Ю. Б. Колесов, Ю. Б. Сениченков

МОДЕЛИРОВАНИЕ СИСТЕМ

ПРАКТИКУМ ПО КОМПЬЮТЕРНОМУ МОДЕЛИРОВАНИЮ

- *Технология разработки моделей сложных динамических систем в среде визуального моделирования MvStudium*
- *Типовые вычислительные эксперименты*
- *Учебные задания по компьютерному моделированию*
- *Численная библиотека среды*
- *Компакт-диск с моделями и демонстрационной версией среды*

+CD

УЧЕБНОЕ ПОСОБИЕ

bhv®

Ю. Б. Колесов
Ю. Б. Сениченков

МОДЕЛИРОВАНИЕ СИСТЕМ

ПРАКТИКУМ ПО КОМПЬЮТЕРНОМУ МОДЕЛИРОВАНИЮ

Рекомендовано Учебно-методическим объединением по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 220100 – «Системный анализ и управление»

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.06
ББК 32.973.26
К60

Колесов, Ю. Б.

К60 Моделирование систем. Практикум по компьютерному моделированию / Ю. Б. Колесов, Ю. Б. Сениченков. — СПб.: БХВ-Петербург, 2007. — 352 с.: ил. + CD-ROM

ISBN 5-94157-580-7

Учебное пособие содержит: описание технологии разработки моделей сложных динамических систем и типовых вычислительных экспериментов, набор учебных заданий и описание численной библиотеки MvStudium. На прилагаемом к книге компакт-диске помещены все рассматриваемые в этом томе модели и демонстрационная версия среды визуального моделирования MvStudium. Книга является последней из трех, объединенных общим названием МОДЕЛИРОВАНИЕ СИСТЕМ.

Для студентов вычислительных специальностей технических вузов

УДК 681.3.06
ББК 32.973.26

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Рецензенты:
Евгенев Г. Б., д.т.н., профессор кафедры «Компьютерные системы автоматизации производств» МГТУ им. Баумана;
Ивановский Р. И., д.т.н., профессор кафедры «Распределенные вычисления и компьютерные сети» СПбГПУ.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.11.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 28,38.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-580-7

© Колесов Ю. Б., Сениченков Ю. Б., 2007
© Оформление, издательство "БХВ-Петербург", 2007

Оглавление

Предисловие **1**

Глава 1. Типовые модели: разработка и отладка **3**

Создание и отладка модели непрерывной системы 3
Создание и отладка модели гибридной системы 31
Создание и отладка структурной модели 55

Глава 2. Типовые вычислительные эксперименты **93**

Реализация определенного сценария поведения 93
Сравнение вариантов 96
Параллельная композиция вариантов 97
Последовательная композиция вариантов 99
Получение параметрической зависимости 100
Параметрическая оптимизация 105
Сравнение результатов моделирования с внешними данными 109
Проведение статистических испытаний 115

Глава 3. Учебные задания **121**

Динамические системы 121
Задание 1 121
Задание 2 123
Задание 3 124
Задание 4 126
Задание 5 132
Гибридные автоматы 134
Задание 6 134
Задание 7 136

Задание 8 138
Задание 9 139
Компонентные модели 141
Задание 10 141
Модели общего вида 144
Задание 11 144
Задание 12 146

Глава 4. Численная библиотека MvStudium 165

Матричные вычисления 166
Использование MvStudium как матричного калькулятора 166
Задача 1 172
Задача 2 184
Задача 3 195
Решение нелинейных уравнений 201
Почему так трудно решать нелинейные уравнения? 201
Задача 4 214
Задача 5 220
Задача 6 220
Задача 7 223
Задача 8 224
Решение обыкновенных дифференциальных уравнений 229
Простейшие методы и их свойства 232
Жесткие задачи 243
Колебательные задачи 259
Странные аттракторы. Отображение Пуанкаре 275
Уравнения, не приведенные к форме Коши 284
Решение алгебро-дифференциальных уравнений 289

Приложение 1. Тестовые матрицы с известной обратной 299

Приложение 2. Библиотека пакета MvStudium 309

Организация библиотеки 310
Структура решателей 312

Автоматические решатели 312
Решатели для задач конкретного типа 312
Решатели для отладки 313
Системы нелинейных алгебраических уравнений 314
Программы для решения нелинейных уравнений 315
Программа-автомат 318
Дифференциальные уравнения 320
Программы для решения нежестких уравнений 322
Программы для решения жестких уравнений 324
Программа-автомат 326
Отладочные программы 328
Алгебро-дифференциальные уравнения 329
Явный способ 330
Неявный способ 331
Программа-автомат для решения алгебро-дифференциальных уравнений 333

Приложение 3. Описание компакт-диска 335

Литература 337

Дополнительная литература по главам 337
К главе 1 337
К главам 3 и 4 338

Предисловие

Вы держите в руках третью книгу по моделированию систем — "Практикум по компьютерному моделированию".

Она является завершающей в серии книг "Моделирование систем. Динамические и гибридные системы" [5], "Моделирование систем. Объектно-ориентированный подход" [6] и демонстрирует, как использовать объектно-ориентированный подход при моделировании сложных динамических систем, а также может служить руководством по моделированию сложных динамических систем в графической среде MvStudium.

Глава 1 посвящена вопросам технологии создания и отладки непрерывных, гибридных и многокомпонентных моделей систем. В ней читатель найдет последовательно усложняющиеся модели движения снаряда, полета неуправляемой и управляемой ракеты. Рассматриваемые модели, конечно же, являются упрощенными, однако описанный процесс их разработки отражает существенные особенности, характерные для моделирования технических систем на ранних этапах проектирования.

Глава 2 посвящена систематизации типичных для проектирования сложных динамических систем вычислительных экспериментов. В ней строятся типовые схемы проведения экспериментов и воплощаются в графической среде MvStudium. Сравнивая между собой примеры из книги "Моделирование систем. Объектно-ориентированный подход" [6], и примеры, приведенные в этой главе, можно сравнить между собой различные реализации и оценить достоинства объектно-ориентированного подхода.

Глава 3 содержит учебные задания, которые можно использовать при изучении курсов "Основы моделирования", "Компьютерные вычисления", "Моделирование механических систем". Задачи классифицированы с точки зрения используемых пользовательских интерфейсов пакета MvStudium.

Глава 4 посвящена использованию MvStudium как специализированной численной библиотеки. Встроенная библиотека для решения алгебро-дифференциальных уравнений и достаточно мощный алгоритмический язык позволяют организовать двухэтапное обучение многим разделам современного численного анализа. Изобразительные возможности среды позволяют формулировать задачи на "естественном" математическом языке и сразу же видеть графики численных решений. При необходимости более углубленно-

го изучения численные алгоритмы могут быть реализованы студентом самостоятельно на языке MVL, а затем можно сравнить полученные результаты с результатами работы встроенных алгоритмов.

Приложение 1 — это набор тестовых матриц, обратные для которых можно построить в явном виде. Тестовые матрицы используются в заданиях из *главы 3*.

В приложении 2 приводятся таблицы кодов завершения программных реализаций численных методов визуальной среды моделирования MvStudium и рекомендации пользователям, что следует делать в том или ином случае аварийного завершения.

Приложение 3 — это описание компакт-диска, прилагаемого к книге. Все примеры выполнены в визуальной среде моделирования MvStudium, демонстрационная версия которой находится на компакт-диске. На этом же диске находятся также файлы всех рассматриваемых в книге примеров. Мы надеемся, что эти материалы понадобятся вам при чтении книги, обучении студентов и научной работе. Дополнительную информацию по пакету MvStudium вы можете найти на сайте **www.mvstudium.com**.

Учебное пособие соответствует содержанию федеральной дисциплины ОПД.Ф.10 "Системное моделирование" государственных образовательных стандартов по направлению подготовки бакалавров 220100 "Системный анализ и управление".

Мы признательны:

- заведующему кафедрой "Баллистика и аэродинамика" МГТУ им. Н. Э. Баумана д.т.н., профессору Лысенко Л. Н., прочитавшему рукопись первой главы и сделавшему ряд ценных замечаний;
- профессорам Евгеневу Г. В. и Ивановскому Р. И., рекомендовавшим книгу к изданию;
- коллективу издательства "БХВ-Петербург", всемерно помогавшему нам при подготовке рукописи к изданию.

Эта книга не могла состояться и без поддержки декана факультета технической кибернетики СПбГПУ профессора Черноруцкого И. Г.

И, конечно же, особая благодарность членам наших семей, еще не потерявшим надежды видеть авторов не только за компьютером днем и ночью.

Глава 1

Типовые модели: разработка и отладка

В этой главе на примере нескольких последовательно усложняющихся моделей демонстрируется технология разработки и отладки непрерывных, дискретных, гибридных и других моделей. Все обсуждаемые модели выполнены в визуальной среде объектно-ориентированного моделирования MvStudium. На прилагаемом к данной книге компакт-диске находятся демонстрационная версия этого пакета и соответствующие файлы для всех примеров. В связи с этим материал данной главы содержит некоторый минимальный объем инструкций по работе с пакетом моделирования MvStudium.

Создание и отладка модели непрерывной системы

В качестве примера рассмотрим процесс создания и отладки модели движения снаряда, выстреливаемого с начальной скоростью V_0 под углом θ_0 к горизонту, которая уже рассматривалась в качестве примера в [5, 6]. Здесь мы достаточно подробно изложим технологию создания и отладки набора последовательно усложняющихся моделей. Мы начнем с простейшей модели движения снаряда для схемы плоской аппроксимации поверхности Земли с постоянным по величине и направлению полем тяготения в безвоздушном пространстве. Далее мы будем вносить в эту модель доработки, учитывая различные факторы:

- сопротивление воздуха с постоянной плотностью;
- изменение плотности воздуха с высотой;

- изменение аэродинамических характеристик в зависимости от скорости;
- изменение силы тяжести с высотой;
- кривизна Земли.

Создадим новый проект (пакет с точки зрения UML) с именем **Снаряд**. Для этого в интегрированной среде MvStudium выполним команду главного меню **Проект | Новый** или нажмем кнопку на инструментальной панели. В появившемся диалоге (рис. 1.1) вводим название папки и имя проекта, а также выбираем вид модели — в данном случае **Непрерывный элементарный объект**.

После нажатия кнопки **Создать** будет создана папка ...\Модели\Снаряд и в ней появится файл Снаряд.mvb, содержащий базу данных проекта. Для удобства мы будем каждую модель размещать в отдельной папке, с помощью команды **Проект | Сохранить как** вы можете в дальнейшем сохранять в той же папке различные модификации данного проекта (например, Снаряд_1.mvb и т. д.). В дальнейшем в этой же папке автоматически будут созданы еще два файла — Снаряд.ini и Снаряд_em.ini, в которых сохраняются соответственно установки проекта и установки визуальной выполняемой модели.

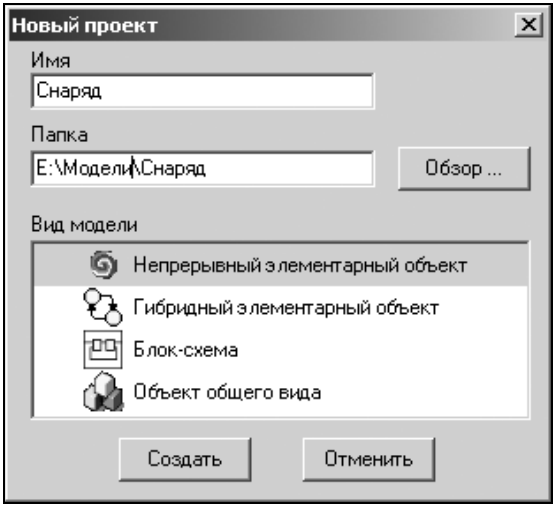


Рис. 1.1. Создание новой модели

На рис. 1.2 показано начальное состояние редактора моделей (главное окно интегрированной среды MvStudium). Для данного простейшего случая он

включает только одно максимизированное окно редактора класса `Model`, который, как показано в томе 2 данной книги, и содержит описание модели для вычислительного эксперимента.

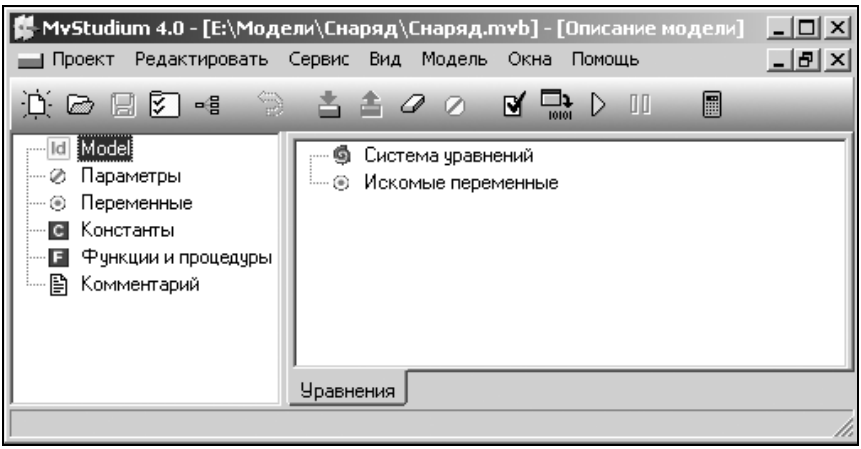


Рис. 1.2. Начальное состояние редактора моделей для модели непрерывной элементарной системы

В общем случае главное окно интегрированной среды включает в себя менеджер проекта в левой части окна и многооконную область редакторов классов (рис. 1.3). Соотношение между ними можно изменить с помощью перемещаемого разделителя (сплиттера) на правой границе менеджера проекта. Менеджер проекта является визуальным представлением элементов проекта (классов, констант, процедур и функций), а также импортируемых пакетов, в виде дерева. В новом проекте всегда присутствует класс `Model` и импортируется пакет стандартных классов `SysyLib`. Вы можете скрыть менеджер проекта с помощью кнопки в его правом верхнем углу или команды меню **Вид | Менеджер проекта**. Вы можете отобразить менеджер проекта с помощью кнопки на инструментальной панели или команды меню **Вид | Менеджер проекта**.

Теперь нужно задать переменные и систему уравнений, определяющую поведение непрерывной модели. Переменные можно определить предварительно, а можно просто использовать их в системе уравнений, и редактор уравнений, встретив неопределенную переменную, будет автоматически воспринимать ее как переменную типа `double` с нулевым начальным значением. Выберем второй путь.

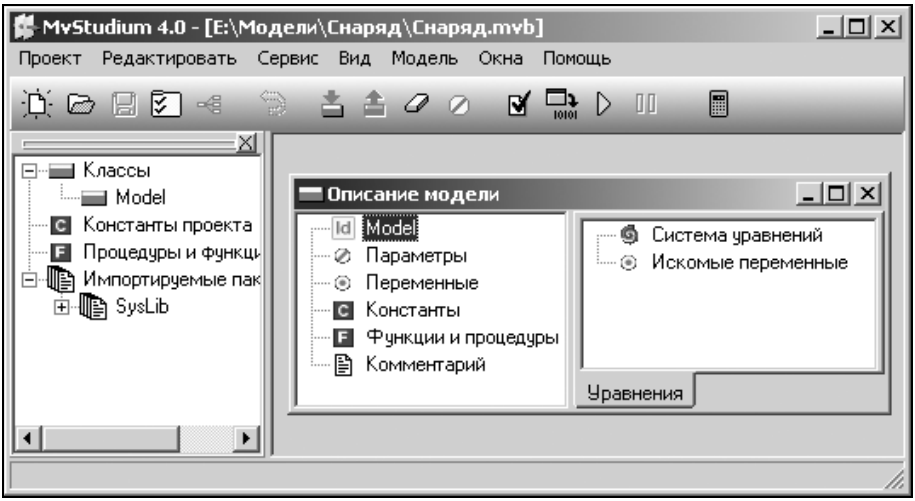


Рис. 1.3. Общий вид редактора моделей

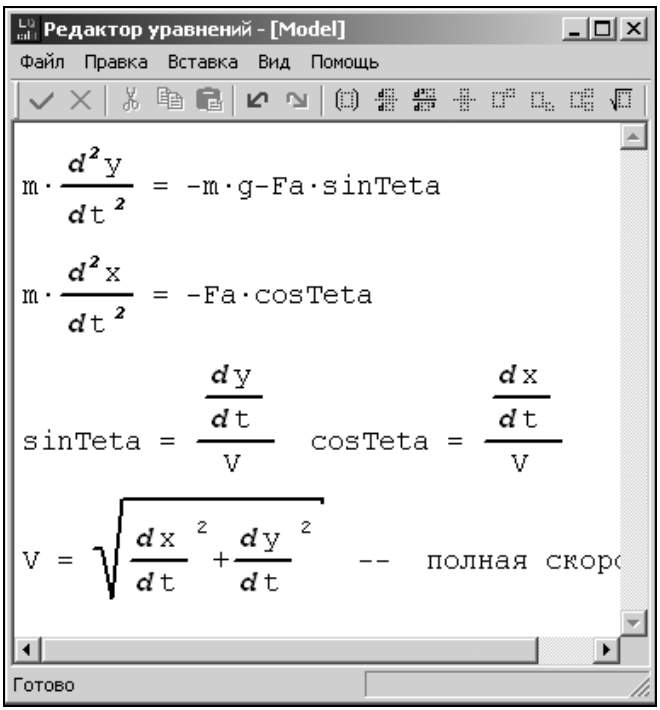


Рис. 1.4. Простейшая система уравнений движения снаряда

Для начала зададим простейшую систему уравнений, описывающую движение снаряда в постоянном по величине и направлению поле тяготения с нулевым значением (пока) силы аэродинамического сопротивления F_a (рис. 1.4). Уравнения можно пояснять комментариями.

Редактор уравнений попросит подтвердить необходимость определения новых переменных, при подтверждении в соответствующем диалоге вводим нужные начальные значения переменных и комментарии. После ввода системы уравнений мы получим соответствующую модель, показанную на рис. 1.5. Для модели элементарной непрерывной системы описание ее поведения содержит только одну вкладку — **Уравнения**. Обратите внимание на то, что для решения дифференциальных уравнений второго порядка необходимы начальные значения первых производных искомых переменных.

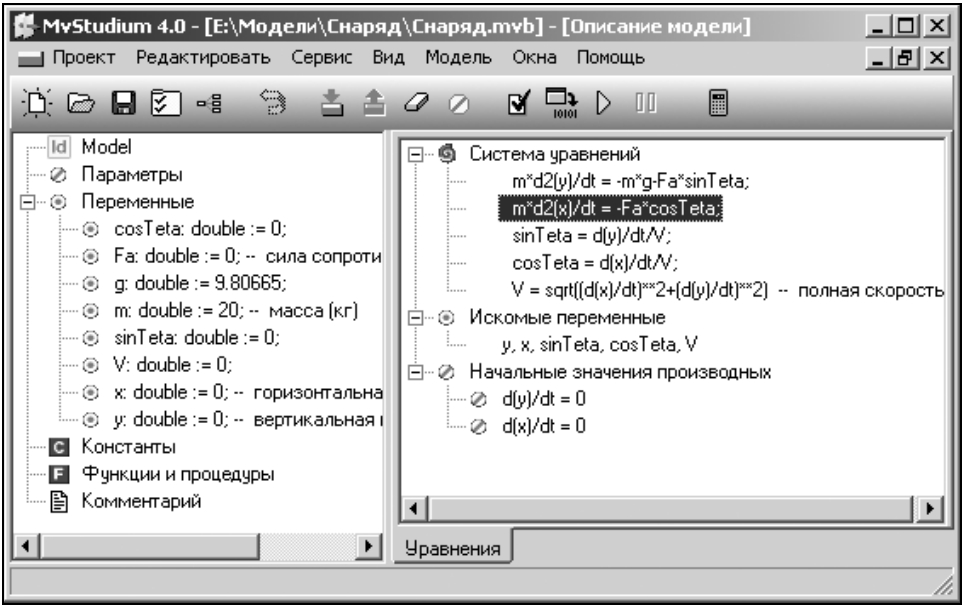


Рис. 1.5. Модель, соответствующая простейшей системе уравнений движения снаряда

Однако получившаяся модель лишена всякого смысла, т. к. при нулевых начальных значениях в ней отсутствует всякая динамика. Введем параметры модели, соответствующие начальной скорости V_0 и начальному углу наклона

траектории θ_0, и зададим соответствующие им начальные значения первых производных (рис. 1.6).

Редактировать уравнения, начальные значения производных, переменные и параметры, а также комментарий к модели можно с помощью команды **Изменить...** (см. рис. 1.6) всплывающего меню, либо с помощью двойного щелчка мышью на изображении соответствующего узла визуального дерева в левой части редактора модели. Добавить или удалить переменные или параметры можно с помощью команд **Добавить...** или **Удалить** всплывающего меню (см. рис. 1.6).

Теперь целесообразно остановиться и провести первый вычислительный эксперимент с простейшей целью проверки правильности нашей модели.

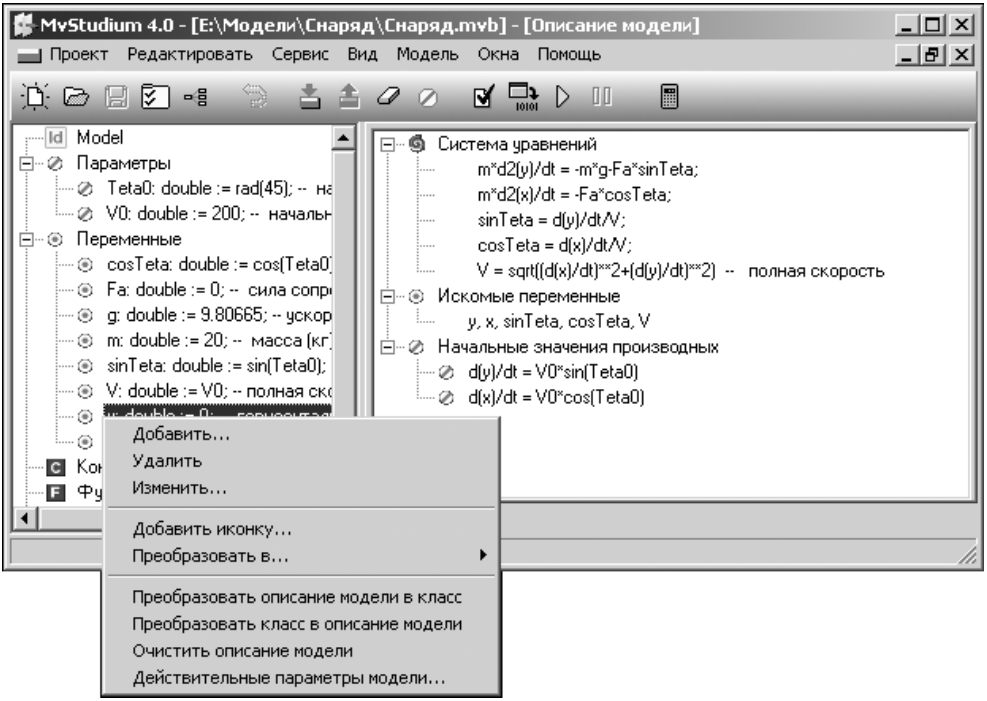


Рис. 1.6. Модель движения снаряда
с ненулевыми начальными условиями

С помощью команды **Модель | Запустить** главного меню или кнопки на инструментальной панели создадим и запустим визуальную модель. Пакет

MvStudium относится к категории "компилирующих": для всех элементов модели генерируется "код" на промежуточном объектно-ориентированном языке программирования, который затем компилируется в машинный код и связывается с исполняющей системой MvStudium с помощью штатного компилятора командной строки этого языка. В данной версии пакета в качестве промежуточного языка используется Borland Object Pascal (Delphi). Вся работа с промежуточным кодом проводится в локальной папке ...\Tmp (в данном примере в ...:\Модели\Снаряд\Tmp), в ней же формируется файл model.exe. Обычно это занимает несколько секунд. При выходе из интегрированной среды все содержимое этой папки автоматически уничтожается. Для того чтобы сохранить модель как независимую от среды разработки выполняемую программу, сначала создайте визуальную модель с помощью команды **Модель | Создать** или кнопки на инструментальной панели, а затем выполните команду **Модель | Сохранить как** главного меню. Если папка проекта недоступна для записи (например, проект находится на компакт-диске), то все временные файлы и файл выполняемой модели формируются в рабочей папке Windows (например, C:\Temp). После создания визуальная модель запускается параллельно с интегрированной средой. В любой момент вы можете прервать выполнение модели из интегрированной среды (например, в случае ее "зависания") с помощью команды **Модель | Стоп** или кнопки на инструментальной панели.

На рис. 1.7 показано начальное состояние главного окна визуальной модели снаряда при первом запуске. Модель находится в точке останова. Для непрерывной модели по умолчанию открываются:

- окно переменных (включая используемые производные);
- окно временной диаграммы для первых четырех искомых переменных.

Не следует забывать, что исходная система уравнений (см. рис. 1.4) автоматически преобразуется к эквивалентной форме, пригодной для численного решения. Для данной модели эквивалентная система имеет следующий вид (перед двоеточием указана переменная, определяемая из данного уравнения):

```
y  : d(y)/dt = y';
y' : d(y')/dt = y'';
x  : d(x)/dt = x';
x' : d(x')/dt = x'';
y'': m*y'' = -m*g-Fay;
x'': m*x'' = -Fax;
```

Вы можете посмотреть численно решаемую систему уравнений, выполнив команду **Показать представление уравнений для численного решения** всплывающего меню на вкладке **Уравнения**.

В нижней части окна указывается информация о типе решаемой в настоящий момент глобальной системы уравнений и ее составе. В данном случае это система дифференциально-алгебраических уравнений (DAE), в которой 4 дифференциальных уравнения и 2 алгебраических.

Замечание

Когда мы станем говорить об элементах описания моделей, то будем вынуждены иногда отступать от обычных обозначений из-за ограниченных возможностей текстового представления. Например, вместо общепринятого обозначения производной по времени $\dot{x}$ мы будем использовать обозначение x'. Знак операции умножения всегда будет указан явно, и `Fax` означает идентификатор переменной, а не произведение $F \cdot a \cdot x$.

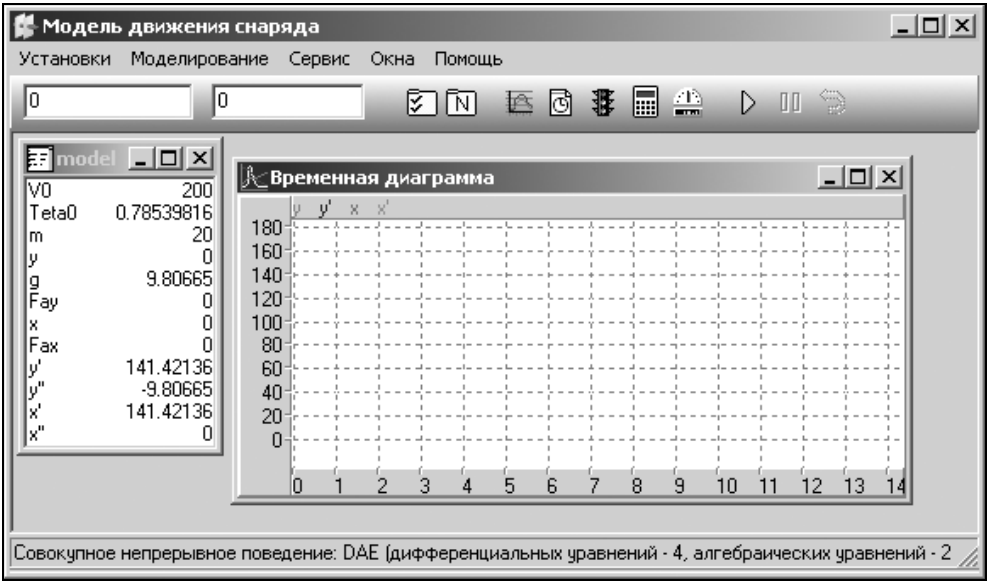


Рис. 1.7. Начальное состояние главного окна визуальной модели снаряда

В заголовок главного окна визуальной модели выводится первая строка комментария для класса `Model` (если комментарий отсутствует, то выводится стандартный заголовок **Визуальная модель MvStudium 4**). В левой части инструментальной панели показаны непрерывная и дискретная составляющие модельного времени.

Значения переменных в окне переменных могут отличаться от начальных значений, указанных в описании модели. Дело в том, что в точках дискретных событий (для непрерывной модели это только точки останова) решается алгебраическая составляющая полной системы уравнений (совокупность алгебраических уравнений и формул), и в окнах показываются согласованные значения переменных. Если переменная является искомой, то ее начальное значение является всего лишь начальным приближением для численного метода.

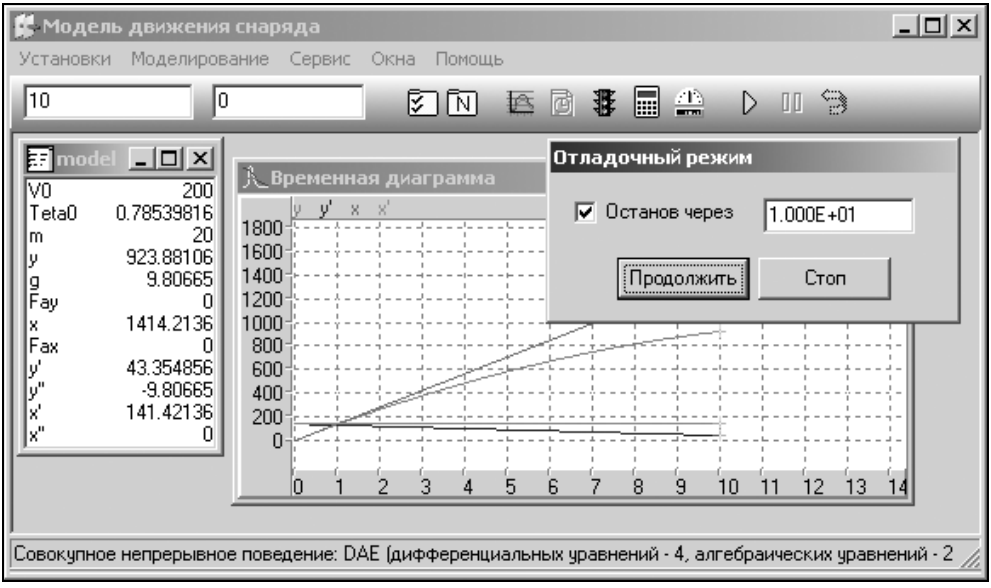


Рис. 1.8. Отладочный останов модели

Запустим продвижение модельного времени. Для этого нужно выполнить команду **Моделирование | Пуск** главного меню или нажать кнопку ▶ на инструментальной панели. Система уравнений модели начнет численно решаться. По умолчанию в опциях численных методов (изменяются с помощью

команды **Установки | Численные методы** или кнопки) выбраны автоматические методы и погрешности 10^{-6}. Эти установки подходят для большинства не очень сложных моделей. Вы увидите, как начнут изменяться значение непрерывной составляющей модельного времени и значения в окне переменных, а также строиться графики на временной диаграмме. По умолчанию в опциях модели (изменяются с помощью команды **Установки | Модель** или кнопки на инструментальной панели) принято соотношение модельного и реального времени "так быстро, как возможно". Для простых уравнений это может быть очень быстро, так что вы ничего не успеете заметить. Поэтому при первом запуске визуальной модели автоматически устанавливается отладочный режим с остановом через 10 единиц непрерывной составляющей модельного времени (см. рис. 1.8).

В нашем случае будет полезнее использовать фазовую диаграмму и отладочный останов по условию. Отменим отладочный останов по времени и остановим эксперимент. В окне диаграммы выполним команду **Настройка** всплывающего меню. В появившемся диалоге настройки (рис. 1.9) щелкнем мышью в столбце **X** строки переменной `x`. Появившийся "плюс" означает, что именно эта переменная будет откладываться по оси абсцисс графика, а не модельное время. Таким образом, мы получаем фазовую диаграмму, на которой изображаются графики зависимостей переменных от выбранной переменной — в данном случае от горизонтальной координаты `x` (если "плюс" убрать, диаграмма вновь станет временной).

С помощью кнопки на инструментальной панели или команды **Сервис | Условия останова** главного меню откроем диалог **Условия останова**, выберем вкладку **По предикату**, с помощью команды **Добавить** всплывающего меню перейдем в диалог редактирования логического предиката и зададим отношение, определяющее момент падения снаряда (рис. 1.10). В этом отношении задано `y<=0`, т. к. определение момента выполнения точного условия `y=0` вызывает большие трудности.

С помощью кнопки на инструментальной панели или команды **Моделирование | Рестарт** главного меню вернем модель в начальное состояние и снова запустим. Теперь сработает отладочный останов в точке падения, и мы получим графики, показанные на рис. 1.11 (график с утолщенной линией соответствует собственно траектории движения снаряда $y(x)$). Характер изменения переменных вполне правдоподобен. Запомним горизонтальную координату точки падения — 4079 м.

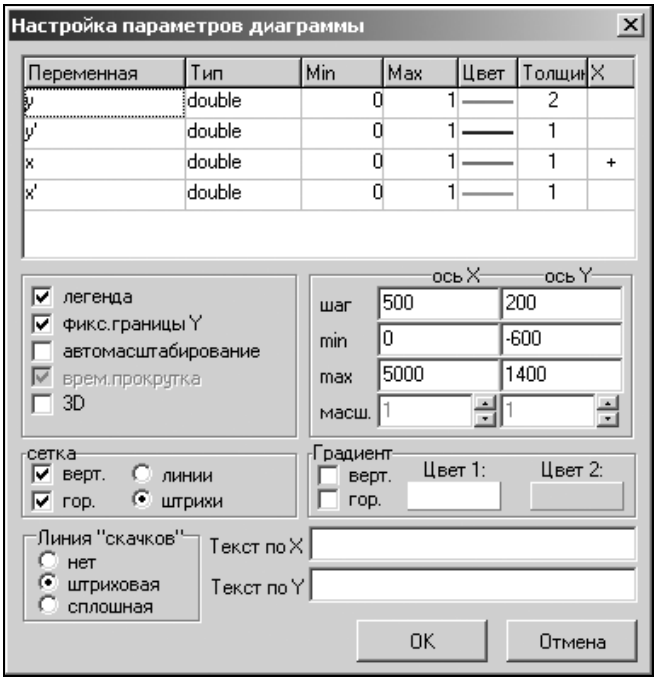


Рис. 1.9. Диалог настройки диаграммы

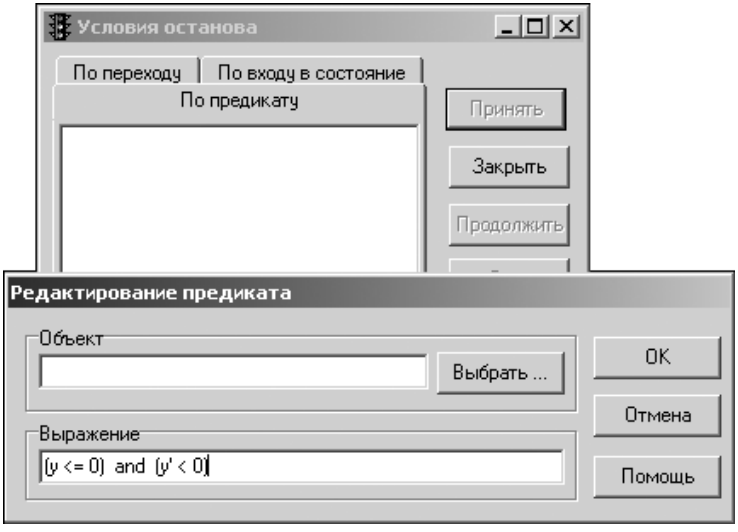


Рис. 1.10. Задание отладочного останова в момент падения снаряда

При отладке динамических моделей часто бывает полезно проверить выполнение различных законов сохранения. В нашем случае сумма кинетической и потенциальной энергии должна быть величиной постоянной при $F_a = 0$ или уменьшаться при $F_a \neq 0$.

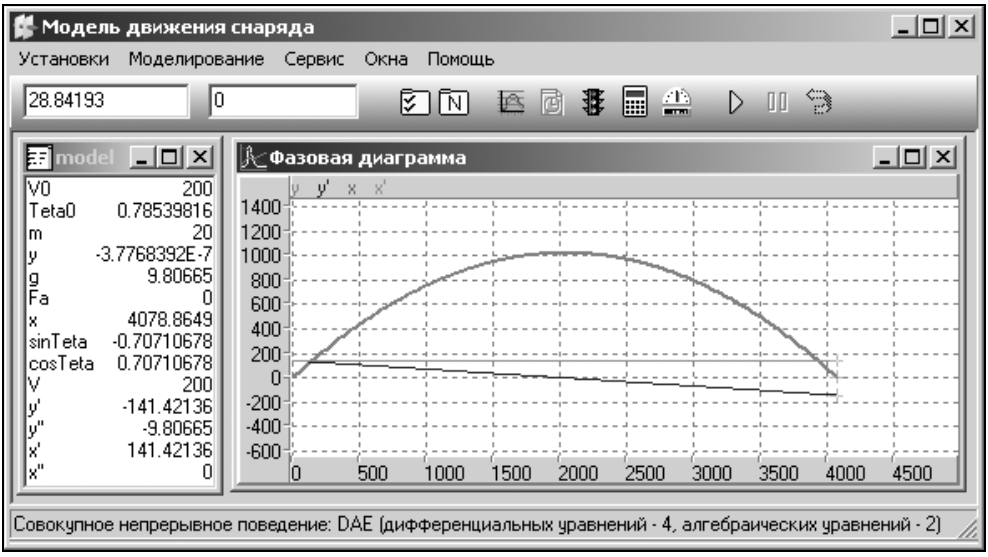


Рис. 1.11. Состояние модели снаряда в момент падения

Для таких целей очень удобно использовать виртуальные переменные. Виртуальной переменной не существует в описании модели, она определяется в визуальной модели вместе с формулой, по которой вычисляется ее значение. Виртуальные переменные можно использовать для построения графиков, анимации, условий останова и т. п. наравне с обычными переменными модели. С помощью команды **Создать виртуальную переменную** всплывающего меню в окне переменных создадим виртуальные переменные $W = \dfrac{m \cdot V^2}{2}$ (рис. 1.12), $Q = m \cdot g \cdot y$ и $E = W + Q$, соответствующие кинетической, потенциальной и полной энергии снаряда.

С помощью кнопки на инструментальной панели или команды **Окна | Новая диаграмма** главного меню создадим новое окно диаграммы (по умолчанию это будет временная диаграмма, т. е. по оси абсцисс будут откла-

дываться значения модельного времени). Методом drag-and-drop перетащим переменные `W`, `Q` и `E` из окна переменных на окно диаграммы. Запустим модель и получим график, показанный на рис. 1.13. График вполне соответствует нашим представлениям о моделируемом физическом процессе.

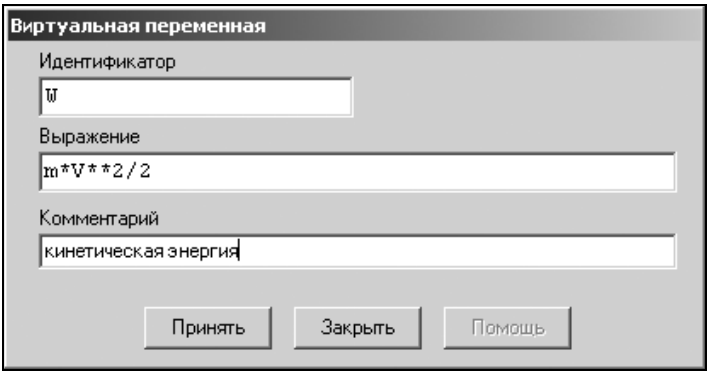


Рис. 1.12. Определение виртуальной переменной

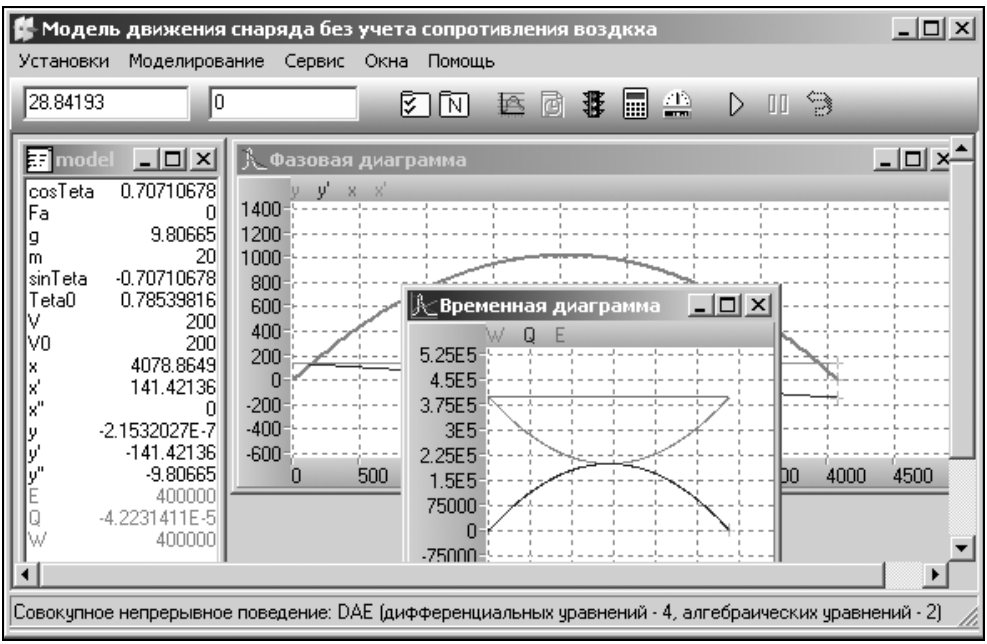


Рис. 1.13. Зависимость энергии от времени

Остается проверить правильность учета сопротивления воздуха. Для простейшей проверки вернем модель в начальное состояние и зададим ненулевое значение переменной F_a (Fa). Активный вычислительный эксперимент предполагает возможность интерактивного вмешательства. Вы можете в любой точке эксперимента изменить значение любой переменной. Для этого нужно выполнить команду **Изменить** всплывающего меню или дважды щелкнуть мышью на нужной строке в окне переменных и затем ввести новое значение в появившемся диалоге (это значение может быть выражением, содержащим другие переменные). После этого вычисляются новые значения зависимых переменных. В случае, если вы изменяете значение параметра, то сначала заново производится вычисление начальных значений переменных, а затем вычисление согласованных значений. Зададим, например, $F_a = 100$. Результаты вычислений показаны на рис. 1.14 и выглядят вполне правдоподобно. Теперь можно перейти к усложнению модели (состояние модели для этой точки разработки сохранено в файле Снаряд_1.mvb).

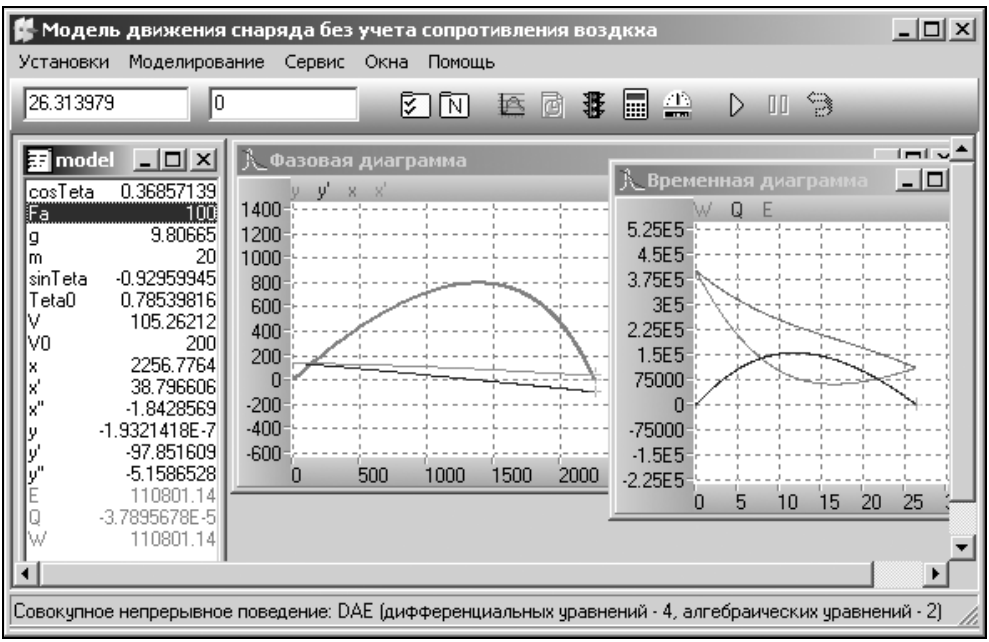


Рис. 1.14. Движение снаряда
при фиксированной силе сопротивления воздуха

Попробуем учесть сопротивление воздуха. Используем зависимость [7]

$$F_a = C_X \cdot S_M \cdot \frac{\rho \cdot V^2}{2}.$$

Здесь S_M — это геометрическая характеристика движущегося объекта, называемая площадью "миделевого сечения". Эту величину сделаем параметром нашей модели и положим $S_M = 0.04\ \text{м}^2$.

C_X — это безразмерный коэффициент силы аэродинамического лобового сопротивления, зависящий от геометрической формы движущегося объекта (строго говоря, здесь мы используем коэффициент сопротивления C_{X_a}, заданный в скоростной системе координат, будем полагать $C_X = C_{X_a}$). Этот коэффициент не является постоянным и зависит от режима обтекания, характеризующегося так называемым "числом Маха" (отношение скорости движения к скорости звука). Поэтому мы введем соответствующую переменную и присвоим ей значение 0.29.

ρ — это плотность воздуха. Пока мы пренебрежем ее зависимостью от высоты и введем соответствующую переменную с начальным значением $\rho = 1.225\ \text{кг/м}^3$ (плотность воздуха на уровне моря при температуре 15 °C).

Добавляем в систему уравнений формулу для F_a и получаем дальность точки падения 2063 м. Это почти в два раза меньше дальности, полученной без учета сопротивления воздуха, значит, предыдущая модель была совсем неадекватной. Состояние модели в данной точке разработки сохранено в файле Снаряд_2.mvb.

Дальнейшее совершенствование нашей модели связано с учетом изменений параметров воздуха в зависимости от высоты. Без этого невозможно также учесть изменение коэффициента C_X, поскольку скорость звука также меняется с высотой.

Используются различные модели атмосферы — "стандартные атмосферы". Все они предусматривают выделение диапазонов высоты ("слоев"), в которых действуют различные зависимости. Для наших примеров достаточно использовать первые три слоя [7]:

- 0 — 11 000 м;
- 11 000 — 25 000 м;
- 25 000 — 46 000 м.

Для первого слоя:

$$T = T_0 + k_1 \times H,$$

$$\rho = \rho_0 \cdot e^{-(1+\frac{G_0}{k_1 \times R_{уд}}) \cdot \ln \frac{T}{T_0}},$$

где

$T_0 = 288.15\ ^\circ K$, $\rho_0 = 1.225$ кг/м^3, $R_{уд} = 287.039$ м^2/град$\cdot$с, $G_0 = 9.80665$ м/с^2, $k_1 = -651\,122 \cdot 10^{-8}$ град/м.

Для второго слоя:

$$T = T_1,$$

$$\rho = \rho_1 \cdot e^{-\frac{G_0}{R_{уд} T}(H - H_1)}.$$

Для третьего слоя:

$$T = T_2 + k_3 \times (H - H_2),$$

$$\rho = \rho_2 \cdot e^{-(1+\frac{G_0}{k_3 \times R_{уд}}) \cdot \ln \frac{T}{T_2}},$$

$$k_3 = 276\ 098 \cdot 10^{-8}.$$

Во всех этих зависимостях под величиной H понимается:

- высота над уровнем моря $H = h$, если не учитывается изменение ускорения силы тяжести с высотой;
- так называемая "геопотенциальная высота" $H = \frac{R_3 \cdot h}{R_3 + h}$, где $R_3 = 6\ 371\ 210$ м — средний радиус Земли, если такое изменение учитывается.

Эти зависимости можно отразить в модели с помощью алгоритмических функций. В данном случае будет вполне достаточно использовать внутренний алгоритмический язык пакета MvStudium. Например, зависимость плотности от высоты может быть задана так:

```
function Rho_H(in H: double) return double
  T: double := T_H(H); -- температура
```

```
begin
  if H<=11000 then
    -- слой 1
    return Rho0*exp(-(1+G0/k1*Rуд)*ln(T/T0));
  elsif H<=25000 then
    -- слой 2
    return Rho1*exp(-G0*(H-H1)/Rуд*T);
  elsif H<=46000 then
    -- слой 3
    return Rho2*exp(-(1+G0/k3*Rуд)*ln(T/T2));
  else
    -- за границей определения модели
    return NaN;
  end if;
end Rho_H;
```

Обратите внимание, что для высот ниже уровня моря неявно распространяется действие зависимостей для слоя 1, а при выходе за верхнюю границу слоя 3 возвращается значение `NaN` ("Not A Number" — предопределенная функция, возвращающая специальное вещественное значение). Существует и второе специальное вещественное значение — `Infinity`, соответствующее очень большой величине, выходящей за границы разрядной сетки компьютера. В окнах переменных выполняемой модели такие значения переменных станут обозначаться соответствующими идентификаторами, а на графиках не будет изображаться ничего.

Заметим, однако, что зависимости параметров атмосферы от высоты имеют изломы, т. е. разрывы производной. Скорее всего, в данном случае это не вызовет каких-либо проблем у численных методов, однако быть уверенными в этом мы не можем. Поэтому подобные зависимости полезно реализовать как гибридный автомат. Таким образом, при более детальном рассмотрении даже чисто непрерывных, на первый взгляд, моделей в них часто появляются зависимости с разрывами, которые приводят к необходимости создания гибридных моделей.

Далее, ясно, что модель атмосферы никак особенно не связана с нашей моделью снаряда и может пригодиться и для других моделей, например, модели ракеты. Поэтому имеет смысл создать отдельный пакет **Земля**, содержащий специфические константы, функции и классы, и импортировать его в специальных моделях.

Для этого снова создадим новый проект в папке ...\Модели\Земля, выбрав вид модели **Объект общего вида**. В менеджере проекта в левой части окна добавим в проект с помощью команды **Добавить...** всплывающего меню нужные константы и функции (в частности, радиус Земли и функцию вычисления геопотенциальной высоты). Теперь добавим новый класс — `СтандартнаяАтмосфера` (рис. 1.15).

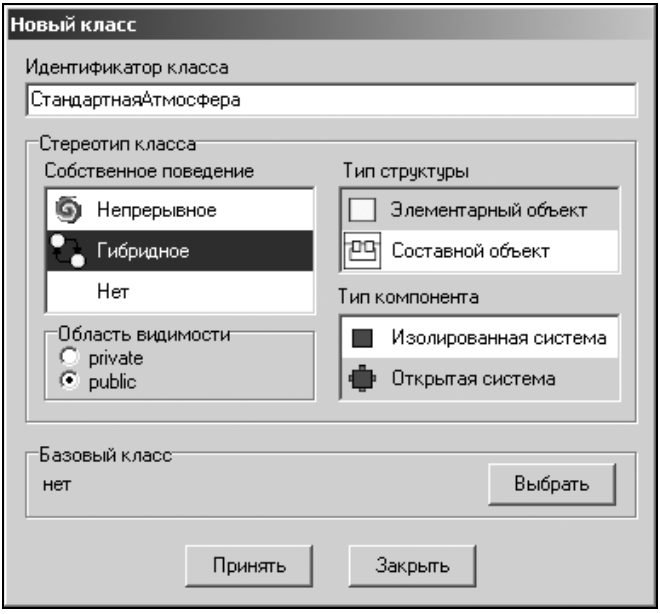


Рис. 1.15. Добавление класса `СтандартнаяАтмосфера`

При создании нового класса указывается начальный стереотип этого класса, включающий тип собственного поведения (в нашем случае **Гибридное**), тип структуры (**Элементарный объект**), тип компонента (в нашем случае это **Открытая система**, поскольку в противном случае этим компонентом не смогут пользоваться другие модели) и область видимости — только в данном пакете (**private**) или в других пакетах, импортирующих данный (**public**). В дальнейшем стереотип класса можно изменить с помощью команды **Преобразовать в ...** всплывающего меню.

Перейдем в окно редактора класса `СтандартнаяАтмосфера`. Оно содержит визуальное дерево атрибутов и методов в левой части и редактор поведения в правой части окна. С помощью команды **Добавить иконку** всплывающе-

го меню визуального дерева присвоим классу подходящий значок, который будет изображаться на всех экземплярах этого класса. С помощью команды **Добавить...** всплывающего меню добавим один вход (высота), три выхода (температура, плотность и скорость звука) и необходимые константы (рис. 1.16).

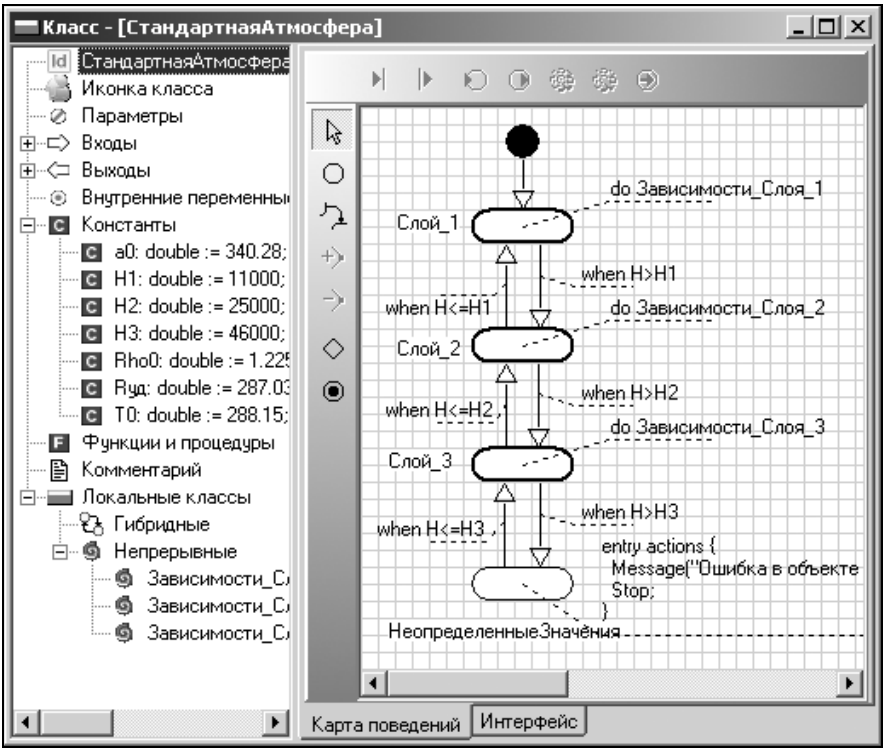


Рис. 1.16. Окно редактора класса `СтандартнаяАтмосфера`

Перейдем теперь на вкладку **Карта поведений**. В редакторе карты поведений имеются две инструментальные панели. Панель слева предназначена для добавления новых элементов: состояний, переходов и опорных точек на переходе. Верхняя панель служит для редактирования свойств выделенного элемента (условий и действий перехода, входных/выходных действий и локальной деятельности в состоянии). Карта поведений нового гибридного класса при первом запуске содержит начальное состояние, обычное состояние `S1` и безусловный переход между ними. Состояние `S1` переименуем в

`Слой_1` и расположим состояния вертикально, переместив с помощью указателя мыши. Положение всех элементов выравнивается по клеткам сетки. Нам будут нужны также состояния `Слой_2` и `Слой_3`, а также состояние `НеопределенныеЗначения`, соответствующее выходу за границы определения модели атмосферы. Для добавления нового обычного состояния нажмем кнопку на левой инструментальной панели и нажмем левую кнопку мыши в точке, соответствующей верхнему левому углу нового состояния. Для добавления нового перехода нажмем кнопку на левой инструментальной панели, нажмем кнопку мыши на изображении исходного состояния перехода, не отпуская ее, переместим курсор мыши на изображение конечного состояния перехода и отпустим кнопку. Переход можно также добавлять "по частям", отпустив кнопку на свободном поле, а затем использовать конечную точку как исходное состояние.

Введем теперь условия срабатывания переходов. Для этого выделим нужный переход и нажмем кнопку на верхней инструментальной панели. В появившемся диалоге введем условие срабатывания (рис. 1.17).

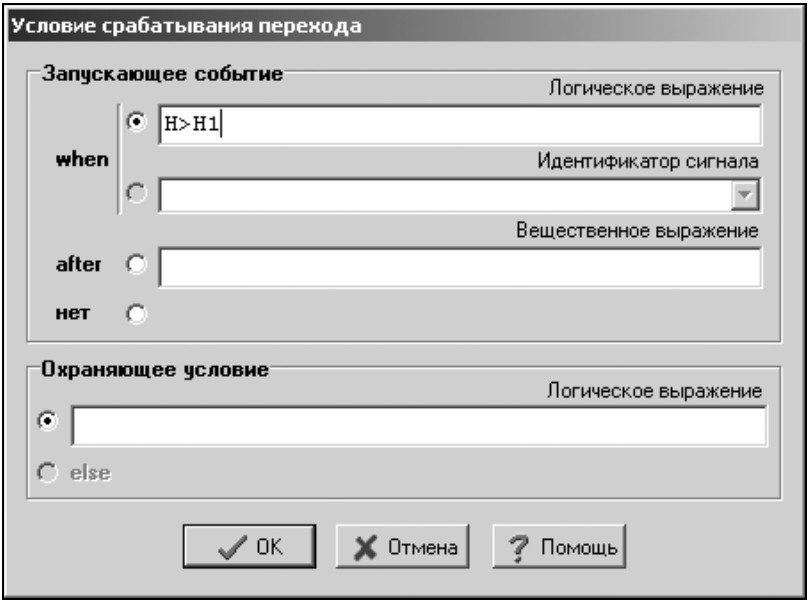


Рис. 1.17. Условие срабатывания перехода

Далее выделяем состояние `НеопределенныеЗначения` и с помощью кнопки верхней инструментальной панели вводим мгновенную последовательность входных действий: при попадании в это состояние выводится сообщение об ошибке (выход за область определения), и модель останавливается.

Теперь нужно задать зависимости температуры, плотности воздуха и скорости звука от высоты для трех слоев. Для этого нужно создать три локальных непрерывных класса, экземпляры которых будут приписаны в качестве деятельности соответствующим состояниям. Выделим, например, состояние `Слой_1` и выполним команду **Создать непрерывную деятельность** всплывающего меню. В классе `СтандартнаяАтмосфера` появится локальный непрерывный класс с именем `Деятельность_в_Слой_1` по умолчанию. Переименуем его в `Зависимости_Слоя_1`. Локальный класс отличается тем, что в нем видимы все элементы описания класса, в который он вложен. В данном случае в определении класса `Зависимости_Слоя_1` видимы все константы и переменные класса `СтандартнаяАтмосфера`. Кроме того, в определении локального класса могут быть введены собственные константы, переменные и функции. Экземпляры локального класса могут использоваться только как деятельности в состояниях карты поведений охватывающего класса. В данном случае введем собственную константу `k1`, необходимую только для задания системы уравнений класса `Зависимости_Слоя_1` (рис. 1.18).

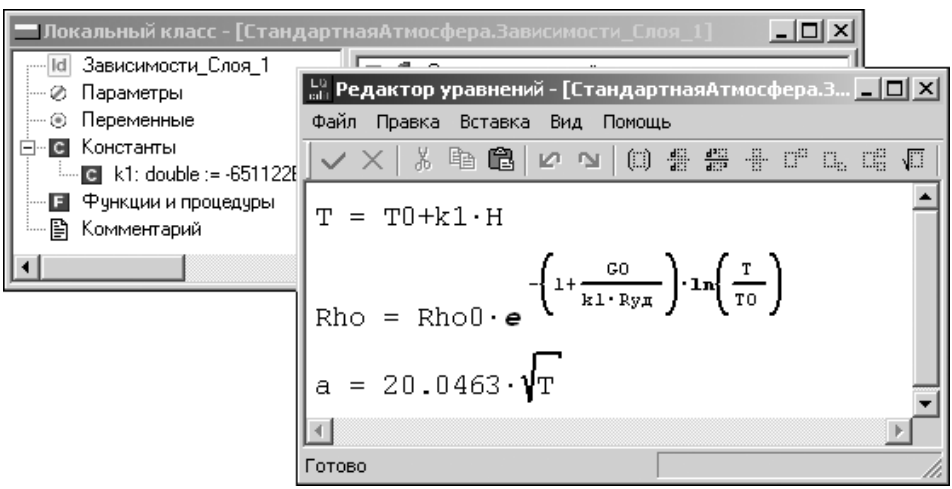


Рис. 1.18. Локальный класс `Зависимости_Слоя_1`

Аналогично введем определения локальных классов `Зависимости_Слоя_2` и `Зависимости_Слоя_3`, экземпляры которых соответственно будут являться деятельностями в состояниях `Слой_2` и `Слой_3`.

Теперь класс `СтандартнаяАтмосфера` определен, и нужно проверить его работоспособность. Откроем вкладку **Структура** класса `Model`, схватим мышью узел класса `СтандартнаяАтмосфера` в дереве классов менеджера проекта, перетащим его в окно редактора класса `Model` и бросим на вкладке **Структура**. В локальной структуре класса `Model` появится объект — экземпляр класса `СтандартнаяАтмосфера`, который назовем `Атмосфера` (рис. 1.19). Однако этого недостаточно. Если мы просто запустим модель в таком виде, то в ней не будет ничего происходить, т. к. модель атмосферы явно не зависит от времени. Для испытаний необходимо изменять значение высоты. Поэтому в дереве **Импортируемые пакеты** менеджера проекта развернем узел `SysLib`, выберем стандартный класс `CRampGenerator` (генератор равномерно нарастающего сигнала) и методом drag-and-drop бросим его на вкладке **Структура** класса `Model`. Появившийся экземпляр этого класса назовем `RG` (см. рис. 1.19).

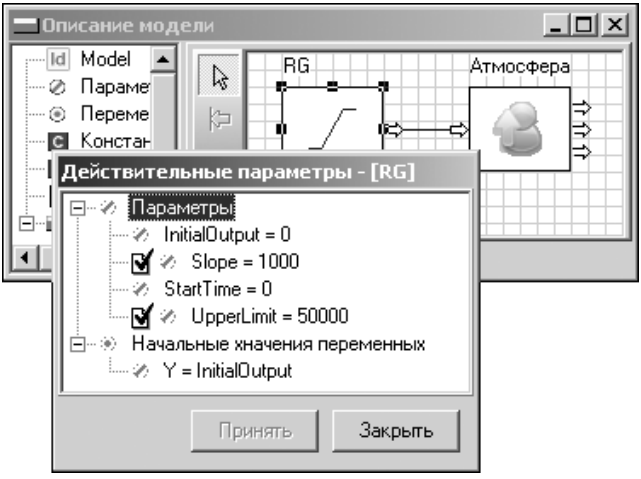


Рис. 1.19. Испытание модели атмосферы

С помощью команды **Действительные параметры** всплывающего меню изменим значения параметров объекта `RG`: нарастание высоты со скоростью 1000 м/с и верхний предел 50 000 м (см. рис. 1.19). Соединим его выход `RG.Y` с входом `Атмосфера.H`. Для этого нажмем кнопку на левой инструментальной панели редактора структуры. Редактор переходит в режим добавле-

ния новой связи, признаком чего является появление крестообразного курсора мыши. Подводим курсор на изображение внешней переменной `RG.Y`, курсор при этом сменится на изображение креста в круге. Затем нажмем левую кнопку мыши и, не отпуская ее, переместим курсор на изображение внешней переменной `Атмосфера.H`. Курсор должен измениться на изображение креста в круге, если выбранные переменные можно соединять. После изменения курсора отпустим кнопку. Возможно и проведение связи в обратном порядке, поскольку ее направленность однозначна.

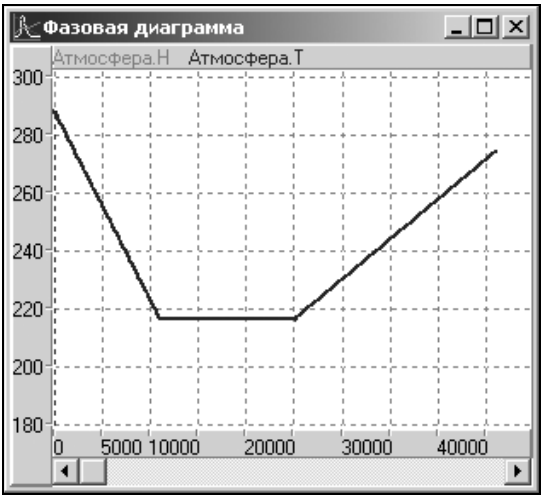
Теперь запускаем визуальную модель и получаем фазовые диаграммы, показанные на рис. 1.20 (следует понимать, конечно, что сразу мы таких зависимостей не получили и исправили множество ошибок в модели).

Существует еще один прием для отладки моделей систем, не зависящих от времени, — интерактивное воздействие. Удалим объект `RG` и запустим снова визуальную модель. С помощью кнопки на инструментальной панели или команды **Сервис | Стандартные 2D-компоненты** главного меню откроем панель стандартных анимационных 2D-компонентов, выберем какой-нибудь компонент непрерывного ввода, например, **Ползунок** (можно и **Круговой регулятор**), перетащим его мышью в окно структуры и бросим на изображении входа `H` объекта `Атмосфера`. В окне структуры появится изображение ползунка, положение которого связано со значением переменной `H`. Настроим его с помощью всплывающего меню: расположим вертикально и установим диапазон значений 0—50 000 (рис. 1.21).

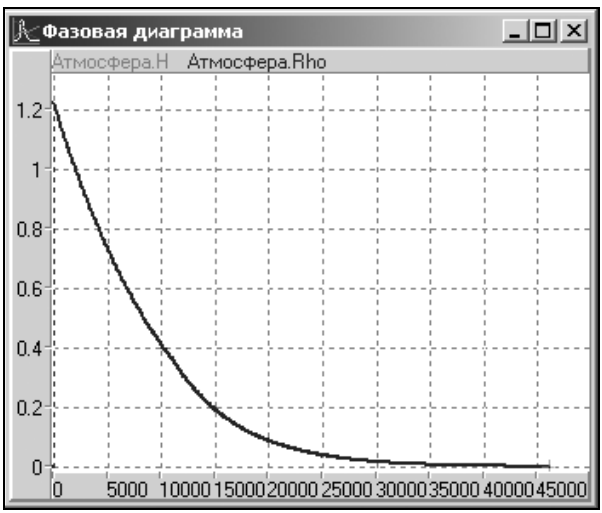
Теперь, перемещая ползунок с помощью мыши, мы можем менять значение переменной `H` в нужном диапазоне значений и получить те же самые графики (см. рис. 1.20). Правда, они будут состоять из точек, т. к. мы перемещаем мышь небольшими рывками, воспринимаемыми моделью как дискретные события. Непрерывная составляющая модельного времени при этом остается равной нулю, а дискретная составляющая изменяется, т. к. при изменении высоты срабатывают переходы в карте поведений объекта `Атмосфера`.

Полученные графики очень похожи на правильные, поэтому считаем отладку законченной и возвращаемся к проекту **Снаряд**.

С помощью кнопки на инструментальной панели интегрированной среды или команды **Вид | Менеджер проекта** главного меню делаем видимым менеджер проекта. Выделяем узел **Импортируемые пакеты** и выполняем команду **Добавить**. В появившемся диалоге **Выберите пакет** сразу показываются пакеты, находящиеся в папке ...\MvStudium4\Lib, поэтому часто используемые пакеты имеет смысл поместить в эту папку.



а



б

Рис. 1.20. Зависимости температуры (*а*) и плотности (*б*) от высоты

В нашем случае используем кнопку **Другие** и укажем путь к пакету **Земля**. Следует отметить, что если вы потом решите переместить импортируемые пакеты в другие папки, то нужно будет исправить пути к папкам импорти-

руемых пакетов в установках проекта. Теперь мы можем использовать в нашем проекте видимые извне элементы пакета **Земля** (рис. 1.22).

Рис. 1.21. Ручное изменение высоты с помощью интерактивного 2D-компонента

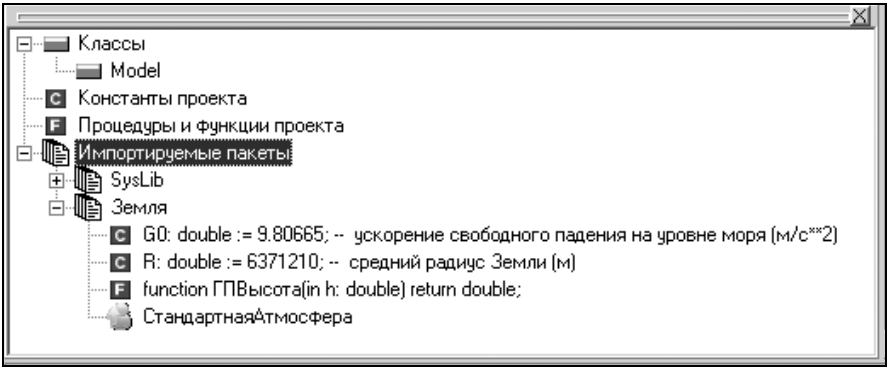


Рис. 1.22. Видимые элементы пакета **Земля**

Теперь нам нужно добавить в модель снаряда локальный объект — экземпляр класса `Земля.СтандартнаяАтмосфера` (все элементы из импортируемого пакета имеют в проекте составные имена с префиксом пакета). Для этого сначала выполним в редакторе класса `Model` команду всплывающего меню **Преобразовать в ... | Составной объект**. После этого в редакторе класса появится вкладка **Структура**. Перетащим на нее мышью методом drag-and-drop узел класса `СтандартнаяАтмосфера` из дерева пакета **Земля** в менеджере проекта (см. рис. 1.22). Появится новый локальный объект, который мы переименуем в `Atm` (рис. 1.23).

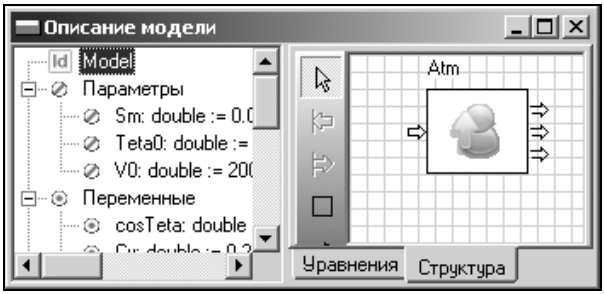


Рис. 1.23. Локальный объект, моделирующий атмосферу

Переходим на вкладку **Уравнения** и добавляем два новых уравнения:

```
Atm.H = y
Rho = Atm.Rho
```

Запускаем визуальную модель и получаем дальность падения 2097 м. Это на 34 м дальше, чем в модели без учета изменения плотности воздуха (результат качественно объясним — на верхней части траектории плотность и, соответственно, сопротивление воздуха меньше). Это уже не двукратная ошибка, но все же приличная величина. Состояние модели в данной точке разработки сохранено в файле Снаряд_3.mvb.

Следующим шагом будет учет зависимости коэффициента аэродинамического сопротивления C_X от числа Маха. Пусть эта зависимость имеет следующий вид (M — число Маха):

$$C_X = \begin{cases} 0.29 & \text{при } 0 \le M \le 0.8, \\ M - 0.51 & \text{при } 0.8 \le M \le 1.068, \\ 0.091 + \dfrac{0.5}{M} & \text{при } M \ge 1.068. \end{cases}$$

Эта зависимость относится исключительно к снаряду и, следовательно, должна отражаться в пакете **Снаряд**. В то же время для снарядов различной формы эта зависимость также меняется. Однако мы сейчас моделируем конкретный снаряд и потому оставим пока эти вопросы в стороне. Реализовывать зависимость $C_X(M)$ в виде алгоритмической функции не очень хорошо, т. к. она не является гладкой.

Зависимости такого вида очень удобно представлять в виде так называемых "условных уравнений":

$$C_X = \texttt{if}\ \ M > 1.068\ \ \texttt{then}\ \ 0.091 + \frac{0.5}{M}\ \texttt{else if}\ M > 0.8$$

$$\texttt{then}\ M - 0.51\ \texttt{else}\ 0.29$$

Условное уравнение принципиально отличается от аналогичного условного оператора в теле алгоритмической функции тем, что при переходе выражения в правой части с одной логической ветви на другую автоматически генерируется дискретное событие и возникает "временная щель". Введем еще одну переменную M и дополним систему уравнений указанным выше условным уравнением, а также формулой $M = \dfrac{V}{Atm.a}$.

Попробуем провести вычислительный эксперимент. Получим тот же результат — 2097 м. Это неудивительно, поскольку начальная скорость существенно меньше скорости звука. Заметим, что те же эксперименты с начальной скоростью 400 м/с дают уже разницу в 342 м. Состояние модели в данной точке разработки сохранено в файле Снаряд_4.mvb.

Теперь попробуем учесть изменение силы тяжести с высотой, используя модель центрального поля тяготения. Зависимость ускорения свободного падения от высоты в этом случае имеет вид

$$g = g_0 \cdot \frac{R_З^2}{(R_З + h)^2},$$

где $R_З$ — средний радиус Земли; g_0 — ускорение свободного падания на уровне моря. Это непрерывная гладкая функция от h, поэтому реализуем ее в пакете **Земля** алгоритмической функцией

```
function g (in h: double) return double is
begin
  return G0*R**2/(R+h)**2;
end g;
```

Вернувшись к пакету **Снаряд**, мы получаем сообщение о том, что изменилась версия импортируемого пакета **Земля**, и необходимо обновить проект. После обновления ссылок в дереве импортируемых элементов появляется новая функция, и мы ее используем, добавив уравнение

```
g = Земля.g(y)
```

Кроме того, теперь вместо геометрической высоты для вычислении параметров атмосферы нужно использовать геопотенциальную высоту, поэтому нужно изменить уравнение

```
Atm.H = Земля.ГПВысота(y)
```

Проведя вычислительный эксперимент, получаем разницу дальностей в 0.2 м. Состояние модели в данной точке разработки сохранено в файле Снаряд_5.mvb.

Наконец, последним уточнением нашей модели будет учет кривизны Земли. Примем сферическую модель Земли (более точная модель — эллипсоид). Ось ординат стартовой системы координат направим по направлению местной вертикали в точке стрельбы, а ось абсцисс по направлению стрельбы (рис. 1.24).

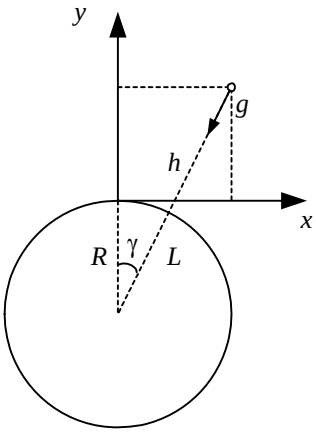


Рис. 1.24. Учет кривизны Земли (сферическая модель)

Из рисунка получаем очевидные соотношения:

$$Y = \sqrt{x^2 + (R + y)^2},$$

$$h = Y - R,$$

$$\sin\gamma = \frac{x}{Y}, \quad \cos\gamma = \frac{y + R}{Y},$$

$$L = R \cdot \gamma,$$

$$g_X = g \cdot \sin\gamma,$$

$$g_Y = g \cdot \cos\gamma,$$

где Y — текущее расстояние до центра Земли; h — текущая высота, которую теперь нужно использовать вместо y для вычисления параметров атмосферы и ускорения свободного падения; L — текущая дальность по поверхности Земли; g_X, g_Y — составляющие ускорения свободного падения по осям стартовой системы координат.

Поправим систему уравнений и проведем вычислительный эксперимент. Условием падения снаряда теперь является выражение $(h \le 0) \wedge (h' < 0)$. Для снаряда вполне можно вместо h' использовать y'. Как и следовало ожидать, влияние кривизны Земли на дальность падения для таких расстояний ничтожно мало. Состояние модели в данной точке разработки сохранено в файле Снаряд_6.mvb.

Создание и отладка модели гибридной системы

В качестве гибридной модели мы рассмотрим модель движения ракеты. Эта модель является гибридной, т. к. траектория движения ракеты в общем случае включает активный участок, на котором работает реактивный двигатель, и пассивный участок, на котором он не работает. Соответственно в модели ракеты обязательно должно фигурировать дискретное событие "отсечка тяги" (это может быть следствием целенаправленного управляющего воздействия или просто выгоранием всего топлива до конца).

Для начала создадим простую модель неуправляемой твердотопливной ракеты с полным выгоранием топлива, стартующей с наклонной направляющей. Создадим новый проект **Ракета** и выберем вид модели — **Гибридный элементарный объект**. Поскольку для расчета траектории ракеты нам понадобятся параметры атмосферы и гравитационного поля, импортируем в новый проект уже разработанный и отлаженный на модели снаряда пакет **Земля**.

При разработке гибридной модели важно правильно выделить качественные состояния моделируемой системы, минимизировав число переходов между ними, а также независимые составляющие поведения. На первый взгляд в нашей задаче можно выделить в качестве состояний активный и пассивный участки, т. е. классифицировать качественные состояния по работе двигателя. Однако более пристальное изучение динамики ракеты показывает, что это будет не слишком удачным решением. Движение ракеты включает две фазы: движение по направляющей и последующий полет до точки падения.

В зависимости от набора стартовых параметров активный и пассивный участки могут сочетаться с этими фазами самым различным образом. При очень малом запасе топлива или очень малой силе тяги двигатель теоретически может закончить свою работу на направляющей. Ракета может также упасть с работающим двигателем. Наконец, ракета может не сойти с направляющей. Правильнее будет выделить как качественные состояния фазы движения.

Во время движения по направляющей на ракету действуют сила тяги двигателя P, сила тяжести $m \cdot g$, сила аэродинамического сопротивления F_a и сила трения скольжения. Динамика ракеты задается уравнениями

$$\begin{cases} m \cdot \dfrac{dV}{dt} = P - k_{\text{тр}} \cdot m \cdot g \cdot \cos\theta_0 - m \cdot g \cdot \sin\theta_0 - F_a, \\ \dfrac{dL}{dt} = V, \\ x = L \cdot \cos\theta_0, \\ y = L \cdot \sin\theta_0. \end{cases}$$

Условие схода с направляющей: $L \geq L_{\text{напр}}$.

Сила аэродинамического сопротивления на всех участках определяется одинаково $F_a = C_X \cdot S_M \cdot \dfrac{\rho \cdot V^2}{2}$. Зависимость аэродинамического коэффициента C_X от числа Маха примем такую же, что и в модели снаряда.

Динамика реактивного двигателя в первом приближении задается уравнениями

$$\begin{cases} P = P_{\text{уд}} \cdot w \cdot g, \\ \dfrac{dM_{\text{т}}}{dt} = -w, \end{cases}$$

где $P_{\text{уд}}$ — так называемая "удельная тяга" — совокупная характеристика конструкции двигателя и энергетических возможностей топлива, показывающая, какая тяга приходится на единицу массового секундного расхода топлива w. В нашем примере положим $P_{\text{уд}} = 2400$ н/(кг/с). Положим начальную массу топлива $M_{\text{т}} = 50$ кг и секундный расход $w = 5$ кг/с. Условие отключения двигателя по выгоранию всего топлива: $(M_{\text{т}} \leq 0) \wedge (w > 0)$.

Полная масса ракеты определяется уравнением $m = M_{\text{н}} + M_{\text{к}} + M_{\text{т}}$, где $M_{\text{н}}$ — масса полезной нагрузки (положим ее равной 20 кг); $M_{\text{к}}$ — масса конструкции двигателя, определяемая как $M_{\text{к}} = \dfrac{M_{\text{т}}(0)}{1 - K_{\text{дв}}}$, где $K_{\text{дв}}$ — коэффициент конструкции двигателя, показывающий долю конструкции в стартовой массе двигателя (положим его равным 0.1).

Во время свободного полета на ракету действуют сила тяги двигателя P, сила тяжести $m \cdot g$ и сила аэродинамического сопротивления F_a. Динамика ракеты задается уравнениями

$$\begin{cases} m \cdot \dfrac{d^2 y}{dt^2} = -m \cdot g - F_a \cdot \sin Teta + P \cdot \sin Teta, \\ m \cdot \dfrac{d^2 x}{dt^2} = -F_a \cdot \cos Teta + P \cdot \cos Teta, \\ \sin Teta = \dfrac{\frac{dy}{dt}}{V}, \quad \cos Teta = \dfrac{\frac{dx}{dt}}{V}, \\ V = \sqrt{(\dfrac{dy}{dt})^2 + (\dfrac{dx}{dt})^2}. \end{cases}$$

Условием падения и окончания вычислительного эксперимента является

$$(y \le 0) \wedge \left(\frac{dy}{dt} < 0 \right).$$

Для вычисления значения плотности воздуха нам понадобится экземпляр класса `Земля.СтандартнаяАтмосфера`. Поэтому с помощью команды **Преобразовать в ... | Составной объект** всплывающего меню преобразуем модель ракеты в составной объект и создадим на вкладке **Структура** объект `Atm` класса `Земля.СтандартнаяАтмосфера`. Затем в редакторе карт поведения создадим карту поведения ракеты (рис. 1.25).

Состоянию `Направляющая` приписан экземпляр непрерывного локального класса `Движение_по_направляющей`, который содержит уравнения движения по направляющей, уравнения для определения силы аэродинамического сопротивления и уравнения двигателя, а также определение локальной константы — коэффициента трения для направляющей.

Состоянию `Полет` приписан экземпляр непрерывного локального класса `Динамика_ракеты`, который содержит уравнения движения ракеты как материальной точки, уравнения для определения силы аэродинамического сопротивления и уравнения двигателя.

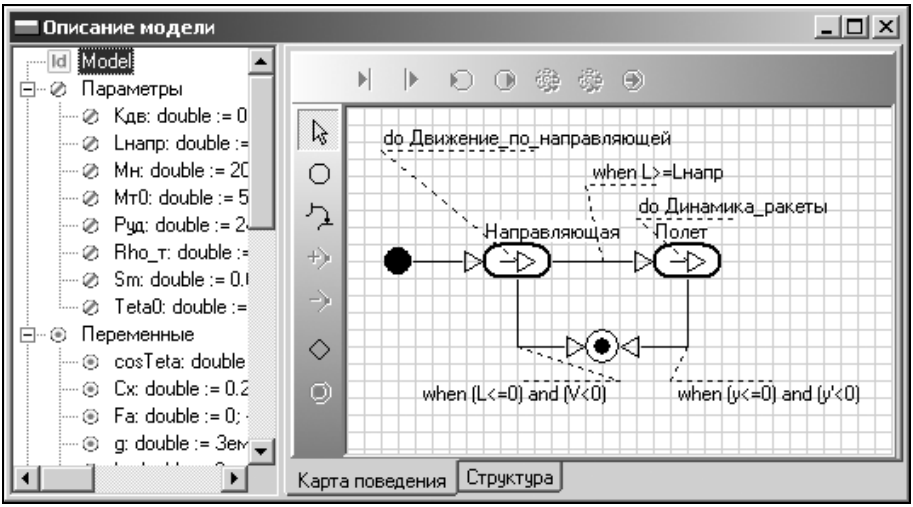


Рис. 1.25. Карта поведения неуправляемой ракеты

Между этими состояниями и конечным состоянием имеются переходы, которые срабатывают при выполнении условий падения ракеты или отката ракеты в начало направляющей (для реальной ракеты последний случай трудно себе представить, но в нашей демонстрационной модели он иллюстрирует необходимость учитывать все возможные качественные переходы). При переходе в конечное состояние карты поведения модели вычислительный эксперимент заканчивается, так что нам не придется вводить специальные отладочные условия останова, как мы это делали при отладке модели снаряда.

Отключение ракетного двигателя моделируется с помощью внутренних переходов в обоих состояниях (рис. 1.26). Для создания внутренних переходов в состоянии нужно выделить это состояние и либо нажать кнопку на верхней инструментальной панели, либо выполнить команду **Внутренние переходы** всплывающего меню.

Срабатывание внутреннего перехода в отличие от внешнего не приводит к выходу из текущего состояния. Таким образом, продолжает решаться теку-

щая система уравнений, но с нулевыми значениями массы топлива и секундного расхода топлива (соответственно и силы тяги).

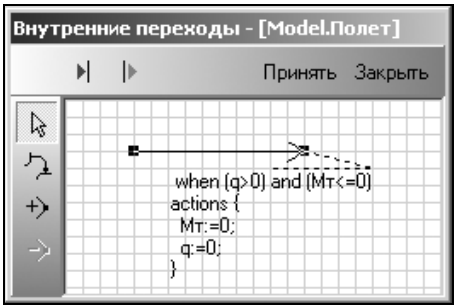


Рис. 1.26. Внутренний переход, моделирующий отключение двигателя

Создадим визуальную модель, запустим ее и получим для наших начальных условий траекторию, показанную на рис. 1.27 (толстой линией показана зависимость силы тяги от дальности).

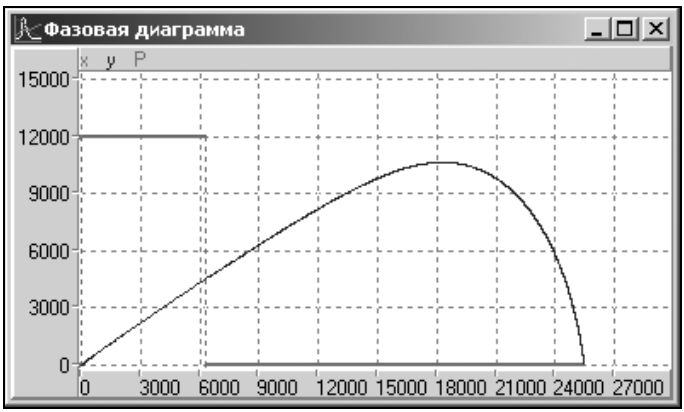


Рис. 1.27. Траектория ракеты для угла старта 45° и секундного расхода 5 кг/с

Несложные эксперименты с моделью показывают, что для ракеты оптимальным углом старта явно является угол, больший, чем 45° — при угле наклона

направляющей в 75° дальность полета увеличивается почти в два раза (рис. 1.28).

Этот результат на первый взгляд кажется весьма странным: известно, что для снаряда при наличии атмосферы оптимальными являются углы бросания, несколько меньшие чем 45°. Такой эффект может свидетельствовать об ошибке в модели, и потому с ним следует разобраться.

