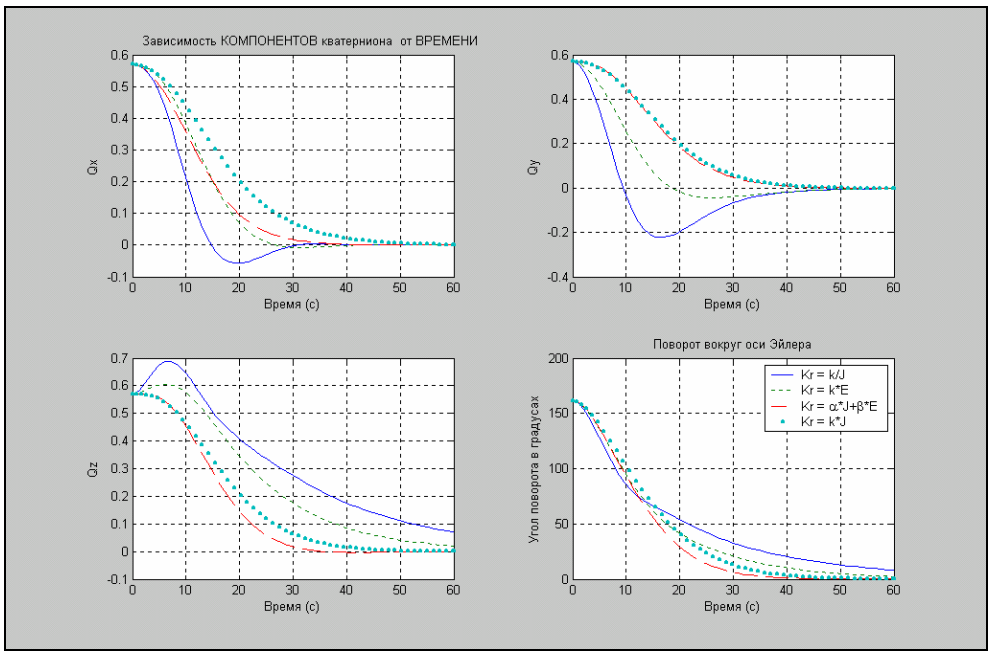


Рис. 8. 41. Зависимость компонентов кватерниона поворота КА от времени

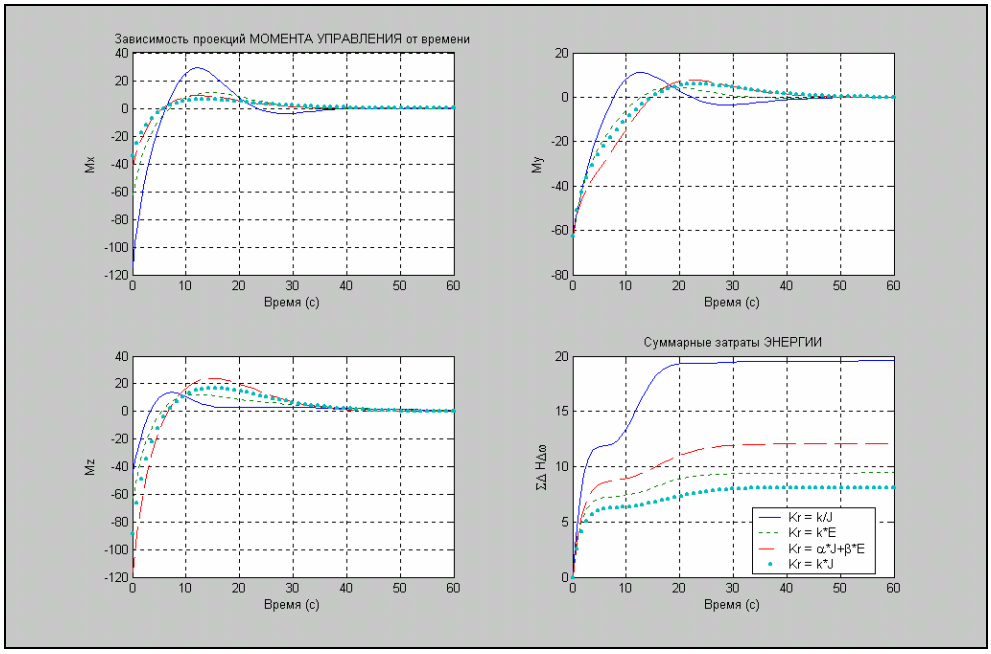


Рис. 8. 42. Зависимость компонентов момента управления ориентацией КА от времени

Рассматривая полученные графики, можно сделать вывод, что наименьшие затраты энергии дает управление по закону, когда матрица позиционного управления пропорциональна матрице моментов инерции КА. Этот закон позволяет сэкономить более чем в два раза по сравнению со случаем, когда матрица коэффициентов позиционного управления обратно пропорциональна матрице моментов инерции.

К такому же выводу можно прийти и сугубо теоретическим путем, если подставить выражение (8.7) момента управления у уравнение (8.1) 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.

8.3.2. Моделирование системы с сухим трением

Всем хорошо известны основные свойства силы трения, возникающие при относительном перемещении двух трущихся друг о друга тел:

- сила трения всегда направлена в сторону, противоположную относительной скорости этих тел;

- величина силы трения не зависит от величины этой относительной скорости, оставаясь одной и той же, как бы не изменялась эта скорость.

Эти свойства достаточно хорошо описываются математически, если использовать известную разрывную элементарную функцию `SIGN`:

$$F_{TR} = -F_T \cdot sign(V),$$

где F_T - некоторый положительный коэффициент, равный величине силы сухого трения, а V - относительная скорость взаимного перемещения трущихся тел.

Однако нам известно еще одно свойство сухого трения:

- если трущиеся неподвижны друг относительно друга, то приложение внешней силы к одному из них не вызовет относительного движения тел до тех пор, пока действующая сила (назовем ее «активной» - F_a) не превысит по величине так называемую силу трения покоя $F_p > 0$.

В этом случае сила трения уже определяется не величиной и направлением скорости, а величиной приложенной активной силы, принимая такое значение и направление, что она полностью компенсирует действие этой силы

$F_a + F_{TR} = 0$, если $V = 0$ и $|F_a| \le F_p$.

Эта особенность сухого трения является причиной целого ряда замечательных свойств систем с сухим трением, в частности, такого, как явление «захватывания», «сцепления» одного тела с другим, когда оба тела начинают двигаться как одно, оставаясь неподвижными друг относительно друга.

Теоретическое изучение этого свойства сил сухого трения наталкивается на значительные трудности, связанные с разрывным характером зависимости силы трения от скорости и указанной сложной зависимостью силы сухого трения от скорости и активной силы.

Сформулируем задачу описания движения механической системы с сухим трением.

Введем в рассмотрение обобщенную координату q (это может быть линейное перемещение или угол при вращательном движении), соответствующую относительному перемещению трущихся тел. Составим обобщенное дифференциальное уравнение по этой координате и разделим все члены этого уравнения на три части:

1) обобщенную силу инерции, в которую включим лишь члены, пропорциональные относительному обобщенному ускорению; поэтому ее можно представить в виде ($-M_q \cdot \ddot{q}$), где M_q имеет смысл обобщенной массы и может зависеть от обобщенной координаты q;

2) обобщенную силу трения $Q_{TR}(\dot{q})$, к которой отнесем все члены уравнения, определяющие влияние сухого трения;

3) «активную» обобщенную силу Q_a, в которую включим все остальные члены уравнения.

Тогда уравнение движения по этой координате можно представить в виде:

$$\ddot{q} = \frac{1}{M_q}(Q_a + Q_{TR}(\dot{q})). \qquad (8.8)$$

Только после такой операции возможно сформулировать математическую зависимость обобщенной силы сухого трения от всех факторов, влияющих на нее в соответствии с установленными свойствами трения:

$$Q_{TR}(\dot{q}) = \begin{cases} -M_{TD}, & если \quad \dot{q} > 0 \\ M_{TD}, & если \quad \dot{q} < 0 \\ -Q_a, & если \quad \dot{q} = 0 \; и \; |Q_a| < M_{TP} \\ M_{TP}, & если \; \dot{q} = 0 \; и \; |Q_a| \geq M_{TP} \end{cases}, \tag{8.9}$$

где положительные постоянные величины M_{TD} и M_{TP} определяют величины обобщенной силы трения движения и покоя соответственно. При этом обычно выполняется соотношение $M_{TP} \geq M_{TD}$.

Как видим, описать полностью все указанные особенности силы сухого трения можно только после того, как заданы (известны) уравнения движения и выделена так называемая активная сила.

Создадим универсальный блок, осуществляющий однократное интегрирование уравнения (8). Входом этого блока должно быть текущее значение активной силы, выходом – текущее значение обобщенной скорости $\dot{q}$.

Параметрами, определяющими процесс интегрирования, являются:

В формуле	В программе	Примечание
M_q	Mq	обобщенная масса
M_{TD}	TrDvig	величина трения движения
M_{TP}	TrPoc	величина трения покоя
$\dot{q}_0$	qt0	начальное значение обобщенной относительной скорости

Оформим блок в виде подсистемы, изображенной на рис. 8. 43 и назовем его **`Suhoe Trenye`**.

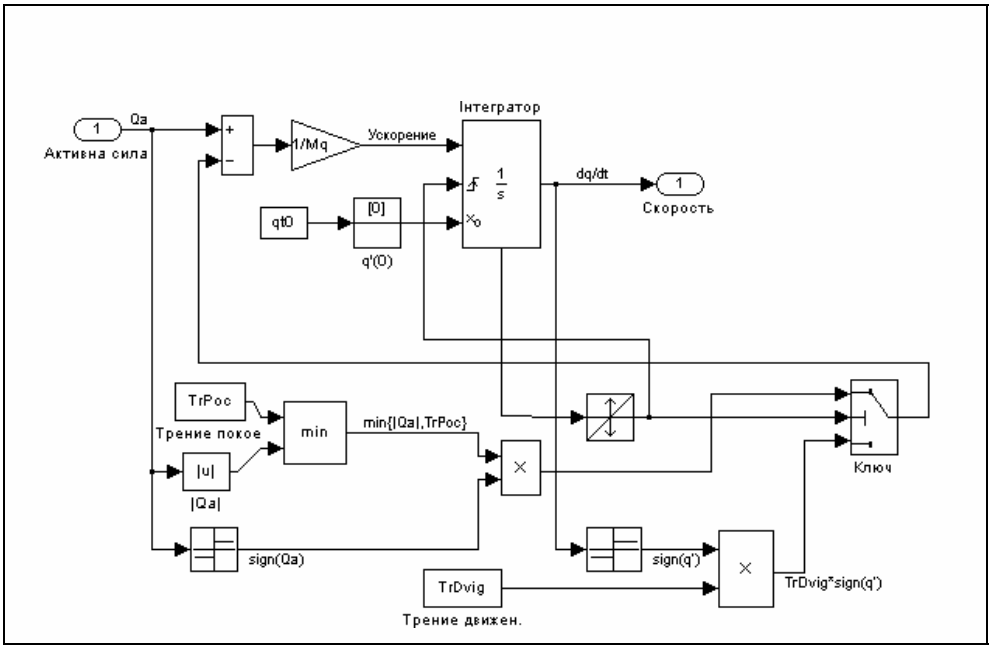


Рис. 8. 43. Блок-схема блока Suhoe Trenye

Основной элемент блока – интегратор. Он осуществляет интегрирование подаваемого на него сигнала относительного ускорения, выдавая сигнал, равный текущему значению обобщенной скорости. Начальное значение обобщенной скорости задается внешним блоком **`IC`**, на который подается постоянный сигнал с блока **`Constant`**, равный начальному значению обобщенной скорости.

Сигнал обобщенного ускорения формируется на сумматоре (блок **`Sum`**), где суммируются активная сила с силой трения, после прохождения через блок Gain, который производит деление суммарного сигнала на обобщенную массу.

Формирование силы трения осуществлено в нижней части блок-схемы.

Если скорость $\dot{q}$ не равна нулю то переключатель в блоке **Ключ** находится в нижнем положении, и сигнал обобщенной скорости проходит через блок **Sign** (нижняя правая часть схемы), умножается на постоянный коэффициент *TrDvig* и передается на сумматор как сила трения с обратным знаком. Этим реализуются первые два соотношения (9).

Значительно сложнее реализовать последние два условия в (9). Для этого, прежде всего, следует как можно точнее определить момент времени, когда относительная скорость проходит через нуль. Это достигается тремя средствами:

1) в блоке интегратора следует открыть порт состояния *Show state port*; при этом на изображении блока внизу появится дополнительный выход – порт состояния; кроме того, следует подключить внешнее управление работой интегратора (параметр *External reset*), установив его значение *Rising*; при этом с левой стороны блока появится изображение еще одного порта (управления);

2) к порту состояния надо подключить вход блока **Hit Crossing**, который осуществляет фиксацию точного момента времени перехода скорости через нуль и выдает в этот момент времени управляющий единичный сигнал;

3) к выходу блока **Hit Crossing** подсоединяем управляющий вход блока **Switch** (второй вход); при этом в качестве порога (параметр *Threshold*) этого блока следует установить значение 0,5; кроме того выход блока **Hit Crossing** надо соединить с портом управления блока **Интегратор**.

Совокупность этих блоков работает следующим образом. Если скорость не пересекает нуля, выходной сигнал блока **Hit Crossing** равен нулю. Он меньше по величине порога блока **Switch** (0,5). Поэтому переключатель соединяет с выходом третий (нижний) вход и на сумматор подается сила трения движения. Как только блок **Hit Crossing** регистрирует пересечения скоростью нуля, на его выходе сигнал становится равным 1, он становится больше порога блока **Switch**, и переключает на сумматор ветвь блок-схемы, формирующую трение покоя (левая нижняя часть блок-схемы). Одновременно сигнал блока **Hit Crossing** поступает на управляющий вход блока **Интегратор**. По нему интегратор начинает интегрирование заново с момента пересечения скоростью нуля с начальным условием, установленным в блоке IC (в нашем случае $\dot{q}$ =0).

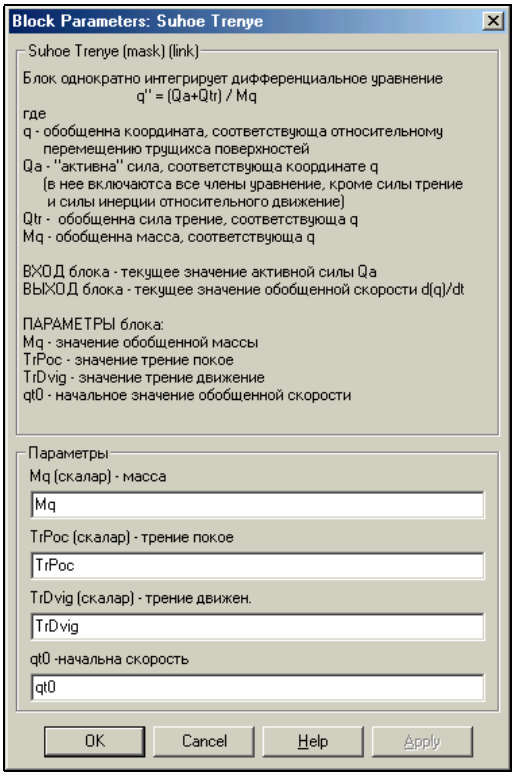


Рис. 8. 44. Окно настраивания блока Suhoe Trenye

Если при дальнейшем интегрировании $\dot{q}$ продолжает оставаться равной нулю, это состояние остается неизменным. Если же $\dot{q}$ на каком-то шаге интегрирования приобретет значение отличное от нуля, блок **Hit Crossing** сбросит значение своего выходного сигнала до нуля, ключ перебросит «рубильник» в нижнее положение и вновь «заработает» трение движения.

Ветвь, формирующая трение покоя, осуществляет следующие функции. Сначала определяется модуль активной силы. Затем он сравнивается со значением трения покоя. Определяется минимальная из этих двух положительных величин. Затем ей присваивается знак активной силы (блоки **Sign** и **Product**). Эта величина и составляет трение покоя и направляется на первый вход переключателя.

Сконструируем маску блока **Suhoe trenye** (рис. 8. 44).

Рассмотрим теперь задачу исследования движения физического маятника, на который действует момент сил сухого трения в опорах его оси вращения, в условиях вращения вокруг этой оси и самого основания.

Обозначим α - угол поворота маятника относительно основания. Тогда уравнение движения (вращения вокруг его оси) маятника можно записать в таком безразмерном виде:

$$\varphi'' + \sin\varphi = \mu_{tr}(\alpha'), \tag{8.10}$$

где, как и ранее φ - угол отклонения маятника от вертикали. Величина $\mu_{tr}(\alpha')$ представляет собой безразмерный момент сил трения, т. е. отношение момента трения к так называемому опорному маятниковому моменту mgl.

Обозначим угол поворота основания относительно вертикали вокруг оси вращения маятника через ϑ. Тогда три угла ϑ, α и φ связаны между собой соотношением

$$\alpha = \varphi - \vartheta. \tag{8.11}$$

Запишем уравнение (10) с учетом (11) в виде:

$$\alpha'' = \{-\vartheta'' - \sin(\alpha + \vartheta)\} + \mu_{tr}(\alpha'). \tag{8.12}$$

Координата α характеризует относительное перемещение (угловое) маятника и основания. Сравнивая (12) с (8) можно прийти к выводу, что в рассматриваемом случае

$$q = \alpha, \qquad M_q = 1, \qquad Q_a = -\vartheta'' - \sin(\alpha + \vartheta) = -\vartheta'' - \sin\varphi.$$

Блок-схема **TRENYE** S-модели, реализующей интегрирование уравнения (12), приведена на рис. 8. 45.

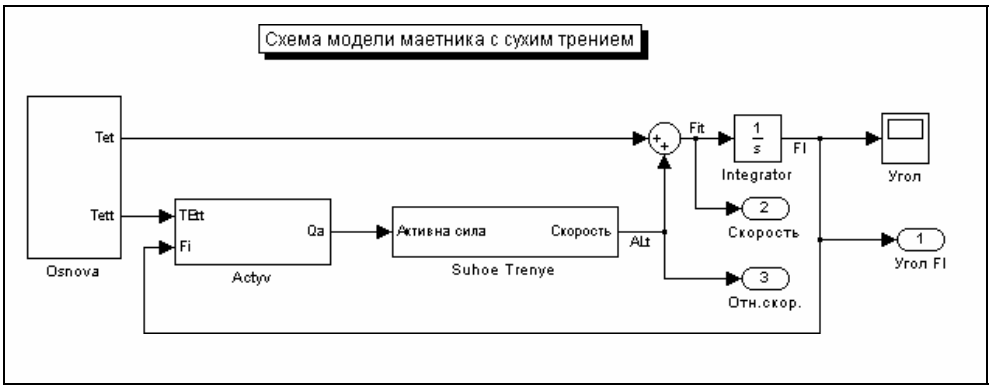


Рис. 8. 45. Блок-схема S-модели TRENYE

Блок Osnova в этой S-модели (рис. 8. 46) формирует сигналы угловой скорости (Tet) и углового ускорения (Tett) вращения основания.

Как видим, угловая скорость основания формируется по закону

$$\vartheta'(\tau) = \theta'_o + \theta'_m \sin(\omega_\vartheta \cdot \tau + \varepsilon_\vartheta).$$

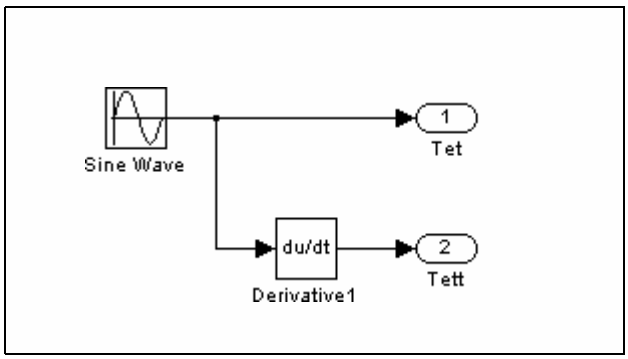


Рис. 8. 46. Блок-схема подсистемы Osnova

Значения используемых констант вводятся в окне настраивания блока Sine Wave (рис. 8. 47).

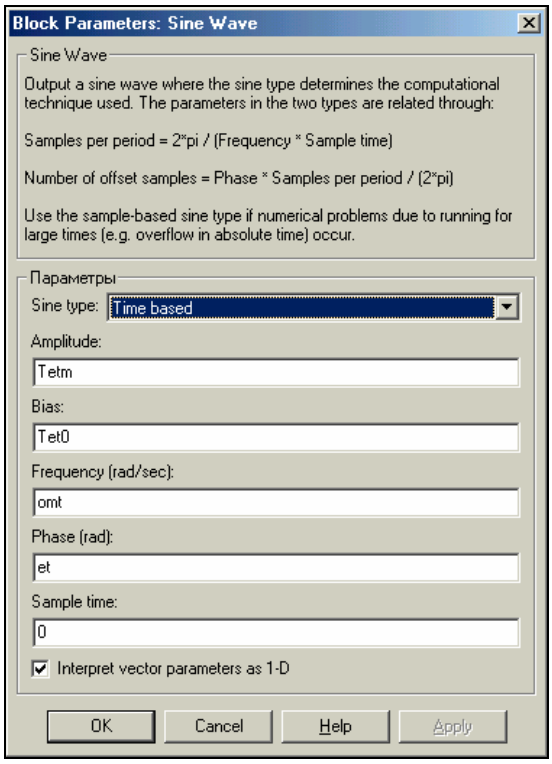


Рис. 8. 47. Окно настраивания блока Sine Wave

Ниже приводится таблица соответствия обозначений в формулах и программе.

В формуле	В программе	Примечание
θ'_o	Tet0	постоянная составляющая угловой скорости
θ'_m	Tetm	амплитуда угловой скорости
ω_ϑ	omt	частота изменения угловой скорости
ε_ϑ	et	начальная фаза угловой скорости

Блок-схема второй подсистемы Activ, формирующей сигнал активной силы, чрезвычайно проста и показана на рис. 8. 48.

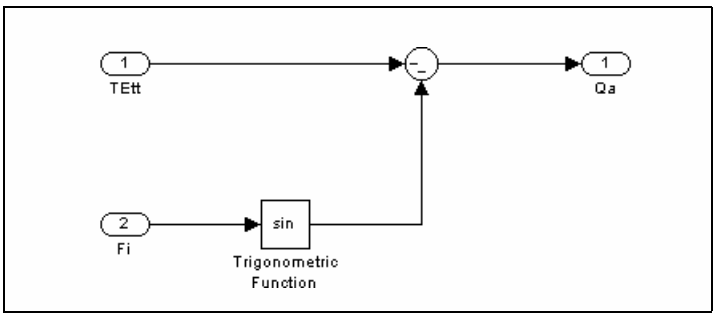


Рис. 8. 48. Блок-схема подсистемы Activ

Как и ранее, создадим управляющую программу TRENYE_upr. m, которая будет выполнять следующие функции:

1) ввода значений всех параметров, однозначно определяющих движение системы;

2) запуск S-модели TRENYE.mdl на моделирование;

3) выведение результатов моделирования в графическое окно MatLab.

Текст программы приведен ниже.

```
% Trenye_upr
% Управляющая программа  для запуска модели TRENYE.mdl
% Лазарев Ю.Ф.  5-2-2004
clear all, clc
% 1. Задание массы и характеристик трения
Mq=1;  TrPoc=0.2;   TrDvig=0.01;
% 2. Задание параметров вращения основания
%    Teta' = Tet0+Tetm*sin(omt*t+et)
Tetm=0;      Tet0=0;      omt=0;       et=0;
% 3. Задание начальных условий
fi0=30*pi/180;      fit0=0;
% 4. Расчет начальной относительной скорости
qt0=fit0-Tet0-Tetm*sin(et);
% 5. Запуск модели на моделирование
   sim('TRENYE');          % МОДЕЛИРОВАНИЕ на S-модели
% 6. Формирование данных для вывода ГРАФИКОВ
FI=yout(:,1)*180/pi;   FIt=yout(:,2);   ALt=yout(:,3);  t=tout;
% 7. Выведение графиков
subplot(2,2,1)
plot(FI,FIt,'.',FI,ALt), grid
set(gca,'fontsize',12)
xlabel('Угол (градусы)'), ylabel('Угловая скорость (б/р)')
legend('относит.','абсолютн.',0)
set(gca,'fontsize',14), title('Фазовый портрет')
subplot(2,2,[3 4])
plot(t,FI), grid
set(gca,'fontsize',12)
xlabel('Время (б/р)'),    ylabel('Угол (градусы)')
set(gca,'fontsize',14),   title('Угол отклонения от вертикали')
subplot(2,2,2)
axis('off')
h=text(0,1,'Маятник с сухим трением','fontsize',16);
h=text(-0.2,0.8,'Вращение основания:
Teta''(t)=Tet0+Tetm*sin(omt*t+et)','fontsize',12);
h=text(-0.2,0.7,['где:      ',...
        sprintf('Tet0 = %g; ',Tet0),sprintf('Tetm = %g;',Tetm),...
        sprintf('omt = %g;   ',omt),...
        sprintf('et =% g градусов',et*180/pi)]);
h=text(-0.2,0.5,'Характеристики трения','fontsize',12);
h=text(-0.1,0.4,[sprintf('Трение покоя = %g;    ',TrPoc),...
        sprintf('Трение движения = %g;   ',TrDvig)]);
h=text(-0.2,0.2,sprintf('Начальная абс. угл. скор. = %g;',fit0),...
                  'fontsize',12);
h=text(-0.2,0.0,'-----------------------------------------');
```

```
h=text(-0.2,-0.1,'Программа TRENYE-upr   5-02-2004  Лазарев Ю. Ф.');
h=text(-0.2,-0.2,'-------------------------------------------');
```

Запуская эту программу на исполнение, получим результат, показанный на рис. 8. 49.

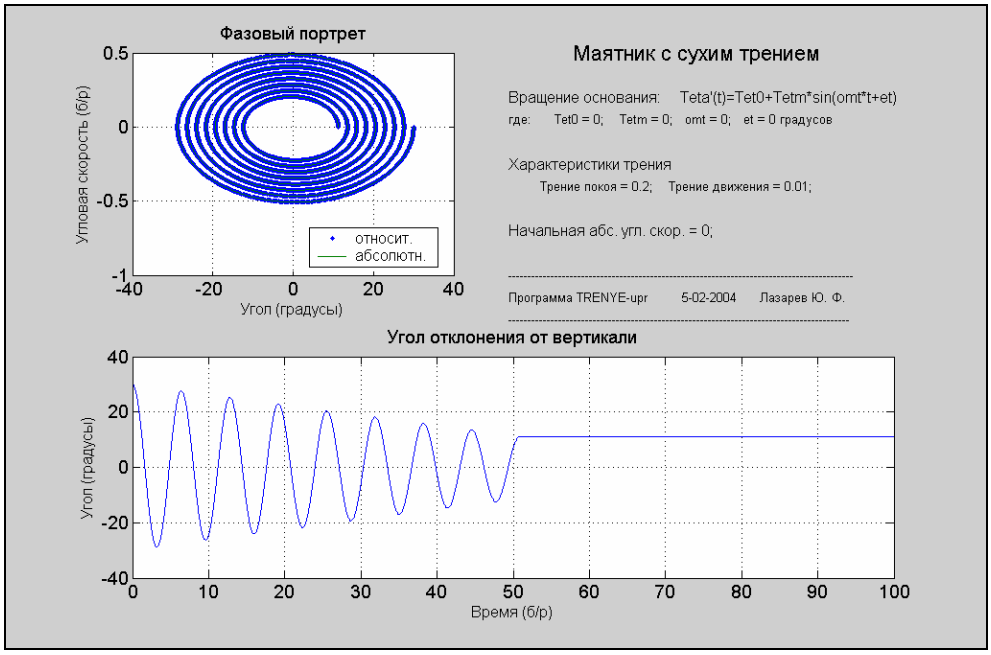


Рис. 8. 49. Свободные колебания маятника с сухим трением

Из рисунка становятся наглядными три основных нелинейных свойства маятника:

- огибающая свободных колебаний является прямой;
- колебания затухают за конечное время;
- маятник останавливается не в положении вертикали, а в смещенном относительно нее положении.

Следующие два рисунка (8. 50 и 8. 51) иллюстрируют поведение маятника при равномерном вращении основания вокруг оси маятника (такой маятник называют маятником Фроуда).

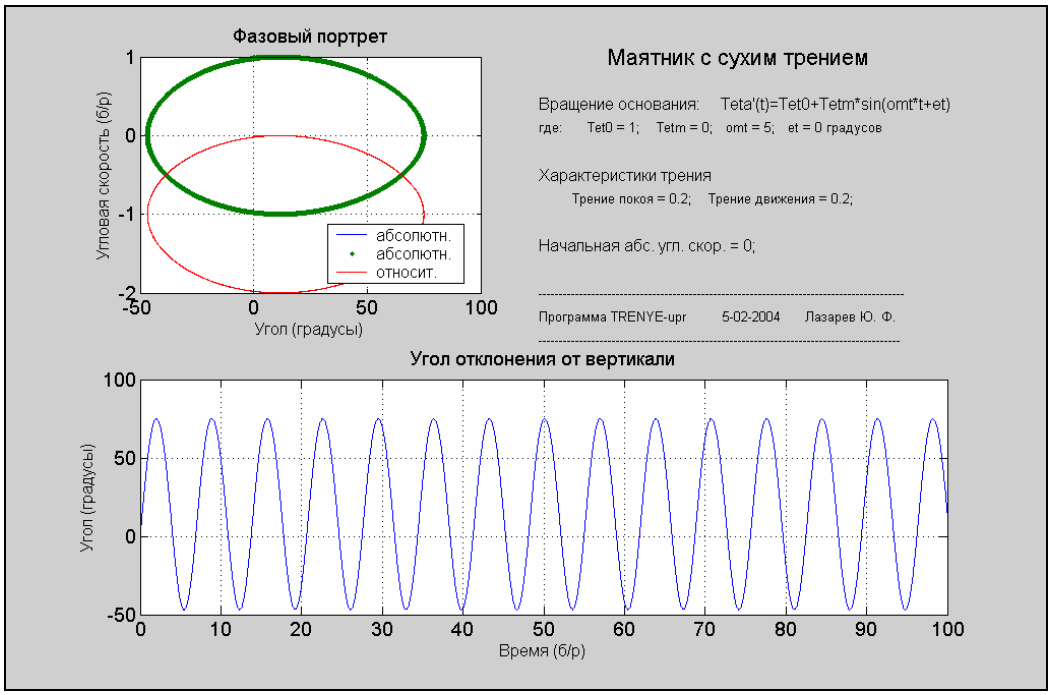


Рис. 8.50. Маятник Фроуда при значительной угловой скорости основания (1)

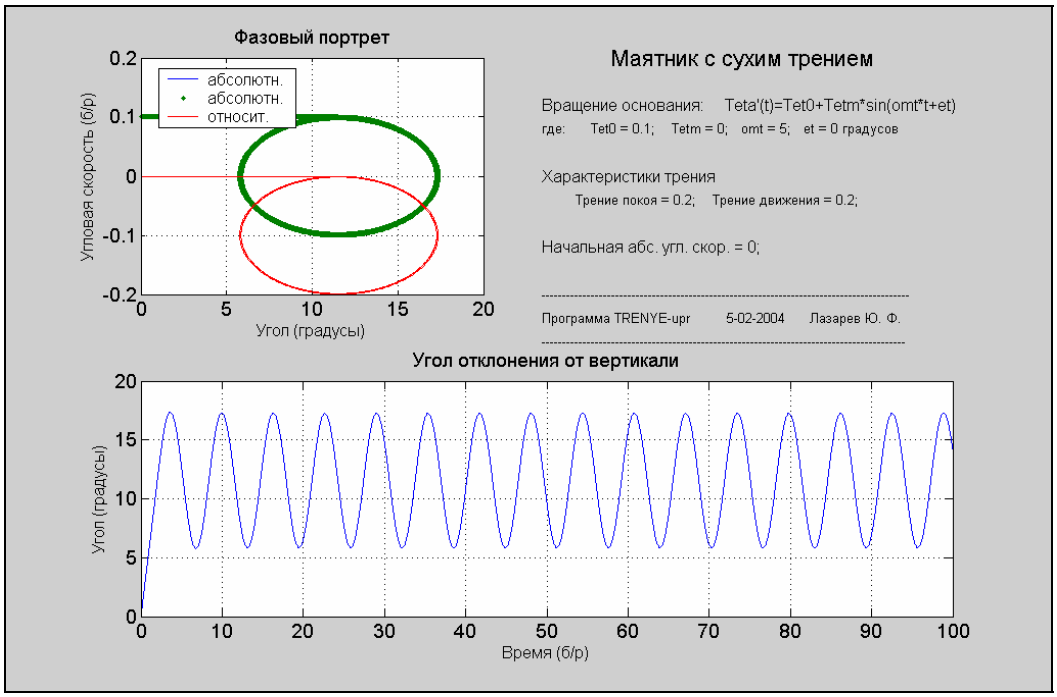


Рис. 8.51. Маятник Фроуда при малой угловой скорости основания (0,1)

Из их рассмотрения следует, что при вращении основания с постоянной угловой скоростью под действием сил сухого трения маятник совершает колебания с частотой его собственных колебаний относительно среднего положения, смещенного относительно вертикали на 11,5 градусов в сторону вращения основания. От величины угловой скорости основания зависит только амплитуда этих колебаний.

Влияние колебаний основания вокруг оси маятника продемонстрировано на рис. 8. 52 и 8. 53.

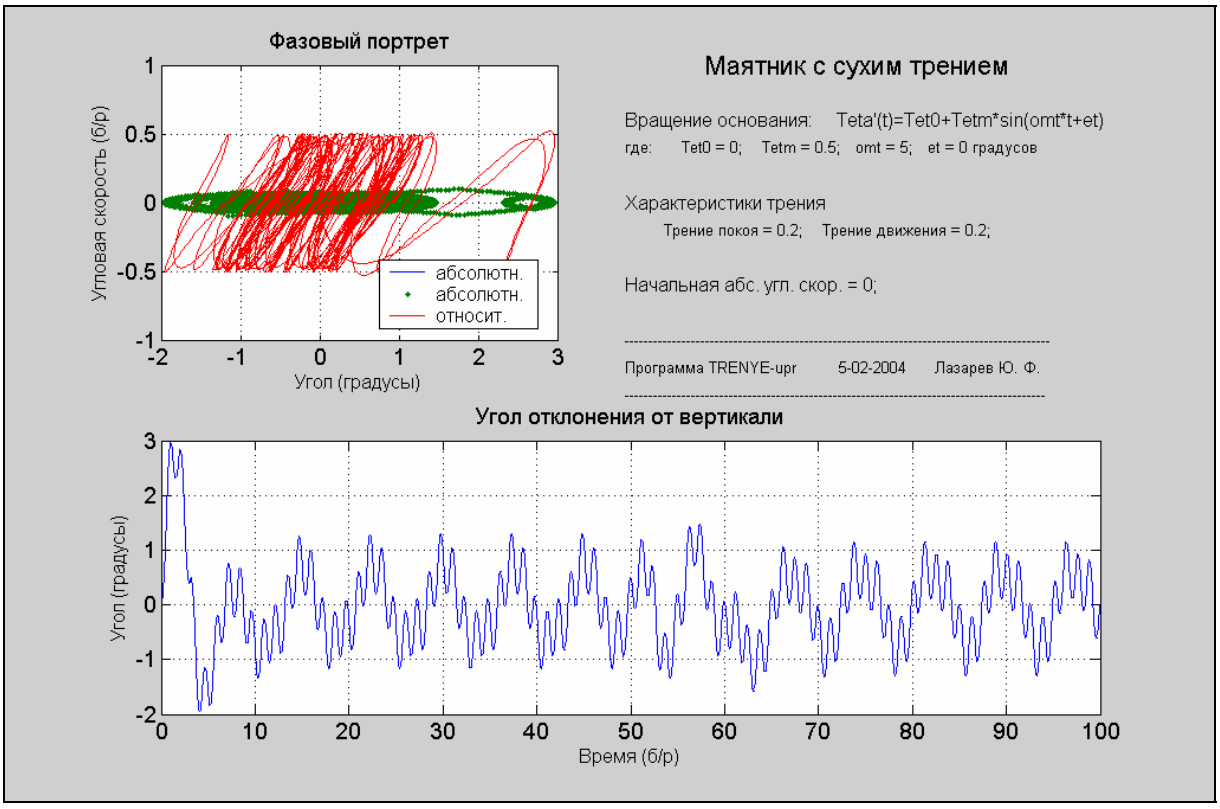


Рис. 8.52. Маятник при колебаниях основания с большой амплитудой

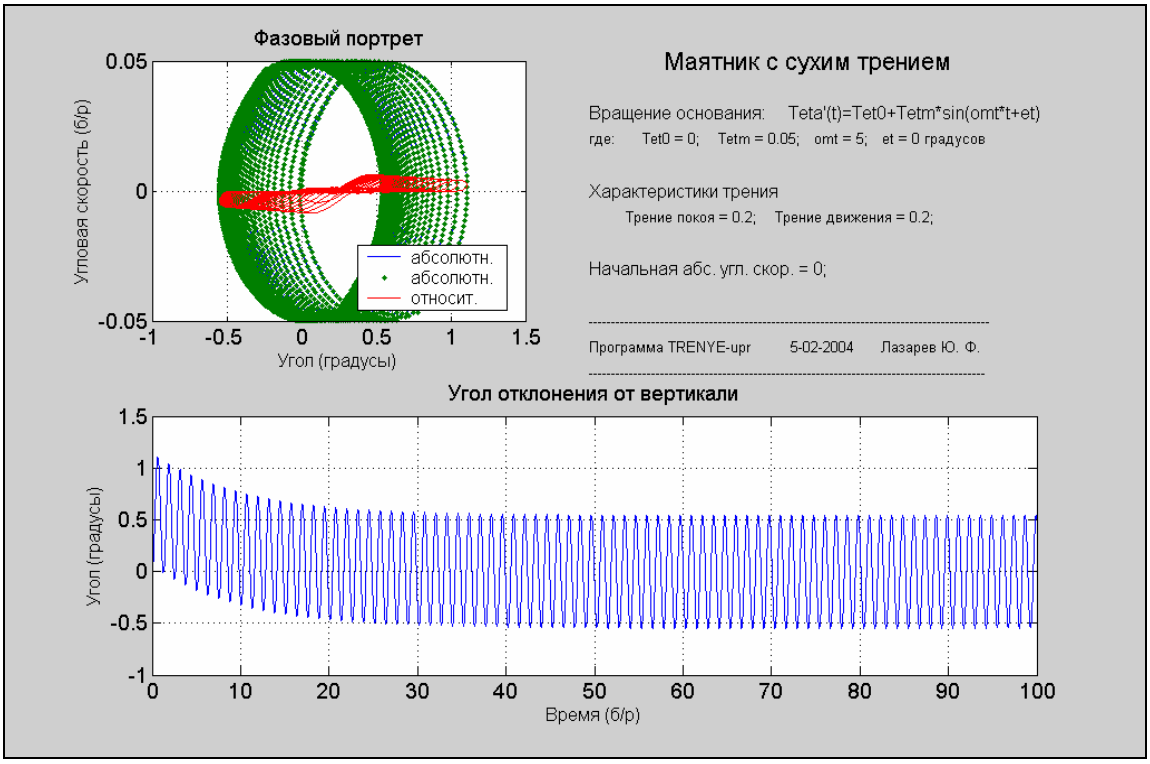


Рис. 8.53. Маятник при колебаниях основания с малой амплитудой

Интересно, что амплитуда вынужденных колебаний маятника практически не зависит от амплитуды колебаний основания. Но при значительных амплитудах колебания основания, на вынужденные колебания накладываются незатухающие собственные колебания маятника.

8.4. Вопросы для самопроверки

1. Как внутри блоков обозначаются входные величины, выходные величины блока и его переменные состояния?

2. Что такое функции пересечения нуля, для чего они служат и в какие блоки входят?

3. Опишите средства пакета **Simulink**, обеспечивающие связь данных из рабочего пространства MatLab и данных, содержащихся в S-модели.

4. Что такое S-функции, для чего они предназначены, и как их создать?

5. Как обеспечить запуск S-модели из программы MatLab?

6. Как обеспечить запуск программы MatLab из S-модели?

7. Что такое маска блока, и для чего она предназначена?

8. Как создать окно настраивания блока?

9. Как создать собственную библиотеку S-блоков?

Урок 9. Моделирование аэрокосмических объектов (библиотека Aerospace)

Мы познакомились с ядром пакета **Simulink** – библиотекой **SIMULINK**. Блоки, входящие в нее, являются основой создания любых S-моделей. Специалисты различного профиля, основываясь на возможностях ядра **Simulink**, разработали ряд S-библиотек, приспособленных для решения специфических задач своей отрасли. Ряд таких библиотек включены в комплект поставки пакета **Simulink**. Одной из них является библиотека **Aerospace Blockset**, предназначенная для моделирования динамики полетов аэрокосмических объектов.

9.1. Общая характеристика библиотеки Aerospace

Войдите в браузер `Simulink` и с помощью контекстного меню библиотеки **Aerospace** вызовите окно aerolibv1 этой библиотеки (рис. 9. 1).

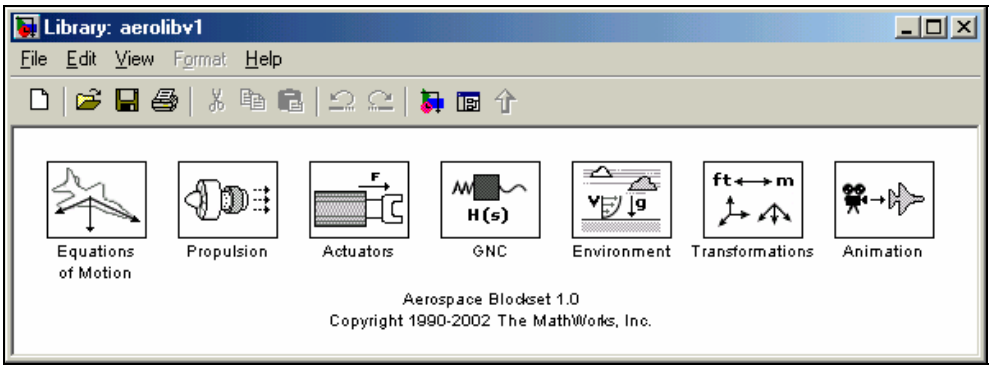


Рис. 9. 1. Окно библиотеки aerolibv1

Как видим, в состав библиотеки входят шесть разделов:

Equatios of Motion	(Уравнения движения) содержит блоки, позволяющие составить модель летательного аппарата;
Propulsion	(Двигатель) включает блоки, моделирующие влияние двигательной установки летательного аппарата;
Actuators	(Привод, Рулевые машинки) содержит блоки, моделирующие поведение привода рулей летательного аппарата;
GNC	Содержит блоки моделирования системы управления движением летательного аппарата
Environment	(Среда) состоит из блоков, моделирующих влияние окружающей среды на движущийся в ней летательный аппарат
Transformations	(Преобразования) содержит блоки преобразования координат
Animation	(Анимация) включает блоки, позволяющие построить анимационные изображения движения летательного аппарата в пространстве.

Раздел Equations of Motion

В разделе Equatons of Motion находятся две группы блоков (рис. 9. 2): 6DoF (6 Degree of freedom – 6 степеней свободы) и 3DoF (3 Degree of freedom – 3 степени свободы).

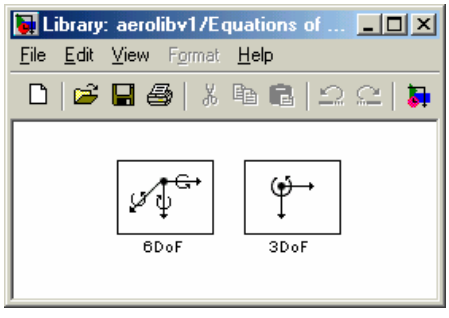


Рис. 9. 2. Содержимое раздела Equations of Motion

В первой группе расположены блоки, позволяющие задать модель пространственного (с шестью степенями свободы – три перемещения вдоль осей декартовой системы координат и три угла поворота ЛА относительно этой системы координат) движения. Дважды щелкнув мышью на изображении группы 6DoF, вы получите на экране окно, представленное на рис. 9.3.

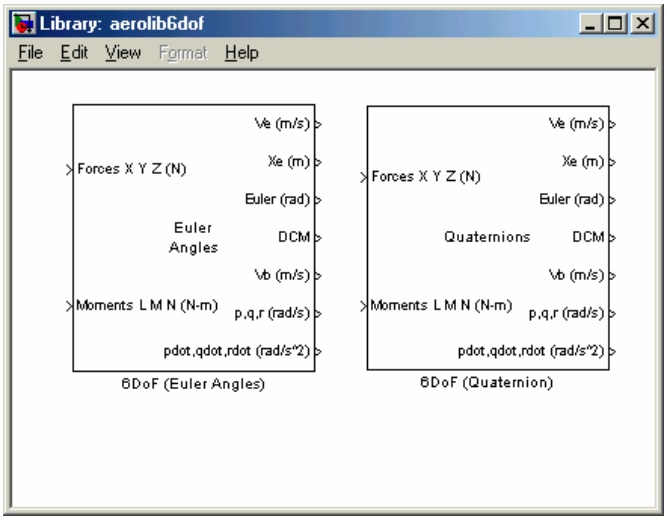
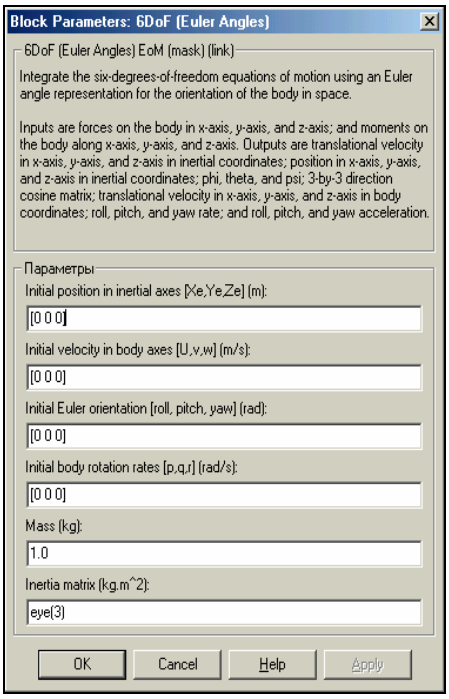


Рис. 9. 3. Блоки подраздела aerolib6dof группы 6DoF

В нем вы обнаружите два блока - 6DoF (Euler Angles) и 6DoF (Quaternion). Оба блока представляют собой модели поведения твердого тела с шестью степенями свободы. Но первый из них осуществляет представление углового движения тела в так называемых углах Эйлера, а второй – в виде кватерниона поворота.
Окна настраивания блоков (рис. 4 и 5) почти не отличаются.



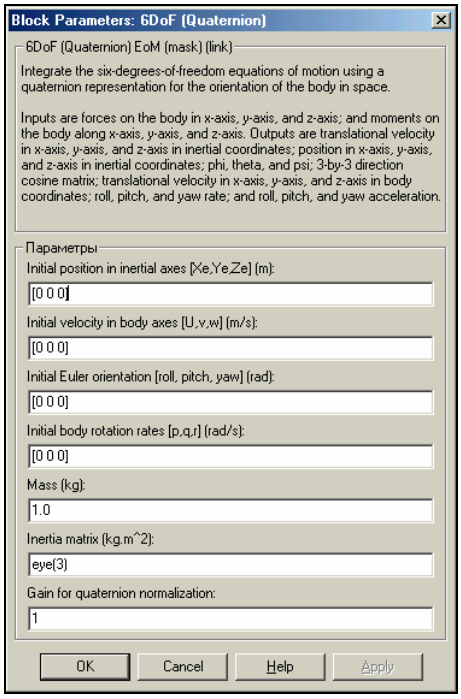


Рис. 9. 4. Окно настраивания блока 6DoF (Euler Angles) Рис. 9. 5. Окно настраивания блока 6DoF (Quaternion)

Прежде чем ознакомиться с содержанием этих окон, следует оговорить особенности и обозначения систем координат, используемых в библиотеке **Aerospace.**

В качестве основной (базовой) системы координат здесь принята система декартовых (взаимно ортогональных) осей $X_eY_eZ_e$, связанная с поверхностью Земли. При этом ось Z_e предполагается вертикальной и

направленной вниз, к центру Земли. Две другие оси лежат в плоскости горизонта. Земля предполагается неподвижной, не вращающейся в пространстве и плоской.

Отсюда следует:

1) система земных осей $X_e Y_e Z_e$ в этих условиях является также и инерциальной;

2) с помощью библиотеки **Aerospace** можно изучать движения вблизи поверхности Земли лишь на небольших расстояниях от начальной точки и в течение небольшого промежутка времени, когда кривизной Земли и ее вращением в пространстве можно пренебречь.

Вторая система координат $X_b Y_b Z_b$ по умолчанию имеет начало в центре масс O летательного аппарата (в дальнейшем – ЛА). Ось X_b направлена по продольной оси ЛА к носу, ось Y_b перпендикулярная ей, лежит в плоскости крыльев и направлена вправо (если смотреть с хвоста на нос ЛА), ось Z_b перпендикулярна плоскости крыльев и направлена вниз.

Проекции вектора $\mathbf{V}$ скорости ЛА на оси $X_b Y_b Z_b$ обозначаются u_b, v_b и w_b соответственно, проекции вектора $\boldsymbol{\omega}$ абсолютной угловой скорости ЛА – соответственно p, q и r, а проекции вектора $\mathbf{M}$ момента внешних сил, действующих на ЛА - L, M и N.

Углы Эйлера, используемые в библиотеке, состоят из углов рыскания ψ (yaw), тангажа ϑ (pitch) и крена φ (roll). Угол рыскания представляет собой угол отклонения в плоскости горизонта продольной оси X_b ЛА от направления оси X_e земной системы координат. Угол тангажа – это угол подъема продольной оси ЛА над плоскостью горизонта, а угол крена является углом поворота корпуса ЛА вокруг продольной его оси.