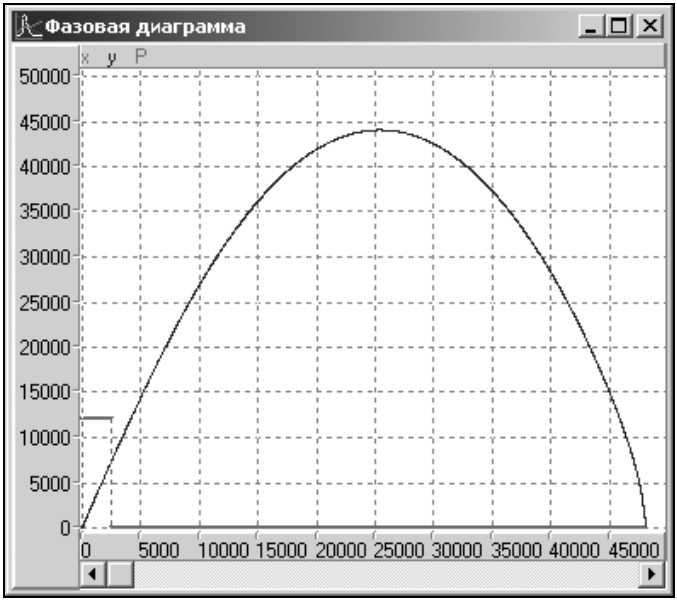
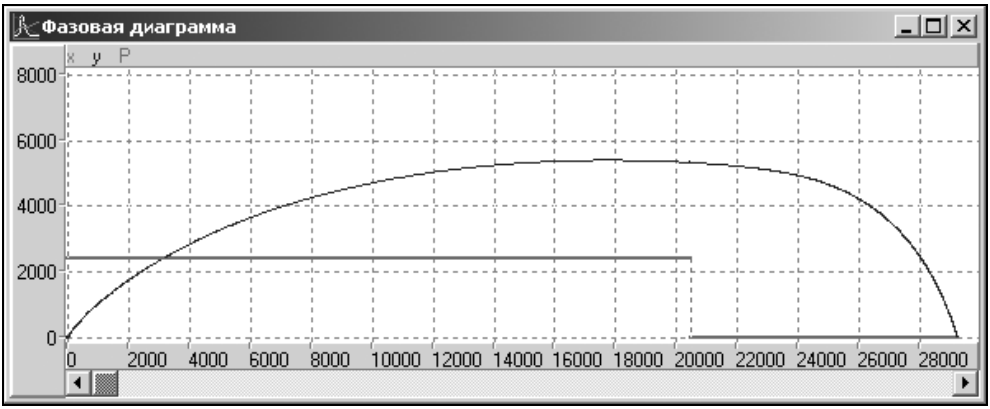


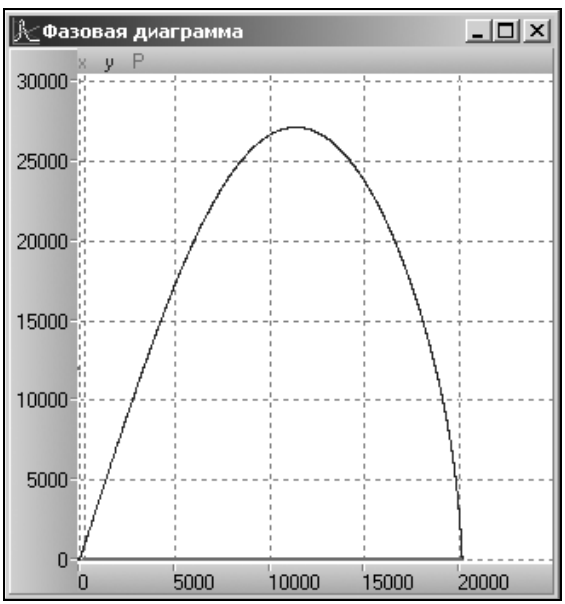
Рис. 1.28. Траектория ракеты для угла старта 75° и секундного расхода 5 кг/с

В отличие от снаряда дальность полета ракеты существенно зависит не только от угла наклона направляющей, но и от секундного расхода топлива (рис. 1.29). Если наша модель правильна, то при очень больших значениях секундного расхода топлива оптимальный угол старта должен приближаться к 45°.

Однако здесь при исследовании различных сочетаний угла старта и секундного расхода возникают неожиданные трудности. При некоторых сочетаниях этих параметров траектория ракеты выходит за границы нашей трехслойной модели атмосферы (рис. 1.30).



а



б

Рис. 1.29. Траектория ракеты для угла старта 75° и секундного расхода 1 кг/с (*а*) и 50 кг/с (*б*)

Поэтому для дальнейших экспериментов нам придется модифицировать класс `СтандартнаяАтмосфера` в пакете **Земля**, введя в модель описание 11 слоев до высоты 200 км. Кроме того, целесообразно ввести в модель и за-

висимости давления от высоты. Состояние пакета **Земля** до модификации сохраним в файле Земля_3.mvb.

Рис. 1.30. Траектория ракеты для угла старта 75° и секундного расхода 2 кг/с — выход за область определения стандартной атмосферы

"Трехслойную" модель атмосферы мы строили, создавая для каждого слоя отдельное состояние, которому приписывали в качестве непрерывной деятельности соответствующие зависимости параметров атмосферы от высоты. Такой подход был очень прост и нагляден для 3-х слоев, однако для 11 слоев он является довольно утомительным. Для упрощения карты поведения класса `СтандартнаяАтмосфера` воспользуемся тем, что для всех слоев на самом деле используются зависимости только двух видов: зависимости для изотермического слоя и зависимости для слоя с линейно изменяющейся температурой (линейного слоя) с набором параметров, уникальным для каждого конкретного слоя.

Удалим все элементы описания поведения в классе `СтандартнаяАтмосфера`. Для изменений, связанных с глубоким преобразованием описания пакета, часто бывает удобно использовать текстовое представление проекта. Для получения текстового представления выполним команду **Проект | Экспорт в MVL** главного меню. После редактирования полученного текстового файла в любом текстовом редакторе выполним команду **Проект | Импорт из MVL**, указав измененный файл в качестве источника, и в результате получим соответствующую внесенным изменениям базу данных проекта.

Прежде всего, добавим в классе `СтандартнаяАтмосфера` новый выход `p` — давление. Далее создадим два непрерывных локальных класса: `ИзотермическийСлой` (рис. 1.31) и `ЛинейныйСлой` (рис. 1.32).

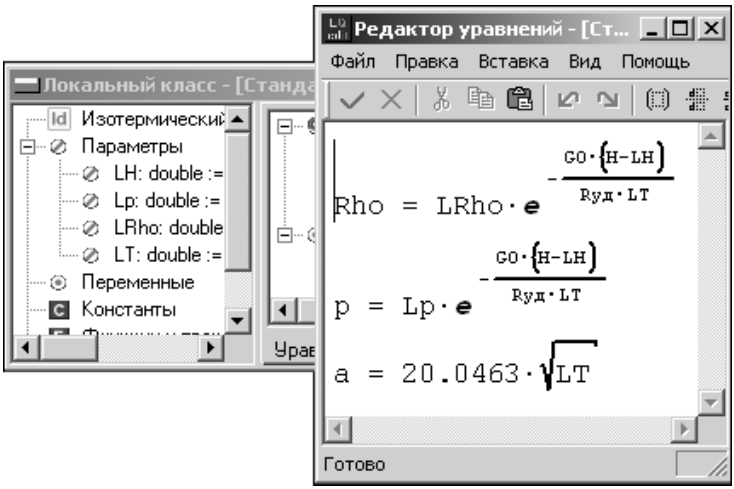


Рис. 1.31. Локальный класс `ИзотермическийСлой`

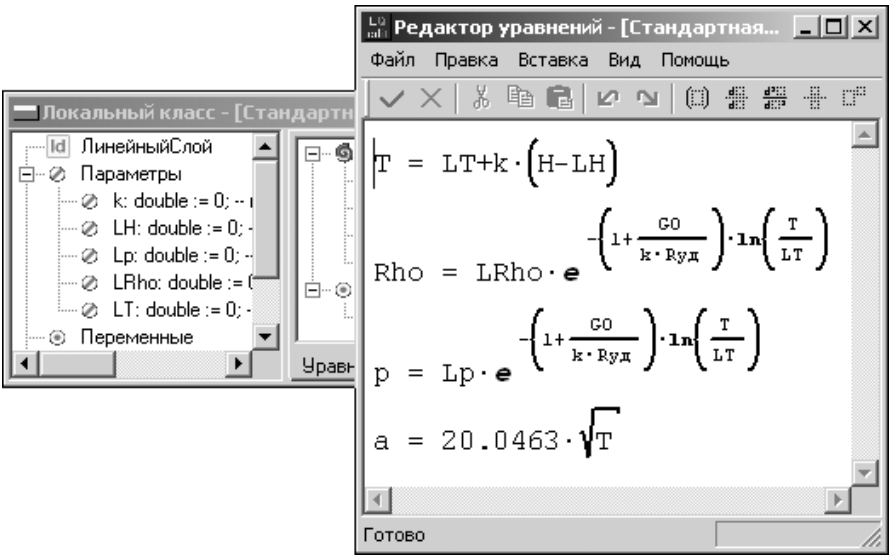


Рис. 1.32. Локальный класс `ЛинейныйСлой`

Эти классы имеют параметры: `LT`, `LRho`, `Lp`, `LH` — температура, плотность, давление и высота для нижней границы слоя соответственно. Класс `ЛинейныйСлой` дополнительно имеет параметр `k` — температурный градиент в данном слое.

Теперь в классе `СтандартнаяАтмосфера` введем несколько констант типа `vector[11]`: `Hmin`, `Hmax` — соответственно нижний и верхний пределы слоев; `K` — температурные градиенты слоев; `LT0`, `LRho0`, `Lp0` — соответственно значения температуры, плотности и давления на нижних границах слоев. Примем `Hmin[1]=-1000`, распространив, таким образом, первый слой на некоторый диапазон высот ниже уровня моря. Введем также целые константы `Nmin=1` и `Nmax=11`, а также целую переменную `N` — текущий номер слоя.

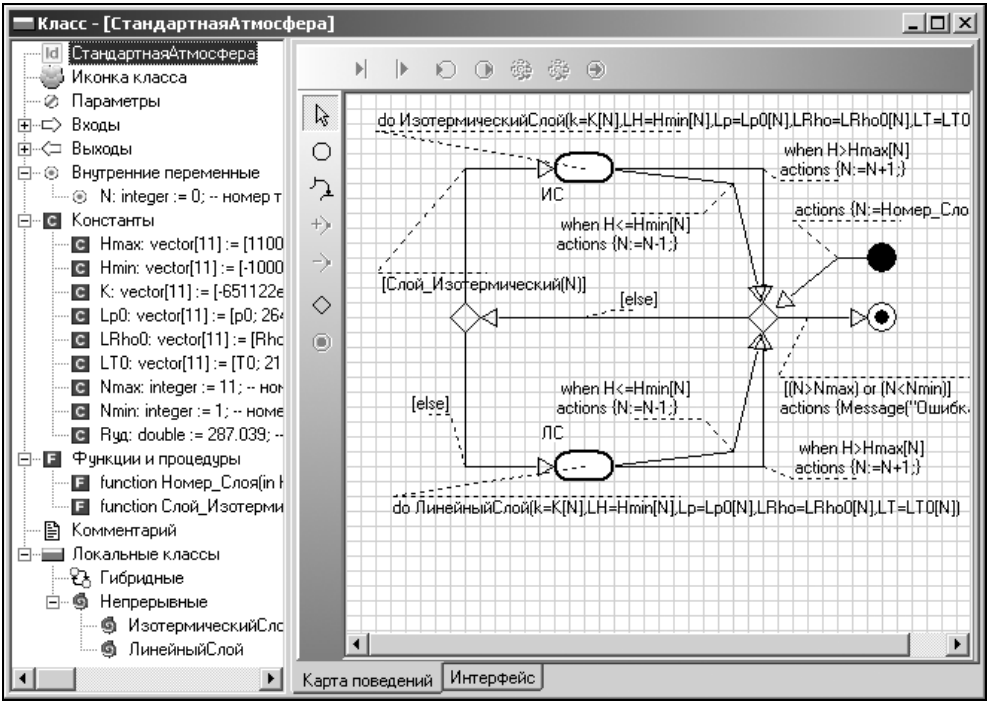


Рис. 1.33. Усовершенствованная модель стандартной атмосферы

На рис. 1.33 показана новая карта поведения для усовершенствованного класса `СтандартнаяАтмосфера` для 11 слоев.

В действиях перехода из начального состояния в первую точку ветвления определяется начальное значение номера слоя `N` для начального значения высоты `H` с помощью алгоритмической функции:

```
function Номер_Слоя(in H: double) return integer is
begin
  if H<Hmin[Nmin] then
    return 0;
  end if;
  for i in 1..Nmax loop
    if H<=Hmax[i] then
      return i;
    end if;
  end loop;
  return Nmax+1;
end Номер_Слоя;
```

Заметим, что использование алгоритмических операторов в последовательности мгновенных действий не влечет за собой никаких неприятностей для численного решения.

В первой точке ветвления выполняется анализ допустимости номера текущего слоя и в случае отрицательного результата выполняется переход на конечное состояние с выдачей сообщения об ошибке, а в случае положительного (условие `else`) выполняется переход на вторую точку ветвления. Во второй точке ветвления определяется вид текущего слоя с помощью алгоритмической функции

```
function Слой_Изотермический(in i: integer) return boolean is
  res: boolean;
begin
  case i of
    when 2!4!6 =>
      res:=true;
    when others =>
      res:=false;
  end case;
  return res;
end Слой_Изотермический;
```

В случае положительного результата выполняется переход в состояние ИС, а в случае отрицательного (условие else) — переход в состояние ЛС. Этим состояниям приписаны в качестве деятельностей экземпляры локальных классов ИзотермическийСлой и ЛинейныйСлой соответственно с действительными значениями их параметров, соответствующими текущему слою. Из состояний ИС и ЛС имеются исходящие триггерные переходы в первую точку ветвления, срабатывающие при выходе текущего значения высоты за пределы нижней или верхней границы текущего слоя. В перовом случае номер текущего слоя уменьшается на единицу, а во втором увеличивается на единицу. Таким образом, в состоянии ИС моделируются зависимости слоев 2, 4 и 6, а в состоянии ЛС — все остальные.

Снова используем отладочную модель атмосферы (см. рис. 1.19) для проверки работоспособности нашего усовершенствованного класса, изменив действительное значение параметра UpperLimit объекта RG на 200 000. Кроме того, поскольку высота при испытаниях линейно увеличивается от нуля, эту же модель можно использовать для последовательного вычисления значений векторов LT0, LRho0, Lp0 (в самом деле, не определять же их с помощью калькулятора!). Для этого внесем в класс СтандартнаяАтмосфера два отладочных изменения:

- сделаем векторы LT0, LRho0, Lp0 переменными, а не константами. Для этого переместим их методом drag-and-drop из секции **Константы** в окне редактора класса в секцию **Переменные**;
- в альтернативном переходе из первой точки ветвления во вторую введем следующую последовательность мгновенных действий

```
LT0[N] := T;
LRho0[N] := Rho;
Lp0[N] := p;
```

Создадим выполняемую модель и после проведения вычислительного эксперимента получим правдоподобные графики (рис. 1.34) и искомые значения векторов (рис. 1.35).

Теперь можно возвращаться к проекту **Ракета**. После открытия этого проекта импортируемый класс Земля.СтандартнаяАтмосфера автоматически обновится, и мы теперь можем моделировать траектории с высотой до 200 км. Вычислительные эксперименты показывают, что при различных сочетаниях угла наклона направляющей и секундного расхода дальность полета ракеты может достигать порядка 200 км. Представляется разумным для таких траекторий учесть кривизну Земли.

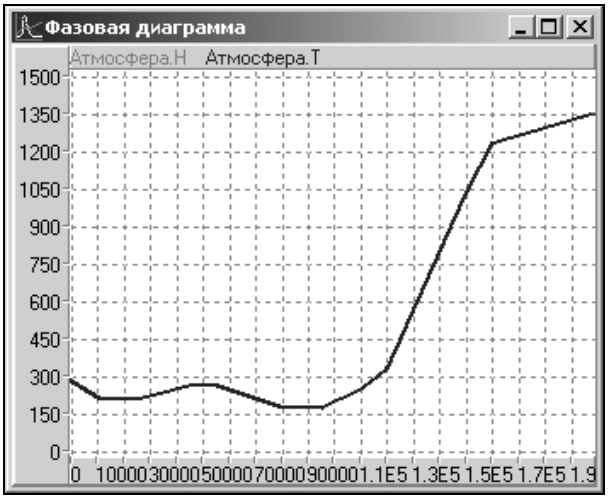
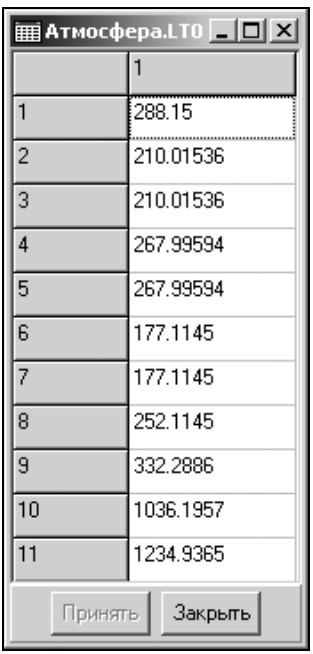


Рис. 1.34. Зависимость температуры от высоты для усовершенствованной модели атмосферы



	1
1	288.15
2	210.01536
3	210.01536
4	267.99594
5	267.99594
6	177.1145
7	177.1145
8	252.1145
9	332.2886
10	1036.1957
11	1234.9365

Рис. 1.35. Начальные значения температуры для слоев, полученные с помощью вычислительного эксперимента

Введем необходимые дополнительные переменные (см. модель снаряда) и модифицируем систему уравнений локального класса `Динамика_ракеты`. Вычислительный эксперимент показывает, что в отличие от снаряда для ракеты эта доработка оказывается существенной: разность между значением горизонтальной координаты точки падения x в стартовой системе координат и значением наклонной дальности D составляет несколько десятков метров.

Вроде бы теперь у нас все успешно летает, однако в созданной нами модели чувствуется некоторая неуклюжесть, ей не хватает некоторой стройности и изящества. Это, прежде всего, связано с тем, что в модели нарушен основной принцип построения "хороших" систем: какая-либо функциональность должна декларироваться один раз и в одном месте. В нашей же модели в двух состояниях дублируются уравнения, связанные с аэродинамикой, уравнения реактивного двигателя, внутренние переходы, фиксирующие выгорание топлива, внешние переходы, фиксирующие момент падения. Такое положение терпимо для нашей относительно простой модели, однако когда мы захотим ее усложнить (например, учесть влияние внешнего давления на силу тяги или ввести второй аэродинамический коэффициент C_y для расчета аэродинамической подъемной силы в зависимости от угла атаки) или обобщить (например, сделать параметрами аэродинамические характеристики), могут возникнуть ошибки.

Для соблюдения принципа "один раз в одном месте" необходимо выделить в моделируемой системе параллельно протекающие процессы и представить их параллельно функционирующими объектами. В модели ракеты такими процессами являются:

- движение ракеты как материальной точки;
- аэродинамика ракеты;
- динамика реактивного двигателя.

В зависимости от начальных условий и параметров возможны достаточно произвольные временные сочетания этих трех процессов. Целесообразно выделить среди этих процессов основной, который и сопоставить с собственным поведением модели. В нашем случае это, безусловно, будет движение ракеты. Другие процессы придется реализовывать в виде локальных объектов, т. к. параллельные карты поведения не допускаются согласно принципу синхронной композиции, изложенному в [5]. Согласно тому же принципу эти локальные объекты не могут быть экземплярами локальных классов (одновременная совместная работа с общими данными может приводить к противоречивым результатам). Следовательно, описание аэродинамики ракеты и

динамки реактивного двигателя нужно выносить в определения отдельных глобальных классов проекта. Эти классы должны иметь стереотип "открытая система", т. к. взаимодействовать с локальными объектами — экземплярами этих классов будет возможно только через внешние переменные.

Сохраним текущее состояние проекта в файле Ракета_2.mvb и начнем преобразования. Новую модификацию модели ракеты назовем модульной. Переименуем локальный класс `Динамика_ракеты` в `Движение_ракеты`, т. к. теперь этот класс будет определять только движение ракеты как материальной точки. Сначала выполним команду **Преобразовать карту поведения в локальный класс** всплывающего меню в редакторе карты поведения модели, чтобы использовать имеющуюся собственную карту поведения ракеты как заготовку для карты поведения нового локального гибридного класса `Динамика_ракеты`. После удаления ненужных состояний и переходов эта карта поведения будет выглядеть так, как показано на рис. 1.36.

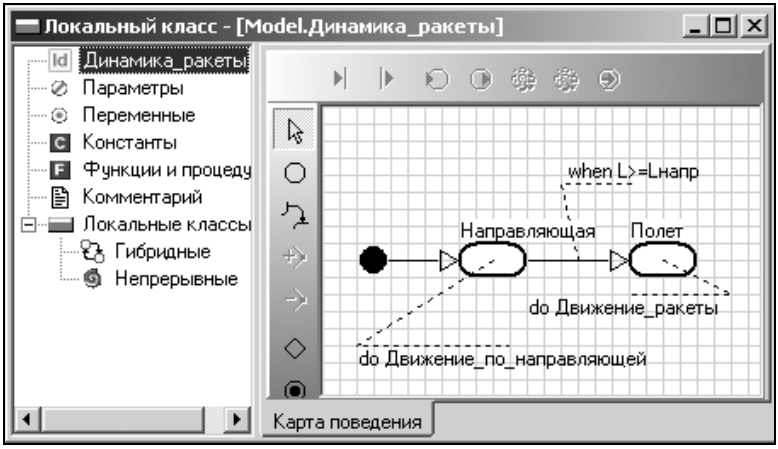


Рис. 1.36. Карта поведения локального класса `Динамика_ракеты`

Теперь модифицируем главную карту поведения модели. Она включает в себя начальное состояние, одно обычное состояние `Движение` и конечное состояние (рис. 1.37). Состоянию `Движение` припишем в качестве деятельности экземпляр локального класса `Динамика_ракеты`. Таким образом, состояние `Движение` становится гиперсостоянием, т. е. состоянием, деятельность которого содержит собственную карту поведения (заметим, что для карт поведения понятие гиперсостояния слегка отличается от аналогичного понятия UML).

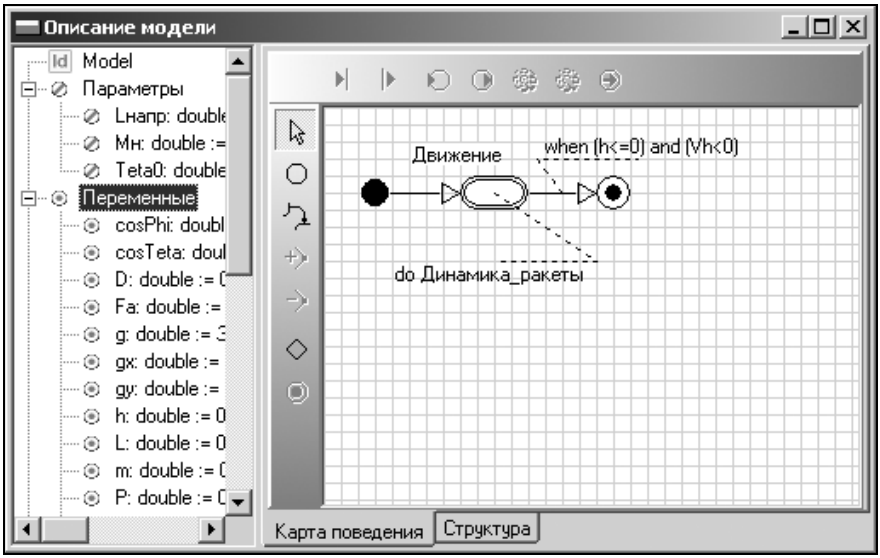


Рис. 1.37. Карта поведения модульной модели ракеты

Переход из состояния `Движение` в конечное состояние срабатывает в точке падения (рис. 1.37). Он срабатывает вне зависимости от того, в каком состоянии находится объект класса `Динамика_ракеты` — во время движения по направляющей или в свободном полете. Таким образом, использование гиперсостояний целесообразно для экономного описания реакций на дискретные события, общих для всех состояний вложенной карты поведения.

Создадим глобальный непрерывный класс `Аэродинамика`, в который перенесем все параметры, переменные и уравнения, относящиеся к аэродинамическим силам (рис. 1.38).

Этот класс должен быть открытой системой: извне должны задаваться скорость летательного аппарата, плотность воздуха и скорость звука, а на выходе должна формироваться сила аэродинамического сопротивления (рис. 1.39).

Далее создадим глобальный класс `РеактивныйДвигатель`, в который перенесем все параметры и переменные, связанные с работой реактивного двигателя. Однако поскольку в двигателе происходит обнуление тяги (дискретное событие), то этот класс должен быть гибридным.

Карта поведения этого класса (рис. 1.40) содержит одно обыкновенное состояние `Работа`, которому в качестве деятельности приписан экземпляр локального непрерывного класса `Работа_двигателя` (рис. 1.41).

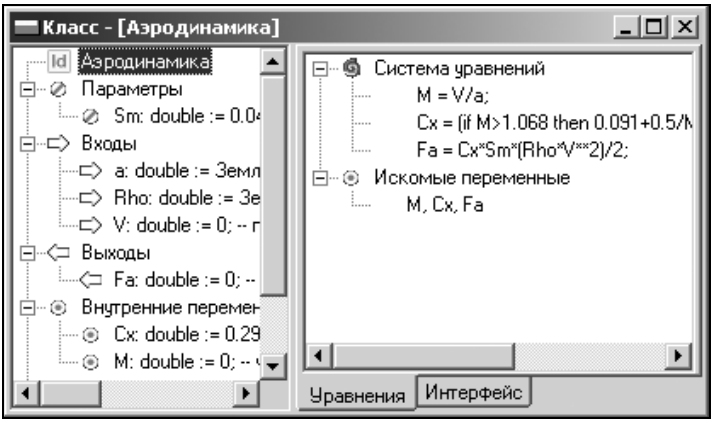


Рис. 1.38. Переменные и уравнения класса `Аэродинамика`

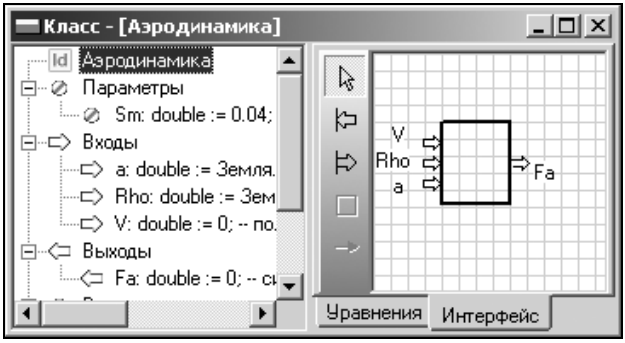


Рис. 1.39. Интерфейс класса `Аэродинамика`

Переход из состояния `Работа` в конечное состояние срабатывает при полном выгорании топлива. При срабатывании выполняются мгновенные действия, в результате которых обнуляется значение тяги и масса двигателя полагается равной массе конструкции двигателя (см. рис. 1.40). Обратите внимание, что в модульной модели эти действия выполняются только один раз и уравнения работы двигателя изымаются из глобальной системы уравнений модели, в то время как в предыдущей модели эти уравнения продолжали решаться с нулевым значением секундного расхода топлива (переменная `q`).

На вход объекта класса `РеактивныйДвигатель` подается внешнее атмосферное давление `pa` (в данной модели оно не используется, но может пригодиться

для уточненных уравнений силы тяги) и на выходе вычисляются значения силы тяги `P` и текущее массы двигателя `m` (рис. 1.42).

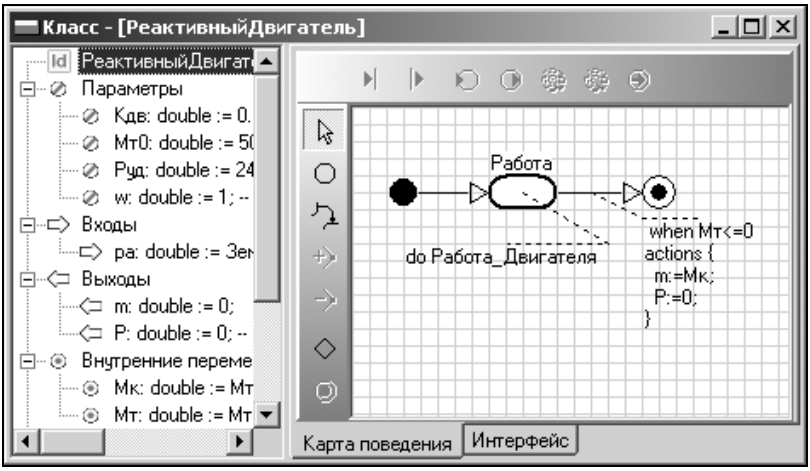


Рис. 1.40. Переменные и карта поведения класса `РеактивныйДвигатель`

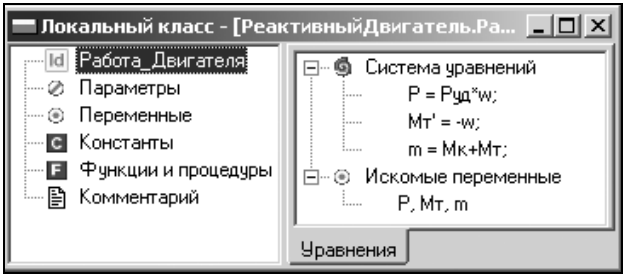


Рис. 1.41. Локальный класс `Работа_двигателя`

Определив необходимые классы, мы теперь можем поместить их экземпляры в качестве локальных объектов `Аэро` и `РДТТ` модели (рис. 1.43).

Часть входных данных — параметров воздуха для этих объектов можно подать непосредственно с выходов объекта `Atm` с помощью связей (рис. 1.43). Значение полной скорости на входе `Аэро.V` должно вычисляться в системе уравнений локальных классов `Движение_по_направляющей` (рис. 1.44) и `Движение_ракеты` (рис. 1.45).

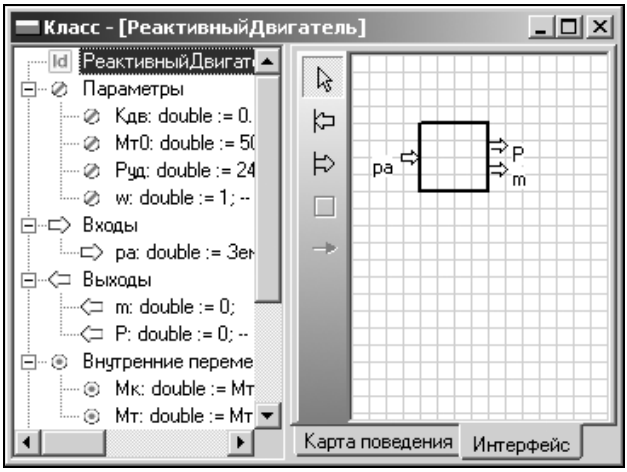


Рис. 1.42. Интерфейс класса `РеактивныйДвигатель`

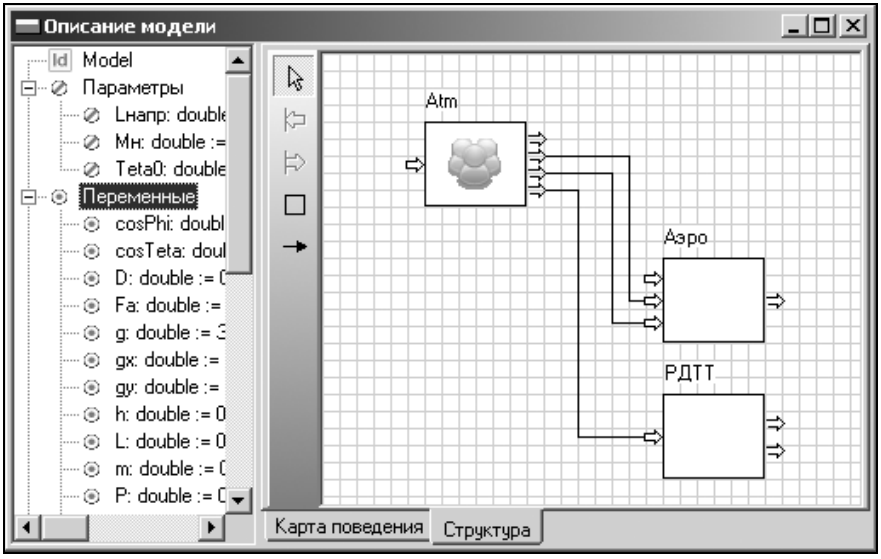


Рис. 1.43. Структура модульной модели ракеты

На этом создание модульной модели заканчивается, и можно приступать к вычислительным экспериментам. Прежде всего, нужно убедиться, что для нескольких сочетаний параметров ["начальный угол", "секундный расход"]

результаты совпадают с результатами предыдущей модели, если это не так, то следует найти и устранить ошибки.

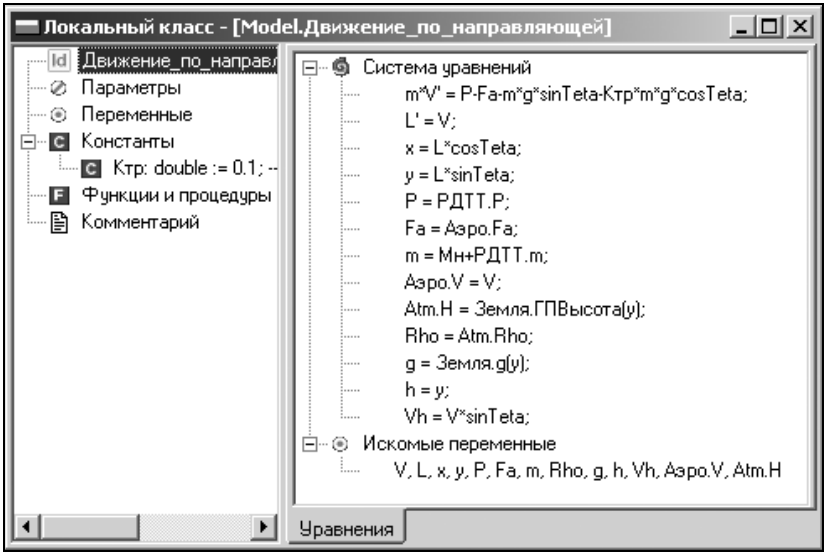


Рис. 1.44. Локальный класс `Движение_по_направляющей`

Результаты вычислительных экспериментов показывают, что дальность полета не очевидным образом зависит от сочетаний параметров ["начальный угол", "секундный расход"]. Возникает естественное желание узнать, какова предельная дальность полета для ракеты с заданными полезной нагрузкой и массой топлива? Выполним команду **Сервис | Оптимизация** главного меню интегрированной среды и перейдем в окно подсистемы оптимизации.

Для проведения оптимизации формируется "скрытая" модель без какой-либо визуализации в виде динамической библиотеки. Подсистема оптимизации взаимодействует с ней по специальному управляющему интерфейсу. Заметим, что скрытая модель может быть использована и для подключения к любой другой прикладной подсистеме пользователя (для создания скрытой модели необходимо выбрать соответствующий тип выполняемой модели на вкладке **Модель** в установках проекта).

Слева в окне подсистемы оптимизации находится дерево переменных модели. Специальным значком указаны свободные переменные, начальные значения которых могут быть использованы в качестве параметров оптимизации.

В нашей задаче такими параметрами являются `Teta0` и `РДТТ.q`. Для того чтобы задать список параметров оптимизации, необходимо с помощью мыши методом drag-and-drop перетащить соответствующие переменные из дерева переменных в поле **Параметры**. (Далее нам нужно указать функционал — величину, которую мы хотим максимизировать или минимизировать. В нашей задаче это значение наклонной дальности D. Соответствующее выражение нужно ввести в текстовом поле **Функционал**. Если функционал, как в нашем случае, представляет собой просто указание переменной, то достаточно перетащить мышью эту переменную в поле **Функционал**.

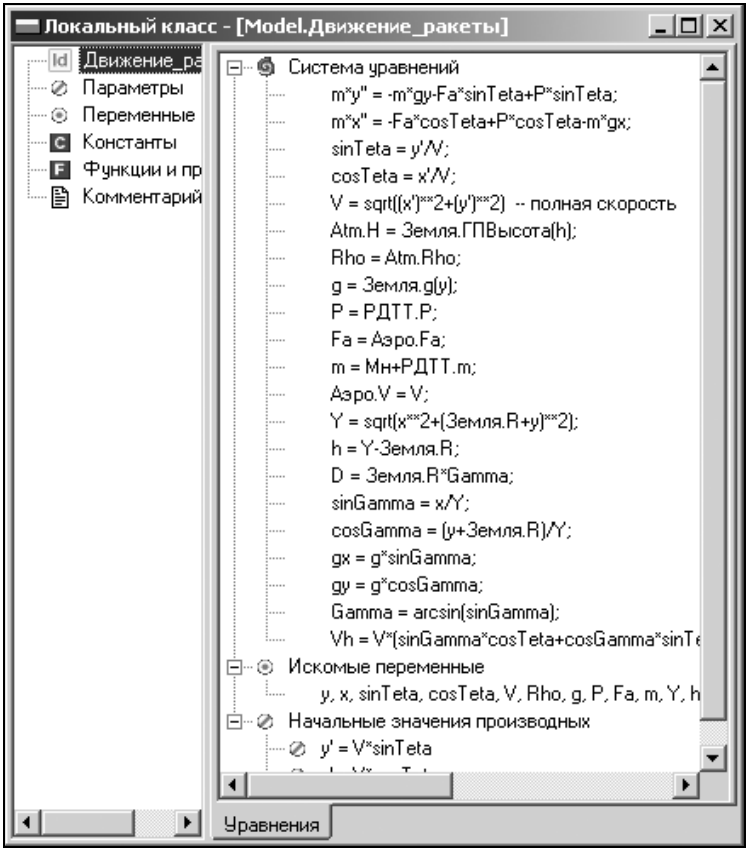
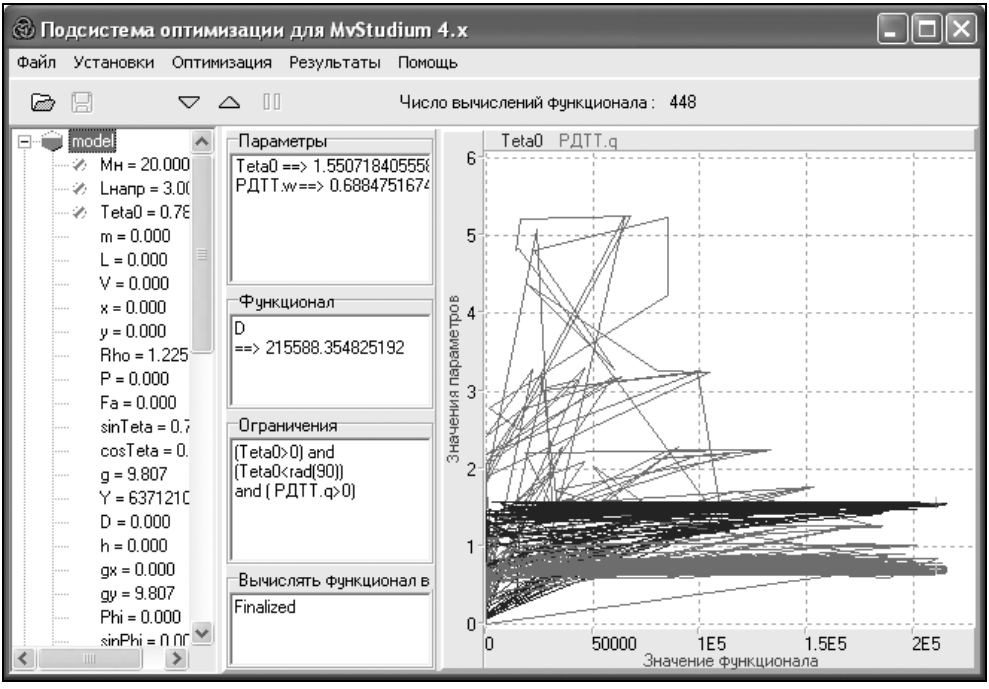


Рис. 1.45. Локальный класс `Движение_ракеты`

Далее нужно указать ограничения на параметры оптимизации, если они существуют. В нашей задаче необходимо указать условие положительности

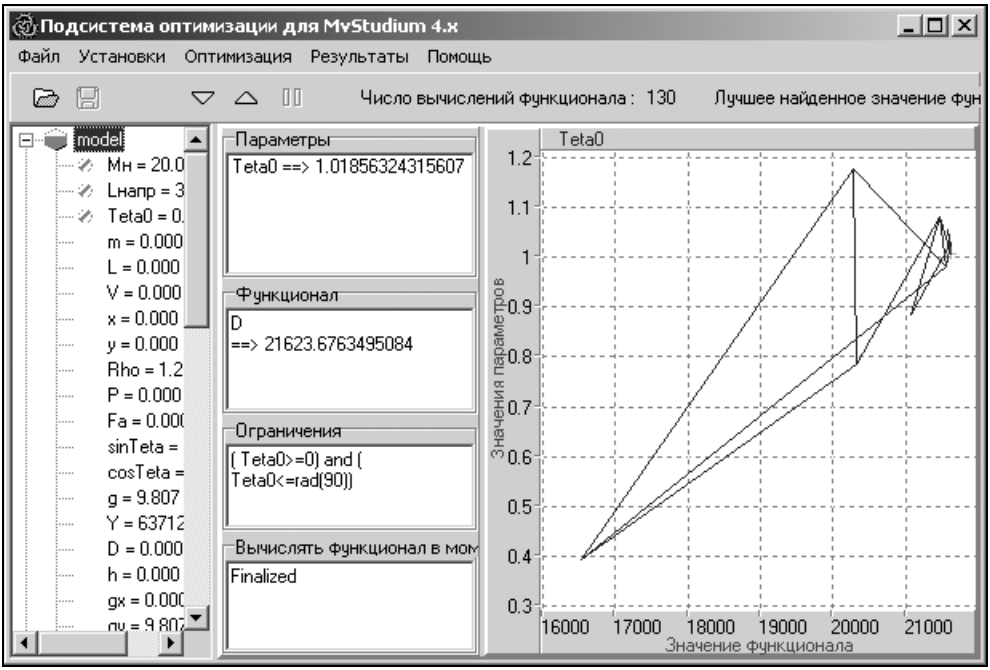
значения секундного расхода топлива и разумно ограничить область поиска начального угла первой четвертью.)

Наконец, нам нужно указать момент модельного времени, в который следует вычислять функционал. Выбранный метод оптимизации согласно своему алгоритму (случайный поиск, градиентный поиск и т. п.) будет изменять значение параметров оптимизации и для этой совокупности значений вычислять значение функционала, и так много раз. В рассматриваемом случае нас интересует не просто наклонная дальность, а дальность в точке падения. Для определения этой величины необходимо для каждой совокупности параметров провести моделирование до точки падения. Точка падения в нашей модели определяется переходом в конечное состояние карты поведения модели. Поэтому в качестве условия, задающего момент вычисления функционала, достаточно указать функцию `Finalized`, которая возвращает `true`, если все процессы в модели завершились.



а

Рис. 1.46. Определение максимальной дальности полета ракеты:
а — по совокупности угла старта и секундного расхода



б

Рис. 1.46. Определение максимальной дальности полета ракеты:
б — по углу старта для значения секундного расхода 5000 кг/с

Заметим, что для чисто непрерывной модели снаряда было бы необходимо указать выражение `(y<=0) and (y'<0)`. С помощью команды **Оптимизация | Найти максимум** или кнопки на инструментальной панели запустим процесс оптимизации. В правой части инструментальной панели начнет изменяться количество вычислений функционала. На фазовой диаграмме в правой части (рис. 1.46) окна будет изображаться траектория поиска (по оси абсцисс диаграммы откладывается значение функционала, а по оси ординат — значения искомых параметров оптимизации). Для нашей задачи после 511 вычислений функционала будет найдено оптимальное значение параметров: 1.546 для угла (88.58°) и 0.713 кг/с для секундного расхода топлива (рис. 1.46, *а*).

На первый взгляд этот результат кажется странным, поэтому проверим его прямым моделированием. Вычислительный эксперимент с указанными значениями параметров, действительно, дает значение дальности 215.4 км (рис. 1.47). Обратите внимание, что в точке падения значение координаты *y*

в стартовой системе координат отрицательно (–3.8 км), что обусловлено кривизной Земли.

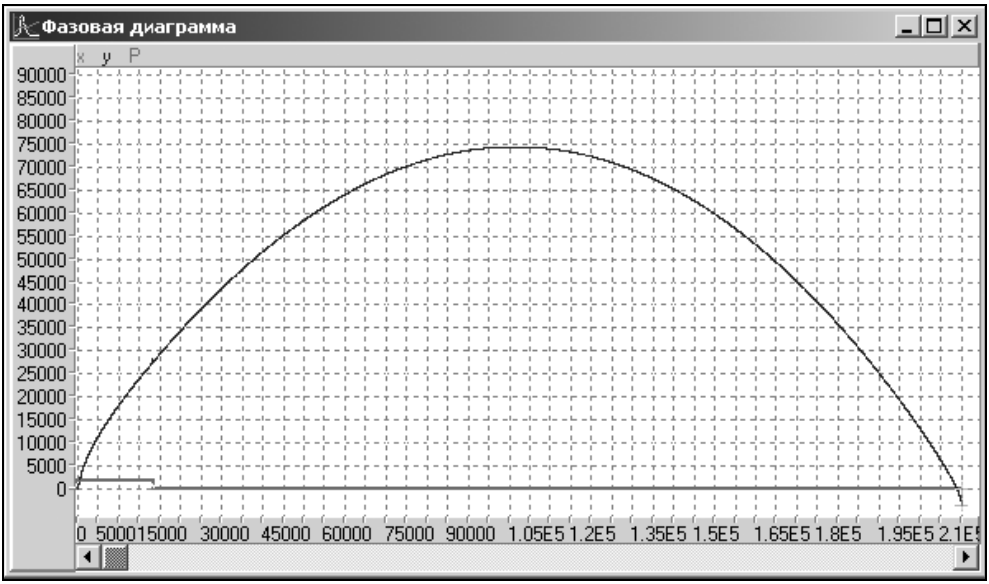


Рис. 1.47. Траектория максимальной дальности

Найдем оптимальное значение угла старта для секундного расхода 5000 кг/с (рис. 1.46, *б*). Оптимальным оказывается угол старта 57.9°, при этом дальность оказывается в десять раз меньшей, чем для траектории максимальной дальности.

Предусмотрим отладочный останов по срабатыванию перехода из состояния `Работа` в конечное состояние в карте поведения объекта `РДТТ` и посмотрим на значения переменных в момент обнуления тяги. Для удобства создадим виртуальную переменную `Teta`, соответствующую текущему значению угла наклона траектории в градусах. Оказывается, что обнуление тяги происходит на высоте 27.8 км при угле наклона траектории 46.5°. Таким образом, ракета переходит на пассивный участок достаточно высоко, при низкой плотности воздуха и с углом наклона траектории, близким к углу максимальной дальности для безвоздушного пространства. Таким образом, на траектории максимальной дальности используется сложное сочетание воздействия трех факторов: гравитационного поля, силы тяги и зависимости плотности воздуха от высоты. Стартуя под таким, казалось бы, странным углом и с небольшим се-

кундным расходом топлива, мы проходим плотные слои атмосферы на небольшой скорости, давая возможность полю тяготения совершить так называемый "гравитационный разворот" вектора скорости ракеты на оптимальный угол. При меньших расходах топлива разворот будет слишком большим (см. рис. 1.29, *б*), а при больших расходах траектория будет близка к вертикальной.

Однако радоваться особо нечему. Вычислительные эксперименты показывают, что при начальном угле 88.0° дальность полета составляет всего 173.4 км, а при 89.0° — 194.7 км (заметим, что проведение статистических экспериментов есть особая тема, и она будет рассмотрена далее). Таким образом, ошибка в полградуса дает разброс точки падения в 40 км, и такая ракета никому не нужна. Поэтому оставим такие совершенные траектории для космической баллистики, где нет сопротивления воздуха.

Однако если из-за "острой" зависимости нельзя использовать гравитационное поле для формирования нужной траектории, то нельзя ли формировать ее с помощью управления? Кроме того, нельзя ли управлять дальностью полета какими-то иными способами кроме изменения угла наклона направляющей? Сохраним текущее состояние проекта как Ракета_3.mvb и продолжим работу над моделью.

Создание и отладка структурной модели

В рассмотренных ранее моделях мы уже использовали локальные объекты и связи, однако это делалось, скорее, для удобства моделирования. Теперь мы рассмотрим модель системы, в которой имеется естественная структура, которую важно отразить в модели. Это модель системы управления ракетой. До сих пор мы рассматривали саму ракету как исследуемый объект и потому помещали ее описание в класс `Model`. Теперь же ракета как физический объект (пусть сложный, со своей внутренней структурой) будет рассматриваться лишь как один из компонентов системы управления, а именно в качестве объекта управления.

Поэтому первое, что мы сделаем, это преобразуем класс `Model` в глобальный класс `УправляемаяРакета`. Для этого выполним команду **Модель | Преобразовать описание модели в класс** главного меню или нажмем кнопку на инструментальной панели. После выполнения преобразования описание модели соответствует начальному состоянию класса соответствующего типа по умолчанию. Заметим, что если мы захотим провести вычислительные экспе-

рименты с экземплярами отдельных глобальных классов, то это можно сделать двумя способами:

- сделать модель составным объектом и поместить в нее в качестве локального объекта экземпляр интересующего нас класса;
- копировать класс в описание модели с помощью команды **Модель | Копировать класс в описание модели** или кнопки на инструментальной панели.

Преобразуем "обновленный" класс `Model` в составной — на вкладке **Структура** описания модели мы и будем собирать нашу систему управления.

Поскольку мы не ставим себе целью освещать здесь методы управления ракетами, используем самые простые методы, позволяющие, тем не менее, проиллюстрировать особенности компонентного моделирования. Интересующиеся читатели могут обратиться, например, к [8, 9]. Попробуем различными способами заставить ракету двигаться по некоторой заданной программной траектории

Сначала нам придется модифицировать модель ракеты, т. к. поведение управляемой ракеты отличается от поведения неуправляемой. Прежде всего, преобразуем также класс `УправляемаяРакета` в открытую систему, поскольку управлять изолированной системой невозможно.

Для реализации операции отсечки тяги предусмотрим существование специального механизма, который по специальной разовой команде обнуляет тягу `РДТТ`. Для этого в классе `РеактивныйДвигатель` введем вход `ОтсечкаТяги` типа `signal` и модифицируем карту поведения (рис. 1.48).

Теперь двигатель прекращает работу не только при выгорании запаса топлива, но и при поступлении извне соответствующей команды (общие действия вынесены в выходные действия состояния `Работа`).

На рис. 1.49 показаны силы, действующие на цент масс ракеты в полете. При описании движения ракеты используются три системы координат:

- стартовая система координат (x, y), в которой ось ординат направлена вертикально в точке старта (см. рис. 1.24);
- связанная система координат (X, Y), в которой ось абсцисс направлена по продольной оси ракеты;
- скоростная система координат (X_a, Y_a), в которой ось абсцисс направлена по вектору скорости центра масс ракеты. Для статически устойчивой ракеты скоростная и связанная системы координат совпадают, если пренебречь угловыми колебаниями ракеты.

Рис. 1.48. Модифицированный класс `РеактивныйДвигатель`

Для управления углом наклона траектории мы должны создавать управляющую силу, перпендикулярную вектору скорости ракеты. Для этого будем отклонять продольную ось ракеты от направления вектора скорости на угол атаки α. При движении с углом атаки появляется аэродинамическая подъемная сила F_{ay}, а также нормальная составляющая силы тяги $P \cdot \sin(\alpha)$ (рис. 1.49).

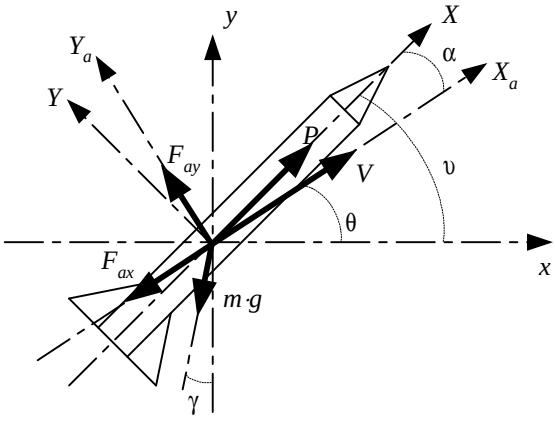


Рис. 1.49. Схема действия сил на центр масс ракеты

Приближенно можно считать, что аэродинамическая сила лобового сопротивления F_{ax} не зависит от угла атаки, а аэродинамическая подъемная сила F_{ay} зависит от угла атаки линейно:

$$F_{ay} = S_M \cdot \frac{\rho \cdot V^2}{2} \cdot C_y^{\alpha} \cdot \alpha,$$

где C_y^{α} — аэродинамический коэффициент подъемной силы, определяемый формой ракеты и зависящий от числа Маха. Для нашей ракеты примем зависимость

$$C_y^{\alpha} = \begin{cases} 2.8 \quad \text{при } M \le 0.25, \\ 2.8 + 0.447 \cdot (M - 0.35) \quad \text{при } 0.25 < M \le 1.1, \\ 3.18 - 0.660 \cdot (M - 1.1) \quad \text{при } 1.1 < M \le 1.6, \\ 2.85 + 0.350 \cdot (M - 1.6) \quad \text{при } 1.6 < M \le 3.6, \\ 3.55 \quad \text{при } M > 3.6. \end{cases}$$

С целью получения более выразительных зависимостей будем полагать, что наша гипотетическая ракета обладает хорошим аэродинамическим качеством и допускает очень большие углы атаки до 20° без срыва потока (для реальных баллистических ракет угол атаки не должен превышать 10—12°).

Для упрощения нашей модели будем считать, что наша `УправляемаяРакета` является "интеллектуальной" и включает в себя встроенный автомат угловой стабилизации, который работает идеально и всегда обеспечивает условие

$$\vartheta = \vartheta_{\text{пр}},$$

где $\vartheta_{\text{пр}}$ — программное значение угла тангажа. Таким образом, создаваемая нами система управления ракетой превращается в систему управления движением центра масс ракеты. Из схемы на рис. 1.49 следует, что при движении в плоскости стрельбы $\vartheta = \theta + \alpha$. Следовательно, системе управления центром масс ракеты достаточно подавать на вход системы угловой стабилизации нужное значение программного угла тангажа $\vartheta_{\text{пр}}$ и получать, таким образом, требуемый угол атаки α.

Для моделирования движения с углом атаки необходимо модифицировать класс `Аэродинамика` (рис. 1.50): ввести новый вход `alpha` и новый выход `Fay`, а также внутреннюю переменную `Cy` и соответствующим образом модифицировать систему уравнений, добавив формулы для вычисления C_y^{α} и F_{ay}.

Рис. 1.50. Модифицированный класс `Аэродинамика`

Теперь займемся модификацией самого класса `УправляемаяРакета` (рис. 1.51). Прежде всего, нужно добавить вход программного значения угла тангажа `teta_пр`, а также внутренние переменные `teta`, `Teta` и `alpha`, соответствующие значениям углов ϑ, θ и α в радианах. Переименуем переменную `Fa` в `Fax` и добавим новую внутреннюю переменную `Fay`.

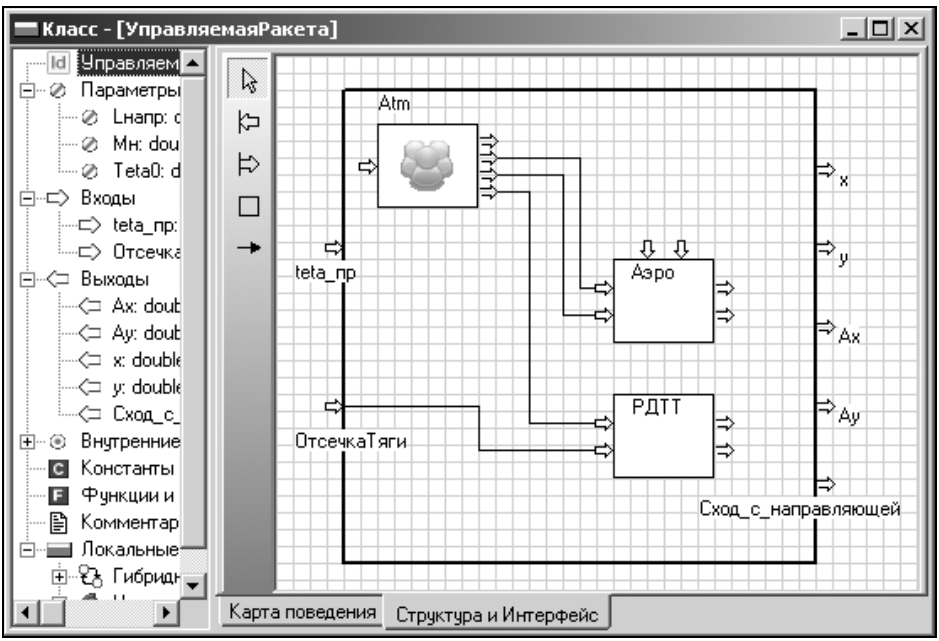


Рис. 1.51. Модифицированный класс `УправляемаяРакета`

Далее, для отработки команды отсечки тяги должен быть предусмотрен вход `ОтсечкаТяги` типа `signal`. Этот вход и вход `ОтсечкаТяги` локального объекта `РДТТ` должны быть связаны локальной связью (рис. 1.51). Переменную `D`, соответствующую дальности полета, для удобства работы с компонентом целесообразно преобразовать из внутренней в выходную. Будем считать, что наша система управления центром масс является автономной. Тогда для определения положения ракеты нам потребуется измерять значения ускорений по осям стартовой системы координат. Поэтому введем выходы `Ax` и `Ay`, соответствующие ускорениям в стартовой системе координат.

Наконец, управлять движением центра масс можно только после схода ракеты с направляющей. Поэтому необходимо ввести выходную переменную `Сход_с_направлящей` типа `signal` и для посылки этого донесения в момент схода с направляющей модифицировать локальный класс `Динамика_ракеты` (рис. 1.52).

Далее необходимо модифицировать систему уравнений локального класса `Движение_ракеты`. Необходимо добавить новые уравнения

$$\begin{cases} \vartheta = \vartheta_{\text{пр}}, \\ \theta = \arcsin(\sin Teta), \\ \alpha = \vartheta - \theta, \\ A_x = \dfrac{d^2x}{dt^2}, \\ A_y = \dfrac{d^2y}{dt^2}, \\ Аэро.\alpha = \alpha, \\ F_{ay} = Аэро.F_{ay}, \end{cases}$$

а также модифицировать два основных дифференциальных уравнения динамики центра масс ракеты с учетом наличия угла атаки:

$$\begin{cases} m \cdot \dfrac{d^2y}{dt^2} = -m \cdot g_y + P \cdot \sin(\vartheta) - F_{ax} \cdot \sin Teta + F_{ay} \cdot \cos Teta, \\ m \cdot \dfrac{d^2x}{dt^2} = -m \cdot g_x + P \cdot \cos(\vartheta) - F_{ax} \cdot \cos Teta - F_{ay} \cdot \sin Teta. \end{cases}$$

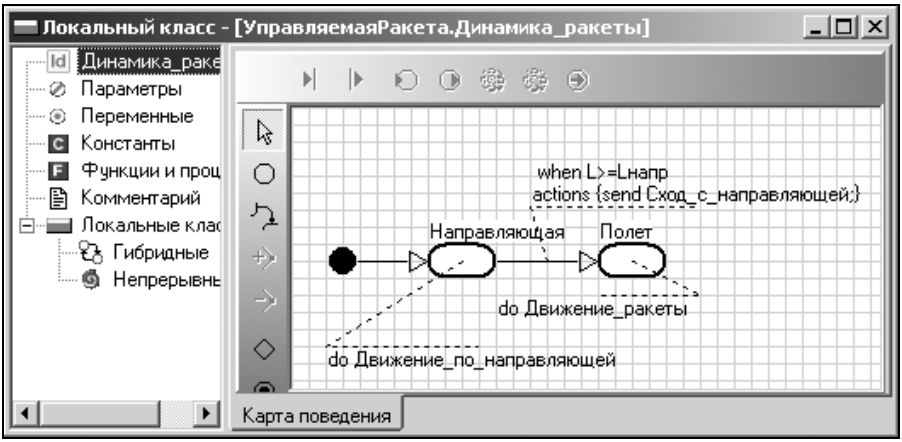


Рис. 1.52. Модифицированный локальный класс `Динамика_ракеты`

Теперь нужно провести контрольный вычислительный эксперимент, подтверждающий, что после стольких изменений наша ракета все еще летает. Проверим, как будет двигаться управляемая ракета с постоянным программным значением угла тангажа и работает ли команда отсечки тяги.

На структурную схему модели поместим первый локальный объект `Ракета` — экземпляр класса `УправляемаяРакета` (рис. 1.53).

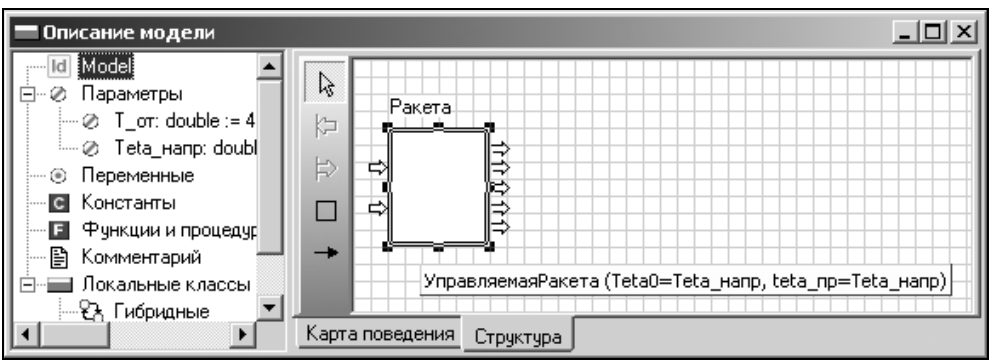


Рис. 1.53. Структура модели
в контрольном вычислительном эксперименте

Положим значение параметра `Teta0` и начальное значение входной переменной `teta_пр` локального объекта `Ракета` равными параметру `Teta_напр` клас-

са `Model` (рис. 1.53). Таким образом, мы попытаемся заставить ракету двигаться по прямой, как бы продолжая движение по направляющей. Для проверки отсечки тяги предусмотрим в карте поведения модели выдачу на вход объекта `Ракета` соответствующего сигнала через `T_от` секунд после старта (рис. 1.54). Положим значение параметра `T_от` равным 40 секундам при времени полного выгорания топлива 50 секунд.

Рис. 1.54. Карта поведения модели
в контрольном вычислительном эксперименте

Проведем вычислительный эксперимент и получим зависимости, показанные на рис. 1.55—1.57. Для получения графика нормальной составляющей силы тяги создадим соответствующую виртуальную переменную `Pyv`.

После схода ракеты с направляющей на еще небольшой скорости она "проседает" под действием силы тяжести, но затем в результате действия подъемной силы и нормальной составляющей силы тяги угол наклона траектории начинает приближаться к углу тангажа, и ракета действительно начинает двигаться практически по прямой линии (рис. 1.57). После отсечки тяги на 40-ой секунде скорость и, соответственно, подъемная сила начинают резко падать (рис. 1.56) и угол наклона траектории уменьшается (рис. 1.55). Далее ракета начинает двигаться с очень большими углами атаки и где-то после 125-ой секунды полета начинается "парашютирующий" спуск. Такое поведение вполне правдоподобно, однако количественные результаты заслуживают доверия только примерно до 80-ой секунды, пока значение угла атаки не превысило предельно допустимого. При больших углах атаки происходит срыв потока, и принятые в модели соотношения для аэродинамической подъемной силы уже не выполняются.

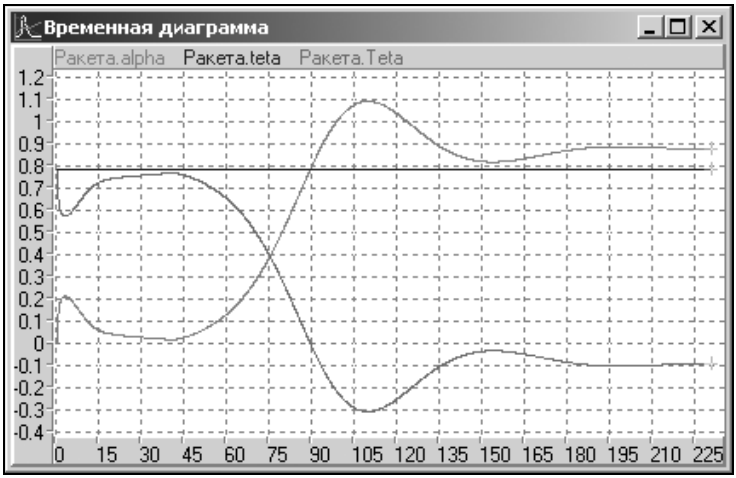


Рис. 1.55. Зависимости углов (сверху вниз) тангажа, наклона траектории и атаки от времени в контрольном эксперименте

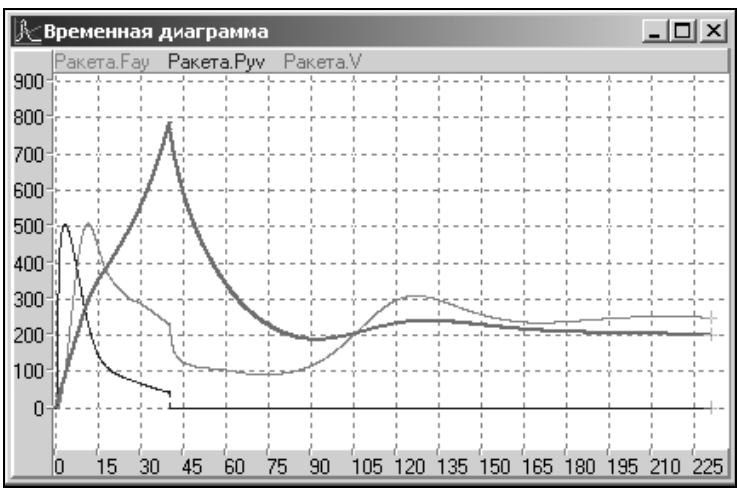


Рис. 1.56. Зависимости нормальной составляющей силы тяги, аэродинамической подъемной силы и скорости ракеты от времени в контрольном эксперименте

Поэтому имеет смысл ввести в модель ракеты контроль абсолютной величины угла атаки и при превышении максимально допустимого значения (например, 20°) выдавать сообщение об ошибке и останавливать эксперимент (рис. 1.58). Кроме того, для удобства планирования эксперимента нужно вве-

сти выходной сигнал `Падение` и выдавать его в момент падения на поверхность Земли (см. рис. 1.58).

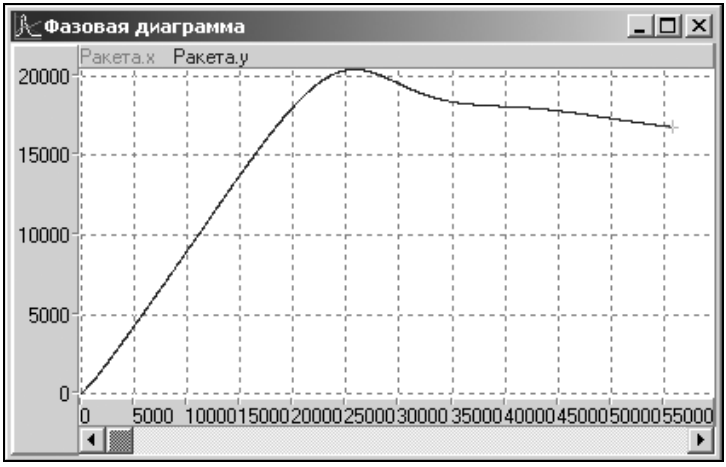


Рис. 1.57. Траектория ракеты в контрольном эксперименте

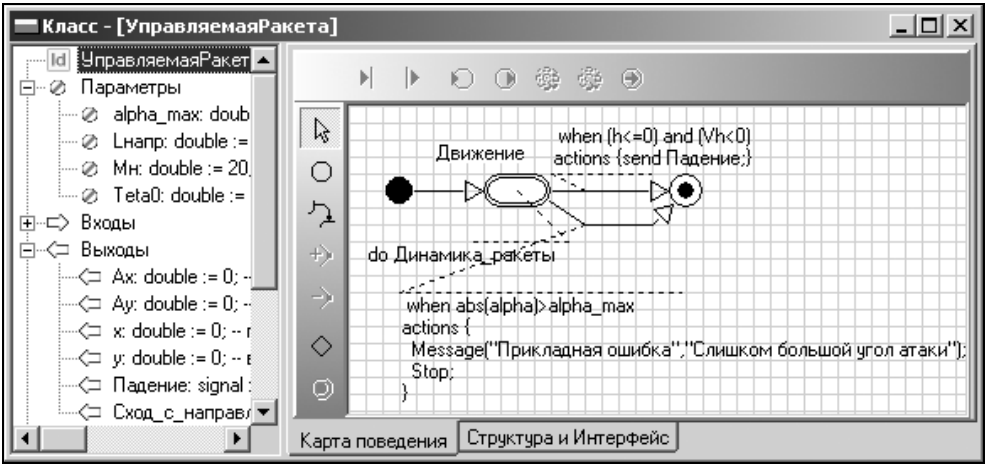


Рис. 1.58. Контроль величины угла атаки и выдача сообщения в точке падения

Переход к нештатному режиму функционирования оказался возможен из-за продолжения отработки системой угловой стабилизации ракеты программного значения угла тангажа после обнуления тяги. Для того чтобы иметь воз-

можность включать и отключать угловую стабилизацию ракеты, введем управляющий вход `Угловая_стаблизация` типа `boolean` и заменим уравнение, имитирующее работу системы стабилизации в локальном классе `Движение_свободное`, на следующее:

$$\vartheta = \texttt{if Угловая_стаблизация then } \vartheta_{\text{пр}} \texttt{ else } \theta$$

То есть мы предполагаем, что ракета является статически устойчивой и после отключения системы угловой стабилизации автоматически развернется по вектору скорости. Соответственно модифицируем карту поведения модели (рис. 1.59).

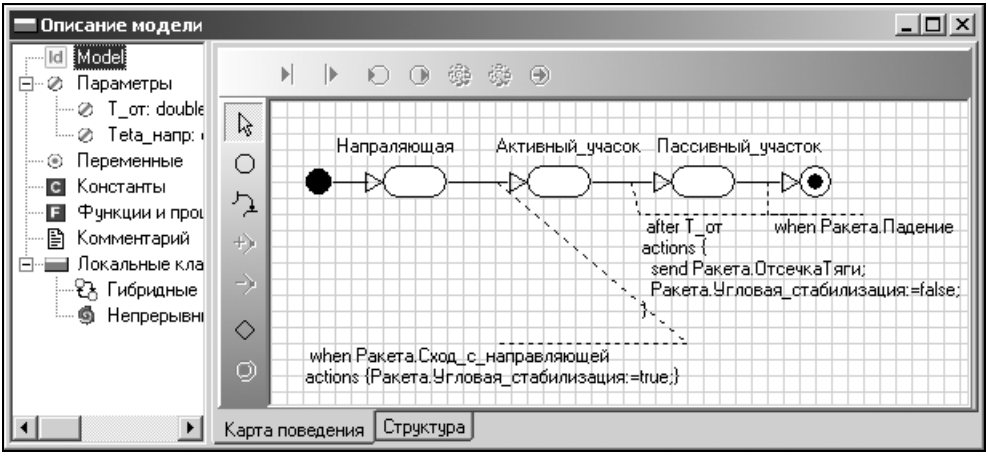


Рис. 1.59. Карта поведения модели в контрольном эксперименте с отключением системы стабилизации

После всех этих доработок снова проводим вычислительный эксперимент и получаем обычную баллистическую траекторию (рис. 1.60).

У внимательного читателя может возникнуть вопрос: почему команду `Угловая_стабилизация` мы сделали типа `boolean`, а донесение `Падение` — типа `signal`? В принципе, выход `Падение` можно сделать и типа `boolean`. Однако в этом случае условие перехода в карте поведения модели (см. рис. 1.59) вычислялось бы на каждом шаге интегрирования, в то время как переходы, ожидающие сигнала, инициируются только при его появлении. В то же время логическую переменную `Угловая_стабилизация` очень удобно использовать в условном уравнении.

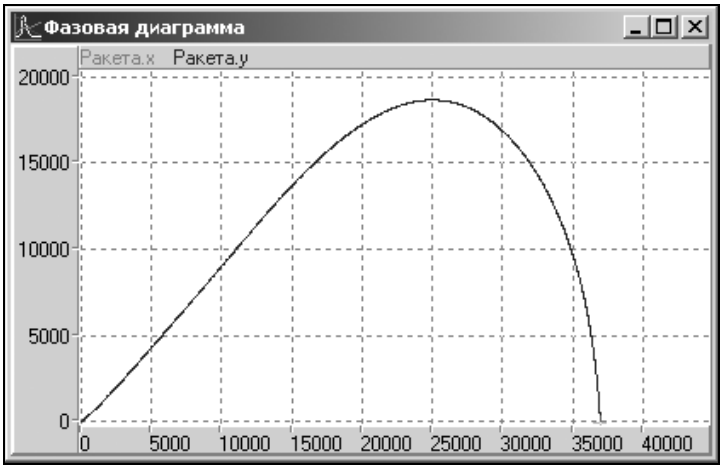


Рис. 1.60. Траектория ракеты в контрольном эксперименте с отключением системы стабилизации

На этом контрольные эксперименты можно закончить и перейти к созданию системы управления. Проект в данной точке разработки сохранен как Ракета_4.mvb.

Проведя несколько вычислительных экспериментов, можно установить, что при обнулении тяги на 45-ой секунде мы получаем дальность полета 71 км, а на 50-ой — 138 км. Следовательно, меняя время отсечки тяги, мы можем управлять дальностью полета при неизменном угле наклона направляющей. Таким образом, мы получаем первый вариант системы управления — жесткое управление моментом отсечки тяги по временной программе без обратной связи. Карта поведения на рис. 1.59, по существу, является алгоритмом работы устройства управления.

Преобразуем класс `Model` в класс `Устройство_управления`. Преобразуем класс в открытую систему. Введем входы и выходы, соответствующие используемым выходам и входам объекта `Ракета`. Модифицируем карту поведения, как показано на рис. 1.61. Далее удалим локальный объект `Ракета` и преобразуем класс в элементарную систему.

Преобразуем класс `Model` в составной объект и соберем на вкладке **Структура** функциональную схему, показанную на рис. 1.62, поместив на нее экземпляры классов `УправляемаяРакета` и `Устройство_управления` и соединив их соответствующие входы и выходы связями.

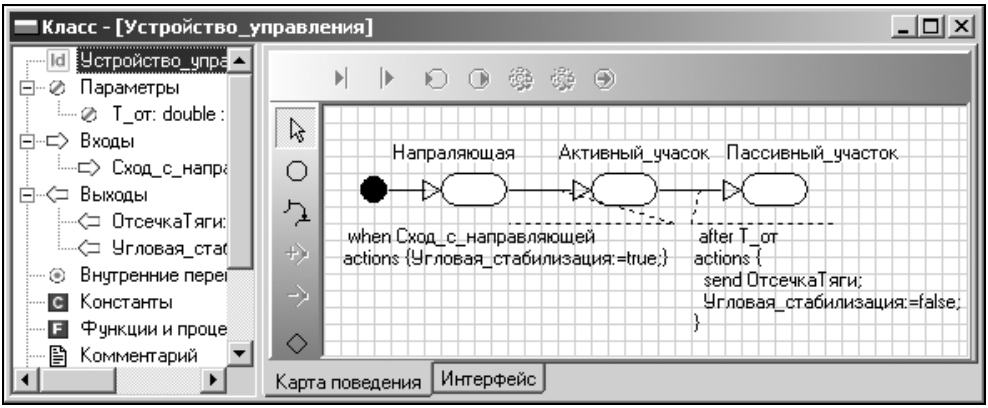


Рис. 1.61. Устройство управления моментом отсечки тяги

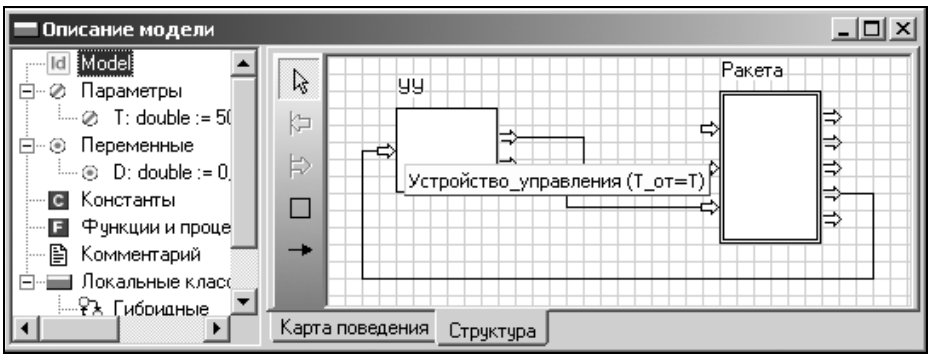


Рис. 1.62. Структура первого варианта системы управления ракетой

С помощью карты поведения модели создадим простейший план вычислительного эксперимента — моделирование до падения ракеты (рис. 1.63).

Проведя вычислительный эксперимент с этой моделью, убеждаемся, что она работает. Таким образом, установив нужное значение параметра `УУ.T_от` (полетное задание), мы должны попасть в требуемую точку. Для того чтобы подготовить такое полетное задание, нужно иметь зависимость дальности полета от времени отсечки тяги. Получить такую зависимость аналитически невозможно, поэтому получим ее с помощью специального вычислительного эксперимента. Сохраним текущее состояние проекта как Ракета_5.mvb.

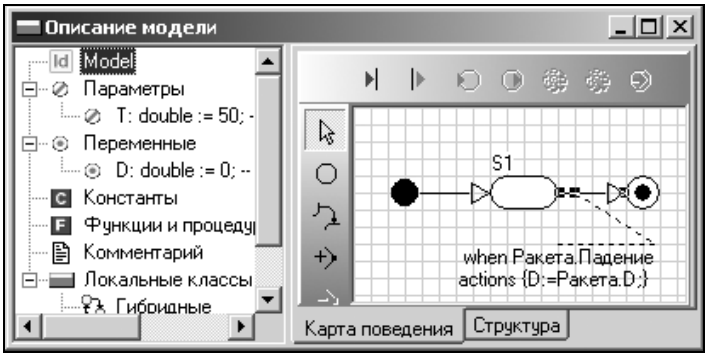


Рис. 1.63. План вычислительного эксперимента с моделью первого варианта системы управления

Преобразуем модель в класс `СУ_т`. План вычислительного эксперимента по определению зависимости дальности полета от времени отсечки тяги зададим с помощью карты поведения модели, показанной на рис. 1.64.

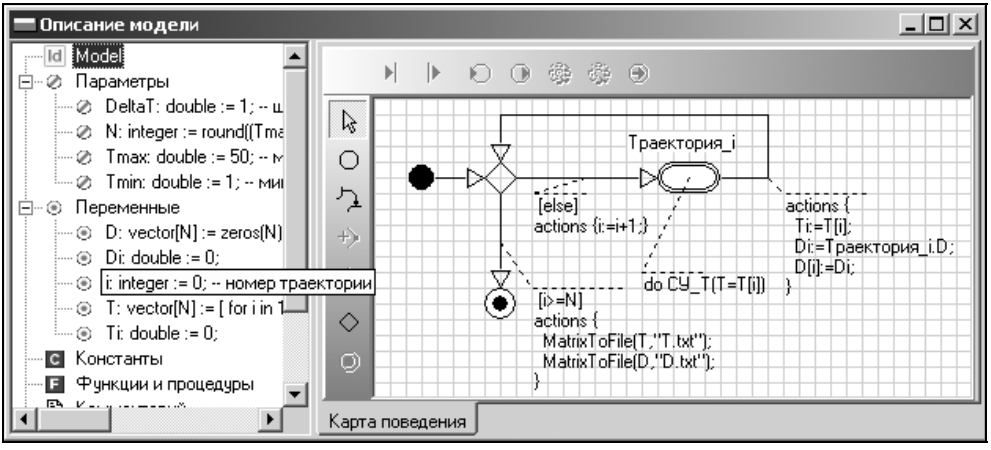


Рис. 1.64. План вычислительного эксперимента по определению зависимости дальности полета от времени отсечки тяги

Эксперимент предполагает моделирование траекторий для последовательности моментов отсечки тяги от `Tmin` до `Tmax` с шагом `DeltaT`. Конкретные значения времени отсечки тяги на `i`-ой траектории и соответствующие им зна-

чения дальности полета запоминаются в векторах `T` и `D` размерности $N = \frac{T_{\max} - T_{\min}}{\Delta_T} + 1$. Вектор `T` заполняется присвоением ему в качестве начального значения итеративного литерала:

```
[ for i in 1..N | Tmin+(i-1)*DeltaT ]
```

Напомним, что переменные цикла внутри итеративных литералов и операторов цикла не требуют предварительного определения, и переменная `i` в этом итеративном литерале не имеет ничего общего с внутренней переменной `i` — номером траектории.

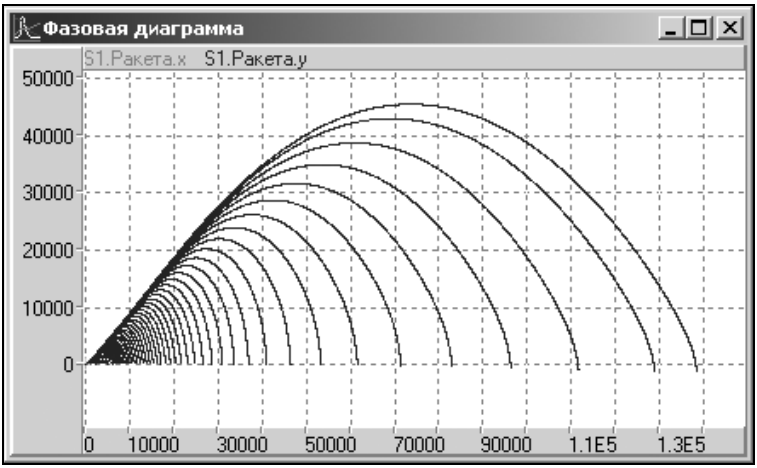


Рис. 1.65. Семейство траекторий

Состоянию `Траектория_i` приписан в качестве деятельности экземпляр класса `СУ_Т`. Всякий раз при входе в это состояние создается новый объект — экземпляр класса `СУ_Т` с `T[i]` в качестве значения параметра `T`. Этот объект получает имя, совпадающее с именем состояния, деятельностью в котором он является, т. е. `Траектория_i`. Поскольку класс `СУ_Т` гибридный, то безусловный переход из состояния `S1` срабатывает только тогда, когда достигается конечное состояние в карте поведения объекта-деятельности, т. е. в момент падения ракеты. Объект-деятельность существует до окончания срабатывания этого перехода, поэтому в действиях этого перехода мы можем занести полученную в `i`-м полете дальность в соответствующую строку вектора `D`. Переменные `Ti` и `Di` нужны только для построения графика (рис. 1.66). После

срабатывания перехода экземпляр класса `СУ_Т` уничтожается и т. д. `N` раз. Цикл организован с помощью специального состояния — точки ветвления. После `N`-го измерения происходит переход в конечное состояние с записью содержимого векторов `T` и `D` в соответствующие текстовые файлы.

Проведя этот эксперимент для интервала `Tmin=1`, `Tmax=50`, `DeltaT=1`, мы получим семейство траекторий, показанное на рис. 1.65, и зависимость `D(T)`, показанную на рис. 1.66.

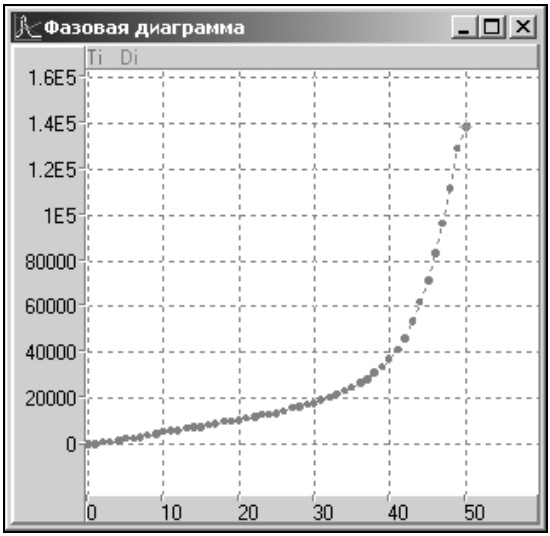


Рис. 1.66. Полученная в результате вычислительного эксперимента зависимость дальности полета `D` от времени отсечки тяги `T`

Отметим, что при проведении этого эксперимента мы воспользовались новым приемом: отлаженную ранее модель использовали как деятельность в карте поведения новой модели. Это соответствует последовательной композиции компонентов. До сих пор мы использовали ранее отлаженные модели как локальные объекты в новых моделях, т. е. параллельную композицию компонентов.

Получив "экспериментально" зависимость `D(T)`, мы теперь можем использовать ее для расчета "полетного задания", соответствующего требуемой дальности. Сохраним текущее состояние проекта как Ракета_6.mvb и вернемся к проекту Ракета_5.mvb, сохранив его как текущий проект Ракета.mvb.

В классе `Устройство_управления` параметр `T_от` сделаем входной переменной. План эксперимента задается картой поведения, показанной на рис. 1.67.

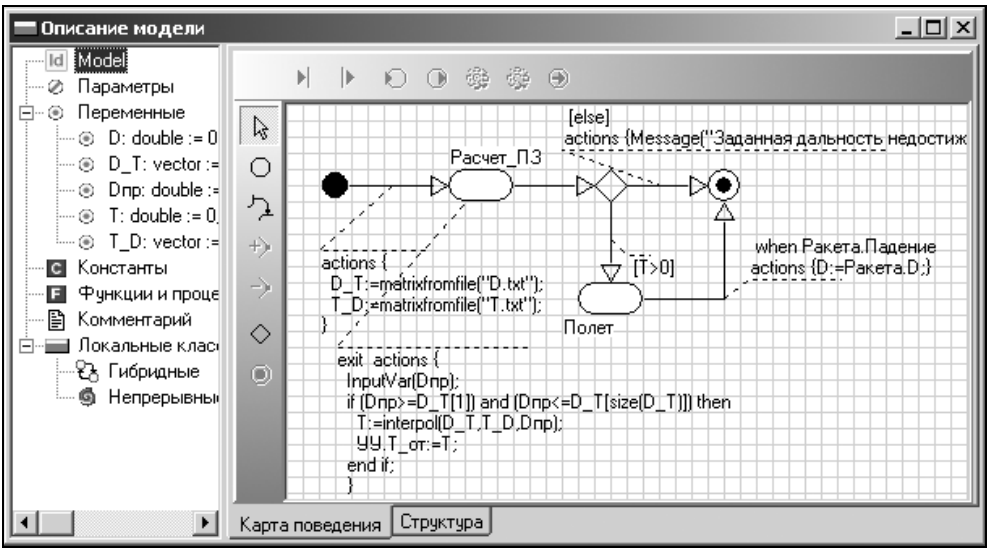


Рис. 1.67. Карта поведения для эксперимента с предварительным расчетом полетного задания

Обратите внимание на то, что векторы `D_T` и `T_D` определены как векторы переменной размерности. Конкретную размерность они приобретают только после загрузки в них данных из файлов. Для задаваемой пользователем дальности проверяется, попадает ли она в диапазон достижимых дальностей, и если попадает, то путем интерполяции определяется соответствующее ей значение времени отсечки тяги `T`, которое и передается на вход `T_от` устройства управления `УУ` (см. рис. 1.67).

Проведя несколько экспериментов, можно убедиться, что наша система управления работает. Точность можно повысить, увеличив число измерений для зависимости `D(T)`. Однако у нашей системы управления есть два серьезных недостатка:

- используемая программная траектория энергетически невыгодна. Вспомним, что дальность полета нашей ракеты может превышать 200 км, а для используемой программной траектории это только 138 км;

- система управления без обратной связи очень чувствительна к разбросам параметров ракеты. Пусть мы стреляем на 50 км зимой и удельная тяга двигателя на 1% меньше номинальной. Оказывается, что мы сразу промахиваемся на 2 км.

Займемся пока вторым недостатком как наиболее неприятным. Дальность полета на пассивном участке траектории полностью определяется начальными условиями, т. е. набором фазовых координат ракеты в конце активного участка: $D = f(x_K, y_K, V_K, \theta_K, m_K)$. Для ракет, у которых конец активного участка находится выше плотных слоев атмосферы, можно не учитывать влияние массы. Известно, что наибольшее влияние на дальность полета оказывает величина скорости V_K. Попробуем выдавать команду отсечки тяги не по времени полета, а по достижении программного значения скорости (скорость на активном участке является монотонно возрастающей функцией времени — см. рис. 1.56).

Прежде всего, нам понадобится зависимость `VK(T)`. Вернемся к модели Ракета_6.mvb, сохраним ее как Ракета_6a.mvb и немного модифицируем. В классе `УправляемаяРакета` предусмотрим выходной сигнал `Конец_АУ` с параметрами `x`, `y`, `Teta`, `V.m` типа `double`. В карте поведения этого класса предусмотрим в состоянии `Движение` внутренний переход, срабатывающий по сигналу `ОтсечкаТяги`, в действиях которого посылается сигнал `Конец_АУ` с текущими значениями координат, угла наклона траектории, скорости и массы в качестве фактических параметров (рис. 1.68).

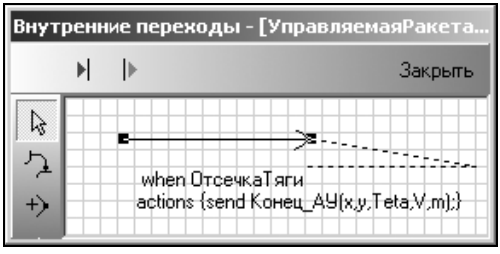


Рис. 1.68. Внутренний переход, посылающий сигнал об окончании активного участка с фазовыми координатами ракеты в качестве параметров

Отметим, что сигналы с параметрами являются очень удобным способом передачи совокупности значений, относящейся к какому-то определенному моменту времени.

Соответственно в состоянии `S1` карты поведения класса `СУ_Т` создадим внутренний переход, срабатывающий по сигналу `Ракета.Конец_АУ`, в действиях которого значения фактических параметров сигнала присваиваются переменным `xK`, `yK`, `TetaK`, `VK`, `mK`, которые тоже нужно добавить. Наконец, в описание модели мы добавим вектор `VK`, в который будем заносить значения переменной `Траектория_i.VK` для каждого измерения. После окончания измерений вектор `VK` сохраняется в файле VK.txt. После проведения эксперимента получим зависимость `VK(T)`, показанную на рис. 1.69.

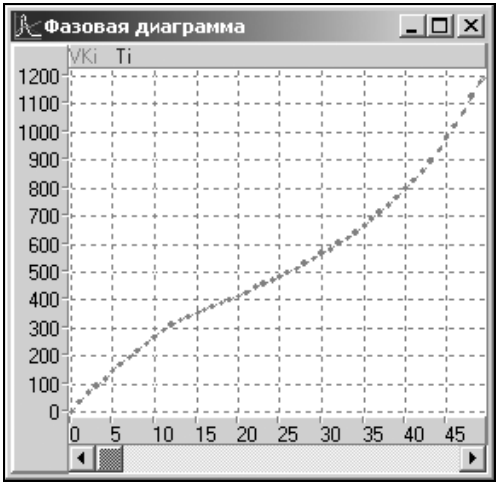


Рис. 1.69. Полученная в результате вычислительного эксперимента зависимость скорости в конце активного участка от времени отсечки тяги

Теперь, имея совокупность значений `D(T)` и `VK(T)`, мы можем найти требуемую зависимость `VK(D)`.

Вернемся к собственно системе управления. Для определения текущей скорости нам придется измерять ускорения центра масс ракеты в стартовой системе координат и затем интегрировать их. Это можно сделать многими различными способами, например, с помощью двух гироинтеграторов, установленных на гиростабилизированной платформе (допустим, что на нашей гипотетической ракете все это укладывается в 20 кг полезной нагрузки). В конечном счете мы должны получить значения составляющих скорости V_x и V_y, а по ним значение полной скорости V. Заметим, что по соотношению составляющих скорости можно определить угол наклона

траектории θ, а после дальнейшего интегрирования узнать текущие значения координат центра масс ракеты.

Модифицируем класс `Устройство_управления`. Заменим вход `Т_от` на вход `Vпр` — программное значение скорости, а также введем дополнительный вход `Vизм` — измеренное значение скорости (рис. 1.70).

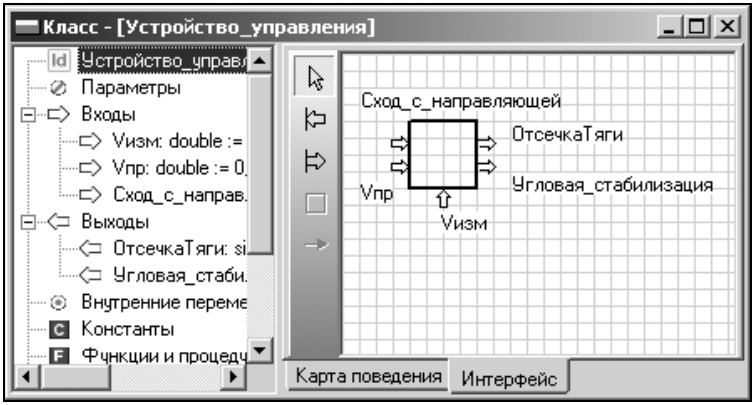


Рис. 1.70. Интерфейс устройства управления отсечкой тяги по скорости

Теперь устройству управления необходимо выдавать команду на отсечку тяги не через заданный интервал времени `Т_от`, а по достижении измеренного значения скорости программного значения (рис. 1.71).

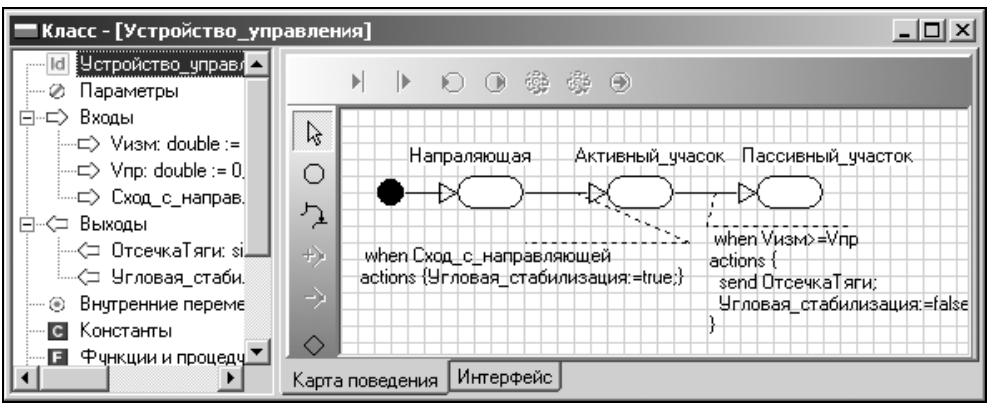


Рис. 1.71. Карта поведения устройства управления отсечкой тяги по скорости

Введем новый глобальный класс `Датчик_координат`. Будем рассматривать экземпляр этого класса как некоторое "интеллектуальное" устройство, которое измеряет значения ускорений центра масс ракеты в стартовой системе координат и затем на их основе вычисляет текущие значения скорости, угла наклона траектории и координат (рис. 1.72).

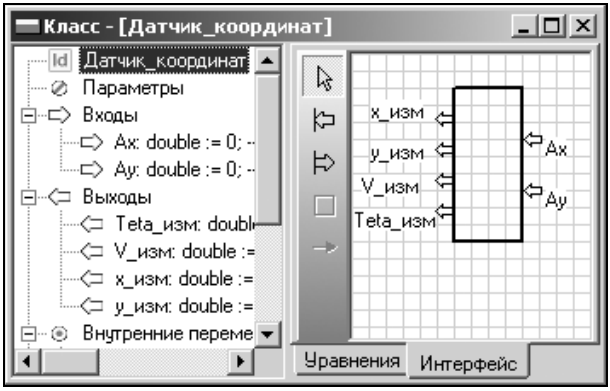


Рис. 1.72. Интерфейс класса `Датчик_координат`

Очевидно, что измерение ускорений осуществляется физическими приборами, а все вычисления, скорее всего, будут возложены на бортовую ЭВМ. Таким образом, объект класса `Датчик_координат` — это некоторый аппаратно-программный комплекс.

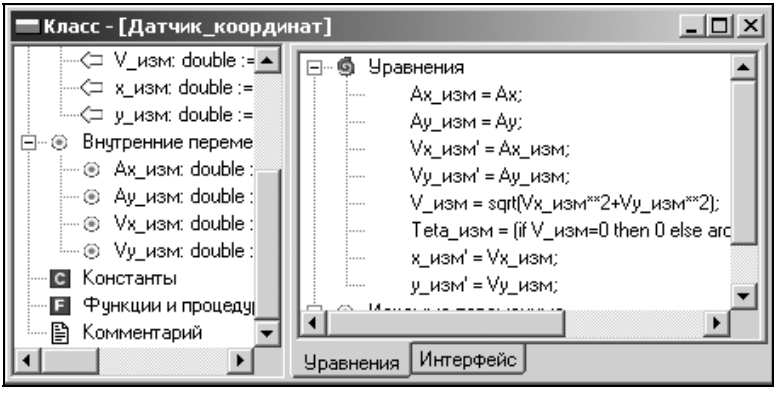


Рис. 1.73. Уравнения класса `Датчик_координат`

Однако для отработки алгоритмов управления это совершенно не важно, главное, что этот объект должен соответствовать заданной системе уравнений. Мы создадим идеальный датчик, в котором физические измерения ускорений будут осуществляться без ошибок (рис. 1.73). Самостоятельно вы можете исследовать вариант системы управления с учетом ошибок измерения.

Теперь можно собрать структурную схему нашего нового варианта системы управления (рис. 1.74).

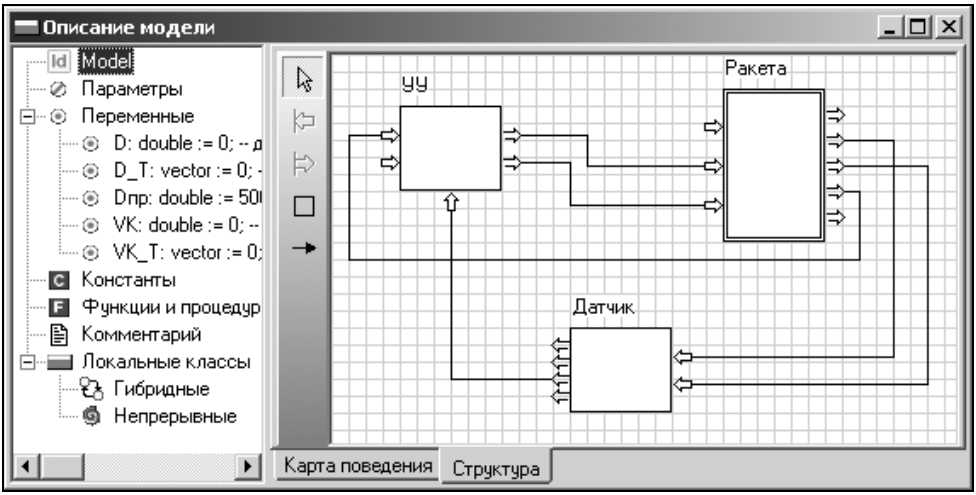


Рис. 1.74. Структурная схема системы управления отсечкой тяги по скорости

В качестве "полетного задания" теперь перед пуском ракеты рассчитывается программное значение скорости для заданной дальности (рис. 1.75).

Проведем вычислительный эксперимент и с некоторым недоумением убедимся, что наш новый продвинутый алгоритм управления при стрельбе на дальность 50 км дает перелет в 2 км, что гораздо хуже результата для старого варианта алгоритма управления. Так быть не может, поэтому начинаем поиск ошибки.

В выполняемой модели с помощью команды всплывающего меню откроем окно карты поведения устройства управления и предусмотрим отладочный останов по срабатыванию перехода из состояния `Активный_участок` в состояние `Пассивный_участок` (см. рис. 1.71). В момент срабатывания этого перехода остановим модель и сравним значения переменных в объектах `Ракета` и `Датчик`. Оказывается, что реальная скорость ракеты в момент отсечки тяги

существенно больше, чем измеренная. Проверяем уравнения датчика и не находим в них ничего неверного. Создадим временную диаграмму для реальной и измеренной скорости. После первых же секунд эксперимента видно, что они изменяются по-разному (рис. 1.76). Посмотрев на значения измеренных ускорений до 0.5 с полета, убеждаемся, что они нулевые. Тут нас озаряет догадка, и мы смотрим систему уравнений локального класса `Движение_по_направляющей`. Обнаруживаем, что формулы для ускорений в нее не добавлены. Добавляем эти формулы

$$A_x = \frac{dV}{dt} \cdot \cos Teta,$$

$$A_y = \frac{dV}{dt} \cdot \sin Teta,$$

и все становится на свои места.

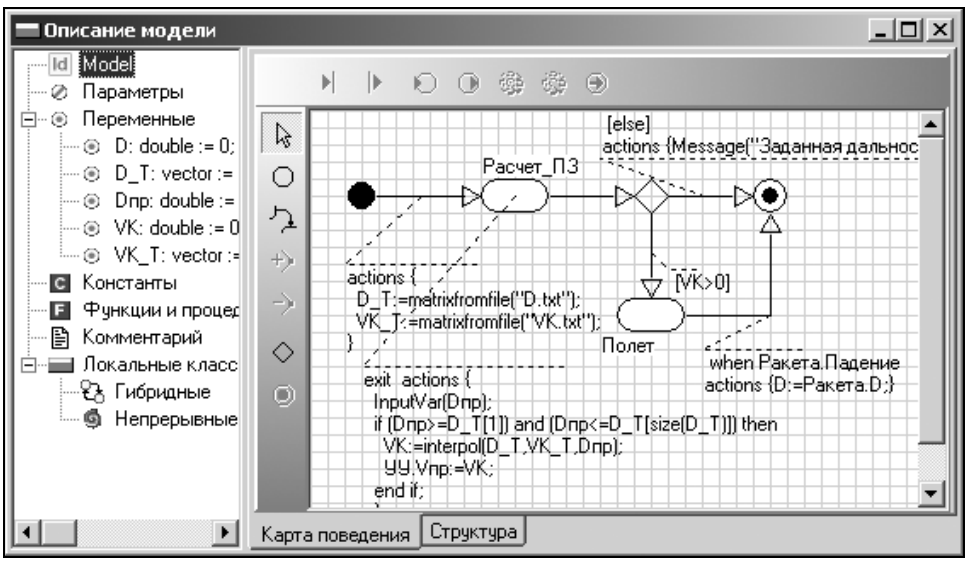


Рис. 1.75. План вычислительного эксперимента
с системой управления отсечкой тяги по скорости

Снова проделываем эксперимент с уменьшением и увеличением удельной тяги на 1%. Получаем промах около 400 м, что гораздо лучше, чем в предыдущем варианте. Несколько парадоксальным кажется то, что при пониженной удельной тяге получаем перелет, а при повышенной, напротив, недолет.

Однако, подумав, находим объяснение: при пониженной тяге двигатель выключается позже номинального времени, и дальность активного участка оказывается увеличенной при практически неизменной дальности пассивного участка (цель нашего управления). При повышенной удельной тяге наоборот двигатель выключается раньше, и дальность активного участка оказывается меньше номинальной.

Рис. 1.76. Зависимости реальной и измеренной скорости от времени

Эти размышления приводят нас к выводу, что для определения момента отсечки тяги недостаточно только величины скорости, и возникает необходимость вычисления так называемого линейного четырехчленного функционала (управляющей функции) [7, 8]. В качестве самостоятельного задания вам предлагается модифицировать ранее использованные модели для определения путем вычислительного эксперимента поправочных коэффициентов по двум координатам, скорости и углу наклона траектории и разработать устройство управления, выдающее команду на отсечку тяги по равенству текущего и программного значения функционала.

Попробуем теперь найти энергетически более выгодную программную траекторию. Предположим, что такая траектория имеет вид, показанный на рис. 1.77, и предусматривает изменение угла наклона траектории согласно некоторой программной зависимости.

Согласно этой программной зависимости, ракета сначала должна двигаться вертикально (или с каким-либо другим начальным углом θ_0) до точки *A*, а затем выполнять разворот, плавно уменьшая угол наклона траектории до не-

которого значения θ_K. После окончания разворота (точка *B*) ракета должна двигаться по прямой (участок выравнивания) до некоторой точки *K*, в которой производится "отсечка тяги". После этой точки ракета продолжает неуправляемый полет с начальными условиями $(x_K, y_K, V_K, \theta_K)$.

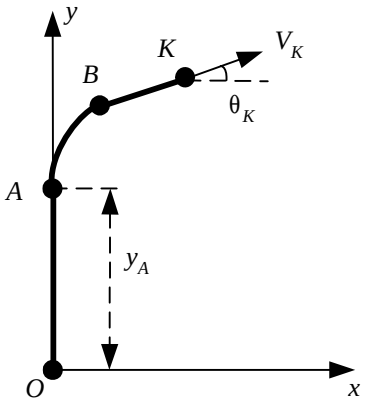


Рис. 1.77. Программная траектория с изменением угла наклона траектории

На практике системы управления ракетами обычно отрабатывают программные зависимости для угла тангажа ϑ, который непосредственно измеряется. Однако теоретически можно управлять и углом наклона траектории θ, что мы и реализуем в нашей гипотетической системе управления (исключительно для демонстрации типовых приемов создания моделей).

Для этого, прежде всего, нужно включить в нашу систему управления новый "контур" управления по θ. Для этого создадим новый класс `Устройство_управления_по_Teta` со стереотипом **Непрерывное**, **Открытая система**. Это устройство управления получает на вход программное и измеренное значения угла θ и на выходе выдает программное значение угла тангажа для системы угловой стабилизации (рис. 1.78).

Реализуем самый простой закон управления, когда управляющее воздействие пропорционально ошибке, и ограничим величину угла атаки 15-ю градусами (рис. 1.79).

Поместим экземпляр `УУ_Teta` этого класса в структурную схему системы управления и соединим его с соответствующими входами и выходами объектов `Датчик` и `Ракета` (рис. 1.80). Немедленно проверим правильность нашей модели: оставим угол наклона направляющей 45°, а `Teta_пр` положим рав-

ным 90° и посмотрим, выведет ли наша система управления ракету на вертикальную траекторию. Эксперимент показывает, что выводит (рис. 1.81).

Рис. 1.78. Интерфейс устройства управления по θ

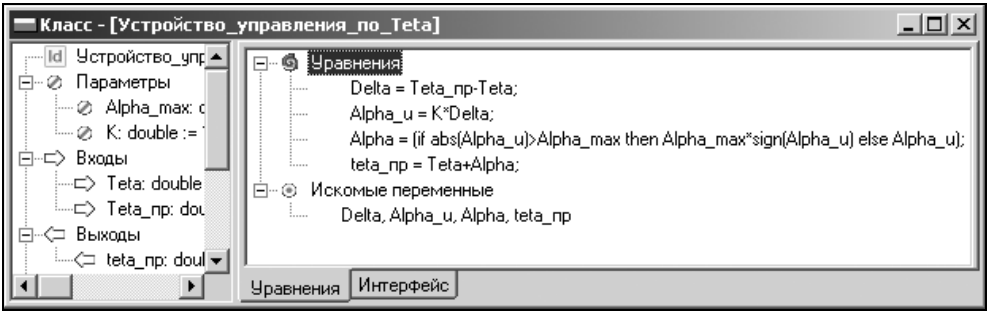


Рис. 1.79. Уравнения устройства управления по θ

Теперь создадим класс `Задатчик_программы_Teta`. Этот класс должен быть гибридным, т. к. программная траектория имеет три участка, а также должен быть открытой системой, т. к. это компонент. На вход задатчика должны поступать текущее значение высоты, а также параметры программной траектории (рис. 1.82).

Будем полагать, что на участке *AB* (состояние `Поворот` на рис. 1.83) программное значение угла меняется с постоянной скоростью в течение интервала τ. Тогда система уравнений локального непрерывного класса `Поворот_по_времени` должна содержать формулу

$$\theta_{\text{пр}} = \theta_0 + \frac{\theta_K - \theta_0}{\tau} \cdot \mathtt{time}.$$

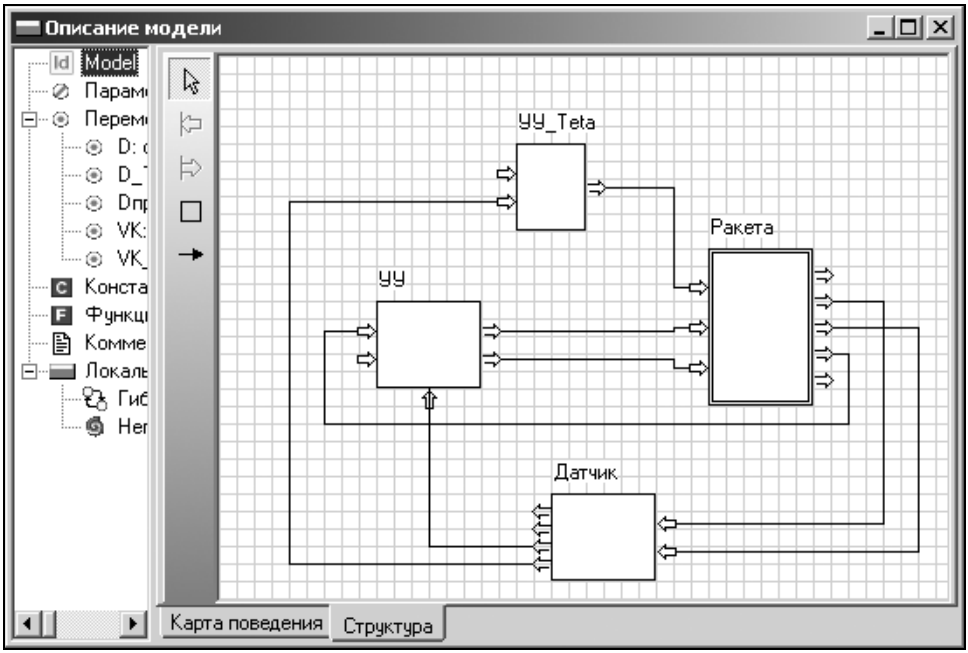


Рис. 1.80. Система управления с контуром управления по θ

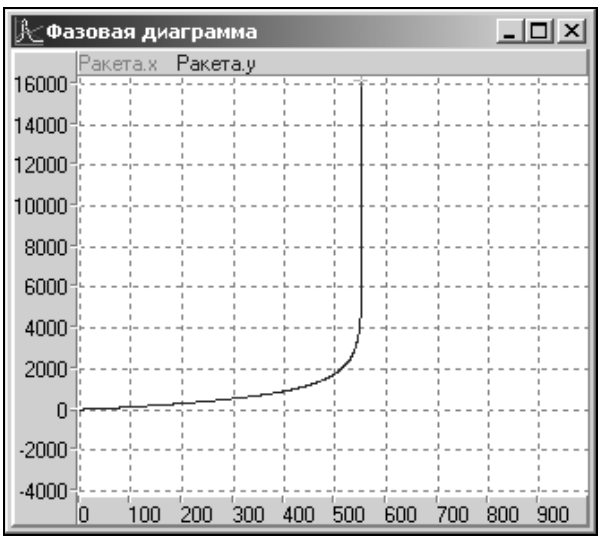


Рис. 1.81. Результат тестового эксперимента:
выход на вертикальную траекторию

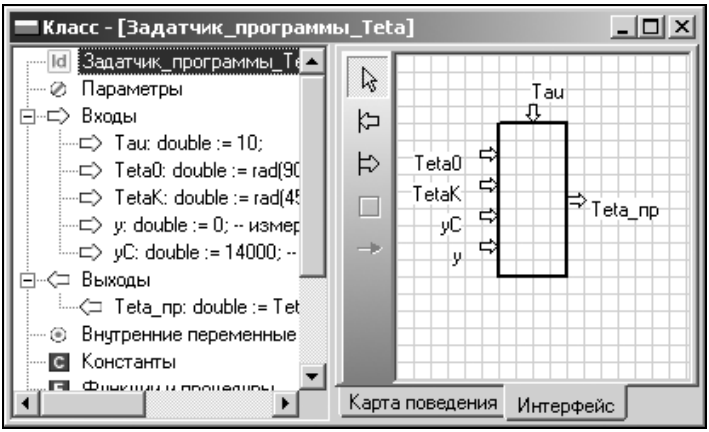


Рис. 1.82. Интерфейс задатчика программы θ

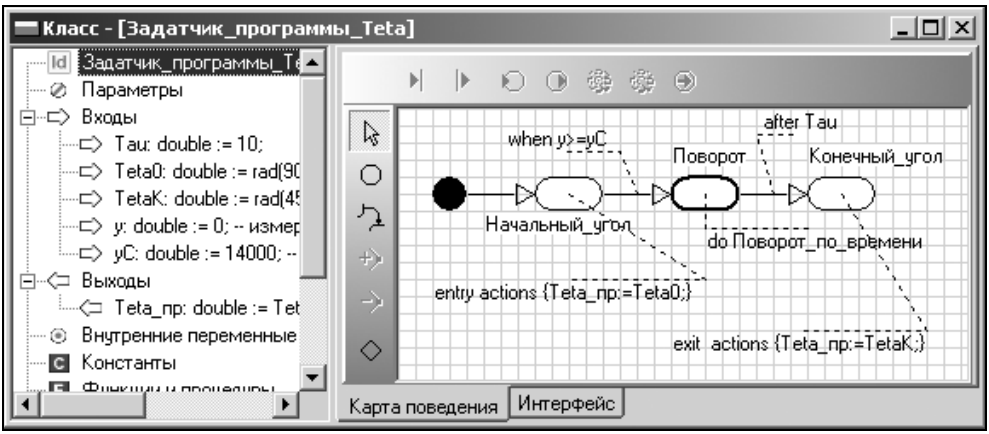


Рис. 1.83. Карта поведения задатчика программы θ

Заметим, что функция `time` возвращает значение локального времени от момента создания данного объекта. Так как экземпляр класса `Поворот_по_времени` создается при входе в состояние `Поворот`, то, следовательно, время отсчитывается от начала поворота.

Поместим объект `Программа_Teta` — экземпляр класса `Задатчик_программы_Teta` на структурную схему системы управления. Измеренное значение высоты подадим с объекта `Датчик`, а программное значение угла наклона траектории подадим на вход объекта `УУ_Teta` (рис. 1.84).

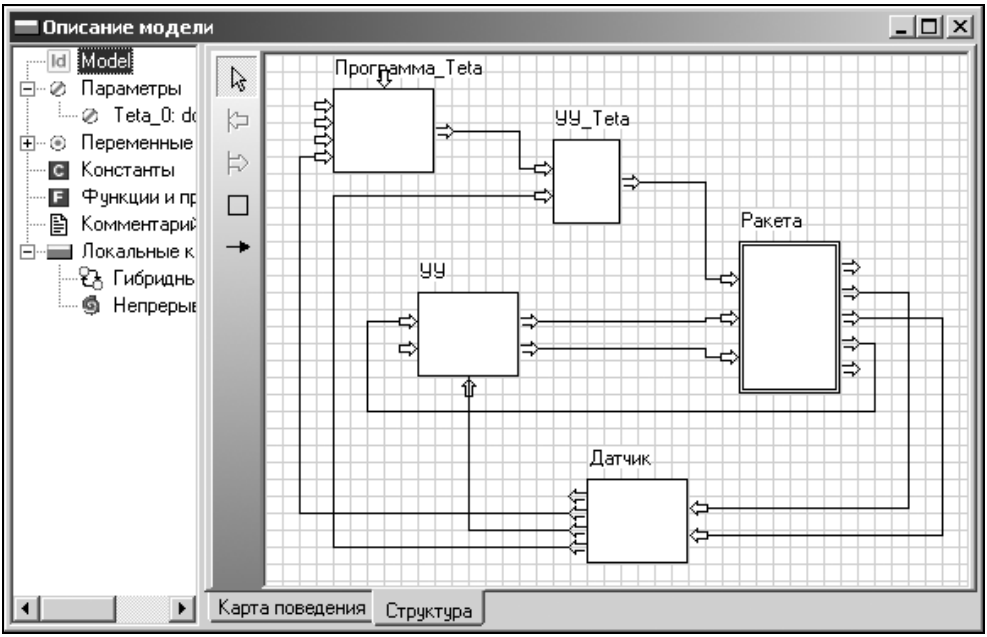


Рис. 1.84. Структура системы управления с программным изменением угла θ

Проведем контрольный эксперимент и убедимся, что программный отворот работает.

Попробуем найти максимально возможную дальность полета. Предварительно модифицируем карту поведения устройства управления отсечкой тяги, добавив переход, который на 50-ой секунде полета (время выгорания всего запаса топлива) отключает систему угловой стабилизации. Входим в подсистему оптимизации и ищем максимум переменной `D` в момент падения, рассматривая в качестве параметров оптимизации значения входов `yC`, `Tau` и `TetaK` объекта `Программа_Teta`. Обнаруживаем, что при $\theta_K = 39.7°$, $y_C = 8559$ м, $\tau = 23.2$ с и полном вырабатывании топлива достигается дальность полета в 228 км (рис. 1.85).

При стрельбе на заданную дальность $D_{пр}$ можно найти сочетание параметров V_K, θ_K, y_C, τ, которое минимизирует функционал $(D - D_{пр})^2$. Например, для дальности 50 км получаем $V_K = 859.2$ м/с, $\theta_K = 26.26°$, $y_C = 5652.6$ м, $\tau = 25.29$ с и траекторию, показанную на рис. 1.86.

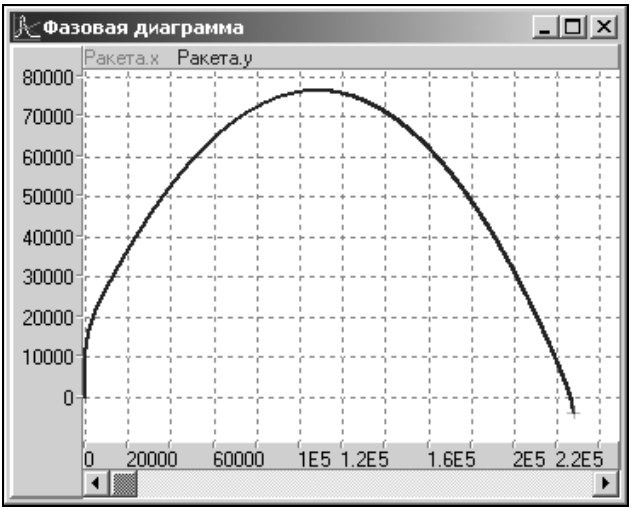


Рис. 1.85. Траектория максимальной дальности с программным изменением угла θ

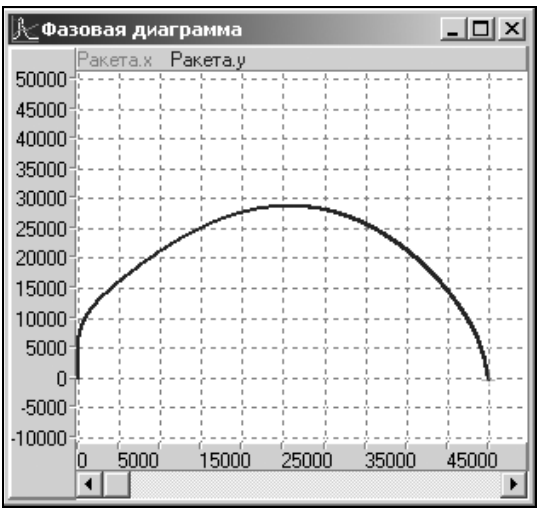


Рис. 1.86. Траектория с программным изменением угла θ для дальности полета 50 км

Сохраним этот вариант модели как Ракета_7.mvb и перейдем к следующему и последнему варианту системы управления ракетой. Попробуем вообще не отключать двигатель и управлять ракетой на всей траектории, стремясь сбли-

зить действительную траекторию ракеты и заранее вычисленную программную траекторию. Эта модель находится в файле Ракета_8.mvb.

Для реализации нового принципа управления нам потребуется совсем новое устройство управления. Удалим классы `Устройство_управления`, `Устройство_управления_по_Teta` и `Задатчик_программы_Teta`. Предварительно необходимо удалить их экземпляры — объекты `УУ`, `УУ_Teta` и `Программа_Teta` в структурной схеме модели. В противном случае редактор проекта не даст вам удалить эти классы, показав места их использования в модели (рис. 1.87). Аналогичная ситуация может возникнуть, например, при попытке изменить тип переменной. Если эта переменная является внешней и участвует в связи, то придется сначала удалить эту связь, затем изменить тип обеих переменных, соединяемых связью, а затем заново создать связь. Следует отметить, что при сложных модификациях модели, одновременно затрагивающих множество элементов проекта, иногда удобнее корректировать текстовое представление проекта, получаемое с помощью команды **Проект | Экспорт в MVL...**.

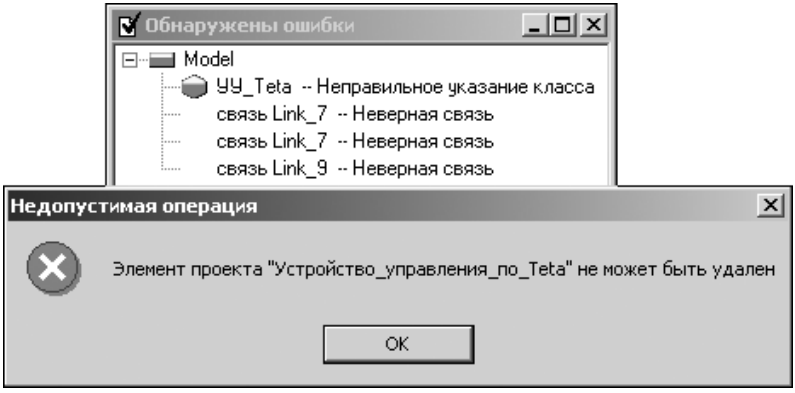


Рис. 1.87. Сообщение об ошибке при попытке удаления класса без удаления его экземпляров

Попробуем задавать программную траекторию параболой вида

$$y = k_1 \cdot (x - k_2)^2 + k_3,$$

проходящей через точку старта, точку цели и среднюю точку с высотой, равной одной трети дальности. В этом случае $\operatorname{tg}\theta = \dfrac{dy}{dx} = 2 \cdot k_1 \cdot (x - k_2)$.

Создадим новый класс `Устройство_управления`. На вход экземпляра этого класса поступают измеренные значения x, y, θ, а также вектор коэффициентов параболы k и на выходе выдается программное значение угла тангажа $\vartheta_{пр}$ (рис. 1.88).

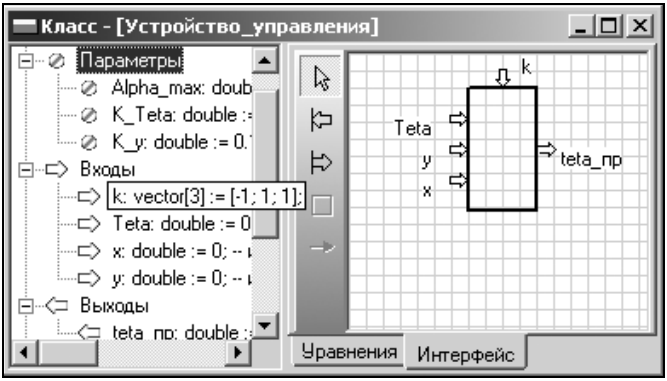


Рис. 1.88. Интерфейс устройства управления на всей траектории

Реализуем простейший закон управления: угол атаки пропорционален сумме рассогласований по вертикальной координате и по углу наклона траектории (рис. 1.89). Угол атаки ограничен по абсолютной величине.

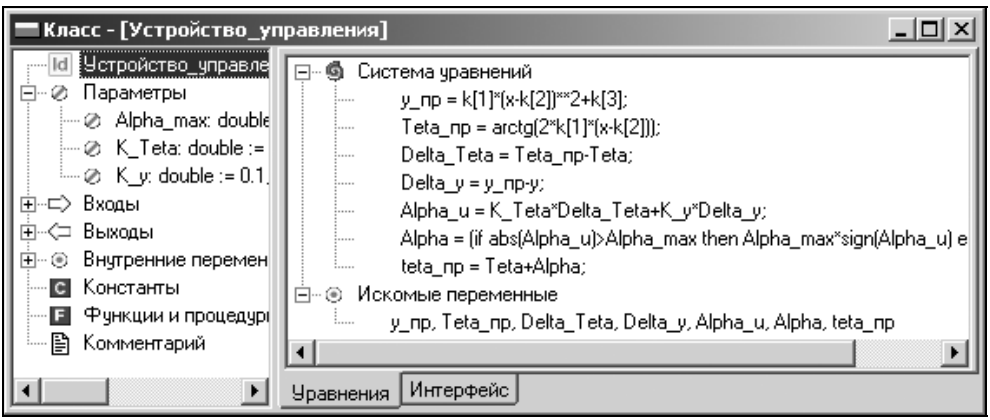


Рис. 1.89. Уравнения устройства управления на всей траектории

Поместим экземпляр класса `УУ` в структурную схему модели и создадим необходимые связи (рис. 1.90).

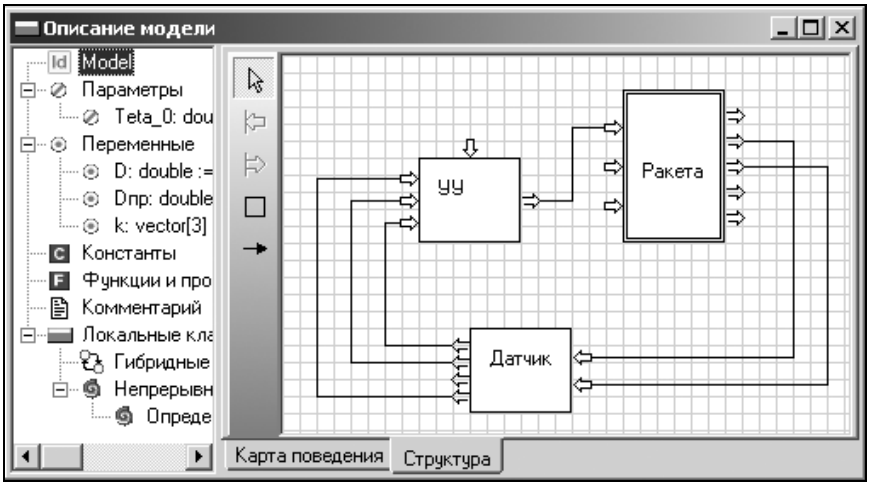


Рис. 1.90. Структура системы управления на всей траектории

"Полетным заданием" в данном случае являются коэффициенты программной траектории, определяемые для заданного значения дальности полета $D_{пр}$, которое вводится пользователем при срабатывании начального перехода в карте поведения модели (рис. 1.91). В состоянии `Полет` создан внутренний переход, срабатывающий по сигналу `Ракета.Сход_с_направляющей`, в действиях которого присваивается значение `true` входу `Ракета..Угловая_стабилизация`.

Коэффициенты программной траектории определяются путем решения системы из трех алгебраических уравнений в локальном классе `Определение_коэффициентов` (рис. 1.92). Здесь использован прием построения гибридных моделей, требующий некоторого пояснения. Экземпляр локального класса `Определение_коэффициентов` приписан в качестве деятельности состоянию `Расчет_ПЗ` (см. рис. 1.91). Существует также безусловный переход из этого состояния в состояние `Полет`. Таким образом, при запуске модели произойдет мгновенный переход из начального состояния в состояние `Расчет_ПЗ` и затем в состояние `Полет`. На этом действия в начальной "временной щели" закончатся при значении непрерывной составляющей модельного времени 0 и значении дискретной составляющей 3. При входе в состояние

`Расчет_ПЗ` был создан экземпляр класса `Определение_коэффициентов`, его система уравнений была добавлена к совокупной системе уравнений модели и была решена алгебраическая составляющая этой совокупной системы уравнений. В результате этого решения было определено значение переменной `k` класса `Model`. При выходе из состояния `Расчет_ПЗ` экземпляр класса `Определение_коэффициентов` был уничтожен, а в выходных действиях значение переменной `k` было присвоено соответствующему входу объекта `УУ` (см. рис. 1.91). Таким образом, для решения чисто алгебраической системы уравнений "в точке" следует использовать состояния с мгновенным выходом.

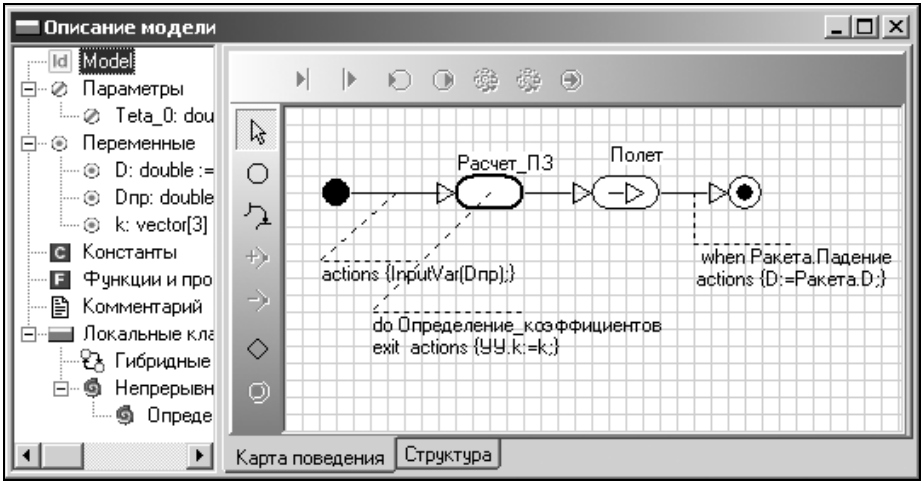


Рис. 1.91. Карта поведения системы управления на всей траектории

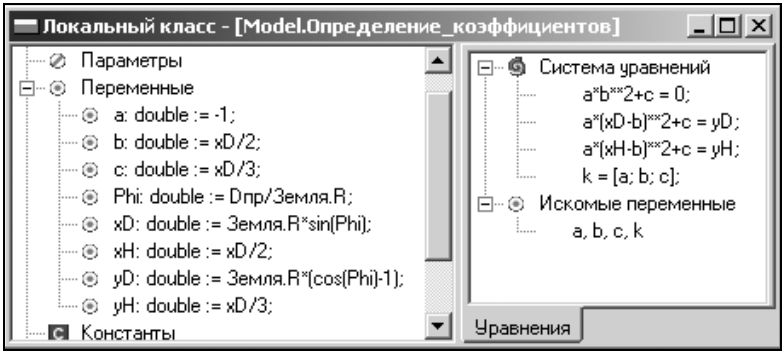
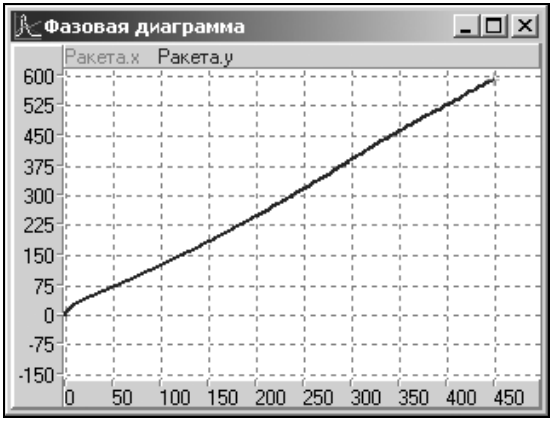
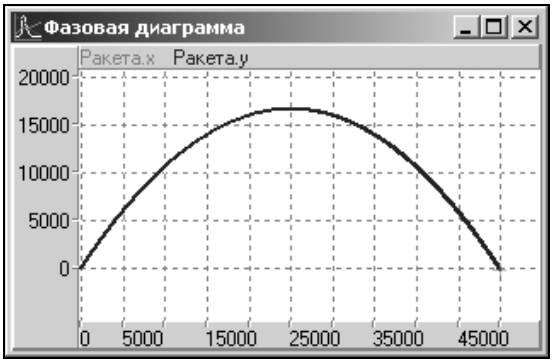


Рис. 1.92. Система уравнений
для определения коэффициентов программной траектории

Следует помнить, что в момент входа в состояние `Расчет_ПЗ` уже существуют локальные объекты "`УУ`, `Датчик` и `Ракета`, и совокупная система уравнений модели уже содержит уравнения этих объектов. Поэтому для успешного решения алгебраической составляющей этих уравнений и задано ненулевое начальное значение вектора `k` в классе `Устройство_управления` (см. рис. 1.88). Альтернативным решением могло бы быть выделение схемы, показанной на рис. 1.90, в отдельный глобальный класс и приписывание его экземпляра состоянию `Полет`. В этом случае вектор коэффициентов можно бы было передать как фактическое значение параметра.



а



б

Рис. 1.93. Траектория ракеты при заданной дальности 50 км:
а — начальный участок; *б* — вся траектория

Проведем вычислительный эксперимент для дальности полета 50 км. Результаты эксперимента показывают блестящий успех нашего нового подхода к управлению: после небольшого переходного процесса на начальном участке траектории (рис. 1.93, *а*) реальная траектория практически совпадает с программной (рис. 1.93, *б*). Зависимости значений рассогласований по θ и y подтверждают это (рис. 1.94).

а

б

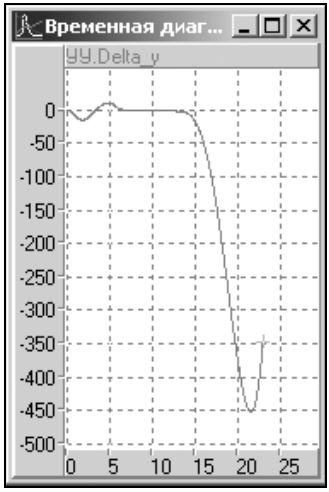
Рис. 1.94. Зависимость рассогласования от времени при заданной дальности 50 км: *а* — по углу наклона траектории; *б* — по вертикальной координате

Начальный переходный процесс обусловлен тем, что угол наклона направляющей (90°) отличается от начального наклона программной траектории.

Самым же важным результатом является то, что при отклонении значения удельной тяги на 5% в обе стороны от номинала отклонение дальности не превышает пяти метров (еще раз напоминаем, что все эти цифры не имеют никакого отношения к управлению реальными ракетами).

Однако абсолютных решений в технике не бывает, и в этом мы легко можем убедиться, проведя вычислительные эксперименты со стрельбой на минимальные дальности, когда падение ракеты происходит с еще работающим двигателем. Для дальностей, меньших 5 км, системе управления просто не хватает управляющих сил, чтобы круто развернуть ускоряющуюся ракету на нисходящем участке траектории (рис. 1.95).

а



б

Рис. 1.95. Зависимость рассогласования от времени при заданной дальности 4.5 км: *а* — по углу наклона траектории; *б* — по вертикальной координате

Подобная проблема возникает и при стрельбе на большие дальности. Однако здесь она обусловлена не большой кривизной программной траектории, а низкой плотностью воздуха на верхнем участке траектории. В результате на верхнем участке возникают значительные рассогласования, отработка которых приводит к колебательным переходным процессам (рис. 1.96).

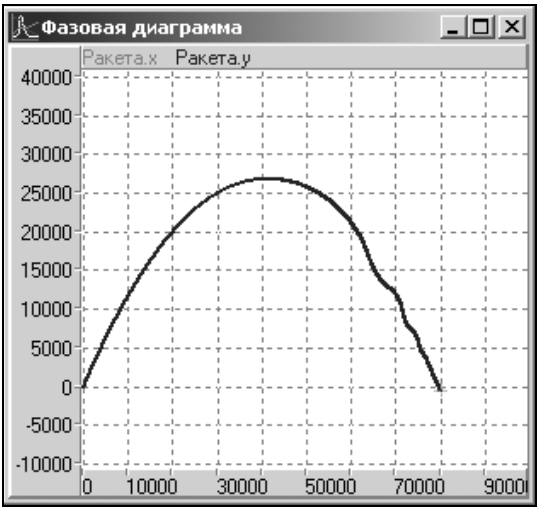


Рис. 1.96. Траектория ракеты при заданной дальности 80 км

В какой-то степени эти проблемы можно решать, понижая высоту программной траектории для малых и больших дальностей (в первом случае для уменьшения кривизны траектории, а во втором случае для помещения верхнего участка траектории в более плотные слои атмосферы). Это позволяет расширить диапазон дальностей нашей гипотетической ракеты до 3—150 км (для реальных ракет, управляемых на всей траектории, этот диапазон значительно уже [8, 9], т. к. в нашей модели не учитывается ряд дополнительных факторов).

Возникает вопрос: зачем при больших отклонениях упорно продолжать стремиться к программной траектории вместо того, чтобы найти новую программную траекторию, проходящую через цель и текущую точку с текущим углом наклона траектории? Подход, подразумевающий изменение программной траектории в ходе реального полета, существует и называется *терминальным управлением*. Интересующийся читатель может найти дополнительную информацию по этому вопросу в [9].

Глава 2

Типовые вычислительные эксперименты

В предыдущей главе рассматривалась технология разработки и отладки моделей различных типов. Предположим, что вы разработали модель, и она ведет себя соответственно вашим представлениям о ее правильной работе. В этой главе мы рассмотрим вычислительные эксперименты, проводимые пользователем с целью исследования построенной модели. Конечно, в предыдущей главе уже рассматривались вычислительные эксперименты, необходимые для отладки моделей. В этой главе мы акцентируем внимание на способах построения плана эксперимента на основе объектно-ориентированного подхода [6] с учетом различных аспектов использования моделей.

Для планирования вычислительного эксперимента мы будем пользоваться исключительно средствами самого языка моделирования, задавая алгоритм проведения эксперимента с помощью карты поведения класса `Model` и используя такие приемы, как сохранение разработанной и отлаженной модели в виде класса, параллельную и последовательную композицию компонентов. Все рассматриваемые в этой главе примеры находятся в папке ...\Модели\Эксперименты.

Реализация определенного сценария поведения

Это наиболее простой план вычислительного эксперимента, сводящийся к подаче в определенные моменты времени на входы испытуемой модели необходимых значений и контролю значений на выходах. В ходе отладки обычно такие планы реализуются вручную в интерактивном режиме, в том

числе с помощью интерактивных анимационных компонентов. Однако последовательность воздействий может быть довольно сложной; кроме того, часто возникает необходимость повторного проведения таких экспериментов после изменений в модели.

В качестве примера тестируемой модели (файл Эксперимент_1.mvb) рассмотрим простейшую систему автоматического регулирования, состоящую из объекта управления (колебательное звено) и устройства управления, вырабатывающего сигнал управления пропорционально ошибке регулирования (рис. 2.1).

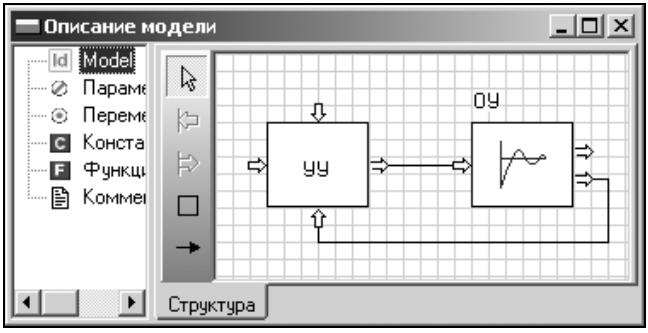


Рис. 2.1. Тестируемая модель

Создадим план вычислительного эксперимента, в котором на вход программного значения устройства управления сначала подается (+1), затем через 10 секунд (0) и еще через 10 секунд синусоидальный сигнал с периодом 5 секунд и амплитудой 0.5. Каждый раз через 5 секунд после подачи нового программного сигнала (предполагаемое время завершения переходного процесса) необходимо в течение 5 секунд проверять, что регулируемая величина отклоняется от программного значения не более чем на 0.05.

Для этого изменим с помощью команды всплывающего меню **Преобразовать в ...** стереотип поведения модели на **Гибридный объект** и нарисуем карту поведения, показанную на рис. 2.2. В состояниях `ПП1`, `ПП2`, `ПП3` на вход устройства управления подается соответствующий программный сигнал и выдерживается пауза 5 секунд на завершение переходного процесса. В состояниях `К1`, `К2`, `К3` в течение 5 секунд непрерывно проверяется условие $\mathrm{abs}(УУ.X_{\text{пр}} - УУ.X_{\text{изм}}) \le E_{\max}$, оформленное в виде алгоритмической функции `В_пределах (Emax: double)`.

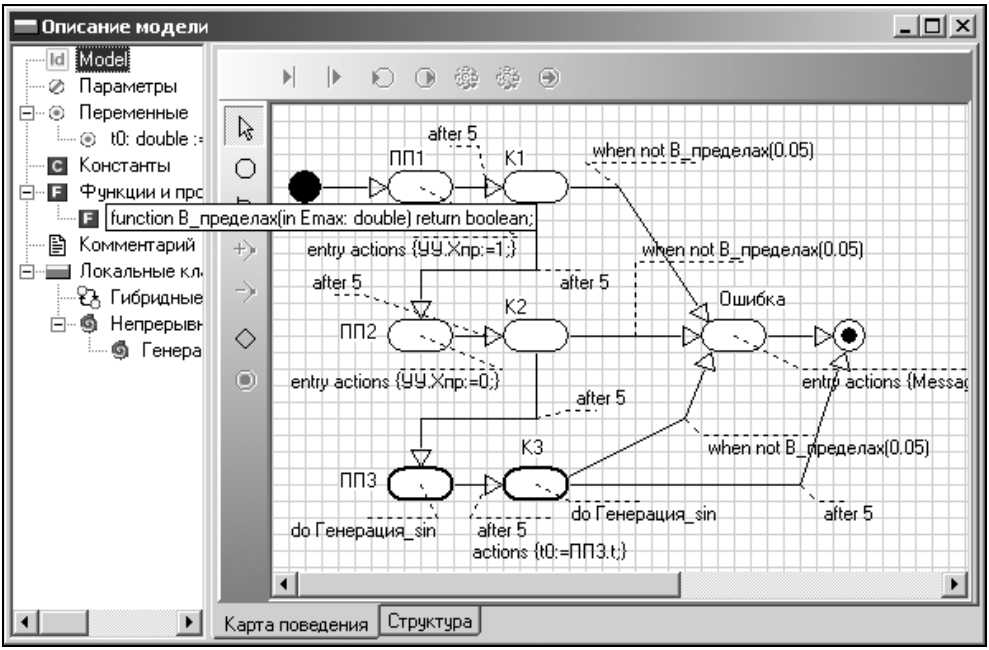
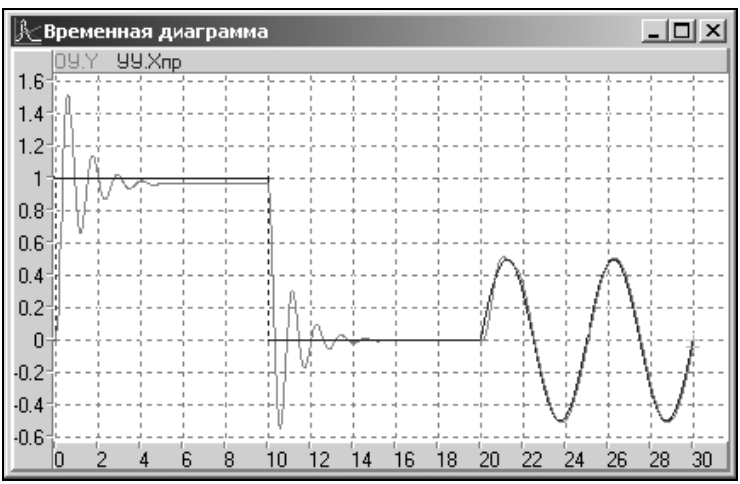


Рис. 2.2. План эксперимента для тестирования системы автоматического регулирования



а

Рис. 2.3. Результаты тестирования системы регулирования:
а — зависимости программного и действительного значения регулируемой величины от времени

б

Рис. 2.3. Результаты тестирования системы регулирования:
б — зависимость ошибки регулирования от времени

При нарушении этого условия происходит переход в состояние `Ошибка` с выдачей соответствующего сообщения. Если условие не нарушалось, то происходит переход к следующему тесту или завершение эксперимента. Переменная `t0` служит для согласования фазы синусоиды в состояниях `ПП3` и `К3`.

Проведя вычислительный эксперимент согласно плану, получаем графики (рис. 2.3), свидетельствующие о том, что требуемые тесты выполняются, и ошибка регулирования на интервалах контроля действительно не превышает 0.05.

Сравнение вариантов

Часто возникает необходимость продемонстрировать функционирование нескольких вариантов исследуемой системы. Пусть, например, мы хотим показать реакции нашей системы регулирования на единичное ступенчатое воздействие при значениях коэффициента усиления 1, 10 и 100. Это можно сделать двумя способами. В обоих случаях нам понадобится модель системы регулирования, преобразованная в глобальный класс `САР`, который сделаем открытой системой и блок-схемой без собственного поведения (рис. 2.4). Коэффициент усиления задается параметром `Крег`.

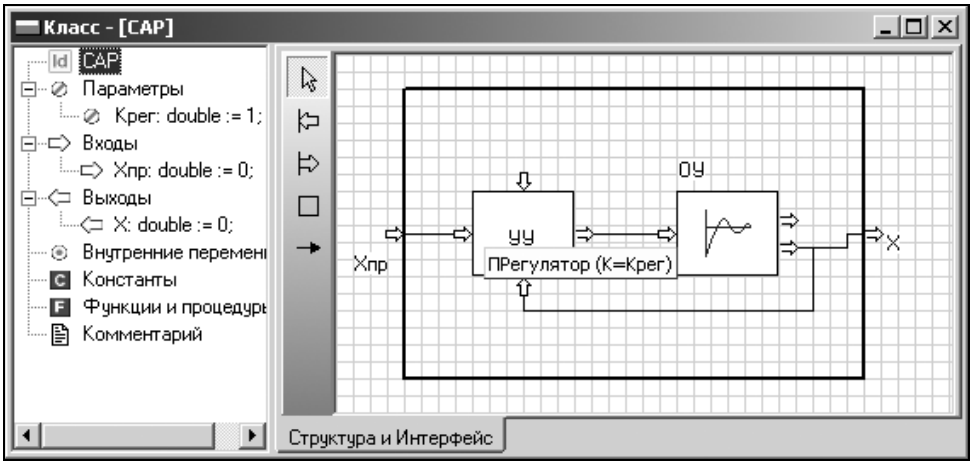


Рис. 2.4. Структурная схема класса `CAP`

Параллельная композиция вариантов

Пример этого эксперимента находится в файле Эксперимент_2a.mvb.

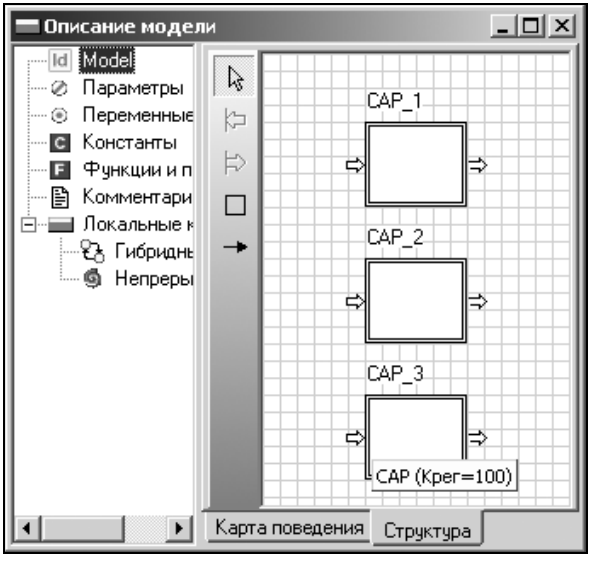


Рис. 2.5. Структура модели при параллельной композиции вариантов

Нам понадобится модель со стереотипом "гибридный составной объект". Создадим три локальных объекта — экземпляры класса `CAP` с различными значениями параметра `Крег` (рис. 2.5).

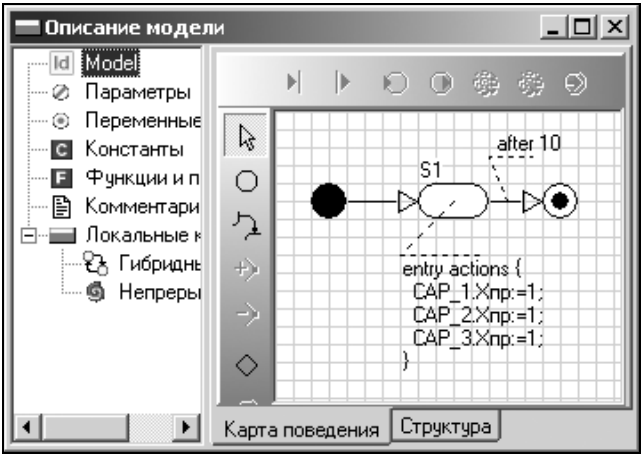


Рис. 2.6. Карта поведения модели
при параллельной композиции вариантов

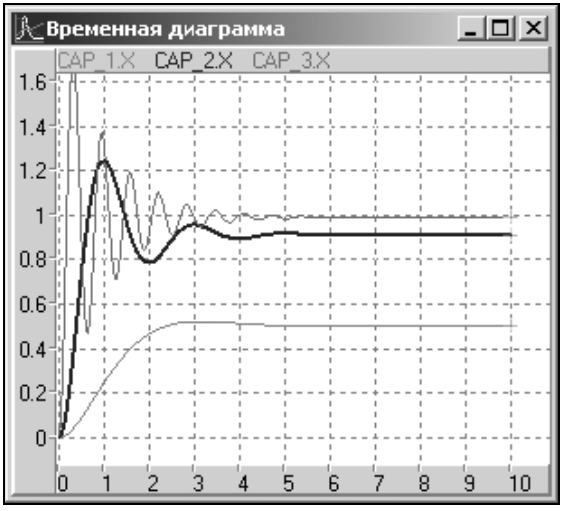


Рис. 2.7. Переходные процессы модели
при параллельной композиции вариантов

Карта поведения модели обеспечивает подачу единичного значения программного сигнала на вход всех трех объектов и наблюдение в течение 10 секунд одновременно всех трех переходных процессов (рис. 2.6 и 2.7).

Последовательная композиция вариантов

Пример этого эксперимента находится в файле Эксперимент_2б.mvb. Нам понадобится модель со стереотипом "гибридный элементарный объект". Карта поведения модели (рис. 2.8) позволяет создать в качестве деятельности в состоянии `Вариант_i` экземпляр класса `CAP` с единичным программным значением на входе и значением коэффициента усиления, равным `i`-му элементу вектора вариантов коэффициента усиления `K`. Переходный процесс для каждого варианта наблюдается в течение 10 секунд. Число вариантов равно размерности вектора `K`.

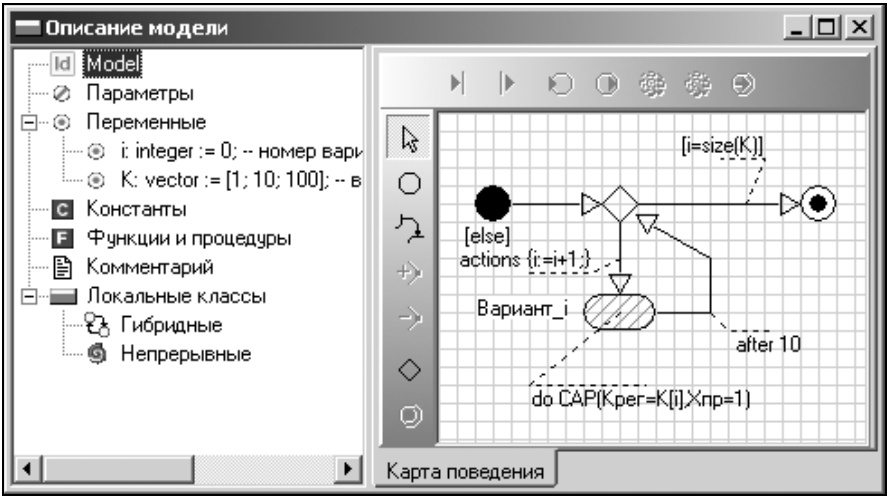


Рис. 2.8. Карта поведения модели при последовательной композиции вариантов

Переходные процессы для различных вариантов наблюдаются последовательно (рис. 2.9).

Рис. 2.9. Переходные процессы модели при последовательной композиции вариантов

Получение параметрической зависимости

Пример этого эксперимента находится в файле Эксперимент_3.mvb. Целью эксперимента является получение зависимости некоторых показателей работы исследуемой системы от параметров. Важно, что для определения значения показателей требуется прогон модели с определенными значениями параметров. В качестве примера рассмотрим вычислительный эксперимент с моделью системы регулирования, целью которого является получение зависимости величины перерегулирования и времени переходного процесса от значения коэффициента усиления регулятора. В данном случае эти показатели непосредственно не вычисляются в исследуемой модели. Поэтому их определение также относится к планированию эксперимента. Для этой цели создадим специализированную версию системы регулирования, воспользовавшись механизмом наследования. Выполним команду **Новый класс** всплывающего меню в менеджере проекта, назовем новый класс `САР_Э` и в появившемся диалоге выберем существующий класс `САР` в качестве базового (рис. 2.10).

Производный класс получает тот же стереотип, что и базовый, но его затем можно изменить в допустимых пределах. Новый класс `САР_Э` наследует все переменные и локальные объекты своего базового класса (рис. 2.11).

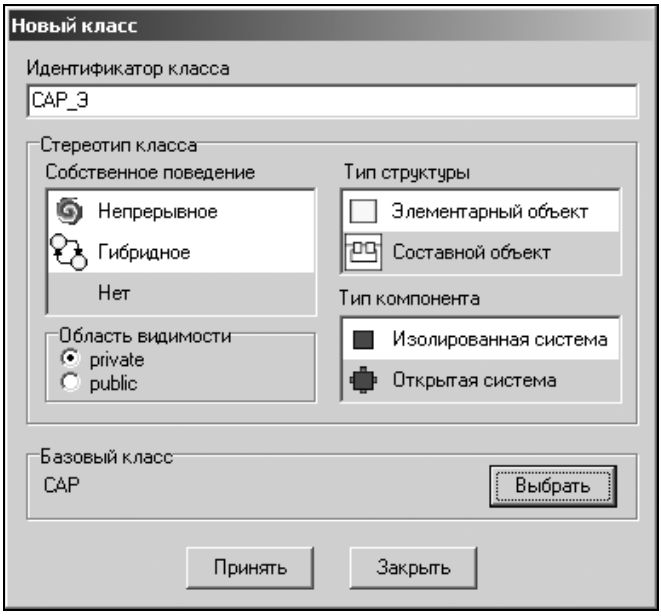


Рис. 2.10. Создание нового класса, производного от класса `САР`

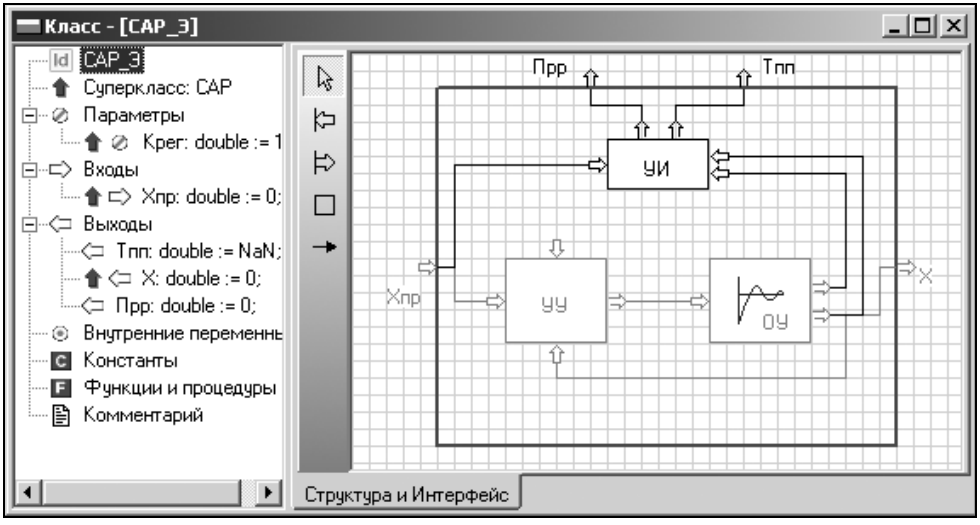


Рис. 2.11. Производный класс `САР_Э`: унаследованные переменные помечены значком ⬆, а унаследованные локальные объекты и связи нарисованы серой линией

Специализация класса заключается в том, что мы добавим два новых выхода (`Прр` — величина перерегулирования и `Тпп` — время переходного процесса, начальное значение `NaN`), а также устройство измерения показателей `УИ` и соответствующие связи.

В состоянии `ПП` карты поведения класса `Устройство_измерения` (рис. 2.12) выполняется измерение величины перерегулирования и времени переходного процесса. По завершении этого измерения (состоянию `ПП` приписана гибридная деятельность) выполняется безусловный переход в конечное состояние с выдачей на выход измеренного значения. Если переходный процесс не завершился в течение времени `Tmax`, то выполняется переход в конечное состояние и присваивание выходу `Тпп` значения "бесконечность", а выходу `Прр` — значения `NaN` (рис. 2.12).

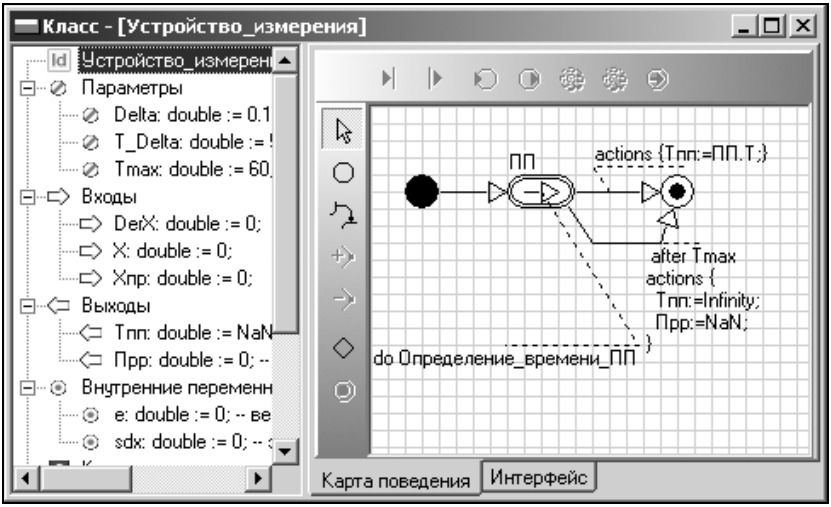


Рис. 2.12. Карта поведения устройства измерения

Переходный процесс считается закончившимся, если разность программного и действительного значения регулируемой величины по абсолютной величине не превышает некоторого значения `Delta`, непрерывного в течение интервала времени `T_Delta`. Этот алгоритм реализуется картой поведения локального класса `Определение_времени_ПП`, показанной на рис. 2.13.

Величина перерегулирования определяется с помощью двух внутренних переходов в состоянии `ПП` карты поведения устройства измерения (см. рис. 2.12). Эти переходы фиксируют моменты максимумов и минимумов за-

висимости $X(t)$ по изменению знака производной регулируемой величины и запоминают максимальное отклонение максимума или минимума от программного значения (рис. 2.14).

Рис. 2.13. Карта поведения
для измерения времени переходного процесса

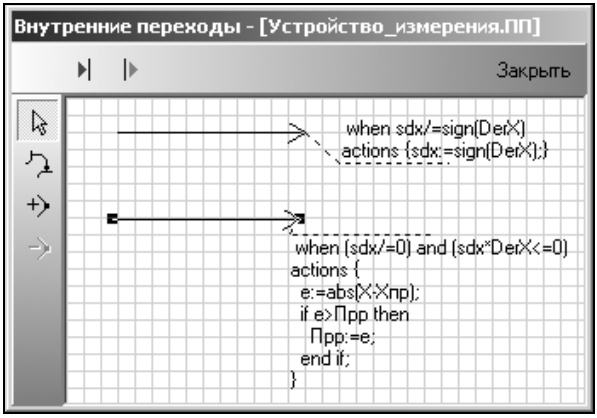


Рис. 2.14. Внутренние переходы,
измеряющие величину перерегулирования

На рис. 2.15 показана карта поведения модели в целом, задающая план эксперимента. Множество значений коэффициента усиления зафиксировано в векторе `K`. При входе в состояние `Вариант_i` создается экземпляр класса `САР_Э` с единичным значением входа `Хпр` и `i`-м значением коэффициента уси-

ления. Выход из состояния происходит либо по завершении переходного процесса, либо по истечении предельного периода измерения (см. рис. 2.12).

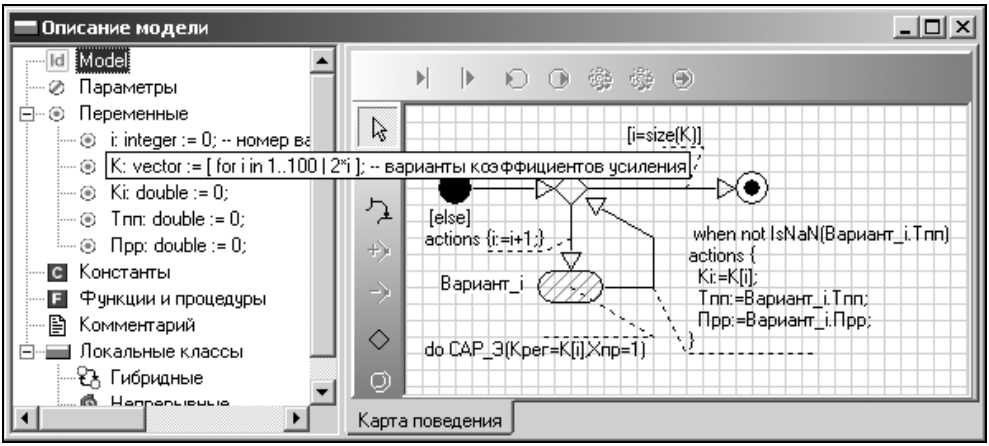
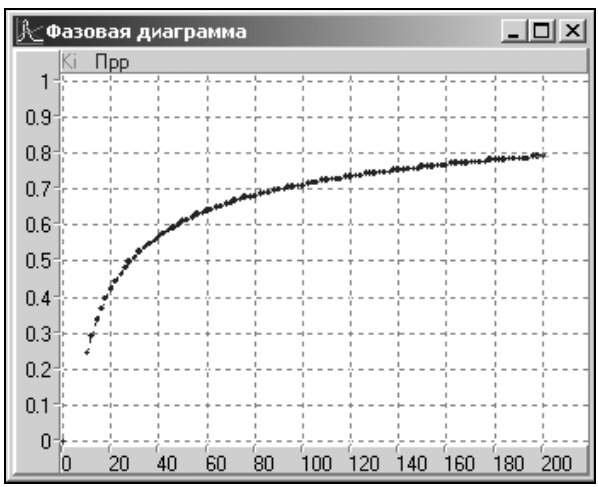
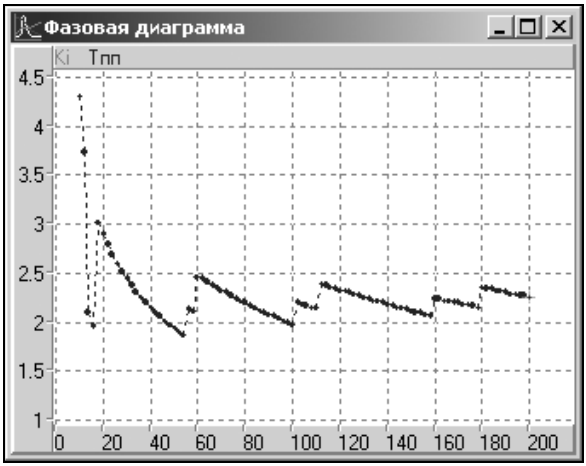


Рис. 2.15. Карта поведения модели, задающая план эксперимента по определению зависимости времени переходного процесса и величины перерегулирования от коэффициента усиления устройства управления



а

Рис. 2.16. Полученные в вычислительном эксперименте зависимости величины перерегулирования (*а*) от величины коэффициента усиления



б

Рис. 2.16. Полученные в вычислительном эксперименте и времени переходного процесса (*б*) от величины коэффициента усиления

Измеренные значения показателей присваиваются переменным `Tпп` и `Пpp`, зависимость которых от текущего коэффициента усиления `Ki` и выводится на фазовые диаграммы (рис. 2.16). Значения `NaN` и `Infinity` на графиках не отображаются.

Параметрическая оптимизация

Пример этого эксперимента находится в файле Эксперимент_4.mvb. Целью параметрической оптимизации является нахождение таких значений параметров оптимизации, при которых достигается минимум или максимум некоторого показателя, называемого функционалом. Каждая конкретная совокупность значений параметров оптимизации задает некоторую фазовую траекторию модели, в определенной точке которой, задаваемой некоторым условием, вычисляется значение функционала. Найдем, например, значение коэффициента усиления регулятора, при котором достигается минимум времени переходного процесса. В данном случае коэффициент усиления будет являться параметром оптимизации, время переходного процесса функционалом, а момент вычисления функционала будет задаваться логической функцией `Finalized`, которая возвращает значение `true`, когда функционирование модели завершилось.

Для проведения оптимизации модифицируем карту поведения модели, как показано на рис. 2.17. Наличие такой карты поведения в классе `Model` обусловлено необходимостью завершения работы модели. Альтернативным решением могло бы быть непосредственное использование экземпляра класса `CAP_Э` в качестве модели. Однако поведение этого объекта представляет собой суперпозицию поведений трех локальных объектов (см. рис. 2.11). Функционирование объекта со стереотипом "блок-схема" завершается только тогда, когда завершается функционирование всех его локальных объектов. В классе `CAP_Э` только объект `УИ` является гибридным и его функционирование завершается по завершении переходного процесса (см. рис. 2.12). Остальные объекты являются непрерывными и завершение функционирования для них не определено.

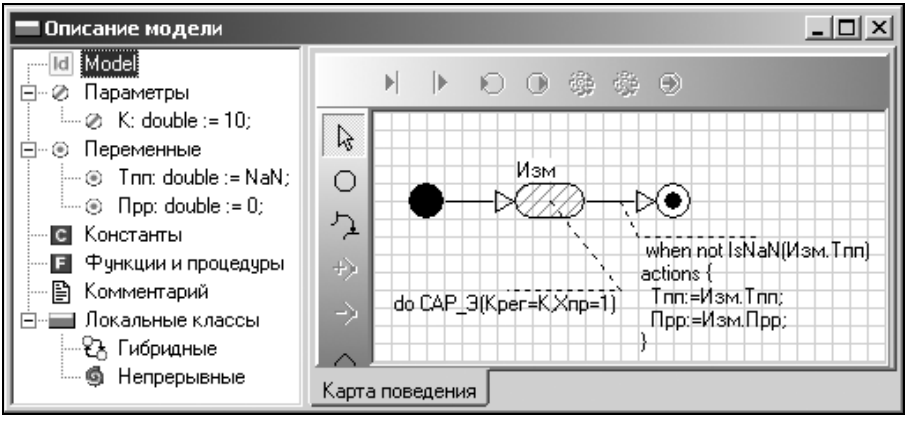


Рис. 2.17. Карта поведения для оптимизации

С помощью команды **Сервис | Оптимизация** вызовем подсистему оптимизации, выберем в ней необходимые параметры оптимизации и функционал и выполним минимизацию функционала (рис. 2.18).

Чтобы получить фазовую траекторию модели, соответствующую найденному оптимальному значению параметра `K`, создадим и запустим визуальную модель (подсистему оптимизации при этом можно не закрывать). Двойным щелчком мыши на окне переменных модели вызовем диалог ввода нового значения, введем оптимальное значение параметра и получим график переходного процесса (рис. 2.19).

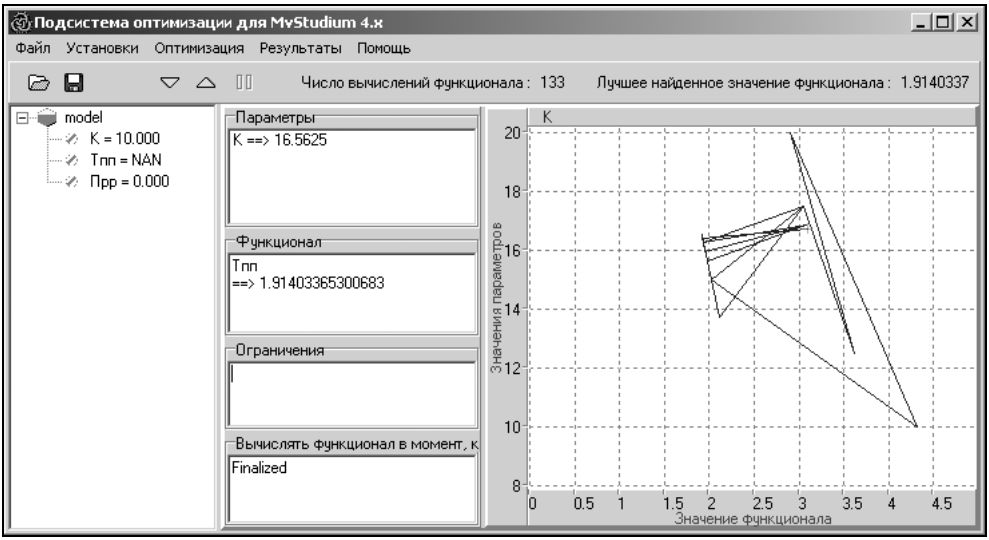
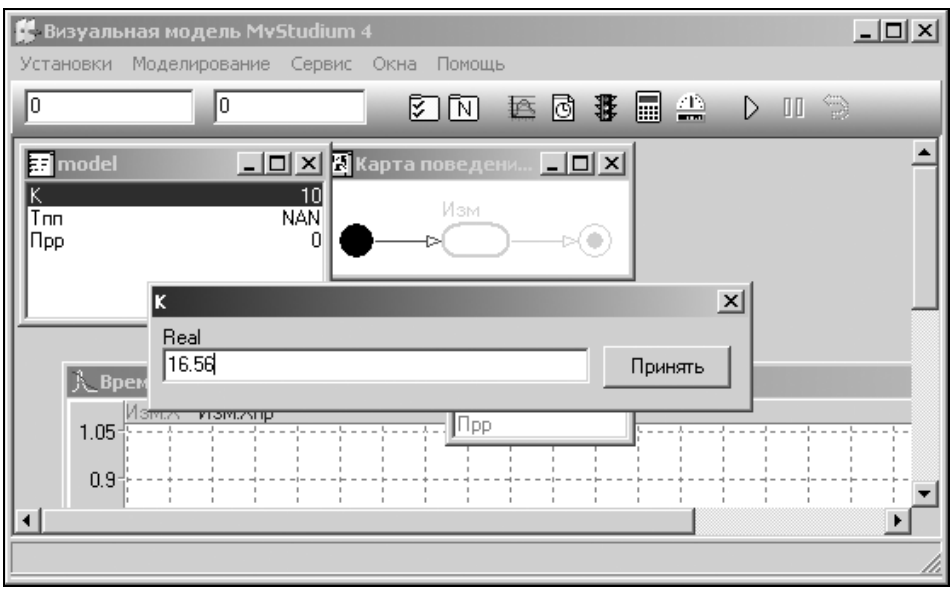
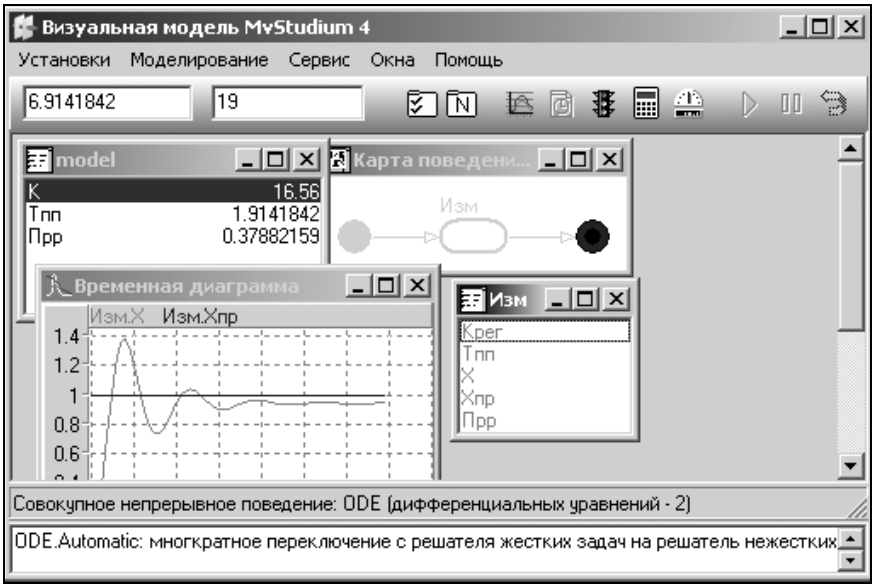


Рис. 2.18. Минимизация времени переходного процесса



а

Рис. 2.19. Фазовая траектория модели, соответствующая оптимальному значению параметра: *а* — интерактивное изменение значения параметра



б

Рис. 2.19. Фазовая траектория модели,
соответствующая оптимальному значению параметра:
б — график переходного процесса для оптимального значения

Следует отметить, что интерактивное изменение значения параметра `K` имеет смысл только до входа в состояние `Изм`, в противном случае экземпляр класса `САР_Э`, являющийся деятельностью в этом состоянии (см. рис. 2.17), будет создан со старым значением. Для этого необходимо снять опцию **Выполнять начальные дискретные действия** в установках модели. Альтернативой является указание действительного значения параметра модели в интегрированной среде с помощью команды **Модель | Действительные значения параметров и переменных модели** главного меню или кнопки на инструментальной панели.

Пусть теперь мы хотим определить значение коэффициента усиления, при котором время переходного процесса равно 3 секундам. Для этого нужно использовать в качестве функционала величину отклонения времени переходного процесса от требуемого значения $abs(T_{пп} - 3)$ или $(T_{пп} - 3)^2$. Второй вариант предпочтительнее, т. к. задает гладкую функцию. Результаты минимизации этого функционала показаны на рис. 2.20.

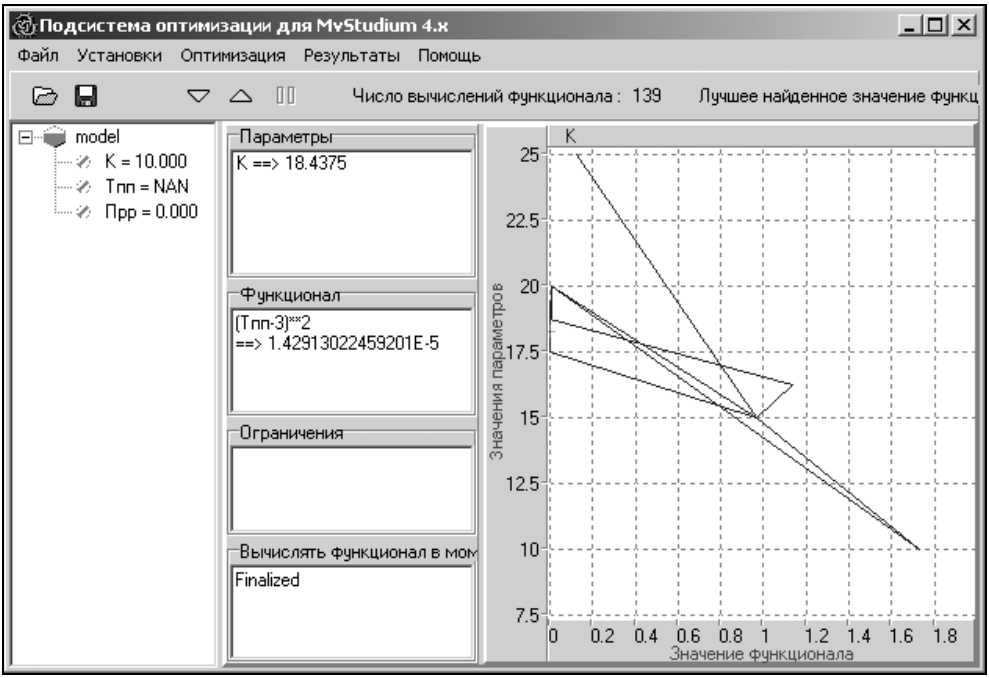


Рис. 2.20. Минимизация отклонения времени переходного процесса от 3 секунд

Сравнение результатов моделирования с внешними данными

Иногда возникает необходимость сопоставить результаты моделирования с аналогичными результатами, полученными другим способом (экспериментально или с помощью другой модели). Вернемся к линейке моделей снаряда, рассмотренной в *главе 1*. В модели `Снаряд_2` не учитывается зависимость параметров атмосферы и ускорения силы тяжести от высоты, а в модели `Снаряд_3` учитывается. Воспользуемся моделью `Снаряд_3` как эталонной для оценки границ адекватности модели `Снаряд_2`.