Возвращаясь к окнам настраивания, отметим, что в параметры настраивания входят такие величины:

Initial position in inertial axes [Xe,Ye,Ze] (m)	Начальное положение в инерциальных (земных) осях. Следует задать начальное отклонение центра масс О ЛА от начала земной системы координат в метрах
Initial velocity in body axes [u,v,w] (m/s)	Начальные скорости в осях тела. Следует задать проекции скорости центра масс ЛА в начальный момент времени на оси, связанные с ЛА в метрах в секунду
Initial Euler orientation [roll,pitch,yaw] (rad)	Начальная ориентация в углах Эйлера. Следует задать начальные углы крена, тангажа и рыскания в радианах
Initial body rotation rates [p,q,r] (rad/s)	Начальные угловые скорости тела. Следует задать начальные значения проекций угловой скорости ЛА на оси, связанные с ЛА, в радианах в секунду
Mass (kg)	Задается значение массы ЛА в килограммах
Inertia matrix (kg.m^2)	Задается матрица 3×3 моментов инерции ЛА относительно осей, связанных с ним

Входные величины у обоих блоков одинаковы. Это вектор текущих проекций на оси ЛА всех внешних сил, действующих на него, и вектор текущих моментов сил относительно осей ЛА.

Выходы обоих блоков также одинаковы. Они перечислены в следующей таблице.

Ve (m/s)	Вектор проекций текущего значения вектора скорости центра масс ЛА на оси земной (инерциальной) системы координат
Xe (m)	Вектор текущих смещений центра масс ЛА относительно начала земной (инерциальной) системы координат
Euler (rad)	Вектор текущих значений углов крена, тангажа и рыскания соответственно
DCM	Текущее значение матрицы направляющих косинусов связанных осей относительно земных осей
Vb (m/s)	Вектор проекций текущего значения вектора скорости центра масс ЛА на оси системы координат, связанной с корпусом ЛА
p,q,r (rad/s)	Вектор проекций текущей угловой скорости ЛА на оси, связанные с ЛА
pdot,qdot,rdot (rad/s^2)	Вектор производных от проекций текущей угловой скорости ЛА на оси, связанные с ЛА

Вторая группа блоков 3DoF позволяет моделировать движение ЛА в одной плоскости (обычно – продольное движение в вертикальной плоскости). Она содержит два блока (рис. 9.6) – Equations of Motion (Body Axes) (Уравнения движения в связанных осях) и Incidence & Airspeed (Угол атаки и воздушная скорость).

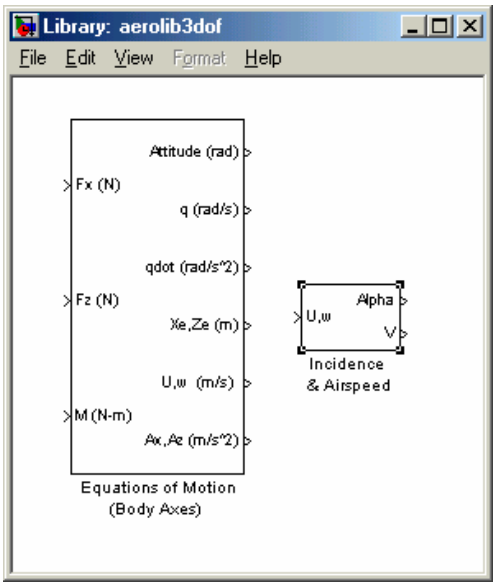


Рис. 9. 6. Содержимое раздела 3DoF

Первый блок позволяет моделировать продольное движение путем численного интегрирования уравнений продольного движения ЛА. Второй вычисляет по заданным проекциям скорости ЛА на оси X_b и Y_b угол атаки Alpha и величину вектора воздушной скорости.

На рис. 9.7. показано окно настраивания блока Equations of Motion.

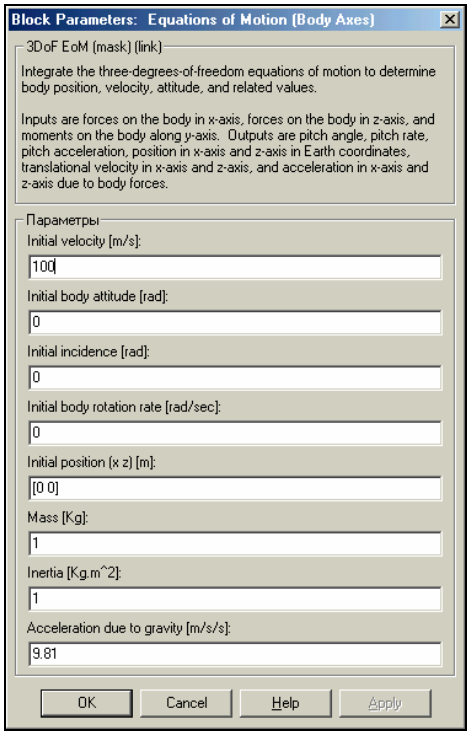


Рис. 9. 7. Окно настраивания блока Equations of Motion

С его помощью вводятся следующие параметры, необходимые для численного интегрирования дифференциальных уравнений продольного движения.

`Initial velocity (m/s)` Начальная скорость

`Initial body attitude (rad)` Начальный угол подъема вектора скорости над плоскостью горизонта

Initial incidence (rad)	Начальный угол атаки
Initial body rotate rate (rad/sec)	Начальная угловая скорость тангажа
Initial position [x z] (m)	Начальное положение центра масс
Mass (kg)	Масса ЛА
Inertia (kg m^2)	Момент инерции ЛА относительно поперечной оси
Acceleration due to gravity (m/s/s)	Ускорение силы тяжести

Входы блока перечислены ниже.

Fx (N) текущее значение силы (в ньютонах), действующей на ЛА вдоль его продольной оси X_b

Fz (N) текущее значение силы (в ньютонах), действующей на ЛА вдоль его нормальной оси Z_b

M (Nm) текущее значение момента сил (в ньютонах на метр), действующего на ЛА вокруг его поперечной оси Y_b

Выходами блока являются нижеуказанные величины.

Altitude (rad)	текущее значение угла между вектором скорости ЛА и плоскостью горизонта (в радианах)
q (rad/s)	текущее значение проекции угловой скорости ЛА на его поперечную ось (в радианах в секунду)
qdot (rad/s^2)	текущее значение проекции углового ускорения ЛА на его поперечную ось (в радианах в секунду в квадрате)
Xe, Ze (m)	текущие координаты центра масс ЛА в продольной плоскости в земной системе координат (в метрах)
U, w (m/s)	текущие значения проекций скорости ЛА соответственно на продольную и нормальную оси ЛА (в метрах в секунду)
Ax, Az (m/s^2)	текущие значения проекций ускорения ЛА соответственно на продольную и нормальную оси ЛА (в метрах в секунду в квадрате)

Рассмотренные блоки являются основными для моделирования движения ЛА. В них сосредоточены программы, осуществляющие численное интегрирование дифференциальных уравнений. Но для их работы необходимо формировать текущие значения сил и моментов сил, действующих на ЛА в течение его полета. Эти силы и моменты можно разделить на три группы:

- силы и моменты, действующие на ЛА со стороны маршевого двигателя;

- силы и моменты, действующие на корпус ЛА со стороны окружающей его среды; сюда относятся силы и моменты аэродинамического сопротивления движению ЛА в атмосфере и сила тяжести ЛА;

- силы и моменты, накладываемые на ЛА со стороны управляющих органов, таких как рули, элероны, элевоны, закрылки и т. п.

В общем случае эти силы и моменты настолько различаются для разных типов ЛА, что нельзя создать универсальные блоки для их вычисления. Поэтому в библиотеке **Aerospace** имеются лишь некоторые блоки, осуществляющие наиболее часто встречающиеся зависимости в законах образования некоторых сил и моментов.

Раздел Environment

В этом разделе библиотеки содержатся три группы блоков, позволяющих учитывать влияние параметров окружающей среды на силы и моменты, действующие на ЛА в его полете (рис. 9.8), - Atmosphere (Атмосфера), Gravity (Гравитация) и Wind (Ветер).

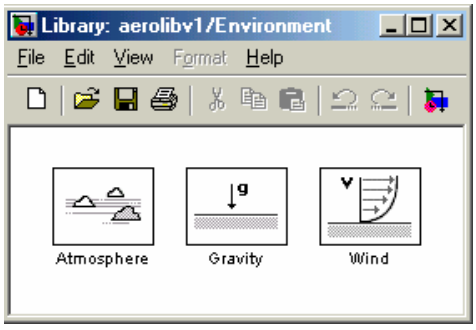


Рис. 9. 8. Содержимое раздела Environment

В первой группе (Atmosphere) размещены блоки (рис. 9) ISA Atmosphere Model (ISA-модель атмосферы) и COESA Atmosphere Model (COESA-модель атмосферы).

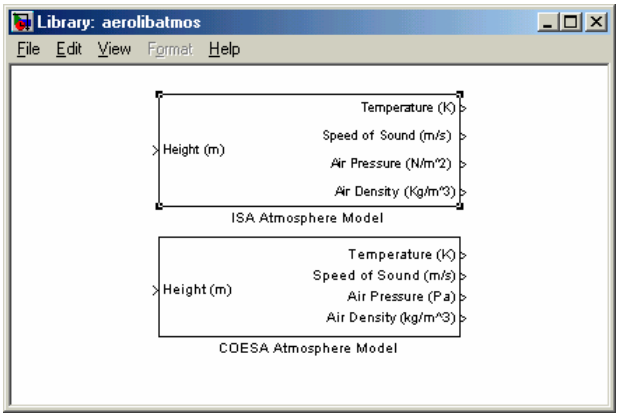


Рис. 9. 9. Содержимое группы Atmosphere

Эти блоки рассчитывают параметры атмосферы на текущей высоте полета. Входным параметром обоих блоков является значение текущей высоты полета ЛА. Выходные параметры приведены ниже.

Temperature (K)	Температура (в градусах Кельвина)
Speed of sound (m/s)	Скорость звука (в метрах в секунду)
Air Pressure (N/m^2)	Давление воздуха (ньютон на метр в квадрате)
Air Density (kg/m^3)	Плотность воздуха (килограмм на метр кубический)

Первый блок производит расчеты по международной модели стандартной атмосферы до высоты полета 20 км. Второй – по американской расширенной модели стандартной атмосферы. Здесь возможен учет особенностей атмосферы до высоты 84852 м.

Параметров настраивания у этих блоков нет.

Вторая группа Gravity – содержит только один блок WGS84 Gravity Model (рис. 10), который рассчитывает значение ускорения свободного падения на текущей высоте полета ЛА и на текущей географической широте.

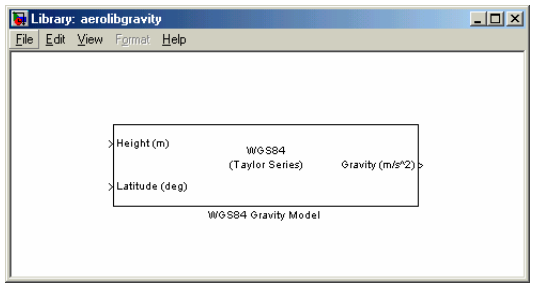


Рис. 9. 10. Содержимое группы Gravity

Раздел Propulsion

Раздел Propulsion (Двигатель) содержит единственный блок (рис. 11) – Turbofan Engine System (Система турбовентиляторного двигателя).

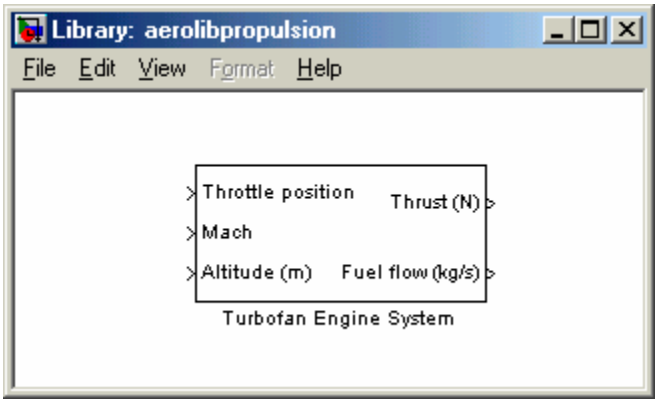


Рис. 9. 11. Содержимое раздела Propulsion

Блок имеет следующие входы:

Throttle position	Текущее положение регулирующего дросселя
Mach	Текущее значение числа Маха
Altitude (m)	Текущее значение высоты полета (в метрах)

и такие выходы:

Thrust (N)	Сила тяги (в ньютонах)
Fuel flow (kg/s)	Расход горючего (в килограммах в секунду)

Окно настраивания блока приведено на рис. 12.

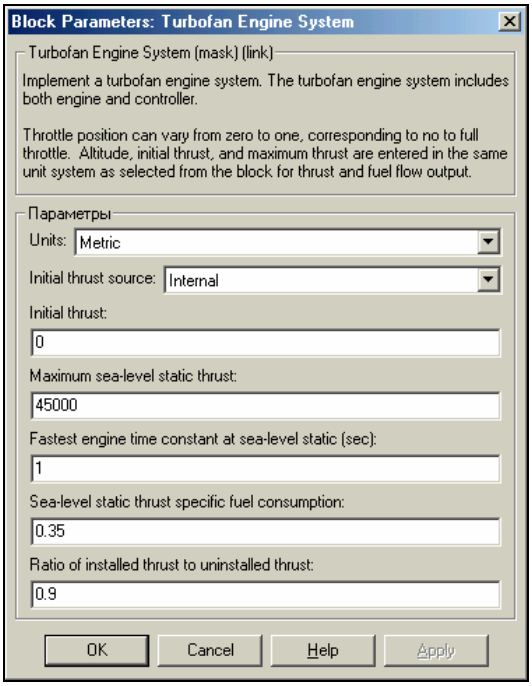


Рис. 9. 12. Окно настраивания блока Turbofan Engine System

В число параметров настраивания блока входят:

Initial thrust source	Источник (внутренний или внешний) начального значения тяги двигателя
Initial thrust	Начальное значение тяги двигателя
Maximum sea level static thrust	Максимальное значение силы тяги на уровне океана
Fastest engine time constant at sea level static (sec)	Постоянная времени двигателя на уровне океана (в секундах)
Sea level static thrust specific fuel consumption	Удельный расход топлива на уровне океана
Ratio of installed thrust to uninstalled thrust	Отношение установленной силы тяги к неустановленной

Разделы Actuators и GNC

Разделы Actuators (Исполнительные элементы) и GNC (Регуляторы управления движением) содержат блоки, помогающие создать модель системы автоматического управления движением ЛА.

В общем случае система автоматического управления движением состоит из следующих частей:

- измерителей параметров движения ЛА – гироскопических приборов, измеряющих углы поворота корпуса ЛА и его угловые скорости, измерителей скорости ЛА, углов атаки и дрейфа и т. п.;
- регуляторов – звеньев системы управления, которые формируют закон управления (требуемые зависимости регулируемых параметров от измеренных величин);
- силового привода, обеспечивающего поворот рулей, элеронов и элевонов на требуемые углы, величины которых определены регулятором;
- исполнительных органов (рулей, элеронов и элевонов), обеспечивающих наложение на ЛА требуемых моментов сил.

Модели измерителей параметров движения и работы исполнительных органов пользователю необходимо создавать самому. Универсальных блоков, моделирующих их поведение в библиотеке нет.

Раздел Actuators содержит блоки, моделирующие движение силового привода рулевых исполнительных органов (рис. 13).

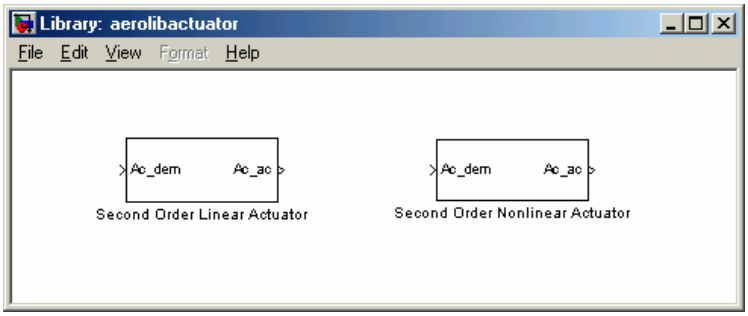


Рис. 9. 13. Содержимое раздела Actuators

В нем находятся два блока – Second Order Linear Actuator (Линейный силовой привод второго порядка) и Second Order Nonlinear Actuator (Нелинейный силовой привод второго порядка). Оба блока моделируют процесс перемещения рулевого органа при подаче на вход силового привода заданного значения этого перемещения как прохождение этого заданного сигнала через звено второго порядка с заданными частотой собственных колебаний и коэффициентом демпфирования.

В обоих блоках входом является текущее требуемое значение (Ac_dem) положения регулирующего органа, а выходом – действительное (Ac_ac) значение его положения.

Окна настраивания блоков изображены на рис. 14 и 15. У обоих блоков два параметра настройки одинаковы:

`Natural frequency` Частота собственных колебаний
`Damping ratio` Относительный коэффициент затухания
`Initial position` Начальное положение регулирующего органа

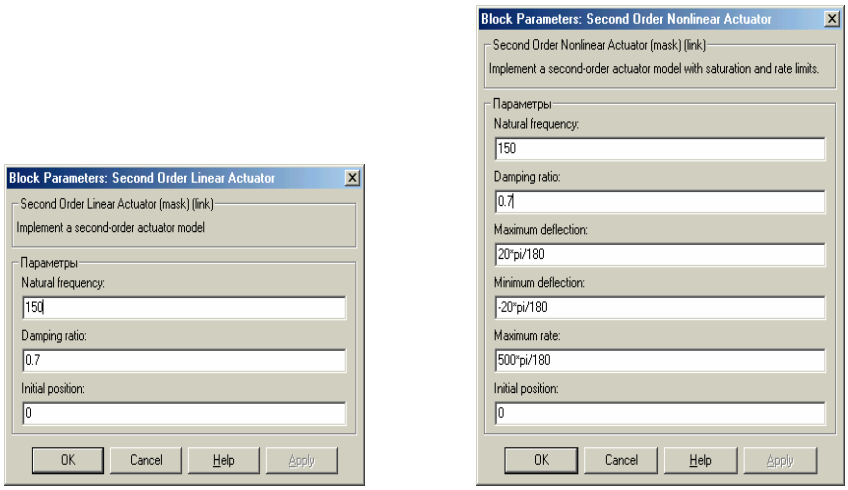


Рис. 9. 14. Окно настраивания блока Second Order Linear Actuator

Рис. 9. 15. Окно настраивания блока Second Order Nonlinear Actuator

Блок нелинейного силового привода содержит еще такие параметры настраивания:

Maximum deflection	Максимальное отклонение регулирующего органа
Minimum deflection	Минимальное отклонение регулирующего органа
Maximum rate	Максимальная скорость отклонения регулирующего органа

Именно наличием указанных ограничений на отклонения регулирующего органа и его скорость отличается нелинейный привод от линейного.

Содержимое раздела GNC показано на рис. 16. В него включены блоки, моделирующие процесс формирования сигналов, управляющих отклонением рулевых органов ЛА.

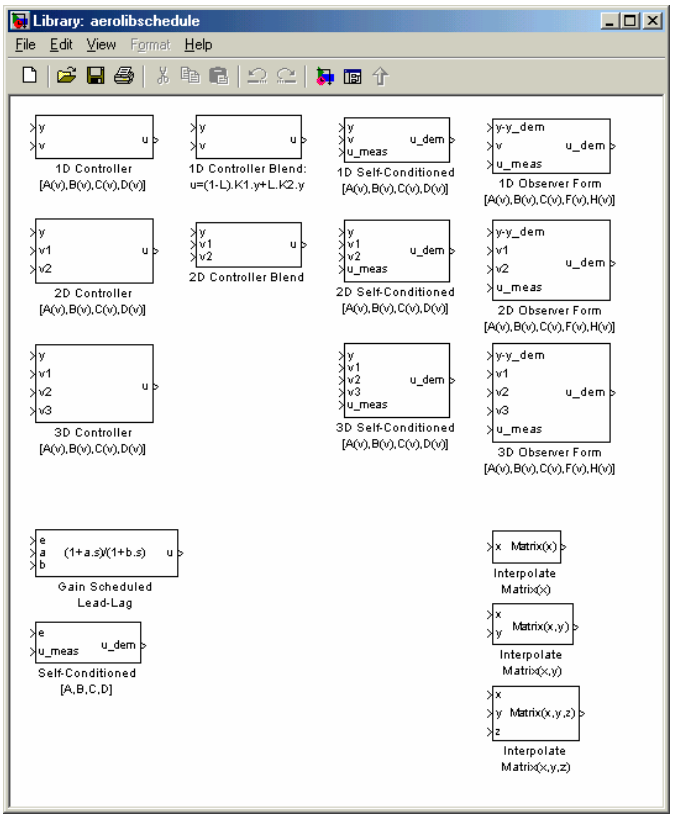


Рис. 9. 16. Содержимое раздела GNC

Кроме трех последних вспомогательных блоков, осуществляющих интерполяцию матриц, 13 блоков этого раздела призваны вырабатывать сигнал, пропорциональный требуемому углу поворота регулирующего органа. Поэтому выход во всех этих блоках один - u - требуемое текущее положение регулирующего органа.

В 11 блоках, представляющих различного вида регуляторы и наблюдатели, основным входом является вектор «y» величин, характеризующих текущее движение объекта и измеряемых на борту ЛА имеющимися измерительными приборами. Более подробное изучение этих блоков представляет интерес, главным образом, для специалистов в области проектирования систем автоматического управления движением ЛА и не входит в задачу предварительного ознакомления с возможностями библиотеки **Aerospace**.

Раздел Transfomations

Сюда помещены блоки двух групп (рис. 17) – Axes (Оси) и Units (Единицы).

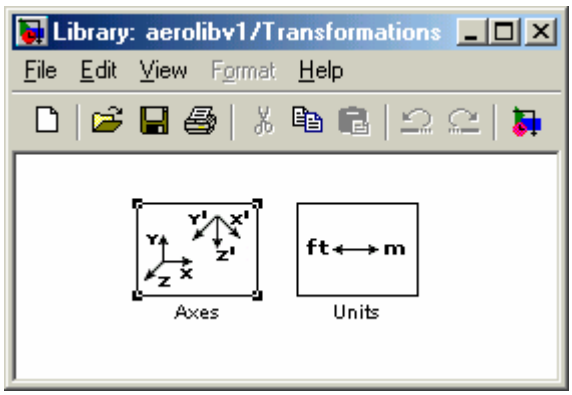


Рис. 9. 17. Содержимое раздела Transfomations

Группа Axes (рис. 18) включает 6 блоков, осуществляющих различные преобразования форм представления углового положения твердого тела в пространстве и 1 блок (3×3 Cross Product) векторного произведения двух трехкомпонентных векторов.

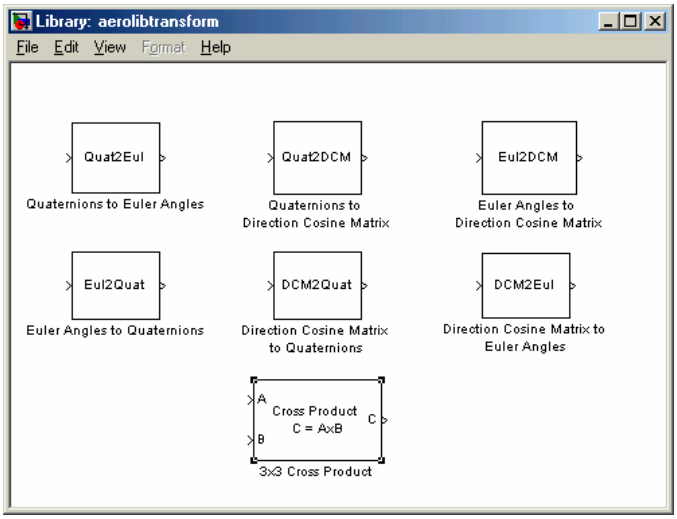


Рис. 9. 18. Содержимое группы Axes

Представим назначение блоков преобразования координат.

Quaternions to Euler Angles	Преобразует кватернион поворота в вектор трех углов Эйлера
Quaternions to Direction Cosine Matrix	Преобразует кватернион поворота в матрицу направляющих косинусов
Euler Angles to Quaternions	Преобразует вектор трех углов Эйлера в вектор кватерниона поворота
Direction Cosine Matrix to Quaternions	Преобразует матрицу направляющих косинусов в кватернион поворота
Euler Angles to Direction Cosine Matrix	Преобразует вектор трех углов Эйлера в матрицу направляющих косинусов
Direction Cosine Matrix to Euler Angles	Преобразует матрицу направляющих косинусов в вектор трех углов Эйлера

Вторая группа блоков – Units – содержит (рис. 19) 11 блоков преобразования величин из одной системы единиц в другую.

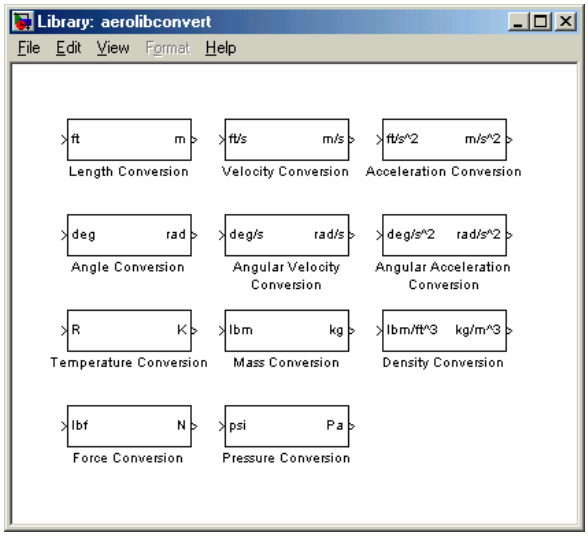


Рис. 9. 19. Содержимое группы Units

Назначение блоков очевидно из их названий. Параметры настраивания во всех блоках отсутствуют.

9.2. Моделирование свободного углового движения космического аппарата (КА)

Работу с библиотекой **Aerospace** начнем с создания простой модели углового движения КА в инерциальном пространстве.

Вариант SvDvigKA.mdl блок-схемы этой модели SvDvigKA.mdl представлен на рис. 20.

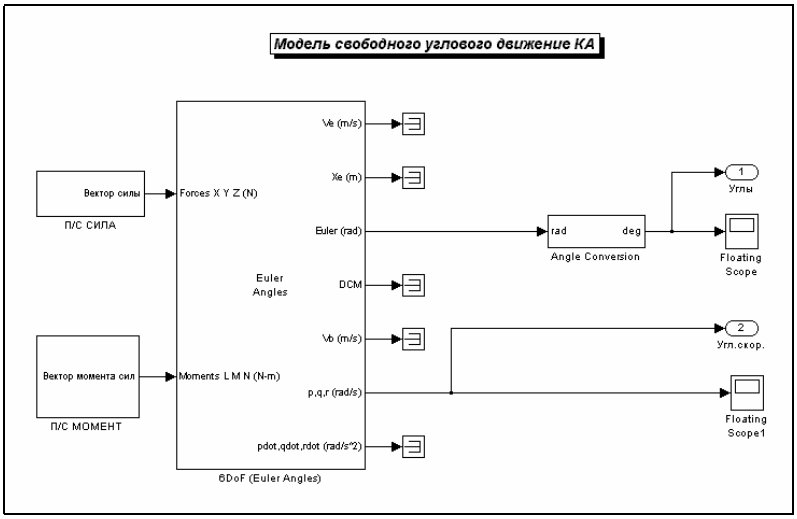


Рис. 9. 20. Модель свободного углового движения КА

В основу модели можно положить блок 6DoF (Euler Angles), обеспечивающий моделирование движение тела с шестью степенями свободы.

Если задачей моделирования является исследование свободного движения, то на вход этого блока следует подать векторные сигналы, соответствующие проекциям внешней силы и внешнего момента сил, равные нулю. Это можно обеспечить с помощью двух подсистем - «П/С СИЛА» (рис. 21) и «П/С МОМЕНТ» (рис. 22).

В дальнейшем будем использовать следующие обозначения.

`m` Масса КА

`J` Матрица моментов инерции КА

`XYZ0` Вектор проекций координат центра масс КА

`V0` Вектор проекций скорости центра масс КА

UG0 Вектор углов поворота КА

UgSk0 Вектор проекций угловой скорости КА на оси КА

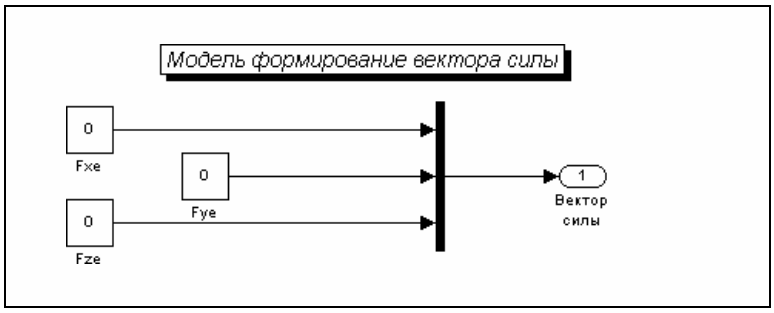


Рис. 9. 21. Подсистема «П/С СИЛА»

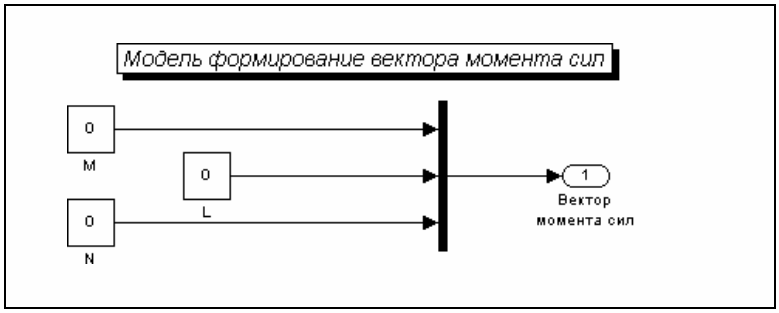


Рис. 9. 22. Подсистема «П/С МОМЕНТ»

На рис. 23 показаны настройки блока 6DoF (Euler Angles).

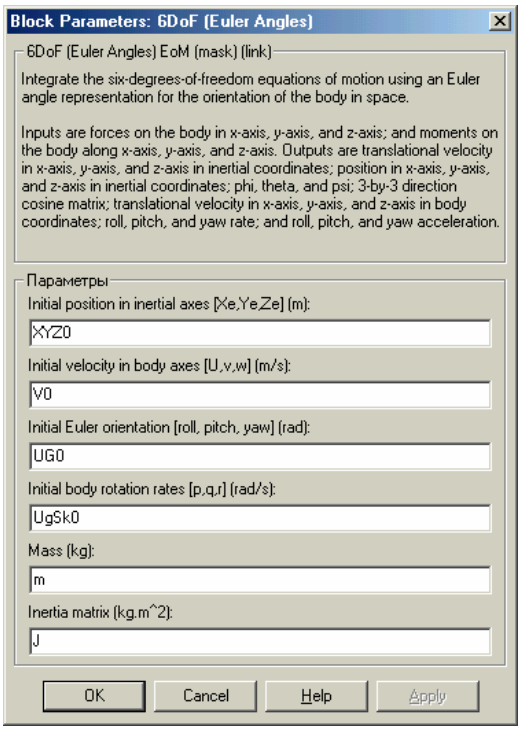


Рис. 9. 23. Настройки блока 6DoF (Euler Angles)

Управление работой модели и выведение результатов осуществим при помощи управляющей программы SvobDvigKA_upr, текст которой приводится ниже.

```
% SvobDvigKA_upr
% Управляющая программа для модели SvDvigKA

% Лазарев Ю.Ф. 29-03-2004

clear all, clc
% Установка параметров КА
J=[3400 300 -200;300 2200 100;-200 100 1400]; %  Матрица моментов инерции КА
m=2000;                       %  Масса КА
% Установка начальных условий
XYZ0=[0 0 0];                 %  Начальное положение КА
V0=[0 0 0];                   %  Начальные скорости КА
UG0=[0 0 0];                  %  Начальные углы КА
UgSk0=[1 0.1 0];              %  Начальные угловые скорости КА
% Установка параметров интегрирования
TK=300;                       %  Конечное время интегрирования
hi=0.1;                       %  Шаг интегрирования
% Запуск модели
sim('SvDvigKA');
%  Запись результатов интегрирования
Fl=yout(:,1);  TE=yout(:,2);  PSI=yout(:,3);
omx=yout(:,4);  omy=yout(:,5);  omz=yout(:,6);
t=tout;
% Графическое представление результатов
subplot(2,2,1)
plot(TE,PSI), grid
axis('equal');        set(gca,'FontSize',12)
title('Движение оси Xb в пространстве');
xlabel('Угол \theta (градусы)');        ylabel('Угол \psi (градусы)');
subplot(2,2,3)
plot(t,TE,t,PSI,'.'), grid
set(gca,'FontSize',12);       title('Углы во времени');
xlabel('Время (c)');          ylabel('Углы (градусы)');    legend(' \theta ',' \psi ',0);
subplot(2,2,4)
plot(t,omy,t,omz,'.'), grid
set(gca,'FontSize',12);       title('Угловые скорости во времени');
xlabel('Время (c)');          ylabel('Угловые скорости (рад/c)');  legend(' omy ',' omz ',0);
subplot(2,2,2)
axis('off');
h=text(-0.3,1.1,'Свободное угловое движение космического аппарата','FontSize',14);
h=text(0.1,0.9,'| ','FontSize',12);        h=text(0.2,0.9,num2str(J(1,1)),'FontSize',12);
h=text(0.4,0.9,num2str(J(1,2)),'FontSize',12);    h=text(0.6,0.9,num2str(J(1,3)),'FontSize',12);
h=text(0.8,0.9,'| ','FontSize',12);        h=text(-0.1,0.8,'J = ','FontSize',12);
h=text(0.1,0.8,'| ','FontSize',12);        h=text(0.2,0.8,num2str(J(2,1)),'FontSize',12);
h=text(0.4,0.8,num2str(J(2,2)),'FontSize',12);    h=text(0.6,0.8,num2str(J(2,3)),'FontSize',12);
h=text(0.8,0.8,'| ','FontSize',12);        h=text(0.1,0.7,'| ','FontSize',12);
h=text(0.2,0.7,num2str(J(3,1)),'FontSize',12);    h=text(0.4,0.7,num2str(J(3,2)),'FontSize',12);
h=text(0.6,0.7,num2str(J(3,3)),'FontSize',12);    h=text(0.8,0.7,'| ','FontSize',12);
h=text(-0.1,0.5,'Начальные углы (градусы)','FontSize',12);
h=text(0.1,0.4,['\psi0 = ',num2str(UG0(3)*180/pi)],'FontSize',12);
h=text(0.4,0.4,['\theta0 = ',num2str(UG0(2)*180/pi)],'FontSize',12);
h=text(0.7,0.4,['\phi0 = ',num2str(UG0(1)*180/pi)],'FontSize',12);
h=text(-0.1,0.2,'Начальные угловые скорости (рад/c)','FontSize',12);
h=text(0.1,0.1,['omx0 = ',num2str(UgSk0(1))],'FontSize',12);
h=text(0.4,0.1,['omy0 = ',num2str(UgSk0(2))],'FontSize',12);
h=text(0.7,0.1,['omz0 = ',num2str(UgSk0(3))],'FontSize',12);
h=text(-0.1,-0.05,'-------------------------------------------------------------------------------------------');
h=text(-0.1,-0.1,'Программа SvobDvigKA-upr       Лазарев Ю. Ф.     29-03-2004');
h=text(-0.1,-0.15,'-------------------------------------------------------------------------------------------');
```

Далее (рис. 24…26) приведены результаты моделирования для трех случаев различного распределения масс КА: 1) матрица моментов инерции является диагональной (динамически симметричный КА); 2) матрица моментов инерции является диагональной с разными моментами инерции; 3) матрица моментов инерции является КА произвольна.

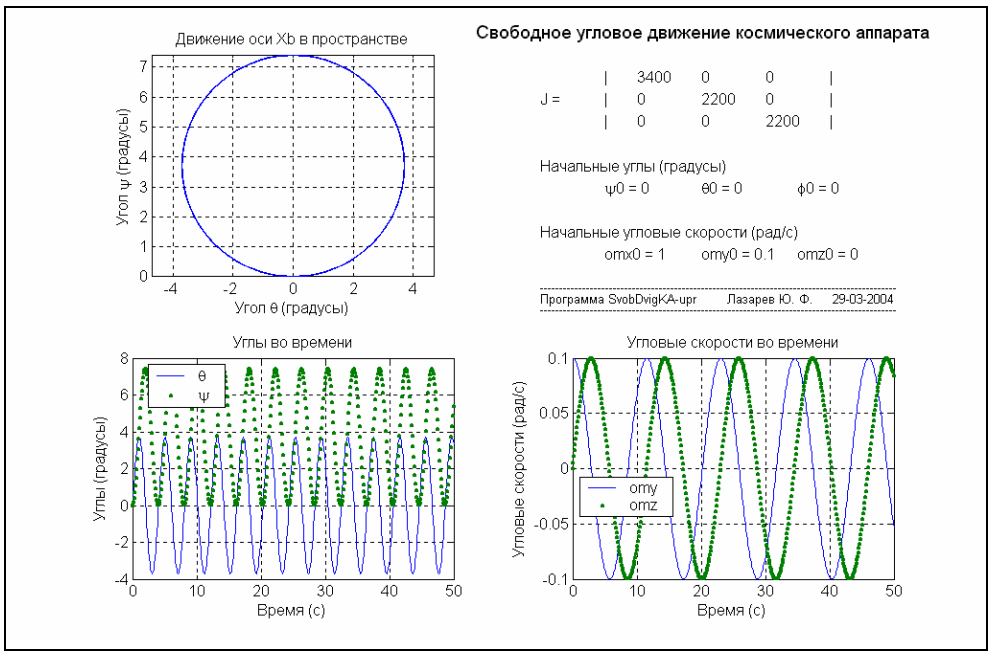


Рис. 9. 24. Свободное движение динамически симметричного КА

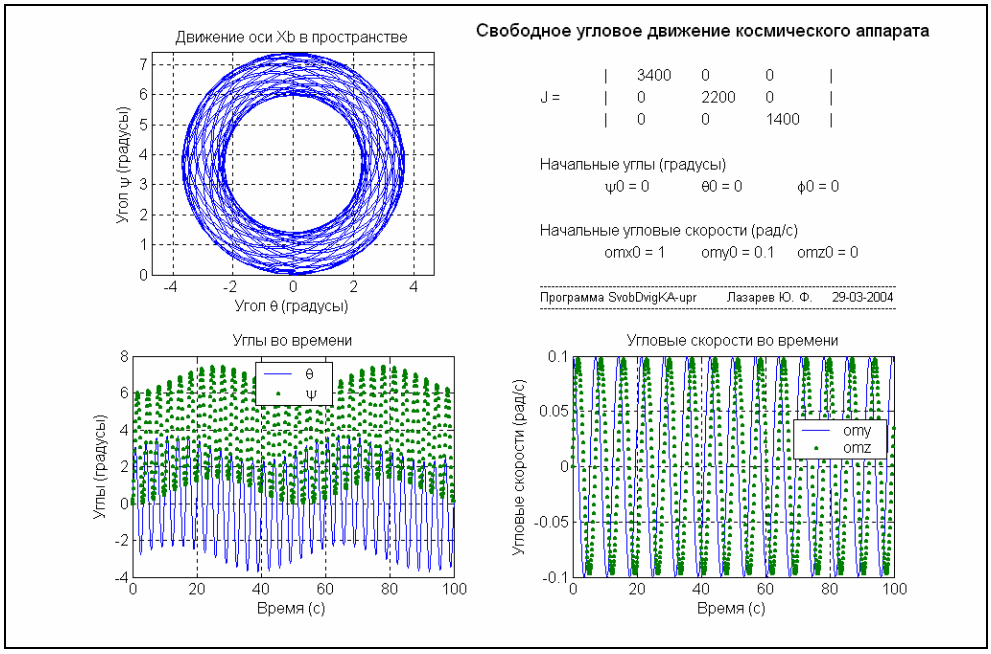


Рис. 9. 25. Свободное движение динамически несимметричного КА

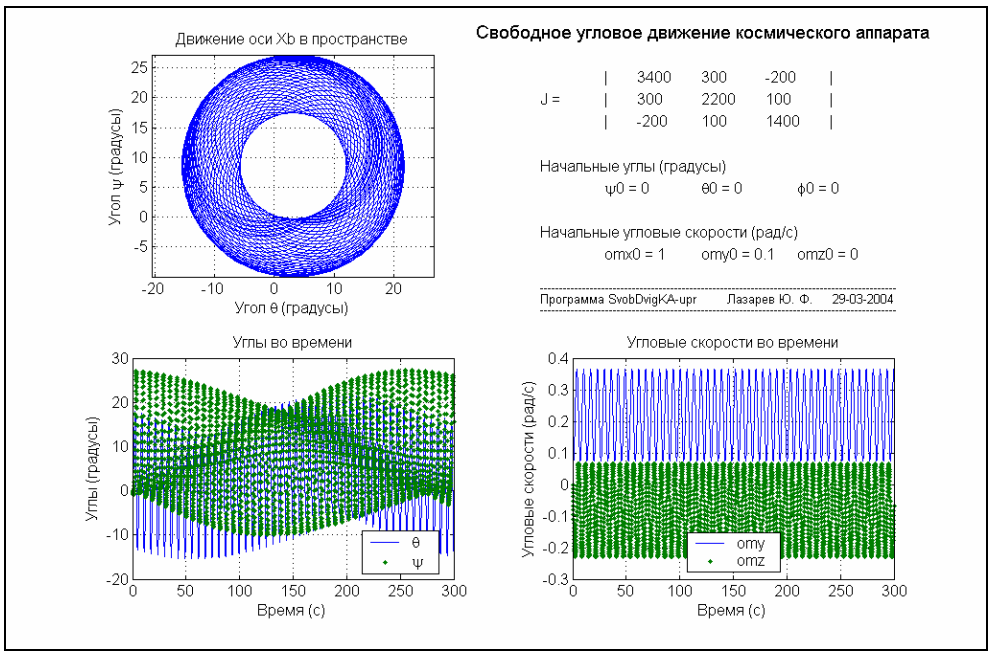


Рис. 9. 26. Свободное движение динамически несбалансированного КА

Полученные результаты подтверждают выводы теоретического анализа, приведенные в [14].

9.3. Моделирование управляемого углового движения космического аппарата

Перейдем теперь к созданию модели управляемого движения КА по углам ориентации. Задачей управления будем полагать приведение КА в некоторое неподвижное в инерциальном пространстве положение.

Для обеспечения управления следует добавить в предыдущую модель контур управления ориентацией КА, предполагая, что углы и угловые скорости поворота КА в инерциальной системе координат измеряются приборами, установленными на его борту, и формируя моменты сил управления на основе этой информации. Контур управления реализован в модели UprDvigKA.mdl, представленной на рис. 27, в виде подсистемы «СУО», блок-схема которой показана на рис. 28.

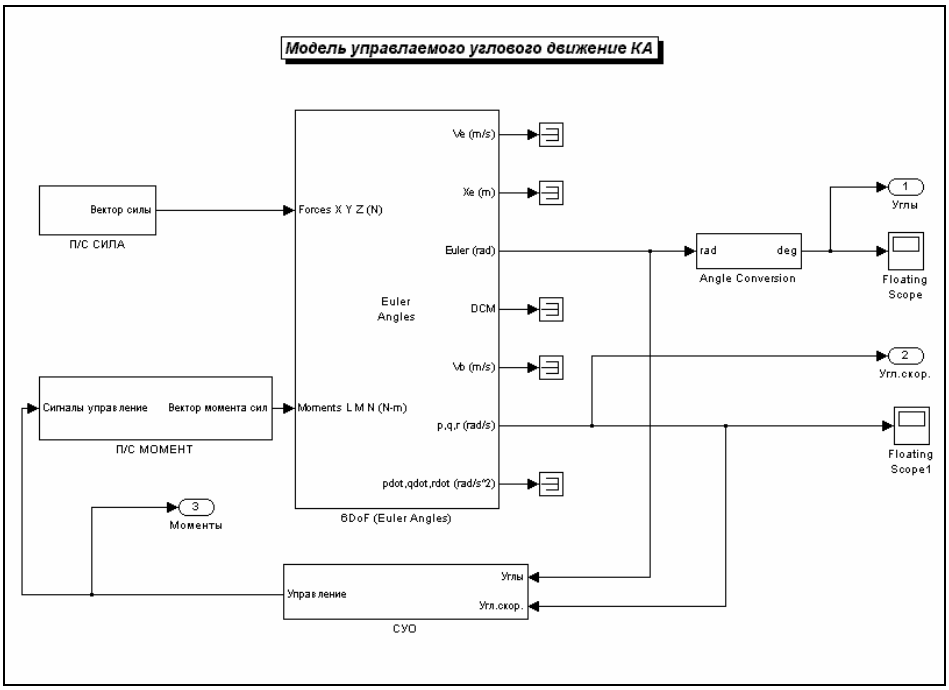


Рис. 9. 27. Блок-схема управляемого процесса ориентацией КА

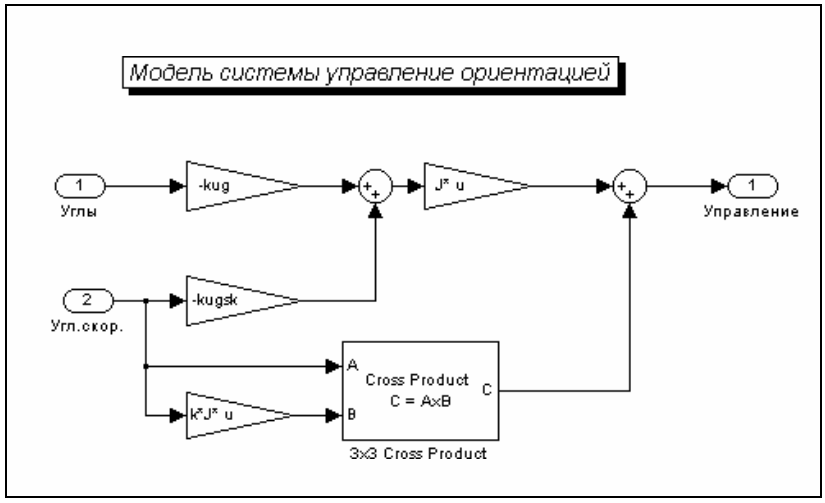


Рис. 9. 28. Блок-схема подсистемы «СУО»

Помимо принятых ранее здесь используются следующие обозначения.

kug коэффициент обратной связи по углу отклонения

Kugsk коэффициент обратной связи по угловой скорости

k коэффициент компенсации гироскопического момента

При этом предполагается, что момент управления ориентацией формируется по закону:

M = - kug*J*Ug – kugsk*J*om – k*(om×)*J*om,

где M – вектор проекций момента управления, Ug – вектор углов Эйлера, om – вектор проекций угловой скорости, (om×) – кососимметричная матрица из проекций угловой скорости. Именно этот закон формируется в подсистеме «СУО».

Проиллюстрируем работу модели на нескольких примерах.

Проверим работу, моделируя движение КА при отсутствии управления (kug=kugsk=k=0). На рис. 29 показаны результаты такого моделирования для КА с матрицей моментов инерции, принятой в [5].

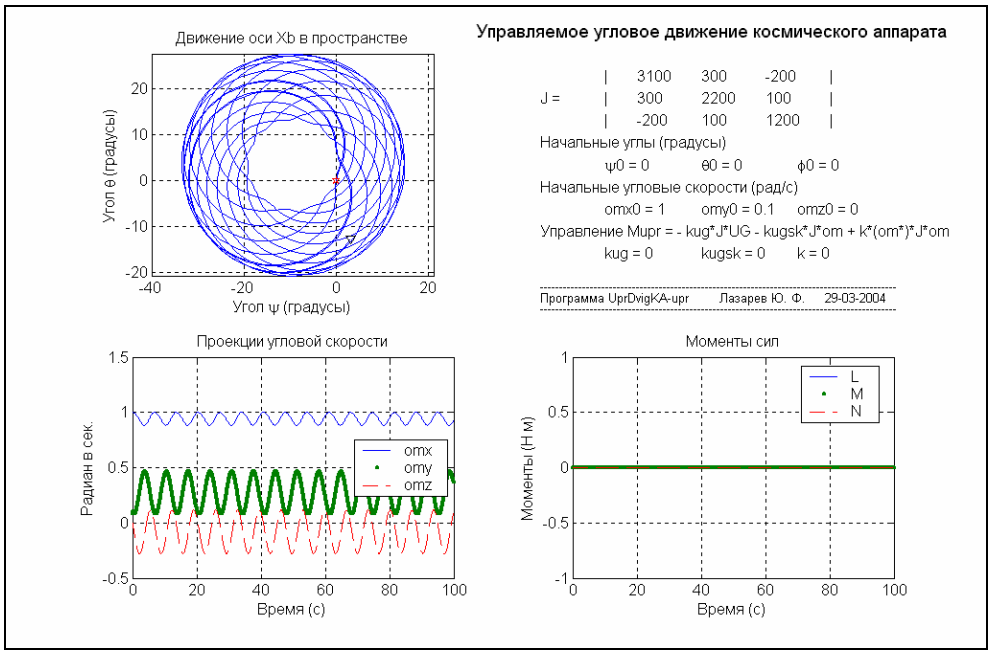


Рис. 9. 29. Неуправляемое движение несбалансированного КА

Нетрудно убедиться, что результат моделирования совпадает с ожидаемым (см. [14]).

Сначала рассмотрим, как повлияет на движение КА, если осуществить компенсацию гироскопического момента (k=1). Результат приведен на рис. 30.