Сначала получим "эталонные" зависимости $x(t)$, $y(t)$ для значений начальной скорости 200, 150 и 100 м/с. Эти зависимости могут быть получены также путем траекторных измерений. Откроем проект Снаряд_3.mvb и запустим визуальную модель. Создадим новую временную диаграмму для переменных `x` и `y` (можно использовать и уже имеющуюся фазовую диаграмму, но на ней

выводится слишком много переменных). Значение начальной скорости по умолчанию равно 200 м/с. Запустим эксперимент и остановимся в точке падения. Нажмем правую кнопку мыши на временной диаграмме и выполним команду **Экспорт** всплывающего меню. В появившемся диалоге укажем файл V_200.txt и нажмем кнопку **Экспорт**. В указанном файле сохранится текстовая таблица с тремя колонками: **Время**, **x**, **y** (см. файл ...\Модели\Эксперименты\V_200.txt). Далее с помощью команды **Моделирование | Рестарт** или кнопки на инструментальной панели возвращаем модель в начальное состояние и меняем значение параметра `V0` на 150. Снова прогоняем модель до точки падения и сохраняем диаграмму в файл V_150.txt. То же самое проделываем для значения 100 и получаем файл V_100.txt. Все три файла переносим в папку ...\Модели\Эксперименты. Создадим новый проект ...\Модели\Эксперименты\Эксперимент_5.mvb со стереотипом "гибридный составной объект".

Мы будем сопоставлять поведение модели снаряда с эталонным поведением путем непосредственного изображения текущей и эталонной траекторий на одной фазовой диаграмме. Для этого нам нужно будет иметь два параллельно функционирующих процесса:

- модель снаряда;
- непрерывный процесс, вычисляющий эталонное значение вертикальной координаты для текущего значения горизонтальной.

Первый процесс реализуем как локальный объект `MC` в структурной схеме класса `Model`, а второй — как элемент собственного поведения класса `Model`.

Для получения объекта `MC` достаточно импортировать пакет `Снаряд_2` и создать экземпляр класса `Снаряд_2.Model`. Однако модель снаряда в проекте `Снаряд_2` является изолированной системой, в то время как для нашего эксперимента переменные x, y а также $\frac{dy}{dt}$ (для определения момента падения) должны быть видимы извне. Поэтому мы создадим новый класс `Модель_снаряда` как производный от базового класса `Снаряд_2.Model` (рис. 2.21).

Теперь изменим стереотип нового класса на **Открытая система** с помощью команды **Преобразовать в ...** всплывающего меню. После этого добавим новые выходные переменные `X`, `Y` и `Vy`, а также добавим в унаследованную систему уравнения три формулы, связывающие новые выходы с x, y и $\frac{dy}{dt}$ (рис. 2.22).

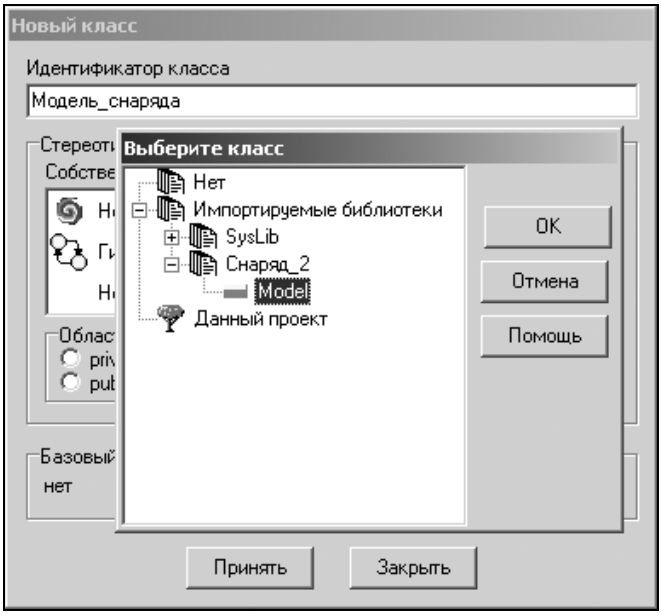


Рис. 2.21. Создание нового класса на основе существующей модели

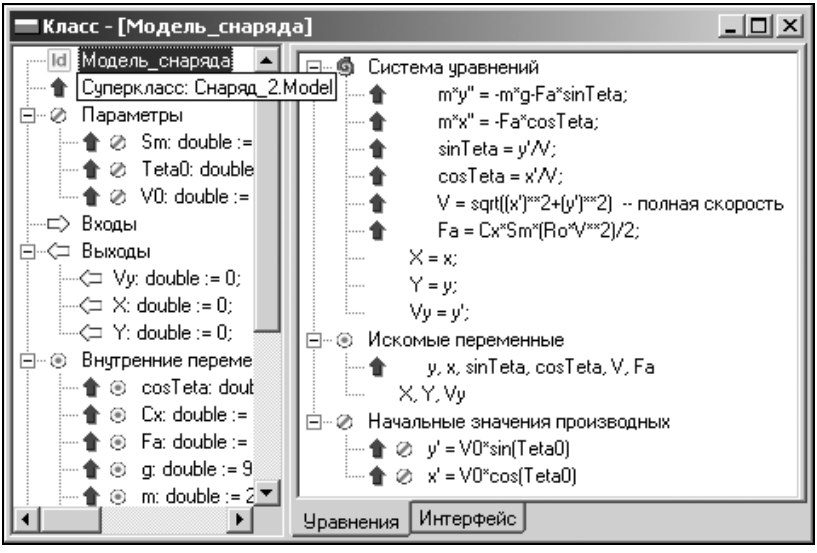


Рис. 2.22. Производный класс,
позволяющий использовать модель снаряда как компонент

Поместим экземпляр этого класса в структурную схему модели (рис. 2.23).

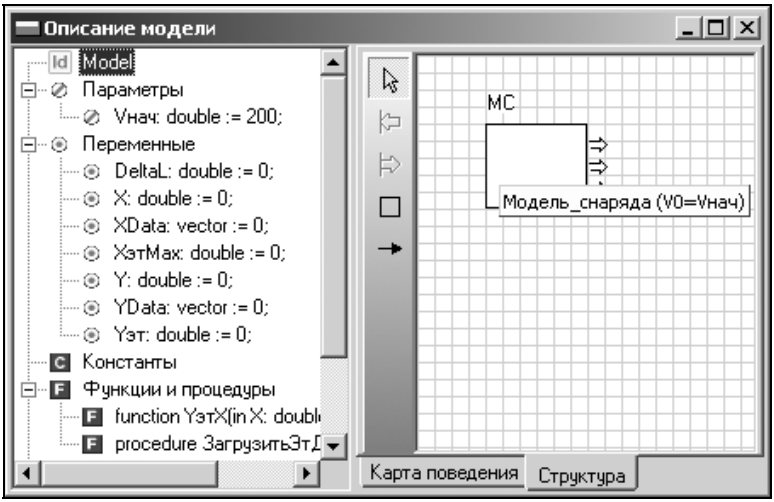


Рис. 2.23. Модель снаряда — локальный объект `MC`

Теперь рассмотрим формирование эталонной траектории. Эталонные значения координат загружаются из соответствующего файла в процедуре `ЗагрузитьЭтДанные`. Имя файла вычисляется как

```
p: string := "V_"+tostring(round(Vнач))+".txt";
```

Алгоритм получения эталонной траектории очевиден: для текущего значения x с помощью функции интерполяции определяется $y_{эт}$. Однако в общем случае текущая и эталонная траектории определены на различных диапазонах значений горизонтальной координаты. При завершении эталонной траектории раньше текущей на всем дальнейшем интервале $x > x_{эт_{max}}$ необходимо использовать последнее значение $y_{эт}$.

```
function YэтX(in X: double) return double is
  res: double;
begin
  if X>XэтMax then
    res:=YData[size(YData)];
```

```
  else
    res:=interpol(XData,YData,X);
  end if;
  return res;
end YэтX;
```

При завершении текущей траектории раньше эталонной необходимо продвигать значение горизонтальной координаты, чтобы достроить завершающий участок эталонной траектории. Это достигается с помощью карты поведения, показанной на рис. 2.24.

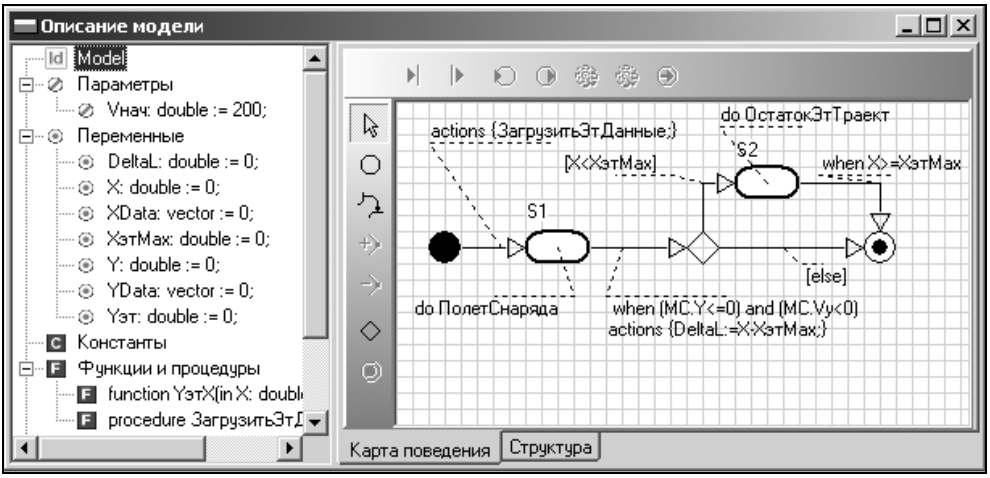


Рис. 2.24. Карта поведения для построения текущей и эталонной траекторий

В состоянии `S1` вычисляется совокупность формул (локальный класс `ПолетСнаряда`):

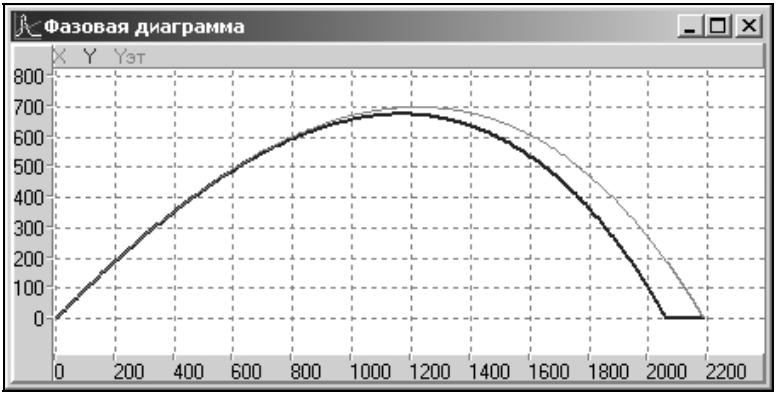
```
X = MC.X
Y = MC.Y
Yэт = YэтX(X)
```

В состоянии `S2` вычисляется другая совокупность формул (локальный класс `ОстатокЭтТраект`):

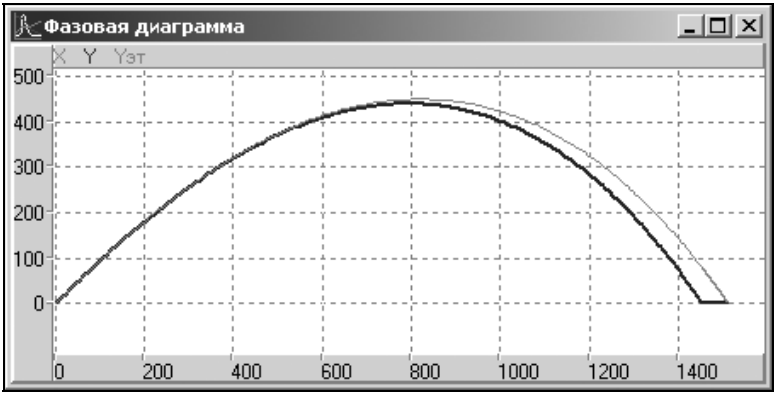
```
X' = 1
Yэт = YэтX(X)
```

В случае, когда эталонная траектория заканчивается раньше текущей, происходит переход в конечное состояние, минуя состояние `S2` (рис. 2.24).

Проведем вычислительный эксперимент для трех значений начальной скорости `Vнач`. В данной модели значения этого параметра необходимо изменять в интегрированной среде с помощью команды **Модель | Действительные значения параметров и переменных модели** главного меню или кнопки на инструментальной панели.

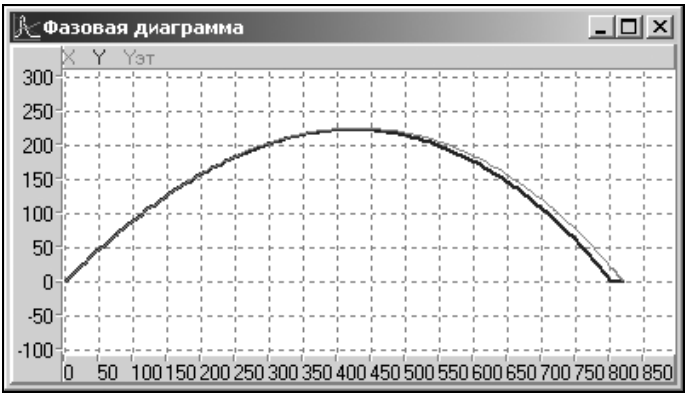


а



б

Рис. 2.25. Результаты вычислительного эксперимента (жирной линией показана текущая траектория): *а* — для `Vнач`=200 м/с; *б* — для `Vнач`=150 м/с



в

Рис. 2.25. Результаты вычислительного эксперимента (жирной линией показана текущая траектория): *в* — для `Vнач`=100 м/с

Интерактивное изменение параметра `Vнач` непосредственно в визуальной модели невозможно, т. к. к моменту открытия главного окна модели статический экземпляр локального объекта `MC` уже создан с тем значением параметра `Vнач`, который определен на момент старта модели. На рис. 2.25 показаны результаты вычислительных экспериментов, из которых следует, что упрощенной моделью снаряда можно пользоваться только в диапазоне начальных скоростей, меньших 100 м/с.

Проведение статистических испытаний

Метод статистических испытаний предусматривает большое число испытаний (вычислений фазовой траектории исследуемой системы) при различных сочетаниях случайных значений параметров с целью определения закона распределения и числовых характеристик этого распределения для ряда показателей работы исследуемой системы.

Пусть, например, начальная скорость снаряда V_0 является случайной величиной, распределенной по нормальному закону с математическим ожиданием $M[V_0] = 200$ м/с и среднеквадратическим отклонением $\sigma(V_0) = 2$ м/с, а начальный угол θ_0 также распределен по нормальному закону с математическим ожиданием $M[\theta_0] = 45°$ и среднеквадратическим отклонением

$\sigma[\theta_0] = 1.5°$. Необходимо определить закон распределения дальности точки падения L и его числовые характеристики.

Общая схема вычислительного эксперимента для проведения статистических испытаний практически совпадает со схемой вычислительного эксперимента для получения параметрических зависимостей: n раз последовательно создается экземпляр изучаемого объекта с нужными значениями параметров. Отличие состоит в том, что значения параметров разыгрываются случайным образом.

Создадим новый проект ...\Модели\Эксперименты\Эксперимент_6.mvb со стереотипом "гибридный элементарный объект".

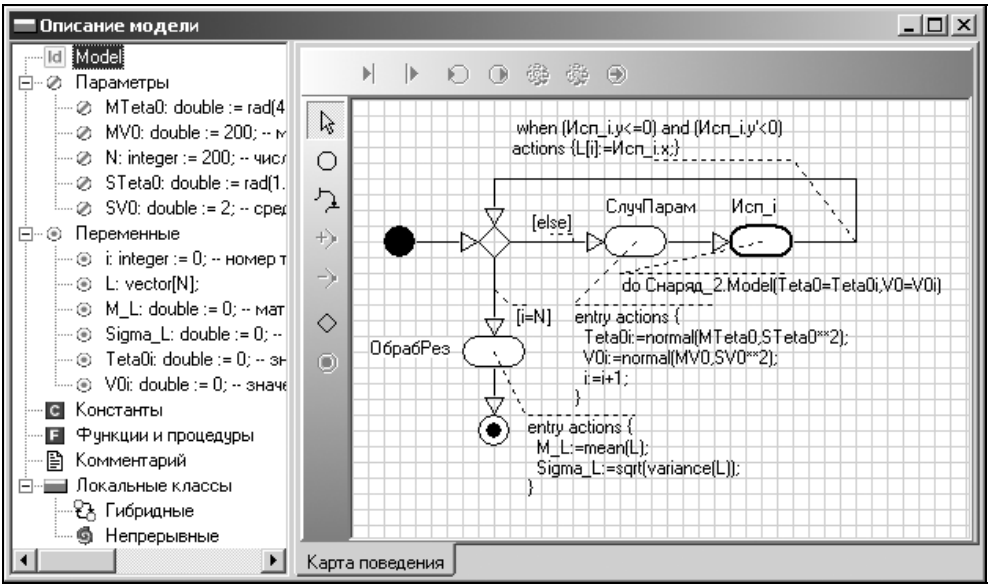


Рис. 2.26. Карта поведения
для проведения статистических испытаний с моделью снаряда

На рис. 2.26 показана карта поведения модели для проведения статистических испытаний с моделью снаряда. В качестве модели снаряда снова используем класс `Model` из проекта `Снаряд_2`, для чего импортируем пакет `Снаряд_2`. В состоянии `СлучПарам` разыгрываются соответствующие `i`-му испытанию значения начальной скорости `V0i` и начального угла `Teta0i`, а также увеличивается на единицу номер текущего испытания `i`. Значения ра-

зыгрываются с помощью встроенного генератора случайных величин, распределенных по нормальному закону. В состоянии `Исп_i` создается экземпляр модели снаряда со случайными значениями начальной скорости и начального угла и проводится моделирование движения снаряда до точки падения. В действиях исходящего перехода вычисленное значение дальности заносится в `i`-й элемент вектора результатов `L`. Экземпляр модели снаряда уничтожается, разыгрываются новые случайные значения параметров и так далее `N` раз. После проведения `N`-го испытания в состоянии `ОбрабРез` вычисляются значения математического ожидания (как среднее значение) и среднеквадратического отклонения дальности точки падения. Для указанных выше параметров законов и `N=200` распределения получаем $M[L] = 2060$ м и $\sigma[L] = 45$ м.

Попробуем теперь построить закон распределения дальности точки падения. Для этого во входные действия состояния `ОбрабРез` добавим обращение к функции создания гистограммы (рис. 2.27). Аргументами этой функции являются вектор результатов испытаний `X` и число интервалов `m`. Результатом является матрица размерности (`m` + 1)×2, в первом столбце которой находятся координаты интервалов, а во втором вероятности попадания в этот интервал:

```
function Гистограмма(in X: vector; in m: integer) return vector is
  res: matrix := zeros(m+1,2);
  n: integer := size(X);
  xmin: double := trunc(min(X));
  xmax: double := trunc(max(X))+1;
  XI: vector := [ for i in 1..m+1 | xmin+(i-1)*(xmax-xmin)/m ];
  F: vector := zeros(m);
  x: double;
begin
  for i in 1..m loop
    for j in 1..n loop
      x:=X[j];
      if x=XI[i] then
        F[i]:=F[i]+0.5/n;
      elsif (x>XI[i]) and (x<XI[i+1]) then
        F[i]:=F[i]+1/n;
      end if;
    end loop;
  end loop;
```

```
  for i in 1..m+1 loop
    res[i,1]:=XI[i];
  end loop;
  for i in 1..m loop
    res[i,2]:=F[i];
  end loop;
  return res;
end Гистограмма;
```

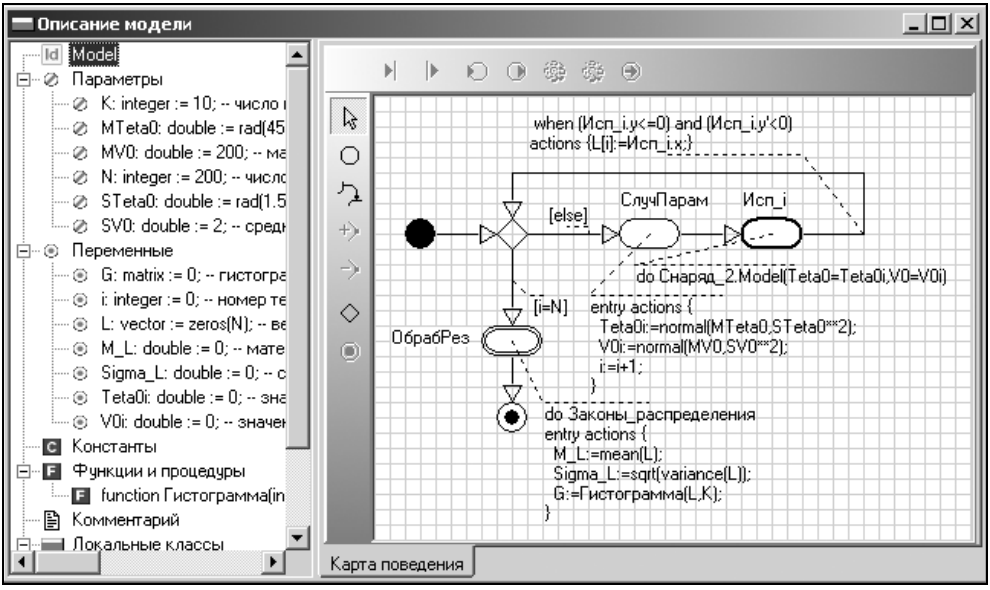


Рис. 2.27. Модифицированная карта поведения статистических испытаний

Из физических соображений мы можем предположить, что дальность точки падения будет распределена по нормальному закону. Построим на одном графике зависимости плотности вероятности от дальности для нормального закона с полученными в результате эксперимента значениями математического ожидания и среднеквадратического отклонения и для полученной гистограммы. Для этого состоянию `ОпредРез` припишем в качестве деятельности экземпляр гибридного класса `Законы_распределения` (рис. 2.28). Карта поведения этого класса обеспечивает продвижение значения переменной `x` по всем интервалам гистограммы из расчета 10 точек на интервал и вычисление теоретической плотности вероятности `fн` для нормального распределения и

экспериментальной `fэ`. Значение `fэ`, вычисляемое с помощью функции `fэ_x`, равно вероятности для текущего интервала, деленной на длину интервала.

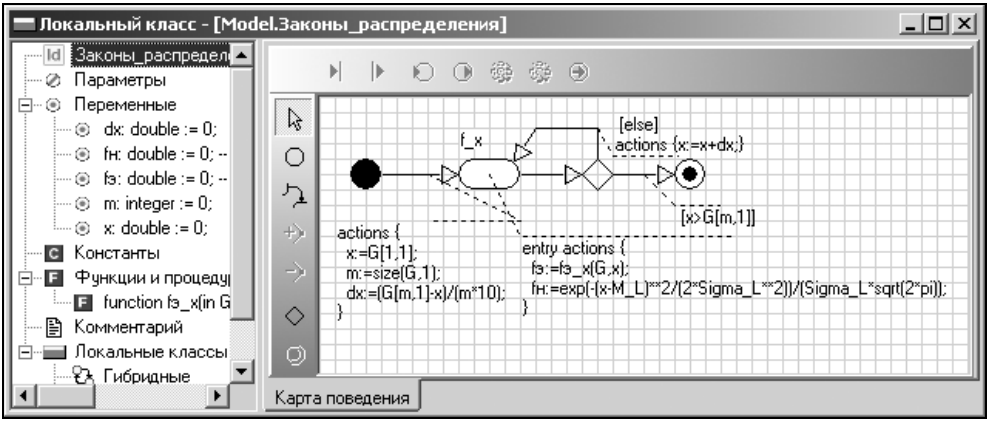
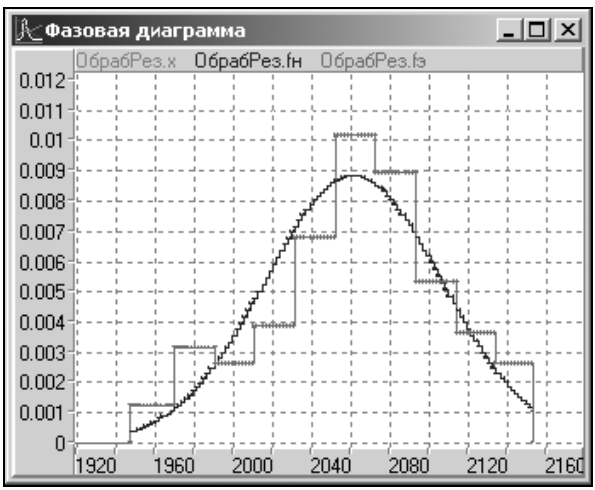


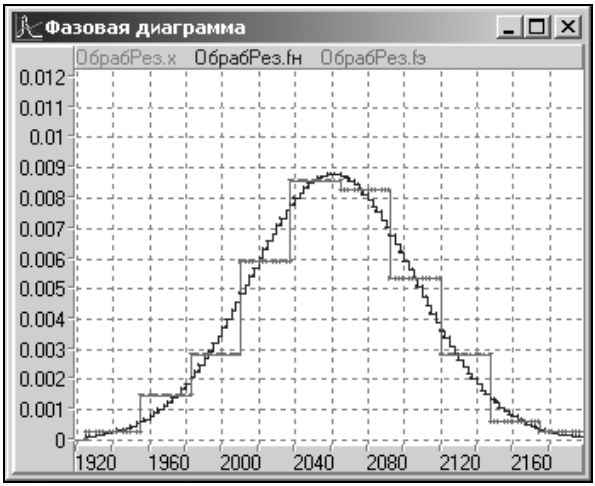
Рис. 2.28. Локальный класс `Законы_распределения`

В результате проведения вычислительного эксперимента получаем графики плотности распределения, показанные на рис. 2.29, *а*.



а

Рис. 2.29. Теоретические и экспериментальные зависимости плотности распределения, полученные для N=200 (*а*)



б

Рис. 2.29. Теоретические и экспериментальные зависимости плотности распределения, полученные для N=1000 (*б*)

Проведя тот же эксперимент для 1000 испытаний, получим зависимости, показанные на рис. 2.29, *б*. Значения математического ожидания и среднеквадратического отклонения в обоих экспериментах практически совпадают.

Самостоятельно модифицируйте модель так, чтобы количественно оценить согласованность результатов с нормальным законом распределения, например, по критерию Пирсона.

Глава 3

Учебные задания

В этой главе приведены учебные задания, которые можно использовать как при обучении работе с математическими пакетами MATLAB, Maple, Mathematica, так и при изучении курсов "Основы моделирования", "Компьютерные вычисления", если для выполнения лабораторных работ используются пакеты моделирования. Задачи классифицированы с точки зрения различных пользовательских интерфейсов пакета MvStudium.

Динамические системы

Интерфейс "Непрерывный элементарный объект".

Задание 1

1. Решить численно уравнения:

$$\frac{dx}{dt} = -x + (a_2 \cdot t^2 + a_1 \cdot t + a_0); \quad t \in [-T, T], \quad x(0) = x_0, \tag{3.1}$$

$$\frac{dy}{dt} = -y + (a_2 \cdot \cos(t) + a_1 \cdot \sin(t) + a_0); \; t \in [-T, T], \; y(0) = y_0. \tag{3.2}$$

Коэффициенты уравнений для конкретного варианта приведены в табл. 3.1.

Таблица 3.1. *Коэффициенты уравнений*

N	a_2	a_1	a_0	T	x_0	Уравнение
1	1	0	0	1	–6	3.1
2	0	1	0	2	–5	3.1
3	1	1	0	3	–4	3.1
4	0	1	1	4	–3	3.1
5	1	0	1	1	–2	3.1
6	1	1	1	2	–1	3.1
7	1	0	0	3	1	3.2
8	0	1	0	4	2	3.2
9	1	1	0	1	3	3.2
10	0	1	1	2	4	3.2
11	1	0	1	3	5	3.2
12	1	1	1	4	6	3.2

Нарисовать графики построенных решений.

2. Найти решение заданных уравнений:

$$\frac{dx}{dt} + f(t) \cdot x = g(t);$$
$$x(0) = x_0;$$

- используя формулу

$$F(t) = \int_0^T f(t)dt;$$

$$x(t) = e^{-F(t)} \cdot \left(x_0 + \int_0^T g(t) \cdot e^{F(t)} dt \right);$$

- как сумму решений — решения однородного уравнения и частного решения неоднородного уравнения.

3. Построить график относительной погрешности численного решения, сравнивая численное решение с найденным в явном виде решением.

Указания

Прежде всего, рекомендуется найти решение вручную, а затем используя математический пакет, например, Maple. Найденные решения — это тесты для пакетов моделирования.

Задание 2

1. Построить графики правых частей дифференциальных уравнений из табл. 3.2 как функций от x и пометить на них особые точки на промежутке $[-2\pi; 2\pi]$. Определить, какие из них устойчивы, а какие — нет. Построить фазовый портрет.

Таблица 3.2. *Уравнения для построения графиков*

№ п/п	Уравнение	№ п/п	Уравнение	№ п/п	Уравнение
1.	$\frac{dx}{dt} = x^3 - x$	5.	$\frac{dx}{dt} = \sin x - x^2$	9.	$\frac{dx}{dt} = x^3 - \sin x$
2.	$\frac{dx}{dt} = \frac{1}{2}x^2 + 2x - 1$	6.	$\frac{dx}{dt} = e^{-x} - \sin x$	10.	$\frac{dx}{dt} = e^{-x} + x^3$
3.	$\frac{dx}{dt} = \sin x$	7.	$\frac{dx}{dt} = e^{-x} + \cos x$	11.	$\frac{dx}{dt} = e^{-x} - x^2 + 2$
4.	$\frac{dx}{dt} = \sin x + x^2$	8.	$\frac{dx}{dt} = x^4 - x^2$	12.	$\frac{dx}{dt} = \cos x + x^2$

2. Написать программу поиска корней функции одной переменной на языке MVL. На первом этапе использовать глобальные методы поиска: деления отрезка пополам и метод золотого сечения, на втором этапе — локальные методы:

   - метод ложного положения

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_0}{f(x_n) - f(x_0)};$$

- метод секущих

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})};$$

- метод Ньютона

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Учебные программы поиска корней функции одной переменной строятся в соответствии с табл. 3.3.

Таблица 3.3. *Сочетание глобальных и локальных методов в учебной программе*

Глобальный метод	**Локальный метод**
Метод деления отрезка пополам	Метод ложного положения
Метод деления отрезка пополам	Метод секущих
Метод деления отрезка пополам	Метод Ньютона
Метод золотого сечения	Метод ложного положения
Метод золотого сечения	Метод секущих
Метод золотого сечения	Метод Ньютона

Сравнить найденные значения со значениями, полученными с помощью встроенной программы пакета MvStudium поиска корней вещественной функции одной переменной.

Задание 3

Построить решение и численное решение системы второго порядка

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

нарисовать их, используя пакет MvStudium, в виде временной и фазовой диаграммы, предварительно построив матрицу $\mathbf{A}$ по формулам:

$$\mathbf{A} = \mathbf{S} \cdot \begin{bmatrix} \lambda & 0 \\ 0 & \mu \end{bmatrix} \cdot \mathbf{S}^{-1}; \quad \mathbf{A} = \mathbf{S} \cdot \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix} \cdot \mathbf{S}^{-1}; \quad \mathbf{A} = \mathbf{S} \cdot \begin{bmatrix} \mu & \nu \\ -\nu & \mu \end{bmatrix} \cdot \mathbf{S}^{-1}.$$

В качестве матриц $\mathbf{S}$ использовать:

а) элементарную матрицу $\mathbf{S} = \mathbf{E} + \mathbf{u} \cdot \mathbf{v}^T$; $\mathbf{S}^{-1} = \mathbf{E} - \mathbf{u} \cdot \mathbf{v}^T$, $(\mathbf{u}, \mathbf{v}) = 0$;

б) элементарную матрицу $\mathbf{S} = \mathbf{E} + \mathbf{u} \cdot \mathbf{v}^T$; $\mathbf{S}^{-1} = \mathbf{E} - \dfrac{1}{1+d}\mathbf{u} \cdot \mathbf{v}^T$, $d = (\mathbf{u}, \mathbf{v})$;

в) матрицу отражения $\mathbf{H} = \mathbf{E} - \alpha \cdot \mathbf{u} \cdot \mathbf{u}^T$; $\alpha = \dfrac{2}{\mathbf{u} \cdot \mathbf{u}^T}$; $\mathbf{H}^{-1} = \mathbf{H}$;

г) матрицу плоских вращений Гивенса

$$\mathbf{G}(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}; \quad \mathbf{G}^{-1}(\alpha) = \mathbf{G}(-\alpha).$$

Собственные числа матрицы $\mathbf{A}$ выбирать так, как указано в табл. 3.4.

Таблица 3.4. *Тип особой точки и вид преобразования*

Вариант	**Тип особой точки**	**Преобразование подобия**
1	Узел устойчивый	Элементарная матрица *а*
2	Узел неустойчивый	Элементарная матрица *б*
3	Звездный узел устойчивый	Матрица отражения
4	Звездный узел неустойчивый	Матрица вращения
5	Вырожденный узел устойчивый	Элементарная матрица *а*
6	Вырожденный узел неустойчивый	Элементарная матрица *б*
7	Седло	Матрица отражения
8	Центр	Матрица вращения
9	Устойчивый фокус	Элементарная матрица *а*
10	Неустойчивый фокус	Элементарная матрица *б*
11	Узел устойчивый	Матрица отражения
12	Узел устойчивый	Матрица вращения

Задание 4

Решение систем нелинейных алгебраических уравнений с заранее известным решением.

В основу всех приведенных далее уравнений положены тождества [20]. Например, рассмотрим тождество

$$\begin{cases} a_{11}x_1 + a_{12}x_2 \equiv a_{11}x_1 + a_{12}x_2; \\ a_{21}x_1 + a_{22}x_2 \equiv a_{21}x_1 + a_{22}x_2. \end{cases}$$

Выберем $x_1 = 1$; $x_2 = \sin(t)$, $t \in [0, T]$, $T = 10$, подставим эти значения в правую часть и получим систему уравнений с известным решением

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = a_{11} + a_{12} \cdot \sin(t); \\ a_{21}x_1 + a_{22}x_2 = a_{21} + a_{22} \cdot \sin(t). \end{cases}$$

В зависимости от коэффициентов a_{ij} эту систему можно сделать хорошо обусловленной, плохо обусловленной, не имеющей или имеющей бесконечно много решений.

Пользуясь этим приемом, постройте тестовые системы, используя тождества, приведенные далее. Для этого выберите для заданного n произвольные значения x_1, x_2, ..., x_{n-1}. Задайте x_n как произвольную функцию от времени $x_n = x_n(t)$. Там где это необходимо, сформируйте правые части приведенных далее уравнений, так чтобы вектор $\mathbf{x}$ стал решением системы. Найдите численное решение $\tilde{\mathbf{x}}$ построенной системы нелинейных алгебраических уравнений и сравните его с известным $\mathbf{x}$. Напишите процедуру вычисления матрицы Якоби на языке MVL по формулам, приведенным для каждого тестового примера.

1. Система уравнений

$$\begin{cases} x_1 + x_2 + \ldots . x_n \equiv x_1 + x_2 + \ldots . x_n = a_1; \\ x_1 \cdot x_2 + x_1 \cdot x_3 + \ldots x_{n-1} \cdot x_n \equiv x_1 \cdot x_2 + x_1 \cdot x_3 + \ldots x_{n-1} \cdot x_n = a_2; \\ \ldots . \\ x_1 \cdot x_2 \cdot x_3 \ldots x_{n-1} \cdot x_n \equiv x_1 \cdot x_2 \cdot x_3 \ldots x_{n-1} \cdot x_n = a_n \end{cases}$$

с матрицей Якоби

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ \sum_{i=2}^{n} x_i & \sum_{i\neq 2}^{n} x_i & & & \sum_{i=1}^{n-1} x_i \\ \dots\dots & & & & \\ P_1 & & \dots\dots & & P_n \end{bmatrix}$$

$$P_1 = \prod_{i=2,N} x_i; P_j = \prod_{\mathrm{i}\neq \mathrm{j}} x_i, \quad P_n = \prod_{i=1,N-1} x_i \ .$$

Найдите условия, обеспечивающие численную вырожденность матрицы Якоби на решении. Одновременно решая систему и вычисляя матрицу Якоби, убедитесь, что трудности с построением численного решения возникают в точках, где матрица вырождена. Обратите внимание на то, что система также обладает еще одним важным свойством — любая перестановка компонентов вектора $\mathbf{x}$ вновь является решением. Как это может отразиться на построении численного решения?

3. Для построения системы вместо симметрических многочленов

$$S_1 = \sum_i x_i; \ S_2 = \sum_{i,j} x_i x_j; \ \dots \ ; S_n = \prod_i x_i$$

используйте симметрические функции $s_1, \dots, s_n$

$$s_1 = \sum_i x_i; s_2 = \sum_i x_i^2; \ \dots \ s_n = \sum_i x_i^n$$

и решайте систему

$$\begin{cases} s_1 = a_1, \\ \dots, \\ s_n = a_n. \end{cases}$$

Матрица Якоби в этом случае равна

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 2x_1 & 2x_2 & \dots & \dots & 2x_n \\ \dots & & & & \\ nx_1^{n-1} & \dots & \dots & \dots & nx_n^{n-1} \end{bmatrix}.$$

Данная форма хороша тем, что значения сумм $\sum_{k=1}^{n} k^q = SUM_q$ известны для всех значений q, а до $q=5$ включительно они имеют чрезвычайно простой вид. Таким образом, если положить $x_i = i$, $i = 1, ..., n$, то легко сформировать правые части уравнений. Действительно

$$\sum_{k=1}^{n} k = \frac{n\cdot(n+1)}{2}; \quad \sum_{k=1}^{n} k^2 = \frac{n\cdot(n+1)\cdot(2n+1)}{6}; \quad \sum_{k=1}^{n} k^3 = [\frac{n\cdot(n+1)}{2}]^2$$

$$\sum_{k=1}^{n} k^4 = \frac{n\cdot(n+1)\cdot(2n+1)(3n^2+3n-1)}{30}; \quad \sum_{k=1}^{n} k^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}.$$

Если воспользоваться любым математическим пакетом, то можно получить нужные значения для произвольного q.

4. Система шестого или седьмого порядка может быть построена на базе следующих явных формул для приведенных далее сумм:

$$\sum_{i=1}^{n} \frac{1}{(p+(k-1)q))(p+kq)} = \frac{n}{p(p+nq)},$$

$$\sum_{i=1}^{n} \frac{1}{(p+(k-1)q))(p+kq)(p+(k+1)q)} = \frac{n(2p+nq+q)}{2p(p+q)(p+nq)(p+(n+1)q)},$$

$$\sum_{i=1}^{n} \frac{1}{(p+(k-1)q))(p+kq)...(p+(k+L)q)} = A,$$

$$A = \frac{1}{(L+1)q}\left\{\frac{1}{p(p+q)...(p+Lq)} - \frac{1}{p(p+nq)(p+(n+1)q)...(p+(n+L)q)}\right\},$$

$$\sum_{i=1}^{n} \frac{1}{(1+(k-1)q))(1+(k-1)q+p)} = \frac{1}{p}\left(\sum_{i=1}^{n} \frac{1}{(1+(k-1)q))} - \sum_{i=1}^{n} \frac{1}{(1+(k-1)q)+p)}\right).$$

Переписывая левые части в виде уравнений относительно неизвестных

$$x_i = i, \; i = 1, ..., 4$$

и вычисляя правые части, получим четыре уравнения. Пока еще здесь L — любое целое, p, q — любые числа.

Четыре полученных уравнения дополняются двумя линейными уравнениями относительно p и q (система шестого порядка):

$$\begin{cases} p+(1+\varepsilon)q=a_1, \\ (1+\varepsilon)p+q=a_2 \end{cases}$$

или любыми другими относительно p, q и L (система седьмого порядка).

5. Для построения системы использовать формулы:

$$\sum_{k=1}^{n}\left(x^k+k\right)\cdot\left(x^k-k\right)=\frac{x^2\cdot\left(x^{2n}-1\right)}{x^2-1}-\frac{1}{6}n\cdot(n+1)\cdot(2n+1);$$

$$\sum_{k=1}^{n}\left(x^k+y^k\right)\cdot\left(x^k-y^k\right)=\frac{x^2\cdot\left(x^{2n}-1\right)}{x^2-1}-\frac{y^2\cdot\left(y^{2n}-1\right)}{y^2-1};$$

$$\sum_{k=1}^{n}\left(x^k+\frac{1}{y^k}\right)\cdot\left(x^k-\frac{1}{y^k}\right)=\frac{x^2\cdot\left(x^{2n}-1\right)}{x^2-1}-\frac{y^2\cdot\left(y^{2n}-1\right)}{y^2-1};$$

$$\sum_{k=1}^{n}\left(x^{k+1}-y^{k+1}\right)=\frac{x^2\cdot\left(1-x^n\right)}{1-x}-\frac{y^2\cdot\left(1-y^n\right)}{1-y}.$$

Число уравнений в этом примере может быть от 4 и выше, если использовать и $x_i\equiv x^i$, x — любое число, и $y_i=y^i$, y — любое число, в качестве неизвестных и добавить необходимое число, например, линейных уравнений относительно y_i.

6. "Замешивание" в одной системе функций с сильно разнящимися производными позволяет построить достаточно трудные для численных методов проверочные примеры. Очередная группа вычисляемых в явном виде сумм позволяет построить систему:

$$\sum_{k=0}^{n}a^{kx}=\frac{a^{(n+1)x}-1}{a^x-1};\ \sin\frac{x}{2}\cdot\sum_{k=1}^{n}\sin kx=\sin\frac{n+1}{2}x\cdot\sin\frac{n}{2}x;$$

$$\sin x\cdot\sum_{k=0}^{n}\cos^2 kx=\frac{n-1}{2}\cdot\sin x+\frac{1}{2}\sin(n+1)x\cdot\cos nx;$$

$$\sin\frac{x}{2}\cdot\sum_{k=1}^{n-1}x\cdot k\cdot\sin kx=x\cdot\left(\sin nx-2n\cdot\cos\left(\frac{2n-1}{2}\right)x\cdot\sin\frac{x}{2}\right);$$

$$\sum_{k=0}^{n-1} p^k \operatorname{sh} kx = \frac{p \cdot \operatorname{sh} x - p^n \cdot \operatorname{sh} nx + p^{n+1} \cdot \operatorname{sh}(n-1)x}{1 - 2p \cdot \operatorname{ch} x + p^2};$$

$$\sum_{k=0}^{n-1} p^k \operatorname{ch} kx = \frac{1 - p \cdot \operatorname{ch} x - p^n \cdot \operatorname{ch} nx + p^{n+1} \cdot \operatorname{ch}(n-1)x}{1 - 2p \cdot \operatorname{ch} x + p^2}.$$

Как и в предыдущем случае, a, p — можно считать параметрами, а неизвестными — величины $x_i = i \cdot x$; $i = 1, ..., n$.

Матрица Якоби

$$\begin{bmatrix} a^{x_1} \ln a & a^{x_1} \cdot \ln a & \dots & a^{x_1} \cdot \ln a \\ \frac{1}{2}\cos\frac{x_1}{2} \cdot \sum_{k=1}^{n} \sin x_k + \sin\frac{x_1}{2} \cdot \cos x_1 & \sin\frac{x_1}{2} \cdot \cos x_2 & \dots & \sin\frac{x_1}{2} \cdot \cos x_n \\ \cos x_1 \cdot \left(\sum_{k=0}^{n} \cos^2 x_k - 2\sin^2 x_1\right) & -2\sin x_1 \cdot \sin x_2 \cdot \cos x_2 & \dots & -2\sin x_1 \cdot \sin x_n \cdot \cos x_n \\ a & \sin x_1 \cdot (\sin x_2 + x_2 \cdot \cos x_2) & \dots & \sin x_1 \cdot (\sin x_n + x_n \cdot \cos x_n) \\ b & (1 - 2p \cdot \operatorname{ch} x_1 + p^2) \cdot p^2 \cdot \operatorname{sh} x_2 & \dots & (1 - 2p\operatorname{ch} x_1 + p^2) \cdot p^2 \cdot \operatorname{sh} x_n \\ c & (1 - 2p \cdot \operatorname{ch} x_1 + p^2) \cdot p^2 \cdot \operatorname{ch} x_2 & \dots & (1 - 2p\operatorname{ch} x_1 + p^2) \cdot p^2 \cdot \operatorname{ch} x_n \end{bmatrix},$$

где:

$$a = 4\sin\frac{x_1}{2} \cdot \cos\frac{x_1}{2} \cdot \sum_{k=1}^{n} x_k \sin x_k + 4\sin^2\frac{x_1}{2} \cdot (\sin x_1 + x_1 \cos x_1),$$

$$b = -2p \cdot \operatorname{sh} x_1 \cdot \sum_{k=0}^{n-1} p^k \operatorname{ch} x_k + (1 - 2p\operatorname{ch} x_1 + p^2) \cdot \operatorname{sh} x_1,$$

$$c = -2p \cdot \operatorname{sh} x_1 \cdot \sum_{k=0}^{n-1} p^k \operatorname{sh} x_k + (1 - 2p\operatorname{ch} x_1 + p^2) \cdot \operatorname{ch} x_1.$$

7. Еще одна группа формул позволяет построить систему шестого порядка:

$$\sin x \cdot \sum_{k=1}^{n} \sin^2 kx = \frac{n}{2} \cdot \sin x - \cos(n+1)x \cdot \sin nx;$$

$$\sin x \cdot \sum_{k=1}^{n} \cos^2 kx = \frac{n}{2} \cdot \sin x + \cos(n+1)x \cdot \sin nx;$$

$$\sin\frac{x}{2} \cdot \sin\frac{3x}{2} \cdot \sum_{k=1}^{n} \sin^3 kx = \frac{3}{4} \cdot \sin\frac{n+1}{2}x \cdot \sin\frac{n}{2}x \cdot \sin\frac{3x}{2} -$$

$$-\frac{1}{4} \cdot \sin\frac{3(n+1)}{2}x \cdot \sin\frac{3n}{2}x \cdot \sin\frac{x}{2};$$

$$\sin\frac{x}{2}\cdot\sin\frac{3x}{2}\cdot\sum_{k=1}^{n}\cos^3 kx=\frac{3}{4}\cdot\cos\frac{n+1}{2}x\cdot\sin\frac{n}{2}x\cdot\sin\frac{3x}{2}+$$

$$+\frac{1}{4}\cdot\cos\frac{3(n+1)}{2}x\cdot\sin\frac{3n}{2}x\cdot\sin\frac{x}{2};$$

$$\sin x\cdot\sin 2x\cdot\sum_{k=1}^{n}\sin^4 kx=\frac{1}{8}\left(3n-\cos(n+1)x\cdot\sin nx\cdot\sin 2x\right)+$$

$$+\frac{1}{8}\cdot\cos 2(n+1)x\cdot\sin 2nx\cdot\sin x;$$

$$\sin x\cdot\sin 2x\cdot\sum_{k=1}^{n}\cos^4 kx=\frac{1}{8}\left(3n+\cos(n+1)x\cdot\sin nx\cdot\sin 2x\right)+$$

$$+\frac{1}{8}\cdot\cos 2(n+1)x\cdot\sin 2nx\cdot\sin x.$$

Относительно неизвестных $x_i=i\cdot x$; $i=1,\ldots,n$, x — любое.

Матрица Якоби

$$\begin{bmatrix} a & 2\sin x_1\cdot\cos x_2\cdot\sin x_2 & \ldots & 2\sin x_1\cdot\cos x_n\cdot\sin x_n \\ b & -2\sin x_1\cdot\cos x_2\cdot\sin x_2 & \ldots & -2\sin x_1\cdot\cos x_n\cdot\sin x_n \\ c & 3\sin\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}\cdot\sin^2 x_2\cdot\cos x_2 & \ldots & 3\sin\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}\cdot\sin^2 x_n\cdot\cos x_n \\ d & -3\sin\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}\cdot\cos^2 x_2\cdot\sin x_2 & \ldots & -3\sin\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}\cdot\cos^2 x_n\cdot\sin x_n \\ e & 4\sin x_1\cdot\sin 2x_1\cdot\sin^2 x_2\cdot\cos x_2 & \ldots & 4\sin x_1\cdot\sin 2x_1\cdot\sin^2 x_n\cdot\cos x_n \\ f & -4\sin x_1\cdot\sin 2x_1\cdot\cos^2 x_2\cdot\sin x_2 & \ldots & -4\sin x_1\cdot\sin 2x_1\cdot\cos^2 x_n\cdot\sin x_n \end{bmatrix},$$

где:

$$a=\cos x_1\left(\sum_{k=1}^{n}\sin^2 x_k+2\sin x_1\right);$$

$$b=\cos x_1\left(\sum_{k=1}^{n}\sin^2 x_k-2\sin x_1\right);$$

$$c=\left(\frac{1}{2}\cos\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}+\frac{3}{2}\sin\frac{x_1}{2}\cdot\cos\frac{3x_1}{2}\right)\cdot\sum_{k=1}^{n}\sin^3 x_k+$$

$$+3\sin\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}\sin^2 x_1\cdot\cos x_1;$$

$$d=\left(\frac{1}{2}\cos\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}+\frac{3}{2}\sin\frac{x_1}{2}\cdot\cos\frac{3x_1}{2}\right)\cdot\sum_{k=1}^{n}\cos^3 x_k-$$

$$-3\sin\frac{x_1}{2}\cdot\sin\frac{3x_1}{2}\cdot\cos^2 x_1\cdot\sin x_1;$$

$$e=\left(\cos x_1\cdot\sin 2x_1+2\sin x_1\cdot\cos 3x_1\right)\cdot\sum_{k=1}^{n}\sin^4 x_k+$$

$$+4\sin x_1\cdot\sin 2x\cdot\sin^3 x_1\cdot\cos x_1;$$

$$f=\left(\cos x_1\cdot\sin 2x_1+2\sin x_1\cdot\cos 3x_1\right)\cdot\sum_{k=1}^{n}\cos^4 x_k-$$

$$-4\sin x_1\cdot\sin 2x\cdot\cos^3 x_1\cdot\sin x_1.$$

Задание 5

Построить решение и численное решение системы

$$\frac{d\mathbf{x}}{dt}=\mathbf{A}\cdot\mathbf{x}+\mathbf{b};\ \mathbf{x}(0)=\mathbf{x}_0;\ t\in[0,T]$$

для одного из следующих частных случаев (4×4)-матриц $\mathbf{A}$ (табл. 3.5).

Таблица 3.5. *Вид матрицы* **A**

№ п/п	Матрица	№ п/п	Матрица
1.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}-\mathbf{E} & 0\\ 0 & \mathbf{R}\end{bmatrix}\mathbf{V}^{-1}$	5.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{C} & 0\\ 0 & \mathbf{G}\end{bmatrix}\mathbf{V}^{-1}$
2.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{R} & 0\\ 0 & \mathbf{C}\end{bmatrix}\mathbf{V}^{-1}$	6.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}-\mathbf{E} & 0\\ 0 & \mathbf{C}\end{bmatrix}\mathbf{V}^{-1}$
3.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{R} & 0\\ 0 & \mathbf{G}\end{bmatrix}\mathbf{V}^{-1}$	7.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{C1} & 0\\ 0 & \mathbf{C2}\end{bmatrix}\mathbf{V}^{-1}$
4.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}-\mathbf{E} & 0\\ 0 & \mathbf{G}\end{bmatrix}\mathbf{V}^{-1}$	8.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{R1} & 0\\ 0 & \mathbf{R2}\end{bmatrix}\mathbf{V}^{-1}$

Таблица 3.5 *(окончание)*

№ п/п	Матрица	№ п/п	Матрица
9.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{G1} & 0\\ 0 & \mathbf{G2}\end{bmatrix}\mathbf{V}^{-1}$	11.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{R} & 0\\ 0 & -\mathbf{E}\end{bmatrix}\mathbf{V}^{-1}$
10.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}-\mathbf{E} & 0\\ 0 & \mathbf{C}\end{bmatrix}\mathbf{V}^{-1}$	12.	$\mathbf{A}=\mathbf{V}\begin{bmatrix}\mathbf{G} & 0\\ 0 & -\mathbf{E}\end{bmatrix}\mathbf{V}^{-1}$

где

$$\mathbf{E}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix};\ \mathbf{R}=\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix};\ \mathbf{G}=\begin{bmatrix}\lambda & 1\\ 0 & \lambda\end{bmatrix};\ \mathbf{C}=\begin{bmatrix}\mu & \nu\\ -\nu & \mu\end{bmatrix}.$$

В качестве матриц $\mathbf{V}$ использовать матрицы из *приложения 1*, где приведены матрицы, обратные для которых известны. Номер матриц из *приложения 1* совпадает с номером задания. Решение строить по формуле:

$$\mathbf{x}(t)=\exp(\mathbf{A}\cdot t)\cdot\mathbf{x}_0+\int_0^t\exp\big(\mathbf{A}\cdot(t-z)\big)\cdot\mathbf{b}\cdot dz=e^{\mathbf{A}t}\cdot\mathbf{x}_0+\mathbf{A}^{-1}\cdot(e^{\mathbf{A}t}-\mathbf{E})\cdot\mathbf{b}\ .$$

Для построения матричных функций $\exp(\mathbf{A}\cdot t)$, $\mathbf{A}^{-1}$ использовать формулу Лагранжа—Сильвестра и следующие правила:

- для матрицы второго порядка и простых собственных чисел формула Лагранжа—Сильвестра выглядит так:

$$\mathbf{f}(\mathbf{A})=\frac{\mathbf{A}-\lambda_1\mathbf{E}}{\lambda_1-\lambda_2}f(\lambda_2)+\frac{\mathbf{A}-\lambda_2\mathbf{E}}{\lambda_2-\lambda_1}f(\lambda_1);$$

- правило 1: $\mathbf{A}=\mathbf{S}\Lambda\mathbf{S}^{-1}\Rightarrow f(\mathbf{A})=\mathbf{S}\,f(\Lambda)\mathbf{S}^{-1}$;
- правило 2: $\mathbf{A}=\begin{bmatrix}\mathbf{A1} & 0\\ 0 & \mathbf{A2}\end{bmatrix}\Rightarrow f(\mathbf{A})=\begin{bmatrix}f(\mathbf{A1}) & 0\\ 0 & f(\mathbf{A2})\end{bmatrix}$;
- правило 3: $\mathbf{J}=\lambda\mathbf{E}+\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\Rightarrow f(\mathbf{J})=\begin{bmatrix}f(\lambda) & f'(\lambda)\\ 0 & f(\lambda)\end{bmatrix}$.

Использовать найденное решение как тестовое для проверки численного решения.

Отчет должен содержать:

- ❒ текст программы на языке пакета;
- ❒ явный вид матриц $\mathbf{A}^{-1}$ и $\mathbf{V}^{-1}$;
- ❒ график любого устойчивого решения, полученного численно и аналитически, и график

$$F(t) = \frac{\left\| \mathbf{x_T} - \mathbf{x} \right\|}{\left\| \mathbf{x_T} \right\|},$$

где $\mathbf{x_T}$ — решение; $\mathbf{x}$ — численное решение.

Гибридные автоматы

Интерфейс "Гибридный элементарный объект".

Задание 6

Реализовать отображения [18, 21]:

- ❒ отображение Хенона

$$\begin{cases} x_{n+1} = -\left(y_n - x_n^2\right) \cdot \sin\left(\dfrac{2\pi}{p_1}\right) + x_n \cdot \cos\left(\dfrac{2\pi}{p_1}\right); \\ x_{n+1} = \left(y_n - x_n^2\right) \cdot \cos\left(\dfrac{2\pi}{p_1}\right) + x_n \cdot \sin\left(\dfrac{2\pi}{p_1}\right); \end{cases}$$

рекомендуемые значения $p_1 = 6$;

- ❒ отображение Чирикова

$$\begin{cases} x_{n+1} = y_{n+1} + x_n; \\ y_{n+1} = y_n + p_1 \sin(x_n); \end{cases}$$

рекомендуемые значения $p_1 = 0.7$;

- ❒ отображение Мира

а) $$\begin{cases} x_{n+1} = p_1 x_n + y_n; \\ y_{n+1} = p_2 + x_n^2; \end{cases}$$

рекомендуемые значения $p_1 = 0.7$;

б) $$\begin{cases} x_{n+1} = (1-p_1)x_n + y_n; \\ y_{n+1} = p_1 p_2 x_n + p_3 y_n - p_1 x_n^3; \end{cases}$$

рекомендуемые значения $p_1 = 2.68$; $p_2 = -0.1$; $p_3 = 0.9$;

в) $$\begin{cases} x_{n+1} = p_2 y_n - F(x_n); \\ y_{n+1} = -x_n + F(x_{n+1}); \end{cases}$$

$$F(x_n) = p_1 x_n + (1-p_1)\frac{2x_n^2}{1+x_n^2};$$

рекомендуемые значения $p_1 = 0.5$; $p_2 = 1$;

г) $$\begin{cases} x_{n+1} = x_n^2 + p_1 x_n - y_n^2 + p_2; \\ y_{n+1} = x_n y_n - \frac{5}{2} p_1 y_n; \end{cases}$$

рекомендуемые значения $p_1 = -0.95126$; $p_2 = -0.8788$;

- отображение Катала

$$\begin{cases} x_{n+1} = p_1 x_n + y_n; \\ y_{n+1} = p_2 + x_n; \end{cases}$$

рекомендуемые значения $p_1 = 0.7$; $p_2 = -0.82$;

- отображение Заславского

$$\begin{cases} x_{n+1} = x_n \cos(p_1) - (y_n + p_2 \sin(x_n)) \cdot \sin(p_1); \\ y_{n+1} = x_n \sin(p_1) + (y_n + p_2 \sin(x_n)) \cdot \cos(p_1); \end{cases}$$

рекомендуемые значения $p_1 = 0.8$; $p_2 = 1$;

- отображение Улама

$$\begin{cases} x_{n+1} = x_n + \cos(2\pi y_n); \\ y_{n+1} = y_n + \frac{p_1}{x_{n+1}}; \end{cases}$$

рекомендуемые значения $p_1 = 5,20$;

- гравитационное отображение Заславского

$$\begin{cases} x_{n+1} = x_n + \cos(2\pi y_n); \\ y_{n+1} = y_n + p_1 x_{n+1}; \end{cases}$$

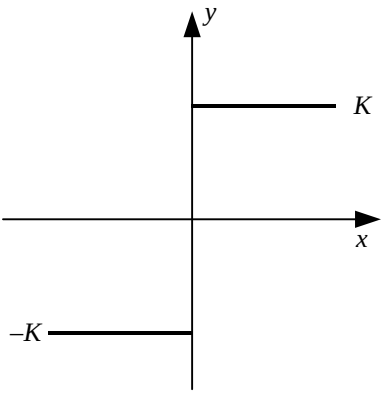
рекомендуемые значения $p_1 = 20$.

Задание 7

Найти численное решение уравнения

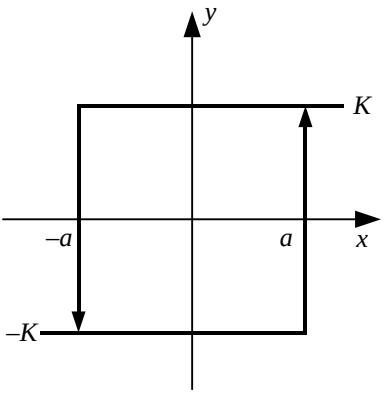
$$\frac{d^2x}{dt^2} = -F - k_2 \cdot x + k_3 \sin(k_4 \cdot t)$$

с константами k_1, k_2, k_3, k_4 и функцией $F = y$ с различными функциями y (рис. 3.1—3.3). Константы, определяющие вид функции: K — максимальное по модулю значение функции, $a > 0$, $b > 0$, $d > 0$ — константы, определяющие нули функций, выбрать самостоятельно.



$y = K \cdot \mathrm{sign}(x)$

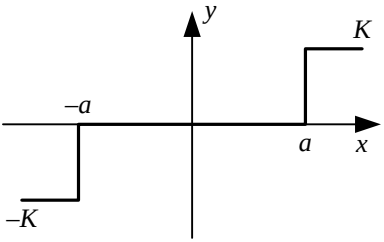
а



$$y = \begin{cases} K \cdot \mathrm{sign}(x-a), \dfrac{dx}{dt} > 0 \\ K \cdot \mathrm{sign}(x+a), \dfrac{dx}{dt} < 0 \end{cases}$$

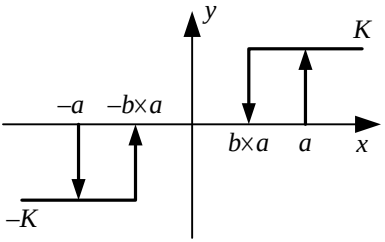
б

Рис. 3.1. Нелинейные функции на базе функции sign() *(начало)*



$$y = \frac{K}{2} \cdot \left(\operatorname{sign}(x-a) + \operatorname{sign}(x+a)\right)$$

в

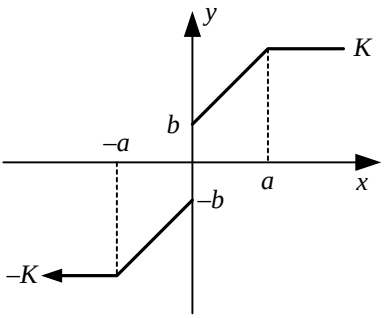


$$y = \begin{cases} \frac{K}{2} \cdot \left(\operatorname{sign}(x-a) + \operatorname{sign}(x+b \cdot a)\right), \frac{dx}{dt} > 0 \\ \frac{K}{2} \cdot \left(\operatorname{sign}(x+a) + \operatorname{sign}(x-b \cdot a)\right), \frac{dx}{dt} < 0 \end{cases},$$

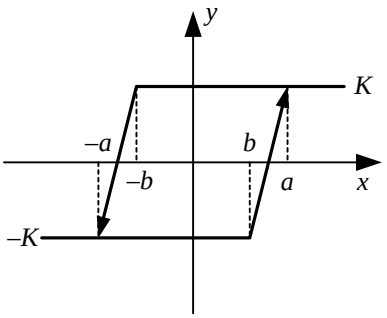
$b < 1$

г

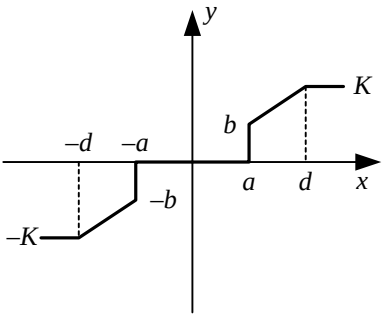
Рис. 3.1. Нелинейные функции на базе функции sign() *(окончание)*



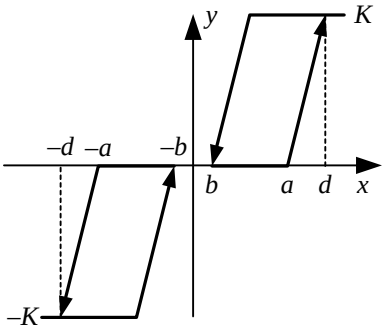
а



б

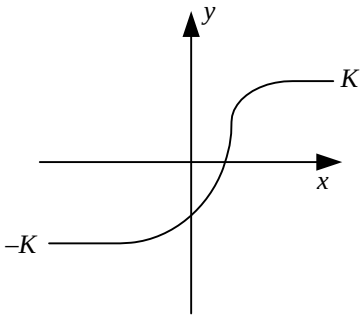


в



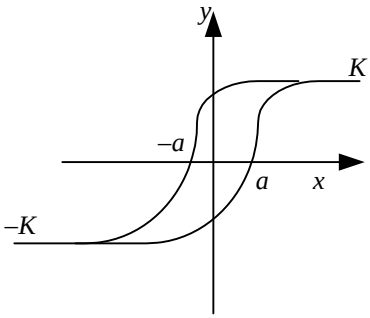
г

Рис. 3.2. Кусочно-непрерывные функции на базе линейных функций



$$y = \frac{2 \cdot K}{\pi} \operatorname{arctg}(\alpha \cdot x)$$

а



$$y = \frac{K}{\pi} \operatorname{arctg}(\alpha \cdot x + a) + \frac{K}{\pi} \operatorname{arctg}(\alpha \cdot x - a)$$

б

Рис. 3.3. Нелинейные функции на базе функции arctg(x)

Задание 8

Построить глобальные фазовые портреты систем

1. $\begin{cases} \dfrac{dx_1}{dt} = x_1(a - bx_2); \\ \dfrac{dx_2}{dt} = -x_2(c - dx_1); \ \ a, b, c, d > 0. \end{cases}$

2. $\begin{cases} \dfrac{dy_1}{dt} = y_1(3 - 4y_2); \\ \dfrac{dy_2}{dt} = -y_2(5 - 6y_1); \end{cases}$

3. $\begin{cases} \dfrac{dx_1}{dt} = x_1(2 - x_1 - 2x_2); \\ \dfrac{dx_2}{dt} = x_2(2 - x_1 - x_2). \end{cases}$

4. $\begin{cases} \dfrac{dx_1}{dt} = x_2; \\ \dfrac{dx_2}{dt} = x_2(1 - x_1^2) - x_1. \end{cases}$

5. $\begin{cases} \dfrac{dx_1}{dt} = x_2 - 1; \\ \dfrac{dx_2}{dt} = -(x_1 - 2)x_2. \end{cases}$

6. $\begin{cases} \dfrac{dx_1}{dt} = x_1 + x_2; \\ \dfrac{dx_2}{dt} = \sin x_1. \end{cases}$

7. $\begin{cases} \dfrac{dx_1}{dt} = -(x_1 - x_2)x_1; \\ \dfrac{dx_2}{dt} = x_1 + x_2; \end{cases}$

8. $\begin{cases} \dfrac{dx_1}{dt} = x_1 - x_2 - x_1^3; \\ \dfrac{dx_2}{dt} = x_1 + x_2 - x_2^2. \end{cases}$

9. $\begin{cases} \dfrac{dx_1}{dt} = 2x_1 - x_2^2; \\ \dfrac{dx_2}{dt} = -x_2 - x_1 x_2; \end{cases}$

10. $\dfrac{d^2x}{dt^2} = -\left(\dfrac{dx}{dt}\right)^2 - x.$

11. $\dfrac{d^2x}{dt^2} = -2a\dfrac{dx}{dt} - b\sin x, \ \ a > 0, b > 0.$

12. $\dfrac{d^2x}{dt^2} = -\dfrac{dx}{dt} - x.$

Задание 9

Найти неподвижные точки, линеаризовать систему в их окрестности и построить фазовые портреты в окрестности особой точки исходной и линеаризованных систем.

1. $\begin{cases} \dfrac{dx_1}{dt} = \dfrac{1}{2}(x_1 + x_2) + {x_2}^2; \\ \dfrac{dx_2}{dt} = \dfrac{1}{2}(-x_1 + 3x_2); \\ \dfrac{dx_3}{dt} = -\dfrac{1}{2}x_3. \end{cases}$

2. $\begin{cases} \dfrac{dx_1}{dt} = x_1(x_1 + 2x_2); \\ \dfrac{dx_2}{dt} = x_2(2x_1 + x_2); \\ \dfrac{dx_3}{dt} = \dfrac{1}{2}x_3. \end{cases}$

3. $\begin{cases} \dfrac{dx_1}{dt} = x_1(x_1 - x_2); \\ \dfrac{dx_2}{dt} = x_2(2x_1 - x_2); \\ \dfrac{dx_3}{dt} = -\dfrac{1}{2}x_3. \end{cases}$

4. $\begin{cases} \dfrac{dx_1}{dt} = -x_2^5; \\ \dfrac{dx_2}{dt} = x_1 + x_2^2; \\ \dfrac{dx_3}{dt} = x_3. \end{cases}$

5. $\begin{cases} \dfrac{dx_1}{dt} = 2x_1 - x_2^2; \\ \dfrac{dx_2}{dt} = -x_2 + x_1 x_2; \\ \dfrac{dx_3}{dt} = \dfrac{1}{2} x_3. \end{cases}$

6. $\begin{cases} \dfrac{dx_1}{dt} = x_1 + x_1 x_2^3 \left(1 + x_1^2\right); \\ \dfrac{dx_2}{dt} = 2x_1 - 3x_2; \\ \dfrac{dx_3}{dt} = \dfrac{1}{2} x_3. \end{cases}$

7. $\begin{cases} \dfrac{dx_1}{dt} = -x_1^2 + \sin x_2 - 1; \\ \dfrac{dx_2}{dt} = \operatorname{sh}\left(x_1 - 1\right); \\ \dfrac{dx_3}{dt} = -x_3. \end{cases}$

8. $\begin{cases} \dfrac{dx_1}{dt} = x_1^2 - e^{x_2}; \\ \dfrac{dx_2}{dt} = -x_2(1 + x_1); \\ \dfrac{dx_3}{dt} = -\dfrac{1}{2} x_3. \end{cases}$

9. $\begin{cases} \dfrac{dx_1}{dt} = 3x_1 + x_2^2 + 2; \\ \dfrac{dx_2}{dt} = x_1^2 - x_2^2; \\ \dfrac{dx_3}{dt} = -\dfrac{1}{2} x_3. \end{cases}$

10. $\begin{cases} \dfrac{dx_1}{dt} = x_2; \\ \dfrac{dx_2}{dt} = -x_1 - x_1^3; \\ \dfrac{dx_3}{dt} = -\dfrac{1}{2} x_3. \end{cases}$

11. $\begin{cases} \dfrac{dx_1}{dt} = \sin(x_1 + x_2); \\ \dfrac{dx_2}{dt} = x_2; \\ \dfrac{dx_3}{dt} = -\dfrac{1}{2} x_3. \end{cases}$

12. $\begin{cases} \dfrac{dx_1}{dt} = x_1 - x_2 + e^{x_1}; \\ \dfrac{dx_2}{dt} = x_1 - x_2 - 1; \\ \dfrac{dx_3}{dt} = \dfrac{1}{2} x_3. \end{cases}$

13. $\begin{cases} \dfrac{dx_1}{dt} = x_2; \\ \dfrac{dx_2}{dt} = \left(1 - x_1^2 - x_1^4\right) x_2; \\ \dfrac{dx_3}{dt} = \dfrac{1}{2} x_3. \end{cases}$

14. $\begin{cases} \dfrac{dx_1}{dt} = \left(x_1 x_2 - 3x_2\right) - 4; \\ \dfrac{dx_2}{dt} = x_2^2 - x_1^2; \\ \dfrac{dx_3}{dt} = \dfrac{1}{2} x_3. \end{cases}$

Компонентные модели

Интерфейс "Блок-схема".

Задание 10

Для заданных преподавателем: схемы (рис. 3.4—3.6), функции $R(i)$, определяющей зависимость переменного сопротивления от тока (табл. 3.6), источника питания (табл. 3.7) и параметров (табл. 3.8):

1. Написать уравнения, определяющие зависимость тока в цепи от времени.
2. Выделить в схеме три компонента: источник питания, переменное сопротивление, оставшуюся часть схемы с постоянными параметрами и построить компонентную модель.
3. Построить панель управления, позволяющую подключать и отключать источник питания по ходу эксперимента с помощью активного элемента — кнопки.

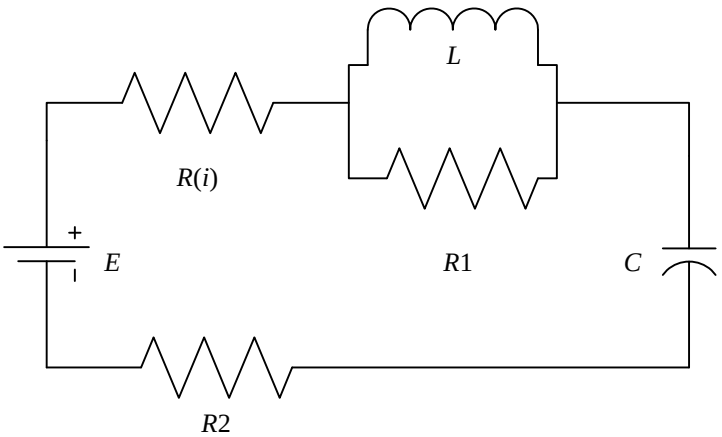


Рис. 3.4. Схема 1

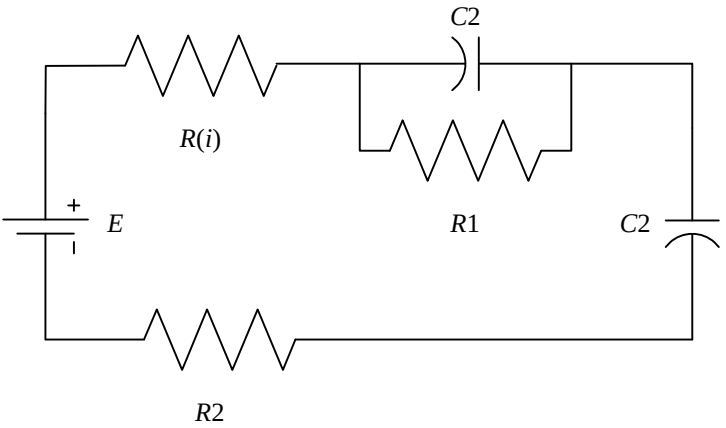


Рис. 3.5. Схема 2

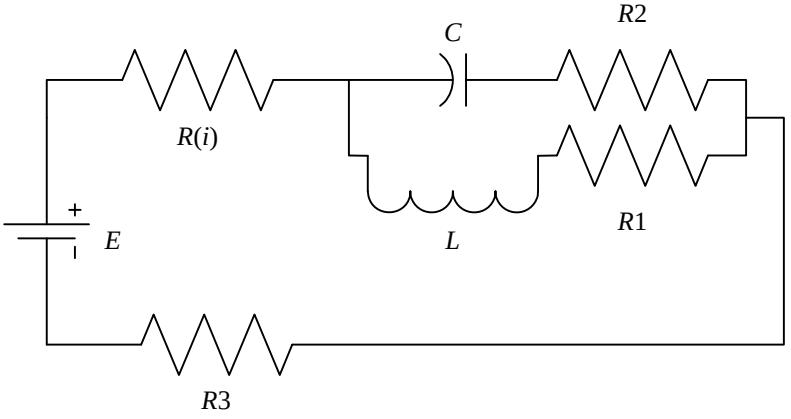


Рис. 3.6. Схема 3

Таблица 3.6. *Нелинейные сопротивления*

Вариант	**Вид функции**
$R_1(i)$	$\begin{cases} A \cdot i, & \lvert i \rvert \le i_{\max} \\ A \cdot i_{\max}, & i > i_{\max} \\ -A \cdot i_{\max}, & i < -i_{\max} \end{cases}$

Таблица 3.6 *(окончание)*

Вариант	**Вид функции**
$R_2(i)$	$\begin{cases} A\cdot i^3, & \lvert i\rvert \le i_{\max} \\ A\cdot i_{\max}^3, & i > i_{\max} \\ -A\cdot i_{\max}^3, & i < -i_{\max} \end{cases}$
$R_3(i)$	$\begin{cases} A\cdot i^3, & \lvert i\rvert \le i_{\max} \\ A\cdot i_{\max}^3 + i - i_{\max}, & i > i_{\max} \\ -A\cdot i_{\max}^3 + i - i_{\max}, & i < -i_{\max} \end{cases}$
$R_4(i)$	$\begin{cases} A\cdot i^2, & i \ge 0 \\ -A\cdot i^2, & i < 0 \end{cases}$

Таблица 3.7. *Источники питания*

Вариант	**Вид функции**
$E_1(t)$	b
$E_2(t)$	$b\cdot\sin(t)$
$E_3(t)$	$b\cdot\lvert\sin(t)\rvert$
$E_4(t)$	$b\cdot t$, $0 \le t \le T$, $E_4(t+T) = E_4(t)$, $T = 1$
$E_5(t)$	$\begin{cases} b\cdot t, & 0 \le t \le T \\ -b\cdot t, & T \le t \le 2T \end{cases}$ $E_5(t+2T) = E_5(t)$, $T = 1$

Таблица 3.8. *Параметры схемы*

Параметры	**Значение**
Индуктивности	$L=1$; $L_1=1$; $L_2=10$
Емкости	$C=1$; $C_1=1$; $C_2=10$
Сопротивления	$R_1=0.1$; $R_2=0.01$
Источник	$b=1$
Нелинейное сопротивление	$A=2$; $i_{\max}=3$

Модели общего вида

Интерфейс "Объект общего вида".

Задание 11

1. Частица начинает движение в плоскости (x, y) с постоянной скоростью $V_x=c_1$ и $V_y=c_2$ из начальной точки (x_0, y_0) и под действием гравитационного поля внутри
   - квадрата со стороной d;
   - равнобедренного прямоугольного треугольника с катетами длиной d;
   - равностороннего треугольника со стороной, равной d.

   Соударение со стенками абсолютно упругое. Что будет происходить с траекториями, если гравитационное поле:
   - отсутствует;
   - периодически "выключается";
   - присутствует.

2. Дано дифференциальное уравнение

$$\frac{d^2x}{dt^2} + F(x) = \sin(t)\,;\ \ x(0) = x_0\,,$$

где

$F(x) = k \cdot \mathrm{sign}(x)\,,\ k > 0\,;$

$$F(x) = \begin{cases} p \cdot x, & \mathrm{abs}(x) \le L,\ p > 0, \\ pL, & x > L, \\ -pL, & x < L; \end{cases}$$

$$F(x) = \begin{cases} 0, & x \le L,\ p > 0, \\ p, & x > L, \\ -p, & x > L; \end{cases}$$

$$F(x) = \begin{cases} px + c, & x > 0,\ p > 0,\ c > 0, \\ px - c, & x < 0; \end{cases}$$

$$F(x) = \begin{cases} p_1 x + c_1, & x > L > 0,\ p_1 > 0,\ c_1 > 0, \\ p_2 x - c_2, & x < L,\ p_2 > 0,\ c_2 > 0; \end{cases}$$

$$F(x) = \begin{cases} 0, & x \le L, \\ p_1 x + c_1, & x > L,\ p_1 L + c_1 = 0, \\ p_1 x + c_2, & x > L, -p_1 L + c_2 = 0. \end{cases}$$

Что будет происходить с траекториями, если синусоидальное воздействие:

- отсутствует;
- периодически "выключается";
- присутствует.

3. Плоские колебания маятника в виде шара массы m, подвешенного на невесомом и абсолютно упругом стержне длины L, ограничены двумя абсолютно упругими стенками, расположенными на расстоянии d от точки подвеса. Удар о стенки или любой упор абсолютно упругие. На маятник действует синусоидальная возмущающая сила

$$\frac{d^2x}{dt^2} + a^2 \cdot x = k^2 \cdot \sin(t)\,,\ a > k\ .$$

Что будет происходить с маятником, если синусоидальное воздействие:

- отсутствует;
- периодически "выключается";
- присутствует.

Задание 12

Приведенная далее серия задач [22] предполагает следующую технологию проектирования. Сначала строится описание проектируемой модели на языке UML. Затем это описание используется для построения моделей в двух различных средах моделирования, например Simulink и MvStudium. Полученные результаты сравниваются.

В отчете должны быть представлены:

- ❒ описание системы, в терминах унифицированного языка моделирования UML;
- ❒ модель системы, реализованная в подсистеме Simulink пакета MATLAB на основе UML-описания;
- ❒ модель системы, реализованная в пакете Studium на основе UML-описания;
- ❒ результаты вычислительного эксперимента и их объяснение.

Примечание

При построении модели любой из задач в пакете MvStudium необходимо реализовать трехмерную анимацию (задачи 1—10) или создать панель управления параметрами, содержащую в себе компоненты двумерной анимации (задачи 11—12).

Задача 1

Переменная сила Q передается на фундамент машиной с неуравновешенным ротором (рис. 3.7).

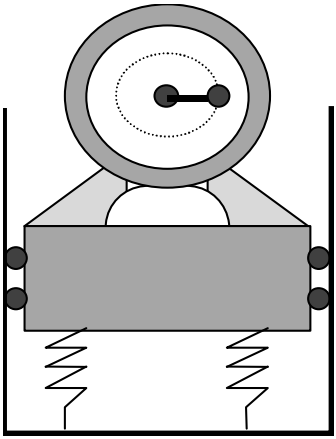


Рис. 3.7. Машина с неуравновешенным ротором

Сила Q меняется следующим образом: через каждые 50 секунд один вид гармонических колебаний

$$Q_1 = H_1 \cdot \sin(\omega_1 t) \tag{3.3}$$

сменяет другой

$$Q_2 = H_2 \cdot \cos(\omega_2 t)\,, \tag{3.4}$$

где H_1 и H_2 — амплитуда возмущающей силы; ω_1 и ω_2 — частота возмущающей силы.

$$H_1 = 2 \text{ см},\ H_2 = 3 \text{ см},\ \omega_1 = 0.3 \text{ с}^{-1},\ \omega_2 = 0.5 \text{ с}^{-1}.$$

Если q — положение центра тяжести машины, то уравнения для различных возмущающих сил запишутся в виде:

$$\ddot{q} + k^2 \cdot q = \frac{Q_1}{a} \tag{3.5}$$

для гармонического закона (3.3):

$$\ddot{q} + k^2 \cdot q = \frac{Q_2}{a} \tag{3.6}$$

для гармонического закона (3.4), где $a = 1 \text{ с}^{-2}$ — инерционный коэффициент; $k = 0.4 \text{ с}^{-2}$ — частота собственных колебаний машины.

Начальные условия: $q\big|_{t=0} = 0$ и $\dot{q}\big|_{t=0} = 0$.

При таком воздействии на систему величина q может превзойти критическую величину $L_k = 100$, и машина выйдет из строя. Предположим, что мы можем управлять частотами возмущающей силы ω_1 и ω_2. Проверить экспериментально, можно ли для избежания аварии, при превышении q по абсолютному значению величины $L = 50$, уменьшать частоты на 10%. Процесс изменения частоты требует 10 секунд модельного времени.

Можно ли таким способом управлять двумя машинами, с разными значениями коэффициентов $k_1 = 0.4$ и $k_2 = 0.5$; $a_1 = 1$ и $a_2 = 0.5$. Машины следует "запускать" поочередно — сначала первую, а затем через 10 секунд модельного времени вторую.

Задача 2

Дана система из двух маятников — маятника 1 и обращенного маятника 2 (рис. 3.8).

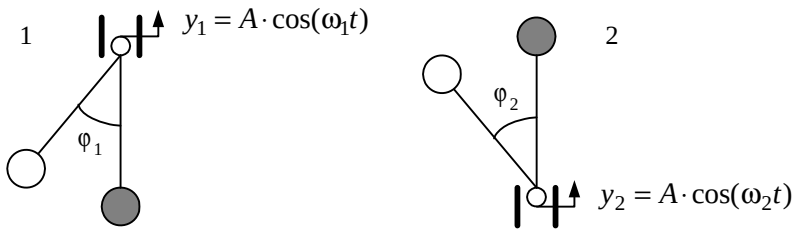


Рис. 3.8. Система маятников

Точка подвеса первого маятника гармонически колеблется по вертикали около своего среднего положения по закону (3.7), а точка подвеса второго маятника — у своего по закону (3.8):

$$y_1 = A \cdot \cos(\omega_1 t); \tag{3.7}$$

$$y_2 = A \cdot \cos(\omega_2 t), \tag{3.8}$$

где A — амплитуда колебаний; ω_1 и ω_2 — частоты колебаний.

$$A = 2 \text{ м}, \ \omega_1 = 3 \text{ с}^{-1}, \ \omega_2 = 3 \text{ с}^{-1}.$$

При этом каждые 10 секунд значение A изменяется на величину ±0.5 поочередно. Длина каждого из маятников равна $l = 40$ м.

Пусть φ_1 — это угол отклонения первого маятника от строго вертикального положения, а φ_2 — это угол отклонения второго маятника от строго вертикального положения. У первого маятника φ_1 изменяется в соответствии с уравнением:

$$\ddot{\varphi}_1 + \left(\frac{g}{l} + \frac{A\omega_1^2}{l} \cdot \cos(\omega_1 t) \right) \cdot \varphi_1 = 0 . \qquad (3.9)$$

Второй маятник колеблется по закону

$$\ddot{\varphi}_2 + \left(\frac{g}{l} + \frac{A\omega_2^2}{l} \cdot \cos(\omega_2 t) \right) \cdot \varphi_2 = 0 . \qquad (3.10)$$

И если $A\omega > \sqrt{2gl}$, где g — это ускорение свободного падения, то маятник со временем приходит в устойчивое состояние $\varphi_2 = 0$.

У обоих маятников в начальный момент времени отклонение от положения равновесия составляет 0.1°. Значения φ_1 и φ_2 изменяются в пределах от 0 до 360°.

Создайте модель системы и убедитесь, что второй маятник приходит в устойчивое положение при этих значениях параметров.

Предположим теперь, что движение точек подвеса осуществляется устройством, которое не может работать постоянно и вынуждено останавливаться через каждые 10 секунд модельного времени на 1 секунду (т. е. в этот момент точки подвеса не двигаются и $\omega_1 = \omega_2 = 0$). Будет ли сохраняться устойчивое положение второго маятника в этом случае, и как изменится поведение первого?

Во всех экспериментах сначала начинает колебаться первый маятник, а через 3 секунды модельного времени второй.

Задача 3

Дан неровный участок со сложным профилем, который периодически повторяется (рис. 3.9). По нему движутся две идентичные "машины", каждая из которых представляет собой подрессорный груз массой $m = 0.7$ кг, прикреп-

ленный к безотрывно движущемуся с постоянной горизонтальной скоростью колесу. Скорость первой машины $V_1 = 0.9$ м/с, второй — $V_2 = 1$ м/с.

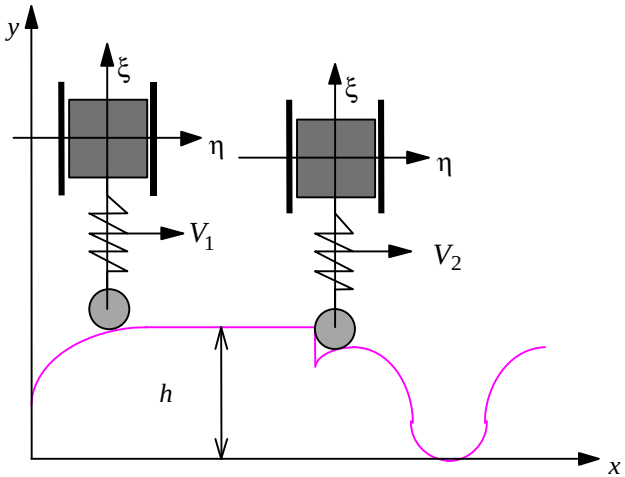


Рис. 3.9. Система "подрессорный груз-колесо"

Профиль участка для одного периода описывается уравнениями

$$y = h \cdot (1 - e^{-\gamma x}),\ 0 \le x \le 1; \tag{3.11}$$

$$y = A_1 \cdot \cos(x),\ A_1 = h \cdot (1 - e^{-\gamma}),\ 1 < x \le 2; \tag{3.12}$$

$$y = A_2 \cdot \sin(x),\ A_2 = A_1 \cdot \cos(2),\ 2 < x < 4, \tag{3.13}$$

где h — высота горизонтального участка; γ — параметр, характеризующий кривизну профиля; A_1 и A_2 — амплитуды колебаний.

$$h = 1 \text{ м},\ \gamma = 1.$$

Дифференциальные уравнения, описывающие вертикальные колебания ξ (в начальный момент времени отсутствующие) центра тяжести машины, движущейся с постоянной скоростью $x = Vt$, записываются в следующем виде:

- для первого участка:

$$\ddot{\xi} = -\frac{c}{m} \cdot \xi + h \cdot \gamma^2 \cdot V^2 \cdot e^{-\gamma V t}; \tag{3.14}$$

- для второго:

$$\ddot{\xi} = -\frac{c}{m} \cdot \xi + A_1 \cdot V^2 \cdot \cos(Vt)\,; \qquad (3.15)$$

- для третьего:

$$\ddot{\xi} = -\frac{c}{m} \cdot \xi + A_2 \cdot V^2 \cdot \sin(Vt)\,, \qquad (3.16)$$

где c — жесткость упругой подвески; $c = 0.5$ кг/с^2.

Первая машина в начальный момент времени расположена в начале пути, а вторая расположена в начале косинусоидального участка, и они начинают движение со скоростями $V_1 = 12$ и $V_2 = 10$.

За движением машин следит система управления. Если расстояние между машинами становится меньше чем $L = 0.2$, то догоняющая машина снижает свою скорость на 50%. Если расстояние оказывается меньшим, чем $L_k = 0.1$, то считается, что произошла авария. Если расстояние между машинами больше $L_b = 1$, то скорость обеих машин увеличивается на 30% от текущего значения.

Построить модель системы.

Задача 4

Жесткая плоская пластинка длиной $l = 5$ м находится в потоке газа (жидкости), скорость $V = 1$ м/с которого горизонтальна и направлена так, как показано на рис. 3.10.

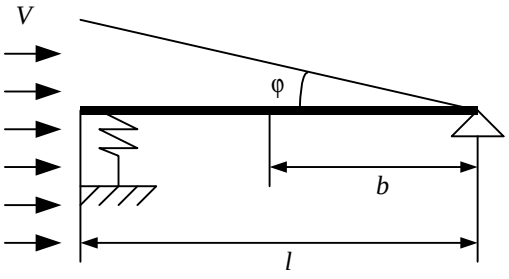


Рис. 3.10. Система "пластина—газ"

В этом положении аэродинамические силы равны нулю, и пластинка находится в равновесии под действием силы тяжести и реакции опор. При отклонении пластинки возникают аэродинамические давления, зависящие от угла отклонения φ.

Предположим, что в начальный момент мы отклонили пластинку от положения равновесия на угол 0.01°. Пусть $I = 1$ кг · м^2 — момент инерции пластинки относительно оси шарнира. Уравнения для угла можно записать в виде:

$$I \cdot \ddot{\varphi} + \left(c_0 \cdot l - k_y \cdot \frac{\rho V^2}{2} \cdot b \right) \cdot l \cdot \varphi = 0 , \qquad (3.17)$$

где c_0 — коэффициент жесткости пружины; k_y — постоянный аэродинамический коэффициент; ρ — плотность газа; b — расстояние от оси шарнира, определяющее точку приложения равнодействующих аэродинамических давлений на пластину.

$$c_0 = 0.5 \text{ кг/с}^2,\ k_y = 0.5 \text{ м} \cdot \text{с}^2,\ \rho = 2 \text{ кг/м}^3,\ b = 1 \text{ м}.$$

Если выполняется условие $c_0 \cdot l - k_y \cdot \frac{\rho V^2}{2} \cdot b > 0$, то в системе наблюдаются возрастающие по амплитуде колебания, в противном случае под действием аэродинамических сил пластинка со временем возвращается в положение равновесия.

Построить модель системы и убедиться, что при выбранных значениях параметров пластина с течение временем приходит в устойчивое состояние.

Предположим теперь, что в начальный момент пластина находится в состоянии равновесия. Каждые 5 секунд скорость подаваемого газа то увеличивается на 50%, то возвращается к прежнему значению. Также каждые 10 секунд плотность газа то увеличивается на 50%, то возвращается к прежнему значению. В некоторый случайный момент времени пластина мгновенно отклоняется на угол 0.01°. Будут ли при таких условиях наблюдаться аварийные ситуации, если аварийным считается отклонение пластины более чем на 45°?

Пусть в рассмотренных условиях пластина проработала 50 единиц модельного времени и вышла из строя. Ее мгновенно заменили новой, у которой конструктивные параметры c_0, k_y отличаются на 10%. Будут ли при таких условиях наблюдаться аварийные ситуации?

Задача 5

Дана система из двух тел с массами m_1 и m_2, соединенных двумя пружинами, жесткости которых равны c_1 и c_2 (рис. 3.11).

$$m_1 = 1 \text{ кг},\ m_2 = 1 \text{ кг},\ c_1 = 1 \text{ кг/с}^2,\ c_2 = 1 \text{ кг/с}^2.$$

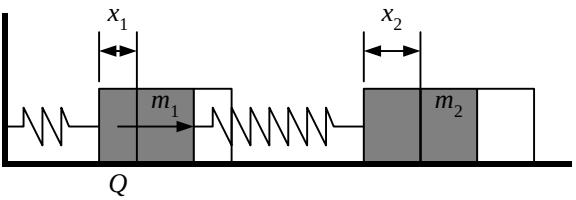


Рис. 3.11. Система двух грузов на пружинах

На левый груз системы действует гармоническая возмущающая сила Q, которая через каждые 20 секунд меняет свой вид:

$$Q = \begin{cases} H_1 \cdot \sin(\omega t), \\ H_2 \cdot \cos(\omega t), \end{cases} \tag{3.18}$$

где H_1 и H_2 — амплитуды колебаний; ω — частота колебаний.

$$H_1 = 1 \text{ м},\ H_2 = 1.5 \text{ м},\ \omega = 2 \text{ с}^{-1}.$$

Пусть x_1 и x_2 — горизонтальное отклонение грузов от положения равновесия (в начальный момент времени отсутствующее). Тогда уравнениями движения будут:

$$m_1\ddot{x}_1 + (c_1 + c_2) \cdot x_1 - c_2 x_2 = Q, \tag{3.19}$$

$$m_2\ddot{x}_2 - c_2 x_1 + c_2 x_2 = 0. \tag{3.20}$$

Построить модель данной системы, создав для каждого груза свой отдельный блок с уравнениями (3.19) и (3.20).

Предусмотреть возможность изменять частоту возмущающей силы в произвольный момент на 25% по желанию экспериментатора. Частота должна автоматически приблизиться к исходному значению через 10 секунд модельного времени. В эти 10 секунд изменение частоты экспериментатором невозмож-

но. При моделировании системы останавливать моделирование всегда, если расстояние между грузами стало меньше 10% от своего первоначального значения L, равного $L = 3$ или $L = 5$.

Задача 6

Дан вертикальный невесомый упругий стержень длиной $l = 5$ м с постоянной жесткостью сечения $EJ = 1$ кг · м^3/с^2. С концом стержня связан сосредоточенный груз массой $m = 1$ кг. Верхней опорой стержня служит неподвижный шарнир, а нижней опорой — подвижная втулка (рис. 3.12).

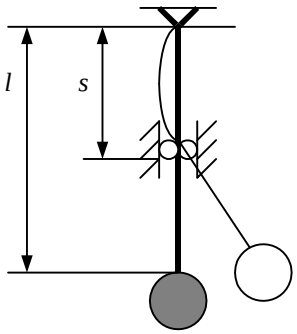


Рис. 3.12. Система "стержень—груз"

Расстояние между опорами $s < l$ не постоянно и меняется с периодом 30 секунд по одному из двух гармонических законов:

$$s = \begin{cases} H_1 \cdot \cos(\omega_1 t), \\ H_2 \cdot \sin(\omega_1 t), \end{cases} \tag{3.21}$$

где H_1 и H_2 — амплитуды колебаний; ω_1 — частота колебаний.

$$H_1 = 1.5 \text{ м}, \; H_2 = 2 \text{ м}, \; \omega_1 = 1 \text{ с}^{-1}.$$

Частота колебаний вне зависимости от закона колебаний каждые 20 секунд то уменьшается на 40%, то возвращается к прежнему значению.

Состоянию равновесия соответствует положение груза на вертикали и прямолинейная форма оси стержня. При возмущении этого состояния груз от-

клоняется в сторону, ось балки изгибается, и последующее движение описывается дифференциальным уравнением:

$$\ddot{x} + \frac{3 \cdot EJ}{m \cdot l \cdot [l - s]^2} \cdot x = 0 . \qquad (3.22)$$

В начальный момент времени происходит отклонение груза на расстояние $x_0 = 0.1$ м. Если x превышает пороговое значение $x_{\max} = 3$ м, то стержень разрушается.

Построить модель данной системы и проверить, есть ли в ней аварийные режимы.

Создать виртуальный тренажер, в котором в начальный момент времени стержень находится в состоянии равновесия. В случайные моменты времени он может мгновенного отклоняться на расстояние от 0.1 до 0.2 м. Задача оператора — отслеживать аварийные ситуации и в случае их возникновения мгновенно менять вышедшее устройство на новое, конструктивные параметры которого l, m, EJ не более чем на 10% отличаются от исходных.

Задача 7

Дана система из груза массой $m = 1$ кг, связанного с невесомой жесткой упруго закрепленной балкой.

Пусть $2l$ — длина балки, c_0 — коэффициент жесткости пружины. Один конец балки закреплен на шарнире, расположенном на неподвижной опоре (рис. 3.13).

$$l = 1 \text{ м}, \; c_0 = 1 \text{ кг/с}^2.$$

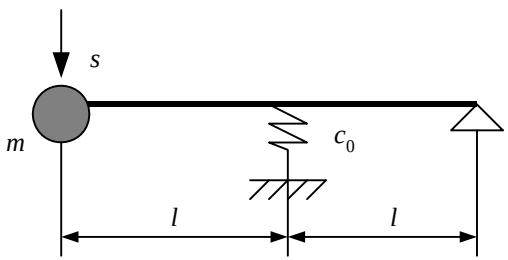


Рис. 3.13. Система "груз—балка—пружина"

В начальный момент времени происходит однократный вертикальный удар по грузу с величиной мгновенного ударного импульса $S = 2$ Н · с. Скорость груза получает мгновенное приращение:

$$\varphi\big|_{t=0} = 0\,,$$

$$\dot{\varphi}\big|_{t=0} = \frac{S}{2 \cdot m \cdot l}\,,$$

где φ — это угол отклонения системы от положения равновесия.

Движение системы описывается следующим уравнением:

$$4 \cdot m \cdot \ddot{\varphi} + c_0 \cdot \varphi = 0\,. \qquad (3.23)$$

Через 25 секунд после начала колебаний масса груза m мгновенно уменьшается на 50%. Далее движение системы продолжается, но уже с грузом новой массы.

Если угол φ становится больше предельного значения $\varphi_{\max} = \dfrac{S}{\sqrt{c_0 m l}}$, то происходит разрушение системы.

Построить модель системы и проверить экспериментально, наступают ли в ней аварийные режимы при заданных значениях параметров.

Построить тренажер, состоящий из двух таких систем, масса которых может меняться с помощью пульта управления экспериментатором, а конструктивные параметры l и $c_0 = 1$ меняются случайным образом не более чем на 10% в начале каждого нового эксперимента. Каждый эксперимент продолжается 100 секунд модельного времени, в течение которых в случайные моменты времени происходит удар то по одному грузу, то по другому (т. е. начальные условия выбираются по формуле (3.22)). Эксперимент считается удачным, если ни в одной из подсистем значение угла не превзошло критического значения.

Задача 8

В воде плавает кусок пробки в форме параллелепипеда с площадью основания $S = 1$ м^2 и высотой $H = 0.5$ м. Пробку погружают в воду на небольшую глубину $x_0 = 5$ см и отпускают (рис. 3.14).

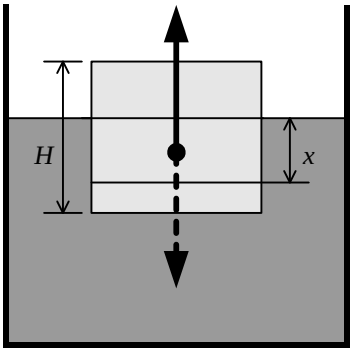


Рис. 3.14. Система "вода—пробка"

В результате пробка начинает совершать колебания. Сопротивление воды не учитывается. Изменение глубины погружения пробки в воду x описывается следующим уравнением:

$$\ddot{x} + \frac{\rho_{\text{в}} \cdot g}{\rho_{\text{п}} \cdot H} \cdot x = 0 \tag{3.24}$$

с начальными условиями:

$$x\big|_{t=0} = x_0\,;\ \dot{x}\big|_{t=0} = 0\,,$$

где $\rho_{\text{в}} = 1000$ кг/м^3 — плотность воды; $\rho_{\text{п}} = 200$ кг/м^3 — плотность пробки; g — ускорение свободного падения.

Пусть существует вторая ванна с пробкой, но в ней в начальный момент пробке, плавающей на поверхности, сообщили скорость, равную $v_0 = 1$ м/с. При изучении поведения пробки во второй ванне учли сопротивление воды, положив силу сопротивления равной $F_{\text{с}} = -r \cdot v$, где $r = 1$ кг/с — коэффициент пропорциональности. Колебания пробки в этой системе описываются уравнением:

$$\ddot{x} + \frac{r}{m} \cdot \dot{x} + \frac{\rho_{\text{в}} \cdot g}{\rho_{\text{п}} \cdot H} \cdot x = 0 \tag{3.25}$$

с начальными условиями:

$$x\big|_{t=0} = 0\,;\ \dot{x}\big|_{t=0} = v_0\,,$$

где m — масса пробки.

Построить модель системы и экспериментально выяснить, какая из пробок скачет выше?

Построить также модель системы, в которой каждые 20 секунд в первой системе вода мгновенно "превращается" в ртуть, а ртуть с той же периодичностью — в воду, а во второй системе каждые 25 секунд происходят аналогичные "превращения" воды в спирт. Плотность ртути — $\rho_р = 1360$ кг/м^3, плотность спирта — $\rho_с = 790$ кг/м^3.

Задача 9

Материальная точечная масса $m = 1$ кг находится в поле тяготения тонкого кольца массы $M = 10^{20}$ кг и радиуса $R = 1$ км. В начальный момент времени материальная точечная масса помещается в точку Q_1, расположенную на оси кольца на расстоянии $x_0 < R$ от плоскости кольца и начинает совершать колебательные движения (рис. 3.15).

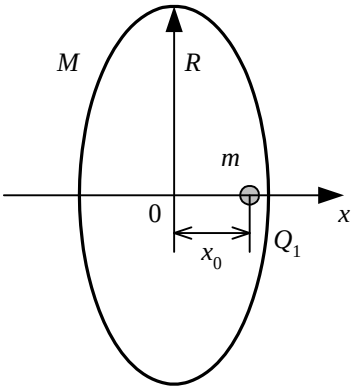


Рис. 3.15. Система "точечная масса — кольцо"

Если $x_0 < 0.1R$, то колебания x описываются следующим уравнением:

$$m \cdot \ddot{x} = -\frac{G \cdot M \cdot m}{R^3} \cdot x\,. \qquad (3.26)$$

Если $x_0 \geq 0.1R$ или в результате колебаний выполняется условие $x \geq 0.1R$, то колебания описываются следующим уравнением:

$$m \cdot \ddot{x} = -\frac{G \cdot M \cdot m}{\left(R^2 + x^2\right)^{3/2}} \cdot x, \tag{3.27}$$

где G — гравитационная постоянная.

Пусть $x_0 = 1$ м.

Каждые 15 секунд радиус кольца поочередно мгновенно расширяется и сжимается в 10 раз. Если точка отклоняется от кольца на расстояние, превышающее в 10 раз его первоначальный радиус, система разрушается.

Построить модель данной системы, а также модель системы, состоящей из двух систем "материальная точка — кольцо", не связанных друг с другом. Вторая система "материальная точка — кольцо" идентична первой, за исключением того, что в ней радиус кольца изменяется в 5 раз каждые 20 секунд.

Задача 10

В длинной вертикальной цилиндрической трубке, закрытой с нижнего конца, может ходить без трения поршень, масса $m = 10$ кг которого велика по сравнению с массой идеального газа, заключенного внутри трубки. В положении равновесия расстояние между поршнем и дном трубки равно $l_0 = 1000$ м (рис. 3.16).

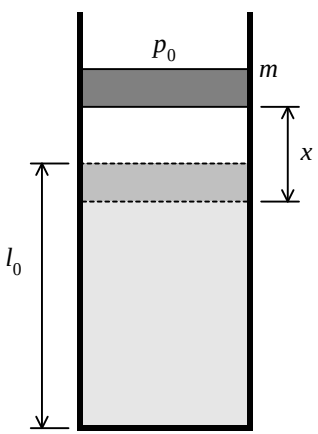


Рис. 3.16. Система "поршень—газ"

Площадь поперечного сечения трубки равна $S = 1$ м^2, на поршень действует нормальное атмосферное давление p_0. В начальный момент времени поршень отклоняют от положения равновесия на расстояние $x << l_0$. В результате этого поршень начинает совершать колебания, описываемые следующими уравнениями:

- если $p_0 \neq 0$, то

$$m \cdot \ddot{x} + \frac{m \cdot g + p_0 \cdot S}{l_0} \cdot x = 0\,; \tag{3.28}$$

- если $p_0 = 0$, то

$$\ddot{x} + \frac{g}{l_0} \cdot x = 0\,, \tag{3.29}$$

где g — ускорение свободного падения.

Пусть $x = 1$ м.

Атмосферное давление каждые 10 секунд мгновенно изменяется с нормального значения на меньшее на 80% (и обратно).

Построить модель данной системы, а также модель системы, состоящей из двух систем типа "трубка с поршнем", не связанных друг с другом. Вторая "трубка с поршнем" идентична первой, за исключением того, что в ней поршень имеет другую массу $m' > m$ и другую площадь поперечного сечения трубки $S' > S$.

Задача 11

Дана однокамерная фармакокинетическая модель с всасыванием (рис. 3.17), где 1 — это место введения лекарственного препарата; 2 — камера.

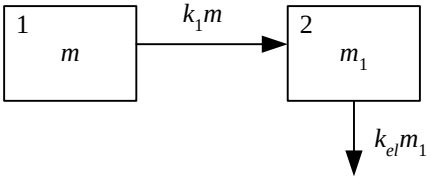


Рис. 3.17. Однокамерная фармакокинетическая модель с всасыванием

Камера представляет собой ограниченный в пространстве объем жидкости (ткани), неизменный с течением времени. Дан определенный объем лекарственного препарата, который всасывается в камеру пропорционально своей массе в соответствии с уравнением:

$$\frac{dm}{dt} = -k_1 \cdot m; \tag{3.30}$$

где m — масса лекарственного препарата в месте введения 1; k_1 — константа скорости поступления препарата в камеру (константа скорости всасывания).

Пусть $k_1 = 0.3\ \text{с}^{-1}$.

Предполагается, что масса лекарственного препарата в месте введения в начальный момент времени равна $m_0 = 30$ мг, причем в самой камере в начальный момент времени препарата нет. Тогда масса лекарственного препарата в камере изменяется в соответствии со следующим уравнением:

$$\frac{dm_1}{dt} = k_1 \cdot m - k_{el} \cdot m_1, \tag{3.31}$$

где m_1 — масса лекарственного препарата в камере; k_{el} — константа элиминации (выведения) лекарственного препарата из камеры.

Пусть $k_{el} = 0.5\ \text{с}^{-1}$.

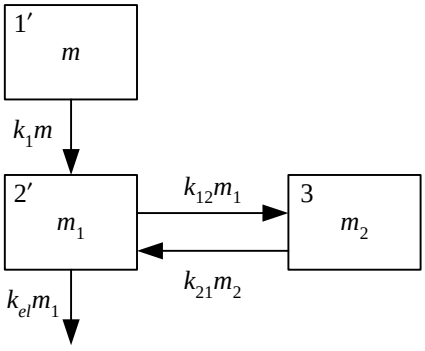


Рис. 3.18. Двухкамерная фармакокинетическая модель с всасыванием

Как только масса лекарственного препарата в месте введения становится меньше порогового значения $\varepsilon = 0.001$ мг, засекается отрезок времени $Time = 10$ с, по истечении которого в месте введения уничтожаются все остатки прежней дозы препарата, и вводится новая доза $m = m_0$.

Существует более сложная система, называемая двухкамерной фармакокинетической моделью с всасыванием (рис. 3.18).

В ней лекарственный препарат вводится из аналогичного ранее описанному места введения $1'$ и выводится из камеры $2'$, но существует еще камера 3, подсоединенная к камере $2'$. Между камерами $2'$ и 3 может циркулировать лекарственный препарат.

В этой модели масса лекарственного препарата в месте введения описывается также уравнением (3.30) с теми же начальными условиями. Массы же в камерах $2'$ и 3 описываются уравнениями (3.32) и (3.33) соответственно:

$$\frac{dm_1}{dt} = k_1 m - \left(k_{el} + k_{12}\right) \cdot m_1 + k_{21} m_2 ; \qquad (3.32)$$

$$\frac{dm_2}{dt} = k_{12} m_1 - k_{21} m_2 , \qquad (3.33)$$

где m_2 — масса лекарственного препарата в камере 3; k_{12} — константа скорости поступления препарата из камеры $2'$ в камеру 3; k_{21} — константа выведения лекарственного препарата из камеры 3 в камеру $2'$.

Пусть $k_{12} = 0.4$ с$^{-1}$, $k_{21} = 0.6$ с$^{-1}$.

Как и в первой системе, в начальный момент времени в камерах $2'$ и 3 лекарственный препарат отсутствует. Как и в первой системе, как только масса лекарственного препарата в месте введения становится меньше порогового значения ε', засекается отрезок времени $Time_1$, по истечении которого в месте введения уничтожаются все остатки прежней дозы препарата и вводится новая доза $m = m_0$.

Построить модель системы, содержащей обе фармакокинетические модели, не связанные друг с другом, с различными местами введения лекарственного препарата. Также построить модель системы, состоящей из обеих фармакокинетических моделей, имеющих общее место введения препарата. Скорость всасывания лекарственного препарата k_1 в обеих фармакокинетических моделях одинаковая, отрезок времени $Time'$ и пороговое значение ε' для места

введения препарата совпадают со значениями, принятыми для однокамерной фармакокинетической модели с всасыванием.

Задача 12

Даны две биологические популяции, оспаривающие одну и ту же пищу. Пусть это будут популяции медведей (численностью N_1) и волков (численностью N_2) (рис. 3.19).

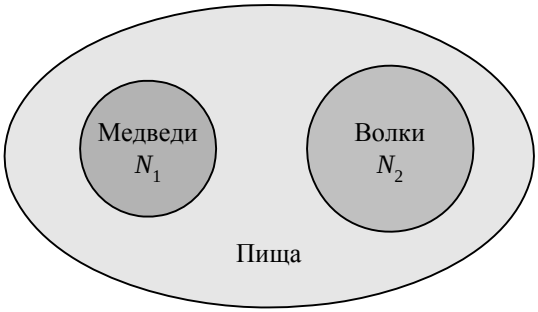


Рис. 3.19. Система двух биологических популяций

Пусть при количестве пищи, достаточном для полного удовлетворения рассматриваемых видов, существуют постоянные положительные коэффициенты прироста популяций: $\varepsilon_1 = 0.7$ мес$^{-1}$ для медведей и $\varepsilon_2 = 0.9$ мес$^{-1}$ для волков. Заданы "коэффициенты прожорливости": $\gamma_1 = 0.7$ кг$^{-1}$ и $\gamma_2 = 0.5$ кг$^{-1}$, соответствующие потребности в пище для каждой из двух популяций.

Пусть $F(N_1, N_2)$ — количество пищи, поедаемой обеими популяциями в единицу времени. Оно задается уравнениями (3.34) и (3.35), где уравнение (3.34) соответствует случаю, когда обе популяции активны, а уравнение (3.35) — когда медведи впадают в спячку. Переключение между режимами (3.34) и (3.35) происходит периодически, причем с режима (3.34) на режим (3.35) переключение происходит через отрезок времени $Time_1 = 9$ мес, а с режима (3.35) на режим (3.35) — через $Time_2 = 3$ мес:

$$F(N_1, N_2) = \lambda_1 \times N_1 + \lambda_2 \times N_2; \tag{3.34}$$

$$F(N_1, N_2) = \lambda_2 \times N_2, \tag{3.35}$$

где λ_1 и λ_2 — некие положительные коэффициенты.

Пусть $\lambda_1 = 0.01$ кг/(мес · шт.), $\lambda_2 = 0.02$ кг/(мес · шт.).

В начальный момент времени популяции обладают начальной численностью $N_1' = 10$ шт. и $N_2' = 20$ шт.

Тогда развитие популяций описывается следующими уравнениями:

- медведи

$$\frac{dN_1}{dt} = [\varepsilon_1 - \gamma_1 \cdot F(N_1, N_2)] \cdot N_1; \tag{3.36}$$

- волки

$$\frac{dN_2}{dt} = [\varepsilon_2 - \gamma_2 \cdot F(N_1, N_2)] \cdot N_2. \tag{3.37}$$

Как только численность той или другой популяции становится меньше 1 (умирает последняя особь), засекается отрезок времени $Time_3 = 3$ мес (если это медведи) или $Time_3' = 5$ мес (если это волки), по истечении которого вместо прежней популяции поселяется новая популяция с новой начальной численностью (N_1', если это медведи, и N_2', если волки).

Построить модель данной системы, а также модель системы, состоящей из двух систем типа "две популяции", не связанных друг с другом. Вторая система "две популяции" идентична первой, за исключением того, что в ней вместо медведей и волков в качестве конкурирующих видов рассматриваются лоси и олени, имеющие иные значения коэффициентов прироста ε_1' и ε_2', другие значения "коэффициентов прожорливости" γ_1' и γ_2', а также иные значения коэффициентов λ_1 и λ_2.

Глава 4

Численная библиотека MvStudium

Начинающий пользователь визуальной среды моделирования редко задумывается над тем, как решаются сформулированные им задачи, потому что в большинстве случаев они решаются достаточно точно и правильно — качественное поведение модели находит разумное объяснение. Более того, если вдруг перестают работать численные методы, то, при внимательном рассмотрении оказывается, что задача плохо сформулирована или допущены ошибки в ее описании. К тому же библиотеки избыточны — всегда можно проверить полученное решение, сравнивая между собой результаты работы нескольких методов. Избыточность хороша и тем, что если отказывает один метод, сохраняется надежда на работоспособность другого. Трудности начинаются потом, когда от успешных предварительных экспериментов с моделью нужно переходить к ее окончательной реализации, когда увеличивается ее размерность, возрастает сложность. Теперь на первый план выступают требования к быстродействию, запрашиваемой памяти, приходится учитывать специфические свойства задачи, и от эффективности кода и численного метода начинает зависеть очень многое.

Среду MvStudium рекомендуется использовать на ранней стадии проектирования, когда требования к эффективности кода менее важны, чем возможность организовать многовариантные расчеты и быстро вносить изменения в модель. Пока численная библиотека MvStudium устраивала пользователей и справлялась с достаточно сложными задачами. Однако всегда наступает момент, когда приходится разбираться, как проводятся вычисления, или усовершенствовать их. Эта глава может подсказать, как лучше организовать вычисления в MvStudium. В то же время, если рассматривать среду MvStudium как численную библиотеку с простым и удобным пользовательским интерфейсом, то ее можно использовать для обучения таким разделам численного анализа, как вычислительные методы линейной алгебры, решение нелиней-

ных алгебраических, обыкновенных дифференциальных и алгебро-дифференциальных уравнений.

Матричные вычисления

Матричные вычисления играют важную роль во многих вычислительных задачах. Существует много пакетов, позволяющих работать с матрицами в простой и удобной форме. Достаточно упомянуть только пакет MATLAB, первоначально создававшийся именно как программа, упрощающая матричные вычисления для пользователей, не знакомых с языком FORTAN. Среду MvStudium можно использовать для матричных вычислений, помня о том, что все-таки основная задача MvStudium — исследовать динамику систем.

Использование MvStudium как матричного калькулятора

В MvStudium определены типы данных: вещественные векторы и матрицы фиксированного (статические) и переменного (динамические) размера. Векторы и матрицы объявляются в языке MVL с помощью операторов, приведенных в листинге 4.1, или с помощью специальных диалогов в окне **Уравнения**.

Листинг 4.1. Объявление векторов и матриц

```
N:integer;
M:integer;
x:vector[N];   -- вектор-столбец постоянного размера
X:matrix[N,M]; -- матрица постоянного размера
xd:vector;     -- вектор-столбец переменного размера
Xd:matrix;     -- матрица переменного размера
```

Над векторами и матрицами можно проводить операции:

- векторные — умножение вектора на число, сложение векторов;
- матричные — умножение матрицы на число, сложение и умножение матриц соответствующих размеров. В отличие от MATLAB, операция "деление" не определена, а матрица размера 1×1 не является одновременно матрицей и вещественным числом.

Векторам можно присвоить значение различными способами:

- задав, например, матричный литерал `[1;2;3]` в качестве значения при описании новой переменной `x:vector[3]` в окне **Уравнения**;
- с помощью оператора явной инициализации `x:=[1;2;3]` (при описании мгновенных действий) или `x=[1;2;3]` (в окне **Уравнения**).

Коэффициенты матриц записываются по строкам `X=[1,2,3; 4,5,6; 7,8,9]` (`X:matrix[3,3]`).

Инициализация также может быть осуществлена с помощью специальных матриц:

- единичной матрицы (`eye(N)`);
- матрицы, все коэффициенты которой равны единице (`ones(N,M)`);
- нулевой матрицы (`zeros(N,M)`),

а также с помощью операторов цикла, в частности, в окне **Уравнения**.

В листинге 4.2 приведен пример инициализации с помощью циклов в окне **Уравнения**. Переменные цикла `i`, `j` являются локальными, их не нужно предварительно описывать. Функция `size()` возвращает число строк или столбцов матрицы, или число координат вектора.

Листинг 4.2. Инициализация матрицы с помощью циклов

```
B = [ for i in 1..size(A,1), j in 1..size(A,2) | A[i,j]**2 ]
```

Матрицы и векторы можно использовать в окне **Уравнения** для записи уравнений в векторно-матричной форме, либо как операнды команд в "мгновенных" входных, выходных и действий на переходах. В обоих случаях можно также использовать библиотечные или пользовательские алгоритмические функции и процедуры.

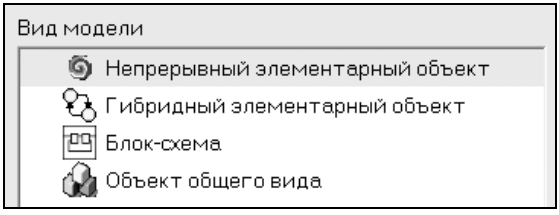
Начнем с использования матричных вычислений в окне **Уравнения**, где описываются длительные действия.

Внимание

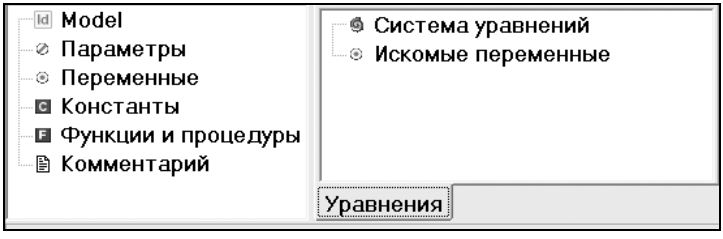
Все вычисления в дальнейшем будут проводиться в окне **Уравнения**, предназначенном для описания непрерывного поведения.

Отсюда вытекают два ограничения.

- Все переменные в окне **Уравнения**, появляющиеся в левой части выражений, должны быть описаны как переменные вещественного типа (`REAL`). Напомним, что выражения, например, `N=size(A,1)`, рассматриваются как уравнения, разрешенные относительно неизвестных (`N`). Здесь `N` — переменная вещественного типа. Если переменная `N` в дальнейшем используется как параметр цикла или как аргумент матричных и векторных функций, то придется использовать операцию `trunk(N)`, чтобы автоматически преобразовать ее к целому типу.
- Поскольку все выражения в окне **Уравнения** рассматриваются как уравнения, последовательность выражений, содержащих одну и ту же переменную в левой части, например, `A=ones(10);A=A*A`, будет считаться переопределенной системой уравнений относительно переменной `A`. Выражение, содержащее одну и ту же переменную и в левой, и в правой частях (`A=2*A`) — как уравнение относительно этой переменной. Если вы не можете обойтись без таких конструкций — вводите дополнительные переменные. Напоминанием, что перед вами уравнения, служит знак равенства, а не присваивания.



а



б

Рис. 4.1. Простейший пользовательский интерфейс:
а — выбор простейшего пользовательского интерфейса; *б* — окно **Уравнения**

При проведении матричных вычислений в окне **Уравнения** проще всего воспользоваться простейшим интерфейсом "Непрерывный элементарный объект" (рис. 4.1, *а*).

На рис. 4.2—4.5 приведены примеры задания векторов и матриц и примеры векторных и матричных вычислений.

```
Model
Параметры
  N: integer := 3;
Переменные
  M_column: matrix[3,1] := [1; 2; 3];
  M_row: matrix[1,3] := [4,5,6];
  N1: double := 0;
  N2: double := 0;
  N3: double := 0;
  s1: double := 0;
  s2: double := 0;
  Vc: vector;
  Vn: vector[N] := [3; 2; 1];
  Vr: vector;
```

Рис. 4.2. Пример описания векторов

```
Система уравнений
    Vc = Vn;
    N1 = size(Vc);
    Vr = transp(Vn);
    N2 = size(Vr,1);
    N3 = size(Vr,2);
    M_row = Vr;
    M_column = Vc;
    s1 = scalar(Vc,Vn);
    — -s2 = scalar( Vr,Vr ) —— Ошибка,Vr - это матрица
Искомые переменные
  Vc, M_column, Vr, M_row, N1, N2, s1, N3
```

Рис 4.3. Пример векторных вычислений в окне **Уравнения**

Обратите внимание на переменную `Vr` (рис. 4.3), описанную как вектор переменного размера. После "присвоения" `Vr=transp(Vn)` ее размер фиксируется, но она становится не вектором-строкой, а матрицей, т. к. в MvStudium нет типа "вектор-строка". Ее размеры с этого момента вычисляются с помощью

операторов `N2=size(Vr, 1)` и `N2=size(Vr, 2)`, в то время как размер вектора `Vc` вычисляется как `N1=size(Vc)`. В приведенном фрагменте использованы две предопределенные операции — транспонирования (`transp()`) и вычисления скалярного произведения векторов (`scalar()`).

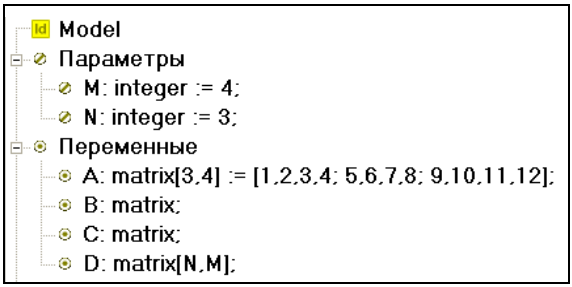


Рис. 4.4. Пример описания матриц

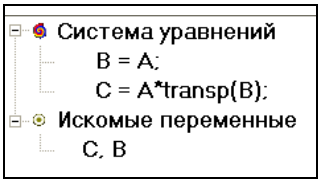


Рис. 4.5. Пример матричных вычислений в окне **Уравнения**

Напомним [5], что выражения и уравнения, расположенные в окне **Уравнения**, предварительно сортируются, для того чтобы получить вычисляемую последовательность. Для того чтобы увидеть вычисляемую последовательность, откройте диалог, установив курсор в окне **Уравнения** и нажав правую кнопку мыши, и выберите команду **Показать уравнения для численного решения**. Вы увидите следующие операторы (листинг 4.3).

Листинг 4.3. Автоматическая сортировка выражений

```
Vc: Vc = Vn;
N1: N1 = size(Vc);
Vr: Vr = transp(Vn);
N2: N2 = size(Vr,1);
N3: N3 = size(Vr,2);
```

```
M_row: M_row = Vr;
M_column: M_column = Vc;
s1: s1 = scalar(Vc,Vn);
```

В каждой строке до разделителя : (двоеточие) стоит имя искомой переменной, а после него — выражение или уравнение, из которого она находится. Вычисляемая последовательность (см. листинг 4.3) в данном случае совпадает с последовательностью, набранной пользователем (рис. 4.3).

Иногда полезно работать непосредственно с текстовым представлением модели, приведенным в листинге 4.4 (см. пример работы с векторами на рис. 4.3).

Листинг 4.4. Класс `Model`, созданный при выборе простейшего интерфейса

```
package ВЕКТОРЫ is
  import SysLib;

  continuous class Model is
    parameter N: integer := 3;
    Vc: vector;
    Vn: vector[N] := [3; 2; 1];
    M_row: matrix[1,3] := [4,5,6];
    M_column: matrix[3,1] := [1; 2; 3];
    N1: double := 0;
    N2: double := 0;
    Vr: vector;
    N3: double := 0;
    s1: double := 0;
    s2: double := 0;
    equations {
      Vc = Vn;
      N1 = size(Vc);
      Vr = transp(Vn);
      N2 = size(Vr,1);
      N3 = size(Vr,2);
      M_row = Vr;
      M_column = Vc;
      s1 = scalar(Vc,Vn);
      --s2 = scalar( Vr,Vr ) ! Ошибка,Vr - это матрица
```

```
      unknown Vc, M_column, Vr, M_row, N1, N2, s1, N3;
    };
  end Model;
  static model: Model;
end ВЕКТОРЫ;
```

Для того чтобы получить доступ к текстовому описанию, выполните команду **Проект | Экспорт в MVL...**, расположенную на командной панели редактора моделей. Отредактируйте текст. Сохраните новое содержимое файла. Выполните команду **Проект | Импорт в MVL...**, и, если редактирование прошло удачно, пакет автоматически преобразует текстовое описание модели в графическое.

Задача 1

Пусть даны вещественная матрица $\mathbf{A}$ и ее обратная $\mathbf{A}^{-1}$. Предположим, что коэффициенты матрицы $\mathbf{A}^{-1}$ могут быть вычислены явно, по заданным формулам. Такие матрицы будем называть *тестовыми*. Попробуем вычислить обратную для тестовой матрицы как численное решение матричного уравнения $\mathbf{AX} = \mathbf{E}$, где $\mathbf{E}$ — единичная матрица, и сравнить ее с "точным" значением, вычисленным по формулам. Операция численного обращения квадратной невырожденной ($n \times n$)-матрицы существует практически в любом пакете, ее традиционно обозначают как `inv()`. Предположим, что за ней скрыто решение матричного уравнения $\mathbf{AX} = \mathbf{E}$, столбец за столбцом, $\mathbf{Ax}_i = \mathbf{e}_i$, $i = 1, ..., n$, методом Гаусса, с однократным разложением матрицы $\mathbf{A}$ на множители.

Точность численного обращения определяется числом обусловленности матрицы $\mathbf{A}$ по отношению к задаче обращения $cond(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$. Предположим, что $\|\mathbf{x}\| = \sum_i |x_i|$ и $\|\mathbf{A}\| = \max_j \|\mathbf{a}_j\|$. Известно, что ошибка вычисления обратной матрицы определяется выражением

$$\|\mathbf{AX} - \mathbf{E}\| \le \sigma_n \|\mathbf{A}\| \cdot \|\mathbf{X}\| \cdot \delta_m ,$$

где δ_m — единичная ошибка округления; σ_n — заданная функция, зависящая от размера матрицы. Как утверждают разработчики пакета Linpack,

функция σ_n может быть аппроксимирована выражением $\sigma_n \approx 0.05n + \beta$, где β — некоторая константа.

Попробуем проверить теоретические выводы на практике. Сформулируем задание.

Средствами изучаемого пакета построить графики функций:

- $T = T(n)$, $T(n)$ — время, затраченное на обращения матрицы, n — порядок матрицы;
- $F = F(cond)$, $F = \dfrac{\left\|\mathbf{A}^{-1} - \mathbf{X}\right\|}{\left\|\mathbf{A}^{-1}\right\|}$ — зависимость относительной ошибки численного решения `X=inv(A)` от числа обусловленности матрицы.

Построить аппроксимацию Fa функции $F = F(cond)$ по методу наименьших квадратов, выбрав в качестве аппроксимирующей функции полином не выше второй степени $Fa = c_2 \cdot cond^2 + c_1 \cdot cond + c_0$.

Рассмотрим, как следует выполнять работу, на следующем примере. Выберем тестовую матрицу [16]:

$$\mathbf{A} = \begin{bmatrix} \alpha & 1 & & & 0 \\ 1 & \alpha & 1 & & \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ & & 1 & \alpha & 1 \\ 0 & & & 1 & \alpha \end{bmatrix}$$

порядка n, для нее известна обратная матрица

$$\mathbf{A}^{-1} = \frac{1}{a_n} \begin{bmatrix} a_0 a_{n-1} & -a_0 a_{n-2} & a_0 a_{n-3} & . \\ -a_0 a_{n-2} & a_1 a_{n-2} & -a_1 a_{n-3} & . \\ a_0 a_{n-3} & -a_1 a_{n-3} & a_2 a_{n-3} & . \\ -a_0 a_{n-4} & a_1 a_{n-4} & -a_2 a_{n-4} & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix},$$

где:

$$a_0 = 1,$$
$$a_1 = \alpha,$$
$$a_i = a \cdot a_{i-1} - a_{i-2}, \quad i \geq 2.$$

Если выбрать значение параметра α в виде $\alpha = 2\cos\varphi$, то рекуррентные соотношения заменяются формулами

$$a_i = \frac{\sin(i+1)\varphi}{\sin\varphi}, \quad i \geq 0.$$

Рассмотрим по отдельности два независимых вычислительных эксперимента. В первом будем изучать зависимость ошибки вычисления обратной матрицы от числа обусловленности, а во втором — зависимость времени вычисления от размера матрицы.

Для проведения первого эксперимента нужно уметь строить такую последовательность коэффициентов $\alpha_1, \ldots, \alpha_k$ чтобы последовательность $cond(A(\alpha_1)), \ldots, cond(A(\alpha_k))$ была строго возрастающей при заданном размере матрицы. Для данной тестовой матрицы известны собственные числа [16] $\lambda_j = \alpha - 2 \cdot \cos\left(\frac{k\pi}{n+1}\right)$, $j = 1, \ldots, n$ и можно попытаться использовать эти формулы для построения нужной последовательности.

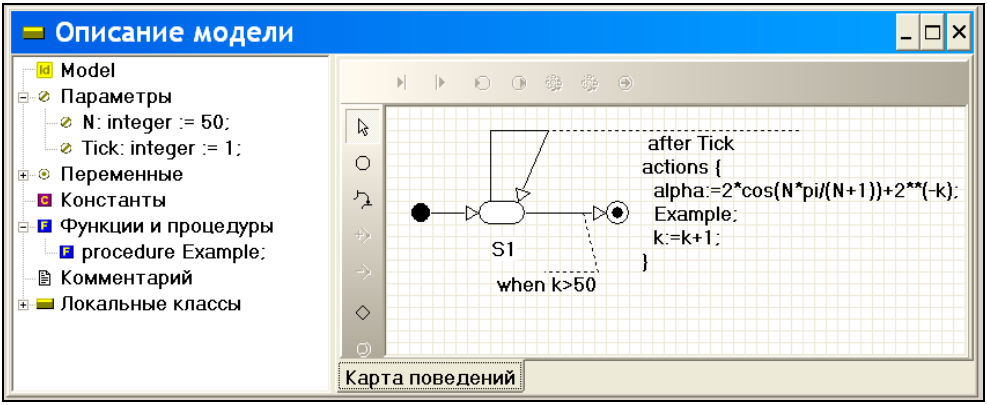


Рис. 4.6. Описание модели
для поведения простейшего вычислительного эксперимента

Простейший вычислительный эксперимент может быть реализован с помощью примитивного гибридного автомата (рис. 4.6).

Гибридный автомат здесь играет особую роль. Он вводит дискретное время, каждый тик которого изменяет значение параметра α, что приводит к построению последовательности $\alpha_1, ..., \alpha_k$. При последнем значении $\alpha_k = 2 \cdot \cos\left(\frac{N \cdot \pi}{N+1}\right) + 2^{-k}$ у рассмотренной тестовой матрицы есть собственное число, приблизительно равное 2^{-k}, и число обусловленности матрицы будет большим.

На рис. 4.7 представлено описание модели.

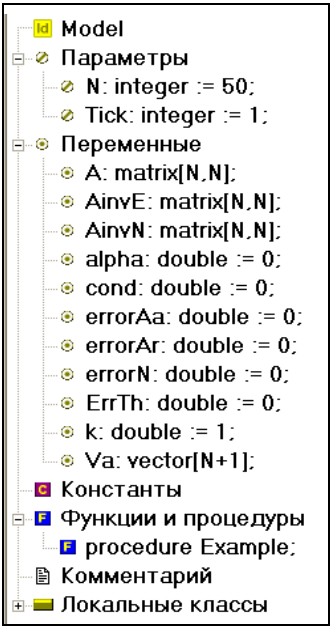


Рис. 4.7. Описание модели

На рис. 4.7 использованы обозначения: `A` — исходная тестовая матрица; `AinvN` — обратная матрица, найденная численно; `AinvE` — матрица, вычисленная по формулам; `cond` — число обусловленности, абсолютная `ErrorAa` и относительная `ErrorAr` ошибки и теоретическая оценка ошибки `ErrorTh`, определяемые выражениями из листинга 4.5.

Листинг 4.5. Использование встроенных функций для вычисления ошибок

```
errorN:=abs(norm2(A*AinvN-eye(N)));
cond:=norm2(A)*norm2(AinvE)/c;
errorAr:=norm2(AinvE-AinvN)/norm2(AinvE);
errorAa:=norm2(AinvE-AinvN)/c;
ErrTh:=0.05*N*norm2(A)*norm2(AinvE)*2**(-52);
```

Константа `c:=1.0E+16` выбрана как нормировочный множитель для построения графиков и равна максимально возможному числу обусловленности тестовых матриц в данной серии экспериментов.

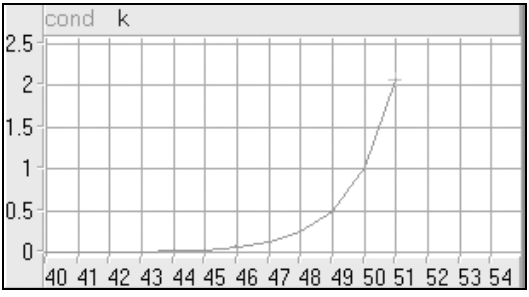
Полный текст процедуры `Example`, вычисляющей тестовую матрицу и обратную для нее, приведен в листинге 4.6.

Листинг 4.6. Процедура `Example`

```
procedure Example is
  ki: integer;
  kj: integer;
  c: double;
begin
  c:=1.0E+16;
  ki:=1;
  kj:=1;
  A:=zeros(N,N);
  A[1,2]:=1;
  A[1,1]:=alpha;
  for i in 2..N-1 loop
    A[i,i-1]:=1;
    A[i,i]:=alpha;
    A[i,i+1]:=1;
  end loop;
  A[N,N-1]:=1;
  A[N,N]:=alpha;
  Va[1]:=1;
  Va[2]:=alpha;
  for i in 3..N+1 loop
    Va[i]:=alpha*Va[i-1]-Va[i-2];
  end loop;
```

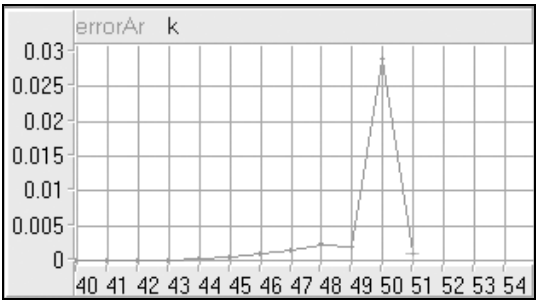
```
  ki:=1;
  for i in 1..N loop
    kj:=ki;
    for j in 1..i loop
      AinvE[i,j]:=kj*Va[j]*Va[N+1-i];
      kj:=-kj;
    end loop;
    ki:=-ki;
  end loop;
  for i in 1..N loop
    for j in i+1..N loop
      AinvE[i,j]:=AinvE[j,i];
    end loop;
  end loop;
  AinvE:=1/Va[N+1]*AinvE;
  AinvN:=inv(A);
  errorN:=abs(norm2(A*AinvN-eye(N)));
  cond:=norm2(A)*norm2(AinvE)/c;
  errorAr:=norm2(AinvE-AinvN)/norm2(AinvE);
  errorAa:=norm2(AinvE-AinvN)/c;
  ErrTh:=0.05*N*norm2(A)*norm2(AinvE)*2**(-52);
end Example;
```

На рис. 4.8 представлены результаты эксперимента с учетом нормировки.

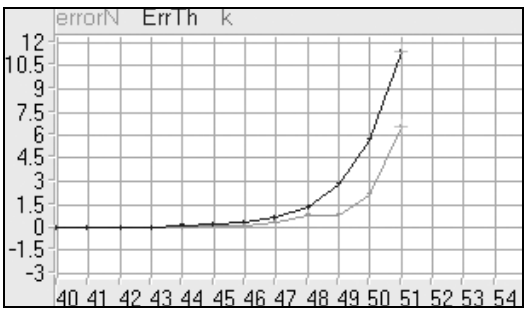


а

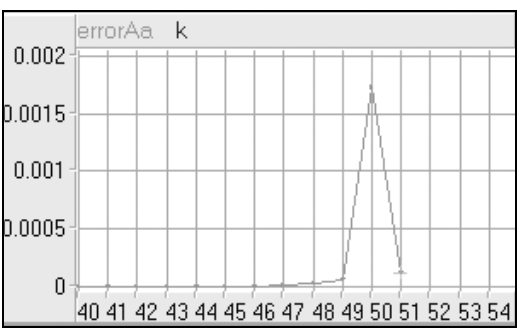
Рис. 4.8. Результаты экспериментов:
а — зависимость числа обусловленности от параметра *k*



б

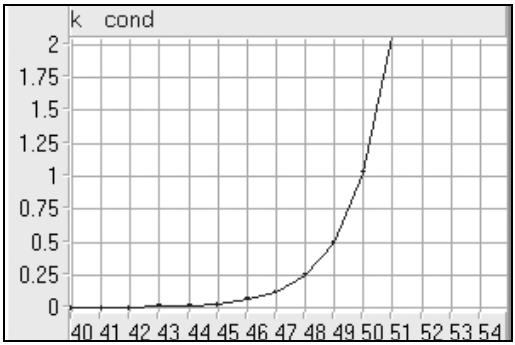


в

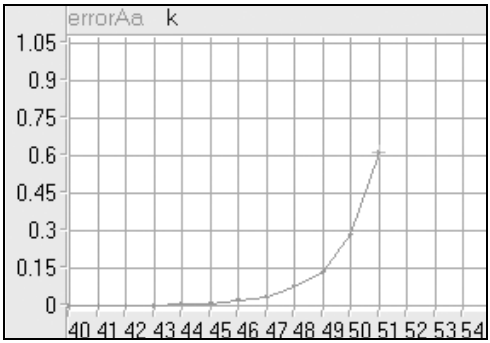


г

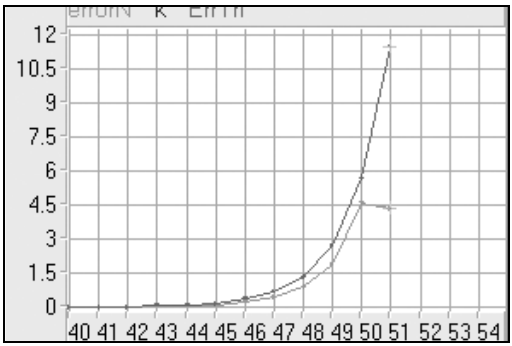
Рис. 4.8. Результаты экспериментов:
б — зависимость относительной ошибки от параметра *k*;
в — сравнение теоретического предсказания ошибки и результатов численных расчетов;
г — зависимость абсолютной ошибки от параметра *k*



а

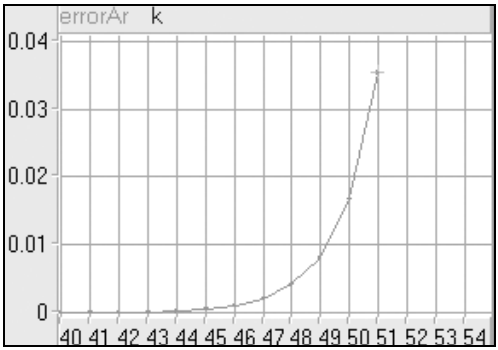


б



в

Рис. 4.9. Результаты экспериментов для обратной матрицы, взятой в качестве исходной:
а — зависимость числа обусловленности от параметра *k*;
б — зависимость относительной ошибки от параметра *k*;
в — сравнение теоретического предсказания ошибки и результатов численных расчетов



г

Рис. 4.9. Результаты экспериментов для обратной матрицы, взятой в качестве исходной: *г* — зависимость абсолютной ошибки от параметра k

Для проверки проведем эксперименты с обратной матрицей, рассматривая ее как исходную (рис. 4.9).

Как видно, результаты вычислений качественно не меняются.

Вопросы

1. Чем объясняется различие результатов в первом и втором экспериментах?
2. Как будет выглядеть карта поведения, если надо провести оба эксперимента одновременно или сразу для нескольких матриц?
3. Что надо изменить в программе, если необходимо менять не только значение параметра k, но и порядок матрицы?

Построим модель для проведения второй серии экспериментов по оценке временной сложности алгоритма вычисления обратной матрицы (рис. 4.10).

Помимо уже использованных переменных, появились вспомогательные переменные, необходимые для вычисления времени нахождения обратной матрицы с помощью алгоритма Гаусса. Вычисление времени происходит, как и в любой другой программе, использующей стандартную функцию `Clock` (рис. 4.11). Функция `Clock` запускается в начале эксперимента и фиксирует начало отсчета времени (это значение хранится в переменной `TB`). В нашем случае это выполняется в секции входных действий состояния `S1`. В самом состоянии ничего не происходит, и после истечения времени `Tick` автомат

покидает текущее состояние. Выходные действия также отсутствуют, а вот при срабатывании перехода происходит вызов процедуры `Example`, а затем функция `Clock` вызывается повторно. Переменная `Tend` хранит время, затрачиваемое на вычисление прямой и обратной матриц. Если пренебречь временем вычисления самой тестовой матрицы, то можно считать, что значение `Tend` и есть время численного нахождения обратной матрицы. К сожалению, это не так, поскольку операционная система Windows — многозадачная система. Именно поэтому в нашем эксперименте мы вычисляем среднее значение (рис. 4.12). Как и следует ожидать, в каждый момент загрузка машины различна, хотя решается только одна пользовательская задача.

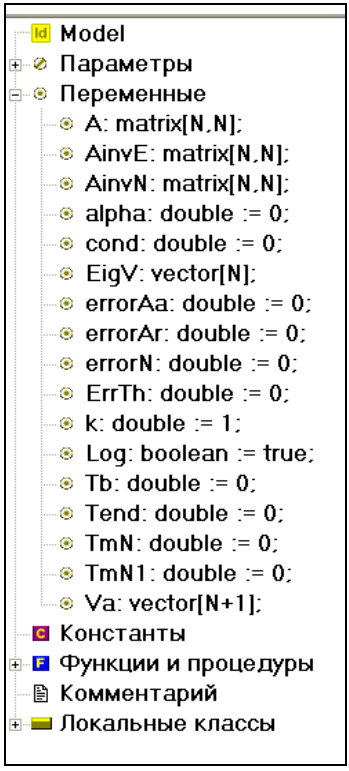


Рис. 4.10. Описание модели для оценки временной сложности

Обратите внимание, что нигде не упомянуто о единицах изменения времени. Абсолютное время не столь важно по сравнению с зависимостью времени от размера матрицы, которая известна из теории. Тем более, что в

данном случае целью было показать, как проводить такого сорта эксперименты в MvStudium.

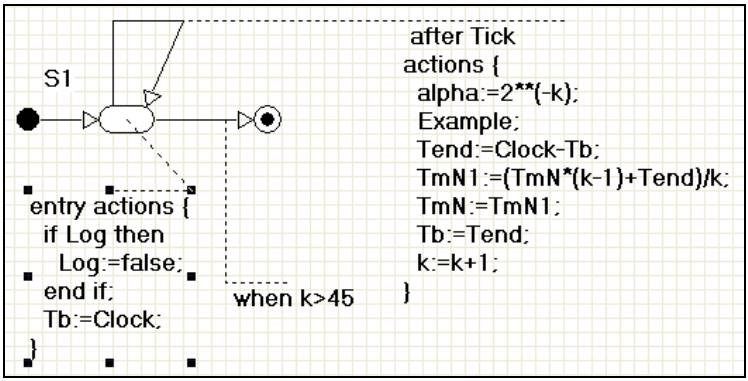
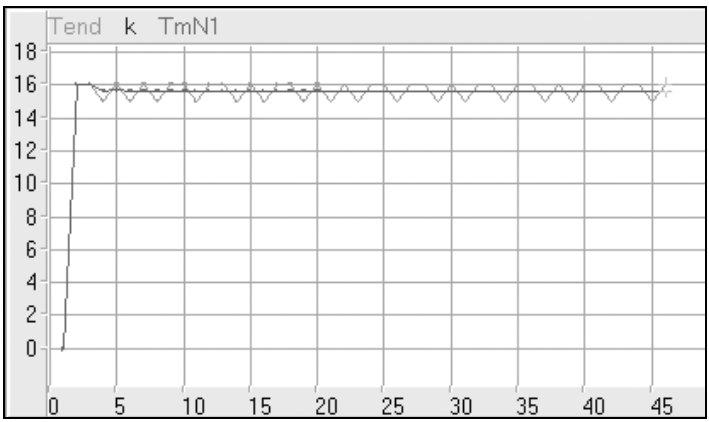


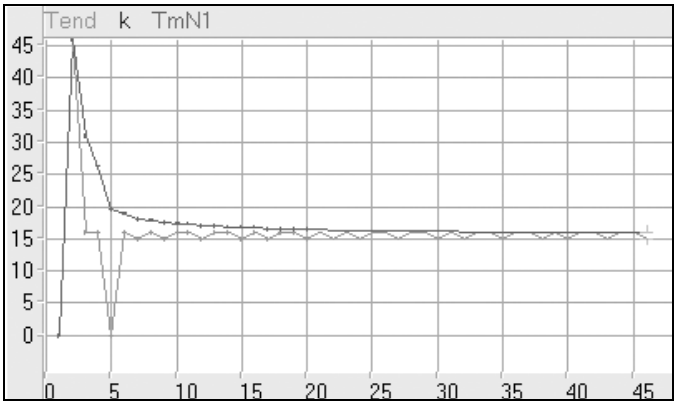
Рис. 4.11. Описание эксперимента по оценке временной сложности

Осталось пояснить, откуда в построенной модели появилась функция `Clock`, которой нет среди встроенных функций пакета MvStudium. Эта функция принадлежит динамически присоединяемой библиотеке (DLL), которая была создана специально для проведения этого эксперимента.



а

Рис. 4.12. Результаты экспериментов по оценке временной сложности: графики *а* и *б* — это результат двух последовательных экспериментов с одной и той же матрицей. Средние значения у них одинаковые



б

Рис. 4.12. Результаты экспериментов по оценке временной сложности: графики *а* и *б* — это результат двух последовательных экспериментов с одной и той же матрицей. Средние значения у них одинаковые

Стандартная функция `Clock` — это стандартная функция языка FORTAN, помещенная в DLL. Подсоединение функции к пакету происходит так, как показано на рис. 4.13.

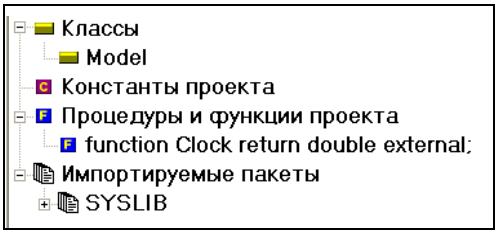


Рис. 4.13. Глобальные функции проекта

В описании всего проекта, как видно из рис. 4.13, помимо списка классов, есть глобальные функции. В нашем случае это функция:

```
function Clock return double is
  external "ClockDLL.dll";
```

описанная как внешняя, с заданным интерфейсом и указанием имени DLL, где ее следует искать. Сама DLL располагается в корневом каталоге конкретного проекта.

Задание

Используя приведенные выше модели, решите задачу 1.

Упражнения

1. Напишите на языке MVL собственную программную реализацию метода Гаусса и оцените ее эффективность.
2. Подсоедините к MvStudium любую из библиотек с программными реализациями численных методов решения систем линейных алгебраических уравнений, реализуйте их аналоги на языке MVL, сравните их эффективность на тестовом наборе из книги [16].

Задача 2

Решить заданную систему линейных обыкновенных дифференциальных уравнений с постоянной вещественной квадратной матрицей $\mathbf{A}$ и вектором $\mathbf{b}$:

$$\frac{d\mathbf{s}}{dt} = \mathbf{A} \cdot \mathbf{s} + \mathbf{b}; \quad \mathbf{s}(0) = \mathbf{x}_0; \quad t \in [0, T].$$

Найти устойчивое решение, если оно существует, оценить погрешность для различных численных методов.

Замечание для преподавателя

Решение приведенной системы можно записать в виде

$$\mathbf{s}(t) = e^{\mathbf{A}t}\mathbf{s}_0 + \mathbf{A}^{-1} \cdot \left(e^{\mathbf{A}t} - \mathbf{E}\right) \cdot \mathbf{b}, \quad \det(\mathbf{A}) \neq 0.$$

Предположим, что существует квадратная матрица $\mathbf{S}$, приводящая исходную матрицу $\mathbf{A}$ к вещественной матрице $\mathbf{\Lambda}_{\text{простой_формы}}$ "простой" формы, $\mathbf{S}^{-1}\mathbf{A}\mathbf{S} = \mathbf{\Lambda}_{\text{простой_формы}}$, обе матрицы $\mathbf{A}, \mathbf{S} \in \Re^{n \times n}$ вещественны и их размер n равен двум.

Возможными формами матрицы $\mathbf{\Lambda}_{\text{простой_формы}}$ будут:

- $\mathbf{\Lambda}_{\text{простой_формы}} = \mathbf{\Lambda}_R = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix};$

- $\mathbf{\Lambda}_{\text{простой_формы}} = \mathbf{\Lambda}_d = \begin{bmatrix} \lambda_0 & 0 \\ 0 & \lambda_0 \end{bmatrix}$;
- $\mathbf{\Lambda}_{\text{простой_формы}} = \mathbf{\Lambda}_G = \begin{bmatrix} \lambda_0 & 1 \\ 0 & \lambda_0 \end{bmatrix}$;
- $\mathbf{\Lambda}_{\text{простой_формы}} = \mathbf{\Lambda}_C = \begin{bmatrix} \mu & \nu \\ -\nu & \mu \end{bmatrix}$.

Зная матрицу $\mathbf{S}$, легко конструировать тестовые примеры с известным решением. Взяв за основу уравнение

$$\frac{d\mathbf{z}}{dt} = \mathbf{\Lambda}_{\text{простой_формы}} \cdot \mathbf{z} + d; \quad \mathbf{z} = \mathbf{S}^{-1} \cdot \mathbf{s}; \quad d = \mathbf{S}^{-1} \cdot \mathbf{b},$$

с заданной матрицей $\mathbf{\Lambda}_{\text{простой_формы}}$, можно построить матрицу $\mathbf{A}$ и вектор $\mathbf{b}$. В качестве матрицы $\mathbf{S}$, можно использовать матрицу $\mathbf{S} = \mathbf{E} + \mathbf{u} \cdot \mathbf{v}^T$. Если $\mathbf{u}^T \cdot \mathbf{v} = 0$, то $\mathbf{S}^{-1} = \mathbf{E} - \mathbf{u} \cdot \mathbf{v}^T$.

Решение.

Пусть матрица $\mathbf{A}$ равна $\mathbf{A} = \begin{bmatrix} -37 & 108 \\ -9 & 26 \end{bmatrix}$, вектор $\mathbf{b} = \begin{bmatrix} -10 \\ +10 \end{bmatrix}$; начальные условия $\mathbf{x}_0 = \begin{bmatrix} +10 \\ -10 \end{bmatrix}$, промежуток интегрирования — $t \in [0, 10]$. Матрица $\mathbf{A}$ подобна матрице $\mathbf{\Lambda}_{\text{простой_формы}} = \mathbf{\Lambda}_R = \begin{bmatrix} -1 & 0 \\ 0 & -10 \end{bmatrix}$, т. к. построена по формуле $\mathbf{A} = \mathbf{S} \cdot \mathbf{\Lambda}_{\text{простой_формы}} \cdot \mathbf{S}^{-1}$; вектор $\mathbf{u} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$; вектор $\mathbf{v} = [1, -2]$.

Вычисляем собственные числа матрицы $\mathbf{A}$ и убеждаемся, что они равны $\lambda_1 = -1$, $\lambda_2 = -10$. Устойчивое решение существует и равно

$$\mathbf{s}(t)\big|_{t\to\infty} = \left(e^{\mathbf{A}t}\mathbf{s}_0 + \mathbf{A}^{-1} \cdot \left(e^{\mathbf{A}t} - \mathbf{E}\right) \cdot \mathbf{b}\right)\Big|_{t\to\infty} = -\mathbf{A}^{-1} \cdot \mathbf{b}.$$

Решение системы можно представить и в форме $\mathbf{s}(t) = \mathbf{S} \cdot \mathbf{z}(t)$,

$$\mathbf{z}(t) = \begin{bmatrix} e^{\lambda_1 t} z_1(0) + \dfrac{d_1}{\lambda_1}\left(e^{\lambda_1 t} - 1\right) \\ e^{\lambda_2 t} z_1(0) + \dfrac{d_2}{\lambda_2}\left(e^{\lambda_2 t} - 1\right) \end{bmatrix}.$$

Последняя формула позволяет оценить погрешность численного решения.

Этот пример (листинг 4.7) ничем не отличается от предыдущего, однако на нем хорошо видны преимущества представления уравнений в графической форме (рис. 4.14).

Листинг 4.7. Решение дифференциальных уравнений, записанных в векторно-матричной форме

```
package дифф_ур is
  import SysLib;
  continuous class Model is
    parameter N: integer := 2;
    parameter s0: vector := [20; 10]; -- [-10,10]
    parameter Kr: double := 1E6;
    A: matrix := S*[-1,0; 0,-10]*Sinv;
    b: vector := [20; 10]; -- [-10,10]
    s: vector := s0;
    z: vector;
    d: vector;
    S: matrix := u*v+eye(N);
    Sinv: matrix := (-1)*u*v+eye(N);
    u: matrix := [2; 1];
    v: matrix := [1,-2];
    Lambda: matrix;
    s_stable: vector := (-1)*inv(A)*b;
    z0: vector;
    s_exect: vector;
    Error_a: double := 0;
    Error_r: double := 0;
    w1: double := 0;
    w2: double := 0;
    equations {
```

```
      z0 = Sinv*s0;
      d = Sinv*b;
      Lambda = Sinv*A*S;
      w1=
exp(Lambda[1,1]*Time)*z0[1]+d[1]/Lambda[1,1]*(exp(Lambda[1,1]*Time)-1);
      w2=
exp(Lambda[2,2]*Time)*z0[2]+d[2]/Lambda[2,2]*(exp(Lambda[2,2]*Time)-1);
      z = [w1; w2];
      s_exect = S*z;
      d(s)/dt = A*s+b;
      Error_r = vnorm2(s-s_exect)/vnorm2(s_exect)*Kr;
      Error_a = vnorm2(s-s_exect)*Kr;
      unknown s, z0, d, Lambda, w1, w2, z, s_exect, Error_r, Error_a;
    };
  end Model;

  static model: Model;

end дифф_ур;
```

z0 = Sinv·s0

d = Sinv·b

Lambda = Sinv·A·S

$$w1 = e^{Lambda_{1,1}\cdot Time}\cdot z0_1 + \frac{d_1}{Lambda_{1,1}}\cdot\left(e^{Lambda_{1,1}\cdot Time}-1\right)$$

$$w2 = e^{Lambda_{2,2}\cdot Time}\cdot z0_2 + \frac{d_2}{Lambda_{2,2}}\cdot\left(e^{Lambda_{2,2}\cdot Time}-1\right)$$

z = [w1; w2]

s_exect = S·z

$$\frac{ds}{dt} = A\cdot s+b$$

$$Error_r = \frac{\mathbf{vnorm2}(s-s_exect)}{\mathbf{vnorm2}(s_exect)}\cdot Kr$$

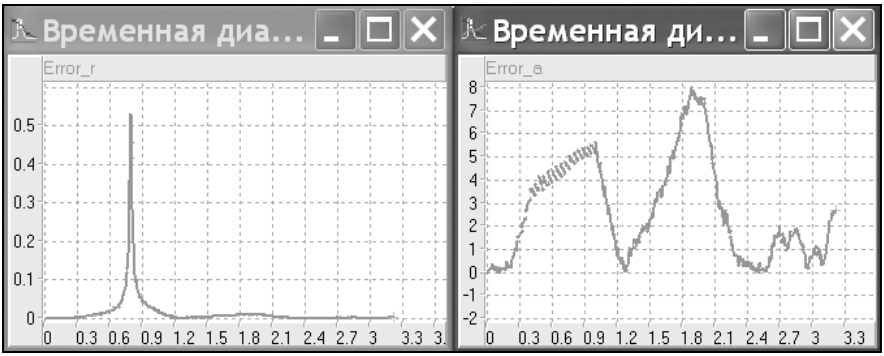
$$Error_a = \mathbf{vnorm2}(s-s_exect)\cdot Kr$$

Рис. 4.14. Графическая форма дифференциальных уравнений

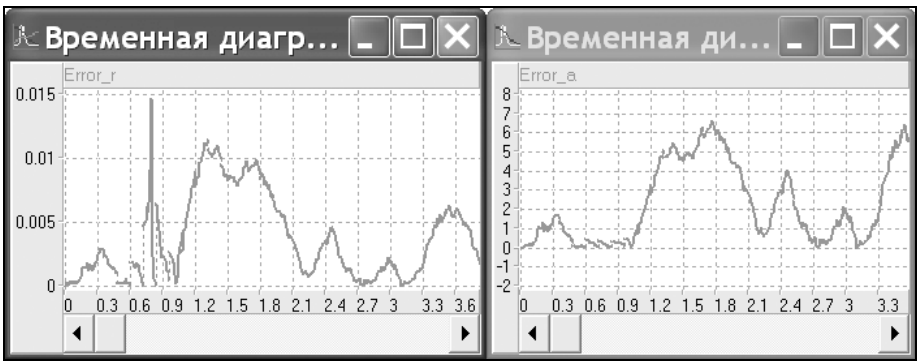
Проведем несколько экспериментов с программными реализациями численных методов, используемых в MvStudium. При анализе результатов экспериментов надо, конечно же, постоянно помнить о простоте тестовых примеров. Выберем программную реализацию, например, методов Рунге—Кутты и построим графики ошибки.

В MvStudium графики рисуются одновременно с воспроизведением поведения, а масштаб изменения переменных обычно неизвестен заранее. Это может вызывать трудности, как, например, в нашем случае. В начальный момент величина ошибки равна нулю, и по умолчанию максимальное значение для такой переменной для отображения на графике будет выбрано равным единице. Для последующих экспериментов параметр `Kr=1.E6` положим равным обратной величине точности, устанавливаемой при работе с численными методами на испытательном стенде (команда **Установки | Численные методы**, вкладка **Погрешности**). По умолчанию в MvStudium выбрано значение `1.E-6` как для относительной, так и для абсолютной локальной погрешности для любого численного метода. В этом случае значение масштабированной погрешности на графике, равное единице, будет означать идеальную характеристику алгоритма выбора шага и оценки погрешности: запрашиваемая и реализуемая погрешности совпадают. Заранее скажем, что на простейших тестовых примерах придется выбрать `Kr=1.E12`, что говорит о завышенных вычислительных затратах. На рис. 4.15 приведены графики решения и его производных для некоторых выбранных отрицательных начальных условий с тем, чтобы решение при стремлении к положительному устойчивому решению пересекало ноль.

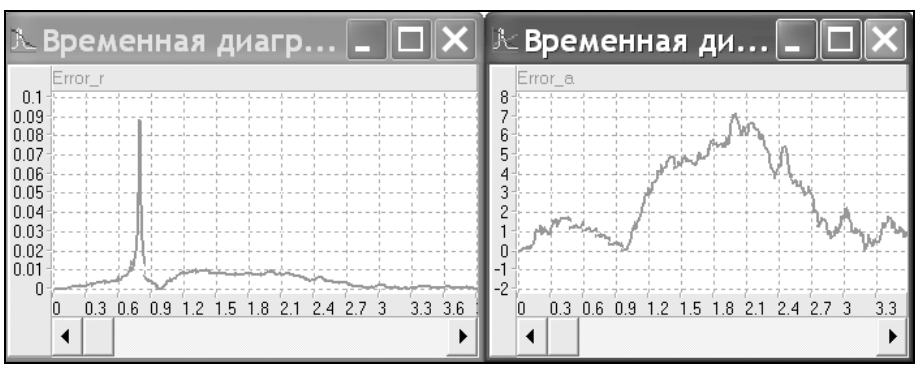
Рис. 4.15. Решение и производная решения тестовой задачи



а



б



в

Рис. 4.16. Экспериментально найденные погрешности (относительная и абсолютная) при различных значениях запрашиваемых погрешностей:

а — $\varepsilon_r = \varepsilon_a = 1.0e^{-3}$; *б* — $\varepsilon_r = \varepsilon_a = 1.0e^{-6}$; *в* — $\varepsilon_r = \varepsilon_a = 1.0e^{-12}$

На рис. 4.16 показаны эксперименты с методом Рунге — Кутты, программа DOPRI853.

Упражнения

Решение задачи 2 можно представить и в другой форме. Введем положительную величину H. Используя формулу $\mathbf{s}(t) = e^{\mathbf{A}t}\mathbf{s}_0 + \mathbf{A}^{-1}\cdot(e^{\mathbf{A}t} - \mathbf{E})\cdot\mathbf{b}$, $\det(\mathbf{A}) \neq 0$, можно вычислить значение $\mathbf{s}(t+H)$, выраженное через $\mathbf{s}(t)$. Действительно,

$$\begin{aligned}\mathbf{s}(t+H) &= e^{\mathbf{A}(t+H)}\mathbf{s}_0 + \mathbf{A}^{-1}\cdot\left(e^{\mathbf{A}(t+H)} - \mathbf{E}\right)\cdot\mathbf{b} = \\ &= e^{\mathbf{A}H}e^{\mathbf{A}t}\mathbf{s}_0 + \mathbf{A}^{-1}\cdot\left(e^{\mathbf{A}H}e^{\mathbf{A}t} - \mathbf{E}\right)\cdot\mathbf{b},\end{aligned}$$

откуда следует

$$\mathbf{s}(t+H) = e^{\mathbf{A}H}\cdot\mathbf{s}(t) + \mathbf{A}^{-1}\cdot\left(e^{\mathbf{A}H} - \mathbf{E}\right)\cdot\mathbf{b}\,.$$

Можно ли использовать эту формулу построения решения исходного дифференциального уравнения?

В этой задаче основная проблема связана с вычислением матричной экспоненты $e^{\mathbf{A}H}$ и матричной функции $\mathbf{A}^{-1}\cdot\left(e^{\mathbf{A}H} - \mathbf{E}\right)$. Разложение в ряд матричных функций $e^{\mathbf{A}H} = \mathbf{E} + \mathbf{A}H + \frac{1}{2}(\mathbf{A}H)^2\ldots$, $\Phi(H) = \mathbf{A}^{-1}\cdot\left(e^{\mathbf{A}H} - \mathbf{E}\right) = H\left(\mathbf{E} + \frac{1}{2}\mathbf{A}^2H + \ldots\right)$ может оказаться весьма трудоемкой операцией. В работе [17] предлагалось использовать следующий алгоритм вычисления матричной экспоненты. Выберем достаточно малый шаг используем отрезок ряда $e^{\mathbf{A}h} \approx \mathbf{E} + \mathbf{A}h + \frac{1}{2}(\mathbf{A}h)^2$ в качестве начального приближения к матричной экспоненте и воспользуемся соотношением $e^{\mathbf{A}(h+h)} = e^{\mathbf{A}h}\cdot e^{\mathbf{A}h}$ для построения $e^{\mathbf{A}H}$. Аналогично можно построить $\Phi(2h) = \left(2\mathbf{E} + \Phi(h)\right)\cdot\Phi(h)$, а затем и $\Phi(H)$.

Прежде чем решать всю задачу, напишем и отладим процедуры для вычисления матричной экспоненты $e^{\mathbf{A}H}$ и функции $\Phi(H)$ (листинг 4.8).

Листинг 4.8. Вычисление матричной экспоненты

```
package Системный_метод_начало is
  import SysLib;
  private hybrid class Model is
    parameter N: integer := 2;
    parameter H: double := 1;
    A: matrix := [-1,0; 0,-10];
    b: vector;
    ExpA: matrix;
    PhiA: matrix;
    w1: double := 0;
    w2: double := 0;
    k: integer := 0;
    h: double := 0;
    w3: double := 0;
    w4: double := 0;
    function mexp_A(in A: matrix; in H: double)
     return matrix is
      expA: matrix;
    begin
      expA:=eye(size(A,1))+h*A*(eye(size(A,1))+h/2*A);
      for i in 1..k loop
        expA:=expA*expA;
      end loop;
      return expA;
    end mexp_A;
    procedure mexp_h(in A: matrix; in H: double) is
      wh: double;
      i: integer;
      j: integer;
    begin
      h:=H;
      wh:=0;
      for i in 1..size(A,1) loop
        for j in 1..size(A,2) loop
          wh:=wh+A[i,i]**2;
        end loop;
```

```
  end loop;
  k:=0;
  if wh/=0 then
    wh:=1/sqrt(wh);
  end if;
  while h>=wh loop
    h:=h/2;
    k:=k+1;
  end loop;
end mexp_h;
function mexp_Phi(in A: matrix; in H: double)
 return matrix is
  PsiA: matrix;
begin
  PsiA:=h*(eye(size(A,1))+h/2*A*(eye(size(A,1))+h/6*A));
  for i in 1..k loop
    PsiA:=PsiA*(2*eye(size(A,1))+A*PsiA);
  end loop;
  return PsiA;
end mexp_Phi;
bchart {
  initial state _InitState;
  final state _FinalState;
  transition _InitTran from _InitState to S1;
  transition Tr_1 from S1 to _FinalState
actions {
      mexp_h(A,H);
      ExpA:=mexp_A(A,H);
      PhiA:=mexp_Phi(A,H);
      w1:=exp(-1);
      w2:=exp(-10);
      w3:=-1*(exp(-1)-1);
      w4:=-1/10*(exp(-10)-1);
    }
  ;
    static model: Model;
end Системный_метод_начало;
```

Этой программе соответствует карта поведения, представленная на рис. 4.17.

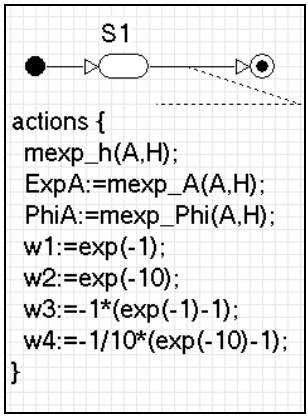


Рис. 4.17. Карта поведения модели
для разработки матричных процедур

В этой программе специально задача настолько упрощена, насколько это возможно: в качестве тестовой матрицы выбрана диагональная, будущий шаг интегрирования H является степенью двойки, для построения начального приближения к искомым матричным функциям в ряде Тейлора учтены только два члена.

Процедура `procedure mexp_h(in A: matrix; in H: double)` вычисляет начальное значение шага `h`, используя оценку максимального по модулю собственного числа матрицы `A`. Одновременно вычисляется количество пересчетов матриц `k`. Обе переменные `h` и `k` являются глобальными по отношению к переменным процедуры и используются в дальнейшем в функциях `mexp_A` и `mexp_Phi`. Формальный параметр `A` в процедуре и функциях объявлен как матрица переменного размера, чтобы их можно было использовать в дальнейшем для матриц любого размера. Матричные функции вычисляются с помощью процедур:

```
function mexp_A(in A: matrix; in H: double) return matrix
function mexp_Phi(in A: matrix; in H: double)return matrix
```

В качестве тестовой выбрана диагональная матрица

```
A: matrix := [-1,0; 0,-10]
```

для которой матричная экспонента равна

```
ExpA: matrix := [exp(-1),0; 0, exp(-10) ]
```

(переменные `w1` и `w2` в модели для диагональных элементов), а функция $\Phi(H)$ записывается в виде

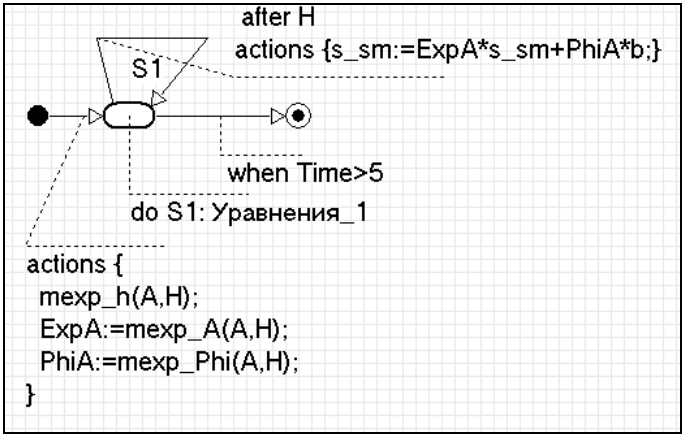
```
PhiA: matrix := [-1*(exp(-1)-1),0; 0, -1/10*(exp(-1/10)-1)] ]
```

(переменные `w3` и `w4` в модели).

Первоначальные вполне удовлетворительные результаты по точности позволяют перейти к дальнейшим исследованиям.

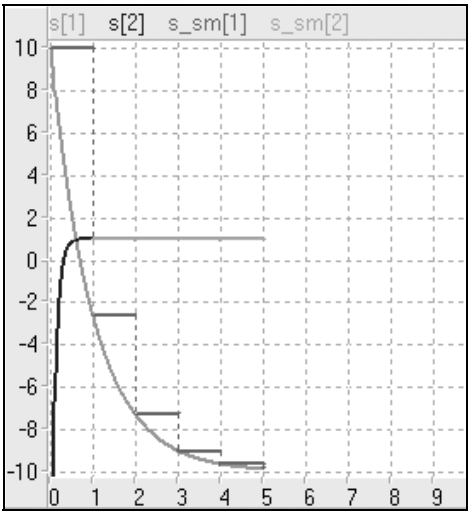
Сохраним модель под именем `Системный_метод_начало` и создадим в той же папке новый вариант модели `Системный_метод_продолжение` (команда **Проект | Сохранить как...** командного меню). Карта поведения нового проекта показана на рис. 4.18, *а*.

Отличие нового проекта от старого в том, что здесь одновременно численно решаются и дифференциальные $\frac{d\mathbf{s}}{dt} = \mathbf{A}\cdot\mathbf{s} + \mathbf{b}$; $\mathbf{s}(0) = \mathbf{x}_0$; $t \in [0, T]$, и разностные уравнения $\mathbf{s}(t+H) = e^{\mathbf{A}H}\cdot\mathbf{s}(t) + \mathbf{A}^{-1}\cdot\left(e^{\mathbf{A}H} - \mathbf{E}\right)\cdot\mathbf{b}$, для чего и использован примитивный гибридный автомат.



а

Рис. 4.18. Для проекта `Системный_метод_продолжение`:
а — карта поведения



б

Рис. 4.18. Для проекта `Системный_метод_продолжение`:
б — результаты эксперимента

Этой подсказки достаточно, чтобы самостоятельно модифицировать проект.

Задание

1. Постройте и протестируйте аналогичную программу, используя дробно-рациональное разложение матричной экспоненты.
2. Можно ли осуществить аналогичный переход от дифференциального уравнения

$$\frac{d\mathbf{s}}{dt} = \mathbf{A}\cdot\mathbf{s} + \mathbf{b}(t)\,;\ \mathbf{s}(0) = \mathbf{x}_0\,;\ t \in [0,T]$$

к разностному, и построить подпрограммы для его решения?

Задача 3

Написать подпрограмму умножения прямоугольных матриц на языке MVL. Сравнить время работы написанной подпрограммы и встроенной операции матричного умножения.

Решение.

В листинге 4.9 приведен текст программы.

Листинг 4.9. Модель "Время вычислений"

```
package Время_вычислений is
--  Время_вычислений
  import SysLib;

  function Clock return double is
    external "ClockDLL.dll";

  hybrid class Умножение is
    parameter K: integer := 8;
    A: matrix;
    B: matrix;
    C: matrix;
    Kloop: integer := 1;
    L: integer := 2;
    M: integer := 2;
    N: integer := 2;
    kloop: integer := 1;
    Logic: boolean := false;
    T: vector[K];
    Tm: double := 0;
    logicT: boolean := true;
    Tbegin: double := 0;
    Tend: double := 0;
    Temp: double := 0;
    function MABC(in A: matrix; in B: matrix) return matrix is
      C: matrix[size(A,1),size(B,2)];
      i: integer;
      j: integer;
      k: integer;
    begin
      for i in 1..size(A,1) loop
        for j in 1..size(B,2) loop
          C[i,j]:=0;
```

```
        for k in 1..size(A,2) loop
          C[i,j]:=C[i,j]+A[i,k]*B[k,j];
        end loop;
      end loop;
    end loop;
    return C;
  end MABC;
  local hybrid class Internal is
    bchart {
      initial state _InitState;
        state S1
        entry actions {
          A:=ones(N,M);
          B:=ones(M,L);
          C:=zeros(N,L);
          if logicT then
            Tbegin:=Clock;
            logicT:=false;
          end if;
        }
        exit  actions {
          C:=MABC(A,B);
          --C:=A*B;
          Tend:=Clock;
          T[kloop]:=T[kloop]-Tbegin+Tend;
          Tbegin:=Tend;
        }
      ;
     transition Tr_1 from S1 to _FinalState when Kloop>K
        actions {
          Logic:=true;
          Kloop:=1;
          T[kloop]:=T[kloop]/K;
          Temp:=T[kloop];
        }
      ;
      transition Tr_2 from S1 to S1 after 1
```

```
          actions {Kloop:=Kloop+1;}
        ;

      };
    end Internal;
    bchart {
      initial state _InitState;

      state S1 do S1: Internal
        entry actions {Logic:=false;}
      ;

      final state _FinalState;

      transition _InitTran from _InitState to S1;

      transition Tr_1 from S1 to S1 when Logic
        actions {
          kloop:=kloop+1;
          N:=2*N;
          M:=2*M;
          L:=2*L;
        }
      ;

      transition Tr_2 from S1 to _FinalState when (kloop>K) and not
Logic;

    };
  end Умножение;

  private continuous class Model is
    static Умножение_1: Умножение;

    equations {
      unknown ;
    };
  end Model;
```

```
  static model: Model;

end Время_вычислений;
```

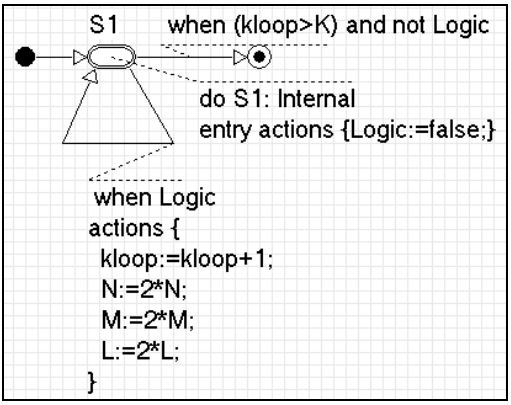


Рис. 4.19. Управление размерами умножаемых матриц

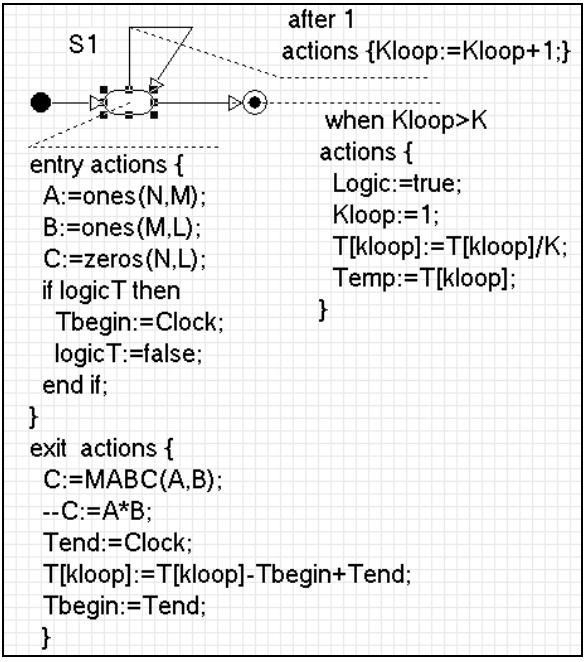
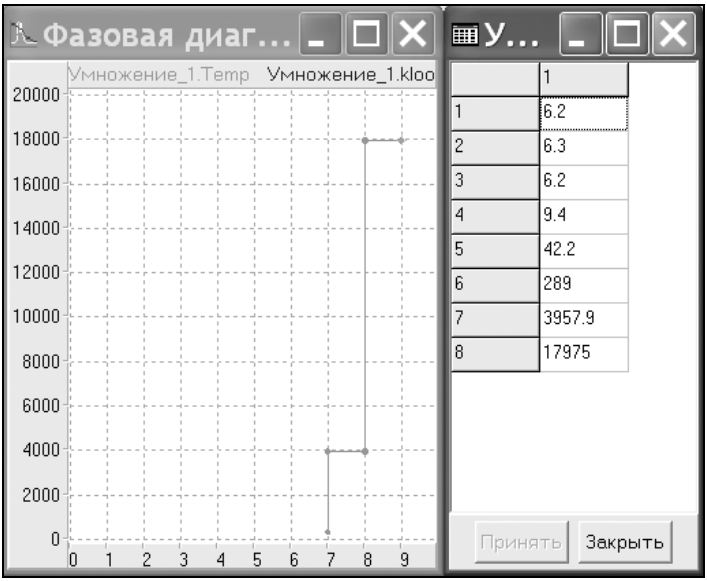
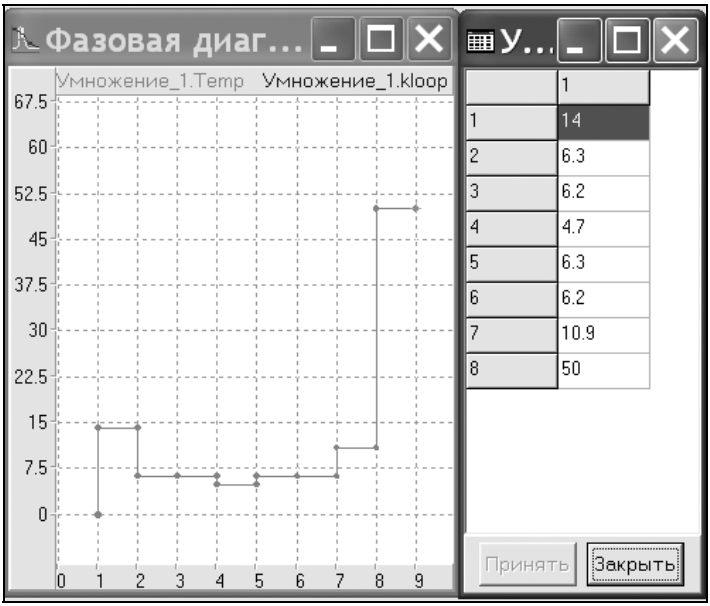


Рис. 4.20. Проведение серии повторяющихся экспериментов для фиксированных размеров



а



б

Рис. 4.21. Абсолютное время выполнения: *а* — процедура на языке MVL; *б* — встроенная операция матричного умножения (FORTAN)

На рис. 4.19 и 4.20 приведены внешняя и внутренняя карты поведения, соответствующие проводимому эксперименту. Внутренняя карта обеспечивает проведение серии экспериментов для заданных размеров матрицы, а внешняя — меняет размер матрицы.