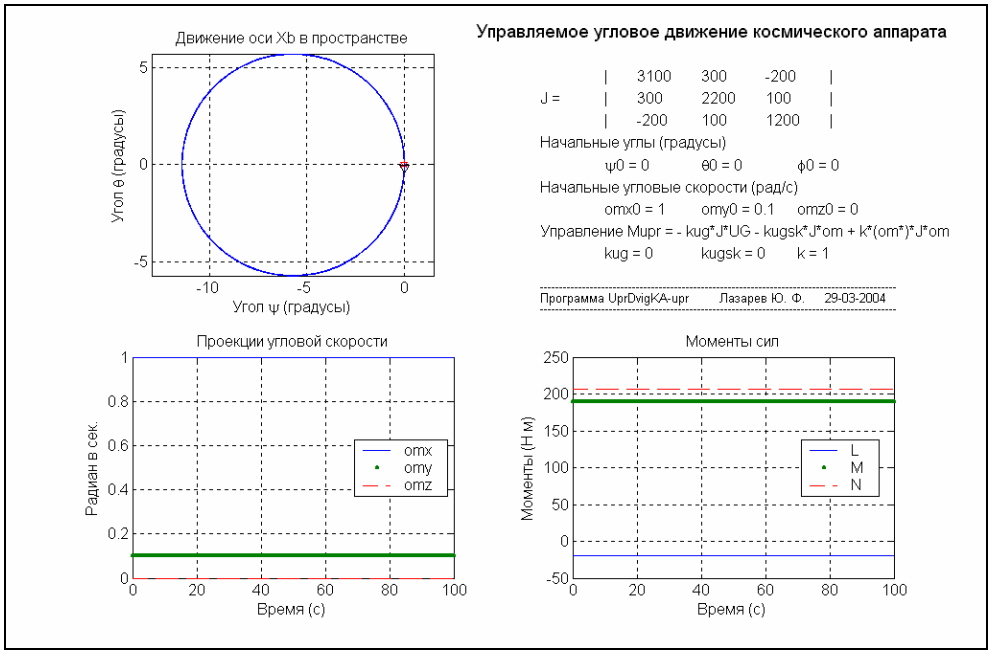


Рис. 9. 30. Влияние компенсации гироскопического момента

Результатом такого управления, как нетрудно убедиться, является значительное уменьшение (примерно в 5 раз) амплитуды колебаний оси X_b.

Теперь рассмотрим задачу демпфирования (точнее – приведения КА в неподвижное относительно инерциального пространства положение). Для этого введем управление (обратную связь) только по скоростям. Если, например, установить kugsk=0.1, а остальные коэффициенты управления равными нулю, то для динамически симметричного КА получим движение, изображенное на рис. 31. Если к демпфирующему моменту добавить компенсацию гироскопического момента, то движение КА будет происходить (рис. 32) со значительно меньшими отклонениями осей от начального положения.

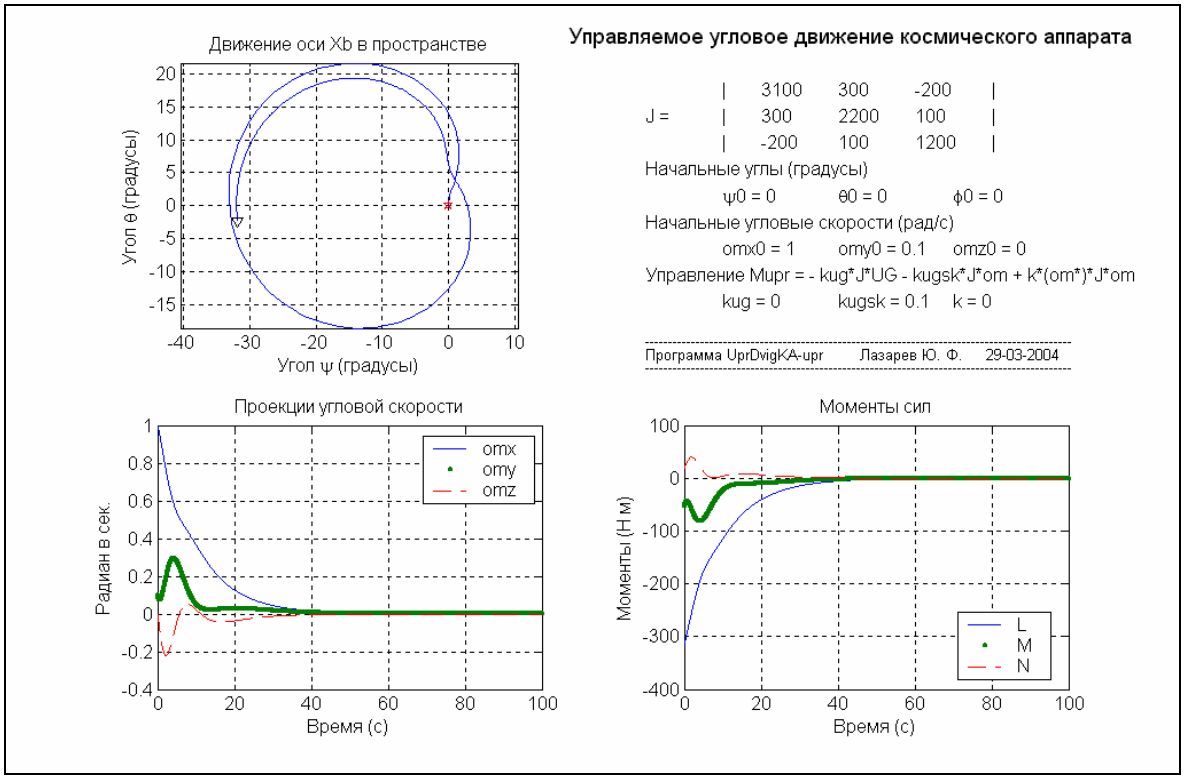


Рис. 9. 31. Режим "обездвиживания" КА

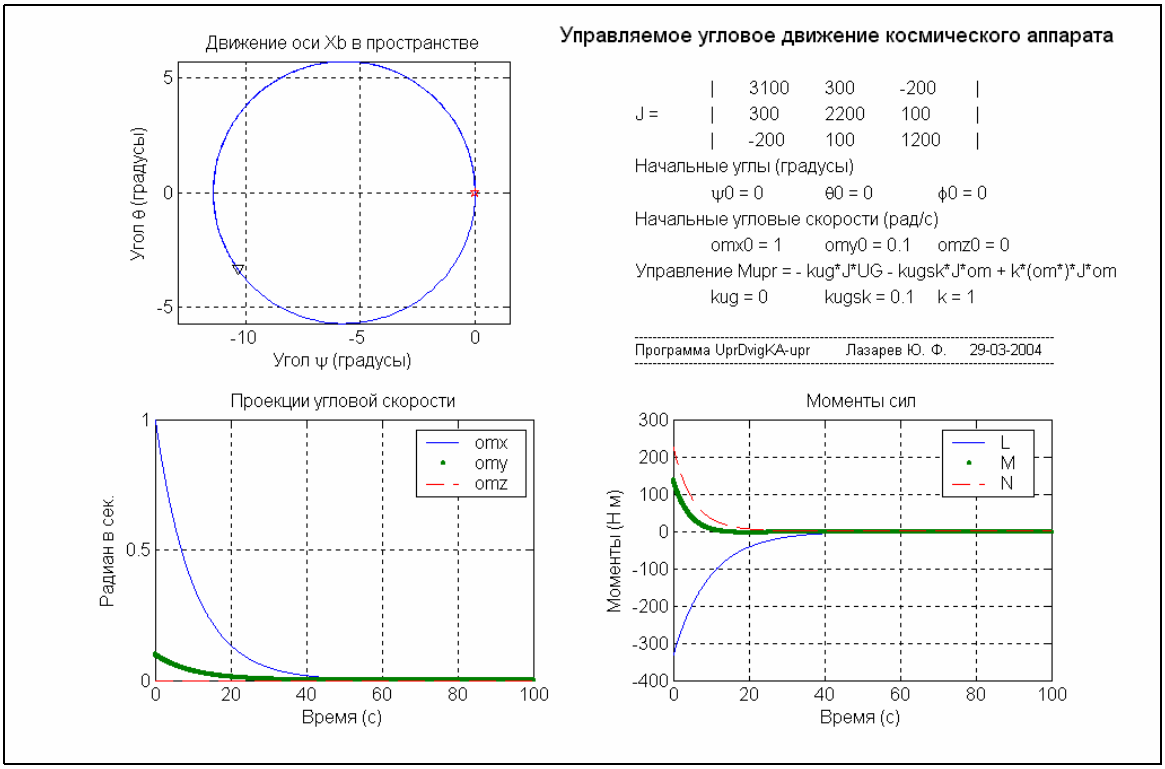


Рис. 9. 32. Режим "обездвиживания" КА с компенсацией гироскопического момента

Как видим, поставленная задача успешно решается. Побочным следствием такого управления является отклонение оси X_b от начального положения на значительные углы.

Перейдем теперь к режиму приведения положения КА к заданному положению, под которым будем понимать положение, когда все три угла будут равны нулю. Для этого, помимо обратной связи по скорости введем обратную связь по углам. Начальные отклонения по углам оси X_b установим равными 0.1 радиан. Тогда при kug=0.1 и kugsk=0.3 придем к результатам, показанным на рис. 33.

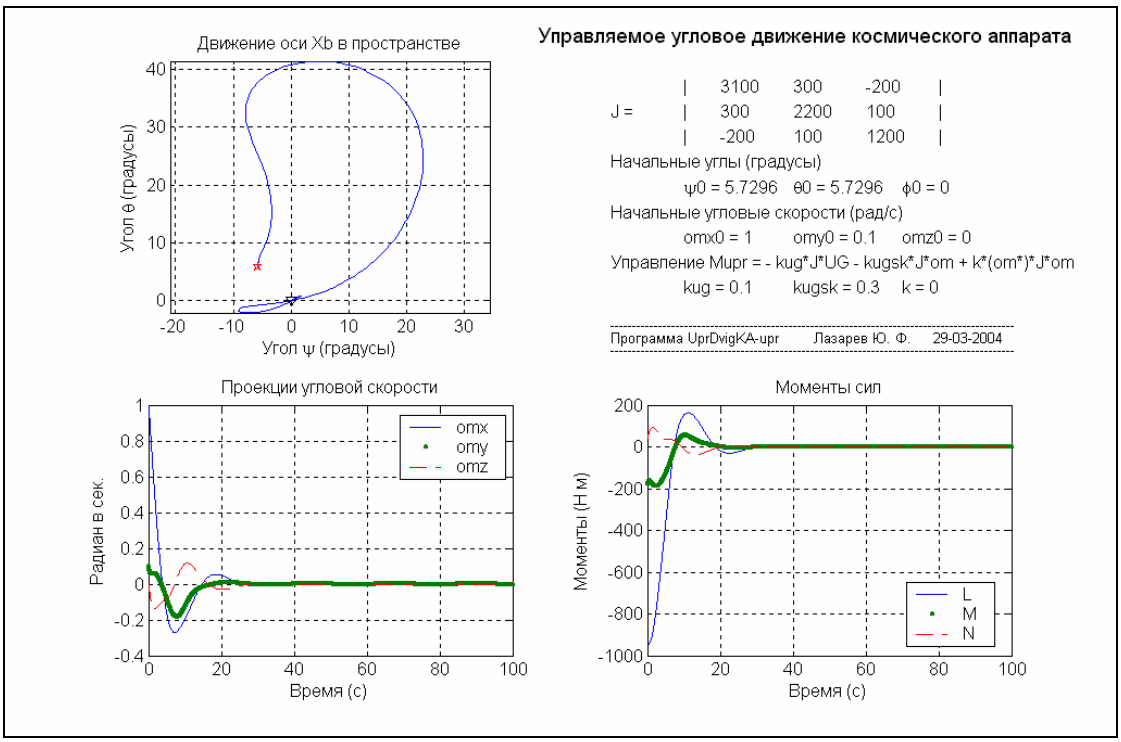


Рис. 9. 33. Режим стабилизации положения КА

Дополнительная компенсация гироскопического момента (рис. 34) и в этом случае улучшает процесс стабилизации.

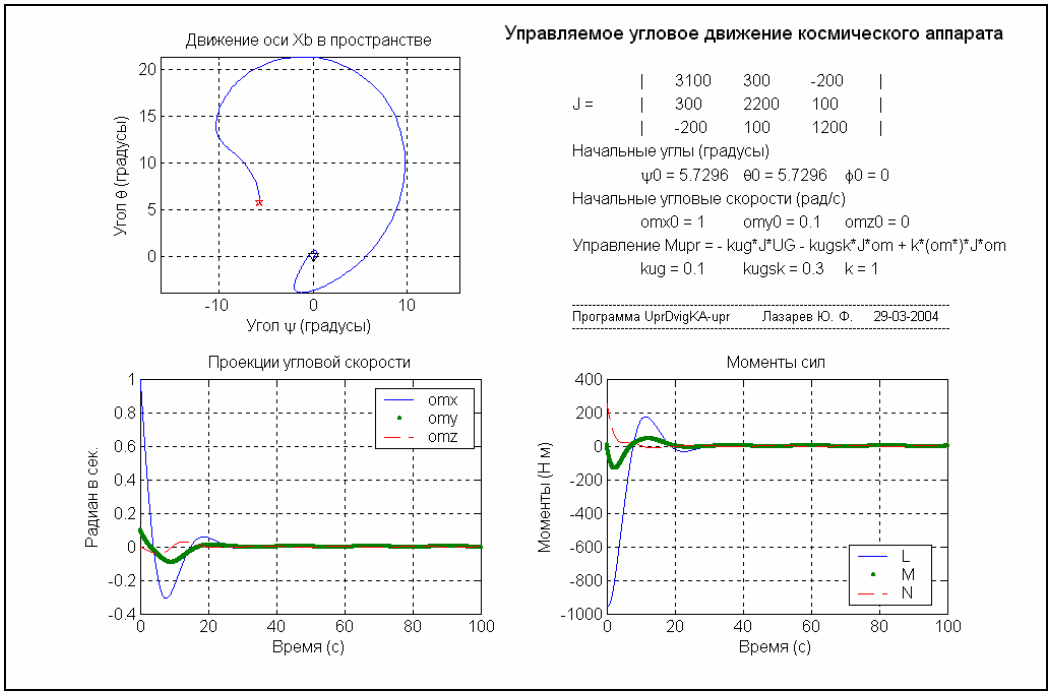


Рис. 9. 34. Режим стабилизации с компенсацией гироскопического момента

Управление по углам отклонения от заданного положения имеет ряд недостатков. К ним относится, прежде всего, отклонение осей поворотов углов (при больших углах) от осей приложения соответствующих управляющих моментов. Это может значительно затормозить процесс стабилизации и даже сделать его неустойчивым (при углах отклонения от заданного положения, больших 90 ^{o}).

Этого недостатка лишено управление по кватернионам, когда управляющие моменты по осям КА пропорциональны составляющим векторной части кватерниона отклонения текущего положения КА от заданного. Эти составляющие, во-первых пропорциональны не углам отклонений, а синусам половинных углов, во- вторых, эти составляющие кватерниона совпадают с осями, по которым направлены управляющие моменты, а это обеспечивает быстрейшее (и без потери устойчивости) приведение КА в заданное положение. Побочным, но тоже важным фактором является конечность величин управляющих моментов при любых значениях углов начального отклонения.

Модель UprDvigKAqw.mdl управляемого по кватернионам процесса ориентацией отличается от модели, приведенной на рис. 27, только содержимым блока «СУО», блок-схема которого для этого случая приводится на рис. 35.

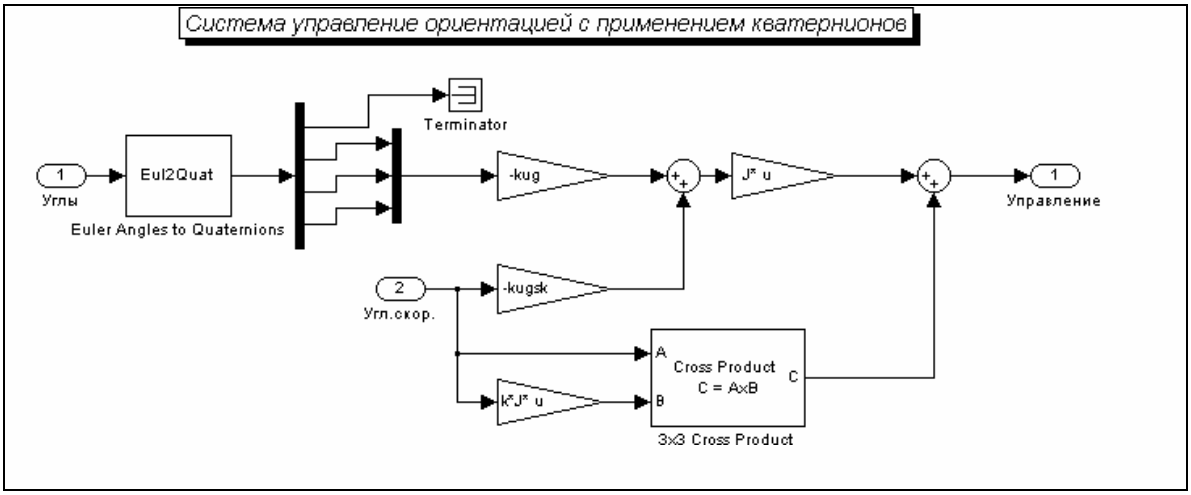


Рис. 9. 35. Блок-схема подсистемы «СУО» для управления по кватерниону

Управление работой модели и выводом результатов осуществляется программой **UprDvigKAqw1_upr**, текст которой приведен ниже.

```
% UprDvigKAqw1_upr
% Управляющая программа для модели UprDvigKA

% Лазарев Ю.Ф. 14-05-2004

clear all
clc
% Установка параметров КА
%J=[3100 0 0;0 2200 0;0 0 2200]; %   Матрица моментов инерции КА
%J=[3100 0 0;0 2200 0;0 0 1200]; %   Матрица моментов инерции КА
J=[3100 300 -200;300 2200 100;-200 100 1200]; %   Матрица моментов инерции КА
m=2000;                    %   Масса КА
% Задание коэффициентов управления
kug=0.1;      % К-нт обратной связи по углам
kugsk=0.3;    % К-нт обратной связи по угловым скоростям
k=0;       % К-нт компенсации гироскопического момента
% Установка начальных условий
XYZ0=[0 0 0];   %   Начальное положение КА
V0=[0 0 0];     %   Начальные скорости КА
UG0=[0 1 1];    %   Начальные углы КА
UgSk0=[0 0 0];    %   Начальные угловые скорости КА
% Установка параметров интегрирования
TK=100;          %   Конечное время интегрирования
hi=0.1;         %   Шаг интегрирования
% Запуск модели
sim('UprDvigKAqw');
%   Запись результатов интегрирования
Fl=yout(:,1);   TE=yout(:,2);   PSI=yout(:,3);
omx=yout(:,4);  omy=yout(:,5);  omz=yout(:,6);
t=tout;
L=yout(:,7);    M=yout(:,8);    N=yout(:,9);
% Графическое представление результатов
subplot(2,2,1)
plot(-PSI,TE,-PSI(1),TE(1),'pr',-PSI(end),TE(end),'kv'), grid
axis('equal');
set(gca,'FontSize',12)
```

```
title('Движение оси Xb в пространстве');
ylabel('Угол \theta (градусы)');
xlabel('Угол \psi (градусы)');
subplot(2,2,3)
plot(t,omx,t,omy,'.',t,omz,'--'), grid%plot(t,L,t,M,'.',t,N,'--'), grid
set(gca,'FontSize',12)
title('Проекции угловой скорости');%title('Моменты сил');
xlabel('Время (с)');
ylabel('Радиан в сек.');%ylabel('Моменты (Н м)');
legend(' omx ',' omy ',' omz ',0);%legend(' L ',' M ',' N ',0);
subplot(2,2,4)
plot(t,L,t,M,'.',t,N,'--'), grid
set(gca,'FontSize',12)
title('Моменты сил');
xlabel('Время (с)');
ylabel('Моменты (Н м)');
legend(' L ',' M ',' N ',0);
subplot(2,2,2)
axis('off');
h=text(-0.3,1.1,'Управляемое по кватернионам угловое движение КА','FontSize',14);
h=text(0.1,0.9,'| ','FontSize',12);
h=text(0.2,0.9,num2str(J(1,1)),'FontSize',12);
h=text(0.4,0.9,num2str(J(1,2)),'FontSize',12);
h=text(0.6,0.9,num2str(J(1,3)),'FontSize',12);
h=text(0.8,0.9,'| ','FontSize',12);
h=text(-0.1,0.8,'J = ','FontSize',12);
h=text(0.1,0.8,'| ','FontSize',12);
h=text(0.2,0.8,num2str(J(2,1)),'FontSize',12);
h=text(0.4,0.8,num2str(J(2,2)),'FontSize',12);
h=text(0.6,0.8,num2str(J(2,3)),'FontSize',12);
h=text(0.8,0.8,'| ','FontSize',12);
h=text(0.1,0.7,'| ','FontSize',12);
h=text(0.2,0.7,num2str(J(3,1)),'FontSize',12);
h=text(0.4,0.7,num2str(J(3,2)),'FontSize',12);
h=text(0.6,0.7,num2str(J(3,3)),'FontSize',12);
h=text(0.8,0.7,'| ','FontSize',12);
h=text(-0.1,0.6,'Начальные углы (градусы)','FontSize',12);
h=text(0.1,0.5,['\psi0 = ',num2str(UG0(3)*180/pi)],'FontSize',12);
h=text(0.4,0.5,['\theta0 = ',num2str(UG0(2)*180/pi)],'FontSize',12);
h=text(0.7,0.5,['\phi0 = ',num2str(UG0(1)*180/pi)],'FontSize',12);
h=text(-0.1,0.4,'Начальные угловые скорости (рад/с)','FontSize',12);
h=text(0.1,0.3,['omx0 = ',num2str(UgSk0(1))],'FontSize',12);
h=text(0.4,0.3,['omy0 = ',num2str(UgSk0(2))],'FontSize',12);
h=text(0.7,0.3,['omz0 = ',num2str(UgSk0(3))],'FontSize',12);
h=text(-0.1,0.2,'Управление Mupr = - kug*J*Qwat - kugsk*J*om + k*(om*)*J*om','FontSize',12);
h=text(0.1,0.1,['kug = ',num2str(kug)],'FontSize',12);
h=text(0.4,0.1,['kugsk = ',num2str(kugsk)],'FontSize',12);
h=text(0.7,0.1,['k = ',num2str(k)],'FontSize',12);
h=text(-0.1,-0.05,'-----------------------------------------------------------------------------------------------');
h=text(-0.1,-0.1,'Программа UprDvigKAqw1-upr       Лазарев Ю. Ф.     14-05-2004');
h=text(-0.1,-0.15,'-----------------------------------------------------------------------------------------------');
```

Результаты работы этой программы представлены на рис. 36. и 37. Коэффициенты управления приняты прежними, но углы отклонения большими в 10 раз (по одному радиану). Как и ранее рассмотрены движения без и с компенсацией гироскопического момента.

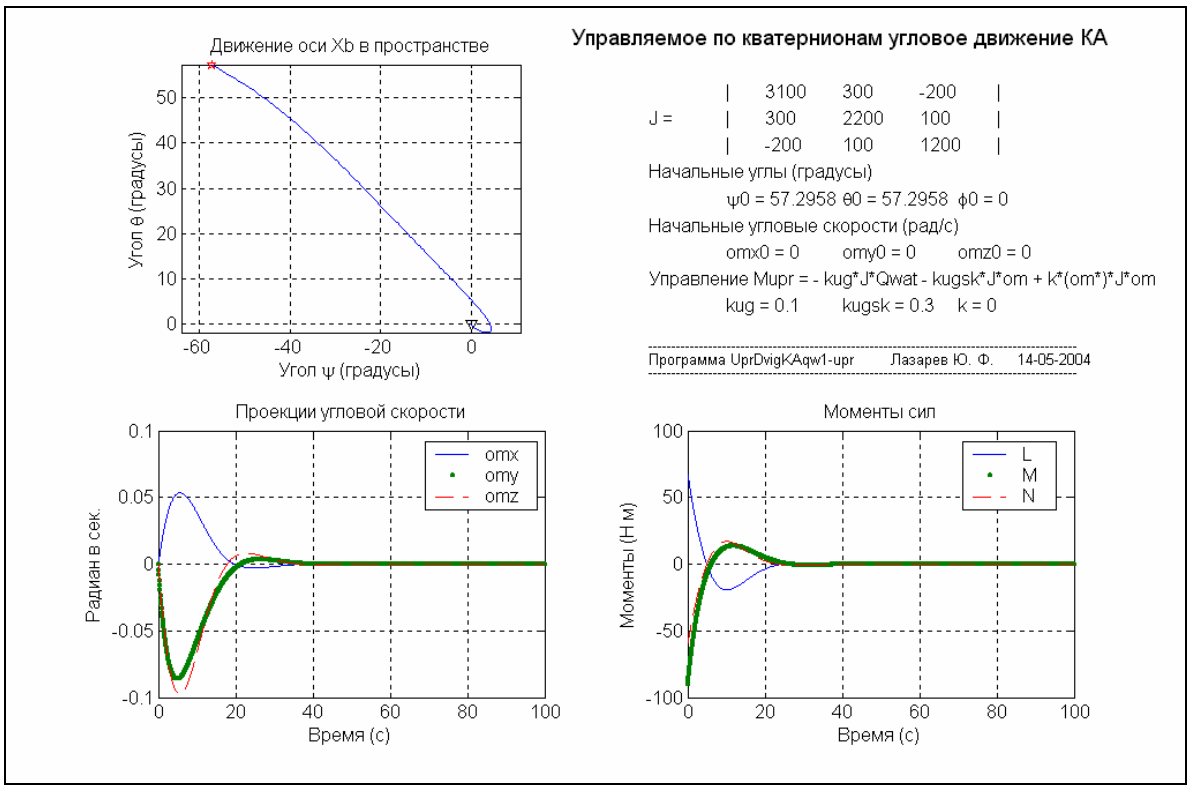


Рис. 9. 36. Управление ориентацией по кватерниону отклонения

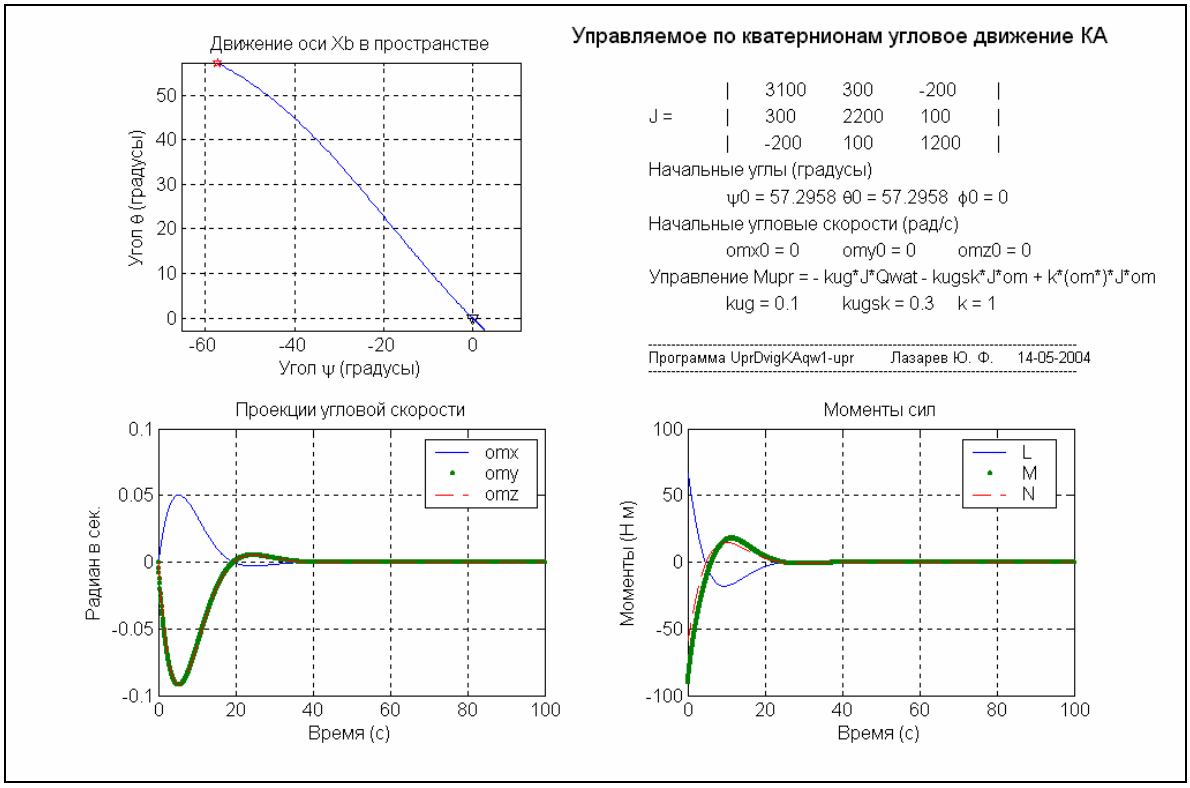


Рис. 9. 37. Управление по кватерниону с компенсацией гироскопического момента

Из приведенных результатов следует:

1) при управлении по кватерниону отклонения движение КА к заданному положению происходит по кратчайшему пути;

2) процесс установления в заданное положение устойчив при весьма больших начальных отклонениях;

3) длительность процесса приведения сохраняется такой же, как и при управлении по углам, хотя величина начального отклонения увеличена в 10 раз;

4) величины моментов управления значительно меньше требуемых для управления по углам;

5) по-прежнему, компенсация гироскопического момента приводит к улучшению динамики процесса ориентации.

9.4. Моделирование движения искусственного спутника Земли

Перейдем к составлению модели орбитального движения искусственного спутника Земли с учетом его угловой стабилизации.

Для этого введем следующие системы координат:

1) инерциальную (неподвижную) систему $X_eY_eZ_e$, начало которой поместим в центр Земли, ось Z_e направим перпендикулярно плоскости орбиты, ось X_e - вдоль линии, соединяющей центр Земли с начальным положением ИСЗ на орбите, а ось Y_e - в плоскости орбиты по вектору начальной скорости ИСЗ;

2) орбитальную систему $X_oY_oZ_o$, начало которой поместим в центр масс ИСЗ, ось Z_o направим параллельно оси Z_e, ось X_o - вдоль линии, соединяющей центр Земли с текущим положением ИСЗ на орбите, а ось Y_o - в плоскости орбиты вперед по движению ИСЗ по орбите;

3) связанную с телом ИСЗ систему $X_bY_bZ_b$, начало которой поместим также в центр масс ИСЗ.

ИСЗ как объект управления имеет шесть степеней свободы – три координаты положения центра масс в инерциальной системе координат и три угловые степени свободы, определяющие угловое положение тела спутника по отношению к инерциальной системе. Управление движением ИСЗ состоит из двух тесно связанных процессов – управления движением центра масс ИСЗ и управления его угловым положением. Первый из них был рассмотрен и промоделирован ранее. Для управления орбитальным движением ИСЗ важно ориентировать и стабилизировать угловое его положение в орбитальной системе координат (см. выше).

Возможный вариант UprDigISZqw1.mdl модели полного движения спутника приведен на рис. 38.

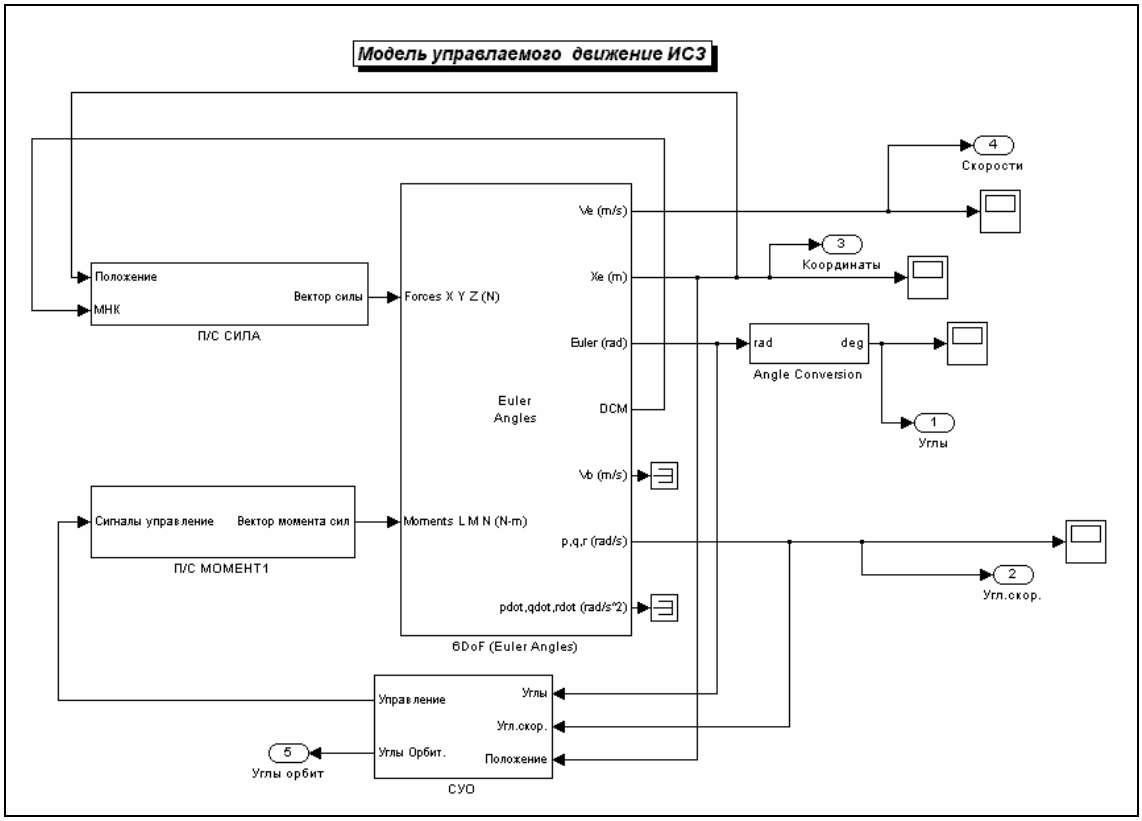


Рис. 9. 38. Блок-схема модели управляемого движения ИСЗ

Основным отличием этой модели от ранее представленных является наличие обратной связи (управления) по вектору силы, действующей на центр масс ИСЗ. Она реализована в виде подсистемы «П/С СИЛА», блок схема которой показана на рис. 39.

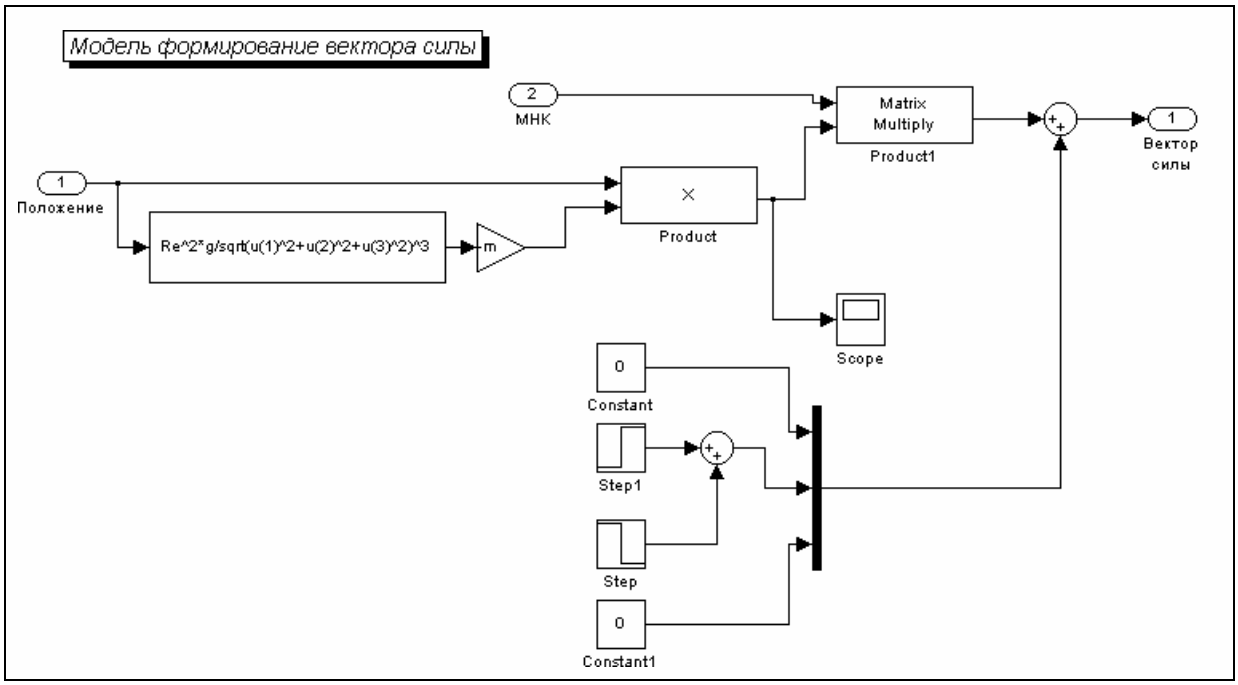


Рис. 9. 39. Блок-схема блока «П/С СИЛА»

Блок-схема подсистемы «СУО» (рис. 40) также изменена, ввиду того, что ИСЗ необходимо ориентировать не в инерциальной, а в орбитальной системе координат.

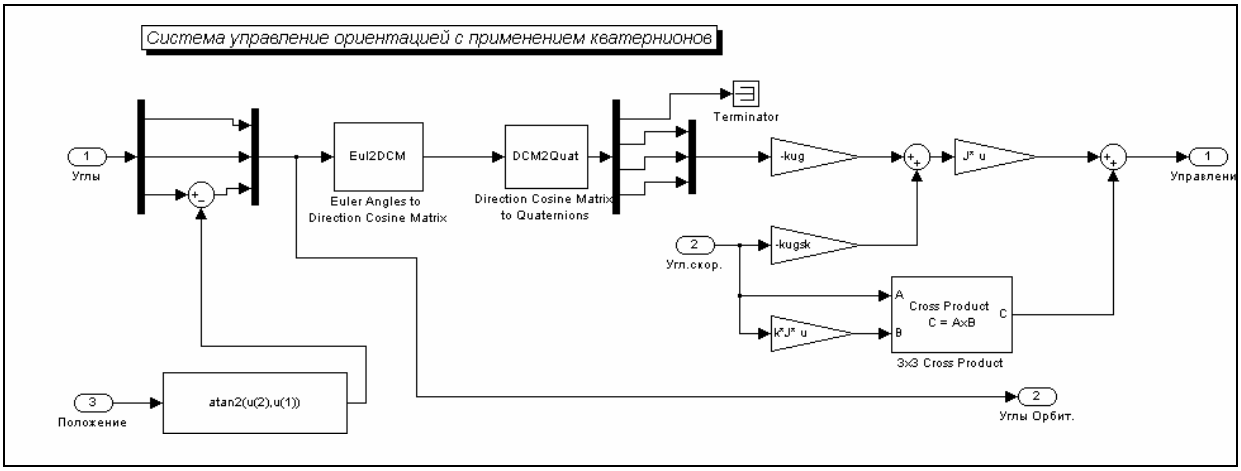


Рис. 9.40. Блок-схема блока «СУО»

Основным назначением блока «П/С СИЛА» является формирование вектора силы гравитации, действующей на ИСЗ со стороны Земли. Для этого вначале следует определить величину ускорения гравитации в той точке пространства, где расположен спутник. Это ускорение определяется выражением:

$$g_c = g\left(\frac{R}{r}\right)^2,$$

где g - ускорение гравитации на поверхности Земли, g_c - ускорение в точке, находящейся на расстоянии r от центра Земли, а R - радиус Земли. Теперь вектор $\mathbf{F}$ силы гравитации, действующей на ИСЗ можно определить из соотношения:

$$\mathbf{F} = -mg_c \frac{1}{r}\mathbf{r}.$$

Здесь m - масса ИСЗ, а $\mathbf{r}$ - радиус-вектор его центра масс.

Если известны текущие координаты центра масс X_e, Y_e, Z_e спутника в инерциальной системе, то

$$r = \sqrt{X_e^2 + Y_e^2 + Z_e^2},$$

и вычисление проекций вектора $\mathbf{F}$ в инерциальной системе можно свести к формуле

$$\mathbf{F} = -mg_c = -mg \frac{R^2}{(\sqrt{X_e^2 + Y_e^2 + Z_e^2})^3}\mathbf{r}.$$

Именно эти операции и обеспечены в блоке «П/С СИЛА» (см. рис. 39). Так как на вход блока 6DoF следует подать вектор силы через его проекции на связанные с ИСЗ оси, то далее полученный вектор силы в инерциальной системе преобразуется в вектор проекций этой силы на связанную систему координат путем умножения на матрицу направляющих косинусов ИСЗ в инерциальной системе.

Вторая задача блока – сформировать силу активного воздействия вдоль оси Y_b связанной системы для осуществления перехода на другую орбиту. Эта цель достигается нижней ветвью в блок-схеме рис. 39. Здесь формируется постоянная по величине сила, начинающая свое действие в заданный момент времени и действующая заданный промежуток времени. При этом использованы следующие обозначения:

Tn Начальный момент времени действия силы коррекции орбиты

tau Промежуток времени, в течение которого действует сила коррекции

DF Величина силы коррекции

Блок «СУО» в представленной модели (рис. 37) отличается от аналогичного блока в предыдущей модели (рис. 35), прежде всего, наличием части, которая вычисляет угловое отклонение связанной с ИСЗ системы координат от орбитальной. Это необходимо для обеспечения управлением ориентацией спутника относительно не инерциальной, а орбитальной системы. Помимо этого, вместо блока непосредственного преобразования углов Эйлера в кватернион поворота используется последовательное преобразование углов Эйлера в матрицу направляющих косинусов, а затем последней в кватернион поворота. Это позволяет избавиться от разрывов в угловых координатах при переходе значений углов через 180 ^{o}.

Текст управляющей программы **UprDviglSZqw1_upr** приведен ниже.

```
% UprDviglSZqw1_upr
% Управляющая программа для модели SvDvigKA

% Лазарев Ю.Ф. 15-05-2004

clear all,  clc
OM=7.292115e-5; Re=6.37814e6;   g=9.78;
% Установка параметров КА
%J=[1800 0 0;0 3400 0;0 0 1800]; %   Матрица моментов инерции КА
%J=[3400 0 0;0 2200 0;0 0 1400]; %   Матрица моментов инерции КА
J=[2200 300 -200;300 3400 100;-200 100 1400]; %   Матрица моментов инерции КА
m=2000;                    %   Масса КА
% Установка начальных условий
XYZ0=[6.4e6 0 0];   %   Начальное положение КА
V0=[0 8e3 0];     %   Начальные скорости КА
UG0=[0 1 0];    %   Начальные углы КА
UgSk0=[0 0.2 0];  %   Начальные угловые скорости КА
% Задание коэффициентов управления
kug=0.1;      % К-нт обратной связи по углам
kugsk=0.3;    % К-нт обратной связи по угловым скоростям
k=1;       % К-нт компенсации гироскопического момента
Tn=3600*2;  tau=1000;    DF=2000;
% Установка параметров интегрирования
TK=6*3600;         %   Конечное время интегрирования
hi=0.5;        %   Шаг интегрирования
% Запуск модели
sim('UprDviglSZqw1');
%   Запись результатов интегрирования
Fl=yout(:,1);   TE=yout(:,2);   PSl=yout(:,3);
omx=yout(:,4);  omy=yout(:,5);  omz=yout(:,6);
Xe=yout(:,7);   Ye=yout(:,8);   Ze=yout(:,9);
Vex=yout(:,10);  Vey=yout(:,11);  Vez=yout(:,12);
Flo=yout(:,13);   TEo=yout(:,14);   PSlo=unwrap(yout(:,15));
t=tout; n=length(t);
k1=0;
for ks=1:n
   if t(ks)<=Tn
      k1=k1+1;    t1(k1)=t(ks);
```

```
   end
end
K1=k1;  Xe1=Xe(1:K1);   Ye1=Ye(1:K1);  k1=0;
for ks=1:n
   if (t(ks)>Tn)&(t(ks)<=Tn+tau)
      k1=k1+1;    t2(k1)=t(ks);
   end
end
K2=k1;  Xe2=Xe(K1+1:K1+K2);   Ye2=Ye(K1+1:K1+K2);   k1=0;
for ks=1:n
   if t(ks)>Tn+tau
      k1=k1+1;    t3(k1)=t(ks);
   end
end
K3=k1;  Xe3=Xe(K1+K2+1:K1+K2+K3);   Ye3=Ye(K1+K2+1:K1+K2+K3);
% Графическое представление результатов
subplot(2,2,1)
plot(Xe1*1e-6,Ye1*1e-6,'--',Xe3*1e-6,Ye3*1e-6,'.',...
   Xe2*1e-6,Ye2*1e-6,'*',0,0,'o',XYZ0(1)*1e-6,XYZ0(2)*1e-6,'pk'), grid
axis('equal');
title('Движение ИСЗ в плоскости X - Y');
xlabel('Координата X (тыс. км)');   ylabel('Координата Y (тыс. км)');
legend(' орбита 1 ',' орбита 2 ',' активный ',0);
subplot(2,2,3)
plot(t1/3600,Xe1*1e-6,'k--',t2/3600,Xe2*1e-6,'k*',t3/3600,Xe3*1e-6,'k.',...
   t1/3600,Ye1*1e-6,'b--',t2/3600,Ye2*1e-6,'b*',t3/3600,Ye3*1e-6,'b.'), grid
title('Инерциальные координаты');
xlabel('Время (часы)'); ylabel('Координаты (тыс. км)');
legend(' орбита 1 ',' активный ',' орбита 2 ',0);
subplot(2,2,4)
plot(t(1:80),PSI(1:80),'*',t(1:80),TE(1:80),t(1:80),PSIo(1:80)*180/pi,'.',...
   t(1:80),TEo(1:80)*180/pi), grid
title('Процесс угловой стабилизации'); xlabel('Время (с)'); ylabel('Углы (градусы)');
legend(' \psi ',' \theta ',' \psio ',' \thetao ',0);
subplot(2,2,2)
axis('off');
h=text(0.0,1.1,'Управляемое движение ИСЗ','FontSize',14);
h=text(0.3,0.95,'| ');  h=text(0.4,0.95,num2str(J(1,1)));
h=text(0.6,0.95,num2str(J(1,2))); h=text(0.8,0.95,num2str(J(1,3)));
h=text(1.0,0.95,'| '); h=text(0.1,0.9,'J = ');
h=text(-0.3,0.9,['m = ',num2str(m)]);
h=text(0.3,0.9,'| '); h=text(0.4,0.9,num2str(J(2,1)));
h=text(0.6,0.9,num2str(J(2,2))); h=text(0.8,0.9,num2str(J(2,3)));
h=text(1.0,0.9,'| '); h=text(0.3,0.85,'| ');
h=text(0.4,0.85,num2str(J(3,1))); h=text(0.6,0.85,num2str(J(3,2)));
h=text(0.8,0.85,num2str(J(3,3))); h=text(1.0,0.85,'| ');
h=text(-0.1,0.75,'Начальное положение:'); h=text(-0.3,0.7,['Xe0 = ',num2str(XYZ0(1)*1e-6)]);
h=text(0.0,0.7,['Ye0 = ',num2str(XYZ0(2)*1e-6)]);
h=text(0.3,0.7,['Ze0 = ',num2str(XYZ0(3)*1e-6)]); h=text(0.7,0.7,'(тыс. км) ');
h=text(-0.3,0.65,['\psi0 = ',num2str(UG0(3)*180/pi)]);
h=text(0.0,0.65,['\theta0 = ',num2str(UG0(2)*180/pi)]);
h=text(0.3,0.65,['\phi0 = ',num2str(UG0(1)*180/pi)]); h=text(0.7,0.65,'(градусы) ');
h=text(-0.1,0.55,'Начальные скорости:');
h=text(-0.3,0.5,['Vex0 = ',num2str(V0(1)*1e-3)]); h=text(0.0,0.5,['Vey0 = ',num2str(V0(2)*1e-3)]);
h=text(0.3,0.5,['Vez0 = ',num2str(V0(3)*1e-3)]); h=text(0.7,0.5,'(км/с) ');
h=text(-0.3,0.45,['omx0 = ',num2str(UgSk0(1))]); h=text(0.0,0.45,['omy0 = ',num2str(UgSk0(2))]);
h=text(0.3,0.45,['omz0 = ',num2str(UgSk0(3))]); h=text(0.7,0.45,'(рад/с) ');
h=text(-0.1,0.35,'К-нты управления ориентацией:'); h=text(-0.3,0.3,['kug = ',num2str(kug)]);
h=text(0.0,0.3,['kugsk = ',num2str(kugsk)]); h=text(0.3,0.3,['k = ',num2str(k)]);
h=text(-0.1,0.2,['Начало активного участка Tn = ',num2str(Tn),'  сек']);
h=text(-0.1,0.1,['Длительность активного участка tau = ',num2str(tau),'  сек']);
h=text(-0.1,0.0,['Тяга F = ',num2str(DF),'  H']);
h=text(-0.1,-0.05,'-------------------------------------------------------------------------------------------');
h=text(-0.1,-0.1,'Программа UprDviglSZqw1-upr       Лазарев Ю. Ф.     15-05-2004');
h=text(-0.1,-0.15,'-------------------------------------------------------------------------------------------');
```

На рис. 41 отображены результаты работы программы и модели.