На рис. 4.21 приведены результаты экспериментов. Основная причина столь разительного отличия — передача матриц "значением".

Решение нелинейных уравнений

Необходимость решать нелинейные уравнения как основную задачу в MvStudium возникает, если изучаемая модель описывается нелинейными алгебраическими уравнениями, и они в явном виде содержат время. Нелинейные уравнения также могут возникнуть при описании новых начальных условий при смене поведения во временной щели, их приходится многократно решать при использовании неявных методов решения систем обыкновенных дифференциальных уравнений. Во всех случаях решение нелинейных уравнений может вызвать большие трудности.

Почему так трудно решать нелинейные уравнения?

На начальном этапе изучения сложной системы ее свойства не известны, и исследователь сталкивается с весьма общей постановкой задачи.

Например, дана система двух уравнений относительно неизвестных x_1, x_2 [12]:

$$\begin{cases} f_1(x_1, x_2) = x_1^2 - x_2 + \alpha = 0; \\ f_2(x_1, x_2) = -x_1 + x_2^2 + \alpha = 0. \end{cases}$$

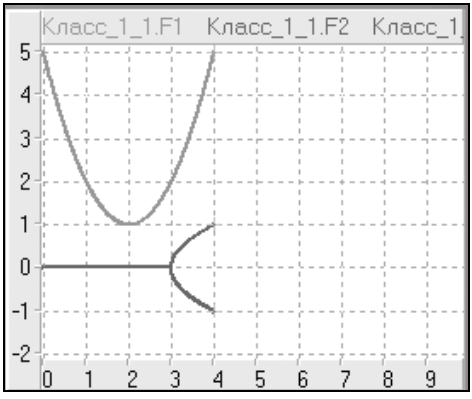
Требуется найти ее вещественные решения. Эта задача параметрическая и, конечно же, численный метод не знает, что если параметр α лежит в пределах $-1 \le \alpha \le 1$, то у системы может быть различное число решений (табл. 4.1).

Таблица 4.1. *Тестовая система*

Параметр	**Решение**
$\alpha = 1$	Нет решений
$\alpha = \frac{1}{4}$	$x_1 = x_2 = \frac{1}{2}$
$\alpha = 0$	$x_1 = x_2 = 0$ и $x_1 = x_2 = 1$
$\alpha = -1$	$x_1 = -1 \quad x_2 = 0$; $x_1 = 0 \quad x_2 = 1$; $x_1 = x_2 = 1 \pm \frac{\sqrt{5}}{2}$

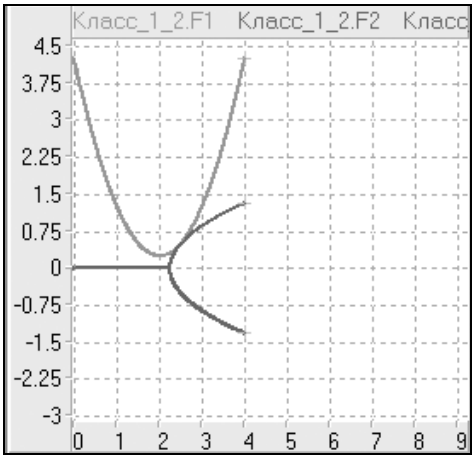
Проверим, не закрались ли в табл. 4.1 ошибки, и посмотрим, как метод будет реагировать на особые случаи — отсутствие решения, множественность корней. Это статическая задача, и она не характерна для MvStudium, но решается весьма просто и часто встречается при изучении численных методов.

Так как речь идет о двух уравнениях, то можно, прежде всего, решить уравнения графически.

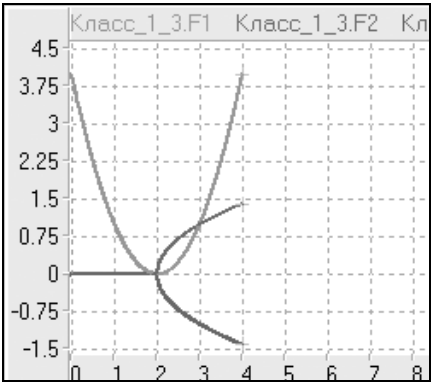


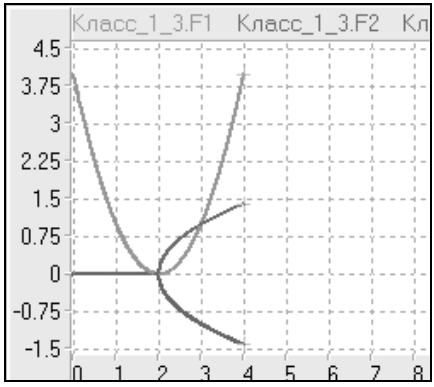
а

Рис. 4.22. Четыре варианта решения, построенные одновременно:
а — корней нет



б



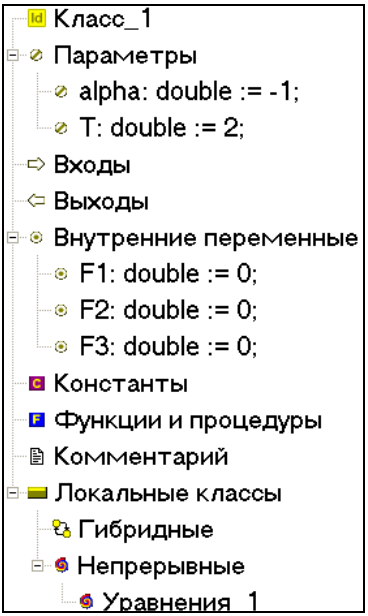


в *г*

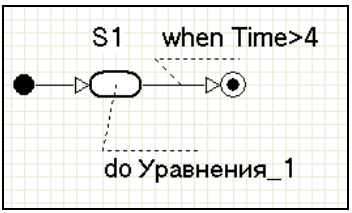
Рис. 4.22. Четыре варианта решения, построенные одновременно:
б — корень один; *в* — два корня; *г* — четыре корня

На рис. 4.22 показаны решения, найденные графически при различных значениях параметра $-1 \le \alpha \le 1$.

Как следует из приведенных графиков, количество корней совпадает с приведенными в табл. 4.1, а значения — нет. Это легко объяснить, если разобраться, как продвигается время в среде MvStudium и как построена головная программа `Model` (Модель). Ее описание приведено на рис. 4.23 и 4.24.



а



б

$$F1 = (\mathbf{Time}-T)^2 + alpha$$

$$F2 = \left(\mathbf{if}\ \mathbf{Time}-T-alpha<=0\ \mathbf{then}\ 0\ \mathbf{else}\ \sqrt{\mathbf{Time}-T-alpha}\right)$$

$$F3 = \left(\mathbf{if}\ \mathbf{Time}-T-alpha<=0\ \mathbf{then}\ 0\ \mathbf{else}\ -\sqrt{\mathbf{Time}-T-alpha}\right)$$

в

Рис. 4.23. Описание класса `Класс_1` для графического решения уравнений: *а* — описание; *б* — карта поведения; *в* — "решаемые" уравнения

Параметр `T` (рис. 4.23, *а*) обеспечивает изменение переменной `x1` (`x1=Time-T`) в пределах от [–2, 2]. Выбранный тип модели — многокомпонентная модель и параметр `alpha` дают возможность включить в Модель (рис. 4.24) сразу четыре экземпляра класса `Класс_1`, что и позволяет одновременно решать четыре системы уравнений с различными значениями параметра `alpha`.

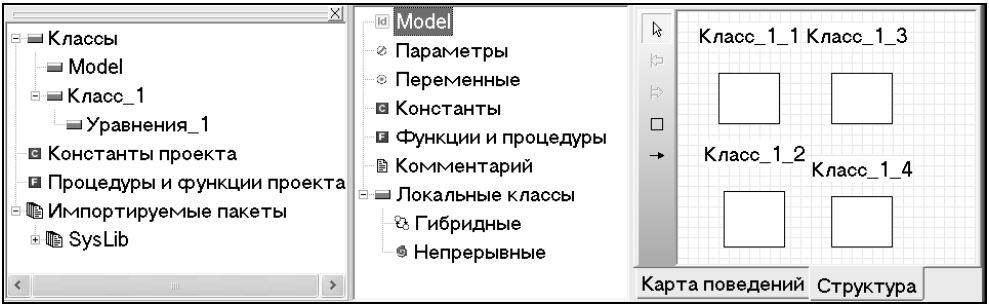


Рис. 4.24. Описание Модели (класс `Model`) для графического решения уравнений

Приведенные графики зависят от переменной `Time`, но было бы лучше, если бы они зависели от переменной `x1=Time-T`. Это и объясняет, почему значения корней на графике не совпадают с приведенными в табл. 4.1. Если не полениться и ввести дополнительную переменную `t=Time-T`, то можно построить "истинные" графики. Для этого достаточно во все окна графиков на испытательном стенде, помимо переменных `F1`, `F2`, `F3`, поместить переменную `t` и вместо временных построить фазовые диаграммы, взяв переменную `t` в качестве абсциссы. Чтобы не вносить изменения в уже разработанную модель, воспользуемся возможностью создавать виртуальные переменные во время вычислительного эксперимента.

Откомпилируйте Модель. Выберите на испытательном стенде образ любого экземпляра класса `Класс_1`, например, `Класс_1_1`. Разместите курсор в поле окна `Класс_1_1`, откройте правой контекстное меню и выберите команду **Переменные**. В появившемся окне вновь откройте диалог и выберите команду **Создать виртуальную переменную** (рис. 4.25).

Далее поместите созданную виртуальную переменную в окно временной диаграммы, откройте диалог **Настройка** для редактирования окна графиков и свяжите виртуальную переменную с осью абсцисс — двойной щелчок в колонке **X** (рис. 4.26), запустите Модель. Вы получите несмещенные графики (рис. 4.27).

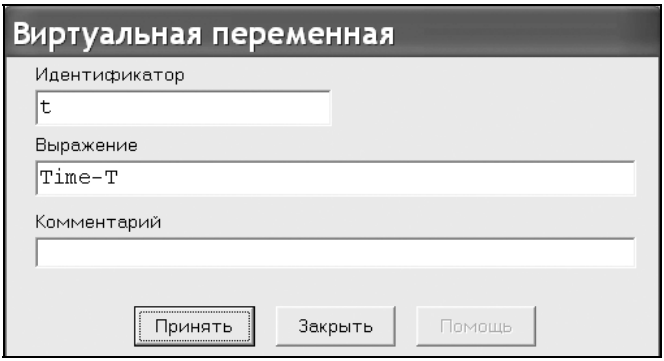


Рис. 4.25. Диалог для создания виртуальных переменных

Рис. 4.26. Диалог **Настройка** для редактирования графиков

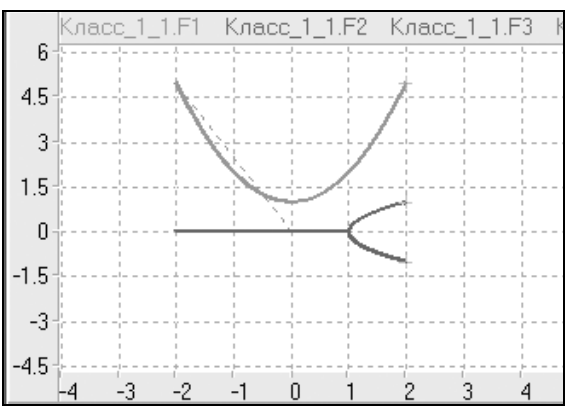
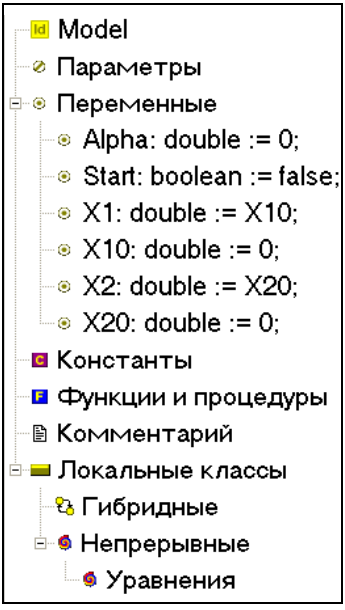


Рис. 4.27. Несмещенный график для случая, когда корней нет

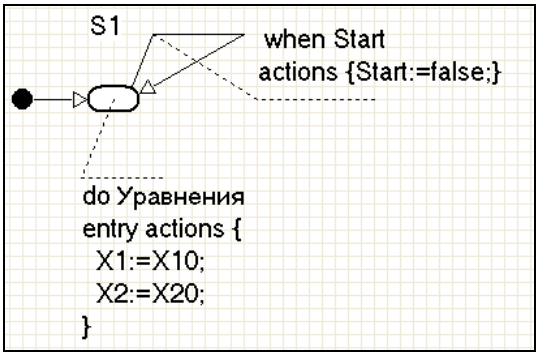
Переменная, помеченная знаком плюс в колонке **X**, будет выбрана в качестве оси абсцисс на фазовой диаграмме.

Теперь можно перейти к численному решению уравнений. Так как ни о какой динамической системе речи не идет, то Модель построена в виды игры, работающей в дискретном, событийном времени (рис. 4.28), — выбирай новые начальные условия, нажимай кнопку, получай решение. Событием является нажатие кнопки на панели выбора начального приближения. Панель создана с помощью команды **Новая 2D-анимация**. Выполните команду **Новая 2D-анимация**. В поле 2D-окна разместите активные 2D-компоненты: два линейных движка различной формы и кнопку. Откройте диалог для редактирования окна, переименуйте его и добавьте нужные подписи под активными элементами. Движки свяжите с переменными `X10` и `X20`, задающими новые начальные приближения, а кнопку — с переменной `Start` *булева* типа, управляющей событиями. Модель можно не запускать.

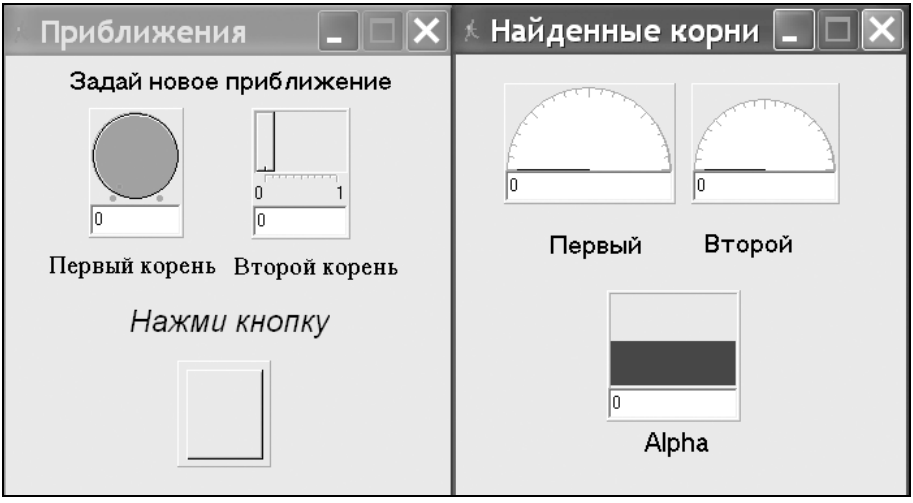


а

Рис. 4.28. Описание Модели:
а — для создания игры использован единственный предопределенный класс `Model`



б



в

Рис. 4.28. Описание Модели:
б — карта поведения класса `Model`; *в* — вид панелей управления экспериментом и визуализации результатов на испытательном стенде

Для продвижения гибридного времени в нашем случае достаточно нажимать кнопку. Количество произошедших событий будет отражаться в окне дискретной составляющей часов гибридного времени, расположенных в левом верхнем углу испытательного стенда.

Прежде чем перейти к экспериментам с разработанной моделью, откроем диалог (команда **Установки | Численные методы...**), предназначенный для

выбора численных методов и задания требуемой погрешности решения (рис. 4.29 и 4.30), и познакомимся с приведенными там методами.

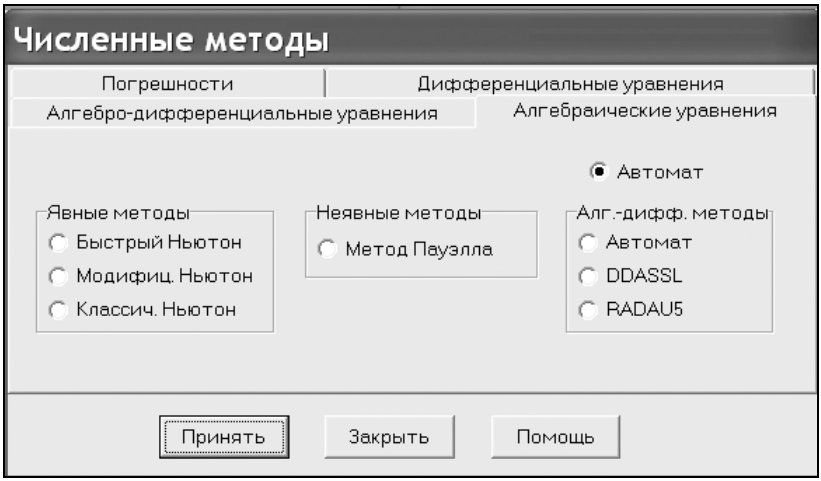


Рис. 4.29. Вкладка численных методов
для решения систем нелинейных алгебраических уравнений

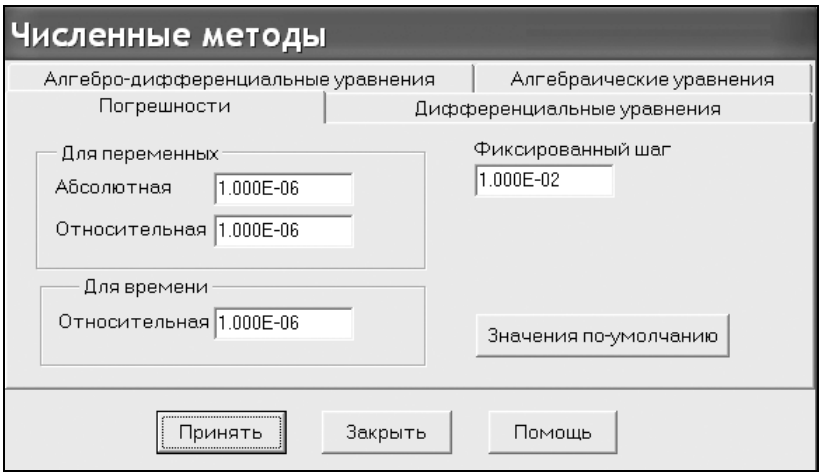


Рис. 4.30. Задание требуемой погрешности решения

Напомним *(см. главу 4 в [5])*, что для решения систем нелинейных алгебраических уравнений по умолчанию используется эвристическая программа Ав-

томат, последовательно применяющая все перечисленные на вкладке методы и одновременно меняющая начальные условия для каждого метода, если он не сходится от начальных условий, заданных пользователем. На этой вкладке перечислены методы ньютоновского типа, с различной стратегией перевычисления матрицы Якоби и поиска приемлемых начальных условий для построения любого решения системы.

Пользователь может использовать любой метод, отличный от программы Автомат, выбираемой по умолчанию.

Методы "быстрый", "модифицированный" и "классический" являются вариантами метода Ньютона с:

- фиксированной матрицей Якоби, вычисленной в начальной точке;
- матрицей Якоби, пересчитываемой после каждых трех итераций;
- матрицей Якоби, вычисляемой на каждой итерации.

В случае медленной сходимости применяемого метода, прежде чем метод будет отвергнут, делается попытка автоматически сменить начальные условия. Если и это не помогает, выдается сообщение о необходимости смены метода.

Наиболее часто методы ньютоновского типа отказывают, если матрица Якоби оказывается вырожденной. Решение системы линейных алгебраических уравнений в этом случае не может быть вычислено, т. к. все решатели систем линейных уравнений оценивают обусловленность матрицы по отношению к задаче обращения и выдают аварийное сообщение, если матрица системы, по их мнению, численно вырождена. Например, в нашем случае при $\alpha = 1/4$ решение равно $x_1 = x_2 = 1/2$, и матрица Якоби в этой точке вырождена. Возникает парадоксальная ситуация, если начать с начального приближения равного решению, — произойдет аварийная остановка, если только ошибки округления не исказят матрицу Якоби настолько, что она станет невырожденной. Это может произойти и по мере приближения к такому решению, если метод случайно вычислит точное решение на очередной итерации (обычно от этого спасают те же ошибки округления). Метод Пауэлла пытается справиться и с этой трудностью и вследствие этого более трудоемок.

В следующей группе (см. рис. 4.20) перечислены методы, предназначенные для решения систем алгебро-дифференциальных уравнений. Дело в том, что их работа всегда начинается с поиска согласованных начальных условий для решаемой системы алгебро-дифференциальных уравнений, т. е. решения системы алгебраических уравнений при заданных начальных условиях для диф-

ференциальных переменных. Таким образом, если дифференциальных переменных нет, то в начальной точке будет решена заданная система алгебраических уравнений. Грешно было не воспользоваться этой возможностью и не предоставить пользователю несколько независимых программ для решения нелинейных алгебраических уравнений.

Выбор желаемой погрешности в решении (рис. 4.30) может существенно повлиять на поиск решения. В данной версии речь идет о евклидовой норме вектора погрешности (конечно, речь идет не о погрешности, а о близости двух последовательных итераций). Во всех программах есть возможность задавать погрешности и для отдельных координат вектора решения, но сейчас она не используется. Дело в том, что многие численные методы имеют тонкие настройки своих параметров, но не понятно, как сделать их доступными пользователю и как их использовать при автоматическом выборе методов. Погрешность, названная "погрешностью по времени", в данном случае ни на что не влияет, т. к. она используется при поиске точки переключения.

Выбрав численный метод, можно начинать играть с Моделью — задавать различные начальные приближения и проверять результат вычисления.

Наконец, интересно посмотреть, что будет с различными методами, если вблизи начального приближения находится несколько решений, как это происходит при $\alpha = 0$ или $\alpha = -1$.

Обсудим еще одну программу, позволяющую посмотреть, что будет в случае, представленном на рис. 4.31.

Приведенная на рис. 4.31 карта поведения реализует следующий вычислительный эксперимент. Задается некоторая область на плоскости, покрывается равномерной сеткой, и ее узлы выбираются в качестве начальных приближений. Основному состоянию `S1` приписаны решаемые нелинейные уравнения. Условие срабатывания перехода — `after tau`, где `tau` — параметр с любым ненулевым значением, который обеспечивает последовательный перебор всех узлов сетки.

В данном эксперименте область начальных значений представляет собой прямоугольник. Начальные приближения, узлы равномерной сетки задаются с помощью операторов

```
x1:=hx1/2+ hx1*(j-1);
x2:=hx2/2+ hx2*(i-1);
```

Здесь `hx1`, `hx2` — параметры. Матрица `M` хранит информацию о найденном решении при начальном приближении в соответствующем узле сетки. Ре-

зультаты эксперимента для области $0.05 \le x_1 \le 0.95$, $0.05 \le x_2 \le 0.95$, $hx_1 = hx_2 = 0.1$ при нулевом значении параметра α показаны на рис. 4.32. В качестве решателя был выбран метод Пауэлла. Любой другой решатель ньютоновского типа в этом случае будет выдавать в некоторых узлах сетки аварийное сообщение.

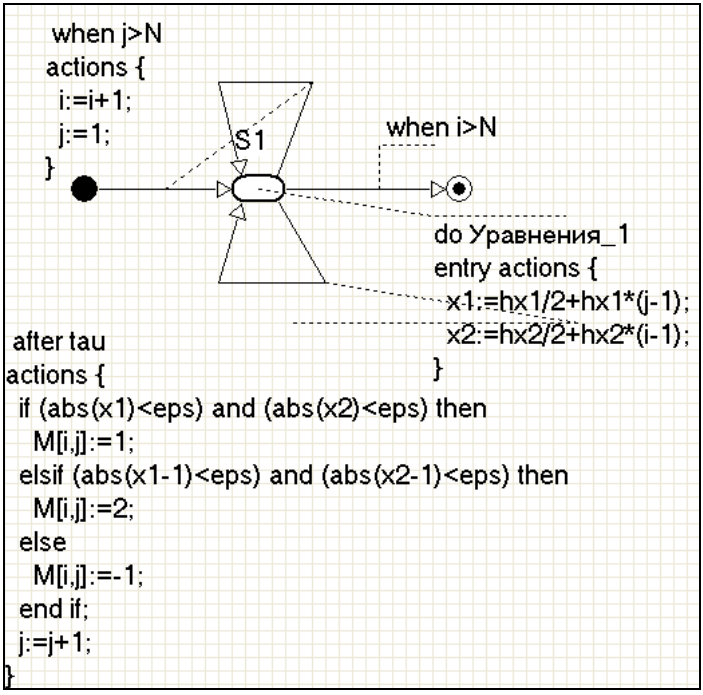
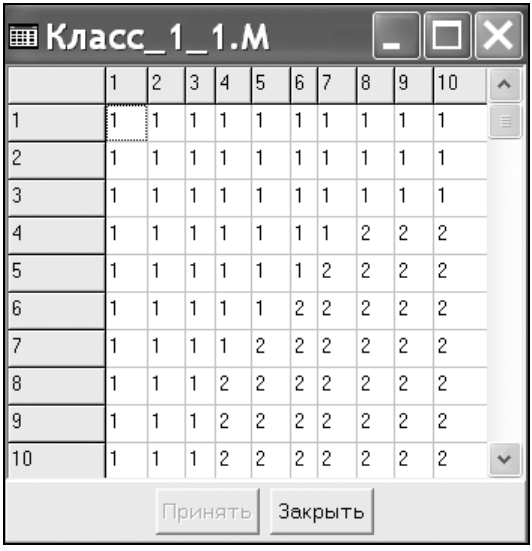


Рис. 4.31. Карта поведения класса `Класс_1` для проведения эксперимента с областью сходимости

Интересно сравнить, что произойдет с областью сходимости при смене метода Пауэлла на методы для решения алгебро-дифференциальных уравнений (рис. 4.33).

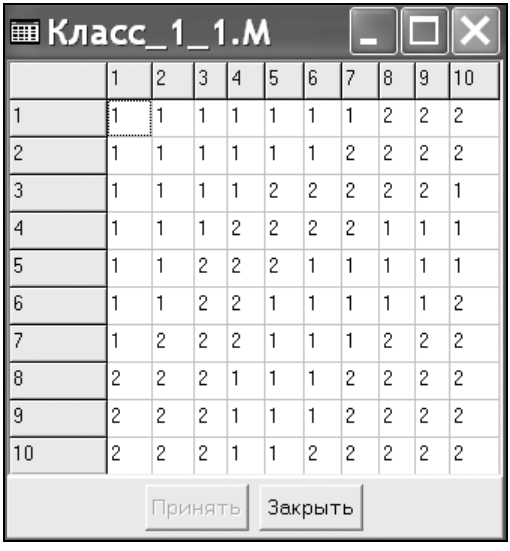
Область сходимости явно меняется. Обратите внимание на то, что программа RADAU5, применяя для поиска согласованных начальных условий методы ньютоновского типа, не выдает аварийного сообщения ни в одном узле сетки! До сих пор с этой задачей справлялся только метод Пауэлла. Разгадка проста, если вспомнить, что у всех методов своя стратегия поиска подходя-

щих начальных условий, когда при заданных начальных условиях метод не сходится.



	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	2	2	2
5	1	1	1	1	1	1	2	2	2	2
6	1	1	1	1	1	2	2	2	2	2
7	1	1	1	1	2	2	2	2	2	2
8	1	1	1	2	2	2	2	2	2	2
9	1	1	1	2	2	2	2	2	2	2
10	1	1	1	2	2	2	2	2	2	2

Рис. 4.32. Построение области сходимости



	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	2	2	2
2	1	1	1	1	1	1	2	2	2	2
3	1	1	1	1	2	2	2	2	2	1
4	1	1	1	2	2	2	2	1	1	1
5	1	1	2	2	2	1	1	1	1	1
6	1	1	2	2	1	1	1	1	1	2
7	1	2	2	2	1	1	1	2	2	2
8	2	2	2	1	1	1	2	2	2	2
9	2	2	2	1	1	1	2	2	2	2
10	2	2	2	1	1	2	2	2	2	2

Рис. 4.33. Изменение области сходимости при смене метода (RADAU5)

У систем нелинейных алгебраических уравнений может быть не только конечное число решений, но и счетное, а также решения могут заполнять целые области. Рассмотрим еще два интересных и важных примера.

Задача 4

Требуется решить графически, построить игру для поиска решения для заданного начального приближения и экспериментально исследовать область сходимости систем уравнений:

$$\begin{cases} \frac{1}{2}\cdot x_1 \cdot \sin\left(\frac{\pi}{2}\cdot x_1\right) - x_2 = 0, \\ -x_1 + x_2^2 + 1 = 0. \end{cases} \tag{4.1}$$

$$\begin{cases} x_1^2 - |x_2| = 0, \\ x_1^2 - x_2 = 0. \end{cases} \tag{4.2}$$

Замечания для самоконтроля

На рис. 4.34 показаны некоторые решения уравнения (4.1). По оси абсцисс откладывается время.

Решения уравнения (4.2) показаны на рис. 4.35. В данном случае по оси абсцисс отложена переменная `Time-T`.

На этом примере следует остановиться особо. Если область определения по переменной x_2 совпадает с положительной полуосью, то система (4.2) переопределена. Обнаружить это автоматически на этапе анализа уравнений до периода исполнения невозможно. Приходится полагаться на программные реализации численных методов, но в этом случае диагностика откладывается на период исполнения. На рис. 4.36 показано типичное сообщение для методов ньютоновского типа, используемых в пакете, появляющееся в этом случае. Метод Ньютона "порекомендовал" обратиться к методу Пауэлла. Мы последовали его рекомендациям, и с помощью метода Пауэлла была исследована область сходимости — построена сетка на прямоугольнике, и ее узлы выбирались в качестве начальных условий. Метод Пауэлла справился с поставленной задачей (рис. 4.37), и трудно сказать, хорошо это или плохо, все зависит от постановки задачи. Не предупреждая о возникающих трудностях, метод находил какое-либо решение. Коэффициентам матрицы на рис. 4.37

присваивается значение суммы модулей найденных корней, и мы видим, что во всех случаях находился свой, "уникальный" корень.

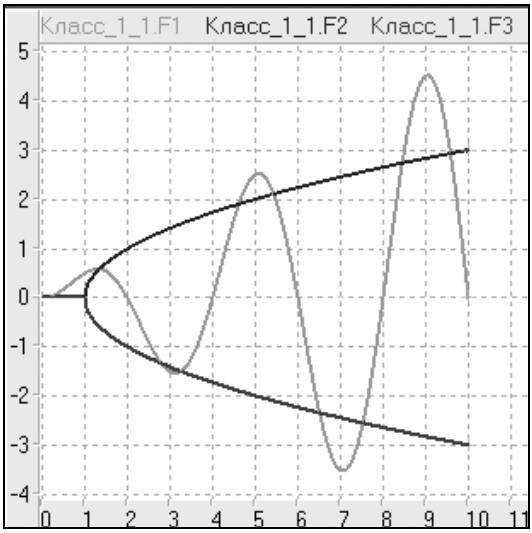


Рис. 4.34. Несколько решений уравнения (4.1)

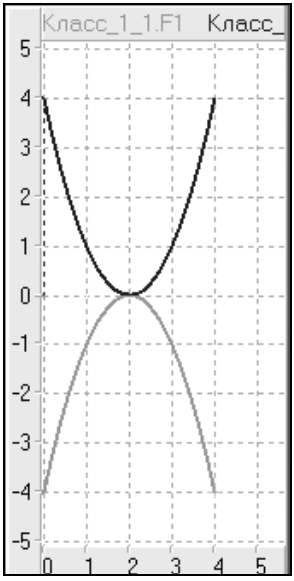


Рис. 4.35. Графическое решение уравнения (4.2)

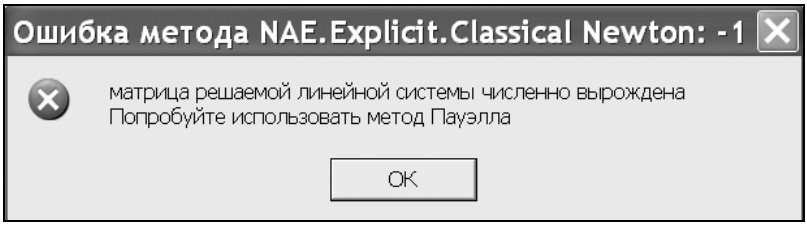
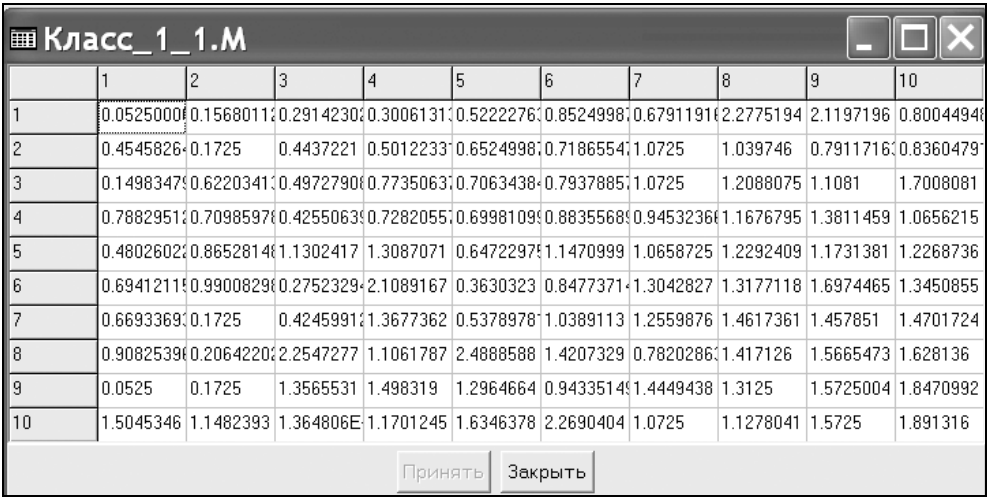


Рис. 4.36. "Классический" метод Ньютона "порекомендовал" обратиться к методу Пауэлла



Класс_1_1.M

	1	2	3	4	5	6	7	8	9	10
1	0.0525000	0.1568011	0.2914230	0.3006131	0.5222276	0.8524998	0.6791191	2.2775194	2.1197196	0.8004494
2	0.4545826	0.1725	0.4437221	0.5012233	0.6524998	0.7186554	1.0725	1.039746	0.7911716	0.8360479
3	0.1498347	0.6220341	0.4972790	0.7735063	0.7063438	0.7937885	1.0725	1.2088075	1.1081	1.7008081
4	0.7882951	0.7098597	0.4255063	0.7282055	0.6998109	0.8835568	0.9453236	1.1676795	1.3811459	1.0656215
5	0.4802602	0.8652814	1.1302417	1.3087071	0.6472297	1.1470999	1.0658725	1.2292409	1.1731381	1.2268736
6	0.6941211	0.9900829	0.2752329	2.1089167	0.3630323	0.8477371	1.3042827	1.3177118	1.6974465	1.3450855
7	0.6693369	0.1725	0.4245991	1.3677362	0.5378978	1.0389113	1.2559876	1.4617361	1.457851	1.4701724
8	0.9082539	0.2064220	2.2547277	1.1061787	2.4888588	1.4207329	0.7820286	1.417126	1.5665473	1.628136
9	0.0525	0.1725	1.3565531	1.498319	1.2964664	0.9433514	1.4449438	1.3125	1.5725004	1.8470992
10	1.5045346	1.1482393	1.364806E	1.1701245	1.6346378	2.2690404	1.0725	1.1278041	1.5725	1.891316

Принять Закрыть

Рис. 4.37. Результаты работы метода Пауэлла. Коэффициентам матрицы присваивается значение суммы модулей найденных корней

Справившись со "статическими задачами", можно перейти к близко связанным с решением нелинейных уравнений задачам построения дискретных отображений.

Рассмотрим отображение Хенона

$$x_{x+1} = x_n \cdot \cos\alpha - \left(y_n - x_n^2\right) \cdot \sin\alpha,$$
$$x_{x+1} = x_n \cdot \sin\alpha - \left(y_n - x_n^2\right) \cdot \cos\alpha,$$

исследуемое в книге [18] с помощью программы WinSet.

Покажем, как построить это отображение с помощью MvStudium. В качестве параметра, влияющего на свойства отображений, берется величина $p = \frac{2\pi}{\alpha}$. Точно так же, как и в предыдущем случае, выберем прямоугольную область, построим на ней сетку и используем ее узлы для построения начальных условий (рис. 4.38).

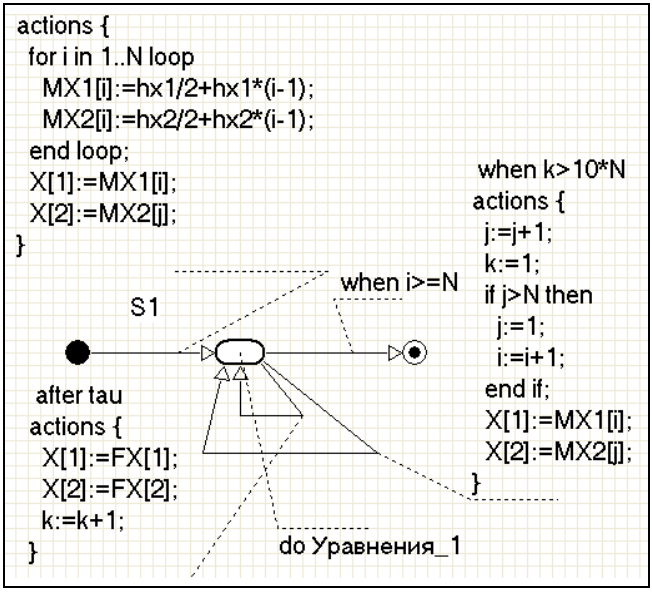


Рис. 4.38. Карта поведения класса `Класс_1` Модели, предназначенной для построения отображений

Отличия от предыдущих программ незначительные — вместо матрицы для хранения начальных условий использованы векторы, и по-другому построен цикл, позволяющий обойти все точки сетки. Можно отказаться и от векторов, формируя начальные условия перед запуском каждого цикла. Количество итераций для каждого начального условия ограничено числом $10 \cdot N$. Необходимость создания класса обусловлена тем, что, как и в предыдущих случаях, мы будем строить несколько отображений одновременно. На рис. 4.39—4.42 показаны построенные отображения при следующих значениях параметра $p = 3.2,\ 3.5,\ 3.95,\ 4$.

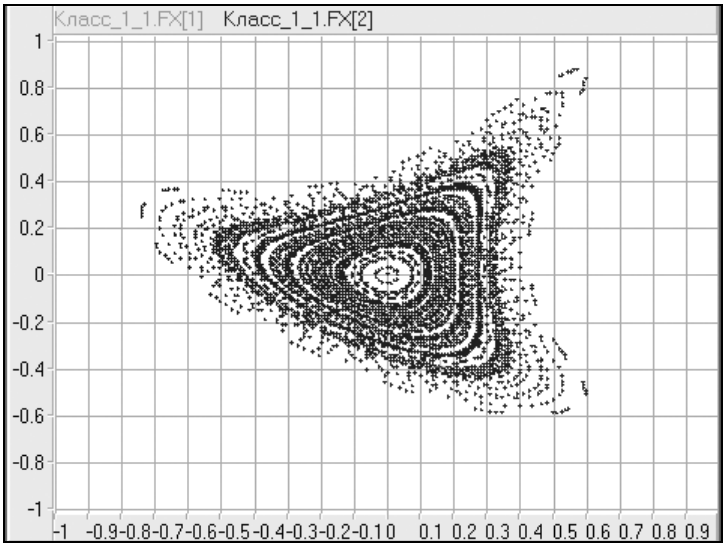


Рис. 4.39. Отображение Хенона, p = 3.2

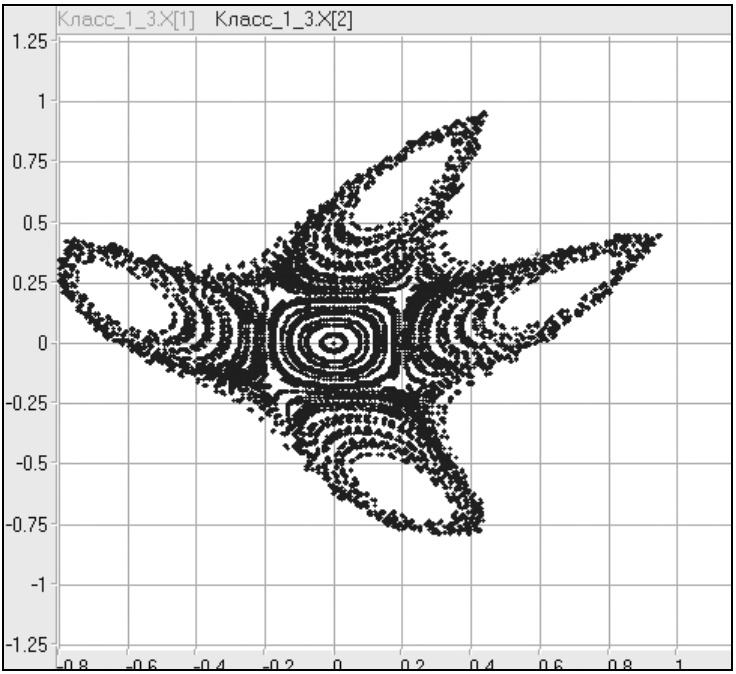


Рис. 4.40. Отображение Хенона, p = 3.95

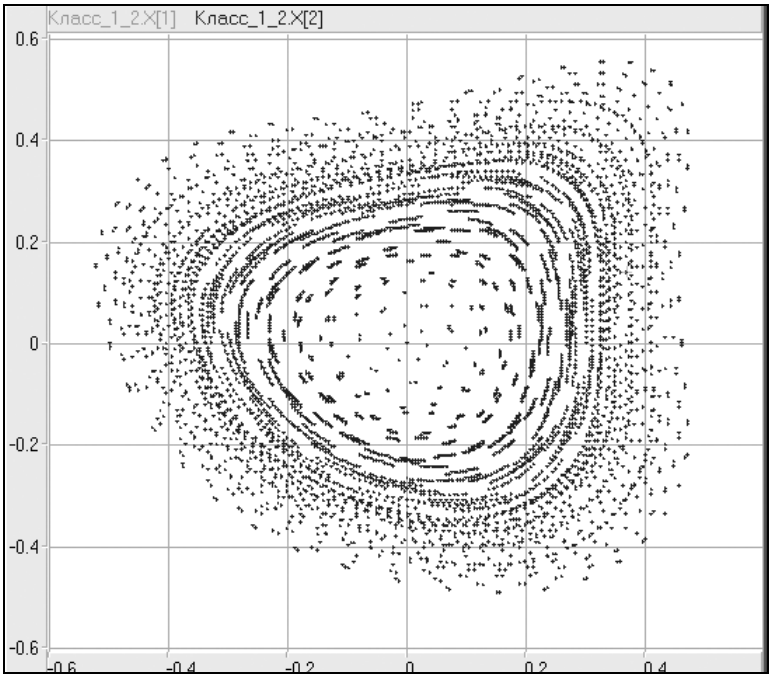


Рис. 4.41. Отображение Хенона, p = 3.5

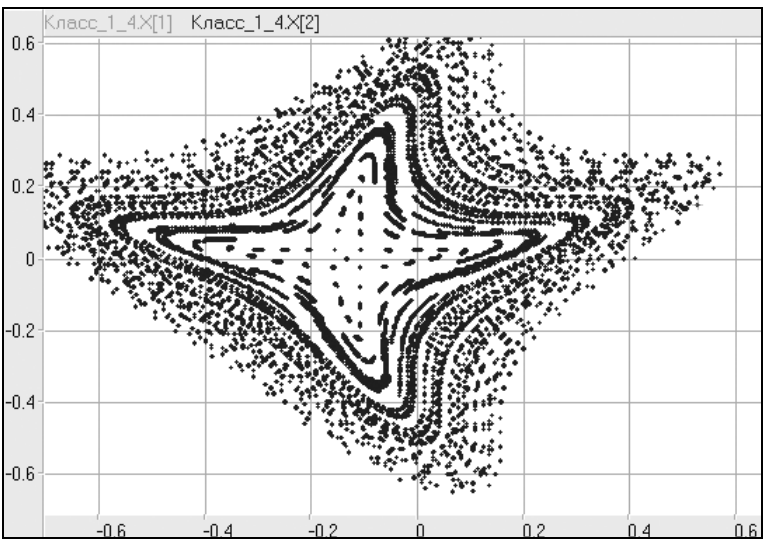


Рис. 4.42. Отображение Хенона, p = 4

Для того чтобы получить приведенные картинки, необходимо предварительно настроить окно фазовой диаграммы (на рис. 4.43 представлен фрагмент окна **Настройка для графиков**). Обратите внимание на группу переключателей **Линия "скачков"**. В данном случае она не отображается.

Рис. 4.43. Настойка отображения линии скачков

Задача 5

Построить отображение Заславского

$$x_{x+1} = x_n \cdot \cos\alpha - \left(y_n - \gamma \cdot \sin(x_n)\right) \cdot \sin\alpha,$$
$$y_{x+1} = x_n \cdot \sin\alpha + \left(y_n - \gamma \cdot \sin(x_n)\right) \cdot \cos\alpha$$

для $p = 7.8$; $\gamma = 0.7, 0.8$; $\alpha = \dfrac{2\pi}{p}$. Выбрать границы для переменных $-40 \le x_1,\ x_2 \le 40$.

Задача 6

Ничто не мешает нам рассматривать формулы метода Ньютона

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left(\frac{\partial \mathbf{f}}{\partial x}\right)^{-1}\Bigg|_{x=x_n} \cdot \mathbf{f}(\mathbf{x}_n)$$

для решения двух уравнений $\mathbf{f}(\mathbf{x}) = 0$ как дискретное отображение. Разработайте программу, которая способна визуализировать такое отображение. Испытайте ее на рассмотренных в начале данного раздела уравнениях.

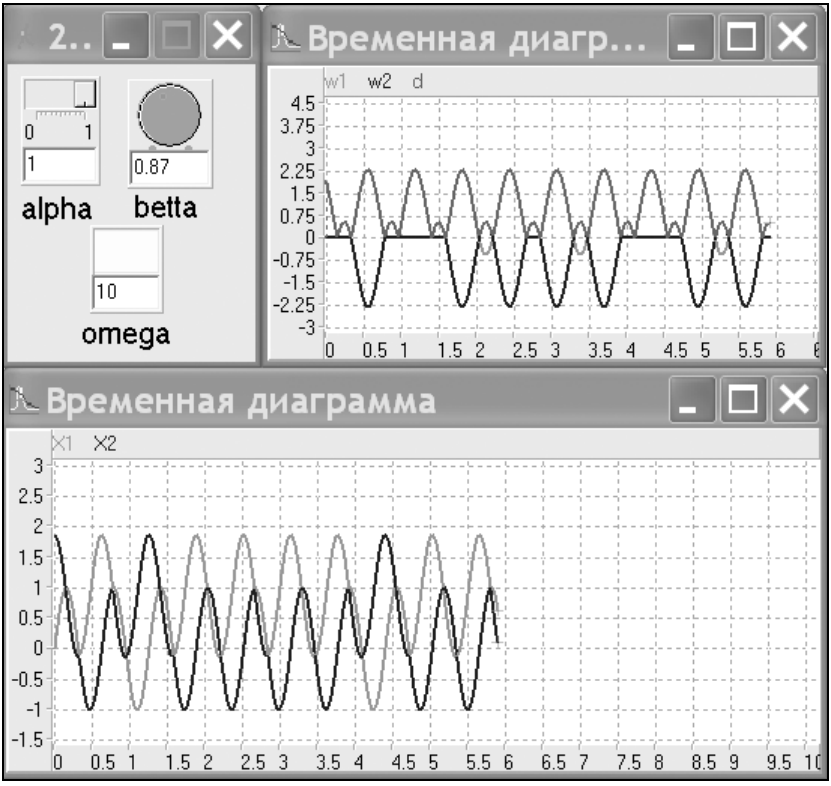
Теперь настало время перейти к основной задаче — моделям, в которых коэффициенты в нелинейных алгебраических уравнениях зависят от времени.

Прежде всего, рассмотрим очень простую систему

$$\begin{cases} f_1(x_1, x_2, t) = x_1 + x_2 + \alpha \cdot \left(\sin(\omega \cdot t) + \cos(\omega \cdot t) + \beta\right) = 0, \\ f_2(x_1, x_2, t) = x_1 \cdot x_2 + \alpha^2 \cdot \sin(\omega \cdot t) \cdot (\cos(\omega \cdot t) + \beta) = 0. \end{cases}$$

У этой системы два решения — $x_1 = \alpha \cdot \sin(\omega \cdot t)$; $x_2 = \alpha \cdot (\cos(\omega \cdot t) + \beta)$ и $x_1 = \alpha \cdot \cos(\omega \cdot t)$; $x_2 = \alpha \cdot (\sin(\omega \cdot t) + \beta)$.

При $\beta > 2$ корни x_1 и x_2 не пересекаются, и решение системы не должно вызывать затруднений. Другое дело, если $\beta \le 2$ — здесь возможны пересечения. Не "перепутает" ли корни численный метод?



а

Рис. 4.44. Численное решение:
а — классический метод Ньютона

б

Рис. 4.44. Численное решение:
б — метод Пауэлла

Создадим очередную игру и построим два решения при $\beta > 2$ и $\beta \le 2$ различными программными реализациями численных методов. На рис. 4.44 и 4.45 показаны решения, полученные "классическим" методом Ньютона, методом Пауэлла и программой DDASL.

Для каждого метода приводятся два графика: график ошибки и значения определителя матрицы Якоби, вычисляемого по ходу решения, и графики компонентов решения. Как видно на рис. 4.44, классический метод Ньютона и метод Пауэлла постоянно "путают" компоненты, причем это происходит в различных точках. Программа DASSL не сходит с выбранного решения. На рис. 4.45 (график значения определителя) хорошо видно, что методы сходят с выбранного решения в точках, где определитель матрицы Якоби близок к нулю.

Рис. 4.45. Численное решение.
Программа DDASL

Задача 7

Вернемся к только что рассмотренным уравнениям и перепишем их в виде

$$\begin{cases} f_1(x_1, x_2, t) = x_1^2 - x_2 + \alpha \cdot \sin(t) = 0, \\ f_2(x_1, x_2, t) = -x_1 + \beta \cdot x_2^2 + \alpha \cdot \sin(t) = 0. \end{cases}$$

При $\alpha = 1$ и $\beta = 1$ число корней будет меняться во времени и система будет иметь от нуля до четырех корней. Проверьте экспериментально, как будут вести себя численные методы для этого уравнения при различных значениях параметров α и β.

Задача 8

Оказывается, что для определенного класса уравнений можно построить специальные варианты метода Ньютона, не требующие ни вычисления матрицы Якоби, ни решения систем линейных алгебраических уравнений.

Рассмотрим уравнение $\widetilde{P}_2(x) = a_0 \cdot x^2 + a_1 \cdot x + a_2$. Представим его в виде:

$$P_2(x) = a_0 \cdot x^2 + a_1 \cdot x + a_2 = 0 \Rightarrow x^2 + \frac{a_1}{a_0} \cdot x + \frac{a_2}{a_0} = (x - \xi_1) \cdot (x - \xi_2),$$

где ξ_1, ξ_2 — вещественные корни

Обычная методика предполагает, что сначала локализуются все корни, а потом их последовательно уточняют. Можно ли искать корни не последовательно, а одновременно?

Применяя формулы Виета для полинома второй степени, получаем:

$$\begin{cases} \xi_1 + \xi_2 = -\dfrac{a_1}{a_0}, \\ \xi_1 \cdot \xi_2 = \dfrac{a_2}{a_0}. \end{cases}$$

Будем рассматривать эти формулы как нелинейную систему уравнений относительно неизвестных x_1 и x_2:

$$\begin{cases} x_1 + x_2 = -\dfrac{a_1}{a_0}, \\ x_1 \cdot x_2 = \dfrac{a_2}{a_0}. \end{cases} \tag{4.3}$$

Оказывается, что метод Ньютона для системы (4.3) можно записать в виде:

$$\begin{cases} x_1^{(k+1)} = x_1^{(k)} - \dfrac{\widetilde{P}_2(x_1^{(k)})}{x_1^{(k)} - x_2^{(k)}}, \\ x_2^{(k+1)} = x_2^{(k)} + \dfrac{\widetilde{P}_2(x_2^{(k)})}{x_1^{(k)} - x_2^{(k)}}, \end{cases}$$

где $k = 0, 1, 2, \ldots$

Действительно, выполним непосредственно все вычисления в формуле

$$x^{(k+1)} = \mathbf{x}^{(k)} - \left(\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right)\Bigg|_{\mathbf{x}=\mathbf{x}^{(k)}}^{-1} \cdot \mathbf{F}\left(\mathbf{x}^{(k)}\right),$$

для рассмотренных уравнений

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_1^{(k)} + x_2^{(k)} + \dfrac{a_1}{a_0} \\ x_1^{(k)} * x_2^{(k)} - \dfrac{a_1}{a_0} \end{bmatrix}, \quad \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\Bigg|_{\mathbf{x}=\mathbf{x}^{(k)}} = \begin{bmatrix} 1 & 1 \\ x_2^{(k)} & x_1^{(k)} \end{bmatrix},$$

$$\left(\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right)\Bigg|_{\mathbf{x}=\mathbf{x}^{k}}^{-1} = \begin{bmatrix} x_1^{(k)} & -1 \\ -x_2^{(k)} & 1 \end{bmatrix} \cdot \frac{1}{x_1^{(k)} - x_2^{(k)}}.$$

Тогда:

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} - \frac{1}{x_1^{(k)} - x_2^{(k)}} \cdot \begin{bmatrix} x_1^{(k)} & -1 \\ -x_2^{(k)} & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1^{(k)} + x_2^{(k)} + \dfrac{a_1}{a_0} \\ x_1^{(k)} \cdot x_2^{(k)} - \dfrac{a_1}{a_0} \end{bmatrix} =$$

$$= \begin{bmatrix} x_1^{(k)} - \dfrac{1}{x_1^{(k)} - x_2^{(k)}} \cdot (x_1^{(k)} \cdot (x_1^{(k)} + x_2^{(k)} + \dfrac{a_1}{a_0}) - x_1^{(k)} \cdot x_2^{(k)} + \dfrac{a_2}{a_0}) \\ x_2^{(k)} - \dfrac{1}{x_1^{(k)} - x_2^{(k)}} \cdot (-x_2^{(k)} \cdot (x_1^{(k)} + x_2^{(k)} + \dfrac{a_1}{a_0}) + x_1^{(k)} \cdot x_2^{(k)} - \dfrac{a_2}{a_0}) \end{bmatrix} =$$

$$= \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} - \frac{1}{x_1^{(k)} - x_2^{(k)}} \cdot \begin{bmatrix} (x_1^{(k)})^2 + \dfrac{a_1}{a_0} \cdot x_1^{(k)} + \dfrac{a_2}{a_0} \\ -(x_2^{(k)})^2 - \dfrac{a_1}{a_0} \cdot x_2^{(k)} - \dfrac{a_2}{a_0} \end{bmatrix} =$$

$$= \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} - \frac{1}{x_1^{(k)} - x_2^{(k)}} \cdot \begin{bmatrix} \tilde{P}_2(x_1^{(k)}) \\ \tilde{P}_2(x_2^{(k)}) \end{bmatrix}.$$

Эти формула распространяется на полиномы любой степени и дает возможность одновременно вычислять корни полиномов с высокой скоростью сходимости.

Реализуйте и исследуйте этот метод. Начните с простейшей реализации (листинг 4.10).

Листинг 4.10. Простейшая реализация метода Ньютона для полиномиальных уравнений

```
function P(in a: vector; in y: double) return double is
      z: double;
      M: integer;
    begin
      M:=size(a);
      if M=1 then
        z:=a[1];
      else
        z:=a[M]*y+a[M-1];
        for i in 2..N loop
          z:=z*y+a[M-i];
        end loop;
      end if;
      return z;
    end P;
    function Pprim(in w: vector; in y: double; in j: integer) return
    double is
      z: double;
    begin
      z:=1;
      for i in 1..size(w) loop
        if i/=j then
          z:=z*(y-w[i]);
        end if;
      end loop;
      return z;
    end Pprim;
    function Roots(in a: vector; in w: vector) return vector is
      y: vector[N];
    begin
      y:=w;
      for j in 1..5*Ki loop
```

```
        for i in 1..N loop
          y[i]:=y[i]-P(a,y[i])/Pprim(y,y[i],i);
        end loop;
      end loop;
      return y;
    end Roots;
```

Постройте полином с переменными коэффициентами, зависящими от времени, с известными корнями (листинг 4.11).

Листинг 4.11. Система уравнений и ее решение

```
local continuous class Уравнения_1 is
  equations {
    x = [uniform(-0.05,0.05); R*sin(Time); R*cos(Time)];
    a = [-x[1]*x[2]*x[3]; x[1]*(x[2]+x[3])+x[2]*x[3];
    -(x[1]+x[2]+x[3]); 1];
    w = R1*x;
    y = Roots(a,w);
    w1 = vnorm2(x-y)/vnorm2(x);
    unknown x, a, w, w1, y;
  };
end Уравнения_1;
```

В приведенных процедурах: `a: vector[N+1]` — вектор, содержащий коэффициенты полинома; `x: vector[N]` — вектор, содержащий корни полинома; `w: vector[N]` — вектор начальных приближений к корням; `y: vector[N]` — вектор, содержащий корни полинома, найденные численно. Таким образом, в каждый момент времени формируется новая система уравнений с известными корнями — вектор `x`. Коэффициенты уравнений задаются через коэффициент полинома — вектор `a`, корни которого надо найти. Обратите внимание, что первый корень (первая координата вектора `x`) — случайное число. Для каждой новой системы уравнений формируется новое начальное приближение — истинное решение умножается на некоторый коэффициент. Затем находится численное решение и сравнивается с известным.

Воспроизведите найденные решения. Карта поведения модели приведена на рис. 4.46.

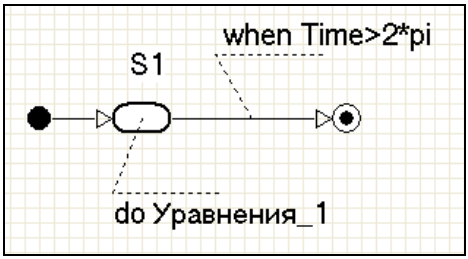


Рис. 4.46. Карта поведения Модели

Посмотрите на полученные графики (рис. 4.47 и 4.48). Почему графики решения и ошибки имеют такой вид, нет ли в программе ошибки?

В приведенной программе число итераций фиксировано и определяется коэффициентом `Ki`. Модифицируйте программу так, чтобы число итераций зависело от задаваемых пользователем относительной и абсолютной погрешностей. При численном решении заранее предполагается, что полином имеет только действительные корни. Можно ли это проверить, не приступая к вычислениям? Можно ли модифицировать программу так, чтобы она вычисляла вещественные корни, даже если у полинома есть и комплексные? Напишите программную реализацию метода Лобачевского и сравните эффективность обеих программ.

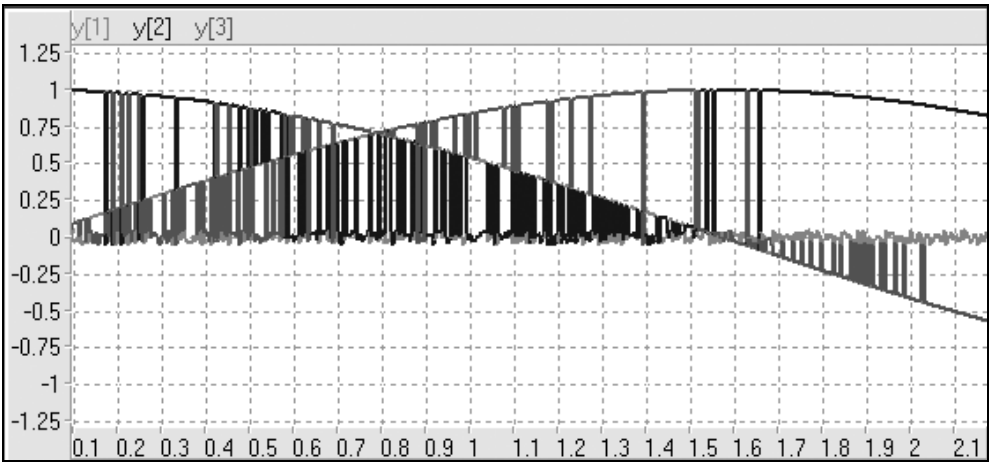


Рис. 4.47. Корни полинома, найденные численно

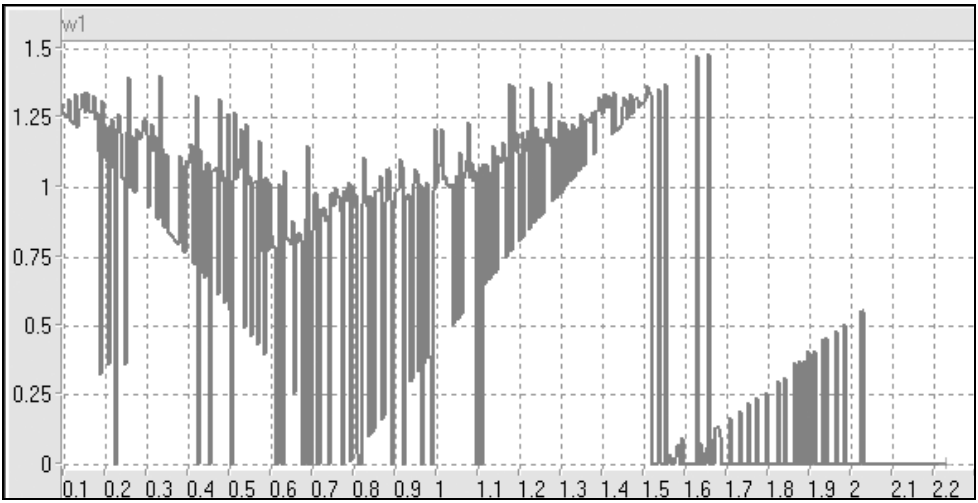


Рис. 4.48. График нормы ошибки

Решение обыкновенных дифференциальных уравнений

Численное решение систем обыкновенных дифференциальных уравнений является основной задачей большинства пакетов моделирования. Сейчас уже можно сказать, что лучшие специализированные библиотеки программных реализаций численных методов решения обыкновенных дифференциальных уравнений, созданные в конце прошлого столетия, наконец-то нашли применение во многих пакетах моделирования. Численные библиотеки пакетов Simulink, Dymola, MvStudium содержат практически одни и те же хорошо зарекомендовавшие себя численные методы, они проверены на близких тестовых наборах, и можно почти не сомневаться в том, что решение системы уравнений, найденное в одном из пакетов, воспроизведется и в другом. Это большое достижение, учитывая, что еще совсем недавно чаще всего использовался метод Рунге—Кутты—Фельберга, программная реализация которого (1974 год) была опубликована в книге [10]. У нас в стране книга была переведена Х. Д. Икрамовым, вышла в 1980 году и была чрезвычайно популярна в эру ручного моделирования благодаря включенным в нее программным реализациям численных методов. Программная реализация метода Рунге—Кутты—Фельберга долго использовалась и в пакете MATLAB, пришедшего

на смену ручным вычислениям, и только относительно недавно она была заменена на реализацию Дормана—Принса.

Разнообразие численных методов, используемых на практике, определяется разнообразием способов аппроксимации дифференциальных уравнений разностными. Изучению разностных схем, соответствующих дифференциальным уравнениям, в свое время уделялось так много внимания, что дискретное время и дискретные модели (разностные уравнения) чуть ли не стали самыми главными в моделировании. Почему-то в пылу споров забывали, что в данном случае разностные уравнения всего лишь аппроксимируют дифференциальные, а то, что именно они применяются для расчетов, не является основанием для отказа от модели непрерывного времени. Если мы умеем вычислять значение функции для любого вещественного значения независимой переменной (времени), то можно говорить о непрерывной модели.

Успехом математиков и программистов можно считать то, что сейчас многие пользователи даже не задумаются над тем, как было найдено решение уравнения в заданной временной точке. К сожалению, этот успех имеет и оборотную сторону — многие из пользователей искренне считают, что любой численный метод, включенный в пакет, обеспечивает нахождение решения с выбранной ими абсолютной или относительной погрешностью.

Перейдем к краткому описанию численных методов решения обыкновенных дифференциальных уравнений. Напомним, что методы бывают одношаговыми и многошаговыми в соответствии с используемыми для аппроксимации разностными уравнениям первого и высшего порядков. Иными словами, в одношаговых методах используется информация только о решении в предыдущей временной точке, а в многошаговых — о решении в нескольких предыдущих точках. К одношаговым методам относятся методы Эйлера, Рунге—Кутты. Вместо одношаговых методов столь же успешно можно применять многошаговые методы, в частности, линейные многошаговые методы Адамса или Куртиса—Хиршфельдера. В некотором смысле выбор между одношаговыми и многошаговыми методами — дело вкуса, т. к. результаты с точки зрения пользователя будут примерно одинаковы, а тонкости реализации — удел профессионалов.

Существует несколько библиотек, где методы общего назначения весьма тщательно реализованы различными авторами. Основные отличия заключаются в стратегии автоматического выбора шага, обеспечивающей требуемую локальную погрешность. Современные численные методы выбирают максимально большой шаг, обеспечивающий не глобальную, этого не может гарантировать не один метод, а локальную погрешность вычислений. Включе-

ние нескольких методов в пакет моделирования, с возможным функциональным дублированием, пока еще оправдано, т. к. повышает доверие к полученным результатам, если результаты расчета различными программами совпадают.

Разнообразие численных методов также есть следствие применения очень простого правила: для решения любой задачи с ярко выраженными особенностями целесообразнее применять метод, учитывающий эти особенности, и чем больше таких методов, тем лучше.

К числу особых задач, которые удается распознавать автоматически, по ходу построения численного решения, относятся системы жестких обыкновенных дифференциальных уравнений. Эти уравнения обладают той особенностью, что практически мгновенно выходят при любых начальных условиях на некоторое особое, часто медленно меняющееся решение. В нашей стране на этот класс задач обратил внимание Ю. В. Ракитский, он предложил свое определение жесткости и свой оригинальный способ решения жестких задач [17]. Многие его идеи до сих пор не нашли должного воплощения, но интерес к ним постоянно растет.

На практике для решения жестких задач наиболее часто используют неявные разностные схемы, в которых на каждом шаге решают нелинейные алгебраические уравнения относительно искомого решения. Именно поэтому они и называются *неявными*, в отличие от явных схем, где искомое решение вычисляется по заранее известным формулам. Наиболее простым методом этой группы является *неявный метод Эйлера*. Применение неявных методов оправдано только в том случае, если накладные расходы на решение уравнений на каждом временном шаге компенсируются большой величиной шага интегрирования. Шаг интегрирования в неявных схемах должен быть много большим, чем в явных, чтобы достигнуть существенного выигрыша. А это так и есть при решении жестких задач. Например, для жестких линейных систем шаг явных методов определяется максимальным по модулю собственным числом. В жестких задачах переходные процессы протекают "мгновенно", и переменные системы быстро выходят на стационарные или почти стационарные значения. Такие системы с "медленными" переменными можно интегрировать неявными методами с большим шагом интегрирования, много превышающим шаг интегрирования явных схем, без потери существенной информации о поведении системы.

Однако разнообразие численных методов, доступных пользователям современных пакетов моделирования, — не главное достижение разработчиков. Современные системы моделирования умеют решать последовательные, сво-

дящиеся к последовательно решаемым системам уравнений различного вида и размерности, и сопрягать их решения в точках смены поведения, и параллельные задачи — параллельно решать несколько систем уравнений, обменивающихся информацией между собой посредством сигналов. Это позволяет в рамках одной модели видеть последовательную смену режимов и параллельную работу компонентов системы.

Особый класс уравнений — это сверхбольшие уравнения, и соответствующий им особый класс численных методов. Сверхбольшие уравнения, возникающие при моделировании сложных многокомпонентных технических систем, обычно оказываются сильно разреженными или имеют специфическую структуру, чем можно воспользоваться для ускорения решения и экономии машинной памяти. Современные программные реализации численных методов обычно имеют специальные блоки, позволяющие работать с разреженными системами уравнений.

Однако и это еще не все. Существенно изменилась форма решаемых уравнений. Многие годы вычислители решали только уравнения, разрешенные относительно производных, а если и попадались уравнения другого типа, то их предварительно руками преобразовывали к нужному виду, к так называемой форме Коши. Сейчас ручные преобразования к форме Коши не обязательны. Пакеты позволяют оставлять уравнения в исходной форме, удобной для составления и интерпретации уравнений. Преобразование их к нужной форме происходит автоматически, без вмешательства пользователя.

Простейшие методы и их свойства

Методы Эйлера

Как утверждает теория, для построения приближенного решения дифференциального уравнения

$$\frac{dx}{dt} = f(x), \; x(0) = x_0$$

можно использовать явный

$$z_{n+1} = z_n + h \cdot f(z_n), \; z_0 = x_0$$

и неявный метод Эйлера

$$z_{n+1} = z_n + h \cdot f(z_{n+1}), \; z_0 = x_0 .$$

При этом в обоих случаях при $h \to 0$ приближенное решение z_n в точках $t_n = n \cdot h$, $n = 1, 2, \ldots$ стремится к решению $x(t_n)$ исходного уравнения.

Явный и неявный методы Эйлера можно получить, заменяя производную $\frac{dx}{dt}$ в точке t_n ее разностным аналогом $\frac{x(t_{n+1}) - x(t_n)}{h}$ при конечном значении h. Обозначая приближения к $x(t_{n+1})$, $x(t_n)$ через z_{n+1} и z_n соответственно, и вычисляя правую часть $f(x)$ либо в точке $f(z_n)$, либо в точке $f(z_{n+1})$, получаем приведенные приближенные формулы. Такая замена вносит в уравнение ошибку, пропорциональную $O(h^2)$ — $\frac{dx}{dt} = f(x) \Rightarrow \frac{z(t_{n+1}) - z(t_n)}{h} - f(z_n) \approx 0$. В случае линейного уравнения $\frac{dx}{dt} = \alpha \cdot x$, $x(0) = x_0$ вместо точного решения $\exp(\alpha \cdot t_n) x_0$ получаем $(1 + \alpha h)^n x_0$. Ошибка аппроксимации уравнения приводит к ошибке в решении, пропорциональной $O(h^2)$.

При малых h явный метод Эйлера "сохраняет" устойчивость исходного уравнения — если решение исходного уравнения устойчиво, то и приближенное решение устойчиво. Действительно, для уравнения $\frac{dx}{dt} = \alpha \cdot x$, $x(0) = x_0$ при $\alpha < 0$ для устойчивости приближенного решения необходимо выполнение неравенства $|(1 + \alpha h)| < 1$, что приводит к ограничению на шаг интегрирования, только при достаточно малых h разностное уравнение устойчиво.

Явный и неявный методы Эйлера с постоянным шагом практически не применяются в современных вычислениях, однако их целесообразно подробно анализировать в учебных целях. Создадим лабораторию для изучения методов Эйлера с помощью MvStudium.

Рассмотрим модельное уравнение

$$\frac{dx}{dt} = \alpha \cdot x + \beta, \; x(0) = x_0; \; t \in [0, T]$$

с известным решением

$$x(t) = e^{\alpha t} x_0 + \frac{\beta}{\alpha}\left(e^{\alpha t} - 1\right).$$

Создадим испытательный стенд для изучения методов. Нам понадобится решение x, численное решение y, с их помощью будем вычислять нормированные (коэффициент `Ker`) относительную `errerxyR` и абсолютную `errerxyA` ошибки (рис. 4.49). На решения `z`, `w` с начальными условиями dx, ddx пока обращать внимание не будем.

```
Система уравнений
    x' = alpha*x+betta;
    y = exp(alpha*Time)*x0+betta/alpha*(exp(alpha*Time)-1);
    z = exp(alpha*Time)*dx+betta/alpha*(exp(alpha*Time)-1);
    w = exp(alpha*Time)*ddx+betta/alpha*(exp(alpha*Time)-1);
    errorxyA = (x-y)*Ker;
    errorxyR = abs(errorxyA)/y;
Искомые переменные
    x, y, errorxyA, errorxyR, z, w
```

Рис. 4.49. Модельное уравнение

Для того чтобы получить приближенное решение явным методом Эйлера, необходимо решать разностное уравнение. Реализуем его с помощью карты поведения (рис. 4.50).

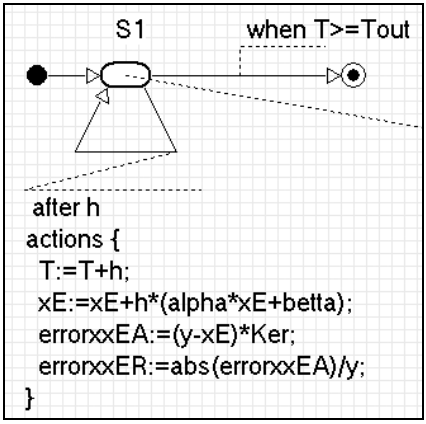
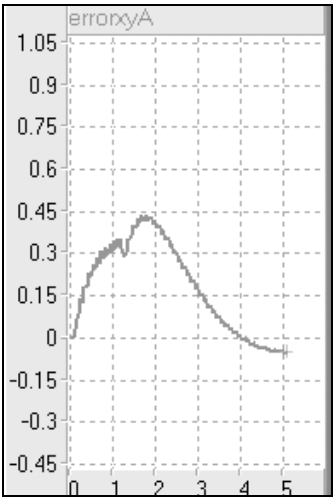
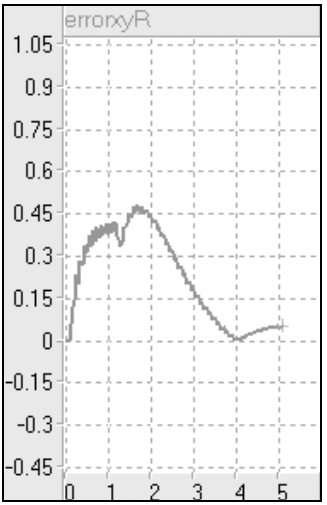


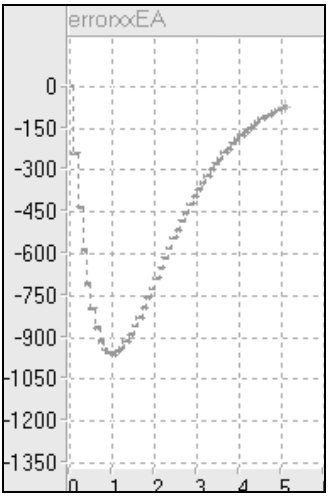
Рис. 4.50. Фрагмент карты поведения, реализующей решение разностного уравнения



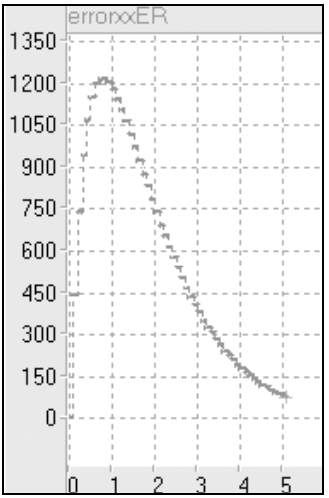
а



б



в



г

Рис. 4.51. Графики: *а* — абсолютной ошибки численного решения x; *б* — относительной ошибки численного решения *x*; *в* — абсолютной ошибки решения xE; *г* — относительной ошибки решения xE

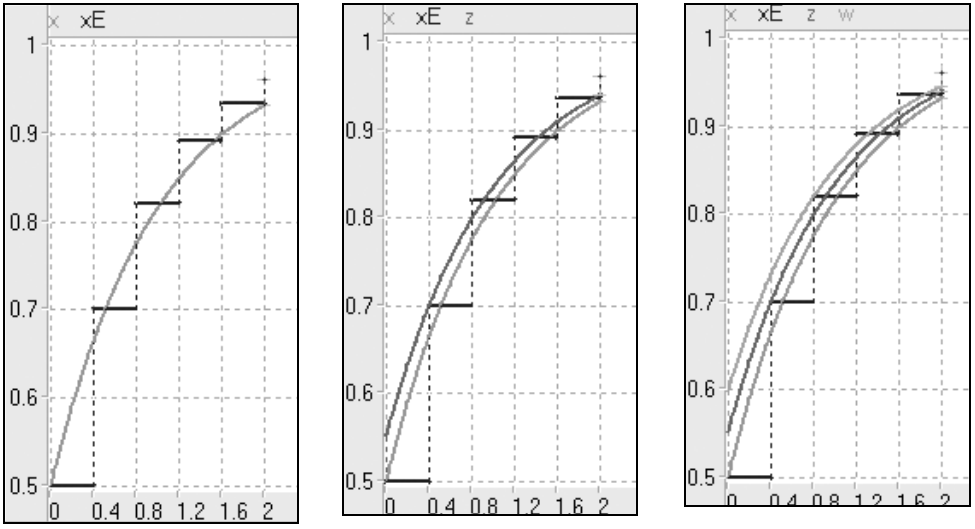
На рис. 4.51 показаны графики абсолютной и относительной ошибок численного решения x, полученного методом Рунге—Кутты с автоматическим вы-

бором шага и решения `xE`, построенного явным методом Эйлера с постоянным шагом $h = 0.1$, масштабный коэффициент `Ker=1E+5`, $\alpha = -1$, $\beta = 1$, $x_0 = 5$.

Даже для столь простого уравнения ошибка метода Эйлера огромна. Конечно, уменьшение шага приведет к ее уменьшению, но на практике разумнее пользоваться готовыми программами, реализующими более сложные методы с автоматическим выбором шага.

Посмотрим более детально, как ведет себя ошибка метода Эйлера.

На рис. 4.52, *а* представлены приближенные решения `x` (непрерывная кривая) и `xE` (ступенчатая кривая с шагом 0.4). Величина шага в данном случае не имеет значения. Хорошо видно, что приближенное решение `xE` быстрее стремится к устойчивому решению $-\dfrac{\alpha}{\beta}$, а также как накапливается глобальная погрешность — она не равняется сумме локальных погрешностей на каждом шаге.



а *б* *в*

Рис. 4.52. Графики точного, приведенных и приближенного решений:
а — для решения `x` и `xE`;
б — для решения `x`, `xE` и `z`; *в* — для решения `x`, `xE`, `z` и `w`

На первом шаге локальная и глобальная погрешности совпадают, т. к. мы начинаем с одной и той же точки x_0, принадлежащей истинному решению. На втором шаге (рис. 4.52, *б*), чтобы получить аналогичную картину, необходимо предположить, что мы интегрируем не исходное уравнение, а уравнение с измененными начальными условиями — приведенное уравнение. Именно оно совпадает с приближенным в точке 0.4. Продолжая дальше, мы видим, что каждый новый шаг соответствует новым начальным условиям dx, ddx (рис. 4.53) для приведенных решений `w`, `z` (см. рис. 4.50).

```
ddx:=x0+h*(alpha*x0+betta);
dx:=exp(-alpha*h)*(ddx-betta/alpha*(exp(alpha*h)-1));
ddx:=ddx+h*(alpha*ddx+betta);
ddx:=exp(-alpha*2*h)*(ddx-betta/alpha*(exp(alpha*2*h)-1));
```

Рис. 4.53. Вычисление начальных условий для приведенных решений (фрагмент карты поведения)

В итоге приближенное решение в данном случае все равно стремится к устойчивому решению, но по траектории, определяемой приведенными решениями, позволяющими вычислить величину глобальной погрешности.

Задания

1. Постройте программу, которая находила бы все приведенные решения, для любого этапа построения приближенного решения явным методом Эйлера.
2. Создайте аналогичную программу для неявного метода Эйлера.

Метод трапеций

Решения разностных уравнений, аппроксимирующих дифференциальные уравнения, должны сохранять основные свойства решений исходных уравнений. Разностные уравнения, соответствующие методам Эйлера, сохраняют например, устойчивость, только при определенных значениях шага интегрирования. Интересно посмотреть, что происходит с другими свойствами при аппроксимации дифференциального уравнения разностным. Например, в решении исходного уравнения есть синусоидальные колебания. Будем ли мы

их наблюдать в решении разностного уравнения? Проведем простейший эксперимент.

Рассмотрит матрицу

$$\mathbf{A} = \begin{bmatrix} \mu & \nu \\ -\nu & \mu \end{bmatrix},$$

где μ, ν — вещественные числа, и систему

$$\frac{d\mathbf{x}}{dt} = \mathbf{A} \cdot \mathbf{x}, \ \mathbf{x}(0) = \mathbf{x}_0 .$$

Решение этой простой системы известно:

$$x_1(t) = e^{-\mu t}\cos(\nu t) \cdot x_{10} + e^{-\mu t}\sin(\nu t) \cdot \left(\frac{2\mu}{\nu} x_{10} + x_{20} \right),$$

$$x_2(t) = e^{-\mu t}\cos(\nu t) \cdot x_{20} + e^{-\mu t}\sin(\nu t) \cdot \left(\frac{2\mu}{\nu} x_{20} - x_{10} \right),$$

и его можно использовать для проверки точности численного решения.

Наше основное утверждение — современным программным реализациям численных методов можно доверять. Испытаем программные реализации на этом простейшем тестовом примере. Рассмотрим численные методы для решения обыкновенных дифференциальных уравнений, включенные в MvStudium (рис. 4.54).

Помимо программы, названной Автомат, существуют явные, неявные и отладочные методы. Автомат — это название программы, пытающейся автоматически решить предложенную задачу различными методами, если один из них или несколько не смогли это сделать самостоятельно. Она всегда начинает с самой экономичной среди включенных в пакет программы явного метода Рунге—Кутты, в предположении, что решаемая задача нежесткая. Обнаружив жесткость, она автоматически переключается на жесткие методы из пакета ODEPACK (неявные линейные многошаговые методы), и обратно, если жесткость уменьшается. Если эта пара методов по какой-то причине отказывается решать задачу, программа автоматически переключается на другие методы из числа перечисленных далее. По умолчанию всегда работает программа Автомат.

Пользователь может отказаться от программы Автомат и сам назначить метод интегрирования. Среди явных и неявных методов предпочтение отдается методам Рунге—Кутты (DOPRI5, DOPRI853 и RADAU5). Особо стоит выде-

лить программу DDASSL, умеющую решать уравнения, не разрешенные относительно производных. Две отладочные программы (программа, названная методом Эйлера, — это модифицированная программа LSODA, умеющая переключаться с явного метода Эйлера на неявный метод Эйлера) предназначены для решения "сырых" задач, у которых правые части уравнений, возможно, еще содержат ошибки. В таких случаях все остальные программы могут вести себя странно с точки зрения пользователя, что мешает отладке.

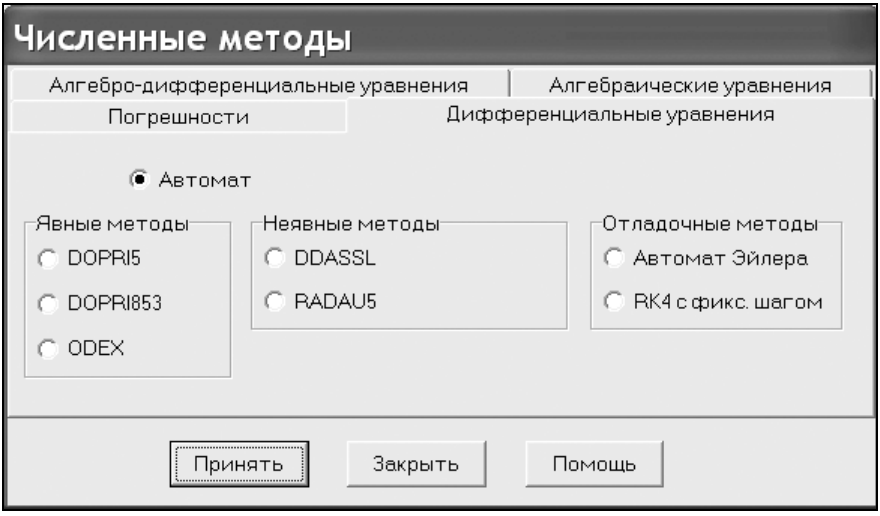


Рис. 4.54. Численные методы для решения обыкновенных дифференциальных уравнений, используемые в MvStudium

Это перечисление методов нужно нам для организации эксперимента по изучению воспроизведения синусоидальных, "чистых" колебаний.