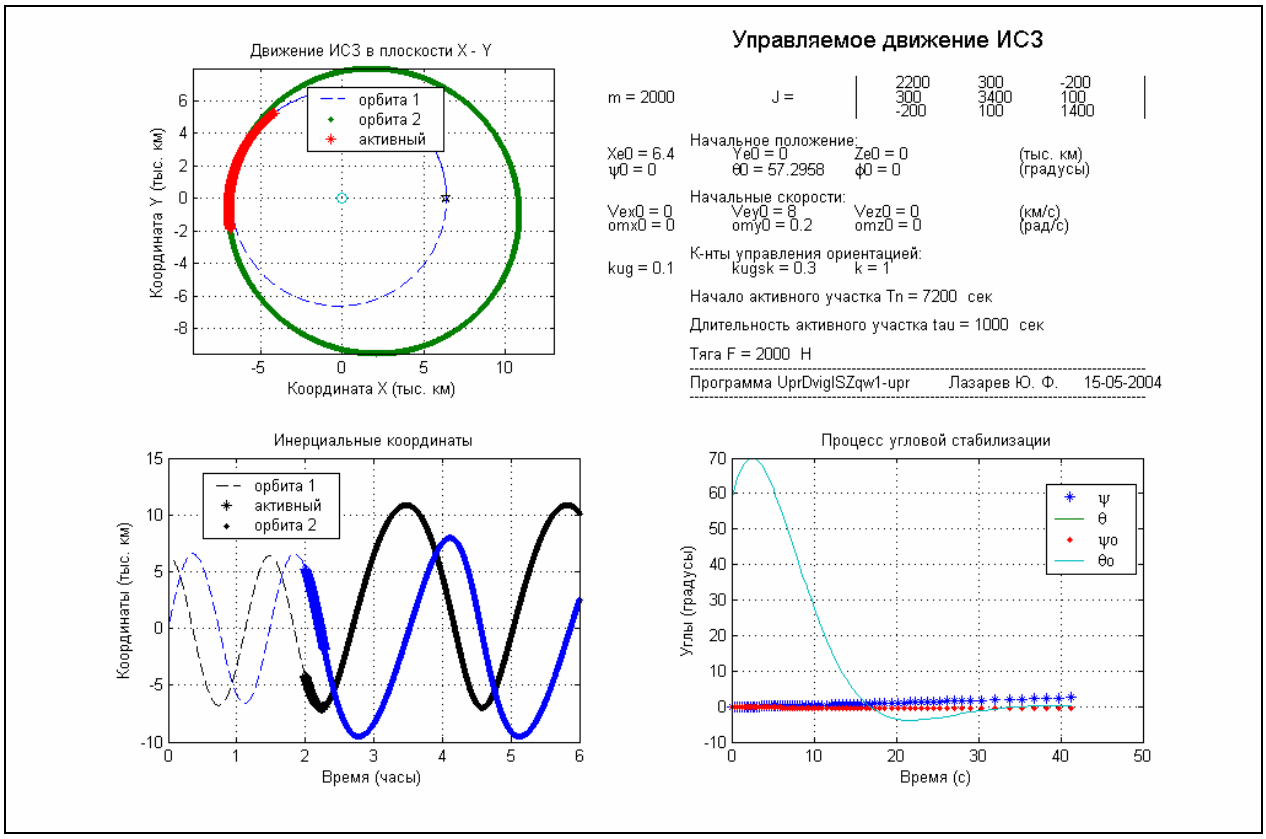


Рис. 9.41. Маневрирование ИСЗ

Полученные результаты подтверждают достаточно хорошие свойства модели, позволяющие исследовать достаточно сложные процессы управления движением ИСЗ.

9.5. Вопросы для самопроверки

1. Каково основное назначение библиотеки **Aerospace**? из каких разделов она состоит?

2. Каково основное назначение блоков раздела Equations of Motion? в чем различие между блоками этого раздела?

3. Для чего предназначены блоки раздела Environment?

4. Какие группы силовых воздействий определяют движение летательного аппарата?

5. Для чего служат блоки раздела Propulsion?

6. Каково основное назначение блоков разделов Actuators и GNC?

7. Какие задачи решают блоки раздела Transfomations?

Урок 10. Моделирование электроэнергетических систем (библиотека SimPowerSystems)

Библиотека **SimPowerSystems** предназначена для моделирования поведения электрических силовых систем, представляющих собой комбинации электрических цепей и электромеханических устройств, таких как электрические двигатели и генераторы. Библиотека функционирует в составе пакета **Simulink** в среде Matlab и содержит модели типовых устройств силовой электроэнергетики, таких как трансформаторы, преобразователи, линии электропередач, электромашины и элементы силовой электроники.

Идеология составления блок-схемы модели в библиотеке **SimPowerSystems** существенно отличается от идеологии составления блок-схем в S-моделях. В S-моделях соединяемые блоки представляют собой программы математического преобразования входных величин блока в выходные величины независимо от их физического содержания. В блок-схемах **SimPowerSystems** соединения блоков **SimPowerSystems** обычно следует рассматривать как имитацию электрических соединений, линии соединения – как проводную связь, осуществляющую передачу электрического сигнала (тока) от выхода одного блока к входу следующего блока, а сами блоки библиотеки **SimPowerSystems** – как модели электрических процессов, протекающих в устройстве, поведение которого моделируется.

Из этого вытекает, что блоки S-модели, в общем случае, не могут быть соединены с большинством блоков библиотеки **SimPowerSystems**. В частности, нельзя непосредственно использовать блоки библиотеки **SIMULINK** для формирования электрических сигналов заданной формы, нельзя непосредственно вывести значения токов и напряжений в обзорные окна Scope и на выходные порты и т. п. Тем не менее, модели библиотеки **SimPowerSystems** функционируют в среде **Simulink**, и потому им должны быть доступны все возможности этой среды, предоставляемые библиотекой **SIMULINK**.

В дальнейшем для простоты будем использовать следующую терминологию. Будем называть S-блоками блоки библиотеки **SIMULINK**, P-блоками - блоки библиотеки **SimPowerSystems**, линии, соединяющие S-блоки, - m-линиями, входы и выходы S-блоков - m-входами и m-выходами, линии, соединяющие P-блоки - p-линиями, входы и выходы P-блоков - p-входами и p-выходами. Напомним, что m-линии переносят сигналы-функции, независимо от их физической природы, а p-линии – электрические сигналы и являются аналогом идеальной проводной связи. Аналогично, m-входы и m-выходы воспринимают или генерируют функциональные сигналы, а p-входы и p-выходы реализуют электрическое соединение проводника p-линии с соответствующим P-блоком.

Для связи P-блоков с обычными S-блоками предназначены лишь некоторые блоки библиотеки **SimPowerSystems**, такие как блоки-измерители электрических сигналов (тока, напряжения и т. п.), которые имеют p-входы и m-выход, некоторые блоки источников электрических сигналов (они имеют m-вход и p-выходы), а также некоторые другие блоки. Именно эти блоки позволяют использовать все богатейшие возможности библиотеки **SIMULINK** для моделирования электроэнергетических систем и, в частности, использовать возможности программирования в MatLab процессов ввода, преобразования и вывода (в том числе – в графической форме) информации.

10.1. Общая характеристика библиотеки SimPowerSystems

В браузере `Simulink` с помощью контекстного меню библиотеки **SimPowerSystems** вызывается окно powerlib2 этой библиотеки (рис. 1).

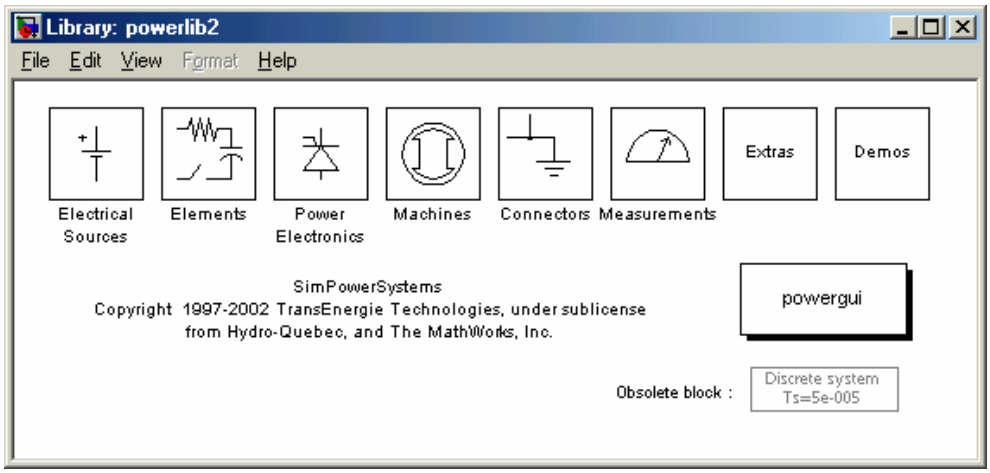


Рис. 10. 1. Состав библиотеки powerlib2

В библиотеку входят девять разделов:

`Electrical Sources` (Источники электричества) содержит блоки, моделирующие источники тока и напряжения

`Elements` (Элементы электрических цепей) включает блоки, моделирующие электрические

	процессы в элементарных электрических элементах (RLC-цепи, реальная проводная связь, электрическая нагрузка, трансформаторы и т. п.)
Power Electronics	(Элементы силовой электроники) содержит блоки, моделирующие поведение коммутирующих элементов силовой электроники (диоды, тиристоры, мосты, управляемые ключи и т. п.)
Machines	(Машины) содержит блоки-модели электрических машин различных типов
Connectors	(Соединители) состоит из блоков, моделирующих электрические соединения различных типов
Measurements	(Измерительные элементы) содержит блоки, имитирующие измерительные электрические приборы (амперметры, вольтметры, омметры и т. п.)
Extras	Библиотека дополнительных блоков
Demos	Демонстрационные программы-модели
Powergui	Интерактивный блок окна графического интерфейса пользователя

Раздел *Electrical Sources*

Содержимое раздела **Electrical Sources** (Источники электричества) представлено на рис. 2.

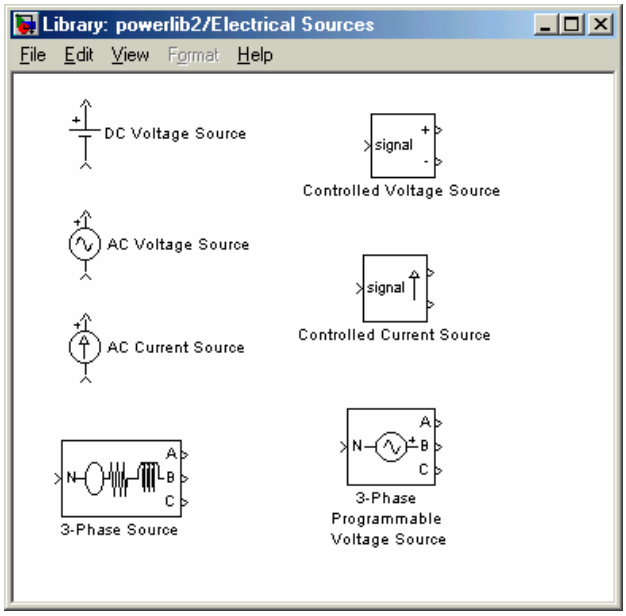


Рис. 10. 2. Содержимое раздела **Electrical Sources**

В него входят 7 блоков.

DC Voltage Source	Источник постоянного напряжения
AC Voltage Source	Источник переменного (синусоидального) напряжения
AC Current Source	Источник переменного (синусоидального) тока
Controlled Voltage Source	Источник управляемого напряжения
Controlled Current Source	Источник управляемого тока
3-Phase Source	Трехфазный источник
3-Phase Programmable Voltage Source	Трехфазный программируемый источник напряжения

Блок **DC Voltage Source** имитирует работу идеального источника постоянного напряжения. Единственный параметр настраивания (рис. 3) – величина напряжения на клеммах источника.

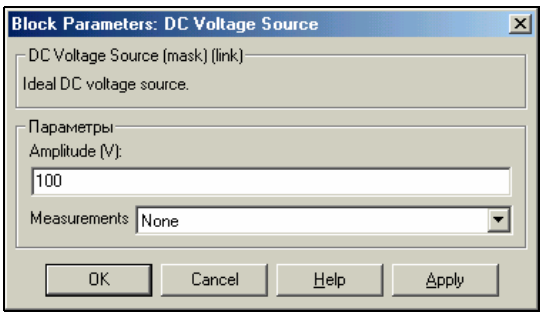
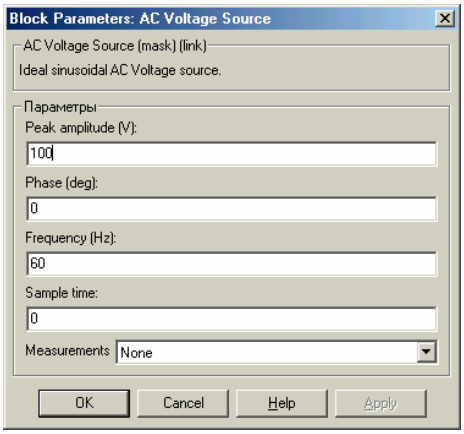


Рис. 10. 3. Окно настраивания блока **DC Voltage Source**

Блоки **AC Voltage Source** и **AC Current Source** имитируют работу источников синусоидального напряжения и тока соответственно. Окна настраивания блоков (рис. 4 и 5) почти не отличаются.



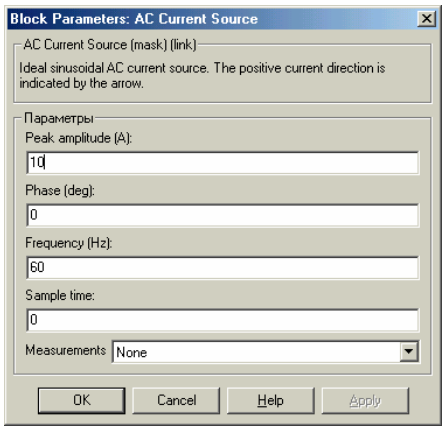


Рис. 10. 4. Окно настраивания блока **AC Voltage Source**

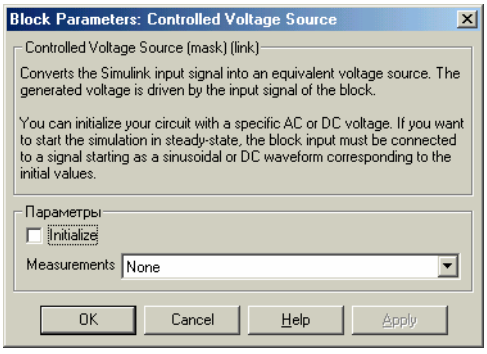
Рис. 10. 5. Окно настраивания блока **AC Current Source**

В параметры настраивания входят такие величины:

Pick amplitude (V)	Амплитуда синусоидального напряжения в вольтах
Pick amplitude (A)	Амплитуда синусоидального тока в амперах
Phase (deg)	Начальная фаза в градусах
Frequency (Hz)	Частота изменения тока (напряжения) в герцах

Вход и выход обоих блоков – это клеммы подсоединения источников к элементам электрической схемы.

Два блока **Controlled Voltage Source** (Управляемый источник напряжения) и **Controlled Current Source** (Управляемый источник тока) позволяют создать источник с произвольным законом изменения напряжения или тока во времени, используя блоки раздела Sources библиотеки **SIMULINK**. В отличие от рассмотренных ранее блоков оба блока имеют m-вход, к которому могут быть подсоединены выходы любого S-блока. На рис. 6 и 7 приведены окна настраивания этих блоков. Они вполне идентичны.



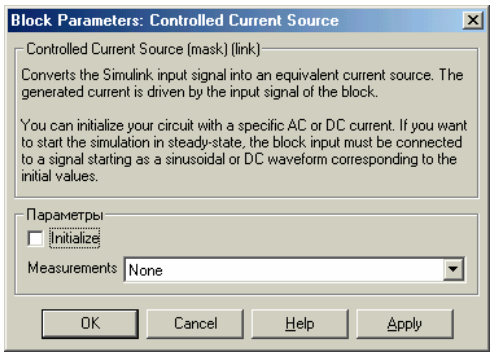


Рис. 10. 6. Окно настраивания блока **Controlled Voltage Source**

Рис. 10. 7. Окно настраивания блока **Controlled CurrentSource**

Блоки вырабатывают на выходных клеммах напряжение или ток, равный тому S-сигналу, который поступает на его вход.

На рис. 8 приведена схема моделирования трех видов источников напряжения.

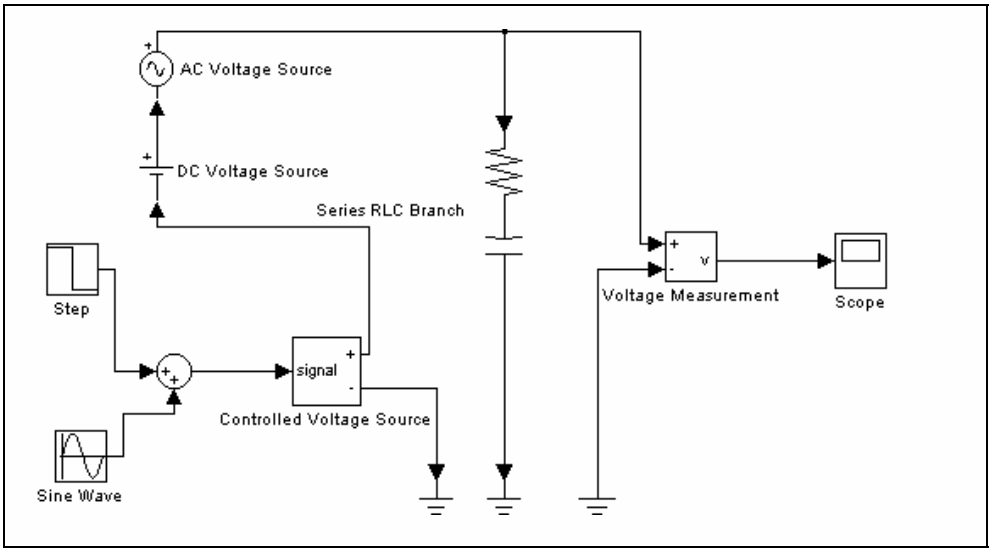


Рис. 10. 8. Схема моделирования работы трех видов источников напряжения

На схеме последовательно соединены три блока источников напряжения:
- блок **AC Voltage Source**, на котором установлена амплитуда 100 В и частота 50 Гц;
- блок **DC Voltage Source**, где установлено постоянное напряжение 50 В;
- блок управляемого источника напряжения **Controlled Voltage Source**, на вход которого (m-вход) подается сумма двух сигналов – гармонического с частотой 20 радиан в секунду и амплитудой 50 В и скачкообразного сигнала, равного 0 до момента времени 0,1 секунды и -200 В – после этого момента.

Эти три источника напряжения замкнуты на RC- цепь (R=1 Ом, C=10^{-6}Ф). Общее напряжение измеряется с помощью измерительного блока **Voltage Measurement** (Вольтметр), выход которого является m- выходом, к которому подключен блок Scope.

Результат моделирования, отображаемый в блоке Scope, представлен на рис. 9.

Нетрудно видеть, что суммарное напряжение, как и следовало ожидать, равно сумме напряжений указанных трех источников.

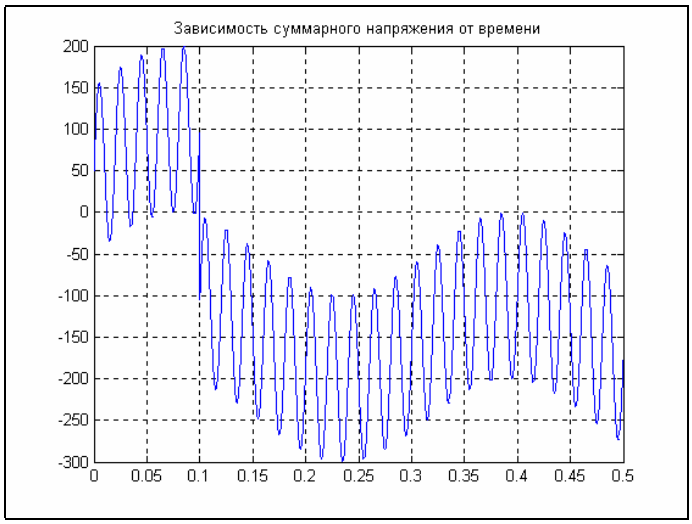
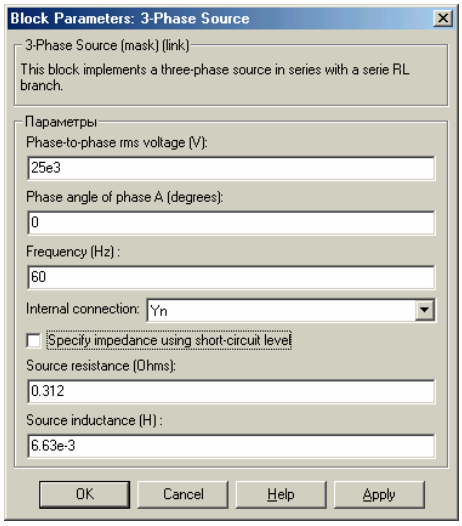


Рис. 10. 9. Результат моделирования по схеме рис. 10. 8

Два последних блока (**3-Phase Source** и **3-Phase Programmable Voltage Source)** имитируют работу трехфазных источников напряжения. В обоих блоках входом является нейтраль (N), а выходов три – фазы A, B и C трехфазного напряжения. Окна настраивания их показаны на рис. 10 и 11.



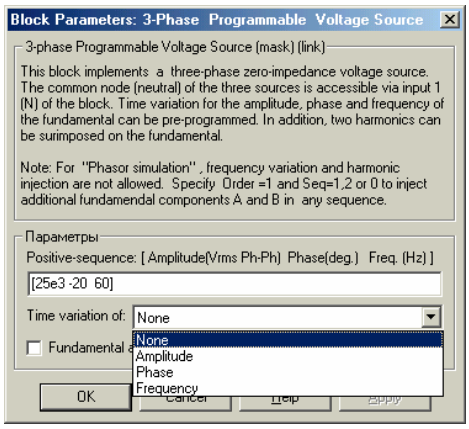


Рис. 10. 10. Окно настраивания блока **3-Phase Source**

Рис. 10. 11. Окно настраивания блока **3-Phase Programmable Voltage Source**

Общими параметрами настраивания являются:

Phase to phase voltage напряжение между фазными клеммами в Вольтах

Phase angle of phase A начальный фазовый угол фазы A в градусах

Frequency частота изменения напряжения в Герцах

В блоке **3-Phase Source** моделируется трехфазный источник с внутренней последовательной RL-цепью, параметры которой задают два остальных параметра настраивания

Source resistance активное сопротивление источника в Омах

Source inductance индуктивность источника в Генри

В блоке **3-Phase Programmable Voltage Source** имитируется источник с внутренним импедансом, равным нулю. Блок дополнительно позволяет модулировать (амплитудно, частотно или фазно) напряжение источника (см. рис. 11). Для этого в число параметров настраивания входит параметр **Time variation of**, который имеет следующий список: **None, Amplitude, Phase, Frequency**, обозначающие вид модуляции напряжения. При выборе одного из видов модуляции появляется дополнительный список, позволяющий установить величины амплитуды (в соответствующих единицах – Вольтах, градусах или Герцах), частоты и фазового угла огибающей.

Раздел *Elements*

В раздел Elements входят четыре группы блоков (рис. 12).

Elements (Элементы) содержит блоки, имитирующие элементарные последовательные и параллельные RLC-цепи

Lines (Линии) содержит блоки, имитирующие линии электропередач

Circuit Breakers (Прерыватели тока) содержит блоки, имитирующие различного рода прерыватели тока

Transformers (Трансформаторы) содержит блоки, имитирующие трансформаторы

Первая группа (**Elements)** состоит из таких блоков:

Series RLC Branch последовательная RLC-цепь

Series RLC Load последовательная RLC-цепь нагрузки

Parallel RLC Branch параллельная RLC-цепь

Parallel RLC Load параллельная RLC-цепь нагрузки

3-Phase Series RLC Branch	трехфазная последовательная RLC-цепь
3-Phase Series RLC Load	трехфазная последовательная RLC-цепь нагрузки
3-Phase Parallel RLC Branch	трехфазная параллельная RLC-цепь
3-Phase Parallel RLC Load	трехфазная параллельная RLC-цепь нагрузки
Mutual Inductance	блок взаимной индуктивности
3-Phase Mutual Inductance	блок трехфазной взаимной индуктивности
3-Phase Dynamic Load	трехфазная динамическая нагрузка
Surge Arrester	Ограничитель пиковых напряжений

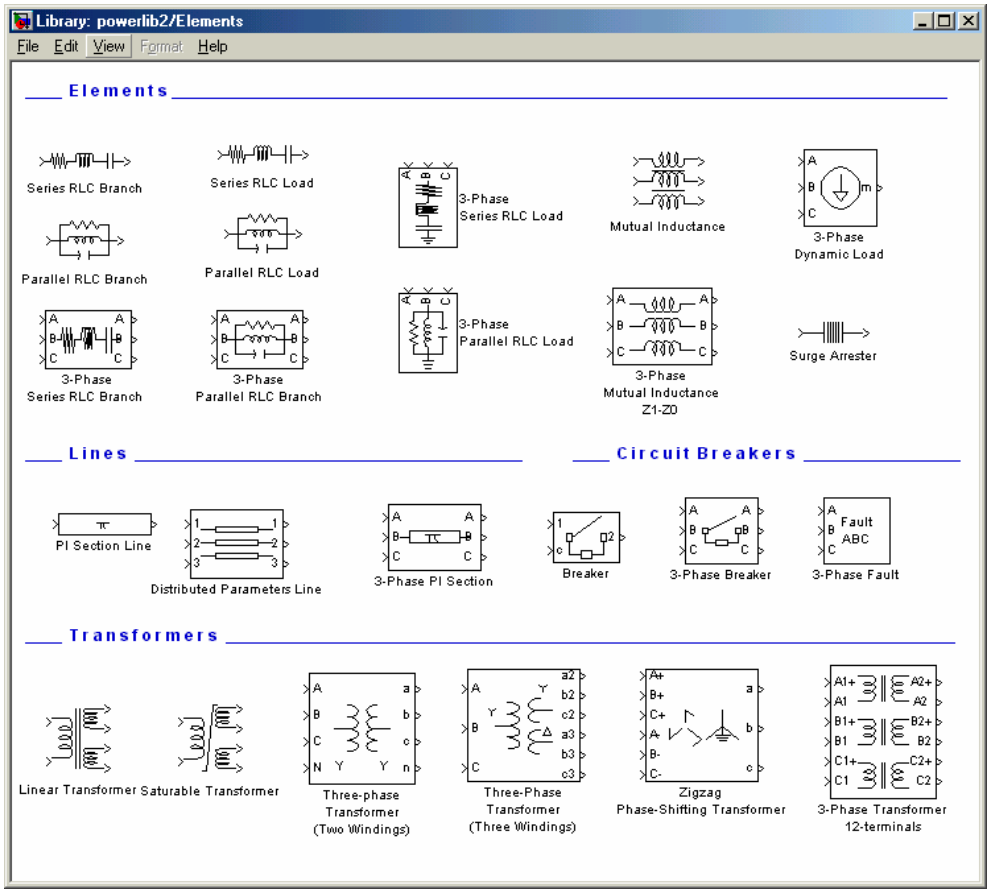


Рис. 10. 12. Содержимое раздела Elements

Обычные RLC-цепи (блоки **Series RLC Branch, Parallel RLC Branch, 3-Phase Series RLC Branch** и **3-Phase Parallel RLC Branch**) задаются тремя параметрами – сопротивлением R, индуктивностью L и емкостью C. Для ввода отдельных элементов (резистора, конденсатора или индуктивности) можно использовать любой из этих блоков, задав в нем параметры, соответствующие отсутствию остальных элементов. Например, для задания резистора с помощью последовательной цепи (блок **Series RLC Branch**) следует задать L=0, а C=inf (бесконечное значение емкости превращает конденсатор в идеальный проводник).

В нагрузочных RLC-цепях (блоки **Series RLC Load, Parallel RLC Load, 3-Phase Series RLC Load** и **3-Phase Parallel RLC Load**) задаются допустимые мощности рассеивания – активная для резистора и реактивная для индуктивности и конденсатора.

В окне настраивания блока **Mutual Inductance** (рис. 13) предусмотрены следующие параметры настройки:

Winding 1 Self Impedance	(Собственный импеданс первой катушки) здесь задается вектор из двух элементов: первый – сопротивление первой катушки в Омах, второй – ее индуктивность в Генри
Winding 2 Self Impedance	(Собственный импеданс второй катушки) задается сопротивление второй катушки в Омах и ее индуктивность в Генри
Winding 3 Self Impedance	(Собственный импеданс третьей катушки) задается сопротивление третьей катушки в Омах и ее индуктивность в Генри
Mutual Impedance	(Взаимный импеданс катушек) здесь задается вектор из двух элементов: первый – сопротивление в Омах, второй – взаимная индуктивность в Генри

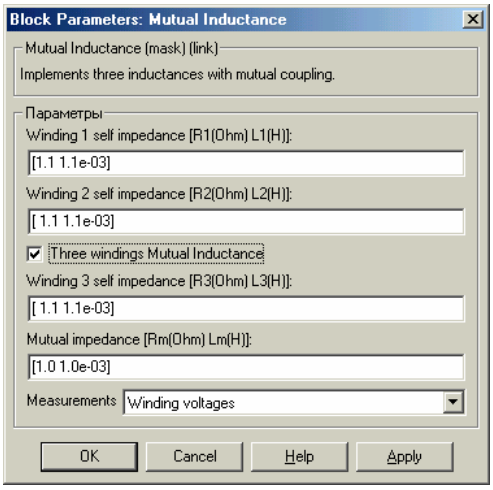


Рис. 10. 13. Окно настраивания блока **`Mutual Inductance`**

Кроме того, предусмотрена возможность создать взаимную индуктивность из двух катушек, для чего в окошке Tree windings Mutual Inductance следует сбросить флажок.

В группе блоков **`Lines`** находятся блоки имитации двух типов линий электропередачи. Первый тип – линии PI Section Line – представляет линию электропередачи как совокупность последовательно соединенных секций с сосредоточенными параметрами. При этом длина линии задается в километрах (рис. 14), а параметрами линии являются сопротивление, индуктивность и емкость одного километра линии.

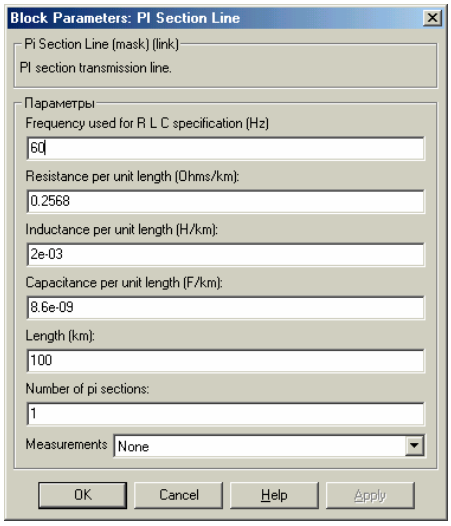


Рис. 10. 14. Окно настраивания блока PI Section Line

Второй тип линии, представленный блоком Distributed Parameters Line (Линия с распределенными параметрами), имитирует электрическую линию более приближенной к реальности системой с непрерывно распределенными вдоль длины линии электрическими параметрами. Параметры настраивания этого блока, в основном совпадают с описанными ранее.

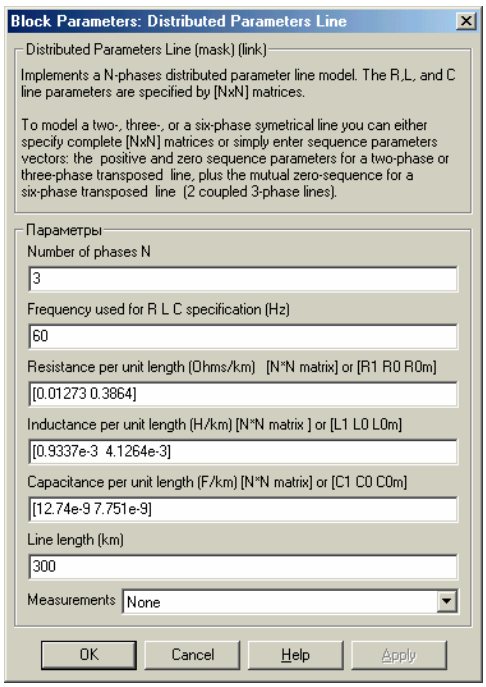


Рис. 10. 15. Окно настраивания блока PI Section Line

Хотя по умолчанию этот блок рассчитан на трехфазную проводную передачу, число линий можно уменьшить до двух или одной, устанавливая это число в окошке Number of Phases N.

Для имитации выключателей (рубильников) предусмотрены блоки третьей группы (Breakers). Они обеспечивают моделирование процессов, возникающих при включении или выключении переменного тока. На рис. 16 показано окно настраивания простейшего из них - блока Breaker.

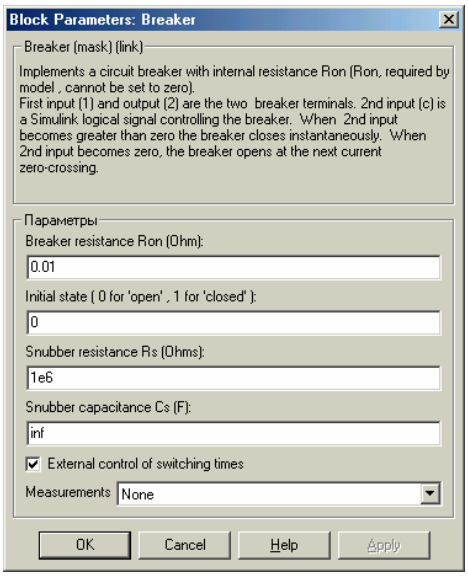


Рис. 10. 16. Окно настраивания блока Breaker

Группа Transformers включает блоки, имитирующие работу трансформаторов. Блок Linear Transformer (Линейный трансформатор) представляет собой линейную модель трансформатора, не учитывающую насыщения в обмотках трансформатора. В число параметров настраивания блока входят (рис. 17):

Nominal power and frequency задается вектор из двух элементов: первый – номинальная мощность трансформатора в Вольт-амперах, второй – частота питания в Герцах

Winding 1 parameters (Параметры первой обмотки) задается вектор из трех элементов: первый – напряжение на обмотке в Вольтах, второй - сопротивление обмотки, третий - индуктивность обмотки

Winding 2 pa- (Параметры второй обмотки) задается вектор из трех элементов: первый – напря-

rameters жение на обмотке в Вольтах, второй - сопротивление обмотки, третий - индуктивность обмотки

Winding 3 parameters (Параметры третьей обмотки) задается вектор из трех элементов: первый – напряжение на обмотке в Вольтах, второй - сопротивление обмотки, третий - индуктивность обмотки

Magnetization resistance and reactance (Активное и реактивное сопротивление намагничивания) здесь задается вектор из двух элементов: первый – сопротивление намагничивания, второй – индуктивность намагничивания

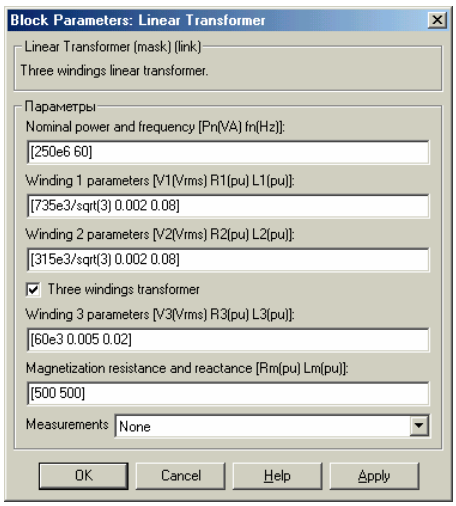


Рис. 10. 17. Окно настраивания блока Linear Transformer

Величины сопротивлений и индуктивностей в этом случае задаются в относительных единицах (pu – per unit), т. е. как величины отношений к базовому сопротивлению и базовой индуктивности. Последние определяются соотношениями:

$Rbase_j = V_j^2 / Pn$; $Lbase_j = Rbase_j / (2\pi fn)$,

где индекс j обозначает номер обмотки, V_j – напряжение на обмотке. Остальные обозначения такие, как изображены в окне настраивания. Базовые сопротивление и индуктивность для параметров намагничивания совпадают с соответствующими базовыми параметрами для первой обмотки.

Раздел *Connectors*

Содержимое раздела Connectors (Соединительные элементы) представлено на рис. 18.

Назначение блоков этого раздела обычно вполне очевидно. Первые два блока служат для подключения элементов электрической цепи к общему проводу – земле. Вторые два блока - нейтрали – отображают также общий провод для нескольких элементов электрической схемы, но, в отличие от заземления, потенциал которого принимается равным нулю, потенциал нейтрали может быть произвольным. Самих нейтралей на схеме может быть несколько. Для отличия их от земли и других нейтралей (с другими потенциалами) нейтрали снабжаются порядковым номером (номером узла), который проставляется в окне настраивания блока.

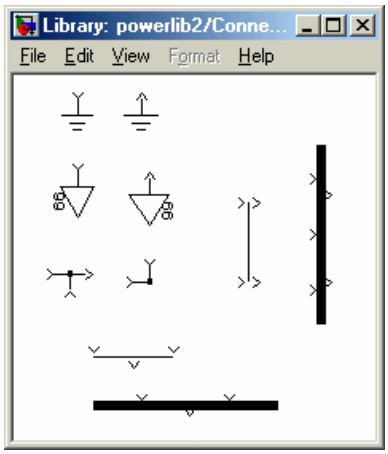


Рис. 10. 18. Содержимое раздела Connectors

Четыре остальных элемента представляют собой различные сочетания разветвлений и соединений электрических линий.

Раздел *Power Electronics*

Окно раздела Power Electronics (Элементы силовой электроники) показано на рис. 19.

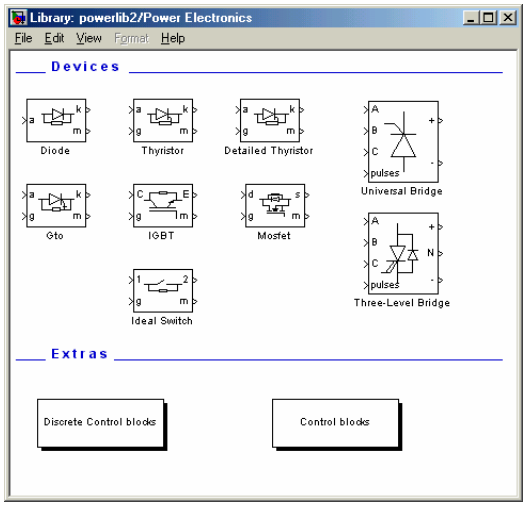


Рис. 10. 19. Содержимое раздела Power Electronics

Здесь расположены такие блоки:

Ideal Switch	(Идеальный ключ) управляемый переключатель
Diode	(Диод) полупроводниковый диод
Thyristor	(Тиристор) упрощенная модель тиристора
Detailed Thyristor	(Детализированный тиристор) детальная модель тиристора
Gto	(Запираемый тиристор) силовой биполярно-полевой блок
IGBT	Комбинация биполярных транзисторов с полевыми
Mosfet	Упрощенная модель полевого транзистора с изолированным затвором
Universal Bridge	(Универсальный мост)
Three Level Bridge	Трехуровневый мост

Все модели содержат имитацию гасящей выбросы напряжения последовательной цепи RsCs (snubber). Блоки имеют m-выход, на котором формируется вектор текущих значений тока, протекающего через соответствующий элемент, и напряжения на клеммах элемента. Как обычный S-сигнал, сигнал с этого выхода может быть использован любыми S-блоками для преобразования и вывода результатов.

В управляемых блоках (тиристоры) предусмотрен также и m-вход, обозначенный g, на вход которого подается управляющий S-сигнал, предварительно сформированный S-блоками.

В блоках мостов для установки величин, которые должны быть измерены и вектор которых будет выдавать m-выход, в окне настраивания в нижней его части находится список Measurement.

Раздел *Machines*

Содержимое раздела Machines (Машины) представлено на рис. 20.

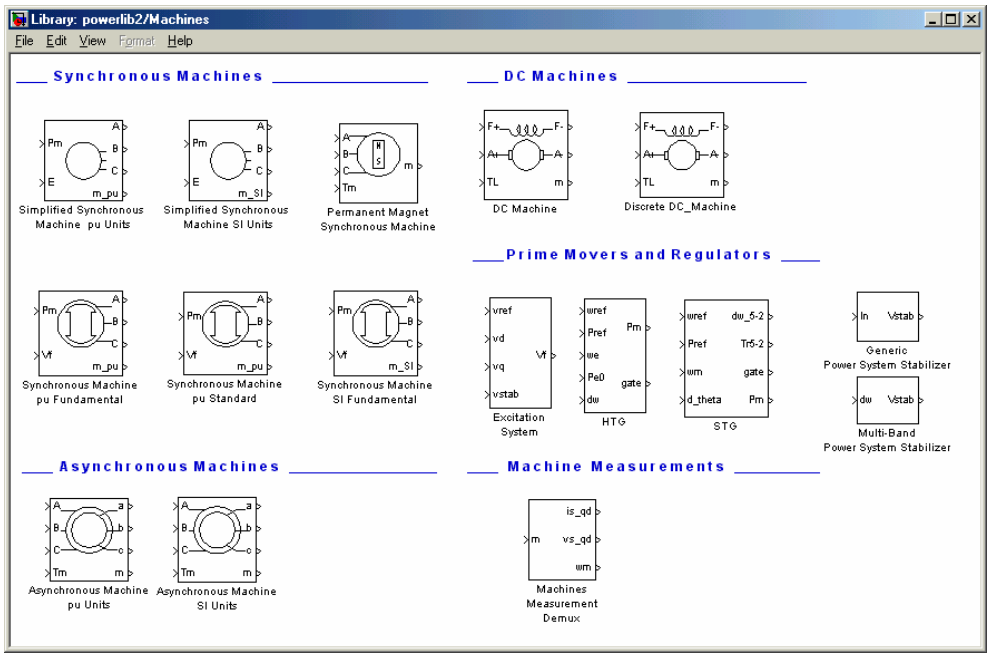


Рис. 10. 20. Содержимое раздела Machines

Далее приводится перечень блоков этого раздела, реализующих различные модели электромашин.

Simplified Synchronous Machine pu Units	Упрощенная модель синхронной машины в относительных единицах
Simplified Synchronous Machine SI Units	Упрощенная модель синхронной машины в единицах системы СИ
Permanent Magnet Synchronous Machine	Синхронная машина с постоянным магнитом
Synchronous Machine pu Fundamental	Основная (полная) модель синхронной машины в относительных единицах
Synchronous Machine SI Fundamental	Основная (полная) модель синхронной машины в единицах системы СИ
Synchronous Machine pu Standart	Стандартная модель синхронной машины в относительных единицах
Asynchronous Machine pu Units	Модель асинхронной машины в относительных единицах
Asynchronous Machine SI Units	Модель асинхронной машины в единицах СИ
DC Machine	Машина постоянного тока
Discrete DC Machine	Машина постоянного тока с дискретным управлением

В отличие от рассмотренных ранее блоков, перечисленные блоки моделируют поведение электромеханических устройств, а потому, помимо электрических связей, необходимо содержат связи механических величин и модели механических процессов. Поэтому они содержат m-входы и m-выходы, позволяющие подключить к модели S-блоки, в которых реализуются динамические процессы преобразования механических величин.

Так, в каждом из блоков электромашин присутствует один m-выход, отмеченный на изображении блока знаком «m». Этот m-выход представляет собой S-сигнал в виде вектора из электрических и механических величин, описывающих текущее состояние выбранного вида электромашины. Состав вектора различен для разных моделей машин. Разобраться в составе этого вектора и выбрать те из величин, которые необходимо использовать для построения модели можно, если подсоединить к m-выходу специальный блок из того же раздела – Machine Measurement Demux (Распределитель машинных измерений). На рис. 21 показано окно настраивания этого блока.

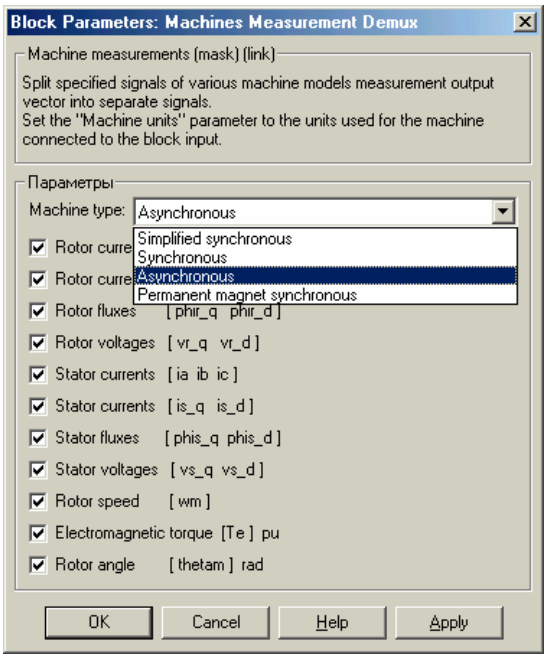


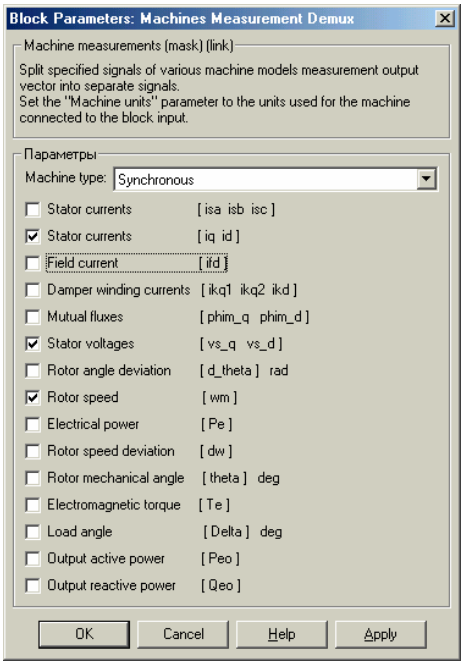
Рис. 10. 21. Окно настраивания блока Machine Measurement Demux (Asynchronous)

В верхней части окна располагается список, позволяющий выбрать тип модели машины (Simplified Sinchronous, Sinchronous, Asinchronous, Permanent Magnet Sinchronous). Из него следует выбрать тот тип машины, к m-выходу которой подсоединен этот блок Machine Measurement Demux. В зависимости от выбора из списка в окне настраивания появится тот или иной набор величин, описывающий измеряемый вектор.

На рис. 21 представлен набор измеряемых величин для асинхронной машины. Видно, что он включает токи и напряжения в различных электрических частях машины и, кроме того, три механические величины:

`Rotor speed`	Угловая скорость вращения ротора
`Electromagnetic torque`	Электромагнитный момент
`Rotor angle`	Угол поворота ротора

Набор измеряемых для синхронной машины величин представлен на рис. 22.



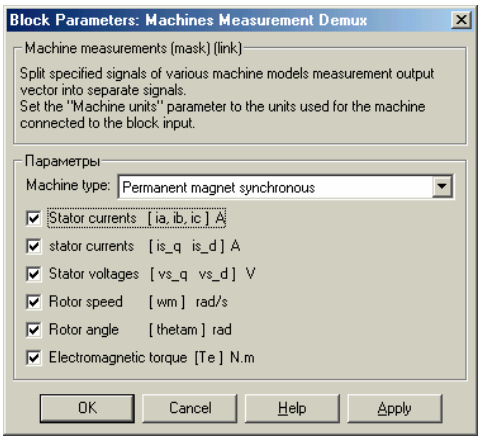


Рис. 10. 22. Окно настраивания блока Machine Measurement Demux (Synchronous)

Рис. 10. 23. Окно настраивания блока Machine Measurement Demux (Permanent Magnet Synchronous)

В нем находятся такие измеряемые механические величины:

Rotor angle deviation	Угол отклонения ротора от равновесного положения
Rotor speed	Угловая скорость вращения ротора
Rotor speed deviation	Угловая скорость отклонения ротора от равновесного положения
Electromagnetic torque	Электромагнитный момент
Rotor mechanical angle	Угол поворота ротора
Load angle	Угол нагрузки

Для машины постоянного тока (рис. 23) предусмотрены те же измеряемые механические величины, что и для асинхронной машины.

Установка флажка рядом с выбранными из представляемого набора величинами обеспечивает наличие этих величин в выходном сигнале блока Machine Measurement Demux. При этом величины в выходном векторе располагаются в порядке, представленном в окне настраивания блока.

В каждом блоке электрической машины предусмотрены p-входы и (или) p-выходы для подключения к электрической схеме. Так, в блоках синхронных машин такими являются выходы A, B и C, имитирующие клеммы обмоток статора. В блоках асинхронных машин – это клеммы статора A, B и C и клеммы ротора a, b и c. Изображения блоков машин постоянного тока содержат четыре клеммы электрических соединения - F+ и F- для подсоединения обмотки возбуждения, а также A+ и A- для подсоединения якоря.

Остальные входы блоков электрических машин представляют собой m- входы, то есть на них должны подаваться S- сигналы. Так, на вход E блока упрощенной синхронной машины подается амплитуда управляющего напряжения, а на вход Pm – сигнал, равный текущему значению механической мощности на валу машины. На вход Vf основной модели синхронной машины подается сигнал возбуждения.

В блоках синхронной машины с постоянными магнитами и асинхронных машин m- входом является механический момент Tm на валу машины, а в блоках машин постоянного тока – момент TL нагрузки на валу.

Сигналы, поступающие на эти входы, должны быть сформированы в модели системы либо как явные функции времени, либо в S-блоках, имитирующих работу блоков управления или динамику механических устройств, являющихся нагрузкой на валу машины. В разделе Machines имеется ряд блоков, осуществляющих эти функции:

Excitation system	(Система возбуждения) осуществляет формирование напряжение возбуждения синхронной машины
Generic Power System Stabilizer	Стабилизатор колебаний
Multi-band Power System Stabilizer	Многополосный стабилизатор колебаний
HTG	(Гидравлическая турбина с регулятором) модель динамики гидравлической турбины с ПИД-регулятором для синхронного генератора
STG	(Паровая турбина с регулятором) модель динамики четырехступенчатой турбины с ПИД-регулятором для синхронного генератора

Раздел *Measurements*

На рис. 24 показано содержимое раздела Measurements (Измерители).

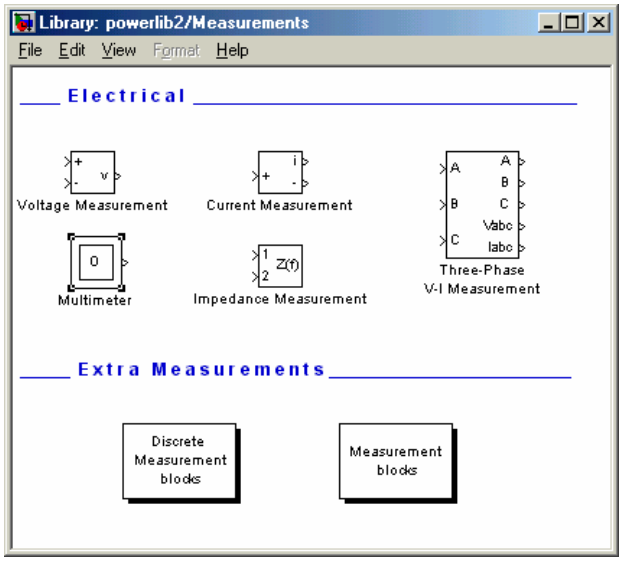


Рис. 10. 24. Содержимое раздела Measurements

Наиболее важными из них являются блоки Voltage Measurement (Вольтметр), Current Measurement (Амперметр) и Multimeter (Многофункциональный измеритель). Важность их заключается в том, что эти блоки имеют, в отличие от других блоков библиотеки **SimPowerSystems** один m-выход (он обозначен знаком v в блоке вольтметра и знаком i - в блоке амперметра). Остальные входы и выходы этих блоков представляют собой, как обычно, клеммы для подсоединения измерителя к электрической цепи.

Благодаря наличию в этих блоках m-выхода, сигнал с этого выхода (напряжение или ток) может быть в дальнейшем преобразован с помощью обычных S-блоков, а затем использован в преобразованном виде вновь в электрической схеме, если его подать на вход блока **`Controlled Voltage Source`** или блока **`Controlled Current Source.`** Кроме того, этот же сигнал может быть транспортирован обычными средствами в рабочее пространство MatLab и затем представлен графическими средствами в окне Figure.

Особое место занимает блок Multimeter (Многофункциональный измеритель). Он предназначен для "измерения" и формирования вектора S-сигнала на его выходе, состоящего из величин, которые выбраны в списках Measurement, находящихся в блоках той блок-схемы, куда помещен блок Multimeter.

Окно настраивания блока Multimeter приведено на рис. 25. Оно состоит из двух полей – Available Measurements (Доступные измерения) и Selected Measurements (Отмеченные измерения). В первом поле автоматически (без вмешательства пользователя) появляются все величины, которые отмечены в списках Measurements окон настраивания блоков, находящихся в блок-схеме с установленным в ней блоком Multimeter. Отмечая в этом поле те из измеряемых величин, которые надо включить в выходной сигнал блок Multimeter, с помощью кнопки » пользователь переводит их в поле Selected Measurements. Таким образом формируется в этом поле список измеряемых величин. Порядок следования в списке может быть изменен на желаемый при помощи кнопок Up, Down и Remove (рис. 25) в окне настраивания.

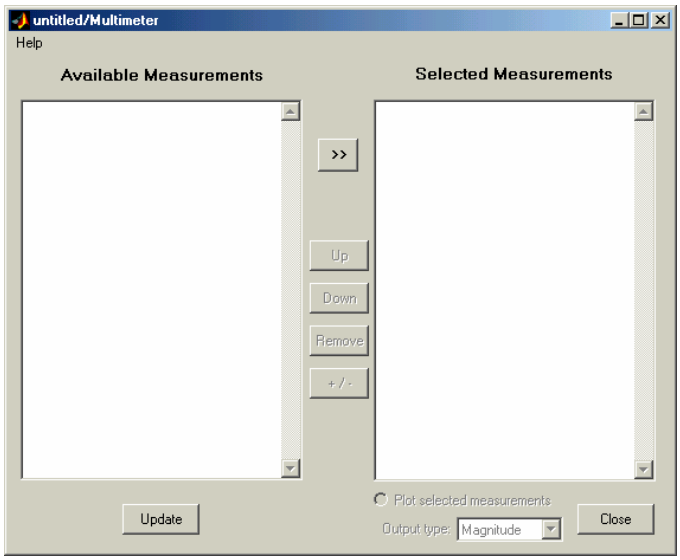


Рис. 10. 25. Окно настраивания блока Multimeter

Для вывода графиков отмеченных величин непосредственно в графическое окно следует установить флажок рядом с надписью Plot selected measurement.

10.2. Модель запуска асинхронного двигателя

Работу по моделированию с использованием библиотеки **SimPowerSystems** рассмотрим на ряде примеров, приведенных в книге С. Г. Герман-Галкина [3], посвященной вопросам моделирования электроприводов.

В качестве первого примера рассмотрим простейшую модель AM_1.mdl асинхронного двигателя с короткозамкнутым ротором (см. модель akzvirt.mdl [3, с. 241-243]). На рис. 26 приведена блок-схема этой модели.

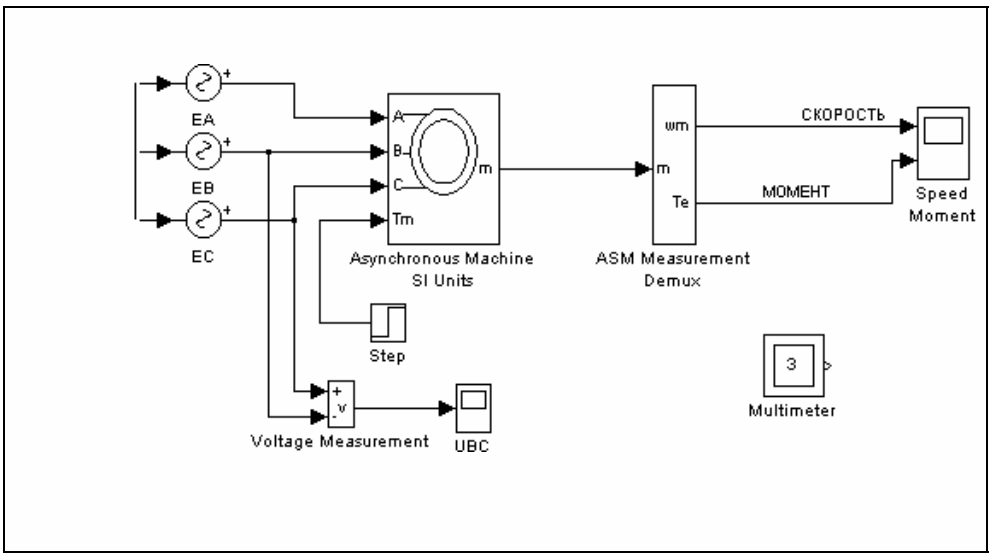


Рис. 10. 26. Блок-схема модели асинхронного двигателя

Основу модели составляет блок **Asynchronous Machine SI Units.** Параметры двигателя приведены на рис. 27.

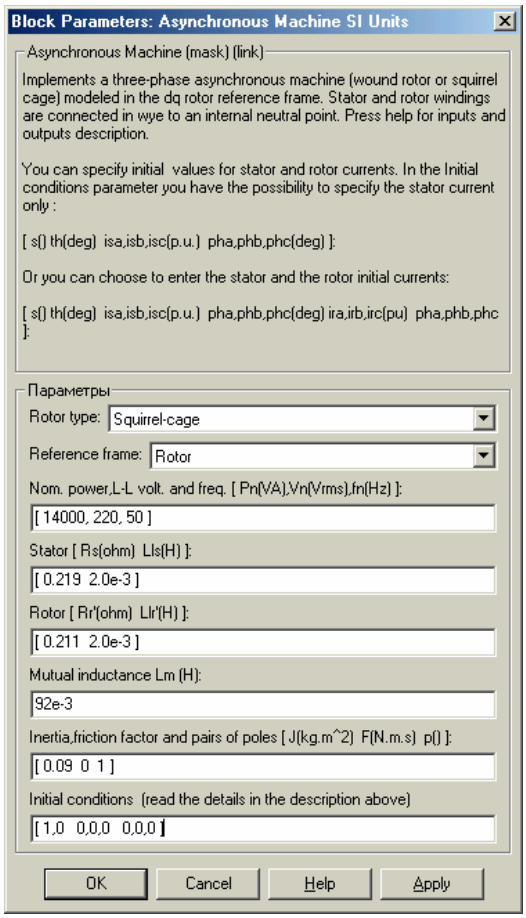


Рис. 10. 27. Окно настраивания блока **Asynchronous Machine SI Units**

Установленный тип ротора Squirrel-cage (Беличья клетка) указывает на то, что ротор является коротко-замкнутым с типом «обмотки» - «беличья клетка».

Блок запитывается от трехфазного источника напряжения, состоящего из трех соединенных звездой блоков **AC Voltage Source.**

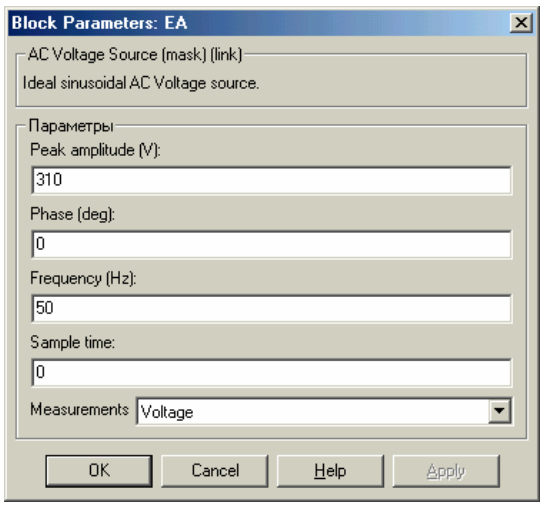


Рис. 10. 28. Окно настраивания блока **AC Voltage Source**

Каждый из этих трех блоков «вырабатывает» переменное напряжение частотой 50 Герц амплитудой 310 Вольт. Начальные фазы в блоках установлены разные, сдвинутые друг относительно друга на 120° (0°, 120° и -120°). Отметим, что в поле Measurement установлен вид измеряемой величины – напряжение (Voltage). Благодаря этому величины напряжений всех трех источников становятся доступными для измерения блоком Multimeter, который установлен в окне блок-схемы (см. рис. 26). При этом в окне настраивания блока Multimeter в поле Available Measurements автоматически появляются обозначения сигналов указанных трех напряжений. Переводя их в поле Selected Measurements, получим вид окна настраивания, приведенный на рис. 29. Для автоматического построения графиков зависимости этих напряжений от времени в поле Plot selection measurements установлен флажок.

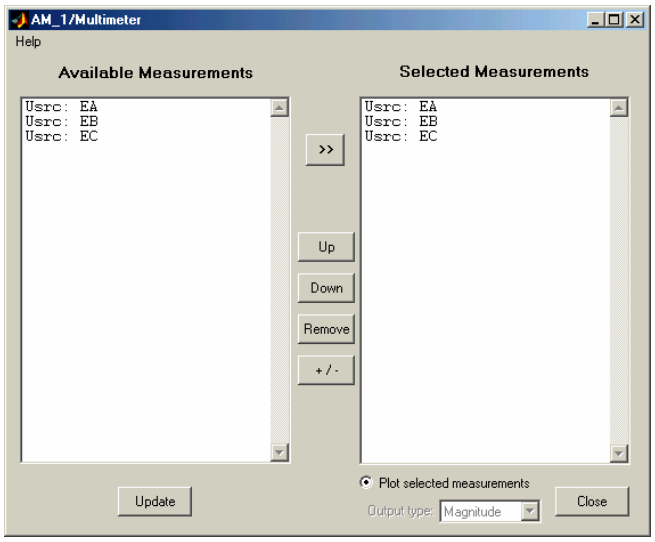


Рис. 10. 29. Окно настраивания блока Multimeter

Величина момента нагрузки на валу двигателя устанавливается блоком Step, параметры которого показаны в его окне настраивания (рис. 29).

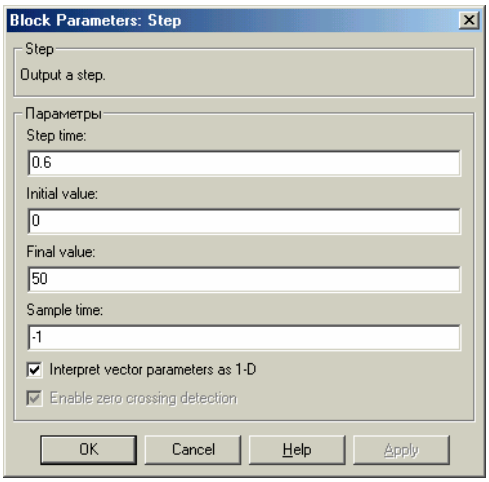


Рис. 10. 30. Окно настраивания блока **`Step`**

В соответствии с ними, в начальный промежуток времени (до 0,6 с) моментная нагрузка предполагается отсутствующей, а в последующее время действует постоянный момент 50 (Нм).

К m-выходу блока **`Asynchronous Machine SI Units`** подсоединен вход блока ASM Measurement Demux. В списке его окна настраивания (рис. 31) отмечены только две величины – Rotor speed (Угловая скорость вращения ротора) и Electromagnetic torque (Электромагнитный момент, развиваемый двигателем).

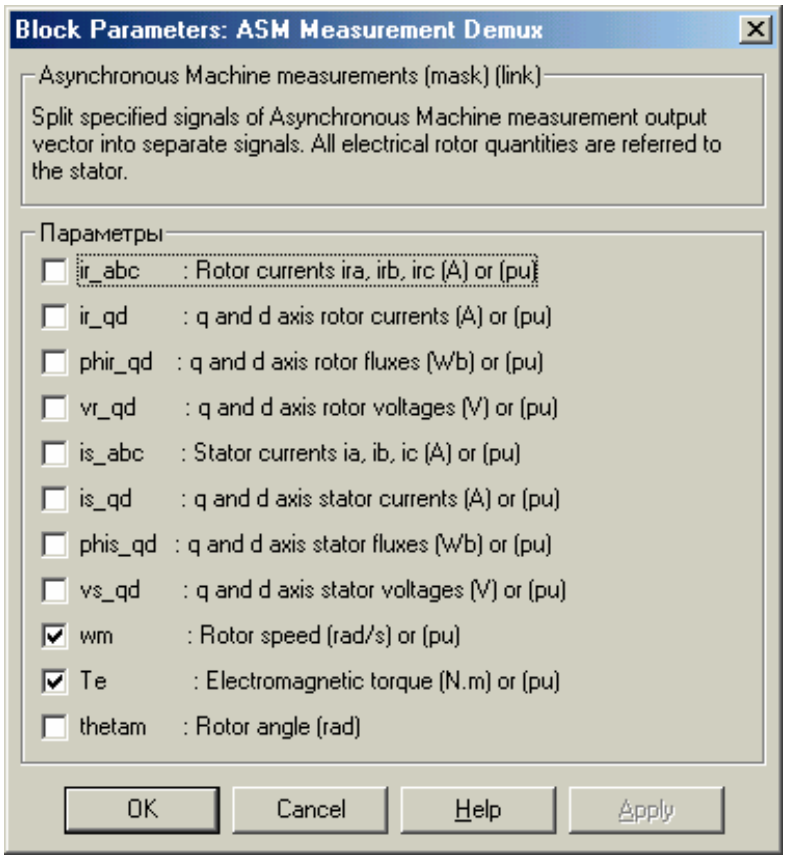


Рис. 10. 31. Окно настраивания блока Machine Measurement Demux

Поэтому именно эти две величины и составят выходы блока ASM Measurement Demux (см. рис. 26). Скорость и момент затем направляются в блок Speed Moment (типа Scope), позволяющий зарегистрировать графики зависимости этих величин от времени.

Для измерения и регистрации межфазного напряжения в блок-схеме предусмотрены блоки Voltage Measurement и UBC (типа Scope).

Установим параметры численного интегрирования описанной модели, показанные на рис. 32.

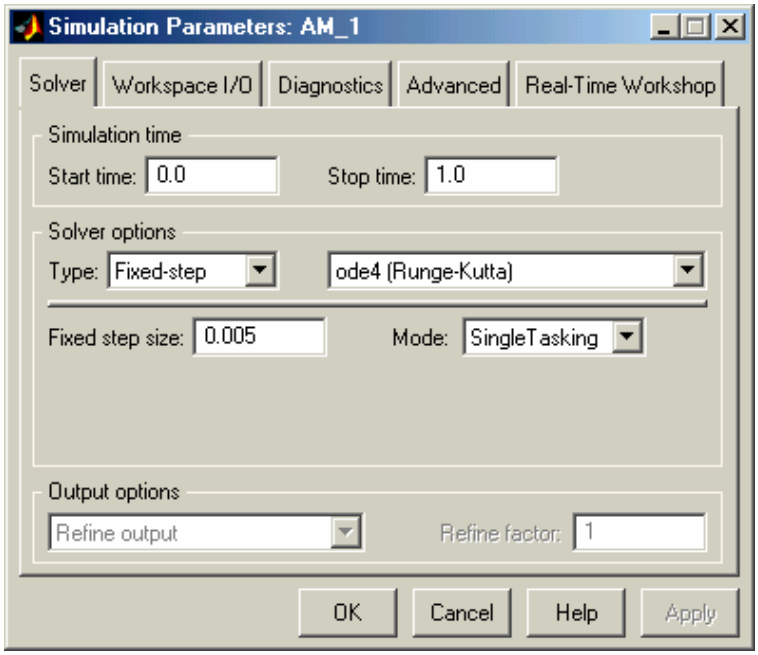


Рис. 10. 32. Окно установки параметров процесса моделирования

После запуска модели на экране автоматически возникает окно, показанное на рис. 33, в котором отображены зависимости питающих напряжений от времени.

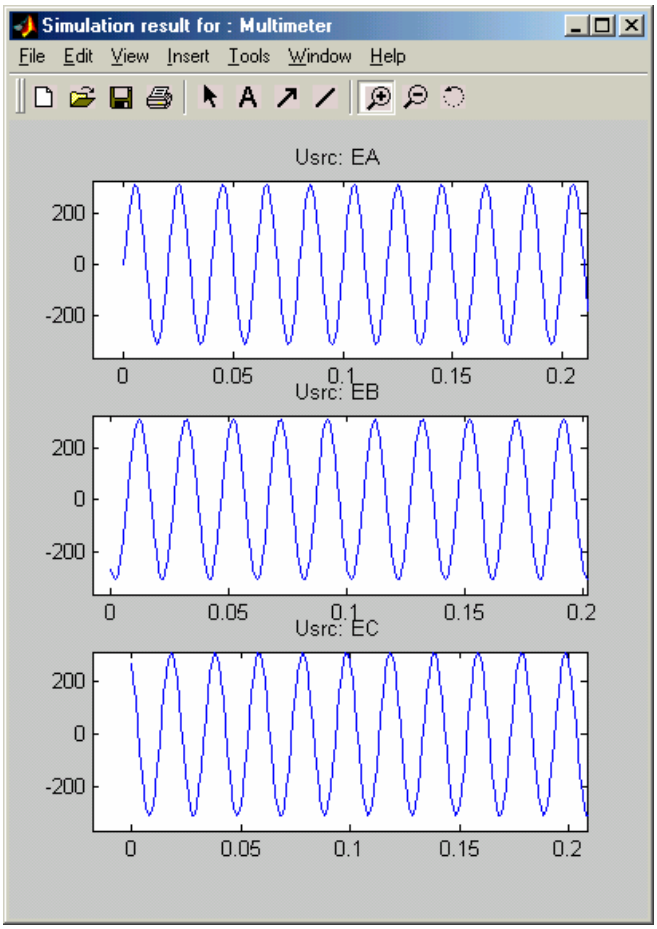


Рис. 10. 33. Результаты моделирования, выведенные блоком Multimeter

Если теперь раскрыть окно блока Speed Moment, то появится окно (рис. 34), в котором изображены графики зависимости от времени скорости вращения ротора и электромагнитного момента, действующего на ротор.

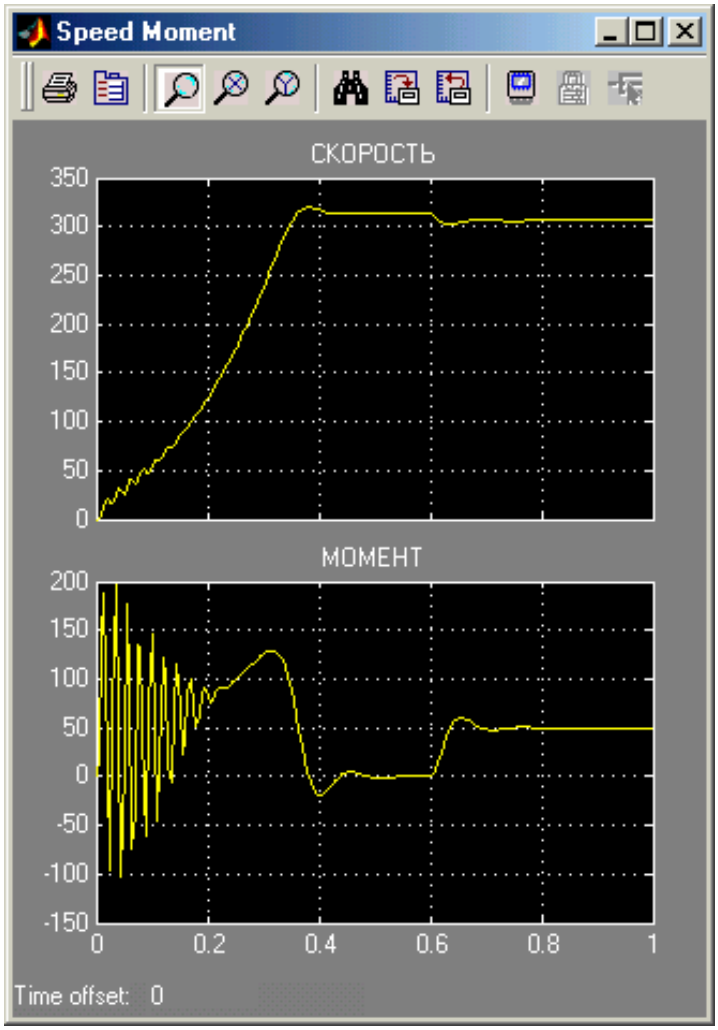


Рис. 10. 34. Результаты моделирования, показываемые в окне блока Speed Moment

Результаты моделирования показывают, что при прямом пуске вначале возникают значительные колебания момента. Такие же колебания наблюдаются по току и скорости. Установление переходного процесса при отсутствии нагрузки на валу двигателя происходит по истечении 0,5 с. В момент времени (0,6 с), когда прикладывается значительный момент нагрузки (50 Нм), происходит скачок по моменту с переходным процессом и наблюдается уменьшение скорости вращения ротора.

10.3. Модель трехфазного мостового управляемого выпрямителя

В качестве второго примера используем модель rect_virt_contr. mdl трехфазного мостового управляемого выпрямителя, приведенную и описанную в [3, с. 143].

Основой модели является блок Thyristor Convertor (Тиристорный преобразователь) типа Universal Bridge (Универсальный мост). Нагрузкой его является активно-индуктивная цепь с противоэдс, обеспечивающая как выпрямительный, так и инверторный режимы работы. Цепь состоит из последовательно соединенных двух блоков типа Series RLC branche (Последовательная RLC цепь) и DC Voltage Source (Источник постоянного напряжения).

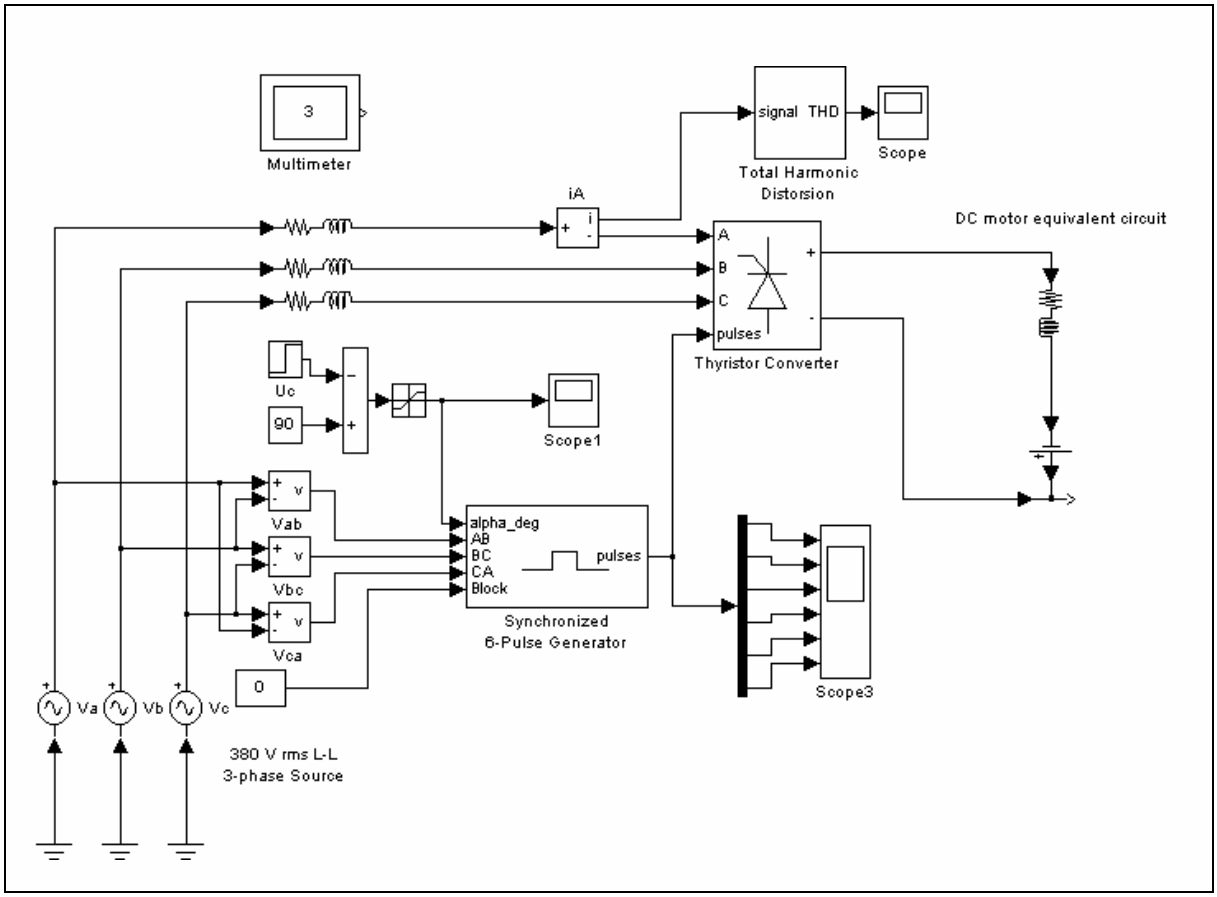


Рис. 10. 35. Блок схема модели трехфазного управляемого выпрямителя

Управление блоком Universal Bridge осуществляется через блок Synchronized 6-Pulse Generator (Генератор 6 импульсных синхронизированных сигналов), а запитывается он трехфазным источником напряжения с индуктивным внутренним сопротивлением. Трехфазный источник моделируется тремя соединенными звездой блоками источников переменного напряжения (типа AC Voltage Source) и последовательно с ними соединенными тремя блоками типа Series RLC branche.

Межфазные напряжения измеряются вольтметрами и подаются в виде S-сигналов на входы генератора импульсов. Управление фазовым запаздыванием импульсов по отношению к опорным межфазным напряжениям осуществляется совокупностью блоков Uc (типа Step), Constant, Sum и Saturation, которые обеспечивают это запаздывание (на схеме - alpha_deg) равным 90^{o} на протяжении первых 0,04 секунды и 65^{o} – на протяжении остального времени.

Рассмотрим окна настраивания блоков. На рис. 36 и 37 представлены установки блоков, моделирующих трехфазный источник питания.

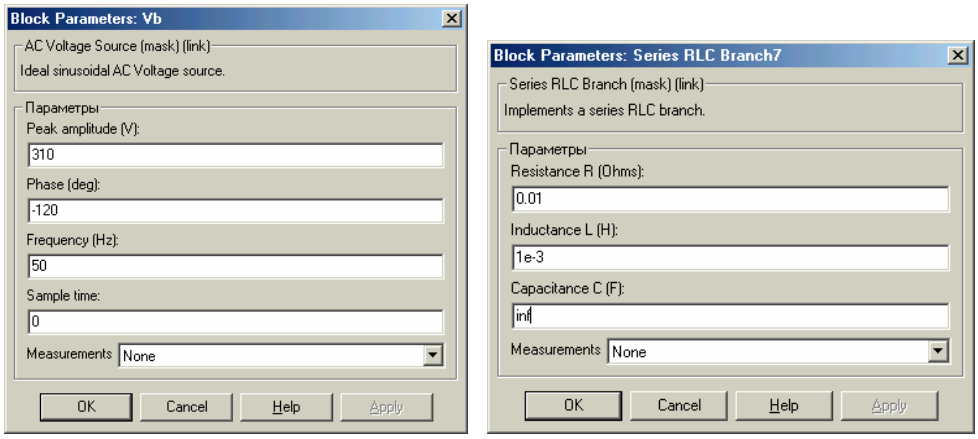
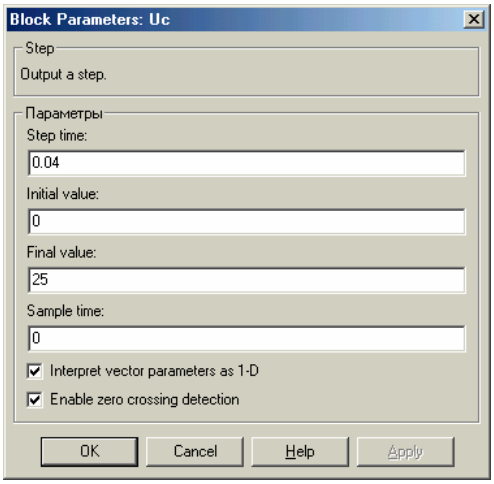


Рис. 10. 36. Окно настраивания блока Vb

Рис. 10. 37. Окно настраивания блока Series RLC Branch

Далее, на рис. 38 и 39, показаны параметры установки блоков, управляющих фазовой задержкой импульсных сигналов.



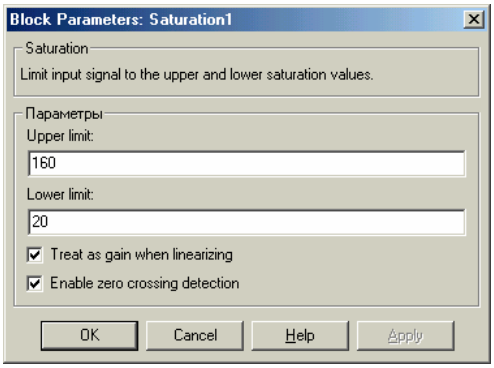
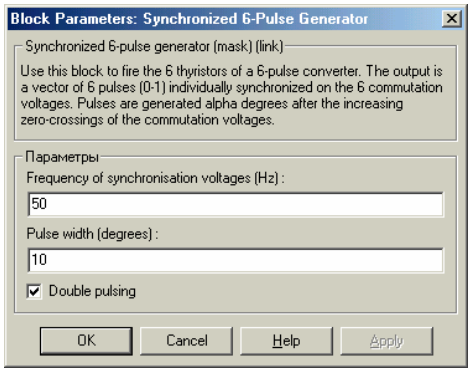


Рис. 10. 38. Окно настраивания блока Uc (Step)

Рис. 10. 39. Окно настраивания блока Saturation

Рис. 40 представляет параметры блока генератора импульсов. Из него следует, что частота напряжения синхронизации установлена 50 Гц, а ширина импульсов – 10^{o} (приведенная к фазовому углу). Параметры основного блока - Thyristor Convertor – приведены на рис. 41. Обратим особое внимание на установку All voltages and currents (Все напряжения и токи) в поле Measurements. Она позволяет вывести все токи и напряжения в ветвях этого моста в графические окна блока Multimeter.



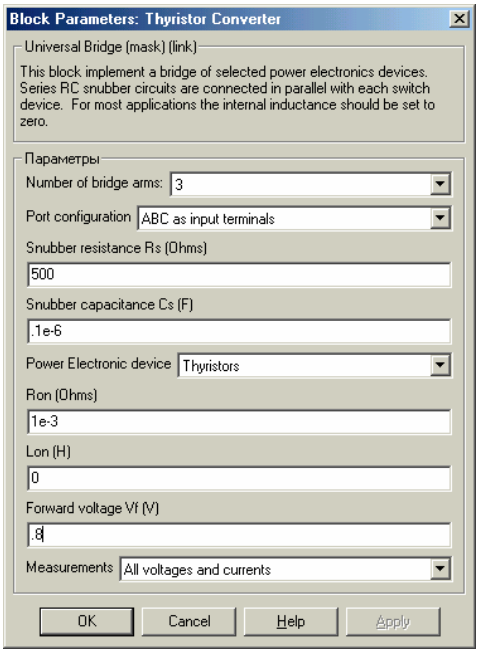
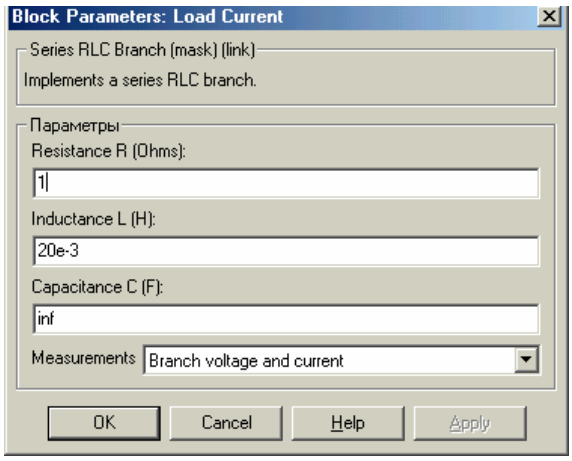


Рис. 10. 40. Окно настраивания блока Synchronized 6-Pulse Generator

Рис. 10. 41. Окно настраивания блока Thyristor Convertor

Параметры цепи нагрузки приведены на рис. 42 и 43. Обратите внимание на то, что в поле Measurement установлено **`Branch voltage and current`**, что позволяет проконтролировать ток и напряжение в цепи нагрузки при помощи блока Multimeter.



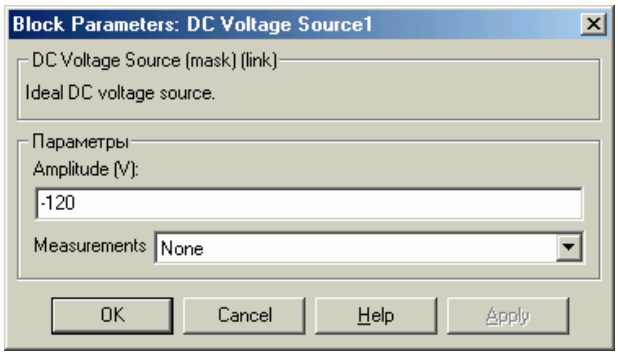


Рис. 10. 42. Окно настраивания блока Series RLC Branch в цепи нагрузки

Рис. 10. 43. Окно настраивания блока DC Voltage Source в цепи нагрузки

В блок-схеме модели предусмотрен еще один специальный блок Total Harmonic Distorsion, который вычисляет коэффициент отклонения формы токового сигнала от гармонической с указанной частотой (рис. 44).

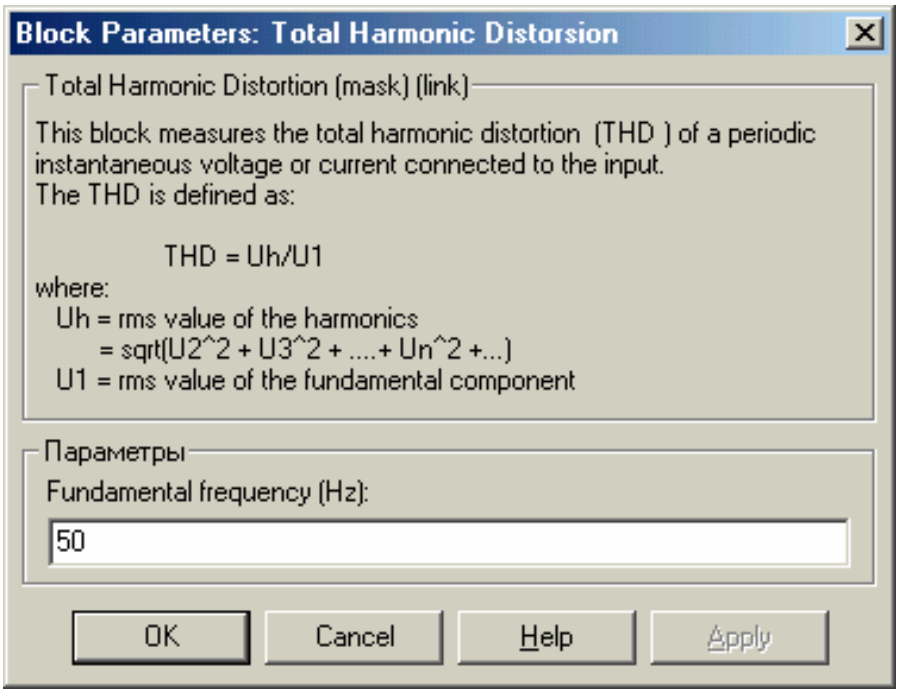


Рис. 10. 44. Окно настраивания блока Total Harmonic Distorsion

В результате предыдущих установок окно настраивания блока Multimeter примет вид, показанный на рис. 45.

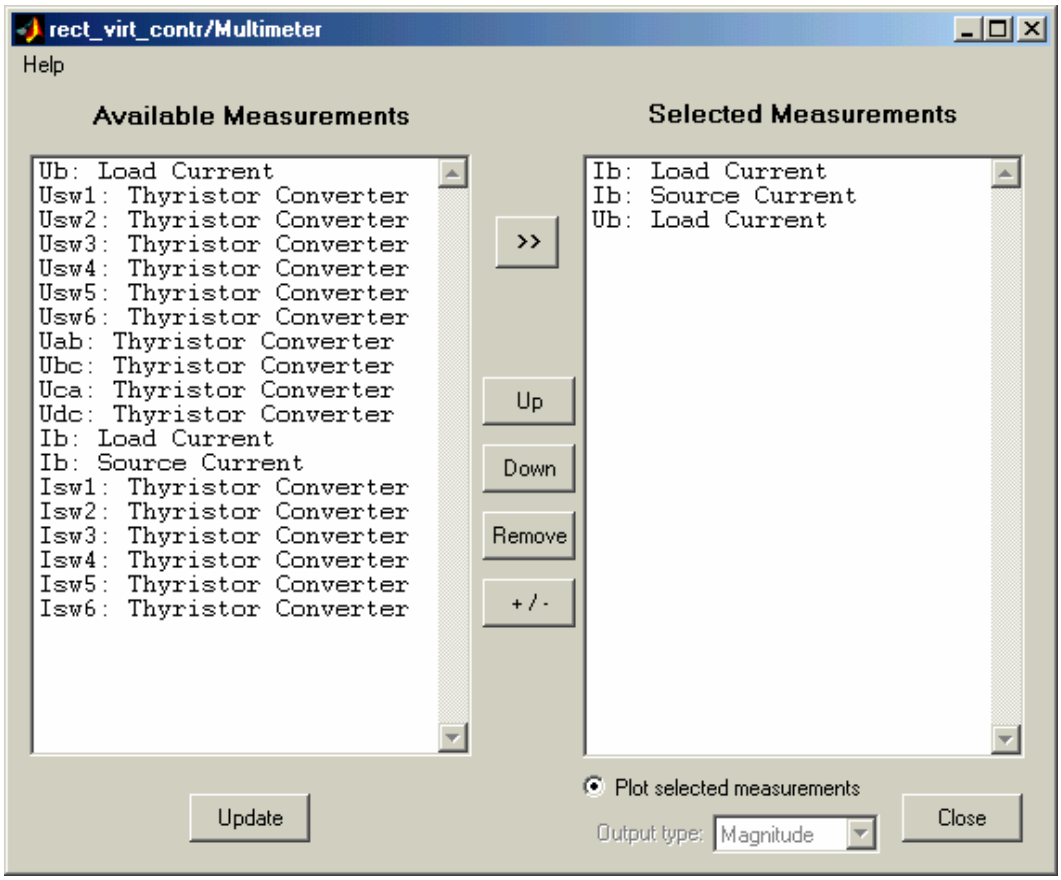


Рис. 10. 45. Окно настраивания блока Multimeter

Установим для представления в графическом окне сигналы тока источника и тока и напряжения нагрузки.

Проводя моделирование при указанных параметрах схемы, получим следующие результаты.

Токи нагрузки и питания и напряжение питания представлены в графическом окне (рис. 46) блока Multimeter.

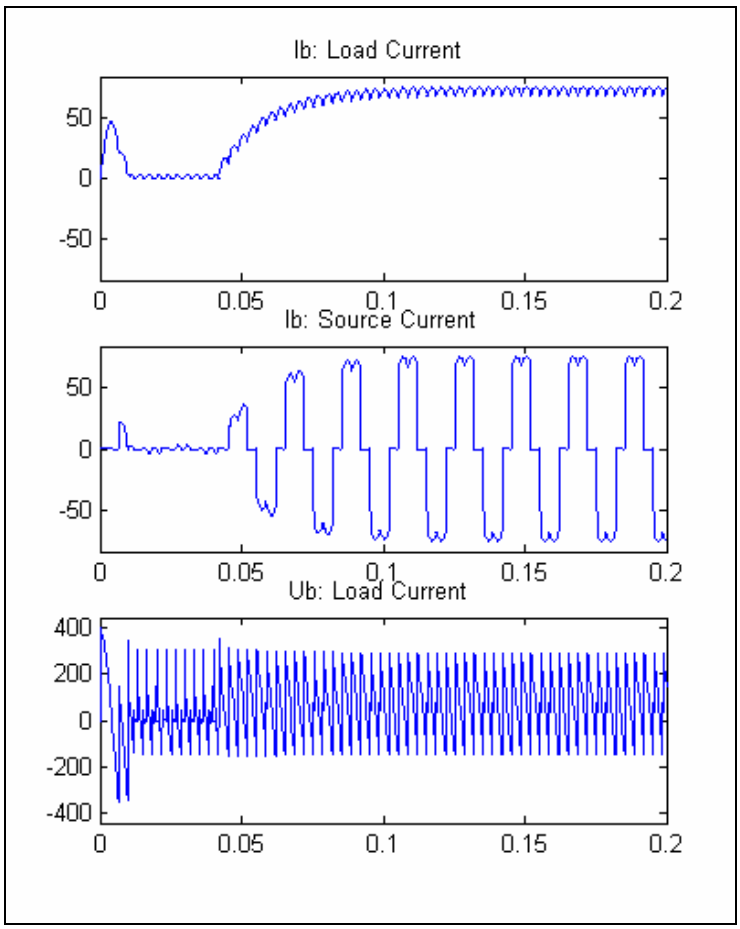


Рис. 10. 46. Результаты моделирования в графическом окне блока Multimeter

На выходе блока Total Harmonic Distorsion получим сигнал, отображенный (рис. 47) в окне блока Scope.

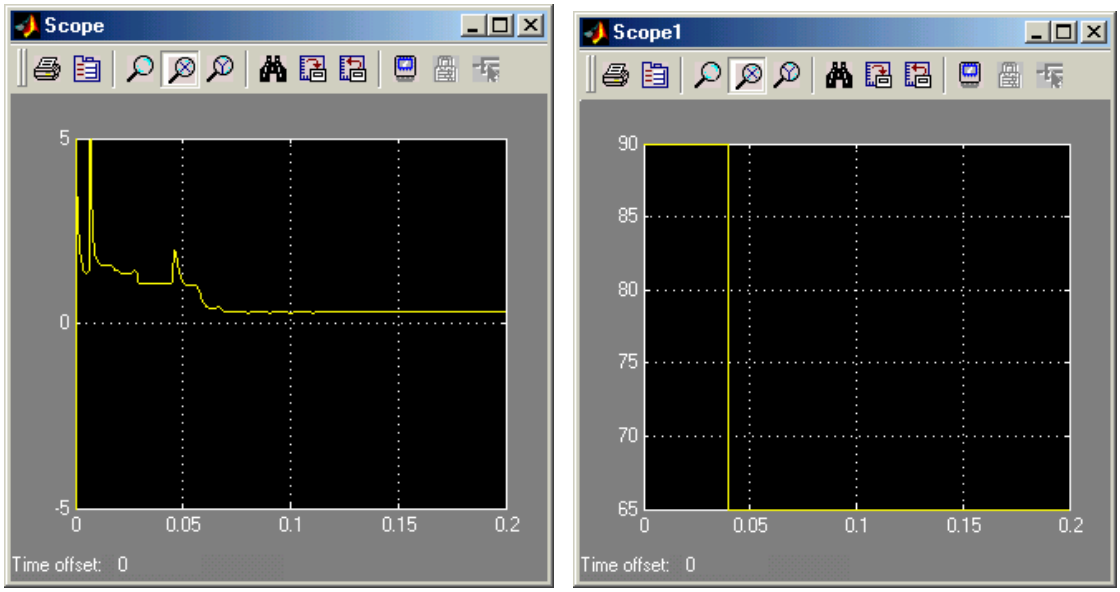


Рис. 10. 47. Зависимость коэффициента искажения гармоничности тока питания от времени в блоке Scope

Рис. 10. 48. Фазовая задержка импульсов

Далее, на рис. 48 и 49 показаны зависимости от времени фазовой задержки импульсов в генераторе импульсных сигналов и самих импульсных сигналов.

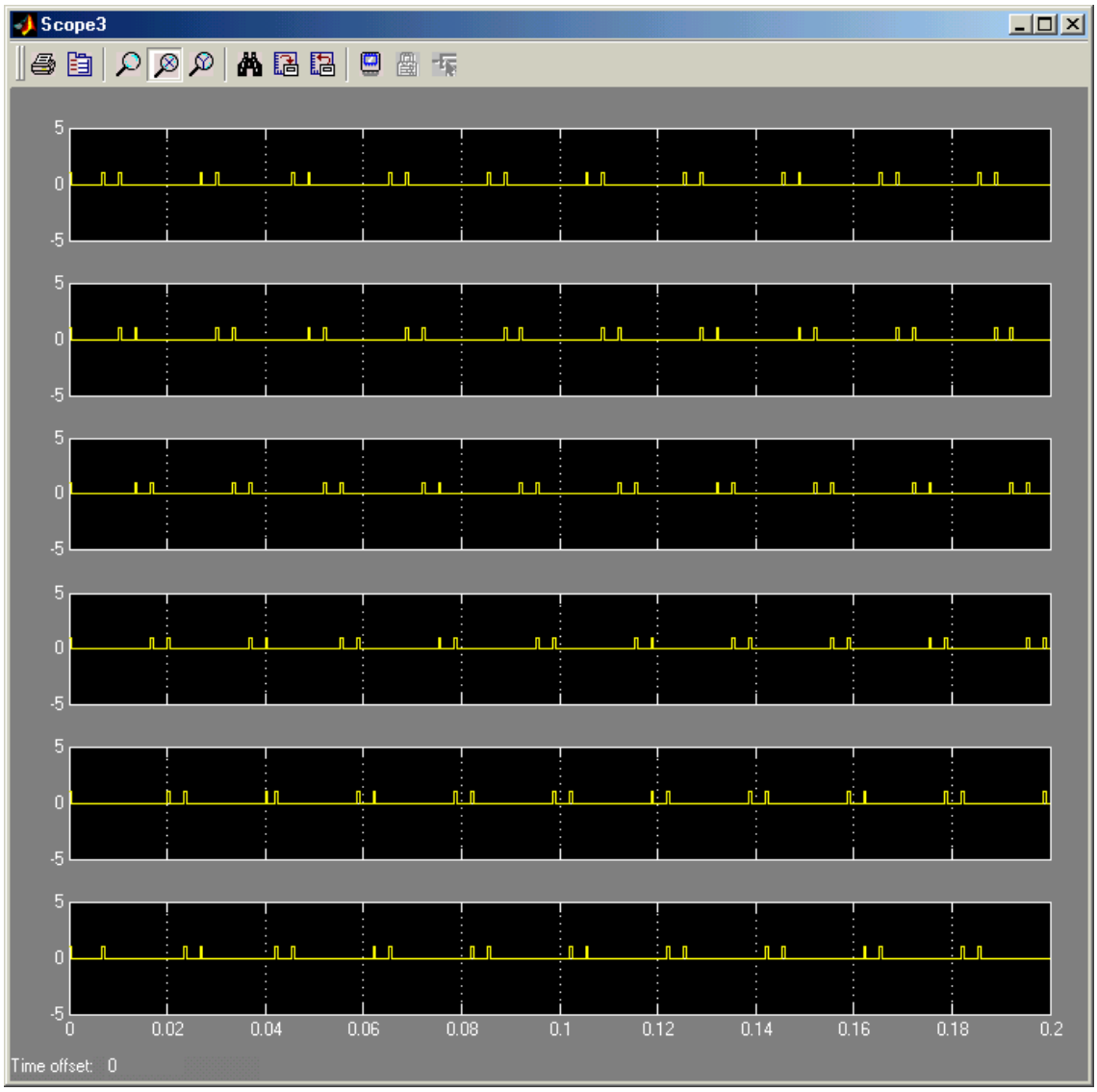


Рис. 10. 49. Выход блока генератора импульсных сигналов

10.4. Вопросы для самопроверки

1. Для чего предназначена библиотека **SimPowerSystems**?

2. Каковы основные принципы формирования блок-схемы, создаваемой на основе библиотеки **SimPowerSystems**? Отличны ли они от принципов создания S-моделей с помощью библиотеки Simulink?

3. Из каких основных разделов состоит библиотека **SimPowerSystems**?

4. В чем состоят основные особенности блоков библиотеки **SimPowerSystems** по сравнению с обычными S-блоками? Каковы преимущества и недостатки такого построения блоков?

5. Какие блоки библиотеки **SimPowerSystems** осуществляют связь с блоками библиотеки Simulink и с системой MATLAB?

6. Каково основное назначение блока Multimeter? Как осуществляется связь этого блока с другими блоками блок-схемы **SimPowerSystems?**

Урок 11. Моделирование машин и механизмов (библиотека SimMechanics)

Библиотека **SimMechanics** пакета **Simulink** содержит программные средства (в виде блоков) для моделирования механического движения механизмов и машин.