Создадим программу для тестирования численных методов на простейшей "колебательной" задаче.

Как видно из рис. 4.55, решение подобрано так, чтобы фазовый портрет представлял из себя круг — в этом случае изменение радиуса `R` будет характеризовать ошибки в воспроизведении амплитуды. Одновременно вычисляется и вектор ошибки численного решения `d`.

На рис. 4.56 показаны соответствующие графики для численного решения, построенного методом Эйлера из группы отладочных программ.

```
Система уравнений
    x' = A*x;
    R = (sqrt(x[1]**2+x[2]**2)-vnorm2(x0))*Ker;
    R1 = R;
    v = exp(-mu*Time);
    v1 = cos(nu*Time);
    v2 = sin(nu*Time);
    w1 = (v1*x0[1]+v2*(2*mu/nu*x0[1]+x0[2]))*v;
    w2 = (v1*x0[2]+v2*(2*mu/nu*x0[2]-x0[1]))*v;
    xT = [w1; w2];
    d = x-xT;
Искомые переменные
    x, R, R1, v, v1, v2, w1, w2, xT, d
```

Рис. 4.55. Тестовое уравнение, его решение, ошибка численного решения

Проверьте, как ведут себя программные реализации методов Рунге—Кутты, и вы убедитесь, что ничего подобного вы не наблюдаете. На достаточно больших временных интервалах у численного решения, найденного с помощью этих современных программ, амплитуда колебаний остается практически постоянной.

Наряду с "рабочими" методами обратимся к простейшим методам Рунге—Кутты, уже рассмотренным методам Эйлера и методу трапеций. Это действительно методы Рунге—Кутты. Если разностную схему метода Рунге—Кутты записать в параметрической форме, с параметром p

$$z_{n+1} = z_n + h \cdot k_1,$$
$$k_1 = f\left(t_n + p \cdot h, z_n + p \cdot h \cdot k_1\right),$$

то при $p = 0$ получим явный метод Эйлера, при $p = 1$ — неявный, а при $p = \frac{1}{2}$ — метод трапеций или метод средней точки.

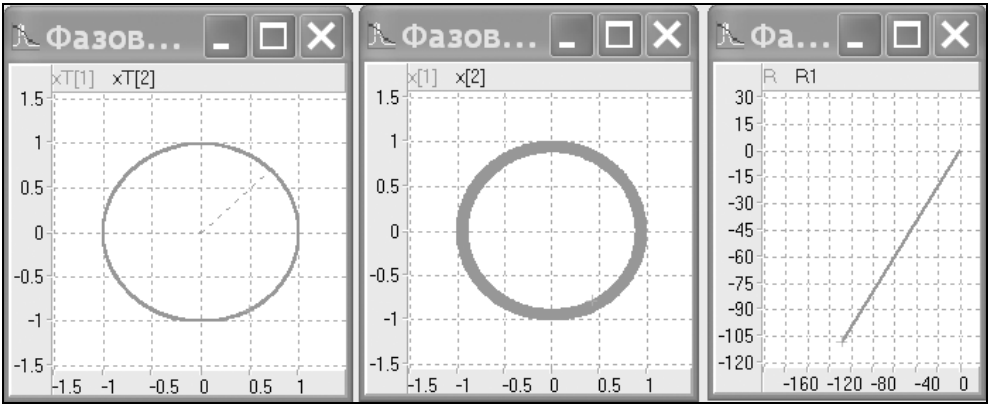
Посмотрим, как они работают на нашей тестовой задаче (рис. 4.57).

Как видно из фазовых портретов, только метод трапеций сохраняет амплитуду неизменной. Это можно строго обосновать, рассматривая простейшее скалярное уравнение

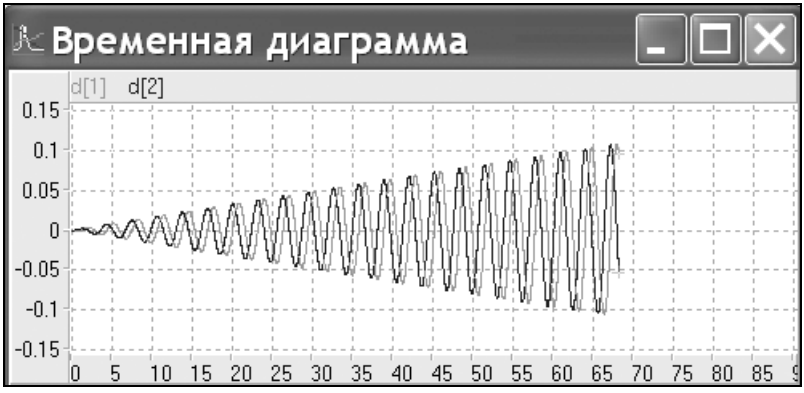
$$\frac{dx}{dt} = \lambda x.$$

Решение разносного уравнения, соответствующее рассмотренным методам с параметром p, можно записать в виде

$$\begin{aligned} z_{n+1} &= z_n + h \cdot k_1 \\ k_1 &= f\left(t_n + p \cdot h, z_n + p \cdot h \cdot k_1\right) \end{aligned} \Rightarrow z_{n+1} = \rho_p(h\lambda) \cdot z_n;\ \rho_p(h\lambda) = \frac{1+(1-p)h\lambda}{1-ph\lambda}.$$

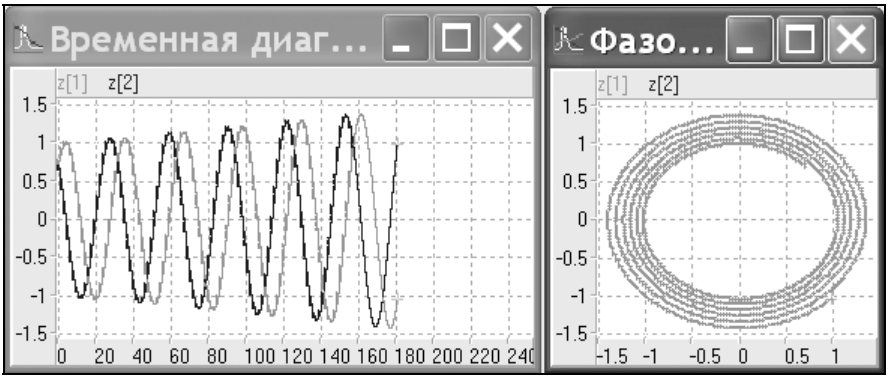


а

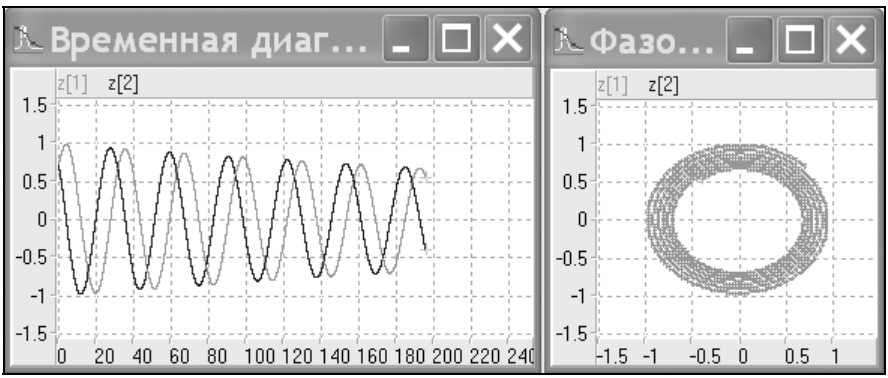


б

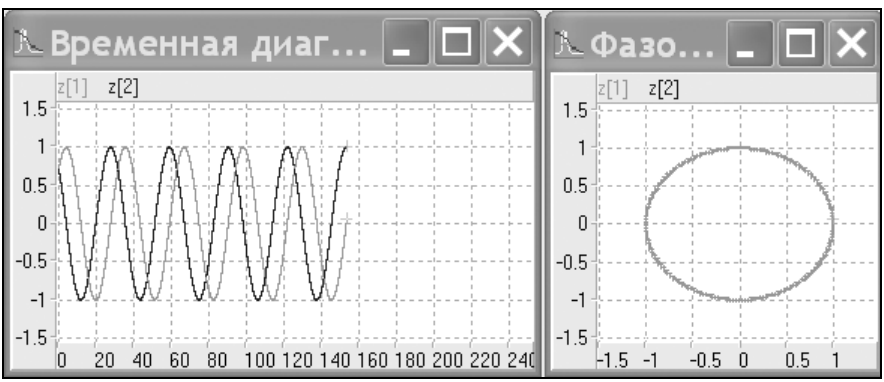
Рис. 4.56. Работа метода Эйлера из группы методов для отладки:
а — ошибка амплитуды "отладочного" метода Эйлера;
б — ошибка численного решения "отладочного" метода Эйлера



а



б



в

Рис. 4.57. Простейшие методы Рунге—Кутты и колебательная тестовая задача: *а* — явный метод Эйлера, $p = 0$; *б* — неявный метод Эйлера, $p = 1$; *в* — метод трапеций, $p = 1/2$

Анализируя множитель $\rho_p(h\lambda)$, можно сказать, методы Эйлера имеют первый порядок точности (в разложении множителя в ряд Тейлора по степеням h совпадение с экспонентой, решением исходного уравнения, до второго члена), а метод трапеций — второго. Условие $\left|\rho_p(h\lambda)\right| < 1$ при комплексных значениях $h\lambda$ позволяет ответить на вопрос об областях устойчивости всех трех методов. Оказывается, что метод трапеций устойчив во всей левой полуплоскости, включая мнимую ось (области устойчивости обычно строят в осях $\mathrm{Im}(h\lambda)$, $\mathrm{Re}(h\lambda)$). Подставляя в выражения для множителя комплексное собственное число $-\mu + i\cdot\nu$, соответствующее устойчивому решению

$$\rho_p(h\lambda) = \frac{[1-(1-p)h\mu] + i[(1-p)h\nu]}{[1-ph\mu] + i[-ph\nu]} = e^{-h\tilde{\mu}} \cdot e^{-ih\tilde{\nu}},$$

и сравнивая его с таким же коэффициентом для решения дифференциального уравнения

$$\rho(h\lambda) = e^{-h\mu} \cdot e^{-ih\nu},$$

можно сказать, насколько искажаются вещественная и мнимая части собственного числа численным методом. Итак, вместо μ, ν мы имеем $\tilde{\mu}$, $\tilde{\nu}$. Вычисляя значения $\tilde{\mu}$, $\tilde{\nu}$ для "чистых" колебаний ($\mu = 0$), можно увидеть, что только метод трапеций не искажает амплитуду, явный метод Эйлера — увеличивает, а неявный — уменьшает с течением времени. Это согласуется с рис. 4.57.

Жесткие задачи

Рассмотрим простейший пример

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} + \mathbf{b};\ \mathbf{x} \in \Re^2;\ \mathbf{A} = \mathbf{S}\cdot\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}\cdot\mathbf{S}^{-1};$$

$$\lambda_1, \lambda_2 \in \Re^2;\ \lambda_1, \lambda_2 < 0\ \ \&\ \ |\lambda_1| >> |\lambda_2|$$

системы второго порядка, с действительными отрицательными собственными числами, одно из которых по модулю много больше второго. Вспоминая, что однородная система имеет два независимых решения $e^{\lambda_1 t}$, $e^{\lambda_2 t}$, и первое из собственных чисел по модулю много больше второго, следует ожидать, что координаты $x_1(t)$ и $x_2(t)$ очень быстро станут линейно-зависимыми уже

для небольших значений времени t. Можно сказать, что такая система ведет себя сначала как обычная система второго порядка, и это продолжается до тех пор, пока первая экспонента не станет много меньше, чем вторая, и ее вклад в координаты $x_1(t)$ и $x_2(t)$ с учетом стоящих перед ней множителей не станет пренебрежимо мал. Далее обе координаты будут зависеть только от второго независимого решения $e^{\lambda_2 t}$ и, следовательно, можно считать, что мы имеем дело с одним дифференциальным уравнением. А еще через некоторое время решение станет равным $\mathbf{x}(t) \approx -\mathbf{A}^{-1} \cdot b$; другими словами, можно говорить, что имеем дело не с дифференциальными уравнениями, а с линейными алгебраическими. Переходные процессы будут затухать тем быстрее, чем больше по модулю собственные числа.

Покажем, как MvStudium справляется с такими жесткими тестовыми задачами.

```
Model
Параметры
  b: vector[2] := [-1; 1];
  lambda1: double := -2;
  lambda2: double := -1;
  Tout: double := 12;
  u: vector[2] := [2; 2];
  v: vector[2] := [3; -3];
  x0: vector[2] := [-10; 10];
Переменные
  A: matrix[2,2] := S*L*Sinv;
  L: matrix[2,2] := [lambda1,0; 0,lambda2];
  S: matrix[2,2] := eye(2)+u*transp(v);
  Sinv: matrix[2,2] := eye(2)-u*transp(v);
  t: double := 0;
  x: vector[2] := x0;
  xprim: vector[2];
  y: vector[2];
  y0: vector[2] := x0;
```

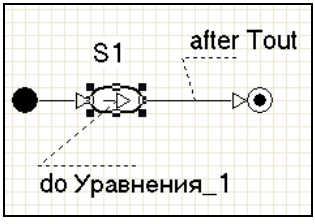
Рис. 4.58. Переменные модели

На рис. 4.58 показаны использованные в модели переменные. Уравнение и его решение представлены на рис. 4.59, они оформлены в виде локального

класса. Обратите внимание, как вычисляется производная решения — переменной `xprim` просто присвоено значение производной. Напомним, что это всего лишь условное обозначение, никакого дифференцирования не происходит, переменной `xprim` автоматически присваивается значение правой части, т. е. производной. Производная нормируется значением евклидовой нормы решения.

```
Система уравнений
    x' = A*x+b;
    xprim = 1/vnorm2(x)*x';
    w1 = exp(lambda1*(Time-t));
    w2 = exp(lambda2*(Time-t));
    W = S*[w1,0; 0,w2]*Sinv;
    y = W*y0+inv(A)*(W-eye(2))*b;
Искомые переменные
    x, xprim, y, w1, w2, W
```

Рис. 4.59. Тестовое уравнение и его решение



а

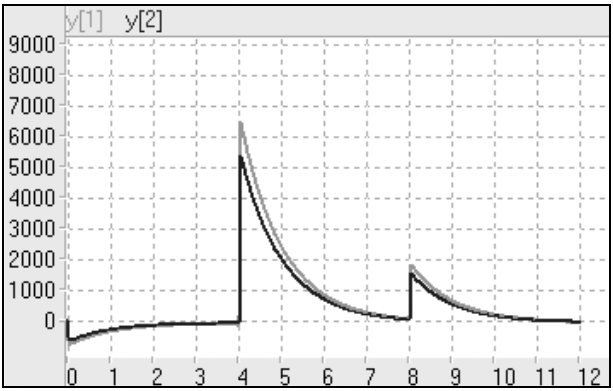
```
after 4
actions {
 t:=Time;
 --Апериодическое движение
 x[1]:=100;
 y0[1]:=100;
 y0[2]:=y[2];
 --y0[2]:=y[2];
 --Периодическое движение
 --x:=x0;
 --y:=y0;
}
```

б

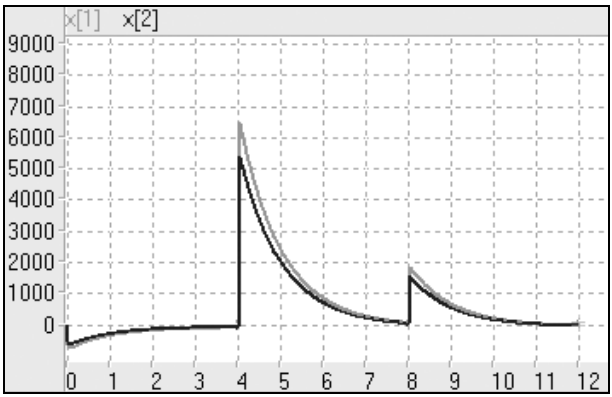
Рис. 4.60. Использование внутреннего перехода:
а — карта поведения; *б* — внутренний переход

На рис. 4.60, *а* показана карта поведения. Впервые в этой главе применена карта поведения с внутренним переходом. Переход нужен для того, чтобы вносить возмущение в решение и наблюдать за реакцией системы. Обратите внимание на то, как формируется возмущение (рис. 4.60, *б*), и на то, что при срабатывании перехода повторно вычисляется значение переменной `t`, нужное для правильного вычисления глобального времени.

На рис. 4.61, *а* показано решение уравнения при $\lambda_1 = -200$, на рис. 4.61, *б* — соответствующе ему численное решение, а на рис. 4.61, *в* — значение нормированной производной.

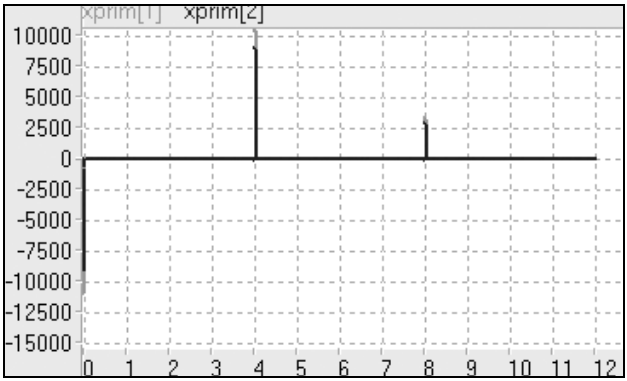


а



б

Рис. 4.61. Решение уравнения при $\lambda_1 = -200$:
а — решение уравнения; *б* — численное решение



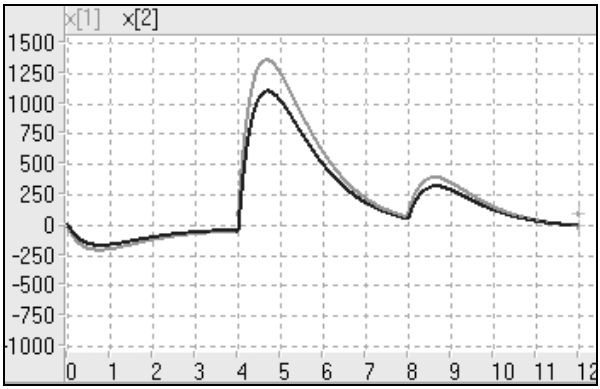
в

Рис. 4.61. Решение уравнения при $\lambda_1 = -200$:
в — нормированное значение производной

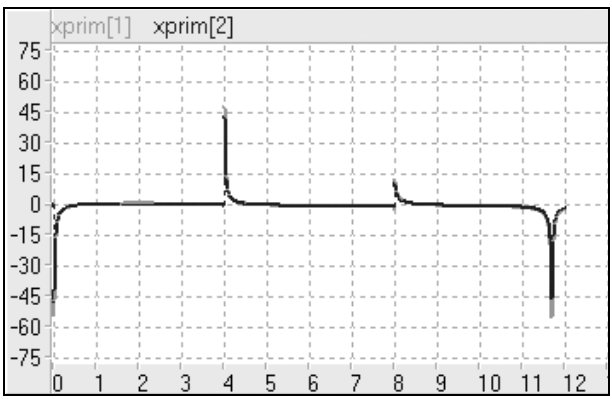
Сравните эти кривые с соответствующими кривыми рис. 4.62. Очевидно, что переходные процессы различны — в одном случае они протекают почти мгновенно, в другом — значительно медленнее. Это и есть проявление жесткости. Система быстро забывает свои начальные условия и стремится к некоторому медленно меняющемуся решению.

а

Рис. 4.62. Решение уравнения при $\lambda_1 = -2$:
а — решение уравнения



б



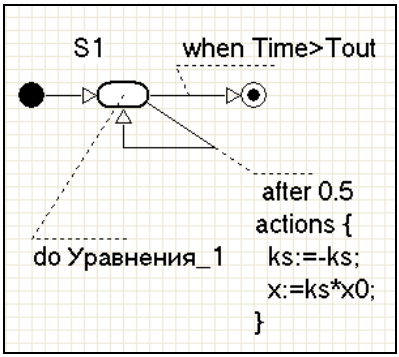
в

Рис. 4.62. Решение уравнения при $\lambda_1 = -2$:
б — численное решение; *в* — нормированная производная

На этих медленно меняющихся участках и возникают связи между переменными с приемлемой для практики точностью. Эти связи можно использовать для понижения порядка решаемой системы.

В рассмотренном примере решение системы стремится к устойчивому "стационарному" и постоянному решению $-\mathbf{A}^{-1} \cdot \mathbf{b}$, однако, как легко себе представить, в общем случае после быстрого протекания переходного процесса дальнейшее движение не обязательно должно быть постоянным. Примером может служить модель, представленная на рис. 4.63 [17]. Ре-

шаемые уравнения показаны на рис. 4.63, *а*, с соответствующим описанием переменных модели — на рис. 4.63, *б*.



а

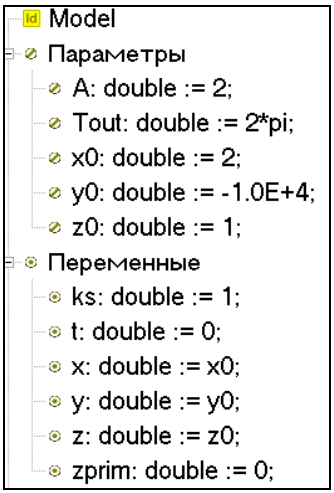
$$\frac{dx}{dt} = y\cdot(x-z)+\frac{dz}{dt}$$

$$\frac{d^2z}{dt^2} = -A\cdot\sin(z)$$

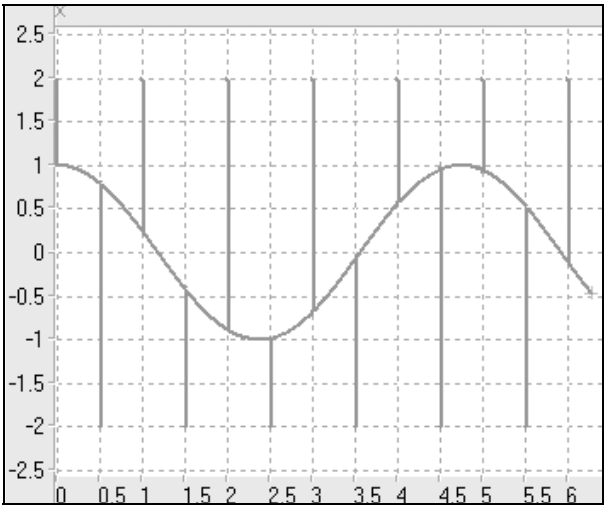
$$\frac{dy}{dt} = -6\cdot(\mathrm{Time}+t)$$

$$zprim = \left.\frac{dz}{dt}\right|$$

б



в



г

Рис. 4.63. Пример жесткой системы: *а* — решаемая система; *б* — переменные модели; *в* — карта поведения; *г* — численное решение

Обратите внимание на карту поведения (рис. 4.63, *в*) — она реализована с помощью внешнего перехода, с помощью которого осуществляется возму-

щение переменной x. Напомним, что отличие карты поведения с внешним переходом, аналогичной рис. 4.63, *в*, от карты поведения с внутренним переходом, подобной рис. 4.60, *а*, заключается в том, что экземпляр локального класса `Уравнения_1`, приписанный состоянию `S1`, уничтожается при выходе из состояния. Переменная `Time`, используемая в уравнениях, показывает текущее время с момента очередного создания экземпляра, поэтому можно воспользоваться глобальной переменной `t` для учета глобального времени. Рисунок 4.63, *г* показывает, что возмущение практически мгновенно "отрабатывается" жестким уравнением, и переменная x выходит на невозмущенное решение $\cos(t)$. Можно легко показать, что $\cos(t)$ и есть решение для переменной x вдали от пограничного слоя, который в данном случае чрезвычайно мал.

Действительно, вместо исходной системы рассмотрим новую систему (рис. 4.64, *а*). В новых уравнениях изменены уравнения математического маятника — они стали линейными, и решение $z(t)$ можно записать в явном виде $z(t)=\cos(t)$ при выбранных начальных условиях. Тогда уравнение для переменной x (положим $A=1$) преобразуется к виду

$$\frac{dx}{dt}=y\cdot(x-z)+\frac{dz}{dt},$$

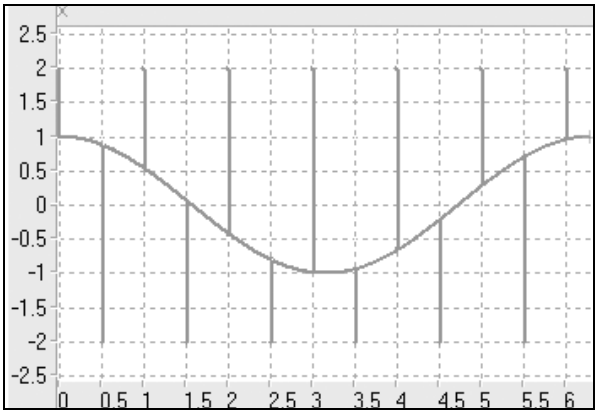
и, если учесть, что и уравнение по y явно разрешимо — $y(t)=y_0-3t^2$, то решение можно найти явно $x(t)=e^{-10^4t-t^3}(x_0-1)+\cos(t)$. Его мы и видим на рис. 4.64, *б*, и оно практически неотличимо от решения, показанного на рис. 4.63, *а*.

$$\frac{dx}{dt} = A\cdot y\cdot\left(x-\cos\left(\mathbf{Time}+t\right)\right)+A\cdot\sin\left(\mathbf{Time}+t\right)$$

$$y = -1.0E+4-3\cdot\left(\mathbf{Time}+t\right)^2$$

а

Рис. 4.64. Упрощенная модельная задача:
а — уравнения новой системы, подготовленные для MvStudium



б

Рис. 4.64. Упрощенная модельная задача:
б — найденное решение

Решение как исходной, так и упрощенной жесткой системы можно найти, используя принцип квазистационарности производных [17] — вне пограничного слоя производные жестких уравнений $\frac{dx}{dt} = f(x)$ малы, и уравнение $0 \approx f(x)$ можно использовать для нахождения связи между переменными. В данном случае для упрощенного уравнения можно найти решение в явном виде. Действительно, решение $\tilde{x} = \cos(t) - \frac{\cos(t)}{10^4 + 3t^2}$ с высокой степенью точности аппроксимирует решение исходного уравнения вне пограничного слоя.

Упражнения

На самом деле в работе [17] утверждается, что для построения связей между переменными можно использовать не только правые части, соответствующие первым производным, но выражения для производных более высокого порядка. Проверьте это утверждение экспериментально на приведенном уравнении и обоснуйте его в общем случае. Покажите, что для рассматриваемого уравнения, если использовать не уравнение $\frac{dx}{dt} \approx 0$, а уравнение $\frac{d^2x}{dt^2} \approx 0$, то аппроксимирующее решение будет иметь вид

$$\tilde{x} = \cos(t) + \frac{\cos(t)}{\left(10^4 + 3t^2\right)^2 - 6t}$$

и стремиться к точному еще быстрее! Что произойдет, если использовать более высокую производную?

В заключение этого раздела обсудим, как включенные в пакет программные реализации методов справляются с решением жестких задач. Для этого создадим тестовую коллекцию. Задачи программы очень просты — пользователь выбирает тестовый пример, устанавливает длину промежутка интегрирования, если нужно, меняет значение параметров примера, указывает, какой метод использовать, и смотрит графики решения. Напомним, что в тестовых примерах точное решение неизвестно и правильность решения можно проверить:

- с помощью расчетов с повышенной точностью (в нашем случае это невозможно, т. к. все программы работают с двойной точностью);
- сверяя решение с решением, найденным другой программой (именно этого в данном случае мы и хотим);
- следя за инвариантами и качественными показателями. Генераторами тестовых примеров называют программы, формирующие параметрические системы дифференциальных уравнений с заданными значениями параметров и известным решением, обладающие требуемым качеством: мягкие (нежесткие), жесткие, колебательные и любые другие.

На рис. 4.65 приведена главная карта поведения разрабатываемой программы. Класс `Model`, ему принадлежит главная карта поведения, имеет две глобальные переменные `N_test` (номер теста) и `Tout` (время проведения эксперимента). Управление экспериментом осуществляется в состоянии `S1`, входные действия которого сводятся к диалогу с пользователем. Пользователь вводит номер теста и время эксперимента. Первый литерал среди аргументов команды `Message` — имя диалогового окна, второй — собственно текст сообщения. Команда `InputVar` вводит нужное значение. Обе команды приостанавливают свое выполнение до тех пор, пока пользователь не введет информацию (окончание ввода — нажатие клавиши <Enter>).

На рис. 4.65 приведен графический вариант оператора `if-else`, операторами которого являются действия и активности гиперсостояний `S2`—`S7`. Каждому из этих состояний приписан один из классов 1—6, содержащий тестовые примеры. Эта фраза означает, что состояние `Si` реализует поведение класса `Класс_i`. Такая схема предполагает динамическое создание экземпляров: если гиперсостояние `Si` становится текущим, создается экземпляр класса `Класс_i`, и управление передается ему. По окончанию работы (истечение времени `Tout`) экземпляр класса `Класс_i` уничтожается.

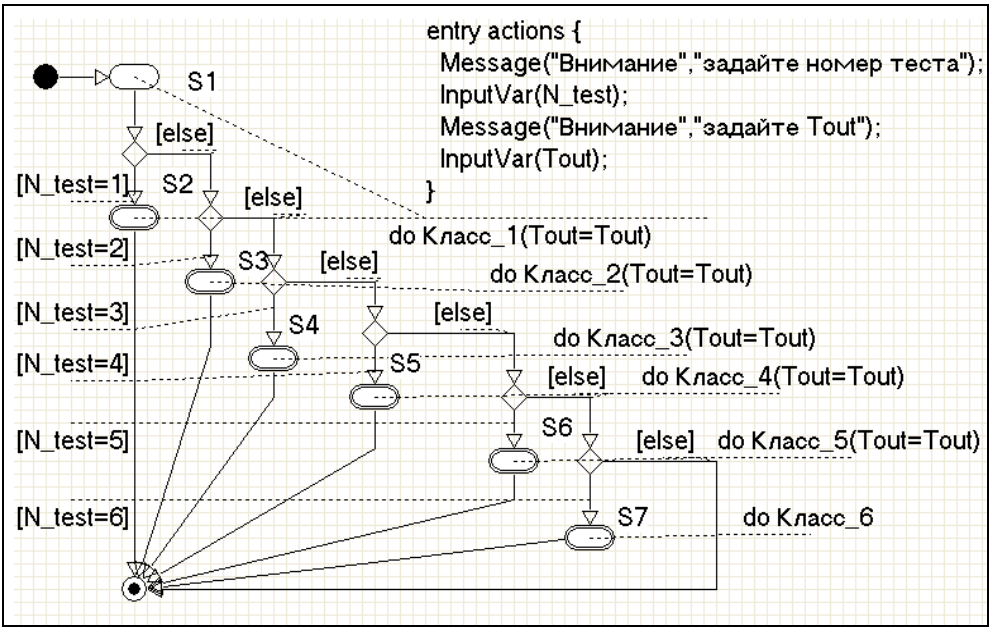


Рис. 4.65. Главная карта поведения класса `Model`

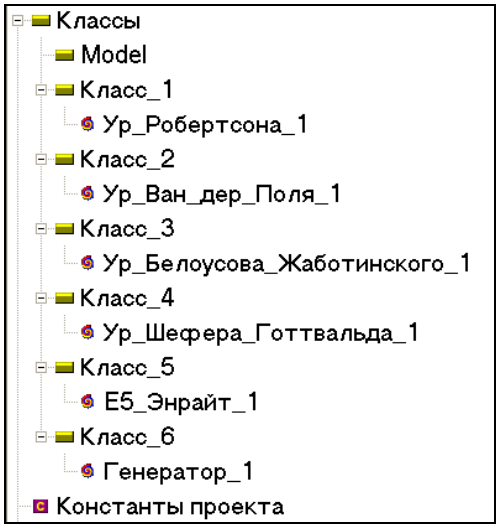


Рис. 4.66. Список классов
и их непрерывных поведений

Передать нужную классу информацию в экземпляр класса в данном случае можно только через механизм параметров. Установите курсор внутри прямоугольника со спрямленными краями, изображающего состояние. Правой кнопкой мыши откройте контекстное меню. Выберите команду **Действительные значения параметров и переменных деятельности**. Выберите и измените любую переменную из видимых в открывшемся окне. В частности, переменным в этом окне можно присвоить любое из значений глобальных переменных. Точно так же можно при завершении текущего состояния значения локальных переменных экземпляра класса присвоить глобальным (в заключительных действиях состояния или действиях перехода, ведущих из состояния).

Список тестовых примеров приведен на рис. 4.66. Первые пять примеров взяты из книги [14], в книге приведены графики их решений, что позволяет убедиться, что они воспроизводятся в пакете MvStudium правильно. Последний пример — это генератор систем линейных дифференциальных уравнений, состоящих из 6-ти уравнений вида

$$\frac{d\mathbf{x}}{dt} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b},$$

с постоянными матрицей $\mathbf{A}$ и вектором $\mathbf{b}$ с заданными собственными числами. Матрица $\mathbf{A}$ с помощью матрицы $\mathbf{S} = \mathbf{E} + \mathbf{u} \cdot \mathbf{v}'$; $\mathbf{u}, \mathbf{v} \in R^n$ и обратной для нее $\mathbf{S}^{-1} = \mathbf{E} - \mathbf{u} \cdot \mathbf{v}'$; $(\mathbf{u}, \mathbf{v}) = 0$ приводится к виду

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu & \nu & 0 & 0 \\ 0 & 0 & -\nu & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_2 \end{bmatrix}.$$

Точнее, задается блочно-диагональная матрица с собственными числами λ, λ_1, λ_2, $\mu \pm i\nu$ и по ней с помощью матриц $\mathbf{S}$, $\mathbf{S}^{-1}$ строится матрица $\mathbf{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}$.

Упражнение

Постройте решение приведенной тестовой системы в явном виде. Воспользуйтесь тем, что решение системы $\frac{d\mathbf{x}}{dt} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$ непосредственно связано с решением системы $\frac{d\mathbf{y}}{dt} = \mathbf{\Lambda} \cdot \mathbf{y} + d$, $\mathbf{y} = \mathbf{S}^{-1} \cdot \mathbf{x}$. Попробуйте выбирать векторы $\mathbf{u}$, $\mathbf{v} \in R^n$ так, чтобы матрица $\mathbf{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{\Lambda}^{-1}$ не содержала ошибок округления, т. е. обеспечьте построение матрицы $\mathbf{A}$ с заданными и точно представленными в машинной арифметике собственными числами λ, λ_1, λ_2, $\mu \pm i\nu$.

Перейдем к составлению программы. Описание переменных приведено на рис 4.67.

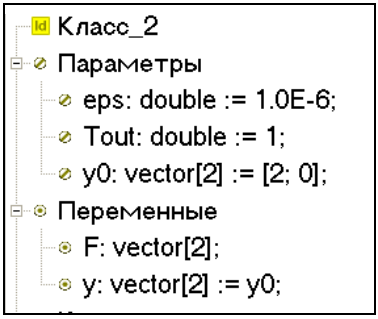


Рис. 4.67. Параметры и переменные отдельных классов

На рис. 4.68 показана главная карта поведения отдельного класса. Она будет одной и той же для всех тестовых примеров.

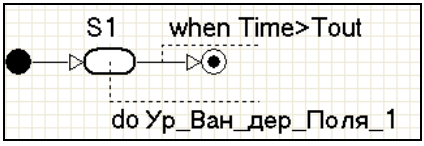


Рис. 4.68. Главная карта поведения отдельного класса

Системы уравнений каждого теста могут быть заданы различными способами.

Описание первой тестовой задачи приведено на рис. 4.69 . Уравнения заданы в матричной форме, коэффициенты матрицы зависят от искомых переменных. Система имеет инвариант `s=1`, все компоненты решения положительны, т. к. речь идет о концентрациях. Масштабный коэффициент `k2` нужен для построения графика.

```
Система уравнений
    A = [-0.04,1.0E+4*y[3],0; 0.04,-3.0E+7*y[2],-1.0E+4*y[2]; 0,3.0E+7*y[2],0];
    y' = A*y;
    y2 = k2*y[2];
    s = y[1]+y[2]+y[3];
Искомые переменные
  y, A, y2, s
```

Рис. 4.69. `Класс_1`. Уравнения, описывающие реакцию Робертсона

К числу тестовых отнесено и уравнение Ван-дер-Поля (рис. 4.70). Здесь мы снова воспользовались векторной формой.

$$F = \left[y_2;\ \frac{\left(1-y_1^2\right)\cdot y_2 - y_1}{eps}\right]$$

$$\frac{dy}{dt} = F$$

Рис. 4.70. `Класс_2`. Уравнение Ван-дер-Поля

Параметр $\varepsilon = 10^{-6}$ в уравнении Ван-дер-Поля достаточно мал, решение имеет форму почти прямоугольных импульсов с очень резким задним фронтом (см. рис. 4.74, слева).

На рис. 4.71 показаны еще одни тестовые уравнения, уравнения Белоусова—Жаботинского. Переменные `g1`—`y3g3` нужны для воспроизведения графиков из книги [14].

`Класс_4` содержит уравнения Шеффера, предложенные Готвальдом в качестве теста (рис. 4.72). Все компоненты решения положительны.

$$F = [k1\cdot\left(y_2+y_1\cdot\left(1-k12\cdot y_1-y_2\right)\right);\ \frac{y_3-\left(1+y_1\right)\cdot y_2}{k2};\ k3\cdot\left(y_1-y_3\right)]$$

$$\frac{dy}{dt} = F$$

$$ylg1 = \lg\left(y_1\right)$$

$$ylg2 = \lg\left(y_2\right)$$

$$ylg3 = \lg\left(y_3\right)$$

Рис. 4.71. Класс_3. Уравнения реакции Белоусова—Жаботинского

```
Система уравнений
    y' = A*y+F;
    F = [0.0007; 0; 0; 0; 0; -280*y[6]*y[8]; 280*y[6]*y[8]; -280*y[6]*y[8]];
Искомые переменные
    y, F
```

Рис. 4.72. Класс_4. Уравнения Шеффера—Готвальда

```
Система уравнений
    y' = Fun(y);
Искомые переменные
    y
```

Рис. 4.73. Класс_5. Уравнения из коллекции Энрайта, Хала и Линберга

Уравнения из коллекции Энрайта, Хала и Линберга представляют очень жесткую задачу с сильно различающимися (на десять порядков) положительными компонентами (рис. 4.73). Для разнообразия они реализованы в виде функции (листинг 4.12).

Листинг 4.12. Реализация уравнений Энрайта, Хала и Линберга в виде функции

```
function Fun(in y: vector) return vector is
  z: vector;
begin
  z:=zeros(size(y));
```

```
  z[1]:=-A*y[1]-B*y[1]*y[3];
  z[2]:=-A*y[1]-M*C*y[2]*y[3];
  z[3]:=A*y[1]-B*y[1]*y[3]-M*C*y[2]*y[3]+C*y[4];
  z[4]:=-B*y[1]*y[3]-C*y[4];
  return z;
end Fun;
```

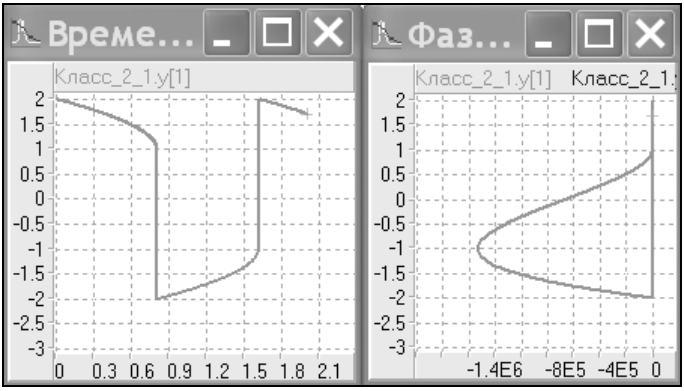
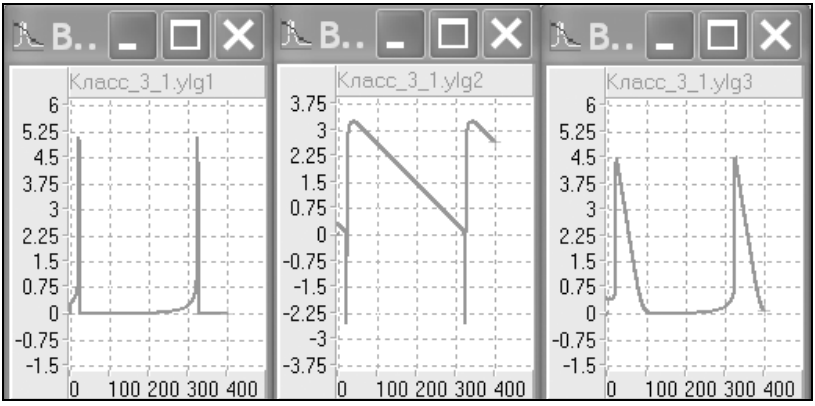
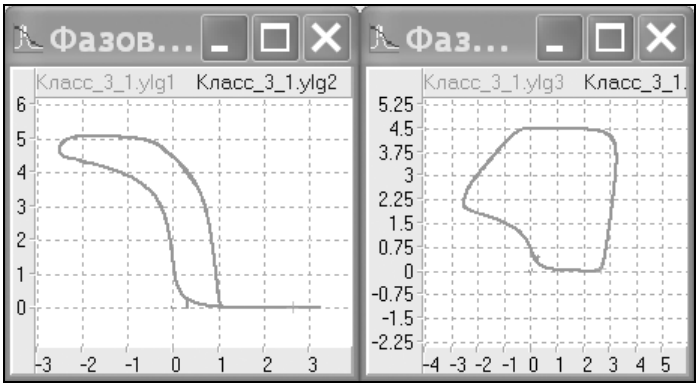


Рис. 4.74. Импульсы с вертикальными фронтами — решение уравнения Ван-дер-Поля



а

Рис. 4.75. Решение и фазовые портреты решения уравнений для реакции Белоусова—Жаботинского:
а — графики координат решения в логарифмическом масштабе



б

Рис. 4.75. Решение и фазовые портреты решения уравнений для реакции Белоусова—Жаботинского:
б — фазовые портреты в логарифмическом масштабе

`Класс_6` — генератор тестовых примеров, реализован в соответствии с приведенными выше формулами.

Вид решений на рис. 4.74 и 4.75 хорошо известен, воспроизведенные в MvStudium кривые узнаваемы, а убедиться в точности решений можно, проведя численные эксперименты.

Колебательные задачи

Теория линейных и нелинейных колебаний содержит множество хорошо изученных сложных задач, которые можно рассматривать как тестовые для пакетов моделирования. Построим несколько примеров воспроизведения колебаний с помощью пакета MvStudium.

Начнем с вынужденных колебаний осциллятора. Построим нелинейное устройство, на вход которого подается функция $x(t)$, а на выходе формируется заданная функция $y(t)$. Предположим, что выходную функцию $y(t) = F(x(t))$ можно представить в виде ряда $y(t) = a_1 x(t) + a_2 x^2(t) + a_3 x^3(t) + \ldots$. Первый член в этом разложении представляет линейное приближение к нашему устройству, второй и третий — приближения с квадратичной и кубической нелинейностями. В теории колебаний обычно рассмат-

ривают периодические входные функции, с периодом T, представленные рядом Фурье

$$x(t) = A_0 + \sum_{k=1}^{\infty} A_k \cos(k\omega \cdot t + \varphi_k), \quad \omega = \frac{2\pi}{T};$$

$$A_0 = c_0; \; A_k = 2|c_k|, \; \varphi_k = \arg c_k,$$

$$c_k = \frac{1}{T}\int_0^T x(t)\exp\left(-\frac{2\pi i k \cdot t}{T}\right)dt.$$

В этом представлении обычно исключают нулевой член с помощью выбора начала отсчета переменной $x(t)$ и говорят о гармонических колебаниях, если в разложении присутствует только первый член (первая гармоника), и ангармонических во всех остальных случаях. Члены ряда обычно называют гармониками. Ангармонические колебания характеризуются коэффициентом нелинейных искажений

$$\chi = \frac{\sqrt{A_2^2 + A_3^2 + ..}}{A_1}.$$

Создадим нелинейное устройство, модель которого можно представить отрезком ряда Тейлора $y(t) = a_1 x(t) + a_2 x^2(t) + a_3 x^3(t)$ с линейным, квадратичным и кубическим членами. Пусть у такого прибора можно менять коэффициенты a_i, $i = 1, 2, 3$ в пределах от нуля до единицы. Таким образом, можно изучать отдельно модели с различными типами нелинейности. На вход устройства будем подавать гармонический сигнал, у которого можно менять амплитуду, круговую частоту и фазу. Пройдя через нелинейное устройство, входной сигнал изменит свои свойства: в случае квадратичной нелинейности в выходном сигнале будем наблюдать постоянную составляющую и вторую гармонику, в случае кубической нелинейности — первую и третью гармоники. Преобразованный сигнал подадим на гармонический осциллятор в качестве вынуждающих колебаний. Если устройство преобразования сигнала линейно, то с помощью гармонического осциллятора можно будет наблюдать явление резонанса. Действительно, если на линейный осциллятор подавать гармоническое возмущение с его собственной частотой при нулевых начальных условиях

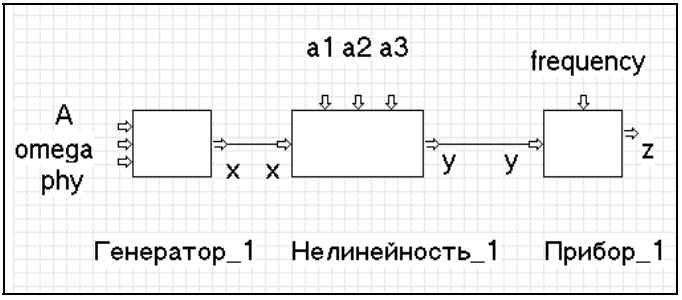
$$\frac{d^2 z}{dt^2} + \omega_0^2 \cdot z = B\sin(\omega_0 t),$$

то мы будем наблюдать колебания

$$z(t) = -\frac{B}{2}(\omega_0 t \cdot \cos(\omega_0 t - \sin(\omega_0 t))$$

с нарастающей амплитудой. На рис. 4.76, *б* представлена функциональная схема такого устройства. Каждый компонент имеет входы, выходы и настраиваемые параметры, реализованные в виде "свободных" входов. Коэффициенты `A`, `omega`, `phy` — это амплитуда, круговая частота и фаза гармонического сигнала, коэффициенты `a1`, `a2`, `a3` — a_i, $i = 1, 2, 3$, `frequency` — собственная частота гармонического осциллятора.

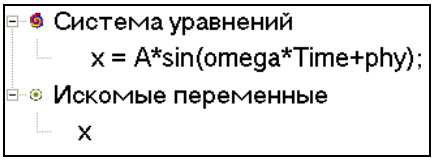
а



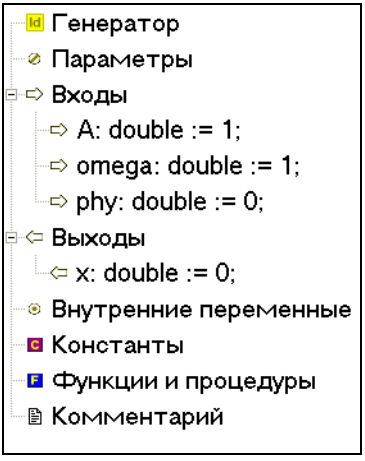
б

Рис. 4.76. Менеджер проекта и функциональная схема.
Тип проекта — "схема": *а* — классы проекта;
б — функциональная схема проектируемого устройства

На рис. 4.76 представлен менеджер проекта с перечислением классов и функциональная схема. На рис. 4.77—4.79 представлены описания переменных классов и решаемые уравнения.



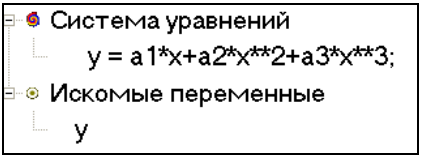
а



б

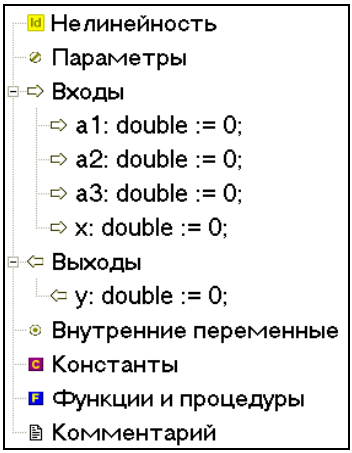
Рис. 4.77. Описание генератора:
а — уравнения генератора; *б* — переменные генератора

Рисунок 4.80 относится уже к испытательному стенду. На испытательном стенде размещено устройство управления коэффициентами модели. В отличие от предыдущих случаев, мы не создавали специального 2D-окна, а воспользовались окном структуры экземпляра класса `Model`.



а

Рис. 4.78. Описание нелинейного устройства преобразования сигнала:
а — уравнения нелинейного устройства

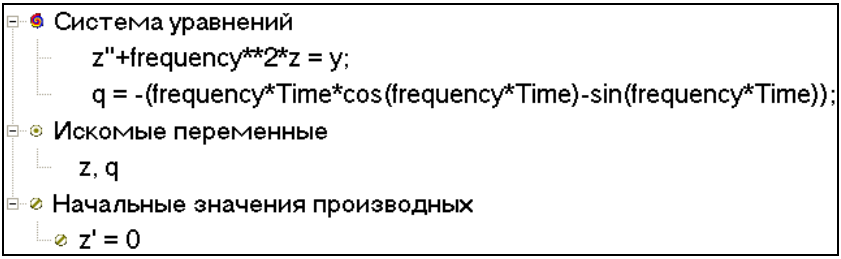


б

Рис. 4.78. Описание нелинейного устройства преобразования сигнала:
б — переменные нелинейного устройства

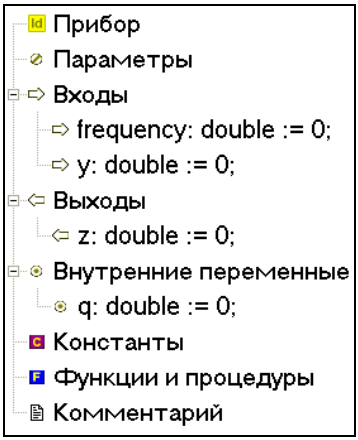
К каждому свободному входу компонентов были присоединены активные "ползунки". Такой способ присоединения активных элементов к свободным входам позволяет менять их значения во время выполнения модели. Если бы коэффициенты были реализованы в виде параметров, их можно было менять только непосредственно перед началом каждого нового эксперимента.

Используем наш прибор для проведения экспериментов. Коэффициенты модели подберем так, чтобы квадратичный и кубический члены в устройстве преобразования сигнала отсутствовали.



а

Рис. 4.79. Описание прибора для изучения явления резонанса:
а — уравнения модели прибора — гармонического осциллятора



б

Рис. 4.79. Описание прибора для изучения явления резонанса:
б — переменные модели прибора

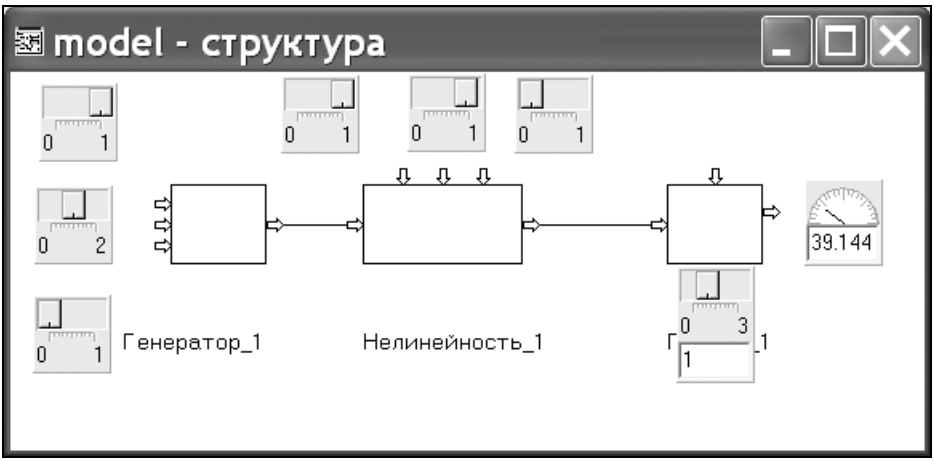


Рис. 4.80. Стенд для управления параметрами модели
по ходу эксперимента

Подадим синусоидальный входной сигнал с частотой, совпадающей с собственной частотой осциллятора. На рис. 4.81 приведены результаты эксперимента: на временной диаграмме слева — входной сигнал, в середине — выходной сигнал созданного прибора, справа — тестовое решение, решение уравнения гармонического осциллятора с вынуждающим гармоническим

входным воздействием, частота которого совпадает с собственной частотой осциллятора. Хорошо видно возникновение резонанса — амплитуда выходного сигнала увеличивается с увеличением времени.

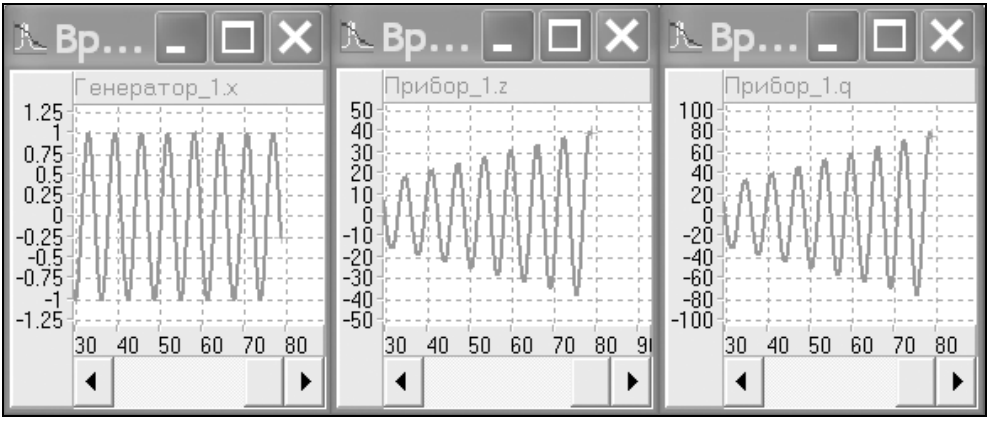


Рис. 4.81. Входной сигнал, выходной сигнал гармонического осциллятора, реализованного численно, точное значение выходного сигнала для гармонического осциллятора в условиях резонанса

Упражнения

1. Изучите поведение гармонического осциллятора с трением, пропорциональным скорости. Для этого достаточно модифицировать класс `Прибор` последнего примера.

2. Создайте испытательный стенд для одновременного изучения осциллятора с трением, и без трения.

   Указания. Сохраните старый проект под новым именем. Создайте еще один компонент — осциллятор с трением. Проще всего это сделать так: с помощью команды **Экспорт в MVL** создайте текстовое представление модели. Скопируйте описание класса `Прибор` и вставьте в текст с новым именем `Прибор_1`. С помощью команды **Импорт из MVL** преобразуйте синглетный класс `Model` в обычный класс. Создайте в `Model` новую функциональную схему, соответствующую устройству с осциллятором с затуханием. Преобразуйте класс `Model` в еще один обычный класс. Создайте в `Model` карту поведения для выбора варианта и назначения параметров.

Теперь естественно обратиться к осцилляторам с нелинейностями. Мы уже рассматривали осциллятор вида

$$\frac{d^2x}{dt^2}+\sin(x)=0.$$

Интересно посмотреть на более сложные осцилляторы

$$\frac{d^2x}{dt^2}+x+x^2=0 \text{ и } \frac{d^2x}{dt^2}+x+x^3=0.$$

Будем называть их осцилляторами с квадратичной и кубической нелинейностями. То, что это осцилляторы, сомнений нет.

У первого две особые точки: $x=0$ и $x=-1$. Потенциальная энергия осциллятора равна $V(x)=\frac{1}{2}x^2+\frac{1}{3}x^3$. В малой окрестности первой точки имеем близкие к гармоническим колебания — эта точка устойчива и имеет тип "центр". Об этом говорит и анализ линеаризованного уравнения, и анализ потенциальной энергии в окрестности нуля. Потенциал имеет в окрестности нуля локальный минимум. Вторая точка неустойчива. Это опять же подтверждает вид потенциальной энергии — в окрестности точки –1 имеется локальный максимум. Фазовая плоскость разделена на три области — область, где происходят колебания, и две области с незамкнутыми траекториями. Этот осциллятор как тестовый пример хорош тем, что можно найти решение уравнения в явном виде.

Осциллятор с кубической нелинейностью называется также уравнением Дюффинга. Потенциальная энергия такого осциллятора имеет вид $V(x)=\frac{1}{2}x^2+\frac{1}{2}x^4$. У такого осциллятора одна устойчивая точка в нуле, и в ее окрестности будут наблюдаться почти гармонические колебания. Можно попробовать поменять знаки коэффициентов уравнения и посмотреть, как изменятся особые точки и какого типа они будут.

Осциллятор $\frac{d^2x}{dt^2}+x-x^3=0$ уже будет иметь три особых очки — в нуле устойчивая точка типа "центр" и две неустойчивые ("седло") в +1 и –1.

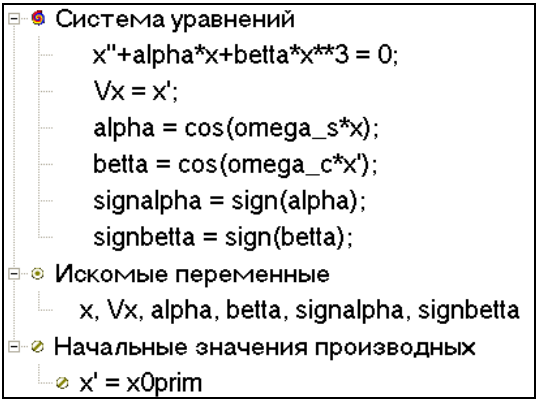
Осциллятор $\frac{d^2x}{dt^2}-x+x^3=0$ тоже имеет три особых очки: в окрестности нуля неустойчивая точка типа "седло", и две устойчивые ("центр") в +1 и –1.

Исследуем эти осцилляторы с помощью MvStudium. Уравнение Дюффинга запишем в форме

$$\frac{d^2x}{dt^2}+\alpha\cdot x+\beta\cdot x^3=0,$$

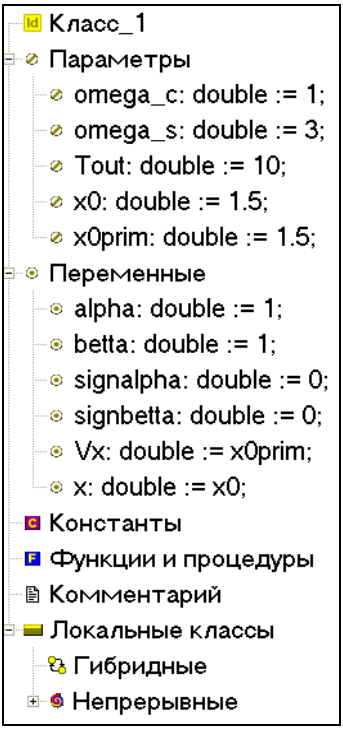
где α, β — произвольные вещественные коэффициенты. Напишем программу, позволяющую строить фазовые портреты. Ее мы будем создавать в два этапа.

На первом этапе построим программу, демонстрирующую различные виды поведения, в зависимости от знаков коэффициентов α, β. На рис. 4.82, *а* показана система решаемых уравнений, у которой коэффициенты α, β зависят от x и $\frac{dx}{dt}$. Зависимости коэффициентов от x и $\frac{dx}{dt}$ выбраны так, чтобы знаки коэффициентов либо периодически менялись, либо оставались неизменными. Как мы видели при рассмотрении различных модификаций уравнения Дюффинга, от знаков коэффициентов меняется число и тип особых точек. Смена различных типов качественного поведения уравнения в зависимости от значений коэффициентов называется *бифуркацией*. Таким образом, уравнение Дюффинга с переменными коэффициентами, меняющими свой знак, должно демонстрировать явление бифуркации.

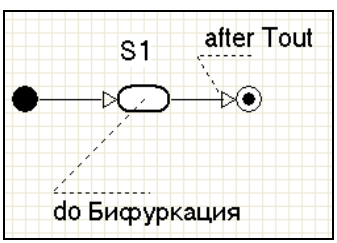


а

Рис. 4.82. Описание класса `Класс_1` для решения уравнения Дюффинга с переменными коэффициентами: *а* — уравнение Дюффинга с переменными коэффициентами



б

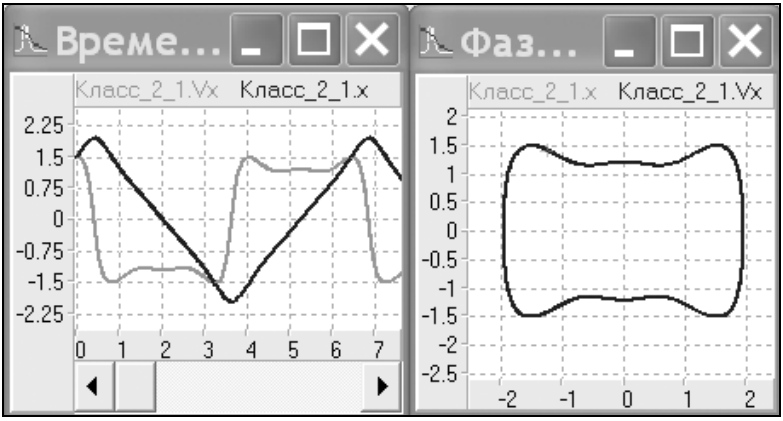


в

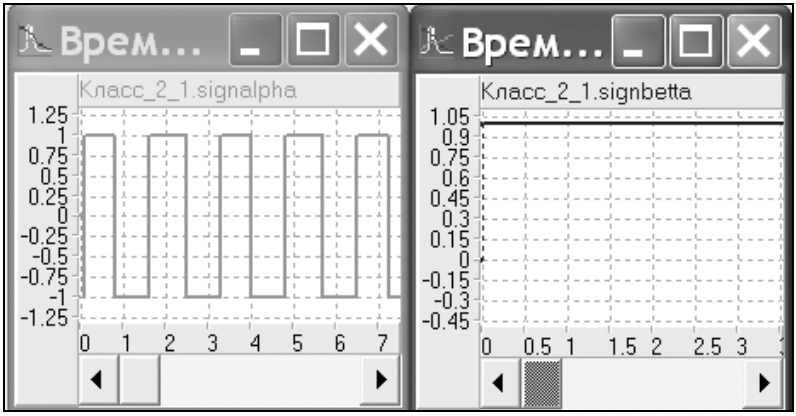
Рис. 4.82. Описание класса `Класс_1` для решения уравнения Дюффинга с переменными коэффициентами: *б* — карта поведения; *в* — описание переменных

На рис. 4.82, *б* и *в* показаны карта поведения и переменные класса `Класс_1`, необходимого нам для построения численного решения уравнения Дюффинга с переменными коэффициентами.

На рис. 4.83—4.85 представлены численные решения и фазовые портреты (кривые *а*)) и знаки коэффициентов (кривые *б*)) для различных значений параметров `omega_c` и `omega_s`.



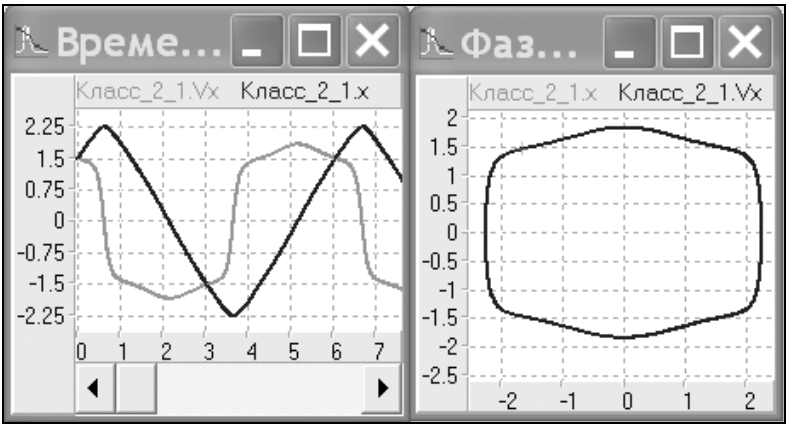
а



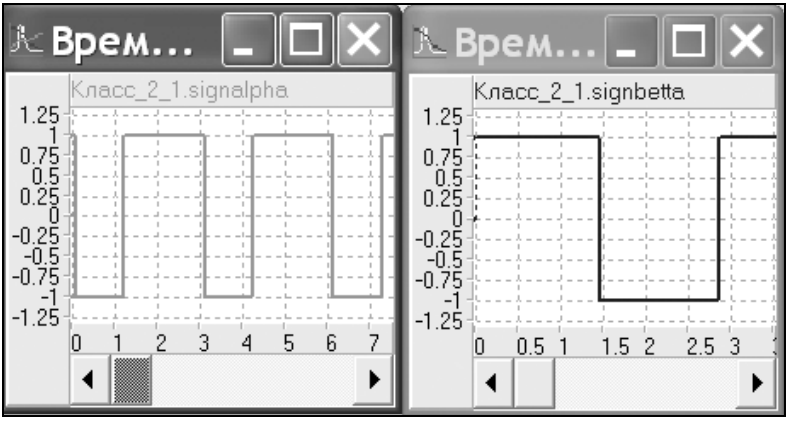
б

Рис. 4.83. Изменение знака α при положительном β: *а* — решение и фазовый портрет; *б* — коэффициент α меняет знак, β положителен

Перейдем теперь к построению фазового портрета. Наметим только решение задачи. На рис. 4.86 показаны описание класса `Model` и его карта поведения.



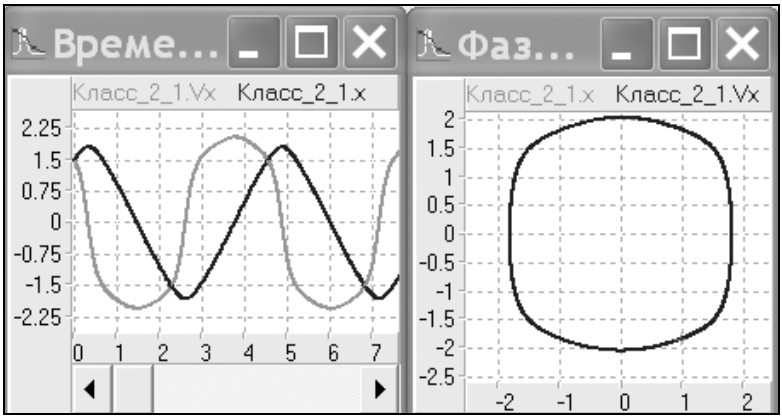
а



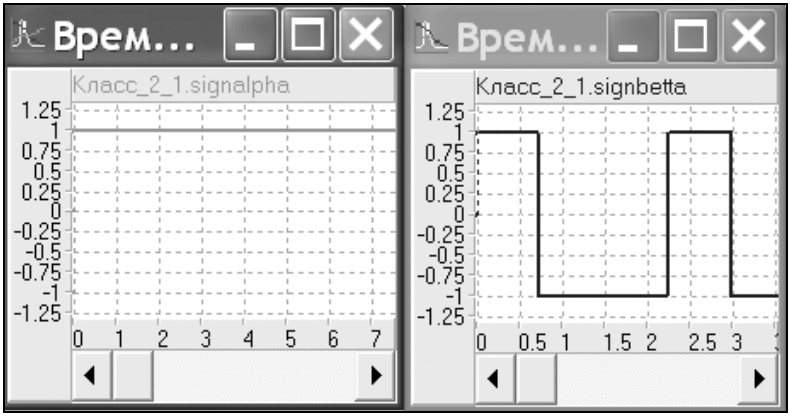
б

Рис. 4.84. Сочетание знаков α и β подобрано так, что присутствуют все три качественно различных поведения: *а* — решение и фазовый портрет; *б* — коэффициенты α и β меняют знак

На рис. 4.87 показано описание класса `Класс_2`. Сравнение описания классов `Класс_2` и `Model` показывает, что описание коэффициентов `alpha`, `betta`, а также начальных условий дублируется. И это не случайно. Первоначально `Класс_2` разрабатывался и проверялся независимо.



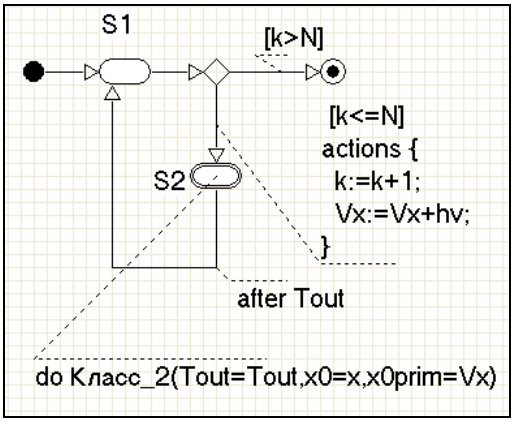
а



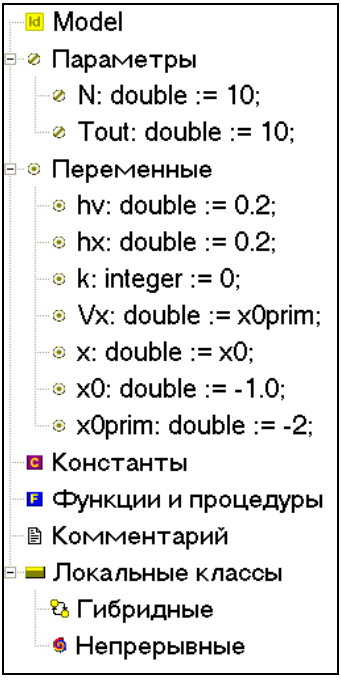
б

Рис. 4.85. Изменение знака β при положительном значении α:
а — решение и фазовый портрет; б — коэффициент α положителен, β меняет знак

При появлении класса `Model` в окончательном виде управление изменением начальных условий было передано ему. Чтобы использовались его переменные при проведении эксперимента, мы воспользовались командой **Установить новые значения параметров и переменных**, открыв правой кнопкой мыши диалог редактирования состояния в карте поведения. Об этом говорит запись `do Класс_2`, связанная с состоянием `S2` (рис 4.86, *б*).

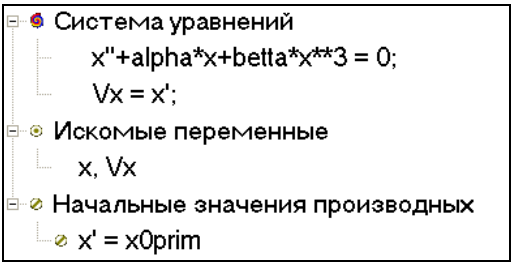


а

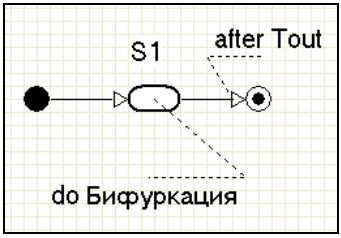


б

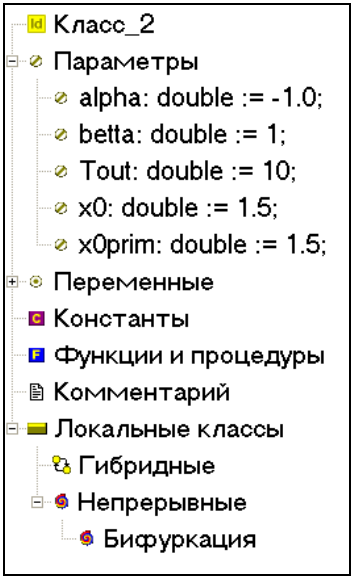
Рис. 4.86. Описание нового эксперимента для построения семейства фазовых портретов: *а* — новое описание класса `Model`; *б* — карта поведения класса `Model`



а



б



в

Рис. 4.87. Описание класса `Класс_2`:
а — новое описание класса `Model`;
б — карта поведения класса `Model`; *в* — описание класса `Класс_2`

На рис 4.88 показан построенный фрагмент семейства фазовых портретов. Как видно и рисунка, на нем не хватает участка, связанного с точкой типа "центр" в точке +1. Но мы и обещали только наметить путь решения.

В примере, которой можно найти на прилагаемом к книге компакт-диске, вы обнаружите один проект, с помощью которого строится только фазовый портрет.

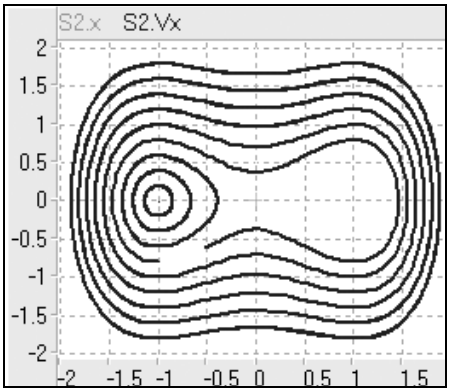


Рис. 4.88. Фрагмент семейства фазовых портретов

Спрашивается, как же проведены два независимых эксперимента — решение уравнения Дюффинга с различными коэффициентами и построение фазового портрета? Очень просто. Для того чтобы реализовать первый эксперимент, был создан проект, который первоначально содержал только `Класс_2`. Он позволял менять коэффициенты и строить только одну фазовую кривую. Когда программа была отлажена, класс `Model` был преобразован в класс, имеющий внутреннюю структуру. В новой модели была реализована главная карта поведения, позволяющая строить семейство фазовых кривых, используя `Класс_2`, приписанный состоянию `S2`. Если следить только за переменными класса `Класс_2`, то можно построить фазовый портрет. В то же время в окне структуры находится экземпляр класса `Класс_1`. Главная карта поведения и экземпляр класса `Класс_1` работают параллельно. Если следить только за переменными класса `Класс_1`, то можно получить любое нужное нам решение уравнения или единственную фазовую кривую. Можно было бы на испытательном стенде создать окна для визуализации первого эксперимента и окна для визуализации второго эксперимента, и наблюдать за ними одновременно. Однако это неудобно, и в то же время не хочется создавать две независимые программы. Выход из положения такой. Нужные опыты проводятся одновременно, но можно создать испытательный стенд с окнами только для первого эксперимента и испытательный стенд с окнами только для второго. Для этого были созданы два отдельных файла с суффиксом `_em` и расширением ini, в которых запоминалась конфигурация окон для каждого эксперимента. При загрузке нужного файла после компиляции программы создается иллюзия, что программа реализует только один эксперимент.

Упражнения

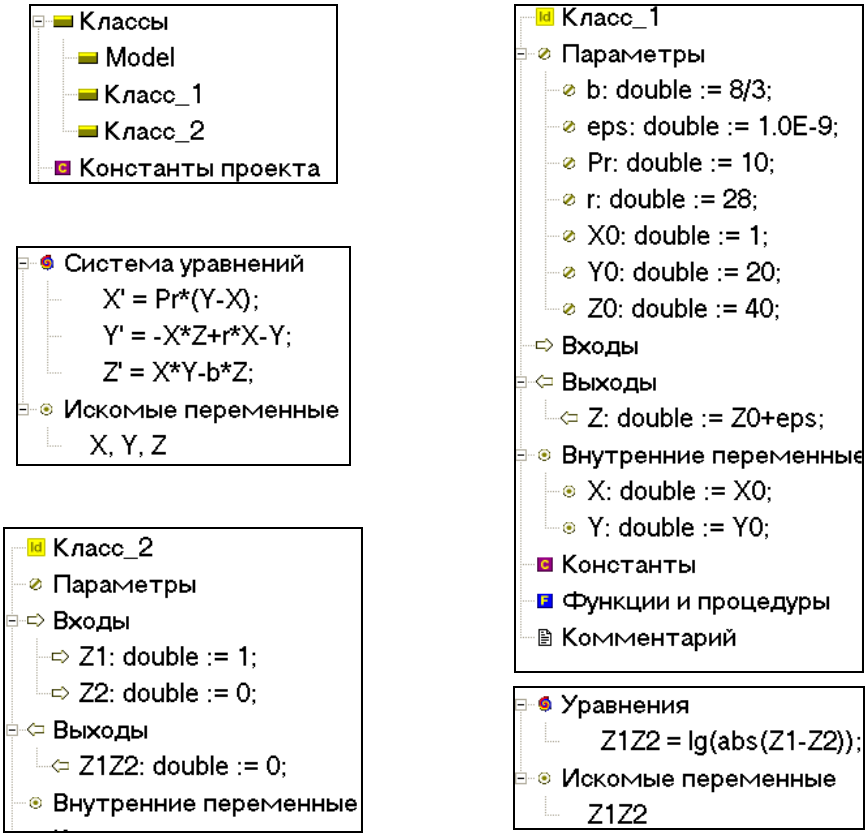
1. Постройте полное семейство фазовых портретов для всех вариантов особых точек.
2. Постройте аналогичную программу для осциллятора с квадратичной нелинейностью.

Странные аттракторы. Отображение Пуанкаре

До сих пор мы имели дело с достаточно простыми притягивающими множествами (аттракторами): либо с точками, либо предельными циклами. Эти аттракторы характерны для систем второго порядка. Для их исследования вполне подходили фазовые портреты. В трехмерном мире появляется новый тип аттрактора (см. уравнения класса `Класс_1` на рис. 4.89 и графики на рис. 4.90) — фазовые кривые не стремятся ни к точке, ни к замкнутой траектории. И близкие в начальный момент фазовые кривые уже не становятся неразличимыми на бесконечности. Эти аттракторы были названы странными. Для странного аттрактора картина другая — близкие по начальным условиям кривые расходятся с течением временем, хотя на первый взгляд они выглядят очень похожими (см. фазовые портреты на плоскости (x, y) на рис. 4.90). Таким образом, кривые не выходят за пределы некоторой области, но и регулярности в их поведении не наблюдается.

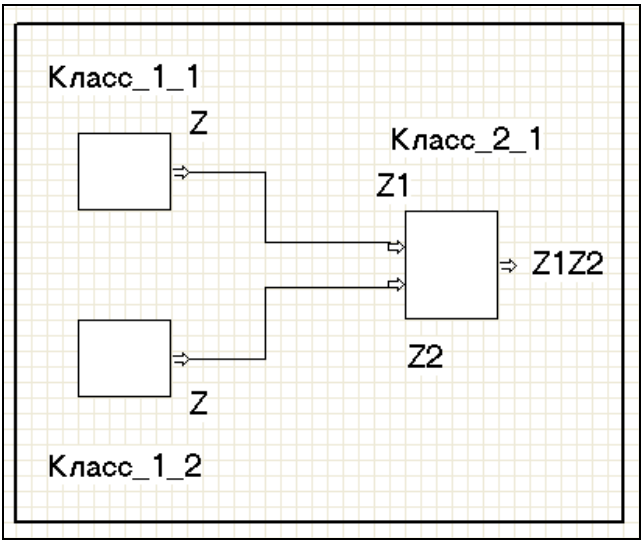
Возможно, вам придется работать со странными аттракторами. Для их исследования нужны другие инструменты. Другие инструменты могут понадобиться и при исследовании бифуркации. Рассмотрим *дискретное отображение* $x_{n+1} = f(x_n)$, $n = 0, 1...$ или просто *отображение*. В качестве примера выберем логистическое отображение $x_{n+1} = \lambda x_n - x_n^2$. Это уравнение может рассматриваться как модель численности популяции: первый член отвечает за рост популяции, а второй — за падение численности. Здравый смысл подсказывает, что если популяция выживает, то ее численность со временем либо стабилизируется на некотором уровне, либо демонстрирует периодическое изменение численности. Первый случай соответствует тому, что существует неподвижная точка $x^* = f(x^*)$, а второй — тому, что существует такое целое $k \geq 2$, что $x^{n+k} = x^n$. А может ли случиться так, что численность будет оста-

ваться ограниченной, но меняться будет хаотически? Оказывается, что так может быть, и даже эта простая модель показывает, что это возможно. Будем медленно менять параметр λ и наблюдать за установившимся движением. Начальный отрезок бифуркационной диаграммы показывает (рис. 4.92), что сначала численность популяции стремится к некоторому постоянному значению, однако после $\lambda > 3$ таких стационарных значений два, затем их становится четыре. Что это означает? Просто численность популяции начинает сначала меняться как последовательность с периодом два — x_1, x_2, x_1, x_2, затем последовательность с периодом четыре и т. д (рис. 4.93).



а

Рис. 4.89. Описание программы для проведения эксперимента с аттрактором Лоренца: *а* — классы программы



б

Рис. 4.89. Описание программы для проведения эксперимента с аттрактором Лоренца: *б* — функциональная схема

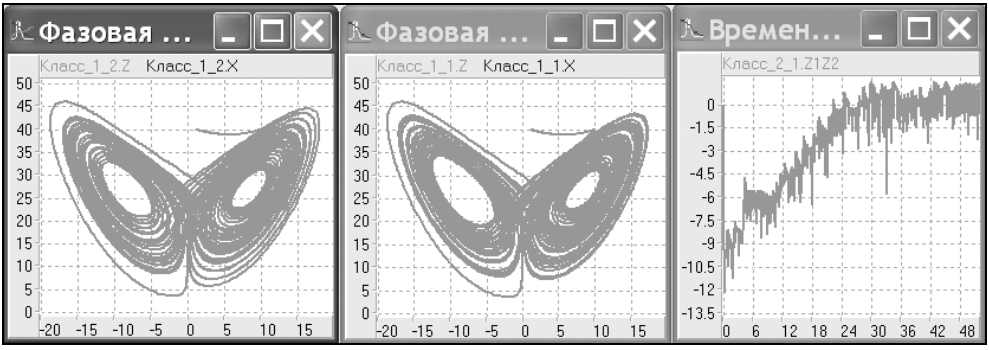
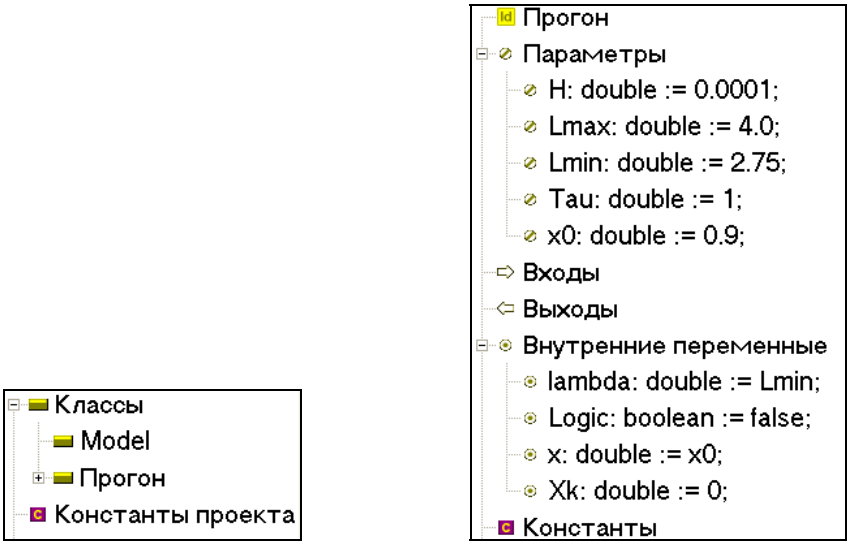


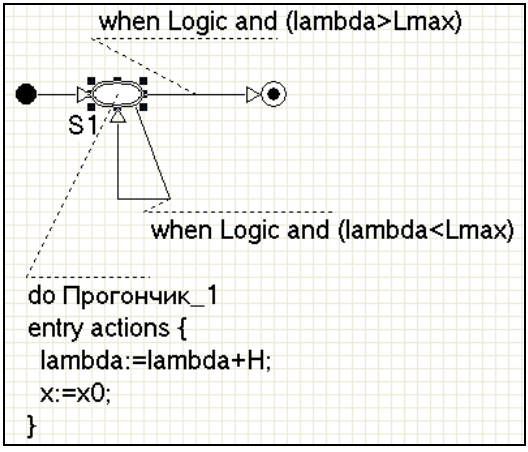
Рис. 4.90. Результаты эксперимента: две кривые с близкими начальными условиями и их разность во времени (на крайнем правом графике по оси ординат выбран логарифмический масштаб)

При больших и очень близких λ замеренные численности популяции в фиксированный момент времени будут заметно отличаться, и можно сказать, что численность будет практически случайным числом. На рис. 4.91 показана программа, с помощью которой осуществлялся эксперимент.

Как дискретное отображение может быть связано с дифференциальными уравнениями? Как можно использовать его для изучения свойств дифференциальных уравнений?

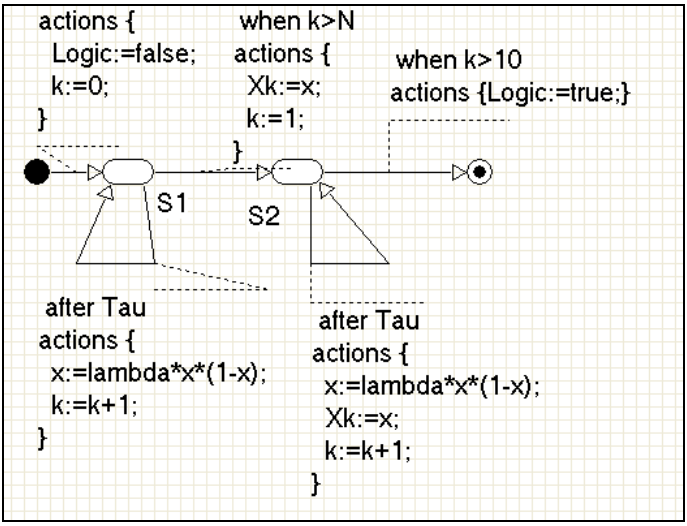


а *б*



в

Рис. 4.91. Программа для экспериментов с логистическим отображением: *а* — описание проекта; *б* — класс `Прогон`; *в* — главная карта поведения



г

Рис. 4.91. Программа для экспериментов с логистическим отображением: *г* — вложенная карта поведения

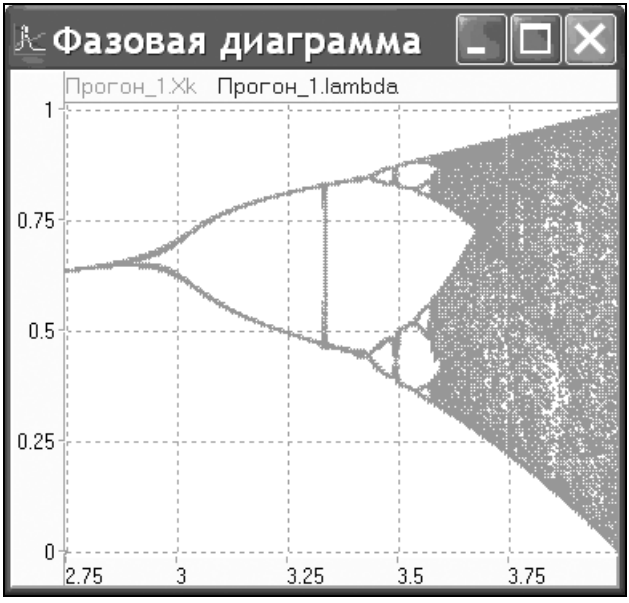


Рис. 4.92. Бифуркационная диаграмма для логистического отображения

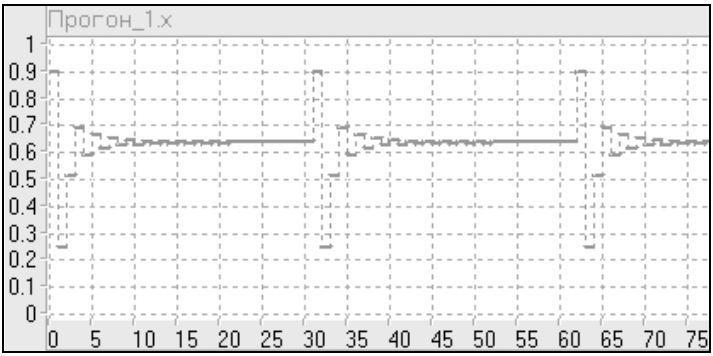
Вернемся в системе с особой точкой типа "центр" в начале координат для уравнений на плоскости. Фазовый портрет такой системы — концентрические окружности. Рассмотрим любую полупрямую с началом в центре координат. Такая полупрямая обязательно будет пересекаться с фазовыми кривыми, окружностями. Выберем любую фазовую кривую. Очевидно, что ее пересечение с полупрямой будет происходить периодически, в одной и той же точке фазовой плоскости. Эта полупрямая и есть сечение Пуанкаре. Для особой точки типа "центр" точка пересечения с заданной фазовой кривой "стоит" на месте. Если теперь рассмотреть движение к предельному циклу, расположенному вокруг начала координат, и следить за точкой пересечения с аналогичной полупрямой, то она будет дискретно перемещаться по ней по закону $x_{n+1} = f(x_n)$, $n = 0, 1...$ Функция f будет задана неявно и определяться дифференциальным уравнением. Стремление решения к предельному циклу, например, будет означать, что функция имеет неподвижную точку. Таким образом, с помощью сечений Пуанкаре можно изучать свойства решений систем дифференциальных уравнений.