Как и в ранее рассмотренной библиотеке **SimPowerSystems**, идеология составления блок-схем в ней существенно отличается от идеологии функциональных блок-схем библиотеки **SIMULINK**, т. е. S-моделей. В блок-схеме **SimMechanics** отдельные блоки фактически следует рассматривать как модели, имитирующие механическое движение одной части моделируемого механизма относительно другой его части. "Входы" и "выходы" блока фактически таковыми не являются, а представляют собой имитацию "посадочного" места соответствующей части механизма. Линии соединения "входов" и "выходов" блоков имитируют жесткие соединения одной ("выходной") части одного механизма с "входной" частью другого механизма. Можно утверждать, что это соединение моделирует передачу силового воздействия между частями разных механизмов. Но, так как, в соответствии с третьим законом Ньютона, сила действия равна силе противодействия, такую передачу силы нельзя рассматривать как однонаправленное воздействие. Поэтому в блок-схемах **SimMechanics** на линиях соединений механических блоков вы не встретите изображений стрелок, указывающих на направление воздействия. По той же причине графические изображения "входов" и "выходов" механических блоков имеют вид не направленных стрелок, а квадратов с диагоналями.

Как и в блоках библиотеки **SimPowerSystems**, "входы" и "выходы" механических блоков нельзя рассматривать как источники и приемники каких бы то ни было сигналов. К линиям их соединений нельзя подсоединить обычные S-блоки, а потому нельзя и сформировать с помощью последних заданные воздействия или вывести информацию о получаемых в результате движениях механизмов (например, в обзорные окна или в систему MatLab). Но, так как любое моделирование механизмов невозможно осуществить без задания нужных исследователю воздействий и без вывода результатов моделирования в среду MatLab, такая идеология построения блок-схем механизмов требует включения в библиотеку блоков, осуществляющих прямую и обратную связь от S-блоков к механическим блокам и обратно. Такие блоки по необходимости должны иметь хотя бы один m-вход и один механический "выход" (для "восприятия" заданного воздействия и перевода его в механическое), либо механический "вход" и m-выход (для отображения результатов моделируемого механического движения в виде информационного сигнала).

11.1. Общая характеристика библиотеки SimMechanics

Если с помощью контекстного меню открыть библиотеку **SimMechanics** из браузера **Simulink**, то на экране появится окно, показанное на рис. 1.

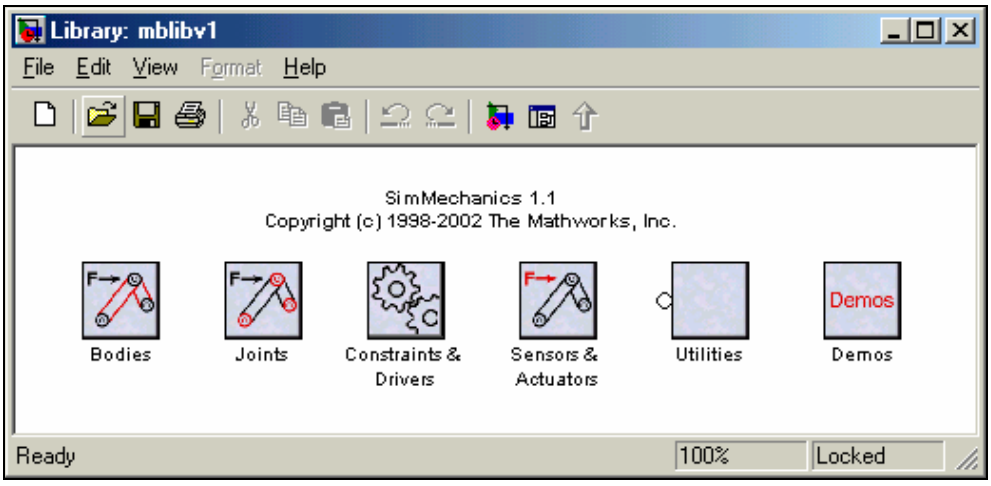


Рис. 11.1. Окно библиотеки SimMechanics

Как видим, библиотека содержит 6 разделов:

Bodies	(Тела) содержит блоки, моделирующие уравнения движения твердых тел
Joints	(Сочленения) включает блоки имитации механических сочленений, обеспечивающих требуемые степени свободы одной части механизма относительно другой
Constraints & Drivers	(Связи) состоит из блоков имитации ограничений на степени свободы механической системы
Sensors & Actuators	(Датчики и приводы) содержит блоки, имитирующие измерители параметров механического движения и блоки задания движения частей механизма
Utilities	(Утилиты) включает вспомогательные блоки, полезные при создании модели механизма
Demos	(Демонстрационные программы) позволяет вызвать на исполнение демонстрацион-

ные модели

Раздел **Bodies** содержит (рис. 2) 2 блока: Ground и Body.

Блок Ground (Основание) является обязательным при построении модели любого механизма. Он представляет неизменные точки основания (Земли), неподвижные в абсолютном (инерциальном) пространстве. Движение отдельных частей механизма задаются или определяются по отношению к системе координат, воплощаемой именно этим блоком.

Блок Body (Тело) представляет отдельную часть механизма, рассматриваемую как твердое тело, движение которого моделируется. В нем задаются масса и матрица инерции этого твердого тела, его начальное положение и ориентация (т. е. положение и ориентация систем координат CS, жестко связанных с ним). В число систем координат, жестко связанных с телом, обязательно входит система CG (Center of Gravity), начало которой совмещено с центром тяжести тела. Именно относительно осей этой системы координат задается матрица моментов инерции тела.

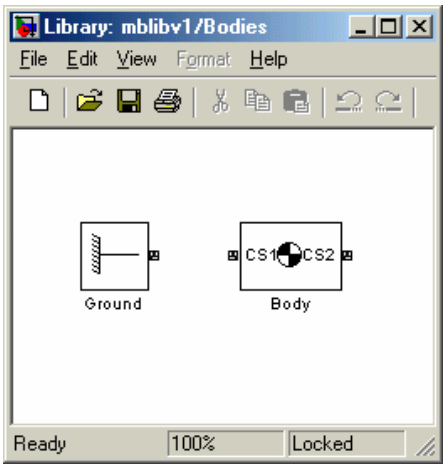


Рис. 11.2. Содержимое раздела Bodies

Раздел Joints (Сочленения) содержит блоки, обеспечивающие возможность относительных движений между телами, представленными отдельными блоками Body, т. е. обеспечение необходимых степеней свободы (DoFs). Сюда входят блоки Prismatic, Revolute, Spherical, имитирующие простейшие призматические, шарнирные и сферические сочленения, и блоки сложных готовых сочленений.

В разделе Constraints & Drivers (Связи и двигатели) собраны блоки, задающие предварительные ограничения на относительные перемещения между телами. Эти ограничения могут быть заданы как независимые от времени связи (блоки Constraints) и как зависящие от времени движения по степеням свободы (блоки Drivers).

Раздел Sensors & Actuators (Датчики и приводы) включает блоки Sensors для измерения относительных движений тел и блоки Actuators для задания относительных движений. Именно эти блоки организуют связь между механическими блоками библиотеки **SimMechanics** и обычными S- блоками, что позволяет, с одной стороны, использовать возможности библиотеки **SIMULINK** для формирования заданных движений, а, с другой, - использовать S-блоки для вывода результатов моделирования движения тел.

При составлении блок-схем механизмов следует принимать во внимание следующие обстоятельства.

1. Основу блок-схемы любого механизма составляет цепь типа

Ground – Joint – Body – Joint - … - Body

с открытой или закрытой топологией, где, по крайней мере, одно из тел представлено блоком Ground. Блоки Body могут иметь более двух соединенных с ним блоков Joint, фиксируя разветвления указанной последовательности. Но каждое сочленение (блок Joint) должно быть подсоединено к двум и только двум телам.

2. Между блоками Body могут быть соединены и блоками Driver или Constraint, имитирующими связи.

3. Блоки Actuator и Sensor могут быть подсоединены к любому из блоков Body, Joint или Driver, но только через дополнительные порты, устанавливаемых в окнах настраивания этих блоков.

4. Задание желаемого закона изменения во времени параметров движения возможно только с помощью блоков Actuator, а вывод результатов в рабочее пространство MatLab – через блоки Sensor, которые связывают блоки **SimMechanics** со средой **Simulink**.

Раздел Bodies

На рис. 3 показано окно настраивания блока Ground.

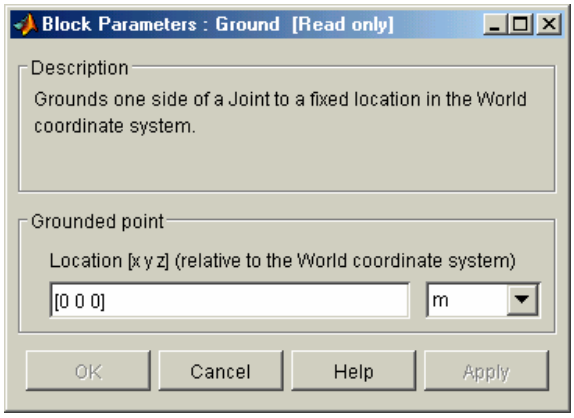


Рис. 11.3. Окно настраивания блока Ground

В нем лишь один параметр настраивания – вектор смещения начала системы координат, связанной с неподвижной частью механизма относительно начала инерциальной системы координат.

Окно настраивания блока Body представлено на рис. 4 и 5.

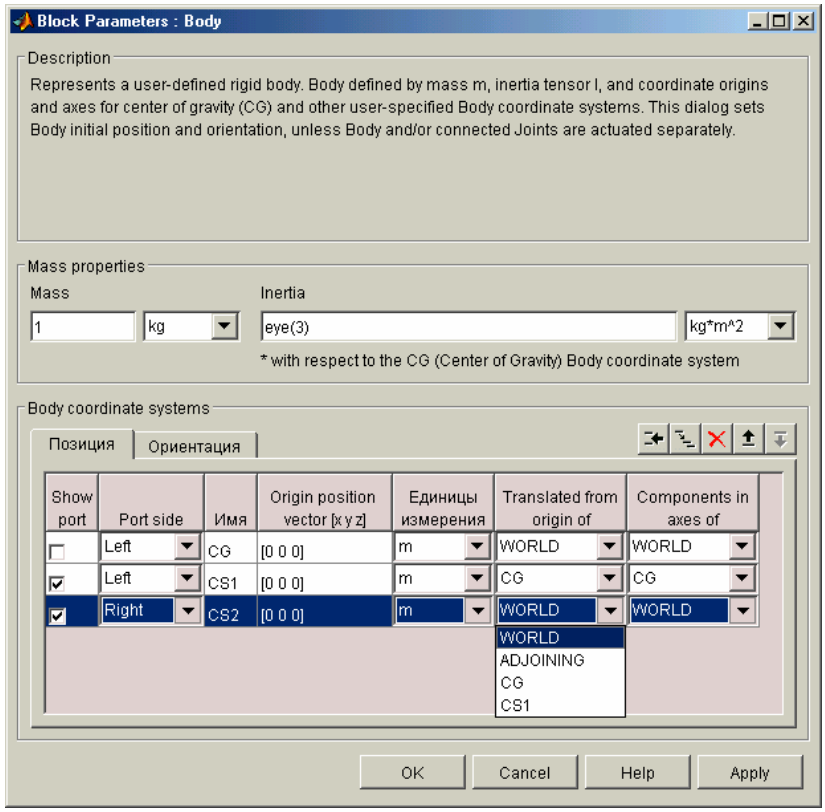


Рис. 11.4. Окно настраивания блока Body. Вкладка Позиция

Оно содержит такие параметры настраивания:

`Mass` (Масса) здесь указывается величина массы той части механизма, которая представляется как твердое тело

`Inertia` (Моменты инерции) тут задается квадратная матрица (3×3) моментов инерции тела относительно ортогональных осей, жестко связанных с телом и проходящих через центр тяжести тела

В поле Body coordinate systems (Системы координат, связанные с телом) расположены две вкладки: Позиция и Ориентация.

Во вкладке Позиция (рис. 4) расположена таблица ввода координат начала систем координат, связанных с телом. По умолчанию, эта таблица содержит три строки и позволяет задать три координатные системы, связан-

ные с телом:
- систему координат CG, начало которой совмещено с центром тяжести тела;
- систему координат CS1, "привязанной" к левому порту ("входу") блока Body;
- систему координат CS2, "привязанной" к правому порту ("выходу") блока Body.

В каждой строке предусмотрено введение 7 характеристик соответствующей системы координат:

Show port	(Показать порт) установка (или сброс) флажка в этой колонке позволяет ввести (или убрать) на изображении блока изображение порта, связанного с соответствующей системой координат тела
Port side	(Сторона порта) позволяет установить изображение порта слева или справа на изображении блока
Имя	здесь устанавливается идентификатор вводимой системы координат
Origin position vector [x, y. z]	(Вектор положения начала) содержит вектор координат начала соответствующей системы координат
Единицы измерения	здесь устанавливается единицы измерения длины, в которых установлены координаты начала
Translated from origin of	(Отсчитывается от начала системы координат…) указывается имя (идентификатор) системы координат, от начала которой отсчитываются координаты устанавливаемой системы координат
Components in axes of	(Компоненты в осях системы координат …) указывается имя (идентификатор) системы координат, в проекциях на оси которой установлены координаты начала устанавливаемой системы координат

Вкладка Ориентация (рис. 5) устроена аналогичным образом и позволяет установить начальную угловую ориентацию вводимой системы координат. Отличия заключаются в следующем. Вместо вектора координат начала в четвертой колонке вводится вектор углов поворота вводимой системы координат относительно исходной, имя которой указывается в шестой колонке. При этом принимаемая последовательность поворотов вокруг координатных осей указывается в седьмой колонке.

Каждому месту соединения тела (блока Body) с другим телом (блоком Body) соответствует своя отдельная система координат CS. Количество точек соединения тела с другими телами (а, значит, и количество связанных с телом систем координат) можно увеличить или уменьшить, пользуясь значками на панели инструментов, находящейся в поле Body coordinate systems.

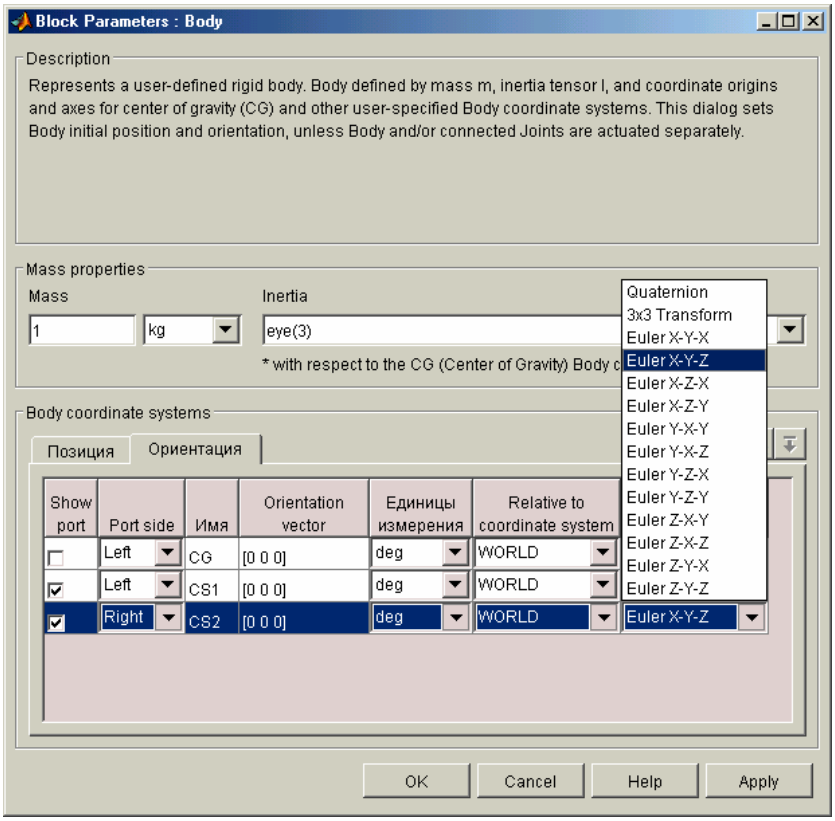


Рис. 11.5. Окно настраивания блока Body. Вкладка Ориентация

На рис. 5 показан список возможного выбора систем координат отсчета. В него входят инерциальная система отсчета WORLD, все имеющиеся на вкладке системы координат, а также система ADJOINING, под которой понимается система координат, связанная жестко с тем сочленением, которое подсоединено к телу.

Раздел Joints

В разделе Joints находятся 15 блоков различных видов сочленений (рис. 6) и два подраздела Disassembled Joints (Разобранные сочленения) и Massless Connectors (Безинерционные соединители).

Блоки этого подраздела имеют два обязательных порта, с помощью которых они подсоединяются к двум блокам из раздела Bodies. Один из этих портов отмечен индексом B (Base – Основной), другой – индексом F (Follower - Следующий). Первый предназначен для соединения с блоком Body, представляющим первое (основное) тело, второй – для подсоединения к следующему телу в связанной цепи тел, составляющих механизм.

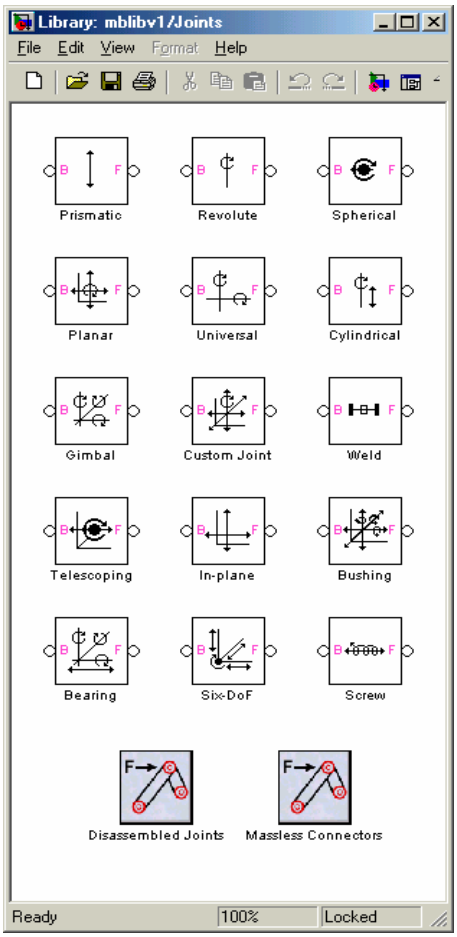


Рис. 11.6. Содержимое раздела Joints

Рассмотрим некоторые наиболее простые и важные блоки.

Начнем с блока Prismatic. Он обеспечивает одну поступательную степень свободы вдоль оси, указанной во вкладке Оси окна его настраивания (рис. 7).

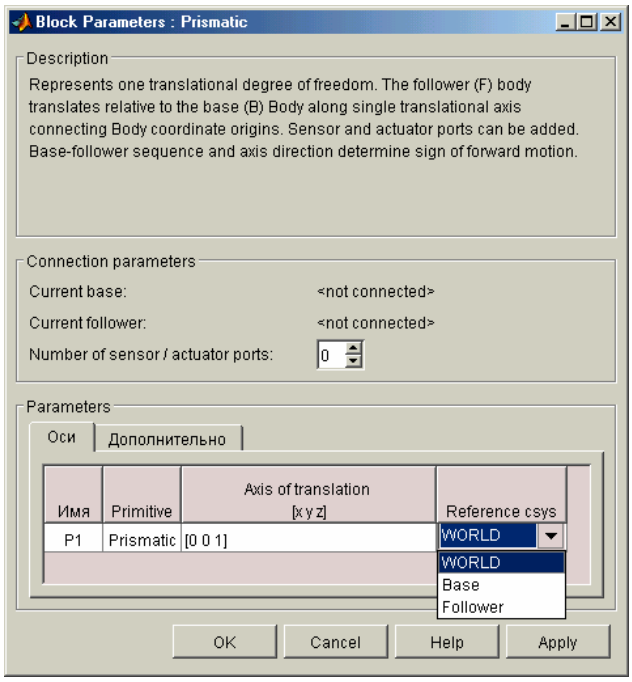


Рис. 11.7. Окно настраивания блока Prismatic

На рис. 7 такой осью свободного перемещения установлена третья ось (z) инерциальной системы координат. Как видно из того же рисунка, имеется возможность связать ось относительного перемещения также с одной из осей первого тела, с которым связан блок Prismatic (выбрав Base из предлагаемого списка), либо с одной из осей системы координат, связанной со вторым телом (выбрав Follower).

В поле Connection parameters (Параметры соединения) указаны три параметра – Current base (Текущая база), Current follower (Текущее ведомое тело) и Number of sensor/actuator ports (Число портов для измерителей и возбудителей движения). Первые два параметра не устанавливаются пользователем и указывают название блока тела, к которому подсоединен соответствующий порт блока Joint. Если блок Joint не подсоединен к телам, то, как видно из рис 7 напротив этих параметров появляется запись <not connected> (не соединен).

Образуем простейшую цепь из трех блоков Ground, Prismatic и Body (рис. 8), одновременно установив в блоке Prismatic два дополнительных порта для подсоединения блоков Actuator и Sensor. Теперь окно настаивания блока Prismatic будет выглядеть так, как показано на рис. 9.

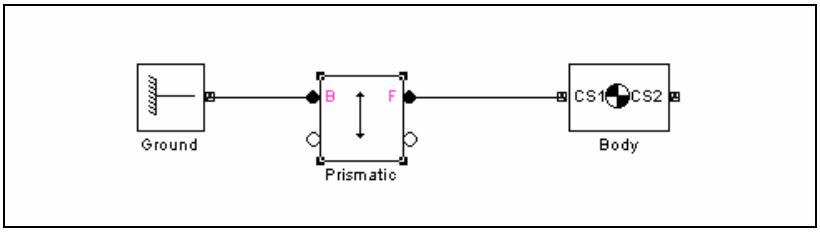


Рис. 11.8. Простейшая механическая цепь

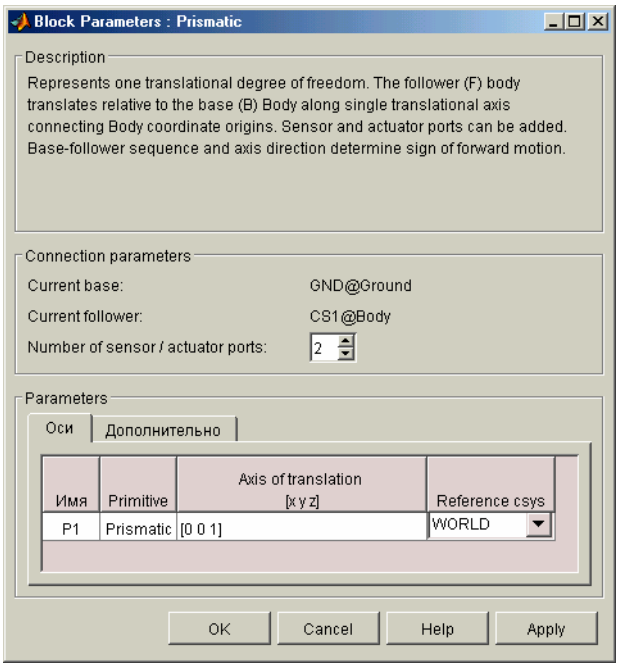


Рис. 11.9. Окно настраивания подсоединенного блока Prismatic

Как видим, теперь текущая база автоматически установлена GND@Ground, что означает, что база отождествлена с системой координат GND блока Ground, а ведомое тело Follower (CS1@Body) связано с системой координат CS1 блока Body.
При этом на изображении блока Prismatic появились изображения двух дополнительных портов. С ними теперь можно соединить блоки измерителей (Sensor) и (или) возбудителей (Actuator) движения.

Окно настраивания (рис. 10) следующего блока - Revolute (цилиндрический шарнир) – практически не отличается от предыдущего. Отличие лишь в том, что в нем устанавливается направление оси вращения тела Follower относительно тела Base.

Следует заметить, что элементарное сочленение типа Prismatic имеет внутреннее обозначение P, а сочленение типа цилиндрического шарнира – обозначение R.

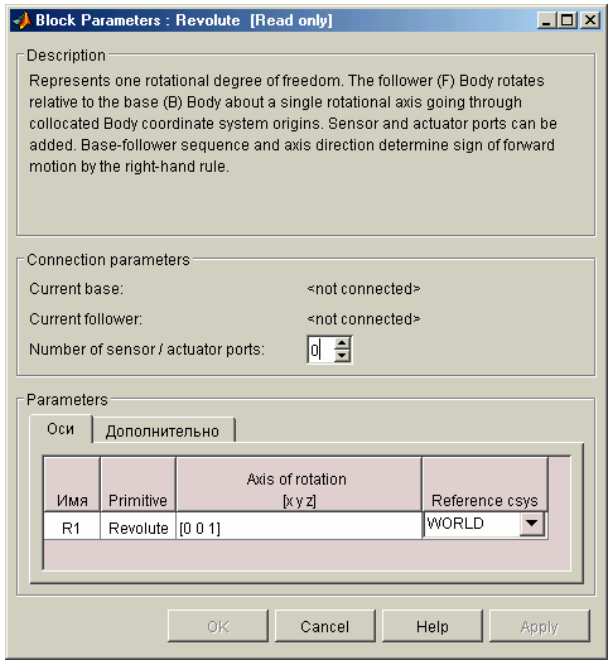


Рис. 11.10. Окно настраивания блока Revolute

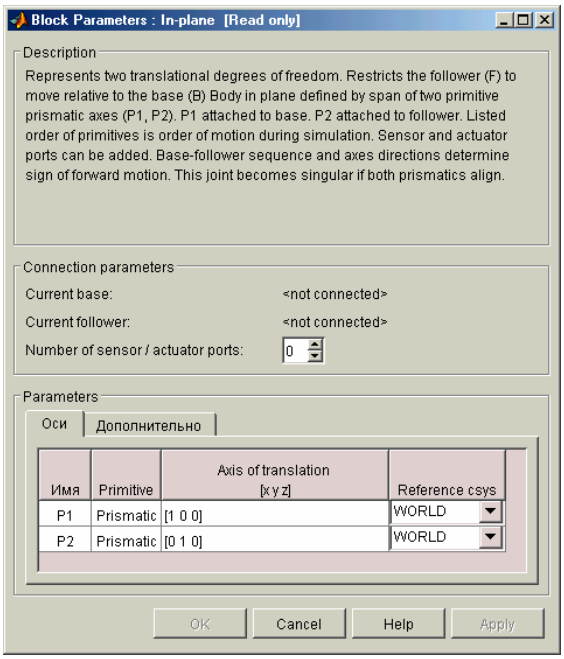


Рис. 11.11. Окно настраивания блока In-plane

Блок In-plane обеспечивает свободу относительного поступательного движения двух тел в плоскости осей, направления которых установлено в окне настраивания (рис. 11). Нетрудно убедиться, что он представляет собой последовательное соединение двух элементарных сочленений P1 и P2 типа Prismatic. Первое из них (P1) обеспечивает свободу перемещения второго (P2) вдоль оси, направление которой указывается в первой строке вкладки Оси окна настраивания. Во второй строке той же вкладки устанавливается направление второй оси свободного перемещения.

Блок Universal обеспечивает свободу углового перемещения тела Follower относительно тела Base относительно двух осей, задаваемых в окне его настраивания (рис. 12). Он представляет собой последовательное соединение двух элементарных сочленений R1 и R2 типа Revolute.

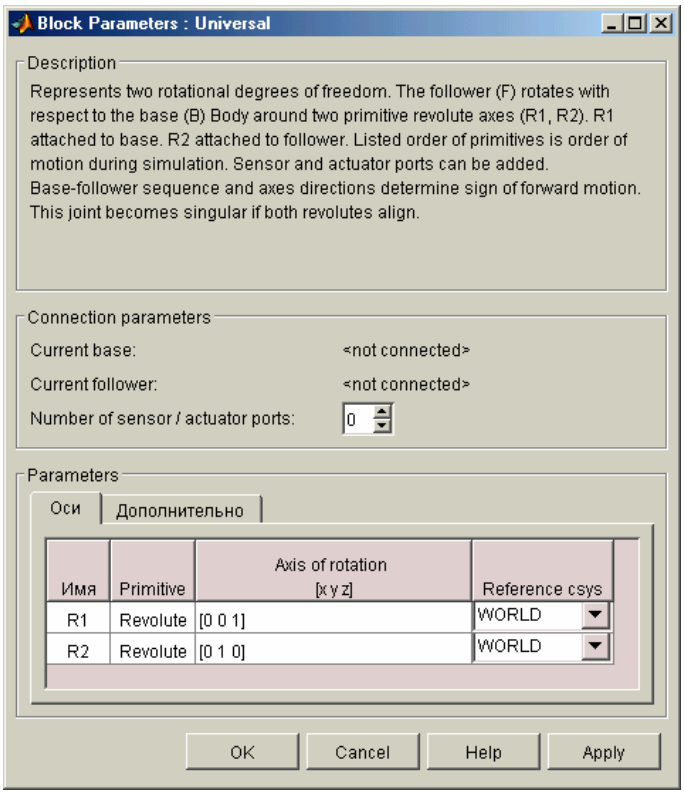


Рис. 11.12. Окно настраивания блока Universal

Следующий блок Gimbal (карданов подвес) представляет собой последовательное соединение трех элементарных сочленений R1, R2 и R3 типа Revolute и обеспечивает свободу углового перемещения одного тела относительно другого вокруг трех, в общем случае некомпланарных осей, указанных в окне настраивания (рис. 13).

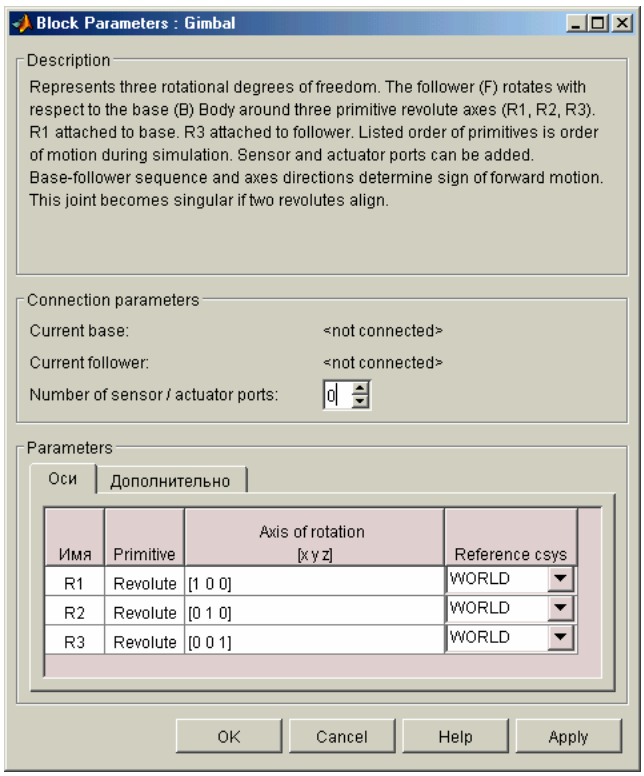


Рис. 11.13. Окно настраивания блока Gimbal

Блок Spherical (Сферический шарнир), как и предыдущий блок, обеспечивает три угловые степени свободы относительного перемещения двух тел. Отличия заключаются в следующем. Во-первых, в блоке Spherical нет явно выраженных осей вращения, и поэтому они не устанавливаются в его окне настраивания. Во вторых, вследствие этой особенности, к блоку Spherical не может быть подсоединен блок возбуждения движения (Actuator), а параметры относительных поворотов тел не могут быть представлены в углах поворотов, а лишь в виде вектора составляющих кватерниона поворота второго тела (Follower) относительно первого (Base).

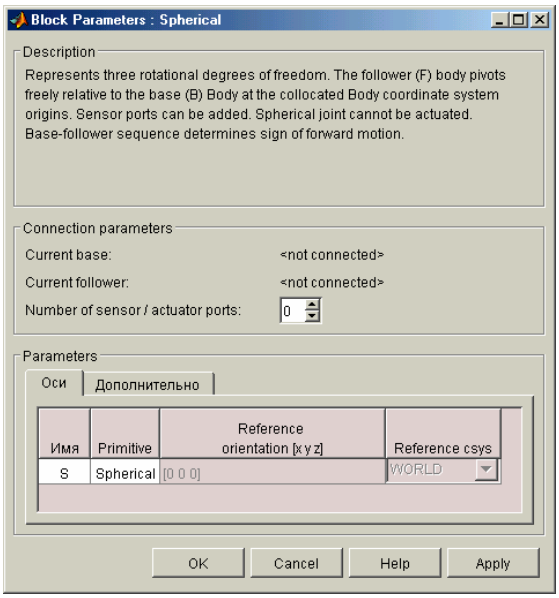


Рис. 11.14. Окно настраивания блока Spherical

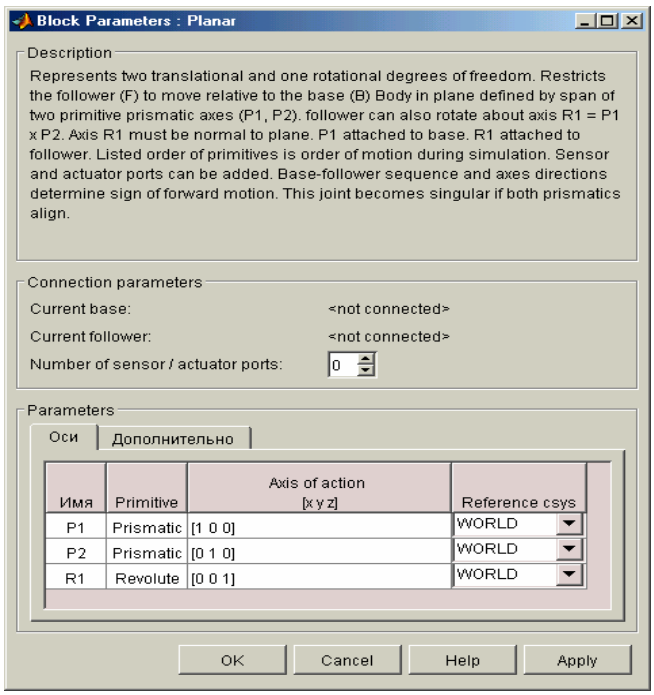


Рис. 11.15. Окно настраивания блока Planar

С помощью блока Planar (Плоское движение) можно обеспечить такое сочленение двух тел, когда одна из плоскостей ведущего и ведомого тела сохраняется неизменной, а точки ведомого тела могут занимать любое положение в соответствующей плоскости ведущего тела. Блок представляет собой последовательное соединение трех элементарных сочленений – двух призматических P1 и P2 и одного цилиндрического шарнира R1. Для обеспечения плоского движения необходимо, чтобы три указанные в блоке настраивания (рис. 15) оси были некомпланарными.

Блок Cylindrical (Цилиндрическое сочленение) представляет возможность имитировать такое соединение двух тел, которое допускает одновременную свободу вращения вокруг указанной оси и поступательного перемещения вдоль этой же оси (рис. 16). Он представляет собой последовательное соединение сочленения P1 типа Prismatic и сочленения R1 типа Revolute.

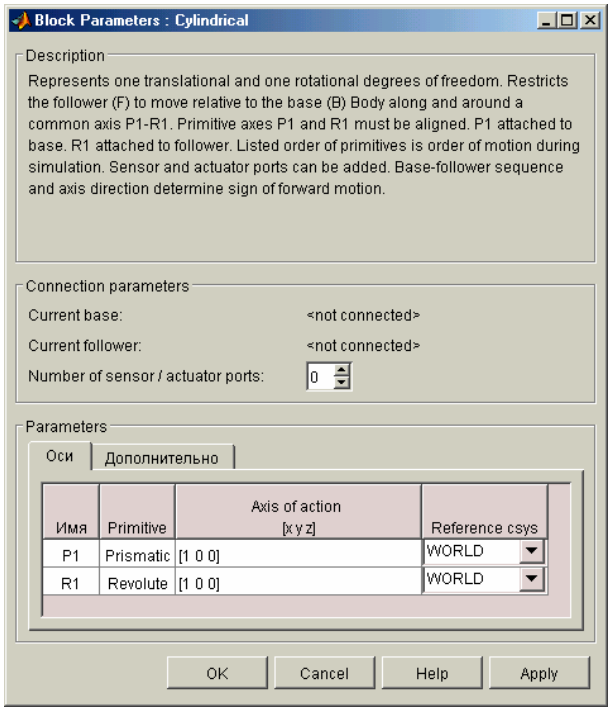


Рис. 11.16. Окно настраивания блока Cylindrical

Особое место занимает блок Custom Joint (Наборное сочленение). Он предоставляет возможность пользователю самому сконструировать произвольную последовательную цепь из элементарных сочленений. Для этого ему предоставляется (в виде списка в первой колонке вкладки Оси, рис. 17) набор из трех примитивов P1, P2, P3 типа Prismatic, трех сочленений R1, R2, R3 типа Revolute и одного примитива S типа Spherical. Ограничения состоят в следующем. Набор должен обеспечивать не более 6 степеней свободы – три угловые и три поступательные. Никакие две оси примитивов Prismatic или две оси примитивов Revolute не должны быть параллельными (в этом случае сочленение вырождается).

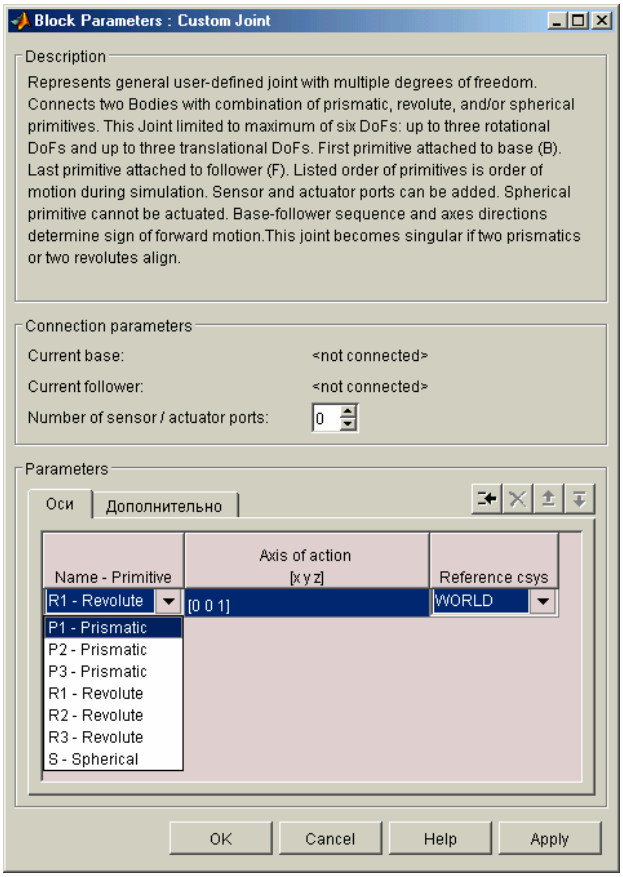


Рис. 11.17. Окно настраивания блока Custom Joint

Блок Weld (Жесткое соединение) служит для имитации жесткого соединения двух тел (рис. 18). Удобен для построения моделей поводковых механизмов.

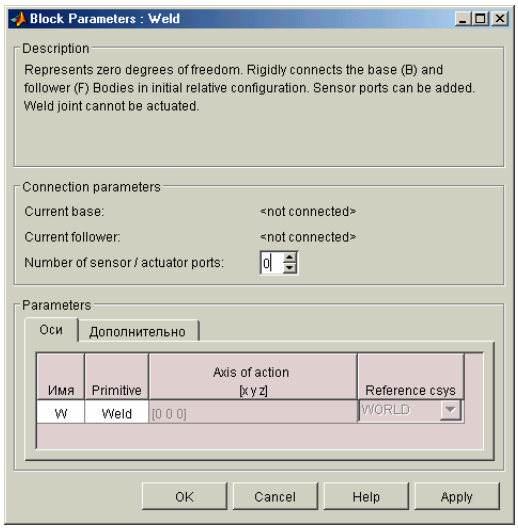


Рис. 11.18. Окно настраивания блока Weld

Блок Bushing (рис. 19) обеспечивает шесть степеней свободы (три поступательные и три угловые) и состоит из последовательно соединенных трех примитивов P1, P2, P3 типа Prismatic, трех сочленений R1, R2, R3 типа Revolute.

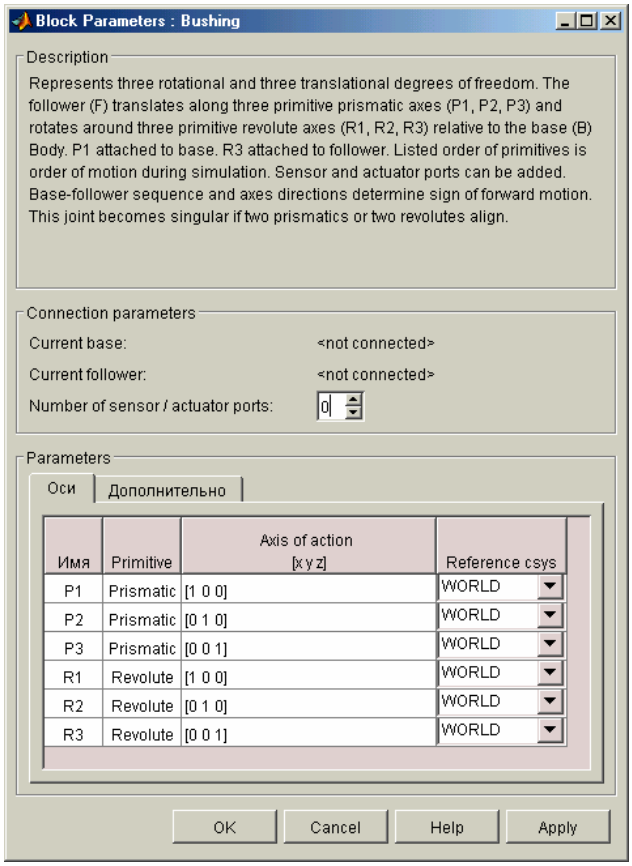


Рис. 11.19. Окно настраивания блока Bushing

Похожую функцию выполняет блок Six-DoF (Шесть степеней свободы). Как видно из рисунка 20, единственным отличием от блока Bushing является то, что для обеспечения трех угловых степеней свободы используется один примитив S типа Spherical вместо трех типа Revolute.

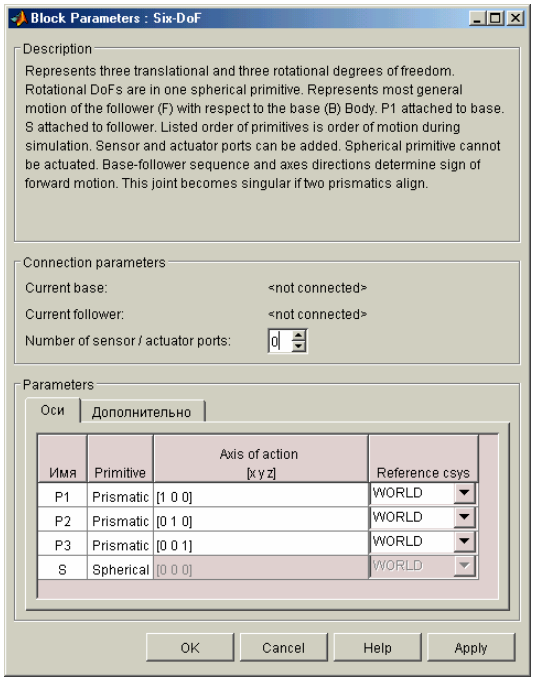


Рис. 11.20. Окно настраивания блока Six-DoF

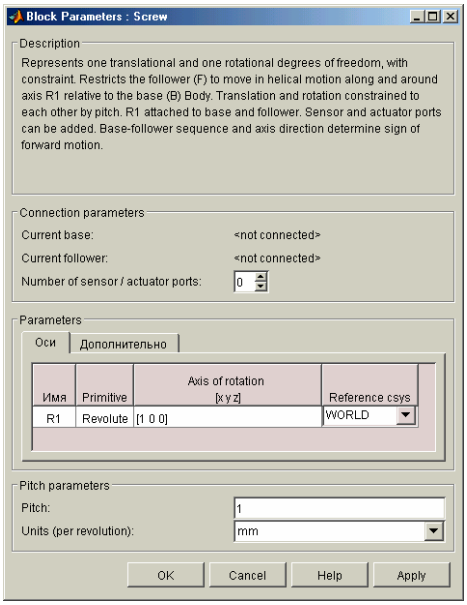


Рис. 11.21. Окно настраивания блока Screw

Особенностью блока Screw (Винт), обеспечивающего винтовую степень свободы относительного перемещения двух тел, является (рис. 21) наличие дополнительного устанавливаемого параметра Pitch (Шаг винта).

Раздел Sensors&Actuators

В этом разделе размещены блоки, позволяющие задать относительные движения тел (блоки Actuator) или измерить характеристики относительного их движения (блоки Sensor). Как ранее было отмечено, блоки типа Joint (Сочленения) могут быть снабжены дополнительными портами для подсоединения к ним блоков Actuator и Sensor. Аналогичная операция возможна и по отношению к блокам Body, Driver и Constraint. Содержимое раздела показано на рис. 22.

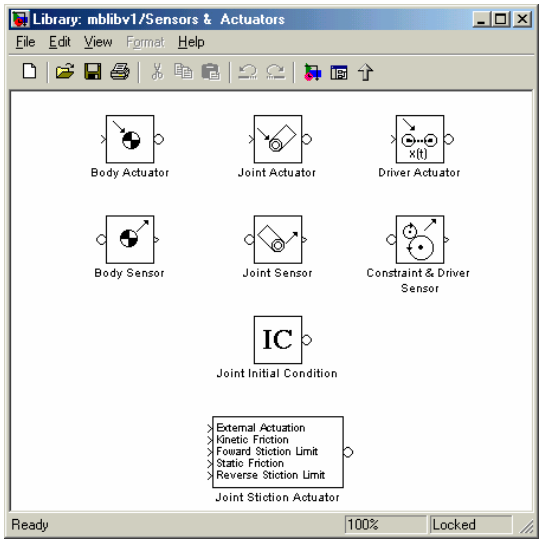


Рис. 11.22. Содержимое раздела Sensors & Actuators

Особенностью рассматриваемых блоков является то, что они являются связующими с обычными S- блоками библиотеки **SIMULINK**, что позволяет использовать возможности этой библиотеки для формирования сигналов, их преобразования и перевода получаемых результатов в рабочее пространство Simulink.

Как видно из рисунка 22, блоки возбудителей (Actuator) и измерителей (Sensor) относительного движения разделяются на три группы:

`Body Actuator` и **`Body Sensor`** — Возбудитель и измеритель движения тела. Предназначены для подсоединения к блокам Body и задают или измеряют абсолютное движение той системы координат, жестко связанной с телом, к которой они подсоединены

`Joint Actuator` и **`Joint Sensor`** — Возбудитель и измеритель движения сочленения. Предназначены для подсоединения к блокам Joint и задают или измеряют относительное движение того при-

митива, который указан в окне настраивания

Driver Actuator и **Driver&Constraint Sensor** — Возбудитель двигателя и измеритель движения двигателя или связи. Предназначены для подсоединения к блокам Driver или Constraint и задают или измеряют относительное движение двигателя или связи, с которыми они соединены

Начнем с рассмотрения блоков второй группы. На рис. 23 и 24 приведены окна настраивания блоков Joint Actuator и Joint Sensor.

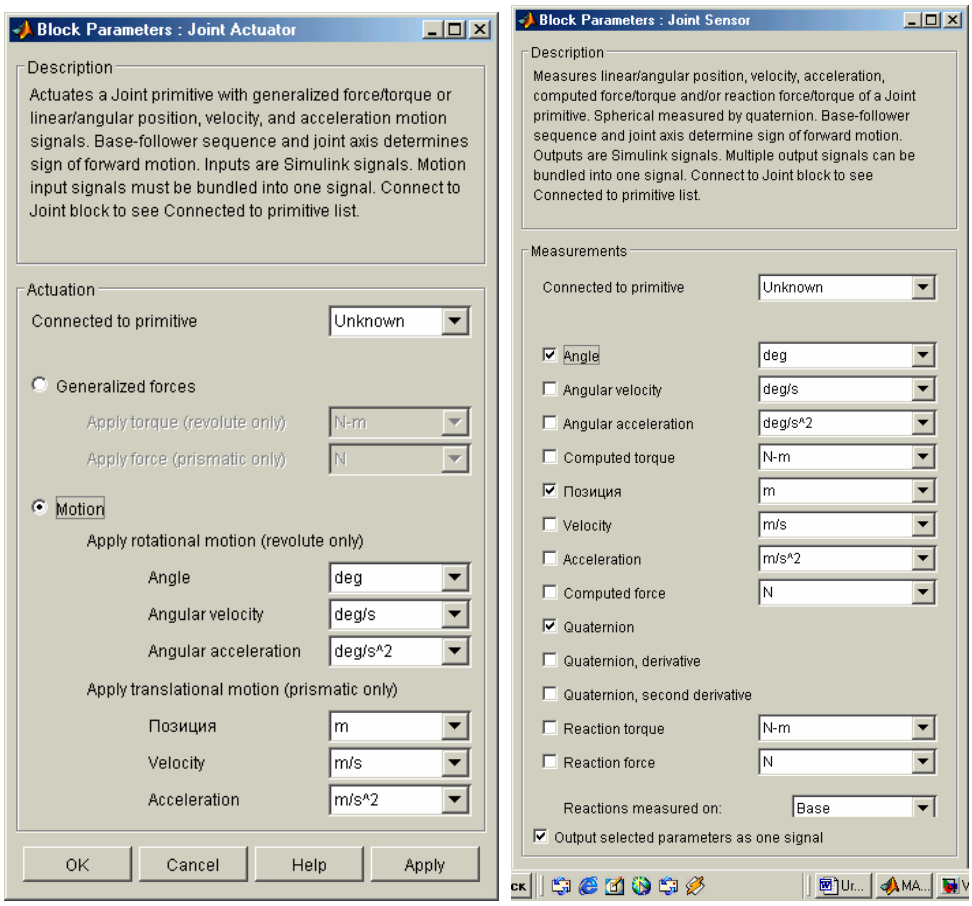


Рис. 11.23. Окно настраивания блока Joint Actuator

Рис. 11.24. Окно настраивания блока Joint Sensor

Из их рассмотрения явствует следующее.

1. С помощью блока Joint Actuator можно в общем случае задать как функцию времени либо силовое взаимодействие между элементами примитива, имя которого указывается в верхнем окошке ввода, либо относительное движение элементов этого примитива. Установление вида возбуждения осуществляется активизацией соответствующего раздела Generalized Forces (Обобщенные силы) или Motion (Движение) в окне настраивания блока кнопкой слева от соответствующего названия раздела.

2. Задание относительного движения частей примитива осуществляется в виде задания векторного сигнала из трех элементов – первый из них определяет относительное перемещение, второй – относительную скорость, а третий – относительное ускорение частей указанного элементарного сочленения.

3. Блок Joint Sensor позволяет в общем случае измерить следующие характеристики относительного движения частей примитива, имя которого устанавливается в верхнем окошке ввода окна настраивания блока:

Angle (Угол) угол поворота выходной части примитива (Follower) относительно его части, соединенной с входом (Base)

Angular velocity (Угловая скорость) – относительная угловая скорость

Angular acceleration (Угловое ускорение) – относительное угловое ускорение

Computed torque (Вычисленный момент) – полный момент сил, вызывающий угловое относительное ускоренное движение

Позиция перемещение выходной части примитива (Follower) относительно его части, соединенной с входом (Base)

Velocity (Скорость) – относительная скорость

Acceleration (Ускорение) – относительное ускорение

Computed force (Вычисленная сила) - полная сила, вызывающая относительное ускоренное движение

Quaternion (Кватернион) – вектор из четырех элементов, характеризующий текущее относительное угловое положение частей примитива

Quaternion, derivative (Производная от кватерниона по времени) – вектор из четырех элементов, являющихся производными по времени от соответствующих элементов кватерниона относительного поворота

Quaternion, second derivative (Вторая производная от кватерниона по времени) – вектор из четырех элементов, являющихся вторыми производными по времени от соответствующих элементов кватерниона относительного поворота

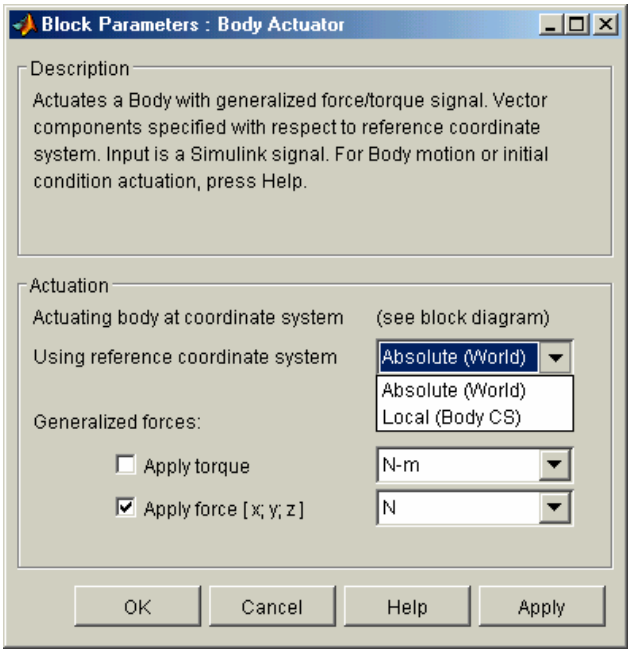
Reaction torque (Момент реакции) – момент реакции относительно оси примитива

Reaction force (Сила реакции) - сила реакции по оси примитива

Примечания.

1. Выбор необходимых для измерения величин производится установлением флажка рядом с названием соответствующей величины.
2. Выходом блока является вектор, элементами которого являются отмеченные величины в том порядке, как они указаны в списке в окне настраивания.
3. При подключении к конкретным элементарным сочленениям ряд окон ввода в окне настраивания становятся неактивными, поэтому выбор ограничивается оставшимися величинами.
4. Примитив типа Spherical позволяет измерить только кватернион поворота и его производные.

Перейдем к рассмотрению блоков возбуждения и измерителей для тел. Их окна настраивания приведены на рис. 25 и 26.



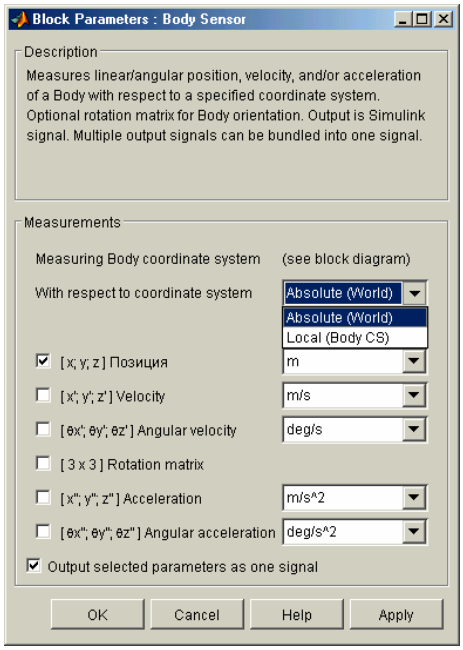


Рис. 11.25. Окно настраивания блока Body Actuator

Рис. 11.26. Окно настраивания блока Body Sensor

Как видим, параметры возбуждения могут быть определены как в абсолютной (инерциальной), так и по отношению к системе координат, связанной с телом в той точке, куда подсоединен блок. При этом возбуждаться могут только силы и моменты, приложенные к точке подсоединения. Измерены же могут быть такие величины:

[x,y,z] Позиция Вектор абсолютного перемещения соответствующей точки тела

[x',y',z'] Velocity (Скорость) – вектор проекций абсолютной скорости соответствующей точки тела

[θx',θy',θz'] Angular velocity (Угловая скорость) – вектор проекций абсолютной угловой скорости тела

[3×3] Rotation matrix Матрица направляющих косинусов углового положения тела

[x",y",z"] Acceleration (Ускорение) – вектор проекций абсолютного ускорения соответствующей точки тела

[θx",θy",θz"] Angular (Угловое ускорение) – вектор проекций абсолютного углового ускорения тела

acceleration

Как и ранее, величины, которые необходимо измерить, должны быть отмечены флажками слева, а выходом блока является вектор всех отмеченных измеряемых величин в порядке, указанном в окне настраивания.

На рисунках 27 и 28 показаны окна настраивания блоков возбуждения и измерения связей.

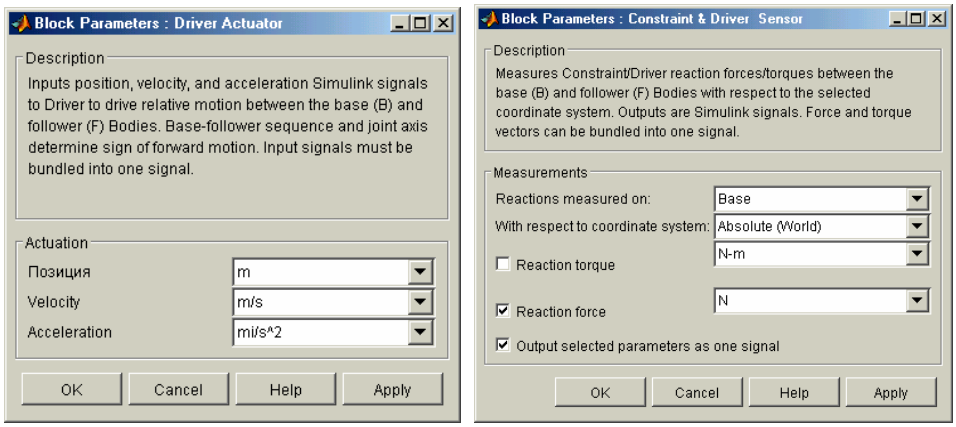


Рис. 11.27. Окно настраивания блока Driver Actuator

Рис. 11.28. Окно настраивания блока Constraint & Driver Sensor

Из них следует:

1) возбуждаться могут только блоки нестационарных связей (Driver), а измерители (Sensor) могут быть подсоединены к любым блокам связей;

2) величины, которые задаются как возбуждение, представляют собой векторы из трех элементов (перемещение, скорость и ускорение); при этом перемещения могут быть линейными (как показано на рис. 27) или угловыми (если имитируется нестационарная связь по углу);

3) измерители (Sensor) могут измерять только силу и момент силы реакции в связи.

Помимо указанных блоков в раздел входят два особых блока. Один из них – блок Joint Initial Condition (Начальные Условия сочленения) - позволяет задать начальное относительное положение и начальную относительную скорость (рис. 29) двух частей того элементарного сочленения (Revolute или Prismatic), к которому он подсоединен.

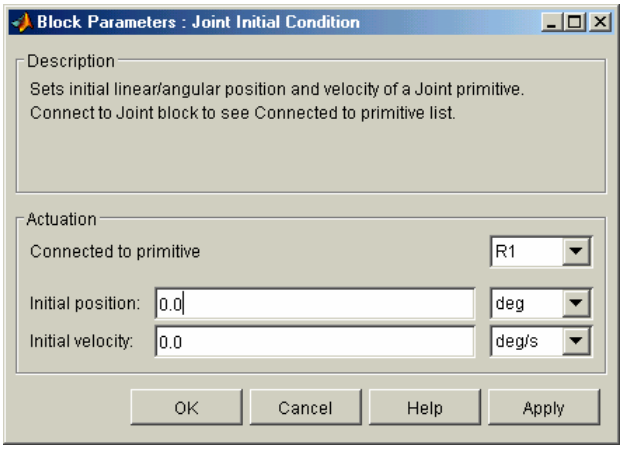


Рис. 11.29. Окно настраивания блока Joint Initial Condition

Следующий блок - Joint Stiction Actuator (Имитатор "захвата" сочленения) – позволяет моделировать силы и моменты сил вязкого и сухого трения в оси элементарного сочленения, включая явление "жесткого" сцепления его частей силами сухого трения. Окно настраивания этого блока приведено на рис. 30.

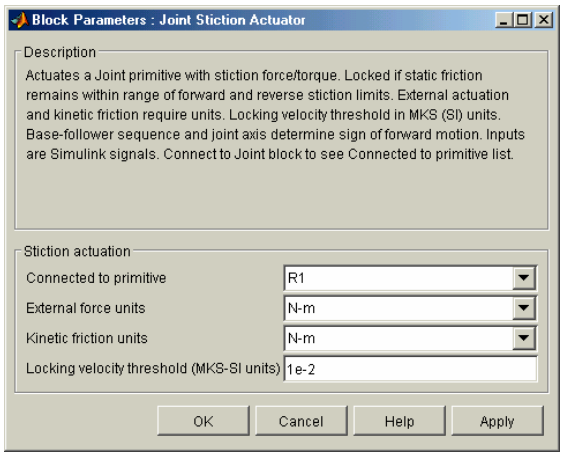


Рис. 11.30. Окно настраивания блока Joint Stiction Actuator

Раздел Constraints & Drivers

Этот раздел содержит блоки, имитирующие наложенные между телами связи – стационарные (блоки Constraint) или нестационарные (блоки Driver). Содержимое раздела приведено на рис. 29.

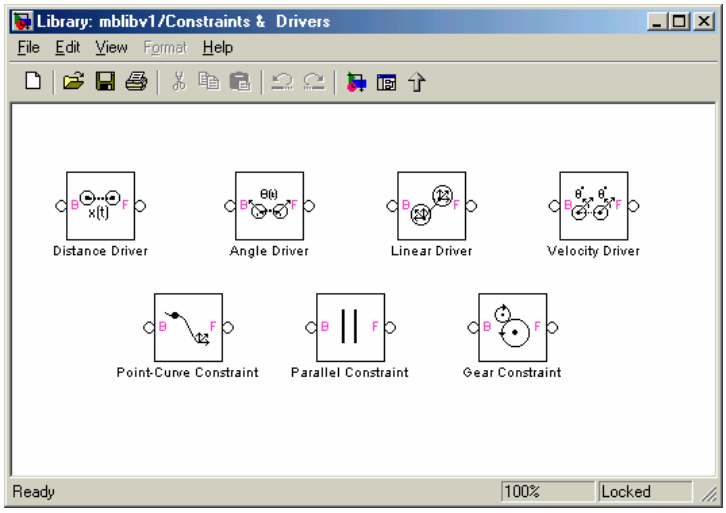


Рис. 11.31. Содержимое раздела Constraints & Drivers

Дадим краткую характеристику блоков.

Distance Driver Задает нестационарную связь между подсоединенными к блоку телами в виде зависимости от времени расстояния между началами систем координат, связанными с этим телами

Angle Driver Задает нестационарную связь между подсоединенными к блоку телами в виде зависимости от времени угла между указанными двумя осями систем координат, связанными с этим телами

Linear Driver Задает нестационарную связь между подсоединенными к блоку телами в виде зависимости от времени проекции расстояния между началами систем координат, связанными с этим телами, на указанную ось инерциальной системы отсчета

Velocity Driver Задает нестационарную связь между подсоединенными к блоку телами в виде зависимости от времени линейной комбинации проекций векторов линейных и угловых скоростей систем координат, связанными с этим телами, на указанные оси

Point Curve Constraint Задает стационарную связь между подсоединенными к блоку телами в виде заданных своими координатами в системе координат, связанной с ведомым (Follower) телом, точек кривой расстояния начала системы координат, связанной с ведущим (Body) телом; кривая определяется сплайновой интерполяцией указанных точек

Parallel Constraint Задает стационарную такую связь между подсоединенными к блоку телами, что указанная ось системы координат, связанной с ведущим телом, остается во все время движения параллельной одноименной оси системы координат, связанной с ведомым телом

Gear Constraint Задает стационарную связь между подсоединенными к блоку телами как зубчатую передачу с задаваемыми радиусами делительных окружностей

Примечания. 1. Задание временной зависимости нестационарной связи в блоке Driver осуществляется через подсоединение к последнему блока Driver Actuator.
2. Если к блоку Driver не подсоединен блок Driver Actuator, он реализуют соответствующую стационарную связь. Например, Distance Driver реализует движение начала координат тела Follower по сфере с центром в начале координат тела Body. При этом радиус сферы определяется заданным начальным положением этих систем координат.

Раздел Utilities

Содержимое раздела показано на рис. 32.

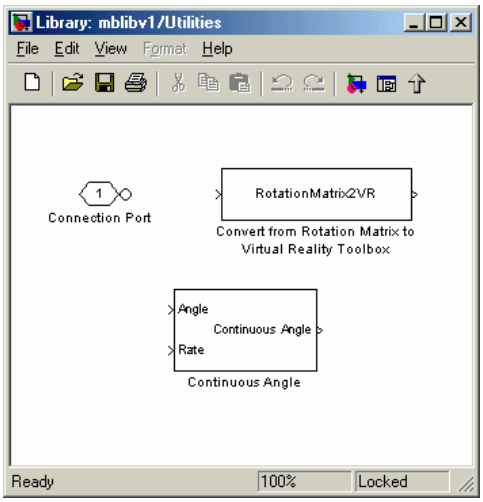


Рис. 11.32. Содержимое раздела Utilities

Блок Connection Port (Соединительный порт) играет в модели SimMechanics ту же роль, какую играют блоки In и Out в обычной S-модели. Благодаря ему, в модели SimMechanics можно создавать подсистемы, формируя входы и выходы этими блоками. Окно настраивания приведено на рис. 33. Параметр настраивания один – место расположение внешнего изображения порта справа или слева на изображении подсистемы.

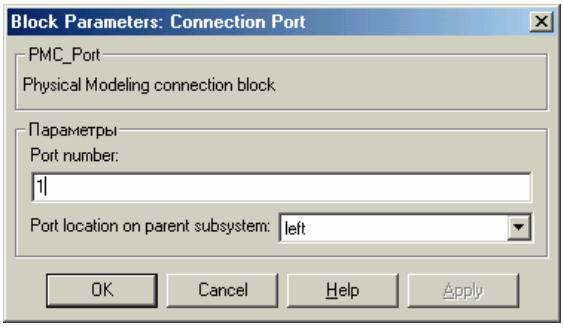


Рис. 11.33. Окно настраивания блока Connection Port

При использовании блоков Joint Sensor для измерения угла относительного поворота следует иметь в виду, что измеритель угла выдает сигнал, пропорциональный измеряемому углу только в диапазоне $\pm\pi$ радиан. При превышении этого диапазона выдаваемое значение угла претерпевает разрыв, равный 2π радиан. Чтобы получить реальное значение угла поворота в этом случае следует использовать блок Continuous Angle. При этом в число измеряемых блоком Joint Sensor величин, помимо угла, следует включить и скорость изменения этого угла и подать соответствующий сигнал на вход Rate. Тогда на выходе блока получится непрерывный сигнал угла. Параметрами настраивания блока Continuous Angle (рис. 34) являются единицы измерения угла и угловой скорости.

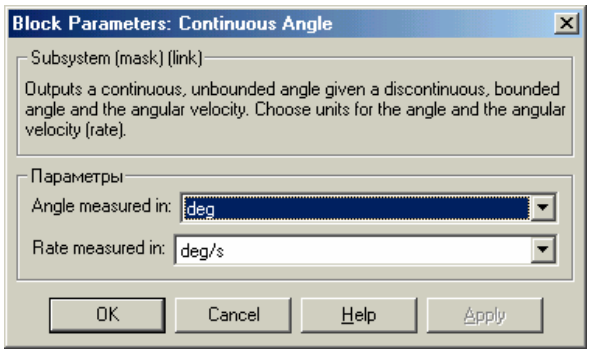


Рис. 11.34. Окно настраивания блока Continuous Angle

11.2. Модель уравновешенного свободного гироскопа

Рассмотрим процесс построения S-модели с помощью библиотеки **SimMechanics** на простейших примерах.

Начнем с модели уравновешенного гироскопа. Под уравновешенным гироскопом понимают устройство, состоящее из одного твердого тела, центр тяжести которого неподвижен в инерциальном пространстве, а само тело может произвольно поворачиваться в пространстве относительно этой неподвижной его точки (точки подвеса), при этом телу сообщено быстрое вращение вокруг одной из осей, жестко связанных с телом (оси собственного вращения гироскопа).

Обеспечить три угловые степени свободы телу можно с помощью двух видов сочленений, предусмотренных библиотекой **SimMechanics** (раздел Joints), – Spherical (Сферический шарнир) и Gimbal (карданов подвес). Удобнее использовать блок Gimbal. Этот вид сочленения представляет собой соединение трех элементарных сочленений вида Revolute (Цилиндрический шарнир), каждый из которых обеспечивает вращение вокруг одной из трех взаимно-перпендикулярных осей.

Модель такого гироскопа представлена на рис. 35.

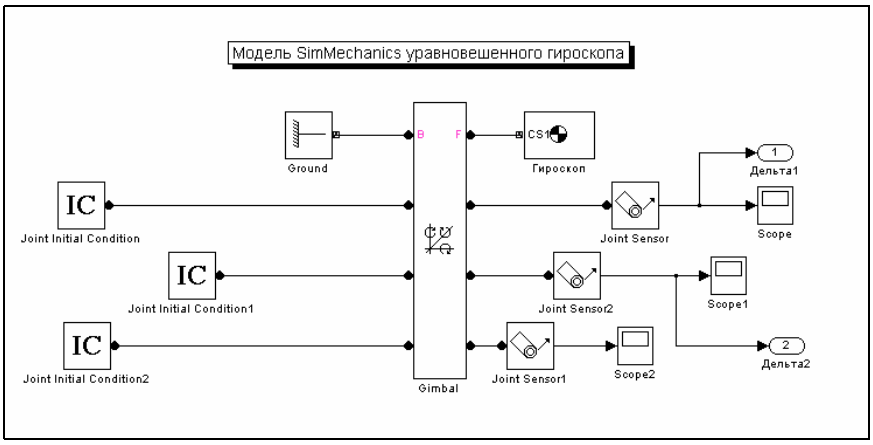


Рис. 11.35. Модель SimMechanics уравновешенного гироскопа

Она состоит из блока Ground инерциальной системы отсчета, блока Gimbal обеспечения трехстепенного подвеса гироскопа, блока Гироскоп (типа Body), трех блоков Joint Initial Condition (IC), каждый из которых устанавливает начальные условия для одного из примитивов R1, R2 и R3 типа Revolute, составляющих подвес Gimbal, и трех измерителей Joint Sensor, каждый из которых подсоединен к одному из указанных примитивов и измеряет угол его относительного поворота.

Для связи с рабочим пространством предусмотрены выходные порты Дельта1 и Дельта2, на которые поступают сигналы, пропорциональные текущим значениям углов поворота гироскопа вокруг осей Z и X соответственно.

Обозначим:

`m`	масса гироскопа
`J`	матрица моментов инерции гироскопа
`OM`	собственная угловая скорость гироскопа (вокруг оси Y)
`omx, omz`	начальные угловые скорости гироскопа вокруг осей X и Z соот-

ветственно

delta10, delta20 начальные углы отклонения оси собственного вращения гироскопа от оси Y инерциальной системы координат (вокруг осей Z и X соответственно)

Ниже (рис. 36…39) приведены окна настраивания основных блоков.

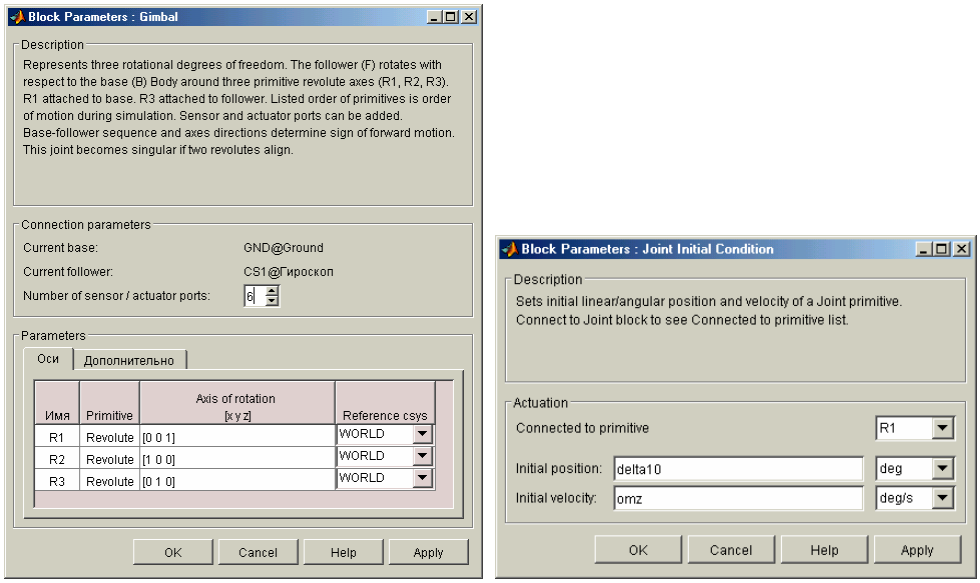


Рис. 11.36. Окно настраивания блока Gimbal

Рис. 11. 37. Окно настраивания блока Joint Initial Condition

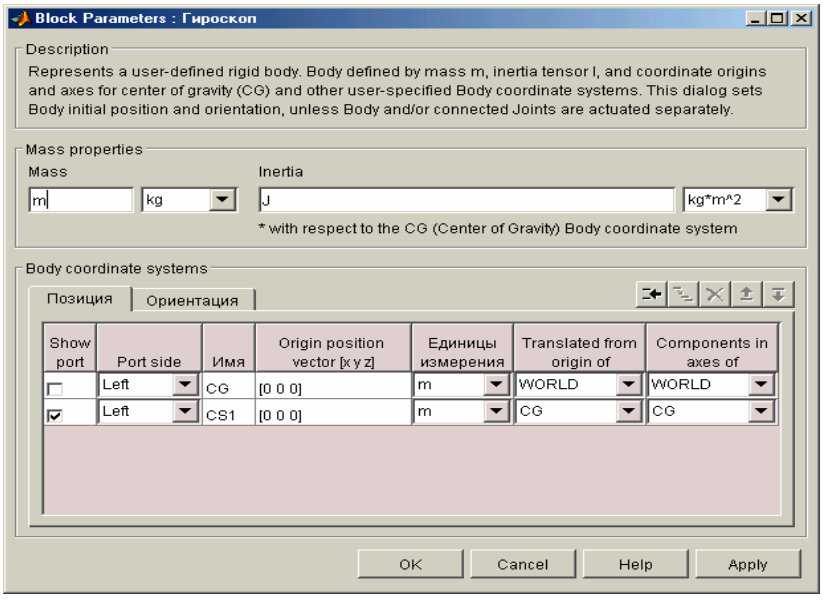


Рис. 11.38. Окно настраивания блока Гироскоп

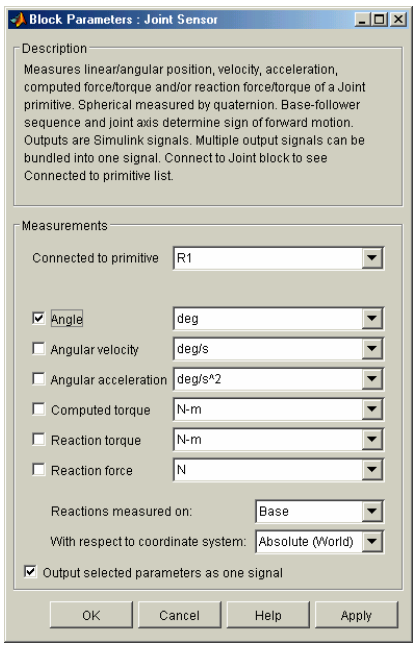


Рис. 11.39. Окно настраивания блока Joint Sensor

Текст управляющей программы **SUG_simMech_upr**, которая осуществляет присвоение значений инерционным характеристикам гироскопа, ввод начальных условий, запуск на выполнение S-модели и выведены в графической форме результатов на экран, приведен ниже.

```
% SUG_simMech_upr

% Лазарев Ю. Ф. 19-04-2004

clc
clear all
% Установка инерционных характеристик гироскопа
m=1;
J=[3 0 0;0 5 0;0 0 3];       %J=[4 0 0;0 5 0;0 0 3];     %J=[4 -0.2 0.1;-0.2 5 -0.2;0.1 -0.2 3];
% Установка начальных условий
OM=20;  omx=1;  omz=0;              delta10=0;  delta20=0;
% Моделирование на модели SimMechanics
sim('SUG_simMech');
% Извлечение данных
t=tout; D1=yout(:,1);   D2=yout(:,2);
% Построение графического вывода результатов
subplot(2,2,1)
plot(D1,D2), grid
set(gca,'FontSize',12)
title('Траектория в картинной плоскости');
xlabel('\delta1 (градусы)'), ylabel('\delta2 (градусы)')
subplot(2,2,2)
axis('off');
h=text(-0.3,1.1,'Уравновешенный гироскоп (модель SimMechanics)','FontSize',14);
h=text(0.1,0.9,'| ','FontSize',12);
h=text(0.2,0.9,num2str(J(1,1)),'FontSize',12);
h=text(0.4,0.9,num2str(J(1,2)),'FontSize',12);
h=text(0.6,0.9,num2str(J(1,3)),'FontSize',12);
h=text(0.8,0.9,'| ','FontSize',12);    h=text(-0.1,0.8,'J = ','FontSize',12);
h=text(0.1,0.8,'| ','FontSize',12);
h=text(0.2,0.8,num2str(J(2,1)),'FontSize',12);
h=text(0.4,0.8,num2str(J(2,2)),'FontSize',12);
h=text(0.6,0.8,num2str(J(2,3)),'FontSize',12);
h=text(0.8,0.8,'| ','FontSize',12);
h=text(0.1,0.7,'| ','FontSize',12);
h=text(0.2,0.7,num2str(J(3,1)),'FontSize',12);
h=text(0.4,0.7,num2str(J(3,2)),'FontSize',12);
h=text(0.6,0.7,num2str(J(3,3)),'FontSize',12);
```

```
h=text(0.8,0.7,'| ','FontSize',12);
h=text(-0.1,0.5,'Начальные углы (градусы)','FontSize',12);
h=text(0.1,0.4,['\delta10 = ',num2str(delta10)],'FontSize',12);
h=text(0.4,0.4,['\delta20 = ',num2str(delta20)],'FontSize',12);
h=text(-0.1,0.2,'Начальные угловые скорости (рад/с)','FontSize',12);
h=text(0.1,0.1,['omx0 = ',num2str(omx)],'FontSize',12);
h=text(0.4,0.1,['omy0 = ',num2str(OM)],'FontSize',12);
h=text(0.7,0.1,['omz0 = ',num2str(omz)],'FontSize',12);
h=text(-0.1,-0.05,'-----------------------------------------------------------------------------------------------');
h=text(-0.1,-0.1,'Программа SUG-simMech-upr       Лазарев Ю. Ф.     19-04-2004');
h=text(-0.1,-0.15,'-----------------------------------------------------------------------------------------------');

subplot(2,2,[3,4])
plot(t,D1,t,D2,'.'),set(gca,'FontSize',12)
title('Изменение углов поворота оси гироскопа со временем');
xlabel('Время (сек)'), ylabel('Углы (градусы)')
legend('\delta1','\delta2',0), grid
```

Далее приводятся результаты работы этой программы.

На рис. 40 представлены результаты моделирования для случая, когда гироскоп является динамически симметричным телом с осью симметрии, совпадающей с осью собственного вращения.

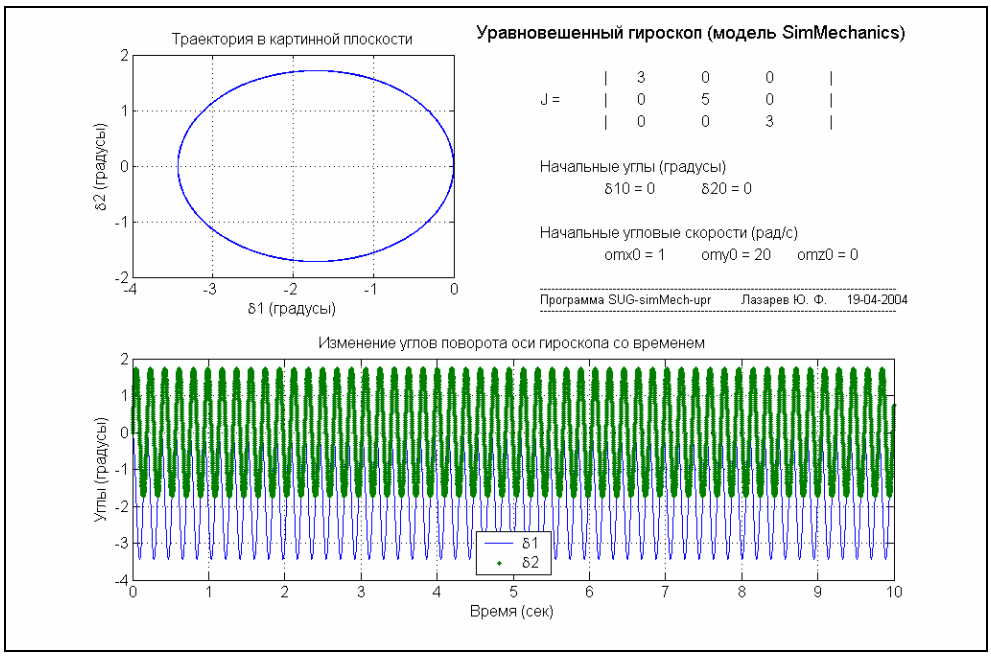


Рис. 11.40. Свободное движение симметричного уравновешенного гироскопа

Рис. 41 показывает результаты для случая несимметричного гироскопа. Наконец, на рис. 42 представлено движение несимметричного и динамически несбалансированного гироскопа.

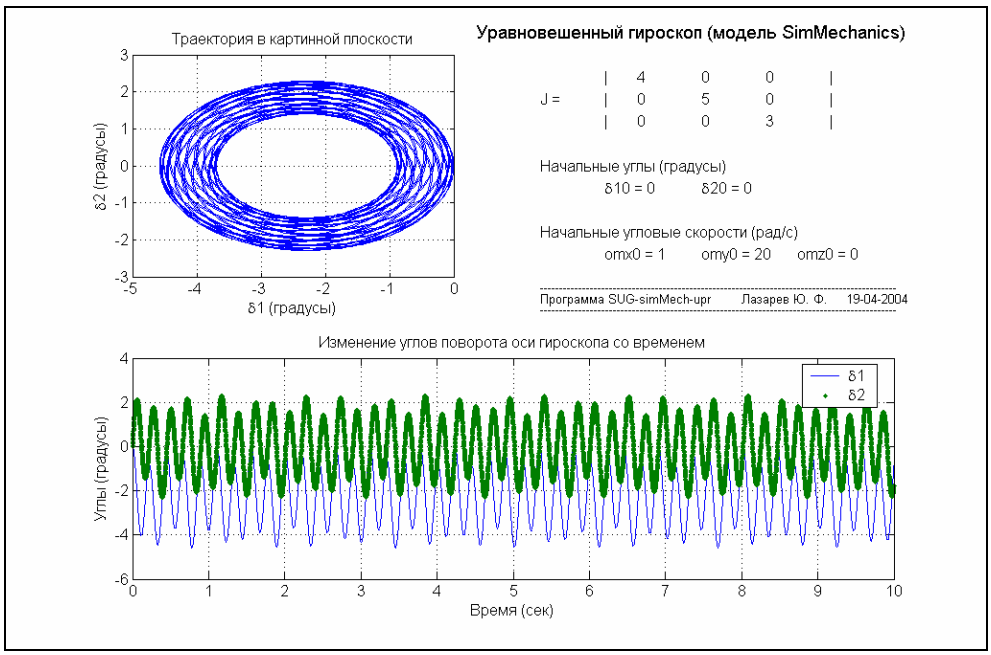


Рис. 11.41. Свободное движение несимметричного уравновешенного гироскопа

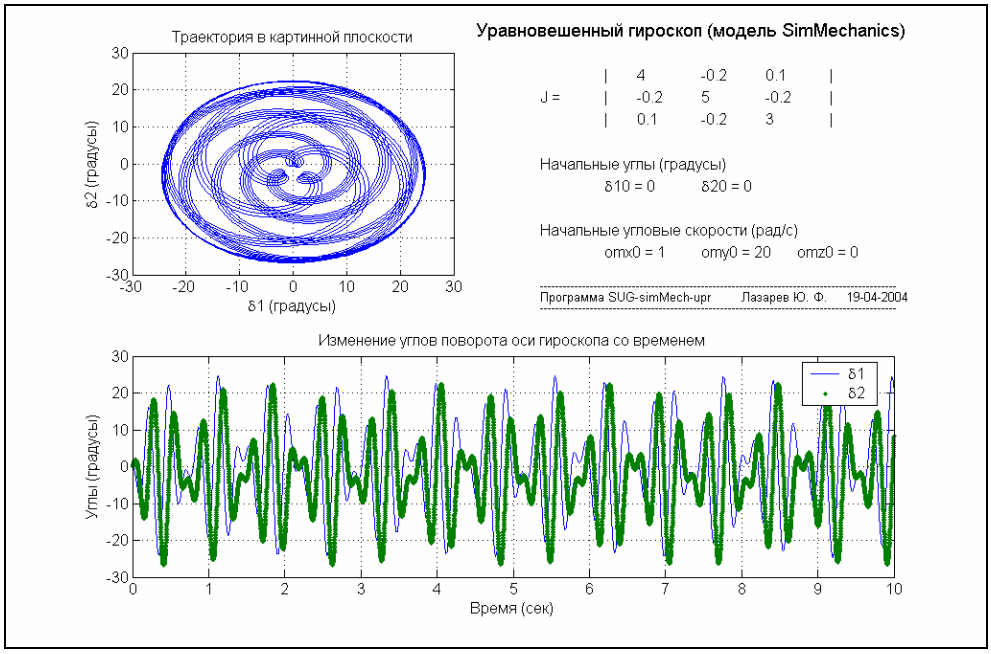


Рис. 11.42. Свободное движение несбалансированного гироскопа

Результаты моделирования хорошо согласуются с результатами теоретического анализа [14].

11.3. Модель кривошипно-шатунного механизма

Перейдем к составлению модели кривошипно-шатунного механизма. Он состоит из ведущего звена (поводка, кривошипа), вращающегося с заданной угловой скоростью вокруг оси Z, ведомого звена (шатуна) и ползуна, перемещающегося поступательно в направляющих, параллельных оси X. Все три тела вместе образуют замкнутую механическую цепь.

Модель может быть реализована в виде схемы, изображенной на рис.

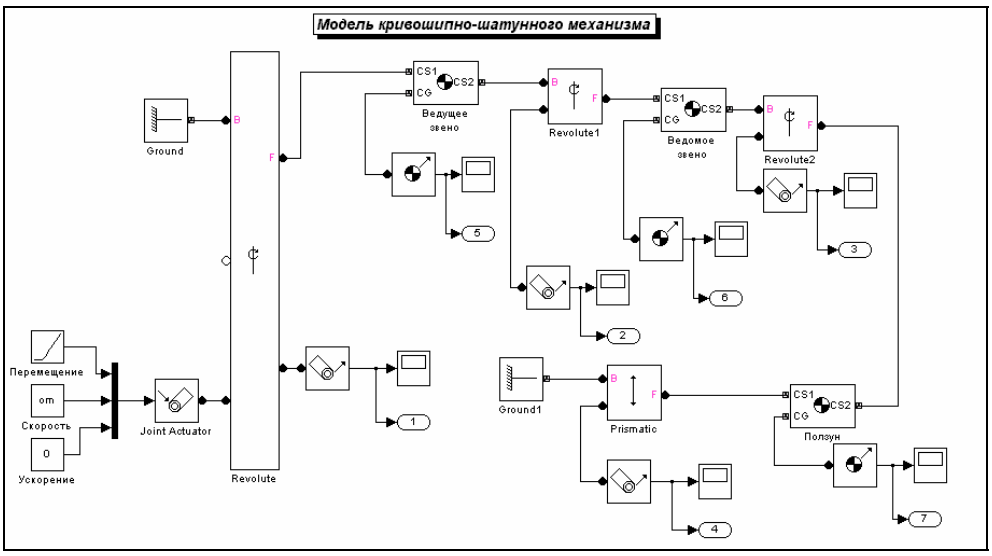


Рис. 11.43. Модель **SimMech_KSM1** кривошипно-шатунного механизма

Она состоит из двух блоков типа Ground, двух блоков типа Body (Ведущее звено, Ведомое звено и Ползун), четырех блоков примитивов сочленений (три блока Revolute и один блок Prismatic), блока Joint Actuator возбудителя вращения ведущего звена, четырех блоков типа Joint Sensor для измерения относительного движения частей четырех сочленений и трех блоков типа Body Sensor для измерения параметров движения тел.

В дальнейшем используем следующие обозначения.

`M1, M2, M3`	Массы ведущего, ведомого звеньев и ползуна соответственно
`J1, J2, J3`	Матрицы моментов инерции указанных тел относительно их центров масс
`L1`	Расстояние от оси вращения кривошипа до шарнира, связывающего его с шатуном
`L2`	Расстояние между двумя шарнирами шатуна (ведомого звена)
`om`	Угловая скорость вращения кривошипа вокруг оси Z
`fi0`	Начальный угол отклонения кривошипа от горизонтальной оси X
`E`	Кратчайшее расстояние от оси вращения кривошипа до оси перемещения ползуна
`Al`	Рассчитываемое начальное значение угла наклона оси шатуна к оси X

С учетом их установлены параметры настройки блоков тел, показанные на рис. 44…48.

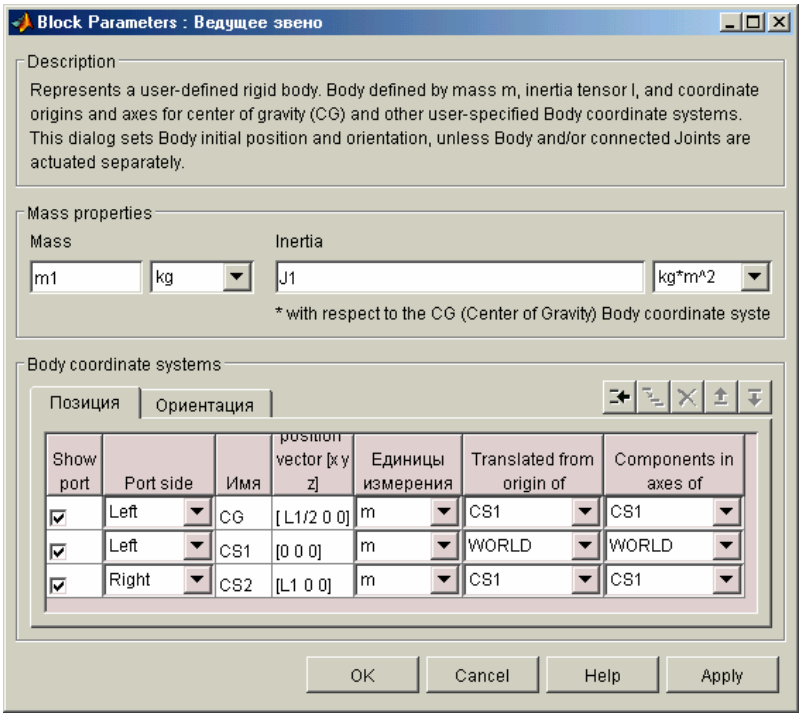


Рис. 11.44. Окно настраивания блока Ведущее звено (вкладка Позиция)

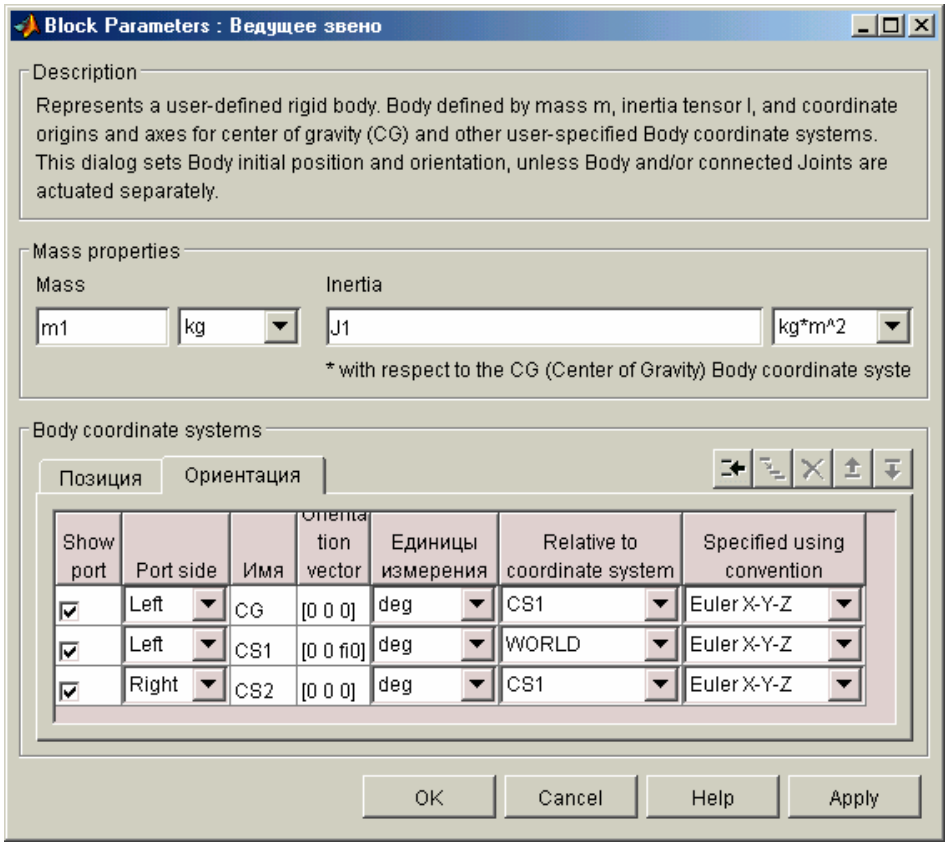


Рис. 11.45. Окно настраивания блока Ведущее звено (вкладка Ориентация)

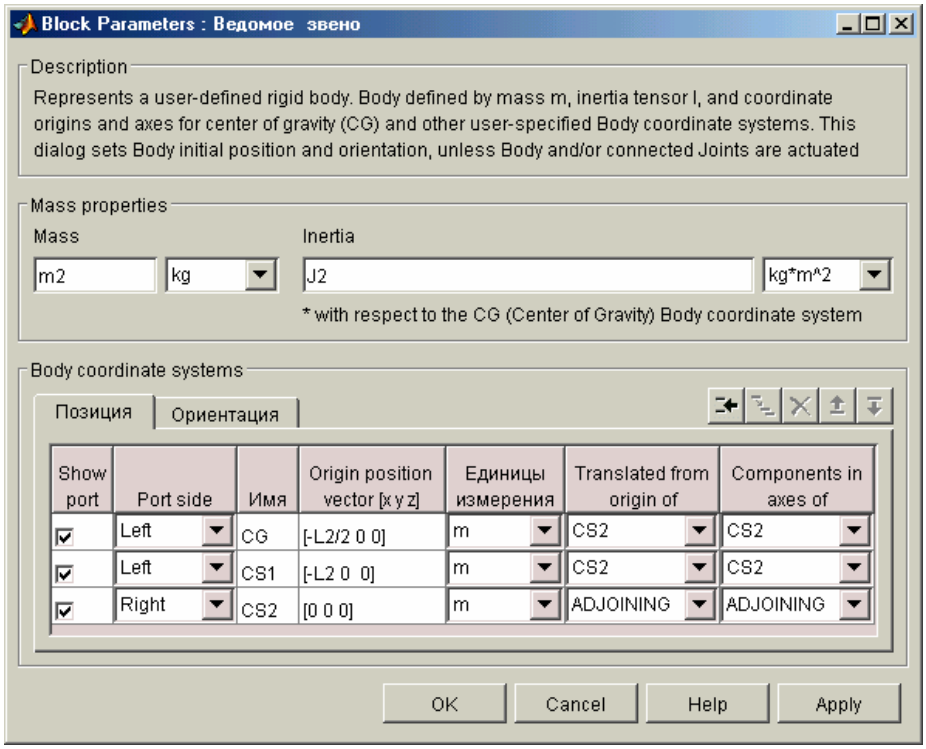


Рис. 11.46. Окно настраивания блока Ведомое звено (вкладка Позиция)

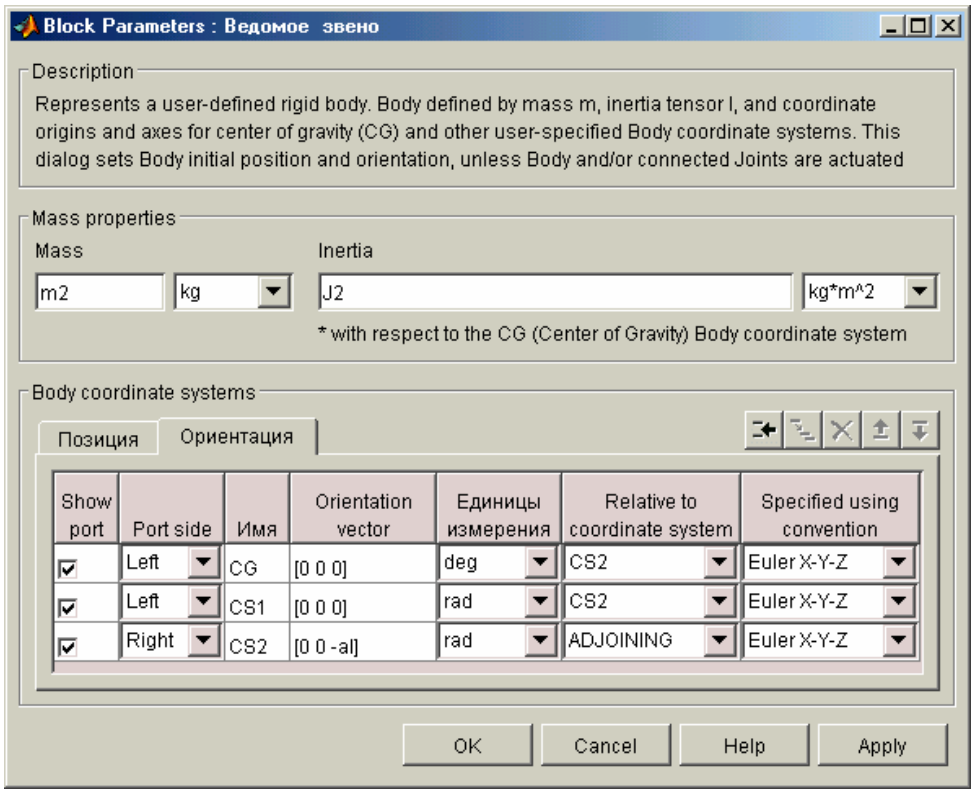


Рис. 11.47. Окно настраивания блока Ведомое звено (вкладка Ориентация)

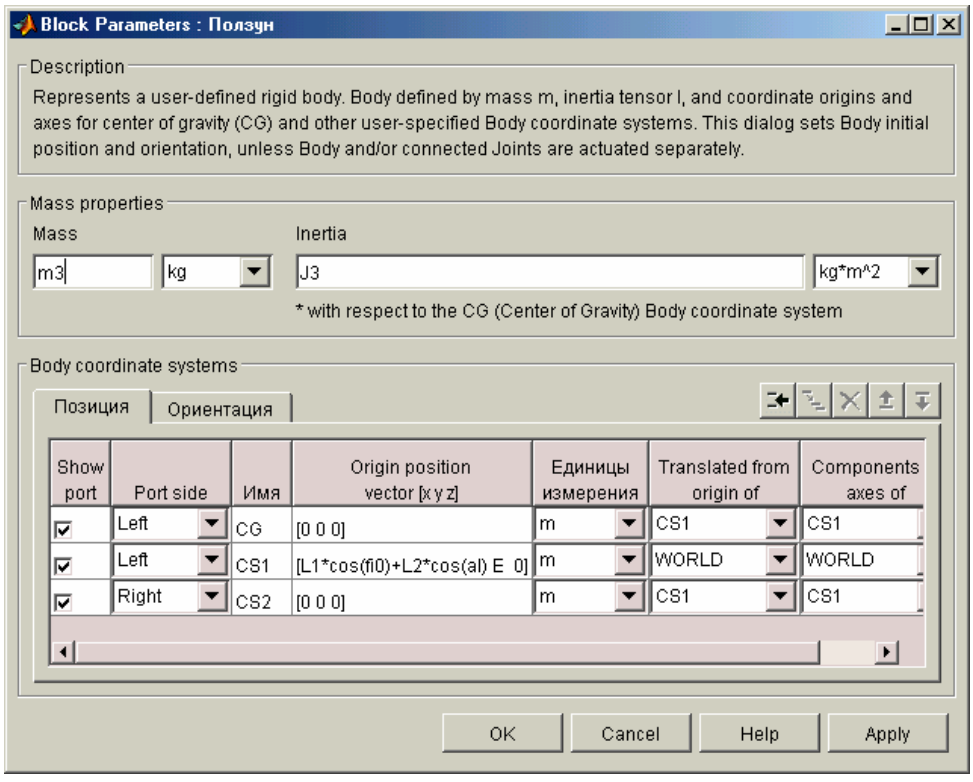


Рис. 11.48. Окно настраивания блока Ползун

Программа **SimMech_KSM1_upr** управления работой этой модели приведена ниже.