Частным случаем отображения Пуанкаре является стробоскопическое отображение. В этом случае непрерывное решение дифференциального уравнения превращают в дискретное отображение, следя за положением вектора решения $x(t)$ через определенные промежутки времени T, т. е. строят последовательность $x(t_0)$, $x(t_0 + T)$, $x(t_0 + 2T)$, ... Опять же для изохронных колебаний, если T совпадает с периодом колебаний, то на фазовом портрете, где откладываются пары $(x(t_0), \dot{x}(t_0))$, $(x(t_0 + T), \dot{x}(t_0 + T))$, $(x(t_0 + 2T), \dot{x}(t_0 + 2T))$, вместо привычных для этих колебаний окружностей мы увидим точки, расположенные на отрезке прямой, проходящей через центр.

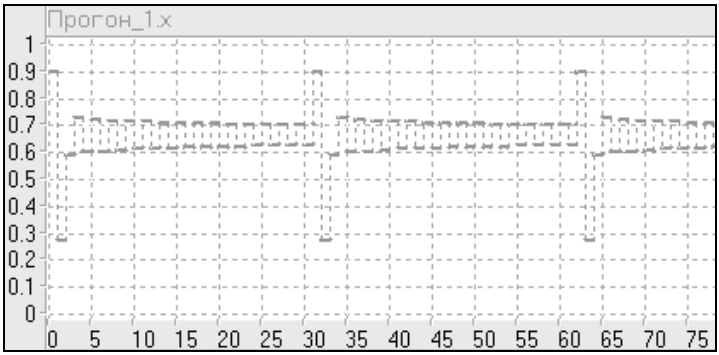
Посмотрим, как использовать отображение Пуанкаре для исследования уравнения Дюффинга. Используем [18] как источник тестовых примеров. Для уравнения Дюффинга

$$\frac{d^2 s}{dt^2} + \alpha \cdot s + \beta \cdot s^3 = \varepsilon \cdot \sin(\nu \cdot t)$$

в книге приводится много конкретных отображений Пуанкаре, при значениях параметров $\alpha = \pm 1$; $\beta = \pm 1$; $\nu = 4$; $\varepsilon = 1.6$. Создадим программу для построения отображения Пуанкаре в MvStudium (листинг 4.13).



а



б

Рис. 4.93. Бифуркация по сценарию удвоение периода:
а — выход на стационарное значение при $\lambda < 3$;
б — колебания периода 2 при $\lambda > 3$

Листинг 4.13. Построение отображения Пуанкаре

```
package О_П is
-- О_П
  import SysLib;


  private hybrid class Model is
    parameter T: double := 11;
```

```
    parameter N: integer := 100;
    parameter M: double := 20;
    parameter hs: double := 0.4;
    parameter hvs: double := 4;
    s: double := s0;
    Vs: double := Vs0;
    x: double := 0;
    y: double := 0;
    k: double := 0;
    s0: double := -4.0;
    Vs0: double := -24;
    EndLoop: boolean := false;
    p: integer := 0;
    local continuous class Внутренний_1 is
      equations {
        s'' = -sin(s);
        Vs = s';
        unknown s, Vs;
      };
      initial {
        s' = Vs;
      };
    end Внутренний_1;
    local hybrid class Внешний_1 is
      bchart {
        initial state _InitState;
        final state _FinalState;
        transition _InitTran from _InitState to S1;
after T
          actions {
            k:=k+1;
            x:=s;
            y:=Vs;
          }
        ;
        transition Tr_2 from _Switchpoint_17760258 to _FinalState [k>N]
          actions {EndLoop:=true;}
        ;
```

```
      transition Tr_3 from _Switchpoint_17760258 to S1 [else];
    };
  end Внешний_1;
  local continuous class Внутренний_2 is
    parameter eps: double := 1;
    parameter vu: double := 4;
    parameter betta: double := 1;
    parameter alpha: double := 1;
    parameter PrT: double := 0;
    equations {
      s'' = -alpha*s-betta*s**3+eps*sin(vu*(PrT+Time));
      Vs = s';
      unknown s, Vs;
    };
    initial {
      s' = Vs;
    };
  end Внутренний_2;
  bchart {
    initial state _InitState;
    state S1 do Внешний_1
      entry actions {EndLoop:=false;}
    ;
    final state _FinalState;
    switch state _Switchpoint_8257538;
    transition _InitTran from _InitState to S1;
      pragma transition(_InitTran) line [(65,90)];
    transition Tr_1 from S1 to _Switchpoint_8257538 when EndLoop;
    transition Tr_2 from _Switchpoint_8257538 to _FinalState [p>M];
    transition Tr_3 from _Switchpoint_8257538 to S1 [else]
      actions {
        s0:=s0+hs;
        Vs0:=Vs0+hvs;
        s:=s0;
        Vs:=Vs0;
        k:=0;
        p:=p+1;
      }
```

```
      ;
    };
  end Model;

  static model: Model;

end О_П;
```

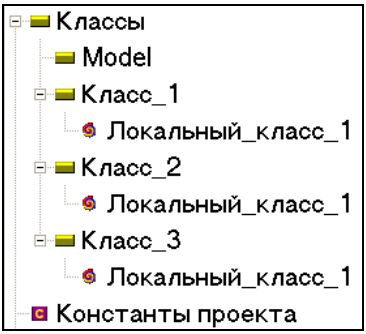
Упражнение

Разберите самостоятельно, как построена программа. Постройте более удобный испытательный стенд для изменения параметров. Сравните фазовые портреты и отображения Пуанкаре для одних и тех же уравнений. Посмотрите, как меняются отображения при изменении параметра т.

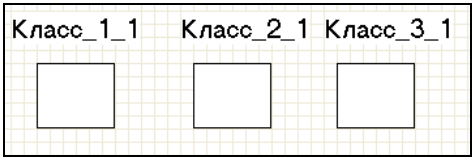
Уравнения, не приведенные к форме Коши

Как мы уже постоянно отмечали, одной из важнейших задач пакетов моделирования является автоматическое преобразование систем уравнений из формы, удобной пользователю, к форме, удобной программным реализациям численных методов.

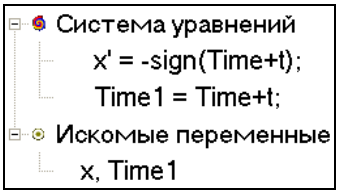
Напомним, что до недавнего времени пользователю предлагалось самому преобразовывать полученные уравнения модели в форму Коши — в автономную $\frac{dx}{dt} = f(x)$ или неавтономную $\frac{dx}{dt} = f(t,x)$ систему уравнений. При этом предполагалось, что функция $f(x)$ обеспечивает существование и единственность решения. К чему приводит несоблюдение этого требования легко убедиться на примере. Уравнение $\frac{dx}{dt} = \sqrt{|x|}$ не имеет единственного решения на промежутке $[-1,1]$, т. к. производная правой части в нуле терпит разрыв. Программы, которые проверку существования и единственности решения перекладывают на пользователя, спокойно берутся решать такие задачи, как и задачи с разрывами функции $\frac{dx}{dt} = \mathrm{sign}(x)$. На рис. 4.94 и 4.95 приведен набор тестовых задач и полученные численные решения. Во всех случаях мы выходим за рамки классических динамических систем, а пакет никак на это не реагирует. Анализ таких систем можно проводить предварительно и предупреждать пользователя о возможных неприятностях.



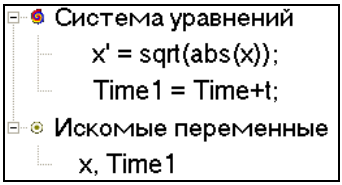
а



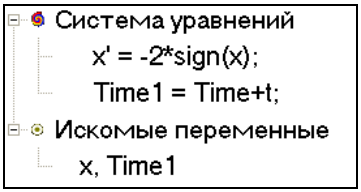
б



в



г

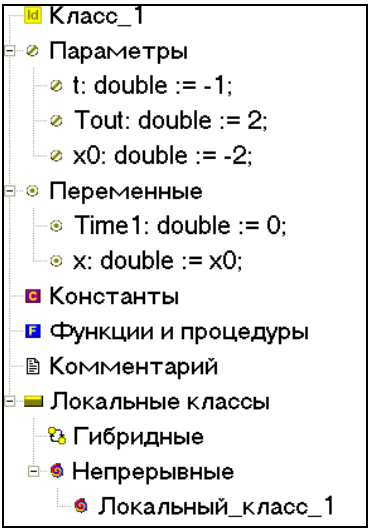


д

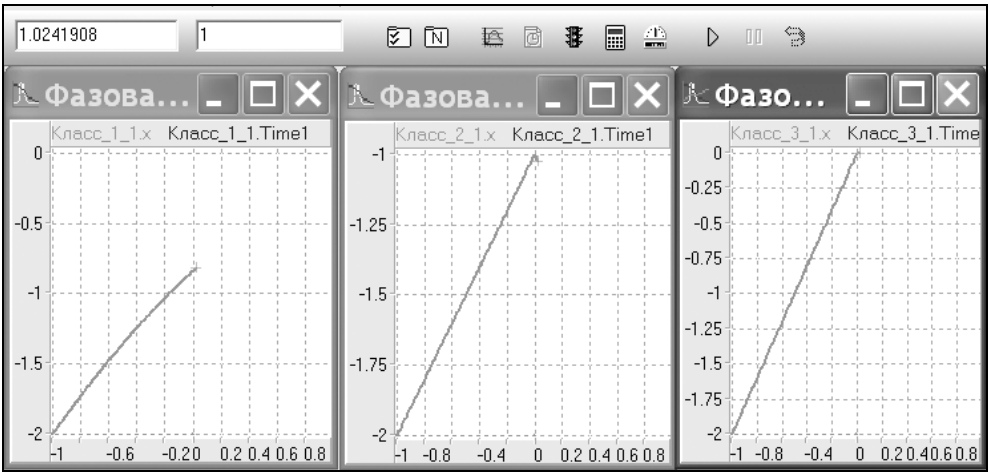
Рис. 4.94. Описание проекта и примеры простейших разрывных правых частей, не контролируемых численными методами: *а* — менеджер проекта; *б* — структурная диаграмма класса `Model`; *в* — разрыв правой части; *г* — не существует производная в нуле; *д* — разрыв правой части

На рис. 4.95, *б* приведены две серии графиков. В верхнем ряду приведены графики, построенные с помощью явных методов. Явные методы не заметили ничего особенного и проинтегрировали уравнения, не обращая внимания на разрыв. В нижнем ряду приведены графики, построенные неявными методами. Графики перехвачены в момент, когда численный метод резко уменьшил шаг интегрирования, и создалось впечатление, что метод прекратил ра-

боту. Это должно подсказать пользователю, что происходит нечто неладное, и стоит проанализировать правую часть.



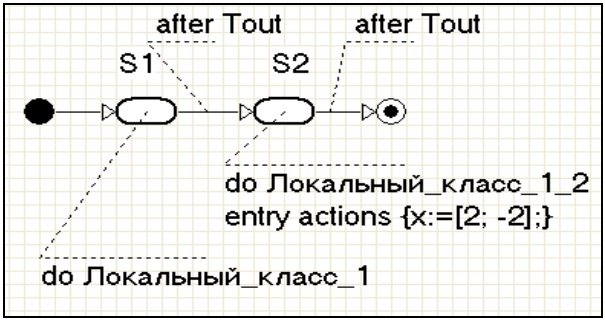
а



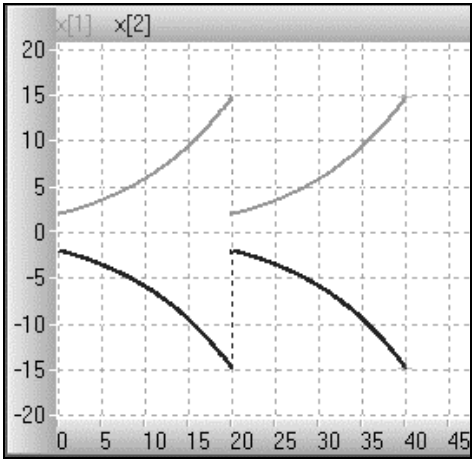
б

Рис. 4.95. Описание класса `Класс_1` и результаты численных экспериментов: *а* — менеджер класса; *б* — численное решение приведенных уравнений

Постепенно, от уравнений, разрешенных относительно производных, стали переходить к более сложным формам. Во многих приложениях стали возникать уравнения вида $\mathbf{A}\cdot\dfrac{dx}{dt}=f(x)$, с квадратной вещественной невырожденной матрицей $\mathbf{A}$, или даже уравнения с матрицей, коэффициенты которой зависят от искомых переменных $\mathbf{A}(x)\cdot\dfrac{dx}{dt}=f(x)$, опять же, с невырожденной на решении матрицей.



а



б

Рис. 4.96. Пример численного эксперимента:
а — карта поведения; *б* — численные решения

Программные реализации подстроились и под эти формы уравнений. Не обсуждая детали реализации, можно считать, что в каждый момент система эквивалентна системе $\frac{dx}{dt} = \mathbf{A}^{-1} \cdot f(x)$ — а это стандартная форма. Конечно же, никто матрицу не обращает, это слишком дорого и "по времени", и "по памяти", главное другое — пользователь может не приводить систему к форме Коши и решать ее в исходном виде.

На рис. 4.96, *а* приведена карта состояния, с помощью которой в пакете MvStudium последовательно решаются две системы

$$\frac{dx}{dt} = \mathbf{A}^{-1} \cdot f(x) \text{ и } \mathbf{A} \cdot \frac{dx}{dt} = f(x),$$

и построенные решения (рис. 4.96, *б*).

Упражнение

Модифицируйте имеющуюся на диске программу так, чтобы можно было сравнить времена, затрачиваемые на решение уравнений в различной форме представления. При проведении этих экспериментов размеры матриц придется существенно увеличить.

Естественно встал вопрос, а если мы имеем систему $F\left(\frac{dx}{dt}, x\right) = 0$ и система однозначно разрешима относительно производной на решении, то можно ли эту форму считать исходной для численного метода? Появились программы, способные решать такие системы. Такие уравнения искусственно превращаются в алгебро-дифференциальные. Можно представить себе, что сначала на каждом шаге решается алгебраическая система, уравнения разрешаются относительно производных, а затем получают численное решение дифференциальных. Предполагается, что при достаточно малом шаге переменные будут меняться незначительно, и трудность, связанная с поиском начального приближения для нелинейных уравнений, будет преодолена.

В листинге 4.14 представлено линейное однородное уравнение в различных формах. Коэффициент выбран так, чтобы решение стремилось к нулю. В листинге 4.15 показаны преобразованные уравнения, которые и решаются с помощью численных методов. Не приступая к решению, попробуйте предугадать, что произойдет на этапе исполнения, все ли уравнения будут решены и какие ответы будут получены?

Листинг 4.14. Различные формы записи линейного уравнения

```
local continuous class Локальный_класс_1 is
      equations {
        -- Вариант 1
        x' = -a*x;
        -- Вариант 2
        a*s+s' = 0;
        -- Вариант 3
        (z')**2-a**2*z**2 = 0;
        -- Вариант 4
        (w'-a*w)*(w'+a*w) = 0;
        -- Вариант 5
        -- ( d( v ) / dt +a*v ) ** 2 = 0
        unknown x, s, z, w;
```

Листинг 4.15. Различные формы записи преобразованного линейного уравнения

```
x: d(x)/dt = x';
s: d(s)/dt = s';
z: d(z)/dt = z';
w: d(w)/dt = w';
x': x' = -a*x;
s': a*s+s' = 0;
z': z'**2-a**2*z**2 = 0;
w': (w'-a*w)*(w'+a*w) = 0;
```

Рассмотренный способ представления уравнений, не разрешенных относительно производных, был предложен Гиром, автором одной из первых успешных программ для решения жестких уравнений. В пакете MvStudium это преобразование выполняется всегда, несмотря на то, что программа DDassl, включенная в пакет, не требует от пользователя такого предварительного преобразования.

Решение алгебро-дифференциальных уравнений

Следуя принципу "пользователь всегда прав", необходимо признать, что алгебро-дифференциальные уравнения — необходимая форма записи моделей,

и программное обеспечение должно справляться с такими задачами. Действительно, если бы форма Коши была единственной узаконенной формой представления дифференциальных уравнений, то даже для простейшей алгебро-дифференциальной системы

$$\begin{cases} \dfrac{dx}{dt} = a \cdot x + c \cdot y + b; \quad x(0) = x_0, \\ \alpha \cdot x + \beta \cdot y = 0; \qquad \beta \neq 0 \end{cases}$$

либо пришлось бы предварительно проделывать следующее преобразование

$$y = -\frac{\alpha}{\beta} \cdot x \;\Rightarrow\; \frac{dy}{dt} = -\frac{\alpha}{\beta} \cdot \frac{dx}{dt} = -\frac{\alpha}{\beta}(a \cdot x + c \cdot y + b); \quad y(0) = -\frac{\alpha}{\beta} \cdot x_0,$$

в результате чего возникает новая система второго порядка

$$\begin{cases} \dfrac{dx}{dt} = a \cdot x + c \cdot y + b; \qquad x(0) = x_0 \\ \dfrac{dy}{dt} = -\dfrac{\alpha}{\beta} \cdot (a \cdot x + c \cdot y + b); \quad y(0) = -\dfrac{\alpha}{\beta} x_0, \end{cases}$$

либо явно разрешить алгебраическое уравнение относительно y и получить уравнение первого порядка с подстановкой

$$\begin{cases} y = -\dfrac{\alpha}{\beta}(a \cdot x + b), \\ \dfrac{dx}{dt} = a \cdot x + c \cdot y + b; \quad x(0) = x_0. \end{cases}$$

В последнем случае ничто не мешает решать алгебраическое уравнение численно: всякий раз, когда вызывается правая часть дифференциального уравнения для построения приближенного значения x, находить y из алгебраического уравнения и подставлять его в правую часть.

Однако есть и радикальное предложение — построить численный метод, который справлялся бы с исходной системой и не требовал ни ручного преобразования, ни отдельного модуля для решения алгебраической системы. Такие методы существуют.

Как и во всех предыдущих случаях, чрезвычайно важно до того как приступать к численному решению задачи узнать, с каким типом уравнения мы имеем дело.

В случае алгебро-дифференциальных уравнений проблема еще больше осложняется, т. к. даже для формы

$$\mathbf{A}\frac{dx}{dt} = f(x)$$

трудно сказать, что это — дифференциальные, дифференциально-алгебраические или алгебраические уравнения, если неизвестно, вырождена или нет матрица **A**.

Преобразование алгебро-дифференциальной системы в чисто дифференциальную имеет важное теоретическое значение.

Уже рассмотренный простейший пример показывает, что полученная система дифференциальных уравнений обладает одной особенностью — начальные условия по y, или алгебраическим переменным, нельзя задавать произвольно. Начальные условия по x и по y должны быть согласованы, т. е. удовлетворять алгебраическому уравнению. В простейшем случае это очевидно, но для системы $\mathbf{A}\frac{dx}{dt} = f(x)$, с вырожденной матрицей, пользователь может задать произвольные начальные условия по всем переменным, не зная, что система алгебро-дифференциальная. Таким образом, перед тем как приступить к численному решению, надо выделить дифференциальные компоненты, алгебраические компоненты и для алгебраических найти согласованные начальные условия.

В рассмотренном примере преобразование осуществляется за один шаг, это и означает, что система имеет индекс 1.

Система двух алгебро-дифференциальных уравнений вида

$$\begin{cases} \dfrac{dx}{dt} = f(x,y);\ x(0) = x_0, \\ 0 = g(x,y) \end{cases}$$

для приведения к системе дифференциальных уравнений потребует дифференцирования второго уравнения

$$\begin{cases} \dfrac{dx}{dt} = f(x,y); x(0) = x_0, \\ \dfrac{\partial g(x,y)}{\partial x}\dfrac{dx}{dt} + \dfrac{\partial g(x,y)}{\partial y}\dfrac{dy}{dt} = 0, \end{cases}$$

и если $\dfrac{\partial g(x,y)}{\partial y} \neq 0$ на решении, то нужная нам система примет вид

$$\begin{cases} \dfrac{dx}{dt} = f(x,y); & x(0) = x_0, \\ \dfrac{dy}{dt} = -\left(\dfrac{\partial g(x,y)}{\partial y}\right)^{-1} \cdot \dfrac{\partial g(x,y)}{\partial x}\dfrac{dx}{dt}; & 0 = g(x_0, y_0). \end{cases}$$

Может случиться так, что $\dfrac{\partial g(x,y)}{\partial y} \equiv 0$, и тогда придется дифференцировать дважды или большее число раз. Если двух шагов достаточно, чтобы получить систему дифференциальных уравнений, то говорят, что система имеет индекс 2 и т. д. Таким образом, индекс показывает, сколько шагов надо сделать для построения эквивалентной системы дифференциальных уравнений и сколько при этом можно будет задавать независимых начальных условий. При автоматическом формировании уравнений и соответствующую проверку следует проводить автоматически.

Наиболее просто дело обстоит с системами линейных уравнений вида

$$\mathbf{A} \cdot \frac{d\mathbf{x}}{dt} = \mathbf{B} \cdot \mathbf{x} + \mathbf{f}(t),$$
$$\mathbf{A}, \mathbf{B} \in \Re^{m \times n}.$$

Как известно [19], эту систему специальными преобразованиями можно привести к каноническому виду, т. е. к совокупности простых форм. Различных "простых" форм уравнений всего несколько.

Не будем рассматривать случай, когда матрицы $\mathbf{A}$, $\mathbf{B}$ квадратные и матрица $\mathbf{A}$ обратима, здесь преобразование очевидно:

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}^{-1} \cdot \mathbf{B} \cdot \mathbf{x} + \mathbf{A}^{-1} \cdot \mathbf{f}(t).$$

В этом случае исходная система — это обычная система линейных дифференциальных уравнений, хоть и не разрешенная относительно производных. Неявные методы справляются с такой системой без всяких преобразований, а явные легко модифицировать, чтобы не обращать матрицу, если она больших размеров и разреженная. В этом случае достаточно воспользоваться тем, что решение линейных систем осуществляется в два этапа, сначала и единожды вычисляются множители в соответствующем разложении, а затем на каждом шаге находится решение для конкретного вектора правой части.

Дальнейшие случаи рассмотрим на примерах систем небольшой размерности, записанных в координатной форме.

Примером чисто алгебраических уравнений является система

$$\begin{cases} \dfrac{dz_2}{dt} = -z_1 + \tilde{f}_1, \\ z_2 = \tilde{f}_2, \end{cases}$$

имеющая решение

$$\begin{cases} z_2 = \tilde{f}_2, \\ z_1 = \tilde{f}_1 - \dfrac{d\tilde{f}_2}{dt}. \end{cases}$$

Решение не зависит от начальных условий! Вдобавок, для того чтобы это решение построить, необходимо знать заранее или вычислять численно производную $\dfrac{d\tilde{f}_2}{dt}$.

Третью форму мы покажем на примере системы из трех уравнений:

$$\begin{cases} \dfrac{dz_1}{dt} = \tilde{f}_1 \\ \dfrac{dz_2}{dt} + z_1 = \tilde{f}_2 \\ 0 \cdot z_3 + z_2 = \tilde{f}_3 \end{cases} \Rightarrow \begin{cases} z_2 = \tilde{f}_3 \\ z_1 = \tilde{f}_2 - \dfrac{d\tilde{f}_3}{dt} \\ \dfrac{dz_1}{dt} = \tilde{f}_1 \end{cases} \Rightarrow \begin{cases} \tilde{f}_1 = \dfrac{d\tilde{f}_2}{dt} - \dfrac{d^2\tilde{f}_3}{dt^2}, \end{cases}$$

у которой:

- нет ни одной свободной постоянной в решении;
- необходима производная $\dfrac{d\tilde{f}_3}{dt}$;
- решение зависит от произвольной функции z_3;
- появляется дополнительное требование согласованности правых частей $\tilde{f}_1 = \dfrac{d\tilde{f}_2}{dt} - \dfrac{d^2\tilde{f}_3}{dt^2}$.

Столько необычных условий по сравнению с динамическими системами нам еще не встречалось.

И, наконец, возможен случай:

$$\begin{cases} \dfrac{dz_1}{dt} + z_2 = \tilde{f}_1 \\ 0 \cdot z_2 = \tilde{f}_2 \end{cases} \Rightarrow \begin{cases} z_2 = z_2(t) \\ \dfrac{dz_1}{dt} = -z_2 + \tilde{f}_1 \\ \tilde{f}_2 \equiv 0 \end{cases}.$$

Здесь решение z_1 зависит от произвольной функции $z_2(t)$, произвольного начального условия z_{10}, а решение системы существует, только если $\tilde{f}_2 \equiv 0$.

Простота систем и очевидность условий существования единственного решения в приведенных системах не должны никого вводить в заблуждение — как только мы столкнемся с практическими задачами, размерности систем возрастут, и увидеть, с каким случаем или, точнее, с комбинацией каких случаев мы имеем дело, станет сложно. Более того, мы должны уметь это делать автоматически.

В случае нелинейных уравнений дело обстоит много сложнее. Формы алгебро-дифференциальных уравнений, с которыми может работать программное обеспечение для решения обыкновенных дифференциальных уравнений, представлены:

- уравнениями вида

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{y}, t), \\ 0 = \mathbf{g}(\mathbf{x}, \mathbf{y}, t), \end{cases}$$

причем неявно предполагается, что в любой точке t можно однозначно найти $y = g_1(x, t)$;

- уравнениями

$$\mathbf{A} \cdot \frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t)$$

с постоянной матрицей $\mathbf{A}$, или матрицей, коэффициенты которой зависят от искомых переменных. Матрица может быть вырожденной.

Решение уравнений в первой форме с дифференциальным и алгебраическим компонентом можно найти, используя стандартное программное обеспечение для решения обыкновенных дифференциальных уравнений, разрешая алгебраические уравнения каждый раз, когда это нужно, численно. Например, для этой цели можно использовать метод Ньютона. Если

решение меняется достаточно медленно, то можно надеяться, что если алгебраическое уравнение будет решаться, то оно будет решаться быстро. Дифференциальный компонент $\mathbf{x}$ для построения решения требует задания начальных условий $\mathbf{x}_0$, а алгебраический — знания согласованных начальных условий $\mathbf{y}_0 = \mathbf{g}_1(\mathbf{x}_0, t_0) \Leftrightarrow \mathbf{g}(\mathbf{y}_0, \mathbf{x}_0, t_0) = 0$, поиск которых численно может вызывать большие трудности, если хорошие начальные приближения к ним заранее неизвестны.

Вторая форма уравнений — уравнений, не разрешенных относительно производных, — интересна тем, что такие системы уравнений можно пытаться решать неявными методами решения систем обыкновенных дифференциальных уравнений, без каких-либо предварительных преобразований. Рассмотрим неявный метод Эйлера применительно к системе

$$\mathbf{A} \cdot \frac{d\mathbf{x}}{dt} = \mathbf{B} \cdot \mathbf{x} \Rightarrow \mathbf{A} \cdot (\mathbf{z}_{n+1} - \mathbf{z}_n) = h_{n+1} \cdot \mathbf{B} \cdot \mathbf{z}_{n+1} \Rightarrow (\mathbf{A} + h_{n+1} \cdot \mathbf{B}) \cdot \mathbf{z}_{n+1} = \mathbf{A} \cdot \mathbf{z}_n .$$

Даже если матрица $\mathbf{A}$ вырождена, можно пытаться подбирать шаг так, чтобы система $(\mathbf{A} + h_{n+1} \cdot \mathbf{B}) \cdot \mathbf{z}_{n+1} = \mathbf{A} \cdot \mathbf{z}_n$ имела бы единственное решение.

К числу допустимых форм можно отнести и систему

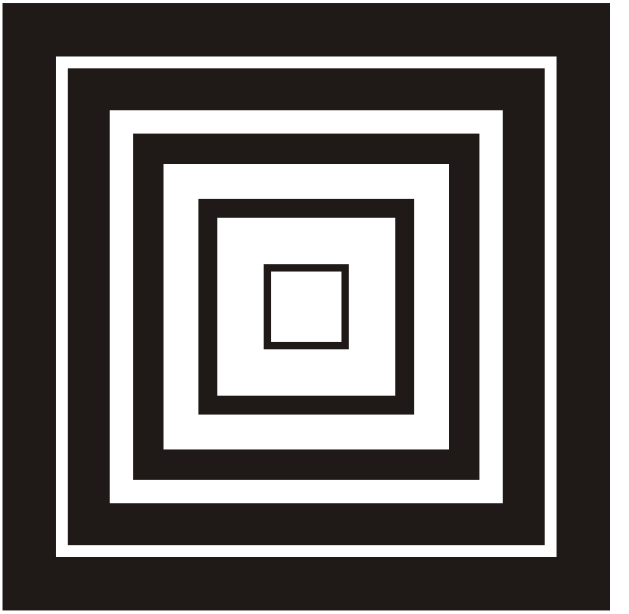
$$0 = \mathbf{f}\left(\frac{d\mathbf{x}}{dt}, \mathbf{x}, t\right),$$

которая может быть представлена в форме

$$\begin{cases} \dfrac{d\mathbf{x}}{dt} = \mathbf{z}, \\ 0 = \mathbf{f}(\mathbf{z}, \mathbf{x}, t). \end{cases}$$

Современные программные реализации численных методов прекрасно справляются с системами индекса 1, а вот с системами большего индекса могут возникать проблемы.

Программное обеспечение для решения систем основано на неявных многошаговых линейных методах, неявных методах Рунге—Кутты. Для практического использования можно рекомендовать программы DDASSL, RADAU5, а также программы из коллекции ODEPACK.



Приложения

Приложение 1

Тестовые матрицы с известной обратной

Тестовые матрицы заимствованы из книги [16]. Номера в скобках соответствуют исходным номерам в книге.

❒ **Пример 1 (2П).**

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & & & & & & 0 \\ -1 & 2 & -1 & & & & & \\ & -1 & 2 & -1 & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & -1 & 2 & -1 \\ 0 & & & & & & -1 & 2 \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1} = \left(a'_{ij}\right),$$

где

$$a'_{ij} = \frac{i(n+1-j)}{n+1},\ i \le j,$$

$$a'_{ij} = a_{ji},\ i > j,$$

$$\det \mathbf{A} = n+1.$$

❒ **Пример 2 (ЗП).**

$$\mathbf{A}=\begin{bmatrix} a & 1 & & & 0 \\ 1 & a & 1 & & \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ & & 1 & a & 1 \\ 0 & & & 1 & a \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1}=\frac{1}{a_n}\begin{bmatrix} a_0a_{n-1} & -a_0a_{n-2} & a_0a_{n-3} & . \\ -a_0a_{n-2} & a_1a_{n-2} & -a_1a_{n-3} & . \\ a_0a_{n-3} & -a_1a_{n-3} & a_2a_{n-3} & . \\ -a_0a_{n-4} & a_1a_{n-4} & -a_2a_{n-4} & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix},$$

где:

$$a_0=1,$$
$$a_1=a,$$
$$a_i=aa_{i-1}-a_{i-2},\ i\geq 2.$$

При значении $a=2\cos\varphi$ рекуррентные соотношения заменяются формулами $a_i=\dfrac{\sin(i+1)\varphi}{\sin\varphi}$, $i\geq 0$.

❒ **Пример 3 (8П).**

$$\mathbf{A}=\begin{bmatrix} x+b & 1 & & & & & & 0 \\ 1 & x & 1 & & & & & \\ & 1 & x & 1 & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & 1 & x & 1 \\ 0 & & & & & & 1 & x+a \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1}=\left(a'_{ij}\right),\quad a'_{ij}=a'_{ji}=\frac{(-1)^{i+j}\,r_{j-1}s_{n-i}}{(x+a)r_{n-1}-r_{n-2}},\quad j\le i,$$

где

$$\begin{aligned} &r_0=1,\\ &r_1=x+b,\\ &r_k=xr_{k-1}-r_{k-2},\ \ k=2,3,\ldots,\ \ n-1,\\ &s_0=1,\\ &s_1=x+a,\\ &s_k=xs_{k-1}-s_{k-2},\ \ k=2,3,\ldots,\ \ n-1. \end{aligned}$$

❒ **Пример 4 (9П).**

$$\mathbf{A}_{n+1}=\begin{bmatrix} 0 & x_1 & \cdot & \cdot & \cdot & \cdot \\ y_1 & 0 & x_2 & \cdot & \cdot & \cdot \\ & y_2 & 0 & x_3 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & y_{n-1} & 0 & x_n \\ \cdot & \cdot & \cdot & \cdot & y_n & 0 \end{bmatrix}$$

четного порядка $n+1$, $x_i\ne 0$, $y_i\ne 0$, $i=1,2,\ldots,n$.

$$\mathbf{A}_2^{-1}=\begin{bmatrix} 0 & \dfrac{1}{y_1} \\ \dfrac{1}{x_1} & 0 \end{bmatrix} \text{ при } n\ge 3,$$

$$\mathbf{A}_{n+1}^{-1}=\begin{bmatrix} & & & & & & & & 0 & b_m \\ & & & & & & & & 0 & 0 \\ & & & & & & & & 0 & b_{m-1} \\ & & & & & & & & . & . \\ & & & & \mathbf{A}_{n-1}^{-1} & & & & . & . \\ & & & & & & & & . & . \\ & & & & & & & & 0 & b_2 \\ & & & & & & & & 0 & 0 \\ 0 & 0 & 0 & . & . & . & 0 & 0 & 0 & b_1 \\ a_m & 0 & a_{m-1} & . & . & . & a_2 & 0 & a_1 & 0 \end{bmatrix}, \text{ где } m=\frac{n+1}{2},$$

$$a_k=\begin{cases} \dfrac{1}{x_n}, k=1, \\ \dfrac{(-1)^{k+1}}{x_n}\prod\limits_{j=1}^{k-1}\dfrac{y_{n-2j+1}}{x_{n-2j}}, k=2,3,\ldots,m. \end{cases}$$

$$b_k=\begin{cases} \dfrac{1}{y_n}, k=1 \\ \dfrac{(-1)^{k+1}}{y_n}\prod\limits_{j=1}^{k-1}\dfrac{x_{n-2j+1}}{y_{n-2j}}, k=2,3,\ldots,m. \end{cases}$$

$$\det\left(\mathbf{A}_{n+1}\right)=(-1)^{\frac{n+1}{2}}\prod_{i=1}^{\frac{n+1}{2}}\left(x_{2i-1}y_{2i-1}\right).$$

❒ **Пример 5 (21П).**

$$\mathbf{A}=\begin{bmatrix} \alpha+\delta & 1 & 0 & . & . & . & 0 \\ \alpha^2 & \alpha+\delta & 1 & . & . & . & 0 \\ \alpha^3 & \alpha^2 & \alpha+\delta & . & . & . & 0 \\ . & . & . & . & . & . & . \\ \alpha^{n-1} & \alpha^{n-2} & \alpha^{n-3} & . & . & . & 1 \\ \alpha^n & \alpha^{n-1} & \alpha^{n-2} & . & . & . & \alpha+\delta \end{bmatrix}. \text{ порядка } n,$$

$$\mathbf{A}^{-1}=\left(a'_{ij}\right);$$

$$a'_{ij} = \begin{cases} (-1)^{i+j} \dfrac{\Delta_{i-1}\Delta_{n-j}}{\Delta_n}, & i \le j; \\ -(\alpha\delta)^{i-j+1} \dfrac{\Delta_{j-2}\Delta_{n-i-1}}{\Delta_n}, & i > j, \end{cases}$$

где

$$\begin{aligned} &\Delta_{-1} = \frac{1}{\delta}, \\ &\Delta_0 = 1, \\ &\Delta_1 = \alpha + \delta, \\ &\Delta_k = \delta\Delta_{k-1} + \alpha\delta\Delta_{k-2}, \\ &\det \mathbf{A} = \Delta_n. \end{aligned}$$

❒ **Пример 6 (48П).**

$$\mathbf{A} = \begin{bmatrix} \alpha & 1 & 1 & . & . & . & 1 \\ 1 & \alpha & 1 & . & . & . & 1 \\ 1 & 1 & \alpha & . & . & . & 1 \\ . & . & . & . & . & . & . \\ 1 & 1 & 1 & . & . & .\alpha & 1 \\ 1 & 1 & 1 & . & . & . & \alpha \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1} = \left(a'_{ij}\right); \quad a'_{ij} = \frac{\alpha + n - 2}{(\alpha - 1)(\alpha + n - 1)}; \; i = j;$$

$$a'_{ij} = \frac{-1}{(\alpha - 1)(\alpha + n - 1)}; \; i \ne j.$$

❒ **Пример 7 (49П).**

$$\mathbf{A} = \begin{bmatrix} 1 & t & t & . & . & . & t \\ t & 1 & t & . & . & . & t \\ t & t & 1 & . & . & . & t \\ . & . & . & . & . & . & . \\ t & t & t & . & . & .1 & t \\ t & t & t & . & . & . & 1 \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1}=\begin{bmatrix} a & b & b & . & . & . & b \\ b & a & b & . & . & . & b \\ b & b & a & . & . & . & b \\ . & . & . & . & . & . & . \\ b & b & b & . & . & .a & b \\ b & b & b & . & . & . & a \end{bmatrix};$$

$$a=\frac{1+t(n-2)}{(1-t)(1+tn-t)};\ b=\frac{-t}{(1-t)(1+tn-t)}.$$

❒ **Пример 8 (37П, матрица Гильберта).**

$$\mathbf{A}=\left(a_{ij}\right) \text{ порядка } n,$$

$$a_{ij}=\frac{1}{i+j-1},\quad i,\ j=1,2,...,n,$$

так что

$$\mathbf{A}=\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & . & . & . & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & . & . & . & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & . & . & . & \frac{1}{n+2} \\ . & . & . & . & . & . & . \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & . & . & . & \frac{1}{2n-1} \end{bmatrix},$$

$$\mathbf{A}^{-1}=\left(a'_{ij}\right),$$

где

$$a'_{ij}=\frac{(-1)^{i+j}(n+i-1)!(n+j-1)!}{(i+j-1)\left[(i-1)!(j-1)!\right]^2(n-i)!(n-j)!}.$$

❒ **Пример 9 (97П).**

$$\mathbf{A}^{-1}=\mathbf{E}+a\mathbf{x}\mathbf{y}^T;\ \mathbf{A}^{-1}=\mathbf{E}-\frac{1}{1+a\cdot(\mathbf{x},\mathbf{y})}a\cdot\mathbf{x}\mathbf{y}^T,$$

где $\mathbf{x}$, $\mathbf{y}$ — произвольные векторы-столбцы; a — число.

❒ **Пример 10 (41П).**

$$\mathbf{A}=\left(a_{ij}\right)=\begin{bmatrix} \frac{1}{p+1} & \frac{1}{p+2} & \frac{1}{p+3} & \cdot & \frac{1}{p+n} \\ \frac{1}{p+2} & \frac{1}{p+3} & \frac{1}{p+4} & \cdot & \frac{1}{p+n+1} \\ \vdots & \vdots & \vdots & \cdot & \vdots \\ \frac{1}{p+n} & \frac{1}{p+n+1} & \frac{1}{p+n+2} & \cdot & \frac{1}{p+2n-1} \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1}=\left(a'_{ij}\right);$$
$$a'_{ij}=d_i a_{ij} d_j,$$

где

$$d_k=(-1)^{k-1}\frac{(p+n+k-1)!}{(n-k)!\ (\mathrm{k}-1)!\ (p+k-1)!};\ k=1,2,\ldots,n.$$

$$\det\mathbf{A}=\frac{p!(p+1)!\ldots(p+n-1)!\left[1!2!\ldots(n-1)!\right]^2}{(p+n)!\ (p+n+1)!\ldots(p+2n-1)!}.$$

❒ **Пример 11 (42П).**

$$\mathbf{A}=\left(a_{ij}\right) \text{ порядка } n,$$

$$a_{ij}=\frac{1}{p+i+j-1},$$
$$i,\ j=1,2,\ldots,n,\quad p\neq -1,-2,\ldots,-(2n-1),$$

$$\mathbf{A}^{-1}=\left(a'_{ij}\right);$$

$$a'_{ij}=\frac{(-1)^{i+j}}{p+i+j-1}\left[\frac{\prod_{k=0}^{n-1}(p+i+k)(p+j+k)}{(i-1)!\ (n-1)!\ (j-1)!\ (n-j)!}\right],$$

$$i,j=1,2,\ldots,n,\quad p\neq -1,-2,\ldots,-(2n-1).$$

❒ **Пример 12 (50П).**

$$\mathbf{A}=\begin{bmatrix} a+b & a & a & . & a \\ a & a+b & a & . & a \\ . & . & . & . & . \\ a & a & a & . & a+b \end{bmatrix} \text{ порядка } n,$$

$$\det \mathbf{A} = b^{n-1}(b+na),$$

$$\mathbf{A}^{-1}=\begin{bmatrix} a'+b' & a' & a' & . & a' \\ a' & a'+b' & a' & . & a' \\ . & . & . & . & . \\ a' & a' & a' & . & a'+b' \end{bmatrix},$$

$$a'=-\frac{a}{b(b+an)},$$

$$b'=\frac{1}{b}.$$

❒ **Пример 13 (77П).**

$$\mathbf{A}=\begin{bmatrix} a & x & . & . & . & x & x \\ x & a & . & . & . & x & x \\ . & . & . & . & . & . & . \\ x & x & . & . & . & a & x \\ x & x & . & . & . & x & x \end{bmatrix} \text{ порядка } n,$$

$$\mathbf{A}^{-1}=\begin{bmatrix} \frac{1}{a-x} & . & . & 0 & . & . & -\frac{1}{a-x} \\ . & . & & & & & . \\ . & & . & & & & . \\ 0 & & & . & & & . \\ . & & & & . & & . \\ . & & & & & \frac{1}{a-x} & -\frac{1}{a-x} \\ -\frac{1}{a-x} & . & . & . & . & -\frac{1}{a-x} & \frac{a+(n-2)x}{x(a-x)} \end{bmatrix},$$

$$\det \mathbf{A} = x(a-x)^{n-1}.$$

❒ **Пример 14 (98П).**

$$\mathbf{A} = \left(a_{ij}\right),\ \ a_{ij} = 1,\ \ j \ge i;\ a_{ij} = b_j;\ \ j < i;$$

$$\mathbf{A}^{-1} = \left(a'_{ij}\right);$$

$$a'_{ij} = \frac{1}{1-b_j};\ i = j;\ \ j \ne n,$$

$$a'_{ij} = \frac{-1}{1-b_j};\ \ j = i+1;\ i \ne n,$$

$$a'_{ij} = \frac{b_{j-1} - b_j}{(1-b_j)(1-b_{j-1})};\ i = n;\ \ j \ne 1, n,$$

$$a'_{ij} = \frac{-b_1}{(1-b_1)};\ i = n;\ \ j = 1;$$

$$a'_{ij} = \frac{-1}{(1-b_{n-1})};\ i = j = n,$$

$a'_{ij} = 0$ во всех остальных случаях.

❒ **Пример 15 (101П).**

$$\mathbf{A} = \begin{bmatrix} (x_1+y_1)^{-1} & (x_1+y_2)^{-1} & . & . & . & (x_1+y_n)^{-1} \\ (x_2+y_1)^{-1} & (x_2+y_2)^{-1} & . & . & . & (x_2+y_n)^{-1} \\ . & . & . & . & . & . \\ (x_n+y_1)^{-1} & (x_n+y_2)^{-1} & . & . & . & (x_n+y_n)^{-1} \end{bmatrix},$$

$$\mathbf{A}^{-1} = \left[a'_{ij}\right],$$

$$\sum_{i,j} a'_{ij} = (x_1 + x_2 + \ldots + x_n) + (y_1 + y_2 + \ldots + y_n).$$

$$a'_{ij} = \frac{\prod\limits_{1 \le k \le n} (x_j + y_k)(x_k + y_i)}{(x_j + y_i)\left[\prod\limits_{\substack{1 \le k \le n \\ k \ne j}} (x_j - x_k)\right]\left[\prod\limits_{\substack{1 \le k \le n \\ k \ne i}} (y_i - y_k)\right]},$$

$$\det \mathbf{A} = \frac{\prod\limits_{1 \le i < j \le n} (x_j - x_i)(y_j - y_i)}{\prod\limits_{1 \le i,\, j \le n} (x_i + y_j)}.$$

Приложение 2

Библиотека пакета MvStudium

Разработка численных методов в MvStudium проводилась одновременно с созданием модификации гибридного автомата, ориентированной на численное исследование моделей. Это позволило не подстраиваться, как это часто бывает, под уже существующий язык, а органично сочетать как требования дружественности языка, так и особенности программных реализаций численных методов.

Первые версии численной библиотеки были созданы в те годы, когда создание DLL было трудоемкой операцией, поэтому оказалось проще переписать основные численные методы на язык Pascal. Библиотека пакета MV 2.1 была ориентирована на вариант гибридного автомата, который сегодня используется в пакете Stateflow.

Появление удобных компиляторов для языка FORTAN позволило отказаться от необходимости переписывать код и сформировать численную библиотеку MVS (MVS или Model Vision Studium) из фортрановских модулей. Хотя практически все модули и подверглись существенным изменениям (что связано в основном с созданием эвристических алгоритмов-автоматов, пытающихся любой ценой решить совокупную систему уравнений, описывающую поведение модели), надежность такой библиотеки существенно выше, чем любой версии этой же библиотеки на другом языке.

Однако при создании пакета AnyLogic, написанного на языке Java, вновь встал вопрос о переводе исходных фортрановских текстов на новый язык. Как показало выборочное тестирование, замедление работы при переходе на язык Java оказалось вполне приемлемым (не более чем в два-три раза), зато пакет теперь создавал "чистые" Java-программы. Наличие прототипа библиотеки на языке FORTRAN позволило тщательно протестировать вновь создан-

ную версию библиотеки. Вместе с библиотекой на язык Java были перенесены и программы тестирования, что позволило сравнивать сначала результаты работы FORTRAN-библиотеки и Java-библиотеки, а затем воспроизводить тесты одновременно в пакетах MVS и AnyLogic. Проверенная таким образом библиотека, с небольшими доработками, и вошла в следующие программные продукты: AnyLogic (Java), OpenSourcePhysics (Java), MvStudium (Fortran).

Особенность моделирования систем в MvStudium в том, что решаемая текущая система уравнений при появлении событий, приводящих к смене поведения, может не только изменить свои численные свойства (жесткая, нежесткая, осциллирующая), но и тип (система нелинейных алгебраических уравнений, система обыкновенных дифференциальных уравнений, система алгебро-дифференциальных уравнений). Более того, по ходу эксперимента выбранная программная реализация численного метода может не справиться с конкретной задачей или превысить заданные по умолчанию ограничения, например, на количество вычисления правых частей при выбранном уровне погрешности. В этих случаях естественно возникает желание сделать программу управления и выбора конкретного численного метода настолько безотказной, насколько позволяет современное состояние теории и практики.

Организация библиотеки

Схематически библиотеку численных методов MvStudium можно изобразить следующим образом (табл. П2.1).

Таблица П2.1. *Состав библиотеки*

Уровень	**Группа**		
1.	BLAS	FLOAT	
2.	LINPACK	EISPACK	MINPACK
3.	LA	GNA	
4.	NAE_SLV	ODE_SLV	DAE_SLV

Программы библиотеки представляют собой иерархическую структуру. Уровни (строки таблицы) являются независимыми группами, использующими только программы предыдущих уровней (табл. П2.2).

Таблица П2.2. *Перечень специализированных коллекций и их назначение*

Название	**Назначение**
BLAS	Систематизированная коллекция подпрограмм, реализует основные элементарные операции линейной алгебры. Используется для структуризации и ускорения работы программ высшего уровня
FLOAT	Подпрограммы этого раздела вычисляют основные характеристики машинной арифметики
LINPACK	Систематизированная коллекция подпрограмм предназначена для решения систем линейных алгебраических уравнений с матрицами различного типа
EISPACK	Систематизированная коллекция подпрограмм предназначена для решения проблемы собственных значений
MINPACK	Систематизированная коллекция подпрограмм предназначена для нахождения минимумов функционалов и решения систем нелинейных алгебраических уравнений
LA	Подпрограммы этой группы реализуют основные матричные операции
GNA	Группа подпрограмм, предназначенная для решения основных задач численного анализа общего назначения
NAE_SLV	Группа подпрограмм, предназначенная для решения систем нелинейных алгебраических уравнений, возникающих при описании поведения
ODE_SLV	Группа подпрограмм, предназначенная для решения систем обыкновенных дифференциальных уравнений, возникающих при описании поведения
DAE_SLV	Группа подпрограмм, предназначенная для решения систем алгебро-дифференциальных уравнений, возникающих при описании поведения

Пользователь не может непосредственно обращаться к подпрограммам библиотеки. Нужная программа подключается автоматически, либо после анализа примененной языковой конструкции, если речь идет о написанных пользователем процедурах и функциях, либо после окончания диалога с пользователем, когда речь идет о выборе метода для решения конкретного типа уравнений. Такой подход является единственно возможным при смене поведения.

Структура решателей

Для решения каждой из основных задач (систем нелинейных алгебраических, дифференциальных и алгебро-дифференциальных уравнений) пользователю предлагается на выбор несколько программных реализаций численных методов. Указать заранее наиболее подходящий метод решения конкретной задачи удается редко, поэтому важной особенностью современных графических оболочек является простота процедуры смены численного метода. В MvStudium это можно делать даже в процессе решения. В то же время существует значительная категория пользователей, которой безразлично, каким методом будет получено решение, лишь бы оно правильно отражало поведение объекта. Аналогичное отношение к численным методам наблюдается и на этапе отладки. При отладке, когда модель демонстрирует "необъяснимое" поведение, бывает важно просто "двигать" время вперед любым доступным способом и смотреть, что произойдет с ключевыми переменными. Программные реализации численных методов, гарантирующие получение решения с заданной точностью, не способны, как правило, справиться с задачами, возникающими при отладке, и приходится применять очень примитивные решатели, вплоть до методов с постоянным шагом.

Исходя из сказанного, в MvStudium пользователю предлагаются три категории программных реализаций численных методов.

Автоматические решатели

Их назначение — автоматически осуществить подбор наименее трудоемкой программной реализации численного метода для решения конкретной задачи, а в случае неудачи предоставить пользователю максимально подробную информацию о встреченных трудностях. Анализируя поведение автоматического решателя, пользователь может затем выбрать конкретный метод.

Решатели для задач конкретного типа

Если пользователю известен тип решаемой задачи, то достаточно просто выбрать один из предлагаемых методов соответствующего класса. Однако и в этом случае пользователь имеет дело не непосредственно с программной реализацией из соответствующей систематизированной коллекции, а с управляющей программой, анализирующей коды завершения вызываемой

подпрограммы и, в случае необходимости, меняющей ее параметры так, чтобы обеспечить получение решения на всем интервале моделирования. Например, если программная реализация не может обеспечить получение решения с указанной пользователем точностью, то управляющая программа сначала попытается изменить точность, а если это не поможет, то только тогда прекратит счет и выдаст аварийное сообщение.

Выбор конкретного метода осуществляется пользователем в диалоговом режиме, простейший вариант которого сводится к использованию "метода по умолчанию". В качестве "метода по умолчанию" предлагается автоматический решатель. Автоматический решатель в заданной последовательности перебирает включенные в него методы, чтобы обеспечить получение решения на всем временном промежутке.

Аварийная остановка любого метода сопровождается диагностическим сообщением. В случае аварийного завершения работы можно попытаться сменить метод, но лучше устранить причину прекращения счета. "Плохие" численные задачи, конечно же, существуют, но практика показывает, что плохо сформулированных задач гораздо больше.

Решатели для отладки

Цель этой группы методов продвигать модельное время вперед как можно дальше, чтобы дать возможность пользователю понять природу ошибок. К этой группе, предназначенной для решения дифференциальных уравнений, относятся явный и неявный методы Эйлера с автоматическим выбором шага (SIMPLE_AUTO) и метод Рунге—Кутты четвертого порядка с постоянным шагом. Для отладки моделей, основанных на нелинейных и алгебро-дифференциальных уравнениях, специальные методы не предусмотрены.

В MvStudium различаются три основные численные задачи и три группы соответствующих методов (табл. П2.3).

Таблица П2.3. *Численные задачи и их методы решения*

Задача	**Автомат**	**Конкретные программы**
Решение систем нелинейных алгебраических уравнений	AUTO_NAE	• Различные модификации метода Ньютона • Гибридные методы

Таблица П2.3 *(окончание)*

Задача	**Автомат**	**Конкретные программы**
Решение систем дифференциальных уравнений	AUTO_ODE	• Нежесткие задачи • Явные методы Рунге—Кутты • Методы Адамса • Жесткие задачи • Неявные методы Рунге—Кутты • Методы, основанные на формулах дифференцирования назад • Отладочные программы • RKF4
Решение систем алгебро-дифференциальных уравнений	AUTO_DAE	• Явные методы • Неявные методы

Системы нелинейных алгебраических уравнений

Решение системы нелинейных алгебраических уравнений, как основная задача, возникает при соответствующем способе описания поведения, и, как вспомогательная, при реализации неявных методов решения систем алгебро-дифференциальных уравнений. В том и другом случае, за исключением начальной точки, начальное приближение к решению можно считать достаточно хорошо известным, что позволяет ориентироваться в основном на различные модификации метода Ньютона. В начальный момент времени при смене поведения ситуация значительно усложняется, т. к. у пользователя может отсутствовать информация об удовлетворительном начальном приближении.

Программы для решения нелинейных уравнений

Для решения систем нелинейных уравнений, возникающих при описании поведения, используются:

- Fast Newton (Newt_F) — "быстрый" метод Ньютона, позволяющий не вычислять матрицу Якоби в очередной временной точке заново, а использовать сохраненное значение с предыдущей точки (пересчет матрицы Якоби через каждые пять итераций);
- Modified Newton (Newt_M) — модифицированный метод Ньютона (пересчет матрицы Якоби через фиксированное число итераций или по мере уменьшения скорости сходимости);
- Classical Newton (Newt_G) — классический вариант метода Ньютона (пересчет матрицы Якоби на каждой итерации);
- Hybrid Powell method (Powell) — гибридный метод Пауэлла (MinPack);
- Auto_NAE — автомат пытается последовательно применять методы (1, 2, 3, 4). Аварийная остановка автоматического решателя означает, что для данного начального приближения, взятого с предыдущего шага, ни один из методов не нашел удовлетворительного решения.

Примечание

В скобках указаны имена подпрограмм, используемые в соответствующей DLL-библиотеке.

Подпрограммы Newt_F, Newt_M, Newt_G являются управляющими для подпрограммы NTDF8.

Подпрограмма NTDF8 представляет собой традиционную реализацию метода Ньютона. На каждом шаге с помощью подпрограмм из коллекции Linpack решается система линейных уравнений относительно приращения к решению, найденному на предыдущей итерации. Матрица Якоби определяется численно. Приближенное решение считается найденным с удовлетворительной точностью, если выполнено условие $\|y_{n+1} - y_n\| \le \|y_{n+1}\|\varepsilon_{\text{отн}} + 100 \cdot \varepsilon_{\text{м}}$. Величина относительной погрешности $\varepsilon_{\text{отн}}$ задается пользователем. Абсолютная погрешность положена равной $100 \cdot \varepsilon_{\text{м}}$. Напомним, что $\varepsilon_{\text{м}}$ — это

минимальное машинное число, при сложении с единицей дающее результат, отличный от единицы.

Задавая различные значения переменным подпрограммы NTDF8 — `ipr`, `inorm`, `nJc`, — можно:

- вычислять матрицу Якоби заново в начальной точке или использовать уже известное значение (`ipr`);
- оценивать величину погрешности, опираясь на различные нормы решения (`inorm`);
- пересчитывать матрицу Якоби через заданное число итераций (`njc`).

Все перечисленные методы ньютоновского типа получены из общей процедуры за счет выбора соответствующих значений этих переменных.

Переменные `nfcnN`, `njacN`, `nitN` общего блока common/stat/nfcnN, `njacN`, `nitN` хранят: `nfcnN` — общее число обращений к правой части; `njacN` — количество вычислений матрицы Якоби; `nitN` — общее число итераций. Пользователь MVS доступа к этим переменным не имеет, они служат для тестирования библиотеки.

Программные реализации метода Ньютона имеют следующие коды завершения (табл. П2.4).

Таблица П2.4. *Коды завершения программных реализаций метода Ньютона*

Код	**Описание**
`ier = 0`	Успешное завершение
`ier = 2`	Прерывание, вызванное арифметическими ошибками, при попытке вычислить правую часть
`ier = -1`	Метод Гаусса обнаружил численную вырожденность матрицы решаемой линейной системы
`ier = -3`	Медленная сходимость
`ier = -4`	Число вычислений правой части превысило допустимое число `nmax=c_itmax*n`

В табл. П2.5 приведены пояснения и рекомендуемые действия при возникновении соответствующих кодов завершения.

Таблица П2.5. *Пояснения к кодам завершения (для методов Ньютона)*

Код	**Пояснение**
`ier = 2`	Любая попытка выполнить недопустимую арифметическую операцию (деление на ноль, попытка вычислить квадратный корень из отрицательного числа и др.) приведет к остановке вычислений и выводу на экран сообщения с указанием ошибочного выражения
`ier = -1`	Код `ier = -1` показывает, что построенная линейная система для определения очередного приближения к решению оказалась вырожденной. В этом случае рекомендуется сменить метод. Постарайтесь понять, используя дополнительные сведенья, почему вашу нелинейную задачу в окрестности приближения не удается заменить линейной
`ier = -3`	При хорошем начальном приближении методы ньютоновского типа теоретически сходятся очень быстро. За одну итерацию можно получить удвоение числа верных знаков в приближенном решении. Однако при поиске хорошего начального приближения или при неудачном выборе последнего возможны ситуации, когда и число верных знаков от итерации к итерации практически не изменяется, и приближенное решение не удовлетворяет заданному условию по точности. Попробуйте поискать иное начальное приближение из "физических" соображений или сменить метод
`ier = -4`	Встроенные в MVS методы относятся к итерационному типу. Число разрешенных итераций ограничено числом допустимых вычислений функции, а именно константой `n*100`, где `n` — размерность решаемой системы. Превышение этого значения без достижения нужной точности приведет к принудительному завершению с кодом `ier = -4`

Помимо методов ньютоновского типа, на практике часто применяют методы минимизации специально построенных функционалов, точка минимума которых совпадает с корнем исходного уравнения. К числу таких методов относится гибридный метод Пауэлла (MinPack). Этот метод часто не требует знания хорошего начального приближения. В то же время, возможно, что программа найдет не корень уравнения, а локальный минимум вспомогательного функционала. Если ваша система допускает нахождение решения различными методами, старайтесь это делать для проверки.

Сообщения об ошибках гибридного метода Пауэлла приведены в табл. П2.6. В табл. П2.7 даны краткие пояснения к сообщениям. Таблица П2.7 отличается от табл. П2.5 только одной строкой, соответствующей коду `ier = -2`, поэтому она (таблица) приводится в сокращении.

Таблица П2.6. *Коды завершения программных реализаций метода Пауэлла*

Код	**Описание**
`ier = 0`	Успешное завершение
`ier = 2`	Прерывание, вызванное арифметическими ошибками, при попытке вычислить правую часть
`ier = -1`	Метод Гаусса обнаружил численную вырожденность матрицы решаемой линейной системы
`ier = -2`	Завышена требуемая точность
`ier = -3`	Медленная сходимость
`ier = -4`	Число вычислений правой части превысило допустимое число `nmax=c_itmax*n`

Таблица П2.7. *Пояснения к кодам завершения (для методов Пауэлла)*

Код	**Пояснение**
`ier = -2`	Попробуйте изменить точность. При заданной точности найти лучшее приближение не удается

Программа-автомат

Управляющая подпрограмма AUTO_NAE последовательно вызывает сначала методы ньютоновского типа, а затем гибридный метод Пауэлла. В случае неудачи очередного метода коды завершения игнорируются, и делается попытка применить очередной метод. На следующем временном шаге делается попытка вновь применить самый простой метод, а именно Fast Newton.

Переменная `istart` управляет только работой "быстрого" модифицированного варианта метода Ньютона (Fast Newton). Значение `istart=0` требует вы-

числения матрицы Якоби в начальной точке. При успешном нахождении решения `istart` на выходе метода полагается равным 1 и при последующем обращении повторное вычисление матрицы не происходит. При смене поведения значение `istart` автоматически полагается равным 0.

Переменная `method` хранит номер (табл. П2.8) примененного на шаге метода. В табл. П2.9 приведены коды завершения.

Таблица П2.8. *Соответствие методов переменной* `method`

Значение	**Метод**
`method = 1`	Fast Newton (Newt_F)
`method = 2`	Modified Newton (Newt_M)
`method = 3`	Classical Newton (Newt_G)
`method = 4`	Hybrid Powell method (Powell)

Таблица П2.9. *Коды завершения программы AUTO_NAE*

Код	**Описание**
`ier = 0`	Успешное завершение
`ier = 2`	Возникли арифметические ошибки при попытке вычислить правую часть
`ier = -1`	Матрица линейной системы вырождена
`ier = -2`	Затребованная точность завышена
`ier = -3`	Слишком медленная сходимость
`ier = -4`	Превышено допустимое число вычислений правой части

`method=1`
`method=2`
`method=3`
`method=4`
	Система решена методом:
	• Newt_F;
	• Newt_M;
	• Newt_G;
	• Powell

Дифференциальные уравнения

Для решения систем дифференциальных уравнений применяются либо одношаговые (для получения решения в новой точке используется информация только с предыдущей временной точки), либо линейные многошаговые методы (используется информация, полученная на нескольких предыдущих точках) методы. И те, и другие могут быть как явными (для решения нежестких задач), так и неявными (для решения жестких задач). Свойство жесткости существенно влияет на время, затраченное на получение решения.

Подпрограммы библиотеки разделены на программы для решения жестких и нежестких уравнений и программы, автоматически настраивающиеся на тип решаемой задачи.

Во всех программах-автоматах, предназначенных для решения задач, тип которых заранее неизвестен пользователю, система первоначально считается нежесткой, т. к. решение нежестких уравнений чаще оказывается менее трудоемким процессом, чем решение жестких (табл. П2.10). При возникновении жесткости такие подпрограммы автоматически переключаются на программные реализации, предназначенные для решения жестких уравнений. Наиболее широкое признание среди подпрограмм автоматов получила подпрограмма LSODA, построенная на линейных многошаговых методах. Подпрограмм автоматов, построенных на одношаговых методах, практически не существует. В MVStudium в качестве автомата предлагается использовать программу AUTO_ODE, которая сначала пытается применить явный одношаговый, а при обнаружении жесткости — неявный многошаговый метод.

Таблица П2.10. *Решение систем обыкновенных дифференциальных уравнений*

По умолчанию	**По выбору пользователя**	**Программы для отладки**
Автомат	Специализированные программы	Отладочные программы
AUTO_ODE	• Нежесткие задачи • Одношаговые, явные методы (явные методы Рунге—Кутты) • Линейные многошаговые, явные (методы Адамса)	Простейший автомат, построенный на явном и неявном методах Эйлера

Таблица П2.10 *(окончание)*

По умолчанию	**По выбору пользователя**	**Программы для отладки**
	• Жесткие задачи • Одношаговые, неявные методы (неявные методы Рунге—Кутты) • Линейные многошаговые, неявные (методы, построенные на формулах дифференцирования назад)	Метод Рунге—Кутты четвертого порядка с фиксированным шагом

Существуют также специализированные программы, предназначенные для решения только соответствующего типа задач. Некоторые программы для решения нежестких уравнений могут обнаруживать возникновение жесткости по ходу решения и предупреждать об этом.

Группа подпрограмм для решения нежестких уравнений содержит различные варианты метода Рунге—Кутты (RKF45, RKF853, RKF78) и одну реализацию линейных многошаговых методов (LSODE_A).

Программы RKF45 и RKF853 автоматически выбирают начальный шаг и предупреждают пользователя о появлении свойства жесткости у решаемой задачи. Программа RKF78 механизма оценки жесткости не имеет. О возникновении этого явления при ее использовании можно судить только косвенно, если число допустимых вычислений правой части превысило заданное значение и при этом задача имеет достаточно медленно изменяющееся решение на значительной части промежутка интегрирования.

Программа LSODE_A реализует явные линейные многошаговые методы и является управляющей для подпрограммы LSODE при выборе значения параметра равным 10 (`mf=10`).

Группа подпрограмм для решения жестких уравнений включает подпрограмму LSODE_B, основанную на неявных многошаговых методах. Подпрограмма LSODE_B является управляющей для подпрограммы LSODE при выборе параметра равным 22 (`mf=22`). В эту группу так же входят подпрограммы из коллекции для решения жестких уравнений, использующие неявные одношаговые методы.

Программы для решения нежестких уравнений

Все подпрограммы, реализующие методы Рунге—Кутты, имеют единый набор вспомогательных программ, общий интерфейс и одни и те же коды завершения. Краткие сообщения (коды завершения) можно найти в табл. П2.11, а пояснения к ним — в табл. П2.12.

Таблица П2.11. *Коды завершения программ, реализующих методы Рунге—Кутты*

Код	Описание
`ier = 0`	Успешное завершение
`ier = 22`	Арифметические ошибки при вычислении правой части дифференциальной системы
`ier = -1`	Переключение поведения
`ier = -2`	Возможно, что задача в точке останова стала жесткой
`ier = -3`	Сделано больше шагов, чем предусмотрено
`ier = -4`	Шаг интегрирования стал недопустимо малым

Таблица П2.12. *Пояснения к кодам завершения программ, реализующих методы Рунге—Кутты*

Код	Пояснение
`ier = 22`	Проверьте правильность описания локального поведения. Формула, приведенная в сообщении, содержит недопустимые арифметические операции
`ier = -2`	Смените метод на один из методов, предназначенных для решения жестких задач
`ier = -3`	Для начала уменьшите запрашиваемую точность. Если это не поможет, перейдите к программе AUTO_DE и выясните, не является ли задача жесткой. Если ничего не помогает, введите в описание локального поведения устройства дополнительный узел с точно таким поведением.

Таблица П2.1 *(окончане)*

Код	Пояснение
	Выход из узла предусмотрите по времени. Значение времени выхода должно быть меньше, чем значение времени, когда появляется данное сообщение. Смена поведения (на точно такое же!) приведет к автоматическому обнулению всех внутренних счетчиков. И постарайтесь понять, почему ваша система так себя ведет в данной точке
`ier = -4`	Шаг интегрирования стал настолько малым, что продвижение вперед стало невозможным. Значение `t+h` оказывается равным `t`. Возможно, что система плохо масштабирована

Коды завершения программы LSODE_a приведены в табл. П2.13, а пояснения к ним — в табл. П2.14. Таблица П2.14 приводится в сокращении, т. к. часть ее строк совпадает с соответствующими строками табл. П2.12.

Таблица П2.13. *Коды завершения программы LSODE_a*

Код	Описание
`ier = 0`	Нормальный выход
`ier = 22`	Арифметические ошибки при вычислении правой части дифференциальной системы
`ier = -1`	Переключение поведения
`ier = -2`	Превышено допустимое число шагов
`ier = -3`	Завышена точность. Может возникать также, если `tin` и `tout` очень близки для данного локального поведения
`ier = -4`	Не прошел тест на точность

Таблица П2.14. *Пояснения к кодам завершения программы LSODE_a*

Код	Пояснение
`ier = -2`	Смените метод. Начните с программы AUTO_ODE. Если она тоже укажет на наличие жесткости, можете переходить к специальным методам решения жестких задач

Таблица П2.14 *(окончание)*

Код	Пояснение
ier = -3	Измените точность или смените метод решения. Возможно так же, что время "жизни" в данном узле карты поведения оказалось чрезвычайно коротким. Проверьте описание поведения
ier = -4	Этот код говорит о невозможности найти решение с заданной точностью в указанной временной точке. Возможно, что эта точка особенная. Проверьте описание локального поведения

Программы для решения жестких уравнений

Подпрограммы для решения жестких задач представлены программой LSODE_b (управляющая для программы LSODE с выбором параметра `mf=22`) и подпрограммой RADAU5 из коллекции.

Линейные многошаговые методы для решения жестких систем в пакете реализованы в подпрограмме LSODE. Вызов подпрограммы LSODE с параметром `mf`, равным 22 (`mf=22`), обеспечивает применение этих методов. Обращение к программе LSODE_b полностью аналогично обращению к программе LSODE_a, за исключением того, что необходимо выделить другое количество памяти рабочим массивам.

Коды завершения программы LSODE_b приведены в табл. П2.15, а пояснения к ним — в табл. П2.16. Таблица П2.16 отличается от табл. П2.14 только одной строкой, а именно строкой, соответствующей коду завершения `ier=-5`, и трактовкой сообщения с кодом `ier=-2`, поэтому она приводится в сокращении.

Таблица П2.15. *Коды завершения программы LSODE_b*

Код	Описание
ier = 0	Нормальный выход
ier = 22	Арифметические ошибки
ier = -1	Переключение поведения
ier = -2	Завышена точность

Таблица П2.15 *(окончание)*

Код	Описание
`ier = -3`	Превышено допустимое число шагов
`ier = -4`	Не прошел тест на точность
`ier = -5`	Не сходится метод Ньютона
`ier = -7`	Очень близки `tin` и `tout` для данного локального поведения

Таблица П2.16. *Пояснения к кодам завершения программы LSODE_b*

Код	Пояснение
`ier = -2`	Скорее всего, ваша задача относится к числу трудных для данного метода. Возможно, что получаемое численное решение не обладает каким- либо важным для задачи свойством
`ier = -5`	Скорее всего, в этой временной точке у решения есть особенность. Проверьте правильность описания локального поведения
`ier = -7`	Время существования системы в данном узле оказалось таким коротким, что численный метод "решил", что вы ошиблись. Проверьте описание поведения

Неявные одношаговые методы представлены подпрограммой RADAU5 [11], реализующей неявный метод Рунге—Кутты. Коды завершения подпрограммы приведены в табл. П2.17, пояснения к ним даны в табл. П2.18. Таблица П2.18 приведена в сокращении (см. табл. П2.14, П2.16).

Таблица П2.17. *Коды завершения подпрограммы RADAU5, реализующей неявный метод Рунге—Кутты*

Код	Описание
`ier = 0`	Нормальный выход
`ier = 22`	Арифметические ошибки
`ier = -1`	Переключение поведения
`ier = -2`	Шаг интегрирования стал слишком маленьким (`h<eps_m`)

Таблица П2.17 *(окончание)*

Код	Описание
`ier = -3`	Превышено допустимое число вычислений правой части
`ier = -4`	Шаг интегрирования стал таким, что `t+h` оказалось равным `t`
`ier = -6`	Не сходится метод Ньютона
`ier = -7`	Недопустимая точность

Таблица П2.18. *Пояснения к кодам завершения подпрограммы RADAU5*

Код	Пояснение
`ier = -2`, `ier = -6`	Скорее всего, в этой временной точке у решения есть особенность. Проверьте правильность описания локального поведения
`ier = -3`	Попробуйте изменить точность или метод
`ier = -4`	Скорее всего, задача плохо масштабирована
`ier = -7`	Увеличьте заданную точность

Программа-автомат

Подпрограмма-автомат представлена программой AUTO_ODE. Назначение данной подпрограммы — избавить пользователя от необходимости выбирать метод и предоставить информацию о типе решаемой численной задачи.

Краткие сообщения об ошибках содержатся в табл. П2.19, а пояснения к ним — в табл. П2.20.

Подпрограмма AUTO_ODE начинает работу с вызова подпрограммы RKF45, предназначенной для решения нежестких задач. Подпрограмма RKF45 способна оценивать жесткость решаемой задачи, что позволяет автоматически переключиться на соответствующий метод. Для решения жестких задач выбрана подпрограмма LSODA, в свою очередь способная "почувствовать", что решаемая задача вновь стала нежесткой. Программа AUTO_ODE следит за значением соответствующих переменных и автоматически переключается с одной подпрограммы на другую. Момент пере-

ключения с методов решения жестких задач на методы решения нежестких может быть выбран из различных соображений. Специальная переменная `imstop`, расположенная в общем блоке common/auto/imstop, управляет моментом переключения. Если `imstop=1`, то переключение происходит немедленно, после того как стало ясным, что задача стала нежесткой. Если `imstop=0`, то управляющая программа переключится только тогда, когда будет достигнут конец очередного промежутка интегрирования.

Переменная `istart` сообщает подпрограмме AUTO_ODE, каким методом осуществлять интегрирование на очередном шаге. При первом обращении ее следует положить равной 10. Дальнейшее ее значение выбирается автоматически. Переменная `method` содержит информацию о выбранном методе, а именно `method=1` означает, что на данном промежутке работала подпрограмма RKF45, а `method=2`, что интегрирование выполнялось подпрограммой LSODA.

Таблица П2.19. *Коды завершения программы AUTO_ODE*

Код	**Описание**
`ier = 0`	Успешное завершение
`ier = 22`	Арифметические ошибки при вычислении правой части дифференциальной системы
`ier = -1`	Внешнее прерывание
`method = 1`	
`ier = -3`	Выполнено больше шагов, чем предусмотрено
`ier = -4`	Шаг интегрирования стал недопустимо малым
`method = 2`	
`ier = -2`	Завышена точность
`ier = -3`	Превышено допустимое число шагов
`ier = -4`	Многократно не прошел тест на точность
`ier = -5`	Многократно не сошелся метод Ньютона
`ier = -7`	Очень близки `tin` и `tout` для данного локального поведения

Таблица П2.20. *Пояснения к кодам завершения программы AUTO_ODE*

Код	**Пояснение**
`method = 1`	
`ier = -3`	Прежде всего, попробуйте изменить точность. Если это не поможет, попробуйте сменить метод. Выберите сначала любой, отличный от RKF45, метод для решения нежестких задач
`ier = -4`	Скорее всего, в этой точке у решения есть особенность
`method = 2`	
`ier = -2`	Попробуйте изменить точность, а затем попробуйте любой из методов для решения жестких задач
`ier = -3`	Попробуйте изменить точность, а затем попробуйте любой из методов для решения жестких задач
`ier = -4`	Скорее всего, в этой точке решение имеет особенность
`ier = -5`	Скорее всего, в этой точке решение имеет особенность
`ier = -7`	Время существования системы в данном узле оказалось таким коротким, что численный метод "решил", что вы ошиблись. Проверьте описание поведения

Отладочные программы

Как уже отмечалось, отладочными программами являются подпрограммы SIMPLE и RKF4.

Отличие программ SIMPLE и LSODA в том, что в первой введено ограничение на порядок применяемых методов, а именно — применяются только явный и неявный методы Эйлера. Такой выбор порядка обусловлен тем, что для задач с сильно меняющейся правой частью одношаговые методы способны преодолеть трудные участки лучше многошаговых методов. Потребность в таких методах возникает и при отладке, когда "плохое" поведение может быть обусловлено просто ошибками. Программу SIMPLE не следует применять для интегрирования отлаженных задач.

Коды возврата программы SIMPLE приведены в табл. П2.21. Пояснения к кодам завершения соответствуют табл. П2.16.

В программе SIMPLE предусмотрен анализ типа решаемых уравнений. О том, как происходило интегрирование, можно узнать по значению возвращаемой переменной `method`. Переменная `method` принимает значение 1, если на промежутке интегрирования применялись только нежесткие методы. Переменная `method` принимает значение 2, если на промежутке интегрирования применялись только жесткие методы. Переменная `method` принимает значение 11, если программа переключалась с жестких методов на нежесткие и обратно.

Таблица П2.21. *Коды завершения программы SIMPLE*

Код	**Описание**
`ier = 0`	Нормальный выход
`ier = 22`	Арифметические ошибки
`ier = -1`	Переключение поведения
`ier = -2`	Завышена точность
`ier = -3`	Превышено допустимое число шагов
`ier = -4`	Не прошел тест на точность
`ier = -5`	Не сходится метод Ньютона
`ier = -7`	Очень близки `tin` и `tout` для данного локального поведения

Алгебро-дифференциальные уравнения

Системы алгебро-дифференциальных уравнений могут представлять значительные трудности для существующих численных методов.

Начнем с того, что алгебро-дифференциальные уравнения могут быть записаны в различной форме. В существующих версиях пакета рассматривается самая простая форма алгебро-дифференциальных уравнений — дифференциальные уравнения должны быть системой уравнений первого порядка, разрешенных относительно производных, и к ним следует добавить для получения замкнутой системы необходимое число алгебраических уравнений.

Выбранное представление позволяет решать систему алгебро-дифференциальных уравнений двумя способами:

- решать систему дифференциальных уравнений любым явным методом, численно разрешая при каждом обращении к правой части систему нелинейных алгебраических уравнений;
- воспользоваться тем, что неявные методы для решения дифференциальных уравнений справляются и с алгебро-дифференциальными уравнениями.

Первый способ будем называть явным, а второй — неявным.

Явный способ

Для реализации первого подхода мы выбрали метод Ньютона для решения алгебраических систем, подпрограмму AUTO_NAE для нахождения согласованных начальных условий и явные методы Рунге—Кутты для решения дифференциальных систем. Все подпрограммы этого класса имеют единый интерфейс и одинаковые коды завершения. В табл. П2.22 приведены коды завершения, пояснения к ним даны в табл. П2.23.

Таблица П2.22. *Коды завершения программы AUTO_NAE*

Код	**Описание**
`ier = 0`	Успешное завершение
`ier = 2`	Арифметические ошибки при вычислении правой части алгебраической системы
`ier = 22`	Арифметические ошибки при вычислении правой части дифференциальной системы
`ier = -1`	Внешнее прерывание
`ier = -2`	Возможно, что задача в точке останова стала жесткой
`ier = -3`	Требуется больше вычислений правой части, чем предусмотрено
`ier = -4`	Шаг интегрирования стал недопустимо малым
`ier = -5`	Не удалось найти согласованные начальные условия
`ier = -6`	Не удалось решить алгебраическую систему по ходу решения
`ier = -7`	Не удалось вычислить начальный шаг

Таблица П2.23. *Пояснения к кодам завершения программы AUTO_NAE*

Код	**Пояснение**
`ier = 22`	Проверьте правильность описания локального поведения. Формула, приведенная в сообщении, содержит недопустимые арифметические операции
`ier = -2`	Смените метод
`ier = -3`	Для начала уменьшите запрашиваемую точность. Если это не помогает, введите в описание локального поведения устройства дополнительный узел, выход из которого предусмотрите по времени. Это время должно быть меньше, чем время, когда появляется сообщение. Смена поведения (на точно такое же) приведет к автоматическому обнулению всех внутренних счетчиков. И постарайтесь понять, почему ваша система так себя ведет в данной точке
`ier = -4`	Шаг интегрирования стал настолько малым, что продвижение вперед стало невозможным. Значение `t+h` оказывается равным `t`. Возможно, что система плохо масштабирована
`ier = -5`	Заданные вами начальные значения не согласованы. Возможно, начальное значение слишком плохое, но, возможно, что нелинейная система не имеет решения. Менять метод интегрирования бессмысленно, т. к. процедура поиска согласованного начального условия у всех методов одинакова
`ier = -6`	В процессе решения дифференциального уравнения не удалось разрешить алгебраическую систему методом Ньютона. Прежде чем выдать это сообщение, программа пыталась уменьшить шаг интегрирования. Попробуйте сменить метод, но скорее всего, что в этой точке алгебраическая система не имеет решения

Неявный способ

Подпрограммы этой группы представлены подпрограммой DDASSL. Подпрограмма рассчитана на много более сложные формы представления уравнений, чем это допустимо в MVS. Коды завершения приведены в табл. П2.24, пояснения к ним — в табл. П2.25.

Таблица П2.24. *Коды завершения программы DDASSL*

Код	**Описание**
`ier = 0`	Нормальный выход
`ier = 22`	Арифметические ошибки при вычислении правой части алгебро-дифференциальной системы
`ier = -1`	Переключение поведения
`ier = -2`	Завышена точность
`ier = -3`	Превышено допустимое число шагов
`ier = -4`	Не прошел тест на точность
`ier = -5`	Не сходится метод Ньютона
`ier = -6`	Матрица Якоби вырождена
`ier = -7`	Невозможно построить согласованные начальные условия
`ier = -8`	Не проходит тест на точность и одновременно не сходится метод Ньютона

Таблица П2.25. *Пояснения к кодам завершения программы DDASSL*

Код	**Пояснение**
`ier = -4`	Многократно не удалость удовлетворить тесту по проверке ошибки на последнем шаге. Скорее всего, что вы имеете дело с особой точкой
`ier = -5`	Многократно не прошел тест на сходимость на последнем шаге. Возможно, что либо неправильно вычисляется матрица Якоби, либо она плохо обусловлена
`ier = -6`	Матрица частных производных вырождена. Возможно, что некоторые из ваших уравнений являются следствием остальных. С такой задачей программа справиться не может. Попробуйте удалить зависимые уравнения. Возможно, что решение не существует либо оно не единственно
`ier = -7`	Невозможно построить согласованные начальные условия

Таблица П2.25 *(окончание)*

Код	Пояснение
`ier = -8`	Многократно не прошел тест на сходимость. Возможно, ваша задача плохо сформулирована, и подпрограмма с ней справиться не может. Возможно, однако, что у решения есть особая точка, либо оно претерпевает разрыв

Программа-автомат для решения алгебро-дифференциальных уравнений

В качестве основной программы для решения алгебро-дифференциальных уравнений выбрана подпрограмма LSODI. Подпрограмма SLODI является управляющей. Ее основная цель — проверить согласованность начальных условий и попытаться найти удовлетворительное решение, если условия не согласованы. Для этого при заданном векторе начальных условий для дифференциальных уравнений ищется значение алгебраической компоненты, удовлетворяющей нелинейным уравнениям. В табл. П2.26 дается перечень кодов завершения программы.

Таблица П2.26. *Коды завершения программы SLODI*

Код	Описание
`ier = 0`	Нормальный выход
`ier = 22`	Арифметические ошибки
`ier = -1`	Переключение поведения
`ier = -2`	Завышена точность
`ier = -3`	Превышено допустимое число шагов
`ier = -4`	Многократно не прошел тест на точность
`ier = -5`	Не сходится метод Ньютона
`ier = -7`	Очень близки `tin` и `tout` для данного локального поведения
`ier = -8`	Не удалось вычислить согласованные начальные условия

Приложение 3

Описание компакт-диска

Компакт-диск, прикладываемый к книге, содержит папки, перечисленные в таблице.

Таблица. *Содержимое компакт-диска*

№ п/п	Имя	Содержимое
1	Licence	Лицензия на право некоммерческого использования демонстрационной версии среды визуального моделирования MvStudium
2	MvStuduim	Файлы для установки демонстрационных версий. Папка содержит две версии пакета: версию 4.013 и новую версию 4.1 betta, демонстрирующую пути дальнейшего развития пакета
3	Doc	Инструкция по установке. Описание новых возможностей среды по сравнению с версией Model Vision 3.2
4	Примеры	Примеры к *главам 1—4*

С последними версиями пакета MvStudium можно ознакомиться по адресу **http://www.mvstudium.com**.

Все сообщения об ошибках и предложения направляйте по адресу: **support@mvstudium.com**.

Литература

1. Колесов Ю. Б, Сениченков Ю. Б.. Визуальное моделирование сложных динамических систем. — СПб.: Мир и семья и Интерлайн, 2000. — 240 с.
2. Бенькович Е. С., Колесов Ю. Б., Сениченков Ю. Б. Практическое моделирование сложных динамических систем. — СПб.: БХВ-Петербург, 2001. — 441 с.
3. Колесов Ю. Б. Объектно-ориентированное моделирование сложных динамических систем. — СПб.: Издательство СПбГТУ, 2004. — 206 с.
4. Сениченков Ю. Б. Численное моделирование гибридных систем. — СПб.: Издательство СПбГТУ, 2004. — 239 с.
5. Колесов Ю. Б., Сениченков Ю. Б. Моделирование с систем. Динамические и гибридные системы. — СПб.: БХВ-Петербург, 2006. — 224 с.
6. Колесов Ю. Б., Сениченков Ю. Б. Моделирование с систем. Объектно-ориентированный подход. — СПб.: БХВ-Петербург, 2006. — 192 с.

Дополнительная литература по главам

К главе 1

7. Дмитриевский А. А, Лысенко Л. Н. Внешняя баллистика. — М.: Машиностроение, 2005. — 608 с.
8. Дмитриевский А. А., Иванов Н. М., Лысенко Л. Н., Богодистов С. С. Баллистика и навигация ракет: Под ред. А. А. Дмитриевского — М.: Машиностроение, 1985. — 312 с.
9. Основы теории систем управления высокоточных ракетных комплексов сухопутных войск / Б. Г. Гурский, М. А. Лющанов, Э. П. Спирин. Под ред. В. Л. Солунина. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2001. — 328 с.

К главам 3 и 4

10. Форсайт Дж., Молер К., Малькольм М. Машинные методы математических вычислений. — М.: Мир, 1980. — 278 с.

11. Каханер Д., Моулер К., Неш С. Численные методы и программное обеспечение. — М.: Мир, 1998. — 575 с.

12. Ортега Дж., Рейнболт В. Итерационные методы решения нелинейных систем уравнений. — М.: Мир, 1975. — 278 с.

13. Хайрер Э., Нерсетт с., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. — М.: Мир, 1990. — 512 с.

14. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие задачи и диффренциально-алгебраические уравнения. — М.: Мир, 1999. — 685 с.

15. Brenan K. E., Campbell S. L., Petzold L. R. Numerical solution of initial-value problems in differential algebraic equations. — Elsevier Science Publishing Co., 1989. — 205 pp.

16. Фаддеева В. Н., Колотилина Л. Ю. Вычислительные методы линейной алгебры. Набор матриц для тестирования. — В 3-х томах // Ленинградское отделение Математического института АН СССР. — Л., 1982.

17. Ракитский Ю. В., Устинов С. М., Черноруцкий И. Г. Численные методы решения жестких систем. — М.: Наука, 1979. — 208 с.

18. Морозов А. Д., Драгунов Т. Н., Бойкова С. А., Малышева О. В. Инвариантные множества динамических систем в Windows. — М.: Эдиториал УРСС, 1998. — 240 с.

19. Гантмахер Ф. Р. Теория матриц. — М: ГИТТЛ, 1953. — 491 с.

20. Градштейн И. С., Рыжик И. М. Таблицы интегралов, сумм, рядов и произведений. — М.: Наука, 1971.

21. Кузнецов А. П., Кузнецов с. П., Рыскин Н. М. Нелинейные колебания: Учеб. пособие для вузов.— М.: Издательство физико-математической литературы, 2005. — 292 с.

22. Пановко Я. Г. Введение в теорию механических колебаний. Пособие для вузов.— М.: Издательство физико-математической литературы, 1971. — 239 с.