% SimMech_KSM1_upr

```
% Лазарев Ю. Ф. 6-05-2004

clear all,          clc
% Задание параметров ведущего звена
m1=1;   J1=[1 0 0;0 1 0; 0 0 1];    L1=0.5;    fi0=0; om=5;
% Задание параметров ведомого звена
m2=1;   J2=[1 0 0;0 1 0; 0 0 1];    L2=1;
% Задание параметров ползуна
m3=1;   J3=[1 0 0;0 1 0; 0 0 1];  E=0.1;
% Предварительные расчеты
al=asin((L1*sin(fi0)-E)/L2);
% Моделирование
sim('SimMech_KSM1')
% Получение данных
t=tout;
% 1. С приводного шарнира
Ug1=unwrap(yout(:,1)*pi/180)*180/pi; UgSk1=yout(:,2);  UgUsk1=yout(:,3);
Mc1=yout(:,4);
% 2. С ведомого шарнира
Ug2=unwrap(yout(:,5)*pi/180)*180/pi; UgSk2=yout(:,6);  UgUsk2=yout(:,7);
Mc2=yout(:,8);
% 3. С шарнира ползуна
Ug3=yout(:,9); UgSk3=yout(:,10);  UgUsk3=yout(:,11);
Mc3=yout(:,12);
% 4. С направляющей ползуна
Xp=yout(:,13); Skp=yout(:,14); Uskp=yout(:,15);
% 5. С центра масс ведущего звена
Xz1=yout(:,16); Yz1=yout(:,17);   SkXz1=yout(:,19);    SkYz1=yout(:,20);
UgSkZ1=yout(:,24);  UgUskZ1=yout(:,27);
% 6. С центра масс ведомого звена
Xz2=yout(:,28); Yz2=yout(:,29);  SkXz2=yout(:,31);    SkYz2=yout(:,32);
UgSkZ2=yout(:,36);  UgUskZ2=yout(:,39);
% 7. С центра масс ползуна
Xz3=yout(:,40); Yz3=yout(:,41);  SkXz3=yout(:,43);    SkYz3=yout(:,44);
UgSkZ3=yout(:,48);  UgUskZ3=yout(:,51);
% Вывод графиков
subplot(2,3,1)
plot(t,Ug1,'o',t,Ug1+Ug2,'.',t,Ug3), grid, title('Углы поворотов шарниров')
xlabel('Время (с)'). ylabel('Градусы')
h=text(0.0,1200,'Кривошипно-шатунный механизм','FontSize',14);
legend('Шарнир1','Шарнир1-2','Шарнир3',0)
subplot(2,3,4)
plot(t,Mc1,'o'), grid, title('Момент сил на ведущем шарнире')
xlabel('Время (с)'), ylabel('Ньютон*м')
subplot(2,3,2)
plot(t,UgSk1,'o',t,UgSk2,'.',t,UgSk3), grid
title('Угловые скорости поворотов шарниров')
xlabel('Время (с)'), ylabel('Градусы в секунду')
legend('Шарнир1','Шарнир2','Шарнир3',0)
subplot(2,3,3)
plot(t,UgSkZ1,'o',t,UgSkZ2,'.',t,UgSkZ3), grid
title('Угловые ускорения поворотов шарниров')
xlabel('Время (с)'), ylabel('Градусы в секунду^2')
legend('Шарнир1','Шарнир2','Шарнир3',0)
subplot(2,3,5)
plot(t,Xp,'r'), grid, title('Отн. перемещение призматич. сочленения')
xlabel('Время (с)'), ylabel('Метры')
subplot(2,3,6)
plot(t,Skp,'.',t,Uskp), grid, title('Скорость и ускорение ползуна')
xlabel('Время (с)'), ylabel('М/с и м/с^2'), legend('скорость','Ускорение',0)

figure
subplot(2,3,1)
plot(t,Xz1,'o',t,Xz2,'.',t,Xz3), grid, title('Перемещения X ц.м. звеньев')
xlabel('Время (с)'), ylabel('Метры'), legend('ведущее','ведомое','ползун',1)
subplot(2,3,3)
plot(t,UgSkZ1,'o',t,UgSkZ2,'.',t,UgSkZ3), grid,
title('Угловая скорость ведущего звена')
xlabel('Время (с)'), ylabel('Градусы в сек.')
subplot(2,3,4)
```

```
plot(t,Yz1,'o',t,Yz2,'.',t,Yz3), grid, title('Перемещения Y ц.м. звеньев')
xlabel('Время (c)'), ylabel('Метры')
subplot(2,3,2)
plot(t,SkXz1,'o',t,SkXz2,'.',t,SkXz3), grid, title('Скорость Vx звеньев')
xlabel('Время (c)'), ylabel('Градусы в сек.')
subplot(2,3,5)
plot(t,SkYz1,'o',t,SkYz2,'.',t,SkYz3), grid, title('Скорость Vy звеньев')
xlabel('Время (c)'), ylabel('Градусы в сек.')
subplot(2,3,6)
plot(t,UgUskZ1,'o',t,UgUskZ2,'.',t,UgUskZ3), grid
title('Угловые ускорения звеньев'), xlabel('Время (c)'), ylabel('Градусы в сек.')
```

Результаты моделирования по этой программе представлены на рис. 49 и 50.

Как видим, средства библиотеки **SimMechanics** позволяют получить достаточно обширную информацию о механическом движении тел, как относительном, так и абсолютном, включая реакции в связях между телами. Последнее обстоятельство особенно важно при проектировании механизмов и машин.

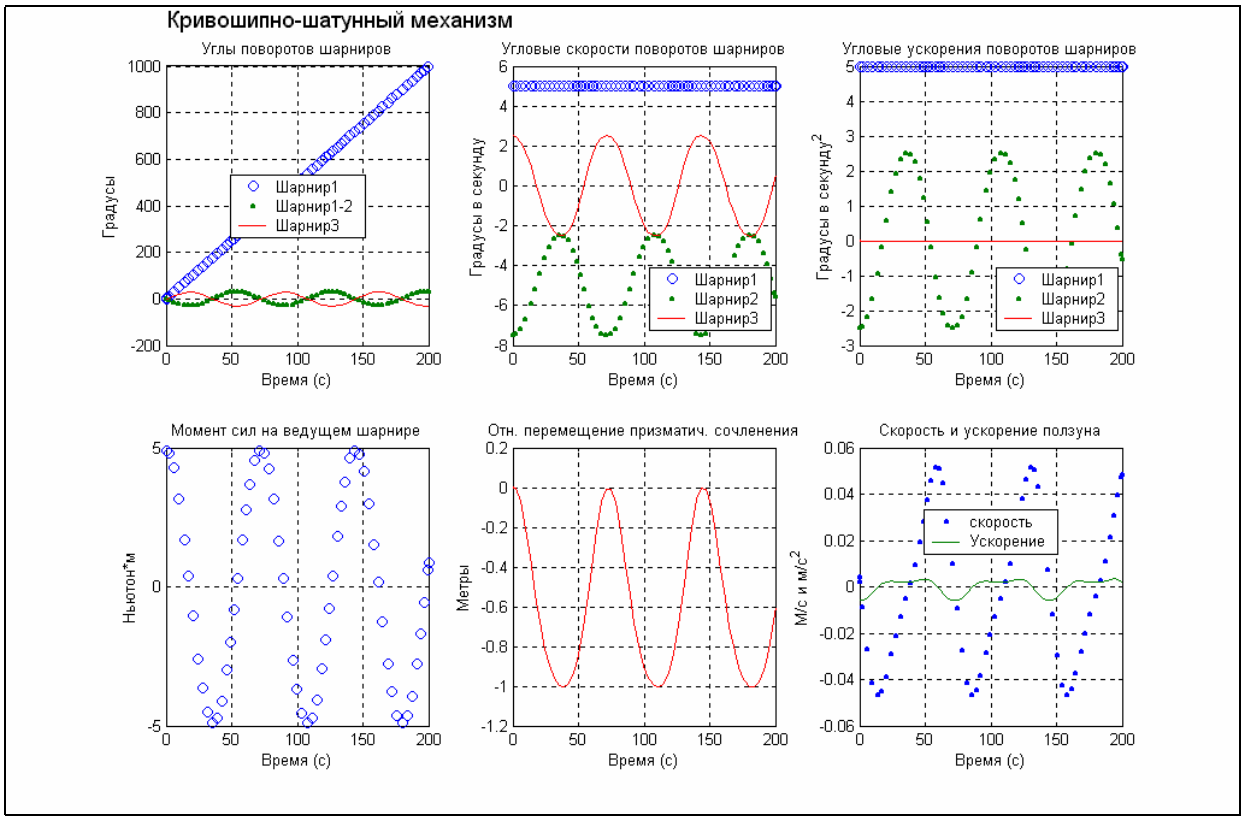


Рис. 11.49. Результаты моделирования по модели **SimMech_KSM1**

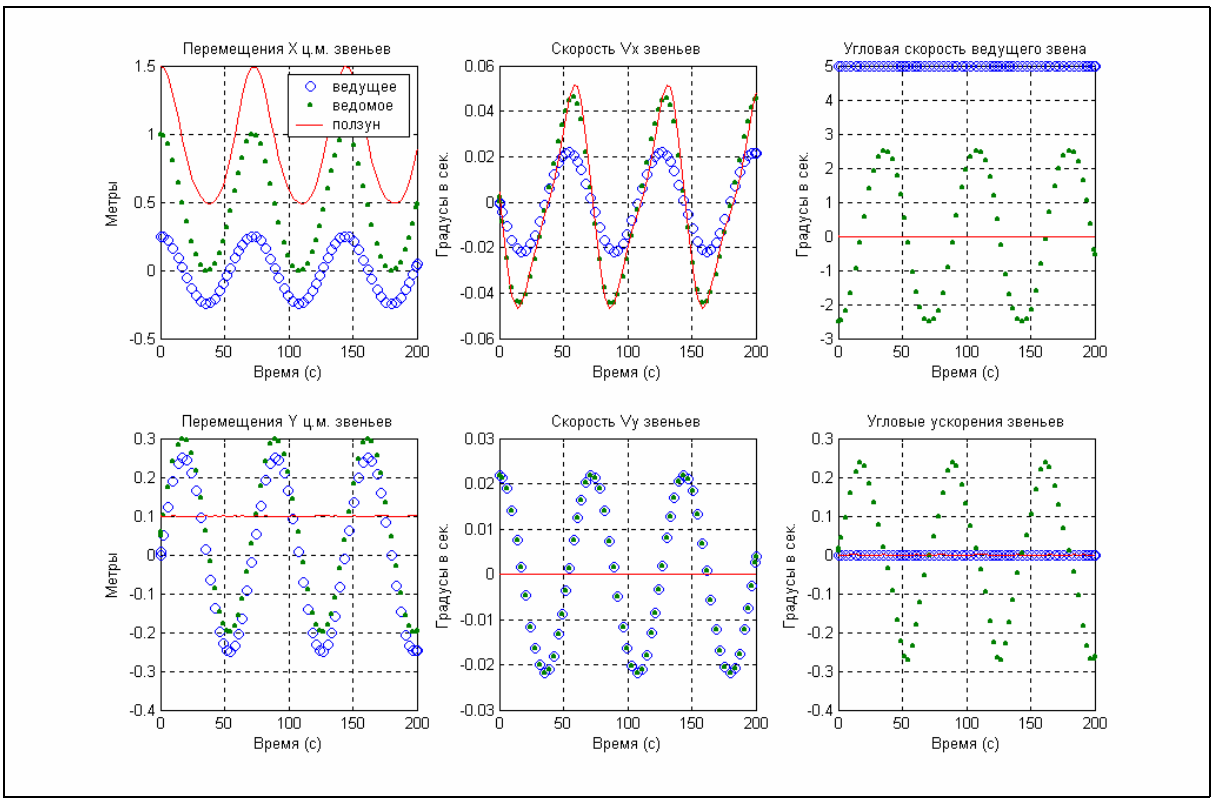


Рис. 11.50. Результаты моделирования кривошипно-шатунного механизма

11.4. Модель движения маятника

Рассмотрим модель движения маятника на поступательно вибрирующем основании, поведение которого было исследовано ранее другими средствами.

Для начала отметим, что по умолчанию модели библиотеки **SimMechanics** автоматически учитывают действие силы тяжести на все звенья механизма. При этом предполагается, что ось Y земной (инерциальной) системы координат направлена вдоль вертикали вверх, а, следовательно, сила тяжести, приложенная в центре масс, направлена по ней в противоположном направлении (вниз). Оси Z и X поэтому лежат в плоскости горизонта.

В дальнейшем будем предполагать, что ось вращения маятника направлена вдоль оси Z, а колебания маятника происходят в плоскости YX.

Для имитации поступательного движения основания в этой плоскости свяжем (рис. 51) тело маятника (блок Тело) с инерциальной системой отсчета (блок Основание) через сочленение Planar, которое представляет собой цепь из трех последовательно соединенных примитивов – P1, P2 и R3.

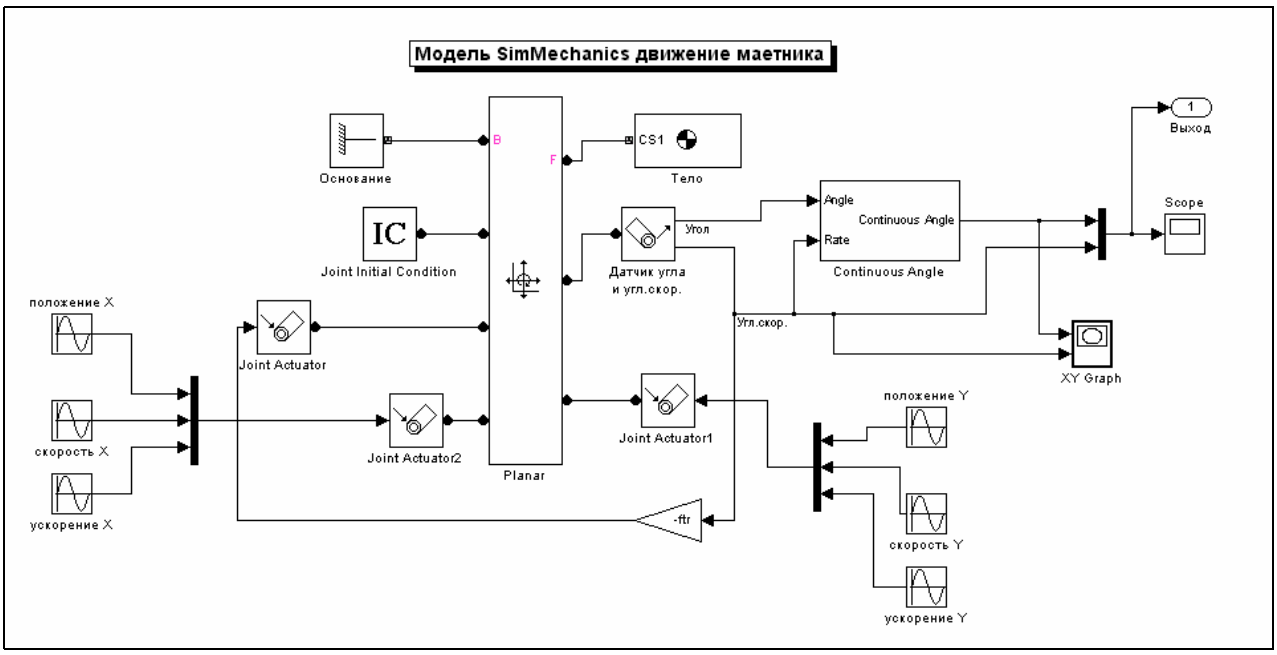


Рис. 11.51. Блок-схема модели **SimMech_FMp** маятника

Первые два (типа Prismatic) реализуют поступательное движение основания соответственно вдоль осей X и Y (см рис. 52). Третий примитив (типа Revolute) реализует вращательную степень свободы маятника вокруг оси Z.

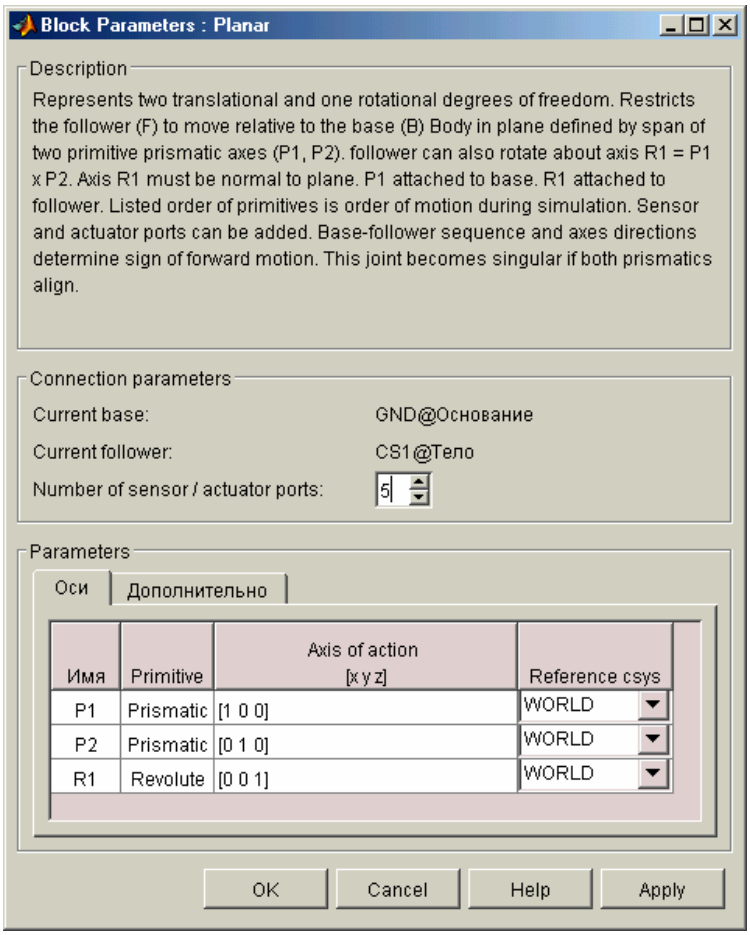


Рис. 11.52. Настройки блока Planar

Для задания параметров использованы следующие обозначения.

`M`	Масса маятника
`J`	Матрица моментов инерции маятника относительно его центра масс
`l`	Смещение центра масс маятника относительно оси его вращения
`ftr`	Коэффициент вязкого трения в оси вращения маятника
`fi0`	Начальный угол отклонения маятника от вертикали
`fit0`	Начальная угловая скорость маятника
`nxm, nym`	Амплитуды виброперегрузок основания соответственно в горизонтальном и вертикальном направлениях
`om`	Частота колебаний основания
`epx, epy`	Начальные фазы колебаний основания
`g`	Ускорение силы тяжести

Ввод начальных условий движения маятника осуществляется блоком Joint Initial Condition, настройки которого приведены на рис. 53.

Окно настраивания блока Тело показаны на рис. 54.

Возбуждение колебаний основания осуществляется двумя блоками Joint Actuator2 (по оси X) и Joint Actuator1 (по оси Y). На рис. 55 отображены настройки одного из них.

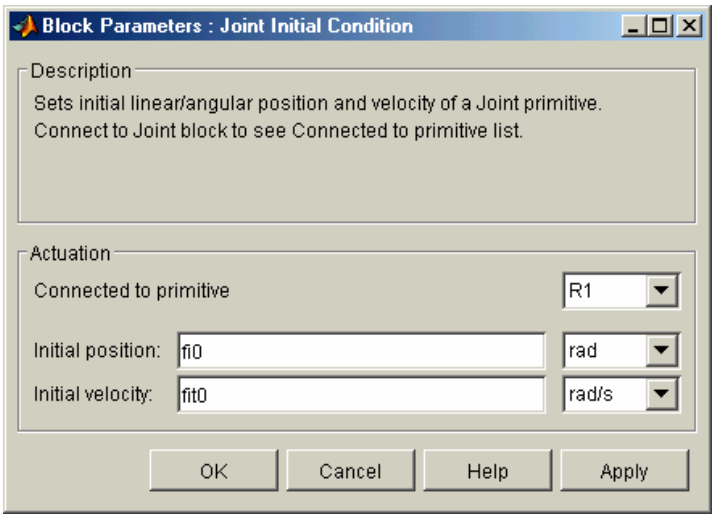


Рис. 11.53. Настройки блока Joint Initial Condition

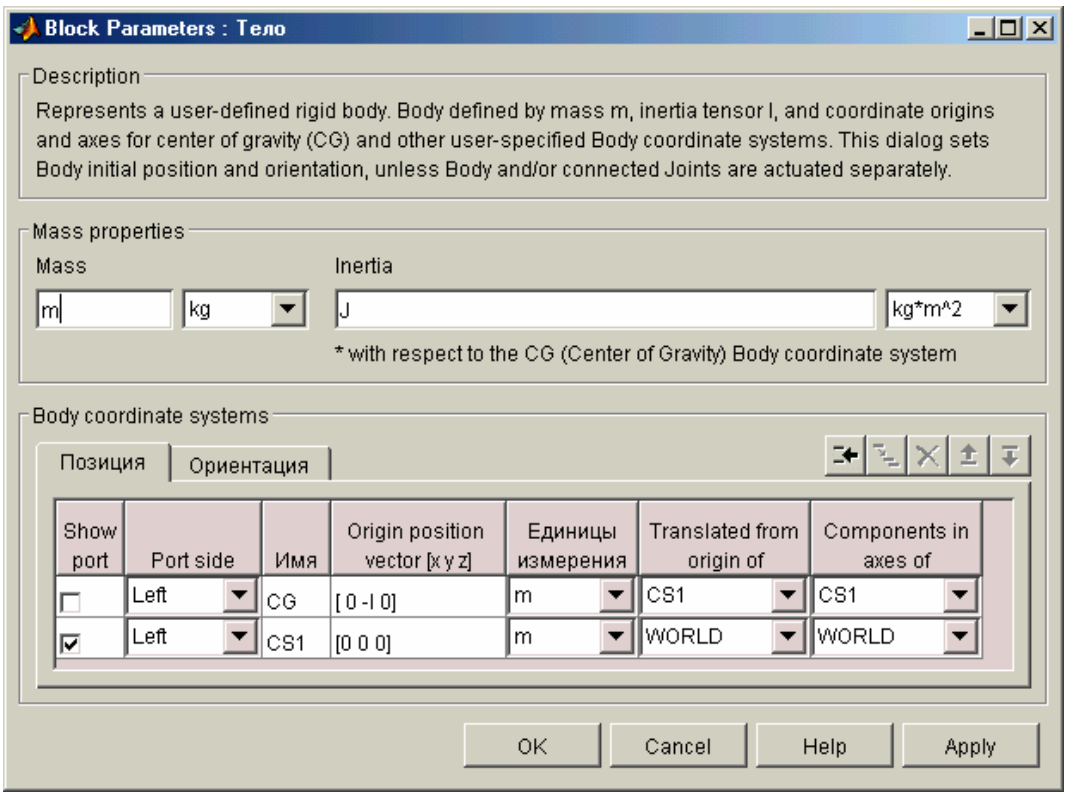


Рис. 11.54. Настройки блока Тело

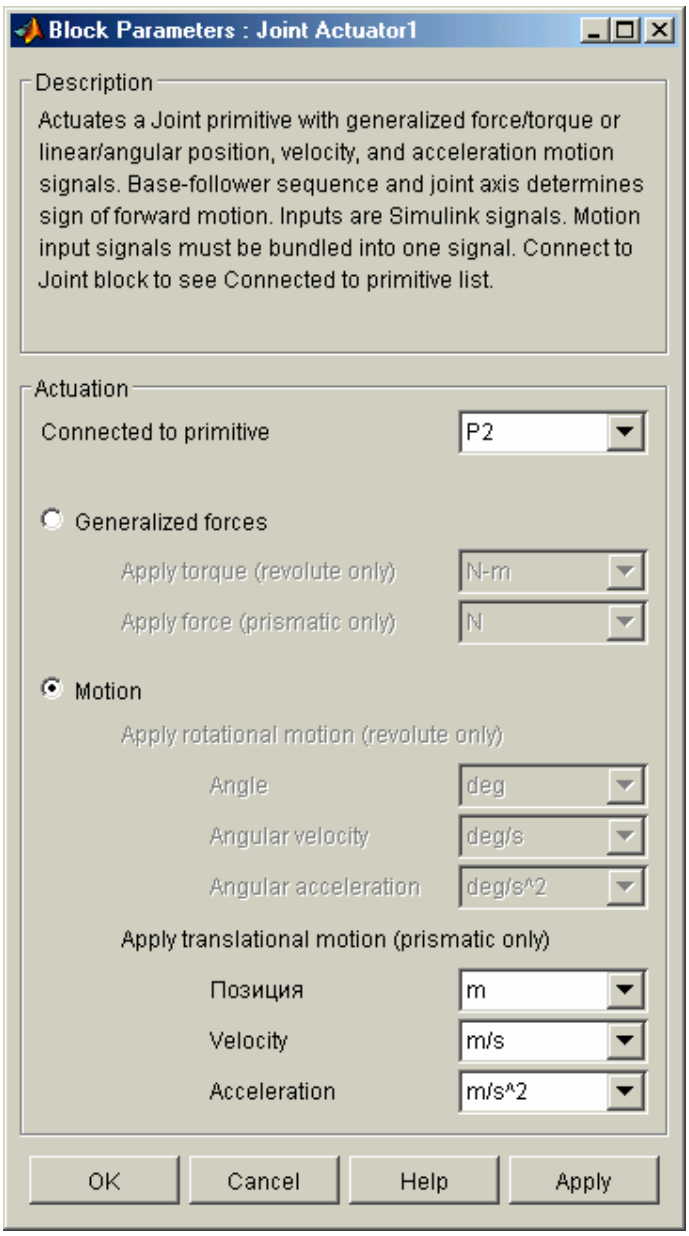


Рис. 11.55. Настройки блока Joint Actuator1

Измерение угла отклонения маятника от вертикали и его угловой скорости осуществляется с помощью блока «Датчик угла и угл. скор.» типа Joint Sensor, настройки которого показаны на рис. 56.

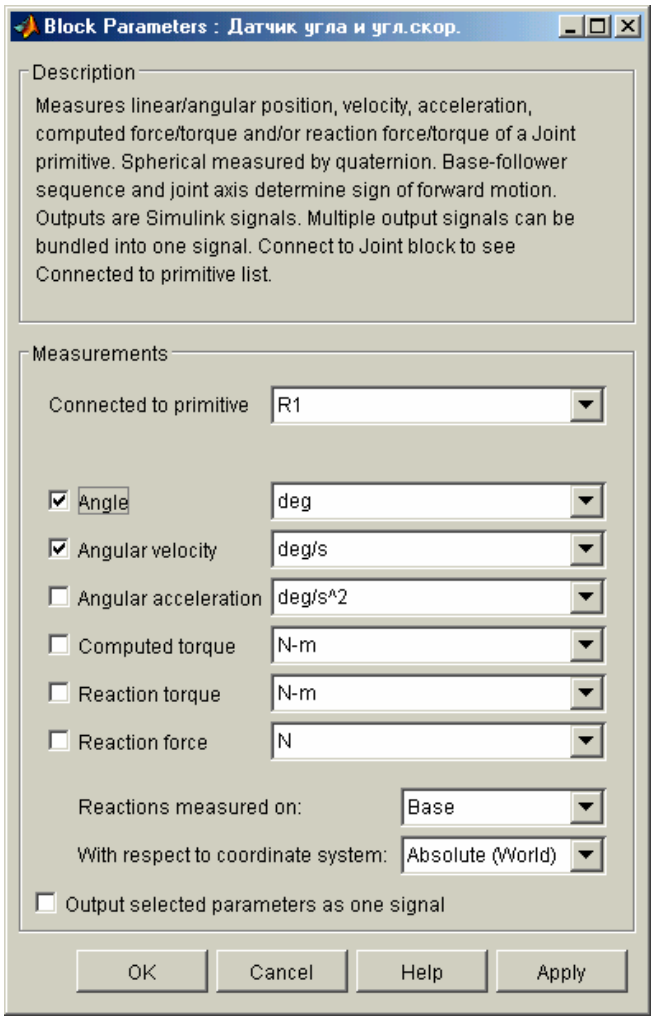


Рис. 11.56. Настройки блока «Датчик угла и угл. скор.»

Для получения непрерывного и большего по модулю 180 ^{o} угла поворота маятника использован блок Continuous Angle.

Сигнал угловой скорости, полученный на выходе блока «Датчик угла и угл. скор.» используется для формирования момента сил вязкого трения на оси вращения маятника при помощи блока Joint Actuator, настройки которого приведены на рис. 57.

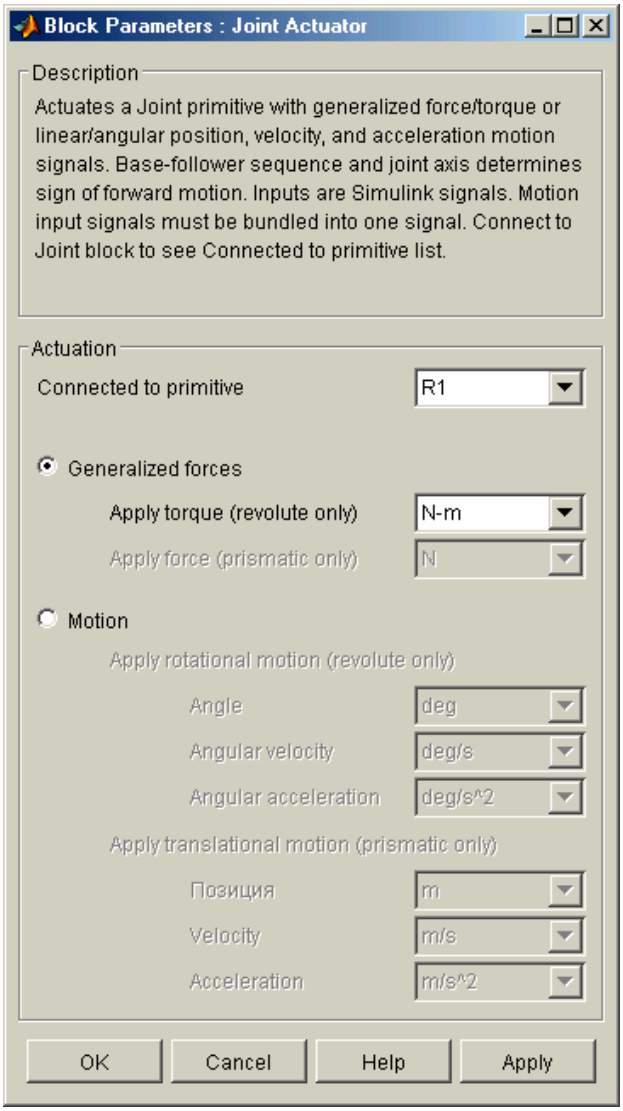


Рис. 11.57. Настройки блока Joint Actuator

Управление вводом данных, запуском модели и выведением результатов можно осуществить с помощью управляющей программы **SimMech_FM_upr**, приводимой ниже.

```
% SimMech_FM_upr
% Управляющая программа для S-модели SimMech_FMp

% Лазарев Ю. Ф., 11-05-2004

clear all, clc
% Ввод параметров маятника
m=1;   Jx=0.01;   Jy=0.01;   Jz=0.02;   l=0.5;
ftr= 0.0%05;
J=[Jx 0 0;0 Jy 0;0 0 Jz];
% Параметры внешних воздействий
nxm=0;  om=10;   epx=0;
nym=0;  om=21.5;   epy=0;   g=9.81;
% Ввод начальных условий
fi0=179.9*pi/180; fit0=0;
% Моделирование на S-модели
sim('SimMech_FMp')
% Получение результатов
t=tout; fi=yout(:,1);   fit=yout(:,2);
% Вывод графиков
subplot(2,2,1)
plot(fi*pi/180,fit*pi/180), grid
```

```
xlabel('Угол (радиан)'), ylabel('Угл. скорость (рад/с)')
set(gca,'FontSize',12), title('Фазовый портрет')
subplot(2,2,[3 4])
plot(t,fi), grid
xlabel('Время (секунды)'), ylabel('Угол (градусы)')
set(gca,'FontSize',12), title('Отклонение от вертикали')
subplot(2,2,2)
axis('off');
h=text(0.1,1.1,'Маятник (модель SimMechanics)','FontSize',14);
h=text(-0.2,1.0,['Масса (кг)   m=  ',num2str(m)],'FontSize',12);
h=text(0.5,1.0,['Смещение ц.м (м)   L=  ',num2str(l)],'FontSize',12);
h=text(-0.2,0.9,'Матрица моментов инерции относительно центра масс (кг м^2)','FontSize',12);
h=text(0.1,0.8,'| ','FontSize',12);
h=text(0.2,0.8,num2str(J(1,1)),'FontSize',12);
h=text(0.4,0.8,num2str(J(1,2)),'FontSize',12);
h=text(0.6,0.8,num2str(J(1,3)),'FontSize',12);
h=text(0.8,0.8,'| ','FontSize',12);
h=text(-0.1,0.7,'J = ','FontSize',12);
h=text(0.1,0.7,'| ','FontSize',12);
h=text(0.2,0.7,num2str(J(2,1)),'FontSize',12);
h=text(0.4,0.7,num2str(J(2,2)),'FontSize',12);
h=text(0.6,0.7,num2str(J(2,3)),'FontSize',12);
h=text(0.8,0.7,'| ','FontSize',12);
h=text(0.1,0.6,'| ','FontSize',12);
h=text(0.2,0.6,num2str(J(3,1)),'FontSize',12);
h=text(0.4,0.6,num2str(J(3,2)),'FontSize',12);
h=text(0.6,0.6,num2str(J(3,3)),'FontSize',12);
h=text(0.8,0.6,'| ','FontSize',12);
h=text(-0.2,0.5,'Начальный угол (градусы)','FontSize',12);
h=text(0.7,0.5,['\phi0 = ',num2str(fi0*180/pi)],'FontSize',12);
h=text(-0.2,0.4,'Начальная угловая скорость (рад/с)','FontSize',12);
h=text(0.7,0.4,['\phi''0 = ',num2str(fit0)],'FontSize',12);
h=text(-0.2,0.3,'К-нт вязкого трения (Н м с)','FontSize',12);
h=text(0.7,0.3,['ftr = ',num2str(ftr*180/pi)],'FontSize',12);
h=text(-0.2,0.2,'Движение основания:','FontSize',12);
h=text(0.5,0.2,['om = ',num2str(om)],'FontSize',12);
h1=text(-0.2,0.1,'   вдоль горизонтали','FontSize',12);
h=text(0.5,0.1,['nxm = ',num2str(nxm)],'FontSize',12);
h=text(0.8,0.1,['epx = ',num2str(epx)],'FontSize',12);
h2=text(-0.2,0.0,'   вдоль вертикали','FontSize',12);
h=text(0.5,0.0,['nym = ',num2str(nym)],'FontSize',12);
h=text(0.8,0.0,['epy = ',num2str(epy)],'FontSize',12);
h=text(-0.1,-0.05,'-----------------------------------------------------------------------------------------------');
h=text(-0.1,-0.1,'Программа SimMech-FM-upr       Лазарев Ю. Ф.     26-04-2004');
h=text(-0.1,-0.15,'-----------------------------------------------------------------------------------------------');
```

Далее приведены два примера работы программы и модели. На рис. 58 изображены результаты работы модели для свободных (без трения и колебаний основания) колебаний маятника с амплитудой, близкой к 180 ^{o}.

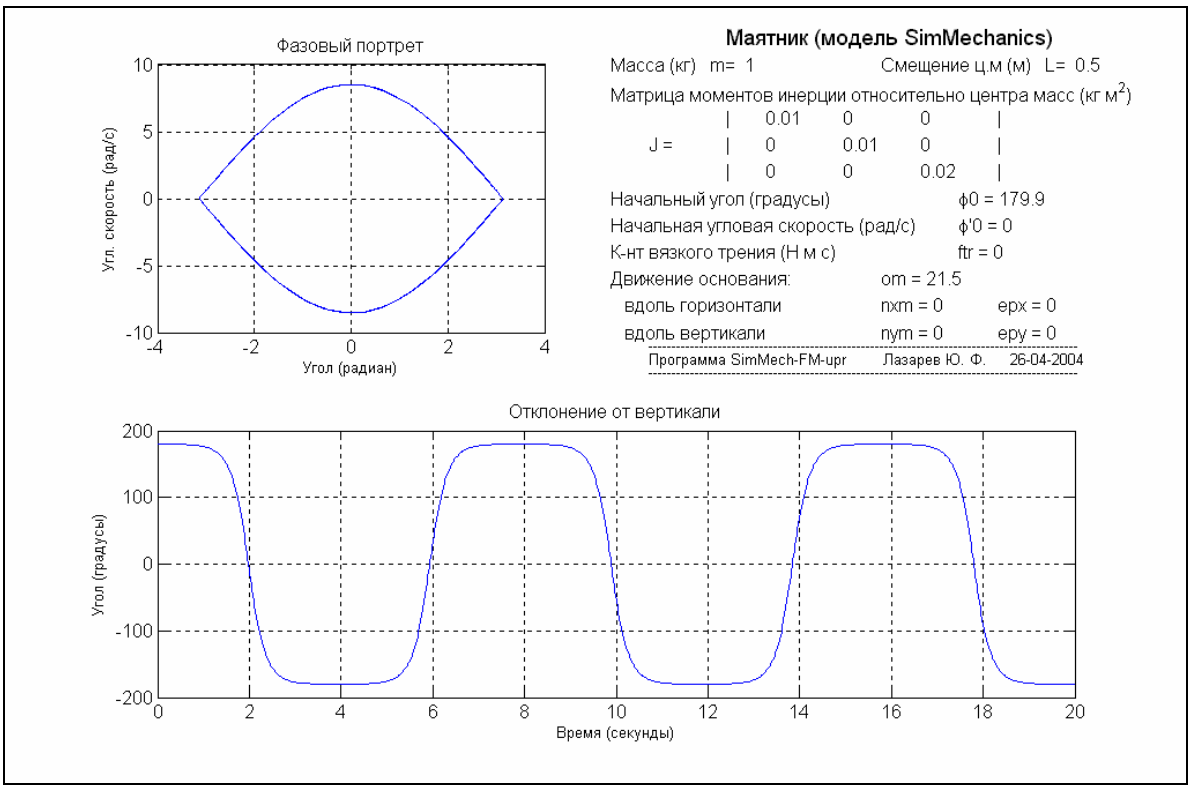


Рис. 11.58. Свободные колебания маятника с большой амплитудой

Рис. 59 представляет результаты моделирования маятника при интенсивной вертикальной вибрации основания и иллюстрирует устойчивость при этих условиях верхнего положения равновесия маятника.

Наконец, на рис. 60 проиллюстрирован выпрямительный эффект маятника при интенсивной горизонтальной вибрации основания.

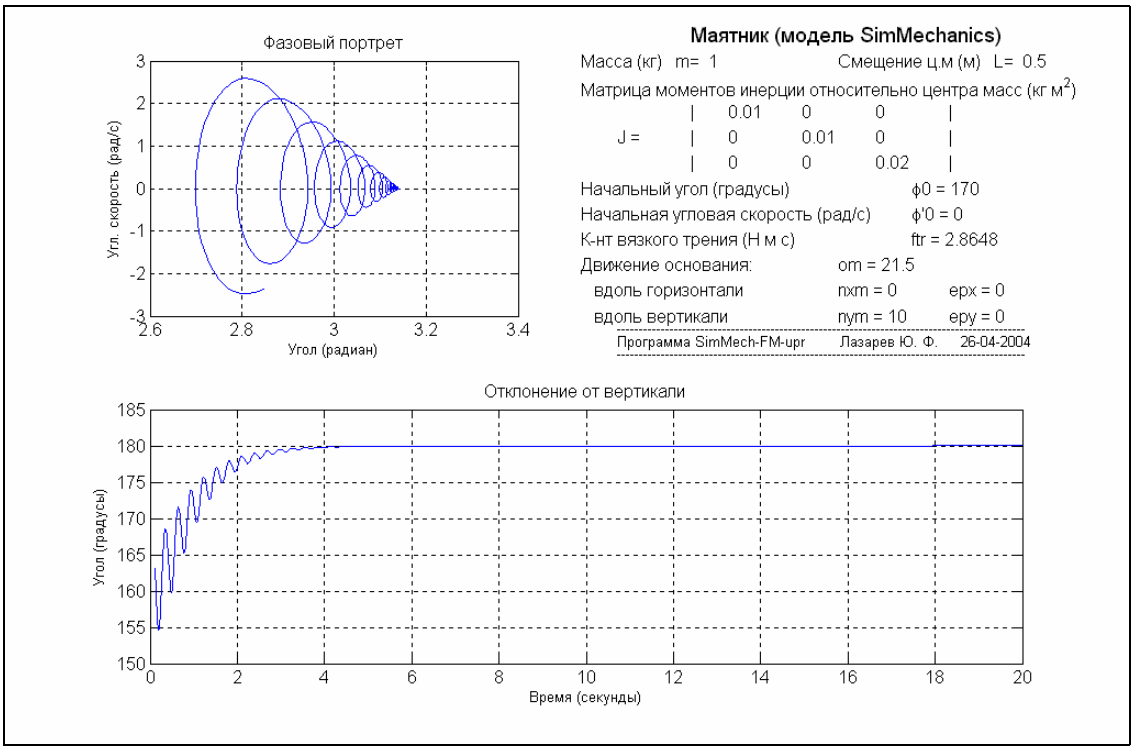


Рис. 11.59. Устойчивость верхнего положения равновесия маятника

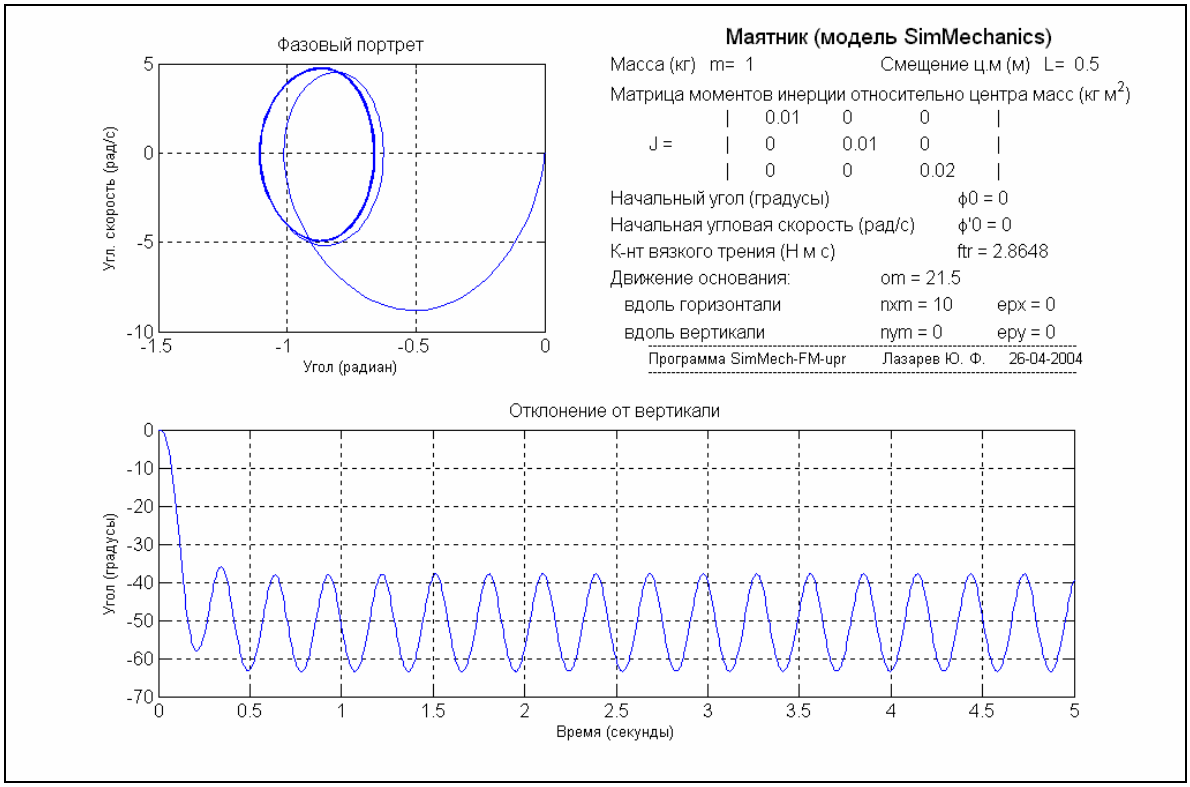


Рис. 11.60. Отклонение положения равновесия маятника от вертикали при горизонтальной вибрации основания

11.5. Вопросы для самопроверки

1. Для чего предназначена библиотека **SimMechanics**?

2. Каковы основные принципы формирования блок-схемы, создаваемой на основе библиотеки **SimMechanics**? Отличны ли они от принципов создания S-моделей с помощью библиотеки Simulink?

3. Из каких основных разделов состоит библиотека **SimMechanics**?

4. В чем состоят основные особенности блоков библиотеки **SimMechanics** по сравнению с обычными S-блоками? Каковы преимущества и недостатки такого построения блоков?

5. Какие блоки библиотеки **SimMechanics** осуществляют связь с блоками библиотеки Simulink и с системой MATLAB?

6. При помощи каких блоков библиотеки **SimMechanics** осуществляется имитация источников механического движения? имитация обеспечения степеней свободы? имитация наложения связей?

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. Там же можно ознакомиться с содержанием некоторых пакетов прикладных программ (ППП), которые поставляются с той или иной версией MatLAB.

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. Ануфриев И. Е. Самоучитель MatLab 5.3/6.x. - СПб.: БХВ-Петербург, 2002. - 736 с.
2. Антонью А. Цифровые фильтры: анализ и проектирование. - М.: Радио и связь, 1983. - 320 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. Визуальное моделирование в среде MATLAB: Учебный курс. - СПб.: "ПИТЕР", 2000. - 430 с.
5. Ви В., Уэйс Х., Эрепостатис Э. Управление поворотами космического аппарата вокруг собственной оси с обратной связью по компонентам кватерниона/ Аэрокосмическая техника, № 3, март, 1990.
6. Дьяконов В. Simulink 4. Специальный справочник. - СПб: "Питер", 2002. – 518 с.
7. Дьяконов В. П., Абраменкова И. В. MatLAB 5.0/5.3. Система символьной математики. - М.: "Нолидж", 1999. – 634 с.
8. Дьяконов В. П., Абраменкова И. В. MatLAB. Обработка сигналов и изображений. Специальный справочник. - СПб.: "Питер", 2002. – 608 с.
9. Дьяконов В., Круглов В. Математические пакеты расширения MatLAB. Специальный справочник. - СПб.: "Питер", 2001. – 475 с.
10. Дьяконов В., Круглов В. MatLAB. Анализ, идентификация и моделирование систем. Специальный справочник. - СПб.: "Питер", 2002. – 444 с.
11. Краснопрошина А. А., Репникова Н. Б., Ильченко А. А. Современный анализ систем управления с применением MATLAB, Simulink, Control System: Учебн. пособие. - К.: "Корнійчук", 1999. – 144с.
12. Лазарев Ю. Ф. Початки програмування в среде MatLAB: Навч. посібник. - К.: "Корнійчук", 1999. - 160с.
13. Лазарев Ю. MatLAB 5.x. – К.: "Ирина" (BHV), 2000. – 384 с.
14. Лазарєв Ю.Ф. Рух несиметричного гіроскопа/ Наукові вісті НТУУ "КПІ", №4(30), 2004, с. 114 -121
15. Мартынов Н. Н. Введение в MATLAB 6. – М.: "Кудиц-образ", 2002. – 348 с.
16. Медведев В. С., Потемкин В. Г. Control System Toolbox. MatLAB 5 для студентов. - М.: "ДИАЛОГ-МИФИ", 1999. – 287 с.
17. Потемкин В. Г. Система MatLAB: Справочное пособие. - М.: "ДИАЛОГ-МИФИ", 1997. - 350 с.
18. Потемкин В. Г. Система инженерных и научных расчетов MATLAB 5.x. Т.1,2. - М.: "ДИАЛОГ-МИФИ", 1999. - 350 с.
19. Потемкин В.Г. MatLAB 5 для студентов: Справ. пособие. - М.: "ДИАЛОГ-МИФИ", 1998. - 314 с.
20. Потемкин В. Г., Рудаков П. И. MatLAB 5 для студентов. - 2-е изд. - М.: "ДИАЛОГ-МИФИ", 1999. - 448 с.
21. Рудаков П. И., Сафонов В. И. Обработка сигналов и изображений. Matlab 5x. - М.: "ДИАЛОГ-МИФИ", 2000. - 414 с.